Designing, Building, and Testing a Physics Simulator with application to the Solar system, programmed in

# C++ using Unreal Engine 4, with a critical comparison of various numerical solvers


*Student*

Dean Sisman

*Supervisor*

Dr Colin Price


*Towards the Degree of*

BSc (Hons) Computing


University of Worcester


April 2021

## Abstract

This is a Design, Build & Test (DBT) project exploring the use of Unreal Engine 4 as a platform for developing a physics simulator. This has been applied specifically to the development of a gravitational simulation of the Solar system. This dissertation will provide a detailed documentation of the design and development process, and then a critical comparison of Euler, Euler-Cromer, Runge-Kutta, and Runge-Kutta-Fehlberg numerical solvers, all applied and tested within the developed artefact. The intended aim is to allow physics simulators to be more effectively built within Unreal Engine 4 by using this work as a basis. Key insights are made regarding the effectiveness of various solvers in accurately simulating Newtonian physics, the effects of timestep changes with these solvers, and the overall effectiveness of the simulation. This leads to conclusions being made on the most effective solver to use and the effectiveness of Unreal Engine 4 as a platform for simulation.



## Preface & Acknowledgements

I would like to thank Colin Price for his guidance while supervising this project. He has been very engaged and proactive from the start; taking time to give detailed answers to my questions and helping to shape the project from a series of rough ideas to the finished result.

# Contents

# Introduction

## Research Questions

There are three main questions that this project aims to provide answers to:

- Is Unreal Engine 4 an appropriate platform for building a physics simulator?
- Which numerical solver provides the most accuracy?
- What are the effects of changes in timestep on accuracy?

## Context

In recent years, there has been an explosion in interest in space exploration as private, free enterprises have drastically reduced the cost of orbital launches. For example, when NASA performed launches using its Space Shuttle, it cost $54,500 per kilogram to put payloads into Low Earth Orbit (LEO). Today, SpaceX offers the same service at a cost of $2,720 per kilogram (Jones, 2018).

Such drastic advances have opened new markets as smaller companies are able to afford launches for their payloads. Human spaceflight has also gained political momentum, with new missions being proposed to send humans to Mars for the first time (NASA, 2015).

As smaller, more cost-conscious businesses and individuals look to embark on orbital missions (Catapult Satellite Applications, 2019), this creates interest in cost effective simulations to visualise the systems during the early exploration stage of development. Videogame engines offer a low-cost platform for such development.

Within the University of Worcester, physics simulators have been developed to model a range of real-world applications, such as car suspension (Price, 2021). However, these simulations are built within the aging Unreal Development Kit (UDK).

This project will explore the development of a physics simulator within the more modern Unreal Engine 4 (UE4). This engine improves upon the UDK by allowing actors to be coded in C++ (Epic, 2021). This allows for highly capable numerical solvers such as Runge-Kutta-Fehlberg to be directly applied (RKF45, 2020).

The simulator will be applied to orbital simulation by creating a Newtonian simulation of the solar system. By doing this, the effectiveness of the platform can be tested within an area of historically high demand. By testing a variety of numerical solvers, a recommendation can be made for the most effective solution.

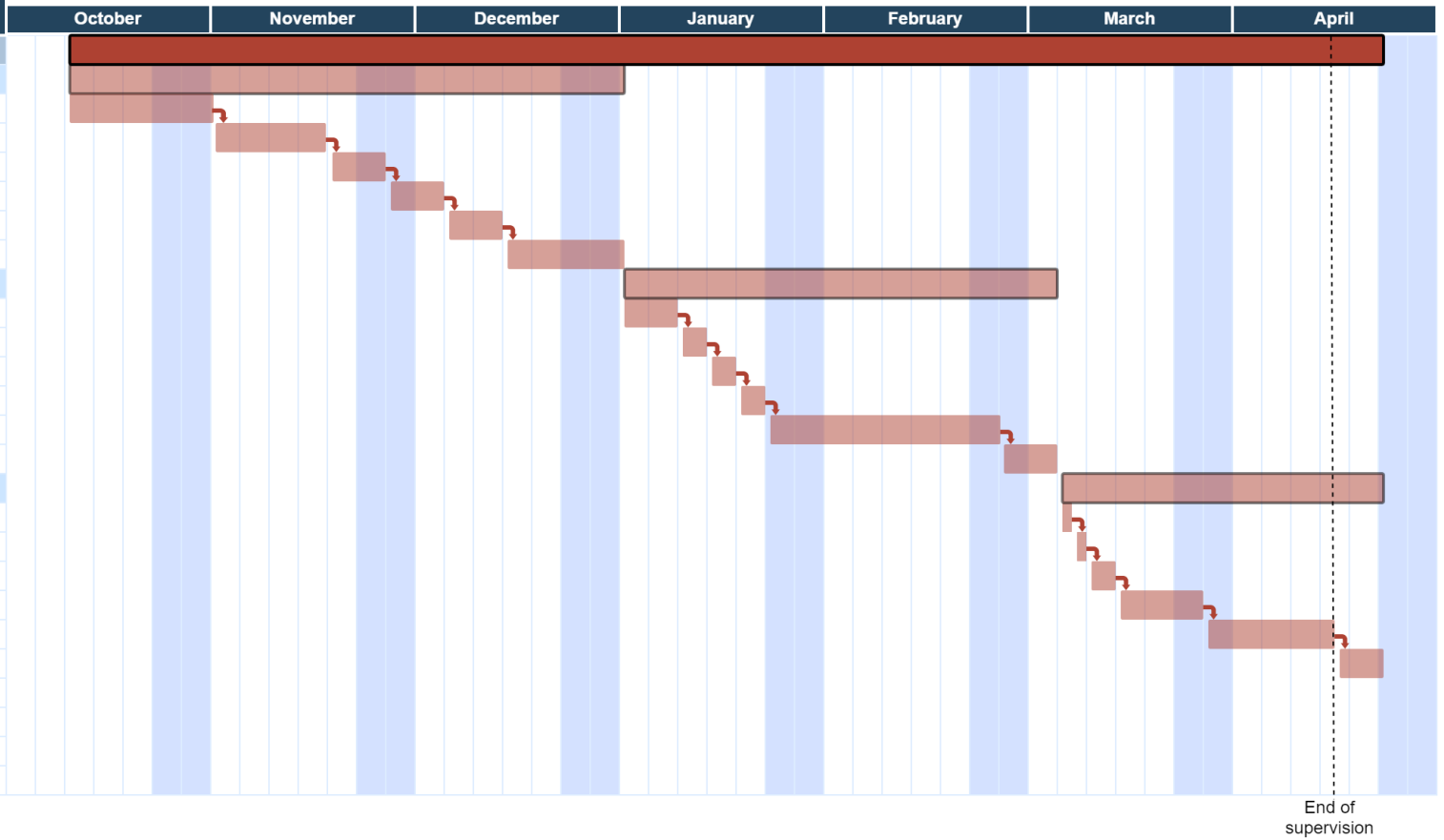
## Contribution

This project aims to contribute to the broader context of Computing by providing an in-depth exploration into building a physics simulator in UE4. Within the University of Worcester, there are pre-existing simulators built within the UDK. By using the findings and explanations from this work, it should be made possible to rebuild these within the more modern UE4.

# Project Gantt Chart

This is the Gantt chart for the project. It is not intended as an exact plan of time allocation, but more as a rough guide on how long various stages are expected to take. It is expected that many tasks will take longer or shorter than expected, and as the project evolves, remaining time will be flexibly allocated to remaining tasks. Due to likely hurdles, such as covid restrictions or technical difficulties, maintaining a flexible attitude to time management will be vital to the project succeeding.

| | Task Name |
|---|---|
| 1 | Complete project execution |
| 2 | Design |
| 3 | Research Unreal Engine 4 development |
| 4 | Research numerical solvers |
| 5 | Research orbital mechanics |
| 6 | Design verification method |
| 7 | Design Validation method |
| 8 | Design simulator |
| 9 | Build |
| 10 | Build tool for fetching real-word data |
| 11 | Start new Unreal project with C++ actor |
| 12 | Develop mechanism for importing data into simulator |
| 13 | Develop mechanism for moving actors with simulation data |
| 14 | Implement solvers |
| 15 | Develop mechanism for generating Octave files at the end |
| 16 | Test |
| 17 | Perform verification |
| 18 | Perform validation |
| 19 | Detail results in dissertation |
| 20 | Discuss results in dissertation |
| 21 | Write rest of dissertation |
| 22 | Refine dissertation and submit |
| 23 | |
| 24 | |
| 25 | |
| 26 | |



# Review of Literature

This literature review aims to analyse writings and works relevant to this project. The topics that will be explored are notable similar projects, the Newtonian model, numerical solvers, and scaling. The literature discussed in this review will be used as a basis during requirements specification and artefact design and development. When testing has been performed, results will be weighed against the principals, concepts and research provided by the reviewed literature.

## Notable similar projects

These are some notable similar projects which will be analysed for functionality. This will aid in building a list of key functionalities which will form the projects requirements specification.

### *UDK Simulations – University of Worcester*

Within the University of Worcester, several physics simulators have been developed within UDK (Price, 2021). These operate in a very similar fashion to how this project is intended to perform; by utilizing a variety of numerical solvers to simulate systems of Ordinary Differential Equations (ODEs). They also have the functionality to automatically generate log files post-runtime which display important data, such as error percentages and velocities. This automates a large part of the verification and validation process.

### *Solar System Scope*

Solar System Scope is an online simulation of the Solar system showing the real positions of the planets and other major bodies in the solar system. The calculations for these are based on orbital parameters from NASA (Solar System Scope, 2021). The bodies are scaled to be much larger than in real life, so they can be clearly seen. This aids in both clarity and visual appeal. Tracing lines are rendered to show the orbital paths of the bodies. No detailed information is given regarding solvers or the programming language used.

### *The Sky Live*

The Sky Live is another online simulation of the Solar System that displays real positions for planets and major bodies in the solar system (The Sky Live, 2021). It also provides detailed information on the planets through its interface, which leads to other pages. The planets are represented with simple images rather than complex models, with tracing lines showing orbital paths.

### *Findings from similar projects*

After reviewing these projects, several key functionalities have been gathered which will aid in requirements specification. There is the ability to solve ODEs with a variety of solvers, as is the case with the UDK simulations. This will help answer the question for the most effective solver. Additionally, the functionality to automatically generate Octave files post-runtime will be of great use for verification and validation. The UI allowing the user to change parameters without having to edit the code makes testing far easier.

The ability to import real world data as shown in both online simulations will be vital for providing accurate positioning and performing validation. Their representations of the solar system, with clear representations of the planets and their orbital paths, will be important for the functionality and visual appeal for this project.

These features will be used in the requirements specification for this simulation, as they have been found to be of high value.

## The Newtonian Model

The gravitational model will be built on Newton's law of gravitation (Fleury, 2019):

$$F = \frac{GMm}{r^2}$$

Which will then be applied to accelerate each body with the equation:

$$a = \frac{F}{m}$$

However, when combined the mass of the body being accelerated ($m$) can be cancelled out leaving the single equation:

$$a = \frac{GM}{r^2}$$

This can be reduced even further, as $G$ (the gravitational constant) and $M$ (the mass of the body that is being accelerated towards) are both constant, so they can be instead stored as a singular constant for each body:

$$a = \frac{otherBodyGM}{r^2} * direction$$

At this point the simulation is simply having to apply the inverse square law to a stored constant according to $r$ (the distance), then applying a direction to the resultant acceleration. When performed every cycle this will save a significant amount of computing compared to the original equation.

However, there are recognized weaknesses to the Newtonian model. In contemporary physics, it is believed to be a very good approximation tool for planetary positions, but with several inaccuracies that could affect the simulation. For example, planets close to the sun, especially Mercury experience perihelion advance that is inaccurately portrayed in the Newtonian model (Malek, 2016). Only after validation will it be seen whether these inaccuracies have a considerable effect on the simulation.

## Numerical solvers

### *Euler and Euler-Cromer*

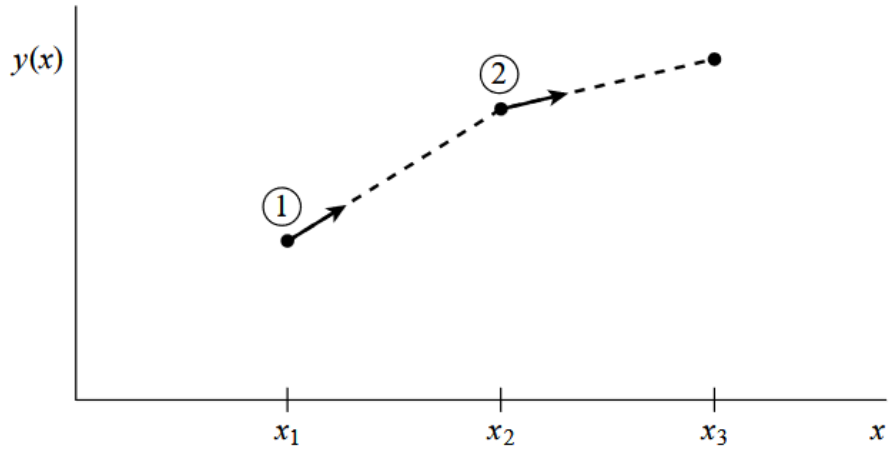



*Figure 1 A visualisation of Euler's method (Press, et al., 1992)*

Both Euler and Euler-Cromer numerical solvers produce an approximation for the next value in an ODE by taking the simplistic approach of applying the full timestep to the function, and giving the result as the approximation (Ydri, et al., 2013):

$$y_{n+1} \cong y_n + \Delta t\, f(x_n, y_n)$$

In practice for the gravitational simulation, it will calculate the change in velocity by calculating accelerations according to current positions and applying them multiplied by the timestep. The difference between Euler and Euler-Cromer is that the Euler solver will update position first before calculating the new velocity, while Euler-Cromer will calculate the new velocity first before updating position.

### *Runge-Kutta and Runge-Kutta-Fehlberg (RKF)*

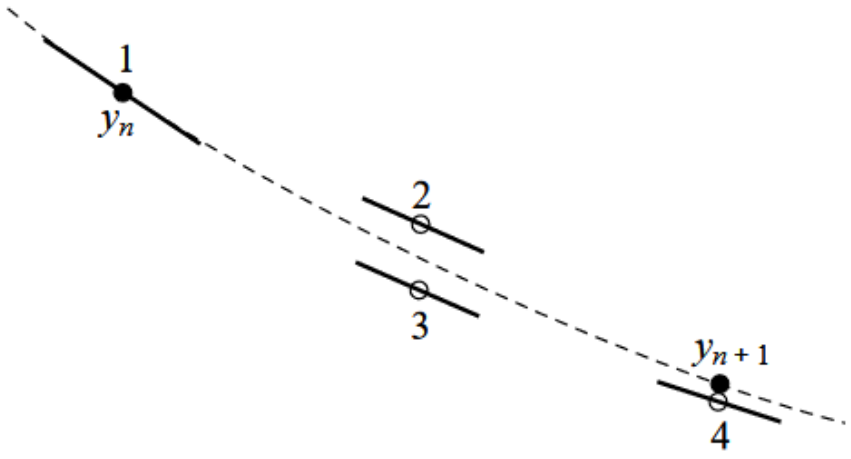



*Figure 2 A visualisation of the Runge-Kutta method (Press, et al., 1992)*

"For many scientific users, fourth-order Runge-Kutta is not just the first word on ODE integrators, but the last word as well." (Press, et al., 1992)

The Runge-Kutta method is a significantly more complex method of integrating ODEs, with the aim of producing far more accurate results. The solver is fourth-order accurate, meaning that a reduction x in timestep results in an increase $x^4$ in accuracy. For example, a halving of timestep would result in a $2^4 = 16$ times improvement.

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\
k_3 &= hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\
k_4 &= hf(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)
\end{aligned}
$$

*Figure 3 The Runge-Kutta method (Press, et al., 1992)*

Above is the formula. Instead of performing a single evaluation and using that as the approximation, as is the case with Euler's method, the fourth order Runge-Kutta method performs four evaluations then gives a weighted average as the approximation. In the case of the gravitational simulation, it would perform this for both position and velocity, performing calculations for $K_{xv}$ and $K_{xr}$ intermittently, as they are dependent on each other's values from the previous evaluation.

As the solver is of fourth-order accuracy, this greatly increases the possible returns from managing timestep in an efficient manner. RKF achieves this by dynamically changing the step-size to maximise accuracy. According to the literature reviewed, it should be proven as the most effective during testing. "We consider adaptive stepsize control, discussed in the next section, an essential for serious computing." (Press, et al., 1992)

## Scaling the system

### *The need for scaling*

As this simulation aims to be as accurate as possible, scaling becomes a very important element of ensuring that accuracy is maximised. One factor which could lead to a loss of accuracy is the very large and small numbers involved when looking at the real-world data for things such as masses, the gravitational constant, accelerations, and distances. As even double precision floats only have limited precision (Sahu & Rao, 2017), this can lead to rounding errors which can compound. This creates a need to scale the various values involved in the system to be much more reasonable sizes, ideally 0-100. The process for performing the ODE scaling has been performed referring to the instructions given in Scaling of Differential Equations (Langtangen & Pedersen, 2016).

### *Scaling distances*

Fortunately, a good coefficient with which to scale distances is already commonly used; Astronomical Units (AU). By measuring distances in AU within the simulation, they will be brought into the desired size range due to the extremely large size of a single AU (Huang, et al., 1994).

### *Scaling GM*

The next parameter to be scaled is each body's standard gravitational parameter (GM). Usually these are extremely high, especially the Sun's due to the very high masses involved. To reduce them, the mass of the Sun will be used as the scaling coefficient, and the gravitational constant will be removed as a factor. This of course makes the Sun's GM exactly 1, and all the planets GM's their mass over the Sun's. This still makes for very small numbers, but much less extreme in scale than originally.

### *Scaling Velocities*

Now that distances and GMs have been scaled, the task of scaling velocities becomes that of finding the value which works within the scaled system to produce correct orbits. The formula which achieves this is as follows:

$$v_c = \sqrt{\frac{GM_{Sun}}{AU}}$$

The velocities are divided by the scaling factors for the GMs, which of course was dividing by G (by simply not including the gravitational constant originally) and the mass of the Sun. These are divided by the distance scaling, in this case AU.

### *Scaling Time*

Interestingly, time has not been explicitly scaled. However, in scaling velocities it has been implicitly scaled as the planets now take a significantly shorter amount of time to perform their orbits. And so, to perform validation, time must be rescaled so comparisons at various points in time can be made with real-world positions. As with velocities, the time scaling coefficient is found by factoring in the scaling coefficients for the other parameters. It becomes:

$$t_c = \sqrt{\frac{GM_{Sun}}{AU^3}}$$

Successfully scaling the system will be vital to ensuring it can retain accuracy at its scaled values. If done incorrectly, it will lead to results drastically different to the real-world data when rescaled back to real scale. This means that it will be a key requirement for the system, and during validation it will be a factor in the overall accuracy of the system.

# Methodology & Sources of Data

Building on the concepts discussed in the introduction and review of literature, the Aims, Objectives, and Requirements for the project have been developed to ensure that the main research questions and contributions are answered, and that the project is effective in the context of current works. This section will also discuss the previously mentioned ideas of verification and validation, and design methodologies to achieve both. Finally, the sources for real-world data will be documented.

## Aims

- To create a physics simulator in UE4 with a focus on numerical solution of systems of Ordinary Differential Equations (ODEs).
- To apply this simulator to the Solar system.
- To implement a range of numerical solvers and critically compare their accuracy.
- To provide documentation of the simulator creation process.

## Objectives

- Review literature on the Newtonian model, solvers, scaling and notable similar projects.
- Design scaling formulas to effectively scale the solar system.
- Design appropriate verification method for deciding on the most effective solver.
- Design a method of validation comparing simulator results to real-world data.
- Create a new UE4 project with C++ code attached to engine actors.
- Develop method for calculating acceleration given a body and its position (Newtonian).
- Develop Euler and Euler-Cromer solvers.
- Develop Runge-Kutta solver.
- Implement Runge-Kutta-Fehlberg (RKF45) solver.
- Implement verification process.
- Run verification process and document results.
- Develop tool for gathering real-world data.
- Develop mechanism for importing real-world data into simulation.
- Develop validation process.
- Run validation process and document results.
- Discuss the results relating to theory.
- Document the development process.

## Functional Requirements

- Ability to simulate systems of ODEs with a variety of solvers.
- Ability to import real-world data for positions and velocities.
- Ability to scale the system before runtime and restore to real scale post-runtime for validation.
- Ability to automatically generate Octave files plotting results post-runtime.

- Ability to display simulation in 3D using Unreal static meshes.
- Ability to adjust key parameters outside of the source code, such as timestep.

## Non-Functional Requirements

- Run at an acceptably high framerate.
- Display the simulation so that the motion of the planets can clearly be seen.

## Verification

Before comparisons to real world data can be performed, it must first be proven that the applied numerical solvers are providing an accurate representation of the Newtonian model. To do this, they must each be verified. Verification is the process of testing a software implementation of a mathematical model to ensure that it is providing an accurate representation of the model (Sargent, 2013). In doing this, error percentages are produced which show how far the simulation results differ from the correct answer according to the model. These error percentages will be used to check whether the solvers are able to accurately represent the Newtonian model, and to compare the various solvers for accuracy.

### *A Circular Orbit*

As stated in the verification definition, the process will compare simulation results to perfect answers according to the model. To do this, those perfect answers must first be calculated. A perfectly circular orbit will always yield a constant correct answer for the radius of its orbit. The formula for circular orbits is: (Kluever, 2003)

$$v = \sqrt{\frac{\mu}{r}}$$

Where "v" is the velocity, "μ" is the standard gravitational parameter (GM) of the centre mass, and "r" is the distance. This means that a perfectly circular orbit can be achieved by setting all these parameters to 1:

$$1 = \sqrt{\frac{1}{1}}$$

And so, a constant perfect answer has been found: with a speed of 1, a distance of 1 and the centre mass having a GM of 1, the distance should remain constant at 1 as the satellite orbits. Deviations in distance from exactly 1 can be used to calculate the error percentage with the formula:

$$Error\ \% = (distance - 1) * 100$$

For example, if at a certain time in the orbit the distance is 1.1, the error is 10% at that time. This process will be performed multiple times on each solver over a fixed timespan, testing its accuracy with different timesteps. This will produce graphs of error percentages over time which can be used for critical comparison of the solvers and the effects changes in timestep have on accuracy.

## Validation

Once verification has been performed, the solver proven to have the highest accuracy will be selected to undergo the process of validation. This will test the simulation by comparing the simulated results to real-world ones, producing error percentages which can be used to judge the overall effectiveness of the simulation (Sargent, 2013).

### *Real Solar System*

In the case of this simulation, the real-world data for comparison will be the positions of the planets at certain times over a timespan. The simulation will be initialized using the real planet's positions and velocities at the 1st January 2000, then it will be ran to simulate 20 years of orbits. The simulated positions will be compared to the real-world positions on the 1st January each year, up to the 1st January 2020. In comparison, the error percentage can be calculated for each planet with the formula:

$$Error\ \% = \left(\frac{distance\ between\ real\ planet\ position\ and\ simulated\ position}{distance\ between\ real\ planet\ and\ sun}\right) * 100$$

For example, the following planet has a distance of 1 between the simulated position and real position, and the real planet is a distance of 10 from the sun. At this time, the simulated planet has an error of 10%.

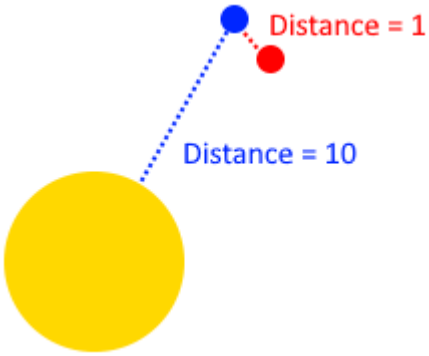



*Figure 4 An example validation error*

Once error percentages for each of the planets has been calculated, an average validation errors for the whole system can be calculated by summing each of the planet's errors then dividing by the number of planets. This will produce the overall validation errors to help judge the overall effectiveness of the simulation. As these errors are all calculated once a year over a 20-year timeframe, the results will be presented in graphs of error percentages over time.

## Obtaining real-world information

### *Positions and Velocities – NASA HORIZONS*

For the validation process to be performed with comparisons to real-world data, the real-world positions of the planets need to be found at the specified dates. Fortunately, NASA's HORIZONS database contains this information and can be freely queried (NASA, 2021). Implementation details for these queries will be documented in the Design & Development section.

### *Other planetary information*

In addition to positioning data, information on planetary masses and other important parameters needs to be gathered. NASA offers these with high precision in its Planetary Fact Sheets (NASA, 2019). The collected values are shown in the table below. Values gathered from other sources are individually referenced.

| Description | Value |
|---|---|
| Mercury Mass ($10^{24}$ kg) | 0.33011 |
| Venus Mass ($10^{24}$ kg) | 4.8675 |
| Earth Mass ($10^{24}$ kg) | 5.9724 |
| Mars Mass ($10^{24}$ kg) | 0.64171 |
| Jupiter Mass ($10^{24}$ kg) | 1,898.19 |
| Saturn Mass ($10^{24}$ kg) | 568.34 |
| Uranus Mass ($10^{24}$ kg) | 86.813 |
| Neptune Mass ($10^{24}$ kg) | 102.413 |
| Solar Mass ($10^{30}$ kg) | 1.9891 (Briggs & Carlisle, 2016) |
| Astronomical Unit (meters) | 149597870700 (Huang, et al., 1994) |
| Gravitational Constant | $6.67384 \times 10^{-11}$ (Milyukov & Fan, 2012) |

# Design & Development

This section will document the process of designing and developing the simulator to meet project requirements. It will start with a brief instructional for creating the new Unreal project and importing UDK assets. This will contribute to achieving the project goal of providing documentation of the simulator creation process. As it is such a significant element of this project and its intended goals, it will form a large part of the dissertation.

## Creating Unreal Projects & File Structure

### *Creating new project*

This is the process for creating the new Unreal project. Of note, the Architecture category is chosen with a blank template. The starter content is disabled, as this drastically increases file size with needless resources.

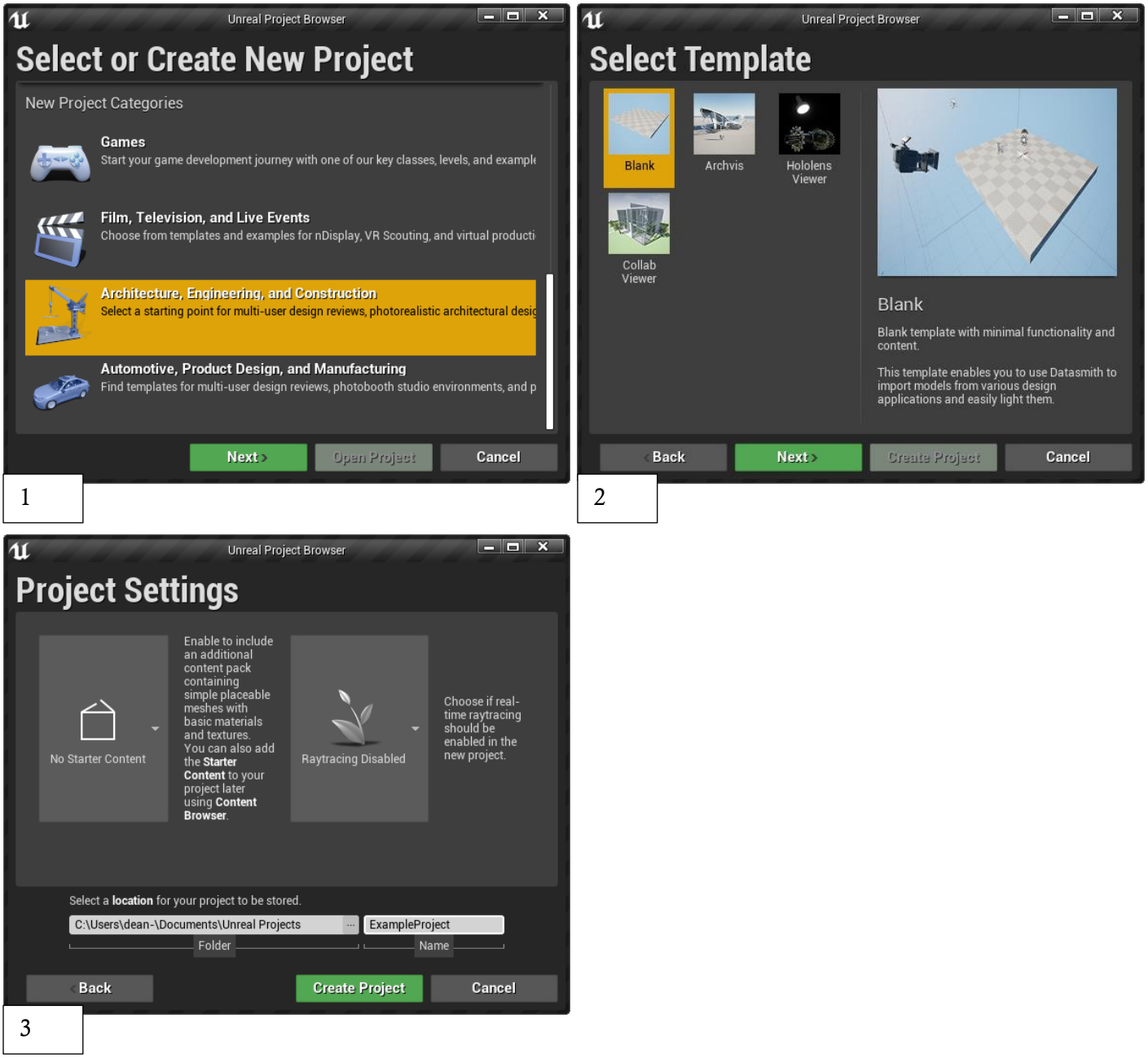



*Figure 5 The process for creating the Unreal project.*

*Creating new C++ class*

This is the process for adding the C++ class. Note that Actor should be chosen as the parent class, as this allows the class to be placed in the scene and given components such as static meshes so it can be visually represented in the simulation. Once created, the generated C++ files are in the project's Source/[Name] folder, demonstrated below in the generated files for MyActor.

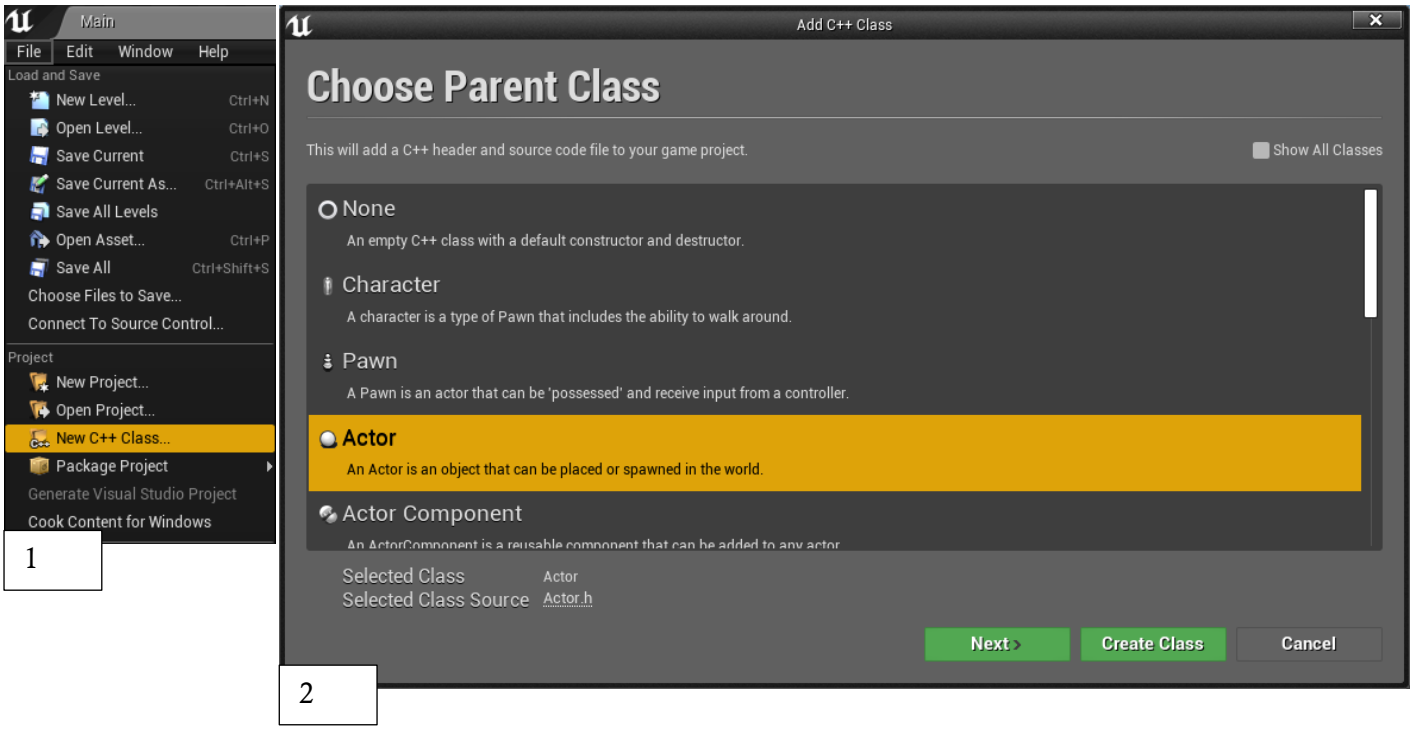



*Figure 6 Adding a new C++ class.*

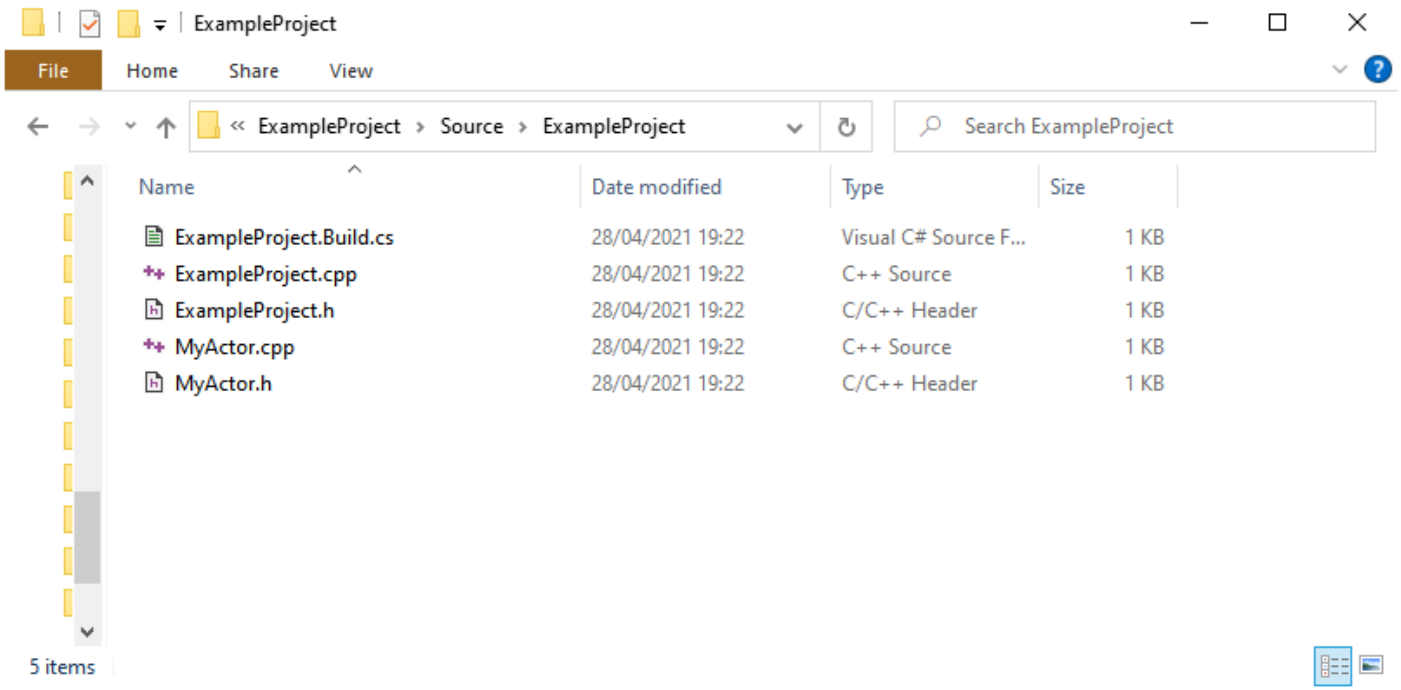



*Figure 7 The file structure for C++ code*

## Importing UDK Assets

Although unused in the project, the process to import UDK assets into UE4 was explored and will be detailed here to aid future works. Firstly, go to File - Import into Level (1), then find the UDK asset to import (2). In the options, select Create one Actor with Components for the hierarchy type (3). This will place an actor in the scene, which should be selected and deleted with the Delete key (4). On the Actors placement tool on the left, find and add the C++ class created earlier (5) then add the static mesh component to it (6). (Epic, 2021)

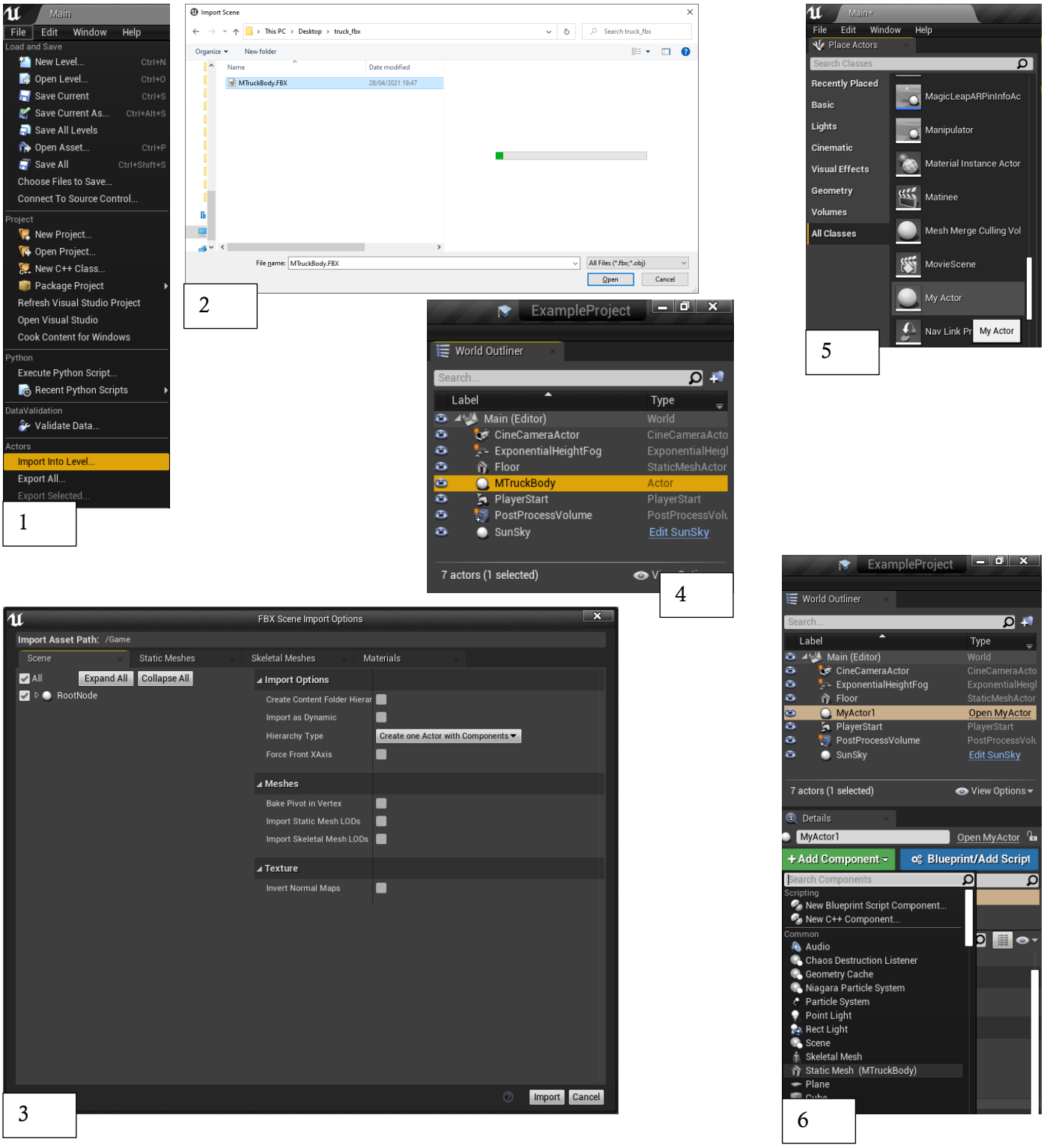



*Figure 8 The process of importing UDK assets*

## Overall System Design

This is the overall structure of the simulator program, showing how the various files are related, as well as the key methods and properties in each. The following pages in this section will document how key functionalities were achieved to meet the system requirements.

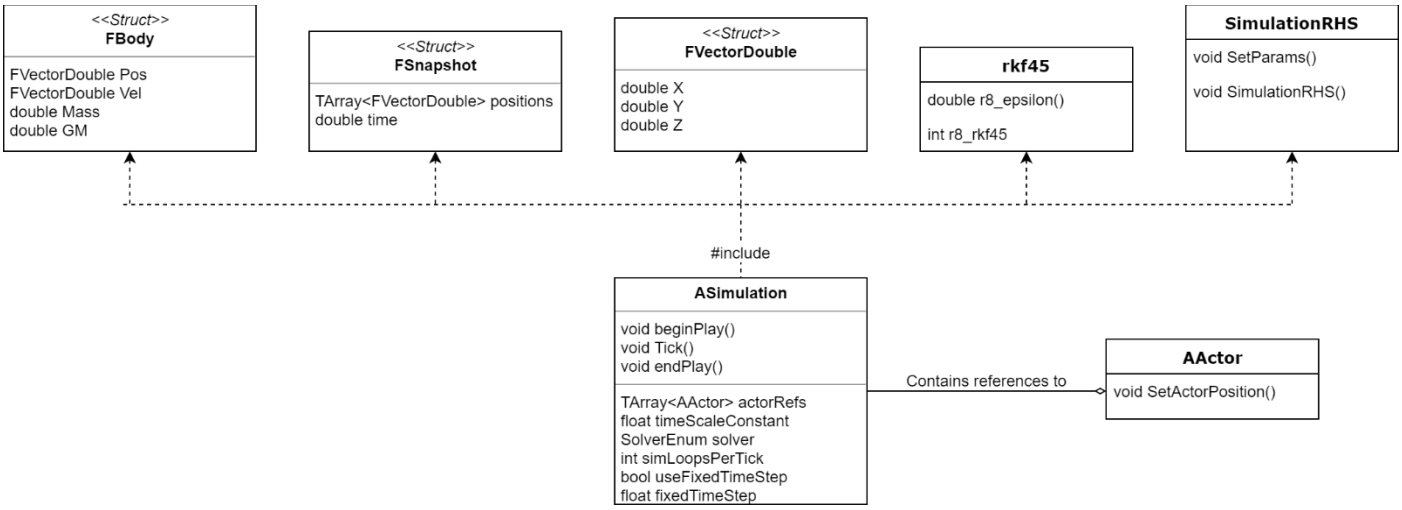



*Figure 9 The structure of the simulation*

Most external files are included in the "ASimulation" class with "#include" tags. The notable exception to this is the references to the actors representing each body in the system. As the "AActor" class is included with the Unreal engine, references to the instances for each body are held in an array. These references are used to call the "SetActorPosition()" method, allowing simulation data to be used to set their positions in the scene without having to use Unreal functionality during simulation calculations.

The simulation will run through the "ASimulation" class "Tick()" method. This is called every frame, and so can be used to continuously call private solver methods to compute the simulation. It also privately holds all of the necessary data for the simulation, such as an array of "FBody" representing each body in the system in double precision.

The "ASimulation" class will also contain the necessary private methods for fulfilling the rest of the required functionality detailed in the requirements specification. Specifically, methods for data importation, scaling and solver initialization will be called in the "beginPlay()" method. Methods for verification, validation and auto-generation of Octave files will be called in the "endPlay()" method.

## Creating Structs

```
#pragma once

USTRUCT(NoExport)
struct FVectorDouble {
    //properties
    double X, Y, Z;

    //multiply with other vector
    FVectorDouble operator*(FVectorDouble vec) {
        return FVectorDouble{
            X * vec.X,
            Y * vec.Y,
            Z * vec.Z
        };
    }

    //multiply with double
    FVectorDouble operator*(double num) {
        return FVectorDouble{
            X * num,
            Y * num,
            Z * num
        };
    }
```

```
//struct for gravitational bodies
USTRUCT(NoExport)
struct FBody {
    FVectorDouble Pos;
    FVectorDouble Vel;
    double Mass;
    double GM;
};

//struct for recording the simulation
USTRUCT(NoExport)
struct FSnapshot {
    TArray<FVectorDouble> positions;
    double time;
};
```

*Figure 10 The structs used in the simulation.*

To maximise accuracy, data must be stored in doubles as they are significantly more precise than floats (Sahu & Rao, 2017). This means that the UE4 built-in vector structs cannot be used for computation as they store values in floats (Epic, 2021). This necessitates the creation of new structs which hold double precision values.

The structs for the simulator are declared in a new "SimulationStructs.h" file and included in the other files with the tag "#include 'SimulationStructs.h'". In the image above, the vector struct is created and declared to the engine with the "USTRUCT" tag. It has X, Y and Z properties all stored as doubles. It is given operator functionalities, with the implementation of multiplication operators shown.

Two additional structs are created which utilize these vectors. First is a struct representing each gravitational body. It has vector properties representing velocity and position, and properties for the bodies mass and standard gravitational parameter (GM). Second is a struct representing a snapshot of the simulation. It contains an array property containing position vectors and a property for time. This will be explained in the Recording the Simulation sub-section.

## Querying Horizons for Real-World Data

```python
#importing the required apis
from astropy.time import Time
from astroquery.jplhorizons import Horizons

#the year to get positions for
year = 2020

#conversion to date for query
datestring = str(year) + "-01-01"
date = Time(datestring).jd

#for each of the 8 planets:
for i, bodyid in enumerate([1,2,3,4,5,6,7,8]):
    #api call to jpl horizons for planetary data at specified time, in vector form
    body = Horizons(id=bodyid, location="@sun", epochs=date, id_type='id').vectors()

    #getting the required data from the returned table
    xPos = body["x"].data[0]
    yPos = body["y"].data[0]
    zPos = body["z"].data[0]
    xVel = body["vx"].data[0]
    yVel = body["vy"].data[0]
    zVel = body["vz"].data[0]

    #writing the data to a txt file
    filepath = "E:/RealPositions/" + str(year) + "positions.txt"
    f = open(filepath, "a")
    stringToWrite = ",".join([str(xPos),str(yPos),str(zPos),str(xVel),str(yVel),str(zVel)]) + ","
    f.write(stringToWrite)
    f.close()
```

*Figure 11 The Python script to generate text files containing real-world positioning and velocity data.*

For gathering the real-world data from Horizons (NASA, 2021), Python was the natural choice for development due to the range of libraries available for simplifying queries to complex databases. This script utilizes Astropy, which is a general-use library for astronomical purposes (Astropy, 2021). Its Horizons method was utilized to perform the query, and the parameters specified the date and the reference point: the sun, as this is used as the stationary centre in the simulation. It can give the values in vector form. Positions are conveniently given in AU, removing the need to rescale. The returned table is parsed into a string with commas separating the values and exported to a text file. This allows the data to be easily imported into the simulation.

## Importing Real-World Data into Simulation

```
//import positions and velocities from txt file
FString importedString;
FString positionFile = FPaths::ProjectConfigDir();
positionFile.Append(TEXT("RealPositions/2000positions.txt"));
FFileHelper::LoadFileToString(importedString, *positionFile);

//parse the imported string into an array of strings
TArray<FString> parsedString;
importedString.ParseIntoArray(parsedString, TEXT(","), true);

//loop through the string aray, casting the strings to doubles, and setting parameters
for (int i = 0; i < 8; i++) {
    //set positions directly to imported values (in AU)
    bodies[i + 1].Pos = FVectorDouble{
        FCString::Atod(*parsedString[i * 6 + 0]),
        FCString::Atod(*parsedString[i * 6 + 1]),
        FCString::Atod(*parsedString[i * 6 + 2])
    };
    //set velocities by converting imported values (in AU/day) to meters/second
    bodies[i + 1].Vel = FVectorDouble{
        (FCString::Atod(*parsedString[i * 6 + 3]) * AU) / 86400,
        (FCString::Atod(*parsedString[i * 6 + 4]) * AU) / 86400,
        (FCString::Atod(*parsedString[i * 6 + 5]) * AU) / 86400
    };
}
```

*Figure 12 The code wrote to import positions and velocities from the Python-generated text files*

To import the Python-generated text files containing real-world positioning and velocity data, the Unreal Engine "LoadFileToString" method was used (Epic Games, 2021). This imports the text file into the selected string. The string was then parsed into an array using the commas as the splitting points. These strings within the array could then be used to set the bodies position and velocity parameters by casting the strings to doubles and converting to other units of measurement where necessary. This same method was also used when importing positions from other years (2001 to 2020) for validation after the simulation has ran.

## Implementing Solvers

By far the most significant part of this project was the successful implementation of the selected ODE solvers: Euler, Euler-Cromer, Runge-Kutta and Runge-Kutta-Fehlberg. The process of doing this took a significant amount of the overall time working on the project. This sub-section will detail how each was implemented according to the general methods described in the literature review.

*Method for calculating accelerations*

```
FVectorDouble ASimulation::CalculateAcceleration(FVectorDouble position, int bodyIndex)
{
    FVectorDouble acceleration = {0,0,0};
    for (int j = 0; j < NR_OBJECTS; j++) {
        if (j != bodyIndex) {
            //calculate square distance between bodies
            FVectorDouble differenceInPositions = position - bodies[j].Pos;
            double distanceBetween = sqrt((differenceInPositions.X * differenceInPositions.X)
                + (differenceInPositions.Y * differenceInPositions.Y)
                + (differenceInPositions.Z * differenceInPositions.Z));
            double squareDistance = distanceBetween * distanceBetween;

            //calculate direction to other body
            FVectorDouble forceDirection = (differenceInPositions / distanceBetween) * -1;

            //calculate acceleration towards other body
            acceleration += (forceDirection * (bodies[j].GM / squareDistance));
        }
    }
    return acceleration;
}
```

*Figure 13 Calculate Acceleration method*

This is the method used to calculate gravitational accelerations. In effect, it is the computerized representation of the Newtonian model, using the simplified version discussed in the review of literature. Importantly, it calculates the acceleration for the requested body according to the position it is passed. It is passed this position rather than directly fetching it from the planet's body struct because more advanced solvers, such as Runge-Kutta, need to call this method for a variety of positions and not just the current one.

### *Euler*

```
void ASimulation::Euler(double DeltaTime)
{
    for (int i = 0; i < simLoopsPerTick; i++) {
        //update positions according to velocity, then update velocities
        for (int j = 1; j < NR_OBJECTS; j++) bodies[j].Pos += bodies[j].Vel * DeltaTime * timeScaleConstant;
        for (int j = 1; j < NR_OBJECTS; j++) bodies[j].Vel += CalculateAcceleration(bodies[j].Pos, j) * DeltaTime * timeScaleConstant;

        //record a snapshot of the simulation
        simTime += (DeltaTime * timeScaleConstant) / timeScale;
        RecordSnapshot();
    }
}
```

*Figure 14 Euler solver*

As Euler's method is so simple, so was its implementation. The method simply updates all the planets positions according to velocities, then updates the velocities by calling the acceleration method with the current position for each planet.

### *Euler-Cromer*

```
void ASimulation::EulerCromer(double DeltaTime)
{
    for (int i = 0; i < simLoopsPerTick; i++) {
        //update velocities, then update positions according to velocity
        for (int j = 1; j < NR_OBJECTS; j++) bodies[j].Vel += CalculateAcceleration(bodies[j].Pos, j) * DeltaTime * timeScaleConstant;
        for (int j = 1; j < NR_OBJECTS; j++) bodies[j].Pos += bodies[j].Vel * DeltaTime * timeScaleConstant;

        //record a snapshot of the simulation
        simTime += (DeltaTime * timeScaleConstant) / timeScale;
        RecordSnapshot();
    }
}
```

*Figure 15 Euler-Cromer solver*

The Euler-Cromer solver is almost identical to Euler, with the exception that the order of operations is swapped so velocities are updated first before positions.

*Runge-Kutta*

```
void ASimulation::RungeKutta(double DeltaTime)
{
    for (int i = 0; i < simLoopsPerTick; i++) {
        TArray<FVectorDouble> deltaVel;
        TArray<FVectorDouble> deltaPos;
        for (int j = 0; j < NR_OBJECTS; j++) {
            deltaVel.Add(FVectorDouble());
            deltaPos.Add(FVectorDouble());

            //Vectors for the four sub-steps, dydt position and velocity
            FVectorDouble K1v,K2v,K3v,K4v,K1r,K2r,K3r,K4r;

            //K1r = v
            K1r = bodies[j].Vel;
            //K1v = a(r)
            K1v = CalculateAcceleration(bodies[j].Pos, j);
            //K2r = v + K1v * h/2
            K2r = bodies[j].Vel + (K1v * (DeltaTime * timeScaleConstant) / 2);
            //K2v = a(r + (K1r * h/2))
            K2v = CalculateAcceleration((bodies[j].Pos + (K1r * (DeltaTime / 2 * timeScaleConstant))) , j);
            //K2r = v + K2v * h/2
            K3r = bodies[j].Vel + (K2v * (DeltaTime * timeScaleConstant) / 2);
            //K3v = a(r + (K2r * h/2))
            K3v = CalculateAcceleration((bodies[j].Pos + (K2r * (DeltaTime / 2 * timeScaleConstant))), j);
            //K4r = v + K3v * h
            K4r = bodies[j].Vel + (K3v * (DeltaTime * timeScaleConstant));
            //K4v = a(r + (K3r * h))
            K4v = CalculateAcceleration((bodies[j].Pos + (K3r * (DeltaTime * timeScaleConstant))), j);

            // NOTE - AS THIS IS THE DELTA IT IS "=" NOT "+="
            //v(i+1) = v(i) + h/6 * (K1v + 2*K2v + 2*K3v + K4v)
            deltaVel[j] = (K1v + (K2v * 2) + (K3v * 2) + K4v) * ((DeltaTime * timeScaleConstant) / 6);
            //r(i+1) = r(i) + h/6 * (K1r + 2*K2r + 2*K3r + K4r)
            deltaPos[j] = (K1r + (K2r * 2) + (K3r * 2) + K4r) * ((DeltaTime * timeScaleConstant) / 6);
        }

        //update velocities and positions according to the calculated vals
        for (int j = 1; j < NR_OBJECTS; j++) {
            bodies[j].Vel += deltaVel[j];
            bodies[j].Pos += deltaPos[j];
        }

        //record a snapshot of the simulation
        simTime += (DeltaTime * timeScaleConstant) / timeScale;
        RecordSnapshot();
    }
}
```

*Figure 16 Runge-Kutta Solver*

The implementation of Runge-Kutta was far more complicated than Euler's method, due to the underlying method being so much more complex. It calls the acceleration method 4 times, firstly with the current position, then with a selection of potential future positions. The final approximation for both velocity and position is given by calculating a weighted average of the four evaluations for each. This solver took a considerable amount of time and effort to get running correctly. It is a notable achievement of this project.

### *Runge-Kutta-Fehlberg (rkf45)*

The implementation for RKF differed from that of the other solvers, due to the solver itself being included from an open-source implementation freely available under the GNU LGPL license (RKF45, 2020). The structure for including the files was shown in the simulation structure diagram.

```
//set the parameters for the rkf solver
void ASimulation::InitializeSolver()
{
    //solver parameters
    flag = 1;
    t = 0.0;
    relerr = 100 * sqrt(r8_epsilon());
    abserr = 100 * sqrt(r8_epsilon());

    //the bodies GM values for the equation
    double GMs[NR_OBJECTS];

    //initial values for solver
    for (int i = 0; i < NR_OBJECTS; i++) {
        y[i * 6 + 0] = bodies[i].Pos.X;
        y[i * 6 + 1] = bodies[i].Pos.Y;
        y[i * 6 + 2] = bodies[i].Pos.Z;
        y[i * 6 + 3] = bodies[i].Vel.X;
        y[i * 6 + 4] = bodies[i].Vel.Y;
        y[i * 6 + 5] = bodies[i].Vel.Z;
        GMs[i] = bodies[i].GM;
    }

    //send parameters to the solver
    SetParams(GMs,timeScaleConstant);
    SimulationRHS(t, y, dydt);
}
```

*Figure 17 Initializing the parameters for the rkf45 solver*

```
void ASimulation::RungeKuttaFehlberg(double DeltaTime)
{
    for (int i = 0; i < simLoopsPerTick; i++) {
        //run the solver
        tout = t + (DeltaTime);
        flag = r8_rkf45(SimulationRHS, NEQN, y, dydt, &t, tout, &relerr, abserr, flag);
        if (flag == 7) flag = 2;

        //update positions according to solver produced values
        for (int j = 0; j < NR_OBJECTS; j++) {
            bodies[j].Pos.X = y[j * 6 + 0];
            bodies[j].Pos.Y = y[j * 6 + 1];
            bodies[j].Pos.Z = y[j * 6 + 2];
            bodies[j].Vel.X = y[j * 6 + 3];
            bodies[j].Vel.Y = y[j * 6 + 4];
            bodies[j].Vel.Z = y[j * 6 + 5];
        }

        //record a snapshot of the simulation
        simTime += (DeltaTime * timeScaleConstant) / timeScale;
        RecordSnapshot();
    }
}
```

*Figure 18 The method which calls the RKF solver and updates positions accordingly*

```
//the function to pass to the solver
void SimulationRHS(double t, double y[], double dydt[])
{
    //makes the sun stationary
    dydt[0] = 0;
    dydt[1] = 0;
    dydt[2] = 0;
    dydt[3] = 0;
    dydt[4] = 0;
    dydt[5] = 0;


    //the rate of change of position is equal to the velocity multiplied by the time scale
    for (int i = 1; i < NR_OBJECTS; i++)  {
        dydt[i * 6 + 0] = y[i * 6 + 3]*timeScaling;
        dydt[i * 6 + 1] = y[i * 6 + 4]*timeScaling;
        dydt[i * 6 + 2] = y[i * 6 + 5]*timeScaling;
    }

    //the rate of change of velocity is the acceleration equation (newtons grav law)
    for (int i = 1; i < NR_OBJECTS; i++)  {
        double accelX = 0.0;
        double accelY = 0.0;
        double accelZ = 0.0;

        //for every other body in the system
        for (int j = 0; j < NR_OBJECTS; j++) {
            if (j =/= i) {
                //calculate differences in x, y and z positions
                double diffX = y[i * 6 + 0] - y[j * 6 + 0];
                double diffY = y[i * 6 + 1] - y[j * 6 + 1];
                double diffZ = y[i * 6 + 2] - y[j * 6 + 2];

                //use pythagoras to calculate the distance and distance squared
                double dist = sqrt( diffX*diffX + diffY*diffY + diffZ*diffZ);
                double dist2 = dist*dist;

                //produce normalized x, y, and z values to give the acceleration a direction
                double normalizedX = -1* diffX/dist;
                double normalizedY = -1* diffY/dist;
                double normalizedZ = -1* diffZ/dist;

                //calculate acceleration towards other body
                accelX += (normalizedX * (bodyGMs[j] / dist2));
                accelY += (normalizedY * (bodyGMs[j] / dist2));
                accelZ += (normalizedZ * (bodyGMs[j] / dist2));
            }
        }
        //the rate of change of velocity is the acceleration multiplied by the time scale
        dydt[i * 6 + 3] = accelX*timeScaling;
        dydt[i * 6 + 4] = accelY*timeScaling;
        dydt[i * 6 + 5] = accelZ*timeScaling;
    }
}
```

*Figure 19 The RHS function to pass to the RKF solver*

Firstly, the RHS function to pass to the solver needed to be developed. This is essentially just the acceleration method but with the vector structures removed and data split into 54 variables (an X,Y,Z for each bodies position and velocity). Then, the solver needs to be initialized with the required data being passed to the RHS function and the other parameters for calling rkf45 being calculated. Finally, the solver itself can be called, passing the RHS function and the other necessary parameters. The positions and velocities for the planets can then be updated according to the results.

## Scaling

```
//scale the system
//sqrt(G*M/AU) M is mass of sun
double velocityScale = sqrt(gravConst * bodies[0].Mass / AU);
for (int i = 0; i < 9; i++) {
    bodies[i].Vel = bodies[i].Vel / velocityScale;
    // Divide masses by mass sun to create scaled gravitational parameters
    bodies[i].GM  = bodies[i].Mass / bodies[0].Mass;
}
//
timeScale = sqrt(gravConst*bodies[0].Mass/(AU*AU*AU));
```

*Figure 20 Scaling*

Implementation of the scaling coefficients was relatively simple, following the formulas designed in the review of literature for scaling.

## Exposure of variables to UE4 editor

```
//variables editable from unreal editor
public:
    UPROPERTY(EditAnywhere)
        TArray<class AActor*> actorRefs;
    UPROPERTY(EditAnywhere)
        float timeScaleConstant;
    UPROPERTY(EditAnywhere)
        SolverEnum solver;
    UPROPERTY(EditAnywhere)
        int simLoopsPerTick;
    UPROPERTY(EditAnywhere)
        bool useFixedTimeStep;
    UPROPERTY(EditAnywhere)
        float fixedTimeStep;
```

*Figure 21 Exposing variables to editor*

Variables were exposed to the editor by using the “UPROPERTY(EditAnywhere)” tag.

### Automatically Generating Octave Files

```
//calculate and export errors in the single body test
void ASimulation::ExportSingleBodyErrors()
{
    TArray<double> errors;
    for (int i = 0; i < tickCount; i++) {
        double distance = sqrt((snapshots[i].positions[0].X * snapshots[i].positions[0].X)
                + (snapshots[i].positions[0].Y * snapshots[i].positions[0].Y)
                + (snapshots[i].positions[0].Z * snapshots[i].positions[0].Z));
        errors.Add((distance - 1) * 100);
    }

    FString logFile = FPaths::ProjectConfigDir();
    logFile.Append(TEXT("GeneratedErrors/CircularOrbit/Errors.m"));
    FString stringToSave = "SingleBodyErr = [\n";
    for (int i = 1; i < tickCount; i++) {
        stringToSave.Append(FString::Printf(TEXT("%e"),snapshots[i].time));
        stringToSave.Append(" ");
        stringToSave.Append(FString::Printf(TEXT("%e"),errors[i]));
        stringToSave.Append("\n");
    }
    stringToSave.Append("];\n");
    stringToSave.Append("plot(SingleBodyErr(:,1), SingleBodyErr(:,2));\n");
    stringToSave.Append("xlabel('Time');\n");
    stringToSave.Append("ylabel('Error %');\n");
    stringToSave.Append("xlim([0,50]);\n");
    FFileHelper::SaveStringToFile(stringToSave, *logFile);
}
```

*Figure 22 Auto-Generating Octave files*

Octave files were generated using the FFileHelper string methods. The above example shows the graphs generated for the verification process.

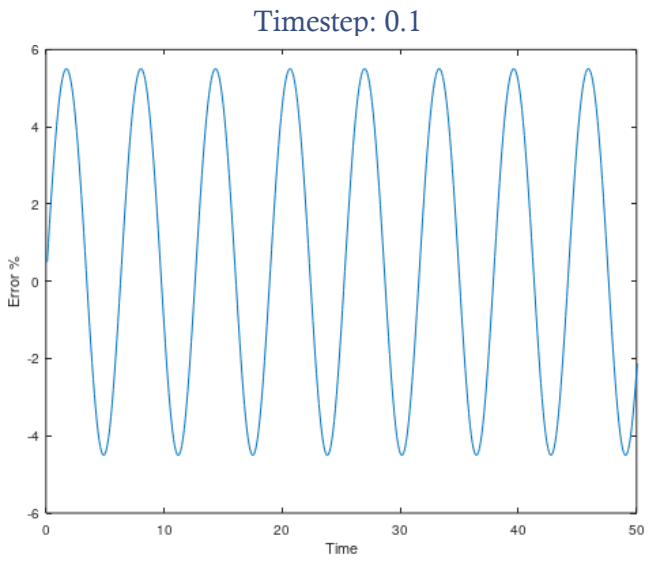
## Analysis

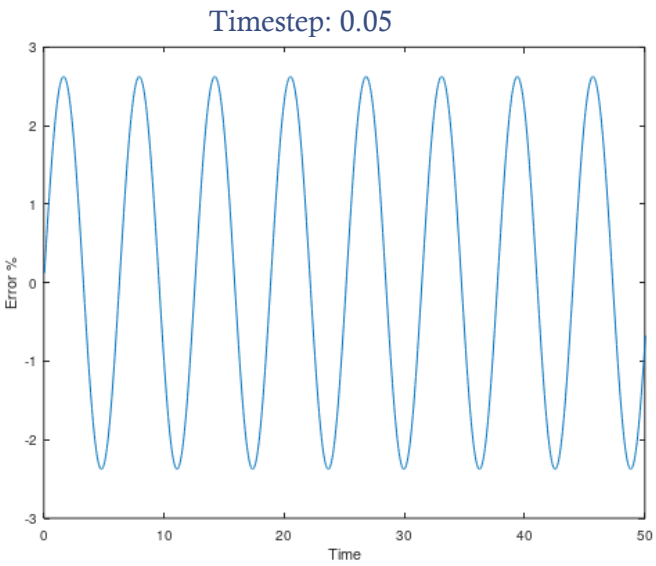
This section will document the results collected from the simulation by using the previously detailed methods of validation and verification. Concise summaries of results will be provided, with a more detailed discussion being given in the next section.

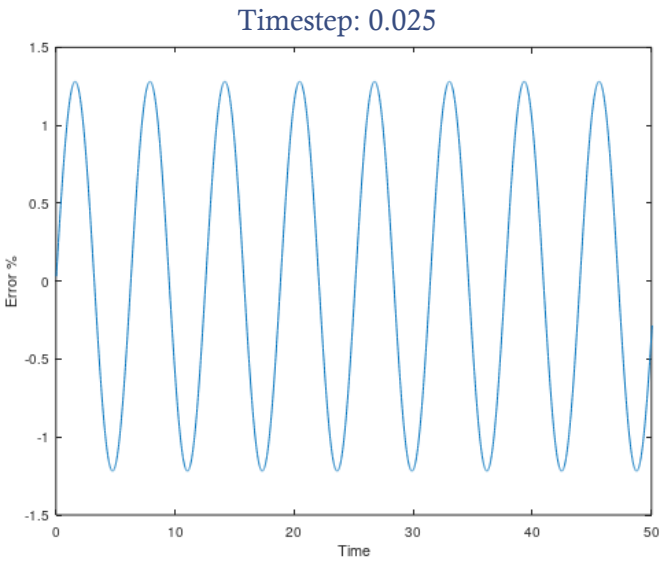
### Verification

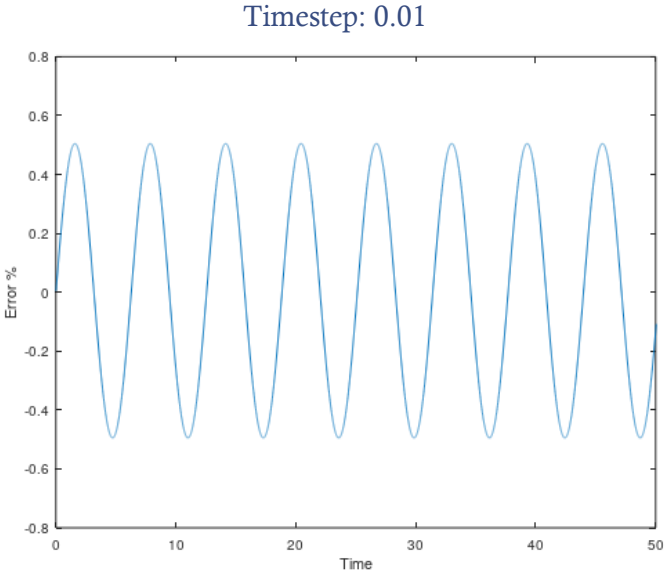
Following are the error percentages for the simulation as it runs the problem of a single body orbiting a fixed mass. It should have a perfectly circular orbit with a radius of 1 unit. Percentage deviations in radius are recorded as the error percentage. Each solver is tested with timesteps of 0.1, 0.05, 0.025 and 0.01.

*Euler*



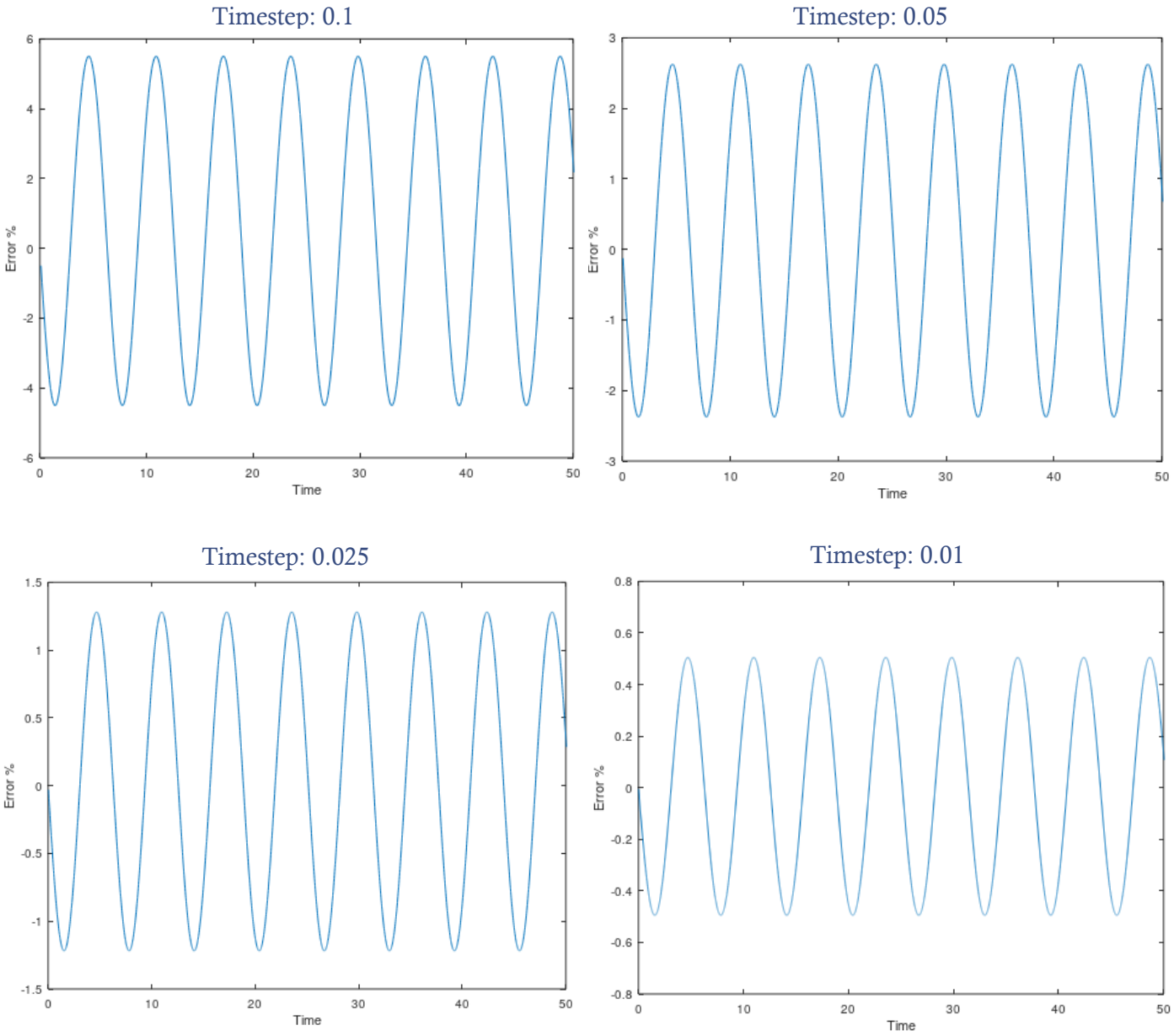




*Euler Summary*

With a timestep of 0.1, the error consistently peaks at 5.5%. When the timestep is halved to 0.05, the error also halves to 2.75%. This is a consistent pattern; a quarter timestep of 0.025 produces a quarter error of 1.38% and a 10x smaller timestep of 0.01 produces a 10x smaller error of 0.55%.

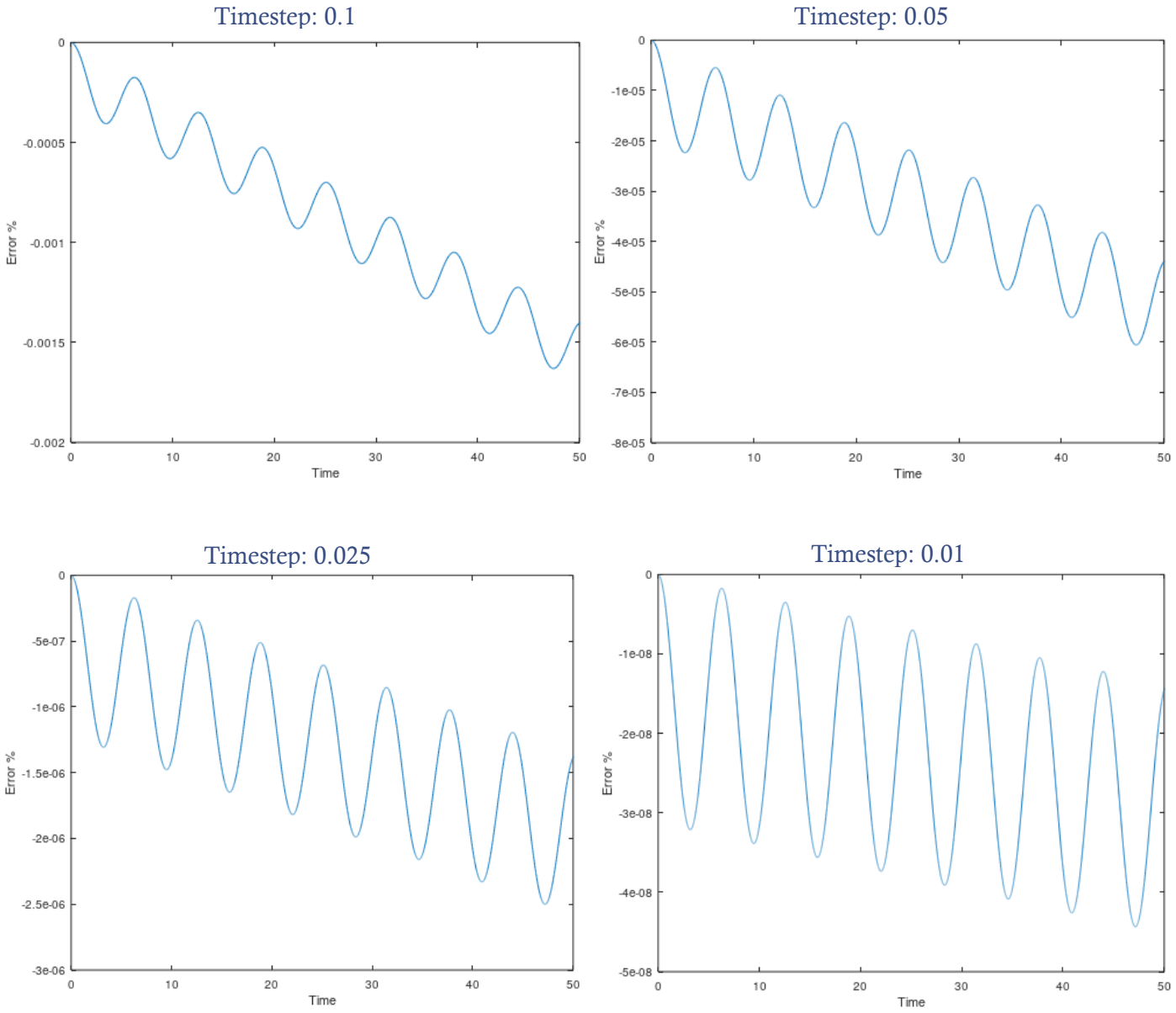
*Euler-Cromer*




*Euler-Cromer Summary*

Interestingly, the Euler Cromer results are near identical to the Euler results. The only difference is that the errors start off progressing towards the negative, whereas the Euler errors start progressing towards the positive. The peak errors and their relationship with the change in timestep are indistinguishable from Euler.

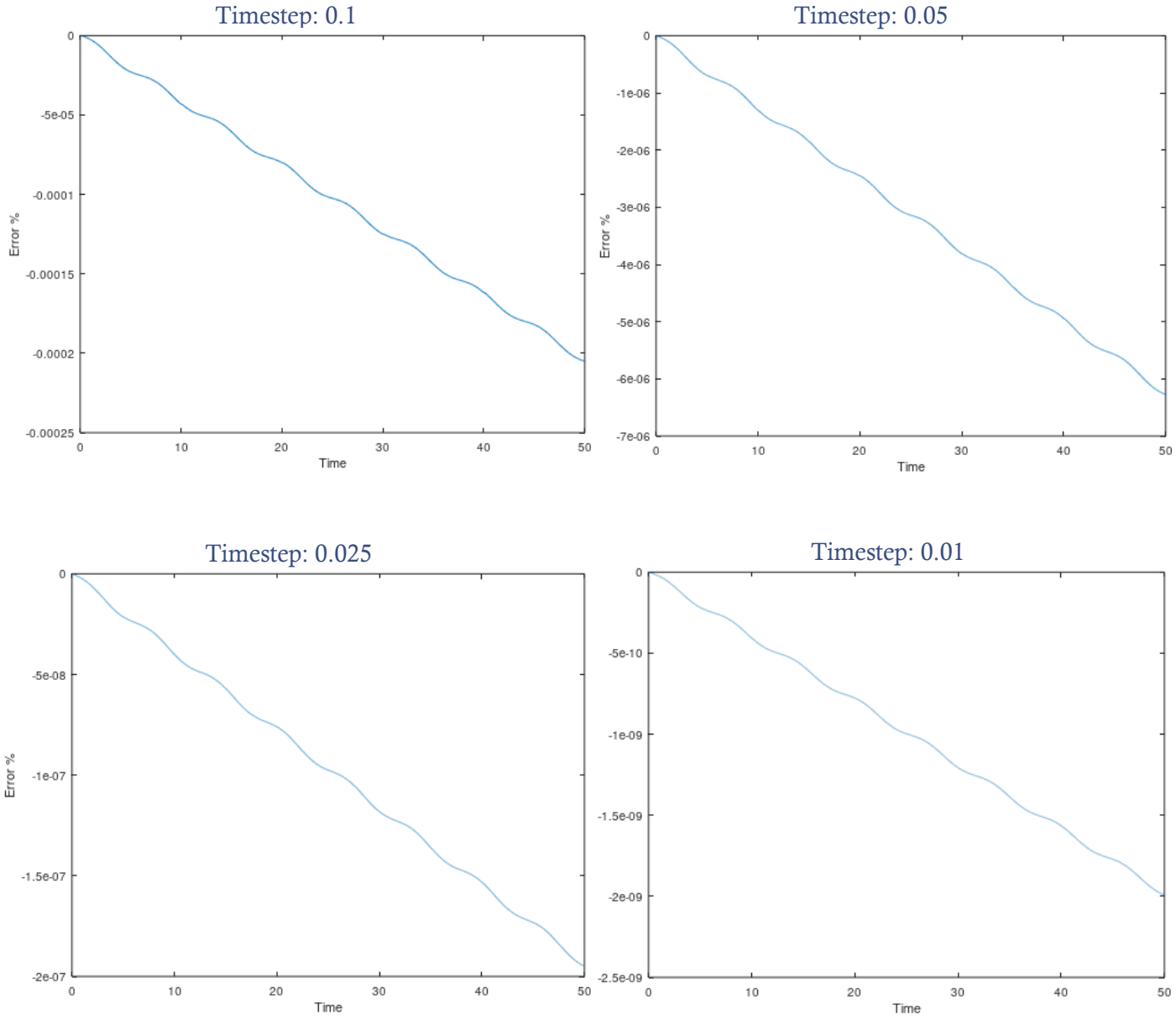
*Runge-Kutta*




*Runge-Kutta Summary*

In general, the error percentages for Runge-Kutta are smaller than the Euler and Euler-Cromer solvers by several orders of magnitude, an extreme increase in accuracy. However, there is a pattern of gradual increase in error percentages over time, which is not the case with Euler and Euler-Cromer. The changes in timestep also have a different effect; halving the timestep reduces errors by ~16x, quartering timestep reduces errors by ~256x, and reducing timestep by 10x reduces errors by ~10000x. However, the variances increase greatly with the reduction in timestep.

*Runge-Kutta-Fehlberg*



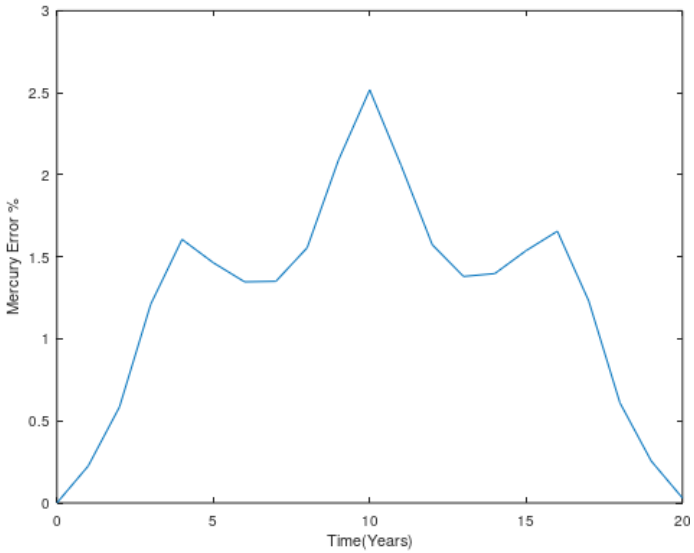

*Runge-Kutta-Fehlberg Summary*

The RFK results are more accurate than even the Runge-Kutta, by roughly 10x. The wave-like variations in accuracy are greatly reduced, with the biggest factor being the gradual decrease in accuracy over time. The effects of changes in timestep are the same as with Runge Kutta, with decreases in timestep having a heavily outsized effect on accuracy improvement.

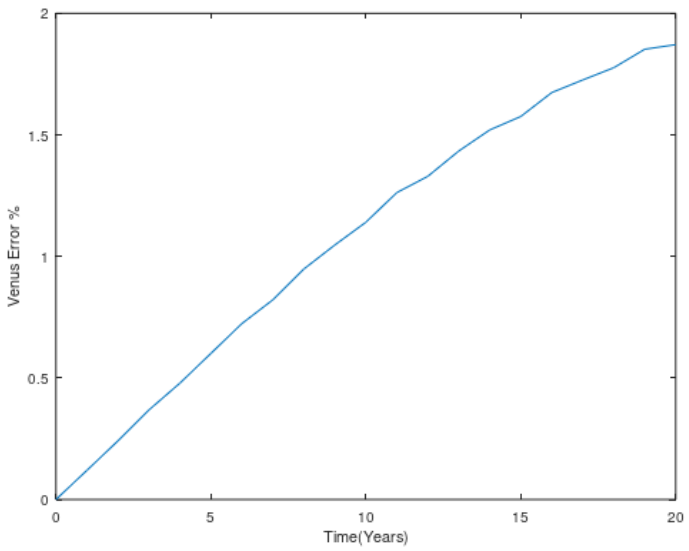
## Validation

Following are the error percentages as a 20-year timeframe of the real solar system is simulated. The simulation is initiated with real-world positions and velocities on the 1st of January 2000 and compared to real positions on the 1st of January each year after, up to 2020. As it was found to be the most accurate after verification, the Runge-Kutta-Fehlberg solver has been used with a timestep of 0.001. There is a graph for each planet's error percentages, followed by a graph for the average errors across the whole system. As with the verification results, the validation results will be discussed in detail in the next section.

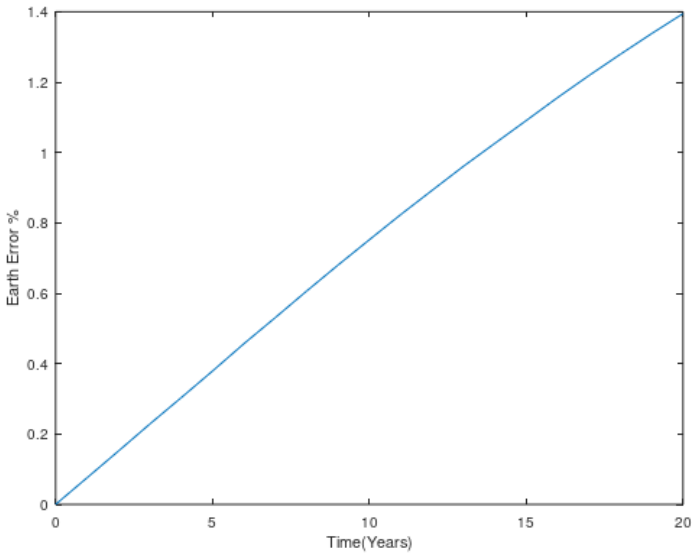
Mercury



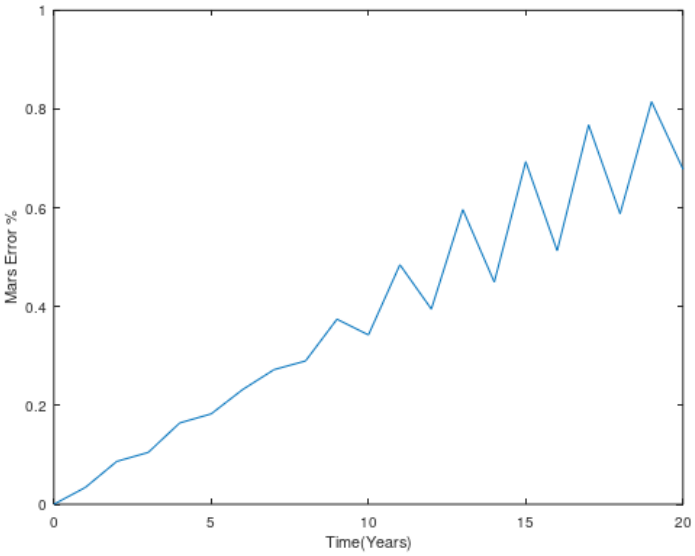

Venus



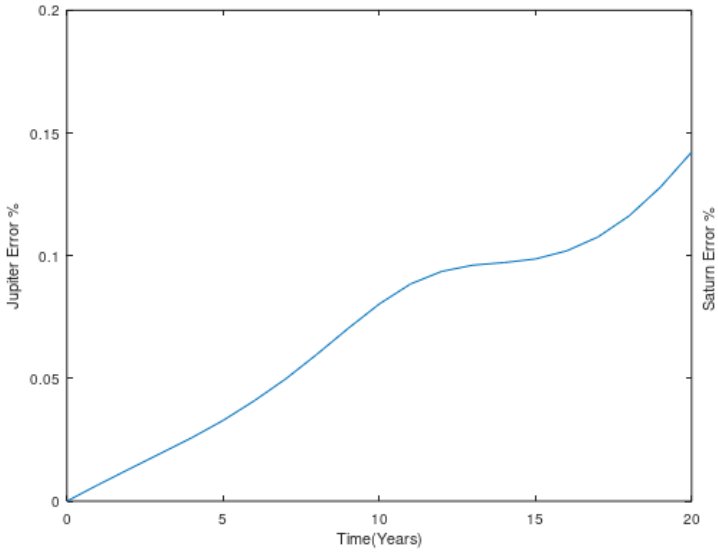

Earth



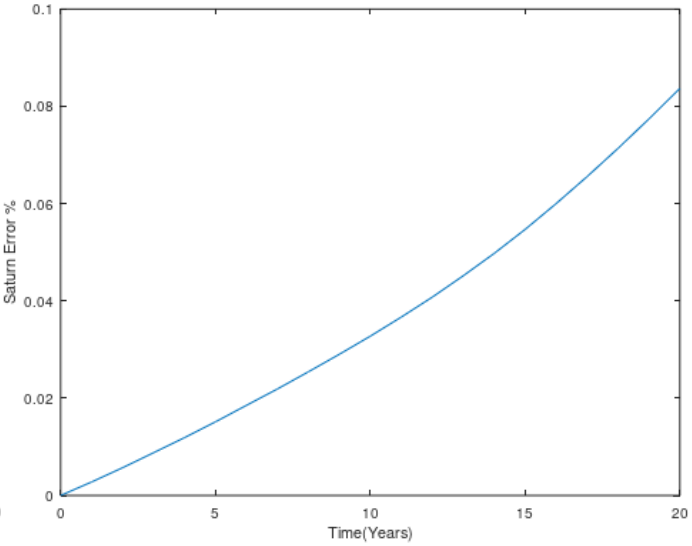

Mars



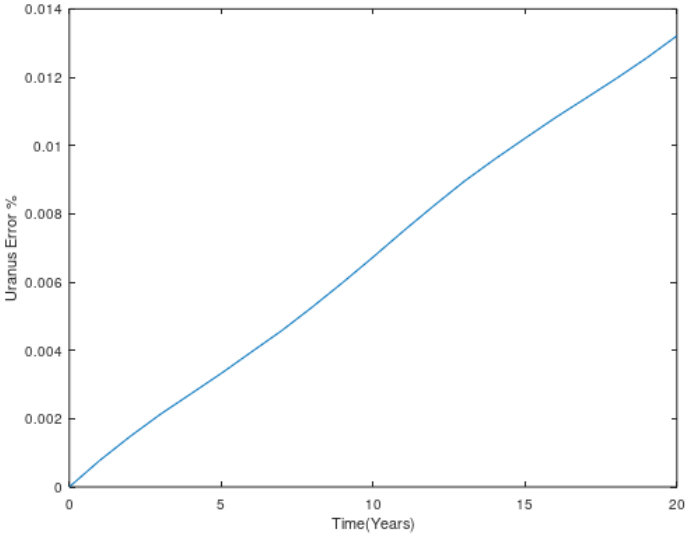
## Jupiter



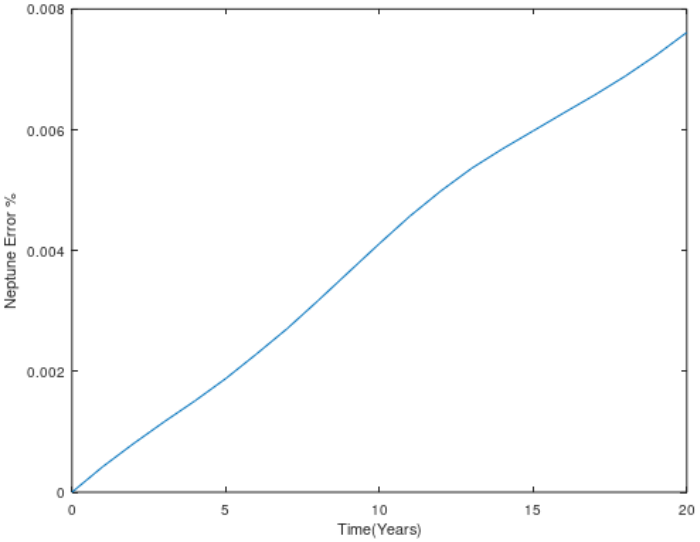

## Saturn



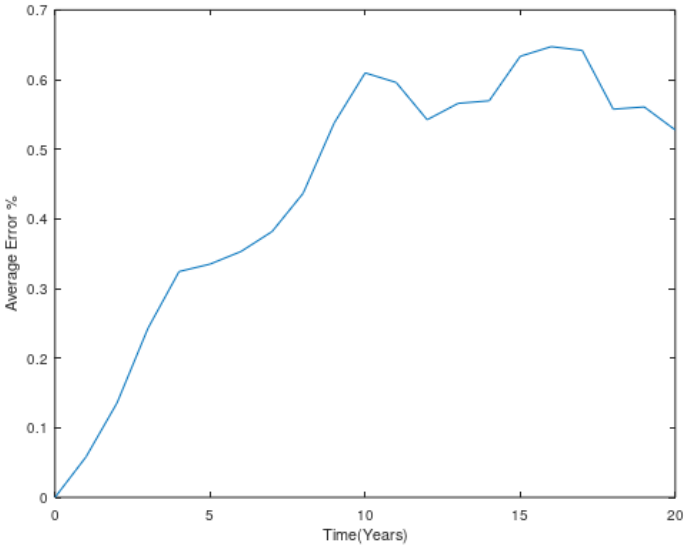

## Uranus



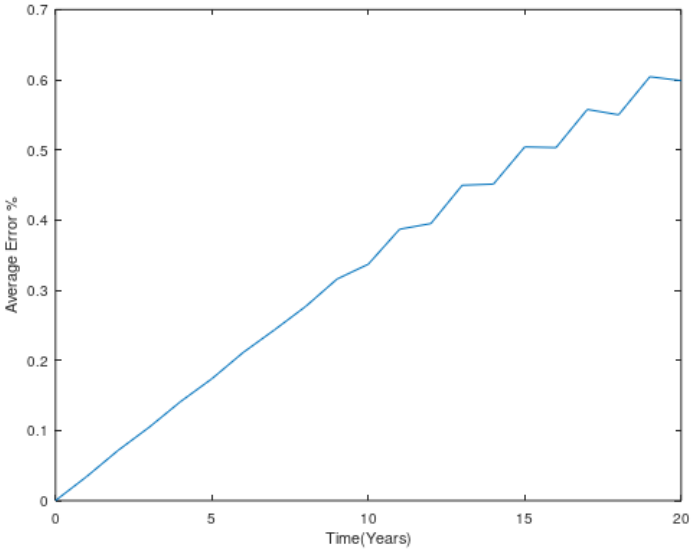

## Neptune




## Average across whole system




## Average excluding Mercury



# Discussion

This section will provide a more in-depth discussion of the results from the verification and validation processes and link the results to the key concepts discussed in the review of literature. This will allow for the research questions to be answered in detail.

## Verification

During verification, several significant concepts were evidenced. Additionally, the data gathered gives a clear answer to the research question on the most effective solver. The results are mostly in line with the concepts discussed in the literature review, with RKF proving why it is the clear favourite.

### *Euler & Euler-Cromer*

Overall, the error percentages for Euler and Euler-Cromer reflected the simplicity of the method. Although reasonable approximations for the body's positions were produced, they were not of the accuracy to be expected of a simulation. Although this was suggested in the reviewed literature (Press, et al., 1992), it still came as a surprise the scale of the inaccuracy. With a timestep of 0.1, errors surpassed 5% and even with a timestep reduction to 0.01 they remained at 0.5%. This gives a good enough approximation to visually represent the system, but for extracting data far more precise methods of solving ODEs are required. Although anecdotally interesting, the mirroring of errors in Euler-Cromer in comparison to Euler are not a significant consideration. As the error percentages remain the same for both, they both equally fail the verification process.

### *Runge-Kutta*

Runge-Kutta surpassed expectations with its achieved accuracy. Even with a timestep of 0.1, errors never surpassed 0.002% in the timeframe of the test. This is remarkably accurate. Even still, the accuracies significantly improved with reductions in timestep, achieving error percentages under $5x10^{-8}$ with a timestep of 0.01. Although there was shown to be a gradual and consistent growth in inaccuracies, the scales of these are so small that it would take an extremely long amount of time before they surpassed the errors of Euler's method. The increases in accuracy with reduction in timestep also evidenced an important concept discussed in the literature: the method achieves fourth-order accuracy. As the timestep was halved, error ratings reduced by roughly 16x, or $2^4$. This was consistent across all timestep reductions, with the 10x smaller timestep of 0.01 achieving roughly $10^4$=10000 times the accuracy. This is what caused the extreme levels of accuracy at this timestep. However, the variations in error percentages also grew significantly with these reductions, causing some of these accuracy gains to be lost.

### *Runge-Kutta-Fehlberg*

It is of little surprise that RKF performed the best out of the solvers tested, as it was clearly highlighted in the literature reviewed that it is the favourite amongst professionals in the field (Press, et al., 1992). Its usage of dynamically adjusted timesteps alongside the Runge-Kutta method led to the most extreme levels of accuracy, achieving error percentages not exceeding $2x10^{-9}$ with a timestep of 0.01. As proven with Runge-Kutta, fourth order accuracy was achieved which is what lead to this performance. Additionally, however, the variation patterns in errors shown in the other solvers, all with fixed timesteps, was reduced drastically. The most significant factor leading to inaccuracy with RKF was the slow decay in accuracy over time. It can be speculated

that, with errors this small, this slow decay in accuracy could be caused by the limitations of double-precision floats (Sahu & Rao, 2017). However, this is outside of the scope of this project, and due to the extremely small growth in errors, this factor can be safely ignored. Compared to Runge-Kutta, RKF achieved errors 10x as small, clearly validating it as the solver to use. It has also been proven that changes in timestep play a very significant factor in accuracy, and a mixture of dynamic adjustment through RKF and general minimisation of the timestep is necessary to achieve the highest accuracy possible.

## Validation

### *Individual planets*

Looking at the individual planets, a clear pattern emerges. The inner planets, with their much higher velocities and shorter orbit time, are significantly more inaccurate than the outer planets. At the furthest ends, Mercury reaches an error of 2.5%, by far the largest, and Neptune does not exceed 0.008, by far the smallest. A potential reason for this is the difference in the amount of orbits the planets undergo within the timeframe. Referring to the Planetary Fact Sheets (NASA, 2019), Mercury will orbit 683.89x more times in a given timeframe. If Neptune's error is multiplied by this to compensate, its error reaches 5.47%, which is much more similar in scale. This does not mean that it was inaccurate, but it shows how the outer planets will naturally be more accurate within a timeframe due to their slower orbits.

### *Causes of errors*

There are several factors which could be leading to the errors in the simulation. Firstly, this simulation only included the major planets and the sun as bodies. This excludes the gravitational effects caused by countless other bodies in the system, and although individually minuscule compared to those of the planets (NASA, 2019), they could cumulatively have a small effect.

Another explanation could be the weaknesses of the Newtonian model itself. As stated in the reviewed literature, the Newtonian model provides a good approximation tool for planetary positions, but for maximum precision a more complex model will be needed (Malek, 2016). This could also explain the strange error behaviours examined with Mercury. Mercury has been singled out in the literature as a particular point of error within Newtonian physics due to its perihelion progression and other real-world factors that are under-represented in the model. Due to the anomalous nature of Mercuries errors, a graph of averages has been produces which excludes Mercury to show the steady erosion of accuracy more clearly over time. All other planets show this in a much clearer fashion.

### *Overall impression*

The validation results show that overall, the simulation achieved a respectable amount of accuracy. In the 20-year timespan simulated, the average error for the system did not exceed 0.7%, with a steady rise in error percentage of roughly 0.03% per year with Mercury excluded. With Mercury included, the decay of accuracy is slightly greater but less consistent. In general, it has been proven that UE4 is an effective platform for simulation, able to support high precision solvers to simulate systems of ODEs.

## Success of scaling system

The scaling methods implemented were successful, and the simulator was able to simulate a scaled version of the solar system which brought the values for distances, velocities and masses to much more reasonable ranges. Time was also successfully rescaled post-runtime for the validation process. This allowed for the simulated system to be analysed as if it had run in full scale. This means that the project requirement for scaling the system has been successfully met.

## Overall Evaluation of Artefact

Overall, the artefact is successful. Referring to the requirements specification, all key functionalities were achieved. In particular, the system was successfully scaled for higher precision solving, before being rescaled back for validation. During validation, it was shown that the simulator was able to achieve a high level of accuracy for most planets, excluding Mercury. For this, potential reasons for the strange results have been given, referring to the outlined literature. The answers for all three research questions have been found, and quotes from this discussion will be taken to directly answer each one:

*Is Unreal Engine 4 an appropriate platform for building a physics simulator?*

"In general, it has been proven that UE4 is an effective platform for simulation, able to support high precision solvers to simulate systems of ODEs."

*What numerical solver provides the most accuracy?*

"Compared to Runge-Kutta, RKF achieved errors 10x as small, clearly validating it as the solver to use."

*What are the effects of changes in timestep on accuracy?*

"It has also been proven that changes in timestep play a very significant factor in accuracy, and a mixture of dynamic adjustment through RKF and general minimisation of the timestep is necessary to achieve the highest accuracy possible."

# Reflection

This section will reflect on the achievements and limitations of the project post-completion. It will also suggest any future improvements which can be made to further explore the research questions and make additional contributions to the field of simulation.

## Deviations from initial plan

### *Controllable spaceship*

The original plans for this project at the very start included the development of a controllable spaceship and mentions of this can still be found in the ethics document. However, this was quickly abandoned as it was unnecessary for achieving the aims of the project. It was also due to a change of ethos towards a much more numerically oriented project rather than an interactive one.

### *Gantt chart*

It was originally stated that the Gantt chart was not intended as an exact plan for time allocation. In practice, the project almost entirely deviated from the timeframes specified in the chart, with some tasks being performed in different orders and entirely different times than originally planned. This was expected however, and the flexible attitude towards time management proved an important factor towards project completion, especially when balancing the project with other university modules and difficulties arising from having to work remotely.

## Achievements

### *Successful development of an accurate gravitational simulation*

The simulation was successfully completed and achieved a high level of accuracy when compared to real-world data. This took a substantial amount of time an effort to learn the workings of both UE4 and the solvers utilized. The simulation also makes good use of Unreal functionality, such as the ability to pass references to other bodies through the engine and assign values to variables on the editor.

### *Detailed documentation of development*

An in-depth documentation of the development process was produced, which gives the information needed for future works to develop their own simulations using this documentation as a reference point. All key aspects were covered, from the import and exportation of data to the process of attaching C++ code to engine actors.

### *Insightful comparison of solvers and timesteps*

By performing the designed process of verification on each solver, useful insights were found regarding their ability to accurately simulate the Newtonian model. The comparisons of various timesteps also led to clear, useful conclusions being made. The proving of Runge-Kutta and RKF solvers as having fourth-order accuracy was a notable achievement of the validation process.

## Limitations

### *Lack of detailed 3D models for planets*

Although the process for doing so has been detailed, this project did not import assets to provide detailed 3D models of the planets. Instead, simple plain spheres with tracing lines were used to visualise the simulation. This was mainly due to time limitations; numerical solvers and scaling experimentation took up most of the time as they were the focus of the project. However, future works using this project as a basis could use the included information to provide a more aesthetically pleasing and visually representative simulation.

### *Lack of user interface*

One area which was not explored was the development of a user interface. This would both convey detailed information to the user during runtime and allow for the editing of parameters without having to use the editor window. The developed simulation makes use of functionality to allow variables to be assigned values within the editor without being hard-coded but given more time a proper user interface would make for a much more elegant solution.

## Future Improvements

### *Further exploration of solver options*

This project tested four options of numerical solvers: Euler, Euler-Cromer, Runge-Kutta, and Runge-Kutta-Fehlberg. There are many more options to explore, and future works could use the processes designed during this project to test a range of other solvers.

### *Further verifications methods*

Although the verification method used in this project led to important insights being made, further methods of verification could be developed to further evidence the conclusions reached in this work. A benchmark simulation could be developed in an environment such as Octave to provide direct comparisons of solvers while simulating the real solar system.

### *Porting existing UDK simulators to UE4*

As stated throughout this work, a significant contribution intended by the project is to allow for existing UDK simulators to be ported to UE4 by using this work as a basis. Now that the foundational knowledge has been established in this paper, this can be achieved and would make a great contribution for future works to achieve.

# Conclusion

In conclusion, the project was a success. The simulator was successfully developed, with the development process clearly documented. Runge-Kutta-Fehlberg was proven to be the most effective of the solvers tested, with fourth-order accuracy improvements with a reduction in timestep. The simulator was shown to be accurate, proving UE4 as an appropriate platform for simulation. This means that all three research questions were answered. Additionally, all key functionality elicited in the requirements specification was successfully implemented. The findings from the verification and validation processes were clearly linked to the theory outlined in the literature review.

# References


Astropy, 2021. *Astropy Documentation.* [Online]
Available at: https://docs.astropy.org/en/stable/
[Accessed 11 April 2021].

Briggs, R. & Carlisle, R., 2016. *Solar Physics and Terrestrial Effects.* [Online]
Available at:
https://www.swpc.noaa.gov/sites/default/files/images/u2/Solar%20Physics%20And%20Terrestrial%20Physics.pdf
[Accessed 20 March 2021].

Catapult Satellite Applications, 2019. *SMALL SATELLITE MARKET INTELLIGENCE REPORT.* [Online]
Available at: https://s3-eu-west-1.amazonaws.com/media.newsa.catapult/wp-content/uploads/2019/05/30124810/19052056-Small-Sat-Market-Intelligence-report-Q1-2019.pdf
[Accessed 5 April 2021].

Epic Games, 2021. *FFileHelper::LoadFileToString | Unreal Engine Documentation.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/Core/Misc/FFileHelper/LoadFileToString/1/index.html
[Accessed 11 April 2021].

Epic, 2021. *FVector.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/Core/Math/FVector/index.html
[Accessed 21 April 2021].

Epic, 2021. *Import Into Level (FBC Scene Import).* [Online]
Available at: https://docs.unrealengine.com/en-US/WorkingWithContent/Importing/FBX/FullScene/index.html
[Accessed 8 April 2021].

Epic, 2021. *Programming with C++.* [Online]
Available at: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/ProgrammingWithCPP/index.html
[Accessed 24 April 2021].

Fleury, P., 2019. *Gravitation: From Newton to Einstein.* [Online]
Available at: https://arxiv.org/pdf/1902.07287.pdf
[Accessed 12 February 2021].

Huang, T., Han, C., Yi, Z. & Xu, B., 1994. *What is the astronomical unit of length?.* [Online]
Available at: https://www.researchgate.net/publication/234506727_What_is_the_astronomical_unit_of_length
[Accessed 16 March 2021].

Jones, H., 2018. The Recent Large Reduction in Space Launch Cost. *International Conference on Environmental Systems.*

Kluever, C., 2003. Spaceflight Mechanics. In: R. Meyers, ed. *Encyclopedia of Physical Science and Technology.* 3rd ed. s.l.:Academic Press, pp. 507-520.

Langtangen, H. & Pedersen, G., 2016. *Scaling of Differential Equations.* [Online]
Available at: https://hplgit.github.io/scaling-book/doc/pub/book/pdf/scaling-book-4screen-sol.pdf
[Accessed 15 April 2021].

Malek, A., 2016. *THE CONCEPTUAL DEFECT OF THE LAW OF UNIVERSAL GRAVITATION OR 'FREE FALL': A DIALECTICAL REASSESSMENT OF KEPLER'S LAWS.* [Online]
Available at:
https://www.researchgate.net/publication/317040165_THE_CONCEPTUAL_DEFECT_OF_THE_LAW_OF_UNIVERSAL_GRAVITATION_OR_%27FREE_FALL%27_A_DIALECTICAL_REASSESSMENT_OF_

KEPLER%27S_LAWS
[Accessed 20 April 2021].

Milyukov, V. & Fan, S., 2012. *The Newtonian gravitational constant: Modern status of measurement and the new CODATA value.* [Online]
Available at:
https://www.researchgate.net/publication/257847131_The_Newtonian_gravitational_constant_Modern_status_of_measurement_and_the_new_CODATA_value
[Accessed 22 February 2021].

NASA, 2015. *Journey To Mars - Pioneering Next Steps in Space Exploration.* [Online]
Available at: https://www.nasa.gov/sites/default/files/atoms/files/journey-to-mars-next-steps-20151008_508.pdf
[Accessed 23 April 2021].

NASA, 2019. *Planetary Fact Sheet.* [Online]
Available at: https://nssdc.gsfc.nasa.gov/planetary/factsheet/
[Accessed 18 March 2021].

NASA, 2021. *HORIZONS System.* [Online]
Available at: https://ssd.jpl.nasa.gov/?horizons
[Accessed 21 April 2021].

Press, W., Teukolsky, S., Vetterling, W. & Flannery, B., 1992. *Numerical Recipes in C: The Art of Scientific Computing.* 2nd ed. Cambridge: Cambridge University Press.

Price, C., 2021. *Comp 3352 Mini-Lectures and Workshops.* [Online]
Available at: https://colin-price.wbs.uni.worc.ac.uk/Courses_2020_21/Comp3352/Workshops.htm
[Accessed 24 April 2021].

RKF45, 2020. *Runge-Kutta-Fehlberg ODE Solver.* [Online]
Available at: https://people.sc.fsu.edu/~jburkardt/cpp_src/rkf45/rkf45.html
[Accessed 14 April 2021].

Sahu, A. & Rao, S., 2017. *Floating Point Numbers.* [Online]
Available at: https://www.iitg.ac.in/asahu/cs101-2017/Lec07.pdf
[Accessed 15 March 2021].

Sargent, R., 2013. Verification and validation of simulation models. *Journal of Simulation,* 7(1), pp. 12-24.

Solar System Scope, 2021. *About | Solar System Scope.* [Online]
Available at: https://www.solarsystemscope.com/about
[Accessed 2 March 2021].

The Sky Live, 2021. *About | TheSkyLive.com.* [Online]
Available at: https://theskylive.com/about
[Accessed 14th March 2021].

Ydri, B., Bouchareb, A. & Chemam, R., 2013. *Lectures on Computational Physics.* [Online]
Available at: https://homepages.dias.ie/ydri/COMPUTATIONALNOTES_FINAL.pdf
[Accessed 9 April 2021].

# Milestone 3: Computing Projects Proposal Form 2020/21

**See the Computing Projects Blackboard VLE for further guidance on how to complete this form.**
**You can change the sizes of the boxes.**

| **A STUDENT DETAILS (to be completed by the student)** | | | |
|---|---|---|---|
| **Name:** | Dean Sisman | | |
| **University email:** | Sisd1_15@uni.worc.ac.uk | | |
| **Student Number:** | 15007088 | **Module code:** | COMP3008 |
| **Project Completion Date:** | 15.00<br>Thursday 29th April 2021 | **Pathway:** | Computing |
| **Part time / Full time:** | Full time | **Year started course:** | 2018 |
| **Single/Major/Joint Honours:** | Single Hons Computing | **If joint/major, other subject:** | |

| B PROJECT DETAILS (to be completed by the student from milestone 1) | |
|---|---|
| **Supervisor:** | Colin Price |
| **Project Title:** | Design, Build and Test a Solar System Simulation |
| **Type of Project:** | Design-Build-Test (DBT) |
| **Project Aims:**<br><br>**(***A broad statement of the expected outcomes)* | To build a gravitational simulation of the solar system<br><br>To test the simulation for numerical accuracy<br><br>To use the testing data to gauge the effectiveness of the solution, and the appropriateness of the platform used to develop the simulation |
| **Research Question(s):** | How accurate is an orbital simulation developed in a game engine? |
| **Proposed Primary Research** | Numerical data from the simulation, such as positioning and acceleration data |
| **Proposed Secondary Research** | Research of videogame engines which could be used to develop the simulation<br><br>Research of other orbital simulations, what they achieved and their effectiveness<br><br>Research of Newtonian physics<br><br>Orbital information of the real solar system, such as orbital distances, inclinations, eccentricities, and velocities. |

| C SMART Project Objectives |
| --- |
| • Complete all milestones within the deadlines<br>• Research examples of orbital simulations before starting development<br>• Research capabilities of various game engines before starting development<br>• Complete MOSCOW analysis of aims<br>• Complete Gantt chart<br>• Research Newtonian physics<br>• Research orbital information of planets and moons in the solar system<br>• Complete first prototype of simulation<br>• Complete first draft of report |

| D REQUIREMENTS (A list of resources you will need such as software, hardware, data sources etc.) | |
| --- | --- |
| A reliable internet connection<br><br>A computer with a reasonably capable processor and GPU<br><br>Access to the student library services for academic sources | |
| **Agreed Word Count** | Max 9000 words |

**Grading Matrix Template**

This matrix captures the assessment criteria for the **Design-Build-Test** Computing Project Assessment.

| **Student Name:** | **Student Number:** | **Academic Year:** 2020/2021 | **Module Code:** COMP300x |
|---|---|---|---|
| **Assessment Criteria**<br>**1.** Demonstrate a critical understanding of a significant computing issue aligned with the degree pathway [LO1].<br>**2.** Provide a comprehensive critical review of relevant and contemporary literature [LO2].<br>**3.** Design and construct an appropriate software or hardware computing artefact [LO3].<br>**4.** Gather and analyse original data [LO4].<br>**5.** Provide an accurate and critical evaluation of project outcomes against the stated objectives [LO5].<br>**6.** Provide a personal assessment of the project contribution to addressing a significant computing problem and a reflection on your management of the project [LO6]. | | | **Assignment Description:** Project Report + Artefact<br>**Assignment Weighting:** 100%<br>**Module Title:** Computing Project |

| **Assessment Criteria** | | | | | | |
|---|---|---|---|---|---|---|
| | **Demonstrate a critical understanding of a significant computing issue aligned with the degree pathway** | **Provide a comprehensive critical review of relevant and contemporary literature** | **Design and construct an appropriate software or hardware computing artefact informed by primary research (requirements elicitation) and/or secondary research and/or critical literature review** | **Gather and analyse primary data, either using the artefact, and/or regarding the design (requirements elicitation) and/or performance (user testing) of the artefact** | **Provide an accurate and critical evaluation of the artefact against the stated design objectives or requirements specification** | **Provide a personal assessment of the project contribution to addressing a significant computing problem and a reflection on your management of the project** |

| **For students:** **The sort of stuff that could be included** These lists are indicative (suggestions). They are not prescriptive (you have to do them all) or exhaustive (include everything you should do) | **Why am I doing this project? Why is it useful? What problem is it solving? How does it link to my specialist pathway and/or employment aspirations? Research question(s)** | **Literature review; citations; reference list; what other work has been done in this area? (Do authors agree with each other?) How does my proposed work link in with this? Best practice** | **Planning, including Gantt chart; methodology; Project aims; Project objectives; product specification; strategy; justification of strategy; design documentation (wireframes, DFDs, ERDs, etc.); commentary on the design and construction process; limitations** | **Data acquisition; justification of data gathering technique(s); data description; data analysis; conclusions based on data; limitations of data gathering techniques, data gathered and conclusions reached** | **Does the project achieve its aims and meet its objectives? What is/are the answer(s) to the research question(s)? Do the findings agree with previous work? Limitations, how might they be overcome in the absence of resourcing restrictions? Suggestions for further work** | **What did the project achieve? What did I learn from doing this project, from my supervisor/ other students/the showcase/…? What went well? Why? What didn't work? Why? How might it be done even better next time? Deviations from planning** |
|---|---|---|---|---|---|---|
| Approximate weighting (%) | **20** | **20** | **20** | **20** | **10** | **10** |
| A | 1. An original research question is formulated independently and without ambiguity and is clearly linked to the investigation of a significant computing problem area.<br>2. A persuasive demonstration of the nature and significance of the problem is presented articulately. | 1. A comprehensive and thorough literature review is presented which includes relevant principles, concepts and research performed in the area of the study.<br>2. The literature review is systematic, articulate and ordered in a clear and coherent fashion. Clear and correctly formatted referencing is used throughout. | 1. A complete and precise system requirements specification is formulated from an articulate synthesis of a full range of functional and non-functional requirements.<br>2. A significant and high quality artefact has been created.<br>3. A systematic and coherent approach to designing an artefact to address a specification is documented. | 1. Primary data is gathered systematically.<br>2. The findings are clearly described and subject to close critical analysis. The analytical method used is clearly demonstrated to be appropriate.<br>3. A coherent interpretation based on a synthesis of a range of results leads to a reasoned and well-articulated critical conclusion. | 1. An articulate and critical evaluation of the artefact is undertaken independently and with full consideration of the specification.<br>2. The evaluation is clearly presented within the context of the research objectives, with clear outcomes demonstrated for each objective.<br>3. A detailed and thorough critical reflection on the effectiveness of the artefact in addressing the project objectives is clearly demonstrated. | 1. An articulate and persuasive argument for the nature and value of the contribution of the project outcomes is made in the context of related literature and theory.<br>2. An assessment of the management and planning of the project is present in the form of a critical analysis of the initial project plan with the actual project life cycle. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | 3. Recognition of potential contribution to the broader context and theory of the subject discipline (degree pathway) is evident. | 3. A full range of contemporary and relevant sources is used. Sources largely consist of high quality and reliable sources such as peer reviewed academic articles.<br>4. The literature review demonstrates an ability to synthesise available evidence and to evaluate conflicting interpretations to reach a critical, independent, resolution. | 4. The development process of an original artefact has been clearly and thoroughly documented.<br>5. A systematic and coherent justification for the design of the system has been provided. This will include a critical and persuasive argument demonstrating the value of formulating a detailed system requirements specification in the context of the proposed design.<br>6. Challenges for the design and implementation of the artefact are acknowledged and addressed. | 4. Specific limitations of the data gathering and analytic approach are acknowledged and addressed. | 4. Limitations of the design-build-test process and the final artefact are acknowledged and addressed through critical reflection.<br>5. A clear and insightful account of how future work might improve or build upon this process and product is provided. | 3. An informed and honest personal reflection of progress on the project overall with key challenges identified and potential improvements outlined. |
| B | 1. A research question is formulated without ambiguity and is clearly linked to the investigation of a significant computing problem area.<br>2. A clear demonstration of the nature and significance of the problem is presented effectively. | 1. A comprehensive literature review is presented which includes relevant principles, concepts and research performed in the area of the study. | 1. A system requirements specification is formulated coherently. A range of functional and non-functional requirements is defined<br>2. A significant artefact has been created.<br>3. The design of the artefact to address a specification is clearly documented. | 1. Primary data is gathered.<br>2. The findings are clearly described and subject to close analysis. The analytical method used is demonstrated to be appropriate.<br>3. A coherent interpretation of the findings leads to a reasoned critical conclusion. | 1. A systematic evaluation of the artefact is undertaken with full consideration of the specification.<br>2. The evaluation is presented within the context of the research objectives, with clear outcomes demonstrated for each objective. | 1. A coherent argument for the nature and value of the contribution of the project outcomes is made.<br>2. An assessment of the management and planning of the project is present in the form of a critical comparison of the initial project plan with the actual project life cycle. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | 3. Recognition of potential contribution to the subject discipline (chosen degree pathway) is evident. | 2. The literature review is well structured and presented in a clear and coherent fashion. Clear and correctly formatted referencing is used.<br>3. A range of contemporary and relevant sources is used. Sources largely consist of high quality and reliable sources such as peer reviewed academic articles.<br>4. The literature review demonstrates an ability to synthesise available evidence to reach a critical resolution. | 4. The development process of an original artefact has been documented fully.<br>5. A critical justification for the design of the system has been provided.<br>6. Some challenges and limitations of the design and implementation of the artefact are acknowledged and addressed. | 4. Some limitations of the data gathering and/or analytic approach are acknowledged. | 3. A critical reflection on the effectiveness of the artefact in addressing the project objectives is clearly demonstrated.<br>4. Some limitations of the design-build-test process and the final artefact are acknowledged and addressed through reflection.<br>5. Some account of how future work might improve or build upon this process and product is provided. | 3. An informed and honest personal reflection of progress on the project overall with some key challenges identified. |
| C | 1. A research question is formulated and is clearly linked to the investigation of a significant computing problem area.<br>2. A clear demonstration of the nature and significance of the problem is presented appropriately. | 1. A literature review is presented which covers a range of relevant topics.<br>2. The literature review is presented in a clear and coherent fashion. Format of referencing is used correctly is most cases. | 1. A system requirements specification is clearly defined and includes some functional and non-functional requirements.<br>2. A functional artefact has been created.<br>3. The design of the artefact to address a specification is evident. | 1. Primary data is gathered.<br>2. The solution or findings are described and subject to analysis.<br>3. An interpretation of the findings leads to a reasoned conclusion.<br>4. An attempt to state some limitations of the data gathering and/or analytic approach | 1. A comprehensive evaluation of the artefact is undertaken with consideration of the specification and research objectives.<br>2. A conclusion is reached regarding the evaluation that addresses most requirements. | 1. An argument for the nature and value of the contribution of the project outcomes is made. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | 3. Relevance to the subject discipline (chosen degree pathway) is strongly evident. | 3. A range of relevant sources is used including peer reviewed academic articles.<br>4. The literature review demonstrates an ability to collate a variety of theories or opinion and present an informed argument. | 4. Documentation of the development process of an artefact is evident but may be lacking some details.<br>5. Some justification for the design of the system has been provided.<br>6. No assessment of limitations of the process. | | 3. Some reflection on the system specification, design and development process is evident.<br>4. Some key challenges and limitations are identified and discussed.<br>5. No indication of how the work may be developed. | 2. An assessment of the management and planning of the project is present in the form of a systematic comparison of the initial project plan with the actual project life cycle.<br>3. An informed and honest personal reflection of progress is discussed. |
| D | 1. A research question is formulated and is aligned with a computing problem.<br>2. A demonstration of the nature of the problem is presented.<br>3. Relevance to the subject discipline (chosen degree pathway) is evident. | 1,3. A literature review is presented which includes some relevant sources.<br>2. Some attempt to format references correctly is evident.<br>4. The literature review demonstrates an ability to collate and present a variety of theories or opinions evident from within the related literature. | 1. An attempt to formulate a requirements specification is evident but may not be complete.<br>2. A limited but functional artefact has been created.<br>3. Some attempt to design the artefact with consideration of the specification is evident.<br>4. Some documentation of the development process of an artefact is evident but may be incomplete.<br>5,6. No justification for the design or consideration of the limitations of the process. | 1. Some primary data is gathered.<br>2. An attempt to describe and analyse the primary data is demonstrated.<br>3. An interpretation of the findings leads to a conclusion.<br>4. No attempt to consider limitations of the data gathering and analysis | 1. An evaluation of the artefact is undertaken with some consideration of the specification and research objectives.<br>2. A conclusion is reached regarding the evaluation that addresses some requirements.<br>3. An attempt to reflect on the system specification, design and development process is evident but may be lacking in detail.<br>4. No consideration of limitations of the design-build-test process or the resulting artefact | 1. An attempt to define the contribution of the project is evident.<br>2. A coherent description of the management and planning of the project is present.<br>3. A reflection of progress is discussed. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | 5. No indication of how the work may be developed. | |
| Narrow Fail (E) | 1,2,3. A problem area is defined but a specific research questions is lacking. | 1,2,3. A literature review is presented which includes a restricted range of sources.<br>4. The literature review demonstrates a limited ability to collate and present a variety of theories or opinions evident from within the related literature. Recapitulation of original sources may contain factual errors. | 1. Limited attempt to formulate a requirements specification.<br>2. An attempt to create an artefact is evident but it may not be fully functional.<br>3. Limited attempt to design the artefact with consideration of the specification is evident.<br>4. Documentation of the development process is largely incomplete.<br>5,6. No justification for the design or consideration of the limitations of the process. | 1. Limited primary data is gathered.<br>2. Limited and unsubstantiated attempt to describe and analyse the primary data is evident.<br>3. A conclusion based on the analysis of primary data is evident.<br>4. No attempt to consider limitations of the data gathering and analysis | 1,2,3. An attempt to evaluate the artefact is evident but may be incomplete or lacking consideration of the specification and research objectives.<br>4. No consideration of limitations of the design-build-test process or the resulting artefact<br>5. No indication of how the work may be developed. | 1. There is limited attempt to define the contribution of the project.<br>2. A description of the management and planning of the project is present.<br>3. No reflection of personal progress. |
| Clear Fail (F-G) | 1,2,3. A problem area is not clearly defined.<br>**Symptoms may include:**<br>• No research question or defined 'deliverable'<br>• No purpose for the work - who will benefit? | 1,2,3,4. No literature review is evident or it lacks sufficient sources of appropriate quality.<br>**Symptoms may include:**<br>• Only a small area covered<br>• Depth very restricted<br>• Sources used are mainly web sites | 1. No attempt to formulate a requirements specification.<br>2,3,4,5,6. No serious attempt to design and/or implement an artefact is evident.<br>**Symptoms may include:**<br>• No requirements specification<br>• Missing, trivial or non-functional artefact | 1,2. No attempt to gather and/or describe primary data is evident.<br>3,4. No conclusion based on the analysis of primary data is provided.<br>**Symptoms may include:**<br>• Primary data is absent or trivial | 1,2,3. No evaluation of the artefact is evident.<br>4,5. No attempt to assess limitations or potential for development.<br>**Symptoms may include:**<br>• No evidence of testing of the artefact (unit testing, system testing, user testing) | 1. There is no attempt to define the contribution of the project.<br>2. No description of the management and planning of the project is present.<br>3. No personal reflection.<br>**Symptoms may include:** |

| | | | | | | |
|---|---|---|---|---|---|---|
| | • It is not clear what is to be done (at the start) and/or what has been done (at the end)<br>• No links to Computing degree | • No obvious structure<br>• Information from sources is presented, but not evaluated<br>• No 'compare and contrast' between sources | • Very limited design documentation<br>• No explanation of or justification for the development process and tools used<br>• No evaluation of the design and development process | • Treatment of data is descriptive, and not analytical<br>• Conclusions are absent or trivial or unjustified<br>• No evaluation of the data gathering and processing - how could they have been improved? | • No comparison of the artefact with the design specification<br>• No evaluation of the design-build process- how might it have been improved?<br>• No suggestions for improving or developing the artefact | • No evaluation of the project outcomes against the objectives - were they achieved?<br>• No evidence of planning, or evaluation of actual progress of project against the plan<br>• No personal action plan: what have I learnt? What went wrong? Why? How am I going to do it better next time? |

## Ethics Procedures

Although not part of the explicitly assessed work in a Computing Project, students are expected to engage with the University's ethics approval process. Failure to do so constitutes academic misconduct, and carries various penalties.

| **Ethics Process:** Supervisors please insert a tick (✓) in the coloured box next to the appropriate description of the student's ethics engagement.<br>If you place a tick in any of the red boxes, the grade for the project should be entered as 'SM' (Suspected Misconduct), and the Academic Integrity Tutor should be informed. | |
|---|---|
| Ethics process completed, and project approved. All changes (e.g. modified questionnaires) also approved. | |
| Ethics process **partly completed**, low-risk project (not involving human participants) | |
| Ethics process **partly completed**, high-risk project (human participants) | |
| No engagement with ethics process, low-risk project (not involving human participants) | |
| No engagement with ethics process, high-risk project (human participants) | |

| | | | |
|---|---|---|---|
| | If the ethics process was only **partly completed**, please indicate the extent of the student's engagement:<br>• what was done<br>• what more should have been done | | |
| **Overall Comments:** | | | |
| **Recommendation for future assignments:** | | | |
| **Employability & Engagement:** | | | |
| **Agreed Assignment Grade:** | **First Marker:** | **Second Marker*:*** | **Third Marker:**<br>*if required* |

**RESULTS ARE PROVISIONAL UNTIL AGREED BY THE BOARD OF EXAMINERS**