Comprehensive Creative Technologies Project: Building a modular simulation of The Simple Pendulum for the Development of Dynamic Swinging Mechanics in Video Games

**Author Name**[Swaiba2.faisal@live.uwe.ac.uk](mailto:Swaiba2.faisal@live.uwe.ac.uk)

Supervisor: Louca Coles

Department of Computing and Creative Technology

University of the West of England  
Coldharbour Lane  
Bristol BS16 1QY

A group of balls from a rope

AI-generated content may be incorrect.

**Abstract**

This project asks two main research questions: “How successfully can the pendulum swing be simulated using the Unity Game Engine?” and “To what extent can the aforementioned simulation be developed in a way that allows for easy use and integration into other video game developers’ projects, particularly those creating their own swinging mechanics?”

Research has been conducted into the literature surrounding models of the pendulum, real-time physics engines for video games, common design patterns for video game development and key principles of User Centred Design. A force-based simulation of The Simple Pendulum was then created, using the Unity Game Engine (Unity Technologies, 2023). The two test scenes provided with the artifact showcase the simulation functioning in different contexts and being used by different interfaces, showing promising success in answering the two research questions.

User testing is recommended for further work on this project to more effectively identify issues with the simulation’s User Experience, as the conclusions made in this report are based on theory. Specialized research into real-time physics engines would also be beneficial in order to fix slight issues with the simulation’s smoothness over time.

**Keywords**: Pendulum, Real-time Physics, Unity 3D, Game Development

**How to access the project** (not included in word count)

Final Video: <https://youtu.be/zqHzXyJcQt4>

Github: <https://github.com/SwaibaFaisal/CCT-Pendulum.git>

Debug controls for both scenes (keyboard):

Pause = P

Reload Scene = R

Quit = Q

Controls for Playground Scene (keyboard):

Move = WASD

Jump = Space

Swing start = Left mouse button click

Swing end = Left mouse button release

**1. Introduction**

Observing video games released in the past two decades reveals the reccurinng appearance of a particular gameplay mechanic: the swing.

This mechanic appears in a number of different contexts. Video games like *Haste* (Landfall Games, 2025) and *Marvel’s Spiderman* (Insomniac, 2018) employ swinging as a traversal mechanic that helps the player to gain momentum, while others only allow the user to swing from certain points, such as *Uncharted 4* (Naughty Dog, 2016) and *God of War 2* (Santa Monica Studios, 2007) (See figure 1).

A screenshot of a video game

AI-generated content may be incorrect.

Fig.1: A screenshot of God of War 2 (Santa Monica Studios, 2007), where the player can only swing from specific pivot points

The recurring addition of the swing into these games implies a long standing demand for similar simulations. Taking this into account, the objectives of this research project are as follows:

1. Conduct in-depth research into real-time physics engines and the two primary models of The pendulum.

1. Use aforementioned research to create a customizable simulation of The pendulum using the Unity game engine (Unity Technologies, 2023).
2. Research common design patterns when developing software and video games, as well as User Centred Design principles
3. Use knowledge gained to develop the simulation to be modular, customizable and easily integrated into other projects

Creating a customizable simulation of The Pendulum swing using Unity (Unity Technologies, 2023) provides developers with a solid base swing that can be adjusted, iterated and expanded upon to create new, innovative gameplay sysytems.

Alongside development of the simulation, two test scenes and interfaces have been developed in order to effectively evaluate the artifact’s performance.

Deliverables for this research project are: This report and a Unity project file containing the following key components:

* A “package” folder containing the simulation and editor script
* A “playground” folder containing the “Playground Scene” and “Character Controller Interface”
* A “Swing from Start” folder containing the “Swing from Start Scene” and “Swing from Start Interface”
* A video summarizing the research, development and evaluation phases of the report.

**2. Research questions**

The research questions for this project are as follows:

How successfully can the pendulum swing be simulated using the Unity game engine?

To what extent can the aforementioned simulation be developed in a way that allows for easy use and integration into other video game developers’ projects, particularly those creating their own swinging mechanics?

**3. Literature review**

**3.1. The Pendulum Swing**

The pendulum is defined as “a rigid body suspended from a fixed point, known as the hinge, which is offset from the body’s centre of mass” (Hafez, 2015)

There are two primary models for the pendulum: The Simple Gravity Pendulum and The Compound Pendulum. While the overall “swing” movement of these models are the same, it is the variables they consider and the assumptions they operate on that set them apart.

Galileo is declared the first to introduce the concept of The Simple Pendulum (see figure 2) in 1583 (Newton, 2005), but Christian Huygen’s book “Horologium Oscillatorium” (Huygens, 1672) provides a list of concrete assumptions that the model operates in accordance with.

These are as follows:

* The cord is massless
* The gravity is uniform
* The bob is a point mass
* The pivot is a fixed point

A diagram of a line

AI-generated content may be incorrect.

Fig. 2: A visualisation of The Simple Pendulum labelled based on Huygens’ “Horologium Oscillatorium” (Huygens, 1672).

Several simulations of The Simple Pendulum have been carried out in a variety of different contexts. Colorado Institute of Technology have developed a 2-Dimensional simulation of the Simple Pendulum for the purpose of improving students’ conceptual understanding (Wijaya *et al*, 2021) (figure 3).

A screenshot of a computer

AI-generated content may be incorrect.

Fig. 3: a screenshot of the Colorado Institute of Technology’s Simple Pendulum Simulation (Wijaya et al, 2021)

Another simulation sees The Simple Pendulum modelled for the purpose of measuring energy (Palka *et al,* 2016), while CAITA 5 (Dassault Systemes, 2020) was used to create a computer-aided design and kinematic simulation of Huygen’s pendulum clock (Del Río-Cidoncha *et al,* 2020). Overall, the simulation of the simple pendulum has been carried out for a variety of different reasons.

In the same book that detailed the principes of The Simple Pendulum, Huygens provides an explanation of The Compound Pendulum (Huygens, 1672). Also referred to as the Physical Pendulum, this model is described as “an extension of the Simple Pendulum” (Hafez, 2015) and takes into account variables that its simple counterpart discards (figure 4).

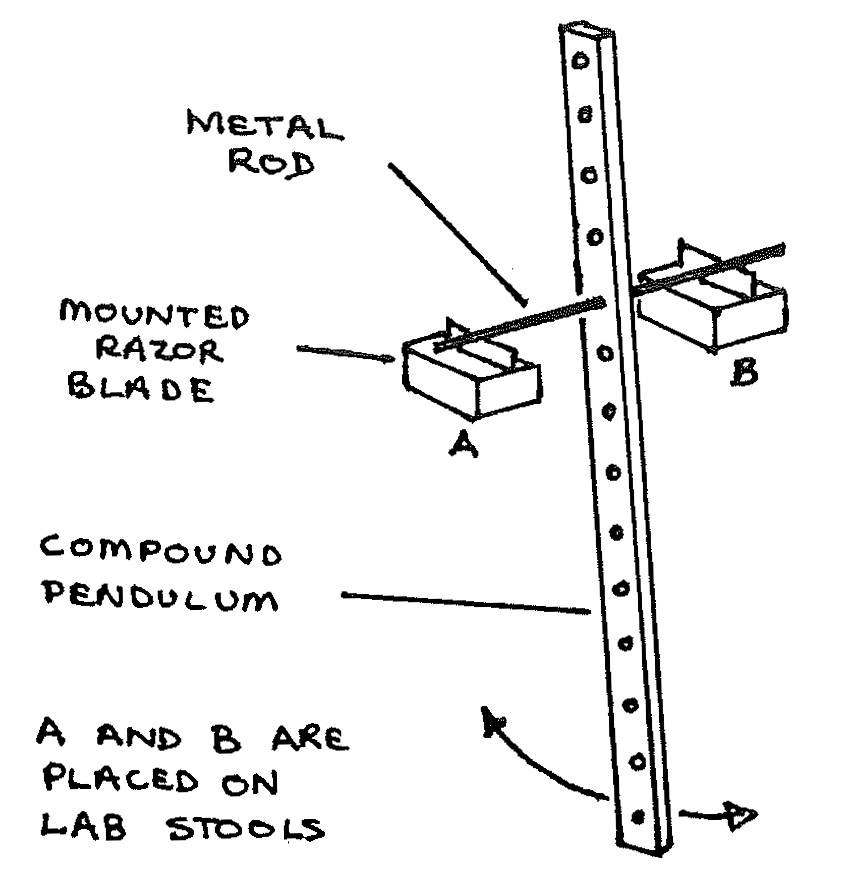


Fig.4: diagram representing a real life compound pendulum as part of a physics experiment (Drach et al, 1987).

Significantly, these variables include the bob’s centre of mass, the pivot’s centre of mass, the weight of the cord and the position of the pendulum’s pivot.

Despite The Compound Pendulum’s complexity in comparison to its simple counterpart, the model has also been simulated using various software. Hafez’s open-source simulation of the pendulum swing was completed using The Compound Pendulum (Hafez, 2015), and Stanford University’s pendulum simulator features options for single and double Compound Pendulum simulations (Johnson, N.A).

Observing the simulations of both primary models of the pendulum overall, it appears that both models are researched and simulated concurrently without a bias of one model over the other.

Also notable is the fact that each simulation of the pendulum swing discussed, bar Stanford University’s “pendulum simulator” as it is unclear what was used, was developed with different software. This, combined with the previous paragraph’s conjecture, can be used to argue that simulation of the pendulum swing can be achieved flexibly and in a variety of different ways.

As indicated in the introduction, this project uses The Simple Pendulum model in its implementation. While The Compound Pendulum is arguably more accurate in its representation of the real-life pendulum swing, taking into account forces that its simple counterpart disregards, this leads to complications when initially implementing the simulation *and* altering it for use by other developers. The equation for calculating the period of a Compound Pendulum (with small-angle approximation) is as follows:

A mathematical equation with black text

AI-generated content may be incorrect.

Fig.5: The equation for calculating the period of a Compound / Physical Pendulum (Nave, N.A)

In contrast, the assumptions that The Simple Pendulum model functions on result in a simpler, cleaner equation (Fig. 6) within the same context:

A black math symbol with a white background

AI-generated content may be incorrect.

Fig.6: The equation for calculating the period of a Simple Pendulum (Huygens, 1672)

Additionally, the two models were compared against swinging mechanics in popular AAA games. Considering the project’s intention to develop the simulation in a way that allows for iteration and integration into other projects, it is logical to utilize the model that aligns the most with that of existing simulations.

A person suspended from a rope

AI-generated content may be incorrect.

Fig.7 A screenshot from God of War 2 (Santa Monica Studios, 2007) where the player must swing across a small chasm.

A person from a rope

AI-generated content may be incorrect.

Fig.8: A screenshot from Uncharted 4: A Thief’s End (Naughty Dog, 2016) where the player must swing across a large chasm.

A person in a blue and red garment from a rope

AI-generated content may be incorrect.

Fig.9: A screenshot from Spiderman 2 (Insomniac Games, 2023) where swinging is the primary movement mechanic of the game.

Examining figures x, y and z together exposes some key similarities to The Simple Pendulum, in both its appearance and its nature:

1. The pivots are fixed points, as opposed to being a free horizontal axis
2. The cord is taut and always remains under tension
3. The player/ bob is its own object with a separate mass to the cord

Currently, the lack of information on how exactly these mechanics were implemented means that the above points can only exist as observations.

This does limit the ability to conclude that AAA studios *are* using this exact model in the development of their gameplay mechanics. However, it is undeniable that the behaviours of these simulations are at the least analogous to that of The Simple Pendulum. Furthermore, it is suggestive that these behaviours are consistent despite the discussed games being released over a period of 27 years.

As stated above, the research questions for this project outline the intention to create a simulation of the pendulum that is useful to video game developers that are creating their own swinging mechanics. After examination of existing simulations alongside the complexity of the two models, it was concluded that The Simple Pendulum (Huygens, 1672) is the model most relevant for this project.

**3.2. Real-Time Physics in Video Games**

Physics simulations are often implemented into video games to simulate physical laws and make players “feel as if they are playing in a real world” (Maranowski, 2024). Fehevari (2024) argues that it is not how “realistic” these physics simulations are that contributes to player immersion, but rather the consistency of the laws put in place throughout the game. This sentiment is echoed by Backer (2011), who draws a connection between immersive experiences and coherence.

Real-time physics engines differ from High-Precision engines due to their prioritisation of performance over physical and numerical accuracy (Maranowski, 2024). Templett (2020) argues that this is due to the fact that video games function as a form of entertainment, removing the need for “safety-critical systems” that would require a high level of accuracy.

An example of this is real-time physics engines’ use of “RigidBodies”. In the context of real-time physics, this term refers to a solid object that cannot be deformed under external sources (Unity Technologies, 2024). These types of objects cannot exist in real life, and yet they are used Nvidia’s *PhysX* (Nvidia Technologies, 2024) and *Havok* (Havok, 2024).

Nvidia PhysX is a real-time, open-source physics engine SDK. Described as “the flagship tool of Omniverse Simulation” (Delise, 2022), the engine is used in the process of deep reinforcement learning (State et al, 2022), and creating metaverse applications (Delise, 2022).

Unity physics’ integration of PhysX uses rigid body dynamics to simulate physics. Rigid bodies can be added as components to game objects and configured to work under Newtonian laws (Unity Technologies, 2024).

As shown in figure 10, the internal physics update loop exists separately to that of the game logic loop. This loop can run multiple times per frame, at the cost of performance, to achieve higher levels of accuracy during physics simulations by simply altering the “Time.FixedDeltaTime” value.

A screenshot of a computer

AI-generated content may be incorrect.

Fig.10: Relevant slice of a flow diagram visualizing Unity Engine’s function execution order (Unity Technologies, 2024).

While also allowing for a high level of flexibility for developers looking to alter the accuracy of their physics simulations, the decoupling of the physics loop from the update loop solves a common performance issue with Nvidia PhysX; described in their documentation as “the well of despair” (Nvidia Corporation, 2017).

It is true that Unity’s integration of a real-time physics engine over a high-precision engine limits the level of accuracy that this project’s artifact can achieve. However, the intention is not to create a simulation that is completely numerically accurate. The Simple Pendulum model (Huygens, 1672) is simulated as the base movement, onto which additional, adjustable forces can be added to provide developers with more flexibility.

**3.3. Software Design: Patterns**

In the context of programming, software development and game development, Object Orientated Design (OOD) is a development paradigm in which software systems are created as “a collection of objects of various types that interact with each other through well-defined interfaces” (Dathan *et al*, 2015).

It can be argued that the reliability of OOD is explicit in its implementation in countless software and programming languages; from Simula’s use of objects, classes, inheritance and virtual methods in 1965 (Johan-Dahl et al, 1962) to Unreal Engine 5’s modular blueprints system (Epic Games, 2025).

In the context of Games Development specifically, OOD is referred to as a useful way of approaching game development by academics (Akkaya et al, 2022; Winggyist et al, 2022) and professionals in the games industry such as Bleeding Edge Studios (2024). Unity also directly details its compatibility with Object Orientated Design through its “Game Object” focused architecture in its documentation (Unity Technologies, 2024).

Arguments have been made in opposition to OOD, in the context of games development, due to its lack of prioritisation of data and performance. Modern video games have increased in scale (A. Kuo et al, 2017) and computational demand (Bayliss, 2022) throughout the years. Fabian (2013) argues that the answer to keeping up with this trend is through development in adherence with the principles of Data Orientated Design.

Data Orientated Design (hereafter DOD) is a design pattern that heavily prioritises data over the object. Rather than making use of common Object-Orientated methods like interfaces and class inheritance (Dathan *et al,* 2015), DOD’s principles involve storing and handling data in a way facilitates high levels of efficiency (Fabian, 2013).

Winggyst et al (2022), while noting the usefulness of OOD in video game development alongside its position as “industry standard”, concluded in their conference that developing a game in adherence with Data Orientated Design principles led to significantly better performance compared to OOD.

Furthermore, Unity’s Data-Orientated Technology Stack (DOTS) is a recent addition that facilitates a more data-orientated approach to development as compared to the default engine’s object-orientated functionality (Unity Technologies, 2025). Freddie Baboard details the use of this system in the development of *Southfield* (Radical Forge, N.A), a multiplayer game in which nearly every object in the game world is affected by physics (Baboard, 2025).

It is true that Baboard also discussed several other optimization methos in his talk that are non-specific to DOD, such as a chunk-based rendering system for the game world. However, the development of a data-orientated addon to the Unity Engine alongside this pattern being integrated into the development of a large-scale commercial game can still be used to argue in favour of DOD’s relevance alongside OOD.

The second research question asks whether the simulation of The Simple Pendulum (Huygens, 1672) can be developed in a way that allows for easy integration into other video game development project. Developing the artifact in adherence to DOD greatly reduces the chances of this question being answered effectively due to this discussed prioritisation of data over reusability, of which the project is trying to maximise.

Considering this alongside the fact that even champions of a Data Orientated Design approach to video game development recognise Object Orientated Design as the industry standard (Bayliss, 2022) (Fabian, 2013) (A, Kuo *et al,*  2017), the decision was made to develop the artifact without intentional use of DOD.

Consequently, the artifact for this project has been developed in accordance with OOD principles. The “practice” section goes into more detail about how exactly this operates.

**3.5 User Centred Design**

User Centred Design (USD) is described as “an approach to design by which the needs as well as preferences of the users are placed at the heart of the design process.” (Mitsiou, 2024).

Design principles found during research that are relevant to this project have been compiled in figure 11:

|  |  |
| --- | --- |
| **Principle** | **Description** |
| Gestalt Proximity Principle (Lowdermilk, 2023) | Elements placed close to each other are seen as more related than those far apart |
| Provide Intuitive Navigation (Browne, 2021) | Users should be able to intuitively understand how to use the application |
| Contextual Progressive Disclosure (Nielson, 2006 ) | Reduce cognitive load by only displaying relevant information |
| Early and frequent user involvement minimises mistakes (Whitman *et al,* 2024) | Early and frequent user testing can help catch issues with the application early |

Fig.11: A table of common design principles found during research

Whitman *et al* (2024)argues that the development of applications in accordance with UCD principles results in increased usability. Considering the project’s intention to create an artifact that is usable by other developers led to the conclusion that development should be carried out in accordance with these principles.

Notably, The Interaction Design Foundation emphasises the importance of user testing when developing user-centred applications, calling development “an iterative process” (Interaction Design Foundation, 2025). Lowdermilk (2023) echoes this sentiment, arguing that user testing can aid in finding more pervasive issues with the user experience of an application.

---

**4. Research methods and Ethics**

UWE Bristol provides students with access to several large databases alongside an extensive library search tool. The library search tool, which contains functionality for searching based on keywords, material type, field and more (see figure 12) allows students to gather research of various types.

A screenshot of a computer

AI-generated content may be incorrect.

Fig.12: UWE Bristol Library Search’s “advanced search” page, showcasing various filters and sorting options

This was the primary research tool for this project, alongside ResearchGate (Madisch *et al,* 2008), Academia.edu (Academia, Inc, 2008) and Google Scholar (Google, 2004).

Regarding the type of research gathered and utilised for this project; a mixture of qualitative secondary research was used. This consisted of academic papers, journal articles, books, developer talks and other publications within the context of video game development, software design, real-time physics and pendulum simulation.

Keywords used when researching the project are as follows:

* Simulating The Simple Pendulum
* Real-Time Physics
* Unity Physics System
* Tool Development

**4.1 The First Research Question**

The first research question is: “How successfully can the Pendulum Swing be simulated using the Unity game engine?”

Answering this question involved the following steps:

1. Review the current literature surrounding simulations of the pendulum swing and real-time physics engines
2. Consider this review in the context of this project to determine the most relevant method of simulating the pendulum swing.
3. Implementing the pendulum simulation

Research into simulations of the two primary models of the pendulum revealed multiple similarities between The Simple Pendulum (Huygens, 1672) and swinging mechanics in video games. Evaluating this alongside the differences in complexity between The Simple and Compound Pendulum models led to the conclusion that The Simple Pendulum is the most relevant model for this context.

Evaluating the success of the artifact in answering the first question has been done through comparison of the behaviour of objects affected by the simulation against the behaviour of objects in The Simple Pendulum (Huygens, 1672).

**4.2 The Second Research Question**

The second research question expands on the first, asking “To what extent can the aforementioned simulation be developed in a way that allows for easy use and integration into other video game developers’ projects, particularly those creating their own swinging mechanics?“

Answering the second research question was then done as follows:

1. Review the current literature surrounding software design patterns and principles
2. Consider research in the context of this project to determine how the simulation can be extended and / or packaged to allow for use by other developers
3. Implement chosen methods of extension

As the success of the artifact in answering the second research question is dependent on its usability by other developers, user testing was considered for this project. Analysis of the primary quantitative and qualitative data that effective user testing generates allows for iteration of the artifact based on in-context feedback alongside theoretical principles.

Despite this, it was ultimately decided that the conducting and evaluation of user testing alongside development of this project was outside the scope of this project. To conduct relevant user testing to a satisfactory level, video game developers familiar with Unity 3D (Unity Technologies, 2023), who are also developing swinging mechanics need to be located.

While this is not impossible, the time and resources needed to locate such a specific focus group would require the artifact to be developed at least two months in advance. This greatly reduces the time available for development and polish, leading to a lower quality research project overall.

Consequently, the decision was made to evaluate the artifact’s success in answering the second research question based on theoretical UCD design principles and relevant industry practice.

**4.3 Ethics**

No ethical considerations regarding data collection and analysis have taken place due to the fact that user testing was not done in this research project. This is in line with the UWE Bristol Handbook of Research Ethics (UWE Bristol, 2024).

Given further time and resources, it is likely that user testing would take place. If this occurs, comprehensive research into the ethics of data collection would need to take place.

**5. Practice**

**5.1. Object Orientated Design and Project Structure**

As justified in the literature review, the artifact for this project has been developed in accordance with Object Orientated Design (OOD) Principles.

All functionality for the simulation itself is contained within the “Pendulum Script” and “Custom Physics Base” script, which the former inherits from . This component, described in OOD terms as the “object” can then be attached to any “GameObject” in the scene. At this point, all that is needed for the simulation to function at runtime is to have an “Interface” call the “StartSwing()” function, passing through a Vector3 pivot position.

In order to demonstrate the simulation’s flexibility as well the script’s functionality as an “object” in the context of OOD, two different interfaces have been made. Section 5.6 and 5.7 describe these interfaces in more detail.

Properties (Unity Technologies, 2024) have been utilised in order to create getters and, in relevant cases, setters for private variables in the “Pendulum Script”. This has been done in order to facilitate extension of the simulation, as interfaces can reference certain values of the “Pendulum Script” and call functions accordingly.

An example of this in the “Character Controller Interface” is the air control functionality. The interface reads the “IsSwinging” property of the pendulum script and restricts player control if the value is true.

**5.2 Simulating the Simple Pendulum – Position Based Approach**

Initially, the “bob” Game Object was made to move along the pendulum arc by directly setting its position. While this method allowed for the first research question to be answered successfully, with the “bob” moving precisely across the arc in two dimensions, issues arose when attempting to answer the second research question with this approach.

Observing the equation for calculating the period of The Simple Pendulum (figure 13) reveals that only two variables are considered: coord length (*L*) and acceleration due to gravity (*g*).

A black math symbol with a white background

AI-generated content may be incorrect.

Fig.13 Equation for calculating the period of The Simple Pendulum (Huygens, 1672).

L is always determined while entering the swing, and does not change in The Simple Pendulum model, removing the possibility of it being an adjustable parameter. Overall, a very limited number of parameters could be added to customize the simulation when utilizing this approach (fig.14).

A screenshot of a computer

AI-generated content may be incorrect.

Fig.14 a screenshot of the customization options available to the user in the position-based approach.

Furthermore, setting the position of the bob directly restricts the simulation’ ability to be integrated into other developer projects. In the event that the user applied the simulation to objects in a physics-based game, the simulation itself would not be affected by any additional forces or effects in the environment (see figure 15). Furthermore, if there was an object in the simulation’s path that was intended to obstruct the bob, the object would move straight through.

A diagram of a diagram of a diagram

AI-generated content may be incorrect.

Fig.15: a visualisation of the various game elements the simulation overrides.

The overriding of collisions and any forces applied by the Unity Physics engine is problematic for this project since the artifact itself becomes unusable in a variety of contexts. In the event that the simulation is applied to a “Game Object” that has a physics-based character controller, for example, all forces acting on the object will be ignored when it is taking place.

Overall, considering the lack of customizability this approach allows for alongside the “SetPosition()” function overriding key features of the engine led to the abandoning of this implementation in favour of a force-based approach.

**5.2 Simulating the Simple Pendulum – Force Based Approach**

The simulation of The Simple Pendulum (Huygens, 1672) included in the artifact functions by calculating the Tension force for the “bob” and adding its centripetal force.

In order to determine what forces needed to be applied to the “bob” object to make it move across the pendulum arc, a diagram of The Simple Pendulum was created (figure 16) and labelled according to the forces acting upon it (Huygens, 1672).

**A diagram of a force

AI-generated content may be incorrect.**

Fig.16: A representation of the forces acting on the Simple Pendulum model, as described by Christiaan Huygens (1672).

From this, the equation for the Tension force shown in Figure 17 was ascertained alongside the general equation for Centripetal Force:

A math equations on a white background

AI-generated content may be incorrect.

Fig.17: Equations for general Centripetal Force and the Tension Force for a Simple Pendulum.

Considering the above equations in the context of this project, certain substitutions and changes need to take place.

For the Tension Force equation, the magnitude of “Unity’s Physics.gravity” Vector3 is used instead of the earth’s gravity: 9.801 (Cook et al, 2025). While using the constant value of the earth’s gravity would create a more realistic movement, it limits the projects that the simulation can be effectively integrated into. Using the engine’s gravity value solves this issue in cases where alternate gravity mechanics are implemented through changing the engine’s physics configurations.

Regarding the Centripetal Force equation, the “radius” variable in this case is be substituted for the coord length as they are functionally the same in this context. As the velocity of a “Game Object” in Unity is also a Vector3 value, the magnitude of the object’s velocity is used instead.

Incorporating these changes into the equations and adding them together results in a final “Net Force” equation:

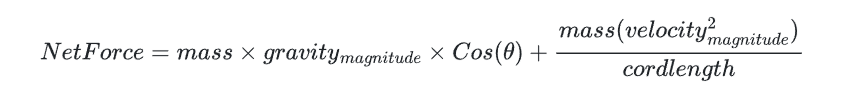


Fig.18: The “Net Force” equation for simulating The Simple Pendulum in Unity 3D

In the artifact, the “Net Force” equation outputs a float which is then multiplied by the normalized distance between the bob and the pivot. This gives the base movement force for the “bob” object as shown in figure 19.

**A diagram of a path

AI-generated content may be incorrect.**

Fig 19: The base pendulum simulation working in-engine

**5.3 Adjustable Parameters**

By adjusting the parameters in the inspector, users of the simulation tool can tweak the simulation without directly altering the code. Figure 20 lays out these parameters alongside their variable type and functions:

|  |  |  |
| --- | --- | --- |
| **Name** | **Variable Type, Default Value** | **Function in Simulation** |
| Mass | Float, 1 | Mass of the bob. Overrides RigidBody.Mass value |
| Drag | Float, 0.001 | Damping on linear motion. Overrides RigidBody.Mass value |
| Interpolate | Boolean, True | If true, turns on smoothing of physics between frames. Overrides RigidBody.Interpolate setting. |
| Min/Max Force | Float, 2  Float, 100 | Clamps force applied to bob between these two values in any direction. |
| Use Custom Timestep | Bool, False | If true, changes Time.FixedDeltaTime to “Fixed Timestep Value”. |
| Fixed Timestep Value | Float, 0.001 | If “Use Custom Timestep” is true, affects number of times the Unity Physics loop runs in a single frame. |
| Jump on start | Bool, True | If true, bob will jump before starting swing |
| Start Jump Value | Vector3, (4,6,4) | If “Jump on start” is true, the bob will jump at this specified force. |
| Swing Force Multiplier | Float, 1.2 | Increases intensity of the swing. Bob gains momentum faster. |

Fig.20: a table detailing all the adjustable parameters in the simulation alongside their type and function.

The ”Swing Force Multiplier” parameter was added to increase the intensity of the swing movement in cases where the amplitude of the swing is especially large, and the “bob” was moving especially slowly. This multiplier is applied to the Tension Force and Centripetal Force equations.

As mentioned in the literature review, real-time physics engines sacrifice accuracy in favour of performance (Maranowski, 2024). This can be seen in the performance of the simulation without interpolation or smoothing (see figure 21).

A white cube with a red string attached to it

AI-generated content may be incorrect.

Fig.21: Early iteration of the artifact with jittering issues caused by high timestep value and low framerate

To mitigate this issue, the “Use Custom Timestep” option was added alongside an option to turn on interpolation between physics frames. While turning on interpolation affects only the “bob” object’s RigidBody, altering the “Fixed Timestep” value affects the entire project.

Also mentioned in the literature review is the effects of altering The Fixed Timestep value on overall performance. In order to account for users favouring performance over smoothness *and* those wanting a smoother simulation, altering the Fixed Timestep was made optional.

Video games such as *Marvel’s Spiderman* (Insomniac Games, 2018), *Marvel’s Spiderman 2* (Insomniac Games, 2023) and *Haste* (Landfall Games, 2025) all feature implementations of the swing that see the player jumping up and towards the pivot point before moving across the arc. In alignment with this trend, a “jump on start” option was added to the simulation. The user can specify the intensity of the jump in all three directions.

**5.4 Editor Extensions**

Unity’s engine attributes (Unity Technologies, 2025) were used alongside the Immediate Mode GUI (IMGUI) editor extension system in order to present information about the simulation and its adjustable parameters in a way that aligns with principles of User Centred Design (UCD).

Figure 22 showcases the final inspector layout for the “Pendulum Script” component:

**A screenshot of a computer

AI-generated content may be incorrect.**

Fig.22: Screenshot of the inspector layout for the “pendulum script” component.

A custom editor script was created for the “Pendulum Script” component in order to add the “Default Values” button above information about the net force equation and its current value during runtime. Pressing the button automatically sets all variables to values that allow the simulation to work straight away. This provides the user with a base simulation to tweak and expand upon using the parameters presented to them below.

In line with the Gestalt Proximity Principle (Lowdermilk, 2024), the header, space and serialize field attributes have been used in order to organize information into relevant groups. Variables that directly affect a specific aspect of the simulation, such as the “Jump On Start Swing” and “Start Jump Force” parameters, are visually grouped together while elements that are relatively unrelated to each other are placed away from each other.

The tooltips attribute has also been used to provide information about how each variable affects the simulation (see figure 23). Alongside the addition of the force equation and current net force at the top of the inspector, this allows users to observe in detail the behaviour of the simulation and adjust parameters accordingly.

A screenshot of a computer

AI-generated content may be incorrect.

Fig.23: Tooltip for the “interpolate” value

Floats restricted between two values, such as the “Custom TimeStep Value”, are displayed using sliders with the range attribute (Unity Technologies, 2024). While using a default serialized float field was passable, allowing the users to adjust values and still restricting said value with the max attribute, inputting a float higher than the specified maximum value and pressing enter would inexplicably reset the value to 0. This goes directly against the visibility principle (Lowdermilk, 2024). Using a slider, however, fixes this issue due to the visual feedback it provides; the slider’s placement on the line indicates how far it is from the minimum / maximum point.

**5.5 Playground Scene**

A playground scene has been created alongside development in order to effectively test the simulation working under various conditions.

The environment for this scene (figure 24) consists of default shapes in various scales, rotations and positions. Each “Game Object” in the scene, excluding the floor object, is part of the “world object” layer. Consequently, any point on the surface of these objects can be passed though as a pivot point by the Character Controller Interface (discussed in 5.6).

A video game of a game

AI-generated content may be incorrect.

Fig.24: The test scene environment

In order to test the flexibility of the simulation and the different ways it can be integrated into a project, two interfaces were created: the “Character Controller Interface” and the “Swing from Start Interface”.

**5.6 Character Controller Interface**

The “Character Controller Interface” contains functionality for a basic, force-based character controller and dynamic target detection system. The player can move in four directions using WASD and press the left mouse button to “swing”.

As mentioned in 5.1, the “StartSwing()” and “EndSwing()” functions are public and can consequently be called by any interface. When the left mouse button is clicked, a Raycast is carried out from the camera’s position to the cursor’s world position. In the event that the Ray hits a “GameObject” with the tag “world object”, the attached script’s “StartSwing()” function is called, and the hit position is passed through as the pivot point. When the left mouse button is released, “EndSwing()” is called which stops the simulation and clears all relevant variables in the references “Pendulum Script”.

The “air control” parameter on the Character Controller Interface was implemented to showcase one way the simulation can be expanded upon. The player can enter a value between 0 and 1, which determines the extent to which they can control the player object during a swing.

In the playground scene, the “GameObject” referenced in the “Character Controller Interface” is a default cylinder shape. The “Pendulum Script” component was applied to the cylinder, referenced in interface, and the following configuration of the “Pendulum Script” values was used:

|  |  |
| --- | --- |
| **Variable** | **Value** |
| Mass | 1 |
| Drag | 0.001 |
| Interpolate | True |
| Use Custom Timestep | True |
| Custom Timestep Value | 0.001 |
| Jump on Start Swing | True |
| Start Jump Force | 4,6,4 |
| Swing Force Multiplier | 1.25 |
| Max Force | 80 |
| Min Force | 5 |
| Show Cord | True |
| Cord Width | 0.1 |

Fig.25 variable configurations for the “Character Controller Interface”

A trail renderer was also added to the cylinder object to observe its path when swinging. As shown in the figures below, this interface paired with the pendulum script allows the player to swing around objects along any axis:

A screenshot of a video game

AI-generated content may be incorrect.

Fig 26: A screenshot of playground scene gameplay where the player is swinging across a horizontal axis.

**A screenshot of a video game

AI-generated content may be incorrect.**

Fig.27: A screenshot of playground scene gameplay where the player is swinging across a vertical axis.

A drawing of a red line on a white surface

AI-generated content may be incorrect.

Fig.28 A screenshot of playground scene gameplay where the player has used the swinging mechanic and the movement system to get up onto a platform.

Figures x,y,z,a all show the simulation functioning across different axes and speeds, showcasing its dynamic nature. Significantly, figure 28 shows the swinging functioning alongside the movement forces added by the character controller interface as well as colliding with other “GameObjects” in the scene. Instead of these additional forces / obstacles being overridden or ignored, as with the position-based approach previously discussed, they work alongside the simulation.

Overall, the “Character Controller Interface” is used in this test scene to effectively showcase the dynamic nature of the simulation. It can work in a variety of different contexts and alongside a variety of different inputs as a result of a force-based approach.

**5.7 Swing from Start Interface**

The “Swing from Start Interface” was developed to showcase the simulation working in isolation from outside forces and over an extended time period.

This interface requires the user to specify a “GameObject” as a pivot in the inspector so that, upon pressing play. “StartSwing()” is called on “Awake()”, and the simulation plays indefinitely.

The values of the parameters for this scene are, excluding the drag value, those entered when clicking the “Default Values” button:

|  |  |
| --- | --- |
| **Variable** | **Value** |
| Mass | 1 |
| Drag | 0 |
| Interpolate | True |
| Use Custom Timestep | False |
| Custom Timestep Value | (N/A) |
| Jump on Start Swing | False |
| Start Jump Force | N/A |
| Swing Force Multiplier | 1 |
| Max Force | 100 |
| Min Force | 5 |
| Show Cord | True |
| Cord Width | 0.1 |

Fig. 29: Variable configurations for the “Swing from Start Interface”

The default drag value is 0.001. This has been set to 0 for this example so the user can observe the simulation’s behaviour over time without damping.

Inside the “Swing from Start Scene” is a background wall and two “GameObjects” acting as the “bob” and the “pivot” (figure 30). The environment is kept intentionally bare, and no additional forces are added to the “bob” object in order to observe the behaviour of the simulation over a period of time.

A white ball and blue cube with green line and a white ball

AI-generated content may be incorrect.

Fig.30: A screenshot of the Swing from Start Scene, after around 15 seconds have passed

Initially the bob swings smoothly and the amplitude is consistent, emulating the Simple Pendulum’s Harmonic Motion in angles of < 15 degrees (Huygens, 1672). However, the “bob” ‘s arc becomes increasingly more inconsistent as time passes (see figure 31).

A blue cube with green and purple lines and a white ball

AI-generated content may be incorrect.

Fig.31: A screenshot of the Swing from Start Scene after around 70 seconds have passed

This could be for several reasons. External forces that have been unaccounted for could be affecting the simulation as time goes on, as a result of the force-based approach. Unity’s (Unity Technologies, 2023) physics system could also be automatically discarding queued physics loops to reduce the time taken to prepare a frame (Unity Technologies, 2024). Further research and examination are needed in this area to identify the root cause of this issue and implement an effective solution.

Slight jittery behaviour was also observed in lower framerate devices. However, this was without the altering of the Time.FixedDeltaTime value. Checking the “Use Custom Timestep” value results in the simulation running for longer without the inconsistent behaviour.

**6. Discussion of outcomes**

Overall, the objective of this research project was to conduct in-depth research into software design principles and the mechanics of The Pendulum swing, to develop a modular simulation of the pendulum that can be integrated into other projects and used in the development of dynamic swinging mechanics in video games.

The final artifact for this project consists of a Unity project containing a “package” folder. This folder holds the “Pendulum Script” component and its editor extension script. To demonstrate the simulation’s flexibility, two test scenes were created featuring two corresponding interfaces. While the “Playground Scene” showcases the simulation acting in a more dynamic fashion, the “Swing from Start Scene” displays the simulation working in isolation from other environment objects and impulses.

As discussed in the literature review, simulations of The Pendulum swing have been carried out using both The Simple and Compound models numerous times. The simulation aspect of the artifact is not new and, due to the reasons discussed in this report, is also not the most successful in achieving a perfectly accurate representation of The Simple Pendulum.

The simulation itself is similar to those created by The Colorado Institute of Technology (Wijaya *et al*, 2021) and The University of Stanford (Johnson, N.A) in how users can adjust its behaviour through the tweaking of certain variables. In both of these examples, as well as the artifact for this research project, parameters like the “bob” ‘s mass and drag values can be adjusted. Unique to this project are the gameplay-specific parameters; namely the “Jump on Start”, “SwingForceMultiplier” and “Max/Min Force” variables.

Also unique to this project, in an academic context, is the development of the Pendulum simulation directly in adherence with principles of OOD and User Centred Design. This has resulted in modular simulation that can be added as a component to any “GameObject” in a Unity scene.

**6.1 Simulating the Pendulum**

The first research question asks “How successfully can The Pendulum Swing be simulated using the Unity game engine?”

This project attempts to answer this question by providing a simulation of The Simple Pendulum (Huygens, 1672) that functions by calculating its tension force and applying it directly to the “bob” ‘s RigidBody.

This method was successful in recreating the pendulum movement. The “Playground Scene” in the artifact can be used to showcase the simulation running in multiple axes and speeds, as discussed in the practice section.

Despite The Pendulum swing being successfully implemented, are small issues with the smoothness of the movement in instances where the simulation is running for an extended period of time. Considering the use of a real-time physics engine for this project, it can be argued that complete numerical accuracy is unobtainable due to its prioritisation of performance over complete physical accuracy. The option to increase the number of times the physics loop runs in a singular frame has been added to try and solve this issue. In hindsight, however, this only mitigates the issue. Instead of eliminating the smoothness issues completely, they are just absent for a longer time period.

Given a larger amount of time and resources, more in-depth research into the functionality of real-time physics engines in order to better understand the topic and develop a more suitable solution to this problem. While research was done into the general behaviour of real-time physics engines, the project would benefit from more specialized knowledge.

**6.2 User Centred Design**

The second research question asks: “To what extent can the aforementioned simulation be developed in a way that allows for easy use and integration into other video game developers’ projects, particularly those creating their own swinging mechanics?”

The project’s attempt to answer this question involved the creation of the artifact as an “object” in line with Object Orientated Design (OOD) principles, alongside the addition of several variables that the user can adjust to affect the simulation. The inspector window for the simulation was then extended to present the variables in a way that aligns with User Centred Design (USD) principles.

Development of the simulation in a way that allows for easy integration into other Unity projects has been mostly successful. The force-based implementation of the Pendulum Swing results in a simulation that functions in the context of its physical environment, reacting to additional forces and colliding with other “GameObjects” (see figure 31).

A computer generated image of a pole with balls and lines

AI-generated content may be incorrect.

Fig.31: A screenshot of Playground Scene gameplay, showing the player colliding with objects while swinging

The “default values” button in the inspector also adds to the project’s success in answering the second research question due to the ease of implementation it allows for. The “Swing from Start” scene in particular demonstrates this.

Admittedly, the overriding of the mass and drag values on the “bob” ‘s “RigidBody” component can be problematic in cases where the simulation is being added to a pre-configured physics object. For example, adding the “Pendulum Script” component with its default configuration to an object that has its mass set to 5 in the inspector would cause the mass to be set to 1 once the play button is pressed. A viable solution to this problem, if the project was to be extended, is to make the overriding optional and implement custom mass and drag values that affect the simulation exclusively.

Unity’s IMGUI system has been used alongside Engine Attributes in an attempt to provide a high level of user experience when adjusting the simulation. This has been moderately successful. The overall layout of the inspector displays relevant variables close to each other (figure 32), in line with the Gestalt Proximity Principle (Lowdermilk, 2023) and makes use of serialized fields in order to allow private variables like the “Start Jump Force” to be altered. Floats restricted between two values are displayed using sliders (figure 32), communicating this restriction without an excessive use of text.

A screenshot of a computer

AI-generated content may be incorrect.

Fig.32: Sliders in the “Pendulum Script” inspector window

However, there are also elements of the inspector that are lacking when considering USD. The lack of progressive disclosure (Nielson, 2006) of variables such as the “Start Jump Force” and “Custom Timestep Value” make it immediately unclear whether these values are in use. Toggling the visibility of these variables based whether their corresponding Booleans are checked is a viable solution for this issue.

It is important to note that, overall, evaluation of the project’s success in answering the second research question is significantly limited by the lack of user testing. Comparison of the artifact’s adherence with principles of OOD and UCD can only prove its theoretical position. Given more time and resources, focus should be placed on conducting extensive user testing to determine the extent to which issues like the overriding of “RigidBody” values and lack of progressive disclosure effect the overall usability of the simulation for developers.

**7. Conclusion and recommendations**

In conclusion, this project presents a dynamic, modular simulation of The Simple Pendulum that can theoretically be integrated into other projects for use by other video game developers.

The initial position-based simulation of The Pendulum, while successful in answering the first research question, could not be used in numerous contexts due to the “Set Position()” function overriding any forces working on the “bob” and not considering collisions. Consequently, this approach was abandoned for a force-based approach.

The force-based approach for this project was mostly successful, with only slight smoothing issues observed after an extended amount of time.

The simulation’s modular nature alongside its ability to work in a variety of contexts and with different interfaces exemplifies the project’s general success in answering the second research question. The artifact’s presentation of the adjustment options available is passable, being developed in line with common UCD principles, but it is not clear when certain variables are used in the simulation and when they are deactivated.

The lack of user testing for this project means that a confident conclusion cannot be reached on the extent to which the simulation is easily integrable into other projects. Further work on this project should prioritise this, preferably on a focus group of video game developers familiar with the Unity Game Engine (Unity Technologies, 2023). Questions asked during this research phase should relate to user experience, the clarity of parameters’ functions, ease of implementation of the simulation into the developers’ own projects and overall User Experience. The data gathered can then be used to ascertain the extent to which issues like RigidBody values being overridden affect the overall user experience, and well-informed, relevant solutions can be carried out.

Alongside user testing, the project would benefit from more specialized research into the functions of real-time physics engines so that the issues with the smoothness of the simulation over time can be more effectively addressed.

Considering all functionality for the simulation is contained within the “Pendulum Script” alongside its modular nature reveals the potential for this artifact to be turned into an official developer tool. After user testing and addressing of the project’s key issues has taken place, the “package” folder containing the simulation functionality can be converted into an asset package and put onto the Unity Asset Store (Unity Technologies, 2025).

**8. References**

Academia, Inc (2008) *Academia.Org* [online]. Available From: <https://www.academia.edu>

Akkaya, A and Akpinar. Y. (2022) Experiential serious-game design for development of knowledge of object-oriented programming and computational thinking skills. *Computer Science Education* [online] 32, p. 476-501

A. Kuo, J. L. Hiler, and R. J. Lutz (2017) From super mario to skyrim: A framework for the evolution of video game consumption, *Journal of Consumer Behaviour.* [online] 16, pp. 101–120.

Baboard, F (2025) *Engineering the wobble: how to (not?) engineer a co-op open world physics-based farming sim* [Talk] Manchester Metropolitan University. April 10.

Backer (2011) *Games, Rules and Immersion.* [Online]. <https://www.gamedeveloper.com/design/games-rules-immersion>

Bayliss, J.D (2022) *The Data-Orientated Design process for game development.*

Bleeding Edge Studios (2024) *Mastering Game Development: Unveiling the power of OOP in Gaming Concepts 2024.* Available From: <https://bleedingedge.studio/blog/game-development-oop-in-gaming-concept-24/>

Browne, C. (2021) *7 User Centred Design Principles*. [online]. Available From: <https://careerfoundry.com/en/blog/ux-design/user-centered-design-principles/#seven-user-centered-design-principles>

Dassault Systemes (2020) *CAITA V5.* [Computer Program]. Available From: <https://www.3ds.com/products/catia/catia-v5>

Dathan, B. and Ramnath, S. (2015) *Object-Oriented Analysis, Design and Implementation: Undergraduate Topics in Computer Science* [online] Springer International Publishing.

Del Río-Cidoncha, G. (2020) Computer-Aided Design and Kinematic Simulation of Huygens’s Pendulum Clock. *Applied Sciences.* [online]. 10, p.538

Delise, F. (2022) *Anyone can build metaverse applications with new beta release of NVidia omniverse*. [online]. Available from: <https://developer.nvidia.com/blog/anyone-can-build-metaverse-applications-with-new-beta-release-of-nvidia-omniverse/>.

Epic Games (2022) *Unreal Engine 5* (2022) [Computer Program] Available from : https://www.unrealengine.com/en-US/download

Fabian, R. (2013) *Data Orientated Design.* [online]. Available From: https://www.dataorienteddesign.com/dodmain.pdf

Fehervari, Z. (2024) *Game Physics: If You Wanna Understand Gaming More* [online]. Available From: <https://medium.com/@fhrvri.mmxiv/game-physics-if-you-wanna-understand-gaming-more-bafae7414e6e>.

Google (2004) *Google Scholar* [online]. Available From: https://scholar.google.com

Hafez, D.Y.I. (2015) *Free Code for Simulation of Pendulum Motion Using Numerical Integration*.

H. Cook and L Nordtvevt, K. (2025) *Gravity.* [online]. Available From: https://www.britannica.com/science/gravity-physics

Havok (2024) *Havok.* [Computer Program]. Available From: https://www.havok.com

Huygens, C. (1672) *Horologium Oscillatorium*. Huygens, C.

Insomniac Games (2018) *Marvel’s Spiderman.* [Video Game]. Sony Interactive Entertainment. Available From: <https://www.playstation.com/en-gb/games/marvels-spider-man/>

Insomniac Games (2023) *Marvel’s Spiderman 2.* [Video Game]. Sony Interactive Entertainment. Available From: <https://www.playstation.com/en-gb/games/marvels-spider-man-2/>

Interaction Design Foundation (2025) *What is Uder Centered Design (UCD)?* [Online]. Available From: https://www.interaction-design.org/literature/topics/user-centered-design

Johan-Dahl, O. and Nygaard, K. (1962) *Simula* [online]. Available from: www.simula67.info.

Johnson, J. (N.A) *Pendulum Simulator.* [online]. Available from: <https://cs.stanford.edu/people/jcjohns/pendulum/>.

Landfall Games (2025) *Haste.* [Video Game]. Landfall Games. Available From: <https://store.steampowered.com/app/1796470/Haste/>

Lowdermilk, T. (2023) *User Centered Design: A Developer's Guide to Building User-Friendly Applications.*[online].O’Reilly Media.

Maranowski, M. (2024) Incorporating Physics in Video Games. [Online]. *Science Buddies*. Available From: <https://www.sciencebuddies.org/science-fair-projects/project-ideas/Games_p031/video-computer-games/physics-in-video-games>

Mitsiou (2024) *What is User-Centered Design (UCD)? W/process* [online] Available From: https://blog.uxtweak.com/user-centered-design/

M. Templett, R. (2020). *Game Physics: An Analysis of Physics Engines for First-Time Physics Developers.* Msc. California State University.

Naughty Dog (2016) *Uncharted 4: A Thief’s End.* [Video Game]. Sony Computer Entertainment. Available From: https://www.playstation.com/en-gb/games/uncharted-4-a-thiefs-end/

Nave, R. (N.A) *Physical Pendulum.* Available From: http://hyperphysics.phy-astr.gsu.edu/hbase/pendp.html

Newton, Roger G. (2005) *Galileo’s Pendulum*. Harvard University Press.

Nielsen, J. (2007) *Progressive Disclosure* [online]. Available From: https://www.nngroup.com/articles/progressive-disclosure/

Nvidia Corporation (2024) *PhysX* (5.0.6) [Computer Program]. Available from: <https://www.nvidia.com/en-us/drivers/physx/physx-9-23-1019-driver/>

Nvidia Corporation (2017) *Best Practices Guide.* Available From: https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/BestPractices.html

Palka, L., Schauer, F. and Dostal, P. (2016) Modelling of the simple pendulum Experiment. *MATEC Web of Conferences* [online]. 76, p. 04037.

Radical Forge (N.A) *Southfield.* [Video Game] Radical Forge. Available From: <https://store.steampowered.com/app/2076210/Southfield/>

Madisch, I Hofmayer, S and Fickenscher, H. (2008) *ResearchGate* [Online]. Available From: <https://www.researchgate.net>

Santa Monica Studio. (2007) *God of War 2*. [Video Game] Sony Computer Entertainment.

State, G. and Handa, A. (2022*) Reinforcing the Value of Simulation by Teaching Dexterity to a Real Robot Hand* [online]. Available from: https://developer.nvidia.com/blog/reinforcing-the-value-of-simulation-by-teaching-dexterity-to-a-real-robot-hand/

Unity Technologies (2023) *Unity.* (2022.3.6f1)[Computer Program]. Available From: <https://unity.com/releases/editor/whats-new/2022.3.6#third-party-notices>

Unity Technologies (2024) *Introduction to RigidBody Physics.* Available From: <https://docs.unity3d.com/Manual/RigidbodiesOverview.html>

Unity Technologies (2024) *Enhanced physics performance for smooth gameplay.* [online]. Available From: https://unity.com/how-to/enhanced-physics-performance-smooth-gameplay

Unity Technologies (2025) *Unity Asset Store.* [Online]. Available From:https://assetstore.unity.com

UWE Bristol (2024) *UWE Bristol handbook of research ethics.* (2024)[Online]. Available From: https://www.uwe.ac.uk/-/media/uwe/documents/research/uwe-bristol-handbook-of-research-ethics.pdf

Whitman, L. Tanovic, A. (2024) *User-centered design: 4 key principles, phases, and examples.* [online]. Available From: https://maze.co/blog/user-centered-design/

Wijaya, P.A., Widodo, A., and Muslim (2021) Virtual experiment of simple pendulum to improve student’s conceptual understanding. *Journal of Physics: Conference Series* [online]. 1806 (1), p. 012133

Winngst, D, Wickstrom, F and Memeti, S (2022) Evaluating the performance of object-oriented and data-oriented design with multi-threading in game development. In: *2022 IEEE Games, Entertainment, Media Conference (GEM)* [online] St. Michael, Barbados: IEEE, pp. 1-6. Available From: https://ieeexplore.ieee.org/document/10017610/

**Appendix A: Project Log**

|  |  |
| --- | --- |
| Month | Notes |
| November |  |
| 15/11/2024 | Base simulation has been implemented by calculating the object’s position next frame and applying it using setPosition(). There is a bug with the speed of the swing though |
| 20/11/2024 | Base simulation inputted but only works across one axes. |
| 25/11/2024 | Commercial Games Development alpha demonstration |
| 26/11/2024 | Meeting with Supervisor highlighted concerns with set position approach overriding other aspects of the game. Consider Second research question. |
| December |  |
| 3/12/2024 | Set Position approach discarded for Force-based approach. Working on devising an equation based on force diagram of The Simple Pendulum |
| 10/12/2024 | Tension Force equation for the simulation has been devised and implemented. Simulation works but is very jittery. Unsure why. |
| 12/12/2024 | Advanced Technologies module deadline |
| 28/12/2024 | Not much work has been done since last log. Research into physics in Unity shows that messing with the timestep might fix the issue? |
| 30/12/2024 | Using a smaller Timestep value solves jittering problems in most cases. |
| January |  |
| 12/01/2025 | Made custom timestep optional. Added options to tweak RigidBody mass, drag and interpolate values also. |
| 16/01/2025 | Poster + progress report submission |
| 20/01/2025 | Poster Presentation |
| 24/01/2025 | Meeting with supervisor to discuss poster presentation feedback. Simulation is solid but user experience and research into design side of things is lacking. |
| February |  |
| 01/02/2025 | Added swing force multiplier and start jump force. Start jump force does not work properly. |
| 02/02/2025 | Found key issue with simulation equation and fixed. |
| 30/02/2025 | Meeting with supervisor. Not much done this month. Start jump force works properly now. |
| March |  |
| 05/03/2025 | Commercial Games Development beta demonstration |
| 06/03/2025 | Began working on Character Controller Interface. Basic character controller implemented. |
| 20/03/2025 | Character Contoller Interface implemented with Raycast based target detection system |
| 25/03/2025 | Created Swing from Start Interface |
| April |  |
| 1/04/2025 | Created Swing from Start Scene |
| 5/04/2025 | Meeting with supervisor for final artifact feedback. |
| 28/04/2025 | Commercial Games Development gold demonstration |
| 29/04/2025 | Finished report. |
| May |  |
| 1/05/2025 | CCTP and Advanced Technologies Submission |

**Appendix B: Project Timeline**

A screenshot of a graph

AI-generated content may be incorrect.

**Appendix C: Assets used in the Project**

All assets, code and text in this project are original.

**Further Appendixes D, E …**