CI601 The Computing Project

Effects of Simplicity on Simulator Realism and Usability

By:	**E.R. Walker**
Student ID:	**1780 2815**
Course:	**BSc Computer Science for Games**
Year:	**2020 / 2021**
Level:	**6**
Supervisor:	**Dr A. Baimagambetov**
2nd Reader:	**Dr S. Fallakhair**

CONTENTS

SECTION 1 – INTRODUCTION

The purpose of this project is to address the contemporary design and workflow of an orbital simulator with a simplified alternative: for some individuals (such as students or the scientifically uninclined), tools with a large scientific scope may be disconcerting to use and affect their ability to learn or understand the phenomena they simulate. To address this, the project seeks to provide a simplified and inherently limited approach towards providing the tools necessary for conducting simulations, such that beginner-friendliness is prioritized over sophistication and broadness.

Henceforth, a case study is conducted comprising of two software created for simulating orbital mechanics – one which is an industry-standard, field-tested tool, and one which is purpose-built as a deliverable for this project. Respectively, these simulators are:

- **GMAT – General Mission Analysis Tool, (complex),**
- **PESO – Physics Engine for Simulating Orbits (simplistic).**

The General Mission Analysis Tool is an industry standard software used by NASA to simulate orbital trajectories for satellites and rockets with existing astronomical bodies within the Solar System. It is open source under the **NASA Open-Source Agreement (v1.3)**, allowing it to be downloaded, modified, and redistributed by anyone world-wide (NASA Open-Source Agreement, 2013), and as of 2016 was maintained by 5 engineers, 5 developers and 1 tester (Lockney, 2016). The version of GMAT that this report covers is **R2020a**.

The aim of the project is to utilize aspects of Agile software development to yield a physics-based simulator (PESO) and observe how it fairs against GMAT. In doing so the project sets out to provide an answer to the following, original research question:

How does simplicity affect an orbital simulator's realism and usability?

and involves an undertaking to develop PESO as an application for simulating the effects of gravity on physical objects. Subsequently the project draws comparisons between PESO and GMAT with the intention of answering this.

There are two deliverables for this project:

1. **an evaluation of an independently developed scientific software, and**
2. **a comparison between it and an industry-standard scientific software.**

This comparison and the PESO application will allow the initial question to be answered. The evaluation will be conducted independently with black box testing, after which quantitative and qualitative comparisons will then be made between PESO and GMAT in the following two areas, each where a score will be provided to numerically examine their overall performance:

- **realism (realistic features of the simulators),**
- **usability (application workflow and appearance),**

where **Gestalt Principals** will be considered in areas where evaluation of usability is subjective, with the aim of applying universally accepted human psychological traits to determine which software is superior in this regard.

The time complexity of algorithms will only be considered for PESO as part of its evaluation as the project software.

SECTION 2 – APPROACH AND DESIGN

PESO: A Basic Orbital Simulator

One of the deliverables for this project is a basic simulator which allows a user to enter physical values for a collection of objects and observe how their trajectories are affected when subject to their collective gravitational forces and respective thrusts. This simulator is PESO, or Physics Engine for Simulating Orbits.

The scope of an industry standard tool, such as GMAT, may be overwhelming for newcomers to use with immediate effect due to their comparatively complicated workflow: some may only want to observe the effects of gravity from an elementary perspective via a tool which will take them minutes to learn, as opposed to advanced orbital mechanics from a tool which would take them weeks to learn effectively, and is perhaps beyond the scope of their requirements. PESO seeks to answer these requirements by taking the stance of a simplified and limited alternative to GMAT.

Developing such a software independently using an inherently simplistic and limited approach provides an uncontrived authenticity to the creation of a simulator of this nature: this is further reflected by PESO in the absence of sophisticated tools used throughout its development, such as **Unity3D** or the **Unreal Engine**, which are purpose-built for rapidly creating either games or simulators. Developing a software from basic components allows for a truly simplified and limited approach, where compromises in design are clearer.

Initial Design and Intended Features

PESO comprises of the following **four** main components (see Table 2.1, entries highlighted in grey represent only partial implementation in the final product):

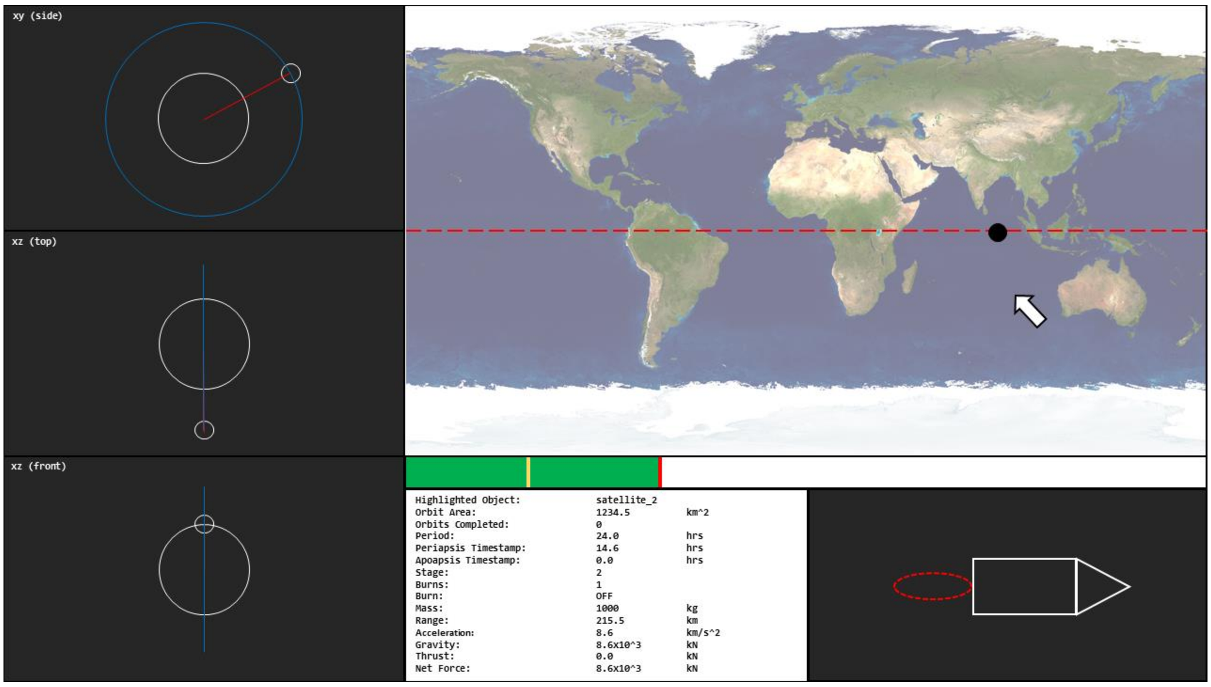
Component	**Outline**
Simulator with 2D graphics	This is the primary component of PESO where simplified depictions of physical objects are exposed to their local gravitational forces. Users may observe these interactions using a combination of x, y, and z axis in orthographic projection.
Data HUD	Alongside simulations must be a section of screen space used for logging the physical parameters of any one object to the screen. Users may select a particular object present in a simulation by highlighting it using the console.
File Subsystem	To add continuity to users' simulations, they must be provided with the option to load predefined parameters into a PESO simulation and save the results for collation. These parameters must be saved to PESO project files which the user can read or write via the console.
Console-based UI	The primary means of communicating with PESO will be with a console that appears on start-up. Users may provide commands registered by PESO as a means of initializing and interacting with simulations.

Table 2.1: Crucial features which PESO is, ideally, expected to have.

PESO's workflow is designed to prioritize user-friendliness so that it may be used quickly and intuitively, comprising of the following three steps:

1. **initialize objects,**
2. **execute simulation,**
3. **review results.**

Additional features include an animation subsystem to enable users to view simulations at any point over a set timeline. A viewport providing closeups of highlighted objects, and another viewport providing views of their orbits as seen from a topological ground track were also considered. Figure 2.1 demonstrates what PESO was expected to look like post-development:



```
Highlighted Object:       satellite_2
Orbit Area:               1234.5       km^2
Orbits Completed:         0
Period:                   24.0         hrs
Periapsis Timestamp:      14.6         hrs
Apoapsis Timestamp:       0.0          hrs
Stage:                    2
Burns:                    1
Burn:                     OFF
Mass:                     1000         kg
Range:                    215.5        km
Acceleration:             8.6          km/s^2
Gravity:                  8.6x10^3     kN
Thrust:                   0.0          kN
Net Force:                8.6x10^3     kN
```

Figure 2.1: Concept design of PESO. Visible are the orthographic viewports (left), the orbital ground tracking map (top right), the simulation timeline below it, the data HUD (bottom) and the satellite closeup (bottom right) (Walker, 2020).

Due to time constraints, simplifications had to be made to allow for a finished prototype for PESO prior to its scheduled development deadline.

Ethics and Legality

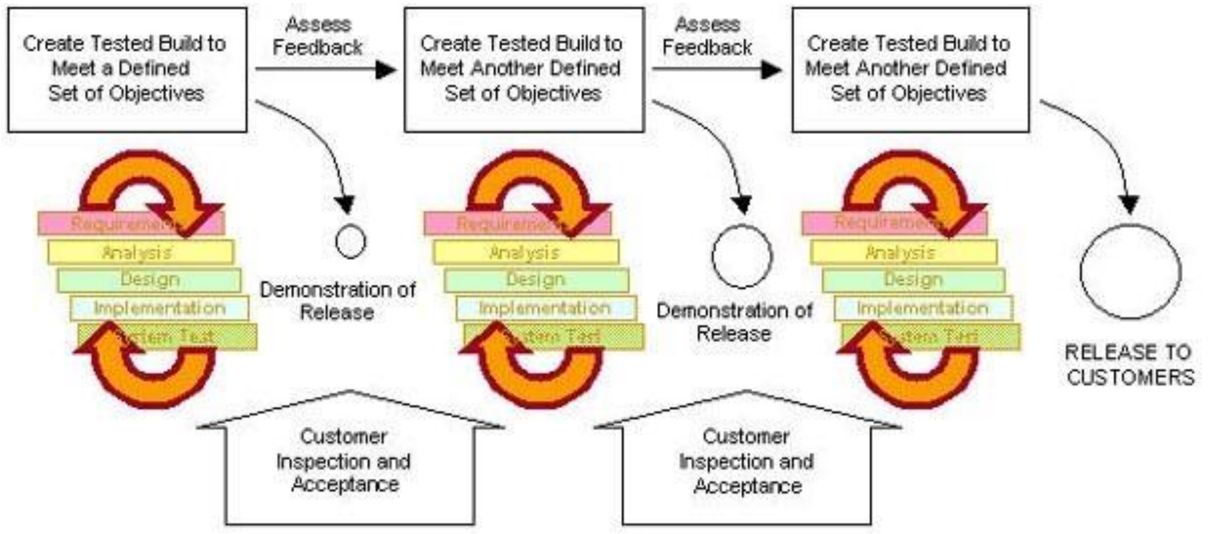
User evaluation for PESO was considered as a deliverable to add an empirical component to its comparison with GMAT, although it was later excluded from the project: an ethics checklist was also submitted. After completing a consent form, participants would have been provided with instructions on how to setup a simulation using GMAT, and then with PESO, subsequently being met with a questionnaire asking them to describe their experience in using both software. This has been replaced with independent, quantitative, and qualitative evaluation to ensure that the project can be completed in good time.

Management Technique

Agile, **iterative development**, and **Kanban** methodologies were used throughout this project's lifespan.

Agile incites developers to provide a working product at the end of each iteration, to adapt to simplifications or outright amendments to its design, and to adjust expectations accordingly. Agile inherently involves gaining user feedback by providing clients with a testable product throughout a its development, such that feedback gained can be applied to ensure that developers are focused toward the client's requirements.

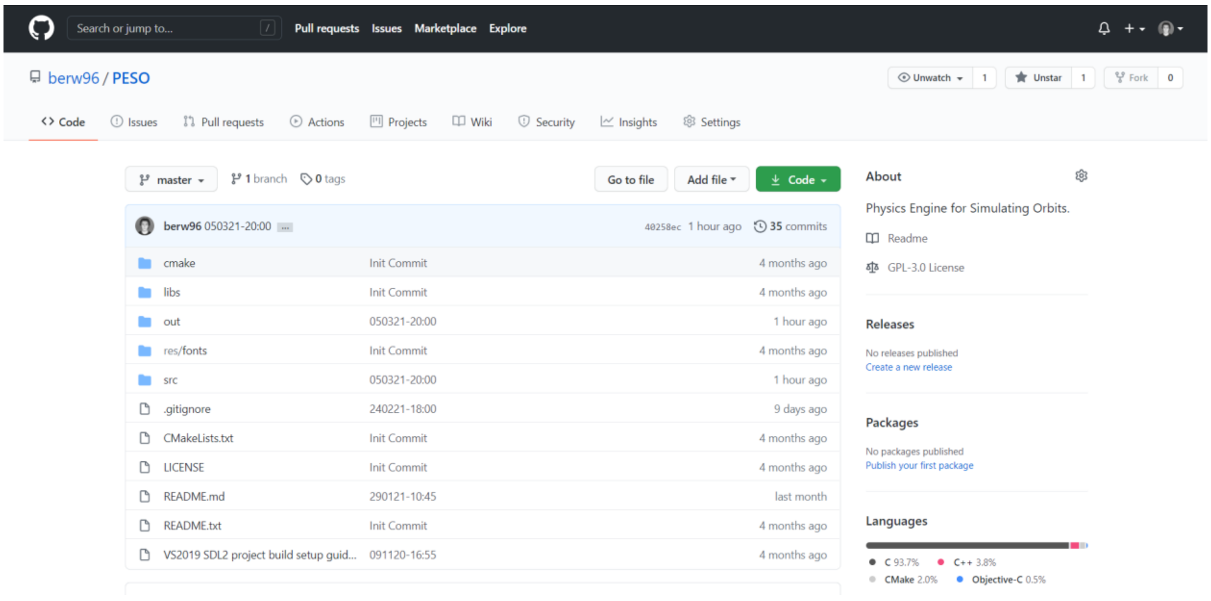
The iterative development approach focuses on the incremental addition and testing of features in a piece of software under a tempered and realistic approach to their viability with the available time and resources (see Figure 2.2).



Create Tested Build to Meet a Defined Set of Objectives
Assess Feedback
Create Tested Build to Meet Another Defined Set of Objectives
Assess Feedback
Create Tested Build to Meet Another Defined Set of Objectives
Requirements
Analysis
Design
Implementation
System Test
Demonstration of Release
Demonstration of Release
RELEASE TO CUSTOMERS
Customer Inspection and Acceptance
Customer Inspection and Acceptance

Figure 2.2: Diagram depicting the iterative development approach from the perspective of the project manager, and how features are presented to clients (in this case, the project's supervisor, second reader, and the School of Computing, Engineering and Mathematics) (IBM, 2005).

To manage the iterative development approach for PESO, **GitHub** version control was used: additions and updates to PESO's source code and resources were committed and pushed to GitHub (see Figure 2.3) using the **GitHub Desktop** application.



berw96 050321-20:00 … 40258ec 1 hour ago 35 commits

cmake	Init Commit	4 months ago
libs	Init Commit	4 months ago
out	050321-20:00	1 hour ago
res/fonts	Init Commit	4 months ago
src	050321-20:00	1 hour ago
.gitignore	240221-18:00	9 days ago
CMakeLists.txt	Init Commit	4 months ago
LICENSE	Init Commit	4 months ago
README.md	290121-10:45	last month
README.txt	Init Commit	4 months ago
VS2019 SDL2 project build setup guid...	091120-16:55	4 months ago

About

Physics Engine for Simulating Orbits.

Readme

GPL-3.0 License

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

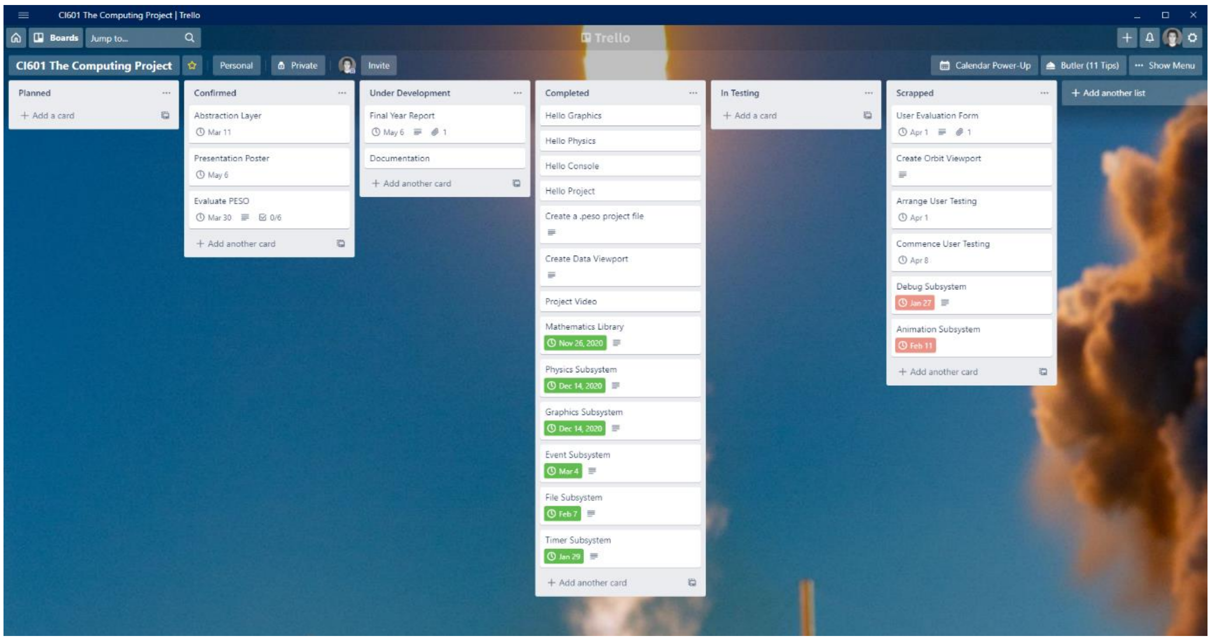
Languages

● C 93.7% ● C++ 3.8% ● CMake 2.0% ● Objective-C 0.5%

Figure 2.3: Screenshot of the GitHub directory used to provide version control during PESO's iterative development (Walker, 2021).

Weekly meetings with the project supervisor (see Appendix A) were conducted as part of this approach to ensure that development goals, expectations, priorities, and progress were adjusted and remained at viable levels, such that the project could be completed in time for the agreed deadline.

Kanban focuses on tracking outstanding tasks and timekeeping with a ‘to-do list’ approach: this was implemented using **Trello**, a popular platform for Kanban (see Figure 2.4). However, throughout PESO’s development basic to-do lists were also employed to accommodate the Kanban approach.



Planned
- Add a card

Confirmed
- Abstraction Layer — Mar 11
- Presentation Poster — May 6
- Evaluate PESO — Mar 30 — 0/6
- Add another card

Under Development
- Final Year Report — May 6 — 1
- Documentation
- Add another card

Completed
- Hello Graphics
- Hello Physics
- Hello Console
- Hello Project
- Create a .peso project file
- Create Data Viewport
- Project Video
- Mathematics Library — Nov 26, 2020
- Physics Subsystem — Dec 14, 2020
- Graphics Subsystem — Dec 14, 2020
- Event Subsystem — Mar 4
- File Subsystem — Feb 7
- Timer Subsystem — Jan 29
- Add another card

In Testing
- Add a card

Scrapped
- User Evaluation Form — Apr 1 — 1
- Create Orbit Viewport
- Arrange User Testing — Apr 1
- Commence User Testing — Apr 8
- Debug Subsystem — Jan 27
- Animation Subsystem — Feb 11
- Add another card

Figure 2.4: Screenshot of the Trello board used to keep track of PESO’s development. Columns representing different stages of development, from left to right these are: Planned, Confirmed, Under Development, Completed, In Testing, and Scrapped (Walker, 2021).

XCube2D: Referenced Software

PESO is developed using **XCube2D** as a reference to reduce its development time.

XCube2D was created under the **GPL (v3.0)** (GNU General Public Licence, 2007) and is a demonstrative piece of software created for the **CI517 Game Engine Fundamentals** module (Baimagambetov, 2020) taught at the University of Brighton during Semester 2 of the 19/20 academic year. The application comprises of a series of subsystems which implement physics, objects, graphics, input, debugging tools, timing, and basic geometry.

Programming Language, Coding IDE, and Graphics Library

The scripting language selected for developing PESO is **C++11**, for its object-oriented focus and one-to-one usability with **SDL**. Development was conducted in **Windows 10** using **Visual Studio (Community Edition) 2019**, chosen for its testing and debugging tools.

The **Simple Direct Media Layer** (SDL) is a graphics library written in the C programming language which may be used under the **ZLib Licence** (SDL, 2021) for high-level access to rudimentary **OpenGL** and **DirectX** features. SDL was chosen as opposed to these alternatives for its simplicity, and PESO makes heavy use of it for its graphics and event handling.

The reason for using C++11 and SDL arises partially from the software on which PESO is based (XCube2D): it was developed using these tools. Hence, this offered an opportunity to recreate a variant purpose-built for simulating orbital mechanics, as opposed to general gameplay.

Functionality and Formulae

PESO is a physics engine and therefore must perform well as one. The linear and rotational mechanics required to simulate gravity between multiple objects are implemented through interpretations of fundamental equations in **Newtonian Mechanics**. These equations all represent a mechanic which PESO applies.

Linear mechanics are employed to simulate basic gravitation, where rotational mechanics are employed to simulate spacecraft pitch, yaw and roll, and axial rotation for celestial bodies (e.g., such as planets spinning on their axis).

As an application which is designed to simulate gravitational phenomena, it requires a suitable equation for determining the gravitational force experienced by n objects: this equation is **Newton's Law of Universal Gravitation**. All the equations used in PESO's physics subsystem are provided in Table 2.2, where equations defined but unused in PESO are shaded in grey:

Equation	Terms	Name/Usage

$a = {}^{F}/_{m}$	a = Acceleration
	F = Force applied
	m = Object′s net mass
		Newton's Second Law of Physics (rearranged for calculating an object's acceleration).

$v = u + at$	v = Resulting velocity
	u = Initial velocity
	a = Acceleration
	t = Time elapsed
		Calculates the resulting velocity of an object after a period of acceleration (time in PESO is implicitly applied, and thus cancels out from this equation).

$p = mv$	p = Momentum
	m = Object′s net mass
	v = Current velocity
		Calculates the instantaneous linear momentum possessed by an object of mass $\boldsymbol{m}$ travelling at a velocity $\boldsymbol{v}$.

$L = I\omega$	L = Angular Momentum
	I = Object′s inertia
	ω = Angular velocity
		Calculates the instantaneous angular momentum possessed by an object with inertia $\boldsymbol{I}$, rotating on its axis at velocity $\boldsymbol{\omega}$.

$F_g = (m_1 m_2 G)/r^2$	$\mathrm{F_g}$ = Gravitational force
	m = Object′s mass
	G = Const of gravitation
	r = Range or displacement
		Newton's Law of Universal Gravitation. Calculates the gravitational force experienced between two bodies of mass over a displacement.

$F_{NET} = \sum_{i=0}^{n} F_i$	$\mathrm{F_{NET}}$ = Net force applied
	n = No. of forces
	i = Force′s ordinal index
		Calculates the sum of all forces applied to an object (particularly useful for determining its overall acceleration).

$s = \sqrt{s_x^2 + s_y^2 + s_z^2}$	s = Resultant displacement $\mathrm{s_x}$ = x displacement $\mathrm{s_y}$ = y displacement $\mathrm{s_z}$ = z displacement	Calculates the displacement of an object in a three-dimensional Cartesian plane. Used for determining the range between two gravitational bodies.

$I = mr^2$	I = Inertia
	m = Object′s net mass
	r = Object′s radius
		Calculates the inertia of an object based on its mass and radius.

$\alpha = {}^{\tau}/_{I} = {}^{\tau}/_{mr^2}$	α = Angular acceleration
	τ = Torque or angular force
	I = Inertia
	m = Object′s net mass
	r = Object′s radius
		Calculates the angular acceleration of an object based on the torque applied and its inertia.

$T = 2\pi\sqrt{{}^{r^3}/_{Gm}}$	T = Orbital period
	r = Range or displacement
	G = Const of gravitation
	m = Object′s net mass
		Calculates the orbital period of a satellite orbiting a body of mass $\boldsymbol{m}$.

$v = \sqrt{Gm/r}$	$\mathrm{v} = \text{Required velocity}$ $\mathrm{G} = \text{Const of gravitation}$ $\mathrm{m} = \text{Object's net mass}$ $\mathrm{r} = \text{Range or displacement}$	Calculates the velocity required for an object to remain in its present orbit (assuming the absence of atmospheric friction and the object is travelling tangentially about its focus).
$r_{cm} = \sum_{i=1}^{n} m_i r_i \Big/ \sum_{i=1}^{n} m_i$	$\mathrm{r_{cm}} = \text{COM on axis } \mathbf{r}$ $\mathrm{n} = \text{Number of objects}$ $\mathrm{i} = \text{Object's ordinal index}$ $\mathrm{m} = \text{Object's mass}$ $\mathrm{r} = \text{Object's position on axis } \mathbf{r}$	Calculates the centre of mass (COM) of a system of masses (e.g., a parent object with children) for a given axis **r**.

Table 2.2: Equations used in PESO, the meaning of their algebraic terms, and their applications. Each equation is associated with an algorithm that occurs as a function in PESO.

The other half of PESO's mechanics comprises of its representations of physical objects, namely the physical parameters they contain. Physical objects possess mutable parameters which serve as the basis for their dynamic behaviour. In Table 2.3 is a list of parameters which apply to objects in PESO. They are designed in a three-dimensional context: the user can simulate both linear and rotational dynamics on all axis of the Cartesian plane.

Parameter
Mass
Position
Radius
Speed
Velocity
Acceleration
Rotation
Thrust
Gravitational Force
Momentum
Net Force
Torque
Inertia
Orbital Period

Table 2.3: The physical parameters associated with objects used in PESO.

SECTION 3 – PLANNING

Projection and Amendments

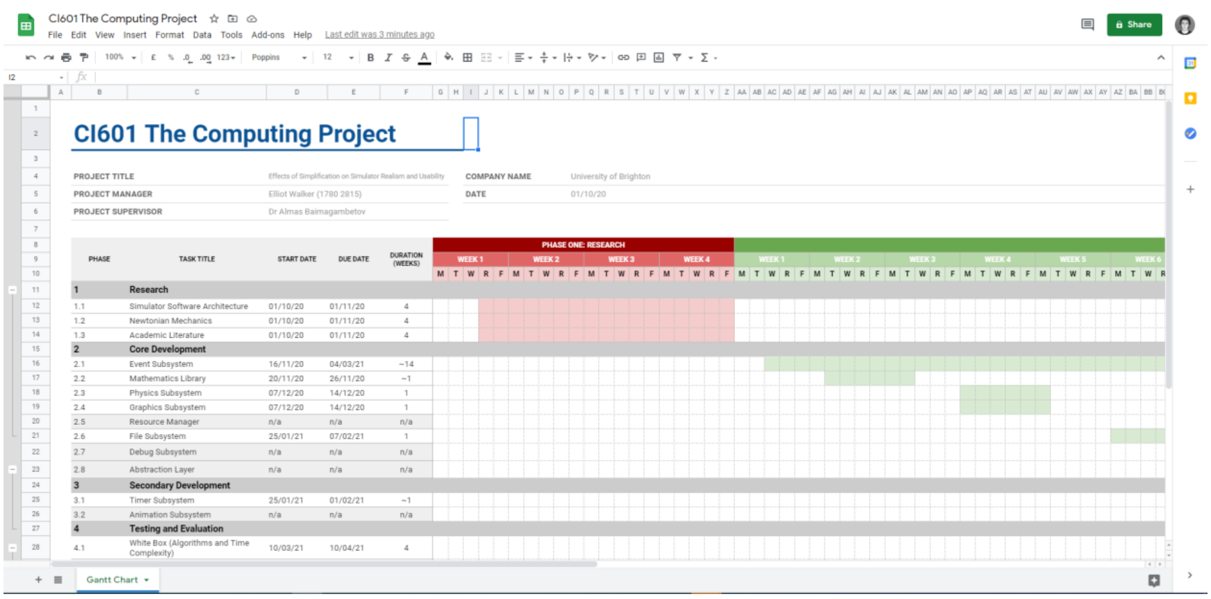
The projected[1] order (and timeline) in which PESO's individual systems would have been developed throughout the duration of the project is detailed in Table 3.1, where Figure 3.1 shows this in the form of a Gantt Chart:

Phase	Start Date	Increment Duration	Total Duration	Contents	Priority
Research Period[2]	Oct 1st 2020	~1 month		Simulation Software Architecture	Very High
				Newtonian Mechanics	
				Academic Literature	
Core Development	Nov 16th 2020	1 month	~3 months	Event Subsystem	
	Nov 20th 2020	1 week		Mathematics Library	
	Nov 23rd 2020	2 weeks		Physics Subsystem	
	Nov 30th 2020	1 week		Graphics Subsystem	
	Dec 7th 2020	2 weeks		Resource Manager	
				File Subsystem	
	Jan 25th 2021	1 week		Debug Subsystem	
	Jan 28th 2021			Abstraction Layer	
Secondary Development	Feb 8th 2021	1 day	~2 weeks	Timer Subsystem	High
	Feb 10th 2021	2 weeks		Animation Subsystem	
Testing and Evaluation	Mar 10th 2021	~1 month		White Box (Algorithms and Time Complexity)	
				Black Box	
				Debugging	
				Realism	
				Usability	
Demonstration	Apr 17th 2021	~1 week		PESO Demo	Medium
	Apr 20th 2021			Presentation / Slideshow	
	Apr 23rd 2021			YouTube Video	Low

Table 3.1: The projected schedule for the project from initial research to its demonstration.

[1] As a projection, this is not explicitly accurate in predicting the details of the ensuing development process. It is an estimation made with the expectations held during the planning phase of the project.

[2] Research into the software architecture of simulators, and the physics of orbital mechanics were conducted spontaneously throughout development where required to reinforce PESO's realism.



CI601 The Computing Project

PROJECT TITLE	Effects of Simplification on Simulator Realism and Usability	COMPANY NAME	University of Brighton	
PROJECT MANAGER	Elliot Walker (1780 2815)	DATE	01/10/20	
PROJECT SUPERVISOR	Dr Almas Baimagambetov			

PHASE	TASK TITLE	START DATE	DUE DATE	DURATION (WEEKS)
1	**Research**			
1.1	Simulator Software Architecture	01/10/20	01/11/20	4
1.2	Newtonian Mechanics	01/10/20	01/11/20	4
1.3	Academic Literature	01/10/20	01/11/20	4
2	**Core Development**			
2.1	Event Subsystem	16/11/20	04/03/21	~14
2.2	Mathematics Library	20/11/20	26/11/20	~1
2.3	Physics Subsystem	07/12/20	14/12/20	1
2.4	Graphics Subsystem	07/12/20	14/12/20	1
2.5	Resource Manager	n/a	n/a	n/a
2.6	File Subsystem	25/01/21	07/02/21	1
2.7	Debug Subsystem	n/a	n/a	n/a
2.8	Abstraction Layer	n/a	n/a	n/a
3	**Secondary Development**			
3.1	Timer Subsystem	25/01/21	01/02/21	~1
3.2	Animation Subsystem	n/a	n/a	n/a
4	**Testing and Evaluation**			
4.1	White Box (Algorithms and Time Complexity)	10/03/21	10/04/21	4

PHASE ONE: RESEARCH

Figure 3.1: The project's Gantt Chart, comprising of phases 1 thru 4. A link to the full Gannt Chart is provided in Appendix B (Walker, 2021).

To reflect amendments to PESO's design made to reduce aspects of the workload unnecessary to producing its MVP, amendments were also made to its development schedule. Non-essential features were either delayed or discarded, where essential features were prioritized throughout its core development phase (December to March) wherein the scheduled development dates for features were altered, or outright discarded (see shaded entries in Table 3.2).

Component	Est. Start Date	Act. Start Date	Est. Finish Date	Act. Finish Date
Abstraction Layer	Jan 28th 2021	n/a	Feb 5th 2021	n/a
Animation Subsystem	Feb 10th 2021		Feb 24th 2021	
Debug Subsystem	Jan 25th 2021		Feb 1st 2021	
Event Subsystem	Nov 16th 2020		Mar 4th 2021	
File Subsystem	Dec 7th 2020	Jan 25th 2021	Feb 7th 2021	
Graphics Subsystem	Nov 30th 2020	Dec 7th 2020	Dec 14th 2020	Dec 12th 2020
Maths Library	Nov 20th 2020		Nov 26th 2020	Nov 21st 2020
Physics Subsystem	Nov 23rd 2020	Dec 7th 2020	Dec 14th 2020	Feb 2nd 2021
Resource Manager	Dec 7th 2020	n/a	Dec 21st 2020	n/a
Timer Subsystem	Feb 8th 2021	Jan 25th 2021	Feb 9th 2021	Feb 1st 2021

Table 3.2: Amendments made to PESO's development schedule: side-by-side comparisons of estimated and actual start and finish dates are provided. Entries labelled with n/a imply components which never made it to the development phase.

SECTION 4 – LITERATURE REVIEW

Simplicity in Educational Software

Research conducted to understand how the learning experiences of students benefits from abstractions presented by educational tools was detailed in the article **Students' ways of experiencing visual program simulation** (Sorva, et al., 2013), and involved a class of programming students being tasked with using **visual program simulation** (VPS), an educational tool providing graphical representations of rudimentary computer programs, in completing coding coursework allocated to them.

The results of this research found that the ability of a student to learn a subject (in this case, coding) is not definitively affected by the abstractions of a learning tool: the latter arising from the subjective premise of the research. The effects which simplifications or alternative formats have on an individual's learning experience, and thus the clarity provided by a tool used to facilitate it, are entirely dependent on their usage of the tool and their motives.

This research provided valuable insight to the effects of simplicity in simulators: its findings served as inspiration for the comparison of both PESO and GMAT, such that subjective aspects could be mitigated, and objective aspects could be highlighted.

Game Engine Architecture

A simulator is an interactive software used as a tool for modelling phenomena experienced in the real world. This implies that simulators are akin to video games, albeit with realistic, analytical undertones: their software architecture is identical, and PESO demonstrates this being directly based off XCube2D (Baimagambetov, 2020)**,** a game engine[3] built using identical tools.

Aside from XCube2D and the university module it accompanied, a cornerstone resource which underpins this architecture is the title **Game Engine Architecture (3rd ed.)**. In it, the author (Gregory, 2018) outlines the hierarchy of subsystems present in a game engine. With these resources it is entirely possible to create a gaming software architecture as a template and include real-world formulae for gravitational effects into it to create a basic orbital simulator.

Due to this parity in software architecture, this project has implications for video games which apply similar physics, such as **Kerbal Space Program** (Take-Two Interactive, Inc., 2018) .

University Modules

The CI517 Game Engine Fundamentals module (Baimagambetov, 2020) aided in the preparation for developing PESO with its coverage of game engine architecture via XCube2D, and its usage of both C++ and SDL. Another module of a similar nature, **CI628 Multiplayer Games Development** (Baimagambetov, 2020), taught (during the 20/21 academic year) further usage of SDL which greatly assisted in the development phase of PESO.

One module which was of particular importance to the evaluation of PESO's runtime efficiency was the **CI583 Data Structures and Operating Systems** module (Burton, 2019) run by **Dr Jim Burton** also during the 19/20 academic year.

Respectively, these modules contributed to the project by teaching the fundamentals of game engine development using C++ and SDL, and by teaching the concept of time complexity used for determining the number of operational steps exhibited by algorithms.

[3] A game engine is software which underlines and implements the fundamental features of a game, such that the logic and appearance of the latter can be developed using the former as a template.

Orbital Mechanics

In making an orbital simulator it is necessary to know the components and mechanics of an orbit. Generally, an orbit is an elliptical or oscillatory trajectory possessed by a body of mass in motion about another (a focus). The components of an orbit are defined as **Kepler's Elements** (Rutgers, 2021), which include:

• **Epoch**	The time required to complete a single orbital cycle, also known as the period of the orbit.
• **Inclination**	The angle between the equator of the gravitational body and its orbit.
• **Ascension**	The angular position at which an orbital body passes between the southern and northern hemispheres of the focus.
• **Eccentricity**	How elliptical the orbit is. Circular orbits possess zero eccentricity ($e = 0$), parabolic orbits $e = 1$, and hyperbolic orbits e > 1.
• **Mean Motion**	The average velocity of an orbiting body.
• **Anomaly**	The location of an orbiting body about the focus as a function of the angle between them.
• **Drag**	The aerodynamic resistance an orbiting body may experience if it is close enough to an atmosphere. Some drag may be caused by cosmic radiation, too.

In orbital mechanics the **Hohmann Transfer** is a fundamental manoeuvre performed to alter the radius of an orbit using thrust: it is the most economic but longest orbital transfer procedure (Bate, et al., 2020) comprising of three components: a small(er) orbit, a large(r) orbit, and a transfer orbit which requires a change in mean motion. The **orbital energy-invariance law** is used for determining the minimum change in velocity required to conduct a Hohmann Transfer:

$$v = \sqrt{Gm(\frac{2}{r} - \frac{1}{a})}$$

where v is the velocity of the orbiting body, r is the displacement between it and the focus, m is the body's mass, a is the semi-major axis (highest displacement) of the orbit, and G is the Universal Constant of Gravitation. $\Delta v = +ve$ produces a larger orbit, and $\Delta v = -ve$ produces a smaller orbit.

The research conducted into orbital mechanics and Keplerian Elements provided insight to the underlying functionality required for a software to simulate it: namely which realistic features constitute an ideal orbital simulator.

SECTION 5 – DEVELOPMENT

Application Loop

During runtime PESO operates via a while-loop which enables the user to provide input via keypresses and is executed based on the value of a **running** Boolean: the PESO application runs until this variable is set to false. Inside this loop is an if statement which is executed based on a **paused** Boolean and enables the physics subsystem to invoke behaviour on physical objects registered in the simulation.

The application loop comprises of the following three-step process:

• **input**	(query user input to application functions),
• **update**	(mutate runtime variables via said functions),
• **render**	(provide visual feedback to the user based on these variables).

While the application itself is running, the user only has the option to play their simulation or quit.

Input (PESO_events)

Input in PESO is console-based and enables user interaction through keypresses handled by the **PESO_events** subsystem. Prior to a simulation the user will be prompted to provide input which PESO can use to initialize their objects for the simulation. After providing all the input parameters the user will then be able to issue basic commands with keypresses.

The event subsystem is defined under the **PESO_Events** class, which makes use of the **SDL_Event** union[4] defined in the SDL.h library, and an enum type **Key** which streamlines the usage of the SDL event subsystem. Only keys which are represented by enumerator values are used in the subsystem, these include:

• **SPACE**	plays a simulation once it has been initialized,
• **ESC**	pauses the application, shuts down its subsystems, then exits it,
• **LAST**	avoids an out of bounds exception when checking the state of a key[5].

The PESO_Events class comprises of two member variables, one private function, and five public functions. The two variables are of type SDL_Event and bool, respectively: the SDL_Event variable stores the most recent keypress detected by the SDL event subsystem, where the bool variable is an array which stores key Booleans: both are used by PESO's event subsystem to determine which keys are being pressed by the user.

Console input is implemented with the event subsystem's **PESO_CreateObjectData** function, which takes an instance of **PESO_Object** as a parameter and returns an instance of **PESO_Data** which is used to initialize each object. Object data is provided by the user in the following order:

1. **name of satellite**	**(string),**
2. **mass of satellite**	**(double),**
3. **relative x-position to origin**	**(double),**
4. **relative y-position to origin**	**(double),**
5. **relative z-position to origin**	**(double),**
6. **x-rotation**	**(double),**
7. **y-rotation**	**(double),**
8. **z-rotation**	**(double),**
9. **x-thrust**	**(double),**
10. **y-thrust**	**(double),**
11. **z-thrust**	**(double),**

[4] A union is a type of C++ struct which holds the value of only one of its member variables at any given time.
[5] This key is not present on the user's keyboard.

12.	**x-torque**	**(double),**
13.	**y-torque**	**(double),**
14.	**z-torque**	**(double).**

All variables are mutable throughout the PESO simulation runtime, although the tags and masses of each object remain (implicitly) constant, as there are no operations which can be applied after initialization to mutate them.

Erroneous console input from the user is managed by the **PESO_CheckInput** and **PESO_ExtractDigitsFromInput** functions: the former scans for any irrelevant characters contained within the provided input string, and the latter extracts only the digits provided. Only digits, decimal points and negative signs are acknowledged when determining a value, with the rest being discarded. Both functions are called from within **PESO_CalculateValueFromDigits**, which returns a double corresponding to the input value. The user can omit digits from their input and the subsystem will return a default value of zero.

Using this subsystem, the user can specify the number of objects they want to be present in their simulation and specify the physical parameters for each one. Once specified, the physics engine iterates through a list containing these objects and initializes each one with the parameters provided by the user.

Maths Library (PESO_math)

For the geometric definitions used in PESO's graphics subsystem a series of mathematical definitions are employed to allow for basic two-dimensional geometries to be drawn to its viewports. Some geometries are defined in the SDL (e.g., **SDL_Rect, SDL_Point**), but a mathematics library streamlines their use and enables more complex geometry to be created (such as ellipses).

In PESO's mathematics library are two definitions, both taking the form of a struct:

• **Point2d**	for two-dimensional points (vertexes),
• **Line2i**	for two-dimensional line segments (edges).

Their naming conventions contain a digit and a character: the digit corresponds to the number of dimensions used (e.g., x and y, horizontal and vertical), where the character corresponds to the data type of those dimensions (e.g., **d** means **double**, **i** means **int**). There are constants defined for **Pi**, and the ratios used for converting between degrees and radians (e.g., Pi over 180 degrees, or the inverse). These are used with the geometric definitions above to draw **ellipses**, featuring properties of width and height. Only two dimensions are used in these structs due to PESO's graphics subsystem: while it provides a three-dimensional experience across three separate viewports, each perspective is inherently two-dimensional.

The Line2i struct for drawing line segments is unused in PESO due to the simplifications made to its design. This struct would have been used to visualise the radii of ellipses drawn using the Point2d struct to communicate the rotation of an object.

Physics (PESO_physics)

While PESO's graphics are fundamentally two-dimensional, its physics are entirely three-dimensional. PESO's physics subsystem is capable of both linear and angular mechanics, and it comprises of two classes and three structs:

• **Vector3d**	**(struct),**
• **PESO_Transform**	**(struct),**
• **PESO_Physics**	**(class),**
• **PESO_Object**	**(class),**
• **PESO_Data**	**(struct).**

Vector3d defines the three-dimensional (Cartesian) standard used by the physics subsystem: all physical values it uses are of the Vector3d type. As with the naming conventions used in the mathematics library, a digit and a character apply here.

The PESO_Transform struct comprises of two member variables of type Vector3d: position and rotation. This struct serves as an abstraction for defining the whereabouts and heading of any object in a PESO simulation. The PESO_Object class makes heavy use of the Vector3d struct in defining the linear and angular fields for each physical object, via the PESO_Data struct. This enables each field to adhere to three-dimensional Newtonian mechanics.

The member variables included in each instance of the PESO_Data class, their respective data types, and their definitions are provided in Table 5.1: only those which are applied in PESO's simulations are listed.

Name	**Type**	**Definition**
tag	std::string	The display name used by the PESO_Object instance.
transform	PESO_Transform	The cartesian transform of the object. Comprises of two Vector3d fields for position and rotation.
timestamp	time_t	The timestamp associated with the object's data.
centre	Vector3d	The Centrepoint of the object. May be used instead of centre of mass (COM).
netLinForce		The overall net force enacted on the object. Constituted by the sum of gravitational force and thrust.
gravForce		The gravitational force experienced by the object.
thrust		The thrust experienced by the object.
linSpeed		The linear speed of the object. Can be considered as the magnitude of its linear velocity.
linAcceleration		The linear acceleration of the object. This field increments the value of its linear speed based on the forces enacted upon it divided by its mass (as per Newton's 2nd Law).
linVelocity		The linear velocity of the object. This increments their position for every tick (frame) which the simulation is running.
linMomentum		The linear momentum of the object. Equal to the product of their mass and their velocity.
torque		The rotational force exerted on an object about its centre.
netAngForce		The net angular force applied to the object, equal to the torque applied to it.
angSpeed		The angular speed of the object. Can be considered as the magnitude of its angular velocity.
angAcceleration		The angular acceleration of the object. This field increments the value of its angular speed based on the rotational forces enacted upon it divided by its inertia (as per Newton's 2nd Law).
angVelocity		The angular velocity of the object. This increments its position for every tick (frame) which the simulation is running.
angMomentum		The angular momentum of the object. Equal to the product of its inertia and angular velocity.
mass	double	The content of matter associated with the object. Determines how the object accelerates under applied forces.
inertia		The object's resistance to rotational forces. Determines how the object accelerates angularly under rotational force (torque).
radius		The radius of the object.

Table 5.1: The member variables which constitute the PESO_Data struct: each one's name, data type, and purpose within the PESO API are provided.

Instances of PESO_Object comprise of two member variables: one to store an instance of type PESO_Data (objectData) containing its physical parameters, and one to store a collection of other instances of PESO_Object (children). Each object may possess other objects as components[6] (e.g., a rocket's final stage would be accompanied by its preceding stages as children).

The physics subsystem itself is defined in the PESO_Physics class and is PESO's core component. Like the PESO_Object class, it comprises of two member variables (both of type **std::vector**): one stores shared pointers to objects registered by the physics subsystem, and one stores object data provided by each object during runtime. Only objects which are registered with the physics subsystem may have mechanics applied to them, thus by default, declaring an instance of PESO_Object requires it to be immediately pushed to this list. This is managed automatically by the **PESO_SelectNumberOfObjects** function defined in the event subsystem: which utilizes the value provided by the user to add a particular number of objects to this list.

Linear and angular mechanics are applied using the following algorithm:

1. calculate the net force experienced by the object,
2. divide net force by object's mass (or inertia for rot. mechanics) to determine its acceleration,
3. increment object's speed by its acceleration,
4. assign value of object's speed to its velocity,
5. increment object's position/rotation with its velocity,
6. increment object's geometric attributes by its transform.

Any graphical representation of the object created by the graphics subsystem shares the same position as the object's centre: when an object moves via the mechanics applied to it, so does its graphical representation.

Graphics (PESO_graphics)

The definitions established in **PESO_math** are applied in the graphics subsystem in functions used to draw single points and line segments. The graphics subsystem contains a variety of member variables, all of which represent a crucial component of PESO's graphical API. Data types include:

• **SDL_Window***	stores the heap memory address of a viewport,
• **SDL_Renderer***	stores the heap memory address of its renderer,
• **SDL_Color**	stores an RGBA colour,
• **TTF_Font***	stores the heap memory address of a TrueType Font.

Collectively the graphics subsystem comprises of four viewports (and four renderers), five colours (four for general drawing and one for clearing the screen) and one font.[7] The viewports and renderers are used to achieve orthographic 3D: three-dimensional graphics may be achieved in one of two ways, using a perspective-based format where depth and field of view (FOV) are considered (as may be experienced in everyday life) with the application of a projection matrix, or alternatively using orthographic projection, by displaying a graphical object from a series of 2D viewpoints, each corresponding to a pair of Cartesian axis.

PESO's graphics opt for the orthographic approach, and use three viewports named **xyViewport**, **xzViewport** and **yzViewport** to display sideways, top-down, and frontal views (respectively) of all geometries which may be rendered to to them, and one viewport to display the telemetry of the initial object registered to the physics subsystem.

[6] While included in the PESO API, this is not a feature which the user has access to from the front-end.
[7] Multiple SDL viewports were implemented as opposed to one to give the user as much control over the layout and setup of the GUI, such that they may operate PESO over multiple monitors.

Output (PESO_filesys)

Object data registered to the physics subsystem throughout PESO's simulation runtime (up to the last second before it is shut down by the user) is printed to a text file which the user can view post-simulation. This document is generated if it does not already exist, and overwritten if it does exist.

The layout and contents of the object data are divided between non-physical data (e.g. the object's name tag), miscellaneous data (e.g. mass and inertia), linear data and angular data. Each set of object data is accompanied with a corresponding timestamp which informs the user when during the simulation runtime its contents were logged by the physics engine.

The PESO file subsystem makes use of a select group of I/O features found in the C Programming Language (which C++ extends), which is used here for its low(er)-level approach to file I/O. The **fprintf** function takes three parameters:

- **File pointer** (FILE*),
- **Format(s)** (const char* or char[]),
- **Variable contents** (const char* or char[]),

and writes the provided contents to the file pointed to by the file pointer, under the provided format. Formats can be chained using C's string concatenation syntax, which allows the following format for printing object data to the output text file:

Data Printing Format (Non-Numerical)
`fprintf(<file>, "<field_name>: %s", <field_value>.c_str);`
Data Printing Format (Numerical)
`fprintf(<file>, "<field_name>: %+Ls", std::to_string(<field_value>).c_str);`

Table 5.2: Non-numerical and numerical formats used for writing session data from instances of PESO_Data to the application's output file.

The placeholder **<file>** refers to the file pointer, **<field_name>** and **<field_value>** refer to the physical field's name and value, respectively. The **%+Ls** is a string formatting expression, in it are two flags (the plus sign and the capital **'L'**) and a type specifier (the **'s'**): the flags in these statements need not be applied if the data is non-numerical (e.g., a name tag or a timestamp). In order, these characters tell the fprintf function to print the contents with the appropriate sign (as negative or positive depending on the value) and print it as a **long double** for maximized accuracy.

The **std::to_string** function is part of C++, and is used here to streamline the conversion of numerical information into strings, which can then be used as **C-strings** (of type const char* or char[]) by invoking the **c_str** function.

Accessing the object data registered by the physics subsystem is achieved by iterating through the contents of its object list (provided that its size is greater than zero). Encapsulating all of this is the **PESO_FileManager** class, which contains a single FILE pointer and a function **PESO_WriteFile** which applies the fprintf function extensively, and accepts two parameters by reference[8]:

- **Session Data** (std::vector<PESO_Data>&),
- **User Satellite** (PESO_Object&).

[8] Pass by reference is used to pass large objects to functions without making copies that would otherwise bloat memory usage.

Timer (PESO_timer)

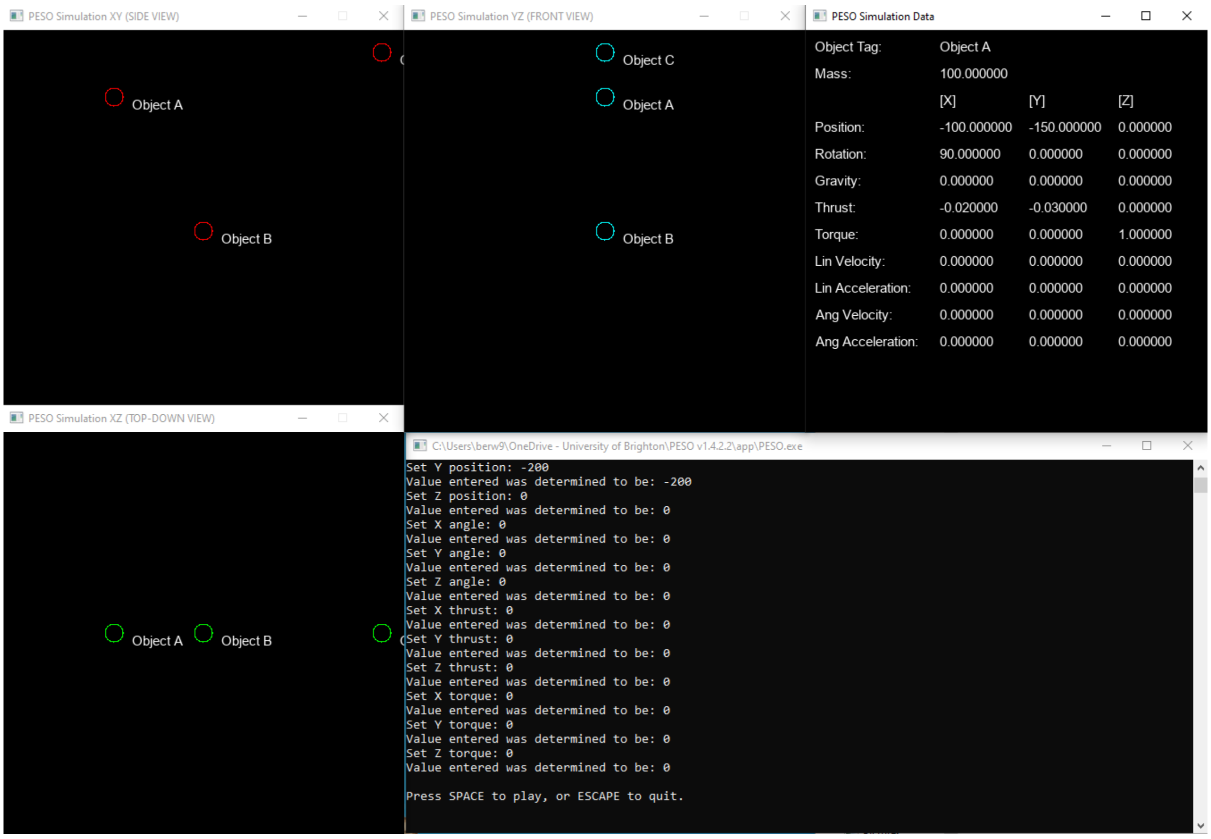
The timer subsystem plays an important role in how PESO keeps track of simulation data, specifically how frequently it updates the list of object data during runtime.

The timer subsystem employs both the **ctime.h** and **SDL.h** libraries for functionality: ctime.h is incorporated by the physics subsystem to retrieve the current date and time as read by the OS: this is used to inform the user when exactly object data has been logged by PESO. SDL.h is used for its **SDL_GetTicks** function, which measures how many milliseconds have elapsed independent of the user's CPU cycle frequency: this is used to detect when a single second (1000 milliseconds) has passed.

Altogether the timer subsystem logs object data associated with each object to a list once every second that the simulation is running. If the simulation is paused, the timer will stop running, where the ticks measured up to that point are preserved: this occurs during shutdown after the user presses ESC.

The End Product

The results of the development phase for PESO are illustrated in Figure 5.1 and demonstrated in the YouTube video linked in Appendix B:



PESO Simulation XY (SIDE VIEW)
Object A
Object B
PESO Simulation YZ (FRONT VIEW)
Object C
Object A
Object B
PESO Simulation Data
PESO Simulation XZ (TOP-DOWN VIEW)
Object A
Object B
C:\Users\berw9\OneDrive - University of Brighton\PESO v1.4.2.2\app\PESO.exe

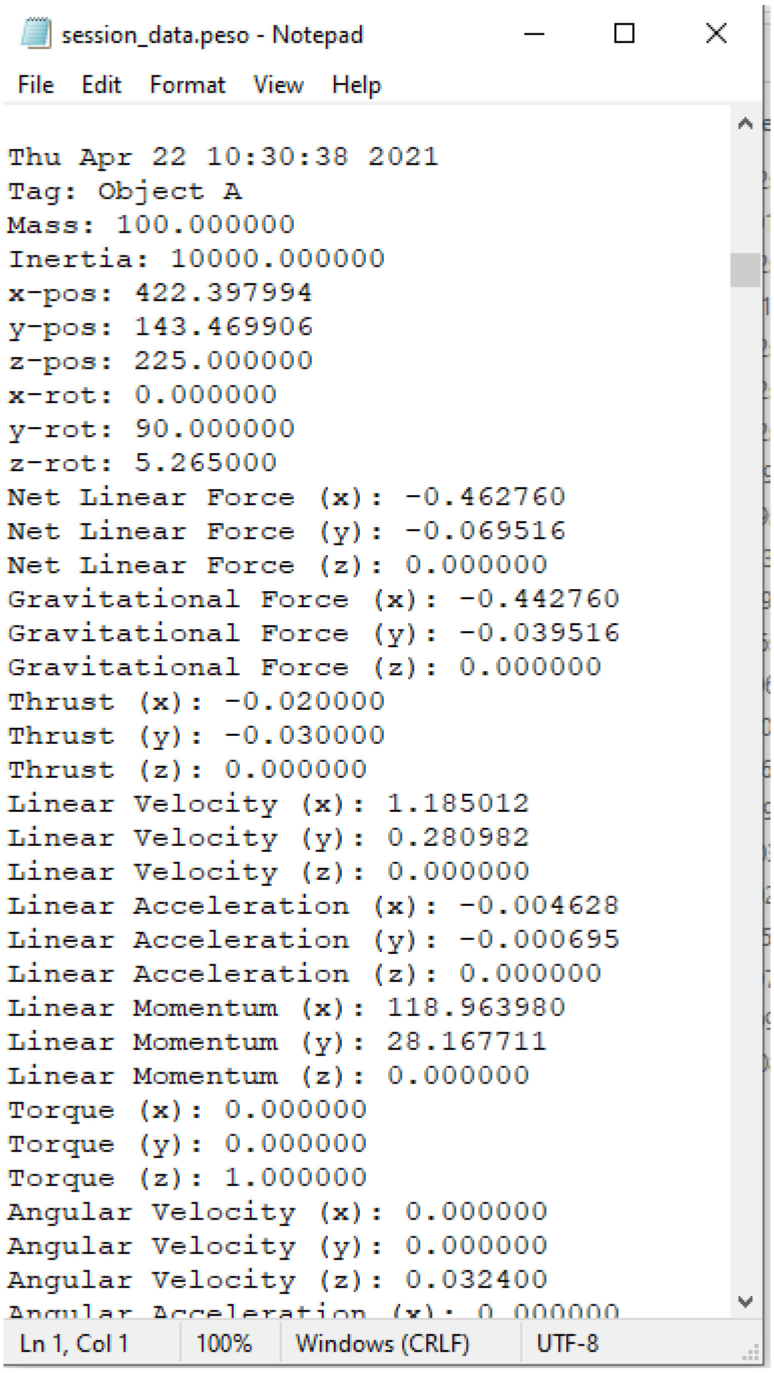
Object Tag:	Object A		
Mass:	100.000000		
	[X]	[Y]	[Z]
Position:	-100.000000	-150.000000	0.000000
Rotation:	90.000000	0.000000	0.000000
Gravity:	0.000000	0.000000	0.000000
Thrust:	-0.020000	-0.030000	0.000000
Torque:	0.000000	0.000000	1.000000
Lin Velocity:	0.000000	0.000000	0.000000
Lin Acceleration:	0.000000	0.000000	0.000000
Ang Velocity:	0.000000	0.000000	0.000000
Ang Acceleration:	0.000000	0.000000	0.000000

```
Set Y position: -200
Value entered was determined to be: -200
Set Z position: 0
Value entered was determined to be: 0
Set X angle: 0
Value entered was determined to be: 0
Set Y angle: 0
Value entered was determined to be: 0
Set Z angle: 0
Value entered was determined to be: 0
Set X thrust: 0
Value entered was determined to be: 0
Set Y thrust: 0
Value entered was determined to be: 0
Set Z thrust: 0
Value entered was determined to be: 0
Set X torque: 0
Value entered was determined to be: 0
Set Y torque: 0
Value entered was determined to be: 0
Set Z torque: 0
Value entered was determined to be: 0

Press SPACE to play, or ESCAPE to quit.
```

Figure 5.1: PESO's MVP. Top three windows are XY/side (red), XZ/top-down (green) and YZ/frontal (cyan) viewports. Top right is the data HUD displaying only the first object's parameters. Bottom right is the PESO runtime console, which serves as the primary interface for the user (Walker, 2021).

With PESO, users can design (in a parameterized manner), run, and analyze simulations for gravitational Newtonian mechanics between any number of objects, using a basic runtime console and two keys, and observe them in real time via orthographic projection. Objects present in PESO's simulations gravitate towards each other under Newton's Law of Universal Gravitation, and exhibit both linear and angular mechanics.



```
Thu Apr 22 10:30:38 2021
Tag: Object A
Mass: 100.000000
Inertia: 10000.000000
x-pos: 422.397994
y-pos: 143.469906
z-pos: 225.000000
x-rot: 0.000000
y-rot: 90.000000
z-rot: 5.265000
Net Linear Force (x): -0.462760
Net Linear Force (y): -0.069516
Net Linear Force (z): 0.000000
Gravitational Force (x): -0.442760
Gravitational Force (y): -0.039516
Gravitational Force (z): 0.000000
Thrust (x): -0.020000
Thrust (y): -0.030000
Thrust (z): 0.000000
Linear Velocity (x): 1.185012
Linear Velocity (y): 0.280982
Linear Velocity (z): 0.000000
Linear Acceleration (x): -0.004628
Linear Acceleration (y): -0.000695
Linear Acceleration (z): 0.000000
Linear Momentum (x): 118.963980
Linear Momentum (y): 28.167711
Linear Momentum (z): 0.000000
Torque (x): 0.000000
Torque (y): 0.000000
Torque (z): 1.000000
Angular Velocity (x): 0.000000
Angular Velocity (y): 0.000000
Angular Velocity (z): 0.032400
Angular Acceleration (x): 0.000000
```

Figure 5.2: PESO simulation data registered to the output file. (Walker, 2021).

SECTION 6 – SOFTWARE EVALUATION

Efficiency of PESO

Algorithms interpreted from linear equations yield linear time complexities: in Big-O notation this is expressed as $O(n)$, where the number of steps required to complete the algorithm corresponds to the number of input elements n applied to it.

The most primitive operation is addition, following is multiplication, then exponentiation. Addition and multiplication each possess linear time complexities, where exponentiation has exponential time complexity ($O(n^2)$ for quadratic complexity, and $O(n^3)$ for cubic complexity) which relies on the values of the exponent and the base: $O(n^x)$.

PESO's fundamental algorithms and their respective time complexities are provided in Table 6.1 below:

Algorithm	**Time Complexity (Worst Case)**	**Constituent Complexities**	**Terminology (n)**
Applying Linear Mechanics	$O(n^2)$	$O(n)$ and $O(n^2)$	Number of objects included in the simulation. This algorithm is run once for each object.
Applying Rotational Mechanics			
Checking User Input Characters	$O(n)$		Number of characters present in the user's input string.
Setting Object Parameters	$O(n^2)$		Number of objects included in the simulation, in addition to the number of physical parameters initialized for each object by the user.
Reading Session Data to Text File			Number of objects included in the simulation, in addition to the number of physical parameters read to the file for each object.

Table 6.1: Table of PESO's algorithms and their respective time complexities provided in Big-O notation.

It is evident that PESO's overall efficiency depends on the complexity of its simulations. Simulations comprising of two objects tend to use 120MB of memory during runtime, and 122MB for three objects. Most of the memory required to run PESO arises from its usage of the SDL.

Evaluation and Comparison Approach

GMAT and PESO are assigned scores based upon analysis of their respective realism and usability. The scores may be either positive or negative in value, affecting how they contribute to their subtotal: for example, the number of realistic features included in a simulator provides a positive effect on its realism, where bugs provide negative effects. Realism scores are calculated as:

$$\mathbf{Realism} = \mathbf{Features} - \mathbf{Bugs}$$

Usability is an inherently subjective aspect as demonstrated by Sorva (Sorva, et al., 2013): it depends on the user and does not adhere to any universal standard when evaluated, individuals are different and would have different experiences with said aspects. To address this and bring usability closer towards being universal and objective, a set of common visual psychological traits defined under the Gestalt Principals are considered, in addition to the number of bugs, and steps present in the application's workflow. Usability scores are calculated as:

$$\mathbf{Usability} = \mathbf{Features} - (\mathbf{Bugs} + \mathbf{Workflow\ Steps}) + \mathbf{Gestalt\ Principals}$$

The Gestalt Principals pertain to the mind's ability to perceive relationships between visuals and geometries in such a way that they possess more meaning than their shared whole. The principals are listed in Table 6.2, those which do not apply in this evaluation are shaded in grey.

Principal	**Effect**	**Example**
Anomaly	The viewer notices that all but one constituent of a visual are similar, symmetrical, or are near one another. This leads the viewer to focus on the one constituent which is out of sync with the others.	A 5 by 5 area of squares: all but one is blue. That one non-blue square becomes the focus. Contrast may produce anomaly.
Continuity	Leads the viewer to focus on and follow the trailing or leading contours of a visual design, such that they reach a destination pointed to by said contours.	An arrow pointing to something.
Closure	The viewer perceives shapes which are not there but would complement the appearance of those that are: filling in the gaps of a visual using their imagination.	A crescent moon may also be viewed as a solar eclipse (the Sun being overshadowed by the Moon).
Figure/Ground	Entices the viewer to witness different perspectives of a visual based on a contrasting foreground and background.	
Proximity (Grouping)	Encourages the viewer to perceive a relationship between multiple objects based on their displacement from each other.	Buttons arranged in a row or column. A tag or label may also serve a similar function.
Similarity	The viewer draws a relationship between visuals with identical or near-identical attributes.	The font style of a string of text. Visuals of the same size, shape, or colour.
Symmetry	Visuals which look undisturbed after being flipped about an axis are considered symmetrical and encourage the viewer to relate them: arises from geometry and positioning.	Mirror images. A butterfly's wings.

Table 6.2: The Gestalt Principals, their effects, and instances where they may occur.

The number of Gestalt Principals exhibited by an application, in each component of its Graphical User Interface (GUI) and/or Heads Up Display (HUD), positively impacts its usability by aiding the user in drawing relationships between their components (e.g., being shown where the control for a feature is located via continuity).

Principals of anomaly, continuity, proximity, and similarity benefit usability. Principals of closure, figure/ground, and symmetry are irrelevant as neither PESO or GMAT possess attributes which exhibit or require them. The number of steps in a workflow negatively impacts usability by contributing to the application's complexity, and the number of bugs inhibits it further by misleading or disabling the user from performing related actions.

Each simulator's overall effectiveness is based on the sum of its realism and usability:

$$\mathbf{Effectiveness} = \mathbf{Realism} + \mathbf{Usability}$$

If a simulator scores zero or more points it may be considered effective in real-world applications, where a simulator which scores below zero points is ineffective and is best replaced with a better one. An application may exhibit various aspects of realism but involve an elaborate workflow (or the converse) which impacts its effectiveness.

It is arguable that both realism and usability benefit from the number of features they possess: an abundance of realistic functions implies an abundance of realism, an abundance of interactive features (particularly those which exhibit Gestalt Principals) implies an abundance of user-friendliness[9]. With these abundances, however, comes the risk of corresponding abundances of bugs or steps constituting a workflow, which decrease effectiveness.

A breakdown of both aspects considered in the evaluation is provided in Table 6.3:

Aspect	**Definition**
Realism	The parity between simulator and reality: realistic features.
Usability	The software's ease of use: workflow and user experience.

Table 6.3: Fields used for the comparison of PESO and GMAT, where realism is the only genuinely objective aspect, with usability being inherently subjective but made more objective with the application of Gestalt Principals.

[9] The number of interactive features can be said to represent a software's flexibility.

Bugs in PESO

Post-development there remain peripheral bugs (a bug being a counter-intuitive or disruptive phenomenon caused by an oversight in the constituent logic of the software) in PESO. Table 6.4 lists each one found via black box testing, and accompanies them with explanations of why they occur, descriptions of their effect on PESO, and solutions which can amend them:

Bug / Effect	Cause	Solution
Affects PESO's **realism**. If one object's displacement from another approaches zero, the intensity of the gravitational force experienced between them approaches infinity. This causes one or both objects to be flung off the screen due to the immense force applied to them.	This is caused by the formula employed for gravitation: where the smaller the resultant range the larger the force. Division by zero ($r^2 = 0$) is prevented with a conditional statement that adjusts this variable to the smallest possible value for a double (DBL_MIN) in the case where the resultant range equals zero. Still, this minute value (approximately 2.22507^{-308}). causes an enormously large value of gravitational force, which results in the bug.	The key here is to prevent the range between any pair of objects reaching close to zero. This can be achieved with the implementation of collision physics which detect when two objects impinge (e.g., if objA.pos.x + objA.radius.x >= objB.pos.x – objB.radius.x, set objA.pos.x to objB.pos.x – objB.radius.x). The mass, velocity, acceleration, and momentum of each object must also be summated, to account for their combined physics.
Affects PESO's **usability**. The telemetry displayed on the data HUD does not scale appropriately as it grows. This causes the digits to overlap and distort each other graphically.	This is caused by the lack of a condition to shift the x-position of a field based on the width of neighbouring fields.	The bug can be solved by shifting the x-position of a field based on the character count (C String size) of the field to its left, for instance: multiplying the value of its x-position offset by this character count. The viewport which displays the object telemetry is resizable, thus large values can still be kept on-screen at the user's discretion.

Table 6.4: Known issues which are present in PESO post-development. None are fatal to the software but impact its realism and usability.

Bugs in GMAT

The online documentation for GMAT (GMAT (R2020a) Documentation, 2020) outlines three known issues[10] considered to be significant, and will be applied in its evaluation:

1. **ID: GMT-5417 (Realism)**
   Adaptive step size control affects GMAT's physical scaling when used in its navigation system, contributing to margins of error.

2. **ID: GMT-6202 (Realism)**
   Up to $10^{-3}ms^{-1}$ spikes are detected in corrections applied to values of Solar radiation pressure, affecting GMAT's ability to simulate it accurately.

3. **ID: GMT-7207 (Usability)**
   Restricts usage of GMAT's Python interface to versions downloaded directly from the official website (python.org), preventing users from interfacing via the Anaconda variant of Python.

[10] A discussion of the causes and solutions to these bugs is beyond the scope of this report and hence omitted here.

Evaluating Realism

The realism exhibited by GMAT and PESO's respective features are summarised in Table 6.5, each tick constitutes 1 point, where a cross constitutes 0 points. The features considered correspond to the attributes of orbital mechanics mentioned in §4, and the features exhibited by GMAT.

Feature	**GMAT**	**PESO**
Multiple bodies	✓	✓
Real world bodies	✓	✗
Star constellations / geography	✓	✗
Linear mechanics	✓	✓
Angular mechanics	✓	✓
Hohmann transfers	✓	✓
Cartesian or Keplerian coordinate systems	✓	✓
Physical scaling (spatial or temporal)	✓	✗
Physical units (S.I., Imperial, or Keplerian)	✓	✗
Rocket burns or thrust	✓	✓
Aerodynamics	✓	✗
Solar radiation pressure	✓	✗
3D physics	✓	✓
Mission optimization	✓	✗
Application	**Subtotal**	
GMAT	**14**	
PESO	**7**	

Table 6.5: The realistic features present in GMAT and PESO, which constitute their simulation capabilities.

PESO can apply 3D Newtonian mechanics to physical objects and simulate rocket burns with its ability to assign a magnitude of thrust to any object during simulation setup: these features enable it to simulate orbital mechanics in a comparatively crude manner to GMAT.

GMAT is capable of much more: it can simulate orbits involving real-world bodies such as the planets of the Solar System, and in an accurately scaled, optimized manner, complete with a selection of coordinate and measurement systems. Alongside this, it can simulate Hohmann Transfers[11]. Minute physical effects which arise from atmospheric drag and pressure from Solar radiation are also considered (at the user's discretion).

[11] It is technically possible to perform Hohmann transfers in PESO provided that a stable orbit is achieved and thrust is applied, although this would not be quite as elegantly performed as in GMAT.

Evaluating Usability

The usability exhibited by GMAT and PESO's interactive features are summarised in Table 6.6, each tick constitutes 1 point towards the application's usability, where a cross constitutes 0 points. The features considered are those which are included in GMAT and PESO.

Feature	**GMAT**	**PESO**
Console-based UI	✓	✓
Graphical UI (GUI)	✓	✗
Scripting API	✓	✗
Telemetry HUD	✓	✓
Orthographic viewport(s)	✗	✓
Full 3D viewport(s)	✓	✗
Ground track viewport	✓	✗
Project file input	✓	✗
Session data output	✓	✓
Project outline or resources panel	✓	✗
Options menu	✓	✗
Animated simulations	✓	✗
Real time simulations	✗	✓
Application	**Subtotal**	
GMAT	**11**	
PESO	**5**	

Table 6.6: The interactive features present in GMAT and PESO, which constitute their user experiences.

The frequency at which Gestalt Principals occur in a simulator can be summarised with its visual components such as its GUI or HUD. Indeed, Gestalt Principals may apply to any of the interactive features mentioned in Table 6.6.

Table 6.7 summarises the Gestalt Principals which apply to GMAT and PESO, and the frequency at which they occur:

Gestalt Principal	**GMAT**	**Freq. (GMAT)**	**PESO**	**Freq. (PESO)**
Anomaly	✓	**3**	✓	**1**
Continuity	✓	**3**	✗	**0**
Proximity	✓	**3**	✓	**4**
Similarity	✓	**3**	✓	**3**
Application	**Subtotal**			
GMAT	**12**			
PESO	**8**			

Table 6.7: The frequency at which Gestalt Principals of anomaly, continuity, proximity, and similarity occur in GMAT and PESO's interactive features.

GMAT's resources panel, GUI toolbar, and 3D viewport all exhibit anomaly (3 points). This arises from the folder icons on the panel, indicating where each aspect of a project's contents (for example, simulation objects) is stored. The GUI toolbar comprises of icons which allow the user to differentiate between interactive functions. The background of the 3D viewport is primarily black due to the depiction of outer space, anomaly arises from the presence of different star constellations and planets present in the simulation. GMAT's resources panel and both 3D and ground track viewports exhibit continuity (3 points). The resources panel shows dashed lines linking the folders to their contents, where the simulation viewports use coloured lines mapping out the orbital trajectories of highlighted objects. The resources panel, GUI toolbar, and options menu each exhibit proximity and similarity (6 points) due to their icons being nearby to each other, and of similar appearance.

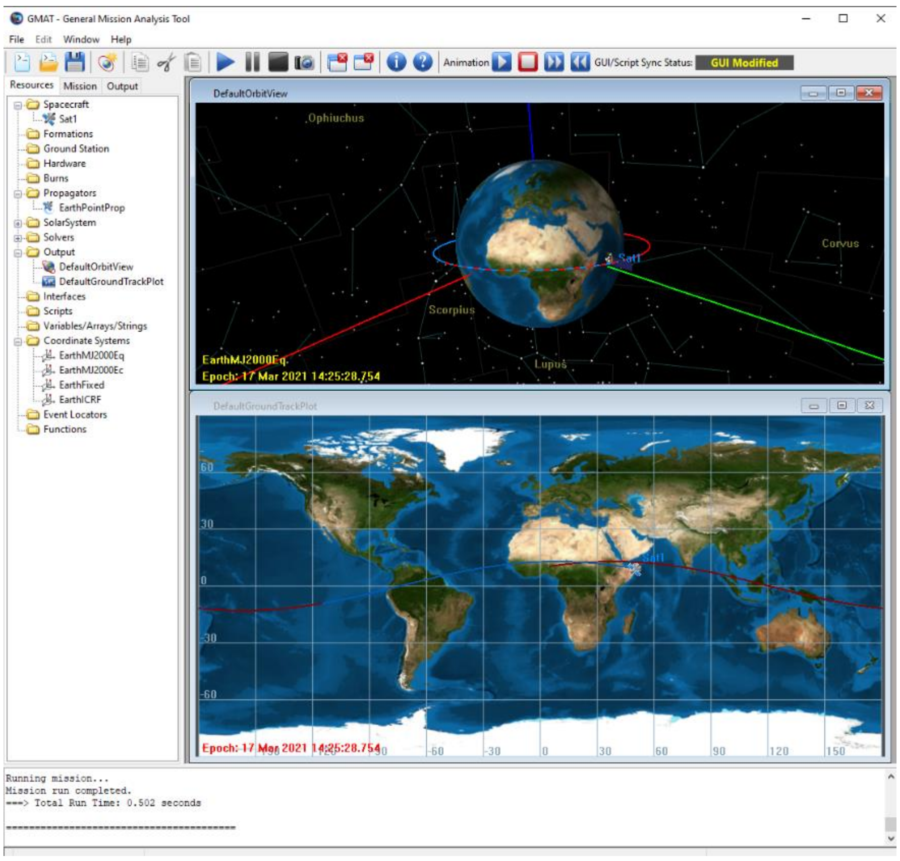
PESO's orthographic 3D viewports collectively exhibit anomaly (1 point): while their geometry is identical, their colours are not, as each viewport constituting the orthographic 3D comprises of its own designated colour. These viewports each exhibit proximity and similarity (6 points) because of the user-specified tags for simulation objects and the identical geometry they apply, respectively. PESO's data telemetry HUD also exhibits proximity (1 point) due to the alignment of the values and their fields.

1. specify no. of objects,
2. initialize objects,
3. adjust viewport layout,
4. execute simulation,
5. stop simulation,
6. review results.

PESO's workflow for creating, running, and collating data from simulations comprises of six steps – double the number from its original design. This workflow must be repeated for each object specified, giving it a worst-case quadratic time-complexity ($O(n^2)$) for the user.

1. create simulation,
2. create spacecraft,
3. create propagator(s) (orbital path(s)),
4. compose mission sequence,
5. create report file for session data,
6. play simulation,
7. analyse session data.

A typical GMAT workflow for creating, running, and collating data from a simulation consists of 7 steps. It is evident that GMAT's realism contributes to the complexity of its workflow, however, its comparatively intuitive GUI and library of predefined objects reduce the steps required from the user to initialize a simulation. Steps 2 thru 4 must be repeated for each spacecraft specified.



```
Running mission...
Mission run completed.
===> Total Run Time: 0.502 seconds

========================================
```

Figure 6.1: Screenshot of a rudimentary GMAT simulation of an Earth orbit (Walker, 2021).

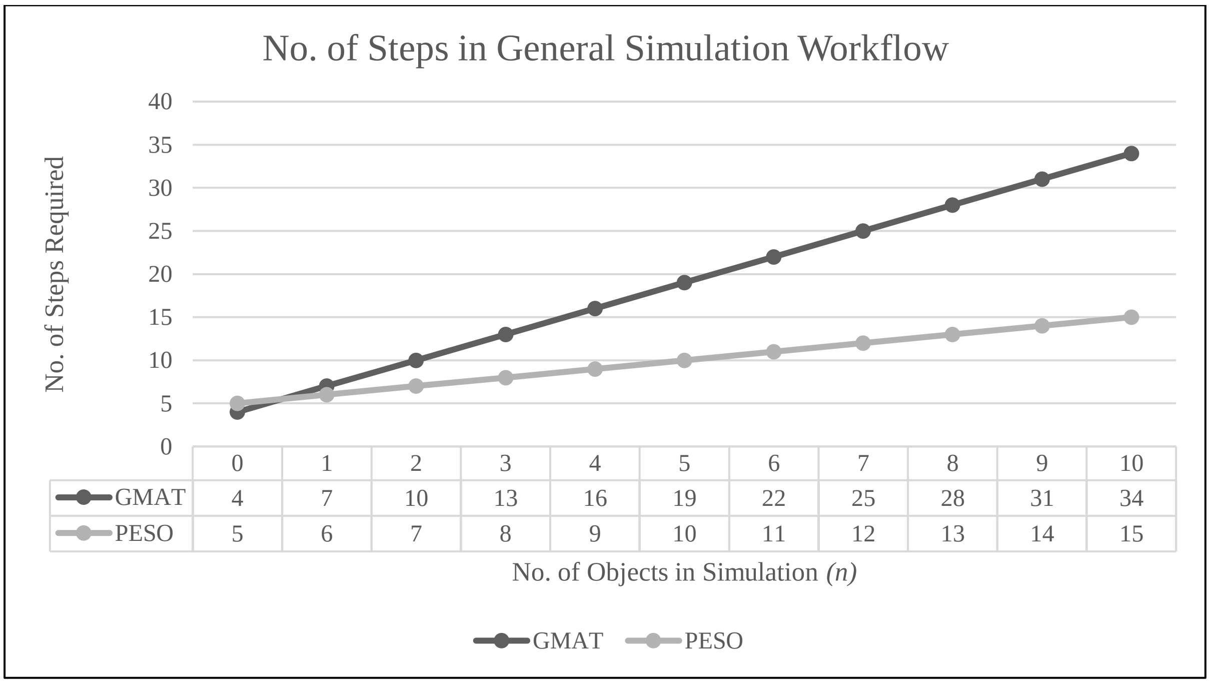
It can be shown that PESO's workflow fairs better than GMAT's as the value of n (the number of objects in a simulation) increases. To simulate gravity using PESO a minimum of two objects are required and must be specified in stage 2 in of its workflow, implying that a $\mathbf{n + 5}$ step workflow is required. A simulation comprising of two objects therefore warrants a 7-step workflow, and a 15-step workflow for ten objects.

One advantage which GMAT has over PESO is that its simulations automatically include a physical body as the focus (Earth by default), putting it one object ahead of PESO in terms of workflow complexity. The minimum value of n required by GMAT is therefore one less than that required by PESO, implying the steps which constitute GMAT's general workflow can be calculated with the expression $\mathbf{3(n-1) + 4}$.

These relationships are further illustrated in Table 6.8 and Graph 6.1:

Application	**Expression**	$\boldsymbol{n = 2}$	$\boldsymbol{n = 5}$	$\boldsymbol{n = 10}$
PESO	$f(n) = n + 5$	$(2) + 5 = 7$	$(5) + 5 = 10$	$(10) + 5 = 15$
GMAT	$g(n) = 3(n-1) + 4$	$3(1) + 4 = 7$	$3(4) + 4 = 16$	$3(9) + 4 = 31$

Table 6.8: The number of steps present in the workflows for PESO and GMAT as a function of how many objects are present in their simulations.



	0	1	2	3	4	5	6	7	8	9	10
GMAT	4	7	10	13	16	19	22	25	28	31	34
PESO	5	6	7	8	9	10	11	12	13	14	15

Graph 6.1: A line graph displaying the relationship between the number of simulation objects and steps necessary for its workflow. From this, PESO's workflow has less steps than GMAT when composing simulations[12].

This superiority displayed by PESO is, however, overshadowed with GMAT's project file input feature, which enables users to create each simulation once rather than spend time recreating it whenever they use GMAT, simplifying its general workflow to exactly 3 steps:

1. load simulation,
2. play simulation,
3. analyse session data.

[12] PESO's workflow is only simpler than GMAT's when users must recreate their simulations from the ground up each time.

Comparing PESO and GMAT

Following evaluations of each software in aspects of realism and usability, a valid comparison between them can be conducted, and is provided in Table 6.9:

PESO – Physics Engine for Simulating Orbits					
Aspect	**Score**				
	Core Features	**Bugs**	**Workflow Steps**	**Gestalt Principals**	**Subtotal**
Realism	**7**	**1**	**n/a**	**n/a**	**6**
Usability	**5**	**1**	**7**	**8**	**5**
GMAT – General Mission Analysis Tool					
Aspect	**Score**				
	Core Features	**Bugs**	**Workflow Steps**	**Gestalt Principals**	**Subtotal**
Realism	**14**	**2**	**n/a**	**n/a**	**12**
Usability	**11**	**1**	**7**	**12**	**15**
Software	**Effectiveness**				
PESO	**11**				
GMAT	**27**				

Table 6.9: Independent comparison of PESO and GMAT in areas of realism and usability. Scores and their derivations are displayed in the right columns.

According to this data (collected and collated using the established evaluation approach) PESO is approximately 2.5 times simpler than GMAT overall: half as sophisticated in realism, and one third as sophisticated in usability.

SECTION 7 – CRITICAL REVIEW

Project Findings

Following the development and evaluation of PESO, and its comparison with GMAT in aspects of realism and usability, it was found that while a simulator's ability to realistically convey aspects of orbital mechanics are impacted by simplifications to its scope, those applied to its workflow and interface complement its usability, such that a user may spend less time figuring out how to use the software and more time genuinely using it – even if only to a limited extent.

Finding 1: Simulators which are more realistic convey a wider variety of information but can run the risk of creating more complicated workflows to stipulate on the effects inherited from this realism. The more simplistic a simulator is, the more comprehensive it is at the expense of its realism.

Finding 2: Simulators which comprise of more features in their interface express greater flexibility, where user-friendliness is complimented by an abundance of Gestalt Principals, which are more prominent in simulators comprising of GUIs: console-based applications are straightforward, but less intuitive than those which apply interactive graphics.

Implications for Games

As mentioned in §4, simulators and video games share identical software architecture, where the former can be considered as video games with realistic and analytical undertones. Therefore, it is logical to assume that the findings of this project concern the design and development of video games which apply similar mechanics to PESO or GMAT in their gameplay.

Such a case is Kerbal Space Program (Take-Two Interactive, Inc., 2018), which challenges the player to design their own space program and spacecraft. Simplifications to its gameplay would make it more comprehensive to players at the expense of its depth, and enjoyment as a tool for entertainment.

Learning from Experience

In retrospect PESO could have shared more parity with GMAT had a game engine such as Unity3D or Unreal been used instead of the low-level tools C++11 and SDL. While using the latter contributed greatly to PESO's simplified and limited approach to simulation, it (particularly its events, file I/O, and graphics subsystems) absorbed a significant amount of development time which could have been utilized in creating a more sophisticated experience for the user, both in terms of the realistic features available, its interactivity, and its graphical presentation.

With more sophisticated tools much of the development time of PESO could have been reduced enough to accommodate user testing, which would have allowed an empirical component to the project's research and provided an insight to the effects of simplification to simulators between individuals, reinforcing the project's findings in this regard.

Were more time provided with the given toolset, however, it would have been possible to implement scaling and physical units, increasing the realistic features of PESO by 2 points. Additional graphical traits, such as a radius drawn across each object's ellipse to visualise their rotation, which would have increased PESO's interactive score by 3 points resulting from the anomaly expressed by each viewport constituting its orthographic 3D.

The inclusion of other orbital simulators aside from GMAT in the comparison would have increased both the relevance and reach of this research, too.

SECTION 8 – CONCLUSION

Throughout the project a simplistic orbital simulator was created using a limited toolset.

A comparison was made between it and an industry-standard orbital simulator in areas of realism and usability, wherein a quantitative and qualitative evaluation of these areas ensued, whose results suggested that the former was approximately 2.5 times less sophisticated than the latter according to the quantitative and qualitative evaluation approach used.

Following the comparison of these two simulators, PESO and GMAT, a verdict comprising of two distinct findings was reached on the effects of simplification on the ability of a simulator to convey scientific information and present itself in a streamlined, user-friendly manner.

It was not PESO's goal to outperform GMAT in realism or usability, but to serve as an unadorned alternative, and in doing so it demonstrated that:

"Simplicity benefits a simulator's ease of use, and negatively affects its flexibility and realism.".

Approximate word count = ***9,967*** *(excluding this line).*

APPENDIX A – MEETINGS

Information on the meetings held between me (the student) and my supervisor.

Date	Topic(s)	Advice / Points
07/10/2020	First meeting of 1st Term. Formal supervisor introduction. Ethics Checklist and Project Proposal submission.	Student should ensure that they are content with their project proposal and are aware of the ethics involved. Submit early to start design and development ahead of schedule.
14/10/2020	Interim Report.	Create interim report template, decide on whether to pursue a 6000 to 10000 words report depending on project complexity/content in mind. Conduct research into similar projects and resources which the student will find helpful/relevant to the report.
21/10/2020		
29/10/2020		*No meeting was held.* *Reminder to submit interim before hard deadline (13th Nov).*
04/11/2020		Final weeks of interim report. Second reader allocated – Dr Sanaz Fallakhair.
11/11/2020		
18/11/2020	Project Viva.	Pre-Viva meeting to discuss purpose of Viva and what to expect from it.
20/11/2020		(Project Viva). Compose material which a newcomer may use as a guide for how to use your application. Elaborate on why the application is important. Why should people use it?
02/12/2020	Collated Viva feedback. Presentation of initial work done for project since submission of interim report.	Legal/Ethical consideration should be explicitly stated within a subsection of your report. Within the introduction, provide 1-2 sentences of your evaluation of your project, stating how the project was successful. Evaluation meaning a comparison between a rival software, or feedback from users. Clearly state projects aims and objectives to readers who are non-specialists in the field/topic (e.g., the project aims to...). Testing means functionality testing – purely implementation-driven (e.g., interact with this and it works or does not work). Showing evidence of learning is a nice proof for evaluation. Product Testing – does it work? Project Testing – is it successful?
09/12/2020	Prudence of meeting.	*Student and supervisor agreed that continuing project development was prudent as opposed to a meeting.* *No meeting was held.*
16/12/2020	Final meeting of Term 1 to discuss current level of progress on project and next steps.	Project should be finished by late February. Getting the project finished by this time will provide time necessary for composing content for the final year report.

17/12/2020	Private meeting to discuss contents of final year report, user testing and the decision to simplify implementation of project.	Decision to simplify project from 10 subsystems to 6 (all of which focusing only on application functionality and UX) is deemed appropriate as it provides more development time for essential features. Said decision may count as a demonstration of Agile project management (accepting and adapting to change). Should focus *less* on visuals and more on UX. Prioritise physics and I/O. Final report should include aspects of interim report as a preface to the section on project development, which should include details of the PESO API (including code snippets). Second reader, Dr Sanaz Fallakhair, is purported to be theory-oriented with regards to project development. It should be kept in mind that they will focus primarily on the project's design and implementation as opposed to its integrity. User evaluation can be split into two parts: theory/setup and results. Both parts should be incorporated into the report.
13/01/2021	First meeting of 2nd Term. Current progress on project development. Next stages of project development. Formal agreement of development timeline.	Suggested to supervisor that I would resume project development no later than February 1st after a period of focusing on other assignments. Priorities stated for development are: • Fixing memory leaks causing gigabytes of memory to be eaten up during runtime. • Enabling the user to define object parameters before running a simulation via console input. • Having object data pushed to a list and written to a text file (peso format) when the simulation has ended. Said data being linked to every second which passes.
27/01/2021	Demonstration to supervisor of current progress in composing report.	*Student and supervisor agreed that project demo of progress should be held in the following week to allow for assignment submissions for other modules using an extension.* *No meeting was held.*
03/02/21		Student updated supervisor on their current level of progress. Mentioned implementation of orthographic 3D, refactored input events subsystem and a refactored representation of PESO object data. Memory leak sapping gigabytes of heap memory had been fixed. Priorities stated for development are: • Fully implementing rotational mechanics and aerodynamic effects into the simulator. • Begin development of file subsystem.

09/02/2021	First meeting of 2nd Semester. General supervision meeting.	Meetings now pushed to Tuesdays at 13:00. Consider theoretical analysis of application rather than user testing for feedback – use pandemic as a reason for not being able to conduct meaningful user-testing (if needed). Metrics may include: • Performance / Time Complexity • Realism / Accuracy • Code Readability* • User Friendliness / Preference* **These aren't universal as they vary between users. Look into the* ***Gestalt Principals*** *for ways to make these metrics universal using commonly shared traits of the human psyche.*
16/02/2021	Final weeks of development	1-to-1 meeting held with supervisor to answer any outstanding questions regarding project development and going forward with the project report. Student and supervisor are both comfortable with the level of progress of the project's development. Supervisor recommended that any ditched features be mentioned within the report to demonstrate planning and contemplation of design.
23/02/2021		*Student and supervisor agreed that continuing project development was prudent as opposed toa meeting.*
02/03/2021	Testing. Report composition. Marking criteria for report.	Formal end date for project development tomorrow at 15:00 GMT. Formal start date for composing report. Marking criteria include: • Technical Grasp • Understanding of Problem Area • Project Management • Report Quality • Evidence of Learning • Research Effort Aiming for A+/A* grade. Supervisor recommends the following: *Discuss the problem(s) which the project set out to solve, the solutions considered and the means in which they were implemented. A working project is preferable, a prototype should suffice even if it is incomplete.* *Demonstrate active planning for the project's development (e.g., its design, creation, and execution). Compare what was expected in theory to what resulted from practice. Use group meetings and Trello (or some other platform) as proof of Agile and Kanban methodologies employed.*

		Writing in the first person is fine but avoid emotional / opinionated text and filler. 8,000 words is a good target. Do not make claims which are not backed up with research and references. Aim to include 25-30 references in the report. *Consider which aspects of your project you managed well and those you did not – justify each aspect in a technical manner.* Develop a report introduction for next week's meeting.
09/03/2021	Showcase of report (introduction). Proof of development.	Uploaded video demonstrating PESO's workflow, appearance, and features implemented up until the deadline for development. Supervisor examined introduction and provided feedback regarding the following: • conciseness of material, • defining the project aim, • defining the project motives. Next meeting will involve examination of development approach.
16/03/2021	Showcase of report (development approach).	Approach sections should contain details of project design, management and planning techniques, and a breakdown of the tools you used throughout the development phase.
23/03/2021	Showcase of report (project research and literature review).	Abundance of quotes in the report is too great and should be reduced. Maximize one's own contents. Remove discussions of GMAT's history and focus instead entirely on its functionality and architecture.
31/03/2021	Submission of report draft to supervisor.	Report draft sent to supervisor for feedback prior to deadline.
01/04/2021	Supervisor's feedback on report draft.	2 major problems and 5 minor problems outlined by supervisor: Major Problems (need fixing): • Demo lacking complexity. Consider adding more objects and an animated background of stars. • Evaluation is weak. Elaborate on scoring format included in software evaluation, focus on objective aspects during evaluation, reduce subjective ones to a paragraph, exclude personal opinions, and expand discussion of Gestalt Principals. Minor Problems: • Section on excluded features is lacking a discussion of how they could be implemented in a second iteration of the project. • Lacking explanations of how to fix bugs in PESO. • Section discussing PESO's algorithms and time-complexities requires clarity on what the variable "n" refers to in each algorithm. • Sections 2 thru 4 comprise of many words, these should be cut down as much as possible. • Report lacks a mention of how research affects games, cite 1 or 2 examples.

16/04/2021	Application of feedback.	Amendments made to PESO’s functionality and appearance, and to the report contents, to adhere to feedback gained from supervisor. Minimum viable report successfully composed.
20/04/2021	Penultimate meeting. Discussion of submission protocol and presentation.	Report should include a demonstrative YouTube video (with no commentary) linked in Appendix B showcasing the project software in action. Contents of presentation may be pre-recorded and provided via Powerpoint slides. According to supervisor’s experience, the presentation Q&A is 20-30 minutes in duration.
27/04/2021	Final meeting. Last minute clarifications on both report and presentation structure.	It was deemed useful to state the project’s originality aspect: implicitly in the Introduction, and Explicitly in the Critical Review. Presentation for project will be held live on Microsoft Teams, with a mixture of presentation slides, a pre-recorded demonstration of the project software, and answers to potential questions prior to the Q&A.

APPENDIX B – LINKS

PESO Codebase:	https://github.com/berw96/PESO
PESO User Guide:	https://tinyurl.com/5xebvbyp
PESO Win-x64 Executable:	https://tinyurl.com/25p7zbcb
Project Gannt Chart:	https://tinyurl.com/kne6zam
Project Trello Board:	https://tinyurl.com/ek2c9dds
PESO Demo Video:	https://youtu.be/4ILwUuJxiKo

BIBLIOGRAPHY

1 NASA Open-Source Agreement v1.3 (NASA-1.3), 2013. *Open-Source Initiative*. [online] Available at: <https://opensource.org/licenses/NASA-1.3> [Accessed January 20th 2021].

2 GNU general Public License v3.0, 2007. *Free Software Foundation, Inc.*. [online] Available at: < https://www.gnu.org/licenses/gpl-3.0.html> [Accessed 18th October 2020].

3 IBM, Spence, I., Bittner, K., 2005. *What is iterative development?* [online] Available at: <https://www.ibm.com/developerworks/rational/library/may05/bittner-spence/> [Accessed 31st October 2020].

4 The BA Guide, Ashenbrenner, J., Khattri V., Khattri P., 2020. Section 7: Kanban Methodology. *Agile Fundamentals: Including Scrum and Kanban – 2020*. [online] Udemy. Available at: < https://www.udemy.com/course/Agile-fundamentals-scrum-kanban-scrumban/> [Accessed 31st October 2020].

5 GMAT Wiki, 2020. *GMAT Wiki – General Mission Analysis Tool (GMAT).* [online] Available at: <https://gmat.atlassian.net/wiki/spaces/GW/Overview> [Accessed 20th October 2020].

6 GMAT (R2020a) Documentation, 2020. *General Mission Analysis Tool (GMAT).* [online] Available at: < http://gmat.sourceforge.net/docs/R2020a/help.html> [Accessed 18th March 2021].

7 GMAT (R2020a) Documentation, 2020. *Known Issues*. [online] Available at: < http://gmat.sourceforge.net/docs/nightly/html/ReleaseNotes.html#N2CCD1> [Accessed 29th April 2021].

8 National Aeronautics and Space Administration (NASA), 2020. General Mission Analysis Tool (GMAT). (version R2020a). [computer program] Sourceforge. Available at: <https://code.nasa.gov/?q=gmat> [Accessed 20th October 2020].

9 Lockney, D., 2016. *Orbital Trajectory Analyzer Takes Mission Planning to New Heights*. [online] Available at: <https://spinoff.nasa.gov/Spinoff2016/t_5.html#:~:text=The%20GMAT%20team%20now%20has,five%20developers%2C%20and%20one%20tester.> [Accessed 17th March 2021].

10 Almas Baimagambetov, 2020. *XCube2D* [computer program and codebase]. GitHub. Available at: <https://github.com/AlmasB/XCube2D> [Accessed 21st October 2020].

11 Elliot Walker, 2020. *Xcube2D Game Engine Fundamentals Assignment Submission* [computer program and codebase]. GitHub. Available at: < https://github.com/berw96/xcube2d> [Accessed 21st October 2020].

12 Elliot Walker, 2021. *PESO*. [computer program and codebase]. GitHub. Available at: < https://github.com/berw96/PESO> [Accessed 5th March 2021].

13 Microsoft, 2020. Microsoft Visual Studio 2019. (version 16.7.6). [computer program] Microsoft. Available at: <https://visualstudio.microsoft.com/> [Accessed 28th October 2020].

14 Gregory, J., 2018. *Game Engine Architecture.* 3rd ed. Boca Raton: Taylor & Francis, CRC Press.

15 Stroustrup, B., 2013. *The C++ Programming Language,* 4th ed. United States: Pearson Education, Inc., Addison Wesley.

16 Stroustrup, B., 2014. *Programming Principals and Practice Using C++*, 2nd ed. United States: Pearson Education, Inc., Addison Wesley.

17 Sedgewick, R., Wayne, K., 2011. *Algorithms,* 4th ed. United States: Pearson Education, Inc., Addison Wesley.

18 Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2009. *Introduction to Algorithms, 3rd ed.* Camebridge, MA: The MIT Press.

19 Kernighan, B.W., Ritchie, D.M., 1988. *The C Programming Language, 2nd ed.* Upper Saddle River, NJ: Prentice-Hall Inc..

20 Prata, S., 2014. *C Primer Plus,* 6th ed. United States; Pearson Education, Inc..

21 Holzner, S., 2011. *Physics 1 – For Dummies*, 2nd ed. Hoboken, New Jersey: John Wiley & Sons.

22 Burton, J., 2019. Semester 1 Module Material, *CI583 Data Structures and Operating Systems*. University of Brighton, unpublished.

23 Baimagambetov, A., 2020. Semester 2 Module Material, *CI517 Game Engine Fundamentals*. University of Brighton, unpublished.

24 Baimagambetov, A., 2020. Semester 1 Module Material, *CI628 Multiplayer Game Development*. University of Brighton, unpublished.

25 Clapham C., Nicholson, J., 2014. *Concise Dictionary of Mathematics*, 5th ed. Great Clarendon Street, Oxford; Oxford University Press.

26 SDL, 2021. *Simple Direct Media Layer Wiki, SDL*. [online]. Available at: <https://wiki.libsdl.org/> [Accessed 29th January 2021].

27 SDL, 2021. *Simple Direct Media Layer Wiki, SDL_Event.* [online]. Available at: <https://wiki.libsdl.org/SDL_Event> [Accessed 29th January 2021].

28 SDL, Atkins, J., 2009. *Simple Direct Media Layer Wiki, SDL_ttf 2.0.10.* [online]. Available at: < https://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf.html> [Accessed 29th January 2021].

29 SDL, 2021. *Licensing the Simple DirectMedia Layer library | zlib license.* [online]. Available at: < https://www.libsdl.org/license.php> [Accessed 29th January 2021].

30 SDL, 2021. *SDL_GetTicks*. [online]. Available at: < https://wiki.libsdl.org/SDL_GetTicks> [Accessed 29th January 2021].

31 Hampton-Smith, S., 2018. *The designer's guide to Gestalt Theory*. [online]. Available at: < https://www.creativebloq.com/graphic-design/gestalt-theory-10134960> [Accessed 4th March 2021].

32 Sorva, J., et al., 2013. Students' ways of experiencing visual program simulation. *Computer Science Education*, [e-journal] 207(38). Available through: < https://www-tandfonline-com.ezproxy.brighton.ac.uk/doi/full/10.1080/08993408.2013.807962> [Accessed 9th March 2021].

33 Bate, Roger R., et al., 2020. *Fundamentals of Astrodynamics,* 2nd ed. Mineola, New York: Dover Publications, Inc.

34 Rutgers, School of Environmental and Biological Sciences, 2021. *Keplerian Elements*. [online]. Available at: <https://marine.rutgers.edu/cool/education/class/paul/orbits.html> [Accessed 23rd March 2021].

35 Null, L., Lobur, J., 2019. *The Essentials of Computer Organization and Architecture*, 5th ed. Burlington, Massachusetts: Jones & Bartlett Learning.

36 Take-Two Interactive, Inc., 2018. *Kerbal Space Program.* [multiplatform game] New York City: Take-Two Interactive.

FIGURES

2.1 Walker, E., 2020. *Concept design for PESO's simulation window.* [online image]. Available at: <https://tinyurl.com/y5tr5zgk> [Accessed 3rd March 2021].

2.2 IBM, 2005. *An iterative project from the project manager's perspective*. [online image]. Available at: <https://www.ibm.com/developerworks/rational/library/may05/bittner-spence/> [Accessed 31st October 2020].

2.3 Walker, E., 2021. *Screenshot of PESOs GitHub Repository as seen on GitHub.com.* [online image]. Available at: <https://tinyurl.com/3avexfr4> [Accessed 5th March 2021].

2.4 Walker, E., 2021. *Screenshot of PESO's Trello board post-development.* [online image]. Available at: <https://tinyurl.com/w5f6f3b9> [Accessed 5th March 2021].

3.1 Walker, E., 2021. *Gannt Chart for PESO's development cycle.* [online image]. Available at: < https://tinyurl.com/2rr75afe> [Accessed 17th April 2021].

5.1 Walker, E., 2021. *Screenshot of PESO's Minimum Viable Product.* [online image]. Available at: <https://tinyurl.com/xbj7e3af> [Accessed 22nd April 2021].

5.2 Walker, E., 2021. *Screenshot of a PESO file containing session data.* [online image]. Available at: <https://tinyurl.com/2whcbaju> [Accessed 22nd April 2021].

6.1 Walker, E., 2021. *Screenshot of a basic two-stage GMAT simulation.* [online image]. Available at: < https://tinyurl.com/nj5hdm74> [Accessed 18th March 2021].