Procedural Planetary System COMP250 - Unity

Callum Metcalfe

Problem

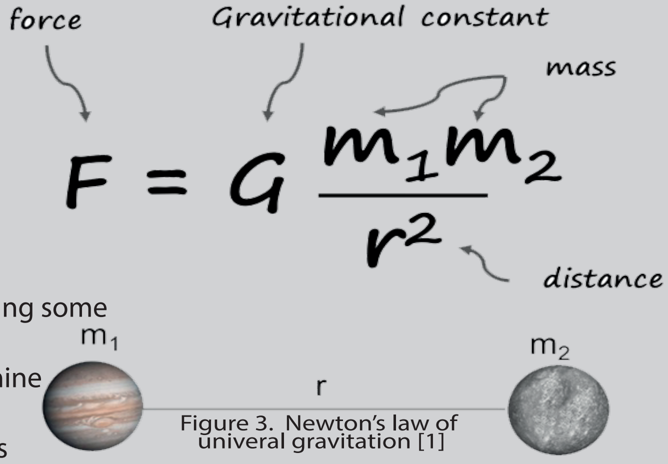
For my project my goal was to make a procedural planetary system. In which I want to use actual physics to simulate the orbits of the planets. Which is usually faked in other systems by generating a orbit and having the planet stick to it making it impossible to disturb the orbit. In addition to this I also want to generate the planets during run time.

Solution

This project is broken up into 2 separate areas the first being the generation of the system its self and the other being the gravity simulation.

To begin work on the project I first looked into some formulas to with the first one being shown in figure 3.

This formula is the most important as it's the one that determine the forces acting on the planets.



$$F = G\frac{m_1 m_2}{r^2}$$

Figure 3. Newton's law of univeral gravitation [1]

I then looked into finding some other formulas that would help me determine a starting velocity and the mass of the planets using its radius and surface gravity.

For the star in the system I will be using our sun as reference and making its mass = 1 solar mass which will be the measurement I use for the star I generate. This value can be used to find the habitability zone where life can be sustained. It can also be used in finding the frost line of which once this point is crossed gas giants form rather than rocky planets.

For the planets I need will only need to generate the planets radius and surface gravity. Whilst for the planets distance my plan is to have a planet generated around the habitability zone as well as the frost line. For the system the first planet generated would be around the frost limit

Planet	Distance from the sun (AU)	distance between previous planet (AU)	distance multiplier
Mercury	0.39	n/a	n/a
Venus	0.72	0.33	1.85
Earth	1.00	0.28	1.39
Mars	1.52	0.52	1.52
Ceres	2.77	1.25	1.82
Jupitor	5.20	2.43	1.88
Saturn	9.55	4.38	1.84
Uranus	19.22	9.67	2.01
Neptune	30.11	10.89	1.57

Figure 4: Planet distance based on multiplier

Based on the data I have presented in figure 4 it shows that the distance between planets varies on a range between 1.39 to 2.01. These values will be used to calculate the positions in which a stable orbit can be achieved by using the planet that was generated at the frost line by multiplying or dividing the planets position by the distance multiplier.

Conclusion

In terms of what has gone well so far I was able to generate 2 successful orbits working which both had moons to accompany them. Whilst for the project so far an area I wish I focused more on the generation of the planetary system earlier on as it is something that slipped my mind whilst working on getting the gravity simulation working. Another area that I will be changing is the fact that I changed the gravitational constant to allow me to have a scaled down simulation. In doing this I have not made any changes to other values to accompany this change and may resort to having a 1:1 scale rather than changing all other values reflect this change .

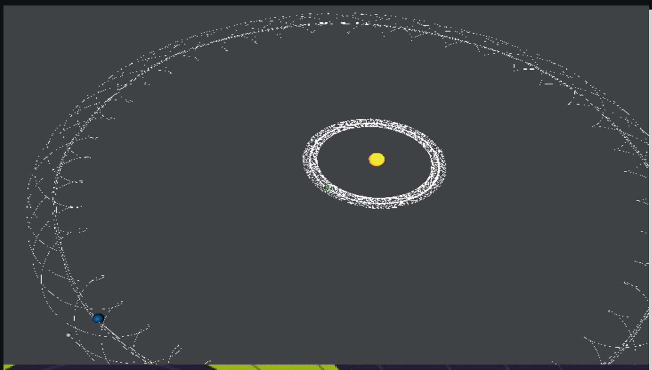


Figure 1. Current implantation of my artefact

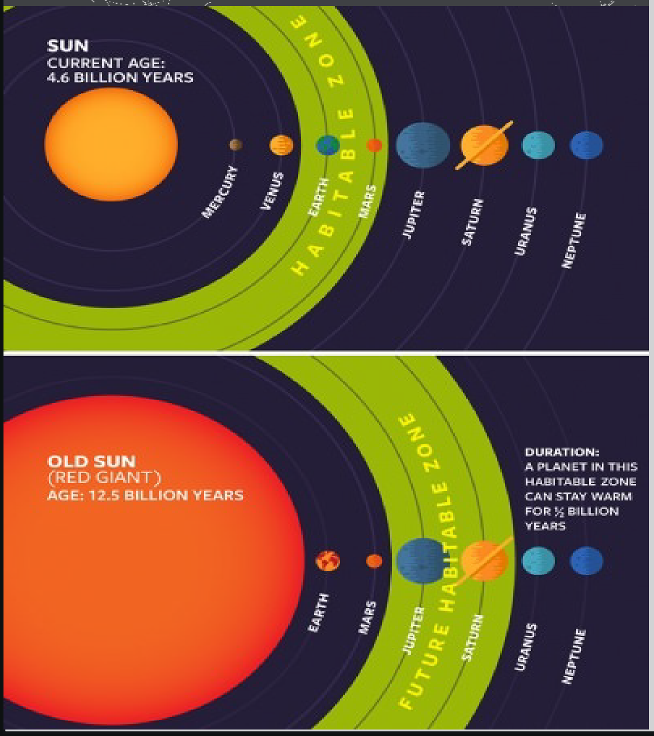
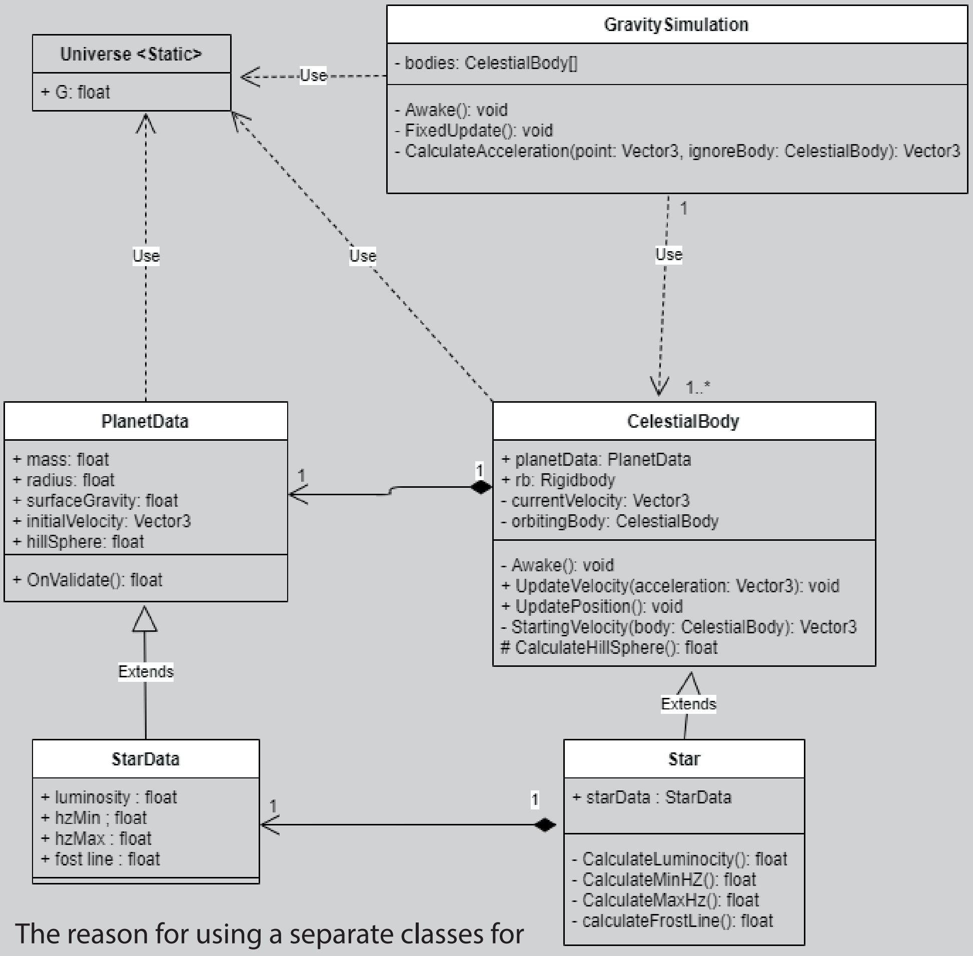


Figure 2. Shows the range in which the habitability zone lies and how it changes over billions of years [2].



Universe <Static>
+ G: float
GravitySimulation
- bodies: CelestialBody[]
- Awake(): void
- FixedUpdate(): void
- CalculateAcceleration(point: Vector3, ignoreBody: CelestialBody): Vector3
Use
Use
Use
Use
1
1..*
PlanetData
+ mass: float
+ radius: float
+ surfaceGravity: float
+ initialVelocity: Vector3
+ hillSphere: float
+ OnValidate(): float
1
1
CelestialBody
+ planetData: PlanetData
+ rb: Rigidbody
- currentVelocity: Vector3
- orbitingBody: CelestialBody
- Awake(): void
+ UpdateVelocity(acceleration: Vector3): void
+ UpdatePosition(): void
- StartingVelocity(body: CelestialBody): Vector3
# CalculateHillSphere(): float
Extends
Extends
StarData
+ luminosity : float
+ hzMin ; float
+ hzMax : float
+ fost line : float
1
1
Star
+ starData : StarData
- CalculateLuminocity(): float
- CalculateMinHZ(): float
- CalculateMaxHz(): float
- calculateFrostLine(): float

The reason for using a separate classes for the data of the planets/stars is that it means I can keep scriptable objects of the planets. Whilst also allowing me to add a custom planet into the generation of the planetary system. I have also separated the GravitySimulation script to prevent each body from finding the other bodies and instead have just the 1 class do the finding as this would help with the performance whilst simulating gravity. I have also used Inheritance to make it easier on changing how the Star is generated as it still uses some of the same variables and functions.

Sources

[1]"F=Gmm/r2 - EWT", EWT, 2021. [Online]. Available: https://energywavetheory.com/equations/fgmmr2/. [Accessed: 15- Mar- 2021].

[2]"Frost Line Archives - Universe Today", Universe Today, 2021. [Online]. Available: https://www.universetoday.com/tag/frost-line/. [Accessed: 18- Mar- 2021].