3D Orbital Motion NEA

Arda Sahbaz
Candidate No: 2434
Centre No: 12676

Table of Contents

Analysis

Problem Identification

Astrophysics is a very common interest among students, however, because it is such a small topic in most GCSE physics specifications, most of the time it is never taught properly so most students are left confused and disappointed. An interactive simulation could assist teachers in teaching the topic and possibly pursue a passion for space for some students. The use of a simulation instead of a video to teach orbital motion would be substantially better due to the interactivity a simulation provides, changing the initial orbital parameters such as the mass, semi-major axis, inclination, etc. would help students understand Kepler's laws of planetary motion.

There are currently a lot of 2D orbital motion simulations for educational use. However, for students needing more assistance in visualising orbital motion, there are no educational 3D simulations online. I could take another 2D orbital motion simulation approach and add more features to make it more educational for students, however this would not be sufficient enough as 2D simulations only show the eccentricity of an orbit but not its inclination as motion is only captured on the x and y axes, losing the z axis which accounts for the inclination. This makes 2D orbital motion simulations useless for education. The solution to this would be an interactive 3D orbital motion simulation that is both educational and fun to play.

Stakeholders

The intended users of this simulation are primarily physics teachers and students, it is specifically designed for educational purposes to aid the teaching of orbital motion in classrooms and make it more engaging. I chose these stakeholders as they would be the best to consult when planning and testing the simulation as they know the most about physics and they will be using it. I could've maybe chosen a university physics professor as my stakeholder, but their use of the simulation would be in a far more advanced level compare to teachers and students in a sixth form/secondary school

Other stakeholders to keep in mind could be space enthusiasts, who may use the simulation for personal learning and exploration, they're input on the simulation function will also be valuable as their satisfaction with how fun the simulation is to play with, will foreshadow the student's engagement in the topic, due to the simulation. I could remove space enthusiasts as stakeholders however this could make my simulation less interactive and more boring to students which could affect the solution to my problem drastically as students are the representative users of this project.

Why is it suitable for a computational approach?

For an interactive 3D orbital motion simulation, a computational method is well suited, as building 3D simulations requires manipulating multidimensional data and performing complex calculations in real-time, which computer technology allows. An interactive and real-time simulation can only be achieved by a computational approach rather than static graphics.

Also, the ability to adjust parameters and observe their effects in real-time will be impossible without a computational approach. A computational approach also allows the simulation to be accessed as an online tool making it readily available for education.

Taking other approaches to solve this problem would be impossible, as the problem itself is one based on amending an improper computational approach.

Interview

To get the most effective and helpful solution to my problem, it will be best to ask the stakeholders of the simulation what they want/need. I've interviewed teachers and students in my school who are involved in physics to help tailor this simulation for educational purposes. As they will most likely be the ones using it and they're the ones I'm solving the problem for. I could skip the interview stage and guess what my stakeholders will like based on stereotypical physics lessons and student needs. However, this approach will not be reliable at all to have a satisfactory final product for the users of the simulation

Suitable Stakeholders:

- Mr Goosen - Physics teacher
  - **Contact:** School, email; will be regularly available.
  - **Why are they suitable?:** They have extensive knowledge of computer simulations for physics and they host the computational physics challenge in our school so can use it in the club.
  - **How will they make use of the solution?:** Can potentially use solutions in lessons to teach
  - **Why is it appropriate to their needs?:** They are a teacher so the interactivity and customizability of a simulation can make it easy for them to teach the topic

- Ibrahim – Physics Student
  - **Contact:** Social media, may not be regularly available.
  - **Why are they suitable?:** They study physics and students are the main target user group. Getting feedback directly from one person provides an opportunity to review desired features and usability considerations.
  - **How will they make use of the solution?:** They can use the simulation to conduct virtual experiments that analyse how different parameters affect orbital motion.
  - **Why is it appropriate to their needs?:** It is appropriate because it provides an engaging, hands-on learning method to complement textbooks. Visualization makes it easier to understand complex physics.

- Dr. Repetto – Physics Teacher
  - **Contact:** School, email; will be regularly available.
  - **Why are they suitable?:** As an astrophysics graduate, they have strong background knowledge in orbital mechanics and physics principles the simulation aims to cover. This enables them to provide informed and relevant feedback.

- **How will they make use of the simulation?:** They could use the simulation to visually demonstrate physics principles in a more interactive way during lessons. Or for supplemented self-learning for students
- **Why is it appropriate to their needs?:** It is appropriate because interactive 3D models improve understanding of complex spatial concepts like orbits. Customization allows tailored lessons. Engages students in lessons more than static diagrams do.

- Eren Ozsayan – Physics Student
  - **Contact:** Phone, email, school; will be regularly available
  - **Why are they suitable?:** They study physics and have a huge passion for astrophysics, their passion for space means their input on the solution can lead to more fun and engaging outcomes.
  - **How will they make use of the solution?:** They can use the simulation to satisfy their personal interest and for exploration. For example, conducting "what if" experiments by changing orbital parameters.
  - **Why is it appropriate to their needs?:** It lets them experience concepts that fascinate them and feeds their curiosity

Interview Questions

1. What is your role or interest in the field of orbital motion? How would you use a 3D simulation tool?

2. What specific features or functionalities would you expect from a 3D orbital motion simulation tool?

3. How important are visual accuracy and realism in the simulation for your needs?

4. How important is the ability to customize parameters such as mass, size, and initial conditions in the simulation?

5. Are there any additional features or functionalities that you would like to suggest for the simulation?

Interview Answers

- Mr Goosen:

1. "I'm a physics teacher and also lead a program that can illustrate orbital motion to kids, so I guess I can use it to show to kids in class"

2. "I would need it to cover everything taught about orbits in the OCR A Level Physics specification"

3. "How the planets/stars look is not too important for educational use, however speaking in the context of realism, I would want the orbital distance of the planets to be correct to scale, but I obviously would not want the size of planets and speed of orbits to scale as they will be very small."

4. “Not that important. I just need it to show the orbits of our solar system.”

5. “I'd like it to clearly demonstrate Kepler's Laws as they are pretty hard for students to understand, maybe having modes in the simulation that are solely used to educate Kepler’s laws will be useful. Another feature could be the use of vectors to show the direction of velocity/acceleration/gravity. Lastly, a display of the planet's absolute speed relative to the planet, or maybe if you can, implementing a binary star system would be fun to see”

- Ibrahim Seker:

1. “As a participant in the computational physics club, an orbital motion simulation will be very useful for me as I need to create a 3D animation. Having seen a 3D simulation can guide me in what I’m doing in my club.”

2. “Features that you should include are an animation of the orbital motion and a point to reference where exactly it is during its motion. I would like to see multiple orbits on the same graph, which are also colour-coded, to put other planets into perspective. It would also be nice to toggle between single-orbit and multi-orbit views.”

3. “High visual accuracy sounds nice; most simulations are very simple and don’t really look good. As a student, I would like to see realistic graphics rather than basic shapes and colours.”

4. “I do not find it that important as the solar system as it is, is enough to show orbital motion, but may be a good extra feature you can add”

5. The ability to toggle different planets in and out of the simulation would be amazing. That way we can focus on specific concepts and comparisons of planets between their motions. Minimalism will help make the simulation clear and easy to learn from

- Dr Repetto:
    1. I have a PhD in astrophysics, and also, I am a physics teacher, so I would probably use an orbital motion simulation for teaching or just to mess around with as I am a space enthusiast.
    2. It would allow the user to change initial orbital parameters such as eccentricity, semi-major axis, inclination, impact parameter
    3. Not very important
    4. It would be very important for me to help see the effects changes have on an orbit so that it can be clearly understood by students
    5. Allow user to control step size of integration to control computational time/resolution

- Eren Ozsayan:

1. “As a student studying physics and interested in astronomy, I think a 3D orbital motion simulator would be really cool! It would help me visualize and better understand abstract concepts like elliptical orbits that are hard to grasp from textbooks alone.”

2. “I'd want to be able to see detailed 3D visualizations of different types of orbits and have control over setting the mass, velocity, distance and other parameters. Easy-to-use controls and clear displays of orbital measurements are important too.”

3. “Visual accuracy is not necessarily that important to me. As long as the physics is modelled correctly and the overall motion looks reasonable, some graphical simplifications are fine. I care more that I can clearly see the shape and features of the orbit.”

4. “Being able to fully customize parameters is very important for an effective educational simulation. I want to be able to freely experiment by changing mass, velocity, distance and other factors to really grasp cause-and-effect relationships. Pre-set scenarios are fine, but I learn more from open-ended simulation.”

5. “Features to speed up time and trace orbital paths would be really helpful. Overlaying orbital parameter plots would also let me geek out on the analytics! The ability to load actual astronomical data to simulate real-world scenarios would take it to the next level.”

Reflection

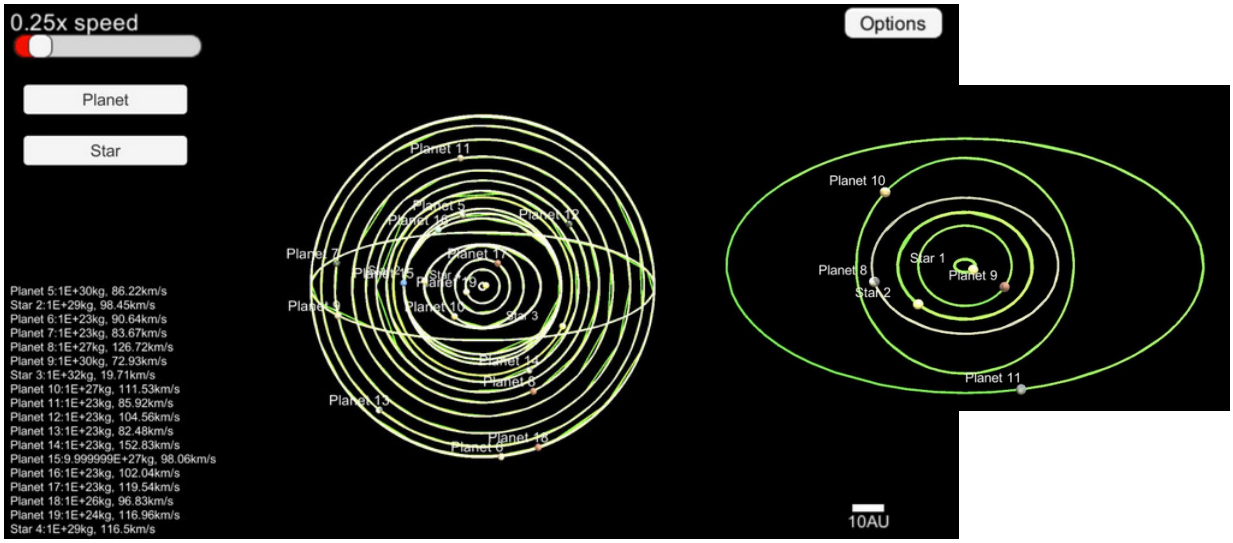
From the answers above I can say that my stakeholders are looking for more of a practical, useful and accurate model rather than something that looks pretty and minimal. Because all of my stakeholders are either students or teachers my solution has to solve an educational problem and I will definitely make it more educational than aesthetic.
What I grasped from the interviews is that the features that I need to most definitely include are: customizations of parameters, change of speed, and display of planet speed. A feature that is not that important and does not need to be included is visual accuracy, as long as it is mathematically accurate.

Research

Similar Existing Solutions

Orbit Simulator by Dino Code Company

https://dino-code-company.itch.io/orbit-simulator



Overview:

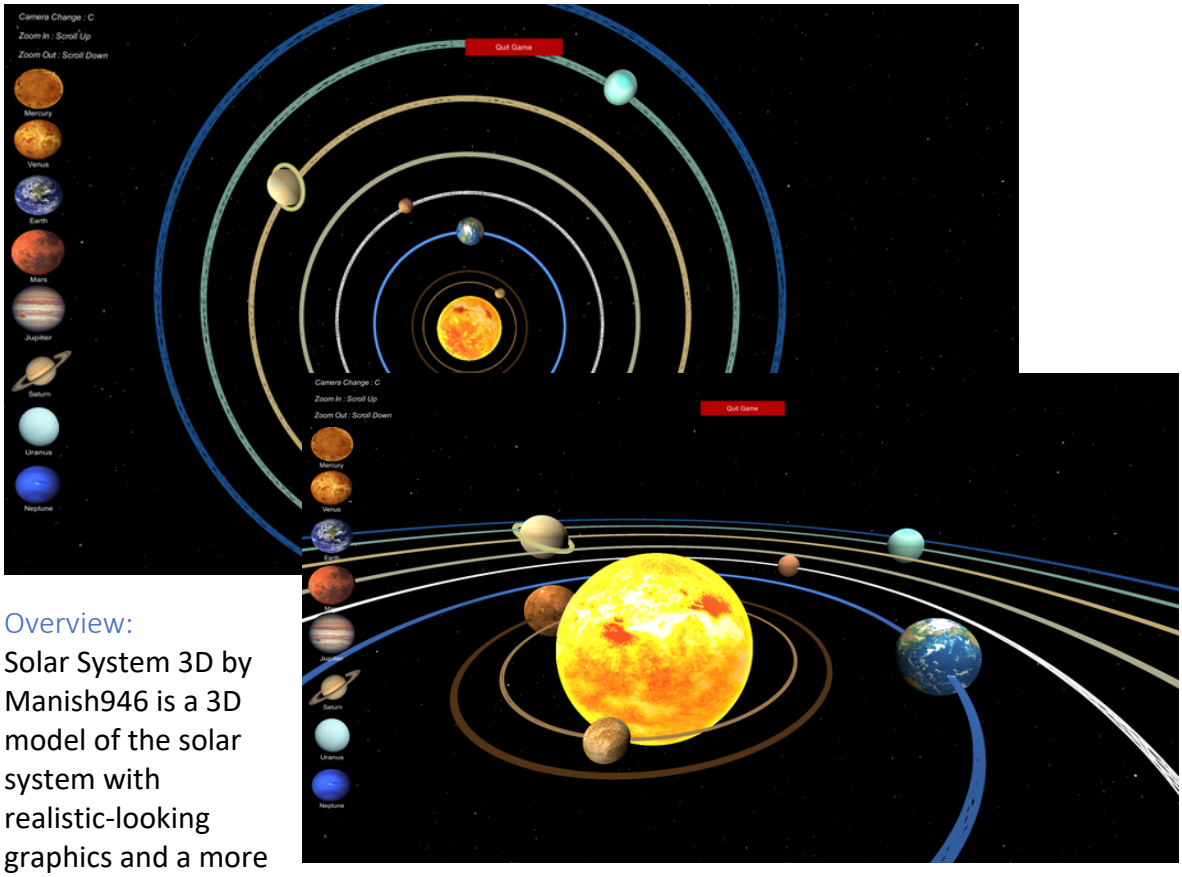
Orbit Simulator by Dino Code Company has a very simple 2D model of the solar system. There is very basic graphics and it displays the speed and mass of every planet. You can also create a custom system by adding multiple planets or suns, but it is very limited and does not really work as all objects are initially placed in the middle and it takes a long time for them to go to the correct spot.

Features I can approach with for my solution based on this research:

Orbit Simulator is similar to my solution; however, it is not 3D. I would definitely apply to my solution some features from Orbit Simulator like the display of the speed and mass of each planet, creating a custom system and a slider to change the speed of the simulation. But would take a slightly different approach. For example, for the planet information, I will not just display it all in a hard-to-read format in the corner. I'd rather allow the user to select a planet and see the information live as this will reduce clutter and make the simulation easier and less of a pain to use which would keep students attention.

Also, the orbital paths in this simulation are not accurate due to its limitation caused by it being 2D, making it useless for educational use and the visual simplicity makes it too hard to view. To not make the same mistake for my solution I would use different colours for orbital paths, larger sizes for planets and an option to zoom.

Solar System 3D by Manish946
https://manish946.itch.io/solarsystem3d



Overview:

Solar System 3D by Manish946 is a 3D model of the solar system with realistic-looking graphics and a more interactive approach. The viewing angle of the solar system can be changed by selecting a planet to look at its perspective, making the simulation more fun to play around with.

Features I can approach with for my solution based on this research:

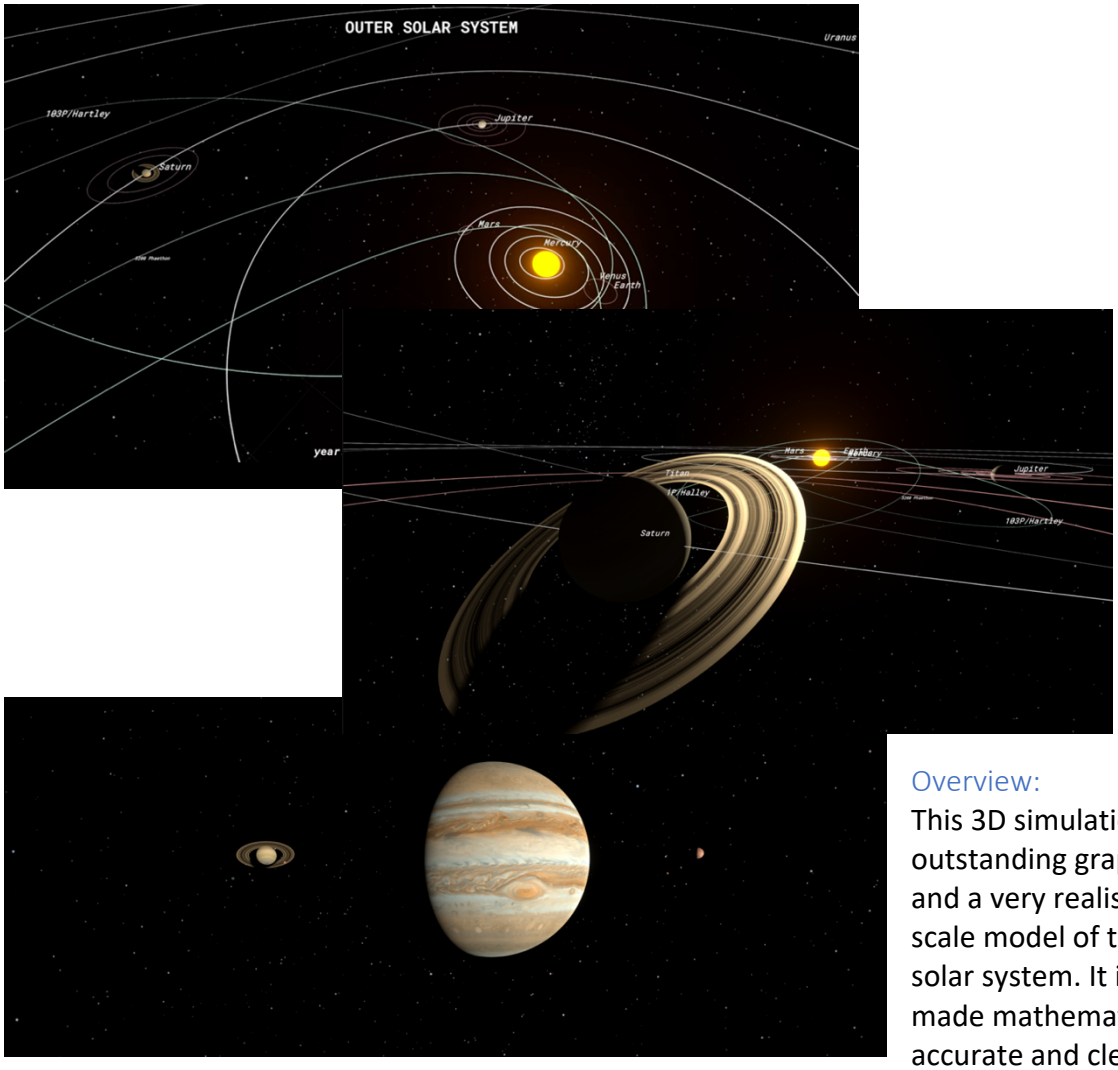
From this simulation, I would definitely try to apply a similar look to the graphics for my simulation, it looks not too GPU intensive but detailed enough to make out what the planets are without knowing their names. Another feature I would like to apply is the range of camera angles available in the simulation, it makes the simulation feel more interactive, realistic and useful. I could have my simulation fixed in a specific angle e.g., a bird's eye view, however this would be very limiting for the user.

The description of each planet when you click on it is a feature I could definitely add to my simulation, but my approach will be done in a more informative and educational way, as my solution is made to solve an educational problem.

The display of all the planets on the left to select them and display the solar system in their POV wastes a lot of space and makes the simulation feel too messy, If I were to implement this feature I would create a slide-down menu and select the planets from there as this feature is not essential and wastes spaces by displaying the same planets orbiting the sun which can also be clicked on directly to view solar system from its POV.

Solar System Simulation by Scot Anderson

https://scottanders.itch.io/solar-system-simulation



Overview:

This 3D simulation has outstanding graphics and a very realistic to-scale model of the solar system. It is made mathematically accurate and clearly presents the orbits of all planets and objects in the solar system. It provides a third-person view of most objects in the solar system including moons of other planets like Jupiter and Mars.

Features I can approach with for my solution based on this research:

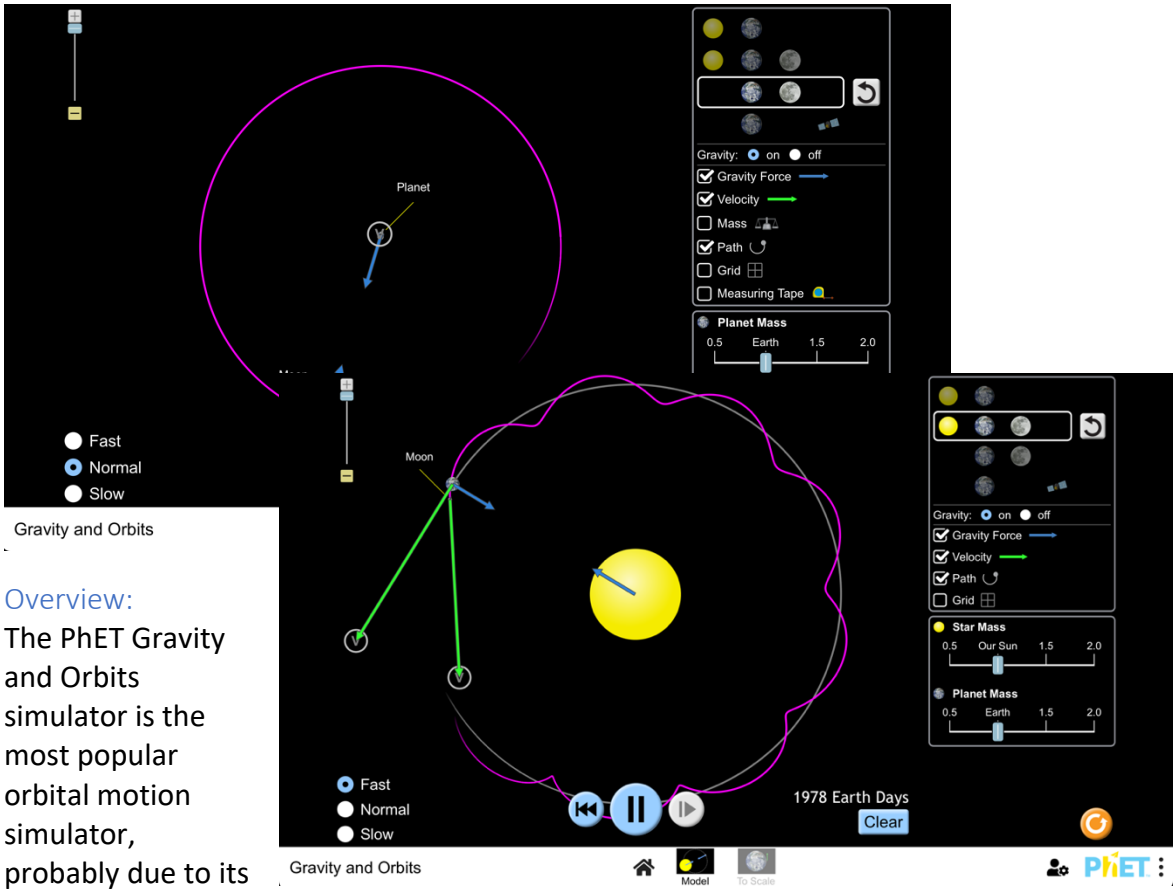
This simulation is very detailed which can be good but there are too many objects so it is hard to focus on the orbit of specific planets and the interactions between them. So my approach, if I were going to add this many celestial bodies, would be to an option to hide the ones I do not want to see.

Also, the graphics behind these simulations and the shadows of the planets not only make the simulation look more realistic but also help the user understand how the position of planets affects day-night cycles which is something I would definitely try to implement into my solution if I can. I could not add this to my simulation, but this would make the educational abilities of my simulation limiting, as light, in space, is something we learn about in physics so it would be useful to have.

It is impressive how mathematically and visually accurate this simulation is. However, I think this is not a good thing as there is no real balance between scientific accuracy and visual polish making it unsuitable for educational use. The planets are too small and hard to see, there are way too many celestial bodies to look at, and there are so many overlapping orbital paths making it confusing to analyse. So, I would make my approach so that there are less celestial bodies, as having more doesn't mean my simulation will be more educational it will just make it more informative

Gravity and Orbits Simulator by PhET

https://phet.colorado.edu/sims/html/gravity-and-orbits/latest/gravity-and-orbits_all.html



Overview:

The PhET Gravity and Orbits simulator is the most popular orbital motion simulator, probably due to its simplicity. There are options for viewing orbital motion for the Earth, Sun, Moon and a satellite. There are also features like a measuring tape helping users understand the distance of the celestial bodies, velocity and gravity vector arrows which most simulations do not have.

Features I can approach with for my solution based on this research:

This simulation being the most used for educational purposes is definitely worth taking inspiration from for my solution. The vector arrows for gravitational force and velocity are definitely the most unique features out of all the simulations. It helps students who use the simulation understand how orbital mechanics work even more clearly compared to other simulations as it tells the user where the celestial body's trajectory is in real time. So this educational approach is one I will 100% try to implement in my simulation as not having it will make my simulation lack in educational tools.

The ability to change the mass of the celestial bodies in a slider is a very suitable approach for an educational simulation I definitely have to include in my solution, as it will be

extremely useful for students to observe the effects of different parameters and this is probably why this simulation is chosen most for educational purposes. Not having this would mean that the students cannot see live how mass effects orbits.

However, the way the simulation speed feature is implemented is not the best as I think having a slider allows for there to be more interactivity and control over the simulation which is what is needed to make this solution a success. This approach has only three options, whereas another approach can be one where the user can change the speed to many multiples of time, not just fixed ones, making the simulation more friendly to the users needs.

One thing this simulation lacks is the ability to change the viewing angles, this makes the users feel more locked in and have less control over what they are seeing making them possibly favour this simulation less. An approach with more freedom in the viewing angles will ensure that the user will see how certain parameters can effect the orbits.

Feedback

I've asked my stakeholders for their opinions on the features I'm going to add based on my research and I also showed them pictures/videos of other solutions. They will each provide feedback on features from a random simulation I've taken inspiration from. Doing this will be very beneficial as it will make my final solution more realistic, as I have looked at existing solutions, and because I showed my stakeholders more features that they maybe couldn't have thought of it will make the simulation more suitable for the users requirements.

Mr Goosen – Orbit Simulator by Dino Code Company

"I appreciate that the Orbit Simulator displays key parameters like planet mass and velocity. However, I definitely agree that this simulation needs a lot more improvement to be used for educational purposes. It looks a bit cluttered and just unappealing. I feel like after looking at this simulation I think visual accuracy is somewhat needed and is a bit important."
"The ability to build custom systems is not that important for an educational simulation. Just being able to change the parameters is enough."

Ibrahim Seker – Solar System 3D by Manish946

"I really like how this simulation looks visually. The different colours for orbital paths make it easier to look at. However, there is no inclination of the planets making it inaccurate."
"The description of planets is not really a necessary feature, the simulation only needs to be made to present orbital motion only, I feel like it will be a waste of time but if it is not hard to do, you should go ahead and add it to your simulation anyways."

Eren Ozsayan – Solar System Simulation by Scot Anderson

"The graphics in the simulation look superb. It looks very interactive and the camera controls make it feel smoother and better to play around with. However, there are way too many planets, moons, and satellites which makes the simulation feel a bit overwhelming."
"I think you can inherit many features from this simulation to yours but tailored for a more educational purpose!'

Dr. Repetto – Gravity and Orbits Simulator by PhET

"The ability to view velocity vectors and gravitational force arrows in real-time would be incredibly helpful for demonstrating the physics concepts visually to students, it will help them understand how centripetal forces work together with gravitational ones."
"I also like the mass adjustment slider it allows students to experiment with the effects different masses have on the trajectory of orbits in a discovery-based learning approach."
"I 100% agree with how limiting the fixed camera is, being able to pan around the orbits would give the students a more immersive experience."

Reflection

The feedback from my stakeholders is very valuable to help shape my simulation and tailor it to fit their preferences best. What I gathered is that users are looking for a not too cluttered but complex simulation with many educational features to provide interactivity for a hands-on learning experience.

Essential features

Based on the above research

Feature	Explanation	Justification
Camera Angles	Changes your POV of the solar system	Providing multiple camera angles helps users better understand orbital motion by visualizing it from different perspectives. An alternative approach is to have the camera fixed in one position, for example a birds eye view, but this would be very limiting for the users making it hard for them to visualise the orbit in terms of incilination.
Speed Slider	Allows you to change speed of animation	A speed slider is a common and easy way to control animation speed. A different approach could be keyboard shortcuts, which may be quicker for some users but less accessible for others. The slider provides a good balance between ease of use and functionality.
Distance	Displays distance camera is from earth in astronomical units	Displaying distances in astronomical units helps users grasp the vast scales involved in orbital motion. An alternative approach could be using units like kilometres, but these numbers would be extremely large making it harder for the user to comprehend sizes of orbits
Scale Bar	A line/bar divided into parts labelled with distances	A scale bar provides a visual reference to help judge distances and sizes within the simulation which would be much easier than a numerical approach, which would make users struggle to comprehend the scale has they have to think more.

Based on the Interviews

Feature	Explanation	Justification
Customizable Parameters	Change initial starting conditions of planets	Allowing users to modify initial conditions like mass, velocity, and position helps them understand how these parameters affect orbital motion. An alternative could be providing a set of pre-defined scenarios, but customizable parameters offer more flexibility and makes learning more fun by adding an exploration aspect to it.
Speed Slider	Allows you to change speed of animation	As mentioned before, a speed slider is a common and intuitive way to control animation speed. The justification for using a slider over alternatives like keyboard shortcuts remains the same.
Planet Information	Displays information about planets such as speed, mass, etc.	Displaying relevant planetary data helps users understand the relationships between properties like mass and the orbital changes they see. An alternative could be allowing users to calculate this themselves, but having it readily available is more convenient for an educational tool.
Mathematical Accuracy	Displays planets in right scales, distances, speeds, and trajectories	Ensuring mathematical accuracy is crucial for an educational simulation, as inaccuracies could lead to incorrect information. There isn't really an alternative approach on the priority of accuracy as it is vital for the simulation to work properly.

Based on Feedback

Feature	Explanation	Justification
Speed Control Via Keyboard Shortcuts	Allows you to change speed using keyboard rather than a slider controlled using mouse	A speed slider may be more accessible for some users, however keyboard shortcuts can provide a quicker way to control speed for users comfortable with keyboard input. The two approaches could be combined to cater to different user preferences.

Colour-coded Orbital Paths	Assigns colours to different orbits of celestial objects	Colour-coding orbital paths makes it easier to distinguish and track multiple objects in the simulation. An alternative approach could be labelling or highlighting paths in other ways, but colour is a highly effective way to differentiate things.

Limitations

- The scope of celestial objects in the simulation - Simulating complex and accurate gravitational physics across an entire solar system can require significant processing power as the gravitational interactions between all celestial objects will be very hard to account for limiting the range of different objects you can have. Instead of attempting to simulate every celestial object within the solar system, a focused approach could be taken. By selecting only the key celestial bodies and their interactions, the demand on the computer can be significantly reduced while still providing a working educational tool.

- Accuracy trade-offs- Precisely simulating celestial objects in the solar system will make it hard to view certain bodies in the solar system as some planets may be too small to look at from different perspectives. So, the approach has to balance scientific accuracy and the visual polish of the simulation e.g., the size of some planets can be enlarged so that you can see them in the simulation making them not 100% accurate but accurate enough to understand the science.

- User engagement- Maintaining students attention to the simulation may be challenging if the simulation does not look appealing, is simple or is not engaging enough. So, my approach is to create a complex but not complicated simulation by prioritising simplicity and interactivity. This can be time and resources intensive however, I think it is worthwhile as nowadays students, are exposed to very modern complex software and they will appreciate something that doesn't look "old school" more.

Hardware/Software Requirements

Hardware

Requirement	Justification
Multicore Processor e.g. Core i7 series	Recommended for orbits with multiple planets, so calculations can be completed simultaneously. Which therefore saves time. Alternatively, a single core processor can be used, however this would make it very hard to run the simulation as parallel processing is an important feature this simulation

	needs due to the multiple objects being rendered at the same time.
At least 8 GB ram	To handle simulation data, which will consist of lots of numbers and calculation results, giving a smoother experience. Alternatively, I can try optimising the simulation for memory efficiency however, this will take a lot of time and may be out of my expertise
Dedicated Graphics Card e.g. Nvidia RTX 30 series	An integrated graphics card may not be capable enough to run a graphical 3d simulation. Alternatively, I could make my simulation less graphics intensive, however this will reduce the attractiveness of my simulation which in turn can affect students attention on the simulation.

Software

Requirement	Justification
Windows 10, MacOS, or Linux	These are the target desktop platforms for the best user experience, a different approach such as a web-based simulation may not work or lag a lot as this simulation is resource-heavy.
Game Engine	Unity will be best suited to build the simulation as it can. Handle 3D rendering and also is compatible with a lot of software's and easily accessible for users compared to a python-based engine which will result in a very resource inefficient.
Programming Language	C# is needed for Unity. Alternatively, using a simpler and easier language like python means you would need to use a lot of external modules causing compatibility issues.

Success Criteria

Criteria No.	Criteria	Explanation	Justification	How to Evidence
C-1	Functioning Main Menu	All buttons open to correct screens	It is essential to guide the user to the simulation,	A video clicking each button

			alternatively I can not have a main menu but this would make it harder to switch through different scenes as there wont be a central location which access all parts of the simulation	
C-2	Options menu features	Graphics, audio, controls settings adjustments	It extends accessibility and allows users to personally tailor the simulation to their preferences, alternatively I could've removed the ability to change these settings, however this would maker there simulation impossible to use for some users who need to lower their graphical settings	Screenshots demonstrating different options and their results
C-3	Consistent Design	Must be a simple and lightweight	Having a consistent design is important to make the simulation feel robust and professionally made a complicated design can be overwhelming for the user, and be unnecessarily resource intensive	Video
C-4	Tutorial functionality	Interactive lessons on core concepts	Crucial for educational engagement & comprehension, alternatively I can remove this feature as a whole, which wont be much of a problem, but would mean that some users would take a while to learn how to use the simulation	Screenshots of tutorial screens, task completion

			which can waste their time which could've been used for education	
C-5	Custom System	A custom solar system that the user can create to their liking	Helpful for user to understand how solar systems work beyond our own and to test out how certain orbital parameters effect the orbital characteristics, alternatively I can simulate the solar system of which the user cannot change parameters, which would mean it will be a bit harder to visualise how they play effect in the orbit, but would still serve as an educational tool	Video of creating a custom system and playing around with it
C-6	Scene switching	Ability to switch to all available scenes	This is needed so that users can access all parts of the program alternatively I could've made the whole simulation a single scene, however this would've made the simulation more likely to crash as there will be components running unnecessarily in the background	Video of switching to all scenes from every scene
C-8	Calculations/equations needed for accurate orbital paths	The simulation should accurately represent the elliptical paths of planets around the sun, following Kepler's laws of	This is needed because the primary goal of an orbital motion simulation is to accurately depict the motion of celestial bodies. There cannot be any other approach without the use of	Showing the code that presents the calculations. Presenting maths needed clearly beforehand

		planetary motion.	maths to make a orbital simulation	
C-9	Select Planet to Track	The simulation should recognise if the user has clicked on a planet and change the cameras viewing angle so that it tracks that planet	Allow for a default viewing method which dynamically moves with the planets. Useful when user provides limited manual camera controls. Alternatively I can remove this feature however this would mean that the user has to keep moving their camera to see what state a planet is in which can make the simulation frustrating to use	A video where different celestial bodies are clicked then tracked.
C-10	Zoom In/out	Allows you to change the viewing angle of the planets	Helps visualise how planets orbits look better and lets you have control in what you see rather than having a fixed view. Alternatively I can remove this feature however this would greatly limit the users control over the simulation making it hard for them to adjust the camera so they can see the orbital characteristics they are looking for	Screen shot of different viewing angles and code for the feature
C-11	Planet Graphics	Shows what planets actually look like rather than a circle	This helps to add realism, alternatively I could have used different colour circle to represent planets but this would have made the simulation	Display images going to be used to present the planets

			very bland and boring, also depicted what a planet is without having to click on it would no longer be possible	
C-12	Planet Orbit Line	Shows what path the planets are going to take	Helps to understand how orbits work as where planet is going is already predicted. Alternatively I could remove this feature, but this would make the simulation impossible to use, the user would not be able to see how the orbital characteristics change if any parameter changes	Screenshot in simulation
C-13	Time counter	Shows years it has been since simulation started	Help to understand orbits and orbital periods better, removing this would limit the user's ability to comprehend how time changes based on orbital parameters	Screenshot in simulation
C-14	Planet Information	Displays information about planets such as speed, mass, etc.	Good to know for orbital motion as you can see quantitively how speed changes in different points in the orbit, allowing you to understand it better. Alternatively, I could've removed this ability, however it would be hard to depict different parameters which can't be seen e.g. Force	Screenshot in simulation

C-15	Scale Bar	A line/bar divided into parts labelled with distances	Displaying distances in astronomical units helps users grasp the vast scales involved in orbital motion. An alternative approach could be using units like kilometres, but these numbers would be extremely large making it harder for the user to comprehend sizes of orbits	Screenshot in simulation
C-16	Speed Slider	Allows you to change speed of animation	A speed slider is a common and easy way to control animation speed. A different approach could be keyboard shortcuts, which may be quicker for some users but less accessible for others. The slider provides a good balance between ease of use and functionality.	Screenshot in simulation

Design

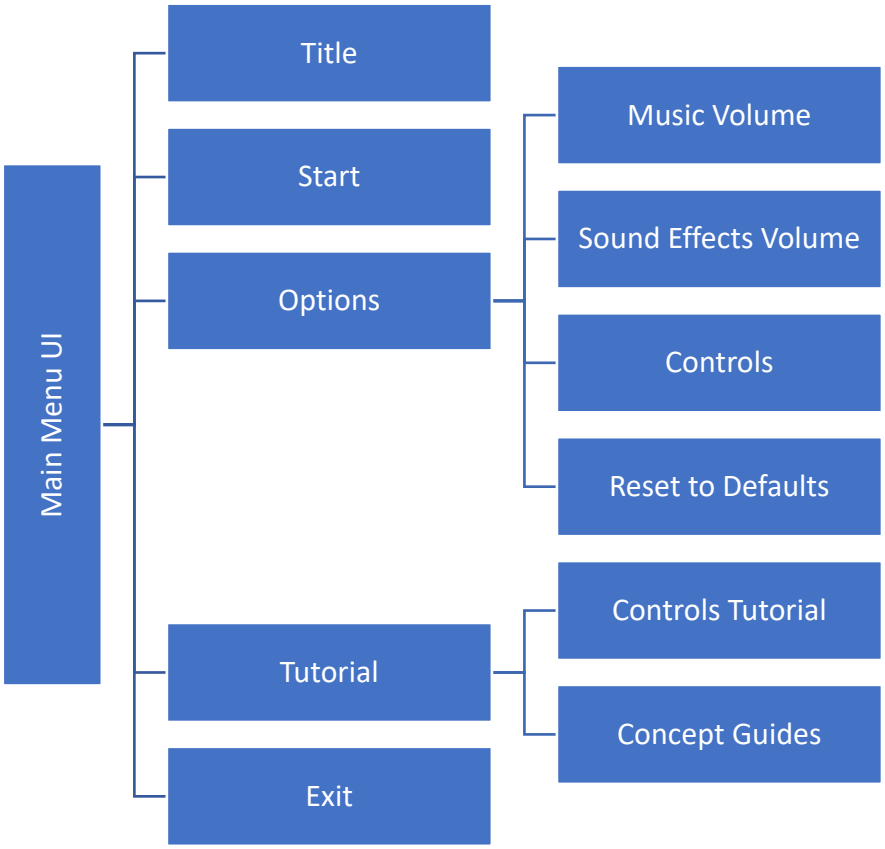
User Interface Design

Main Menu UI

Decomposing the Problem

The main menu will be the first thing the user will see when they launch the program. It will have many functionalities linked to it. So, I will be decomposing the problem by breaking it into smaller tasks so that they can be dealt with separately.

Hierarchy Chart of Game:



Coding the game without decomposing the problem will be extremely tricky and time consuming, so I created a hierarchy chart, alternatively, by creating a complex program without breaking it down to more manageable tasks you will not know where to start from

and possibly miss out parts you haven't thought about before leading to a less efficient lower quality program.

Main Menu UI Mock Up

I've created a mock-up of the main menu. Having a colourful space-themed design is much more appealing to the eye compared to me implementing a solid colour background which would have been less representative of the program.

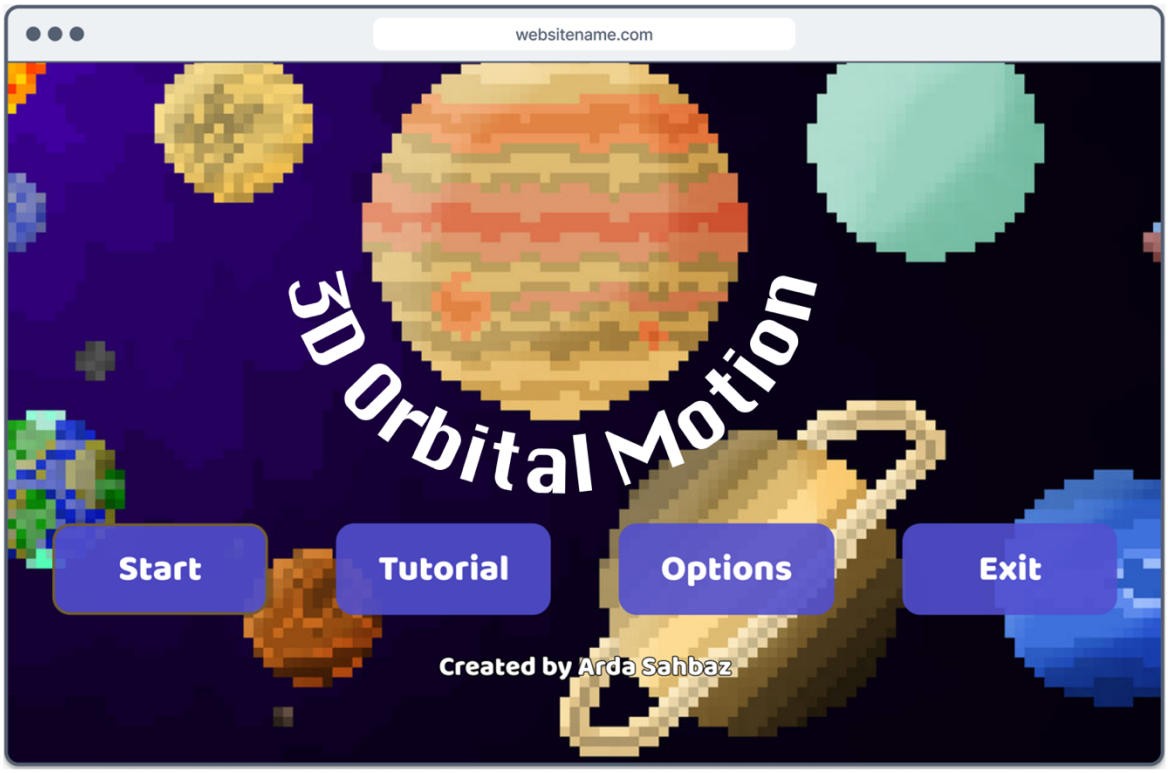


Figure 1: Main Menu UI Design

The menu needed buttons to let users navigate to various pages:

Start – Essential to allow users to have access to the simulation from the main menu. If I didn't have this the user cannot switch to the simulation scene
Tutorial – Very useful for younger students who may not know much about orbital motion. It will open up guided lessons that teach core concepts related to orbital motion. Different tutorials can cover basics like orbital shapes, Kepler's Laws, and gravitational effects. More advanced ones could dive into orbital mechanics formulas and parameters.
Options – Allows the user to tailor things to their preference. This will be great for education as it will make it easier to learn for different students with different needs. For example, some may not like loud music with their simulation so may turn it down or off to concentrate better.
Exit – Closes the simulation so that users are not stuck in it.
I've made the text white in the buttons and title to contrast with the background colour as it would aid legibility for people who have visual impairments.

Feedback

I asked my stakeholders for their opinions on the main menu UI. Although this doesn't serve a huge function to the simulation and how it works, it will be nice to know what a user's first

impression will be when opening the program as it can be a determining factor to them proceeding with using the program.

Eren Ozsayan:
"It lacks a hint of professionalism, and looks as if it is projected towards younger age groups, however, the design is simple, efficient and works."

Dr Repetto:
"I like the tutorial button straight next to the start button It will be useful for students who don't know much about orbital motion yet. The choice for graphical layout is clear and essential"

Ibrahim:
"I really like the space artwork; it makes sense."

Mr Goosen:
"The menu aesthetic is appropriate for the subject matter. I like how the title is big and curves around the planet."

Reflection

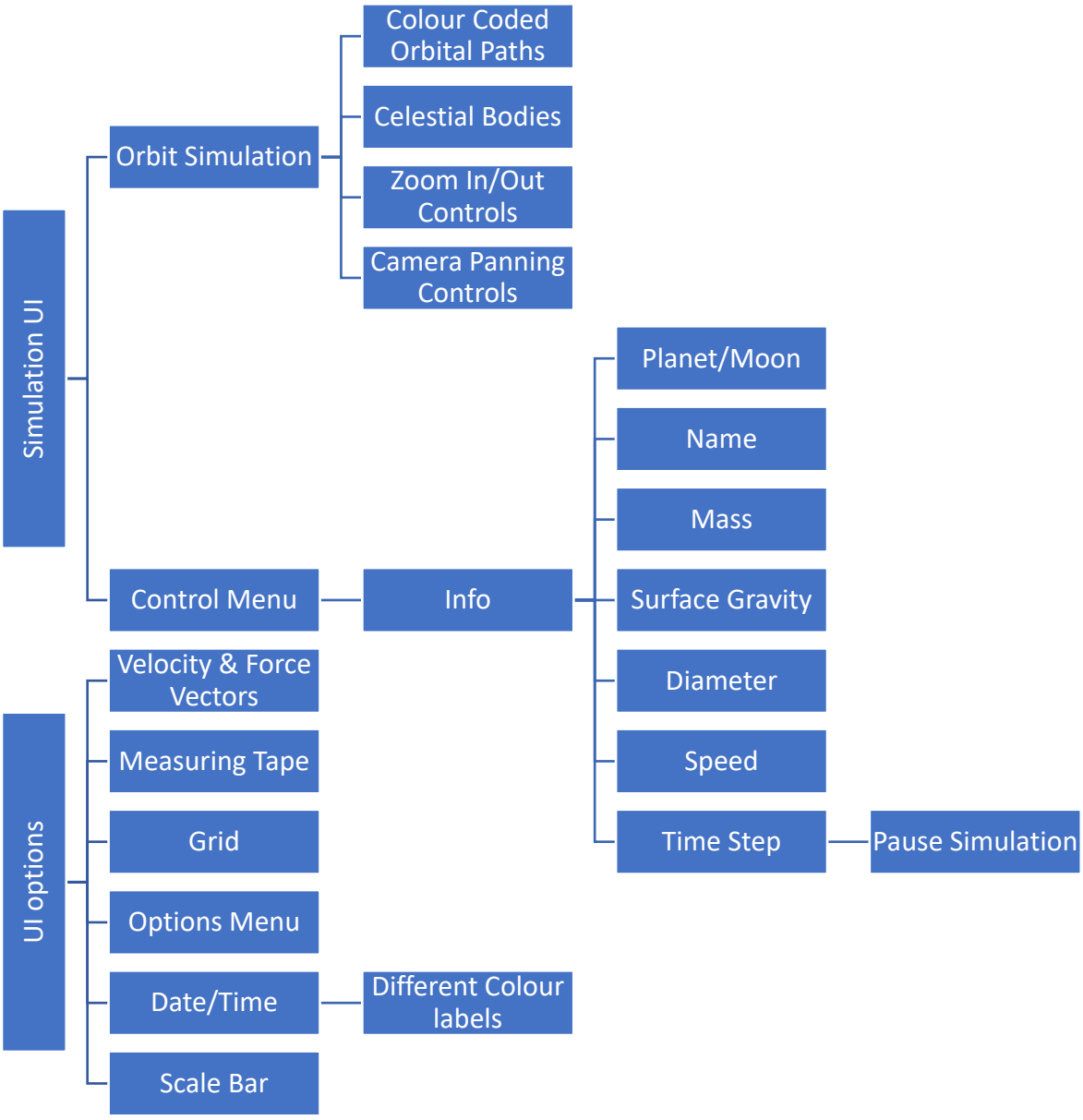
Overall, the stakeholders seem to be happy about my design choices. Eren thinks it lacks professionalism however since it is going to be used for education for a wide range of age groups, I would rather sacrifice professionalism for younger students, who tend to concentrate better with more colourful-looking educational tools rather than bland and simple ones.

Simulation UI

Decomposing the Problem

The simulation UI is the part of the program that will be used by the users most and, therefore is the most important section. So, breaking down the task and planning out what to do is essential to achieve a bug-free functioning simulation.

Hierarchy Chart



UI Mock-Up

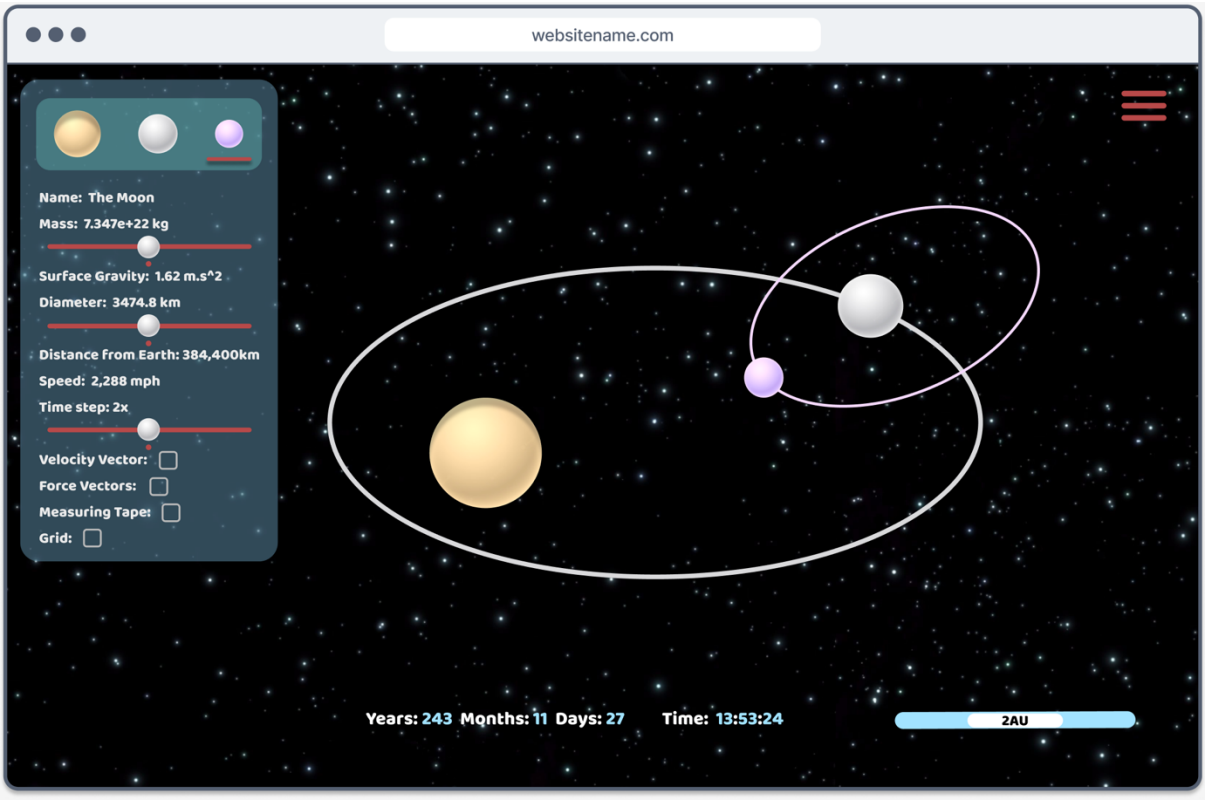


Figure 2: Solar System Scene UI Mock Up

This is a mock-up of what the simulation UI will look like. I believe a simple but not too minimal design will be appropriate for educational use as too many options will be too overwhelming for the user and will make the program hard to navigate.

Orbit Simulation

The design for the orbits is made to be easy to look at and analyse. I've used different colours for the orbital paths of planets, if I had multiple planets in the same colour orbits can cause confusion as it will be hard to spot which orbit belongs to which planet when looking at bigger orbits which are closer to other orbits. Also, it makes the simulation easier to teach if it is to be used by a teacher.

Options Menu

I've added a pause menu to access other pre-made orbital systems and to quit the app. Incorporating these features into one button makes the application more minimal and saves space for the simulation. If I had separate buttons for each option that were to be in the pause menu, it will take up a lot of space and make the simulation UI have a lot of clutter which can diminish the user satisfaction as their orbits will be covered by many buttons.

Date/Time

I've placed a date & time counter at the bottom in the middle of the simulation so it is out of the way but still accessible. By making the actual values of time a different colour on the labels I think it stands out more and makes the simulation look better to the eye which can possibly catch more students' attention.

Compared to them being all the same colour which would waste their time when trying to figure out the data.

Control/Info Menu

The control menu displays the planets at the top and allows you to select the one you want to view information for and to edit parameters of. The number of planets that fit on the top will be up to 4. If there are more planets you can scroll horizontally through the planets at the top of the menu and select the one you want to edit. For the mass and diameter slider, I set the original values in the middle of the slider and made it so that the user can move it to the left to decrease mass/diameter and right to increase. This makes the simulation easier to learn from and use because if I had it so that you have to type in the mass you want, it would be hard to use for people with not much knowledge about space as they might not know what masses to use to see sufficient enough effects in the simulation to learn from/analyse. By using a slider, the user can see the effects of changing these parameters live and it can make the simulation not only feel much more interactive but much easier to learn from to see the effects. Compared to having set options which would limit the user greatly in the parameters they can choose.

Figure 3: Side Bar

For the speed of the simulation, I added a timestep counter at the bottom where 2x is the default value and you can increase and decrease like the same with the mass/diameter sliders. However, I made it so that sliding the slider to the leftmost side pauses the simulation. This will be useful as the students can take a second to analyse the motion of the planets and also it saves space for the UI allowing more room to focus on the simulation.

I made the display of vectors for the simulation optional by making it a simple check-boxed control. Having the vectors as an optional setting allows for more versatility for students who may not need the vector arrows which prevents clunking up the view of the simulation. Finally, by adding a measuring tape and a grid as an option as well, the students can select these options to mathematically analyse the simulation. For example, to measure the distances between celestial bodies by utilising both the measuring tape and the grid.

Feedback

I asked my stakeholders for their opinions on the UI design choices for my orbital motion simulation. Their input will help improve the layout, visuals, controls and overall ease of use. I'm hoping to hear their opinions and iterate through their feedback so that the final interface is perfectly tailored for educational use. I could've implemented my design without stakeholder feedback however this could lead to a simulation that is less user friendly to target users.

Mr. Goosen:
“Visually it seems clear enough for classroom use. I appreciate the simplicity. Labelling the planets would help students with identification. The control menu parameters cover the basics we'd need to demonstrate the principles of orbital motion. Using sliders rather than inputs simplifies experimentation for students. Overall, it provides a focused interface to engage students without overwhelming them.”

Ibrahim:
“It looks alright, the coloured orbits definitely help make it less confusing. The controls seem easy enough to figure out. Being able to pause and change speeds is useful. I do think the vector display option should be more obvious since that's useful for learning. More speed settings could be nice too. It's a good start, I'd play around with it.”

Eren:
“I really like the clean visual style of the UI design. One feature addition I think could be useful is providing some keyboard shortcuts to control the speed of time like the spacebar to pause/resume. But overall, you clearly put thought into simplifying the interface to focus on comprehension rather than overloading it, which I appreciate.”

Dr. Repetto:
“I'd add an option to show the mass relative to the earth's mass. I also see that you have tilted the plane of the orbit. Is this a parameter that can be changed? Don't forget to change the unit for the surface gravity. Also, consider displaying some key measurements live, like apogee, perigee, period, etc. to further educational value.”
Me – “I definitely could add the mass relative to the earth’s mass. The orbit shown in the simulation isn’t Earth’s I just made a random system so it’s not accurate. I’m not sure if I should make it a parameter that can be changed as it may be hard to code but if you think I should I would definitely try to. Thank you for pointing the units out.”
“Yes, the orbital inclination would be too hard to code because you would need to add corrections to Newton's law of gravitation accounting for the shape of the planets and their angular momentum - perhaps in a few years :)”

Reflection
Overall, the stakeholders seem to be mostly happy with my design choices but also offered valuable feedback so to best match my stakeholder's needs I would definitely prioritise implementing Dr Repetto’s suggestion on displaying more key features like apogee, perigee and period for higher education users. As a teacher who not only teaches astrophysics but has a PhD in it, their input highlights how important displaying those parameters can be. Students could understand orbital behaviours better by visually viewing these measurements.
Also, as Mr. Goosen said, adding text labels directly next to the planets in the simulation will be much more convenient. It will avoid the extra step of having to click on the planets to learn their names which causes inconvenience to the users of the simulation.

Usability Features
Having my simulation accessible to everyone is very important, especially for one that is made to be used in an educational setting.

As you can see in my UI design I've made the buttons very big, and the text in the buttons and in the UI overall are contrasting. This is very important for people with bad vision as they may not be able to read what they are pressing or the values that they are seeing in the simulation sidebar. Alternatively, I could've designed my simulation so that it looks more aesthetically pleasing, and more minimal, thus having less contrast, smaller and cleaner buttons. But this would've made it unnecessarily difficult for the user to utilise the simulation.

I will make the keyboard and mouse controls in my simulation fully customizable. Users can open the settings menu and remap all controls to their personal preferences. This is extremely important for accessibility, because if I didn't do this people with certain physical disabilities may not be able to use the default control mappings and it also makes the simulation less of a hassle to use for people, instead making it more enjoyable to use, and this is important to keep students' attention to it.

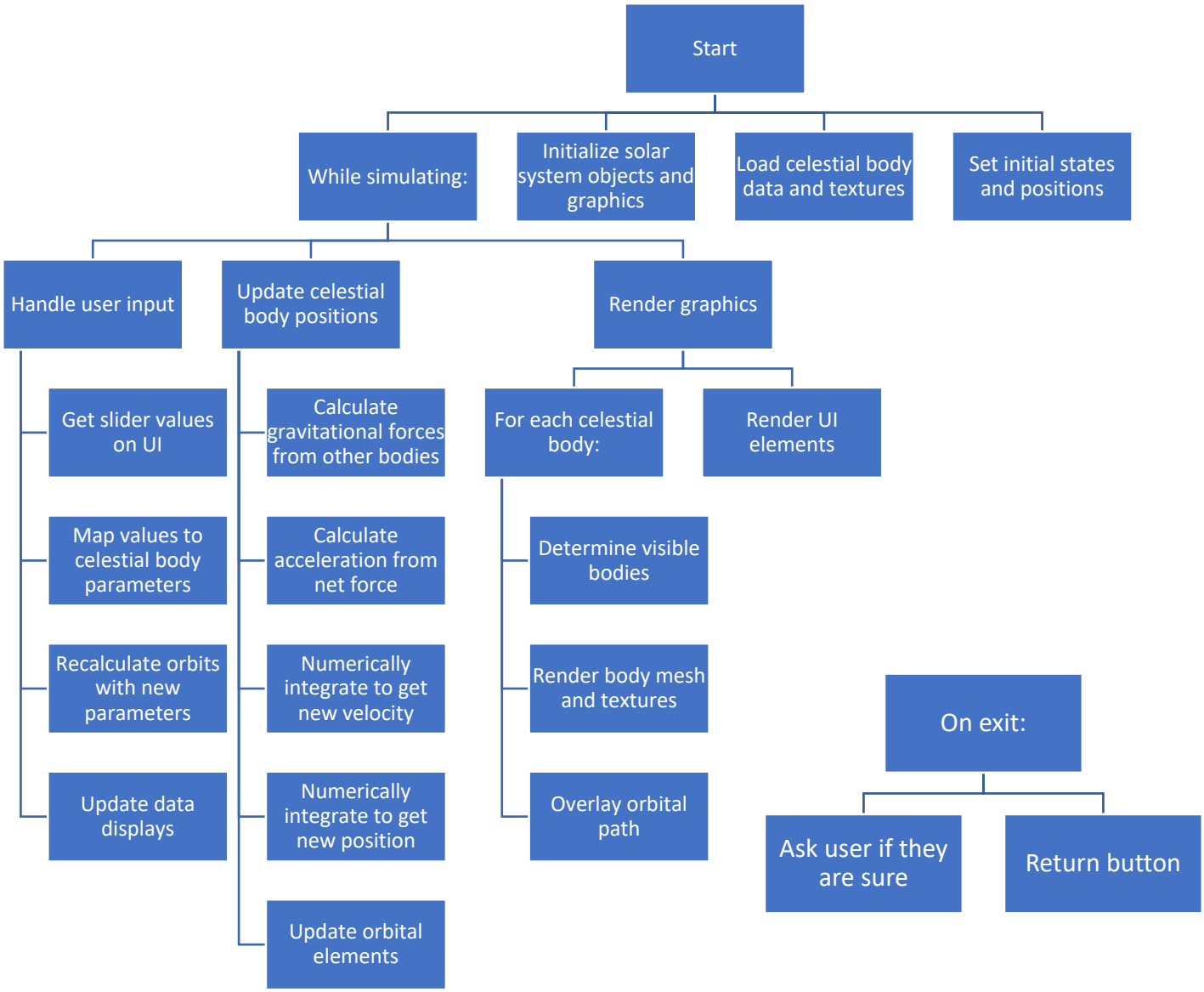
I will implement keyboard shortcuts for most common UI actions. For example, pressing 'P' will pause the simulation, 'R' will reset to default values, and arrow keys can be used to navigate menu options. This keyboard to UI interactivity speeds up the use of the simulation and avoids constant mouse movement for frequent actions. Alternatively, relying solely on mouse interaction would slow down usability and the student's productivity, also some student may not be able to use a mouse, so having only keyboard functionality in the simulation would be very important for them.

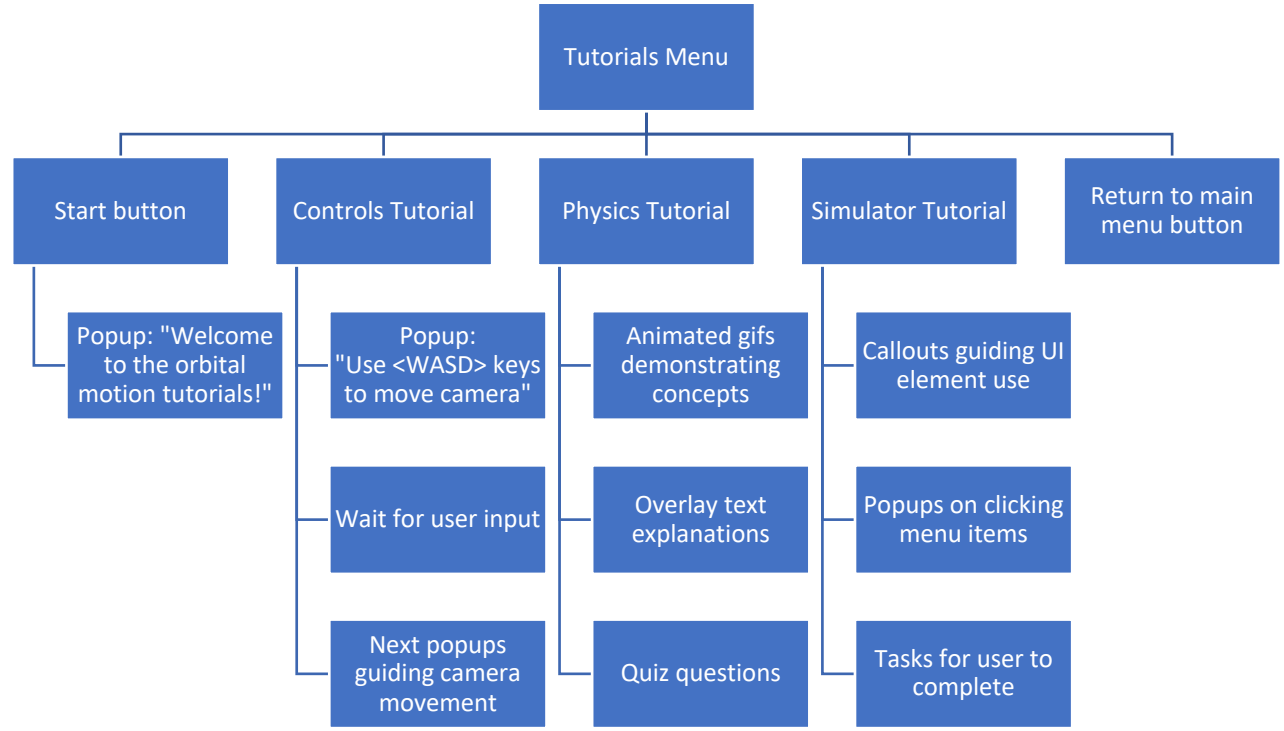
I will add sound effects to my simulation, this makes it more interactive and engaging. This will be useful for people who have bad vision, they may click the edge of a button and not sure if they have clicked it, so having sound effects for UI interactions will be an extra step of validation for them. If I didn't include UI sound effects it could make my simulation feel stale and boring due to the absence of audio feedback and could possibly make it less accessible for a niche group of users.

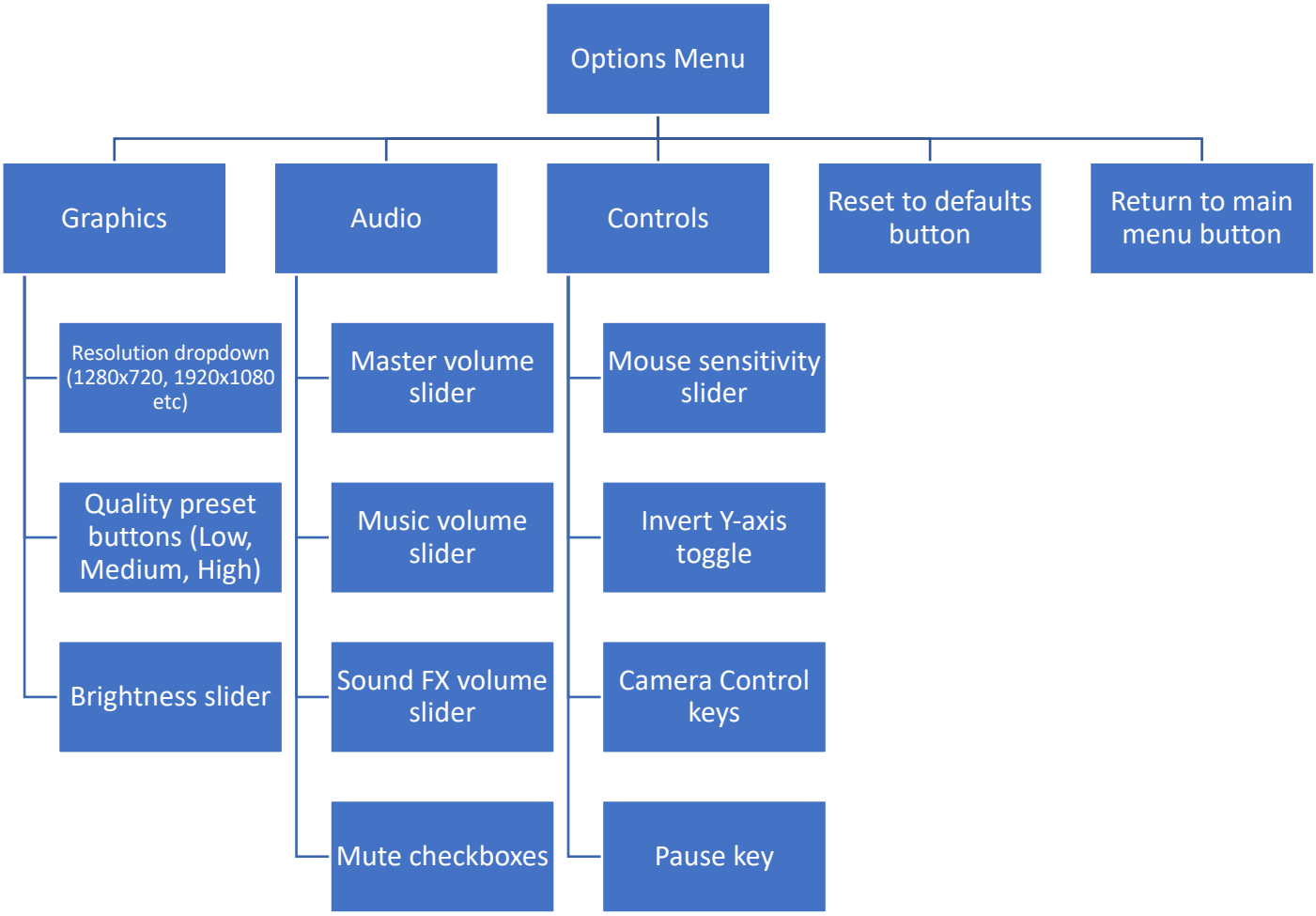
Algorithms

Modular Design Diagrams

To logically break down complex features into easy-to-manage steps I created flow charts for the algorithms that will be needed for the simulation to function. This is a useful way of programming as a modular approach allows for guided development and robust code because you can debug modules separately without affecting others which can possibly make the program unusable and waste even more time trying to fix. Alternatively, I could've taken a more sequential approach, however this would've made the code more difficult to maintain and debug compared to the reusability, maintainability, and testability a modular approach offers.




Tutorials Menu
Start button
Controls Tutorial
Physics Tutorial
Simulator Tutorial
Return to main menu button
Popup: "Welcome to the orbital motion tutorials!"
Popup: "Use <WASD> keys to move camera"
Wait for user input
Next popups guiding camera movement
Animated gifs demonstrating concepts
Overlay text explanations
Quiz questions
Callouts guiding UI element use
Popups on clicking menu items
Tasks for user to complete


Options Menu
Graphics
Audio
Controls
Reset to defaults button
Return to main menu button
Resolution dropdown (1280x720, 1920x1080 etc)
Quality preset buttons (Low, Medium, High)
Brightness slider
Master volume slider
Music volume slider
Sound FX volume slider
Mute checkboxes
Mouse sensitivity slider
Invert Y-axis toggle
Camera Control keys
Pause key

Data Structure Design

Objects

Attribute	Data Type	Description	Justification
Name	String	Name of celestial body	Providing a name for each celestial body is useful for identifying and distinguishing them within the simulation. An alternative approach could be using the planets graphics only in which the users can determine what a planet is from what it looks like however this could cause confusion.
Mass	float	Mass in kilograms	The mass of a celestial body is a crucial parameter needed to accurately calculate gravitational forces and simulate orbital motion. There is no alternative to including this attribute, as mass is a fundamental property in physics.
Radius	float	Radius in meters	The radius of a celestial body is necessary for collision detection of the camera and the realistic rendering of planets. An alternative approach could be using very simplified shapes like squares instead of spheres but this would be very inaccurate as there is no realistic and educational material.
Density	float	Density in kg/m^3	Including density as an attribute is important for the realism of the simulation, as it affects the mass thus the gravitational interactions between the celestial bodies. An alternative could be using a constant density for all bodies, but varying densities reflect the

			variety of objects in the solar system better.
Sideral Year	float	Time taken for object to orbit the sun with respect to the fixed star	This attribute is essential for accurately simulating the orbital period of a celestial body around the sun. There is no viable alternative, as orbital periods are a fundamental aspect of the simulation.
Rotation Period	float	Rotation period in days	Including rotation periods allows for realistic rotational motion of celestial bodies, enhancing the visual experience and educational value of the simulation. An alternative could be using static, non-rotating bodies, but this would be less realistic and engaging.
Semi major axis	float	Longest radius of an ellipse	Together with the semi-minor axis, these attributes define the shape and size of a celestial body's orbital path. There is no alternative to including these attributes, as they are fundamental parameters for describing elliptical orbits.
Semi minor axis	float	Shortest radius of an ellipse	*Mentioned in semi major*
Inclination	float	Angle of orbital plane	Having the inclination angle would not affect how the simulation works but will effect the accuracy, not including the inclination angle will be very stupid as it goes against the problem I'm trying to solve as a whole as it is the main orbital characteristic that would want to be seen in a 3D orbit
eccentricity	float	Shape of orbits variation from a perfect circle	Eccentricity is a crucial parameter for determining the shape of a celestial body's orbit. There is no alternative to including this attribute for an accurate simulation of orbital motion.

Mean Anomaly	float	The angle between an elliptical orbit's current position and the line between periapsis and sun	This attribute is essential for accurately calculating the orbital positions of celestial bodies at any given time. There is no alternative to including this parameter for a precise simulation of orbital motion.
Axial Tilt	float	Axial tilt in degrees	Including axial tilt allows for realistic simulation of seasonal rotations, enhancing the educational value and visual appeal of the simulation. An alternative could be keeping the axis static but this would make the simulation less accurate and engaging.

Celestial Body Class

Method	**Description**	**Justification**
CalculateGravity()	Calculates gravitational force vector	Calculating the gravitational forces acting on celestial bodies is essential for simulating their orbital motion accurately. There is no alternative to including this method, as gravity is the fundamental force behind orbital mechanics.
addGravity()	Updates acceleration from gravity	This method is necessary for applying the calculated gravitational forces to update the acceleration of celestial bodies. An alternative approach could be combining this function with the update velocity one, but separating the calculations leads to code modularity and maintainability.
UpdateVelocity()	Updates velocity vector based on acceleration	Updating the velocity based on the calculated acceleration is needed to simulate the motion of celestial bodies. There is no alternative to including this method for an accurate physics simulation.
UpdatePosition()	Updates world position from velocity	After updating the velocity, this method is required to update the position of celestial bodies based on their new velocity vectors. There is no alternative, as updating positions is fundamental to simulating motion.

drawPath()	Renders the orbital path of planet	Rendering the orbital paths of celestial bodies is a necessary feature for an educational simulation, as it helps visualize and understand their motion. An alternative could be not including orbital paths, but this would reduce the simulation's educational purpose.
eccentricAnomaly()	Calculates eccentric anomaly using Kepler's laws and Newtons methods	Calculating the eccentric anomaly is necessary for determining the position of celestial bodies in elliptical orbits. There is no alternative to including this method for accurately simulating elliptical orbital motion.
True Anomaly	Finds true anomaly by using eccentric anomaly	This method is required to convert the eccentric anomaly into the true anomaly, which represents the actual position angle in an elliptical orbit. There is no alternative to including this step for accurate elliptical orbit simulations.

Simulation Class

Variable	Type	Description	Justification
celestialBodies[]	array	Array of bodies in simulation	Storing all celestial bodies in an array is a necessary, as it allows iteratively access different objects in the simulation. An alternative could be using a different data structure, but arrays are a simple and well understood choice by me for this purpose.
startTime	float	Simulation start time in seconds	Having a defined start time is important to ensure consistent behaviour across different runs of the simulation, and it is also needed to have a reference point in my physics calculations, like initial velocity, which is something needed in motion with constant acceleration. There is no alternative approach as it is needed to intiliase orbits.
timeStep	float	Physics timestep in seconds	The timestep will control the precision and performance of the simulation. A smaller timestep increases accuracy but requires more computations. An alternative could be using a dynamic

			timestep which changes for different bodies, for example Pluto which is a much larger orbit in terms of size and time taken to complete could have a very small timestep to make it more accurate, however this would make my simulation open to a lot of errors in terms of time and will be very hard to implement
timeElapsed	float	Current elapsed time	Tracking the elapsed time is fundamental for updating the positions and velocities of celestial bodies in the simulation. There is no alternative to including this variable for an accurate simulation of motion over time.
mainCamera	Camera	Main scene camera	Having a main camera is necessary for rendering the simulation scene from a specific viewpoint. An alternative could be using multiple cameras for multiple views but this would be very resource intensive.

Method	**Description**	**Justification**
UpdateBodies()	Updates all body positions/velocities	Updating the positions and velocities of all celestial bodies is a fundamental requirement for the simulation to work. There is no alternative to including this method, as it is the core mechanism for an orbital motion simulation.
DateCalc()	Presents Date in DD/MM/YY 00:00:00 format	Displaying the current date in a readable format is a useful feature for an educational simulation, as it helps users understand the timeline of the simulation. An alternative approach could be displaying the date in a different format or removing it entirely, but this format is the one used in most countries so is the most user-friendly choice.
IsLeapYear()	Returns correct value for year to match Gregorian calendar	Handling leap years correctly is necessary for accurately simulating celestial body positions over long periods of time. An alternative could be ignoring leap years, but this would load to inaccuracies in the simulation.

Option Menu

Variable	Type	Description	Justification
resolution	Resolution	Stores the current resolution	Allowing users to select the display resolution is an essential feature, as it ensures compatibility with different screen sizes and resolutions. An alternative approach could be using a fixed resolution, but providing options means the simulation can be used on different types if displays and can be viewed to the users preference.
quality	int	Stores the current graphics quality level 0-3	Providing different graphics quality levels allows users to adjust the simulations performance based on the hardware capability of their device. An alternative could be using a fixed quality level, but offering options improves accessibility and user experience.
volume	float	Stores the value for the overall volume of the simulation 0-1	Including a master volume control is a allows users to adjust the overall sound level to their preference. Alternatively, I can remove this and keep separate sound controls however this would create an extra step for a user that wants no audio output at all
musicVolume	float	Music volume 0-1	Having a sperate volume slider allows users to control the balance between background music and other audio elements. An alternative could be relying solely on the master volume, but having individual controls makes it more flexibility.
sfxVolume	float	Sound effects volume 0-1	Same for music volume

Method	Description	Justification
SetResolution()	Changes display resolution	Providing methods to change these simulation options is the necessary approach, as it allows users to select the options that suits their hardware and preferences. I don't think there is an alternative to changing these settings without using methods.
SetQuality()	Changes graphics quality preset	
SetVolume()	Sets master volume	

SetMusicVolume()	Sets music volume	
SetSFXVolume()	Sets sfx volume	
SaveOptions()	Saves options to file	Implementing a method to save the users setting Is necessary to provide a consistent experience across multiple sessions. An alternative could be resetting options on each launch, but having persistent user preferences is a more user-friendly approach.
LoadOptions()	Loads options from file	A method to load saved user settings is required to restore the simulation to the user's preferences at launch. An alternative could be making the user manually load their saved settings, but automatically loading settings saves them time and makes launching the simulation less of a hassle.

Camera Class

Attribute	Type	Description	Justification
transform	Transform	Position, rotation, scale, angle	The transform attribute is needed to change the position of the camera. There is no alternative to including this attribute.
fieldOfView	float	Field of view in degrees, the higher the more the user can see	The field of view attribute allows users to change of much of the around them they can see. An alternative could be using a fixed field of view, but providing a variable that is changeale can mean the user can customise the simulation to their needs.
nearClip	float	Makes objects too close to the camera disappear	The near clipping plane removes objects that are too close to the camera, preventing the camera being blocked and improving performance. An alternative could be rendering all objects regardless of distance, but this could make the simulation unusable in some cases, for example when the camera is blocked, or when the simulation crashes because there are too many objects to render.
farClip	float	Camera doesn't recognise object too far to see.	The far clipping plane removes objects that are too far away to be seen. This helps improve performance by avoiding unnecessary rendering of objects that are too small to be recognised. An alternative could be rendering all objects regardless of

			distance, but this would make the simulation very hard to run.

Method	Description	Justification
SetPosition()	Sets world position	Providing a method to set the camera's world position is a fundamental requirement for any camera system, as it allows for positioning the camera's viewpoint within the scene. An alternative could be relying solely on transforms or manual positioning, but a dedicated method for setting the position is a more direct and user-friendly approach.
SetRotation()	Sets world rotation	Similar to the position, a method for setting the camera's rotation is necessary for orienting the camera's viewpoint in the desired direction. An alternative could be relying on transforms or manual rotation, but a dedicated method offers a more straightforward way to control the camera's orientation.
LookAt()	Rotates camera to look at point	The LookAt method is a valuable feature for camera systems, as it allows for automatically aiming the camera at a specific target point within the scene. An alternative could be manually adjusting the rotation to achieve the desired view, but the LookAt method provides a more convenient and intuitive way to orient the camera.
Zoom()	Changes field of view	Providing a method to change the camera's field of view is a crucial feature, as it allows for adjusting the visible viewing angle and effectively zooming in or out. An alternative could be using a fixed field of view, but offering control over this parameter through a dedicated method provides more flexibility for different use cases and user preferences.

Tutorials Menu Class

Attribute	Type	Description	Justification
scenes	List<TutorialScene>	List of tutorial scenes	Having a list of tutorial scenes is a necessary to have a flow in the tutorial. An alternative could be using a different data structure but they don't provide any additional benefits so no point.

completedScenes	List<TutorialScene>	List of completed scenes	Having separate list of completed tutorial scenes is a allows for tracking of the user's progress. An alternative is not storing whether the user has completed a scene or not but his would mean when the user relaunches the program they would have to start the tutorial back from the beginning.

Method	Description	Justification
NextScene()	Advances to next tutorial scene	A method to load the next tutorial scene is necessary to guide the user throughout the tutorial in a structured manner. An alternative could be relying on the user manually going to the next scene however this can waste their time as there would be breaks inbertween scenes
ResetProgress()	Resets tutorial progress	A method to reset the tutorial progress allows users to restart the tutorial from the beginning or review content they may have missed. An alternative could be making the tutorial a one-time option, but there isn't any benefit to doing this.

Key Variables

These are the main variables that will be used for the 3D orbits:

Variable	Data Type	How It's Used
celestialBodies[]	array	Stores all celestial bodies to be simulated so that they can be access by methodS which construct the orbits
startTime	float	Records simulation start time for time based calculations and the time bar shown at the button of the screen
timeStep	float	Defines how fast the simulation will be, this is used In live calculations to change the rate at which the simulation renders planets.

timeElapsed	float	Tracks current time minus startTime so that the elapsed time can be used in time based calculations
mainCamera	Camera	Used so that methods can access the camera so that zoom/tracking controls can be added
Semi major axis	float	Used in calculations
Semi minor axis	float	Used in calculations
Inclination	float	Sets the orientation of the orbital plane of the celeStial bodies
eccentricity	float	Specifies how circular the orbit, used in calculation to change how this is shown in the simulation
Name	String	Used to reference specific planets in code. Useful for planets with other celestial bodies attached to it like Earth's moon and Saturn's rings
Mass	float	Used to calculate the gravitational forces
Radius	float	Used to construct the orbits

Subroutines

Below, I have thought ahead by creating pseudocode for tricky physics calculations that would've normally taken a lot of time to construct on the spot.

addGravity()

This procedure will be used to change the value of the net force any given celestial object is experiencing. It first finds the distance between the centres of the two objects, squares it, finds its direction and plugs it into $F = \frac{Gm_1m_2}{r^2}$. To get the force. The force is then added to the previous net force.

```
Procedure addGravity(attractor,attractee)

    massProduct = attractor.mass*attractee.mass
    difference = attractor.position - attractee.position
    distance = difference.magnitude
    force = (G*massProduct)/distance**2
    attractee.addForce(force)

End Procedure
```

Figure 4: Gravity Subroutine

The velocity vector is then updated by multiplying force by time and dividing by mass in the addForce() function, because Newtons 2nd Law states that the rate in change in momentum is the net force on an object, or in other words $F = ma$ so dividing the force by the mass gives the acceleration and multiplying it by time will give you the speed

This function would need to be used for every object in a system, and in a system, every single object has a gravitational attraction between one another, so using this function would be tricky when handling multi body celestial orbits. Therefore, an approach is creating a gravity handler which I will talk about later.

addForce()

This procedure will add the force to an object and change the velocity of the object based on that force, it does this by using newtons second law of motion to find acceleration by dividing the force by mass, then it multiples that acceleration by the timestep to get the change in velocity in that time This change in velocity can then be added onto attractees.

```
1 Procedure addForce(netForce, mass, timestep)
2
3     acceleration = netForce / mass
4
5     velocity += acceleration * timestep
6
7     return velocity
8
9 End Procedure
```

Figure 5: addForce function

Without the addForce()
method the gravitational forces will not be able to change the celestial bodies motions at all. As velocity would not be calculating meaning the objects new position cant be added on.

Validation

Velocity now being a part of the physical calculations means that kinetic energy can now be calculated if we already have mass. To check whether these velocity calculations are correct or not I can take advantage of the law of conservation of energy to see if the total kinetic energy in the system remains the same throughout the system. If it does it would mean that the velocities are all correct as no planets would be moving slower or faster than normal meaning no energy is destroyed or created within the simulation.

Here is what the psuedocde validation function looks like.

```
Function checkEnergyConservation()

    // Calculate total kinetic energy
    kineticEnergy = 0
    for each body in simulation:
        kineticEnergy += 0.5 * body.mass * body.velocity.magnitude^2

    // Calculate total potential energy
    potentialEnergy = 0
    for each body in simulation:
        for each otherBody in simulation:
            if body != otherBody:
                r = distance(body, otherBody)
                potentialEnergy += -G * body.mass * otherBody.mass / r

    // Compare current & initial total mechanical energy
    currentTotal = kineticEnergy + potentialEnergy
    initialTotal = getInitialSystemEnergy()
    energyDifference = abs(currentTotal - initialTotal)

    // Allowed error tolerance
    epsilon = 0.01

    // Check if difference exceeds threshold
    if (energyDifference > epsilon)
        print("Warning: System Energy Not Conserved")

    else
        print("System Energy Conserved")

End Function
```

Figure 6: Validation function

It creates a variable for the total kinetic energy and potential energy in the system and checks for each body in the simulation the kinetic and potential energy it has, and adds it to this total. This total is then compared to the initial total energy and it checks if that take away the calculated energy is zero with a tolerance of 0.01. If the energy difference is more than the tolerance the system does not pass validation as it breaks the laws of physics because it suggests that energy has been created of destroyed.

Alternatively, I could not validate, but this would mean that I don't know whether my simulation is in line with the laws of physics. Which would mean my simulation is inaccurate.

Gravity Handler

This procedure will iterate through all celestial bodies in the simulation and add to the velocity vector of every body's interaction with every other body using the addGravityForce() preocedure. Every body in space is an attractor and an attractee even if their gravitational force is very minute, so doing this for even small objects increases the accuracy and realism of the simulation, but makes it harder to implement and more resource dependant. I think this is worth doing even though the smaller objects may not have much of an effect on the motion of the larger ones However, having this code implemented will be very useful when creating custom orbital systems.

```
class GravityHandler
  G = 6.67428*10**-11
  public static attractees = new List(Rigidbody())
  public static attractors = new List(Rigidbody())
  public static isSimulatingLive = true
  Procedure fixedUpdate():
    if isSimulatingLive == true
      Call simulateGravities()
    endif
  End Procedure
  Procedure simulateGravities()
      for attractor in attractors
         for attractee in attractees
           if attractor != attractee
             Call addGravityForce(attractor,attractee)
           endif
         next attractee
      next attractor
  End Procedure
```

Figure 7: Gravity Handler

The class has a list of all celestial bodies in the simulation, which is public to make it accessible from other parts of the program. So, it iterates through the lists of attractors and attractees using a nested loop to run the addGravity() procedure on all of them making sure the attractor/attractee is not the same to avoid self-interaction

Anomaly Functions

The pull of gravity from different celestial bodies can change the shape of a bodies orbit, causing it to stretch, this makes some orbits elliptical. Eccentricity measures how much the shape of an orbit deviates from a circular one. Having an eccentric orbit causes the body to have non uniform circular motion meaning the speed of the body changes at different point in the orbit. Calculating the speed and position of the actual body is very hard compared to just find the shape of the orbit. I could remove some anomaly functions like eccentric anomaly and make it static but I want to make my simulation as realistic as possible meaning I want to calculate the bodies actual position instead of estimating it using different functions.

MeanAnomaly()

First, we must calculate the mean anomaly. This is the angle made between the orbit's centre, and a point on an imaginary circle with the same radius as the semi major axis.
This is the point where an object would be if the orbit was perfectly circular and the orbiting object was moving at constant speed. It is calculated with the equation $M = M_0 + n(t - t_0)$.

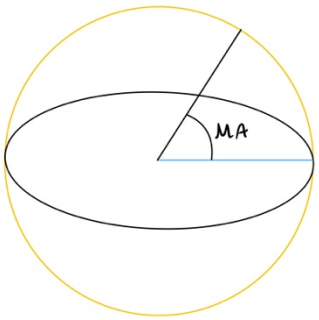


Figure 8: Mean Anomlay Diagram

```
private procedure MeanAnomaly() {
    _meanAnomaly = _meanAnomaly + (_angularVelocity * deltaTime)
    //Checks if angle is in range of 0 to 360 degrees if not subtracts 360 till it is
    if _meanAnomaly > 2*PI:
        _meanAnomaly = _meanAnomaly - (2*PI)
}
```

Figure 9: Mean Anomaly Procedure

I've made sure to have remove any multiples of 2pi to keep the angle in range for a small circle to avoid any errors when simulating eccentric orbits. If I didn't do this the mean anomaly will be calculated wrong as the value that will be given will be invalid as it wouldn't be in range with the angles possible
The mean anomaly function is vital to simulate orbits, there is no over alternative which could produce a 3d orbit.

EccentricAnomaly()

Eccentric Anomaly is defined as the angle between the objects position, the centre of the ellipse, and the point on a circle, with the radius being the semi major axis of the ellipse. This point on the circle is the point directly above the object on an elliptical orbit.
The relationship between the eccentric anomaly and mean anomaly is given by: $M = E - e \sin E$. Solving for Kepler's equation requires numerical methods/iterative techniques, as it is not simple to find a solution when E is used twice in the equation.

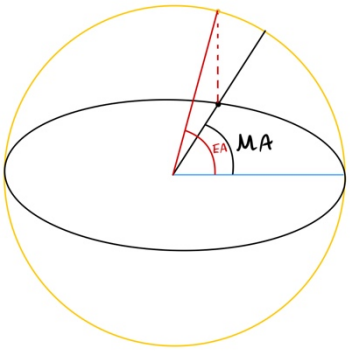


Figure 10: Eccentric Anomaly Diagram

So, I will use newtons method to approximate the eccentric anomaly, the equation for this is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

When applying it to find the eccentric anomaly we get the equation:

$$E_{i+1} = E_i - \left(\frac{E_i - M + e\sin(E_i)}{-1 + e\cos(E_i)}\right)$$

```
function EccentricAnomaly(meanAnomaly, eccentricity, maxIterations = 100)
    h = 0.0001 // Step size for approximating gradient of the function
    acceptableError = 0.00000001
    guess = meanAnomaly

    for i = 0 to maxIterations
        y = KeplerEquation(guess, meanAnomaly, eccentricity)

        if Abs(y) < acceptableError
            break

        slope = (KeplerEquation(guess + h, meanAnomaly, eccentricity) - y) / h
        step = y / slope
        guess = guess - step
    end for

    return guess

function KeplerEquation(E, M, e)
    // Here the equation has been rearranged. We're trying to find the value for E where this will

    return M - E + e * Sin(E)
```

Figure 11: Eccentric Anomaly Function

This function is an implementation of newtons method. It works by declaring a step size to approximate the gradient of the function, I chose 0.0001 as it is the most sensible value that can be used without compromising time, used resources or precision in unequal amounts. Then I declared the uncertainty of my guess for E as 0.0000001 this value allows for an extremely precise value of E sensible for a simulation. The for loop then iterates through different values of E with step size of h to see which iteration returns an answer closest to zero, then outputs the guess for E.

The eccentric anomaly function works with together with other function to provide a 3d orbit that is eccentric. Not having this function would mean my orbits won't be eccentric which would beat the purpose of my simulation as it will no longer be accurate because there is no other way to implement eccentricity.

TrueAnomaly()

Now that we can determine values for eccentric anomaly, we can create a function for the true anomaly. The true anomaly of an object is the angle between the current position of the object, the focus of the ellipse, and the periapsis. It is calculated using the equation:

$$\tan v = \frac{\sqrt{1-e^2}\sin E}{\cos E - e}$$

```
function TrueAnomaly(eccentricity, eccentricAnomaly)
    x = SquareRoot(1 - eccentricity**2)) * Sine(eccentricAnomaly)
    y = Cosine(eccentricAnomaly) - eccentricity
    return ArcTangent2(x, y)
```

Figure 12: true Anomaly Function

Here I have chosen to use the arctan2 function. The arctan2 function is specifically designed to address the issue of quadrants and division by zero. Whereas the tan function doesn't consider quadrants or negative signs which can lead to incorrect values when dealing this the cartesian coordinate system. Therefore, I chose arctan to get more precise values and to avoid errors in my calculations for the true anomaly.

How the algorithms form a complete solution to my problem

These methods and classes will help form the foundations of my simulation. The addGravity() method is essential for calculating the gravitational force between two celestial bodies based on their masses and the distance between them, using Newton's law of universal gravitation. This calculation allows for there to be orbital motion by providing the accelerations need to produce curved paths. Without computing and applying these gravitational forces, there would be no central force causing the orbits, thus no orbital motion to simulate.

The gravity handler iterates these orbital calculations for every pair of orbital bodies because in multi body systems, every body has an effect on every other body even if it is a tiny tug and these effects have to be put into account for an accurate solution.

The addForce() function transforms the accelerations calculated for every body in the gravity handler to changes in velocity using newtons second law. This is done so that a change in the orbits can be seen as the accelerations aren't the quantities that move the bodies it is the velocity which is changed by the acceleration.

So far the necessary functions to have moving orbits have been formed and justified, but orbits aren't circular, these functions assume that they are and that the orbits have a constant acceleration, to have these orbits more realistic I need to implement planetary motion with eccentricity, this is done using Keplers laws instead of newtons.

Newtonian mechanics don't work with ellipses, and inputting an eccentricity alone isn't enough to have a non circular orbit. I have to find by what angle the orbit changes with time. So, I first created the mean anomaly function to find the eccentric anomaly the mean anomaly is used to iteratively approximate the eccentric anomaly based on the eccencricity this anomaly is then used to find the true anomaly which is the actual angle the orbit sweeps with time, this angle changes in different intervals with time which constructs an elliptical orbit.

Together, these algorithms form a complete solution for an accurate 3D orbital motion simulation by modelling gravitational interactions, acceleration and elliptical motion. Alternatively, I could've used more basic formulas and ignore the existence of elliptical orbit and assume they are all circular, this would've made my simulation less accurate and go against the problem I'm trying to solve. I could've removed the gravity handler as a whole and make body interactions only based between the sun and the body instead of other bodies orbiting the sun, this would've made my simulation less resources intensive but the trade-off would be my simulations accuracy.

Also, having the total energy in my system validated is the most robust method of checking whether the velocities, thus the orbits are correct. Any other method wouldn't have been effective as there is no physical law easier to prove than the law of conservation of energy.

Testing Methodology

In Development Test Plan:

I would have to test every function in my simulation to avoid bugs. Doing this whilst developing my game will make finding a problem in the future easier rather than testing it at the end when I finished my simulation.

Options Menu:

Test ID	Test Case	Description	Expected Results	Justification
OM-01	Accessing Options Menu	Verify that users can access the Options Menu from the main menu.	Selecting "Options" from the main menu should open the Options Menu.	If there is no access to the options menu it would mean the user cannot change their settings so it is needed.
OM-02	Audio Settings	Test the functionality of audio settings within the Options Menu.	Adjusting audio settings (volume, sound effects, music) should have a noticeable effect on the audio experience.	If the user cannot access the audio settings they cannot control what they hear which can me frustrating
OM-03	Quality Settings	Confirm the availability and functionality of graphics quality settings.	Adjusting graphics settings should impact the visual quality and performance.	If the user cannot access quality settings they cannot change the way the simulation is rendered which may mean they cannot use the simulation at all as their system may not be able to handle a higher graphis setting
OM-04	Resolution Settings	Test the functionality of resolution settings.	User is able to configure the resolution	If the user cannot change the resolution and it is set to a very small value, they may not get the best experience
OM-05	Control Settings	Test the control settings within the Options Menu.	Users should be able to configure control options such as keybindings or controller settings.	If there were no control setting the user cannot modify the control for their preferences which can

				be an accessibility problem.
OM-06	Saving Options	Verify the capability to save configured options.	After adjusting settings, users should be able to save their preferences, which persist across sessions.	If the users settings aren't saved when they relaunch the simulation they would have to change each setting one by one again which can be frustrating for them.
OM-07	Reset to Default	Test the "Reset to Default" option within the Options Menu.	Selecting "Reset to Default" should restore all settings to the default values.	If there were no reset to default and the user made a mistake in their settings they cant go back to the original setting which can be frustrating as they may not be able to remember what it is
OM-08	Applying Changes	Confirm that changes made in the Options Menu are applied immediately.	After adjusting settings, users should see and hear the effects of those changes without needing to restart the application.	If there users changes didn't work then they cannot change the simulation to their needs creating accessibility problems
OM-09	Accessibility Settings	Test accessibility options within the Options Menu.	Users should be able to modify settings related to accessibility, such as text size and contrast.	Having accessibility settings is important for users who cannot use the application the same way as others, this makes the simulation more inclusive which is important in an educational environment
OM-11	Visual Consistency	Check for visual consistency in the Options Menu.	The Options Menu should maintain a consistent visual style with the rest of the application.	Visual consistency makes the simulation look more polished and professionally made which can increase user satisfaction

OM-12	Help and Documentation	Verify the availability of help or documentation within the Options Menu.	Users should be able to access explanations or tooltips for each setting if they require additional information.	Having help and documentation is very important for if there is a bug that needs fixing which developers need to be contacted about or if they user needs further help in using the simulation

Main Menu:

Test ID	Test Case	Description	Expected Results	Justification
MM-01	Launch Application	Verify that the application launches successfully from the main menu.	The application should load without errors, and the main menu should be displayed.	If the application can't be launched the simulation can't be accessed so this test is needed
MM-02	Main Menu Elements	Confirm the presence and functionality of main menu elements (buttons, options, etc.).	All main menu elements are visible and responsive to user interactions.	The main menu elements are needed for the user to interact with the application and launch the simulation so testing this is important
MM-03	Options Menu	Confirm the "Options" menu from the main menu.	Accessing "Options" should lead to settings for adjusting audio, graphics, controls, and other game preferences.	Being able to access the options menu form the main. Menu is important as the suer may want to change certain settings such as graphical settings before launching the simulation
MM-04	Credits and About	Test the "Credits" and "About" sections in the main menu.	Selecting "Credits" should display information about the development team, and "About" should provide	Ensures that users can access information about the application and its developers.

			details about the application.	
MM-05	Exit Game	Verify the "Exit" or "Quit" option in the main menu.	Selecting "Exit" or "Quit" should gracefully close the application.	Being able to exit the program from the main menu is important for when user accidentally launches the program and wants to close it without launching the simulation
MM-06	Main Menu Navigation	Test the navigation flow within the main menu.	Users should be able to move between menu options, submenus, and return to the main menu without issues.	If the main menu is hard to navigate the user may not be able to launch the simulation which can be frustrating for them
MM-07	Visual Consistency	Check for visual consistency and branding in the main menu.	The main menu should maintain a consistent visual style, including colors, fonts, and graphics.	Visual consistency makes the simulation look more polished and professionally made which can increase user satisfaction
MM-8	Accessibility	Assess accessibility features and options in the main menu.	The main menu should provide options for adjusting text size, screen resolution, and other accessibility settings.	Having accessibility settings is important for users who cannot use the application the same way as others, this makes the simulation more inclusive which is important in an educational environment
MM-9	Error Handling	Test error handling and user feedback in the main menu.	In the event of errors or user interactions that can't be completed, the application should display	Ensuring that users can receive clear feedback in case of errors or invalid actions is important for the usability of the simulation.

			informative error messages.	

Simulation

Test ID	Test Case	Test Data	Description	Expected Results	Justification
S-1	Render Body	Sun	Tests to see if you able to render planets	Earth added with correct texture	The ability to render bodies is very important to see orbital characteristics
S-2	Axial Rotation	Sun	Tests to see if the sun rotates on its axis	Sun rotating at specific speed	Axial rotation is important to add realism to the simulation
S-3	Single body test	Mass: 5000 kg, Velocity: 10 m/s	Motion of a single celestial body.	The body should move in a straight line with constant velocity	A single body test ensures that the functions used for the fundamental motion for any celestial body is in working order.
S-4	Zero Inclination Test	Planet: Mass=6e24 kg, Moon: Mass=7e22 kg, Initial positions/velocities	Two celestial bodies (e.g., a planet and its moon) with known masses, initial positions, and velocities, set up with zero inclination.	True if value is from 0-1 non-inclusive False otherwise	This test is a preventative measure for zero errors in calculations
S-5	Gravitational Interactions Test	Earth: Mass=6e24 kg, Moon: Mass=7e22 kg, Initial positions/velocities	Motion of dual celestial bodies test. For example, I can test a well-known orbit like earth and moon rather than made up celestial bodies	Accurate gravitational interaction	This test verifies if the interactions between the planets are accurate as an already know interaction between the earth and moon is being tested.

S-6	Multi body test	Real masses, positions, velocities of solar system planets	Motion of multiple celestial bodies i.e., the solar system with gravitational interaction	Accurate representation of our solar system	Having more than 2 orbital bodies, tests the gravity handler and checks if it is changing the motion vector for each celestial body based on its other interacting bodies
S-7	Edit body mass test	Set earth mass = 3×10^{24}	Change bodies mass to see effects	Moon moves slightly further away due to weaker gravitational pull	Doing this helps check if our functions can handle normal, erroneous, extreme and abnormal data correctly.
S-8	Asteroid Belt Generation	Cube Prefab	Renders an asteroid belt that moves	Asteroid belt is accurately placed and random	Most students don't know that asteroid belts exist so would be a nice quick feature to implement.
S-9	Accurately Represented asteroid belts	N/A	Renders asteroid belts correctly	Renders inner asteroid belt and Kuiper belt	Having a realistic representation of asteroid belt can be educationally beneficial as most student don't have knowledge on asteroid belts.
S-10	Star Generation	Star Data	Generates a star field	Accurate representation of stars around the solar system	Star generation needs to be tested as it adds realism to the simulation instead of their being a plane boring background
S-11	Star Colour	Star Game Objects	Adds colours to stars	Adds colours which represents how stars can be different	Star colours add educational value to the simulation by representing the doppler effect.

				colours e.g. blue/red	
S-12	Constellation	N/A	Constellations are properly mapped	Lines connected to correct stars to give correct shape of constellation	This is needed for accuracy as constellations only exist between specific stars and having them connected to different ones would be incorrect

UI:

Test ID	Test Case	Test Data	Description	Expected Results	Justification
UI-01	Orbital Path	N/A	Shows the path that orbital bodies will take	Orbital paths stay static, move when parameters or camera changes	Doing this is vital for the user to see a change in the orbit without waiting for the simulation to run.
UI-02	Parameter labels	N/A	Displays the orbital parameters based on the focus selected orbits in simulation	Returns orbits name, and other parameters.	This is important as there is no over way for the user to find out the values of the orbital paramters.
UI-03	Accurate Labels	Angular Veloity	Converts the orbital parameters ot the correct unit	Velocity of earth in radians per second	Having the units set to a standard means the user wouldn't need to do any unnecessary conversion if they want to calculate something using the imulation
UI-04	Live Updates	Mean Anomaly	Displays non static orbital parameters new	Mean anomaly changes values	If this wasn't automatically done the user

			values within every frame	throughout the simulation	would constantly have to click on a planet to see new values which would be frustrating and could be an issue for people with accessibility needs.
UI-05	Asteroid Prefabs	Multiple Different prefabs	Allows for there to be different types of asteroids	Renders a variety of asteroids in a belt not just one that looks the same for all asteroids	This makes the asteroid belt look more realistic as not all asteroids in space look the exact same

Physics Calculation:

Test ID	Test Case	Test Data	Description	Expected Results	Justification
PC-01	Radial Distance	a=$1.49598 \times 10^{11}\ m$ e=0.0167 θ=30°	Semi-major axis, eccentricity, true anomaly plugged into $r = \frac{a(1-\varepsilon^2)}{1-\varepsilon\cos\theta}$	Calculated r matches expected which is: $r = 1.52 \times 10^{11}$	Allows you to know the distance of a celestial body from the object it is orbiting. Helps understand how circular motion works and proofs the relationship of velocity and radius in circular motion
PC-02	Orbital period	$G = 6.67430 \times 10^{-11}$ $M = m_1 + m_2 = 1.99 \times 10^{30}$ $a = 1.49598 \times 10^{11}\ m$	Gravitational Field Strength and Mass plugged in: $T = \frac{2\pi a^{\frac{3}{2}}}{\sqrt{GM}}$	The calculated orbital period matches one of the Earth's which is a year	Testing this verifies the time, velocity, and gravity functions. Doing this allows me to test if the interactions between

					different function works.
PC-03	Kepler's 3rd Law	$r_1 = 1.496 \times 1011\ m$ $r_2 = 2.279 \times 1011\ m$ $P_1 = 3.156 \times 107\ s$ $P_2 = 5.928 \times 107\ s$	$\frac{P_2^2}{P_1^2} \propto \left(\frac{r^2}{r^1}\right)^2$	Ratio matches within 10% (It may not be exact due to rounded numbers)	Checks for keplers third law which has to be proven to students using the simu;lation correctly as it is a vital part of their education in orbital motion.
PC-04	Newton's Law of Gravitation	$m_1 = 1.989 \times 10^{30}\ kg$ $m_2 = 5.972 \times 10^{24}\ kg$ $r = 1.49598 \times 10^{11}\ m$ $G = 6.67430 \times 10^{-11}$	Mass of sun, mass of earth, distance from earth to sun plugged in to find gravitational force $F = \frac{Gm_1m_2}{r^2}$	Calculated for matches one between earth and sun so: $F = 3.54 \times 10^{22}\ N$	This is a vital check as it determines how the orbital bodies will behave since gravity is the main force that is keeping them in orbit
PC-05	Escape Velocity	$m = 5.972 \times 10^{24} kg$ $R = 6.371 \times 10^{6} m$	Gravitational constant, mass of earth, and radius of earth to find escape velocity $v_{escape} = \sqrt{2\,G\,\frac{m}{R}}$	The calculated velocity matches the expected value for Earth, which is approximately $1.186 \times 10^4\ ms^{-1}$	This is tested for custom system where the mass of a planet may be too low to stay in the field of gravity causing it to escape
PC-06	Gravitational Potential Energy	$m = 5.972 \times 10^{24} kg$ $M = 1000\ kg$ $r = 6.371 \times 10^{6} m$	Gravitational constant, mass of earth, mass of random object, and distance from centre of earth to find gravitational potential energy	The calculated gravitational potential energy matches the expected value for the given values of mass and distance.	Needed to test if the system is physically accurate as energy should be conserved within any system

			$G.P = -\frac{GMm}{r}$		
PC-07	Conservation of Angular Momentum	$m = 5.972 \times 10^{24} kg$ $v = 3 \times 10^{4} ms^{-1}$ $r = 1.49598 \times 10^{11}\ m$	Mass of earth, distance of earth from sun at specific point, velocity of earth around the sun to find angular momentum $L = mrv$	The value for L from different radius vectors from the centre of the orbit of earth remains constant	Checks if the total momentum changes if so there is a mistake in the simulation as this would imply that energy has been removed or added ot the system
PC-08	Standard Gravitational Parameter	$G = 6.67430 \times 10^{-11}$ $M = 5.972 \times 10^{24}\ kg$	Product of gravitational constant and mass of earth $\mu = GM$	The value for μ for earth will be $3.986004418 \times 10^{14} m^3 s^{-2}$	This is a parameter that needs to be tested as it shows up in nearly all physics calculations
PC-09	Mean Angular Motion	$\mu = 3.99 \times 10^{14} m^3 s^{-2}$ $a = 149.6 \times 10^{9}\ m$	Standard gravitational parameter and semi major axis for mean angular motion a.k.a angular velocity $n = \sqrt{\frac{\mu}{a^3}}$	The value for n for earth will be $\approx 1.99 \times 10^{-7}$	This needs to be tested as it helps test to see if the motion of the celestial bodies are correct
PC-10	Mean Anomaly	$M_0 = 0$ $n = 3.99107 \times 10^{14}$ $t - t_0 = 175{,}200$	Initial mean anomaly, mean motion and time of perigee passage to give mean anomaly at specific time $M = M_0 + n(t - t_0)$	The value for M will be 0.035 radians. For mean anomaly at 15th of July 2023 12:00:00 UTC	This is a very important anomaly as the other two depend on this so having this tested is very important as it determines if the simulation will e accurate as a whole or not

PC-11	Eccentric Anomaly		$M = E - e \sin E$		Having this tested is important for accuracy as orbits are normally eccentric so having this wrong would mean that the eccentricity of the orbits will be shown wrong
PC-12	True Anomaly		$\tan v = \frac{\sqrt{1-e^2}\sin E}{\cos E - e}$		This anomaly determines the actual absolute angle of the orbital body so testing this is like testing to see if the other two anomalies worked.

Camera

Test ID	Test Case	Test Data	Description	Expected Results	Justification
C-1	Camera Initialization	N/A	Tests if the camera is initialized correctly.	Camera is positioned at the default location.	Not having a camera would mean that the user cannot see what is going on in the simulation. There is no other approach to this.
C-2	Camera Orbit Functionality	Target: Sun	Tests if the camera orbits around a celestial body.	Camera orbits the specified body accurately.	This is needed to keep an orbital body in focus. Alternatively, the camera can be static however this would mean the player has to manually change the position of the camera so that they can see celestial bodies closer which can become an

					inconvenience for them.
C-3	Zoom In/Out Functionality	N/A	Tests if the camera can zoom in and out.	Camera zooms in and out smoothly.	This ensures the user can closely analyse orbit, alternatively the camera can be set to one zoom level, however this can be a problem for very large or very small orbit like pluto and mercury
C-4	Camera Reset	N/A	Tests if the camera can be reset to its default position.	Camera returns to the default position.	This ensures that the camera start from the same original position, the sun, every time the simulation relaunches. If I didn't have this I would have to save the last position of camera and put it back into that position when relaunching the game which opens the simulation to more errors
C-5	Camera Limits	Limits: MinDistance=10, MaxDistance=10000	Tests if the camera stays within specified distance limits.	Camera does not go beyond the set limits.	Having limits keeps the simulation intact, if I didn't the user would be able to infinitely zoom out and will not see any of the orbits, this will be hard to find again for them
C-6	Camera Follow Functionality	Target: Earth	Tests if the camera can follow a celestial body.	Camera smoothly follows Earth	This is needed to keep an orbital body in focus. Alternatively, the camera can be static however this would mean the player has to manually change the position of the camera so that they can see celestial bodies closer which can become an inconvenience for them.

C-7	Camera Collision Avoidance	Obstacle: Asteroid, Collision: True	Tests if the camera avoids collisions with obstacles.	Camera adjusts its position to avoid collision.	This is important for when looking at orbits closer to the sun, as a planet orbiting futher may be very close to the camera's positon, this can block the cameras view so the camera would ahve top avoid rendering thios to keep the simulation usable.
C-8	Camera Switching	Target: Moon	Tests if the camera can smoothly switch between celestial bodies.	Camera transitions smoothly to the new target.	This is important for when switching the view between diferent plannet focuses. Having this being snappy will make the ismulation feel less of a 3D one but instead like switching through differet videos, this is mainly for user satisfaction and wouldnt ereally have much of an educational value but i think it is still worth adding.

Control Intentions

Test ID	Test Case	Test Data	Description	Expected Results	Justification
I-1	Pause Simulation	N/A	Tests if the user can pause the simulation.	Simulation pauses, and celestial bodies freeze.	Pausing the simulation is needed so that the users can properly analyse orbits. Another approach to this could be making the simulation

					extremely slow so it looks paused, however this has no benefit over actually pausing the simulation as it just wastes computational resources
I-2	Resume Simulation	N/A	Tests if the user can resume a paused simulation.	Simulation continues from the paused state.	If there is a pause option a resume option is needed so that the user can get back to the simulation after their observations
I-3	Adjust Simulation Speed	Speed: 2x	Tests if the user can adjust the simulation speed.	Simulation runs at the specified speed.	Adjusting the simulation speed helps user as they can see orbital effects quicker, and don't have to wait because they are stuck at a set speed
I-4	Toggle Celestial Body Labels	Labels: On, Labels: Off	Tests if the user can toggle labels for celestial bodies.	Labels are displayed or hidden as per the setting.	This option is beneficial to suer as it can minimise clutter in the view of the orbital characteristics they are trying to observe
I-5	Change Camera Target	Target: Moon	Tests if the user can change the camera's target.	Camera focuses on the specified celestial body.	The user changing the camera target is vital component, as them being limited to one view limits how many orbital characteristics they can view in turn limiting the educational

					benefits of the simulation
I-6	Reset Controls	N/A	Tests if the user can reset all controls to default settings.	All controls are reset to their default values.	Resetting controls gives the user an easier way to go back to the original settings which can be important for when they don't like their changes and forgot what the original settings were
I-7	Monitor Control Input State	Launch simulation	Tests if the system accurately monitors user control input.	Control input state is detected and reflected in the simulation.	The control inputs not being monitored will mean the simulation will not respond to any inputs by the user, this will make the simulation unusable so checking this is important
I-8	Toggle Constellations	Constellations	A constellation is toggled on or off	When a key corresponding to a constellation is pressed, the constellation is turned on or off	This reduces clutter, not all users would want to see lines in-between the stars, this cold clutter the orbital characteristics they are trying to view

Audio:

Test ID	Test Case	Test Data	Description	Expected Results	Justification
A-1	Background Music	N/A	Tests to see if the background music plays	Background music plays	This is not needed, but i think it can be beneficial to keep students engaged as they will be less bored

A-2	Consistent Background Music	N/A	Tests to see if background music runs consistently between all scenes	Isn't interrupted between scene switches	Makes the simulation more consistent, change in music can be triggering to students with special needs.
A-3	SFX Audio	Buttons	Test to see if a click sound is played when a button is pressed	Click sound is played when button is pressed	Having sound effects is an additional verification step for the user so that they can know if they're input has gone through
A-4	Celestial Body Focus SFX Audio	Celestial Bodies	Test to see if a selection sound is played when a celestial body is pressed	Selection sound is played when a celestial body is focused on	Having this informs if the suer has mis clicked a celestial body which will lead them to click closer from the edge of a celestial body
A-5	Master Volume Control	N/A	Tests to see if the user can change the volume of all types of audios centrally	User can change volume	This can be more convenient for users as they can do it all at once, and is important for accessibility for people with different hearing abilities
A-6	SFX Volume Control	N/A	Tests to see if the user can change the volume of the sound effects	User can change sound effect volume	It allows for more customisability as having just one master volume control for all audio types can be frustrating for people with accessibility needs
A-7	Background Music Volume Control	N/A	Tests to see if the user can change the volume of the	User can change background music volume	This can be useful for people who prefer to use the simulation in

			background music		silence but still want to hear button clicks so they know if their clicks are registered

Post Development Testing

Having post development testing on top of in development testing helps find bugs that couldn't be found whilst making the simulation. Doing this is very important because it adds an extra layer of certainty on whether my simulation is robust or not, and it also helps me see how robust the simulation is from the user's perspective rather than just the developers which can help me find bugs that I couldn't spot during development which ultimately prevents bugs being present in the simulation and not being fixed.

Test ID	Test Case	Description	Expected Results	Justification
1	Launch Application	Ensure the application starts correctly from the main menu.	The application should load smoothly without errors, displaying the main menu.	If the application doesn't launch then the program is unusable
2	Start Button	Click the "Start" button on the main menu.	The 3D orbital motion simulation should begin running.	This button is needed to launch the simulation which is the main thing the user is looking for. There is no other alternative
3	Tutorial Button	Select the "Tutorial" button on the main menu.	The tutorial menu should open, listing available tutorial sections.	This is needed to access the tutorial section so users needing extra support can learn, I can integrate a tutorial section within the main simulation but this can be frustrating for users who already know how to use it
4	Tutorial Pop Ups	Navigate through each tutorial section and interact with pop-ups or interactive elements.	Pop-ups and interactive elements should function as intended, providing relevant information or guidance.	These are needed to help educate users, and are a better way than having a long document explaining how to use the simulation

5	Options Menu Button	Click the "Options" button on the main menu.	The options menu should open, showing the available settings categories.	Being able to access the options menu form the main. Menu is important as the suer may want to change certain settings such as graphical settings before launching the simulation
6	Exit Button	Click the "Exit" button on the main menu.	The application should close gracefully without errors or crashes.	Being able to exit the program from the main menu is important for when user accidentally launches the program and wants to close it without launching the simulation
7	Credits and Abouts Button	Click the "About" button on the main menu.	The credits and about information should display correctly.	Ensures that users can access information about the application and its developers. This can be needed for when the user needs to inform the developer of a bug.
8	Keyboard Functionality	Interact with buttons using keyboard not just mouse	The buttons in the simulation should respond to keyboard interactions	Confirms that the user can access with the simulation with a keyboard only set up which is vital for people with accessibility issues
9	Change Audio settings	In the options menu, adjust audio settings like SFX, Master, and Music.	The audio settings should apply correctly, and changes should be audible in the simulation.	Ensures the user has the ability to customize audio settings which is needed as if they can't it can be really frustrating for them to hear music they don't want to hear.
10	Change Control Settings	In the options menu, modify control settings (e.g.,	The control settings should apply correctly, and the simulation	The ability to change control settings is needed for users who have accessibility issues.

		keyboard bindings, mouse sensitivity).	should respond accordingly.	This feature can be removed but will limit the accessibility.
11	Change Graphical Settings	In the options menu, adjust graphical settings (e.g., resolution, quality, FPS).	The graphical settings should apply to the simulation correctly, and the simulation's visuals should update accordingly.	Changing the graphical setting is a feature that needs to be tested as some users rely on this as they may have computers which can handle complicated graphical operations
12	Save Settings	In the options menu, modify settings and click the "Save" button.	The modified settings should save and apply to different launches of the application.	Testing to see if the user can save settings is important for them to relaunch the simulation easily to a setting they prefer
13	Reset To Default	In the options menu, click the "Reset to Default" button.	All settings should revert to their default values.	Not having this can lead to a problem for some users as for example, they can select a setting that is not compatible for their computer and they can forget what setting to go back to making the simulation not usable for them
14	Toggle Sidebar	Within the simulation, the user can toggle the visibility of the sidebar.	The sidebar should appear or disappear based on the toggle action.	The sidebar shows all the celestial body parameters not having it would reduce the educational value of the simulation
15	Toggle Pause Menu	The user can press the paus button to access pause menu	The pause menu should appear or disappear based on the users click, allowing the user to pause or resume the simulation and access other menus.	If there were no pause menu the user can access in the simulation this would mean that they can't exit, and change options once they have entered the simulation which can make the program really frustrating to use.

16	Focus on Body	Within the simulation, the user can focus the view of the camera on a specific celestial body. And see the paramters of that body in the sidebar	The camera should smoothly transition and centre the view on the focused celestial body.	Testing to see if a body can be focused on is important as some users may want to simply track a celestial bodies motion instead of constantly manually moving the camera.
17	Updating Orbital Parameters	Within the simulation the user can see if the orbital parameters are being updated with time appropriately and correctly	Orbital parameters change with time scale	Having the orbital parameter updated is important as if they were static and set to the start time of the simulation then the simulation would be very useless as the user cant quantitively see orbital characteristics.

Development

Main Menu UI

The main menu will the first thing that appears when the user opens the program. Each button will link to other menus and the simulation. When a button is clicked a sound will be played to confirm to the user that it has been clicked this makes the game more accessible as if I didn't have this a user with vision problem such as double vision will not be able it precisely places the pointer over the button even though they can see it. Having a main menu is essential as it prevents inconveniences like the music playing as soon as the simulation starts, when the user actually wants it to be off the whole time while simulation is running. Or, a user wanting to change the graphics of the simulation because their computer will struggle to run it on the default graphics so they would have to change the graphics setting before the simulations starts as It can cause the program to crash before the user even being able to access the options menu to turn the graphics down.

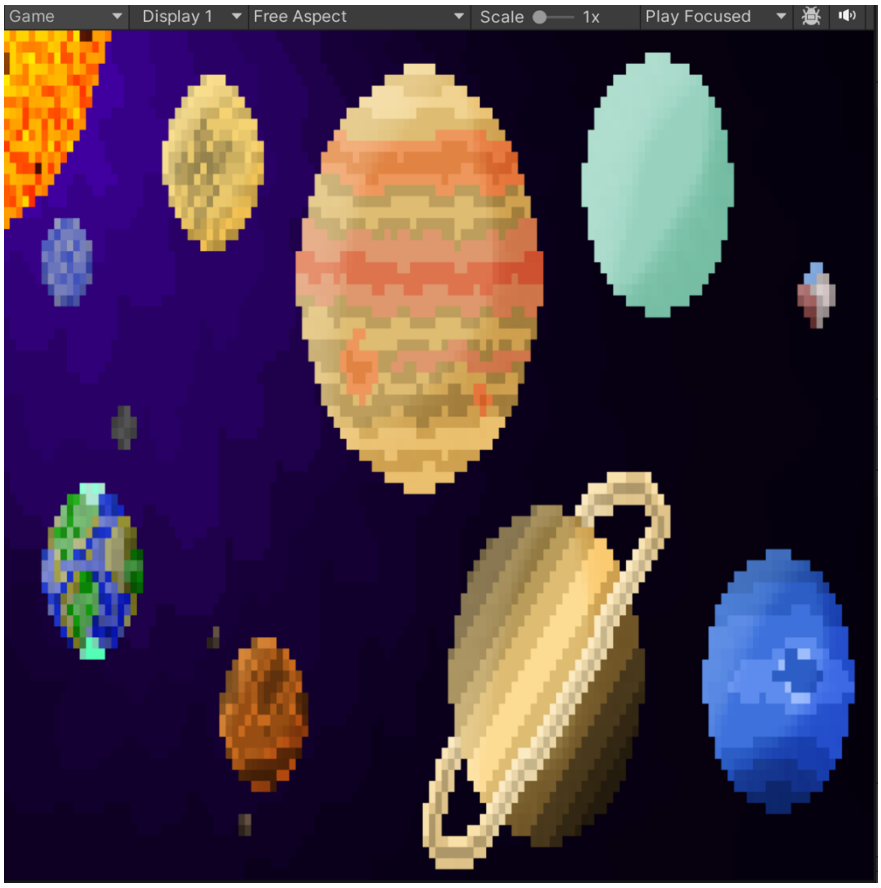


Figure 13: Free Aspect Ratio Background

Background

When trying to add my background image to the scene I realised that when loading the game form different aspect ratios, the background will look squashed making the planets look like ovoid's rather than spheres. This inaccurate and seems inappropriate to have in an educational setting. So, I changed my approach to the main menu design as a whole and made it more suitable for different aspect ratios without compromising any features the original design had.

First, I made sure that the background was significantly bigger than the area the main camera covers so that it can support multiple aspect ratios. Taking this approach means that the original background doesn't have to be adjusted to fit different size screens, rather the section of the background you view changes. I could've taken a different approach where I make the actual simulation fit to any screen by making it a static size and having black bars

in the sides of the screen. However, I chose not to do this as having a full screen experience will make the simulation more engaging and look better to the eye.

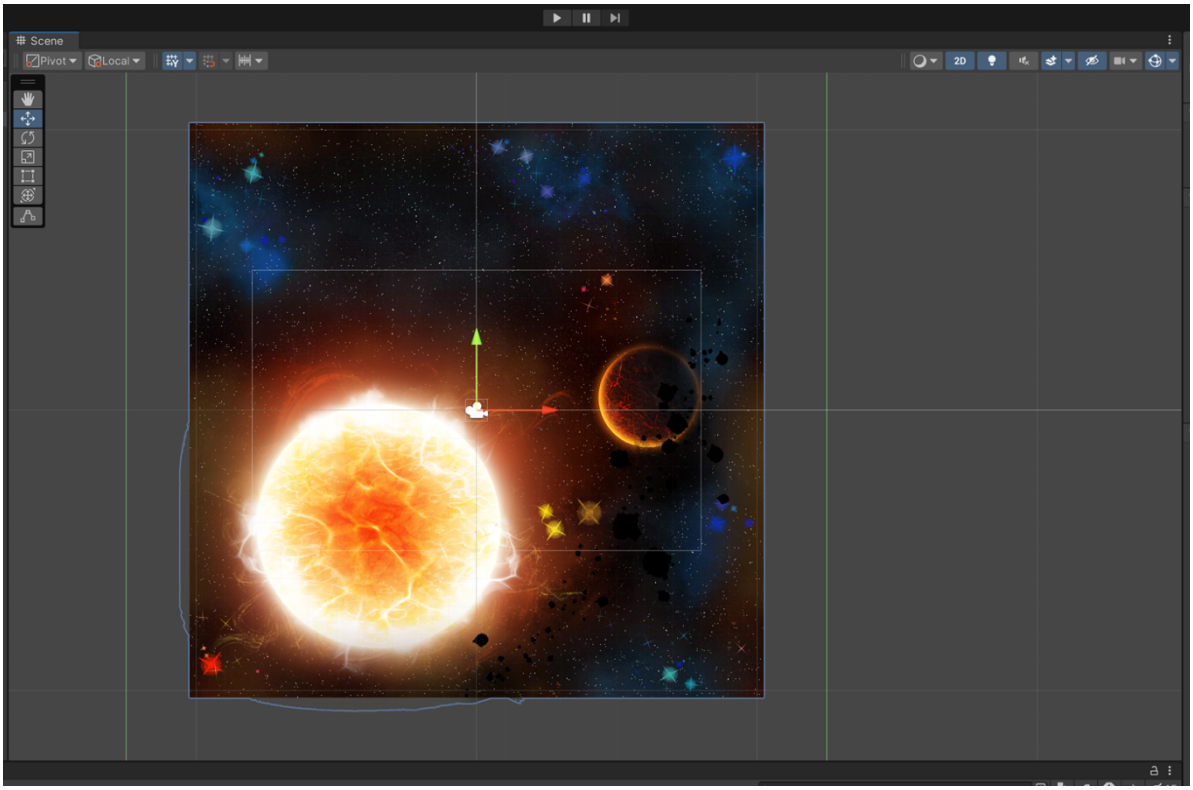


Figure 14: Extended Background

Buttons

Because I changed my background, I had to change the design as a whole, what I opted in for was a less colourful more minimal design. The buttons are highlighted when a mouse is hovered onto it and when it is clicked. I took this approach as it is a beneficial accessibility feature for people with vision problems as they may not be able to see the pointer on the button. It also helps show that the button is actually functional, rather than having no visual response which can leave the user clueless

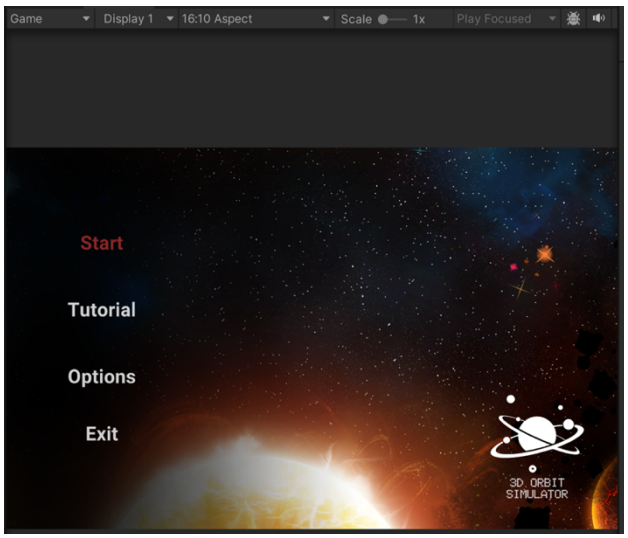


Figure 15: Start Menu Buttons

When the start button is clicked a pop up is displayed presenting two options, a simulation of the solar system and one for a custom system. For the button to be clicked and perform an action I had to create a function that loads the new scene i.e the simulation. I done this in the visual studio. The function PlaySolarSystem() loads the scene of the orbital simulation of the solar system by using the SceneManager class' LoadScene function where a scene is called by searching for its name through a list in the build settings section in unity.

Users > ardasahbaz > Documents > GitHub > Orbital-Motion-NEA > Computer Science NEA > Assets > Scenes > MainMenu.cs > ...

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

0 references
public class MainMenu : MonoBehaviour
{
    0 references
    public void PlaySolarSystem(){
        SceneManager.LoadScene("SolarSystem"); //This function is called by a button to load the scene of a s
    }
    0 references
    public void PlayCustomSystem(){
        SceneManager.LoadScene("CustomSystem"); //This function is called by a button to load the scene of an
    }
    0 references
    public void QuitGame(){
        Application.Quit(); //Called to quir the game
    }
    0 references
    public void OptionsMenu(){
        SceneManager.LoadScene("OptionsMenu"); //Called to load the options menu on the screen
    }
}
```

Figure 16:Main Menu Functions

Another approach I could've taken is to create a more general startScene() function where each button holds the string value of the scene they are going to start and is sent by value to the string a parameter of the startScene() function. However, this approach is much more complicated and will take longer to code, as id have to make every button hold a string value, which will lead to more code, making it less efficient, and adding customisability to what happens when specific scenes open will mean I'd have to create more functions which defeats the purpose of having a reusable component.

Pop Ups

Exit and Title is clicked I made it so that there is a pop up that displays further options. Having pop ups rather than other scenes is much less time and resource consuming to make as loading to a different scene after every button click means id have to create separate

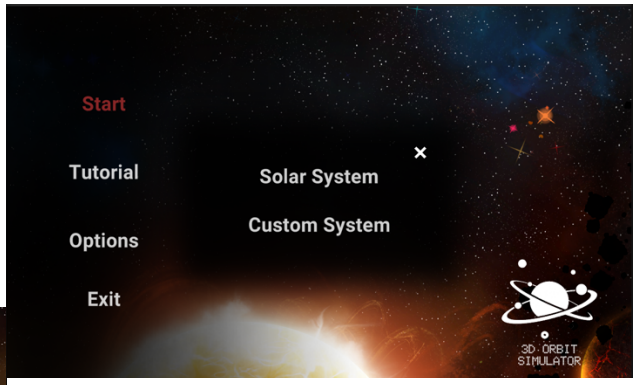


Figure 19: Start Pop Up

Figure 17: About Pop Up

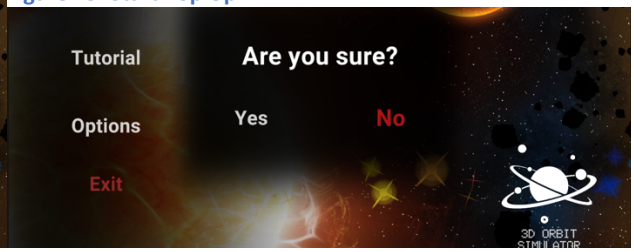


Figure 18: Exit Pop Up

backgrounds and more scenes for every action which would be unnecessary and a waste of time.

For the button in the main menu display a pop up the on click function is nested with the SetActive function in unity which is set to true for it to unhide specific elements in the scene which are the pop ups.

Figure 20: Unity On Cick() Actions Options

Tests

Test ID	Test Case	Result	Passed
MM-01	Launch Application	Application opens to main menu	✓
MM-02	Main Menu Elements	All elements can be interacted with	✓
MM-03	Options Menu	Options button leads to options scene in unity	✓
MM-04	Credits and About	When logo/title is clicked credits and about opens	✓
MM-05	Exit Game	Exit button first asks if sure then leave when user presses yes	✓
MM-06	Main Menu Navigation	Can repetitively go back and forth between pop ups and click buttons using mouse with no glitches. However, I cannot use arrow keys on keyboard.	
MM-07	Visual Consistency	All buttons have same colour when highlighted and pressed	✓
MM-8	Accessibility	Main menu is accessible, simple and not confusing	✓
MM-9	Error Handling	No errors	✓

Test MM-06 Iteration 1

To add keyboard input functionality to my main menu UI I downloaded the Input system package in the unity registry. This allows for navigation of more complicated main menus like mine with pop ups. Keyboard functionality was somewhat working without altering any settings. However, one major problem was you had to pick a button with a mouse first to enable the keyboard functionalities. So, setting deselect on background click to false in unity's Input system script allows for a keyboard only use of the main menu.

Test MM-06 Iteration 2

Using the keyboard for the main buttons in the main menu worked, but for pop ups, it was quite a struggle. When start is selected and entered, I couldn't directly access the buttons in the pop up with the keyboard as the scope for the Event System's Input Module wasn't locked to just the pop-up screen, but it was registering the original buttons behind the screen instead of only the new ones. Which can be seen in this video: https://youtu.be/5VbDMQZJPUg

To combat this issue for every button I changed the navigation settings in unity so that they can only have access to other specific buttons.

I also changed the selected colour of every button in my Main Menu to the same as the highlighted on to help verify that the button is selected when a user used the keyboard, this is vital because if a user with accessibility needs tries to use the keyboard they will not know what button they're clicking and accidentally click the wrong one.

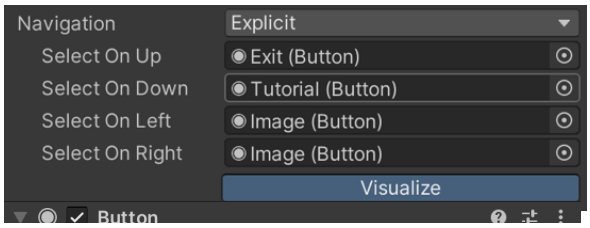


Figure 21: Main Menu Buttons Configurations

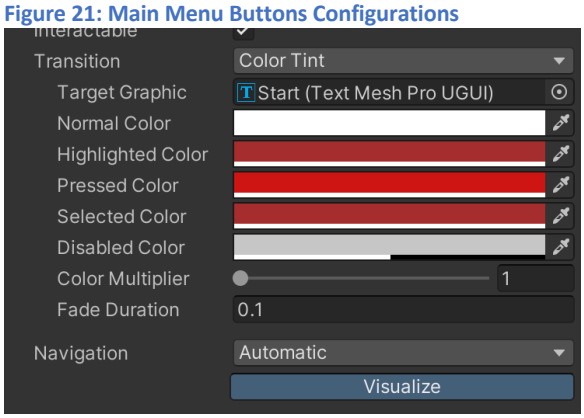


Figure 22: Main Menu Button Configurations

Test MM-06 Iteration 2

Changing the selected colours of the button did help with navigating with a keyboard but because I have pop up the main menu UI couldn't navigate between buttons smoothly. When start is selected and entered, I couldn't directly access the buttons in the pop up with the keyboard as the scope for the Event System's Input Module wasn't locked to just the pop-up screen but it was registering the original buttons behind the screen instead of only the new ones. Which can be seen in this video: https://youtu.be/5VbDMQZJPUg

To try combat this issue for every button I changed the navigation settings in unity so that they can only have access to other specific buttons. This approach did not work, instead I was not able to control the pop-up screens at all.

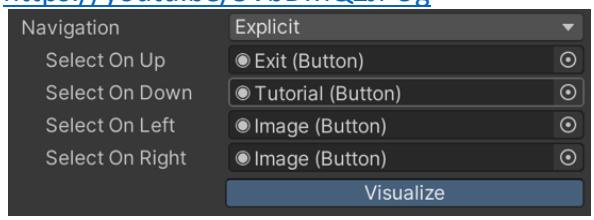


Figure 23: Main Menu Keyboard Navigation Configurator

Test MM-06 Iteration 3

This time i wanted to tackle it with a coding approach. Problems I had to solve were: Keyboard functionality being lost if mouse is used and not being able to control the pop-up menu.

```
public class MainMenu : MonoBehaviour
{
    //Declared as reference to handle keyboard input for buttons in the unity editor
    private Button button;

    // Start is called before the first frame update
    void Start()
    {
        button = GetComponent<Button>();
    }

    // Update is called once per frame to contionously check for any user inputs
    void Update()
    {   //Ensures action is only executed once per key press. So if the user held the key down it would only e

        if (Keyboard.current.enterKey.wasPressedThisFrame && button != null && button.interactable)
        {
            // Triggers the button click when Enter key is pressed
            button.onClick.Invoke();
        }
    }

    //keeps a specific button selected
    public void SetFocus()
    {
        button.Select();
    }

    //references to variables so that they can be assigned values to in the unity editor
    public GameObject startPopUp, exitPopUp, aboutPopUp;
    public Selectable exitPopUpFirstButton, startFirstButton, startPopUpFirstButton, aboutPopUpFirstButton;

    // Opens the start pop-up and sets focus to the first button inside
    public void OpenStartPopUp()
    {
        startPopUp.gameObject.SetActive(true);
        startPopUpFirstButton.Select();
    }

    // Opens the exit pop-up and sets focus to the first button inside
    public void OpenExitPopUp()
    {
        exitPopUp.gameObject.SetActive(true);
        exitPopUpFirstButton.Select();
    }

    // Opens the about pop-up and sets focus to the first button inside
    public void OpenAboutPopUp()
    {
        aboutPopUp.gameObject.SetActive true;
        aboutPopUpFirstButton.Select();
    }
```

Figure 24: Main Menu Class Code

By creating function to set the pop up for certain buttons activate instead of using unity's built in On Click() function in the editor I was able to make the button selected correspond to the buttons in the pop up window when it is displayed. Now the along with the explicit navigation settings selected in unity allowing me to limit the scope of the keyboards access to specific buttons and the instruction set to select a specific first button for each window I have successfully solved the problem of not being able to control the pop-up menu. Alternatively, I could've skipped adding keyboard functionality to my main menu UI but

especially because this simulation is being created for an educational purpose, I would want it to be accessible for all types of people because not every can use a mouse.

Test MM-06 Iteration 4

Although I solved the pop-up menu problem. Keyboard functionality still wasn't perfect. When the user first clicks with a mouse and wants to then use the keyboard, they keyboard doesn't select any button. To fix this I had to create function to close the pop up via code

```
// Closes the start pop up and set the focus to the start button in the main menu
public void CloseStartPopUp()
{
    startPopUp.gameObject.SetActive(false);
    startButton.Select();
}

// Closes the exit pop-up and sets focus to the exit button in the main menu
public void CloseExitPopUp()
{
    exitPopUp.gameObject.SetActive(false);
    exitButton.Select();
}

// Closes the about pop-up and sets focus to the about button in the main menu
public void CloseAboutPopUp()
{
    aboutPopUp.gameObject.SetActive(false);
    aboutButton.Select();
}
```

Figure 25: Main Menu Pop Up Functions

rather than using unity's built in setActive toggles in the editor.

I could've skipped this step as it doesn't serve a lot for people without accessibility needs as they clearly can use a mouse in the first place because they unselected the selected button via keyboard. But doing this makes the program more robust and less annoying for someone to use.

Review

I have constructed a working main menu and have met the first criteria which is "A functioning main menu" and have adhered to the in-development test plan.

In summary, the main menu has keyboard functionality, mouse functionality, working buttons, and is robust.

Changes from my original design

The main menu looks nothing, in terms of UI, like my original design, I opted in for a less childish design as my target users aren't primary school kids. Other than the UI my main menu functions the same way like in my original design.

Solar System Scene

To create a realistic simulation of the solar system I have to create code to calculate many different physical quantities. First, I have to apply newtons gravitational law for every object in the simulation, this will be done using unity's rigid body component for more realistic results. Not using unity's rigid body but instead implementing my own components will be extremely time and resource consuming and will be much less reliable as there is more room for bugs to occur.

Attractor Class

Every object in the universe has a gravitational field around them this means every object interacts with one another gravitationally. To realistically simulate orbital motion, I have to incorporate newtons gravitational force equation to my code to calculate the gravitational force each object experiences and simulate the objects based on their gravitational interactions. Since the solar system is very well studied and we know the orbital path of every object in it, I could just simulate the solar system without calculating quantities such as the velocity, acceleration, etc. since they are all known. However, this will mean that I won't have the ability to change certain parameters of my simulation live in action which is one of the most vital features needed to learn how orbital motion works.

Prototype 1

```
public class Attractor : MonoBehaviour {

    //Gravitational Constant
    const float G = 6.67430e-11;

    public Rigidbody rb;

    //Periodically updates the position of planets.
    void FixedUpdate ()
    {
        //Finds any object in the simulation that is defined as an attractor
        // and applied the asttract function on it with every other attractor
        Attractor[] attarctors = FindObjectsOfType<Attractor>();
        foreach (Attractor attractor in attractors){
            Attract(attractor);
        }
    }

    //Applied Newton's Gravitational Law to the bodies
    void Attract (Attractor objToAttract)
    {
        Rigidbody rbToAttract = objToAttract.rb;
        //Finds distance between the two objects
        Vector3 direction = rb.position - rbToAttract.position;
        float distance = direction.magnitude;
        //Applied newtons equation for gravity
        float forceMagnitude = G * (rb.mass * rbToAttract.mass) / Mathf.Pow(distance, 2);
        Vector3 force = direction.normalized * forceMagnitude;
        rbToAttract.AddForce(force);
    }

}
```

Figure 26: Attractor Class Code

The attractor class uses unity's Rigidbody component which is connected to game objects in the unity editor. It works by periodically updating the position of the object by adding the gravitational force experience by each object periodically. This gravitational force changes within time as the distance between the planets change which makes it necessary to periodically add the new force experienced. I chose the data type for my distance and gravitational constant to be a float instead of a double. Using a double would've made my simulation more precise since it can store up to 64 bits instead of 32. However, unity's vector3 class structure cannot handle the double data type, so I though it will be a sensible compromise because 32 bits is definitely enough for our solar system but for custom systems it may set a boundary to by how much the user can change orbital parameters to avoid rounding to zero which can lead to zero errors.
The attract function is applied to rigid bodies it takes another rigid body in as a parameter and finds the distance between the two. It then inserts this distance into newtons equation for gravity which is $F = \frac{Gm_1m_2}{r^2}$ and adds the force to the vector3 component of the rigidbody which calculates its new position based on the force.

Test

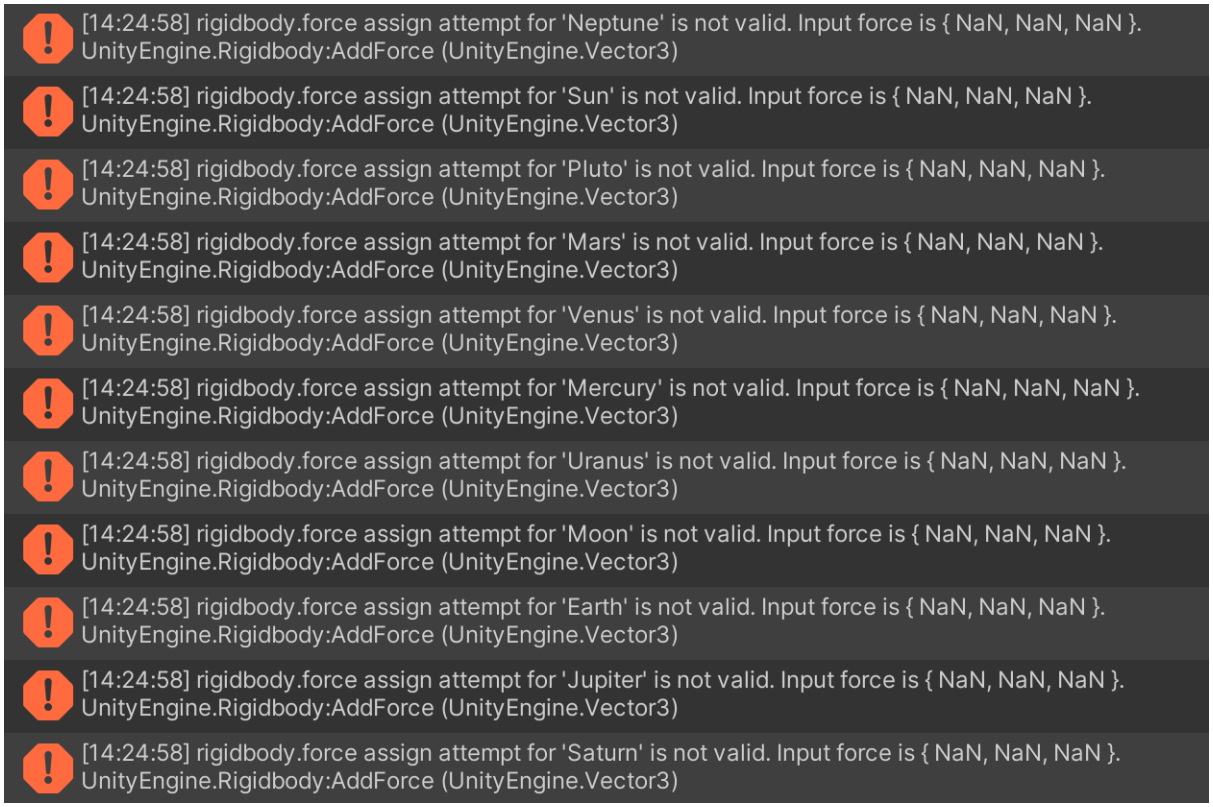


Figure 27: Unity Console Errors

Test ID	Test Case	Test Data	Result	Passed
S-01	Render Body	Sun	Sun is rendered	✓
S-03	Single body test	Mass: 5000 kg, Velocity: 10 m/s	Single body can move	X
S-05	Gravitational Interactions Test	Earth: Mass=6e24 kg, Moon: Mass=7e22 kg, Initial positions/velocities	Accurate motion of celestial bodies	X
PC-04	Newton's Law of Gravitation	$m_1 = 1.989 \times 10^{30}\ kg$ $m_2 = 5.972 \times 10^{24}\ kg$ $r = 1.49598 \times 10^{11}\ m$ $G = 6.67430 \times 10^{-11}$	Equation outputs correct value	X

Prototype 2

The code couldn't successfully simulate gravitational interactions between multiply celestial bodies. This is because of a logical mistake, when looping through the list of attractors to add the force of attraction between them, we also loop through the same attractor that we are trying to add attractive forces to. This means when calculating the distance between the two objects in the attract function we get s value of zero since they are the same objects o they are in the same position. To avoid this, another approach i can to is to add an if statement to the update function which checks if the celestial bodies are the same and if they are it prevents the Attract function from running.

```
//Periodically updates the position of planets.
void FixedUpdate ()
{
    //Finds any object in the simulation that is defined as an attractor
    // and applied the asttract function on it with every other attractor
    Attractor[] attractors = FindObjectsOfType<Attractor>();
    foreach (Attractor attractor in attractors){
        if (attractor != this)
            Attract(attractor);
    }
}
```

Figure 28: Fixed Update Function

Test

Test ID	Test Case	Test Data	Result	Passed
S-01	Render Body	Sun	Sun is rendered	✓
S-03	Single body test	Mass: 5000 kg, Velocity: 10 m/s	Single body can move	✓
S-05	Gravitational Interactions Test	Earth: Mass=6e24 kg, Moon: Mass=7e22 kg, Initial positions/velocities	Accurate motion of celestial bodies	X
PC-04	Newton's Law of Gravitation	$m_1 = 1.989 \times 10^{30}\ kg$ $m_2 = 5.972 \times 10^{24}\ kg$ $r = 1.49598 \times 10^{11}\ m$ $G = 6.67430 \times 10^{-11}$	Equation outputs correct value	✓

Prototype 3

This method of coding the attractor class is not physically accurate. The rigid body class only accepts mass' under $1.0 \times 10^{9} kg$. The sun and other planet are much greater in mass that the limit set. So, to overcome the limitations set, I could use the planetary data with earth ratios. Planetary data based on earth ratios is already given in the NASA website: https://nssdc.gsfc.nasa.gov/planetary/factsheet/

	MERCURY	VENUS	EARTH	MOON	MARS	JUPITER	SATURN	URANUS	NEPTUNE	PLUTO	SUN
Mass	0.0553	0.815	1	0.0123	0.107	317.8	95.2	14.5	17.1	0.0022	333,000
Diameter	0.383	0.949	1	0.2724	0.532	11.21	9.45	4.01	3.88	0.187	109
Density	0.985	0.951	1	0.606	0.714	0.241	0.125	0.230	0.297	0.336	0.255
Gravity	0.378	0.907	1	0.166	0.377	2.36	0.916	0.889	1.12	0.071	27.94
Escape Velocity	0.384	0.926	1	0.213	0.450	5.32	3.17	1.90	2.10	0.116	55.2
Rotation	58.8	-244	1	27.4	1.03	0.415	0.445	-0.720	0.673	6.41	N/A

Period											
Length of Day	175.9	116.8	1	29.5	1.03	0.414	0.444	0.718	0.671	6.39	25.449
Distance from Sun	0.387	0.723	1	0.00257*	1.52	5.20	9.57	19.17	30.18	39.48	N/A
Perihelion	0.313	0.731	1	0.00247*	1.41	5.04	9.23	18.58	30.40	30.16	N/A
Aphelion	0.459	0.716	1	0.00267*	1.64	5.37	9.91	19.73	29.97	48.49	N/A
Orbital Period	0.241	0.615	1	0.0748*	1.88	11.9	29.4	83.7	163.7	247.9	N/A
Orbital Velocity	1.59	1.18	1	0.0343*	0.808	0.439	0.325	0.228	0.182	0.157	7.33 (around milky way)
Orbital Eccentricity	12.3	0.401	1	3.29	5.60	2.93	3.38	2.74	0.677	14.6	0.015
Obliquity to Orbit	0.001	0.113*	1	0.285	1.07	0.134	1.14	4.17*	1.21	2.45*	7.25
Surface Pressure	0	92	1	0	0.01	Unknown*	Unknown*	Unknown*	Unknown*	0.00001	5500
Number of Moons	0	0	1	0	2	92	83	27	14	5	0

Ring System?	No	No	No	No	No	Yes	Yes	Yes	Yes	No	No
Global Magnetic Field?	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	Unknown	N/A

By using this data instead of the actual metric values of earth, I stay in within the boundaries of the mass for the rigid body component without losing any accuracy in my simulation. However, whenever I want to present the data to users of my simulation, I wouldn't give it in earth ratios instead I would calculate the actual values and display it in my UI because it will be very frustrating for students to calculate the actual values of the celestial bodies parameter without having known earth's.

Another approach i could've taken is to use values from the metric system. However, this would mean that there will be very big values, and this would make the simulation much harder to run, also unity caps the parameters passed into it at 3.4x10^38. Some values in astronomy can be even greater than this. So, if I took this approach my simulation will have data missing in turn making it unfunctional.

Celestial Body

I've realised that using a gravity handler class would be very unnecessary.

Using earth ratio's does help escape the rigid body components set boundaries. However, when inserting earth ratios into newtons equation for gravitation, there is no set value to use to write the gravitational constant in terms of earth ratios as the constant is not dependant on earth.

```
public class CelestialBody : MonoBehaviour
{
    public float mass;
    public float radius;
    public Vector3 initialVelocity;
    Vector3 currentVelocity;
    public Rigidbody rb;
    const float G = 6.67430e-11F;
    // When time is 0 current velocity is equal to initial
    void Start(){
        currentVelocity = initialVelocity;
    }

    public void UpdateVelocity (CelestialBody[] allBodies, float timeStep){
        //Iterates through all orbital bodies in the simulation
        foreach (var otherbody in allBodies){
            //Makes sure velocity of an orbital bdoy isnt being updated based on that same body
            if (otherbody != this){
                //Finds the distancd betweemn the two bodies and squares it
                float distanceSqr = (otherbody.rb.position - rb.position).sqrMagnitude;
                //Finds direction of the vector
                Vector3 forceDirection = (otherbody.GetComponent<Rigidbody>().position - rb.position).normalized;
                //Uses newtons law of graviattaion to find the force
                Vector3 force = (forceDirection * G * mass * otherbody.mass) / distanceSqr;
                // Finds accelration because F=ma
                Vector3 acceleration = force / mass;
                //New velocity is added on because acceleration times time is change in velocity
                currentVelocity += (acceleration * timeStep);
            }
        }
    }
    //Updates the position of the orbital bdoy by adding the new calculated velocity
    public void UpdatePosition(float timeStep){
        GetComponent<Rigidbody>().position += currentVelocity * timeStep;
    }
```

Figure 29: Celestial Body Class

To make my simulation work my approach would have to have the value changed for the gravitational constant as the gravitational attractiveness between the celestial bodies will be very weak so a very slow simulation will be presented, but doing this creates a physically inaccurate simulation as I would be changing a physical constant which means the calculations for my simulation will just be false and inaccurate.

What I can do instead is to introduce time to my simulation. By doing this I eliminate the need to change the value for the gravitational constant. As mentioned before newtons equation for the force of gravitation is $F = \frac{Gm_1m_2}{r^2}$ and we know that Newtons second law is defined by $F = ma$ if we divide the force by the mass of the object being attracted, we get acceleration and the mass' cancel out. If we have time, we can easily calculate the change in velocity because: $a = \frac{\Delta v}{t}$ so multiplying acceleration by the timestep we can find the change in velocity and add this change to the rigid body.

If I want to update the velocity of the rigid body, I'd have to change multiple variables of the Vector3 component, so I think it be most appropriate to create a separate class for every celestial body instead of defining them as rigid bodies in the unity editor. Taking this approach will be beneficial in many ways for example it can give modularity to my program, having celestial bodies as modular component means that their parameters can more easily

be changed, this will be very useful when having custom system as a play may want to change the name or texture of a planet. It can also allow for more custom implementations of the celestial bodies, for example a user can change the axial rotation speed of the planet, or number of moons/ rings etc. Which wouldn't be as easy to do without a celestial body class because components like axial tilt is independent from the attractor class, it would be hard to modify from there.

So, I think I should make the gravity handler a function incorporated into each and every celestial body instead of having it as a class will be much more efficient as not every celestial body will have an attractor class attached to it which will be a waste of memory, and also it still allows for reusability of the attractor instructions

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CelestialBody : MonoBehaviour
{
    public CelestialBody[] allBodies;
    public float mass;
    public float radius;
    public Vector3 initialVelocity;
    Vector3 currentVelocity;
    Rigidbody rb;
    Transform meshHolder;
    const float G = 6.6743e-11F;
    public float timeStep = 0.1f;
    // When time is 0 current velocity is equal to initial
    void Start(){
        currentVelocity = initialVelocity;
        rb = GetComponent<Rigidbody> ();
        rb = gameObject.AddComponent<Rigidbody>();
        rb.mass = mass;
    }

    public void UpdateVelocity (CelestialBody[] allBodies){
        //Iterates through all orbital bodies in the simulation
        foreach (var otherbody in allBodies){
            //Makes sure velocity of an orbital bdoy isnt being updated based on that same body
            if (otherbody != this){
                //Finds the distancd betweemn the two bodies and squares it
                float distanceSqr = (otherbody.rb.position - rb.position).sqrMagnitude;
                //Finds direction of the vector
                Vector3 forceDirection = (otherbody.GetComponent<Rigidbody>().position - rb.position).normalized;
                //Uses newtons law of graviattaion to find the force
                Vector3 force = (forceDirection * G * mass * otherbody.mass) / distanceSqr;
                // Finds accelration because F=ma
                Vector3 acceleration = force / mass;
                //New velocity is added on because acceleration times time is change in velocity
                currentVelocity += (acceleration * timeStep);
            }
        }
    }
    //Updates the position of the orbital bdoy by adding the new calculated velocity
    public void UpdatePosition(){
        GetComponent<Rigidbody>().position += currentVelocity * timeStep;
    }
    //Updates the velocity and position of each object within every frame
    void Update(){
    UpdateVelocity(allBodies);
    UpdatePosition();
    }

}
```

Figure 30: Celestial Body Class Updated

The UpdateVelocity function in the new CelestialBody class takes in a parameter that fetches all present celestial bodies in the unity scene and creates an array called allBodies,

and also the time step of the solution. It loops through all bodies in the array and calculates the gravitational force between each and adds the new velocity calculated through the force onto the Vector3 component of the objects making sure that the same object in the list is skipped to avoid finding the force between the same objects. I chose this approach because the force between the same object will be zero which can cause zero error in when performing calculation based on the gravitational force.

Test S-5

Simulation is not moving at all. No reaction to script.

Test ID	Test Case	Test Data	Result	Passed
S-5	Gravitational Interactions Test	Earth: Mass=6e24 kg, Moon: Mass=7e22 kg, Initial positions/velocities	Accurate gravitational interaction	X

S-5 Iteration 1

Executing my current function didn't perform anything. The planets were stationery and were not affected by my script even though I've changed the initial velocity values for each celestial body.

The problem must have been caused by my updatePosition() function. For unity to update the status of the scene it only recognises an update function with the name Update(). So, what I did was nest in the UpdateVelocity() and UpdatePosition() function into the Update() function so that unity can execute it.

```
void Update(){
    UpdateVelocity(allBodies);
    UpdatePosition();
    }
```

Figure 31: Celestial Body Class Update Function

S-5 Iteration 2

The celestial bodies are now moving, but it is not an accurate representation of the solar system. The initial position of the planets hasn't been set; therefore, they are attracted to each other in different ways compared to the solar system. To make my simulation a representation of the solar system I'd have to find data which shows the coordinates of every planet relative to the sun at a specific point in time. Although I've accurately represented most parameters of the celestial bodies like, initial velocity, mass, and distance from the sun, it doesn't necessarily mean that the solar system can be represented the same way straight away as it all depends on the initial conditions of the solar system. Since we don't know the initial conditions for the solar system like where each planet came from and how each planet formed, I wouldn't be able to accurately represent the solar system from the start to the present, so I'd have to pick a point in time to call it the initial existence of the solar system.

First, I had to find reliable data of each planet relative to the sun. So, I found a NASA website: https://ssd.jpl.nasa.gov/horizons/app.html#/ which has an app that generates precise data for nearly very parameter a celestial object can have. Here is an example of Mars relative to the Sun with parameters given in AU and days:

****

```
 Revised: June 21, 2016                Mars                              499
/ 4

 PHYSICAL DATA (updated 2019-Oct-29):
  Vol. mean radius (km) = 3389.92+-0.04   Density (g/cm^3)      =
3.933(5+-4)
  Mass x10^23 (kg)      =    6.4171       Flattening, f         =
1/169.779
  Volume (x10^10 km^3)  =   16.318        Equatorial radius (km)=  3396.19
  Sidereal rot. period  =   24.622962 hr  Sid. rot. rate, rad/s =
0.0000708822
  Mean solar day (sol)  =   88775.24415 s Polar gravity m/s^2   =  3.758
  Core radius (km)      = ~1700           Equ. gravity  m/s^2   =  3.71
  Geometric Albedo      =    0.150

  GM (km^3/s^2)         = 42828.375214    Mass ratio (Sun/Mars) =
3098703.59
  GM 1-sigma (km^3/s^2) = +- 0.00028      Mass of atmosphere, kg= ~ 2.5 x
10^16
  Mean temperature (K)  =  210            Atmos. pressure (bar) =    0.0056
  Obliquity to orbit    =   25.19 deg     Max. angular diam.    =  17.9"
  Mean sidereal orb per =    1.88081578 y Visual mag. V(1,0)    =  -1.52
  Mean sidereal orb per =  686.98 d       Orbital speed,  km/s  =  24.13
  Hill's sphere rad. Rp =  319.8          Escape speed, km/s    =   5.027
                               Perihelion  Aphelion    Mean
  Solar Constant (W/m^2)       717         493         589
  Maximum Planetary IR (W/m^2) 470         315         390
  Minimum Planetary IR (W/m^2)  30          30          30
*******************************************************************************
****


*******************************************************************************
****
Ephemeris / WWW_USER Fri Nov  3 03:18:30 2023 Pasadena, USA      / Horizons
*******************************************************************************
****
Target body name: Mars (499)                      {source: mar097}
Center body name: Sun (10)                        {source: DE441}
Center-site name: (user defined site below)
*******************************************************************************
****
Start time      : A.D. 2000-Jan-01 12:00:00.0000 TDB
Stop  time      : A.D. 2000-Feb-01 12:00:00.0000 TDB
Step-size       : 1440 minutes
*******************************************************************************
****
Center geodetic : 0.0, 0.0, 0.0                   {E-
lon(deg),Lat(deg),Alt(km)}
Center cylindric: 0.0, 6.96E5, 0.0                {E-
lon(deg),Dxy(km),Dz(km)}
Center pole/equ : IAU_SUN                         {East-longitude positive}
Center radii    : 696000.0, 696000.0, 696000.0 km {Equator_a, b, pole_c}
Output units    : AU-D
Calendar mode   : Mixed Julian/Gregorian
Output type     : GEOMETRIC cartesian states
Output format   : 1 (position only)
Reference frame : Ecliptic of J2000.0
*******************************************************************************
****
JDTDB
```

```
   X     Y     Z
*******************************************************************************
****
$$SOE
2451545.000000000 = A.D. 2000-Jan-01 12:00:00.0000 TDB
 X = 1.391416854130100E+00 Y =-1.800992385068067E-02 Z =-
3.469784120300255E-02
2451546.000000000 = A.D. 2000-Jan-02 12:00:00.0000 TDB
 X = 1.393108964513796E+00 Y =-2.504877974901379E-03 Z =-
3.451967823617148E-02
2451547.000000000 = A.D. 2000-Jan-03 12:00:00.0000 TDB
 X = 1.394538721096286E+00 Y = 1.326072941106634E-02 Z =-
3.431608490334090E-02
2451548.000000000 = A.D. 2000-Jan-04 12:00:00.0000 TDB
 X = 1.395646045862153E+00 Y = 2.924998423041104E-02 Z =-
3.408077627885758E-02
2451549.000000000 = A.D. 2000-Jan-05 12:00:00.0000 TDB
 X = 1.396381237557081E+00 Y = 4.541222652734608E-02 Z =-
3.380917537872171E-02
2451550.000000000 = A.D. 2000-Jan-06 12:00:00.0000 TDB
 X = 1.396708011228207E+00 Y = 6.168603854040298E-02 Z =-
3.349869458101577E-02
2451551.000000000 = A.D. 2000-Jan-07 12:00:00.0000 TDB
 X = 1.396605720951333E+00 Y = 7.800288901127811E-02 Z =-
3.314889576884007E-02
2451552.000000000 = A.D. 2000-Jan-08 12:00:00.0000 TDB
 X = 1.396070630205438E+00 Y = 9.429121153230920E-02 Z =-
3.276151943061374E-02
2451553.000000000 = A.D. 2000-Jan-09 12:00:00.0000 TDB
 X = 1.395116152423734E+00 Y = 1.104806682691363E-01 Z =-
3.234038094234621E-02
2451554.000000000 = A.D. 2000-Jan-10 12:00:00.0000 TDB
 X = 1.393772047044398E+00 Y = 1.265063390930267E-01 Z =-
3.189114035502152E-02
2451555.000000000 = A.D. 2000-Jan-11 12:00:00.0000 TDB
 X = 1.392082620074305E+00 Y = 1.423125807063514E-01 Z =-
3.142095972396902E-02
2451556.000000000 = A.D. 2000-Jan-12 12:00:00.0000 TDB
 X = 1.390104038880409E+00 Y = 1.578563204658530E-01 Z =-
3.093806887489386E-02
2451557.000000000 = A.D. 2000-Jan-13 12:00:00.0000 TDB
 X = 1.387900924933172E+00 Y = 1.731095840765297E-01 Z =-
3.045126608414545E-02
2451558.000000000 = A.D. 2000-Jan-14 12:00:00.0000 TDB
 X = 1.385542432252119E+00 Y = 1.880611030446319E-01 Z =-
2.996938411866746E-02
2451559.000000000 = A.D. 2000-Jan-15 12:00:00.0000 TDB
 X = 1.383098050662681E+00 Y = 2.027169038920428E-01 Z =-
2.950075419301809E-02
2451560.000000000 = A.D. 2000-Jan-16 12:00:00.0000 TDB
 X = 1.380633389754280E+00 Y = 2.170998432220484E-01 Z =-
2.905270052850605E-02
2451561.000000000 = A.D. 2000-Jan-17 12:00:00.0000 TDB
 X = 1.378206200607186E+00 Y = 2.312481170022800E-01 Z =-
2.863109633411213E-02
2451562.000000000 = A.D. 2000-Jan-18 12:00:00.0000 TDB
 X = 1.375862877856430E+00 Y = 2.452128349772323E-01 Z =-
2.824000828326415E-02
2451563.000000000 = A.D. 2000-Jan-19 12:00:00.0000 TDB
 X = 1.373635655370582E+00 Y = 2.590548081277677E-01 Z =-
2.788145116371548E-02
```

```
2451564.000000000 = A.D. 2000-Jan-20 12:00:00.0000 TDB
 X = 1.371540666529622E+00 Y = 2.728407450814389E-01 Z =-
2.755526765997789E-02
2451565.000000000 = A.D. 2000-Jan-21 12:00:00.0000 TDB
 X = 1.369576987367680E+00 Y = 2.866390894171729E-01 Z =-
2.725914059843618E-02
2451566.000000000 = A.D. 2000-Jan-22 12:00:00.0000 TDB
 X = 1.367726720915438E+00 Y = 3.005157517029813E-01 Z =-
2.698873690843424E-02
2451567.000000000 = A.D. 2000-Jan-23 12:00:00.0000 TDB
 X = 1.365956117587277E+00 Y = 3.145299965225952E-01 Z =-
2.673797452045117E-02
2451568.000000000 = A.D. 2000-Jan-24 12:00:00.0000 TDB
 X = 1.364217663283558E+00 Y = 3.287307352967744E-01 Z =-
2.649939592579824E-02
2451569.000000000 = A.D. 2000-Jan-25 12:00:00.0000 TDB
 X = 1.362453007871910E+00 Y = 3.431534509626315E-01 Z =-
2.626462561887306E-02
2451570.000000000 = A.D. 2000-Jan-26 12:00:00.0000 TDB
 X = 1.360596555469824E+00 Y = 3.578179420458300E-01 Z =-
2.602488352895159E-02
2451571.000000000 = A.D. 2000-Jan-27 12:00:00.0000 TDB
 X = 1.358579497596869E+00 Y = 3.727270236955415E-01 Z =-
2.577152313462372E-02
2451572.000000000 = A.D. 2000-Jan-28 12:00:00.0000 TDB
 X = 1.356334043259188E+00 Y = 3.878662648998166E-01 Z =-
2.549656144828632E-02
2451573.000000000 = A.D. 2000-Jan-29 12:00:00.0000 TDB
 X = 1.353797588019446E+00 Y = 4.032047779181592E-01 Z =-
2.519316855329575E-02
2451574.000000000 = A.D. 2000-Jan-30 12:00:00.0000 TDB
 X = 1.350916567826481E+00 Y = 4.186970118093159E-01 Z =-
2.485608684292548E-02
2451575.000000000 = A.D. 2000-Jan-31 12:00:00.0000 TDB
 X = 1.347649762601983E+00 Y = 4.342854407053314E-01 Z =-
2.448195439750254E-02
2451576.000000000 = A.D. 2000-Feb-01 12:00:00.0000 TDB
 X = 1.343970848132930E+00 Y = 4.499039829231093E-01 Z =-
2.406951278145403E-02
$$EOE
******************************************************************************
****

TIME

  Barycentric Dynamical Time ("TDB" or T_eph) output was requested. This
continuous coordinate time is equivalent to the relativistic proper time
of a clock at rest in a reference frame co-moving with the solar system
barycenter but outside the system's gravity well. It is the independent
variable in the solar system relativistic equations of motion.

  TDB runs at a uniform rate of one SI second per second and is independent
of irregularities in Earth's rotation.

CALENDAR SYSTEM

  Mixed calendar mode was active such that calendar dates after AD 1582-
Oct-15
(if any) are in the modern Gregorian system. Dates prior to 1582-Oct-5 (if
any)
```

```
are in the Julian calendar system, which is automatically extended for
dates
prior to its adoption on 45-Jan-1 BC.  The Julian calendar is useful for
matching historical dates. The Gregorian calendar more accurately
corresponds
to the Earth's orbital motion and seasons. A "Gregorian-only" calendar mode
is
available if such physical events are the primary interest.

REFERENCE FRAME AND COORDINATES

  Ecliptic at the standard reference epoch

    Reference epoch: J2000.0
    X-Y plane: adopted Earth orbital plane at the reference epoch
               Note: IAU76 obliquity of 84381.448 arcseconds wrt ICRF X-Y
plane
    X-axis   : ICRF
    Z-axis   : perpendicular to the X-Y plane in the directional (+ or -)
sense
               of Earth's north pole at the reference epoch.

  Symbol meaning [1 au= 149597870.700 km, 1 day= 86400.0 s]:

    JDTDB    Julian Day Number, Barycentric Dynamical Time
      X      X-component of position vector (au)
      Y      Y-component of position vector (au)
      Z      Z-component of position vector (au)

ABERRATIONS AND CORRECTIONS

 Geometric state vectors have NO corrections or aberrations applied.

Computations by ...

    Solar System Dynamics Group, Horizons On-Line Ephemeris System
    4800 Oak Grove Drive, Jet Propulsion Laboratory
    Pasadena, CA  91109   USA

    General site: https://ssd.jpl.nasa.gov/
    Mailing list: https://ssd.jpl.nasa.gov/email_list.html
    System news : https://ssd.jpl.nasa.gov/horizons/news.html
    User Guide  : https://ssd.jpl.nasa.gov/horizons/manual.html
    Connect     : browser
https://ssd.jpl.nasa.gov/horizons/app.html#/x
                  API            https://ssd-
api.jpl.nasa.gov/doc/horizons.html
                  command-line   telnet ssd.jpl.nasa.gov 6775
                  e-mail/batch
https://ssd.jpl.nasa.gov/ftp/ssd/hrzn_batch.txt
                  scripts        https://ssd.jpl.nasa.gov/ftp/ssd/SCRIPTS
    Author      : Jon.D.Giorgini@jpl.nasa.gov
*******************************************************************************
****
```

Figure 32: Celestial Body Data

The data above is data for mars relative to the sun at the equinox of j2000 which is the data January 1 2000. This data is very useful because I can take advantage of the extra information it gives and test my physics calculation methods more reliably.

```
$$SOE
```

```
2451545.000000000 = A.D. 2000-Jan-01 12:00:00.0000 TDB
 X = 1.391416854130100E+00 Y =-1.800992385068067E-02 Z =-
3.469784120300255E-02
```

This section of the data provides the cartesian coordinates of Mars relative to (0,0,0) which is the sun. So, by using the Euclidean distance formula, I can figure out the distance of Mars from the sun to confirm if this data is correct. I do this by calculating

$$\sqrt{(1.391416854130100)^2 + (-1.800992385068067 \times 10^{-2})^2 + (-3.469784120300255 \times 10^{-2})^2}$$

This gives me 1.39196593331 which is in the range 1.381 AU - 1.666 AU which is the minimum and maximum distance Mars is from the sun.

When I put the x, y, and z components into the position component of the celestial bodies in the unity editor, the planets were still not following the orbits in the solar system. This is probably because I thought other parameters such as aphelion, perihelion, eccentricity, inclination, etc. would be measured parameters that exist due to gravity rather than ones that are independent of gravity and need to be present to stimulate the solar system.

S-5 Iteration 3

This implementation of the celestial body class was a failure. There were gravitational interactions, but they weren't applicable to the solar system. Ss you can see this is how the solar system was position before the simulation is started:

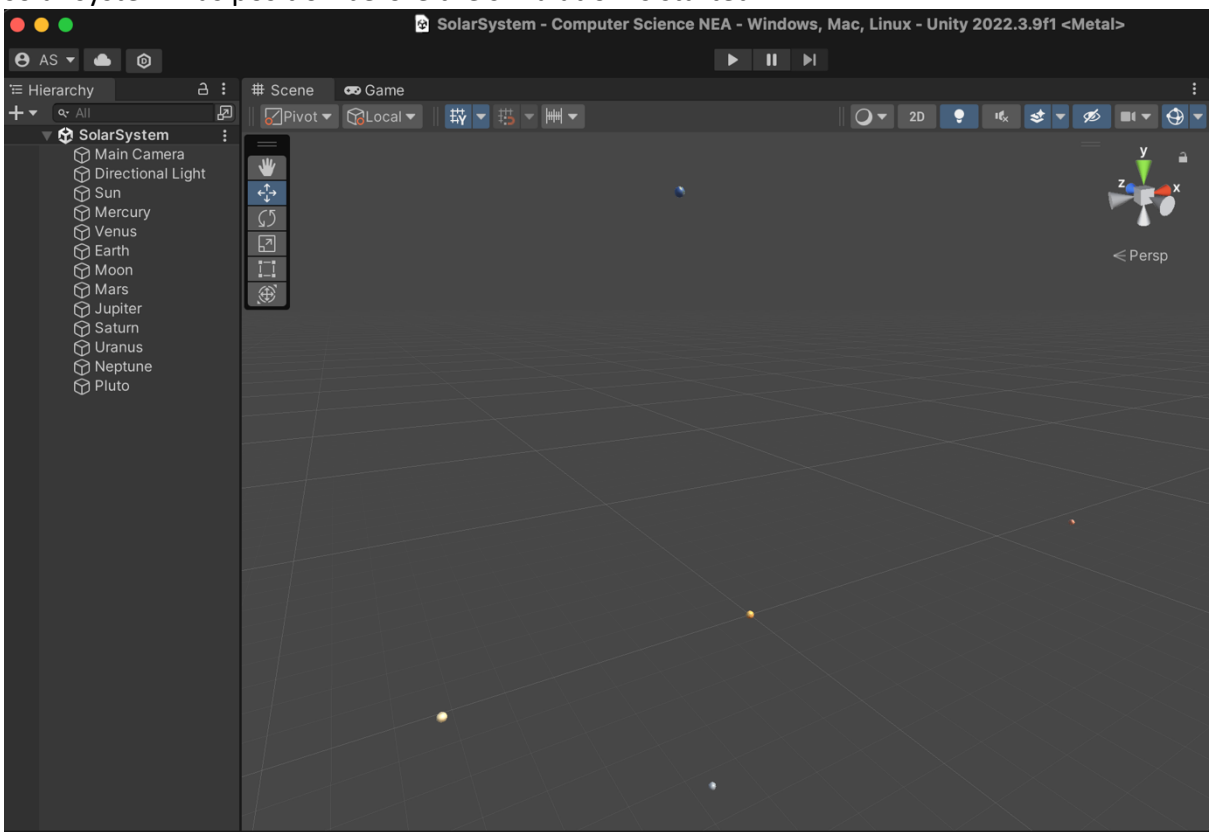


Figure 33: Unity Scene Showing Celestial Body Positions at epoch j2000

After I started the simulation, the planets weren't moving in a circular orbital around the sun, rather they were moving away/towards the sun linearly. As you can see from the 2nd shot Venus has moved away from the sun. The X position of Venus has increased drastically compared to the Y and Z which means Venus isn't travelling in a circular path so test S-6 is a fail.

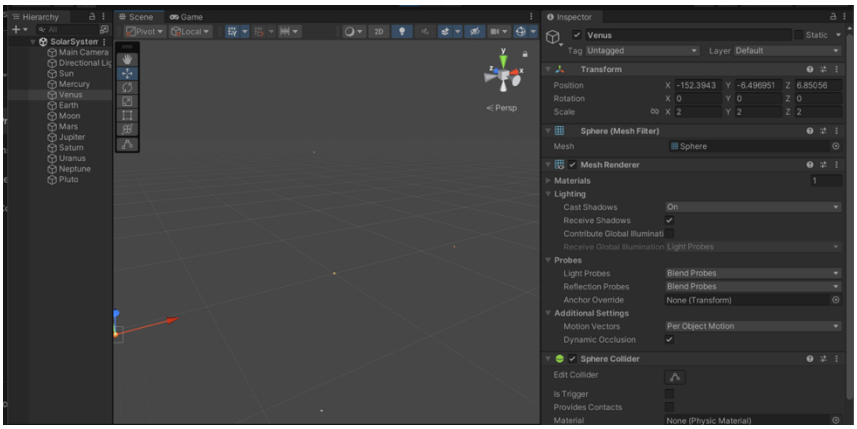
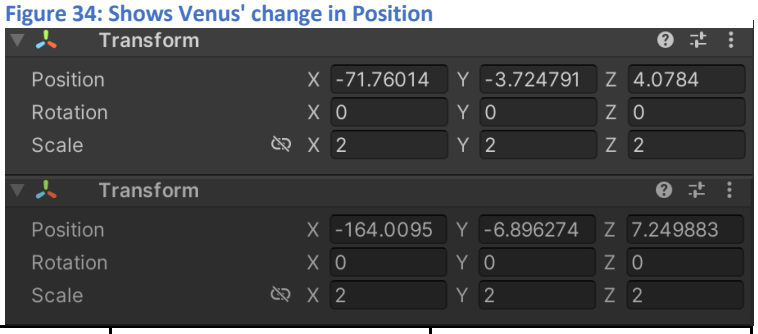


Figure 34: Shows Venus' change in Position



Test ID	Test Case	Test Data	Result	Passed
S-5	Gravitational Interactions Test	Earth: Mass=6e24 kg, Moon: Mass=7e22 kg, Initial positions/velocities	Accurate gravitational interaction	✓
S-6	Multi body test	Real masses, positions, velocities of solar system planets	Accurate representation of our solar system	X

Test S-6

Test S-6 Iteration 1

Updating the positions of the celestial bodies using gravity didn't really work. So, I wanted to take advantage of other parameters and code my simulation using those.
Parameters like true, eccentric, and mean anomaly can help accurately presenting the solar system. So first I defined the necessary parameter is my CelestialBody class:

```
public class CelestialBody : MonoBehaviour
{
    // Variables related to the orbital axis.
    [SerializeField] private float _sideralYear;
    [SerializeField] private float _semiMajorAxis = 2f;
    [SerializeField] private float _inclination = 0f;
    [SerializeField] [Range(0f, 1f)] private float _eccentricity = 0f;
    [SerializeField] private float _argAscending = 0f; // longitude
    [SerializeField] private float _argPerihelion = 0f; // argument
    [SerializeField] private float _startMeanAnomaly = 0f; // J2000
    [SerializeField] private float _meanAnomaly;

    // Variables related to the orbital body.
    [SerializeField] private float _dayLength = 1f;
    [SerializeField] private float _startDayRotation = 0f;
    [SerializeField] private float _size = 1f;
    [SerializeField] private float _rightAscension = 0f;
    [SerializeField] private float _declination = 0f;
```

Figure 35: Celestial Body Variable Definitions

[SerializeField] has been put before defining these parameters so that they can be accessed by Unity because they are private. Most of these parameters are set to some standard value so that they don't cause errors when Unity is launched and can be changed later.

Next, I coded the

```
private float EccentricAnomaly(float M, int dp = 5) {
            // Mathematical Model is as follow:
            // E(n+1) = E(n) - f(E) / f'(E)
            // f(E) = E - e * sin(E) - M
            // f'(E) = 1 - e * cos(E)
            // we are happy when f(E)/f'(E) is small enough.

            int maxIter = 20;  // we make sure we won't loop too much
            int i = 0;
            float e = _eccentricity;
            float precision = Mathf.Pow(10, -dp);
            float E, F;

            // If the eccentricity is high we guess the Mean anomaly for E, other

            E = (e < 0.8) ? M : Mathf.PI;
            F = E - e * Mathf.Sin(M) - M;  //f(E)

            // We will interate until f(E) higher than our wanted precision (as d

            while ((Mathf.Abs(F) > precision) && (i < maxIter)) {
                E = E - F / (1f - e * Mathf.Cos(E));
                F = E - e * Mathf.Sin(E) - M;
                i++;
            }

            return E;
```

Figure 36: Eccentric Anomaly Function

EccentricAnomaly function:

Since this function has to use newtons method of approximation it has to have multiple iterations to get a precise answer. I took 20 iterations as I think it would be enough, alternatively too many will make the program very resource intensive and hard to run as this will have to be done for every celestial body.

Now that I have created a function to find the eccentric anomaly of an orbit, I can create one for the true anomaly. Here it is:

```
private float TrueAnomaly(float E) {
            // from wikipedia we can find several way to solve TA from E.
            // I tried sin(TA) = (sqrt(1-e*e) * sin(E))/(1 -e*cos(E)) but it didn

            float e = _eccentricity;
            float numerator = Mathf.Sqrt(1f - e * e) * Mathf.Sin(E);
            float denominator = Mathf.Cos(E) - e;
            float TA = Mathf.Atan2(numerator, denominator);

            return TA;
        }
```

Figure 37: True Anomaly Function

Now that I have functions for the more complicated physics calculations for some orbital parameters. I can begin coding the initial state of my simulation. As mentioned before, I have data from the equinox J2000 which is a standard point in time used as a reference in astronomy, it is treated like the beginning of time where everything exists as it is. I could've tried using a different point in time, however there isn't as much data readily available to set the starting conditions of the solar system. So, I created a procedure which sets the orbital body's position to one corresponding with J2000. Here it is:

```
private void SetJ2000() {
            transform.Rotate(new Vector3(0, 0, (90 - _declination))); //Rotates o

            transform.Rotate(new Vector3(0, -_rightAscension, 0));
            transform.Rotate(new Vector3(_eclipticTilt, 0, 0));
            transform.Rotate(new Vector3(0, _startDayRotation, 0), Space.Self);

            _angularVelocity = Mathf.Deg2Rad * (360 / _sideralYear); // How much

            _meanAnomaly = _startMeanAnomaly * Mathf.Deg2Rad; //Converts to radia

        }
```

Figure 38: Set epoch J2000 function

This procedure transforms the body's position based on the given right ascension, elliptical tilt and declination. It also calculates the angular velocity from a given sideral year (how long it takes for an orbit to be completed) to find the rate of change of the angle between the orbital plane and object.

I now have the necessary function to be able to produce a moving orbit. However, for it to move accurately I need a time and space script that handles the scale and dimension of space and time for any given space instance.

Review

These functions alone are not enough to run tests yet, it wouldn't make sense, as these functions are not utilised to move the orbital bodies, to satisfy test S-6. So, I'd have to create a function that changes the position of the orbital bodies.

GetPosition() – Test S-6 Iteration 2

Now that I have created functions to calculate the true and eccentric anomaly, I can find the position of any celestial body based on these parameters. To do this, I will find the distance between the focus of an ellipse to the object orbiting around it. To find this distance I would use the equation: $r = \frac{a(1-e^2)}{1+e\cos TA}$ then using the focus radius I find the X , Y, and Z components of the coordinates for the celestial bodies.

```
public Vector3 GetPosition(float M) {
    float e = _eccentricity;
    float a = OrbitScaled; // semiMajorAxis
    float N = _argAscending * Mathf.Deg2Rad; // not const as might vary with precession
    float w = _argPerihelion * Mathf.Deg2Rad;
    float i = _inclination * Mathf.Deg2Rad;

    float E = EccentricAnomaly(M);
    float TA = TrueAnomaly(E);
    float focusRadius = a * (1 - Mathf.Pow(e, 2f)) / (1 + e * Mathf.Cos(TA)); //distance from focus to object

    // parametric equation of an elipse using the orbital elements
    float X = focusRadius * (Mathf.Cos(N) * Mathf.Cos(TA + w) - Mathf.Sin(N) * Mathf.Sin(TA + w)) * Mathf.Cos(i);
    float Y = focusRadius * Mathf.Sin(TA + w) * Mathf.Sin(i);
    float Z = focusRadius * (Mathf.Sin(N) * Mathf.Cos(TA + w) + Mathf.Cos(N) * Mathf.Sin(TA + w)) * Mathf.Cos(i);

    Vector3 orbitPoint = new Vector3(X, Y, Z);

    return orbitPoint;
}
```

Figure 39: GetPosition Function

Body Rotation Functions

AdvanceOrbit() – Test S-6 Iteration 3

This function moves the planets using the GetPosition() function:

```
private void AdvanceOrbit() {
  _meanAnomaly += _angularVelocity * SpaceTime.Instance.DeltatTime;
  if (_meanAnomaly > 2f * Mathf.PI)  // we keep mean anomaly within 2*Pi
      _meanAnomaly -= 2f * Mathf.PI;

  Vector3 orbitPos = GetPosition(_meanAnomaly);
  Vector3 parentPos = _stellarParent.transform.position; // position of the parent to offset the calculated pos (used for mo

  transform.position = new Vector3(orbitPos.x + parentPos.x, orbitPos.y + parentPos.y, orbitPos.z + parentPos.z);
}
```

Figure 40: AdvanceOrbit Function

Angular displacement is added to the original mean anomaly of the celestial body with respect to time. Then the mean anomaly is kept within 2 pi so that it can be used in the getposition function. Not doing this would lead to calculation errors as orbits can only follow a path of angle form 0-360º. The new positions of the planet is then updated by adding the new orbital position to the position of the orbital body, the positions are added because the orbitPos vector only represents the change in the orbital position from the parent body, not from the origin in the scene set in unity, meaning I have to add the parent body's position so the updated celestial body is in the correct place.

Review

When executing the scene with my newly implemented functions, I can see that, the orbital bodies are moving around the sun. However, because my camera movement function is not fully implemented yet I cannot fully verify if the planets are orbiting the sun correctly

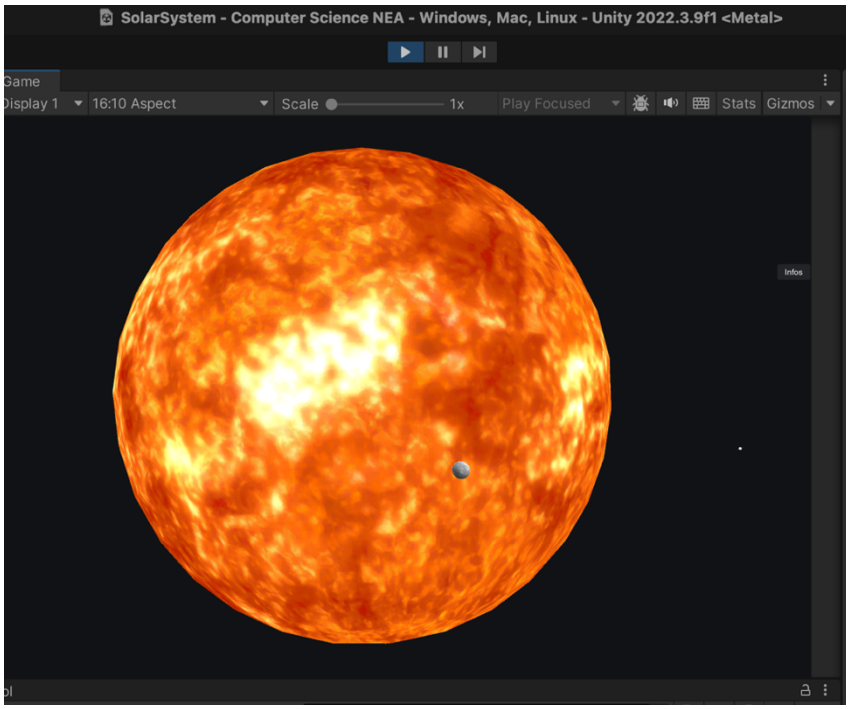


Figure 42: Shows Mercury's Updated Position now Infront of the Sun

Test PC-10, PC-11 Iteration 1

To make sure if my anomaly functions work. I added these lines of code to my Update function in the celestial bodies script.

```
Debug.Log("Eccentric Anomaly: " + EccentricAnomaly(meanAnomaly));
Debug.Log("True Anomaly: " + TrueAnomaly(EccentricAnomaly(meanAnomaly)));
```

Figure 43: Code to update debug values in console log

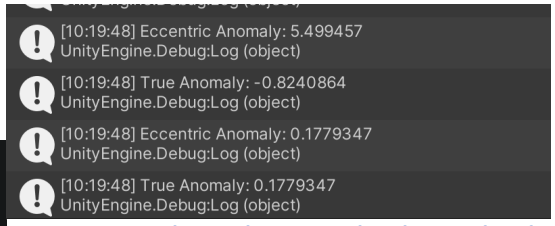


Figure 41: Console Log showing updated anomaly values

The values for true and eccentric anomalies for all celestial bodies in the simulation is printed, the values look correct as they are in the range of -360º to 360º but I can't be sure form which orbital bodies these values are from, so to make sure I made a debugging class in the celestial bodies script.

Test PC-10, PC-11 Iteration 2

The debug class looks like this:

```
public class OrbitalDebug {
  public string name;
  public float eccentricAnomaly;
  public float trueAnomaly;
}Pos.x + parentPos.x, orbitPos.y + parentPos.y, orbitPos.z + parentPos.z);
}
```

Figure 44: Orbital Debug Class Initilisation

I called the debug class inside the orbital body script, so that I can actually see it in the console. When the program is run this is what the console looks like:

Figure 45: Errors from console log

From the console errors I can see that me defining a class in a namespace and trying to access it in the namespace is causing a problem, so i think to fix this I should create a separate debugging script and attach it to the namespace.

Test PC-10, PC-11 Iteration 3

This is what the OrbitalDebug script looks like:
I imported the solsyssim namespace to the script so that the OrbitalBody class can access it and debug the anomaly functions inside.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using solsyssim;


public class OrbitalDebug {
        public string name;
        public float eccentricAnomaly;
        public float trueAnomaly;
    }
```

Figure 46: Orbital Debug class with solyssim namespace access

```
OrbitalDebug debug = new OrbitalDebug();
debug.name = gameObject.name;
Debug.Log(debug);
```

Figure 47: Orbital Debug called in OrbitalBody

Then in the orbitalBody class I called the debug logs in the start and update function:

```
//For debugginf reference
GameObject gameObject;
public string name;

// registering to some events and initialising stuff
private void Start() {
    SpaceTime.Instance.ScaleUpdated += UpdateScale;
    FindObjectOfType<InterfaceManager>().OrbitToggle += TogglePath;
    FindObjectOfType<InterfaceManager>().FullStart += TogglePath;

    SetScales();
    SetJ2000();
    AdvanceOrbit();
    //Debugging

    OrbitalDebug debug = new OrbitalDebug();
    debug.name = name;
    debug.eccentricAnomaly = EccentricAnomaly(_meanAnomaly);
    debug.trueAnomaly = TrueAnomaly(EccentricAnomaly(_meanAnomaly));
    Debug.Log(debug);
}

private void Update() {
    AdvanceOrbit();
    AdvanceDayRotation();
    Debug.Log("Name:"+ name);
    Debug.Log("Eccentric Anomaly: " + EccentricAnomaly(_meanAnomaly));
    Debug.Log("True Anomaly: " + TrueAnomaly(EccentricAnomaly(_meanAnomaly)));
}
```

Figure 48: Logs added to update function to see change in anomalies

Doing this gave me the eccentric and true anomaly for each celestial body along with its name, so now I can verify if the values are true.

Figure 49: Correct values returned from console log

The true anomaly for Venus in the console is 0.8860931 and eccentric is 0.880629 in radians. This value is in the range for true anomaly at epoch j200, by 1º. So, my anomaly functions work.

AngularVelocity()

My current implementation of finding the angular velocity used is not accurate, it assumes that the orbital bodies move the same amount every day which is not true for eccentric orbits. For eccentric orbits planets move faster near the perihelion and slower near the aphelion. To simulate this, I'd need to create a new function to find the angular velocity of an orbit.

```
public float CalculateAngulaVelocity() {
    float m1 = GameObject.Find("Sun").GetComponent<OrbitalBody>()._mass;
    float m2 = _mass;

    float sum = _G * (m1+m2);
    float _angVel = (radius()*m1*(Mathf.Sqrt(sum*((2/radius())-(1/_semiMajorAxis)))))/(m1*radius()*radius());

    return _angVel;
```

Figure 50: Angular velocity calculation function

This function uses the equation $\omega = \frac{\sqrt{G(m_1+m_2)\left(\frac{2}{r}-\frac{1}{a}\right)}}{r}$ to find the angular velocity real time, since radius is going to be a function that changes with time.

Test

The angular velocity isn't updating with time, this is because the function for angular velocity is dependent on the radius, the radius is dependent on the eccentric and true anomaly which are calculated using the mean anomaly. However, the mean anomaly is calculated using the angular velocity, which creates a circular dependency on the variables meaning that they cannot change.

Also, the current implementation of angular velocity makes me simulation inaccurate. It assumes that all orbits are completely circular which is not true. The current implementation just divides 360º by the sideral year of the orbit to find the change in angle every second.

Test PC-09 Iteration 1

To break this circular dependency I'm going to be calculating angular velocity using Newtonian methods. Here is the derivation for angular velocity:

$$F_c = G\frac{Mm}{r^2}$$

$$a_c = \omega^2 r$$

$$G\frac{Mm}{r^2} = m\omega^2 r$$

$$\omega = \sqrt{G\frac{M}{r^3}}$$

Because radius is a value that changes with time, and there is no other quantity in the equation that depends on it, the angular velocity will now be able to change with time as seen here in the updated AdvanceOrbit() function:

```
1  private void AdvanceOrbit() {
2      // Calculate the distance to the sun
3      r = radius();
4
5      float m1 = _stellarParent.gameObject.GetComponent<OrbitalBody>()._mass;
6      float m2 = _mass;
7      float G = 6.67e-11F;
8
9      // Calculate angular velocity based on current radius
10     _angularVelocity = Mathf.Sqrt(G * (m1 + m2) / Mathf.Pow(r, 3));
11
12     // update mean anomaly
13     _meanAnomaly += _angularVelocity * Time.deltaTime;
14     if (_meanAnomaly > 2f * Mathf.PI)  // we keep mean anomaly within 2*Pi
15         _meanAnomaly -= 2f * Mathf.PI;
16
17     Vector3 orbitPos = GetPosition(_meanAnomaly);
18     Vector3 parentPos = _stellarParent.transform.position; // position of the
19   parent to offset the calculated pos (used for moons)
20     transform.position = new Vector3(orbitPos.x + parentPos.x, orbitPos.y + parentPos.y,
   orbitPos.z + parentPos.z);
21 }
```

Figure 51: AdvanceOrbit function updated

And in this video:
https://youtu.be/A2FXLom0IYk

Test ID	Test Case	*Test Data*	Result	Passed
PC-09	Mean Angular Motion (Angular Velocity)	μ = Mean anomaly of pluto a = radius of pluto with time	Angular velocity of Pluto changing with time and increasing closer to sun	✓

SpaceTime

Having a space-time class is useful because it provides a central point of control for managing scales related to space and time, making it easier for the user to adjust these aspects of the game. It offers many more features like pausing the simulation, which if I didn't have my simulation would be less educationally beneficial as for example a specific point in time can't be discussed, and provides default scale values which means that if a user changes a lot of parameters they can go back to the original state of the solar system without having to reset the application which will waste a lot of time.
Space-time will be a singleton class which means it can only exist once, this is the only approach I can take because in real life we have only one instance of space-time in the universe and also if there are multiple instances of space-time the parameter of the celestial bodies in orbits will change at different rates making the simulation very unrealistic and simply not work.
Here is how I made SpaceTime a singleton class:

```
// Singleton instatiation
        private static SpaceTime _instance;
        public static SpaceTime Instance {
            get {
                 if (_instance == null) {
                     if (_instance == null){
                         Debug.LogError("SpaceTime Singleton not yet instanciated.");
                         _instance = new SpaceTime();
                     }
                 }
                return _instance;
            }
        }

        private void Awake() {
            if (_instance != null && _instance != this) {
                Debug.LogWarning("Double instance of SpaceTime Singleton!");
                Destroy(this.gameObject);
            } else {
                _instance = this;
            }
        }
```

Figure 52: SpaceTime Singleton Initilisation

This class checks if an instance of the SpaceTime class exists and if it doesn't it returns an error and creates an instance. If there is and that doesn't equal the current instance it means there must be more than one instance so what happens in the Awake() procedure is that it returns an error saying there is two instances of the space-time singleton and destroys the current instance that is trying to be created. Otherwise, it just creates the instance. I made it so that the new instance being created is deleted instead of the one that existed previously

To be able to change the sizes of orbit, time, and celestial bodies in real time I'd have to implement functions that change them into the space-time class so that it happens dynamically.

Body & Orbit Scale

The implementation of changing body and orbit scale is very similar. To apply the changes from my BodyScale/OrbitScale I've created a method that returns the base body scale but updates it if needed. How this is done is shown below:

```
public float BodyScale {
            get {
                return _bodyScale;
            }
            private set {
                //clamps the values the scale can be to a specific range
                _bodyScale = Mathf.Clamp(value, MinDefaultScale, MaxDefaultScale);
                //Updates scale if needed
                if(ScaleUpdated != null)
                    ScaleUpdated.Invoke(Scale.Body, _bodyScale);
            }
        }
```

Figure 53:Body scale setter

The setter in the code checks if the scale is being updated and if so, it changes the body scale.

Time Scale

The time scale function is implemented a bit differently because time is a function that you can pause compared to the scale of the orbital body and orbit. So, in the setter procedure in the function, I first check if the simulation is paused or not and change the speed of time according.

```
public float TimeScale {
           get { return _timeScale; }
           //checks if game is unpaused, if so it clamps the values the scale for the speed of time can be and updates it

           private set {
               if(!_timePause){
                   _timeScale = Mathf.Clamp(value, MinTimeScale, MaxTimeScale);
                   if(ScaleUpdated != null)
                       ScaleUpdated.Invoke(Scale.Time, _timeScale);
               }
           }
       }
```

Figure 54: Time scale setter

I also created additional time functions that will come in handy:

I can make it so that elapsed time is calculated for every function that needs it but nearly all physics calculations need it so making a reuasble component would make sense. It is also important to have elapsed time as a whole so that users who are using the simulation can get a sense of how orbits change over time.

```
private double _elapsedTime = 0;
public double ElapsedTime {
    get { return _elapsedTime; }
    private set { _elapsedTime = value; }
}

public float DeltatTime {
    get { return Time.deltaTime * TimeScale; }
}
```

Figure 55: Elapsed and Delta Time getters

I've also created a delta time function, having a change in time variable already existing will be very useful when performing physics calculations which involve rates of change.
The elapsed time will be updated in the update function which will be called within every frame.
Other values like the scales will update in a different function using a switch statement. Here is how I will do it:

```
private void UpdateScale(Scale scale, float value) {
  switch(scale) {

      case Scale.Body:
      BodyScale *= 1f + value;
      break;

      case Scale.Orbit:
      OrbitScale *= 1f + value;
      break;

      case Scale.Time:
      TimeScale *= 1f + value;
      break;

      default:
      Debug.LogWarning("Unknown scale when updating scales.");
      break;
  }
  }
```

Figure 56:UpdateScale selection method

Using switch statements instead of if statements are much more efficient because the function directly goes to the correct scale instead of checking if it is the correct one to be updated. This approach saves time and makes it faster to update parameters which is essential when creating an educational tool.
This statement works only for the three scales, if another one is attempted at being updated an error is returned.
Lastly, for the time functions, I had to implement a pause time method:

```
public void PauseTime(bool pause) {
  if(pause){
      _lastTimeScale = TimeScale;
      TimeScale = MinTimeScale;
      _timePause = pause;
  } else {
      _timePause = pause;
      TimeScale = _lastTimeScale;
  }
}
```

Figure 57: PauseTime method

I did this by making the lastTimeScale equal to the current one so that where the time stopped is recorded, this approach will come in very handy for the user as they can know at what time their simulation is paused and therefore analyse what happens at the time known. I then made the current timescale equal to the minimum which is 0. Then set _timePause to pause so that the simulation is no longer updated. When the simulation is resumed, the TimeScale is set to its previous value, effectively resuming time at the speed that it was at before the pause.

Review

In this video I test all the scale functions:
https://youtu.be/IxDylqZ0bBE
As seen in the video they all work!

Control Intentions Class

Before I create my camera class, I need to be able to monitor user inputs and run functions depending on them so that my users can actually interact with the simulation. To do this im going to be creating a new ControlIntentions script and add it to the solsyssim namespace so that other class' can monitor inputs and run functions depending on what is inputted. I created a sperate script for controls as it is going to be used universally throughout the program, so having it attached to one script, for example only the solar system scripts would mean id have to use and switch between multiple script which wastes time and open my simulation to errors. I also had to make the script a singleton as if I didn't user interactions may be registered multiple times and there can be clashes in inputs which can make the simulation unusable.

Before I start implementing anything to my class I need to create a singleton initiation script like I did in my SpaceTime class, to make sure that there are not more than one instances of control intentions causing the simulation to malfunction and user inputs to be not work properly, for example if there were two singletons then each click I user initiates will cause a certain action to happen twice which would make the simulation very frustrating to use. Here is what I did:

```
// Singleton instantiation
  private static ControlIntentions _instance;
  public static ControlIntentions Instance {
      get {
           if (_instance == null){
               if (_instance == null){
                   Debug.LogError("Singleton not yet instanciated.");
                   _instance = new ControlIntentions();
               }
           }
          return _instance;
      }
  }

  // checks the singleton instance and deletes current one if there is already one
  private void Awake() {
      if (_instance != null && _instance != this) {
          Debug.LogWarning("Double instance of ControlIntentions Singleton!");
          Destroy(this.gameObject);
      } else {
          _instance = this;
      }
  }
```

Figure 58:Control Intentions Singleton Instantiation

This class checks if an instance of the ControlIntentions class already exists and if it doesn't, it returns an error and creates an instance. If there is and that doesn't equal the current instance it means there must be more than one instance so what happens in the Awake() procedure is that it returns an error saying there is two instances of the space-time singleton and destroys the current instance that is trying to be created. Otherwise, it just creates the instance. I made it so that the new instance being created is deleted instead of the one that existed previously as setting communication with unity for a new instance may not work and lead to errors.

The solar system scene is going to be broken down into 3 control sections, pause menu control, game control, and scaling control. The game control responds to users interacting with the actual simulation like clicking planets. The pause menu control will be the user pressing esc to open the pause menu etc, and the scaling control will be used for the user to change the time, zoom and orbit scaling. Here's how I implemented it:

```
public enum State { Game, Menu, Scaling };
        private State _state = State.Menu;

        public event Action<SpaceTime.Scale, float> Scaling;
        private void RaiseScaling(SpaceTime.Scale scale, float value) {
            if(Scaling != null)
                Scaling.Invoke(scale, value);
        }
        public event Action<string, float> CamRotation;
        private void RaiseCamRotation(string axe, float dir) {
            if(CamRotation != null)
                CamRotation.Invoke(axe, dir);
        }
        public event Action<float> CamTranslation;
        private void RaiseCamTranslation(float value) {
            if(CamTranslation != null)
                CamTranslation.Invoke(value);
        }
        public event Action<Vector3> FocusSelection;
        private void RaiseFocusSelection(Vector3 mousePosition) {
            if(FocusSelection != null)
                FocusSelection.Invoke(mousePosition);
        }
        public event Action<bool> MenuCall;
        private void RaiseMenuCall(bool call) {
            if(MenuCall != null)
                MenuCall.Invoke(call);
        }
        public event Action<bool> PauseGame;
        private void RaisePauseGame(bool pause) {
            _gamePaused = pause;
            if(PauseGame != null)
                PauseGame.Invoke(_gamePaused);
        }
        private bool _gamePaused = false;
```

Figure 59: Control Intentions enum

I defined a public enum which defines the three states in the solar system scene, I set the initial state to the menu (options pop up), I took this approach because I wanted another step to be there before the user starts playing with the actual simulation so that they can change any settings they want to before the simulation starts as some users may have accessibility needs.

Then I created public event for multiple actions, the first one changes the time scaling based on user input, this is done by creating a public event which can be accessed by any class, this allows for other classes to be notified when there is a change in SpaceTime by. Implementing it in the function RaiseScaling() allows other classes to invoke/trigger the scaling event when a change occurs. This is done by first checking for any subscribers of the Scaling state, if there are none, nothing is invoked as there is no point. If there are any, the scaling is changed by a specific value and all classes are notified because the event has changed. This same logic is repeated for all different possible controls of the simulation. Because this same logic is repeated, I thought of approaching this problem by creating a function to allow for modularity, however this wouldn't really work since it is responsible for a small set of state based events, custom raising code is likely needed for each one anyway. For a class with lots more events or complex shared logic, it would make more sense. But for ControlIntentions, because of the small scope and central role of the class modularization may not yield enough advantage to justify increased abstraction.

Now that I have my control events, I need to create an update function that constantly checks within every frame if there is an update to the control events. To do this I broke down the Update function into three cases, I used a switch case implementation to reduce the use of any unnecessary resources as for example it would be pointless to constantly check for changes in the Menu state when the user is in the game state. Here is what I done:

```
private void Update () {
  switch(_state){

      case State.Game:
          CheckGameInput();
          break;

      case State.Menu:
          CheckMenuInput();
          break;

      case State.Scaling:
          CheckScalingInput();
          break;

      default:
          Debug.LogError("Unknown game state when checking for input.");
          break;
  }
}
```

Figure 60:Control Intentions Update Function

I nested separate check functions into my update function instead of writing them separately as this allows for modularity within my simulation, CheckGameInput may be used by other classes' not just controlIntentions. Also, this will make it much easier to find and fix any errors in the control monitoring functions compared to all being in the update function.

Test I-7

The update function returns an error in the console log if there is no state initialised, this would mean that the user has zero access to the simulation and they cannot control its current state.

Test ID	Test Case	Test Data	Result	Passed

I-7	Monitor Control Input State	Launch simulation	Control input state is detected and reflected in the simulation	✓

Now, I have to create the check functions for each state in the ControlIntentions class.

CheckGameInput()

This function will need to check specific conditions in the game to allow for a change in the states, it will utilise the private methods created earlier to check if there are any subscribers to the event and then it will invoke the new value onto the event. Here how I did this:

```
private void CheckGameInput() {

    if (Input.GetKeyDown(KeyCode.Escape) || SimulatedInput == "menu" || Input.GetButtonDown("menu")) {
        _state = State.Menu;
        RaisePauseGame(true);
        RaiseMenuCall(true);
    } else if ((Input.GetButtonDown("scale time") || Input.GetButtonDown("scale orbits") || Input.GetButtonDown("scale bodies

        _state = State.Scaling;
    }

    // pause is not a full state as besides time, control should reamin operational
    if (Input.GetButtonDown("pause game"))
    RaisePauseGame(!_gamePaused);

    // Rotation of the cam around the center horizontally (on the y axis of Axis).
    if (Input.GetAxis("rotate cam horizontally") != 0)
        RaiseCamRotation("horizontal", Input.GetAxis("rotate cam horizontally"));

    // Rotation of the cam around the center vertically (on the x axis of Pole).
    if (Input.GetAxis("rotate cam vertically") != 0)
        RaiseCamRotation("vertical", Input.GetAxis("rotate cam vertically"));

    // Translation of the cam on the z axis (from and away)
    if (Input.GetAxis("translate cam (zoom)") != 0)
        RaiseCamTranslation(Input.GetAxis("translate cam (zoom)"));

    // Focus Body Selection
    if (Input.GetMouseButtonDown(0))
        RaiseFocusSelection(Input.mousePosition);
}
```

Figure 61: CheckGameInput Procedure

The first if statement checks if the escape key or any other button assigned in unity to raise the pause menu is pressed, if so, it will call the private functions previously created to invoke a change in the states else, it will change the state to one of the scaling states of time, orbits or bodies. Instead of having multiple if statements for each scaling state I put them all into one since they all fall into the same enum, doing this saves time when coding and also reduces the time complexity of the function as there are only one check for GetButtonDown.

Next, I coded camera rotation functions. To allow the player to continuously rotate the camera horizontally and vertically. Horizontal rotation makes the camera orbit around the focus point, while vertical tilting changes the angle up or down. To enable smooth continuous rotation, I used Input.GetAxis rather than GetButtonDown or GetKeyDown. GetAxis returns a value between -1 to 1 each frame based on input. For example, holding left returns -1, holding right returns +1. This allows for a gradual change in the rotation over multiple frames based on how long a key is held, rather than just on a single keypress event.

Two axes are used, "rotate cam horizontally", and "rotate cam vertically". These are mapped to the input controls set in Unity's input manager. It then checks if the horizontal axis value is non-zero if not the GetAxis("rotate cam horizontally") value is passed through as the rotation amount. This is also done similarly for the vertical axis/tilt. Taking this approach and having the axis separated means the controls can be mapped independently which is useful to allow further accessibility for different users who wish to use different input devices to control the simulation. So, in summary as the player holds left/right or up/down, the axis values will continuously change from -1 to 1 over multiple frames, resulting in smooth rotation.

Tests I-1 to I-7

Test ID	Test Case	Test Data	Results	Passed
I-1	Pause Simulation	N/A	Simulation pauses, and celestial bodies freeze.	✗
I-2	Resume Simulation	N/A	Simulation continues from the paused state.	✗
I-3	Adjust Simulation Speed	Speed: 2x	Simulation runs at the specified speed.	✗
I-4	Toggle Celestial Body Labels	Labels: On, Labels: Off	Labels are displayed or hidden as per the setting.	✗
I-5	Change Camera Target	Target: Moon	Camera focuses on the specified celestial body.	✗

Iteration 1

The simulation seems to recognise the control intentions state but doesn't recognise the user's inputs. When running the simulation and trying to press buttons that are pre-assigned in the control intentions script, i.e escape button for pulling up the menu and

```
1  private void CheckMenuInput() {
2    // check condition for changing state
3    // hardcoded to always be able to access menus and quit
4    if (Input.GetKeyDown(KeyCode.Escape) || SimulatedInput == "menu" || Input.GetButtonDown("menu")) {
5        _state = State.Game;
6        RaiseMenuCall(false);
7    }
8  }
9
10 if (Input.GetKeyDown(KeyCode.Escape) || SimulatedInput == "menu" || Input.GetButtonDown("menu")) {
11     _state = State.Menu;
12     RaisePauseGame(true);
13     RaiseMenuCall(true);
14 } else if ((Input.GetButtonDown("scale time") || Input.GetButtonDown("scale orbits") || Input.GetButton
15     _state = State.Scaling;
```

Figure 62: CheckMenuInput Procedure

pausing the game, which can be seen in lines of code, 4 and 10, where the actual keycode is specified instead of a reference to it:

Having references to escape directly in the script is a better approach because, escape is a universal control for pausing games, so keeping it as a standard would be very important when creating any piece of software for the public to use, as it would make the game less complicated it understand and more user friendly compared to an unkownn irrevelant button like T. It also allows for me to debug the code like I am doing here, there are clear argument exception errors shown in the unity console for thew buttons that I assigned key codes to in the script.

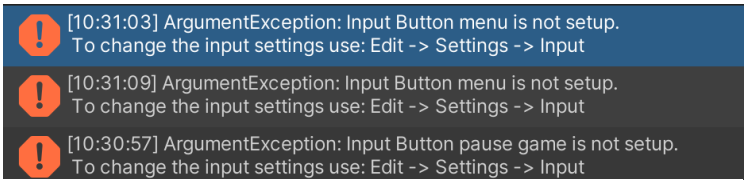


Figure 63: Unity Console Errors

When looking at the errors in more detail you can see that at:
/Users/bokken/build/output/unity/unity/Modules/InputLegacy/Input.bindings.cs:373
There is an error in the unity input manager package.

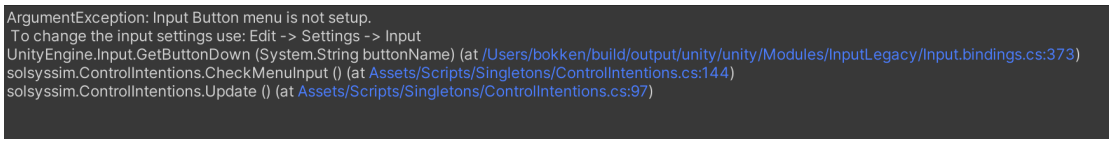


Figure 64: Error Details/Location

So, I went into the project settings in unity and checked the input manager and there was these pre-set inputs that are recognisable by unity:

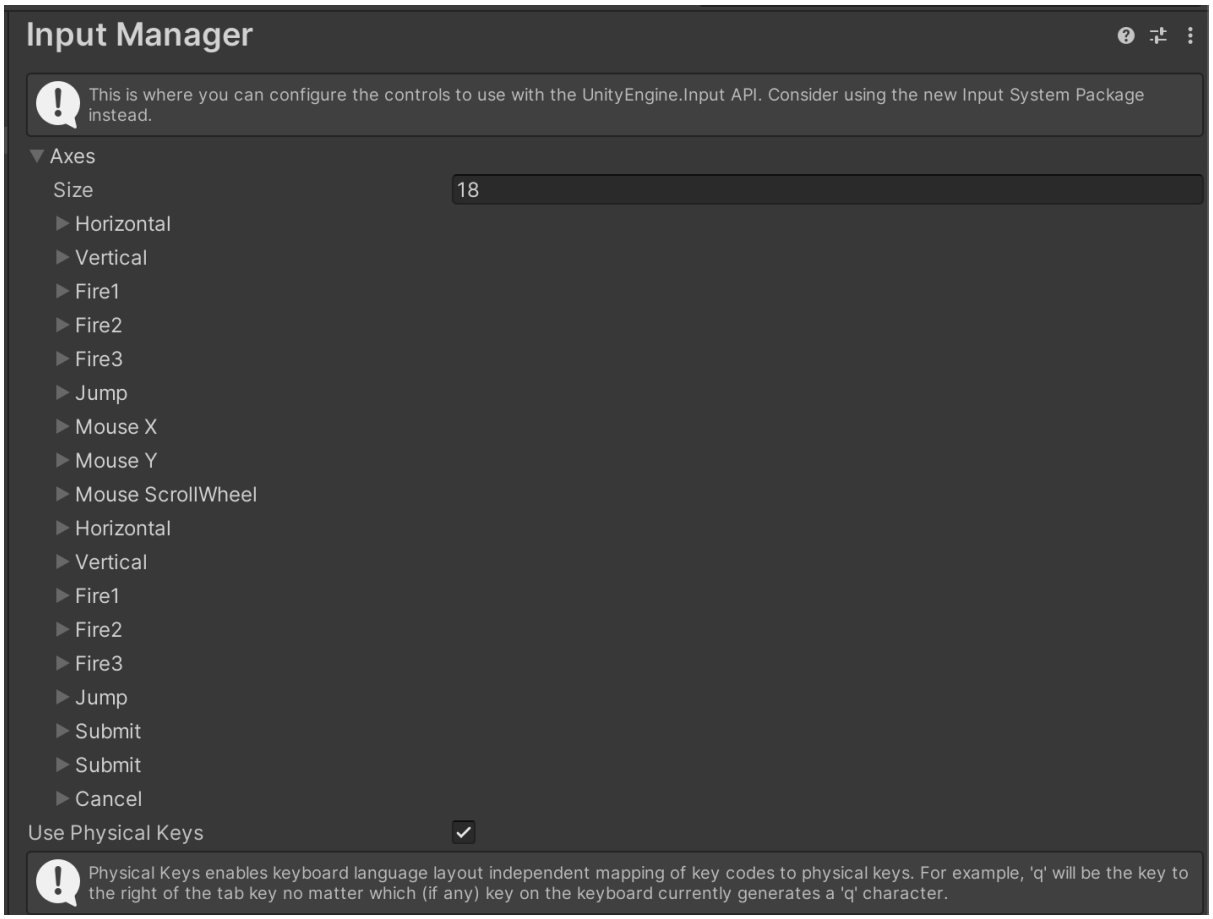


Figure 65:Unity Input Manager

So, the reason why I have been getting these errors is because there are no keycodes assigned to values which are referenced in my control intentions script. To prevent this error from happening again I assigned values to the references of keycodes in my control intentions script. As you can see here:

I choose specific values for Gravity, Dead, and Sensitivity in my input manager to fine tune my user's experience. These constants to different things, Gravity represents the rate at which the input axis returns to its default state when the user releases the input. For example, when using the w, a, s, d keys to change the camera pan, the camera still pans for a very small amount in the direction of the button the user just let go of, so having a very large value will make the button less responsive The reason why I didn't make the value for gravity zero when changing the camera direction horizontally/ vertically is because it allows for a smoother more natural feel to the simulation, whereas having it stop instantly makes the simulation look roughly made.

The Dead of the input represents a range around the input axis where the input is considered inactive. This is important when creating precise inputs, where very small movements like a joystick or the movement of the mouse matters a lot, in an orbital motion simulation the mouse/joystick movement is not very important as it is not changing the real world values in the simulation rather changing the camera position, so I made the Dead of the input a very small value because the range at which it is inactive is not very important as

accidental inputs do not matter much when controlling the camera as it does not affect the accuracy of the simulation.

The sensitivity of the inputs allows for a larger range of the input being represented by a smaller physics input. I kept the sensitivity of the camera at 1000 which is fast enough and not too slow to pan the camera. Having a very fast sensitivity will make the simulation hard to use and a slow simulation will make the simulation frustrating to use, so through trial and error I chose the value 1000 for sensitivity as it felt most appropriate.

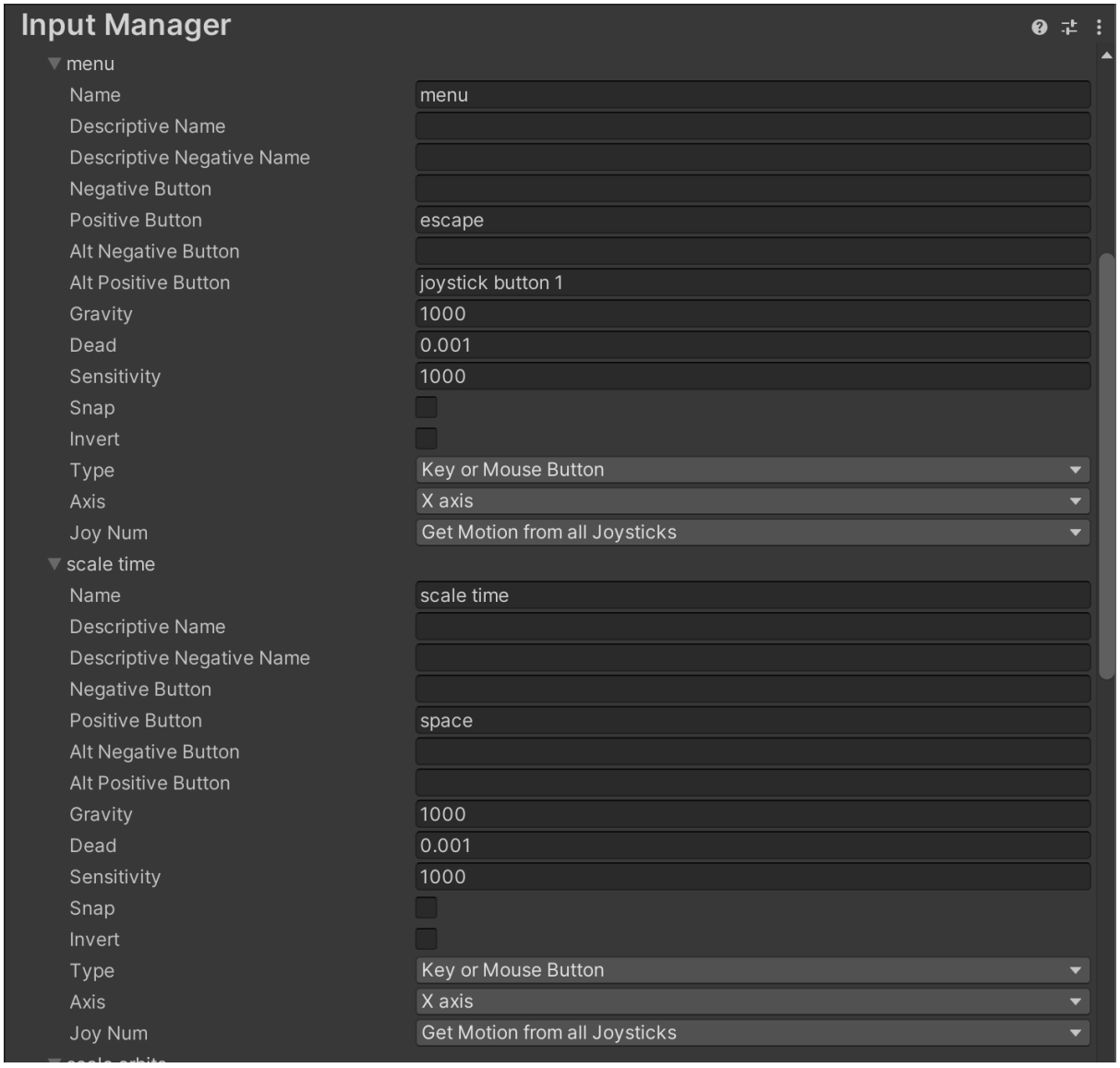


Figure 66: Scale Time and Menu input options

Doing this now allow for me to move my camera which I am going to further test in my Camera class section which I will talk about later on.

Review

Test ID	Test Case	Test Data	Results	Passed

I-1	Pause Simulation	N/A	Simulation pauses, and celestial bodies freeze.	✓
I-2	Resume Simulation	N/A	Simulation continues from the paused state.	✓
I-3	Adjust Simulation Speed	Speed: 2x	Simulation runs at the specified speed.	✓
I-4	Toggle Celestial Body Labels	Labels: On, Labels: Off	Labels are displayed or hidden as per the setting.	✓
I-5	Change Camera Target	Target: Moon	Camera focuses on the specified celestial body.	✓

All inputs are now recognised by unity. This will be further tested in the classes relating to the inputs (i.e. camera target would be further tested in camera development section)

Review

The solar system component of the scene has met all of the success criteria for it and has been thoroughly tested with the tests plan.

Criteria Met:

Criteria No.	Criteria	Criteria Met
C-7	Calculations/equations needed for accurate orbital paths	Completed
C-8	Select Planet to Track	Completed
C-9	Zoom In/out	Completed
C-10	Planet Graphics	Completed
C-17	Simulation Speed	Completed
C-18	Orbit Scale	Completed
C-19	Body Scale	Completed

Changes from my original design

The orbits are better than my original design, they now have scales for size of orbit and size of orbital bodies and react to the speed of time.

UI

My UI is going to consist of a retractable side bar which displays all celestial parameters and toggle that are need to control the simulation.

Sidebar

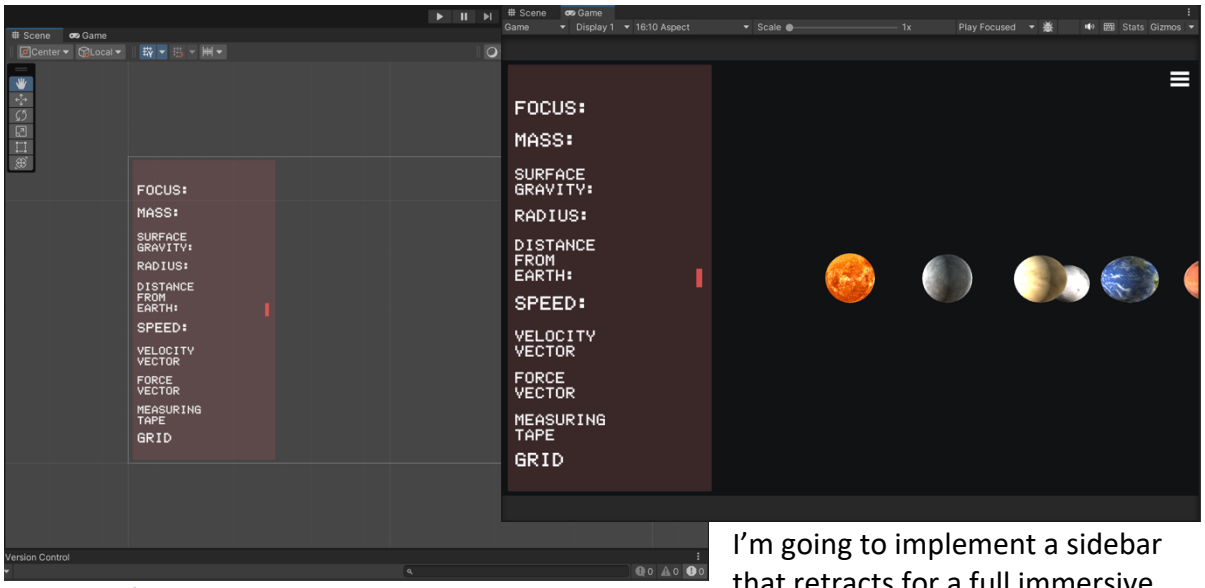


Figure 67: SideBar Layout

I'm going to implement a sidebar that retracts for a full immersive view of the solar system instead of having a static menu which covers the simulation. This will be better because it gives the user more control and comfortability with the simulation and it will be essential for people with smaller screens so they can see the full solar system.

When run nothing shows up, this is because nothing is assigned to these texts, so here's how I tried to implement it first:

```
using TMPro;
private TextMeshProUGUI _focus;
private void Start() {

            _camPole = _cam.transform.parent;
            _focus = GetComponent<TextMeshProUGUI>();

private void CheckSelection(Vector3 selectorPos) {
    // Ray is casted onto the UI to see if we catched one of the icons
    Ray ray = _cam.ScreenPointToRay(selectorPos);
    RaycastHit hit;

    if (Physics.Raycast(ray, out hit, RayLength)) {
        if (hit.collider != null) {
            _selectedBody = hit.collider.transform;

            if (NewFocus != null)
                NewFocus(_selectedBody);
            _focus.text = _selectedBody.name;
        }
    }

}
```

Figure 68: Updated Start and CheckSelection method

For the user to know which focus is selected in the camControl script I try to implement a way for it to change the focus text in the unity scene. What it does is it find any component of TextMeshPro which is a component that is assigned to my text labels in the scene which have the CamControl script attached to it. I chose to do it this way as creating a new script for the UI seemed unnecessary since it was a few lines of code.

However, this implementation did not work, also having a script would allow for more modularity and make the game easier to update in the future if I want to add more parameters that I want the user to see.

Method 2

I created a InfoDisplay script which extracts all parameters from other scripts, like speed form OrbitalBody, to change what the parameter labels display in the unity scene.

```
public class InfosDisplay : MonoBehaviour{
    // UI references
    public TextMeshProUGUI speedText;
    public TextMeshProUGUI massText;
    public TextMeshProUGUI radiusText;
    public TextMeshProUGUI aphelionText;
    public TextMeshProUGUI perihelionText;
    public TextMeshProUGUI meanAnomalyText;
    public TextMeshProUGUI inclinationText;
    public TextMeshProUGUI eccentricityText;
    public TextMeshProUGUI semiMajorAxisText;
    public TextMeshProUGUI angularVelocityText;
    private CamControl camControl;

    private void Start()
    {
        // Find the CamControl component in the scene
        camControl = FindObjectOfType<CamControl>();
        if (camControl != null)
        {

            // Subscribe to the NewFocus event
            camControl.NewFocus += OnNewBodyFocused;
        }
    }

    private void OnDestroy()
    {
        // Unsubscribe from the NewFocus event
        if (camControl != null){
            camControl.NewFocus -= OnNewBodyFocused;
        }
    }

    private void OnNewBodyFocused(Transform focusedBody)
    {
        // Get the OrbitalBody component from the focused body
        OrbitalBody orbitalBody = focusedBody.GetComponent<OrbitalBody>();
        if (orbitalBody != null)
        {
            // Update UI elements with the parameters from the OrbitalBody
            speedText.text = "Speed: " + orbitalBody.GetSpeed().ToString("F2") + " km/s";
            massText.text = "Mass: " + orbitalBody.mass.ToString("F2") + " kg";
            radiusText.text = "Radius: " + orbitalBody.radius.ToString("F2") + " km";
            //.....
            //removed other parameters fromm this image to save space
        }
    }
}
```

Figure 69: InfoDisplay script

It starts by subscribing to the newFocus event of camControl to see if there has been an update in the focus, so that it can run the OnNewBodyFocused method which gets the orbital body component from the focused body, so that it has access to the paramters of it, it then updates the UI elements in unity by referencing to the names of the text objects and changing it to the parameter name folloed by a 2 decimal place value of it. I chose to use two decimal places as not having the decimal places capped at a certain number of places would mean my UI will look very unprofessional when I have parameters which are very precise. Also have a consistency in the number of decimal places is important in physics as it makes the uncertainty in calculations easier to figure out.

UI – 02 Test Iteration 1

When I assign this new script to my UI components in the unity scene, there Is no difference in their state, the labels aren't updated.

So, to troubleshoot this issue I had to check if the newFocus event is changing states when there is a selection. The root of this event is under the CheckSelection method in the CamControl script. Where newFocus is updated. For newFocus to be updated the checkSelection method must be called. So, for me to check if it is being called I added a debug log method under its initialisation so I know it Is being used as soon as it is called:

```
private void CheckSelection(Vector3 selectorPos) {
    Debug.Log("CheckSelection method called.");

.....
```

Figure 70: Check Selection Debug

When the simulation is run again, and I click on a celestial body the log console verifies that the CheckSelection method is called:

Figure 71: Console Debug Results

However, this doesn't necessarily mean that the raycast recognised that an object has been pressed, CheckSelection is only being called because there was a click In the simulation, so for me to check if the raycast actually recognises this click as a celestial body, id need to add more debug scripts in the CheckSelection method:

```
private void CheckSelection(Vector3 selectorPos) {
    Debug.Log("CheckSelection method called.");
    // Ray is casted onto the UI to see if we catched one of the icons
    Ray ray = _cam.ScreenPointToRay(selectorPos);
    RaycastHit hit;
    float rayLength = RayLength;

    if (Physics.Raycast(ray, out hit, RayLength)) {
        if (hit.collider != null) {
            Debug.Log("Hit object: " + hit.collider.gameObject.name);
            _selectedBody = hit.collider.transform;

            if (NewFocus == null)
                NewFocus(_selectedBody);
            _focus.text = _selectedBody.name;
        }
    } else {
        Debug.Log("Raycast did not hit any object.");
    }
}
```

Figure 72: Better debugging code

These debugging scripts return the name of the object that has been hit. This 100% verifies if a celestial body has been hit. So, when running my simulation and trying to select mercury I get these I the console log:

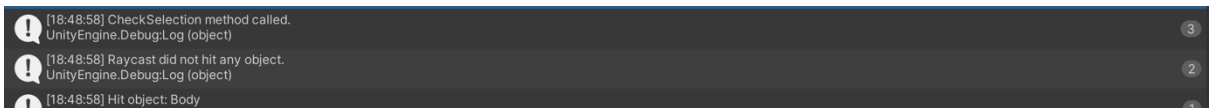


Figure 73: Debug Results

At first it says raycast didn't hit any object because I mis clicked the celestial body because it was moving. Maybe this implementation of selecting a body to focus on is a bit unreliable as the user would have to try and get the perfect click which can be frustrating, so maybe possibly in the future of my development I will make it so that if the user clicks on the orbital path the body will be focused on but for now I am going to be using the celestial body as the primary selection method as it is easy to work with and avoids creating more scripts which can make it harder to find the root issue in the test of my function.

If the celestial body is being clicked on, and this is clearly recognised by the checkselection method therefore the OnNewBodyFocused method then there is another issue in the method.

Test ID	Test Case	Test Data	Results	Passed
UI-02	Parameter labels	Perihelion	Labels are unchanged	X

UI – 02 Test Iteration 2

When trying to click on a celestial body I noticed this NullReferenceException error. Having know that checkselection is functional, this must mean that in the OnNewBodyFocused method there is an error in reference to the celestial body.

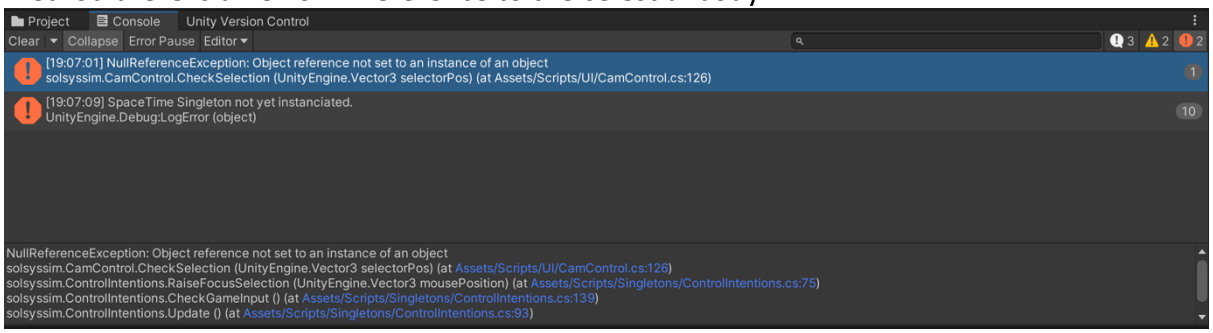


Figure 74: Unity console errors

So instead of using the original reference to the celestial body which checks for the celestial body component in the game object that is selected. I'm going to change to the _selectedBody variable in the CamControl script which is known as a definite reference to the focused body as I've already tested it in the 1st iteration of the test.

```
1    private void OnNewBodyFocused(Transform focusedBody)
2    {
3        // Get the OrbitalBody component from the focused body
4        Debug.Log("New body focused: " + focusedBody.name);
5        OrbitalBody orbitalBody = camControl._selectedBody.parent.GetComponent<OrbitalBody>();
6        if (orbitalBody != null)
7        // .........cut to save space in image........
```

Figure 75: Change in reference for OnNewBodyFocused

To do this I declared _selectedBody as a public variable in the CamControl class so that InfoDisplay can access it and I referenced selectedbody's parent instead of focusedbody. I done this because the orbitalBody script component in unity is attached to the parent gameobject of the selected body.

Also, if there is a null reference that must mean that newFocus is not being updated in the CheckSelection method. Looking at my method, I realised that the newFocus variable isn't

being updated.

```
if (Physics.Raycast(ray, out hit, RayLength)) {
    if (hit.collider != null) {
        Debug.Log("Hit object: " + _selectedBody.parent.name);
        _selectedBody = hit.collider.transform;

        if (NewFocus == null) //SOLUTION IS TO REMOVE THIS IF STATEMENT AS IT DOESNT RUN
            NewFocus(_selectedBody);
        _focus.text = _selectedBody.name;
    }
```

Figure 76: CheckSelection Selection Statement Removal

This was because I made it so that the variable is only updated if newFocus is equal to null. But I realised that this would be unnecessary as it doesn't matter what newFocus is equal to before updating it. Removing the if statement means that newFocus can never be equal to null as the new selected body is going to be applied either way. So, this part of my new method:

```
if (Physics.Raycast(ray, out hit, RayLength)) {
    if (hit.collider != null) {
        Debug.Log("Hit object: " + _selectedBody.parent.name);
        _selectedBody = hit.collider.transform;
        NewFocus(_selectedBody);
        _focus.text = _selectedBody.name;
    }
```

Figure 77: CheckSelection Updated

When run I get a value from my test data which is the perihelion. As seen here:

Figure 78: Perihelion Display in game

To get data for other orbital parameters that are more prominent in Newtonian mechanics, like speed, and gravitational field strength etc, compared to using Kepler's laws I will create functions in the orbitalbody script to do so.

Test ID	Test Case	Test Data	Results	Passed
UI-02	Parameter labels	Perihelion	Labels are unchanged	✓

UI – 03 Test Iteration 1

There are orbital parameters displaying in the unity scene. However, they are not in the correct units.

Angular Velocity

My first test case would be angular velocity, in the orbitalBody script angular velocity is in units of radians per day.

```
angularVelocityText.text = " Angular Velocity:" + ((orbitalBody._angularVelocity)/(24*60*60)).ToString("G5") + "rad/s";
```

Figure 79: InfoDisplay angular velocity value

I will convert this value to radians per second in the InfoDisplay script instead of another unit as I do not want to cause errors in the orbital body script because changing it to radians per second would mean that the simulation thinks the new value for angular velocity is still radians per day making it much smaller than it would actually be, therefore a slower speed for planets.

I also set the number to 5 significant figures as having more would be unnecessary and also causes there to be a multi-line display for the velocity which makes the simulation look unprofessionally made.

Semi-Major Axis

The semi major axis is a 1000x less than its actual value for scaling reasons, having the scale to the real size would mean that orbits have a huge distance between them making them harder to see together, thus to compare.

Stakeholder Feedback

I'm not 100% sure which orbital parameters would be useful to display in the sidebar for an educational setting. So, I will be getting stakeholder opinion, mainly from the teachers.

Mr Goosen - "As an A levels teacher I'd use this simulation to make students find the change In energy of a system, so having the absolute velocity relative to the sun, distance from sun, and time would be very important to me"

I already have the time implemented, so the extra parameters which I will calculate then display is the absolute velocity relative to the sun and the radius, not the semi major axis, as semi major axis is the radius from the furthest point.

Radius

The radius will be the first parameter that will be constantly updated within each frame when displayed, to allow for a constant update of parameters id have to create an update function.

```
private void Update()
{
    if (camControl != null && camControl._selectedBody != null)
    {
        OnNewBodyFocused(camControl._selectedBody);
    }
}
```

Figure 80: Update method

Update()

In my update function I checked if CamControl exists and that there is a selected body, so that if there is, within every frame the OnNewBodyFocused function runs which retrieves all orbital parameters from the OrbitalBody script.

Having an update function implemented is important because if I didn't have an update function users would have to constantly focus on an orbital body to see its new parameters. This would be very frustrating for users looking to analyse changes in variables over time and also for users with accessibility needs whom can't be constantly clicking planets it will make the simulation unusable.

Test

https://youtu.be/tcUV_yfJ6-k

Test ID	Test Case	Test Data	Results	Passed
UI-04	Live Updates	Mean Anomaly/ Radius	Mean anomaly/Radius changes values throughout the simulation	✓

Radius

Now that o have an update() function I can create one to calculate the radius. Here is my first implementation of it:

```
1 public float radius() {
2     // from wikepedia its says that radius cna be solved with TA, eccentrity, and semi major axis
3     // r = a ((1-e^2)/(1+e*cos(TA)))
4     float e = _eccentricity;
5     float a = _semiMajorAxis;
6     v = TA;
7     r = (1000*a) *((1-(e*e))/(1+(e*Mathf.Cos(v))));
8     return r;
9 }
```

Figure 81: Radius function

I chose to calculate radius from true anomaly as it is also a variable that changes with time. If I chose a different variable then radius will return the same value within every frame.

I also made it so that the radius() function is called in the update function in the OrbitalBody script.

After doing this and inserting radius into the InfosDisplay script, I got a 0 value for the radius of the orbits. So, I had to test to see why this is.

Test Iteration 1

Since I couldn't see the radius in the scene, I inserted a debug.log that will be called within every updated value of r.

```
public float radius() {
    // from wikepedia its says that radius cna be solved with TA, eccentrity, and semi major axis
    // r = a ((1-e^2)/(1+e*cos(TA)))
    float e = _eccentricity;
    float a = _semiMajorAxis;
    v = TA;
    r = (1000*a) *((1-(e*e))/(1+(e*Mathf.Cos(v))));
    Debug.Log("Radius:" + r);
    return r;
}
```

Figure 82: Radius debug line

```
[12:14:44] Radius:3628800
UnityEngine.Debug:Log (object)
[12:14:44] Radius:4.46391E+09
UnityEngine.Debug:Log (object)
[12:14:44] Radius:2.067407E+08
UnityEngine.Debug:Log (object)
[12:14:44] Radius:7.393176E+08
UnityEngine.Debug:Log (object)
[12:14:44] Radius:0
UnityEngine.Debug:Log (object)
[12:14:44] Radius:1.074505E+08
UnityEngine.Debug:Log (object)
[12:14:44] Radius:1.349344E+09
UnityEngine.Debug:Log (object)
[12:14:44] Radius:4.597974E+07
UnityEngine.Debug:Log (object)
[12:14:44] Radius:1.46606E+08
UnityEngine.Debug:Log (object)
[12:14:44] Radius:2.742788E+09
UnityEngine.Debug:Log (object)
```

Figure 83: Console log showing debug values

What I realised after looking at the console is that having radius being called in the update() function in the OrbitalBody script means that it is being called for every orbital body. I have to remove this as this will be a very unnecessary toll on the hardware as it has to do multiple times more calculations then necessary because the user only wants to see the radius of the orbital body being focused on.

Test Iteration 2

```
    private void Update()
    {
        if (camControl != null && camControl._selectedBody != null)
        {
            OnNewBodyFocused(camControl._selectedBody);
            float r = camControl._selectedBody.parent.GetComponent<OrbitalBody>().radius();
        }
    }
```

Figure 84: Update function radius implementation

Calling the radius function in InfoDisplay worked! The radius displayed in the sidebar is no longer zero, and it is only displaying the radius for the planet that is focused on. However, since most orbits are nearly circular I can't 100% determine if the radius is actually changing based on the body orbital positions.

To combat this issue, I could add Pluto to the list even though it is not a planet. Pluto has the most eccentric orbit in the solar system, it is actually much more eccentric then most, meaning the distance between Pluto and the sun varies a lot with time. I could skip this and not test if the radius is actually updating, by assuming that it is because the orbits are basically circular, but not knowing if the function actually works can be an issue if the user happens to pause the simulation at a time which changes the radius by a very small amount.

Test Iteration 3

Now that I added Pluto I can check if the radius is changing with time. When starting the simulation, the radius shown in sidebar didn't change. So, I checked every variable that the radius function uses, and when debugging I realised that TA was always 0, because cos(0) equals one there is no change in the radius as TA is not changing. I think this was because I used the value that TA was declared with which was 0 instead of calculating TA in the radius function. Here is what my radius function now looks like:

```
1  public float radius() {
2      // from wikepedia its says that radius cna be solved with TA, eccentrity, and semi major axis
3      // r = a ((1-e^2)/(1+e*cos(TA)))
4      float e = _eccentricity;
5      float a = _semiMajorAxis;
6      float E = EccentricAnomaly(_meanAnomaly);
7      float v = TrueAnomaly(E);
8      r = (1000*a) *((1-(e*e))/(1+(e*Mathf.Cos(v))));
9      //Debug.Log("Radius:" + r);
10     return r;
11 }
```

Figure 85: Updated true anomaly parameter

The radius for Pluto is now changing with time which you can see form this video: https://youtu.be/_O8pcoBr1bQ

Test ID	Test Case	Test Data	Results	Passed
UI-04	Live Updates	Mean Anomaly/ Radius	Mean anomaly/Radius changes values throughout the simulation	✓

DrawPath()

For the user to see and determine the path that the celestial body is going to take I have to make a function that will draw the path and make it dynamically change as other parameters change. Having a path drawn for the user makes the simulation much more useful for them than watching a whole orbit to realise parameters like the eccentricity have changed because they changed another parameter in the simulation.
I can draw paths using Unity's low-level graphics library. Here's how I did it:

```
private void DrawPath() {
    const float PathDetail = 0.03f;
    GL.PushMatrix();  // Push the current transformation matrix onto the matrix stack.
    _line.color = UiVisual;
    _line.SetPass(0);  // Set the material pass to render the line.
    GL.Begin(GL.LINES);  // Begin rendering lines using OpenGL.
    Vector3 parentPos = _stellarParent.transform.position;  // Get the position of the parent object.
    // Loop through a full circle by the specified path detail and create a line for each step.
    for (float theta = 0.0f; theta < (2f * Mathf.PI); theta += PathDetail) {
        float anomalyA = theta;
        Vector3 orbitPointA = GetPosition(anomalyA);
        GL.Vertex3(orbitPointA.x + parentPos.x, orbitPointA.y + parentPos.y, orbitPointA.z + parentPos.z);

        float anomalyB = theta + PathDetail;
        Vector3 orbitPointB = GetPosition(anomalyB);
        GL.Vertex3(orbitPointB.x + parentPos.x, orbitPointB.y + parentPos.y, orbitPointB.z + parentPos.z);
    }
    GL.End();
    GL.PopMatrix();
}
```

Figure 86: DrawPath() function

The function starts by pushing a new transformation matrix onto the matrix stack to isolate each instance of DrawPath() for every celestial body making each path separate so that they do not interact with each other causing inaccurately produced paths. It then sets the colour of the line and renders a line connecting two points in the orbit with a distance determined by the value stored in the variable PathDetail. At the end the Matrix is popped so that it doesn't interfere with other path lines.

I chose OpenGl to implement the graphics rendering as it is a widely supported platform, also it is not very resource heavy and also allows for the customisation of path detail and colour, I could've implemented it using. Unity's trail renderer however this platform is very resource dependant, also I feel like having the orbit predetermined for the user rather than them having to wait for a complete rotation so that the trial can render a full circle would save them a lot more time if they want to visualise orbital parameters like eccentricity quicker.

Test

This is what the simulation looks like with orbital paths rendered. However, if you look carefully, you can see that there are duplicate paths rendered that do not have an orbital body attached to them.

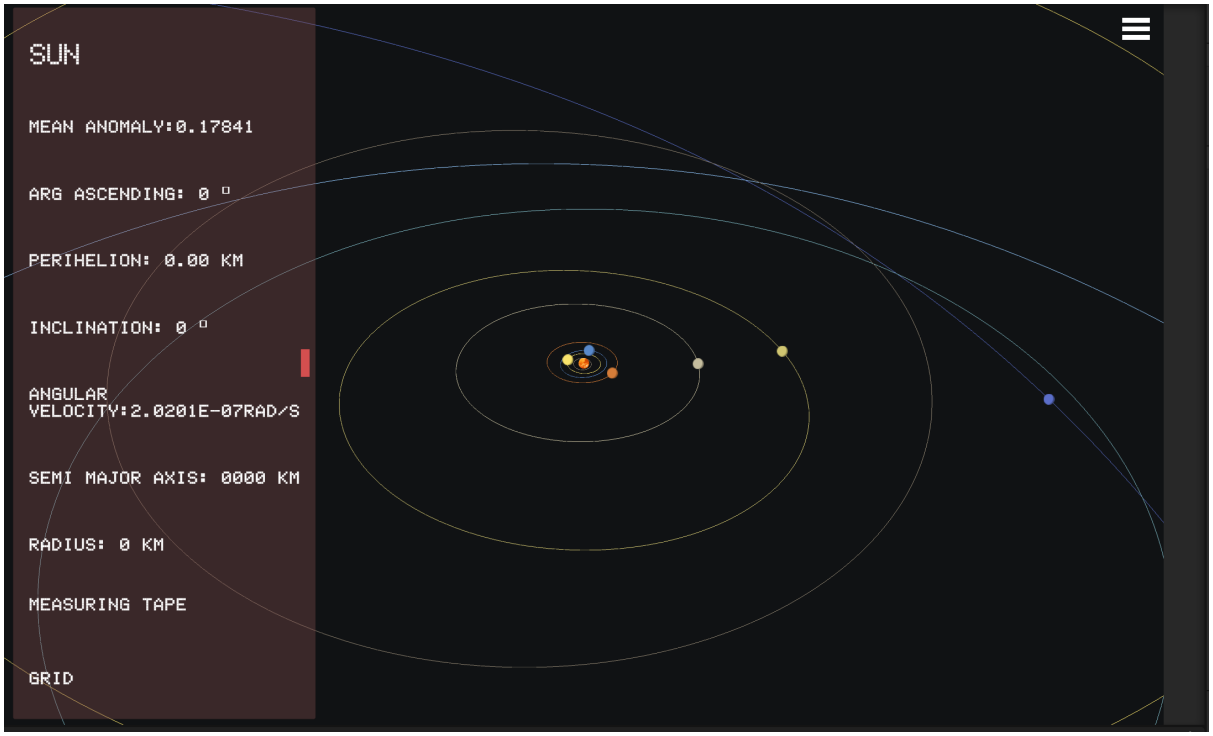


Figure 87:Unity Game Screenshot

I couldn't figure out why there was duplicates for only Saturn and Jupiter, so instead of using GL Lines I'm going to use Unity's Line renderer, this would be better as it will allow for the change of the line width however it may be more hardware intensive but since it's only the rendering a line of single colours instead of complex 3d images the extra load on the computer will not be noticeable for most computers and for computers with the recommended specifications shown in my success criteria

Test ID	Test Case	Test Data	Results	Passed
UI-01	Orbital Path	Jupiter & Saturn	They are duplicated	X

UI-01 Unity Implementation

To implement the line rendering using unity id first have to create a function that calculates points in which the small lines in the circular path should connect. As unity's line render, like openGL, cannot create curved lines or follow a planet.

With Unity's line renderer I don't have to use Newton's method to iterate through points, I can just use the eccentric anomaly to divide the orbit evenly. I would rather use the eccentric anomaly than newtons method as it will be less performance intensive as I will be using an already calculate value to estimate points rather than creating an additional method which will eat more CPU cycles, making the simulation more hardware intensive.

Since, I am going to be completely replacing the current implementation of calculating orbit lines I am going to create a new script completely separate form OrbitalBody called OrbitalUI.

In OrbitalUI, I am going to be making a function to calculate the points of the line using Keplerian elements. These quantities are updated within every frame of the simulation, so the more calculations I do on different values of data for different functions the more hardware intensive my simulation will be and using an already calculated value instead of recalculating something else would be an efficient way of reducing the CPU load.

Semi Constants

To create a balance between performance, accuracy, and usability I will be introducing semi-constants to the calculation of these lines. Semi constants are values that are derived from the orbital elements and remain somewhat constant for the duration of a single orbit, they may change over time due to perturbations or control manoeuvres but these changes are very minimal so they are calculated once and treated as constants values.

I am using semi constants because repeatedly calculating the position of an orbital body at any point in its orbit involves complex equations that are very hardware intensive especially when done multipole times. By precalculating certain values that remain constant for at least one orbital period, we reduce the number of calculations needed for each frame, thus improving performance. The use of abstraction by getting rid of the unnecessary complexities of calculating real values within every frame and instead turning them into semi-constants, the code becomes more maintainable and easier to test as there are significantly less calculations to look at and check.

I am going to be calculating semi constants in the orbital body script as I want to decompose my code properly and keep orbital calculations separate from the UI to make the script more readable and easier to update in the future.

Here is what my semi constant's function in orbital body will look like:

```
public void CalcSemiConstants()
    {
        SGP = parentBody.mass * gravConst;
        MAM = Mathf.Sqrt(SGP / (float)Mathf.Pow(semiMajorAxis, 3));
        TAC = Mathf.Sqrt((1 + eccentricity) / (1 - eccentricity));

        CosLOAN = Mathf.Cos(longitudeOfAscendingNode);
        SinLOAN = Mathf.Sin(longitudeOfAscendingNode);

        CosI = Mathf.Cos(inclination);
        SinI = Mathf.Sin(inclination);
    }
```

Figure 88: CalccSemiConstant method

This function will be called when the OrbitalBody script starts to initialise the constants as shown here:

```
private void Start() {

    SpaceTime.Instance.ScaleUpdated += UpdateScale;
    FindObjectOfType<InterfaceManager>().OrbitToggle += TogglePath;
    FindObjectOfType<InterfaceManager>().FullStart += TogglePath;

    SetScales();
    SetJ2000();
    CalcSemiConstants();
    AdvanceOrbit();

    //Debugging
    //OrbitalDebug debug = new OrbitalDebug();
    //debug.name = name;
    //debug.eccentricAnomaly = EccentricAnomaly(_meanAnomaly);
    //debug.trueAnomaly = TrueAnomaly(EccentricAnomaly(_meanAnomaly));
}
```

Figure 89: Start funciton for orbital body script

Test

Although the method seems quite simple I still want to test if the values being provided are correct, doing this test not only shows if the semi constants are accurate but also shows if the quantities used to calculate them are as well.

So in the CalcSemiConstants() function I added these lines of code:

```
    //Debbuging
    Debug.Log(name +" CosLOAN:"+ CosLOAN);
    Debug.Log(name +" SinLOAN:"+ SinLOAN);
    Debug.Log(name +" TAC:"+ TAC);
}
```

Figure 91: Semi Constant debugging code

And when the simulation is run I get these values in the console log for pluto:

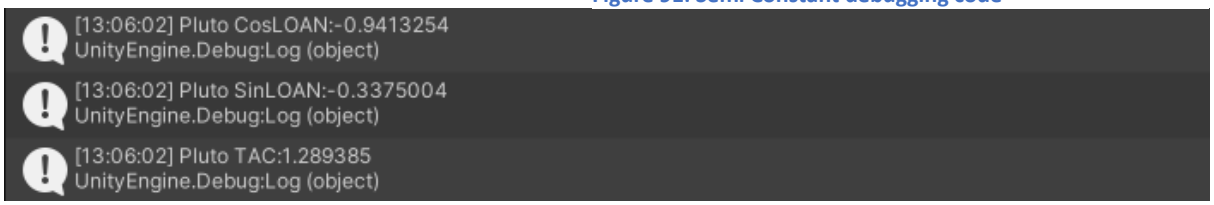


Figure 90: Semi constant debug results in console

When I calculate the sin and cos for longitude of ascending node, I got different values. This is because the longitude of ascending node is written in degrees instead of radians and the function treats it as if it is in radians. So, I will use Mathf.Deg2Rad to convert the values into

radians before finding sin and cos. Here is how I done it:

```
public void CalcSemiConstants(){

    float argrad = _argAscending * Mathf.Deg2Rad;
    float incrad = _inclination * Mathf.Deg2Rad;
    SGP = _stellarParent.gameObject.GetComponent<OrbitalBody>()._mass * G;
    MAM = Mathf.Sqrt(SGP / (float)Mathf.Pow(_semiMajorAxis, 3));
    TAC = Mathf.Sqrt((1 + _eccentricity) / (1 - _eccentricity));

    CosLOAN = Mathf.Cos(argrad);
    SinLOAN = Mathf.Sin(argrad);

    CosI = Mathf.Cos(incrad);
    SinI = Mathf.Sin(incrad);

    //Debbuging
    Debug.Log(name +" CosLOAN:"+ CosLOAN);
    Debug.Log(name +" SinLOAN:"+ SinLOAN);
    Debug.Log(name +" TAC:"+ TAC);
}
```

Figure 92: Updated CalcSemiConstants value

Now the values shown in the console correspond to my calculated values.

UpdateLines()

Update lines is a very short function that iterates through all the points in the point list one by one and updates the linerender's position.

```
private void UpdateLines()
{
    for (int i = 0; i < points.Count; i++)
    {
        lineRenderer.SetPosition(i, points[i]);
    }
}
```

Figure 93: UpdateLines() function

Calculating Points

Since I'm calculating points id first have to create a list to store them in. Then a function to actually come up with the points based on a given intensity. Here how I implemented the GetPoints() function and declared the points list:

```
using System.Collections.Generic;
using UnityEngine;
namespace solsyssim {
    [RequireComponent(typeof(LineRenderer))]
    public class OrbitLineRenderer : MonoBehaviour
    {
        [SerializeField] private LineRenderer lineRenderer;
        [SerializeField] private OrbitalBody orbitalBody;
        [SerializeField] private int resolution = 32;

        private List<Vector3> points = new List<Vector3>();

        public void GetPoints()
        {
            points.Clear();
            float eAnomaly = orbitalBody.EccentricAnom;
            float trueAnom = orbitalBody.TA;

            Vector3 centre = orbitalBody._stellarParent.transform.position; // Make sure this method exists in OrbitalBody

            float orbitFraction = 1f / resolution;
            for (int i = 0; i < resolution; i++)
            {
                float eAnomaly = i * orbitFraction * Mathf.PI * 2f;

                float trueAnomaly = 2 * Mathf.Atan(orbitalBody.TAC * Mathf.Tan(eAnomaly / 2f));
                float distance = orbitalBody._semiMajorAxis * (1 - (orbitalBody._eccentricity * Mathf.Cos(eAnomaly)));

                float cosAOPPlusTA = Mathf.Cos(orbitalBody._argPerihelion + trueAnom);
                float sinAOPPlusTA = Mathf.Sin(orbitalBody._argPerihelion + trueAnom);

                float x = distance * ((orbitalBody.CosLOAN * cosAOPPlusTA) - (orbitalBody.SinLOAN * sinAOPPlusTA * orbitalBody.CosI));
                float z = distance * ((orbitalBody.SinLOAN * cosAOPPlusTA) + (orbitalBody.CosLOAN * sinAOPPlusTA * orbitalBody.CosI));
                float y = distance * (orbitalBody.SinI * sinAOPPlusTA);

                points.Add(centre + new Vector3(x, y, z));
            }

            lineRenderer.positionCount = points.Count;
            UpdateLines();
        }
//.....
```

Figure 94: GetPoints() function

I first made sure that this script only works only objects that have the line render component, as doing this avoids unnecessary operations on other game objects without the line render component thus avoiding errors within my simulation.

When the GetPoints() function is first called all points from the list from any previous runs is first deleted to avoid extra points from being rendered which can be very intensive on the systems hardware. Then the centre of the orbit of the orbital body is found, which is the stellar parent of the body, so that the rendered vector positions can be added relative to the centre, like how the orbital body is travelling relevant to their stellar parent. Then they could be connected together to produce a trial behind the orbital body.

The vector positions are calculated using basic trigonometry, the radius and argument angles are used to find the position of the planet. The number of times the vectors are calculated is based on the resolution of the orbital line which I have set as 32 as it looked like the most sensible value when running the simulation through trial and error. Then, finally the number of position to be rendered are given to unity's line renderer and the UpdateLines() function is run.

Test Iteration 1

When trying to run the simulation I got this error:

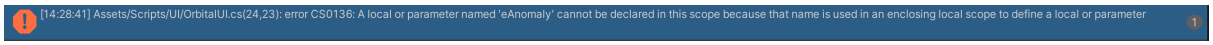


Figure 95: Console error log

This was because I was redeclaring eAnomaly within the for loop, to avoid this I changed the name of it to eAnom outside the for loop and now my simulation runs. But I don't see the lines at all, even though I have the line renderer component added to each orbital body in unity.

Test Iteration 2

To first see why I can't see any orbit lines I am going to print the values for x,y and z. with a. debug.log operation.
The values for X, Y, and Z is extremely large. When you look at the simulation there is a faint line of a planets orbit that is very far from the solar system ass seen in this image:

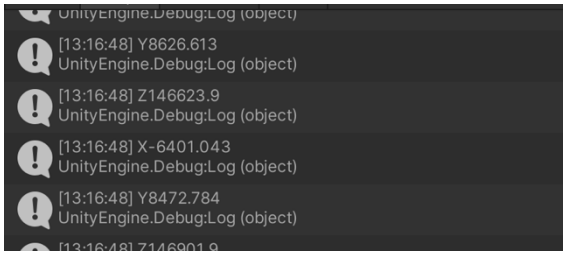


Figure 96: Console Debug Log

To combat this issue, I can try introducing a scale to the x, y, and z components to see if this is caused by a scaling issue rather than miscalculations.

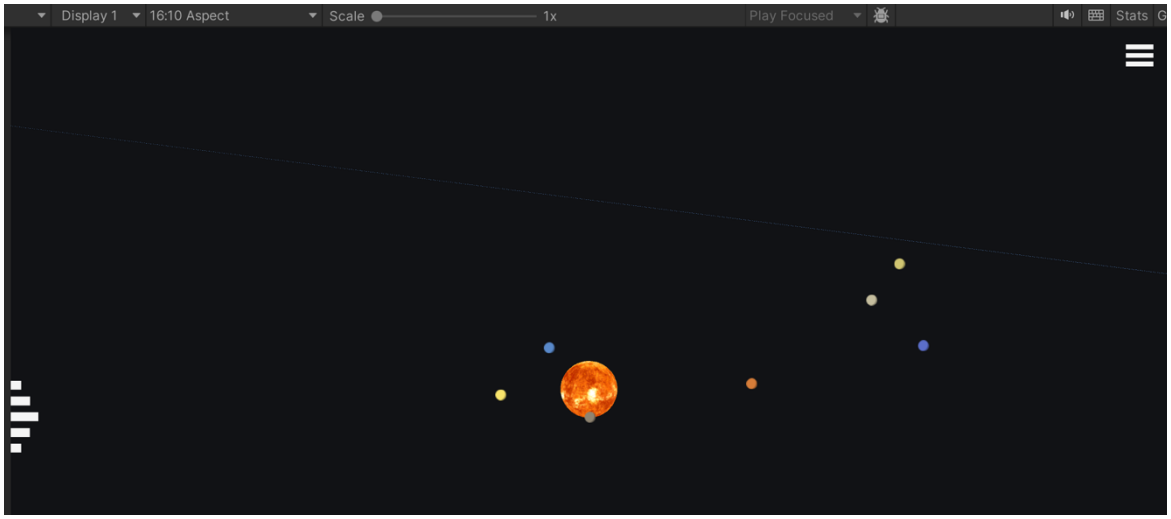


Figure 97: Faint orbit lines

Through trial and error, I found that multiplying the coordinates by 0.0001 allowed me to see the orbits and the values of the coordinates. As shown here:

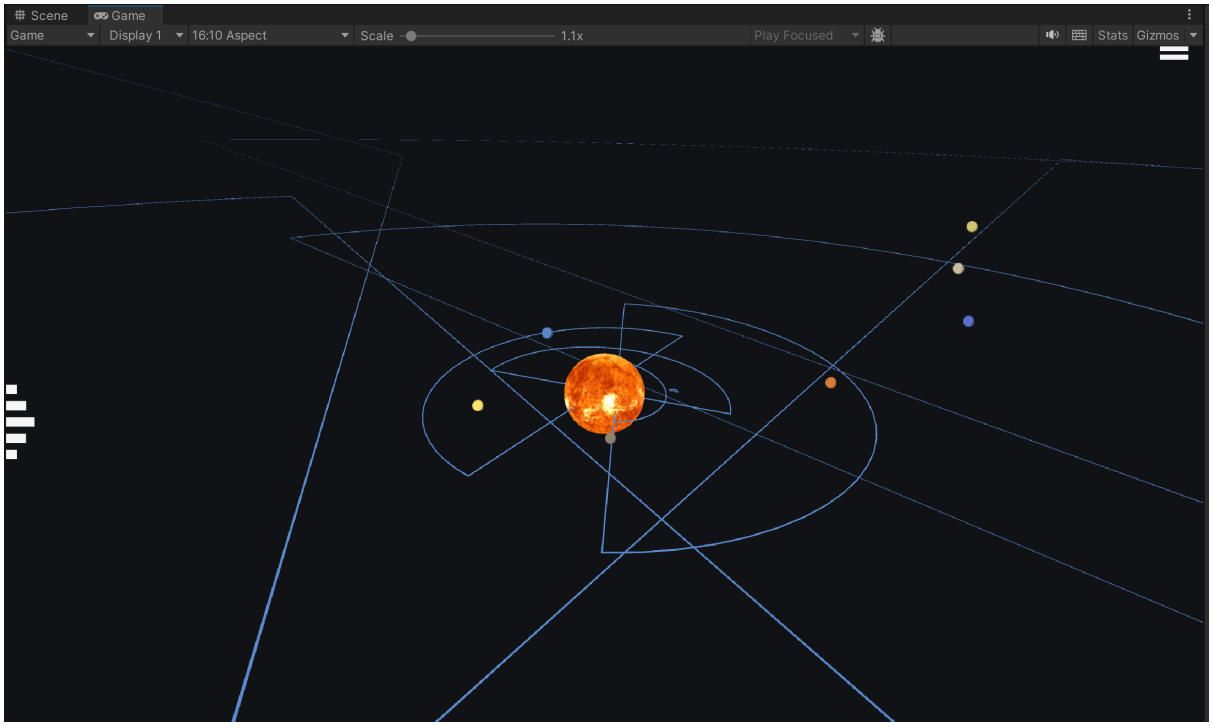


Figure 98: Abnormal orbit lines

The orbits are definitely wrong, they look abnormal. I think it is caused by a calculation error or the logic of my method is wrong as it doesn't seem to be a scaling issue, because if so my orbits will look circular.

Test Iteration 3

So, instead of calculating the positioning of the orbit lines separately from the celestial bodies I thought of utilising the GetPosition function in the OrbitalBody script, as I have a modular approach to my program I can reuse this component in a different script, saving me

time, and resources as less RAM will have to be used when calculating the position of the orbital bodies again for the orbital lines.

I nested GetPosition() into my GetPoints() function my code is much shorter and much easier to run. This makes the readability of my program better and also increases its performance.

```
public void GetPoints()
{
    points.Clear();

    Vector3 centre = orbitalBody._stellarParent.transform.position;

    float orbitFraction = 1f / resolution;
    for (int i = 0; i < resolution; i++)
    {
        float meanAnomaly = i * orbitFraction * 2f * Mathf.PI;
        Vector3 pointPosition = orbitalBody.GetPosition(meanAnomaly, "getpoints");
        points.Add(centre + pointPosition);
    }
    lineRenderer.positionCount = points.Count;
    UpdateLines();
}
```

Figure 99: GetPoints function GetPosition nesting implementation

When running the simulation with my updated code, I again don't see any orbit line. To check why I'm going to print the value from my GetPosition function to see what it returns for the orbit line.

Figure 100: GetPoints coordinate debug

I just get the coordinates of zero, but I don't know if this is for the orbit lines or for the planets so here is how I modified the GetPosition function to improve my debugging methods:

```
public Vector3 GetPosition(float M, string callerName) {

    float a = OrbitScaled; // semiMajorAxis
    float N = _argAscending * Mathf.Deg2Rad; // not const as might vary with precession
    float w = _argPerihelion * Mathf.Deg2Rad;
    float i = _inclination * Mathf.Deg2Rad;

    float E = EccentricAnomaly(M, _eccentricity);
    float TA = TrueAnomaly(E);
    float focusRadius = a * (1 - Mathf.Pow(_eccentricity, 2f)) / (1 + _eccentricity * Mathf.Cos(TA)); //distance from f

    Debug.Log($"{callerName} - Eccentric Anomaly: {E}");
    Debug.Log($"{callerName} - True Anomaly: {TA}");
    Debug.Log($"{callerName} - Focus Radius: {focusRadius}");
    // parametric equation of an elipse using the orbital elements
    float X = focusRadius * (Mathf.Cos(N) * Mathf.Cos(TA + w) - Mathf.Sin(N) * Mathf.Sin(TA + w)) * Mathf.Cos(i);
    float Y = focusRadius * Mathf.Sin(TA + w) * Mathf.Sin(i);
    float Z = focusRadius * (Mathf.Sin(N) * Mathf.Cos(TA + w) + Mathf.Cos(N) * Mathf.Sin(TA + w)) * Mathf.Cos(i);

    Vector3 orbitPoint = new Vector3(X, Y, Z);

    return orbitPoint;
}
```

Figure 101: GetPosition debugging with caller name

Because the function is called multiple times by both planets and the orbital line script. I added a caller name parameter to the function so I know from where it is being called so I don't mix up the position of orbit lines with the positioning of planets. Doing this makes me 100% sure when pinpointing where there is a problem as I can easily mix up which points the console is showing. When adding these caller names this is what my console looks like:

Figure 102: Console log verifying coordinated are from getPoints caller

The issue is clearly based on the OrbitalUi script, my planet positions are printing the correct values. Only coordinates from the GetPoints() function is wrong.
I think this is because GetPoints Is only being called at the start of the program, this is when everything is being initialised, and at the start the calculations in the OrbitalBody script haven't been done yet. This is most likely the reason why I am seeing zero as my positions.

To verify if this is true I created an update() procedure for my OrbitalUI script, so I can see if the orbits will show at a later point when orbital parameters have been calculated:

```
1 private void Update(){
2     GetPoints();
3     UpdateLines();
4 }
```

Figure 103: Orbital UI update script

When running the simulation I can see the orbit lines, and they look correct!

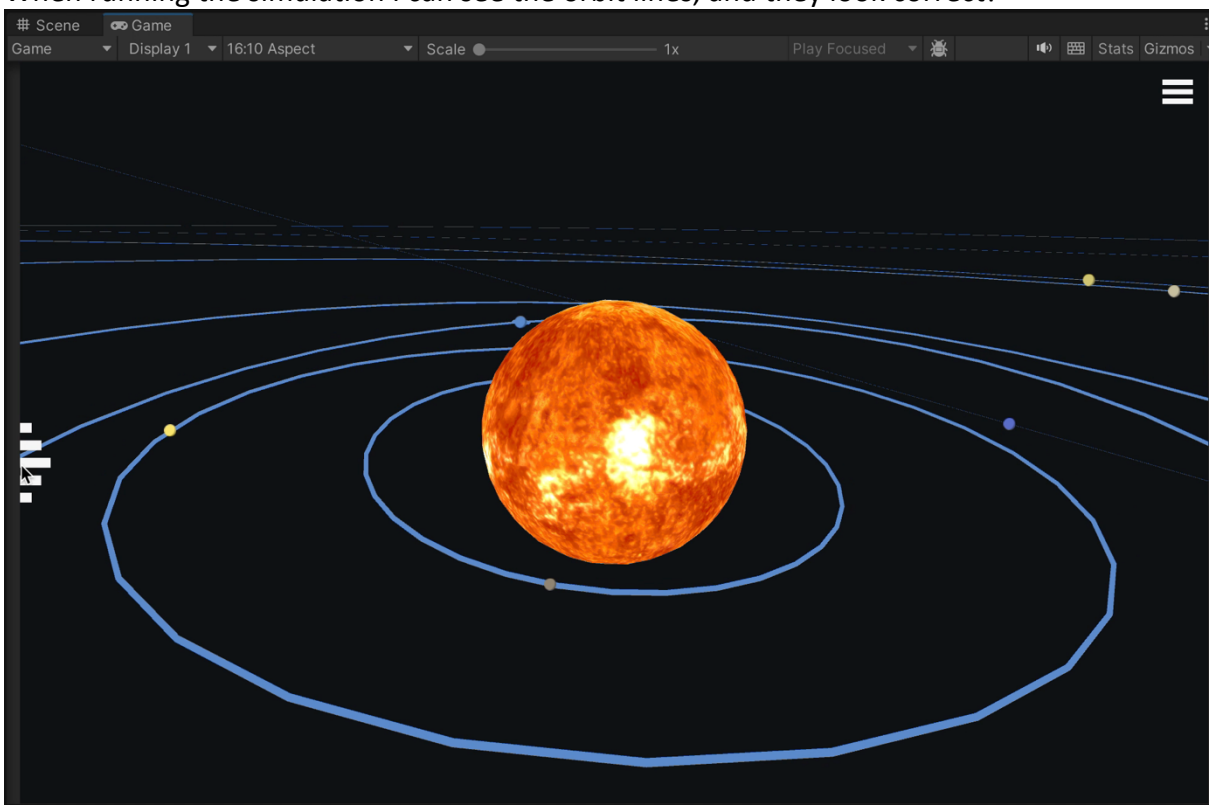


Figure 104: Correct orbit lines shown in game view

However, my computer is struggling a lot with keeping up with the orbit line calculations for each planet, the RAM is simply not enough.

Here is a video showing how much the simulation lags: https://youtu.be/Uw1G0PrWygU

Although this method is the only one that has worked so far it has taken away the functionality and interactivity of my simulation, it is now very hard to run, but now that I know that there is no issue with my code, and that I have verified that the issue is based on orbital parameters being zero at the beginning I can find a different way to make the orbit lines show.

Test Iteration 4

To ensure that the necessary values are there to help the OrbitalUi script create lines I have to change the time at which the script is executed. It must be executed after the orbital body script has calculated all the orbital body parameters.

I can try changing the script execution order in unity. I can make the OrbitalBody script start before the OrbitalUI one so that is has the correct orbital parameters. Here's how I did it in Unity's Projects Settings:

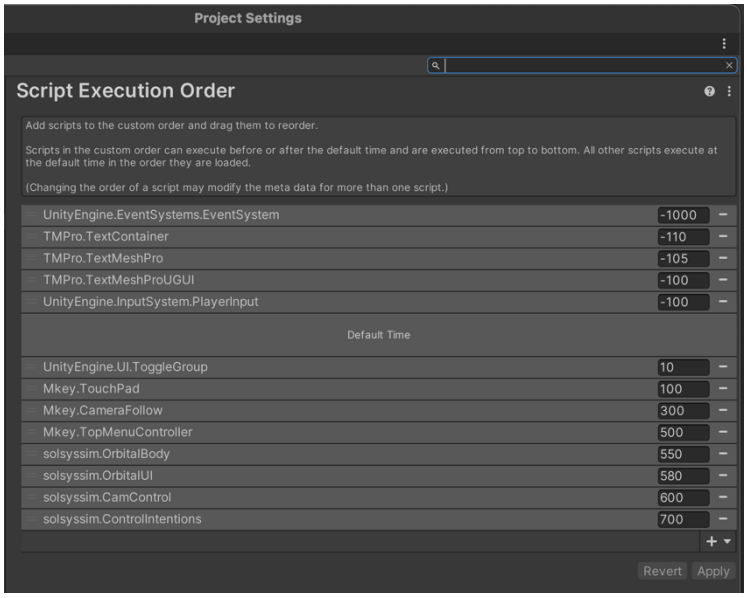


Figure 105: Unity script execution order settings

Now that I have changed my script execution order I can look if my simulation runs properly. As you can see in this video: https://youtu.be/qr1tTSQmJp4

The orbit lines are visible and in the correct positions, however there is only problem, the orbit lines are not updated with time, this means that the orbit for orbital bodies with stellar parents that move, their orbit lines will stay in a static position just like every other orbit line. In this case the moons orbit line just stays in one position.

Test Iteration 5

So, in my OrbitalUI class I am going to create an update function that updates the orbital line position for only the moon. Here is my code:

```
1 private void Update() {
2     if (orbitalBody._stellarParent.name != "Sun") {
3         GetPoints();
4         UpdateLines();
5     }
```

Figure 106: Orbital Line method with live moon orbit updates

For my if statement I made it so that it doesn't just apply to the moon but to all celestial bodies that don't have the sun as their stellar parent. I done because I am thinking ahead for when I need to add other celestial bodies in the future like satellite and moons for other planets not just earth.

Test ID	Test Case	Test Data	Results	Passed
UI-01	Orbital Path	Moon	Orbit line for all planets is shown and the orbit line	✓

			for the moon is travelling around earth.	

Default Settings

When I add user based customisation to my simulation In the options menu, I would want a reset() function to change the changes made to the orbit line creator back to default. Here is what I have set the default values as:

```
1 private void Reset()
2 {
3     lineRenderer = GetComponent<LineRenderer>();
4     lineRenderer.shadowCastingMode = UnityEngine.Rendering.ShadowCastingMode.Off;
5     lineRenderer.startWidth = 0.1f;
6     lineRenderer.endWidth = 0.1f;
7     lineRenderer.loop = true;
8
9     orbitalBody = GetComponent<OrbitalBody>();
10 }
```

Figure 107: Reset method showing default values for the orbit lines

By doing this I not only set a default to my orbital lines but I am also thinking ahead for if when I have an options menu and the user wants to change the UI for the orbit lines.

Collider

I want the user to be able to click on an orbital line of a specific planet and to be able to select a focus from it. To do this I'd have to add a collider to the rendered lines. I can skip this step however; planets are small and will be frustrating to click on so having the whole orbital line to be a collider means the user can easily focus on a selected body.

Creating colliders for my orbital bodies was easy as they were game objects in unity so I was just able to add the collider component to it. However, since the orbital line are a the linerenderer component of the orbital bodies and are not gameobjects I will need to code a function that generates a mesh collider for the orbital line in my OrbitalUi script.

Here is the GenerateMeshCollider() function:

```
1 public void GenerateMeshCollider()
2 {
3     MeshCollider collider = GetComponent<MeshCollider>();
4     if (collider == null)
5     {
6         collider = orbitalBody._stellarParent.gameObject.AddComponent<MeshCollider>();
7     }
8
9     Mesh mesh = new Mesh();
10
11    lineRenderer.BakeMesh(mesh, Camera.main, false);
12    collider.sharedMesh = mesh;
13
14 }
```

Figure 108: GenerateMeshCollider function

What it does is, it first checks if there are any mesh colliders present, if not, it generates a new mesh collider and adds the component to the stellar parent of the planet, then a new

mesh is generated using the BakeMesh function which generates a mesh based on the lineRenderer.

Test

To check if my mesh collider has been generated. I looked at the scene view in unity and this is what it shows:

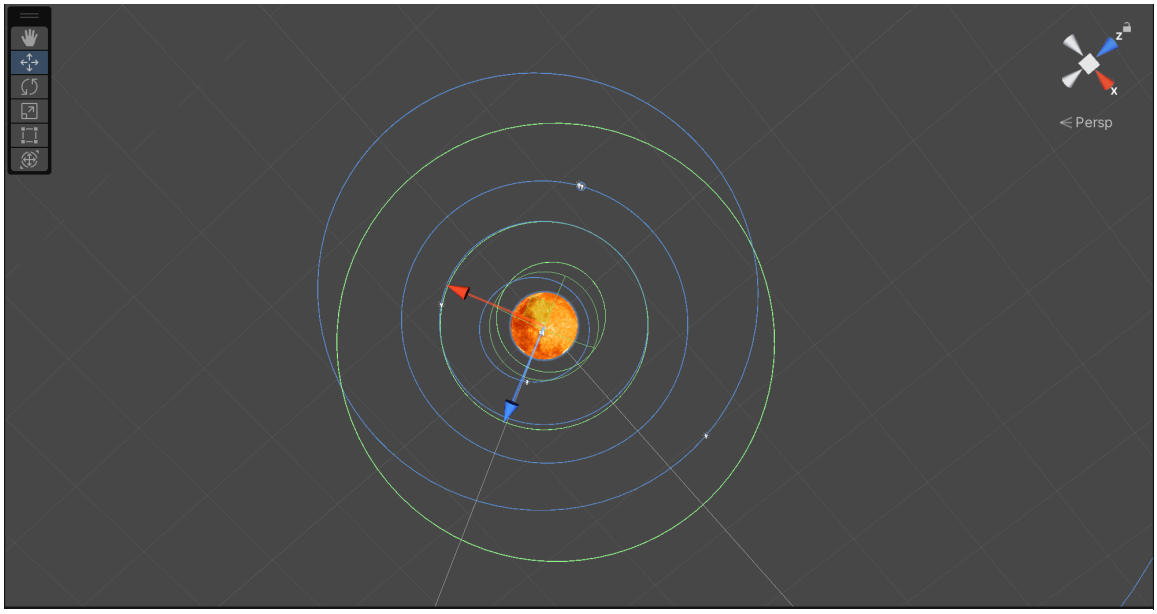


Figure 109: Scene view in unity showing offset mesh collider

The mesh collider isn't in the same position as the orbital line, they are offset. I'm not sure why as I don't have much experience coding with colliders so I'm going to not implement this feature.

Instead, I can make the planets bigger, or I can create a slider on the top of my menu with the images of the planets and select them from there like in my original design, this would be much less performance intensive as I wouldn't have to generate mesh colliders for every orbit, especially orbits that travel around another like the moons. So, I think having colliders for the orbit lines is quite an unnecessary feature and isn't 100% needed.

Grid Lines

I want to add a 3D grid within my simulation scene. This will help with users comprehending the difference in size and distance of the orbits.

To do this I'm going to be creating a new script that handles the grid in the simulation.

GridManager.cs

```
using UnityEngine;
public class GLGridRenderer : MonoBehaviour
{
    public Material lineMaterial;
    public int gridSize = 400;
    public float cellSize = 10.0f;
    public Color gridColor = Color.white;
    private void CreateLineMaterial()
    {
        if (!lineMaterial)
        {
            // Create a material that supports line rendering
            Shader shader = Shader.Find("Unlit/Color");
            lineMaterial = new Material(shader);
            lineMaterial.color = gridColor;
        }
    }

    private void OnRenderObject()
    {
        CreateLineMaterial();
        lineMaterial.SetPass(0);
        GL.Begin(GL.LINES);
        GL.Color(gridColor);
        DrawGrid();
        GL.End();
    }

    private void DrawGrid()
    {
        for (int y = -gridSize; y <= gridSize; y++)
        {
            for (int x = -gridSize; x <= gridSize; x++)
            {
                GL.Vertex(new Vector3(x * cellSize, y * cellSize, -gridSize * cellSize));
                GL.Vertex(new Vector3(x * cellSize, y * cellSize, gridSize * cellSize));
            }
            for (int z = -gridSize; z <= gridSize; z++)
            {
                GL.Vertex(new Vector3(-gridSize * cellSize, y * cellSize, z * cellSize));
                GL.Vertex(new Vector3(gridSize * cellSize, y * cellSize, z * cellSize));
            }
        }
    }
}
```

Figure 110: New GridManager script

I opted into using GL Lines for the grid instead of line render, this is because the grid is going to be the biggest component in the scene so far, it is going to go beyond the stars so I would need a very efficient way of rendering these lines. What this method does is it draws horizontal lines at multiple levels along the Y-axis, which creates a layered grid structure. This is done by iterating through a range given by gridSize, which determines the extent of the grid along the X, Y, and Z axes. For each layer along the Y-axis, the method draws two sets of lines: one set parallel to the X-axis and another set parallel to the Z-axis. These lines are spaced according to cellSize, ensuring uniform grid cells.

Test

When running the simulation after attaching the gridline component to a game object, this is what the simulation looks like:

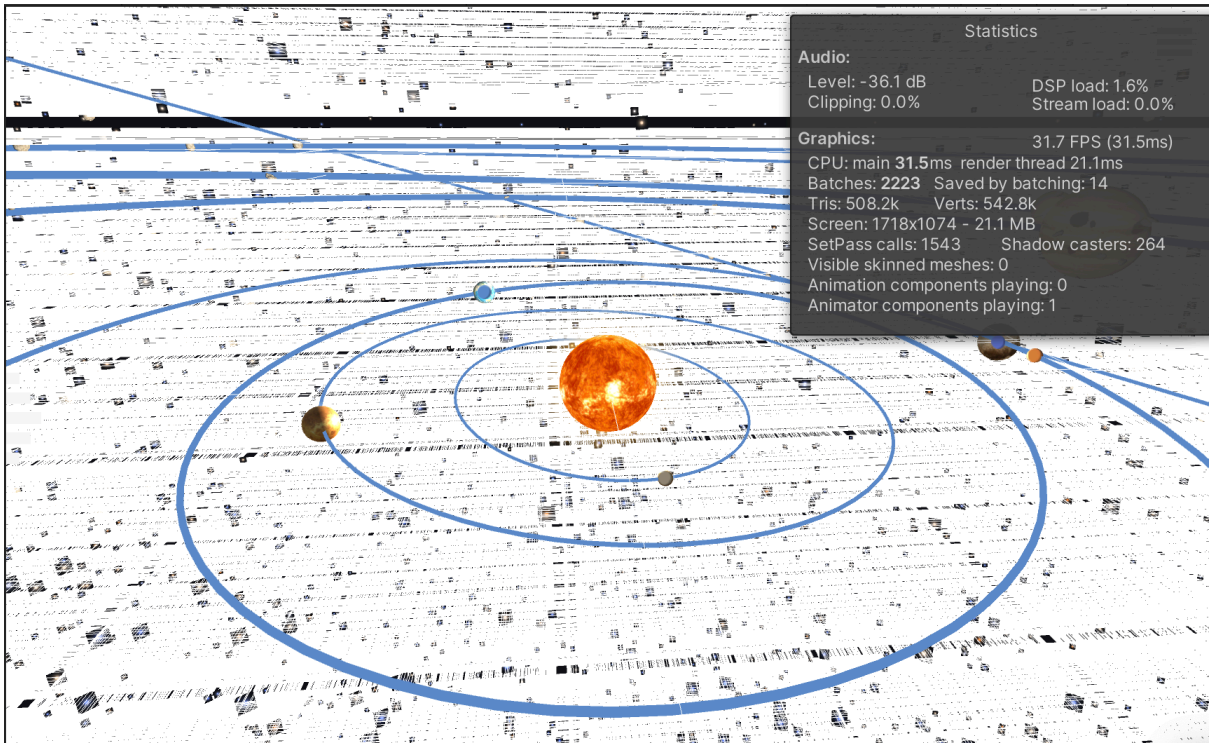


Figure 111: Unity game and stats view

The fps has dropped significantly from 100fps to about 30, also the grid lines don't work properly, the spacing of the grid lines have to be updated within every frame depending on the zoom of the camera. This would be code able, but it will eat too much processing power. So, I think I cannot sacrifice performance for grid lines which will not have much of an educational benefit. Therefore I will not be implementing it to my simulation.

Pause Menu

For the user to exit out of the simulation or go into the options menu whilst in the simulation, I'll need to add a pause menu interface within the simulation so that the user can access the options scene and the quit button.

This pause menu will be opened up by a button in the top right corner of the simulation as shown in figure 111.

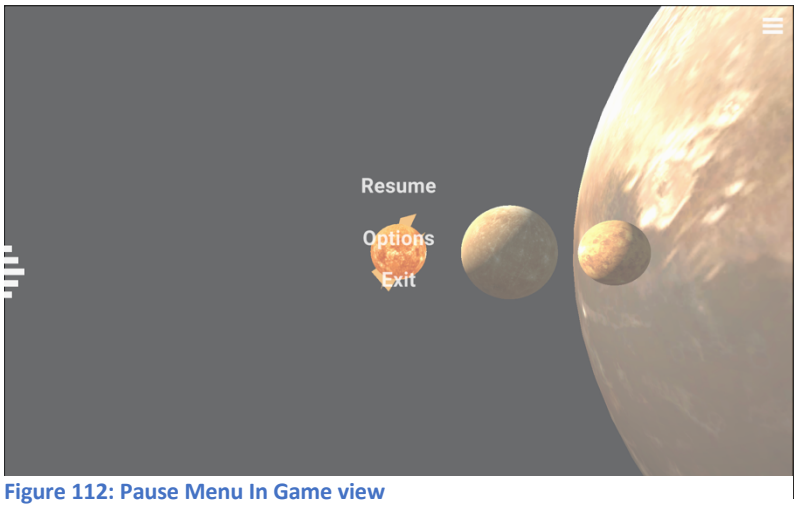


Figure 112: Pause Menu In Game view

Figure 113: Pause button icon

This is what pops up when the pause button icon is pressed, I made it a very minimal, simple easy to use pause menu. I kept the UI consistent with the same font and highlight/select colours for the button.

For these buttons to work I created a PauseMenu script:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class PauseMenu : MonoBehaviour
{
    public void PlayOptionsMenu(){
        SceneManager.LoadScene("OptionsMenu");
    }
    public void QuitGame(){
    Application.Quit();
    }
    public void PlayMainMenu(){
    SceneManager.LoadScene("MainMenu");
    }
}
```

Figure 114: Pause Menu script

It is a very short script; it provides the necessary functions needed so that when the buttons are pressed specific scenes are loaded using the scene manager. I chose to have a separate script for the pause menu instead of having a whole script for all scene changes because this allows for modularity, this means that if there were to be an error when switching scenes, I would be able to pinpoint in which scene management instance this is happening, also other menus have their own specialised instruction, for example the options menu script plays background music when it is loaded, if I wanted to share this script with my pause menu, there will be two instances of the background music playing and this would be annoying for the user to hear.

Test I-1 & I-2

Here is a video testing the pause menu:
https://youtu.be/kp-T674eVeA
As you can see in the video every aspect of the pause menu is functional.

Date Label

The final UI element I need to add is the date label, this will allow the user to see the current time in the simulation, it will start at epoch j2000 and will change with the time scale. So, if the user speeds up the simulation the date label will also move at a faster rate. To add a date label I created a separate script called DateCalc which calculates the year, month, day, hour, minute, and second from spaceTime in a readable format.

I decided to calculate the year and month separately, this is because of leap years, and also the fact that different months have different days. Time (hours, minutes, and seconds) always have a constant amount, the number of hours in a day doesn't change. Putting leap years into account when calculating time is very important, this is because it can lead to inaccuracy in time, maybe for the first year time may seem accurate but as it goes on the days unaccounted for will build up and later in time in the simulation the date will be wrong,

this is an error that cannot be ignored ad my simulation can be sped up, so the user will be faced with an inaccurate time very early on into the simulation.

First, I created the functions to compute the years:

```
private int ComputeYear(ref double pool, int baseYear) {
    int year;
    int dayCheck;
    int leapCheck = (IsLeapYear(baseYear)) ? 1 : 0;

    year = Mathf.FloorToInt((float)(pool / 365.2425f));
    dayCheck = Mathf.FloorToInt((float)(pool - (365 * year + (year - 1) / 4 - (year - 1) / 100 + (year - 1) / 400 + leapCheck)));

    if (dayCheck < 0)
        year -= 1;

    if (year <= 0)
        year = 0;
    else
        pool -= (365 * year + (year - 1) / 4 - (year - 1) / 100 + (year - 1) / 400 + leapCheck);

    return year + baseYear;
}

private bool IsLeapYear(int year) {
    if (year % 400 == 0)
        return true;
    else if (year % 100 == 0)
        return false;
    else if (year % 4 == 0)
        return true;
    else
        return false;
}
```

Figure 115: Compute Year Functions

The function takes in two parameters, one is the time pool which is the elapsed time in days, and the base year, which in our case is 2000, as the simulation starts at epoch 2000. It first checks if the first year is a leap year, then it divides the elapsed day by 365.2425 which is the average of how many days there are in a year. Then dayCheck is used to calculate the number of days it has been since the base year, it accounts for the number of days left after the sum of complete years have been removed (non leap years) then from the remaining sum is the days accumulated from leap years, this sum is then used to determine whether the base year is a leap year or not. And based on the daycheck the function adjusts how many years are to be added onto baseYear.
IsLeapYear checks if the current year is a leap year based on the rules of the Gregorian calander which is a year is a leap year if it is divisible by 4, except for years that are divisible by 100, unless they are also divisible by 400. This means that every 4th year is generally a leap year except for century years, which are leap years only if they are divisible by 400.

Then I created the ComputeMonth() function here it is:

```
private int ComputeMonth(ref double pool, bool leap) {
    int[] daysToMonthEnd = new int[] { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };
    int[] daysToMonthEndLeap = new int[] { 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 };
    int month;

    for (month = 1; month <= 12; month += 1) {
        if (!leap && pool < daysToMonthEnd[month]) {
            pool -= daysToMonthEnd[month - 1];
            break;
        } else if (leap && pool < daysToMonthEndLeap[month]) {
            pool -= (daysToMonthEndLeap[month - 1]);
            break;
        }
    }

    return month;
}
```

Figure 116: ComputeMonth function

The time pool and whether the year is a leap year or not is taken in as parameters. It has two lists, where each index is how many days it has been since the first month for every month, there is two because one is for leap years and the other list is for non-leap years. The function then iterates though months and subtracts the number of days in that month from the time pool, if the remaining time pool is less than the cumulative days to the end of the current month, then the current month has been found.

The function then subtracts the cumulative days up to the previous month from the time pool. This then reflects the remaining days within the current month, which is the day part of the date label.

Finally, I created a function to update the date label

```
private void UpdateDateLabel() {
    double timePool = SpaceTime.Instance.ElapsedTime;

    int year = ComputeYear(ref timePool, th);
    int month = ComputeMonth(ref timePool, IsLeapYear(year));
    int day = Mathf.FloorToInt((float)timePool) + 1;

    int hours = Mathf.FloorToInt((float)(timePool % 1 * 24));
    int minutes = Mathf.FloorToInt((float)((timePool % 1 * 24 - hours) * 60));
    int seconds = Mathf.FloorToInt((float)((((timePool % 1 * 24 - hours) * 60) - minutes) * 60));

    string dd = day < 10 ? "0" + day.ToString() : day.ToString();
    string mm = month < 10 ? "0" + month.ToString() : month.ToString();
    string yyyy = year.ToString();

    string timeString = $"{hours:D2}:{minutes:D2}:{seconds:D2}";

    _dateLabel.text = $"{dd}/{mm}/{yyyy} {timeString}";
}
```

Figure 117: UpdateDateLabel function

The UpdateDateLabel function helps show the current date and time in the game. It grabs the total time that's passed in the game from the SpaceTime instance. With this number, which is the total days since the game started, it figures out the year and moth by calling ComputeYear and ComputeMonth. Then the number of days Is calculated which is easy as the timepool is that itself but rounded down, so 1 is added as days start at 1, not 0. To find hours, minutes, and seconds the function breaks down the day's fraction into hours by multiplying by 24, then takes what's left to get minutes, and does it one more time to get seconds.

Now that all the data is ready a string format of dd mm yyyy is created. Days and months get a zero in front if they're less than 10 to keep things uniform. The year is straightforward, just four digits. For the time, it uses it rounds them to 2 decimal places using “D2” to make sure hours, minutes, and seconds always show up as two digits, even if they're less than 10.

Finally, it puts the strings into a readable format called _dateLabel.text, this variable is a parameter in the GetDateString function:

```
public string GetDateString()
{
    if (_dateLabel != null)
    {
        return _dateLabel.text;
    }
    else
    {
        return "DateLabel is not set.";
    }
}
```

What it does is, it checks whether there is a date label to set, else returns an error.

For the data label to be created and updated I created start() and update function for the DateCalc script:

```
private void Start() {
    _dateLabel = GetComponent<TMP_Text>();
}

void Update() {
    if (SpaceTime.Instance._timePause == false) {
        UpdateDateLabel();
    }
}
```

The start function gets the TMP text component where datecalc is going to be displayed. And, the update function checks if the simulation is paused, if not it updated the date label, otherwise the date label is not updated and stays as it is before the pause.

Test

In this video the data label is fully tested:
https://youtu.be/VraeGGB5jdg

The rate at which it is updated changes with the time scale as I speed up the simulation, and when I pause the simulation the date label also stops updating.

Review

The UI part of my solar system scene has met most of its criteria, I removed the scale bar and speed slider feature. I removed the scale bar feature because I couldn't implement the visual aids toggling, so it would've been a useless feature. I couldn't implement the visual aids toggling due to lack of time and because of how complicated some parts of it was. I removed the speed slider feature and made it so that the speed is controlled using the keyboard, taking this approach saved me time, made the simulation look less cluttery and more minimal.

Criteria No.	**Criteria**	**Criteria Met**
C-3	Consistent Design	Completed
C-6	Scene switching	Completed
C-11	Planet Orbit Line	Completed
C-12	Time counter	Completed
C-13	Planet Information	Completed
C-14	Scale Bar	Feature Removed
C-15	Visual aids toggling	Couldn't Complete, removed.
C-16	Speed Slider	Feature removed

Changes from my original design

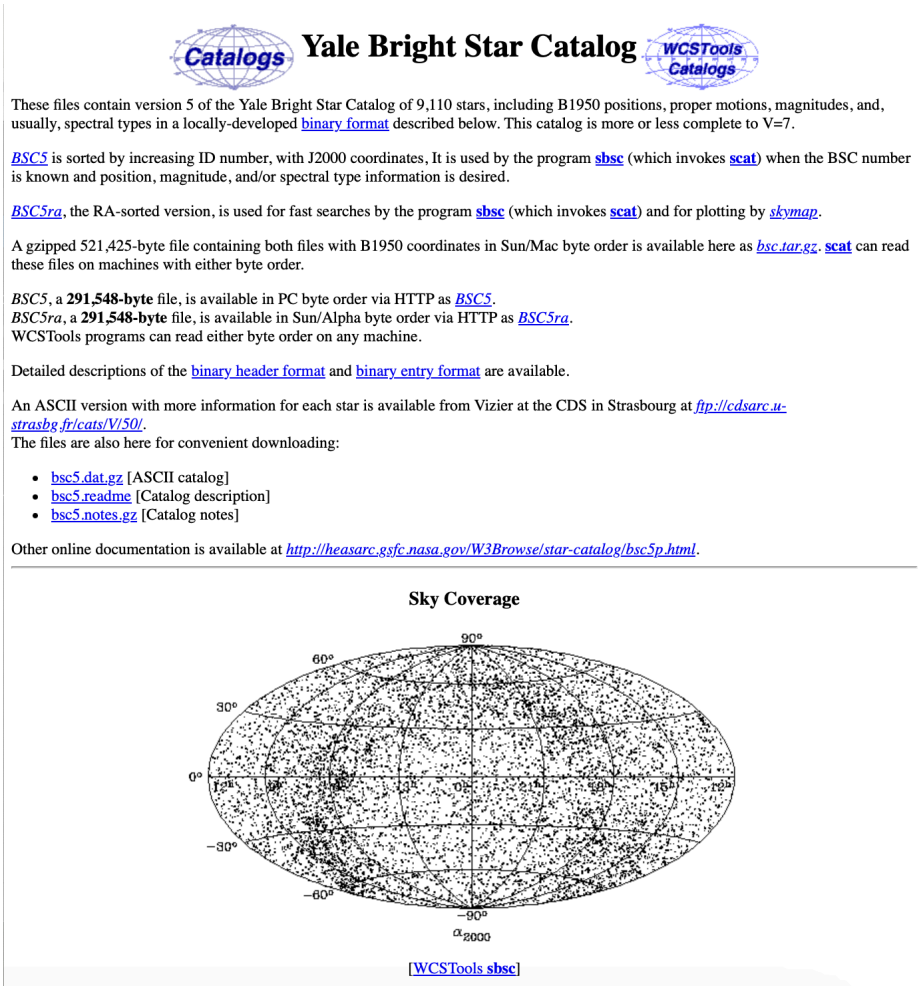
My UI has changed a bit from my original design, like I mentioned before some UI features were removed, however I outweighed these cons by improving some features, like the orbit line feature, where I made the orbit lines dynamically switch in thickness so that they can be seen from afar.

I also made it so that the sidebar is hideable, this gives a more immersive view of the simulation compared to having it statically placed and covering half the screen which would've made the simulation harder to use.

Star Sky

Instead of using a skybox, which is just an image that is turned into the sky, I want to create individual game objects for the stars and show constellations within the system. I chose to do this instead as the main aim of my project is to improve education of space, and since star constellations are a very popular but not really know part of this subject. I want to add it to my simulation, I first need data of all the known stars that can be seen from our solar system. I can find this data from the yale bright star catalogue, as shown in the image to the right, which contains 9110 stars that can be seen with the naked eye from earth.

To use this I'd have to download the BSC5 file, which has all the data in a binary format, then I have to put it into my resources folder in unity. Having the data in a binary format is very useful, because although the readability and the ability to modify the data is a bit limited, this format is much more compact and faster to read which is crucial for large datasets, like stars. Compared to a human readable format which would require a lot more translation taking up more CPU cycles which would make the simulation harder to run



Yale Bright Star Catalog

These files contain version 5 of the Yale Bright Star Catalog of 9,110 stars, including B1950 positions, proper motions, magnitudes, and, usually, spectral types in a locally-developed binary format described below. This catalog is more or less complete to V=7.

BSC5 is sorted by increasing ID number, with J2000 coordinates, It is used by the program **sbsc** (which invokes **scat**) when the BSC number is known and position, magnitude, and/or spectral type information is desired.

BSC5ra, the RA-sorted version, is used for fast searches by the program **sbsc** (which invokes **scat**) and for plotting by *skymap*.

A gzipped 521,425-byte file containing both files with B1950 coordinates in Sun/Mac byte order is available here as *bsc.tar.gz*. **scat** can read these files on machines with either byte order.

BSC5, a **291,548-byte** file, is available in PC byte order via HTTP as *BSC5*.
BSC5ra, a **291,548-byte** file, is available in Sun/Alpha byte order via HTTP as *BSC5ra*.
WCSTools programs can read either byte order on any machine.

Detailed descriptions of the binary header format and binary entry format are available.

An ASCII version with more information for each star is available from Vizier at the CDS in Strasbourg at *ftp://cdsarc.u-strasbg.fr/cats/V/50/*.
The files are also here for convenient downloading:

- bsc5.dat.gz [ASCII catalog]
- bsc5.readme [Catalog description]
- bsc5.notes.gz [Catalog notes]

Other online documentation is available at *http://heasarc.gsfc.nasa.gov/W3Browse/star-catalog/bsc5p.html*.

Sky Coverage

[WCSTools **sbsc**]

Figure 118: Yale bright star catalog sky coverage map

To manage this data, I will be creating a StarDataLoader script. This script will change the binary data format of the file into one that is readable and useable by unity

StarDataLoader.cs

First, I constructed a class to be used for each star. This is what it looks like:

```
public Star(float catalog_number, double right_ascension, double declination, byte spectral_type,
            byte spectral_index, short magnitude, float ra_proper_motion, float dec_proper_motion) {
  this.catalog_number = catalog_number;
  // Save the location parameters.
  this.right_ascension = right_ascension;
  this.declination = declination;
  this.ra_proper_motion = ra_proper_motion;
  this.dec_proper_motion = dec_proper_motion;
  // Set the position
  position = GetBasePosition();
  // Set the Colour.
  colour = SetColour(spectral_type, spectral_index);
  // Set the Size.
  size = SetSize(magnitude);
}
```

Figure 119: Star class

It initializes a Star object with given parameters, calculating its position, colour, and size based on the data in from the yale bright star catalogue. To find the position it calls the GetBasePosition() function and the instances variables that are defined in the contructor are passed into the function.

LoadData()

Before I try to do anything with the data, id obviously have to be able to load it into my simulation first. I will do this using the LoadData() function. Heres what it looks like:

```
public List<Star> LoadData() {
  List<Star> stars = new();
  // Open the binary file for reading.
  const string filename = "BSC5";
  TextAsset textAsset = Resources.Load(filename) as TextAsset;
  MemoryStream stream = new(textAsset.bytes);
  BinaryReader br = new(stream);
  // Read the header
  int sequence_offset = br.ReadInt32();
  int start_index = br.ReadInt32();
  int num_stars = -br.ReadInt32();
  int star_number_settings = br.ReadInt32();
  int proper_motion_included = br.ReadInt32();
  int num_magnitudes = br.ReadInt32();
  int star_data_size = br.ReadInt32();

  // Read one field at a time.
  for (int i = 0; i < num_stars; i++) {
    float catalog_number = br.ReadSingle();
    double right_ascension = br.ReadDouble();
    // Angular distance from celestial equator.
    double declination = br.ReadDouble();
    byte spectral_type = br.ReadByte();
    byte spectral_index = br.ReadByte();
    short magnitude = br.ReadInt16();
    float ra_proper_motion = br.ReadSingle();
    float dec_proper_motion = br.ReadSingle();
    Star star = new(catalog_number, right_ascension, declination, spectral_type, spectral_index, magnitude, ra_p

    stars.Add(star);
  }

  return stars;
}
```

Figure 120: LoadData() function

It begins by loading the binary file from the resources folder, then the file is converted into a TestAsset which is a flexible type that can handle binary data. Then I chose to create a memory stream, which stores the data in a byte array within memory, this allows for very fast random access, as the data doesn't have to be moved in and out of memory to be processed. Making it much

more efficient compared to having a copy of the data stored in a file in the SSD, which will have much more latency and will be slower.

Then a BinaryReader is instantiated which provides methods to read binary data in various formats, making it suitable for extracting data from a binary file. The function then reads the header of the file, this allows for the function to understand the structure of the file, it does this by assuming the file has a structured format, this is done using BinaryReader's built in br.ReadInt32() function which reads a 4 byte signed integer from the current stream, then advances the stream by 4 bytes, so that the next one can be read when the function is called again.

Once the structure of the data is understood, it is time to actually read the star data, pass it into the star constructor and return a list of instances of star. It does this by iterating through each field in the file, and reading different date types in those fields, i.e., declination would be a double, since it is a decimal point number, and spectral type is read as a byte because it is a string. Then these read parameters for each star is passed into the constructor and appended into a list of stars.

.NET classes like MemoryStream and BinaryReader are designed for efficient manipulation of binary data. MemoryStream allows for fast in-memory data access, and BinaryReader simplifies the process of extracting structured data from a binary stream. Thus, why I initially chose to convert the file into a binary format.

GetBasePosition()

The function calculates the star's position using trigonometry and projects it onto a sphere. It uses the right ascension and declination to place stars accurately on a celestial sphere representation. It does this by first calculating the x and z coordinates using the cosine and sine of the right ascension. Then the Y coordinate by using the declination's sine component.

The values of the x and z components are then adjusted for the declination, this is done so that when the stars are mapped, they are mapped as a sphere around the central axis rather than a cylinder. It does this by multiplying x and x by the cosine of the declination. The values are then returned.

SetColour()

Stars have different colours, obviously not completely different colours, but more like different shaded of white, for example a white with a blue or red hue to it. This is because of the doppler effect, stars that are moving away from the solar system have a red shift effect to them and stars that are moving toward the solar system have a blue shift effect. So, to have the most realistic simulation possible I will also be adding stars the simulation, this will also educate learner on the red and blue shift of stars which is part of their curriculum. Rendering nearly 10,000 stars in accurately places positions is already very performance intensive. Calculating colours based on physical properties like the temperature of the star would involve more complex physics-based methods to calculate the correct colour which is much more computationally intensive compared to a simple linear interpolation method which uses already existing data about the colour of the star to determine the exact

colour/shade/hue of the star. This introduces a trade-off with the accuracy of the exact colour of the star and the performance of the simulation, I think this trade-off is definitely worth it as the shade/hue of the stars are not a required for educational purposes, being able to pinpoints differences in colours is enough, so removing these details will reduce unnecessary complexities within my simulation.
This is what this method looks like:

```
private Color SetColour(byte spectral_type, byte spectral_index) {
  Color IntColour(int r, int g, int b) {
    return new Color(r / 255f, g / 255f, b / 255f);
  }
  Color[] col = new Color[8];
  col[0] = IntColour(0x5c, 0x7c, 0xff); // O1
  col[1] = IntColour(0x5d, 0x7e, 0xff); // B0.5
  col[2] = IntColour(0x79, 0x96, 0xff); // A0
  col[3] = IntColour(0xb8, 0xc5, 0xff); // F0
  col[4] = IntColour(0xff, 0xef, 0xed); // G1
  col[5] = IntColour(0xff, 0xde, 0xc0); // K0
  col[6] = IntColour(0xff, 0xa2, 0x5a); // M0
  col[7] = IntColour(0xff, 0x7d, 0x24); // M9.5
  // defualt value if no spectral type given
  int col_idx = -1;
  if (spectral_type == 'O') {
    col_idx = 0;
  } else if (spectral_type == 'B') {
    col_idx = 1;
  } else if (spectral_type == 'A') {
    col_idx = 2;
  } else if (spectral_type == 'F') {
    col_idx = 3;
  } else if (spectral_type == 'G') {
    col_idx = 4;
  } else if (spectral_type == 'K') {
    col_idx = 5;
  } else if (spectral_type == 'M') {
    col_idx = 6;
  }

  // If unknown, make white.
  if (col_idx == -1) {
    return Color.white;
  }

  // Map second part 0 -> 0, 10 -> 100
  float percent = (spectral_index - 0x30) / 10.0f;
  return Color.Lerp(col[col_idx], col[col_idx + 1], percent);
}
```

Figure 121: Color setting function

How this method works is, it retrieves the stars spectral type from the star catalogue, and linear interpolated between two different pre-set colours, that are defined using unity's colour constructor. These pre-set colours are made based on 8 different spectral types of stars, and they're rgb values are passed into unity's color constructor class.
I found the rgb values of these colours online in this article:
https://arxiv.org/pdf/2101.06254.pdf .

TABLE 5 Linear RGB and Hex color codes of solar metallicity main sequence stars for spectral types (SpT) M9.5V - O1V.

SpT	T_{eff}	log(g)	RGB	Hex
M9.5V	2300	5.0	1.0,0.491,0.144	#ff7d24
M9V	2400	5.0	1.0,0.518,0.179	#ff842d
M8V	2500	5.0	1.0,0.542,0.202	#ff8a33
M7.5V	2600	5.0	1.0,0.607,0.255	#ff9a41
M6.5V	2700	5.0	1.0,0.648,0.286	#ffa548
M6V	2800	5.0	1.0,0.649,0.285	#ffa548
M6V	2900	5.0	1.0,0.644,0.285	#ffa448
M5.5V	3000	5.0	1.0,0.641,0.289	#ffa349
M4.5V	3100	5.0	1.0,0.638,0.293	#ffa24a
M4V	3200	5.0	1.0,0.638,0.3	#ffa24c
M3.5V	3300	5.0	1.0,0.638,0.308	#ffa24e
M3V	3400	5.0	1.0,0.638,0.315	#ffa250
M2.5V	3500	5.0	1.0,0.637,0.322	#ffa251
M2V	3600	5.0	1.0,0.635,0.327	#ffa153
M1V	3700	4.5	1.0,0.637,0.34	#ffa256
M0.5V	3800	4.5	1.0,0.635,0.346	#ffa158
M0V	3900	4.5	1.0,0.636,0.354	#ffa25a
K8V	4000	4.5	1.0,0.641,0.369	#ffa35e
K7V	4100	4.5	1.0,0.65,0.389	#ffa563
K6.5V	4200	4.5	1.0,0.662,0.411	#ffa868
K5.5V	4300	4.5	1.0,0.677,0.439	#ffac6f
K5V	4400	4.5	1.0,0.696,0.47	#ffb177
K4.5V	4500	4.5	1.0,0.717,0.501	#ffb67f
K4V	4600	4.5	1.0,0.739,0.533	#ffbc87
K3.5V	4700	4.5	1.0,0.761,0.565	#ffc18f
K3V	4800	4.5	1.0,0.781,0.595	#ffc797
K3V	4900	4.5	1.0,0.802,0.626	#ffcc9f
K2.5V	5000	4.5	1.0,0.821,0.657	#ffd1a7
K1.5V	5100	4.5	1.0,0.84,0.691	#ffd6b0
K1V	5200	4.5	1.0,0.857,0.722	#ffdab8

Figure 122:Example of star spectral types

It has the rgb values corresponding to every spectral type of stars. So, what I did was I chose 8 different spectral types that are the most different in colour, and converted the rgb values given in the table to hex colour code values for the unity colour class. Then, I created a selection statement which gives the index of a colour from a list which corresponds to the spectral type of a star.

Then I calculate the interpolation factor which represents how far the star's colour should be between the two base colours. It takes the spectral index, subtracts the base value, and divided by 10 to normalise it to a range between 0 and 1.

With this method the colour of any star is not 100% accurate, there is a trade off with accuracy bit this trade-off is not noticeable to the human eye as two concurrent base colours look practically the same, as you go down the list of the colours the just get warmer. Also, the implementation of custom stars rather than using a wallpaper as a skybox was so that I can represent sky constellations properly in my simulation which will have an educational benefit, I don't think having accurate colours of the stars is as important in fact I could've just made all the stars white, but this will make the simulation less visually pleasing, and teachers won't be able to pinpoint the fact that stars have different colours due to the doppler effect.

Test ID	Test Case	Test Data	Results	Passed
S-11	Star Colour	Star Game Objects	Adds colours which represents how stars can	✓

			be different colours e.g. blue/red	

StarField.cs

Now that I have created a script to load my data in to the simulation, then to create classes of stars from. I need to actually present these stars in the simulation, I will do these using a separate script, which generates star game objects for the field. This is how the scrip starts:

```
using System.Collections.Generic;
using UnityEngine;
namespace solsyssim{
  public class StarField : MonoBehaviour {
    [Range(0, 100)]
    [SerializeField] private float starSizeMin = 0f;
    [Range(0, 100)]
    [SerializeField] private float starSizeMax = 5f;
    private List<StarDataLoader.Star> stars;
    private List<GameObject> starObjects;

    private readonly int starFieldScale = 400;

    void Start() {
      // Read in the star data.
      StarDataLoader sdl = new();
      stars = sdl.LoadData();
      starObjects = new();
      foreach (StarDataLoader.Star star in stars) {
        // Create star game objects.
        GameObject stargo = GameObject.CreatePrimitive(PrimitiveType.Sphere);
        stargo.transform.parent = transform;
        stargo.name = $"HR {star.catalog_number}";
        stargo.transform.localPosition = star.position * starFieldScale;
        stargo.transform.localScale = Vector3.one * Mathf.Lerp(starSizeMin, starSizeMax, star.size);
        Material material = stargo.GetComponent<MeshRenderer>().material;
        material.color = star.colour;
        starObjects.Add(stargo);
      }
    }
  }
}
```

Figure 123: StarField script

The script starts by initialising an instance of star data, which is stored in a new list called stars. Then the start function iterates through each star in the list, for each star It creates a spherical gameobject, then it sets the parent of this star to the sun, and labels the star with the catalogue number found in the star catalogue file, the positioning of the star is than changed based on a scale, the larger the scale the larger the field will be, thus further the position of the star from the parent. Then, linear interpolation is used again to set a random size of the star between the minimum and maximum size given. Finally, the material of the game object is set and the colour is changed based on the colour that was calculated I the star loader function, then added to the starObject list.

Test

When I run the simulation this is what it looks like:

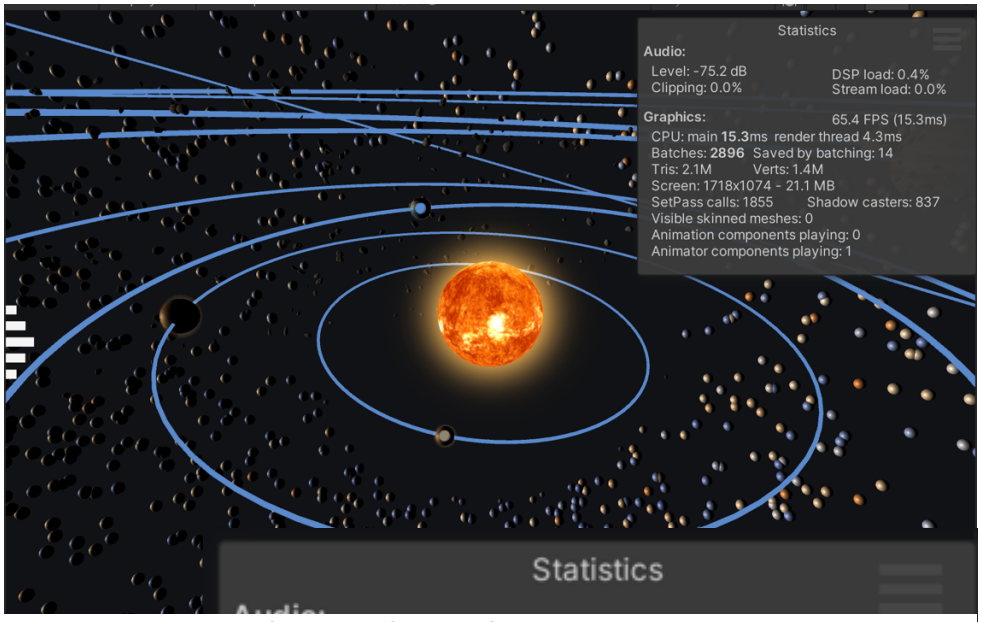


Figure 124: Game view with stars as spheres, and stats view

Right now, the stars are just spheres, there are no shaders added an they do not look very realistic

The fps of the simulation has dropped significantly compared to what it was like without the stars:

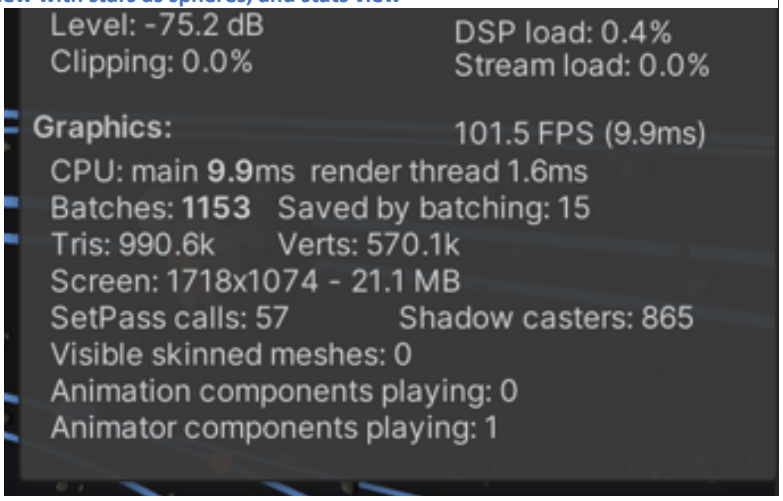


Figure 125: Closer look a stats view

This is about a 40 fps drop, the computer I'm running this simulation on is pretty high end, and with stars it is averaging 60fps. On computers with lower specs, rendering the stars will be a problem, the simulation may not be able to run. I would definitely have to solve this problem as not all educators and students are likely to have high performing computers, I also want my simulation to be accessible to a wide variety of demographics, I don't want the price of their computer to be a factor. So, I could try to change the way my game objects are rendered.

Start() Attempt 2

Rendering 3D object can be very performance intensive. Since each star I a 3d sphere, I think changing the type of object the sphere is to a quad can save a lot of performance. A quad is a 2d object that exists in a 3d space, what makes it replaceable to a sphere is the fact that it always faces the camera, this makes it seem like a 3d object when viewing it. A quad is more like a square, not a sphere, however when viewing from very long distance the shape of the star doesn't really matter, what is important is the light that comes out of the star. So, having a different shape for the game object wouldn't be important, as it wouldn't make the stars look drastically different.

Now, as you can see, the star field is made up of a bunch of squares that face one direction, I obviously would not want my stars to look like this.

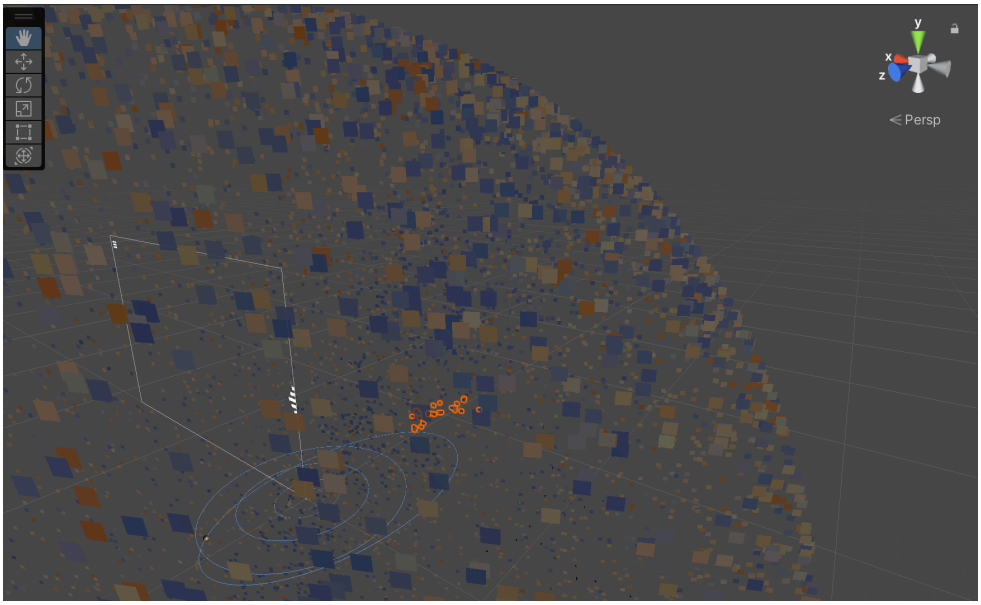


Figure 126: Unity star quad prefab view

So first I would have to make the stars actually face the sun, to do this I used the LookAt function so that the stars face the sun. This is the two lines I've added to the start method, I rotated the stars by 180º because when I first added the lookAt function for some reason it was doing the complete opposite the stars were facing away from the sun although they were directed to it.

```
stargo.transform.LookAt(transform.position);
stargo.transform.Rotate(0, 180, 0);
```

Figure 128: Cube prefab tansform code

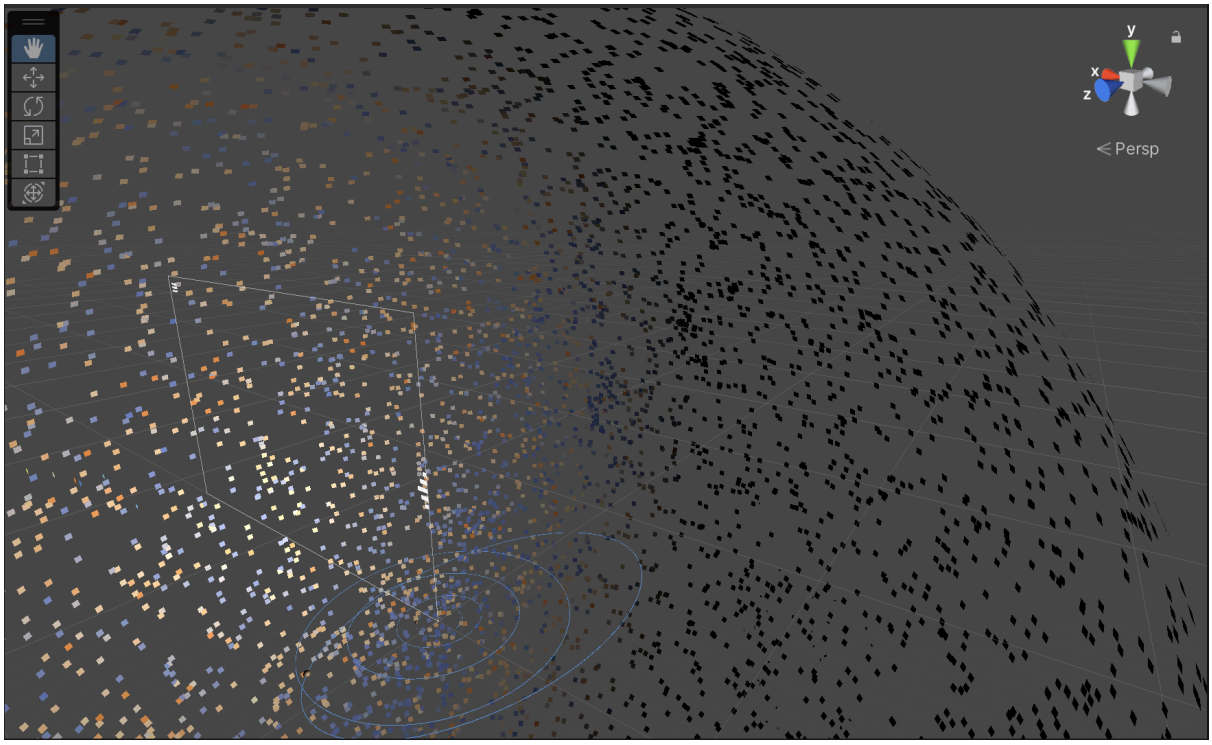


Figure 127: Unity scene cub prefabs facing correct direction

Now as you can see in Figure 121 the stars are facing towards the sun, however they are still squares. To change this, I will have to add a shader to them.

Now that I have added the shaders this is what the stars look like:

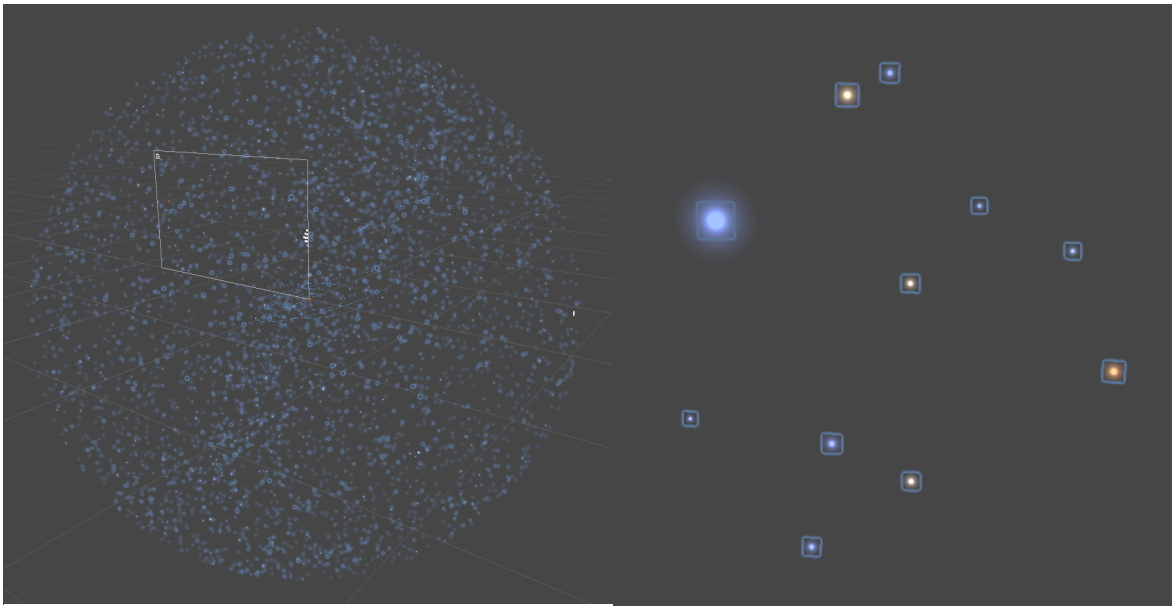


Figure 129: Cupre prefabs look view shaders in scene view

As you can see, I've made them smaller, and they now look like circles, with different colours, and sizes.

However, when I look at it from the cameras perspective you cannot see the stars, only at the corners you can.

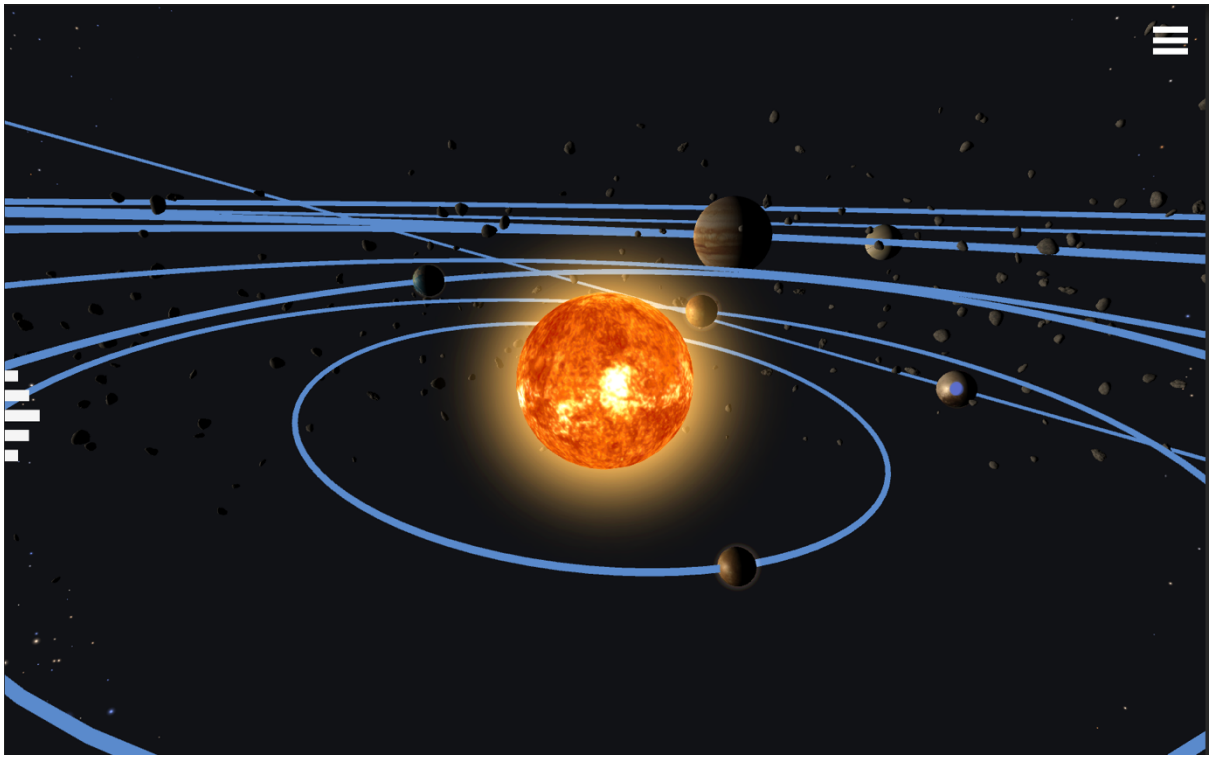


Figure 130: Unity game view, stars only showing in corners of screen

I think the problem lies within the camera, as when I look at the scene view, as I've previously shown, you can see the stars, but in the game view you cannot, since it is viewed from a camera.

Cameras in unity have clipping planes, this means that at certain minimum and maximum distances you cannot see objects in the scene. Since the stars are far away in the scene, it'd have to set the maximum clipping plane to a larger value so that when focused on any planet, and stars are a far distance form that focused body, which the camera would be on, I would want those stars to be viewed even though they are far away. So, I changed the "Far" value of the clipping plane to 3000 this value is definitely more than enough for the camera to recognise the existence of those stars further away from the camera.

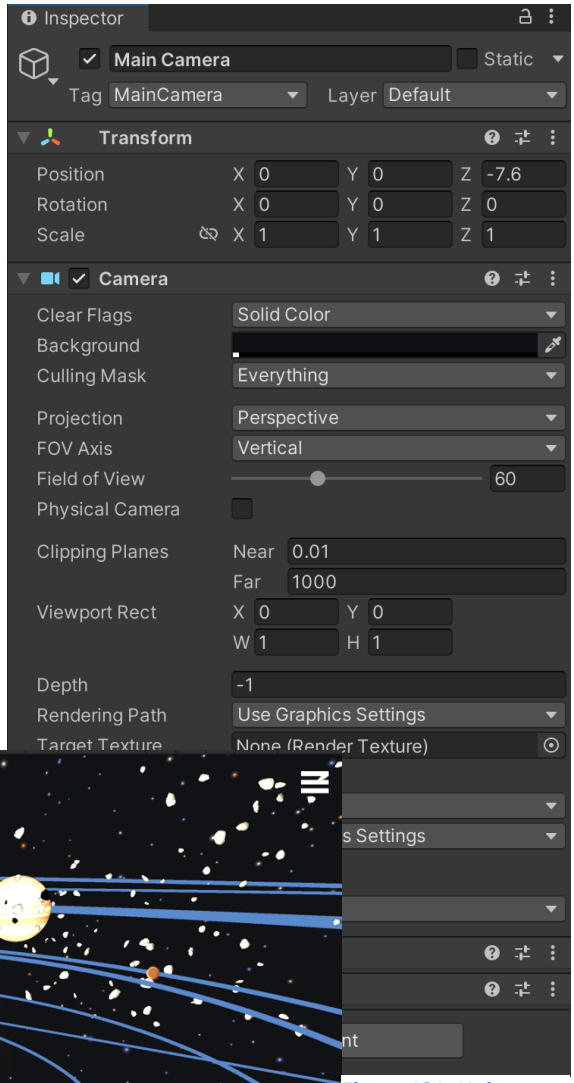


Figure 131: Unity camera component camera clipping plane settings

And now the stars are visible from any camera angle. As you can see here:

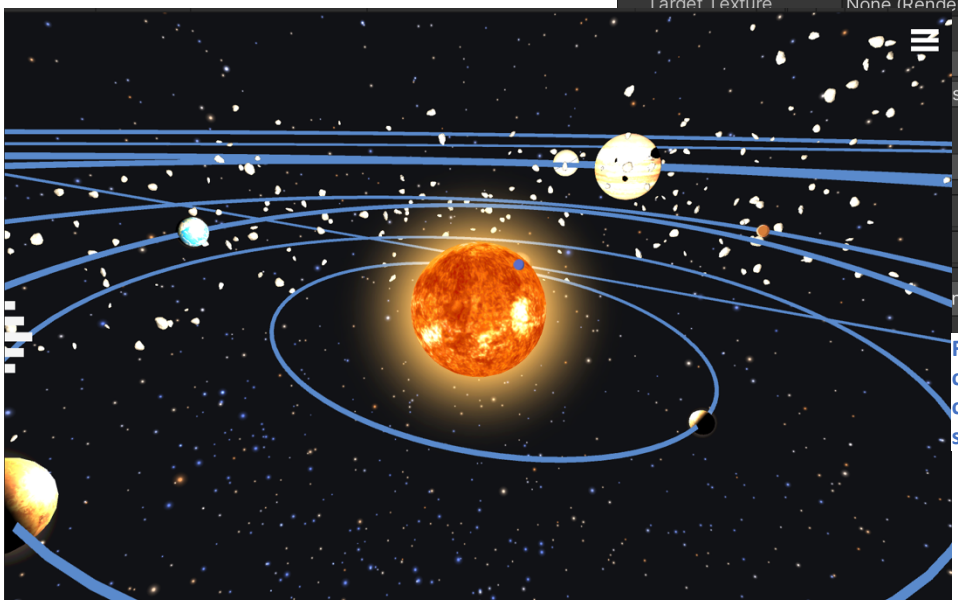


Figure 132: Unity game view with stars visible

Star Constellations

Since I already created my own skybox with each individual star as a game object it makes sense to actually utilise implementing it this way (compared to using an image) and make star constellations. I will make it so that the star constellations automatically spawn when the cameras ray box hits any stars within that constellation.

Constellation Data

Before I create any constellations, I would obviously have to find data on which stars the constellations exist within.

This is what the constellation data looks like each constellations has two lists, one list is the list of stars that actually exist within the constellations represented as their catalogue id's, and the 2nd list is the order in which the lines of the constellations are to be connected, each line made up of a pair of catalogue id's in the list. I had to implement a second list because if I connected lines, in a non-specific order the shape of the constellation will not match to what it actually is. To better explain this, I will show what it looks like if I mapped the constellation using the first list compared to the first.

```
private readonly List<(int[], int[])> constellations = new() {
  // Orion
  (new int[] { 1948, 1903, 1852, 2004, 1713, 2061, 1790, 1907, 2124,
               2199, 2135, 2047, 2159, 1543, 1544, 1570, 1552, 1567 },
  new int[] { 1713, 2004, 1713, 1852, 1852, 1790, 1852, 1903, 1903, 1948,
              1948, 2061, 1948, 2004, 1790, 1907, 1907, 2061, 2061, 2124,
              2124, 2199, 2199, 2135, 2199, 2159, 2159, 2047, 1790, 1543,
              1543, 1544, 1544, 1570, 1543, 1552, 1552, 1567, 2135, 2047 }),
  // Monceros
  (new int[] { 2970, 3188, 2714, 2356, 2227, 2506, 2298, 2385, 2456, 2479 },
  new int[] { 2970, 3188, 3188, 2714, 2714, 2356, 2356, 2227, 2714, 2506,
              2506, 2298, 2298, 2385, 2385, 2456, 2479, 2506, 2479, 2385 }),
  // Gemini
  (new int[] { 2890, 2891, 2990, 2421, 2777, 2473, 2650, 2216, 2895,
               2343, 2484, 2286, 2134, 2763, 2697, 2540, 2821, 2905, 2985},
  new int[] { 2890, 2697, 2990, 2905, 2697, 2473, 2905, 2777, 2777, 2650,
              2650, 2421, 2473, 2286, 2286, 2216, 2473, 2343, 2216, 2134,
              2763, 2484, 2763, 2777, 2697, 2540, 2697, 2821, 2821, 2905, 2905, 2985 }),
  // Cancer
  (new int[] {3475, 3449, 3461, 3572, 3249},
  new int[] {3475, 3449, 3449, 3461, 3461, 3572, 3461, 3249}),
  // Leo
  (new int[] { 3982, 4534, 4057, 4357, 3873, 4031, 4359, 3975, 4399, 4386, 3905, 3773, 3731 },
  new int[] { 4534, 4357, 4534, 4359, 4357, 4359, 4357, 4057, 4057, 4031,
              4057, 3975, 3975, 3982, 3975, 4359, 4359, 4399, 4399, 4386,
              4031, 3905, 3905, 3873, 3873, 3975, 3873, 3773, 3773, 3731, 3731, 3905 }),
  // Leo Minor
  (new int[] { 3800, 3974, 4100, 4247, 4090 },
  new int[] { 3800, 3974, 3974, 4100, 4100, 4247, 4247, 4090, 4090, 3974 }),
  // Lynx
  (new int[] { 3705, 3690, 3612, 3579, 3275, 2818, 2560, 2238 },
  new int[] { 3705, 3690, 3690, 3612, 3612, 3579, 3579, 3275, 3275, 2818,
              2818, 2560, 2560, 2238 }),
  // Ursa Major
  (new int[] { 3569, 3594, 3775, 3888, 3323, 3757, 4301, 4295, 4554, 4660,
               4905, 5054, 5191, 4518, 4335, 4069, 4033, 4377, 4375 },
  new int[] { 3569, 3594, 3594, 3775, 3775, 3888, 3888, 3323, 3323, 3757,
              3757, 3888, 3757, 4301, 4301, 4295, 4295, 3888, 4295, 4554,
              4554, 4660, 4660, 4301, 4660, 4905, 4905, 5054, 5054, 5191,
              4554, 4518, 4518, 4335, 4335, 4069, 4069, 4033, 4518, 4377, 4377, 4375 }),
};
```

Figure 135: Star constellation data

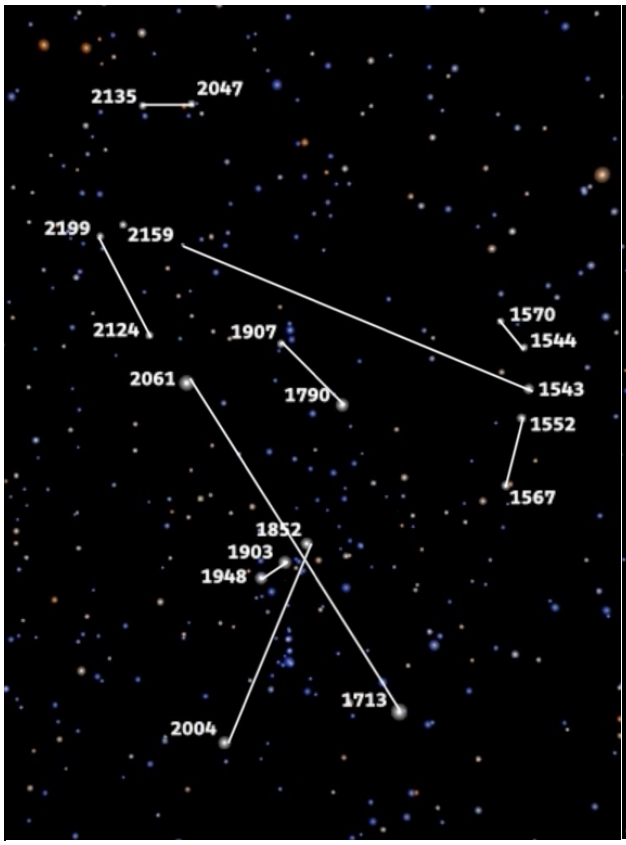


Figure 134: 1st List Constellation Mapping

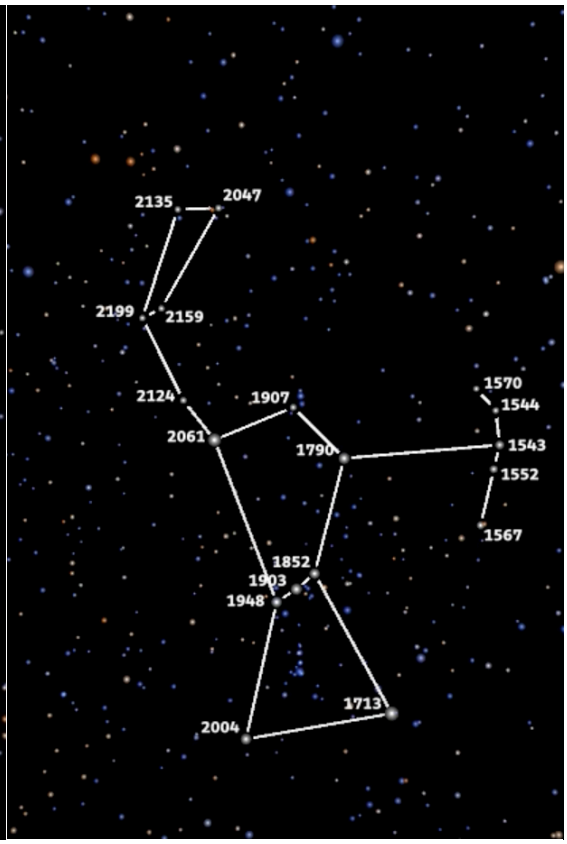


Figure 133: 2nd List Constellation Mapping

As you can see with the constellation mapping using the first list, it doesn't look correct, this is because the line rendering function will only connect lines in pairs sequentially. Whereas with the second list because the pairs are sorted sequentially and follows the logic of the line rendering function, the constellation looks as it should.

CreateConstellation()

Now that I have my constellation data ready in a format that will be suitable for a line rendering function, I can now create my constellations.

Here is what the code looks like:

```
private void CreateConstellation(int index) {
      int[] constellation = constellations[index].Item1;
      int[] lines = constellations[index].Item2;

      GameObject constellationHolder = new($"Constellation {index}");
      constellationHolder.transform.parent = transform;
      constellationVisible[index] = constellationHolder;

      for (int i = 0; i < lines.Length; i += 2) {
        GameObject line = new("Line");
        line.transform.parent = constellationHolder.transform;
        LineRenderer lineRenderer = line.AddComponent<LineRenderer>();
        lineRenderer.material = new Material(Shader.Find("Legacy Shaders/Particles/Alpha Blended Premultiply"));

        lineRenderer.useWorldSpace = false;
        Vector3 pos1 = starObjects[lines[i] - 1].transform.position;
        Vector3 pos2 = starObjects[lines[i + 1] - 1].transform.position;

        Vector3 dir = (pos2 - pos1).normalized * 3;
        lineRenderer.positionCount = 2;
        lineRenderer.SetPosition(0, pos1 + dir);
        lineRenderer.SetPosition(1, pos2 - dir);
      }
    }
```

Figure 136: CreateConstellation method

The function works by taking in an index, this index is representative to which constellation is to be mapped. Then this index is used to pass in the two lists into the function from the constellation data.

Before the lines are created a new gameobject is created which acts as a folder for all the lines that are to be created to form a constellations. These lines will be child components of the contellationHolder gameobject. I did this so I have a structured layout in my unity editor, this will be useful for when I need to edit or test any child component of the star, for example if I wanted to fix an asteroid belt, I wouldn't want to scroll through 10,000 stars before reaching the belt in unity.

Then the procedure iteratively goes through the stars list, in pairs, and creates a new line gameobject with its parent being the constellation holder, then a lineRenderer component is added to the line game object and its material is set. Then useWorldSpace is set to false, as I want the lines to move with the constellation holder's position.

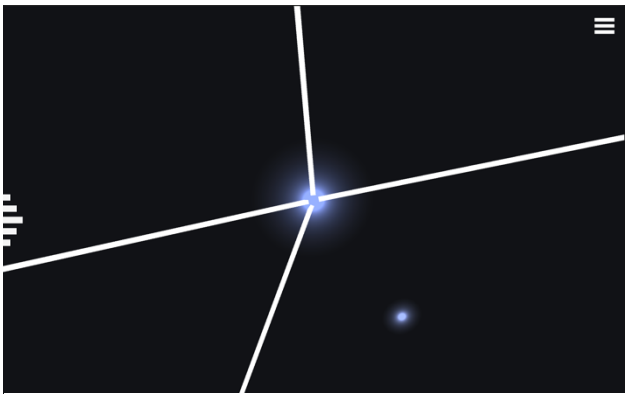


Figure 137: Star constellation lines not offest

I then set the position of the lines and created them, I also multiplied vector by 3, done this because I wanted to offset my lines. This is because the lines can go over the star a little bit, as you can see for stars with a lot of line nodes connected to them, most of the star is covered by the lines. To combat this, I can offset my stars which will make them start rendering a bit outside of the centre of the star so it doesn't look like the lines are touching it. Through trial-and-error multiplying by 3 was the best value to achieve this look as shown:

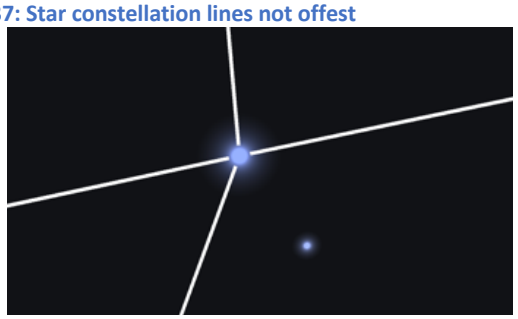


Figure 138: Star constellation lines offset

Now the lines are rendered properly and the stars under the constellation can now be seen.

Test ID	Test Case	Test Data	Results	Passed
S-12	Constellation	N/A	Lines connected to correct stars to give correct shape of constellation	✓

ToggleConstellation()

Some users may prefer not to see the constellations, so I'm going to make toggling them optional. I can do this by using the number keys in the keyboard as none of them are used. Because I initially set the constellations as lists of stars, I can just use the index of those lists to toggle a constellation.

This is what the Toggle Constellation procedure looks like. It first checks if the index is valid by checking if it is less than zero or greater than the number of constellations, if so, it will just return from the function meaning the if statement doesn't run to create a constellation.

```
1   private void ToggleConstellation(int index) {
2     // Safety check the index is valid.
3     if ((index < 0) || (index >= constellations.Count)) {
4       return;
5     }
6
7     // Toggle on or off.
8     if (constellationVisible.ContainsKey(index)) {
9       Destroy(constellationVisible[index]);
10      constellationVisible.Remove(index);
11    } else {
12      CreateConstellation(index);
13    }
14  }
```

Figure 139: Toggle Constellation method

If the index is valid then the constellationVisible list is checked to see if any constellations of that index already exists, if so they are removed, so the constellations is toggled off, else, if the index doesn't exist in the constellation visible list, a new constellations is created from that index.

The update function in this script constantly check is any number keys have been pressed so that toggleConstellation can be run.

```
1   private void Update() {
2     // Check for numeric presses and toggle the constellation highlighting.
3     for (int i = 0; i < 10; i++) {
4       if (Input.GetKeyDown(KeyCode.Alpha0 + i)) {
5         ToggleConstellation(i);
6       }
7     }
8   }
```

Figure 140: Star script update function

Test

Test ID	Test Case	Test Data	Results	Passed
I-8	Toggle Constellations	Constellations	When a key corresponding to a constellation is pressed, the constellation is turned on or off	✓

All star tests have been passed and now I can move onto another part of my simulation.

Review

Stars are something that weren't in my original design, I added them because they can be educationally beneficial, they show the Doppler effect and star constellations, this also makes the simulation look much more realistic and aesthetically pleasing.

The stars being accurately placed, with each star being a separate component makes my simulation more complex, in a good way, as I didn't use a picture background of stars which would've been slightly inaccurate and wouldn't have had star constellation features. Not implementing custom stars would've saved me time but I think the educational benefits outweighs the cons, and also I think my stakeholders would be really surprised with this feature and like it a lot as they haven't thought of it.

Camera

Right now, I have no camera controls implemented in my simulation. The main camera in the scene is placed directly in front of the sun, and the user cannot move it at all. Having a static camera which the user cannot move will make the simulation very hard to use and will deem the simulation useless because it removes the ability for students/educators to analyse specific celestial bodies' orbits and to look at the orbits in a bird's eye view to visually see parameters like eccentricity and inclination.

Implementing a static camera would be very simple, however it would be very restricting in an orbital motion simulation, so when coding a camera that the user can move, and zoom with I'd have to have complex animations that would allow for a dynamic change in scenery. To do this I will create an CamAnimator class which handles all the animations, and a separate CamAnimation class which holds the data and logic for the animation itself.

CamAnimator Class

Without a cam animator class, I wouldn't be able to focus on specific planets by moving my camera with them. So, to implement this focus feature I'd have to add animations that track planets and move the camera accordingly.

```
1  using UnityEngine;
2  namespace solsyssim {
3
4    public class CamAnimator : MonoBehaviour {
5
6        [SerializeField] private Camera _cam;
7        // checks if there is an animation
8        private CamAnimation _currentAnim;
9        public bool IsAnimating { get { return _currentAnim != null; } }
10       //constants
11       const float CamFinalPos = -20;
12       const float CamFinalRot = 25;
13       const float LerpIntensity = 0.1f;
14       const float LerpEnd = 0.1f;
15       const float AnimEnd = 0.1f;
16       //checks if there is an animation currently playing if not it plays the next anima

17       private void Update() {
18           if(_currentAnim != null) {
19               if (_currentAnim.AnimStatus >= AnimEnd) {
20                   _currentAnim.Animate ();
21               } else {
22                   NextAnim (warpToEnd:true);
23               }
24           }
25       }
26   }
27 }
```

Figure 141: Unity CamAnimator script

I started doing this by creating the camera animation handler which I added to the solsyssim namespace. I added these constants so the cameras final position at the end of an animation has a limit, because once an animation is finished and the user clicks on a different celestial body to focus on, I want the camera to be panned at a certain angle that makes sense.

I put my constants to use by creating a constructor for my class which would set specific values when focussing on a celestial body:

```
public CamAnimation (Transform cam, float camFinalDepthZPos = 0f, float poleFinalHorizontalYRot = 0f, float poleFi
    _cam = cam;
    _pole = _cam.parent;

    if(camFinalDepthZPos == 0f)
        camFinalDepthZPos = _cam.localPosition.z;

    _camFinalDepthZPos = new Vector3 (0, 0, camFinalDepthZPos);

    if(poleFinalVerticalXRot == 0f)
        poleFinalVerticalXRot = _pole.localEulerAngles.x;
    if(poleFinalHorizontalYRot == 0f)
        poleFinalHorizontalYRot = _pole.localEulerAngles.y;

    _poleFinalXYRot = new Vector3 (poleFinalVerticalXRot, poleFinalHorizontalYRot, _pole.localEulerAngles.z);
    _lerpIntensity = lerpIntensity;
    AnimStatus = 1f;
}
```

Figure 142: CamAnimation function

This constructor sets the initial camera position to the ones set as parameters in the CamControl script when CamAnimation is called, then it initiates an animation for the linear interpolation of the camera.

The parameters in the CamControl script is set to the Normal values I found in Tests C-5. I chose the default values of CamFinalPos (the distance) to be 20 and for CamFinalRot to be 25 (the angle) I chose this angle because it seems most natural, when an a user is selecting a body to focus on and the camera is position let's say I put an angle of 0 as the default it would be very silly as the user won't be able to see other celestial bodies as their FOV will be covered by the sun so they'd have to move the camera by themselves, this makes the simulation seem more unprofessional and more frustrating to use.

Test C-5 Normal

Found using trial and error for values, this value is most satisfactory as most orbits can be seen from the sun as focus in this specific angle.

CamFinalPos = -20

CamFinalRot = 25º

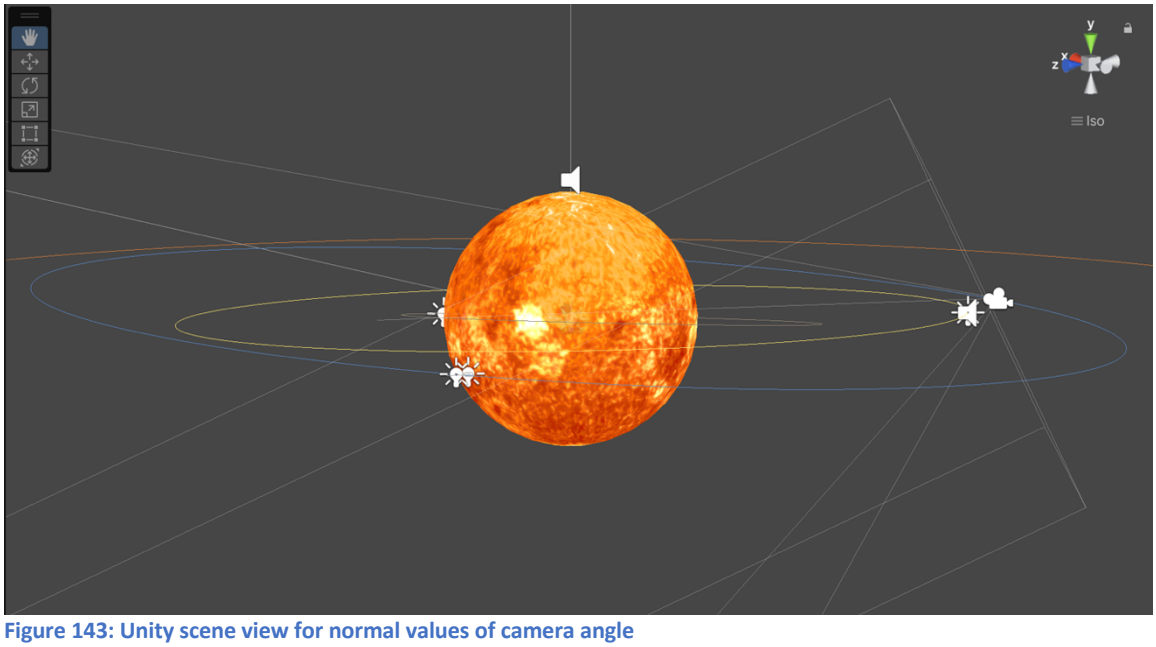


Figure 143: Unity scene view for normal values of camera angle

Link: https://youtu.be/ywIiVU9AOrw

Test C-5 Boundary

I decided these angles should be the boundary for these variables as you can see only a few orbits from the sun as a focus.
CamFinalPos = -70
CamFinalRot = 60º
Link: https://youtu.be/iDgKL0Yr3gs

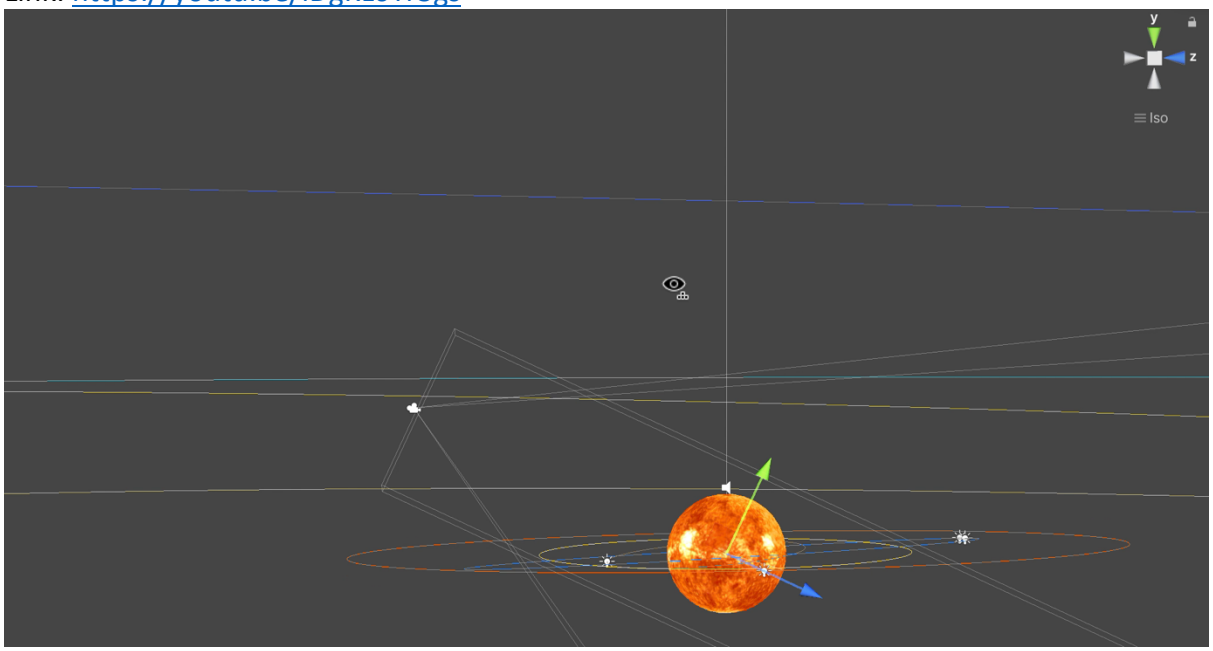


Figure 144: Unity scene view of boundary values of camera angle

Test C-5 Erroneous

These values for the variables causes logical errors as shown in the videos, where the orbit of the cam is very fast, it is very hard to study planets as it is moving constantly.
CamFinalPos = -200
CamFinalRot = 250º
Link: https://youtu.be/vLZUJFZ_TSA

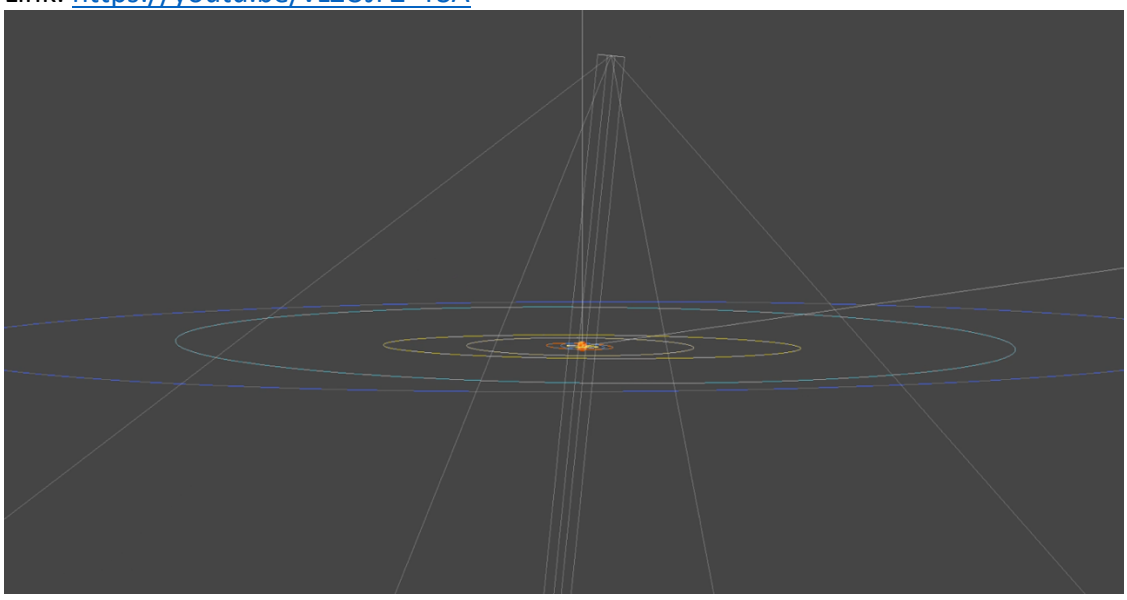


Figure 145: Unity scene view of erroneous values of camera angle

Review:

Based on the values for boundary and erroneous data for CamFinalRot and CamFinalPos, I chose the normal values for these variables to be CamFinalPos = -20, and CamFinalRot = 25º.

Animation() - Linear Interpolation

Linear interpolation techniques are used to provide a smooth animation for the camera when it is moving/zooming. It works by creating new points the camera can be in between the original point and the new location the camera wants to be in.

Here is how I used it in my code:

```
public void Animate() {
    // first we set the cam position on the pole
    Vector3 pos = _cam.localPosition;
    _cam.localPosition = Vector3.Lerp(pos, _camFinalDepthZPos, _lerpIntensity);
    Vector3 rot = _pole.localEulerAngles;
    // making sure the rotation stays between 0 and 360 for the lerping to work
    if (rot.x > _poleFinalXYRot.x)
        rot.x -= 360;
    float newRotX = Mathf.Lerp(rot.x, _poleFinalXYRot.x, _lerpIntensity);
    if (newRotX < 0)
        newRotX += 360;

    if (rot.y > _poleFinalXYRot.y)
        rot.y -= 360;
    float newRotY = Mathf.Lerp(rot.y, _poleFinalXYRot.y, _lerpIntensity);
    if (newRotY < 0)
        newRotY += 360;

    _pole.localEulerAngles = new Vector3(newRotX, newRotY, 0);

    //update the Animation status
    AnimStatus = Mathf.Max ((_cam.localPosition - _camFinalDepthZPos).magnitude, (_pole.localEulerAngles - _poleFi
}
```

Figure 146: Animate() function

Before using the linear interpolation function I had to make sure that the values for the angles entering the Mathf.Lerp() function are between 0-360 so I used if statements that remove/add multiples of 360 based on the value of the angle. I done this because the lerp function does not accept values outside of the range 0-360 and if I did put it in I will get errors and my simulation wouldn't work.

The LerpIntensity and LerpEnd constants influence the visual representation of the camera's movement as it tracks the celestial body's orbital motion. LerpIntensity increases the number of points created between the original and new point the camera wants to travel to and LerpEnd represents the threshold value at which the linear interpolation should end ensuring the camera reaches its final position Here's how these constants might affect the simulation:

LerpIntensity & LerpEnd Test:

Variable	Boundary	Normal	Erroneous
LerpIntensity	With a low LerpIntensity, the camera's movement would be slow and gradual. The camera would smoothly follow the body's path, providing a visually pleasing representation of the orbital dynamics.	Moderate Value: At a moderate LerpIntensity, the camera's movement would be reasonably paced, allowing it to smoothly track the celestial body's orbital motion without appearing too slow or too fast.	A high LerpIntensity would cause the camera's movement to be rapid, potentially resulting in a quick and abrupt tracking of the celestial body's orbital motion. Making the camera's movement appear less smooth and more jittery.
LerpEnd	A low LerpEnd would cause the camera's interpolation process to end very quickly, resulting in abrupt transitions. This could lead to less smooth and visually appealing representations of the orbital dynamics.	At a moderate LerpEnd, the camera's interpolation process would smoothly and gradually track the celestial body's orbital motion, providing a visually pleasing and seamless representation of the orbital dynamics.	A high LerpEnd would make the interpolation process very long, causing the camera to continue moving even after it has reached its final position. This would be very resource dependant and make the program hard to run.

By testing through different values of LerpEnd and LerpIntensity I found satisfactory values for the angles, I calculated the new rotation of the camera for both its x and y components created a new vector3 component for the pole of the camera. I then updated the AnimStatus which represents the animation status based on the remaining distance to be covered by the linear interpolation(if greater than zero the animation plays, when it reaches zero it stops). It takes the maximum magnitude of the differences between the current and final positions of the camera and the parent "pole", and assigns it to the AnimStatus.

WarpToEnd()

The WarpToEnd() function warps (teleports) the camera to the starting position when focusing on a celestial body. I done this by first checking if there is a camera to teleport and if so, when called by the update method it updates the focus of the camera and also sets the AnimStatus variable to zero. By doing this it ends the animation that was playing previously

as in the update function the value of AnimStatus is check and if equal to zero it move onto the next one (so the focus body that is pressed).

```
public void WarpToEnd () {
    if (_cam != null) {
        _cam.localPosition = _camFinalDepthZPos;
    }

    if (_pole != null) {
        _pole.localEulerAngles = _poleFinalXYRot;
    }

    AnimStatus = 0f;
}
```

Figure 147: WarpToEnd function

CamControl Class

To create more complicated camera controls, I'd have to code them myself rather than using Unity's pre-set ones. So, I created the CamControl class and added it to the solsyssim namespace. First, I assigned references to the main camera in Unity so that it could be controlled by my script. Then I created constants for the cam/zoom speed etc. as you can see here:

```
namespace solsyssim {

    public class CamControl : MonoBehaviour {
        [SerializeField] private Camera _cam;
        private Transform _camPole;
        private CamAnimator _camAnimator;

        // cam control togle
        private bool _userControl = false;

        // cam focus variable and event action
        private Transform _selectedBody;
        public event Action<Transform> NewFocus;
        const float CamSpeed = 25f;
        const float ZoomSpeed = 2f;
        const float ZoomMin = -0.1f;
        const float ZoomMax = -500.0f;
    }
}
```

Figure 148: CamControl class initialisation in solyssim namespace

Obviously, this much code wouldn't be enough to have a moving camera, so first instead of having the camera locked into the sun and not being able to have the camera focus onto a celestial body I implemented a focusing feature which pans the camera on a bird's eye view at an angle, from the celestial body, so that the user can not only see the celestial body that they are focusing on but the other planets besides/behind it as well. Implementing it this way means that users can actively analyse the motion of planets form other perspectives and also be able to comprehend distances of planets from earth, which they can't do in real life due to the earth's atmosphere and light pollution form cities. Being able to visually see these distances will be the best way for a student to understand the size of the solar system.

I done this by creating the start function which subscribes to the ControlIntention events:

```
private void Start() {

    _camPole = _cam.transform.parent;
    _camAnimator = GetComponent<CamAnimator>();

    ControlIntentions.Instance.CamRotation += RotateCam;
    ControlIntentions.Instance.CamTranslation += TranslateCam;
    ControlIntentions.Instance.FocusSelection += CheckSelection;
    ControlIntentions.Instance.MenuCall += PanCam;
    // Set the first body transform to istelf (null) to avoid error with cam zoo

    _selectedBody = transform;

    }
```

Figure 149: Start function for camControl class

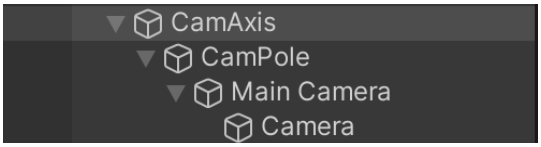


Figure 150: Unity scene hierarchy

And assigns them to specific functions I created in the CamControl class. The start procedure first assigns the _camPole variable to the parent of the actual camera. Instead of having just a main camera I had to create a multi-level camera system so that my camera can follow objects when it focuses on one. For example, _camPole has been made as an separate object from the camera axis in unity, this implementation of a multi layered camera means that the actual camera can move around its axis instead of moving the whole axis of the camera. I chose to do this to allow for more complicated camera control, other implementations of this would be moving the camera independently from the axis and having it not be a child of the _camAxis object. However, this implementation would be very hard to do as it can lead to a larger room for errors, because the objects not being attached together means there will be a very likely chance of miscommunication between them, e.g., _camPole may translate its self around a planet's origin with a different instance of the axis (the axis' centre may not be the same as _camPole's).

The _camAnimator variable's function will be explained in the _camAnimator class.

I've assigned the _selectedBody to "transform "(the camPole) to avoid a null error when linearly interpolating the camera with the selected body's vector position to allow for a smooth transition between bodies in the update() function.

Update()

The only thing that has to be constantly checked in the camera control class is if the selected body has changed.

```
1 private void Update() {
2     // Changes the positon of the cam axis to the selected bodies position
3     transform.position = Vector3.Lerp(transform.position, _selectedBody.position, 0.3f);
4 }
```

Figure 151: CamControl class update function

So what the update() procedure does is, it sets the CamAxis position to the same position as the new body that we want to track but instead of doing this statically, it dynamically implements this change by linearly interpolating to find the vector between the current focus body and the new one so that the position of the camera is changed within every frame to allow for a smooth transition of focused bodies. I done it this way because it wasn't very hard to implement and also, it makes it seem like the camera is actually travelling in space rather than spawning at different points making the simulation more satisfying to use.

StopAnimation()

When switching between different body's to focus on I'd need a function that stops the current animation that the camera is in to avoid two animation running at the same time on the same time which can cause the camera to malfunction as it is being forced to orbit around two bodies at the same time. I implemented the stop feature by creating a void animation and forcing it onto the camAnimator handler.

```
1 // stop any ongoing animation by simply creating a new void one
2 private void StopAnimation(){
3     if (_camAnimator!= null && _camAnimator.IsAnimating)
4         _camAnimator.NextAnim ();
5 }
```

Figure 152: StopAnimation function for camControl

PanCam()

```
1 public void PanCam(bool unused){
2     if(_camAnimator != null)
3         _camAnimator.NextAnim (new CamAnimation(_cam.transform, camFinalDepthZPos:-20f, poleFinalVerticalXRot:25f));
4
5     ControlIntentions.Instance.MenuCall -= PanCam;
6 }
```

Figure 153: PanCam() function

PanCam() sets the initial position of the camera when the game is started, It is subscribed to the menu call event, so it Is notified when there is a change (when the user leaves the menu). When notified it calls the constructor for the camAnimator class to activate camera movements. This implementation is vital for a working camera as having the _camAnimator initialised without the user even starting the simulation will cause many errors in the simulation as the camera animator class will try to linearly interpolate over a bunch of null values which will prevent the game from running and cause it to crash.

RotateCam()

To allow for manual rotation of the camera from the user I created this function:

```
1 private void RotateCam(string axis, float dir) {
2     if (axis == "horizontal")
3         transform.Rotate(new Vector3(0, dir * CamSpeed * Time.deltaTime, 0));
4
5     else if (axis == "vertical")
6         _camPole.Rotate(new Vector3(dir * CamSpeed * Time.deltaTime, 0, 0));
7
8     else
9         Debug.LogWarning("Cam rotation called on unknown axis: " + axis);
10
11    StopAnimation();
12 }
```

Figure 154: RotateCam() function

When notified by control intentions it transforms the camera by creating a new vector3 component for the cam axis and pole, the camera is rotated at a specific speed, this speed is set as a constant in the CamControl class, but I will allow the user to change it when I create my options menu. If there is another type of input that calls the RotateCam event an error is printed in the console log stating the axis at which the input satisfies, this is a very unlikely error but I added it just in case for a more robust program.

StopAnimation() is then called within the RotateCam function, to interrupt any active camera animations, if present, before applying the requested rotation of the camera. This ensures that the camera's rotation can occur independently of any ongoing animations, preventing visual inconsistencies or unexpected behavior that can occur.

TranslateCam()

Zooming is translating the cam in the Z axis. To implement the zooming feature I'd have to change the z component of the camera's position vector. I done it like this:

```
1 private void TranslateCam(float translation) {
2     // The idea is to use the actual distance from the cam to the focus body
3     // as a factor in how fast the cam is moving othe pole.
4     Vector3 pos = _cam.transform.localPosition;
5     float Z = pos.z + ZoomSpeed * translation * _cam.transform.localPosition.magnitude;
6     pos.z = Mathf.Clamp(Z, ZoomMax, ZoomMin);
7     _cam.transform.localPosition = pos;
8
9     StopAnimation();
10 }
```

Figure 155: TranslateCam() function

The z component of the camera is a sum of its original position and the translation amount multiplied with the zoom speed and magnitude of original position. The new z position is then clamped to ensure that it fall between ZoomMin and ZoomMax so that the user doesn't go to far away from the simulation. It is then transformed onto the position of the camera and ongoing animations are stopped so that they can adjust to the camera's new position.

Tests

Test ID	Test Case	Test Data	Expected Results	Passed

C-1	Camera Initialization	N/A	Camera is positioned at the default location.	✓
C-2	Camera Orbit Functionality	Target: Sun	Camera orbits the specified body accurately.	✓
C-3	Zoom In/Out Functionality	Zoom In: True, Zoom Out: True	Camera zooms in and out smoothly.	✓
C-4	Camera Reset	N/A	No option has been created yet	X
C-5	Camera Limits	Limits: MinDistance=10, MaxDistance=100	Camera does not go beyond the set limits.	✓
C-6	Camera Follow Functionality	Target: Planet, Zoom In: True	Camera smoothly follows the specified body.	✓
C-7	Camera Collision Avoidance	Obstacle: Asteroid, Collision: True	Asteroid haven't been added yet	X
C-8	Camera Switching	Target: Moon	Simulation does not respond to clicks	X

The simulation doesn't respond when a celestial body is clicked. As seen in the video here: https://youtu.be/e1G9AltYptc.This is because a function to switch the focus body based on user input is not created.

Test C-8

I could implement this function in many different ways, one way could be to have a menu on the side of the screen where the user can select the celestial body they want to focus on, however this implementation is very basic, it makes the simulation feel less interactive because the user isn't directly interacting with the simulation. So, what I can do instead is make it so that the user clicks on celestial bodies in the actual simulation, this makes the simulation feel more interactive, and more minimal. I think this implementation would be better as interactivity is very important in an educational setting, people's attention will be drawn for longer if they have more control over the simulation.

To do this I'd first have to make it so that my celestial bodies have a collider component to them in unity. This allows for the simulation to know when celestial body is clicked. As you can see the position of the collider is 0,0,0 this is relative to the celestial body it belongs to and not the sun. If I made it so that it was relative to the sun the colliders components will also have to be updated with time which is extra processor power being wasted.

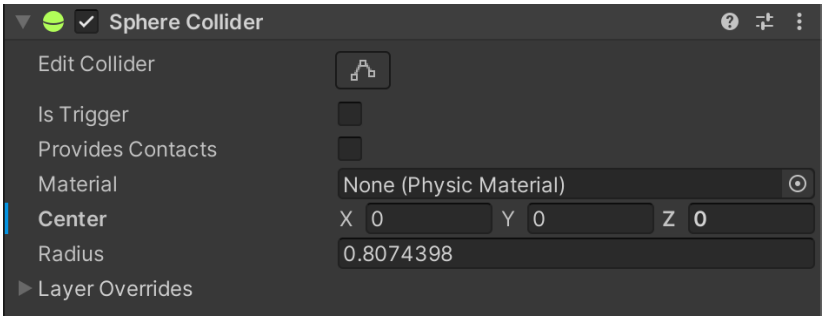


Figure 156: Sphere collider component of celestial bodies

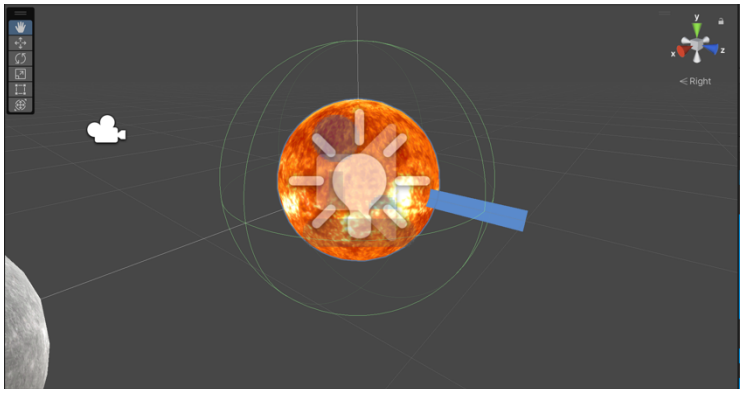


Figure 157: Unity scene view showcasing sphere collider radius

The collider component looks like this, its radius is a bit bigger than the radius of the sun as the user may not be able to click on the sun perfectly when they are zoomed out and the sun isn't the focused body. I could've made it the same size as the sun but this would mean that the user would struggle clicking on the sun and would have to click on it multiple times to get it correct which causes inconvenience for them. The radius of the collider component doesn't affect the overall performance of the simulation but it still has to be set to a sensible value that is not much greater than the celestial body itself as the simulation can misregister a click ona different celestial body as the one of another due to the size of the collider

For the simulation to know where a collider actual is relative to the camera of the scene id have to use unity's raycast function in the physics class, what it does is that it casts a ray from a selected origin (the location of the camera) against all colliders in the scene for a certain distance.

```
1  private void CheckSelection(Vector3 selectorPos) {
2      // first we cast a Ray on the UI to see if we catched one of the icon
3      Ray ray = _cam.ScreenPointToRay(selectorPos);
4      RaycastHit hit;
5
6      if (Physics.Raycast(ray, out hit, RayLength)) {
7          if (hit.collider != null) {
8              _selectedBody = hit.collider.transform;
9
10             if (NewFocus != null)
11                 NewFocus(_selectedBody);
12
13             if (!GetComponent<AudioSource>().isPlaying)
14                 GetComponent<AudioSource>().PlayOneShot(_selectionSound);
15         }
16     }
17 }
```

Figure 158: CheckSelection method for focussing

So, I created a checkSelection() function which is subscribed to the FocusSelection event In ControlIntentions. So, when the user clicks on the simulation this function runs, what it does is that if the click corresponds with a celestial body the _celestialbody variable is set to the celestial body that is clicked, then it notifies the newFocus controlIntentions event saying that there has been a change in focus. The last if statement is irrelevant to the camera class, it is for buttons sounds which I will talk about in the audio section.

Review

The camera works as intended in my design, it focuses, on planets, zooms in and out and can be moved freely around the solar system.

Criteria Met:

Criteria No.	Criteria	Criteria Met
C-8	Select Planet to Track	Completed
C-9	Zoom In/out	Completed

There are no changes from my original design, this is because the camera section is a pretty simple section anyways, there isn't much additional features I can add nor is there any better approaches I could've taken as my camera design was already well thought out.

Asteroids

To make my simulation look realistic I want to add asteroids. However, there are currently over 1.3 million known asteroids in our solar system, and I obviously cannot simulate that as it would be very performance intensive.

So, I am going to be using abstraction to make my simulation representative of the solar system by removing unnecessary details of asteroid to save resources for my simulation.

This is an image representative of the asteroid belts in our solar system. From this I can infer that id have to create two belts, a dense smaller inner asteroid belt, and a less large outer asteroid belt.

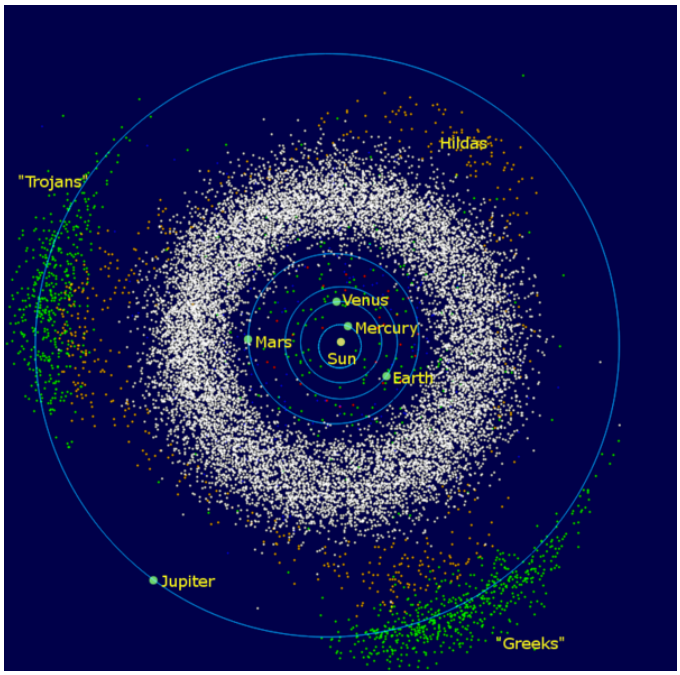


Figure 159: Asteroid Belts of our solar system

To do this I'd have to create a script that procedurally generates asteroids which vary in shape and size. I opted in to implement it this way because it allows for the asteroid to be more representative of real life (Not all asteroids are the same size in real life). And also, it can help show the density of the orbital belt better as asteroids of same shape have a fixed gap between them compared to ones have different shapes which can have varying gaps.
This can help students and educators better understand the composition and dynamics of asteroid belts in our solar system.

While it might seem counterintuitive, having asteroids of different sizes can actually help with rendering performance. Smaller asteroids can be rendered with fewer details when they are far away from the camera, which can save on processing power. So having varying sizes will be useful for computers with lower system specifications then recommended, as smaller asteroids can be loaded faster than if all asteroid were large and the same size.

BeltSpawner.cs

This is what the initialisation of the BeltSpawner class looks like. It takes in values for the density of the belt, inner and outer radius of the belt, height of the belt, the rotation direction, the minimum and maximum size of the asteroids in the belt, the maximum inclination of an asteroid in the belt, the range of orbital speeds asteroid in a belt can take and the local position and position relative to the sun of the asteroids.

```
1  public class BeltSpawner : MonoBehaviour
2  {
3      public GameObject cubePrefab;
4      public int cubeDensity;
5      public int seed;
6      public float innerRadius;
7      public float outerRadius;
8      public float height;
9      public bool rotatingClockwise;
10     public float minScale = 0.1f; // minimum scale for asteroid
11     public float maxScale = 0.2f; // maximum scale for asteroid
12     public float maxInclination = 30f; // maximum orbital inclination in degrees
13
14     public float minOrbitSpeed;
15     public float maxOrbitSpeed;
16     public float minRotationSpeed;
17     public float maxRotationSpeed;
18
19     private Vector3 localPosition;
20     private Vector3 worldOffset;
21     private Vector3 worldPosition;
22     private float randomRadius;
23     private float randomRadian;
24     private float x;
25     private float y;
26     private float z;
```

Figure 160: BeltSpawner script

When the script starts this is what happens:

```
private void Start()
{
    Random.InitState(seed); // passes in a seed so belt looks the same within each launch. This works like it wo

    for (int i = 0; i < cubeDensity; i++)
    {
        do
        {
            randomRadius = Random.Range(innerRadius, outerRadius);
            randomRadian = Random.Range(0, (2 * Mathf.PI));

            y = Random.Range(-(height /2), (height / 2));
            x = randomRadius * Mathf.Cos(randomRadian);
            z = randomRadius * Mathf.Sin(randomRadian);
        }
        while (float.IsNaN(z) && float.IsNaN(x));

        // Add random inclination
        float inclination = Random.Range(-maxInclination, maxInclination);
        Quaternion rotation = Quaternion.Euler(inclination, 0, 0);
        localPosition = rotation * new Vector3(x, y, z);
        worldOffset = transform.rotation * localPosition;
        worldPosition = transform.position + worldOffset;

        GameObject _asteroid = Instantiate(cubePrefab, worldPosition, Quaternion.Euler(Random.Range(0, 360), Ran

        // Add random scale
        float scale = Random.Range(minScale, maxScale);
        _asteroid.transform.localScale = new Vector3(scale, scale, scale);
        _asteroid.AddComponent<BeltObject>().SetupBeltObject(Random.Range(minOrbitSpeed, maxOrbitSpeed), Random.

        _asteroid.transform.SetParent(transform);
    }
}
```

Figure 161: BeltSpawner script start function

The script first sets the “randomness” of the simulation to a given seeds. I chose to implement the use of seeds to my asteroid belt generator because I want the belt to look the same within every launch of the simulation, this allows for consistency, and it doesn’t effect the performance of the simulation. I wanted to do this because at epoch j200 planets always start at the same position so it will make sense to also start the asteroids at the same place.

Then the function iteratively creates asteroids and appends it to the stellar parent (in this case the sun). It does this until the number of asteroids generated is equal to the cube density. It creates a random radius, and random angle for inclination of the asteroid and based on those it generates an x,y, and z coordinates for it in a specific height range on the condition that the coordinates are zero. This stops until there are no coordinates that are 0. I implemented a while loop because if it was only iterative based on count, there can be random numbers generated which are zero, and having a coordinate of 0,0,0 would be a wasted rendering of an asteroid so to have the most efficient asteroid generation algorithm I opted in for a while loop.

The coordinates obtained from the while loop is then translated from the local position to the world position, this is done by generating a random inclination, which Is then used in the Quaternion.Euler function which is used to represent the asteroid rotation around the central axis. This position is then used to find the local position of the asteroid, which works by multiplying it by the randomly generated x, y, and z axes to get the new randomly inclined value of the local position. I chose to have random inclinations of my asteroids as

this will allow for a larger variety in positions, this represents a real-life asteroid belt better which makes my simulation more realistic

Then the world offset is calculated by applying the parent object's rotation to the local position of the asteroid. This gives the position of the asteroid relative to the parent object, but in world space instead of local coordinates.

The world position is calculated by adding the world offset to the parent object's position. This gives the absolute position of the asteroid in the scene.

Then a new asteroid Game Object is created using the world position coordinate and the rotation of this asteroid is randomised.

After the game object has been created it has to be adjusted to look like an asteroid. I first generate a random scale for the asteroid as I want them to be different sizes, this is because it will make the density of the belt look more realistic and also in real life not all astroids are the same.

After a BeltObject component is added to the gameobjects of the asteroids then the parent of the asteroid is set.

BeltObject.cs

To make my code more modular and easier to manage. I created another script, this allows me to change how individual asteroids behave. The BeltSpawner script is responsible for creating and positioning the asteroids in the scene, while the BeltObject script is responsible for controlling the behavior of each individual asteroid. It also makes it easier to pinpoint where any errors are coming from when testing the asteroid belt as I can know if the errors or of a spawning issue or of an object issue this saves me time and gives me the possibility to create a more precise asteroid belt in the future thanks to the individuality provided by the encapsulation of the script

This will also allow for my code to be reused. If I want to use the same behaviour for other objects in the scene, like the rings of planets like Jupiter, which would require different asteroids but would consist of the same code from the BeltSpawner. I can simply attach the BeltObject script to those objects and still use the same BeltSpawner script. This will save me huge amounts of time if I want to implement my own rings for planets.

This is the whole Belt Object script:

```
public class BeltObject : MonoBehaviour
{
    [SerializeField] private float orbitSpeed;
    [SerializeField] private GameObject parent;
    [SerializeField] private bool rotationClockwise;
    [SerializeField] private float rotationSpeed;
    [SerializeField] private Vector3 rotationDirection;

    public void SetupBeltObject(float _speed, float _rotationSpeed, GameObject _parent, bool _rotateClockwise)
    {
        orbitSpeed = _speed;
        rotationSpeed = _rotationSpeed;
        parent = _parent;
        rotationClockwise = _rotateClockwise;
        rotationDirection = new Vector3(Random.Range(0, 360), Random.Range(0, 360), Random.Range(0, 360));
    }

    private void Update()
    {
        if(rotationClockwise)
        {
            transform.RotateAround(parent.transform.position, parent.transform.up, orbitSpeed * Time.deltaTime);
        }
        else
        {
            transform.RotateAround(parent.transform.position, -parent.transform.up, orbitSpeed * Time.deltaTime);
        }

        transform.Rotate(rotationDirection, rotationSpeed * Time.deltaTime);
    }
}
```

Figure 162: BeltObject class

I set the variable as serialised fields as, they cannot be public, this is because there are more than one asteroids and if the fields are public It may introduce a risk of their being an error within the spawning of asteroids.

Then I created a public function called SetUpBeltObject, this is used by the BeltSpawner script. It takes in values for the orbit speed, rotation speed, parent object, and rotation direction, and assigns these values to the corresponding variables. The rotation direction is set to a random vector, which makes the asteroid rotate in a random direction, like how asteroids in a belt will be.

The update method then checks if rotationClockwise is true, the asteroid is rotated around the parent object's position in the direction of the parent object's up vector. If rotationClockwise is false, the asteroid is rotated in the opposite direction. The speed of the rotation is controlled by orbitSpeed and Time.deltaTime, which makes the rotation frame rate independent. The asteroid is also rotated around its own axis according to rotationDirection and rotationSpeed.

Inner Belt Test S-8

To create my first orbital belt, I attached the orbital spawner to my sun gameObject. I done this because the centre of the orbital belt will be the centre of the solar system which is the sun.

This is what the I put the presets as in unity for now, the asteroid prefab is set to a white cube temporarily as I still haven't found a prefab of an asteroid. Also, I wanted to test out if the asteroid belt actually works before polishing it.

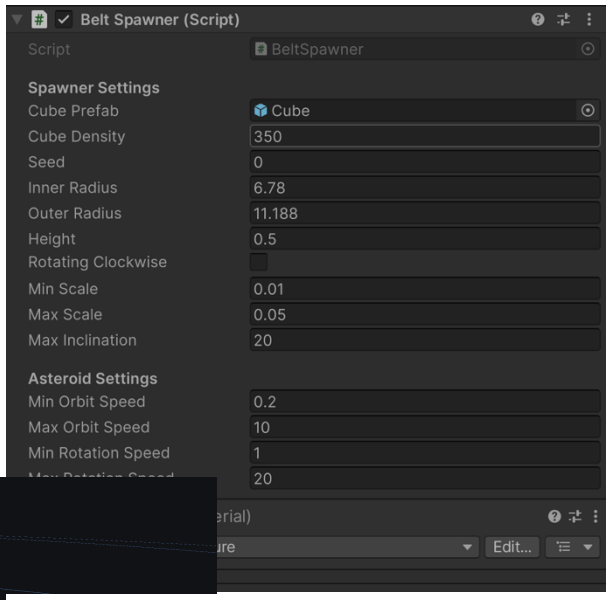


Figure 163: Belt Spawner component settings

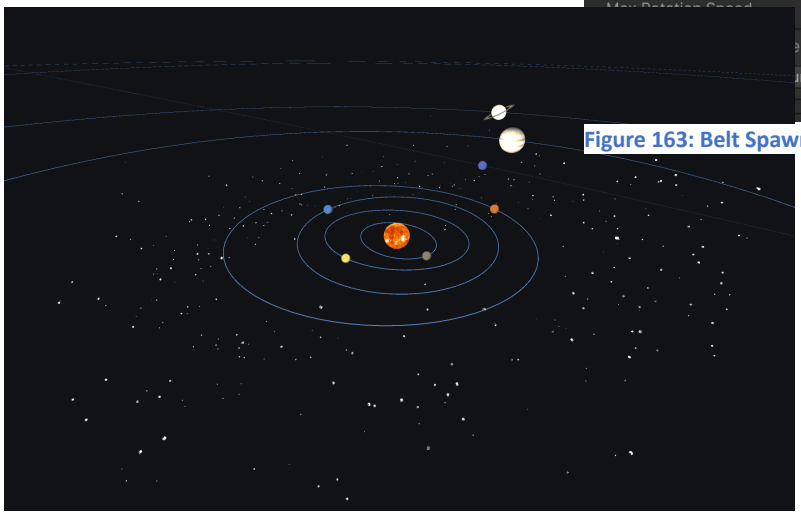


Figure 164: Inner Belt

This is what the asteroid belt looks like, it rotates with time and I can change the direction and speed of the belt as a whole.

Right now, I don't have an asteroid prefab, I found one on the internet and applied it to my simulation, the performance was not affected as much as I thought it would. And this is what the asteroid belt looks like with the prefab:

Test ID	Test Case	Test Data	Expected Results	Passed
S-08	Asteroid Belt Generation	Cube Prefab	Asteroid belt is accurately placed and random	✓
UI-05	Asteroid Prefabs	Multiple Different prefabs	Renders a variety of asteroids in a belt not just one that looks the same for all asteroids	X
S-9	Accurately Represented asteroid belts	N/A	Renders inner asteroid belt and Kuiper belt accurately	X

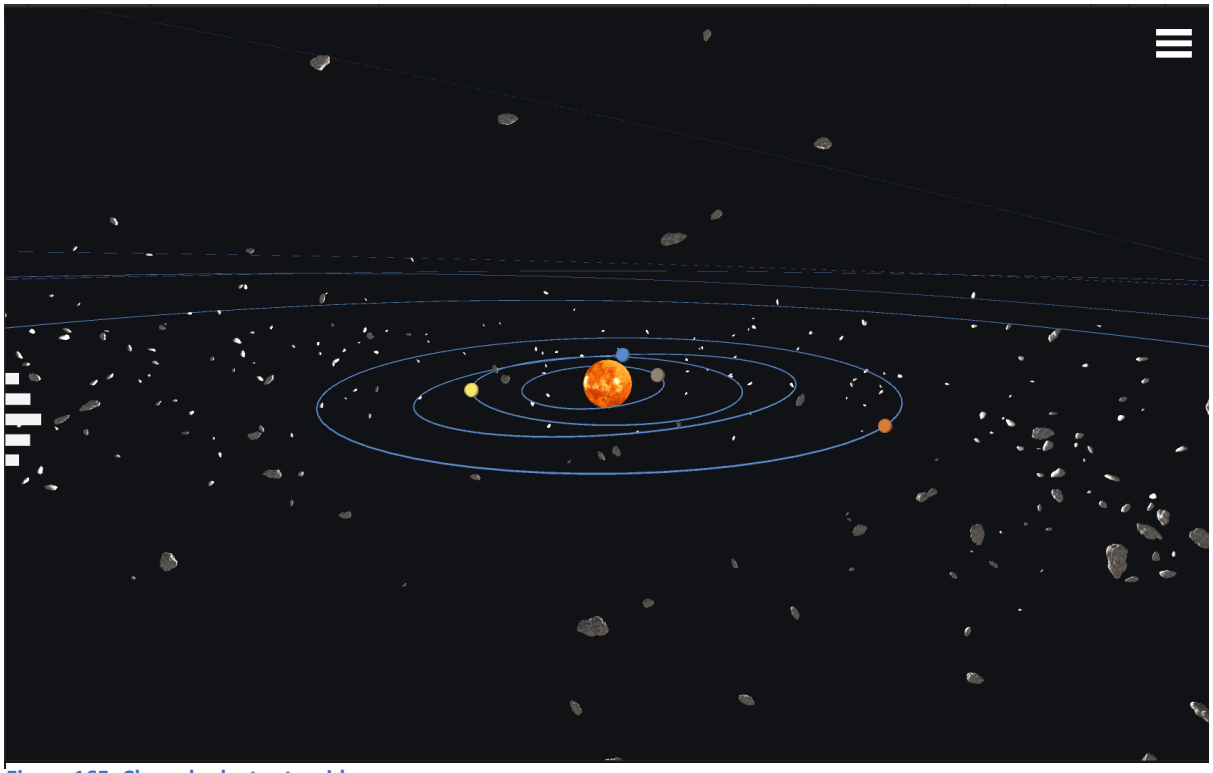


Figure 165: Clsoer look at asteroids

Here is a closer look at the asteroid, what bothered me with this. Prefab is that it is the only one, meaning all the asteroids in the belt look the exact same. So, I wanted to add more prefabs and randomly generate an asteroid with a randomly chosen prefab to represent reality more, as not all asteroids are the same in space.

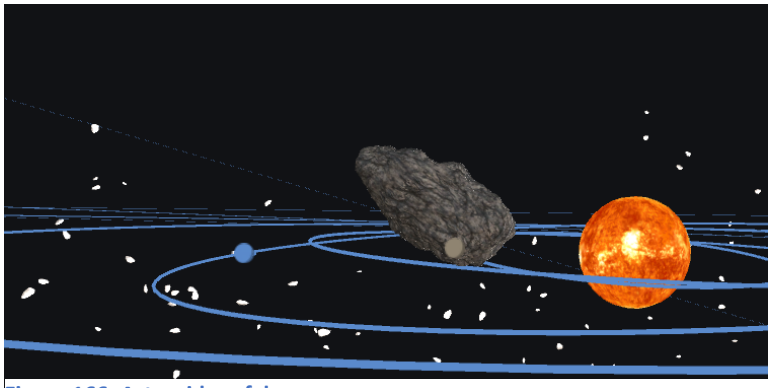


Figure 166: Asteroid prefab

Test UI-05

My asteroids all look the same, to fix this I'd have to change my BeltSpawner script so that it randomly chooses a prefab from a list. Here is how I changed my code:

The changes are highlighted in the

```
public class BeltSpawner : MonoBehaviour
{
    public GameObject[] cubePrefabs; // Array of prefabs
    ...
    private void Start()
    {
        ...
        for (int i = 0; i < cubeDensity; i++)
        {
            ...
            // Select a random prefab
            GameObject selectedPrefab = cubePrefabs[Random.Range(0, cubePrefabs.Length)];

            // Instantiate the asteroid with the selected prefab
            GameObject _asteroid = Instantiate(selectedPrefab, worldPosition, Quaternion.Eu

            ...
        }
    }
}
```

Figure 167: Random prefab selecting variable

image, I wanted to create a list of different asteroid and randomly iterate through them so that my simulation is more representative of reality, this would be very useful for students as most don't know that asteroid belts exist, so having a realistic looking asteroid belt would be very good for educational purposes

Test ID	Test Case	Test Data	Expected Results	Passed
UI-05	Asteroid Prefabs	Multiple Different prefabs	Renders a variety of asteroids in a belt not just one that looks the same for all asteroids	X

I also created a second component:
I done this because the orbital belt is not representative enough of one in reality, the asteroids are uniformly distributed.
I made it so that this second component has a larger radius than the first belt but is less dense, I done this because the asteroid belt between mars and Jupiter is denser closer to mars and less dense closer to Jupiter. So doing this means that that closer to mars there is a separate belt and top of that one there is the second belt which makes it look even more dense. I implemented it this way because altering my code so that I can change density of the asteroid belt in different sections would be very time and resource consuming, and require a lot of complex maths. So, by spawning belts on top of each other I save time, resources and achieve the same thing without wasting too much resources.

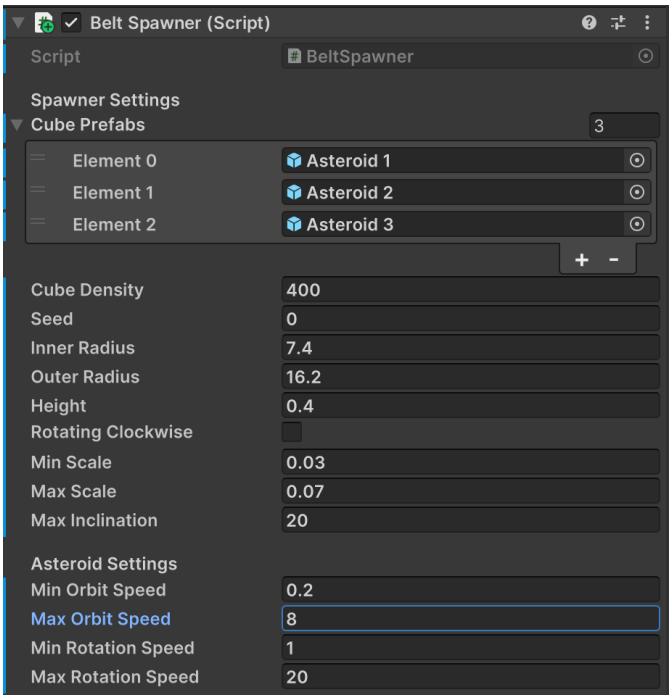


Figure 168: Updated Belt Spawner component settings

This is what the belt looks like now. As you can see the inner side of the belt is much more dense compared to the outside, but what I realised is that some side of the outer skirts of the belt have much more empty space, this is because Jupiters orbit is a bit eccentric, so to fill this space and make it more realistic I want to implement eccentricity to my asteroid belt.

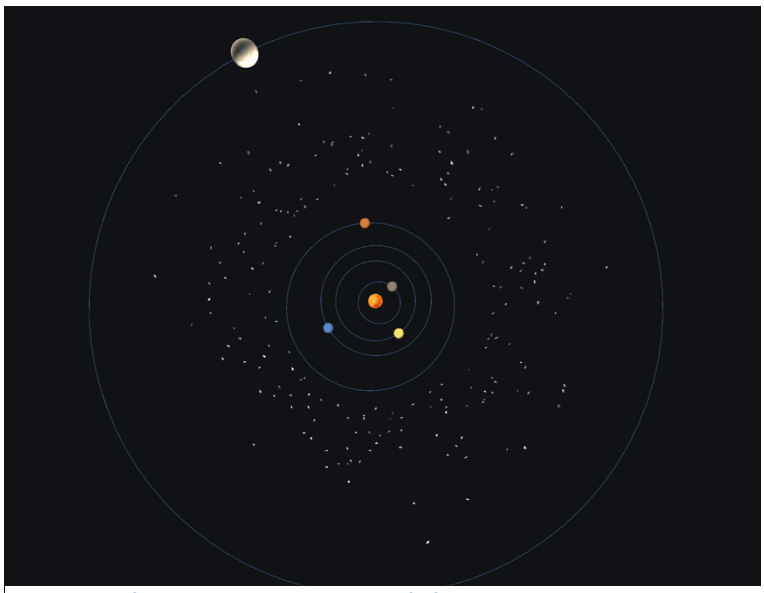


Figure 169: Showing non eccentric inner belt

Test ID	Test Case	Test Data	Expected Results	Passed
UI-05	Asteroid Prefabs	Multiple Different prefabs	Renders a variety of asteroids in a belt not just one that looks the same for all asteroids	✓

Test S-08 Iteration 2

I implemented eccentricity to my asteroid belt by changing how the x and z coordinates are calculated. Here's what I changed them to:

```
1 x = (1 + eccentricity) * randomRadius * Mathf.Cos(randomRadian);
2 z = randomRadius * Mathf.Sin(randomRadian);
```

Figure 170: Updated x and z variables accounting for eccentricity

After Implementing this I had to test if it actually works because it is much harder to see eccentricity in an asteroid belt compared to a celestial body's orbital line, as asteroid are randomly distributed whereas a line is uniformly distributed so I would be able to easily see changes to it. I felt like I need to test it as I wanted to make sure that my asteroid belt is eccentric to make my simulation as realistic as possible, also adding eccentricity to the belts do not affect the performance that much so it is a good trade off for realism

Boundary Data

Here I set the eccenctricty of both belts to 0.99 which is ther highest it can be, This should produce asteroid that follow a staright line:

Like it did here, so this verify's that the eccentricity implementation works

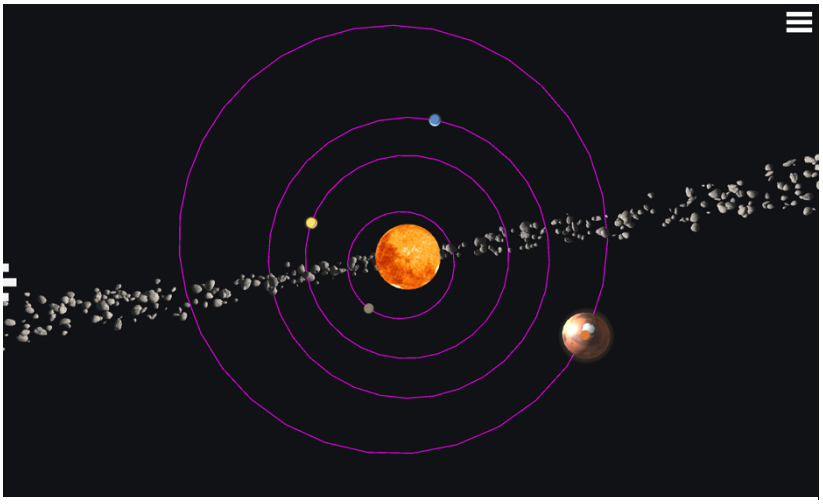


Figure 171: Boundary value for eccentricity

Normal Data

Now I want to add the actual eccentricity's that the asteroid belt would be. I set the eccenctrity of the second belt to 0.2 and the 1st to 0.05. And this is what the asteroid belt looks like now:

As you cans see the asteroids are now distributed better.

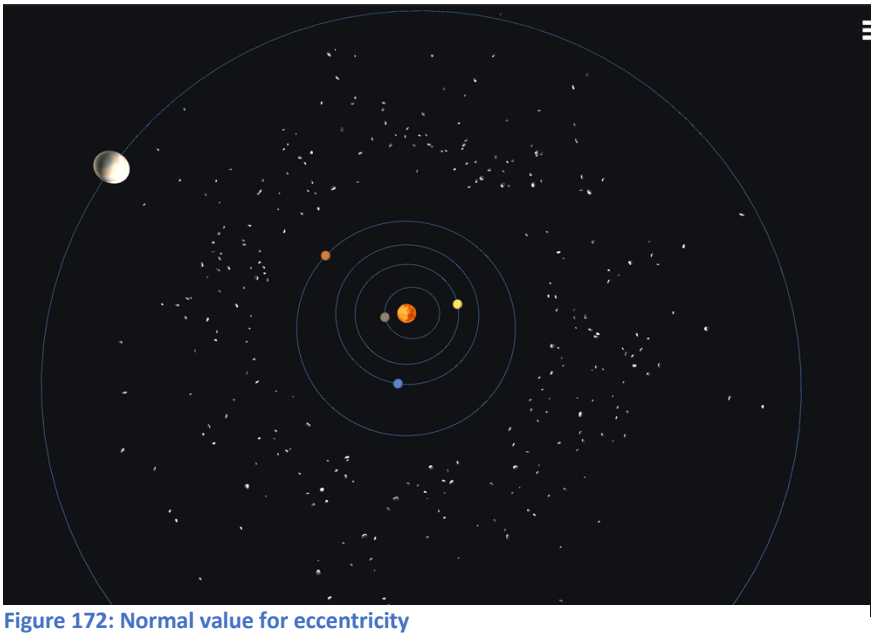


Figure 172: Normal value for eccentricity

Test ID	Test Case	Test Data	Expected Results	Passed
S-9	Accurately Represented asteroid belts	N/A	Renders inner asteroid belt and Kuiper belt accurately	✓

The Kuiper Belt

Rendering the Kuiper belt will probably be the most challenging, this is because it stretches very far away, further than Pluto, which makes it giant, so it would have to wrap around the whole solar system. Also, the asteroid being visible all the time would create a lot of clutter when viewing planets in the inner solar system, so by thinking ahead, I would also have to make it so that the Kuiper Belt only renders at specific distances of the camera.
To do this I'm going to be creating a new script, called BeltVisibilityController:

BeltVisibilityController.cs

This script will only work on the assigned Belt Spawner.

```
public class BeltVisibilityController : MonoBehaviour
{
    public BeltSpawner specificBeltSpawner; // assign the specific BeltSpawner in the inspector
    public float thresholdDistance = 100f; // set the threshold distance
    private Camera mainCamera;

    private void Start()
    {
        mainCamera = Camera.main; // get the main camera
    }

    private void Update()
    {
        float distance = Vector3.Distance(mainCamera.transform.position, specificBeltSpawner.transf

        if (distance > thresholdDistance)
        {
            specificBeltSpawner.gameObject.SetActive(true);
        }
        else
        {
            specificBeltSpawner.gameObject.SetActive(false);
        }
    }
}
```

Figure 173: BeltVisibilityController class

How it works is, it sets a threshold distance, which is the maximum distance of the camera at which the asteroids don't appear. Then within every frame the script checks if the distance of the camera and compares it to the threshold, if greater it activates the belt spawner and if not, it deactivates it.

Test Iteration 1

When I attach the script and add it as a component to my sun object, then select the BeltSpawner for the Kuiper belt, the sun and all asteroids disappear.
As shown here:

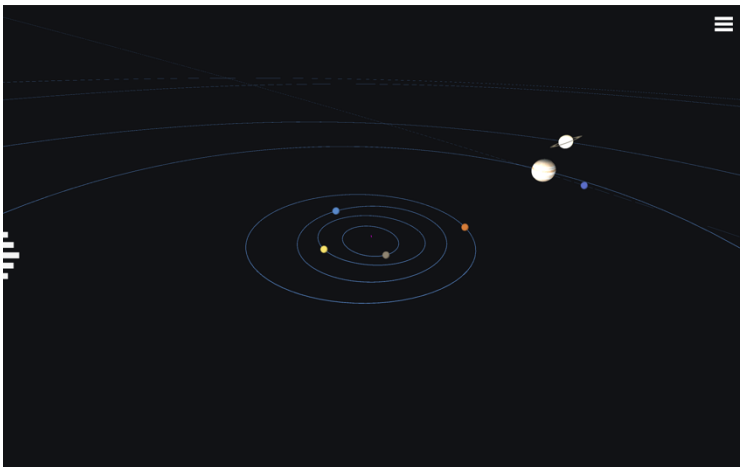


Figure 174: Unity fame view sun and asteroids missing

I think this is because when I attach the BeltSpawner script to the specificBeltSpawner attribute, it represents the game object of that spawner, which is the sun and other components attached to it (the asteroids). To combat this problem I have to change the way the beltspawner scripts are attached in unity.

Test Iteration 2

This is what I changed it to:

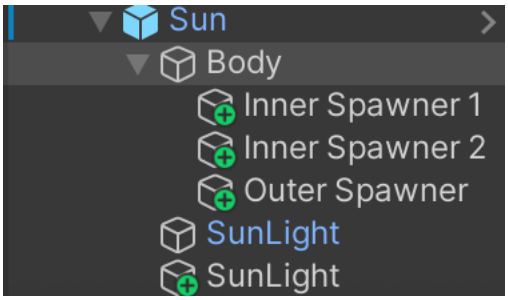


Figure 175: Change of scene hierarchy

Each child gameobject has its own sperate beltspawner script attached to it, meaning the beltvisibilitycontroller script can deactivate the parent of the asteroids which would be these gameobjects.
Another way I could've fixed this is by adding a name attribute to the belt spawner script and made it so that a belt with the specific name disappears, but this method is less modular and not reusable compared to a new script and new gameobjects. I opted in for this method as I may need to reuse the beltvisibilitycontroller script if I want to change the visibility of other smaller asteroid belts or for when I want to create rings of planets myself. Also this method allows me to separate each asteroid cluster from one another by creating child objects for ewach which will come in handy for when I may need to debug.

When I run the simulation, I see the inner asteroid belt but not the Kuiper belt, when I zoom in and out this doesn't change. I think this is because my threshold distance it too is high.

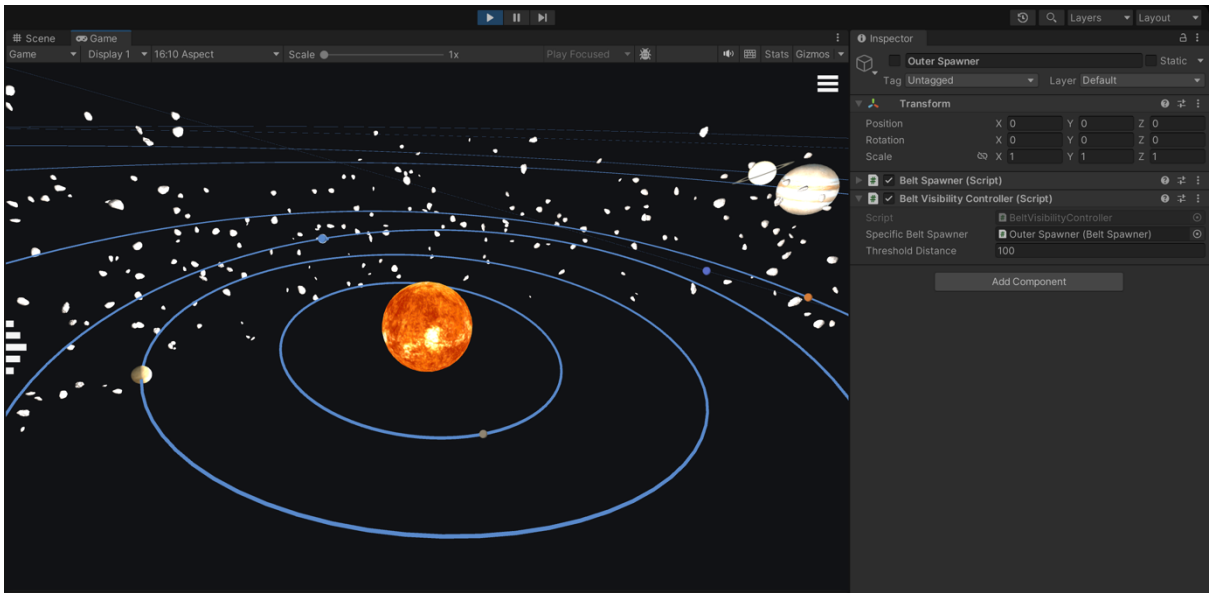


Figure 176: Kuiper belt not showing

Test Iteration 3

When I decrease my threshold distance to an extremely small value the asteroid belt appears again, but then when I zoom in and out it disappears and doesn't come back. I think this is because of the update method, it must not be working properly I know this because when the simulation starts my threshold distance is smaller than the default distance of the camera that is set and when I zoom in then zoom back out the asteroid belt doesn't come back because the update method hasn't captured the change in distance of the camera. Looking at my update method I realised that the belt spawner is set to deactivate as it should when the camera is less than the threshold distance, but what it deactivates is the game object that actually has the script containing the update method, meaning update cant run after this.

To fix this I have to change the order of my components in the unity editor, The beltvisibilty controller should be a component of a game object that always stays active, and this is the sun. So, after moving my component to the sun's gameobject the belt visibility controller works as intended.

Review

Asteroids were an additional section of my simulation that isn't in my design that I thought to later add. I added it because it makes my simulation more realistic and makes my simulation more educational.
Most students don't know where asteroids are in our solar system, so I thought adding this would make my stakeholders really grateful as only the orbital motion of celestial bodies were thought out, not asteroids.
The asteroids in the asteroid belts have random motion, just like in real life and have the correct densities and eccentricities.

Adding the asteroid belt did take up a lot of my development time, but I think it is definitely worth it and makes my simulation feel more realistic whilst at the same time not taking a big toll on its performance. I could've not added it at all, but I felt like my simulation had something missing and wasn't representative of the solar system enough.

Options Menu Scene

The final scene I have to create is the options menu scene. This scene will allow the user to customise various aspect of the simulation, like resolution, audio, framerate, control bindings etc. It will have 3 sections, graphics, audio, and controls.

I want to have consistency within my simulation, so im going to make the options menu have the same design as the main menu. This will make it quicker for me to construct the

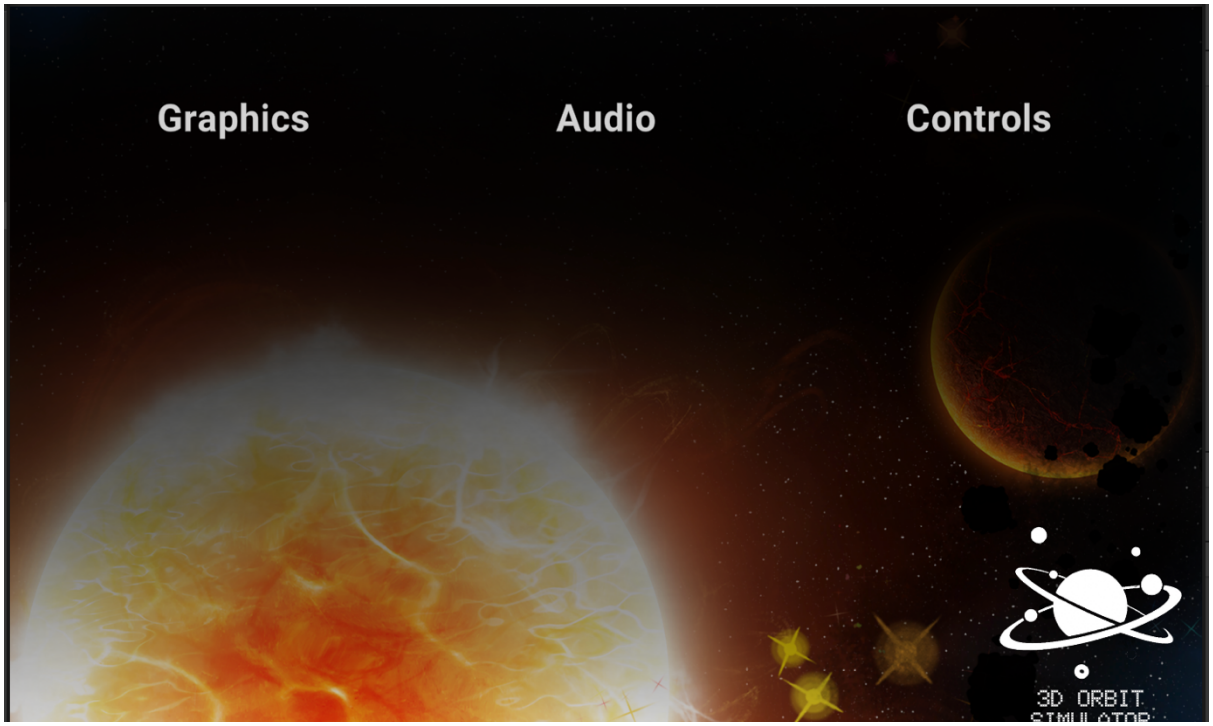


Figure 177: Options menu scene buttons

options menu in unity as the unity components are reusable components and I can just paste them over to the options menu scene. As you can see this is what o copied over:

I copied the buttons over and changed their name to graphics, audio and control, I done this instead of creating new buttons to save time, as it is time consuming to write different hex codes for buttons in the button component of text object, to represent different colours for different actions within the buttons like highlighting, selecting, etc.

When the options menu is opened the first tab that will open will be the graphics tab, it will be places left most in the tab selector section. And the graphics tab will be the first one opened, to allow for quick access to the settings. I could leave the options menu like it is as shown in the image but it will be useless to have it like this as the user will have to click on the tab they want anyways, also I don't think it looks as appealing compared to having options already there.

I created separate scripts for the three tabs instead of having one script for the whole options menu, this keeps my code separated and allows for modularity within my simulation. Also, it makes my code more maintainable as the isolation of the tab reduces the risk of introducing bugs to the simulation and if I wanted test where a bug is, it would be much easier to pinpoint

Graphics Tab

I first have to create buttons and dropdowns in the grapics tab so that the user can access different graphical settings. I created dropdowns for resolution, quality, FPS, and anti-aliasing. These options are vital for users with slower performing computers, they can lower graphical settings so that their simulation runs smoother.

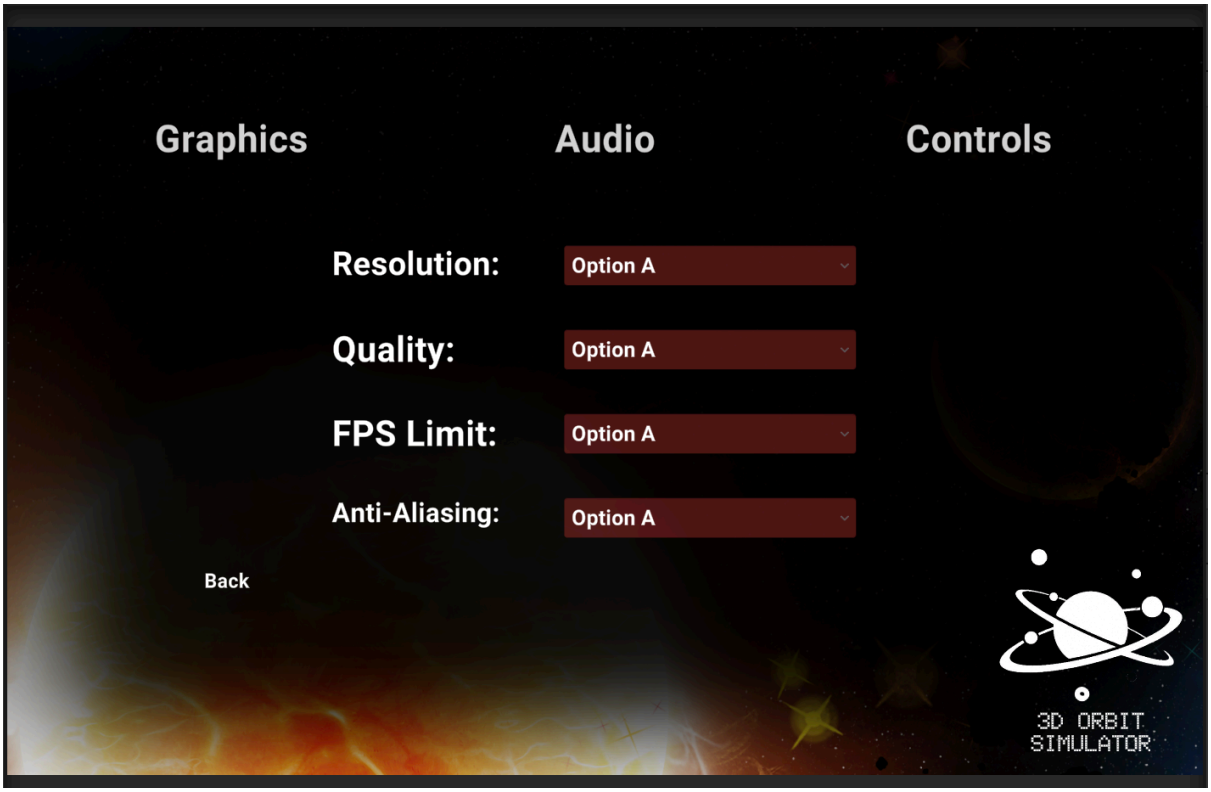


Figure 178: Graphics Tab

If I didn't have these settings, users with lower performing computers would not be able to run the simulation properly, assuming most users are students, whom most likely use chromebooks in school they would definitely need to have these settings so that they can run this simulation.

Quality

I wanted to add named options to the quality dropdown. Instead of representing them as option a, b, and c. User readability is important, I wouldn't want to have them guessing which graphics wuality is suitable for their computer. I chose to have 6 graphics options, very low, low, medium, high, very high, and ultra. I could've just chosen 3 levels low, medium, and high. But, as mention previously, having graphical settings suitable for low performing computers is very important, so a very low option is definitely needed for those students. I added very high, and ultra for the smaller group of users who may want to use this simulation for aesthetic or entertainment reasons.

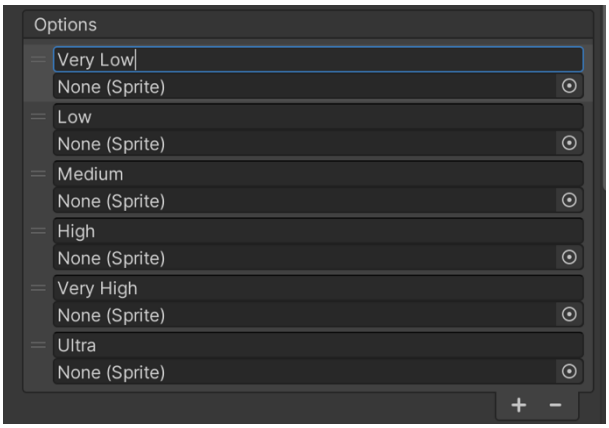


Figure 179: Quality options

Here is how I started the graphics tab script. I created a setquality() procedure which changes the quality of the simulation based on the index of the dropdown selection.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class GraphicsTab : MonoBehaviour
6 {
7     public void SetQuality(int qualityIndex)
8     {
9         QualitySettings.SetQualityLevel(qualityIndex);
10    }
11 }
```

Figure 180: Set quality function

In my unity project settings, this what the quality settings looks like, I followed the order of quality levels in the project settings, for my dropdown menu so that the index's match.

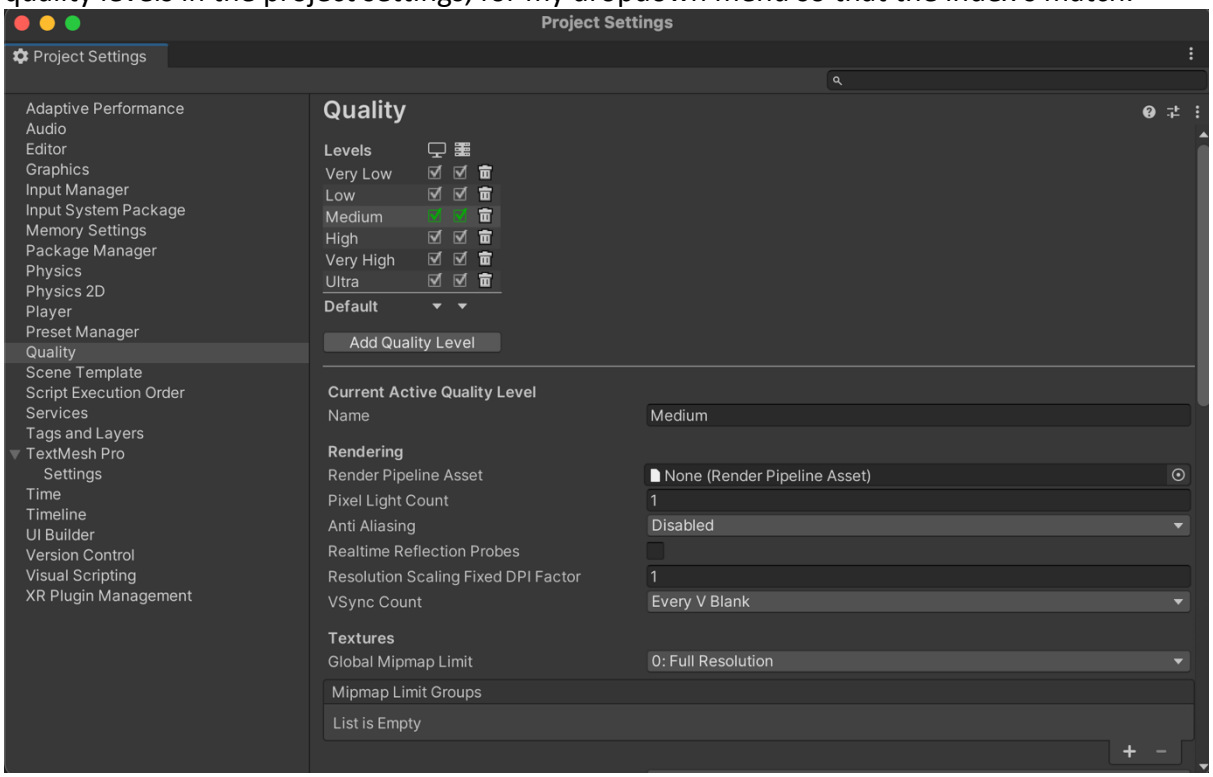


Figure 181: Unity project settings quality options

Test OM-03

Now that I have adjusted my script, I can test if the quality of the simulation changes. As you can see in this video, when changing the option in the dropdown the current quality shown in the unity project settings changes to one corresponding to the current option in the dropdown:
https://youtu.be/LhqG0r6HpT8

Resolution

Graphics quality is a parameter that can be universally changed within all computers; however, resolution isn't, different computers have different resolution settings. So, I can't just create my dropdown options easily like I did by checking the unity project settings. When the script is called id have to check for all resolution options for the specific computer the script is being run on.

So I'm going to use the start() procedure which will retrieve all the resolutions and also clear the dropdown for the resolution before adding new ones, in case there is still options left from a previous run.

```
1    void Start(){
2        resolutions = Screen.resolutions;
3        resolutionDropdown.ClearOptions();
4    }
```

Figure 182: Start procedure retrieving resolutions before scene starts

Now that I have retrieved all the resolutions available, I have to edit the dropdown so that I can actually display the different resolution options. Here is how I modified the script so I can change the dropdown:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class GraphicsTab : MonoBehaviour
{
    public TMPro.TMP_Dropdown resolutionDropdown;
    Resolution[] resolutions;
    public void SetResolution(int resolutionIndex)
    {
        Resolution resolution = resolutions[resolutionIndex];
        Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
    }
    public void SetQuality(int qualityIndex)
    {
        QualitySettings.SetQualityLevel(qualityIndex);
    }

    void Start(){
        resolutions = Screen.resolutions;
        resolutionDropdown.ClearOptions();
        List<string> options = new List<string>();
        int currentResolutionIndex = 0;
        for (int i = 0; i < resolutions.Length; i++)
        {
            string option = resolutions[i].width + " x " + resolutions[i].height;
            options.Add(option);
            if (resolutions[i].width == Screen.currentResolution.width && resolutions[i].height == Screen.currentResolution.h

            {
                currentResolutionIndex = i;
            }
        }
        resolutionDropdown.AddOptions(options);
        resolutionDropdown.value = currentResolutionIndex;
        resolutionDropdown.RefreshShownValue();
    }
}

```

Figure 183: Graphics Tab class

I'm using a TMPro dropdown instead of unity's one to keep the way my simulation is built constant as I have my buttons set as a TMPro type. So, I loaded in the TMPro library so that my script can reference an TMPro type dropdown which is named as the resolutionDropdown in this case. Then I created an empty array for the different types of resolutions which will store all the different types of screen resolutions when the start() function appends the resolutions to it. Once these resolutions are in the list, for me to create buttons from the resolutions in this list I'd have to create a new list called options which has the resolutions as a string. I then made it so that the function iteratively goes through the list and adds a new option to the list with the width of the resolution as a string concatenated with the letter x and the height of the string as a string. After the for loop has gone through all the resolutions the options list is added to the resolution dropdown.

Then, just like with setting the quality settings, I created setResolution() method which retrieves the correct resolution from the resolution list and sets this to the current screen resolution.

Test OM-04

Now that I added the needed code to see the resolution options, I can test if it work.

As you can see when I run my simulation the defualt option is now a resolution instead of "Option A" and when I press the dropdown all available resolutions are shown:

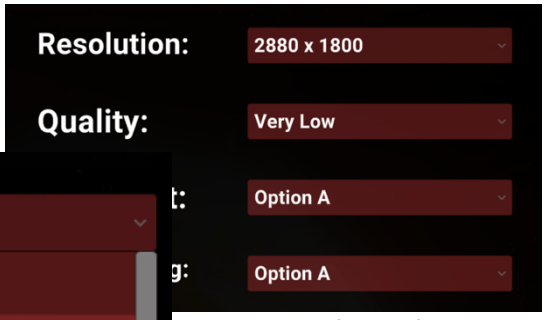


Figure 184: Graphicas Tab dropdowns

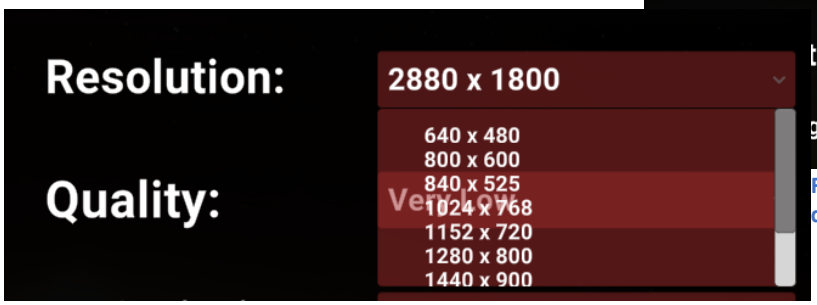


Figure 185: Resolution dropdown options

Test ID	Test Case	Test Data	Expected Results	Passed
OM-04	Resolution Settings	N/A	User is able to configure the resolution	✓

Full Screen

I wanted to do add this feature for users who want to multitask and have the simulation set as a tab rather than a full screen. This will be a very useful feature for teachers who may want to present the simulation and run another educational app whilst teaching. As you can see, I changed the UI to accommodate for a full screen toggle, currently the toggle doesn't have any state (true or false) so there is no circle on either the left or right side to represent that, but when the simulation runs there will be one. For this to happen I have to create a new SetFullScreen() method in my script:

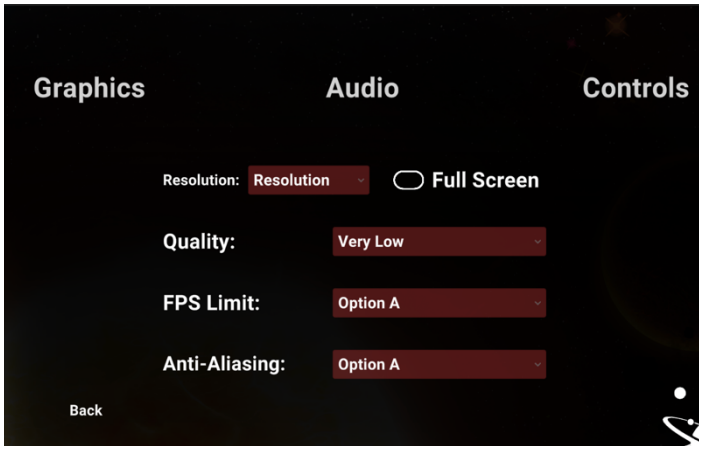


Figure 186: Graphics options, full screen added

As you can see this method is very simple, this is thanks to me using unity. It has built in methods for full screen toggles as it is a feature that is used in nearly all programs.

```
1    public void SetFullScreen(bool isFullScreen)
2    {
3        Screen.fullScreen = isFullScreen;
4    }
```

Figure 187: SetFullScreen function

This is what the slider looks like when on and off:

Figure 188: Full screen toggle

Max FPS

For user who don't want to put a lot of load onto their computer, I added a Max Fps option. I chose to add this because teachers/students won't have high refresh rate screens, their screen are mostly likely about 30 or 60 Hz. So, capping the simulation at 30 or 60 fps will be very useful for them as the simulation won't have to unnecessarily run at a higher framerate which will put a lot of the CPUs resource to waste, because having a high framerate and low refresh rate screen would make it seem like the simulation is running at the refresh rate of the screen instead of the fps of the simulation as that's what the users actually see.

Adding FPS options to the simulation isn't as simple as setting the quality, however isn't as complicated as setting the resolution where you have to retrieve the options. FPS, has standardised values (30, 60, 120, 144, 240) so it is not different for every screen or computer. Here is what I added to my script so I can customise the fps dropdown:

```
void InitializeFPSDropdown()
{
    List<int> fpsOptions = new List<int> { 30, 60, 120, 144, 240 };
    fpsDropdown.ClearOptions();
    List<string> options = new List<string>();
    foreach (int fps in fpsOptions)
    {
        options.Add(fps.ToString() + " FPS");
    }
    fpsDropdown.AddOptions(options);
    fpsDropdown.value = fpsOptions.IndexOf(Application.targetFrameRate);
    fpsDropdown.RefreshShownValue();
}
```

Figure 189: InitialiseFPSDropdown function

It clears all the options already existing In the dropdown, then it adds fps options as strings to the dropdown so the user can select an option then then the selected option is shown in the dropdown.

For the FPS change to actually take place I created the setFPS method like with all other dropdowns:

```
public void setFPS(int fpsIndex)
{
    Application.targetFrameRate = fpsIndex;
}
```

Figure 190: SetFPS function

Anti-Aliasing

The last option to add to the graphic tab is the anti-aliasing option, anti-aliasing allows for smoother snd more realistic images, this will be useful for when the camera is closer to the planet, and instead of the user seeing rough edges of the celestial body assets they can see a smooth version of it, this makes the simulation feel more realistic and have a better quality.

Similar to the resolution and max FPS dropdown, I had to also change the dropdown options, here is the InitialiseAntiAliasingDropdown() method along with the setAntiAliasing() method:

```
1 void InitialiseAntiAliasingDropdown()
2 {
3     List<int> aaOptions = new List<int> { 0, 2, 4, 8 };
4     antiAliasingDropdown.ClearOptions();
5     List<string> options = new List<string>();
6     foreach (int aa in aaOptions)
7     {
8         string option = aa > 0 ? aa.ToString() + "x" : "Off";
9         options.Add(option);
10    }
11    antiAliasingDropdown.AddOptions(options);
12    antiAliasingDropdown.value = aaOptions.IndexOf(QualitySettings.antiAliasing);
13    antiAliasingDropdown.RefreshShownValue();
14 }
15
16 public void setAntiAliasing(int aaIndex)
17 {
18     QualitySettings.antiAliasing = aaIndex;
19 }
```

Figure 191: InitialiseAntiAliasingDropdown function

Audio

I want the user to be able to control the audio to their preferences. This can have many benefits, for example, if a teacher wanted to present the simulation to the class it would be frustrating to have music that they cannot turn down playing whilst they are trying to talk, so having the option to mute the background music will be useful.

Another case where audio is important is for accessibility needs, as a user may not know if they've clicked a planet or a button, so having a click sound play in response to a valid click would be useful for the user to know if they're mis clicking or if the simulation is not working properly.

Background Music

I have three scenes that the user is going to be switching through, and I want my background music to play seamlessly between the scene switching. To do this I will be creating a new script called AudioManager. Here is what it looks like so far:

```
1 using UnityEngine;
2
3 public class AudioManager : MonoBehaviour
4 {
5     private static AudioManager instance;
6     public static AudioManager Instance { get { return instance; } }
7     public AudioClip backgroundMusic;
8     private AudioSource audioSource;
9     //singleton instantation
10    void Awake()
11    {
12        if (instance != null && instance != this)
13        {
14            Destroy(this.gameObject);
15            return;
16        }
17        else
18        {
19            instance = this;
20        }
21        DontDestroyOnLoad(this.gameObject);
22        audioSource = gameObject.AddComponent<AudioSource>();
23        audioSource.clip = backgroundMusic;
24        audioSource.loop = true;
25        audioSource.Play();
26    }
```

Figure 192: Audio Manager Script

I made it so that the script is a singleton, this is because I don't want multiple instances of the audiomanager running, this will be a problem because I want my audio to be controlled centrally by the options menu scene, having different audio managers for different scenes would make my simulation inconsistent and I wouldn't be able to achieve a smooth transition with background music across scenes.
Then I made it so that this instance of audio manager is never destroyed, as I don't want the audio to be cut when there is a scene change, then I added my preferred options for audio, like looping and the actual background music, I obviously chose to loop the background music as I don't want it to cut midway through the simulation, as this will be unsatisfactory and may make the user think that the simulation is buggy or their speakers aren't working.

Test

Now, when I switch through different scenes the background music plays without any interruptions. You can hear the music and see me switching scenes in this video: https://youtu.be/fS55gy8MpxE

Test ID	Test Case	Test Data	Expected Results	Passed
A-1	Background Music	N/A	Background music plays	✓
A-2	Consistent Background Music	N/A	Isn't interrupted between scene switches	✓

Sound effects

Now I can add sound effects to my simulation, I want to add sound effects for planet focusing and button clicks. To do this I'm going to be creating a play sound effect function:

```
private void PlaySoundEffect(AudioClip clip)
{
    if (clip != null)
    {
        AudioSource.PlayClipAtPoint(clip, Camera.main.transform.position, sfxVolume * masterVolume);
    }
}

public void PlayButtonClickSound()
{
    PlaySoundEffect(buttonClickSoundClip);
}

public void PlayPlanetFocusSound()
{
    PlaySoundEffect(planetFocusSoundClip);
}

public void PlayOnSound(){
    PlaySoundEffect(onSoundClip);
}

public void PlayOffSound(){
    PlaySoundEffect(offSoundClip);
}
```

Figure 193: Sound effect functions

The PlaySoundEffect function plays the sound effect clip at the specific SFX and Master volume (I will show how I implemented these later) if the clip passed into the function isn't null. Then I created specific procedures for the different types of sound effects as they are likely to be reused for different buttons.

Test A-3 & A-4

In this video I test out if the button/toggle sound effects work, and the planet focus sound effects:
https://youtu.be/hIpkyt7VZl4

As you can hear from the video, the button sound effects are working perfectly fine, however when focusing on a planet far from the camera, you cannot here the planet focus sound effects, then when I moved to a planet closer to the sun the planet focus audio can be heard.

Test ID	Test Case	Test Data	Expected Results	Passed
A-3	Button SFX Audio	Buttons	Click sound is played when button is pressed	✓
A-4	Celestial Body Focus SFX Audio	Celestial Bodies	Selection sound is played when a celestial body is focused on	✗

Case A-4 Fix

I think this problem is caused by the audio listener, which is attached to the camera, I think the distance of the audio listener is too small. To fix this I can completely remove distance as a factor to the volume of the sound effect played as it is a UI sound effect, which should be the same volume anywhere in the game because it doesn't make sense to have different volumes of clicks next to different planets as the UI doesn't relate to the actual solar system in anyway.

Here is what I added to the AudioManager script so it works:

```
private AudioSource sfxSource;
        void Awake()
        {
            if (instance != null && instance != this)
            {
                Destroy(this.gameObject);
                return;
            }
            else
            {
                instance = this;
            }
            DontDestroyOnLoad(this.gameObject);
            LoadVolumeSettings();
            //sound effects
            sfxSource = gameObject.AddComponent<AudioSource>();
            sfxSource.spatialBlend = 0; // Set to 0 for 2D sound
            //background music
            audioSource = gameObject.AddComponent<AudioSource>();
            audioSource.clip = backgroundMusic;
            audioSource.loop = true;
            UpdateMusicVolume(); // Apply music volume
            audioSource.Play();
        }
private void PlaySoundEffect(AudioClip clip)
{
    if (clip != null)
    {
        sfxSource.PlayOneShot(clip, sfxVolume * masterVolume);
    }
}
```

Figure 194: Updated AudioManager Script

What I did is, I created a new audio source for the sound effects, this is because sound effect has the same volume everywhere, also I thought it would be nice to have my different types of audios separated for a more modular approach, this makes it more useful to pinpoint any errors in audio, and to update audio settings without effecting other parts like the background music. Then I changed the spatial blend of the audio to 0, this makes the audio sound 2D which is what I'm trying to achieve, it sounding 2d means that it sounds the same in every plan as my plane is 3D. I also changed the PlaySoundEffect function so that it now uses the sfxSource instead of the audioSource and I made it so that the passed in audio clip is played as a one shot, instead of being played at a point as I want it to be played the same everywhere instead of a specific point.

Review:

Now everything works! As shown in this video:
https://youtu.be/uHpCQo_6eMg

Test ID	Test Case	Test Data	Expected Results	Passed
A-4	Celestial Body Focus SFX Audio	Celestial Bodies	Selection sound is played when a celestial body is focused on	✓

Audio Tab

To change my audio setting I need the options menu to be able to communicate with the audio manager script. So I first referenced the audiomanager and the audio tab sliders in my options menu script so they can be modified as shown:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.EventSystems;
using UnityEngine.InputSystem;
using UnityEngine.UI;

namespace solsyssim{

    public class OptionsMenu : MonoBehaviour
    {
        [SerializeField] private AudioManager _audioManager;

        // References to sliders
        [SerializeField] private Slider _masterVolumeSlider;
        [SerializeField] private Slider _musicVolumeSlider;
        [SerializeField] private Slider _sfxVolumeSlider;

        private void Start(){
            // Initialize sliders' values from AudioManager and add listeners
            _masterVolumeSlider.value = _audioManager.GetMasterVolume();
            _masterVolumeSlider.onValueChanged.AddListener(SetMasterVolume);

            _musicVolumeSlider.value = _audioManager.GetMusicVolume();
            _musicVolumeSlider.onValueChanged.AddListener(SetMusicVolume);

            _sfxVolumeSlider.value = _audioManager.GetSFXVolume();
            _sfxVolumeSlider.onValueChanged.AddListener(SetSFXVolume);
        }


        public void PlaySolarSystem(){
        SceneManager.LoadScene("StellarSystem");
        }
        public void QuitGame(){
        Application.Quit();
        }
        public void PlayMainMenu(){
        SceneManager.LoadScene("MainMenu");
        }
        public void SetMasterVolume(float volume) {
            _audioManager.SetMasterVolume(volume);
        }

        public void SetMusicVolume(float volume) {
            _audioManager.SetMusicVolume(volume);
        }

        public void SetSFXVolume(float volume) {
            _audioManager.SetSFXVolume(volume);
        }
    }
}
```

The highlighted sections show the changes I've made to the options menu script so that the audio settings can be changed. I

referenced the audio manager and the different audio sliders then in my start function I called the current running singleton instance of the audio manager. Then I added listeners to the sliders in the unity scene for the audio tab, these listeners will be called when the sliders values are changed, and if they are they will call the corresponding methods in thew audio manager to changes the volume to the values of those sliders.

Test OM-02

Now when I run my simulation the volume of the sfx and music are changed as shown in this video:
https://youtu.be/0Siiz0b4gvc

Review

My options menu is now complete, but it hasn't fully met my "Options Menu Features" criteria. I couldn't implement the change of control settings feature. Otherwise, everything works as intended for my options menu.

Evaluation

The final version of my program has all the essential features needed to solve the problem as stated in my analysis. However, some major sections of the program, planned in the design section, have been removed. These are the ability to create a custom system, and to have a functioning tutorial section. These sections were originally planned to be implemented to serve for a more interactive simulation, and at the same time to have an educational benefit, but I had to remove these features due to a huge lack of time. I think it was a sensible choice to choose these two features to remove, as I think they serve more in the interactivity aspect of the simulation rather than solving the educational problem. I think the solar system simulation is enough to learn and understand all aspects of orbital motion.

Also, in my interview stage, my stakeholders didn't request a tutorial or a custom system, their needs were catered for a more accurate, precise simulation which display orbital parameters.

Stakeholder Review

I have given my stakeholders a copy of my simulation so that they can give their opinions, state any limitations, and any usability features that they have liked. So that I can evaluate my simulation with the users that will be using it. Having user input in the evaluation stage can be very beneficial as It will help me know which parts of the simulation

Mr Goosen:

"The simulation looks like it has been professionally made, I love how there are star constellations and how they show the doppler effect. The menus look very nice and I like the added keyboard functionality"

"Limitations wise, I feel like the quantities shown may not be enough, having the option to choose which quantities are displayed would be nice. Also, the ability to change controls would have been nice as well"

"I like how you can change the background music; it would have been very annoying to have it play whilst I'm trying to show the simulation to student"

Dr. Repetto

"This is an amazing simulation Arda, adding asteroid belts were a very good choice and seeing the shadows caused from the sun is pretty cool"

"There are moons missing from other planets, I can only see the earth's moon, so definitely think of adding the moons of all planets"

"I realised that there Is a sound effect whenever I click on a button, this interactive aspect is makes the program look like it's been professionally made"

Criteria Met:

Features added during development are in blue text.

Criteria No.	Criteria	How to Evidence	Criteria Met
C-1	Functioning Main Menu	A video clicking each button	Completed
C-2	Options menu features	Video	Partially Completed
C-3	Consistent Design	Screen shot	Completed
C-4	Tutorial functionality	Screenshots of tutorial screens, task completion	Feature Removed
C-5	Custom System	A video creating a custom system and playing around with it	Feature Removed
C-6	Scene switching	Video of switching to all scenes from every scene	Completed
C-7	Calculations/equations needed for accurate orbital paths	Showing the code that presents the calculations. Presenting maths needed clearly beforehand	Completed
C-8	Select Planet to Track	A video where different celestial bodies are clicked then tracked.	Completed
C-9	Zoom In/out	Screen shot of different viewing angles	Completed
C-10	Planet Graphics	Display images going to be used to present the planets	Completed
C-11	Planet Orbit Line	Screenshot in simulation	Completed
C-12	Time counter	Screenshot in simulation	Completed
C-13	Planet Information	Screenshot in simulation	Completed

C-14	Scale Bar	Screenshot in simulation	Feature Removed
C-15	Visual aids toggling	Option to toggle vectors, grids etc	Couldn't Complete, removed.
C-16	Speed Slider	Screenshot in simulation	Feature removed
C-17	Simulation Speed	Video showing rate of change of time counter	Completed
C-18	Orbit Scale	Video	Completed
C-19	Body Scale	Video	Completed
C-20	Asteroid Belts	Screenshot in simulation	Completed
C-21	Custom Star Background	Screenshot in simulation	Completed
C-22	Star Constellations	Screenshot in simulation	Completed

C-1 Functioning Main Menu

The first criteria have been fully met, the main menu has keyboard and mouse functionality. This is important for users who have different accessibility needs.

Here is the link to the video showing the keyboard and mouse functionality of the main menu: https://youtu.be/fclxA-RuEQk

Figure 195: Start Menu Screenshot

C-2 Options Menu Features

For my options menu, I didn't have time to implement the ability to change the simulation controls, so instead I still kept the controls tab but showcased the controls I chose to implement when creating the game. This is what my options menu looks like: https://youtu.be/DHvkZRvyJl4

This unmet criteria can be resolved in further development by creating a controls script that will be attached to the options menu, and will have changes saved to it for the user to come back to when relaunching the program, instead of having to change the controls within every launch.

C-3 Consistent Design

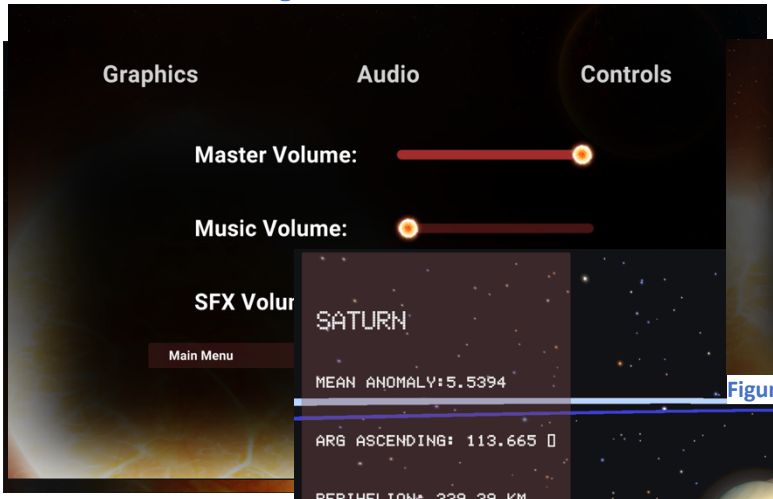


Figure 196: Options Menu Audio Tab Screen Shot

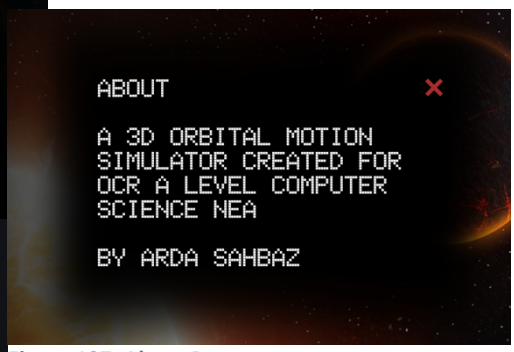


Figure 197: About Pop-up

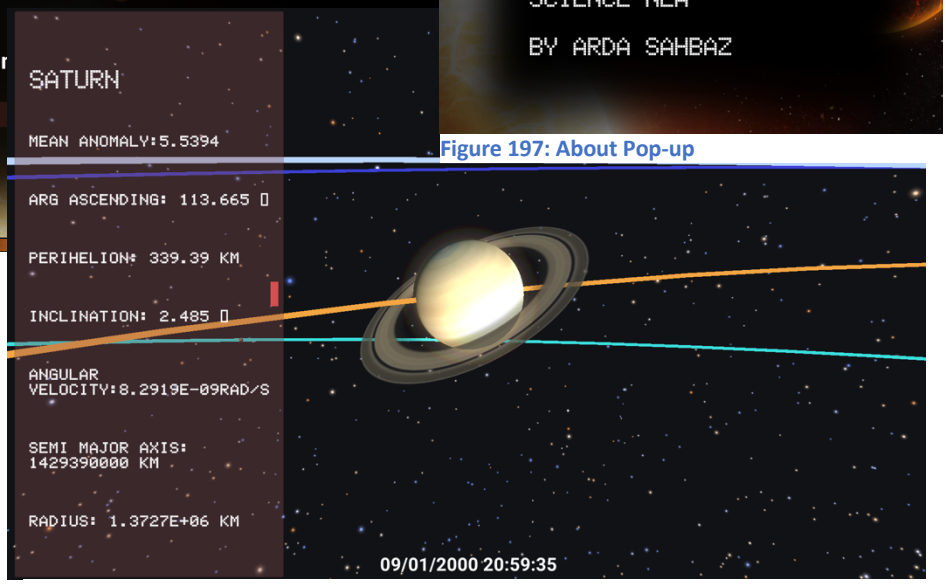


Figure 198: Simulation screenshot with sidebar

As you can see there is a consistent design within. My simulation, red and white are the main colours used within it. The options menu and main menu have the exact same background and look very similar in UI elements. All pop ups are a black rectangles with fading edges.

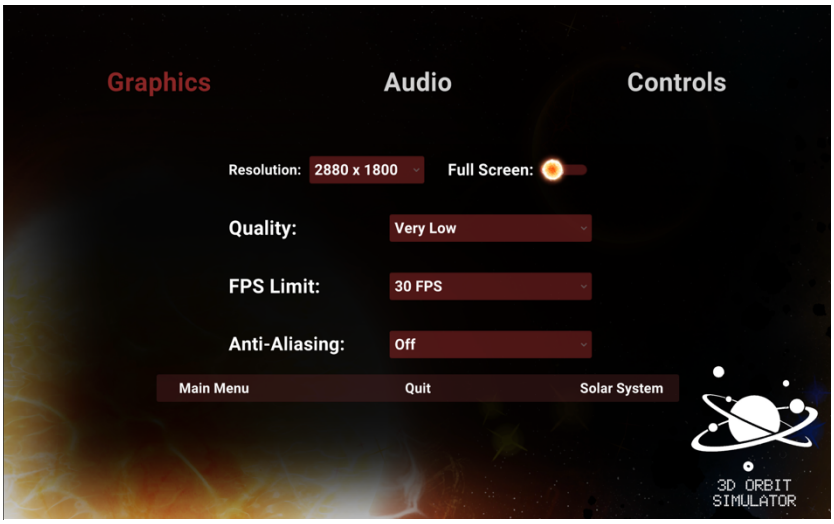


Figure 199: Options Menu Graphics Tab Screen Shot

C-4 Tutorial Functionality

I removed the tutorial feature as a whole as I did not have time, not having a simulation feature doesn't affect my problems solution, as at the end of the day my simulation works and there is control information in the options menu. Also, the tutorial scene being missing means that the simulation doesn't teach users about orbital motion but rather makes the simulation a teaching tool which was what it was intended for anyways.

I think the time I saved by not implementing this feature outweighed the benefits this feature could have brought as I was able to use this spare time to implement more useful features.

However, if I were to implement this unmet criterion in further development I would've done so by creating a duplicated scene of the solar system one and adding tutorial pop ups on how to use the simulation and the physics behind it like it would be in any game tutorial.

C-5 Custom System

Implementing a custom system was definitely beyond my abilities, it would've required a lot more complex physics, due to the lack of already know data about the system that would've been created, which would be impossible to know anyway as this feature would have been completely custom.

If I were to implement this unmet criteria in the future, I would implement with the same design as the solar system, but I do not know how I could make it work with further development as like I said, it is beyond my coding abilities and understanding of physics.

C-6 Switching Scenes

As you can see in this video, I can successfully switch between the 3 scenes:
https://youtu.be/vfsMtVpHOpU

C-7 Calculations/equations needed for accurate orbital paths

```
    public float radius() {
        // from wikepedia its says that radius cna be solved with TA, eccentrity, and semi major axis
        // r = a ((1-e^2)/(1+e*cos(TA)))
        float e = _eccentricity;
        float a = _semiMajorAxis;
        EccentricAnom = EccentricAnomaly(_meanAnomaly, _eccentricity);
        float v = TrueAnomaly(EccentricAnom);
        r = (a) *((1-(e*e))/(1+(e*Mathf.Cos(v))));
        return r;
    }

    private void AdvanceOrbit() {
        // Calculate the distance to the sun
        r = radius();

        float m1 = _stellarParent.gameObject.GetComponent<OrbitalBody>()._mass;
        float m2 = _mass;

        // Calculate angular velocity based on current radius
        _angularVelocity = (Mathf.Sqrt(G * (m1 + m2) / Mathf.Pow(r, 3))) * 0.0001f; //(2f * Mathf.PI / Mathf.Sqrt(Math

        //Debug.Log("angvel" + _angularVelocity);
        // update mean anomaly
        _meanAnomaly += _angularVelocity * SpaceTime.Instance.DeltatTime;
        if (_meanAnomaly > 2f * Mathf.PI)  // we keep mean anomaly within 2*Pi
            _meanAnomaly -= 2f * Mathf.PI;

        Vector3 orbitPos = GetPosition(_meanAnomaly, "advanceOrbit");
        Vector3 parentPos = _stellarParent.transform.position; // position of the parent to offset the calculated pos

        transform.position = new Vector3(orbitPos.x + parentPos.x, orbitPos.y + parentPos.y, orbitPos.z + parentPos.z)

    }

    private static float EccentricAnomaly(float M, float _eccentricity, int dp = 5) {
        // Mathematical Model is as follow:
        // E(n+1) = E(n) - f(E) / f'(E)
        // f(E) = E - e * sin(E) - M
        // f'(E) = 1 - e * cos(E)
        // we are happy when f(E)/f'(E) is small enough.

        int maxIter = 20;  // we make sure we won't loop too much
        int i = 0;

        float precision = Mathf.Pow(10, -dp);
        float E, F;

        // If the eccentricity is high we guess the Mean anomaly for E, otherwise we guess PI.
        E = (_eccentricity < 0.8) ? M : Mathf.PI;
        F = E - _eccentricity * Mathf.Sin(M) - M;  //f(E)

        // We will interate until f(E) higher than our wanted precision (as devided then by f'(E)).
        while ((Mathf.Abs(F) > precision) && (i < maxIter)) {
            E = E - F / (1f - _eccentricity * Mathf.Cos(E));
            F = E - _eccentricity * Mathf.Sin(E) - M;
            i++;
        }

        return E;
    }



    /// Computes the True Anomaly. Angles to be passed in Radians.

    /// <returns>The True Anomaly.</returns>
    /// <param name="E">Eccentric Anomaly.</param>
    private float TrueAnomaly(float E) {
        // from wikipedia we can find several way to solve TA from E.
        // I tried sin(TA) = (sqrt(1-e*e) * sin(E))/(1 -e*cos(E)) but it didn't work properly for some reason,
        // so I sued the followlng as one of the my sources(jgiesen.de/Kepler) tan(TA) = (sqrt(1-e*e) * sin(E)) / (cos


        float e = _eccentricity;
        float numerator = Mathf.Sqrt(1f - e * e) * Mathf.Sin(E);
        float denominator = Mathf.Cos(E) - e;
        float TA = Mathf.Atan2(numerator, denominator);
        return TA;
    }




    //Compute a point's position in a given orbit. All angles are to be passed in Radians and returns the point coordi

    public Vector3 GetPosition(float M, string callerName) {

        float a = OrbitScaled; // semiMajorAxis
        float N = _argAscending * Mathf.Deg2Rad; // not const as might vary with precession
        float w = _argPerihelion * Mathf.Deg2Rad;
        float i = _inclination * Mathf.Deg2Rad;

        float E = EccentricAnomaly(M, _eccentricity);
        float TA = TrueAnomaly(E);
        float focusRadius = a * (1 - Mathf.Pow(_eccentricity, 2f)) / (1 + _eccentricity * Mathf.Cos(TA)); //distance f

        // Debug.Log($"{callerName} - Eccentric Anomaly: {E}");
        // Debug.Log($"{callerName} - True Anomaly: {TA}");
        // Debug.Log($"{callerName} - Focus Radius: {focusRadius}");
        // parametric equation of an ellipse using the orbital elements
        float X = focusRadius * (Mathf.Cos(N) * Mathf.Cos(TA + w) - Mathf.Sin(N) * Mathf.Sin(TA + w)) * Mathf.Cos(i);

        float Y = focusRadius * Mathf.Sin(TA + w) * Mathf.Sin(i);
        float Z = focusRadius * (Mathf.Sin(N) * Mathf.Cos(TA + w) + Mathf.Cos(N) * Mathf.Sin(TA + w)) * Mathf.Cos(i);


        Vector3 orbitPoint = new Vector3(X, Y, Z);

        return orbitPoint;
    }


}
```

Figure 200: OrbitalBody.cs code with calculation scripts highlighted

C-8 Select Planet to Track

This video shows me selecting different planets at different angles
https://youtu.be/sE2Lr8btzHc

C-9 Zoom In/out, C-10 Planet Graphics, C-11 Planet Orbit Lines, C-12 Time Counter

These pictures show all the criterias listed above In the heading.

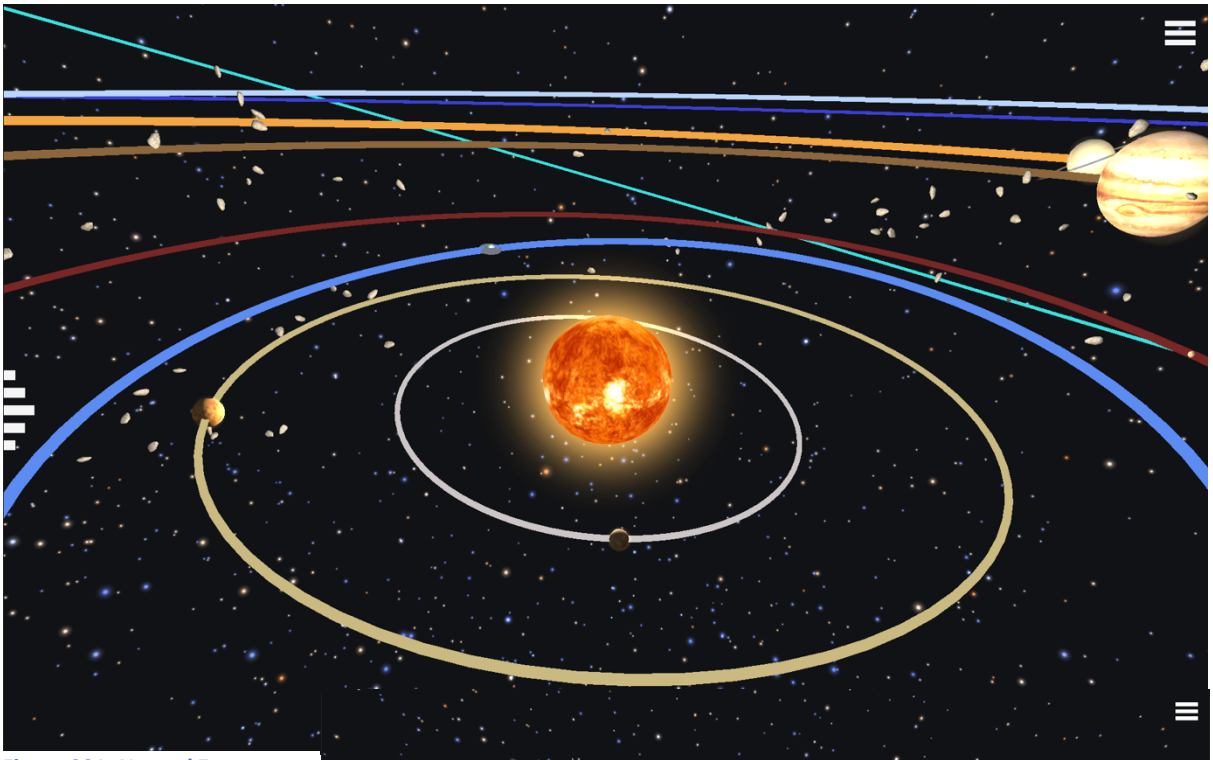


Figure 201: Normal Zoom

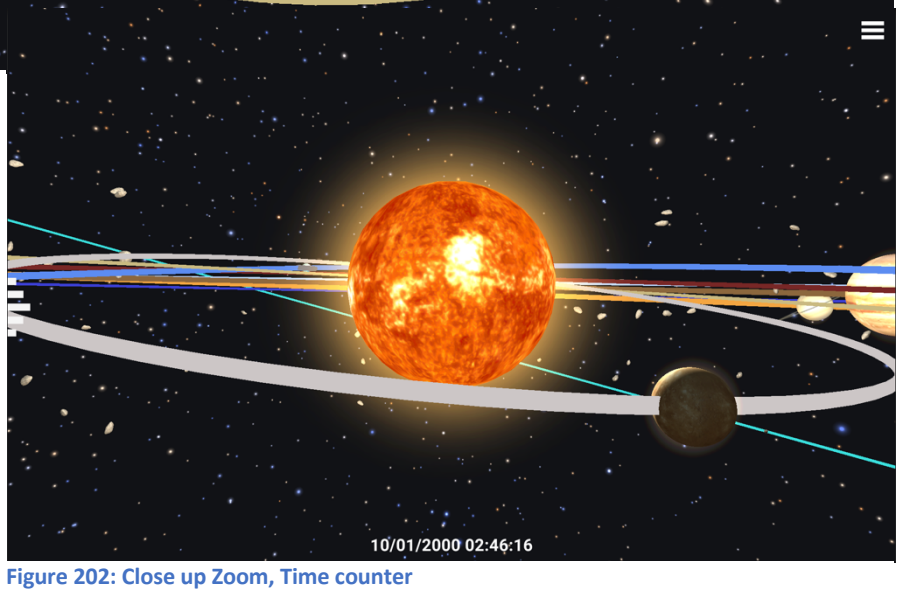


Figure 202: Close up Zoom, Time counter

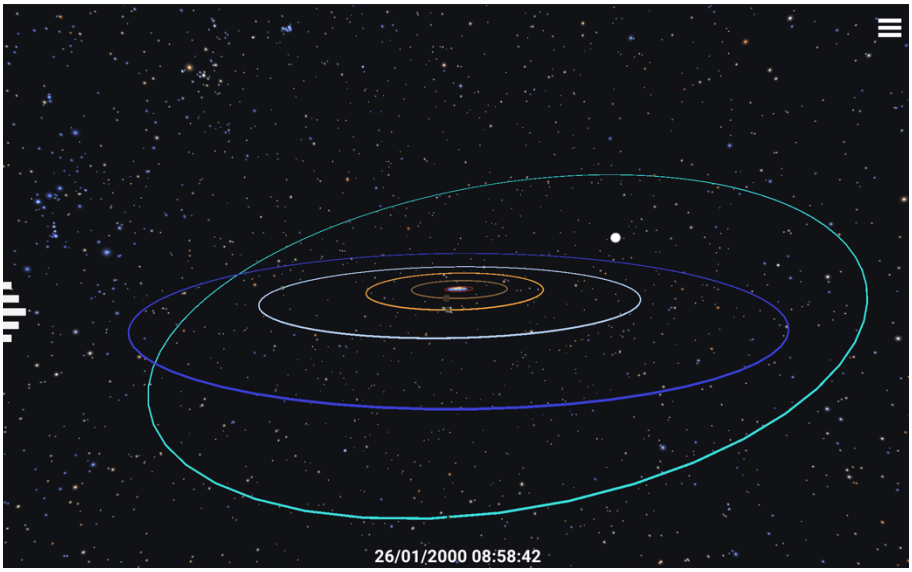


Figure 203: Far away zoom

C-13 Planet Information

Here is a picture with the sidebar open showing all the planet information:

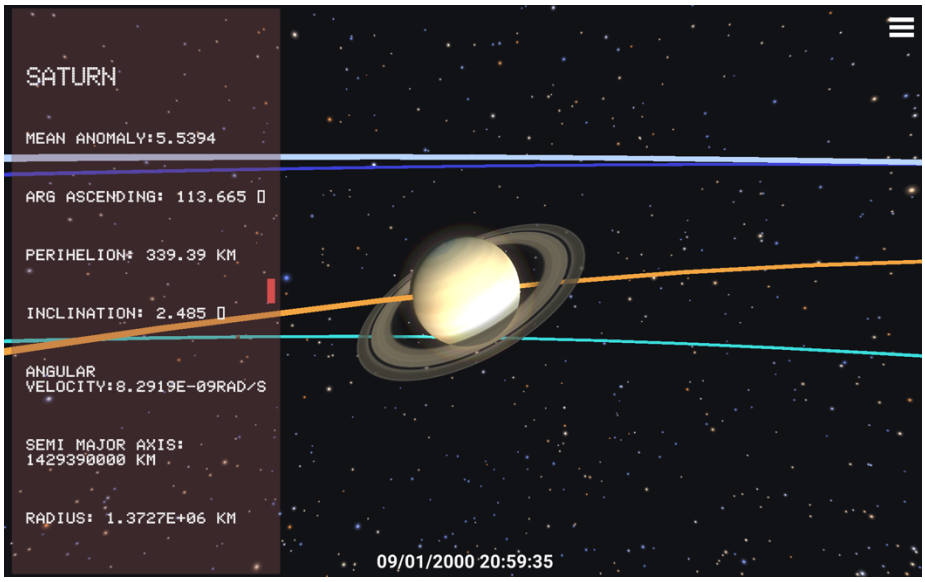


Figure 204: Saturn focus with sidebar displaying

C-14 Scale Bar

I did not implement this feature not only because of lack of time but I now think it is very useless, this is because I couldn't implement the grid lines, without a grid line having a scale bar will be pointless as there is nothing to use it with.

C-15 Visual Aids Toggling

I struggled to implement the grid lines feature, it didn't work properly my whole simulation was white as there were too many gridlines being processed by the camera, thus overloading my RAM and making the simulation extremely laggy, so having a 3D grid line was impossible. I could've added a 2D one in the y = 0 plane but I don't think this would be

been very useful and would just make the simulation look unpleasing, and because the simulation is 3D and the orbits are inclined I would be very hard to use, which is why I decided to scrap the feature as a whole.

With the vectors, I simply didn't have enough time, it would've been very time consuming to plan out and create as I couldn't find any implementations of 3D vectors in orbits online, also the positioning of vectors can be easily explained by a teacher when teaching.

This unmet criteria can be resolved in further development by utilising newton's laws and the existing mass data to display vectors of varying sizes in the simulation with the same scale as the body scales

C-16 Speed Slider

I scrapped this feature as I think it is not needed, this is because it is a waste of space, the user can already change the speed with their keyboard and mouse, so having this control be on screen when others aren't would've made my simulation inconsistent and cluttery.

C-17 Simulation Speed, C-18 Orbit Scale, C-19 Body Scale

As you can see in this video:
https://youtu.be/IxDylqZ0bBE

All the scaling functions work, the time orbit and body scale changes depending on the user's input. However due to timing restrictions these scaling functions aren't perfect, I don't have limits put in place for how much the user can change the scale by, this can make the simulation unusable in some scales, for example, in the body scale I made the bodies extremely big that it was hard to see the orbit lines, however this lack of restriction in scaling isn't a big problem, the user will likely not go that far in their scaling. So, having this feature not perfected isn't a big problem and doesn't affect the solution to my problem in any major way.

Body and orbit scale were features that I decided to implement for the user to have more customisability over the simulation, I felt like having the simulation be an extremely fixed single perspective view of the solar system would've made the simulation non engaging, thus making my simulation less attractive to students.

C-20 Asteroid Belts

Asteroid belts were one of the biggest additional features I added to my simulation. It is something that I am very glad I didn't miss, as it is a vital part of the educational problem I'm trying to solve, many students don't know the existence of asteroid belts in our solar system, and which planets they are in between in our solar system, so these celestial objects being added, which are normally ignored in most solar system simulations, makes my simulation much more realistic and informative compared to others, which yet again helped solve the problem mentioned in my analysis.

I implemented the Kuiper belt and the inner asteroid belt between mars and Jupiter and this is what they look like:

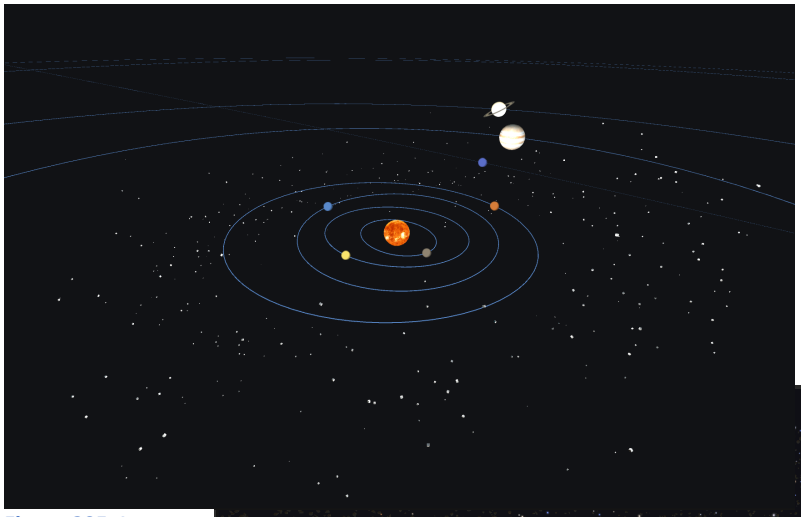


Figure 205: Inner asteroid belt

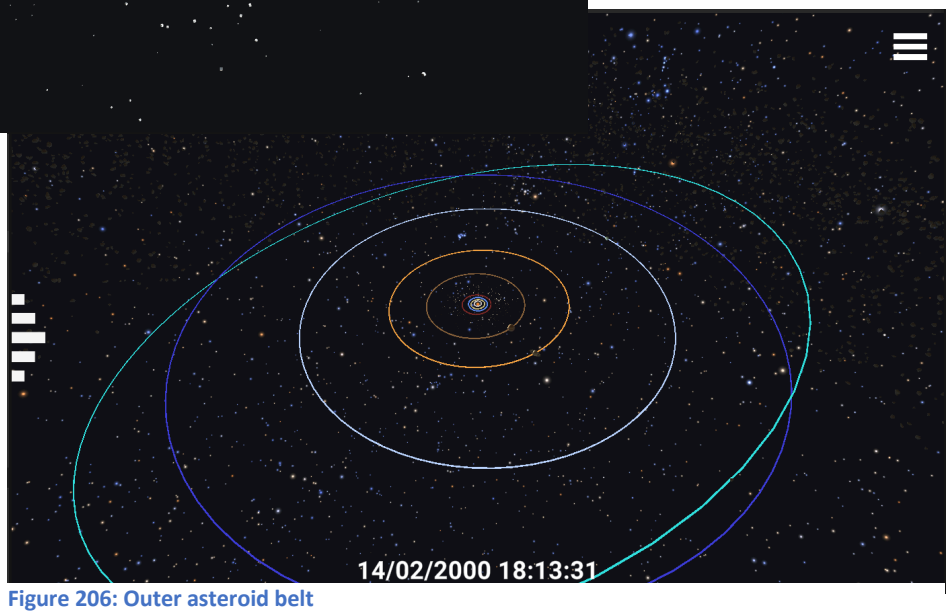


Figure 206: Outer asteroid belt

C-21 Custom Star Background, C-22 Star Constellations

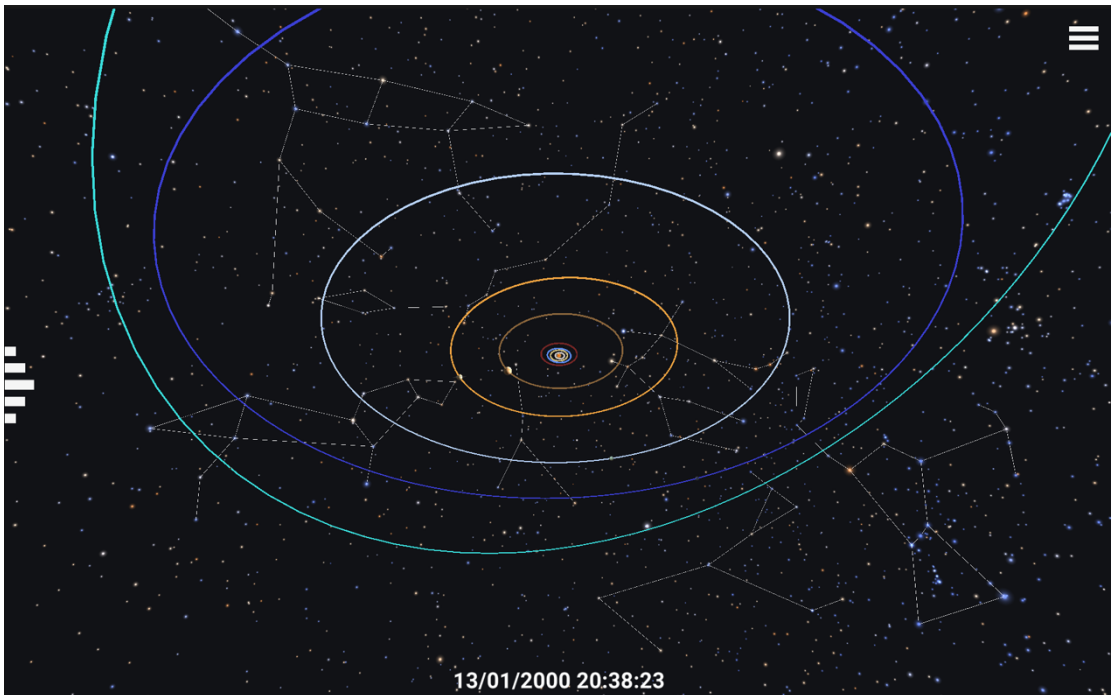


Figure 207: Screenshot of all star constellations toggled on

The star constellations are toggled on using the number keys. Figure 206 shows what they look like.

Usability Features

Button Highlighting & Keyboard Accessibility

Buttons being highlighted when hovered over by a mouse or selected by a keyboard is a useful feature for users who don't have a mouse and want to use a keyboard. It allows them to know which button is selected before they click on them and to know if they have clicked on a button.

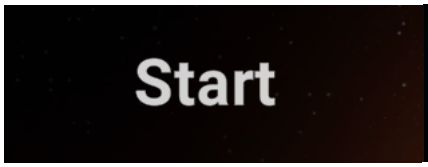


Figure 210: Button Normal Colour

Figure 209: Button Highlighted Colour

Figure 208: Button Pressed Colour

Alternatively, I could've saved time by not Implementing this usability feature, but it would've made my simulation very hard to launch through a keyboard, it would almost be like a guessing game to know which button is hovered over by the keyboard. This would've made it extremely inconvenient to use a keyboard thus less accessible for people who cannot use a mouse.

Also, I added keyboard functionality to my buttons as some people may not be physically able to use a mouse. For example, in secondary school my teacher had RSI, meaning she couldn't hold pens or use a mouse, having multiple ways of controlling the simulation is very important for people like her with accessibility needs, especially when my simulation is being used in an educational setting, where everyone should be able to receive the same level of education.

The combination of these two features is extremely effective together, from me testing the simulation and using it, I've realised that its much quicker to launch it and interact with it. There being only a few keys that are used in the simulation means my hands are constantly in the same position, this makes to extremely quick to find what I'm looking for in the simulation.

Pre Access to Options Menu

Having access to the options menu before launching the simulation is crucial for usability, providing an upfront settings panel which the user can configure for their accessibility needs before facing any issues which could inhibit their experience. For example, users with hearing problem can turn down the music volume as it can be too loud or irritating for them.

This usability feature is effective for problems outside of accessibility, for example changing the resolution, quality, and fps before the simulation starts which can prevent crashes if the user has a weak system. However, when talking about accessibility, because I removed the option to change the user controls this usability feature doesn't have the same effect as planned, like I mentioned in the previous paragraph it can help people with hearing people,

but one demographic it could've been really effective for is for people who can't use keyboard or mice, they cannot pair their own input device specialised for their disabilities to use in my system.

With future development, this usability feature can be more effective with a more complex access to the simulation settings. Having the ability to change the control options for people with physical disabilities, and having the option for a colourblind mode will make having pre access to the simulation setting before launching it more useful.

SFX Audio

Having sound effects for button clicks as shown in this video: https://youtu.be/hIpkyt7VZl4 is very useful for people with visual impairments, if they are struggling to see where they are clicking they can know anyways through the sound effects.

This usability feature is effective, it prevents the simulation from solely relying on visual feedback, which could cause unnecessary confusion for those who cannot see the screen clearly. Having menu selections and focus selections emit a confirmation tone assists user on whether they have clicked a button or not, without visual button highlighting.

Dynamic Line Thickness

This is one usability feature I didn't plan for but ended up implementing through testing of my orbital line functions. Having the line thickness change for different zoom positions is an extremely important accessibility feature this is because when zoomed out the orbital line for further planets would like extremely tiny, this would be a pain to look out for people with visual impairments.

This usability feature is extremely effective, this is because without the feature when focused on Pluto and zoomed out to the maximum distance, the orbital line for nearly every planet is barely visible and is about a pixel thin, this usability feature solving this problem and preventing eye strain for people with visual impairments deems it very effective.

Limitations

Limitations of my simulation has been pinpointed with robust testing and user feedback.

User Customisability

One of my biggest limitations is the lack of user customisability. Currently, the simulation does not offer custom control options for users to tailor the way they use the simulation to their preferences.

Another customisability limitation is the ability for the user to select which celestial bodies are included in the simulation, and which orbital parameters are displayed in the sidebar, along with the unit they are displayed in, for them to retrieve the data they need of the celestial bodies.

Also, saving these preferences would have made the simulator more interactive and personalised.

How I can deal with this limitation

I can deal with this limitation with more development time, adding user customisability is a process that can open the simulator to a lot of bugs and errors, and for this limitation to be scrapped a lot of time would need to be invested to have a robust simulator.

Scale Context

This simulation obviously had to be scaled, as the solar system is way too big, but a big limitation that comes with this is the ability for users to intuitively grasp the actual sizes and distance involved within the simulation.

How I can deal with this limitation

With more development time, incorporating scale bars (which I originally scrapped from my design due to lack of time) and a set to scale function which will provide an actual representation of the solar system would solve this limitation.

Accessibility

Another limitation, which I tried to prevent as much as possible, is user accessibility. I didn't consider users who use touch only devices, like tablets, which are commonly found in educational settings. Expanding my simulation to support touch and maybe controller interactivity will increase inclusivity which is a very important thing in educational settings.

How I can deal with this limitation

I can deal with this simulation by surveying other programs in unity, looking at how they use controllers in simulations would help me find which buttons are commonly used for common control in simulations.

Also, with more development time I could have solved this limitation. However, although inclusivity is important, this limitation is only a problem for a very small minority, so solving other implementations would be more worthwhile.

Missing Celestial Bodies

A limitation which one of my stakeholders mentioned is the lack of celestial bodies. There is a lack of artificial satellites and only the earth has its moon in the simulation and no other planets do.

However, I don't think this isn't too big of a limitation as the other celestial bodies are enough to show the physics of orbital motion which is what my simulation is aiming to do anyways.

How I can deal with this limitation

This limitation can be dealt with by finding more data, which is also reliable, on other celestial bodies and increasing development time so that these celestial bodies can be implemented into the simulation.

Maintenance

My program is very friendly to future maintenance, every component has its own script making parts of the program very modular and easy to upgrade/fix. Every script has comments within every line so that future developers can follow the code and know what they are dealing with.

The whole simulation being built upon object-oriented programming makes it very maintainable, there is many reusable methods, which will come in handy for when new features are added in as methods don't have to be recoded which saves time. Or, for when a method needs improving, since it is reusable not every part of the code that has that method needs to be changed, only one portion of the code needs to be changed. For example, the way GetPosition is calculated can be changed to a more efficient way without having to repeat the change all over the program which saves time.

The program is also split into 3 different scenes, options menu, main menu, and stellar system. Having 3 different scenes for three major parts of the simulation keeps the program organised and allows for changes to be implemented within unity very easily as the scenes are standalone components meaning a change in one scene would play zero effect in another.

Post Development Testing

https://youtu.be/jPjB0pW0VTI

Robustness Test

https://youtu.be/xK-bgk9lA6w

References:

Star Data:
http://tdc-www.harvard.edu/catalogs/bsc5.html

Planet Data:
https://nssdc.gsfc.nasa.gov/planetary/factsheet/index.html

Unity Menu Background:
https://assetstore.unity.com/packages/tools/particles-effects/mk-space-background-parallax-maker-93514

UI Pack:
https://assetstore.unity.com/packages/2d/gui/icons/ux-flat-icons-free-202525

Planet Pack:
https://assetstore.unity.com/packages/3d/environments/planets-of-the-solar-system-3d-90219

Asteroid Pack:
https://assetstore.unity.com/packages/3d/environments/asteroids-pack-84988

Star Colour:
https://arxiv.org/pdf/2101.06254.pdf

Final Code

OrbitalBody.cs

```
using UnityEngine;
using solsyssim;

namespace solsyssim {

    // Script managing the behaviour of the Orbital axis and
it's child orbital body.
    // Sets the initial body parameter.
    // Updates the scales of all parameters if necessary.
    // Updates the orbit rotation and the body's day rotation.

    public class OrbitalBody : MonoBehaviour
    {
        // reference to object in the hierarchy
        [SerializeField] public GameObject _stellarParent;
        [SerializeField] public GameObject _body;
        [SerializeField] private Material _line;


        // Variables related to the orbital axis.
        [SerializeField] public float _sideralYear;
        [SerializeField] public float _mass = 0f;
        [SerializeField] public float _semiMajorAxis = 10f;
        [SerializeField] public float _inclination = 0f;
        [SerializeField] [Range(0f, 1f)] public float
_eccentricity = 0f;
        [SerializeField] public float _argAscending = 0f; //
longitude
        [SerializeField] public float _argPerihelion = 0f; //
argument
        [SerializeField] public float _periapsis = 0f;
        [SerializeField] public float _apoapsis = 0f;
        [SerializeField] public float _startMeanAnomaly = 0f;
// J2000
        [SerializeField] public float _meanAnomaly;


        // Variables related to the orbital body.
```

```
        [SerializeField] private float _dayLength = 1f;
        [SerializeField] private float _startDayRotation = 0f;
        [SerializeField] private float _size = 1f;
        [SerializeField] public float _rightAscension = 0f;
        [SerializeField] public float _declination = 0f;
        private const float _eclipticTilt = 23.44f;
        public float _angularVelocity;
        public float EccentricAnom;
        public float r;
        public float TA;
        private const float G = 6.674e-11F;
        public float v;

        // Semi Constants
        public float SGP;
        public float MAM;
        public float TAC;
        public float CosLOAN;
        public float SinLOAN;
        public float CosI;
        public float SinI;

        // Scales from the StellarSystem script.
        public float SizeScaled { private set; get; }
        public float OrbitScaled { private set; get; }

        // planet UI color and path variables
        [SerializeField] private Color _uiVisual;
        public Color UiVisual { get { return _uiVisual; } }
        private bool _showPath = false;

        //For debugginf reference
        DateCalc dateCalc;
        GameObject gameObject;
        public string name;

        // registering to some events and initialising stuff
        private void Start() {

            SpaceTime.Instance.ScaleUpdated += UpdateScale;
            //
FindObjectOfType<InterfaceManager>().OrbitToggle +=
TogglePath;
            // FindObjectOfType<InterfaceManager>().FullStart
+= TogglePath;

            SetScales();
            SetJ2000();
            CalcSemiConstants();
```

```
            AdvanceOrbit();

            //Debugging

            //OrbitalDebug debug = new OrbitalDebug();
            //debug.name = name;
            //debug.eccentricAnomaly =
EccentricAnomaly(_meanAnomaly);
            //debug.trueAnomaly =
TrueAnomaly(EccentricAnomaly(_meanAnomaly));

        }

        private void OnDestroy() {
            SpaceTime.Instance.ScaleUpdated -= UpdateScale;
        }

        public void CalcSemiConstants(){

            float argrad = _argAscending * Mathf.Deg2Rad;
            float incrad = _inclination * Mathf.Deg2Rad;
            SGP =
_stellarParent.gameObject.GetComponent<OrbitalBody>()._mass *
G;
            MAM = Mathf.Sqrt(SGP /
(float)Mathf.Pow(_semiMajorAxis, 3));
            TAC = Mathf.Sqrt((1 + _eccentricity) / (1 -
_eccentricity));

            CosLOAN = Mathf.Cos(argrad);
            SinLOAN = Mathf.Sin(argrad);

            CosI = Mathf.Cos(incrad);
            SinI = Mathf.Sin(incrad);

        }

        // advance the body on it's orbit and it's rotation on
itself separately
        private void Update() {
            AdvanceOrbit();
            AdvanceDayRotation();
            radius();

            //For UI

            //Debug.Log("Name:"+ name);
            //Debug.Log("Eccentric Anomaly: " +
EccentricAnomaly(_meanAnomaly));
```

```
            //Debug.Log("True Anomaly: " +
TrueAnomaly(EccentricAnomaly(_meanAnomaly)));
        }

        // renders or not the orbit path
        // private void OnRenderObject() {
        //     if (_showPath)
        //         DrawPath();
        // }


        // Initialises the orbital body state as per the J2000
data.

        private void SetJ2000() {
            transform.Rotate(new Vector3(0, 0, (90 -
_declination))); //Rotates object around its local axis in
Euler angles
            transform.Rotate(new Vector3(0, -_rightAscension,
0));
            transform.Rotate(new Vector3(_eclipticTilt, 0,
0));
            transform.Rotate(new Vector3(0, _startDayRotation,
0), Space.Self);
            _angularVelocity = Mathf.Deg2Rad * (360 /
_sideralYear); // How much degree is covered each day.
            _meanAnomaly = _startMeanAnomaly * Mathf.Deg2Rad;
//Converts to radians
        }



        // Initialises the scales from the StellarSystem.

        private void SetScales() {
            OrbitScaled = _semiMajorAxis *
SpaceTime.Instance.OrbitScale;
            SizeScaled = _size * SpaceTime.Instance.BodyScale;
            _body.transform.localScale = Vector3.one *
SizeScaled;
        }


        // Toggles the orbit path. Called on events.

        // public void TogglePath() {
        //     _showPath = _showPath;
        // }
```

```
        // Updates the scale. Called on event from the
StellarSystem script.

        // <param name="variable">Which scale.</param>
        public void UpdateScale(SpaceTime.Scale scale, float
value) {
            switch (scale) {

                case SpaceTime.Scale.Body:
                    SizeScaled = _size * value;
                    _body.transform.localScale = Vector3.one *
SizeScaled;
                    break;

                case SpaceTime.Scale.Orbit:
                    OrbitScaled = _semiMajorAxis * value;
                    break;

                case SpaceTime.Scale.Time:
                    break;

                default:
                    Debug.LogWarning("Wrong variable name in
Body.updateScales() .");
                    break;
            }
        }


        // Calculates how much the body has moved in it's
orbit and rotate it's axis accordingly.



        //public float CalculateAngulaVelocity() {
        //    float m1 =
GameObject.Find("Sun").GetComponent<OrbitalBody>()._mass;
        //    float m2 = _mass;

        //    float sum = _G * (m1+m2);
        //    float _angVel =
(radius()*m1*(Mathf.Sqrt(sum*((2/radius())-
(1/_semiMajorAxis)))))/(m1*radius()*radius());

        //    return _angVel;

        //}

        //Finds radius from True Anomaly
```

```
        public float radius() {
            // from wikepedia its says that radius cna be
solved with TA, eccentrity, and semi major axis
            // r = a ((1-e^2)/(1+e*cos(TA)))
            float e = _eccentricity;
            float a = _semiMajorAxis;
            EccentricAnom = EccentricAnomaly(_meanAnomaly,
_eccentricity);
            float v = TrueAnomaly(EccentricAnom);
            r = (a) *((1-(e*e))/(1+(e*Mathf.Cos(v))));
            return r;
        }

        private void AdvanceOrbit() {
            // Calculate the distance to the sun
            r = radius();

            float m1 =
_stellarParent.gameObject.GetComponent<OrbitalBody>()._mass;
            float m2 = _mass;

            // Calculate angular velocity based on current
radius
            _angularVelocity = (Mathf.Sqrt(G * (m1 + m2) /
Mathf.Pow(r, 3))) * 0.0001f; //(2f * Mathf.PI /
Mathf.Sqrt(Mathf.Pow(r, 3)/G*m1)) ;
            //Debug.Log("angvel" + _angularVelocity);
            // update mean anomaly
            _meanAnomaly += _angularVelocity *
SpaceTime.Instance.DeltatTime;
            if (_meanAnomaly > 2f * Mathf.PI)  // we keep mean
anomaly within 2*Pi
                _meanAnomaly -= 2f * Mathf.PI;

            Vector3 orbitPos = GetPosition(_meanAnomaly,
"advanceOrbit");
            Vector3 parentPos =
_stellarParent.transform.position; // position of the parent
to offset the calculated pos (used for moons)
            transform.position = new Vector3(orbitPos.x +
parentPos.x, orbitPos.y + parentPos.y, orbitPos.z +
parentPos.z);
        }


        // Rotates the body on itself as per it's day length.
        private void AdvanceDayRotation() {
            float rot = SpaceTime.Instance.DeltatTime * (360 /
_dayLength);
```

```
            _body.transform.Rotate(new Vector3(0, -rot, 0));
// counter clockwise is the standard rotation considered
        }


        // Draws the orbit by calculating each next point in a
circle.
        // Then using the GL.Lines to draw a line between each
point.


        // private void DrawPath() {
        //         const float PathDetail = 0.03f;
        //         GL.PushMatrix();  // Push the current
transformation matrix onto the matrix stack.
        //         _line.color = UiVisual;
        //         _line.SetPass(0);  // Set the material pass
to the first file in the ui folder to render the line.

        //         GL.Begin(GL.LINES);  // Begin rendering
lines using OpenGL.
        //         Vector3 parentPos =
_stellarParent.transform.position;  // Get the position of the
parent object.
        //         // Loop through a full circle by the
specified path detail and create a line for each step.
        //         for (float theta = 0.0f; theta < (2f *
Mathf.PI); theta += PathDetail) {
        //             float anomalyA = theta;
        //             Vector3 orbitPointA =
GetPosition(anomalyA);
        //             GL.Vertex3(orbitPointA.x + parentPos.x,
orbitPointA.y + parentPos.y, orbitPointA.z + parentPos.z);

        //             float anomalyB = theta + PathDetail;
        //             Vector3 orbitPointB =
GetPosition(anomalyB);
        //             GL.Vertex3(orbitPointB.x + parentPos.x,
orbitPointB.y + parentPos.y, orbitPointB.z + parentPos.z);
        //         }
        //         GL.End();
        //         GL.PopMatrix();
        // }



        // Computes the Eccentric Anomaly. Angles to be passed
in Radians.
```

```
        // The Newton method used to solve E implies getting a
first guess and itterating until the value is precise enough.

        //  returns the Eccentric Anomaly.
        private static float EccentricAnomaly(float M, float
_eccentricity, int dp = 5) {
            // Mathematical Model is as follow:
            // E(n+1) = E(n) - f(E) / f'(E)
            // f(E) = E - e * sin(E) - M
            // f'(E) = 1 - e * cos(E)
            // we are happy when f(E)/f'(E) is small enough.

            int maxIter = 20;  // we make sure we won't loop
too much
            int i = 0;

            float precision = Mathf.Pow(10, -dp);
            float E, F;

            // If the eccentricity is high we guess the Mean
anomaly for E, otherwise we guess PI.
            E = (_eccentricity < 0.8) ? M : Mathf.PI;
            F = E - _eccentricity * Mathf.Sin(M) - M;  //f(E)

            // We will interate until f(E) higher than our
wanted precision (as devided then by f'(E)).
            while ((Mathf.Abs(F) > precision) && (i <
maxIter)) {
                E = E - F / (1f - _eccentricity *
Mathf.Cos(E));
                F = E - _eccentricity * Mathf.Sin(E) - M;
                i++;
            }

            return E;
        }



        // Computes the True Anomaly. Angles to be passed in
Radians.
        private float TrueAnomaly(float E) {
            // from wikipedia we can find several way to solve
TA from E.
            // I tried sin(TA) = (sqrt(1-e*e) * sin(E))/(1 -
e*cos(E)) but it didn't work properly for some reason,
            // so I sued the following as one of the my
sources(jgiesen.de/Kepler) tan(TA) = (sqrt(1-e*e) * sin(E)) /
(cos(E) - e).
```

```
            float e = _eccentricity;
            float numerator = Mathf.Sqrt(1f - e * e) *
Mathf.Sin(E);
            float denominator = Mathf.Cos(E) - e;
            float TA = Mathf.Atan2(numerator, denominator);
            return TA;
        }




        //Compute a point's position in a given orbit. All
angles are to be passed in Radians and returns the point
coordinates
        public Vector3 GetPosition(float M, string callerName)
{

            float a = OrbitScaled; // semiMajorAxis
            float N = _argAscending * Mathf.Deg2Rad; // not
const as might vary with precession
            float w = _argPerihelion * Mathf.Deg2Rad;
            float i = _inclination * Mathf.Deg2Rad;

            float E = EccentricAnomaly(M, _eccentricity);
            float TA = TrueAnomaly(E);
            float focusRadius = a * (1 -
Mathf.Pow(_eccentricity, 2f)) / (1 + _eccentricity *
Mathf.Cos(TA)); //distance from focus to object
            // Debug.Log($"{callerName} - Eccentric Anomaly:
{E}");
            // Debug.Log($"{callerName} - True Anomaly:
{TA}");
            // Debug.Log($"{callerName} - Focus Radius:
{focusRadius}");
            // parametric equation of an elipse using the
orbital elements
            float X = focusRadius * (Mathf.Cos(N) *
Mathf.Cos(TA + w) - Mathf.Sin(N) * Mathf.Sin(TA + w)) *
Mathf.Cos(i);
            float Y = focusRadius * Mathf.Sin(TA + w) *
Mathf.Sin(i);
            float Z = focusRadius * (Mathf.Sin(N) *
Mathf.Cos(TA + w) + Mathf.Cos(N) * Mathf.Sin(TA + w)) *
Mathf.Cos(i);

            Vector3 orbitPoint = new Vector3(X, Y, Z);
```

```
            return orbitPoint;
        }


    }

}
```

OrbitalUI.cs

```
using System.Collections.Generic;
using UnityEngine;
namespace solsyssim {
    // This script is responsible for managing UI elements
    //It uses linerender to draw the orbit lines
    [RequireComponent(typeof(LineRenderer))]
    public class OrbitalUI : MonoBehaviour
    {
        [SerializeField] private LineRenderer lineRenderer;
        [SerializeField] private OrbitalBody orbitalBody;
        //Higher value means smoother orbit lines
        private int resolution = 80;

        private List<Vector3> points = new List<Vector3>();
        public LineRenderer Line;
        //ises the getposition function in orbitalbody to
calculate the points of the orbital path
        public void GetPoints()
        {
            //clears existing points just in case to avoid any
errors
            points.Clear();

            Vector3 centre =
orbitalBody._stellarParent.transform.position;

            float orbitFraction = 1f / resolution;
            //iterates through based on the resolution and
adds a specific number of points corresponding to that
resolution
            for (int i = 0; i < resolution; i++)
            {
                float meanAnomaly = i * orbitFraction * 2f *
Mathf.PI;
                Vector3 pointPosition =
orbitalBody.GetPosition(meanAnomaly, "getpoints");
                // Debug.Log(pointPosition);
                points.Add(centre + pointPosition);
            }
            lineRenderer.positionCount = points.Count;
```

```
            UpdateLines();
        }


        //resets to defualt settings
        private void Reset()
        {
            lineRenderer = GetComponent<LineRenderer>();
            lineRenderer.shadowCastingMode =
UnityEngine.Rendering.ShadowCastingMode.Off;
            lineRenderer.startWidth = 0.1f;
            lineRenderer.endWidth = 0.1f;
            lineRenderer.loop = true;

            orbitalBody = GetComponent<OrbitalBody>();
        }

        private void Start()
        {
            orbitalBody = GetComponent<OrbitalBody>();
            orbitalBody.CalcSemiConstants();
            //skips adding orbit lines to sun as it is not
needed
            if (orbitalBody.name != "Sun") {
                Reset();
                GetPoints();
                UpdateLines();
                //GenerateMeshCollider();
            }

        }

        private void Update() {
            GetPoints();
        }

        //updtes the orbital line
        private void UpdateLines()
        {
            for (int i = 0; i < resolution; i++)
            {
                lineRenderer.SetPosition(i, points[i]);
            }

            // Calculate the distance from the camera to the
object
            float distance =
Vector3.Distance(Camera.main.transform.position,
transform.position);
```

```
            // Adjust the line width based on the distance
            float lineWidth = Mathf.Clamp(distance / 100,
0.1f, 2f); // Adjust the divisor and limits as needed

            lineRenderer.startWidth = lineWidth;
            lineRenderer.endWidth = lineWidth;
        }

        //scrapped feature

        // public void GenerateMeshCollider()
        // {
        //     MeshCollider collider =
GetComponent<MeshCollider>();
        //     if (collider == null)
        //     {
        //         collider =
orbitalBody._stellarParent.gameObject.AddComponent<MeshCollide
r>();
        //     }

        //     Mesh mesh = new Mesh();

        //     lineRenderer.BakeMesh(mesh, Camera.main,
false);
        //     collider.sharedMesh = mesh;

        // }

    }
}
```

StarDataLoader.cs

```
using System.Collections.Generic;
using System.IO;
using UnityEngine;

public class StarDataLoader {
  //this script loads the star data from the harvard yale star
catalogue
  public class Star {

    // Three variables used to define the star in the game.
    public float catalog_number;
    public Vector3 position;
    public Color colour;
    public float size;
```

```
    // Keep the original points so we can recalculate based on
dates.
    private readonly double right_ascension;
    private readonly double declination;
    private readonly float ra_proper_motion;
    private readonly float dec_proper_motion;

    // Constructor
    public Star(float catalog_number, double right_ascension,
double declination, byte spectral_type,
                byte spectral_index, short magnitude, float
ra_proper_motion, float dec_proper_motion) {
      this.catalog_number = catalog_number;
      // Save the location parameters.
      this.right_ascension = right_ascension;
      this.declination = declination;
      this.ra_proper_motion = ra_proper_motion;
      this.dec_proper_motion = dec_proper_motion;
      // Set the position
      position = GetBasePosition();
      // Set the Colour.
      colour = SetColour(spectral_type, spectral_index);
      // Set the Size.
      size = SetSize(magnitude);
    }

    // Get the starting position shown in the file.
    public Vector3 GetBasePosition() {
      // Place stars on a cylinder using 2D trigonometry.
      double x = System.Math.Cos(right_ascension);
      double y = System.Math.Sin(declination);
      double z = System.Math.Sin(right_ascension);

      // Pull in ends to make the sphere
      // Work out y-adjacent and use this to scale (as on unit
sphere)
      double y_cos = System.Math.Cos(declination);
      x *= y_cos;
      z *= y_cos;

      // Return as float
      return new((float)x, (float)y, (float)z);
    }

    private Color SetColour(byte spectral_type, byte
spectral_index) {
      Color IntColour(int r, int g, int b) {
        return new Color(r / 255f, g / 255f, b / 255f);
      }
```

```
      // OBAFGKM colours from:
https://arxiv.org/pdf/2101.06254.pdf
      Color[] col = new Color[8];
      col[0] = IntColour(0x5c, 0x7c, 0xff); // O1
      col[1] = IntColour(0x5d, 0x7e, 0xff); // B0.5
      col[2] = IntColour(0x79, 0x96, 0xff); // A0
      col[3] = IntColour(0xb8, 0xc5, 0xff); // F0
      col[4] = IntColour(0xff, 0xef, 0xed); // G1
      col[5] = IntColour(0xff, 0xde, 0xc0); // K0
      col[6] = IntColour(0xff, 0xa2, 0x5a); // M0
      col[7] = IntColour(0xff, 0x7d, 0x24); // M9.5

      int col_idx = -1;
      if (spectral_type == 'O') {
        col_idx = 0;
      } else if (spectral_type == 'B') {
        col_idx = 1;
      } else if (spectral_type == 'A') {
        col_idx = 2;
      } else if (spectral_type == 'F') {
        col_idx = 3;
      } else if (spectral_type == 'G') {
        col_idx = 4;
      } else if (spectral_type == 'K') {
        col_idx = 5;
      } else if (spectral_type == 'M') {
        col_idx = 6;
      }

      // If unknown, makes it white.
      if (col_idx == -1) {
        return Color.white;
      }

      // Maps second part 0 -> 0, 10 -> 100
      float percent = (spectral_index - 0x30) / 10.0f;
      return Color.Lerp(col[col_idx], col[col_idx + 1],
percent);
    }

    private float SetSize(short magnitude) {

      return 1 - Mathf.InverseLerp(-146, 796, magnitude);
    }
  }

  public List<Star> LoadData() {
    List<Star> stars = new();
    // Openss the binary file for reading.
```

```
    const string filename = "BSC5";
    TextAsset textAsset = Resources.Load(filename) as
TextAsset;
    MemoryStream stream = new(textAsset.bytes);
    BinaryReader br = new(stream);
    // Reads the headers
    int sequence_offset = br.ReadInt32();
    int start_index = br.ReadInt32();
    int num_stars = -br.ReadInt32();
    int star_number_settings = br.ReadInt32();
    int proper_motion_included = br.ReadInt32();
    int num_magnitudes = br.ReadInt32();
    int star_data_size = br.ReadInt32();

    // Reads one field at a time.
    for (int i = 0; i < num_stars; i++) {
      float catalog_number = br.ReadSingle();
      double right_ascension = br.ReadDouble();
      //Finds angular distance from celestial equator.
      double declination = br.ReadDouble();
      byte spectral_type = br.ReadByte();
      byte spectral_index = br.ReadByte();
      short magnitude = br.ReadInt16();
      float ra_proper_motion = br.ReadSingle();
      float dec_proper_motion = br.ReadSingle();
      // Creates new star class
      Star star = new(catalog_number, right_ascension,
declination, spectral_type, spectral_index, magnitude,
ra_proper_motion, dec_proper_motion);
      stars.Add(star);
    }

    return stars;
  }

}
```

StarField.cs

```
using System.Collections.Generic;
using UnityEngine;
namespace solsyssim{
  //this method uses the star data script and renders the
star. It also renders constellations
  public class StarField : MonoBehaviour {
    [Range(0, 100)]
    [SerializeField] private float starSizeMin = 0.1f;
    [Range(0, 100)]
    [SerializeField] private float starSizeMax = 3f;
    private List<StarDataLoader.Star> stars;
```

```
    private List<GameObject> starObjects;
    private Dictionary<int, GameObject> constellationVisible =
new();

    private readonly int starFieldScale = 400;

    void Start() {
      // Read in the star data.
      StarDataLoader sdl = new();
      stars = sdl.LoadData();
      starObjects = new();
      foreach (StarDataLoader.Star star in stars) {
        // Create star game objects.
        GameObject stargo =
GameObject.CreatePrimitive(PrimitiveType.Quad);
        stargo.transform.parent = transform;
        stargo.name = $"HR {star.catalog_number}";
        stargo.transform.localPosition = star.position *
starFieldScale;
        stargo.transform.localScale = Vector3.one *
Mathf.Lerp(starSizeMin, starSizeMax, star.size);
        stargo.transform.LookAt(transform.position);
        stargo.transform.Rotate(0, 180, 0);
        Material material =
stargo.GetComponent<MeshRenderer>().material;
        material.shader = Shader.Find("Unlit/StarShader");
        material.SetFloat("_Size", Mathf.Lerp(starSizeMin,
starSizeMax, star.size));
        material.color = star.colour;
        starObjects.Add(stargo);
      }
    }
//rotates the star field around the camera so stars are in
correct position from user perspective
    private void FixedUpdate() {
      if (Input.GetKey(KeyCode.Mouse1)) {

Camera.main.transform.RotateAround(Camera.main.transform.posit
ion, Camera.main.transform.right, Input.GetAxis("Mouse Y"));

Camera.main.transform.RotateAround(Camera.main.transform.posit
ion, Vector3.up, -Input.GetAxis("Mouse X"));
      }
      return;
    }

    private void OnValidate() {
      if (starObjects != null) {
        for (int i = 0; i < starObjects.Count; i++) {
```

```
            // Updates the size set in the shader.
            Material material =
starObjects[i].GetComponent<MeshRenderer>().material;
            material.SetFloat("_Size", Mathf.Lerp(starSizeMin,
starSizeMax, stars[i].size));
        }
      }
    }

    // A constellation is a tuple of the stars and the lines
that join them. So second list is the lines joining stars in
order and first is just a list of the stars that exist within
the constellation
    private readonly List<(int[], int[])> constellations =
new() {
      // Orion
      (new int[] { 1948, 1903, 1852, 2004, 1713, 2061, 1790,
1907, 2124,
                   2199, 2135, 2047, 2159, 1543, 1544, 1570,
1552, 1567 },
      new int[] { 1713, 2004, 1713, 1852, 1852, 1790, 1852,
1903, 1903, 1948,
                   1948, 2061, 1948, 2004, 1790, 1907, 1907,
2061, 2061, 2124,
                   2124, 2199, 2199, 2135, 2199, 2159, 2159,
2047, 1790, 1543,
                   1543, 1544, 1544, 1570, 1543, 1552, 1552,
1567, 2135, 2047 }),
      // Monceros
      (new int[] { 2970, 3188, 2714, 2356, 2227, 2506, 2298,
2385, 2456, 2479 },
      new int[] { 2970, 3188, 3188, 2714, 2714, 2356, 2356,
2227, 2714, 2506,
                   2506, 2298, 2298, 2385, 2385, 2456, 2479,
2506, 2479, 2385 }),
      // Gemini
      (new int[] { 2890, 2891, 2990, 2421, 2777, 2473, 2650,
2216, 2895,
                   2343, 2484, 2286, 2134, 2763, 2697, 2540,
2821, 2905, 2985},
      new int[] { 2890, 2697, 2990, 2905, 2697, 2473, 2905,
2777, 2777, 2650,
                   2650, 2421, 2473, 2286, 2286, 2216, 2473,
2343, 2216, 2134,
                   2763, 2484, 2763, 2777, 2697, 2540, 2697,
2821, 2821, 2905, 2905, 2985 }),
      // Cancer
      (new int[] {3475, 3449, 3461, 3572, 3249},
```

```
      new int[] {3475, 3449, 3449, 3461, 3461, 3572, 3461,
3249}),
      // Leo
      (new int[] { 3982, 4534, 4057, 4357, 3873, 4031, 4359,
3975, 4399, 4386, 3905, 3773, 3731 },
      new int[] { 4534, 4357, 4534, 4359, 4357, 4359, 4357,
4057, 4057, 4031,
                  4057, 3975, 3975, 3982, 3975, 4359, 4359,
4399, 4399, 4386,
                  4031, 3905, 3905, 3873, 3873, 3975, 3873,
3773, 3773, 3731, 3731, 3905 }),
      // Leo Minor
      (new int[] { 3800, 3974, 4100, 4247, 4090 },
      new int[] { 3800, 3974, 3974, 4100, 4100, 4247, 4247,
4090, 4090, 3974 }),
      // Lynx
      (new int[] { 3705, 3690, 3612, 3579, 3275, 2818, 2560,
2238 },
      new int[] { 3705, 3690, 3690, 3612, 3612, 3579, 3579,
3275, 3275, 2818,
                  2818, 2560, 2560, 2238 }),
      // Ursa Major
      (new int[] { 3569, 3594, 3775, 3888, 3323, 3757, 4301,
4295, 4554, 4660,
                  4905, 5054, 5191, 4518, 4335, 4069, 4033,
4377, 4375 },
      new int[] { 3569, 3594, 3594, 3775, 3775, 3888, 3888,
3323, 3323, 3757,
                  3757, 3888, 3757, 4301, 4301, 4295, 4295,
3888, 4295, 4554,
                  4554, 4660, 4660, 4301, 4660, 4905, 4905,
5054, 5054, 5191,
                  4554, 4518, 4518, 4335, 4335, 4069, 4069,
4033, 4518, 4377, 4377, 4375 }),
    };

    private void Update() {
      // Checks for numeric presses and toggles the
constellation highlighting.
      for (int i = 0; i < 10; i++) {
        if (Input.GetKeyDown(KeyCode.Alpha0 + i)) {
          ToggleConstellation(i);
        }
      }
    }

    private void ToggleConstellation(int index) {
      // Safety check to see if the index is valid.
      if ((index < 0) || (index >= constellations.Count)) {
```

```
            return;
        }

        // Toggles constellations on or off.
        if (constellationVisible.ContainsKey(index)) {
            Destroy(constellationVisible[index]);
            constellationVisible.Remove(index);
        } else {
            CreateConstellation(index);
        }
    }

    private void CreateConstellation(int index) {
        int[] constellation = constellations[index].Item1;
        int[] lines = constellations[index].Item2;

        // Change the colours of the stars
        // foreach (int catalogNumber in constellation) {
        //   // Remember list is 0-up catalog numbers are 1-up.
        //   starObjects[catalogNumber -
1].GetComponent<MeshRenderer>().material.color = Color.white;
        // }

        GameObject constellationHolder = new($"Constellation
{index}");
        constellationHolder.transform.parent = transform;
        constellationVisible[index] = constellationHolder;

        // Draws the constellation lines.
        for (int i = 0; i < lines.Length; i += 2) {
            // Parent it to our constellation object so we can
delete them all later.
            GameObject line = new("Line");
            line.transform.parent = constellationHolder.transform;
            // Defaults to white and width 1 which works for us.
            LineRenderer lineRenderer =
line.AddComponent<LineRenderer>();
            // Doesn't get assigned a material so just dig out one
that works.
            lineRenderer.material = new
Material(Shader.Find("Legacy Shaders/Particles/Alpha Blended
Premultiply"));
            // Disable useWorldSpace so it will track the parent
game object.
            lineRenderer.useWorldSpace = false;
            Vector3 pos1 = starObjects[lines[i] -
1].transform.position;
            Vector3 pos2 = starObjects[lines[i + 1] -
1].transform.position;
```

```
        // Offsets them so they don't occlude the stars, 3
chosen by trial and error.
        Vector3 dir = (pos2 - pos1).normalized * 1;
        lineRenderer.positionCount = 2;
        lineRenderer.SetPosition(0, pos1 + dir);
        lineRenderer.SetPosition(1, pos2 - dir);
      }
    }

    private void RemoveConstellation(int index) {
      int[] constallation = constellations[index].Item1;

      // Toggling off set the stars back to the original
colour.
      foreach (int catalogNumber in constallation) {
        // Remember list is 0-up catalog numbers are 1-up.
        starObjects[catalogNumber -
1].GetComponent<MeshRenderer>().material.color =
stars[catalogNumber - 1].colour;
      }
      // Remove the constellation lines.
      Destroy(constellationVisible[index]);
      // Remove from our dictionary as it's now off.
      constellationVisible.Remove(index);
    }
  }
}
```

SpaceTime.cs

```
using UnityEngine;
using System;

namespace solsyssim {

    // Handles all scales and dimention of space and time for
the current gameinstance.

    public class SpaceTime : MonoBehaviour {


        private static SpaceTime _instance;
        public static SpaceTime Instance {
            get {
                 if (_instance == null) {
                     if (_instance == null){
                         Debug.LogError("SpaceTime Singleton
not yet instanciated.");
                         _instance = new SpaceTime();
                     }
```

```
                }
                return _instance;
            }
        }

        private void Awake() {
            if (_instance != null && _instance != this) {
                Debug.LogWarning("Double instance of SpaceTime
Singleton!");
                Destroy(this.gameObject);
            } else {
                _instance = this;
            }
        }

        // enum of all the different accessible scales. made
it enum as it will be useful for a swtich statement
        public enum Scale{Time, Orbit, Body};
        private float _bodyScale = BaseBodyScale; //defualt
scale

        public float BodyScale {
            get {
                return _bodyScale;
            }
            private set {
                //clamps the values the scale can be to a
specific range
                _bodyScale = Mathf.Clamp(value,
MinDefaultScale, MaxDefaultScale);
                //Updates scale if needed
                if(ScaleUpdated != null)
                    ScaleUpdated.Invoke(Scale.Body,
_bodyScale);
            }
        }

        private float _orbitScale = BaseOrbitScale;

        public float OrbitScale {
            get { return _orbitScale; }
            private set {
                //clamps the values the scale can be to a
specific range
                _orbitScale = Mathf.Clamp(value,
MinDefaultScale, MaxDefaultScale);
                //Updates scale if needed
                if(ScaleUpdated != null)
```

```
                    ScaleUpdated.Invoke(Scale.Orbit,
_orbitScale);
                }
            }

            // time scale allows you to pause the game
            // it also has a totaltime counter and the last scaled
deltatime
            //simulation starts with time not paused
            public bool _timePause = false;
            private float _lastTimeScale = BaseTimeScale;
            private float _timeScale = BaseTimeScale;

            public float TimeScale {
                get { return _timeScale; }
                //checks if game is unpaused, if so it clamps the
values the scale for the speed of time can be and updates it
                private set {
                    if(!_timePause){
                        _timeScale = Mathf.Clamp(value,
MinTimeScale, MaxTimeScale);
                        if(ScaleUpdated != null)
                            ScaleUpdated.Invoke(Scale.Time,
_timeScale);
                    }
                }
            }

            private double _elapsedTime = 0;
            public double ElapsedTime {
                get { return _elapsedTime; }
                private set { _elapsedTime = value; }
            }

            public float DeltatTime {
                get { return Time.deltaTime * TimeScale; }
            }

            // constant parameters
            private const float BaseBodyScale = 0.01f;
            private const float BaseOrbitScale = 0.0001f;
            private const float BaseTimeScale = 0.5f;
            private const float MinTimeScale = 0.01f;
            private const float MaxTimeScale = 500.0f;
            private const float MinDefaultScale = 0.0001f; //
probably replace default by specifics for body and orbits
            private const float MaxDefaultScale = 1.0f;

            // SpaceTime Signals
```

```
        public event Action<Scale, float> ScaleUpdated;

        // registering to events
        private void Start() {
            ControlIntentions.Instance.Scaling += UpdateScale;
            ControlIntentions.Instance.PauseGame += PauseTime;
        }

        private void OnDestroy() {
            ControlIntentions.Instance.Scaling -= UpdateScale;
            ControlIntentions.Instance.PauseGame -= PauseTime;
        }

        // updating the elapsed time each frame depending on
frame and scale
        private void Update() {
            ElapsedTime += Time.deltaTime * TimeScale;
        }

        // listening to events from inputs
        // modify the correct scale accordingly
        private void UpdateScale(Scale scale, float value) {

            switch(scale) {

                case Scale.Body:
                BodyScale *= 1f + value;
                break;

                case Scale.Orbit:
                OrbitScale *= 1f + value;
                break;

                case Scale.Time:
                TimeScale *= 1f + value;
                break;

                default:
                Debug.LogWarning("Unknown scale when updating
scales.");
                break;
            }
        }

        public void TogglePause() {
            bool isPaused = SpaceTime.Instance.TimeScale == 0;
            SpaceTime.Instance.PauseTime(!isPaused);
        }
```

```
        // Sets timescale to the minimum when pause is
pressed.
        public void PauseTime(bool pause) {
            if(pause){
                _lastTimeScale = TimeScale;
                TimeScale = 0;
                _timePause = true;
            //and resets it back to original when pressed
again
            } else {
                _timePause = false;
                TimeScale = _lastTimeScale;
            }
             ScaleUpdated?.Invoke(Scale.Time, _timeScale);

        }

        // Resets the different scales to their default
constant values.
        public void ResetAllScales() {
            BodyScale = BaseBodyScale;
            OrbitScale = BaseOrbitScale;
            TimeScale = BaseTimeScale;
        }

    }
}
```

ControlIntentions.cs

```
using UnityEngine;
using System;

namespace solsyssim {

    // This script handles all control input and checks the
game state it sends a signal to whoever is interested in what
inpu has been given by the user.
    public class ControlIntentions : MonoBehaviour {

        // Singleton instantiation
        private static ControlIntentions _instance;
        public static ControlIntentions Instance {
            get {
                // if (_instance == null){
                //     if (_instance == null){
                //         Debug.LogError("Singleton not yet
instanciated.");
```

```
                //          _instance = new
ControlIntentions();
                //      }
                // }
                return _instance;
            }
        }

        // checks the singleton instance
        private void Awake() {
            if (_instance != null && _instance != this) {
                Debug.LogWarning("Double instance of
ControlIntentions Singleton!");
                Destroy(this.gameObject);
            } else {
                _instance = this;
            }
        }

        // Property to let the programe create fake key press
input
        private static String _simulatedInput = "";
        public static String SimulatedInput {
            set {
                _simulatedInput = value;
            }
            get {
                string _key = _simulatedInput;
                _simulatedInput = "";
                return _key;
            }
        }

        // Game/input state var and related events
        public enum State { Game, Menu, Scaling };
        private State _state = State.Menu;

        public event Action<SpaceTime.Scale, float> Scaling;
        private void RaiseScaling(SpaceTime.Scale scale, float
value) {
            if(Scaling != null)
                Scaling.Invoke(scale, value);
        }
        public event Action<string, float> CamRotation;
        private void RaiseCamRotation(string axe, float dir) {
            if(CamRotation != null)
                CamRotation.Invoke(axe, dir);
        }
        public event Action<float> CamTranslation;
```

```
        private void RaiseCamTranslation(float value) {
            if(CamTranslation != null)
                CamTranslation.Invoke(value);
        }
        public event Action<Vector3> FocusSelection;
        private void RaiseFocusSelection(Vector3
mousePosition) {
            if(FocusSelection != null)
                FocusSelection.Invoke(mousePosition);
        }
        public event Action<bool> MenuCall;
        private void RaiseMenuCall(bool call) {
            if(MenuCall != null)
                MenuCall.Invoke(call);
        }
        public event Action<bool> PauseGame;
        private void RaisePauseGame(bool pause) {
            _gamePaused = pause;
            if(PauseGame != null)
                PauseGame.Invoke(_gamePaused);
        }
        private bool _gamePaused = true;
        private void Update () {
            switch(_state){

                case State.Game:
                    CheckGameInput();
                    break;

                case State.Menu:
                    CheckMenuInput();
                    break;

                case State.Scaling:
                    CheckScalingInput();
                    break;

                default:
                    Debug.LogError("Unknown game state when
checking for input.");
                    break;
            }
        }

        private void CheckGameInput() {
            // check condition for changing state
            if (Input.GetKeyDown(KeyCode.Escape) ||
SimulatedInput == "menu" || Input.GetButtonDown("menu")) {
                _state = State.Menu;
```

```
                RaisePauseGame(!_gamePaused);
                RaiseMenuCall(true);
            } else if ((Input.GetButtonDown("scale time") ||
Input.GetButtonDown("scale orbits") ||
Input.GetButtonDown("scale bodies"))){
                _state = State.Scaling;
            }

            // pause is not a full state as besides time,
control should reamin operational
            if (Input.GetButtonDown("pause game"))
                RaisePauseGame(!_gamePaused);// wrote this
instwead of true so that s asingle button cna iterate through
true and false

            // Rotation of the cam around the center
horizontally (on the y axis of Axis).
            if (Input.GetAxis("rotate cam horizontally") != 0)
                RaiseCamRotation("horizontal",
Input.GetAxis("rotate cam horizontally"));

            // Rotation of the cam around the center
vertically (on the x axis of Pole).
            if (Input.GetAxis("rotate cam vertically") != 0)
                RaiseCamRotation("vertical",
Input.GetAxis("rotate cam vertically"));

            // Translation of the cam on the z axis (from and
away)
            if (Input.GetAxis("translate cam (zoom)") != 0)
                RaiseCamTranslation(Input.GetAxis("translate
cam (zoom)"));

            // Focus Body Selection
            if (Input.GetMouseButtonDown(0)) {
                //Debug.Log("Attempting to focus on a body at
position: " + Input.mousePosition);
                RaiseFocusSelection(Input.mousePosition);
    }
        }

        private void CheckMenuInput() {
            // check condition for changing state
            // hardcoded to always be able to access menus and
quit
            if (Input.GetKeyDown(KeyCode.Escape) ||
SimulatedInput == "menu" || Input.GetButtonDown("menu")) {
                _state = State.Game;
                RaiseMenuCall(false);
```

```
            }
        }

        private void CheckScalingInput() {

            // check condition for changing state
            if (!Input.GetButton("scale time") &&
!Input.GetButton("scale orbits") && !Input.GetButton("scale
bodies"))
                _state = State.Game;

            // check user input
            if (Input.GetButton("scale bodies"))
                RaiseScaling(SpaceTime.Scale.Body,
Input.GetAxis("scale axis")); //"scale axis" is/should be the
"Mouse ScrollWheel"
            else if (Input.GetButton("scale orbits"))
                RaiseScaling(SpaceTime.Scale.Orbit,
Input.GetAxis("scale axis"));
            else if (Input.GetButton("scale time"))
                RaiseScaling(SpaceTime.Scale.Time,
Input.GetAxis("scale axis"));
        }

    }
}
```

BeltObject.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace solsyssim {
    // this script uss the beltspawner script to construct a
whole asterid belt as one obejct in unity
    public class BeltObject : MonoBehaviour
    {
        // Serialized fields for object properties
        [SerializeField] private float orbitSpeed;
        [SerializeField] private GameObject parent;
        [SerializeField] private bool rotationClockwise;
        [SerializeField] private float rotationSpeed;
        [SerializeField] private Vector3 rotationDirection;

        // Initializes the belt object with given parameters
        public void SetupBeltObject(float _speed, float
_rotationSpeed, GameObject _parent, bool _rotateClockwise)
        {
            orbitSpeed = _speed;
            rotationSpeed = _rotationSpeed;
```

```
            parent = _parent;
            rotationClockwise = _rotateClockwise;
            rotationDirection = new Vector3(Random.Range(0,
360), Random.Range(0, 360), Random.Range(0, 360));
        }

        // Handles orbit and self-rotation of the object
        private void Update()
        {
            float deltaTime =
(float)SpaceTime.Instance.DeltatTime;
            Vector3 rotationAxis = rotationClockwi(se ?
parent.transform.up : -parent.transform.up;
            transform.RotateAround(parent.transform.position,
rotationAxis, orbitSpeed * deltaTime);
            transform.Rotate(rotationDirection, rotationSpeed
* deltaTime);
        }
    }
}
```

BeltSpawner.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace solsyssim {

    // The BeltSpawner script is responsible for generating an
asteroid belt within specified parameters.
    //It creates asteroids with varied positions, scales, and
rotations to simulate a belt around a central point.

    public class BeltSpawner : MonoBehaviour
    {
        [Header("Spawner Settings")]
        //list of prefabs for variation in asteroids
        public GameObject[] cubePrefabs;
        public int cubeDensity;
        public float eccentricity;
        public int seed;
        public float innerRadius;
        public float outerRadius;
        public float height;
        public bool rotatingClockwise;
        //asteroid size range
        public float minScale = 0.1f;
        public float maxScale = 0.2f;
        public float maxInclination = 30f;
```

```
        // rotation and orbit speeds
        [Header("Asteroid Settings")]
        public float minOrbitSpeed;
        public float maxOrbitSpeed;
        public float minRotationSpeed;
        public float maxRotationSpeed;

        // Internal variables for calculations.
        private Vector3 localPosition;
        private Vector3 worldOffset;
        private Vector3 worldPosition;
        private float randomRadius;
        private float randomRadian;
        private float x, y, z;

        private void Start()
        {
            Random.InitState(seed);

            for (int i = 0; i < cubeDensity; i++)
            {
                do
                {
                    randomRadius = Random.Range(innerRadius,
outerRadius);
                    randomRadian = Random.Range(0, (2 *
Mathf.PI));

                    y = Random.Range(-(height / 2), (height /
2));
                    x = (1 + eccentricity) * randomRadius *
Mathf.Cos(randomRadian);
                    z = randomRadius *
Mathf.Sin(randomRadian);
                }
                while (float.IsNaN(z) && float.IsNaN(x));

                float inclination = Random.Range(-
maxInclination, maxInclination);
                Quaternion rotation =
Quaternion.Euler(inclination, 0, 0);
                localPosition = rotation * new Vector3(x, y,
z);
                worldOffset = transform.rotation *
localPosition;
                worldPosition = transform.position +
worldOffset;
```

```
                GameObject selectedPrefab =
cubePrefabs[Random.Range(0, cubePrefabs.Length)];
                GameObject _asteroid =
Instantiate(selectedPrefab, worldPosition,
Quaternion.Euler(Random.Range(0, 360), Random.Range(0, 360),
Random.Range(0, 360)));

                float scale = Random.Range(minScale,
maxScale);
                _asteroid.transform.localScale = new
Vector3(scale, scale, scale);


_asteroid.AddComponent<BeltObject>().SetupBeltObject(Random.Ra
nge(minOrbitSpeed, maxOrbitSpeed),
Random.Range(minRotationSpeed, maxRotationSpeed), gameObject,
rotatingClockwise);
                _asteroid.transform.SetParent(transform);
            }
        }
    }
}
```

BeltVisibilityController.cs

```
using UnityEngine;

namespace solsyssim {

    // Controls the visibility of the asteroid belt based on
the distance from the main camera.

    public class BeltVisibilityController : MonoBehaviour
    {
        public BeltSpawner specificBeltSpawner; // Reference
to the BeltSpawner to control visibility.
        public float thresholdDistance = 100f; // Distance
threshold for toggling visibility.
        private Camera mainCamera; // Reference to the main
camera in the scene.

        private void Start()
        {
            mainCamera = Camera.main; // Initialises the main
camera reference.
        }

        private void Update()
        {
            // Calculate the distance between the camera and
the belt spawner.
```

```
            float distance =
Vector3.Distance(mainCamera.transform.position,
specificBeltSpawner.transform.position);
            // Toggle the visibility based on the distance
exceeding the threshold.
            specificBeltSpawner.gameObject.SetActive(distance
> thresholdDistance);
        }
    }
}
```

CamControl.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using TMPro;


namespace solsyssim {
    //Manages camera control including rotation, translation,
and focus selection within the simulation.
    public class CamControl : MonoBehaviour {

        [SerializeField] private Camera _cam;
        private Transform _camPole;

        private CamAnimator _camAnimator;
        [SerializeField] private AudioManager _audioManager;


        // cam control togle
        private bool _userControl = false;

        // first menu related variables
        private bool _initializeCam = false;

        // audio reference and variables
        [SerializeField] private AudioClip _selectionSound;

        // cam focus variable and event action
        public Transform _selectedBody;
        private TextMeshProUGUI _focus;

        public event Action<Transform> NewFocus;

        const float CamSpeed = 25f;
        const float ZoomSpeed = 2f;
```

```
        const float ZoomMin = -0.1f;
        const float ZoomMax = -800.0f;
        const float RayLength = 1000000f;

        private void Start() {

            _cam = Camera.main;
            _camPole = _cam.transform.parent;
            _focus = GetComponent<TextMeshProUGUI>();

            _camAnimator = GetComponent<CamAnimator>();

            ControlIntentions.Instance.CamRotation +=
RotateCam;
            ControlIntentions.Instance.CamTranslation +=
TranslateCam;
            ControlIntentions.Instance.FocusSelection +=
CheckSelection;

            _selectedBody = transform;
            _selectedBody =
GameObject.Find("Sun").transform.GetChild(0);

            ControlIntentions.Instance.MenuCall += PanCam;

            // Set the first body transform to istelf (null)
to avoid error with cam zoom function.

        }

        // private void OnDestroy() {
        //     ControlIntentions.Instance.CamRotation -=
RotateCam;
        //     ControlIntentions.Instance.CamTranslation -=
TranslateCam;
        //     ControlIntentions.Instance.FocusSelection -=
CheckSelection;
        // }

        // update the cams pole position to the focus body
position
        private void Update() {
            // We always set the cam axis position to the
selected celestial object's position.
            transform.position =
Vector3.Lerp(transform.position, _selectedBody.position,
0.3f);
        }
```

```
        // the animatino when first starting the game, it is
triggerred by an event call
        // we simply unregister the method so it is not called
again
        public void PanCam(bool unused){
            if(_camAnimator != null)
                _camAnimator.NextAnim (new
CamAnimation(_cam.transform, camFinalDepthZPos:-20f,
poleFinalVerticalXRot:25f));

            ControlIntentions.Instance.MenuCall -= PanCam;
        }

        // stop any ongoing animation by simply creating a new
void one
        private void StopAnimation(){
            if (_camAnimator!= null &&
_camAnimator.IsAnimating)
                _camAnimator.NextAnim ();
        }


        // Listen to control event and manages the orbital
rotation of the cam.

        private void RotateCam(string axis, float dir) {
            // Rotation of the cam around the center
horizontally (on the y axis of Axis).
            if (axis == "horizontal")
                transform.Rotate(new Vector3(0, dir * CamSpeed
* Time.deltaTime, 0));

            // Rotation of the cam around the center
vertically (on the x axis of Pole).
            else if (axis == "vertical")
                _camPole.Rotate(new Vector3(dir * CamSpeed *
Time.deltaTime, 0, 0));

            // Warning just to be sure
            else
                Debug.LogWarning("Cam rotation called on
unknown axis: " + axis);

            StopAnimation();
        }
        private void TranslateCam(float translation) {
            // Uses the actual distance from the cam to the
focus body as a factor in how fast the cam is moving onthe
pole.
```

```
            Vector3 pos = _cam.transform.localPosition;
            float Z = pos.z + ZoomSpeed * translation *
_cam.transform.localPosition.magnitude;
            pos.z = Mathf.Clamp(Z, ZoomMax, ZoomMin);
            _cam.transform.localPosition = pos;

            StopAnimation();
        }
        private void CheckSelection(Vector3 selectorPos) {
            Debug.Log("CheckSelection method called."); //
This should log when the method is called
            // Ray is casted onto the UI to see if we catched
one of the icons
            Ray ray = _cam.ScreenPointToRay(selectorPos);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit, RayLength)) {
                if (hit.collider != null) {
                    // Play sound regardless of whether the
selected body has changed
                    _audioManager.PlayPlanetFocusSound();

                    // Check if the hit object is different
from the currently selected body
                    if (_selectedBody !=
hit.collider.transform) {
                        Debug.Log("Hit object: " +
hit.collider.name);
                        _selectedBody =
hit.collider.transform;
                        _audioManager.PlayPlanetFocusSound();
                        NewFocus?.Invoke(_selectedBody); //
Use ?.Invoke to safely invoke the event
                        _focus.text = _selectedBody.name;
                    }
                }
            } else {
                Debug.Log("Raycast did not hit any object.");
// Add this line for debugging
            }
        }
        }

    }
```

CamAnimator.cs

```
using UnityEngine;

namespace solsyssim {
```

```
    // Handles the different camera animation which can be
called from this class' methods.
    public class CamAnimator : MonoBehaviour {

        // reference to the camera
        [SerializeField] private Camera _cam;

        // reference to the current animation
        private CamAnimation _currentAnim;
        public bool IsAnimating { get { return _currentAnim !=
null; } }


        // Camera limit and calculations constants.
        const float CamFinalPos = -20;
        const float CamFinalRot = 25;
        const float LerpIntensity = 0.1f;
        const float LerpEnd = 0.1f;
        const float AnimEnd = 0.1f;

        // We check everyframe if their is an animation on
going and if yes we update it
        private void Update() {
            if(_currentAnim != null) {
                if (_currentAnim.AnimStatus >= AnimEnd) {
                    _currentAnim.Animate ();
                } else {
                    NextAnim (warpToEnd:true);
                }
            }
        }

        // Switches to next animation and either stop the
current one or wrap it to the end
        public void NextAnim(CamAnimation nextAnim = null,
bool warpToEnd = false) {
            if (_currentAnim != null && warpToEnd)
                _currentAnim.WarpToEnd ();

            _currentAnim = nextAnim;
        }
    }


    // Animation object holding the data and logic for the
animation itself.
    // It is created and stored in the currentAnimation of the
CamAnimator script.
```

```
    public class CamAnimation {

        // animation detail variable
        private Vector3 _camFinalDepthZPos;
        private Vector3 _poleFinalXYRot;
        private float _lerpIntensity;

        // reference to the cam that needs to be animated
        private Transform _cam;
        private Transform _pole;

        // status of the animation ending in 0
        public float AnimStatus { private set; get; }


        // constructor changes float angles to Vector3
EulerAngles
        public CamAnimation (Transform cam, float
camFinalDepthZPos = 0f, float poleFinalHorizontalYRot = 0f,
float poleFinalVerticalXRot = 0f, float lerpIntensity = 0.1f)
{
            _cam = cam;
            _pole = _cam.parent;

            if(camFinalDepthZPos == 0f)
                camFinalDepthZPos = _cam.localPosition.z;

            _camFinalDepthZPos = new Vector3 (0, 0,
camFinalDepthZPos);

            if(poleFinalVerticalXRot == 0f)
                poleFinalVerticalXRot =
_pole.localEulerAngles.x;
            if(poleFinalHorizontalYRot == 0f)
                poleFinalHorizontalYRot =
_pole.localEulerAngles.y;

            _poleFinalXYRot = new Vector3
(poleFinalVerticalXRot, poleFinalHorizontalYRot,
_pole.localEulerAngles.z); // rotation on z should not happen,
but we never know

            _lerpIntensity = lerpIntensity;

            AnimStatus = 1f;
        }

        // Animation logic called each fram from the
CamAnimator script.
```

```
        // Lerps the animation to the given parameters.
        // Updates the AnimStatus as the maximum remaining
distance to be covered by a lerp.
        public void Animate() {

            // first we set the cam position on the pole
            Vector3 pos = _cam.localPosition;
            _cam.localPosition = Vector3.Lerp(pos,
_camFinalDepthZPos, _lerpIntensity);

            Vector3 rot = _pole.localEulerAngles;

            // then we set the pole rotation
            // we need to make sure the rotation stay between
0 and 360 for the lerping to work
            if (rot.x > _poleFinalXYRot.x)
                rot.x -= 360;
            float newRotX = Mathf.Lerp(rot.x,
_poleFinalXYRot.x, _lerpIntensity);
            if (newRotX < 0)
                newRotX += 360;

            if (rot.y > _poleFinalXYRot.y)
                rot.y -= 360;
            float newRotY = Mathf.Lerp(rot.y,
_poleFinalXYRot.y, _lerpIntensity);
            if (newRotY < 0)
                newRotY += 360;

            _pole.localEulerAngles = new Vector3(newRotX,
newRotY, 0);


            AnimStatus = Mathf.Max ((_cam.localPosition -
_camFinalDepthZPos).magnitude, (_pole.localEulerAngles -
_poleFinalXYRot).magnitude);
        }

        // Simply warps the animation to the final status and
sets the status to 1.

        public void WarpToEnd () {
            if (_cam != null) {
                _cam.localPosition = _camFinalDepthZPos;
            }

            if (_pole != null) {
                _pole.localEulerAngles = _poleFinalXYRot;
            }
```

```
            AnimStatus = 0f;
        }

    }
}
```

DateCalc.cs

```
using UnityEngine;
using TMPro;

namespace solsyssim {

    // Calculates and displays the current date in the
simulation based on elapsed time.

    public class DateCalc : MonoBehaviour {
        private TMP_Text _dateLabel; // UI element for
displaying the date.
        private int _baseYear = 2000; // The year simulation
starts from.

        private void Start() {
            _dateLabel = GetComponent<TMP_Text>(); //
Initialize the date label.
        }

        void Update() {
            if (!SpaceTime.Instance._timePause) {
                UpdateDateLabel(); // Update the date label if
the game is not paused.
            }
        }


        // Updates the date label with the current date
calculated from the elapsed simulation time.

        private void UpdateDateLabel() {
            double timePool = SpaceTime.Instance.ElapsedTime;
// Total elapsed time in the simulation.

            int year = ComputeYear(ref timePool, _baseYear);
            int month = ComputeMonth(ref timePool,
IsLeapYear(year));
```

```
            int day = Mathf.FloorToInt((float)timePool) + 1;

            // Calculate hours, minutes, and seconds from the
remaining timePool fraction.
            int hours = Mathf.FloorToInt((float)(timePool % 1
* 24));
            int minutes = Mathf.FloorToInt((float)((timePool %
1 * 24 - hours) * 60));
            int seconds = Mathf.FloorToInt((float)((((timePool
% 1 * 24 - hours) * 60) - minutes) * 60));

            // Format the date and time string.
            string dd = day < 10 ? "0" + day.ToString() :
day.ToString();
            string mm = month < 10 ? "0" + month.ToString() :
month.ToString();
            string yyyy = year.ToString();
            string timeString =
$"{hours:D2}:{minutes:D2}:{seconds:D2}";

            _dateLabel.text = $"{dd}/{mm}/{yyyy}
{timeString}"; // Update the UI element.
        }


        // Returns the current date string.

        public string GetDateString() {
            return _dateLabel != null ? _dateLabel.text :
"DateLabel is not set.";
        }


        // Computes the current year based on elapsed time.

        private int ComputeYear(ref double pool, int baseYear)
{
            int year;
            int dayCheck;
            int leapCheck = IsLeapYear(baseYear) ? 1 : 0;

            year = Mathf.FloorToInt((float)(pool /
365.2425f));
            dayCheck = Mathf.FloorToInt((float)(pool - (365 *
year + (year - 1) / 4 - (year - 1) / 100 + (year - 1) / 400 +
leapCheck)));

            if (dayCheck < 0) year -= 1;
```

```
            pool -= year > 0 ? (365 * year + (year - 1) / 4 -
(year - 1) / 100 + (year - 1) / 400 + leapCheck) : 0;

            return year + baseYear;
        }


        // Determines if a given year is a leap year.

        private bool IsLeapYear(int year) {
            return year % 400 == 0 || (year % 100 != 0 && year
% 4 == 0);
        }


        // Computes the current month based on elapsed time
and whether it's a leap year.

        private int ComputeMonth(ref double pool, bool leap) {
            int[] daysToMonthEnd = { 0, 31, 59, 90, 120, 151,
181, 212, 243, 273, 304, 334, 365 };
            int[] daysToMonthEndLeap = { 0, 31, 60, 91, 121,
152, 182, 213, 244, 274, 305, 335, 366 };
            int month;

            for (month = 1; month <= 12; month++) {
                if ((!leap && pool < daysToMonthEnd[month]) ||
(leap && pool < daysToMonthEndLeap[month])) {
                    pool -= leap ? daysToMonthEndLeap[month -
1] : daysToMonthEnd[month - 1];
                    break;
                }
            }

            return month;
        }
    }
}
```

InfoDisplay.cs

```
using UnityEngine;
using UnityEngine.UI;
using solsyssim;
using TMPro;

public class InfosDisplay : MonoBehaviour
{
    // Displays information about the currently focused
celestial body in the UI.
    // UI references
```

```
    public TextMeshProUGUI speedText;

    //public TextMeshProUGUI massText;

    public TextMeshProUGUI radiusText;
    public TextMeshProUGUI focusText;
    public TextMeshProUGUI aphelionText;
    public TextMeshProUGUI perihelionText;
    public TextMeshProUGUI meanAnomalyText;
    public TextMeshProUGUI inclinationText;
    public TextMeshProUGUI eccentricityText;
    public TextMeshProUGUI semiMajorAxisText;
    public TextMeshProUGUI angularVelocityText;

    private CamControl camControl;
    private void Start()
    {
        // Find the CamControl component in the scene
        camControl = FindObjectOfType<CamControl>();
        if (camControl != null)
        {

            // Subscribe to the NewFocus event
            camControl.NewFocus += OnNewBodyFocused;
        }
    }

    private void Update()
    {

        if (camControl != null && camControl._selectedBody !=
null)
        {
            OnNewBodyFocused(camControl._selectedBody);
            float r =
camControl._selectedBody.parent.GetComponent<OrbitalBody>().ra
dius();
        }
        Debug.Log("True Anomaly:" +
camControl._selectedBody.parent.GetComponent<OrbitalBody>().v)
;
    }
    private void OnDestroy()
    {
        // Unsubscribe from the NewFocus event
        if (camControl != null)
        {
            camControl.NewFocus -= OnNewBodyFocused;
        }
```

```
    }

    private void OnNewBodyFocused(Transform focusedBody)
    {
        // Get the OrbitalBody component from the focused body
        Debug.Log("New body focused: " + focusedBody.name);
        OrbitalBody orbitalBody =
camControl._selectedBody.parent.GetComponent<OrbitalBody>();
        if (orbitalBody != null)
        {

            // Update UI elements with the parameters from the
OrbitalBody
            focusText.text = orbitalBody.name;
            //speedText.text = "Speed: " +
orbitalBody.GetSpeed().ToString("F2") + " km/s";
            //massText.text = "Mass: " +
orbitalBody.mass.ToString("F2") + " kg";
            radiusText.text = "Radius: " +
orbitalBody.r.ToString("G5") + " km";
            aphelionText.text = "Arg Ascending: " +
orbitalBody._argAscending + " °";
            perihelionText.text = "Perihelion: " +
orbitalBody._argPerihelion.ToString("F2") + " km";
            meanAnomalyText.text = "Mean Anomaly:" +
orbitalBody._meanAnomaly.ToString("G5");
            Debug.Log("Updating inclination text to: " +
orbitalBody._inclination.ToString("F2"));
            inclinationText.text = "Inclination: " +
orbitalBody._inclination + " °";
            semiMajorAxisText.text = "Semi Major Axis: " +
orbitalBody._semiMajorAxis + "000 km";
            angularVelocityText.text = "Angular Velocity:" +
((orbitalBody._angularVelocity)/(24*60*60)).ToString("G5") +
"rad/s";
        }
        else
        {
            Debug.LogError("OrbitalBody component not found on
focused object.");
        }



    }
}
```

OptionsMenu.cs

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.EventSystems;
using UnityEngine.InputSystem;
using UnityEngine.UI;

namespace solsyssim{

    public class OptionsMenu : MonoBehaviour
    {


        // References to sliders
        [SerializeField] private Slider _masterVolumeSlider;
        [SerializeField] private Slider _musicVolumeSlider;
        [SerializeField] private Slider _sfxVolumeSlider;

        private void Start(){
            // Load saved volume settings or use default
values
            _masterVolumeSlider.value =
PlayerPrefs.GetFloat("MasterVolume",
AudioManager.Instance.GetMasterVolume());
            _musicVolumeSlider.value =
PlayerPrefs.GetFloat("MusicVolume",
AudioManager.Instance.GetMusicVolume());
            _sfxVolumeSlider.value =
PlayerPrefs.GetFloat("SFXVolume",
AudioManager.Instance.GetSFXVolume());

            // Initialises sliders and adds listeners

_masterVolumeSlider.onValueChanged.AddListener(SetMasterVolume
);

_musicVolumeSlider.onValueChanged.AddListener(SetMusicVolume);

_sfxVolumeSlider.onValueChanged.AddListener(SetSFXVolume);
        }


        public void PlaySolarSystem(){
        SceneManager.LoadScene("StellarSystem");
        }
        public void QuitGame(){
        Application.Quit();
        }
        public void PlayMainMenu(){
```

```
        SceneManager.LoadScene("MainMenu");
        }
        public void SetMasterVolume(float volume) {
            AudioManager.Instance.SetMasterVolume(volume);
            PlayerPrefs.SetFloat("MasterVolume", volume);
        }

        public void SetMusicVolume(float volume) {
            AudioManager.Instance.SetMusicVolume(volume);
            PlayerPrefs.SetFloat("MusicVolume", volume);
        }

        public void SetSFXVolume(float volume) {
            AudioManager.Instance.SetSFXVolume(volume);
            PlayerPrefs.SetFloat("SFXVolume", volume);
        }
    }
}
```

GraphicsTab.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class GraphicsTab : MonoBehaviour
{
    public TMP_Dropdown resolutionDropdown;
    public TMP_Dropdown fpsDropdown;
    public TMP_Dropdown antiAliasingDropdown;
    Resolution[] resolutions;

    void Start()
    {
        InitializeResolutionDropdown();
        InitializeFPSDropdown();
        InitialiseAntiAliasingDropdown(); // Initialise anti-
aliasing dropdown
        LoadGraphicsSettings();
    }
//Retrieves all available resolutions for system and displays
them in dripdown
    void InitializeResolutionDropdown()
    {
        resolutions = Screen.resolutions;
        resolutionDropdown.ClearOptions();
        List<string> options = new List<string>();
        for (int i = 0; i < resolutions.Length; i++)
```

```
        {
            string option = resolutions[i].width + " x " +
resolutions[i].height;
            options.Add(option);
        }
        resolutionDropdown.AddOptions(options);
    }
// Retrieves all available FPS options and adds it to the
dropdown
    void InitializeFPSDropdown()
    {
        List<int> fpsOptions = new List<int> { 30, 60, 120,
144, 240 };
        fpsDropdown.ClearOptions();
        List<string> options = new List<string>();
        foreach (int fps in fpsOptions)
        {
            options.Add(fps.ToString() + " FPS");
        }
        fpsDropdown.AddOptions(options);
    }
// aadds the antialiasing options to the dropwdown
    void InitialiseAntiAliasingDropdown()
    {
        List<int> aaOptions = new List<int> { 0, 2, 4, 8 };
        antiAliasingDropdown.ClearOptions();
        List<string> options = new List<string>();
        foreach (int aa in aaOptions)
        {
            string option = aa > 0 ? aa.ToString() + "x" :
"Off";
            options.Add(option);
        }
        antiAliasingDropdown.AddOptions(options);
    }
//changes the resolution based on what the dopdown has been
changed to
    public void SetResolution(int resolutionIndex)
    {
        Resolution resolution = resolutions[resolutionIndex];
        Screen.SetResolution(resolution.width,
resolution.height, Screen.fullScreen);
        PlayerPrefs.SetInt("ResolutionIndex",
resolutionIndex);
    }
//toggles fullscreen on/off based on the dropdown
    public void SetFullScreen(bool isFullScreen)
    {
        Screen.fullScreen = isFullScreen;
```

```
        PlayerPrefs.SetInt("FullScreen", isFullScreen ? 1 :
0);
    }
//sets the quality based on the one chosen in the dropdown
    public void SetQuality(int qualityIndex)
    {
        QualitySettings.SetQualityLevel(qualityIndex);
        PlayerPrefs.SetInt("QualityIndex", qualityIndex);
    }
//changes the fps based on the one chosen from the dropdown
    public void SetFPS(int fpsIndex)
    {
        Application.targetFrameRate = fpsIndex;
        PlayerPrefs.SetInt("FPSIndex", fpsIndex);
    }
//sets the antialiasing based on one chosen from dropwdown
    public void SetAntiAliasing(int aaIndex)
    {
        QualitySettings.antiAliasing = aaIndex;
        PlayerPrefs.SetInt("AAIndex", aaIndex);
    }
//loads the graphics settings into the unity engine
    void LoadGraphicsSettings()
    {
        resolutionDropdown.value =
PlayerPrefs.GetInt("ResolutionIndex", 0);
        resolutionDropdown.RefreshShownValue();

        Screen.fullScreen = PlayerPrefs.GetInt("FullScreen",
1) == 1;

        int qualityIndex = PlayerPrefs.GetInt("QualityIndex",
2);
        QualitySettings.SetQualityLevel(qualityIndex);

        fpsDropdown.value = PlayerPrefs.GetInt("FPSIndex", 1);
        fpsDropdown.RefreshShownValue();

        antiAliasingDropdown.value =
PlayerPrefs.GetInt("AAIndex", 0);
        antiAliasingDropdown.RefreshShownValue();
    }
}
```