

Comparison of game engines for PC game development

Ayman Abdullah Jawhar

Bachelor's thesis
2023



Tomas Bata University in Zlín
Faculty of Applied Informatics

ASSIGNMENT OF BACHELOR THESIS

(project, art work, art performance)

Name and surname: Ayman Abdullah Jawhar
Personal number: A19902
Study programme: B0613A140021 Software Engineering
Type of Study: Full-time
Work topic: Porovnání herních engine pro vývoj počítačových her
Work topic in English: Comparison of Game Engines for PC Game Development

Theses guidelines

1. Provide a brief description of a game development process, including standard game development requirements. Keep the focus on game development for personal computers.
2. Describe the general parts of game engines, their types and programming languages used for game development.
3. Define a set of parameters suitable for the game engine comparison, including a game scene to implemented in selected engines.
4. Select game engines properly and compare them.
5. Discuss the comparison results and provide proper recommendations.

Form processing of bachelor thesis: **printed/electronic**

Recommended resources:

1. HALPERN, Jared. Developing 2D Games with Unity: Independent Game Programming with C#. APress, 2018. ISBN 1484237714.
2. BARRERA, Ray, Aungh Sithu KYAW a Thet Naing SWE. Unity 2017 Game AI Programming. 3rd ed. Birmingham: Packt Publishing, 2018. ISBN 978-1-78847-790-1.
3. O'DONNELL, Casey. Developer's dilemma: the secret world of videogame creators. Cambridge, Massachusetts: The MIT Press, [2014], 1 online zdroj (xii, 337 pages). Inside technology. ISBN 9780262322836.
4. DORAN, John P. Unity 2020 mobile game development: discover practical techniques and examples to create and deliver engaging games for Android and iOS. Second edition. Birmingham: Packt>, 2020, iv, 453 s. ISBN 978-1-83898-733-6.
5. SALMOND, Michael. Video game design: principles and practices from the ground up. London: Fairchild Books, 2018, 1 online resource (272 p.). ISBN 9781474222426.
6. NAKATSU, Ryohei, Matthias RAUTERBERG a Paolo CIANCARINI. Handbook of digital games and entertainment technologies. Singapore: Springer, [2016], 1 online resource (2 volumes). Dostupné z: doi:9789814560504

Supervisors of bachelor thesis: **Ing. Peter Janků, Ph.D.**
Department of Informatics and Artificial Intelligence

Date of assignment of bachelor thesis: **October 5, 2022**

Submission deadline of bachelor thesis: **May 12, 2023**

doc. Ing. Jiří Vojtěšek, Ph.D. m.p.
Dean



prof. Mgr. Roman Jašek, Ph.D., DBA m.p.
Head of Department

In Zlín October 10, 2022

I hereby declare that:

- I understand that by submitting my Bachelor's Thesis, I agree to the publication of my work according to Law No. 111/1998, Coll., On Universities and on changes and amendments to other acts (e.g. the Universities Act), as amended by subsequent legislation, without regard to the results of the defence of the thesis.
- I understand that my Bachelor's Thesis will be stored electronically in the university information system and be made available for on-site inspection, and that a copy of the Bachelor's Thesis will be stored in the Reference Library of the Faculty of Applied Informatics, Tomas Bata University in Zlín, and that a copy shall be deposited with my Supervisor.
- I am aware of the fact that my Bachelor's Thesis is fully covered by Act No. 121/2000 Coll. On Copyright, and Rights Related to Copyright, as amended by some other laws (e.g. the Copyright Act), as amended by subsequent legislation; and especially, by §35, Para. 3.
- I understand that, according to §60, Para. 1 of the Copyright Act, TBU in Zlín has the right to conclude licensing agreements relating to the use of scholastic work within the full extent of §12, Para. 4, of the Copyright Act.
- I understand that, according to §60, Para. 2, and Para. 3, of the Copyright Act, I may use my work - Bachelor's Thesis, or grant a license for its use, only if permitted by the licensing agreement concluded between myself and Tomas Bata University in Zlín with a view to the fact that Tomas Bata University in Zlín must be compensated for any reasonable contribution to covering such expenses/costs as invested by them in the creation of the thesis (up until the full actual amount) shall also be a subject of this licensing agreement.
- I understand that, should the elaboration of the Bachelor's Thesis include the use of software provided by Tomas Bata University in Zlín or other such entities strictly for study and research purposes (i.e. only for non-commercial use), the results of my Bachelor's Thesis cannot be used for commercial purposes.
- I understand that, if the output of my Bachelor's Thesis is any software product(s), this/these shall equally be considered as part of the thesis, as well as any source codes, or files from which the project is composed. Not submitting any part of this/these component(s) may be a reason for the non-defence of my thesis.

I herewith declare that:

- I have worked on my thesis alone and duly cited any literature I have used. In the case of the publication of the results of my thesis, I shall be listed as co-author.
- That the submitted version of the thesis and its electronic version uploaded to IS/STAG are both identical.

In Zlín; dated:

.....
Student's Signature

ABSTRAKT

Toto téma je zaměřeno na veřejně dostupné herní enginy, které jsou často první volbou programátorů, kteří chtějí vyvíjet nové hry a/nebo aplikace s 3D prostředím. Každý den jsou vytvářeny nové herní enginy a stávající enginy jsou aktualizovány. Cílem této práce je vybrat vhodné a aktuální herní enginy, sestavit soubor pravidel pro jejich porovnání a provést srovnání. Výsledek této práce by měl být užitečný pro každého programátora, který se chce zorientovat ve světě herních enginů.

Klíčová slova: herní motory, 3D, programátor

ABSTRACT

This topic is focused on public available game engines, which are often the first pick of programmers, who want to develop the new games and/or applications with 3D environments. Every day new game engines are created and the existing engines are updated. The aim of this thesis is to select suitable and recent game engines, build a set of rules for its comparison and do the comparison. The result of this work should be useful for every programmer who wants to become oriented in the world of game engines.

Keywords: game engines, 3D, programmer

ACKNOWLEDGEMENTS

I would like to give all of my thanx to my family. They have faced too many hardships to enable me to achieve this. I will forever be indebted to them.

I hereby declare that the print version of my Bachelor's/Master's thesis and the electronic version of my thesis deposited in the IS/STAG system are identical.

Contents

INTRODUCTION	11
I. THEORY.....	12
1 GAME DEVELOPMENT	13
1.1 GAME DEVELOPMENT REQUIREMENTS	13
1.1.1 Hardware Requirements	13
1.1.2 Knowledge Requirements.....	14
1.1.3 Game Development Requirements Summary.....	14
1.2 GAME DEVELOPMENT PROCESS.....	14
1.2.1 Pre-production Stage.....	15
1.2.2 Production Stage	15
1.2.3 Testing Stage.....	16
1.2.4 Release Stage	16
1.2.5 Post-release Stage	17
1.2.6 Game Development Process Summary.....	17
2 GAME ENGINES	18
2.1 GAME ENGINE PARTS	18
2.1.1 Rendering Engine	18
2.1.2 Physics Engine.....	19
2.1.3 Audio Engine	19
2.1.4 Artificial Intelligence.....	19
2.1.5 Networking	19
2.1.6 Game Engine Parts Summary	20
2.2 GAME ENGINE TYPES	20
2.2.1 Game engines based on supported dimensions.....	20
2.2.2 Game engines based on target platforms.	20
2.2.3 Game Engines based on availability.....	21
2.3 GAME ENGINE PROGRAMMING LANGUAGES	22
2.3.1 C++	22
2.3.2 C#.....	22
2.3.3 JavaScript.....	23
2.3.4 Java	23

2.3.5	Lua	23
2.3.6	Game Engine Programming Languages Summary	24
2.4	VISUAL PROGRAMMING IN GAME ENGINES	24
2.4.1	Unreal Engine	24
2.4.2	Unity	25
2.4.3	Godot	25
2.4.4	Construct 3	25
2.4.5	GameMaker Studio 2	25
2.4.6	Scratch	25
2.4.7	Visual Programming in Game Engines Summary	26
3	POPULAR GAME ENGINE	27
3.1	UNITY	27
3.1.1	History.	27
3.1.2	Games made with Unity.	27
3.1.3	Programming languages and visual programming.	27
3.1.4	Unity's Financial Model.	27
3.1.5	Supported game types.	28
3.1.6	Supported platforms.	28
3.1.7	Unity's Community, Learning, and Documentation.	28
3.1.8	Unity's Asset Store.	28
3.1.9	Job prospects.	28
3.1.10	Unity's Drawbacks.	28
3.1.11	Latest updates.	29
3.1.12	Unity Summary.	29
3.2	UNREAL ENGINE	29
3.2.1	History.	29
3.2.2	Games made with Unreal Engine.	29
3.2.3	Programming languages and visual programming.	29
3.2.4	Unreal Engine's Financial Model.	30
3.2.5	Supported game types.	30
3.2.6	Supported platforms.	30
3.2.7	Unreal Engine's Community, Learning, and Documentation.	30
3.2.8	Unreal Engine's Market Place.	30

3.2.9	Job prospects.....	31
3.2.10	Unreal Engine's Drawbacks.	31
3.2.11	Latest updates.	31
3.2.12	Unreal Engine Summary.....	31
3.3	CRYENGINE	31
3.3.1	History.	32
3.3.2	Games made with CryEngine.	32
3.3.3	Programming languages and visual programming.	32
3.3.4	CryEngine's Financial Model.....	32
3.3.5	Supported game types.....	32
3.3.6	Supported platforms.....	32
3.3.7	CryEngine's Community, Learning, and Documentation.	32
3.3.8	Job prospects.....	33
3.3.9	CryEngine's Drawbacks.	33
3.3.10	Latest updates.	33
3.3.11	CryEngine Summary.....	33
3.4	GADOT.....	33
3.4.1	History.	34
3.4.2	Games made with Godot.....	34
3.4.3	Programming languages and visual programming.	34
3.4.4	Godot's Financial Model.	34
3.4.5	Supported game types.....	34
3.4.6	Supported platforms.....	34
3.4.7	Godot's Community, Learning, and Documentation.....	35
3.4.8	Godot's Asset Library.....	35
3.4.9	Job prospects.....	35
3.4.10	Godot's Drawbacks.....	35
3.4.11	Latest updates.	35
3.4.12	Godot Summary	35
II.	PRACTICAL.....	37
4	SHOWCASING CHOSEN GAME ENGINES	38
4.1	UNITY	39
4.2	UNREAL ENGINE 5	40

5	TESTS FOR BENCHMARKING GAME ENGINES.....	41
5.1	CPU LOAD TESTING	41
5.1.1	Unity	42
5.1.2	Unreal Engine 5	45
5.1.3	Results Analysis.....	48
5.2	GPU LOAD TESTING	49
5.2.1	Unity	51
5.2.2	Unreal engine 5	54
5.2.3	Results Analysis.....	56
5.3	MEMORY USAGE TESTING	57
5.3.1	Results and Analysis	58
6	SOFT DATA COMPARISONS.....	60
6.1	EASE OF USE.....	61
6.2	ACCESSIBILITY	61
6.3	FIDELITY.....	62
6.4	IN ENGINE PERFORMANCE	63
6.5	SCRIPTING LANGUAGE.....	64
6.6	REGULAR UPDATES AND FUTURE ROADMAP	64
6.7	TOOLS AND FEATURES.....	65
	CONCLUSION	67
	BIBLIOGRAPHY.....	69
	LIST OF ABBREVIATIONS	74
	LIST OF FIGURES	76
	LIST OF TABLES	77
	APPENDICES.....	78

INTRODUCTION

Video games have emerged as a popular and influential form of entertainment. More and more people are starting to get into game development with game engines now being much more accessible than in the past. Game engines have also advanced enough that it's possible for solo developers to create immersive video game experiences or in general develop the 3D applications they want.

The aim of this thesis is to give a general introduction to game development, and dive deeper into the game engines that are used to create games. Explaining game engines in the theoretical part and talk about some popular game engines, while doing comparison between the two most popular of those game engines in the practical part, and giving a clear recommendation for those who want to get into game engines, but are not yet familiar enough with the industry to know where to start.

The theoretical part starts with giving a brief introduction to game development, then talks about game development requirements and the game development process. Next, the thesis talks about game engines, their parts, types, programming languages, and visual programming options. Having explained what game engines are to the reader, the thesis then talks about some examples of popular game engines and talks about them in a bit more detail and transitioning to the practical part.

The practical part opens with giving example of a scene being rendered in the two chosen game engines and showing their rendering capabilities. Then move to the hard data comparisons between the two game engines. The chapter will go over three tests and describe their results. The tests are CPU load testing, GPU load testing, and memory usage testing. However, the comparison is not limited to just hard data comparisons. The thesis next starts comparing the two game engines in areas that would be difficult to give an objective numerical evaluation of, thus making this chapter in small parts influenced by the author and then closing the practical part.

The author finally gives a recommendation of which game engine is best to choose. However, the reader should now be familiar enough with game engines and the industry to be able to also make their own independent decisions with the backup of hard data.

I. THEORY

1 GAME DEVELOPMENT

In simple terms, game development is the process of creating video games, and the individuals who take part in creating the game are called game developers. Video games are some of the most complex pieces of software where many disciplines come together and work as a team in order to create a complete product. Those disciplines may include 2D/3D artists, programmers, level designers, audio producers, story writers, etc. Game development is usually handled by large teams in large game development studios, that are responsible for creating big-budget AAA games with the highest qualities. However, there are still individuals who work independently or in small teams to create small games with the limited budget that they have, and they are usually called indie developers. Regardless of the size of the game, game development usually follows certain steps that are recognized by the game development industry. In general, those steps can be summarized into concept design, production, testing, release, and post release support. The focus of this chapter is to provide a brief introduction to game development, talk about different game development requirements, and describe the different stages of game development in more detail. [1]

1.1 Game Development Requirements

Conventional wisdom holds that one cannot simply step into the world of game development without any prior experience in any of its fields. For someone to become a game developer, they first would have to pass two major requirements. Those requirements being hardware requirements and knowledge requirements.

1.1.1 Hardware Requirements

To learn game development and creating video games, one should have good hardware that makes the process comfortable. In order to create video games using the latest technologies with decent details and fidelity, it is required to have at least a mid-tier modern gaming laptop or the equivalent. Modern game engines demand hardware with powerful graphics and processing power to have a decent game development experience. Attempting to develop games using low-tier or very old hardware will make the game development experience terrible.

1.1.2 Knowledge Requirements

To become a game developer, it is recommended to know at least one popular scripting language in game development like C# or C++ on a medium level and core principles like OOP. Although modern game engines like Unity and Unreal Engine have visual scripting options built in, and examples have been shown of creating games entirely using visual scripting options provided by these game engines, knowing one of the mentioned programming languages only compliments those visual scripting options and makes learning them easier. Also, it is recommended to be familiar with the latest trends in the game development world. The game development world keeps changing with new trends and new technologies coming out. It is necessary for the game developer to keep a lookout for them if they want to advance their knowledge and keep making better games in the future that are competitive with other games in the market. Finally, having decent artistic skills in terms of game, audio, 3D model design, etc. While developing games, the game developer will continuously be interacting with these areas of game development, and although some of them can be outsourced or handled by other members of a team, it is still required to know how to interact with them and integrate them into your game. Having at least enough knowledge to recognize when a 3D model or a sound is out of place in the game world is a must, and being able to fix or improve them would make for a better game developer. [3][4]

1.1.3 Game Development Requirements Summary

It is recommended to have at least a mid-tier gaming machine, decent programming skills in a scripting language, keeping up with the latest trends, and finally, knowing the basics of the other fields of game development than programming.

1.2 Game Development Process

As was mentioned before, games are some of the most complex pieces of software, and developing these games requires a certain amount of planning. It is not possible to just start making a game without any planning and expect to finish within a timely manner while also adhering to a limited budget. When developing a game, most studios go through a very similar development process, and there certainly are studios that might have a development process unique to them, but the majority of the studios follow five well-defined stages. Those stages are pre-production, production, testing, release, and post-release. Each one of these

stages includes their own steps and workflows that are overall significant and necessary for creating a finished product.

1.2.1 Pre-production Stage

Usually, the first step starts with a proposal/idea of a game, and if the proposal is approved, the game development process starts. The process begins with a small team, usually referred to as the Core Team (which usually include a designer, artist, and a programmer) who are in-charge of identifying the requirements and the cost of developing the game.

These include creating the Game Design Document (GDD) which is a document that provides an outline of the game and how it should be developed moving forward. Defining the art-direction of the game, which includes the style of the characters, environment, animation, visual effects, etc. and identifying the technical requirement of the assets like polygon count, resolution, and animation. Defining the technologies that would be used, that includes choosing a game engine (choosing an already available game engine or creating a new one depending on the requirements) and defining the overall project architecture. Finally, identifying the cost, like how long it will take, the number of team members needed, and the budget needed to develop the game. [6]

1.2.2 Production Stage

Arguably the most important stage of the development process. This is the stage where most of the development time takes place.

The production stage also goes through its own stages. The most important stages being Prototype stage, where a basic concept of the game is created to show the core elements of the game. This stage decides if the game idea is worth pursuing further, since many ideas may seem promising, but after creating a concept, prove to be a failure. First playable stage, where it's possible to get a feel for how the game will turn out. Higher quality assets are put in the game and it's possible to test gameplay features. The vertical slice stage, where a small portion of the game is prepared to be fully playable. The playable portion of the game is then used to show to the investors and partners, and to advertise the game to create publicity. The Alpha stage, where it's possible to play the game entirely from start to finish. Although some of the art assets may be unfinished, all the main elements like functions and mechanics should be complete. The Beta stage, where all the game features and assets should be

complete. This stage focuses on testing the game, fixing bugs, and optimizing the game, rather than adding new features to the game.

Important developments in the production stage. These developments could include Art production, where the visual artists create the visual assets. The visual assets could be 2D and 3D models, textures, animations, VFX, etc. Audio production, where sound designers create sound effects which create an immersive sound environment that with the visual assets bring life into the game world. The sound designers are also responsible for creating soundtracks, background noise, menu sound effects, etc. Level Design, where game and level designers create the game world and make sure the player has a good time and won't get into problems or confused when traveling through the game world. Programming, where game developers and programmers create the functions and mechanics to play the game. They bring together all the elements that were mentioned before and create an engaging gameplay experience for the player. [5]

1.2.3 Testing Stage

In this stage, testing and QA make sure that the game has as little number of bugs as possible. In the game development world, it's always impossible to publish a game without any bugs since games are some of the most complex pieces of software, so testing is one of the most important stages to make sure that every player has as close to a bug free experience as possible. Testing is done by stress testing the game world, checking for possibilities where the player can exploit the game world to get unintended advantages, or play testing to see if playing the game is an enjoyable experience. To find and identify as many issues as possible, sometimes game studios create private or public betas, where they invite as many players as they need to test the game and find the issues that couldn't be found with the limited and comparatively small number of internal testers. [5]

1.2.4 Release Stage

At this stage, everything should be finished, testing should have identified as many issues as possible, and now the game is ready to be released to the public. Shortly before the game is made available to the public, studios that have the budget for it, do marketing and advertisement for their game and try to capture as much attention as possible. This could be through creating cinematic or gameplay trailers, paying influencers to play their games, or sometimes even host big in-person events where influencers, regular people, and sometimes even celebrities are invited to participate in the event and play their game for the first time

and generating as much publicity as possible. However, since usually most of the profit from a game comes from the first few weeks of launching the game, and the current pre-order culture, there have been many examples of studios overhyping their game through showing gameplay that was not representative of the final product and profiting from the hype that was built up. Some of the most high-profile examples are No Man's Sky from Hello Games, Watch Dogs from Ubisoft, and the most recent cyberpunk 2077 from CD Project RED. [5]

1.2.5 Post-release Stage

After the game is released to the public, there are most likely still many bugs and problems that were not discovered in testing which show up now that many people were able to get their hands on the game and explore every small part of it. Developers usually keep supporting the game for some time after it is released to fix those issues. However, there have been many games recently that have a live-service model where the developers keep introducing new content to the game to retain player attentions and make most of their profit through microtransactions like skins, and other in-game items. Even if the game will have a rough start at release, developers can still make up for their mistakes at this stage. There are many examples of studios that kept supporting and fixing their game for many years after it was launched. Some of the most high-profile examples are No Man's Sky, Tom Clancy's Rainbow Six Siege, and Final Fantasy XIV: A Realm Reborn. [7]

1.2.6 Game Development Process Summary

Making a game is not a simple task, it needs proper planning and a lot of dedication to bring it from an idea into a finished product. However, it is difficult to predict how the public will receive the game until it is released, and even if the first impression is negative, developers still have a chance to curb those bad first impressions through post-launch support and make a comeback.

2 GAME ENGINES

A game engine is a software framework that's comprised of many tools which are meant to help create video games more easily and efficiently. Some examples of those tools are a level editor, a physics engine, a rendering engine, artificial intelligence, an audio engine, etc. It is certainly possible to develop a game without using a game engine and its tools, but it would most certainly be more difficult and take more time. Depending on the provided tools, game engines can also be used for more than just creating games. They could be used to make AR applications, architectural representations, simulations, etc. Game engines can be divided into different types. Some of them try to do everything, and some of them try to excel in a specific area like 2D, 3D, VR/AR, mobile, etc. Some engines are publicly available, but more engines are owned by game development studios that are private and only used by the developers in the company. The publicly available game engines have different financial models. Some of them are completely free, some of them are subscription based, and some of them are royalty based. Almost all of the game engines need a way to handle logic and a way to manipulate objects inside the game, and most of them use a programming language to do that. The two most popular programming languages used are C++ and C#. However, there are also some game engines that use a visual programming option instead, while some game engines provide support for both. [9]

2.1 Game Engine Parts

Although much more difficult, it is possible to make games without using a game engine. A game engine provides a collection of tools in a framework that makes the game development experience much less difficult. Some of the most core tools and modules provided by game engines are the rendering/graphics engine, the physics engine, the audio engine, artificial intelligence, and networking.

2.1.1 Rendering Engine

The rendering engine is a critical component of the game engine that is responsible for rendering the visual assets of the game. Depending on the way the rendering engine handles lighting, shadows, asset maps and materials, etc., it gives a unique characteristic for how everything looks and feels in the game world. A good rendering engine that handles the previously mentioned elements very well is detrimental to creating a believable look and feel to the game. [10]

2.1.2 Physics Engine

The physics engine is responsible for handling how the objects in the game world should behave. It is responsible for handling movement, velocity, acceleration, gravity, collision, etc. How well in terms of realism and accuracy the physics engine handles those elements is essential. Some physics engines of game engines also come with advanced features like ragdoll physics, soft body simulation, and vehicle physics. [9]

2.1.3 Audio Engine

The audio engine handles and controls the sounds that are generated in the game world. Audio files are stored and with the help of programming, they are triggered when interactions are happening and when they are needed. The sound engine also handles how sound should be moving depending on its source and the objects around it. A realistic sound engine is crucial in giving an immersive feel to the game world, and it is considered a necessity for competitive games that rely heavily on sound. [9]

2.1.4 Artificial Intelligence

Artificial Intelligence plays a greater role than ever in the game development world and in the world in general. Artificial Intelligence helps make everything in the game world feel more realistic when interacting with the player and with each other. This could be non-playable-characters (NPCs) reacting to the player's decisions and with each other, enemy behavior, companion AI, and puzzle-solving systems. [10]

2.1.5 Networking

Networking is responsible for connecting players across the internet or through a local network and it enables them to share the same game world creating experiences that would not be possible without it. Networking is essential especially for big multiplayer games. It makes sure that the players have a smooth experience in terms of lag and latency and that the game world and the players' movement and interactions are in sync with everyone in the game world. [9]

2.1.6 Game Engine Parts Summary

A game engine is comprised of many parts, and they are all working together to create immersive and enjoyable games. Each component of the game engine is detrimental, and it would be much more difficult to create a complete experience when one of the components is missing.

2.2 Game Engine Types

Game engines can be classified into certain types based on factors like the types of games created with them, the programming language they use, and the platforms they support. However, there are also game engines that cover more than one type. Depending on what part of the game engines are used to define types of game engines, there are way too many types to cover, However, it is possible to group some of the most popular types under three categories. Those categories are game engines based on the supported dimensions, game engines based on the platforms they target, and game engines based on availability. [40]

2.2.1 Game engines based on supported dimensions.

There are only two types of game engines that could be mentioned in this section since as humans, it's not possible to comprehend 1st dimensional, the 4th dimensional, or any other number of dimensions other than 2D and 3D.

- **2D game engines.** These engines specialize specifically in creating two-dimensional games. They often include features to aid with the creation of 2D sprites and animations, and physics tailored to a 2D environment. Examples include GameMaker Studio 2 and Godot.
- **3D game engines.** These engines have features and tools that are designed primarily for creating three-dimensional games. They typically include complex rendering systems, 3D physics engines, and support for advanced lighting and shadows. Examples include Unity, Unreal Engine, and CryEngine.

2.2.2 Game engines based on target platforms.

There are game engines that try to focus on optimizing for targeting a specific platform. Some of these platforms are AR/VR, mobile, and web.

- **AR/VR game engines.** These are engines that have specific features or extensions designed to create VR (virtual reality) or AR (augmented reality) experiences.

Although there still isn't a game engine that is designed specifically with VR/AR in mind yet, many popular game engines provide very good support for creating VR/AR games. Two of the more popular options are Unity and Unreal Engine.

- **Mobile game engines.** While many game engines support mobile platforms, some are specifically designed for mobile game development. These engines may include features for touch input, mobile ad integration, and optimization for mobile hardware. Examples include Solar2D and Cocos2d.
- **Web game engines.** These engines are designed to create browser-based games using HTML5. They include features for dealing with the limitations and advantages of browser-based game development. Examples include Phaser, Construct, and Babylon.js.

2.2.3 Game Engines based on availability.

The two types in this category are publicly available game engines like Unity, Unreal Engine 5, Godot, O3DE, etc. privately owned, multi-purpose game engines like the Frostbite engine from EA, Snowdrop engine from Ubisoft, etc. and privately owned, proprietary game engines like Overwatch, Legend of Grimrock 2, etc.

- **Publicly available game engines.** These engines are open to anyone interested in game development. They offer a suite of tools and reusable software components that simplify game creation. They are usually free to use but apply royalties or extra fees once reaching a certain level of profit.
- **Privately owned multi-purpose game engines.** These engines are very similar to the publicly available game engines in their capabilities. However, they are owned by big development studios and can only be used by developers working in those studios.
- **Privately owned, proprietary game engines.** These are game engines that are created from scratch to be used for only a single game. They offer the advantage of having very high performance compared to other game engines, but the disadvantage of being too expensive to build and maintain.

To summarize, there are game engines that try to cover as many areas as possible, and there are game engines that try to excel on one specific area. It is recommended that the developer

realizes the specific area of game development they want to become an expert at and choose the game engines that would be best suited for what games they are trying to develop.

2.3 Game Engine Programming Languages

All game engines support one or more ways for handling logic, and the most popular method is by using a programming language. Two of the most popular languages are C++ and C#. There are other languages that are used in game development that are less popular compared to C++ and C# for game development. Languages like JavaScript, Java, and Lua.

2.3.1 C++

It is by far the most popular language used in the game development world. It is adopted by many game engines and also used by many studios with their in-house built game engine. C++ is an object-oriented programming (OOP) language that is highly performant and optimized, and it provides low-level access to hardware. When used properly, it can yield very high results. It is a cross-platform language, meaning that code written in C++ can be compiled and run on a wide variety of platforms. Since it has been around for a long time, it has a big developer community and many open-source libraries that help developing games easier. C++ is a very difficult programming language to learn compared to other modern programming languages, and it is mostly not recommended as a first programming language to learn. However, it can be said that it is necessary to know how to write code in C++ in order to succeed in the game development world as a programmer. [11][12]

2.3.2 C#

It is a high-level general-purpose programming language that has better syntax and is easier to learn compared to C++. It has useful features like garbage collection, and its compiler reduces runtime errors by warning the developer in advance. It is also a cross-platform language like C++ that can be run on all types of systems. It has been popularized in the game development world for being the primary programming language of the Unity game engine, which is currently one of the most popular game engines in the market. It is also used in other parts of the software development world like web development and mobile app development with .NET, which opens up a lot of work opportunities for programmers who can write code in C#. It has a rich ecosystem with a large developer community, and it's continuously supported by Microsoft which gives it a lot of longevity for the future. [11]

2.3.3 JavaScript

JavaScript is one of the most popular programming languages. It is used in many areas in the software development world, mainly in Web development and for creating mobile web applications in a limited fashion. In terms of game development, it is used to create games that can be run in the browser with HTML5. Since those games can be run in the browser, it means that they can technically run on any device that can access the internet through a browser. JavaScript programmers take advantage of 3D libraries like Three.js and Famo.us for making games. JavaScript is also used as the main programming language in some 2D game engines, notably Construct and Phaser. JavaScript is a comparatively easy language to learn, and given its popularity, it has a very large community of developers that create libraries for practically anything. It is used by many companies in many applications making job opportunities widely available. [11]

2.3.4 Java

It is also a general-purpose high-level programming language and is similar to C# in many ways. It is used by smaller game development studios and one of the most popular games, Minecraft, is written in Java. It offers many of the same advantages as C#. In the game development world, it is mainly used in creating mobile games for Android devices since it is natively supported by Android mobile operating system. It is also popular outside of the game development world, being used in Servers and creating APIs giving it good job availability. [11]

2.3.5 Lua

Lua is lightweight, flexible, and very easy to learn programming language. It is not as popular as the previously mentioned programming languages. However, it is still responsible for some popular games like angry birds and Age of Conan. One of the most popular game platforms, Roblox, also uses a modified version of Lua called Luau. Lua, being very lightweight, makes it very fast and efficient while also having a small memory footprint. Given its flexibility, it is also easily integrated with other programming languages in game development. However, although it is lightweight and flexible, it does not offer as much functionality as the previously mentioned programming languages. Lua is used in Love2D, which is a free, easy-to-learn gam engine. [11]

2.3.6 Game Engine Programming Languages Summary

There are many programming language options for someone who wants to get into game development. but given that performance is one of the most important requirements for game development, C++ is the most widely used, given that it is one of the most powerful programming languages. However, C# is also very popular in game development and is starting to get adopted by many game engines, while also being recommended as the first programming language to learn when getting into game development.

2.4 Visual Programming in Game Engines

Although the most popular way to write game logic in game engines is by using a programming language, some of the popular game engines also offer visual programming options. Visual programming provides a graphical interface where graphical elements are used to simulate logic. One of the more popular methods is drag-and-dropping already made graphical elements and connecting them through links. Visual programming offers an easy entry for people who do not know programming to step into the game development world. They are also very useful for collaborating with other team members like level designers, 3D artists, etc. in big game development studios who don't understand programming language since visual programming is easier to understand. Some downsides of visual programming would be that it is much slower to create a game using visual programming rather than just writing the code. It takes more time to drag the elements to their place, naming them, connecting them, repositioning them, etc. However, there are examples of games that have only been created using visual programming. Two of the most notable examples are Out of Action and Bright Memory Infinite. Some examples of game engines that offer visual programming options are Unreal Engine, Unity, Godot, Construct 3, GameMaker studio, and Scratch. [13]

2.4.1 Unreal Engine

The Unreal Engine provides support for visual programming through its Blueprint system. The Unreal Engine's Blueprint system is probably the most flushed out system compared to the other visual programming options from other game engines. It uses a node-based interface using a visual graph editor to drag-and-drop the nodes and connect them to create gameplay elements.

2.4.2 Unity

The Unity game engine supports visual programming through its system called Visual Scripting. It was available as a separate package in previous versions of Unity called Bolt, but it was bought by Unity, and it's now integrated into the game engine and is available by default. It also uses a node-based system same as the Blueprint system in the Unreal Engine.

2.4.3 Godot

Godot offers a system called VisualScript for visual programming. However, it is famous for not being able to compete with other visual programming options of other game engines. It was introduced in Godot version 3.0 in 2018, but it was not able to catch enough audience and has been discontinued in the latest Godot 4.0 version. [14]

2.4.4 Construct 3

Construct 3 is a very easy-to-use 2D only game engine, and it only supports visual programming, meaning it does not have a traditional programming option. Unlike Godot, which its visual programming option was not integrated into the game engine very well, Construct 3 was built from the start with visual programming in mind. This makes creating games with Construct 3 very intuitive and a great platform for learning game development and creating 2D games very quickly.

2.4.5 GameMaker Studio 2

GameMaker studio 2, although it does have a traditional programming option, its main focus is on its integrated visual programming option. It offers a drag-and-drop graphical interface that is very user friendly and easy to use and understand. Same as Construct 3, it is a good game engine to learn game development and make 2D games.

2.4.6 Scratch

Scratch is a 2D only game engine and was developed mainly as a learning tool and it only supports visual programming. Scratch has probably the most intuitive and easy to use system for making simple 2D games. Scratch is a good example of what visual programming represents. Easy to understand and make games with, but difficult to make big projects with only visual programming. [15]

2.4.7 Visual Programming in Game Engines Summary

Visual programming offers a broad and accessible entry into game development. Its intuitive approach simplifies game creation, enhancing collaboration and opening the field to a wider audience. Despite certain challenges, such as slower development speed, its benefits are evident. As engines continually improve their visual scripting systems, it remains a valuable tool in the evolving landscape of game development.

3 POPULAR GAME ENGINE

Currently, there are many publicly available, free to download game engines in the market for game developers to choose from. There are engines like Unreal Engine 5 that focuses on 3D and tries to cover as many types of games as possible, GameMaker Studio 2 that mostly focuses on 2D games, and Unity that tries to do everything. This chapter will focus on going over some of these engines in more detail. Going over their history, games they produced, programming languages they use, their financials model, etc. [16]

3.1 Unity

Unity is one of the most popular game engines used right now. It is mostly used by indie game developers. However, there are examples of some big game development studios being able to gain financial success using Unity. It is beginner friendly and recommended by many people working in the game development industry.

3.1.1 History.

Unity game engine was launched by a company currently based in San Francisco named Unity Technologies in 2005 for the Mac operating system, but it has since been adopted to support Windows and other platforms. Unity Technologies was initially based in Denmark, but it has since relocated its headquarters to San Francisco, USA. [17]

3.1.2 Games made with Unity.

Examples of some popular games made with Unity are Subnautica, Beat Saber, Escape from Tarkov, Genshin Impact, and Fall Guys.

3.1.3 Programming languages and visual programming.

The engine itself is written in C++, but it uses C# as its main programming language and has limited support for C++, although it also supports visual programming using its system called Visual Script.

3.1.4 Unity's Financial Model.

The Unity game engine is free to download and use, and it has paid subscription tiers for those who want extra features and for large teams. However, the free tier is only available if the developer has not made more than \$100k in revenue during 12 months with their games made with Unity. [21]

3.1.5 Supported game types.

Unity has support for making games in both 2D and 3D. It also supports creating many types of games like action-adventure, FPS shooters, racing, etc.

3.1.6 Supported platforms.

Unity has a great list of supported platforms. It supports all major PC operating systems (Windows, Linux, and Mac), has great support for mobile devices, supports deploying games on all modern game consoles, and is one of the better game engines for creating games for Artificial and Virtual Reality with supporting most of the major AR and VR devices. [19]

3.1.7 Unity's Community, Learning, and Documentation.

Unity has arguably the largest community of game developers, which is a big help for beginner developers who want to ask questions about problems they run into. Also, the documentation provided by the engine developers is much better than that of other game engines. Given the large community, there are also many free online resources that make it easy for someone to start learning making games with Unity. [17]

3.1.8 Unity's Asset Store.

Unity also has an asset store where designers and artists can sell their creations ranging from 3D assets to specific sounds while keeping 70% of the profit, and game developers can find the assets they need for creating their games. [17]

3.1.9 Job prospects.

In terms of job opportunities, since Unity is very popular amongst developers and many game development studios use it, it has good job prospects. Learning Unity also has the additional benefit of learning C#, which is very popular in the fields of web and mobile development with .NET giving it another boost regarding finding jobs in the future. [22]

3.1.10 Unity's Drawbacks.

Some of the drawbacks of the Unity game engine are its performance and memory issues for when working on large projects for mobile, some issues with not being able to link external libraires, and its terrain editor being mediocre. [20]

3.1.11 Latest updates.

The current version of Unity is Unity 22.2.9. The next version, Unity 23.1 is promised to come out during 2023 bringing some anticipated new features and improvements like support for ray tracing, improvements to the UI toolkit, improvement to performance and networking, and many others. [23]

3.1.12 Unity Summary.

Unity is a great engine to learn right now. It is used by many developers, supports many platforms, has a great and vibrant community, and an accessible pricing model. It has a few drawbacks, but it's continuously being improved and promises many new improvements in its latest upcoming update.

3.2 Unreal Engine

Unreal Engine is currently in its fifth iteration and is one of the most influential players in the game development industry. It has been responsible for many AAA games and many more upcoming games are using Unreal Engine 5. However, it has a steep learning curve and needs powerful hardware in order to have a decent development experience.

3.2.1 History.

Unreal Engine was developed and launched in 1998 by Epic Games which is located in North Carolina, US. Epic Games's founder, Tim Sweeney, is known to be one of the most influential individuals in the game development industry. Unreal Engine has expanded its capability to support various genres of games beyond its initial focus on FPS titles. [28]

3.2.2 Games made with Unreal Engine.

Some examples of popular games made with Unreal Engine are Fortnite, Gears of War franchise, Unreal Tournament, Borderlands 3, and Batman Arkham Knight. [24]

3.2.3 Programming languages and visual programming.

The Unreal Engine is written in C++ and also uses it as its main programming language to write code. However, it also has great support for visual programming through the Blueprint system, which is currently one of the best examples of visual programming in the game development world.

3.2.4 Unreal Engine's Financial Model.

The Unreal Engine offers three subscription tiers. A free tier for personal use, a paid tier for non-game development studios that use the engine for film making and real-time simulations, and another paid tier for game development studios that use it for making games. The free tier applies only if the developer has made less than \$1 million in revenue during the last 12 months, otherwise, royalty fees are applied. [27]

3.2.5 Supported game types.

The newest version of Unreal Engine officially only supports 3D, but it is also possible to make 2D games, and 2D hybrid games, although with some workarounds. With the many features and functionalities in the Unreal Engine, it gives the developer the option to create many genres of games.

3.2.6 Supported platforms.

Same as Unity, the latest version of Unreal Engine has support for all of the desktop operating systems, deploying to all modern gaming consoles, mobile systems, and popular VR and AR systems.

3.2.7 Unreal Engine's Community, Learning, and Documentation.

The Unreal Engine has a great community of developers that have explored many of the problems that would face beginner developers and are very supportive. However, the official documentation provided by the developers leaves much to be desired, and the engine's complexity makes it difficult for independent creators to make extensive educational content. Although with the latest increase in popularity, the developers are starting to care more about the documentation, and more creators are making free online educational content. [25]

3.2.8 Unreal Engine's Market Place.

The Unreal Engine also has a notable aspect being its Market Place, which includes many free assets provided by Epic Games, and the creators who sell their creations in it, retain 88% of the sales. This provides a great example for all online marketplaces whether related to game development or not, that usually retain 30% of the sales and only pay the creators 70%. [25]

3.2.9 Job prospects.

Unreal Engine, being used by so many big studios and in many enterprise fields, provides great job prospects for people who know how to use the engine. Also, knowing C++, which is a difficult language to learn, but still in demand in all different industries, makes the job prospects of Unreal Engine game developers even brighter. [22]

3.2.10 Unreal Engine's Drawbacks.

The Unreal Engine also has some notable disadvantages. Those disadvantages being the complexity of the engine and the UI, and its use of C++, which is a difficult language to learn, the high system requirement for running the engine and running games that are made with it, and finally, even though the engine is practically free for small creators, the licensing and royalty fees for big studios is very expensive. [25]

3.2.11 Latest updates.

Epic games just recently released Unreal Engine 5.2. which comes with some exciting new features like a new material framework called Substrate, procedural environment generation, a new unified marketplace, a new way to record facial animations using only mobile phones, and many more small improvements. [46]

3.2.12 Unreal Engine Summary.

Unreal Engine stands as an exemplar in the realm of game development. It has some great features, accessible financial models, and a great community. Even though the engine has some drawbacks, the advancements it's making in game development make it a great engine to learn and keep watching for the future.

3.3 CryEngine

CryEngine is one of the most powerful game engines currently available in the game development industry. However, it hasn't received any major updates for too long and is starting to get overshadowed by other game engines. The engine still has some powerful tools, and the potential for a major update coming in the future, makes the engine worthwhile to discuss in more detail.

3.3.1 History.

CryEngine was released in 2002 by Crytek, which is a game and software development company based in Frankfurt, Germany. The game engine was first showcased in 2004 with the release of the game Far Cry that showed impressive features for the time.

3.3.2 Games made with CryEngine.

Some popular examples of games made with CryEngine are the Crysis Franchise, Prey, Ryse: Son Of Rome, and Hunt: Showdown. [29]

3.3.3 Programming languages and visual programming.

The programming languages the engine supports are C++, C#, and Lua, while also having support for visual programming with its system called Flow Graph. However, to take full advantage of the power and capabilities of the engine, it is necessary to use C++.

3.3.4 CryEngine's Financial Model.

The licensing terms and conditions of CryEngine are simple, where the engine is free to download and use, but if annual revenue for a project exceeds \$5k, the developers will have to pay a 5% royalty fee. [31]

3.3.5 Supported game types.

CryEngine has support for both 2D and 3D games, and the game engine is versatile enough to support many genres of games.

3.3.6 Supported platforms.

Developers can run the engine on Windows and Linux systems. The engine also supports deploying to all the modern Consoles, Oculus Rift, OSVR, PSVR, and HTC Vive. Unfortunately, CryEngine does not yet support making games for mobile devices. [33]

3.3.7 CryEngine's Community, Learning, and Documentation.

The community of CryEngine is smaller compared to the communities of the previously mentioned game engines, and it is more difficult to find free comprehensive online tutorials for beginners who are trying to learn the engine with no previous experience. Although Crytek have tried to provide tutorials made by themselves, there still are not enough tutorials, and the documentation although well made, there is still much to be added. [32]

3.3.8 Job prospects.

Because of the decline of popularity of CryEngine, finding jobs is starting to become more and more difficult for CryEngine developers. However, the skills learned from making games in CryEngine can easily be transformed to other game engines, and knowing C++ provides its own advantages as mentioned previously in Unreal Engine section. [22]

3.3.9 CryEngine's Drawbacks.

Even though CryEngine has one of the best rendering engines in the market right now, the game engine still has some big disadvantages. Some of these disadvantages are it being resource intensive for creating large games, not enough learning materials, its heavy reliance on C++ which is a difficult language to learn, and not having mobile support.

3.3.10 Latest updates.

The latest version update of CryEngine, CryEngine 5.7 LTS, only added some small updates and features, even though it has been the only update in more than 3 years. CryEngine is quickly falling behind its competition and is in dire need of a big version update if they want to remain competitive in the current market, but there is still no news on when the next major version update will be. [30]

3.3.11 CryEngine Summary.

CryEngine has been one of the most influential game engines in the game development industry. It has support for a good range of programming options, an accessible financial model, and support for many types of games. However, a shrinking community, limited learning materials and job opportunities, and lack of mobile device support necessitate a major update soon.

3.4 Godot

Godot is a free and open-source game engine that is starting to become more and more popular with game developers. With each new update, the engine is introducing new features and it's starting to compete with other big-name game engines in the market. However, it is still mostly used by indie game developers and is yet to produce a highly profitable game. Nonetheless, it is still worth talking about it in more detail.

3.4.1 History.

The Godot game engine was developed and released by two Argentinian game developers Juan Linietsky and Ariel Manzur. They started development on the engine in 2001 and was mainly used as an in-house game development engine for their studio name OKAM that is based in Buenos Aires, Argentina. Godot was released to the public in 2014 as a free and open-source game engine. [35]

3.4.2 Games made with Godot.

Some examples of popular games made with Godot are Kingdoms Of The Dump, Romino's Adventure, Gravity Ace, and Resolutiion. [34]

3.4.3 Programming languages and visual programming.

For programming, Godot uses its own proprietary language called GDScript, while also having support for C#, and C/C++ thorough its system called GDNative. Godot used to have support for visual programming, but it was discontinued in its latest version Godot 4.0.

3.4.4 Godot's Financial Model.

Godot uses the MIT license which mean that anyone is free to basically do anything with the source code of the engine under the condition that the resulting software from the original source-code must also be under the MIT license and must mention the name of the author. Game developers using Godot can freely make and publish their games made with Godot, while not having to pay any royalties or any other fees, keeping all of the profit for themselves. [36]

3.4.5 Supported game types.

Godot is great for creating 2D games while also having support for 3D. The newest version, Godot 4.0, overhauled the entire 3D engine and made many improvements, but it is still very behind compared to other popular and 3D capable game engines.

3.4.6 Supported platforms.

Godot developers can port their games to Windows, Linux, mac, iOS, Android, and web, but is not possible to port their games to gaming consoles, unless the developers publish their game with the help of a game publisher.

3.4.7 Godot's Community, Learning, and Documentation.

The Godot game engine has a vibrant community of developers, and the engine being free and continuously receiving updates, helps a lot with growing the community even further. As a result, there are many independent creators that make helpful content and online tutorials. However, Godot still being a new engine compared to other game engines, the documentation provided by the developers is lacking compared to older game engines. [36]

3.4.8 Godot's Asset Library.

Godot offers an asset library where developers can download and share assets they create, but since it has a requirement that all content must be free, not many developers are creating content for it, making it very much lacking compared to similar marketplaces of other game engines. [36]

3.4.9 Job prospects.

In terms of job availability, since Godot is still being considered as a new engine, not many studios use it for their games, and its programming language GDScript being proprietary, does not help with job prospects outside of Godot. However, with the game engine continuously getting updated and its community growing steadily, the job prospects of Godot developers may improve in the future. [22]

3.4.10 Godot's Drawbacks.

Godot, same as all the other game engines, has its disadvantages. Its biggest disadvantages are as mentioned before, its weak 3D rendering engine, its proprietary language that can't be used outside of Godot, and its lack of visual programming options. [36]

3.4.11 Latest updates.

The current version of the engine is Godot 4.0 which brought a lot of improvements to the engine when it was released, notably its 3D rendering engine. The next version, Godot 4.1, is expected to be released before the end of 2023, and bring with it many new features. [37]

3.4.12 Godot Summary

In summary, Godot is an open-source and free-to-use game engine that has become increasingly popular, particularly in indie game development and 2D game creation. Despite its limitations in 3D capabilities and proprietary GDScript language, its flexibility, active

community, and commitment to free use make it an attractive choice for budding developers. With its ongoing development and improvements, Godot shows great promise for the future of game design.

II. PRACTICAL

4 SHOWCASING CHOSEN GAME ENGINES

Three game engines have been chosen for comparison. The focus was to choose amongst game engines that are available to download for free and that have capable 3d rendering engines. A secondary focus was also choosing game engines that use C++ as their main programming language since these engines are known to be more performant. The reason only three game engines were chosen is because it takes too much time for one person to learn three game engines from scratch, create tests for them, analyze their capabilities, and give proper conclusion to the comparison.

For showcasing the game engines, one scene has been chosen and imported to the engines. Each engine is running on the highest graphics settings when doing this showcase. Since each engine handles assets and materials differently, it was difficult to make the scenes look exactly the same, but an engine not supporting a material is inherently a drawback. The main purpose of this showcasing is to give a proper feel of the quality of the rendering engine in these game engines.



Figure 1 Image of the scene rendered in Blender.

Figure 1 shows the scene being rendered in Blender. The scene has all the original assets and materials without any modifications having been done to them.

4.1 Unity

The main reason Unity was chosen is because it is arguably the most popular game engine with game developers. Unity uses URP (Universal Rendering Pipeline) and HDRP (High Definition Rendering Pipeline). URP is better for performance while HDRP is used more for realistic scenes. Both rendering pipelines are very powerful, and it depends on the developer to choose which suits their game the best. Even though it is possible with some workarounds to use C++ with Unity, C# is by far the best way to write scripts in Unity. [38]

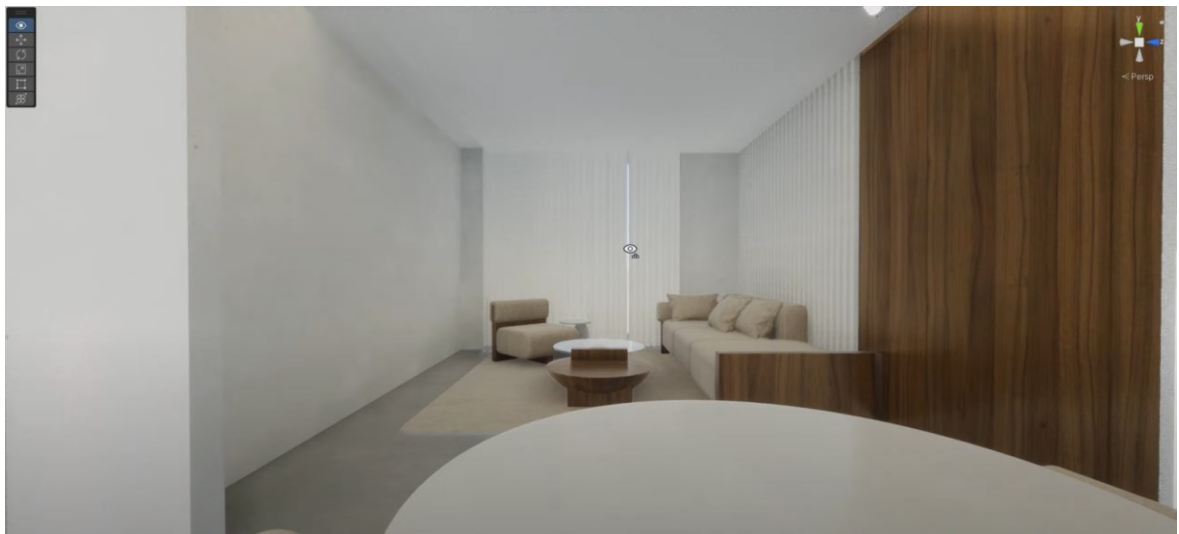


Figure 2 Image of the scene rendered in Unity.

The render in Figure 2 shows the scene being rendered in Unity. It must be highlighted that Unity does not translate all of the materials very well, so the materials have to be changed to some degree in order for everything to show up correctly in the editor.

4.2 Unreal Engine 5

Unreal Engine 5 was chosen because it arguably has one of the most powerful rendering engines right now. Unreal Engine currently uses two major features to achieve this. One is called Nanite, which replaces LOD (Level of Detail), with dynamic LOD, that adjusts the number of polygons in an asset dynamically based on the distance. The other one is called Lumen, And Lumen, which provides dynamic real-time in-engine global illumination and reflection, helps independent developers and small studios to focus more on other important parts of the game other than lighting. UE5 also has much better support for materials and meshes compared to other engines specifically because of its latest technology called Substrate, that enables an asset to have multiple layers of materials. On top of its impressive features, UE5 also uses C++, which is the most used by game engines for its performance and the language being object-oriented. [26][39]



Figure 3 Image of the scene rendered in Unreal Engine 5.

Figure 3 shows the scene being rendered in Unreal Engine 5 with the Cinematic graphic quality. It must be said that considering the scene is being rendered in real-time, the scene looks very good.

5 TESTS FOR BENCHMARKING GAME ENGINES

One of the best ways to make a comparison between these engines is to do some performance testing. Testing the engines this way will provide a clear numerical number indicating how well they perform against each other. The comparisons that will be done are CPU load testing, GPU load testing, and Memory usages testing. These tests will be able to compare some of the most common performance areas in game engines.

5.1 CPU load testing

This chapter will show how well these engines handle CPU/Calculation heavy scenarios. Tests should focus on stressing the physics engine of these game engines that perform most of those calculations using the power of the CPU. Almost every game does these types of calculations using the CPU, and the engine performing well in these tasks is necessary.

Methodology. In each of the game engines, a test has been created where cubes are spawned in random locations within an area, the cubes then gradually move towards the center and collide with each other. The test will be measuring average PFS (Frames Per Second) when the number of cubes reaches 2000, 4000, 6000, and 8000. The software used to do the measurement is called MSI afterburner which provides great tools to do these measurements and is widely used for benchmarking performances of games.

Testing Environment. All of the tests were run in the same room temperature. The only programs running were the game and the software used to measure the average FPS. Since some of the hardware used for the were limited in terms of power, in some cases, lower graphics were used to enable the CPU to not be limited by the power that the GPU is drawing.

Testing Hardware. The tests were done Using an ASUS ROG STRIX G17 laptop with Ryzen 7 5800h for the CPU, a RTX 3060 for the GPU, 32Gb of DDR4 RAM, and 2.5 TB of M.2 NVMe SSD. The resolution of the screen is 1980x1080 and the refresh rate is 144Hz.

Steps to do the tests.

1. The game is built and run.
2. After the desired number of cubes is reached, the test is left to run for 10 minutes.
3. After the 10 minutes are finished, the measuring software is used to start measuring performance for 15 minutes.
4. After the measuring, let the computer cooldown before doing the next test.

5.1.1 Unity

The first test didn't have any problems running in Unity. Figure 4 shows the results of the tests with how many FPS the engine was able to reach with the number of cubes for each test.

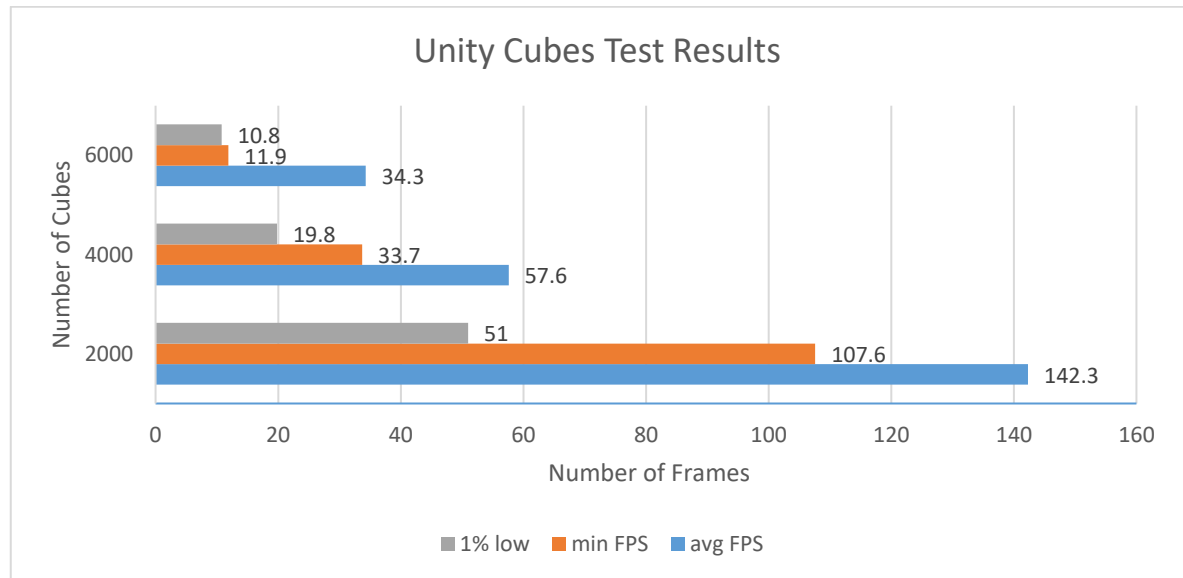


Figure 4 Results of the cube tests in Unity.

The results show that in the test with 2000 cubes, Unity was just under the maximum refresh rate of the monitor. Going from 2000 cubes to 4000 cubes, the performance decrease is ~147% for the average FPS hovering around 60FPS. The performance decrease was much less when going from 4000 cubes to 6000 cubes at ~67%. At 6000 cubes, the average frame rate was getting close to 30FPS which is the minimum number for average FPS for a game to be considered playable.

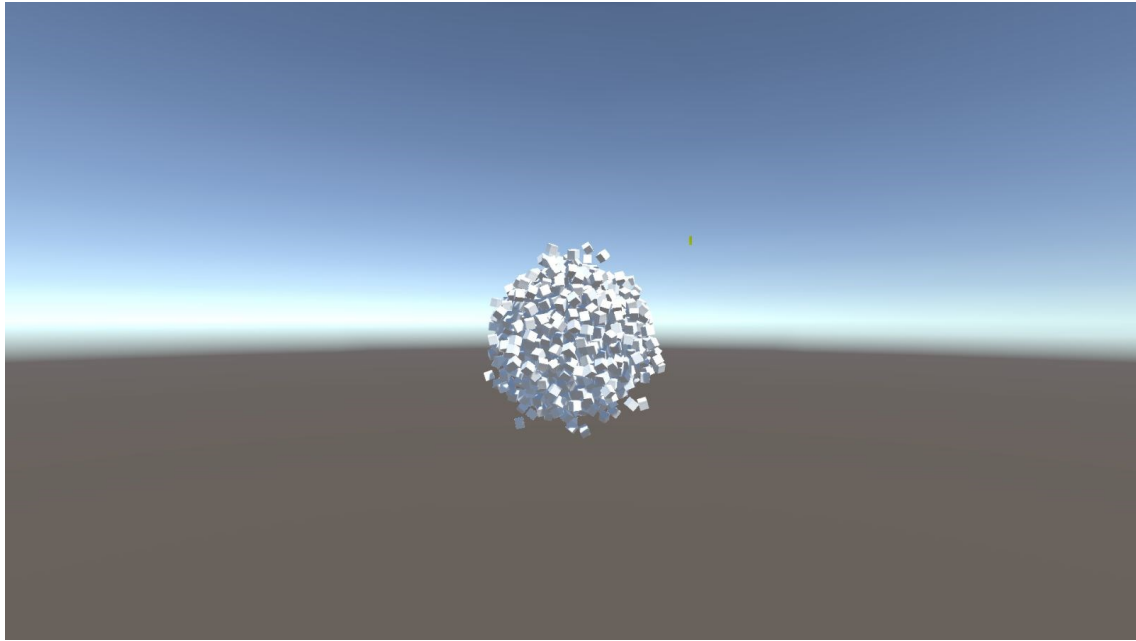


Figure 5 Image of the cubes test running.

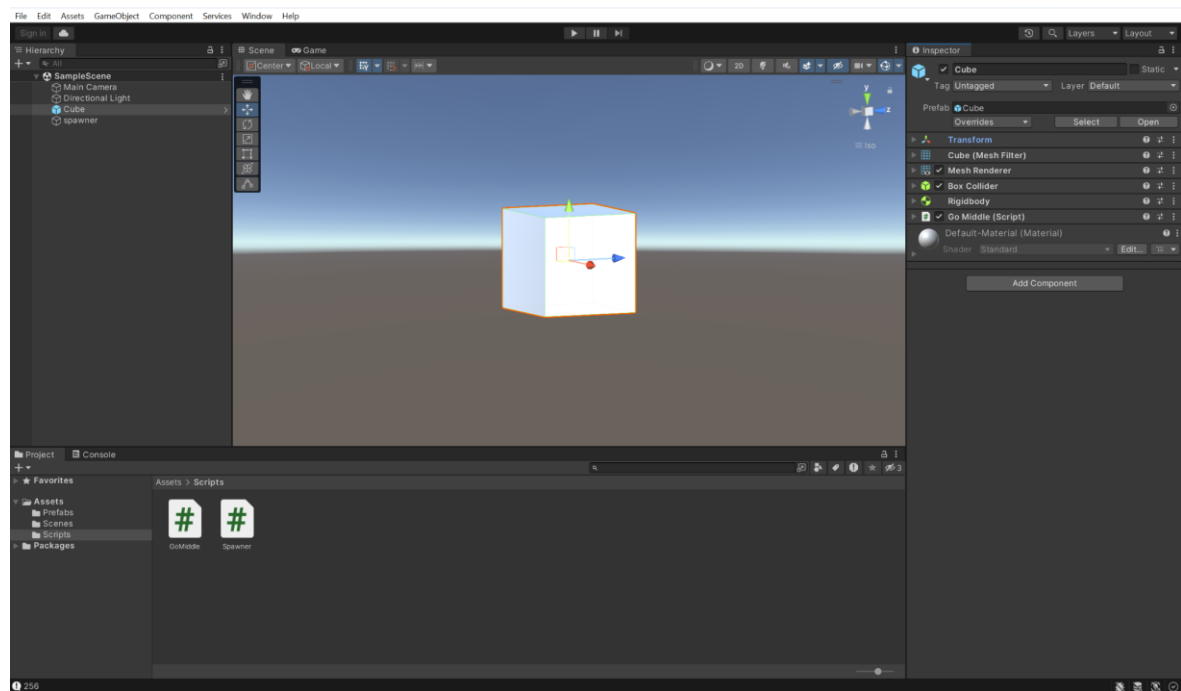


Figure 6 Image of the cubes test open in the Unity editor.

Figure 5 shows the test running with 2000 cubes in Unity. The game world does not contain any extra assets. The only objects in the world are the cubes, camera, and lighting.

Figure 6 shows the test open in the editor not running. The editor shows what components are applied to the cube, the scripts files, and the objects in the game world.

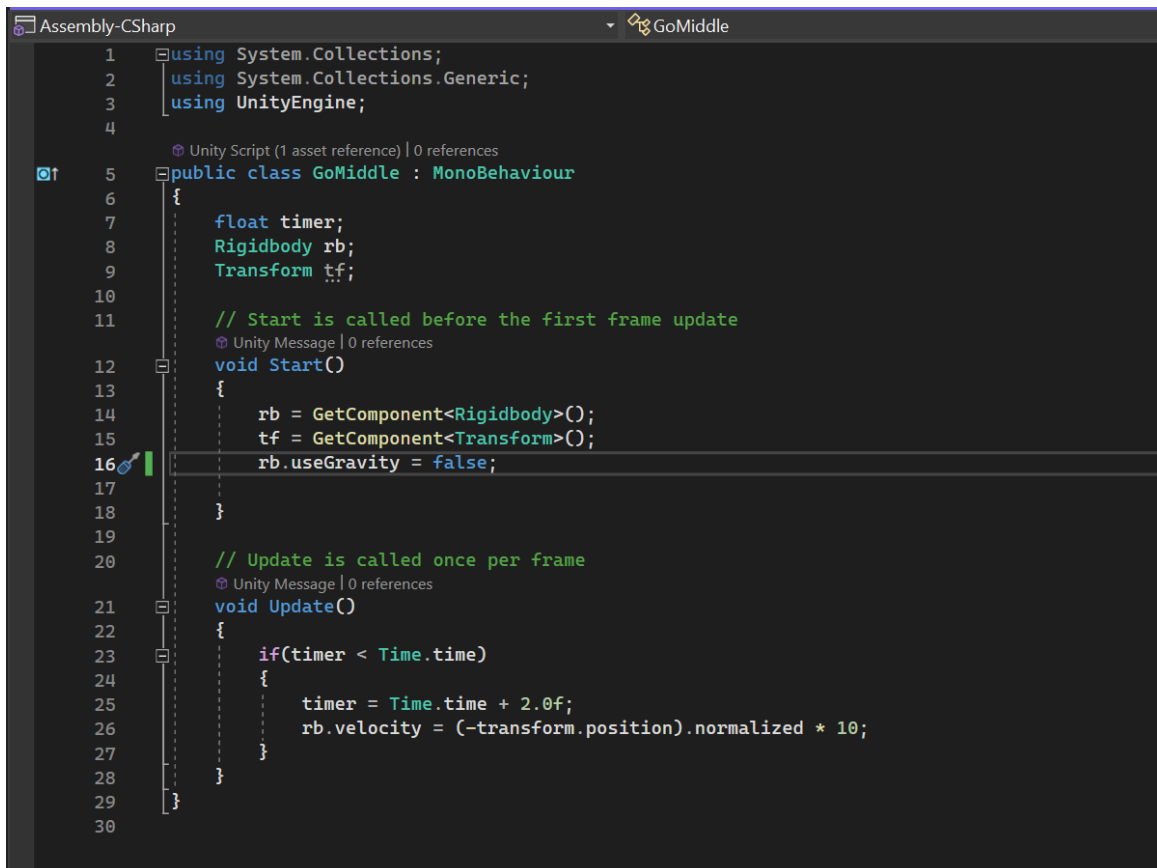


Figure 7 Image of the code responsible for moving cubes to the middle.

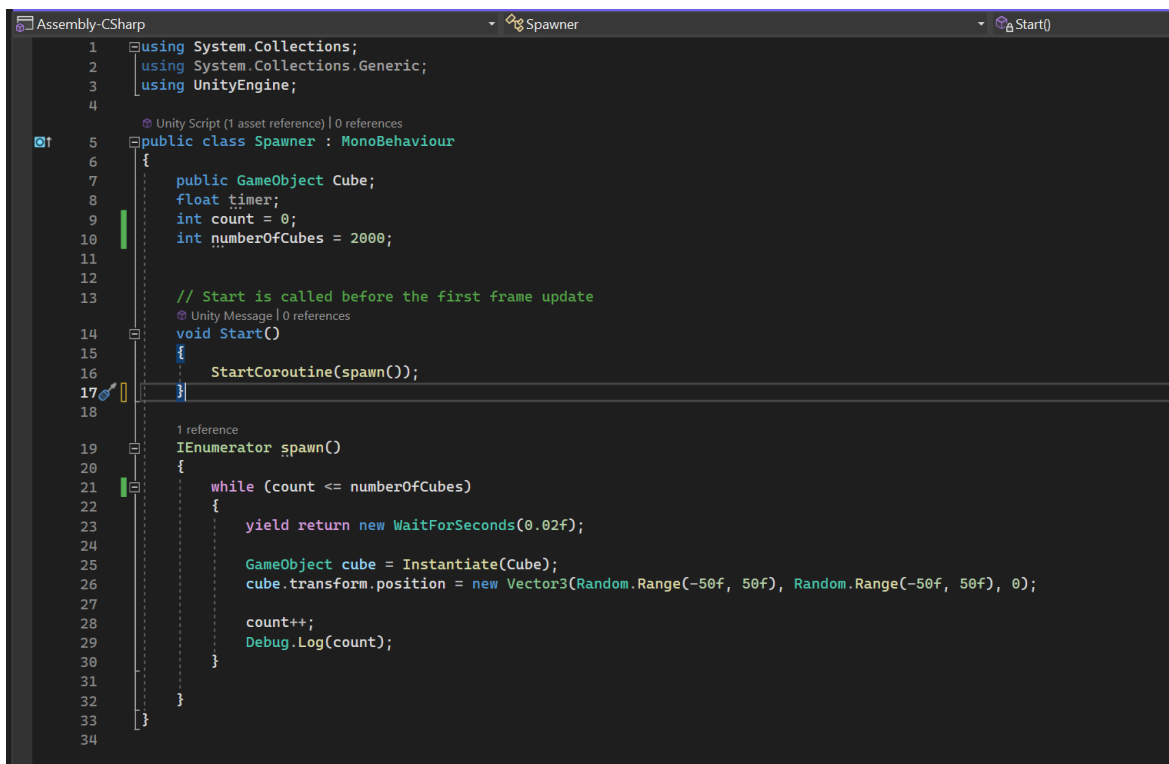


Figure 8 Image of the code responsible for spawning the cubes.

The code shown in Figure 7 is responsible for moving the cubes towards the middle. When the cube is created, its components are initialized, and the gravity disabled. Then every two seconds, the cube will move towards the center.

The code shown in Figure 8 is responsible for spawning the cubes. A function has been created that every 0.02 seconds a cube is spawned at a random location on the Y and X axis. The count is incremented, and the number of cubes is logged. The function is put inside a Coroutine and starts running when the game is started.

5.1.2 Unreal Engine 5

Same as in Unity, the test didn't have any problem running in Unreal Engine 5. The tests for the number of cubes were able to run successfully and the results were recorded as shown in Figure 9.

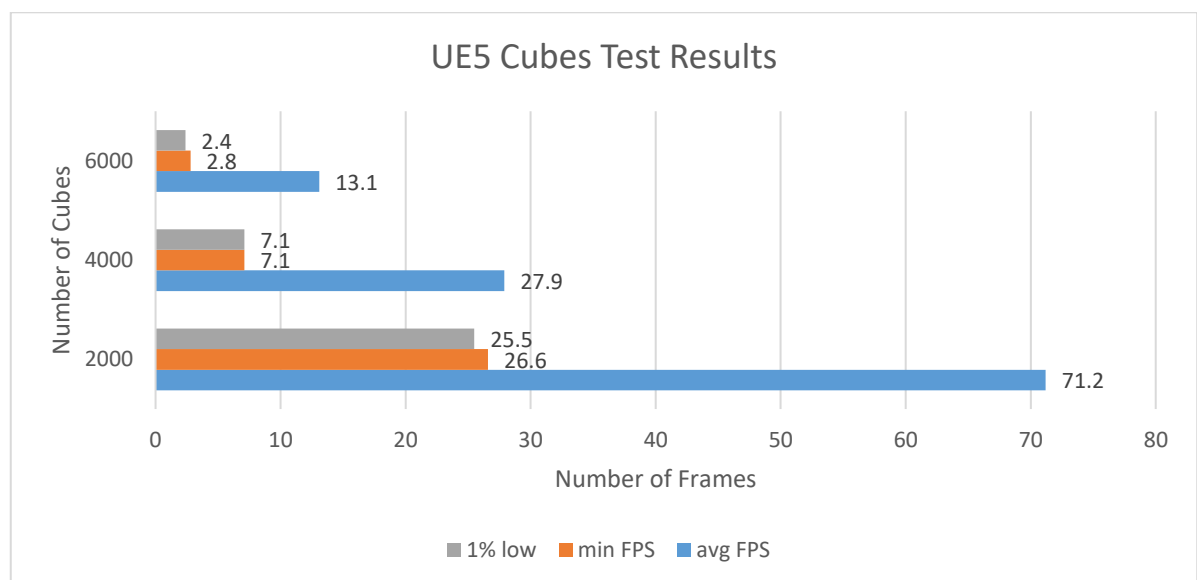


Figure 9 Result of the cube tests in Unreal Engine 5.

The results of the test show that in terms of this specific test and for CPU tests in general, Unreal Engine 5 has lower performance numbers. The chart shows that at 2000 cubes, Unreal Engine was just above 60FPS at 71.2FPS. The decrease in performance going from 2000 cubes to 4000 cubes is ~155%, and the performance decrease going from 4000 cubes to 6000 cubes is ~129%. UE5 was already below 30FPS when testing for 4000 cubes which shows big decrease compared to Unity. Furthermore, at 6000 cubes averaging around 13.1FPS is not a playable framerate, especially with the minimum and 1% low framerates at ~2.8 and ~2.4.

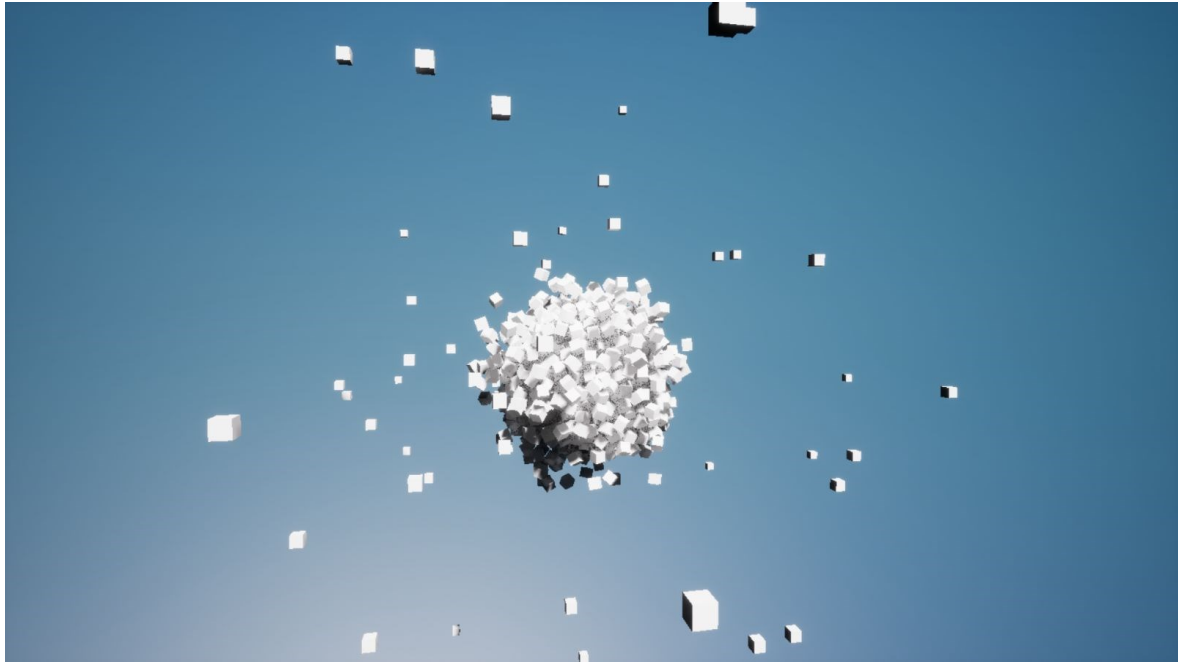


Figure 10 Image of the cubes test running in Unreal Engine 5.

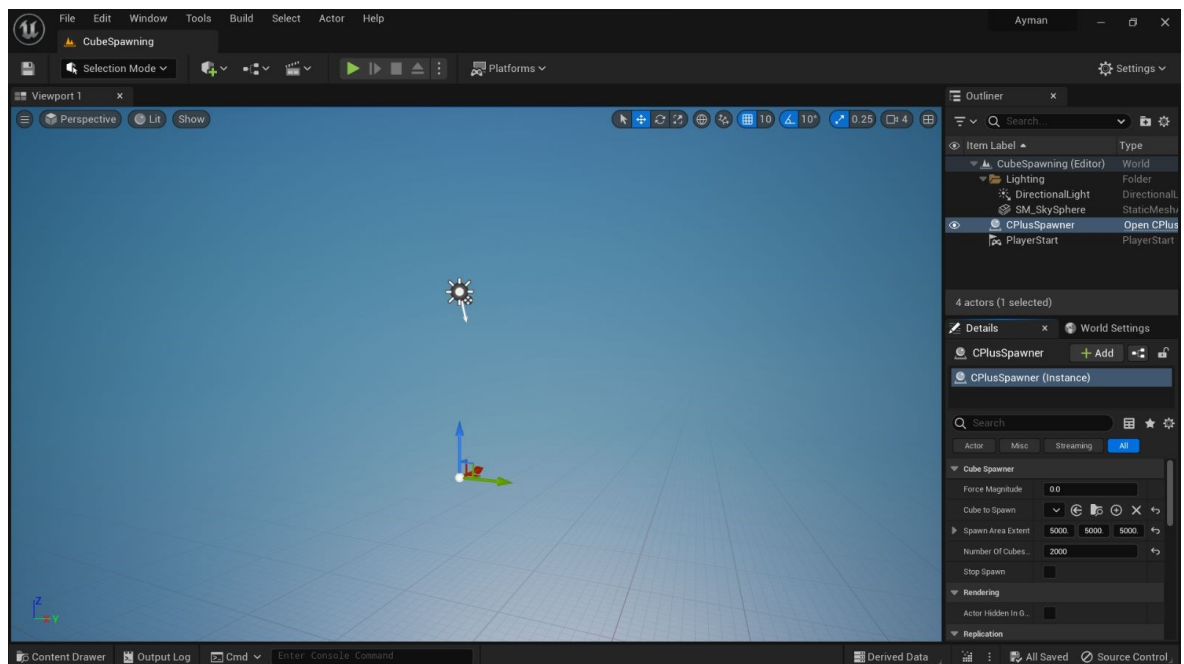


Figure 11 Image of the cubes test open in the Unreal Engine 5 editor.

Figure 10 shows the test running in UE5 for 4000 cubes while the desired number of cubes has not been reached yet. The game world contains only the necessary objects to illuminate the game world, the camera, and the cubes.

Figure 11 shows the test open in the UE5 editor while not running. It is possible to see the objects that are currently in the game world, and some options that can be modified for spawning the cubes.

```

12
13 // Called when the game starts or when spawned
14 void ACPlusSpawner::BeginPlay()
15 {
16     Super::BeginPlay();
17     SpawnCubes();
18 }
19
20 // Called every frame
21 void ACPlusSpawner::Tick(float DeltaTime)
22 {
23     Super::Tick(DeltaTime);
24 }
25
26 void ACPlusSpawner::SpawnCubes()
27 {
28     float SpawnDelay = 0.02f;
29     SpawnCubesWithDelay(SpawnDelay, NumberOfCubesToSpawn);
30 }
31
32 void ACPlusSpawner::setStopSpawn()
33 {
34     stopSpawn = true;
35 }
36
37 void ACPlusSpawner::SpawnCubesWithDelay(float Delay, int32 RemainingCubes)
38 {
39     UWorld* World = GetWorld();
40     FVector RandomDirection = FMath::VRand();
41     FVector SpawnLocation = SpawnAreaCenter + SpawnAreaExtent * RandomDirection;
42     AActor* SpawnedCube = World->SpawnActor<AActor>(CubeToSpawn, SpawnLocation, FRotator::ZeroRotator);
43     if (SpawnedCube)
44     {
45         UStaticMeshComponent* MeshComponent = SpawnedCube->FindComponentByClass<UStaticMeshComponent>();
46         if (MeshComponent)
47         {
48             MeshComponent->SetMobility(EComponentMobility::Movable);
49             MeshComponent->SetSimulatePhysics(true);
50             MeshComponent->SetEnableGravity(false);
51             FVector ForceDirection = SpawnAreaCenter - SpawnLocation;
52             ForceDirection.Normalize();
53             MeshComponent->AddForce(ForceDirection * ForceMagnitude);
54         }
55         SpawnedCubes.Add(SpawnedCube);
56     }
57
58     // Spawn the cubes
59     if (RemainingCubes > 1 && !stopSpawn)
60     {
61         FTimerDelegate TimerDelegate;
62         TimerDelegate.BindLambda( [=]()
63         {
64             SpawnCubesWithDelay(Delay, RemainingCubes - 1);
65         });
66         GetWorld()->GetTimerManager().SetTimer(TimerHandle_SpawnCubes, TimerDelegate, Delay, false);
67     }
68 }
69
70
71
72
73
74
75
76
77

```

Figure 12 Image of the code responsible for spawning the cubes.

```

1  // Fill out your copyright notice in the Description page of Project Settings.
2
3
4  #include "GoMiddle.h"
5
6  void UGoMiddle::GoMiddle(AActor* ActorToMove, FVector TargetLocation, float ForceMagnitude)
7  {
8      UStaticMeshComponent* StaticMeshComponent = ActorToMove->FindComponentByClass<UStaticMeshComponent>();
9
10     FVector Origin = FVector::ZeroVector;
11     FVector ComponentLocation = StaticMeshComponent->GetComponentLocation();
12     FVector Direction = (Origin - ComponentLocation).GetSafeNormal();
13
14     FVector DesiredVelocity = Direction * ForceMagnitude;
15
16     StaticMeshComponent->SetPhysicsLinearVelocity(DesiredVelocity);
17 }

```

Figure 13 Image of the code responsible for moving the cube towards the middle.

Figure 12 shows the code for spawning the cubes. The code repeats the same behavior as in Figure 6, but this time in C++. Some extra steps have to be taken to recreate a similar behavior in C++. When the game is started, a function is called to spawn the cubes, which calls another function to spawn the cubes with the specified delay. The function gets a pointer to the game world, chooses a random direction, and gets a spawn location based on the random direction. The cube then is spawned and the meshes are applied. The function will continue spawning the cubes until the game is stopped or the number of desired cubes is reached.

Figure 13 shows the code responsible for moving the cubes towards the center. The function is periodically called on the cube. The function gets the vectors for the center of the game world and the location of the cube, and then moves the cube towards the direction of the center of the game world.

5.1.3 Results Analysis

The results of the tests show that in terms of CPU performance and the specific tests that were used, Unity is better for games that are heavy on using calculations that would be done on the CPU. Unity on average was around 100% faster than Unreal Engine 5 while being around 160% faster in the test with 6000 cubes.

However, it must be said that CPU usage almost never went over 30% when running the tests, while Unity was consistently hovering ~45% CPU usage. It would be possible to increase the performance numbers of the Unreal Engine tests by using extensive multi-threading, but it was way past the scope of this comparison being for beginner. Moreover, it would also be possible to do the same thing to increase the performance of Unity which would circle back to the same problem.

5.2 GPU load testing

This chapter will show the rendering capabilities of the engines purely in terms of how much frames-per-second they can achieve. The tests will mostly be focusing on placing many assets in the world and rendering them in real-time, which should push the rendering capabilities of the engines. This chapter should show how well the engines will perform when there are many high-polygon assets in the frame at the same time.

Methodology. In each of the game engines, two tests have been created. In the first test, many static assets are put in the game world. All of the assets must be in view when running the test. In the second test, the same thing is repeated as the first test, but this time for assets that are playing an animation. The software used to do the measurement is called MSI afterburner which provides great tools to do these measurements and is widely used for benchmarking performances of games. For all of the engines, LOD has been disabled so there is no performance difference when the camera's distance to the assets varies.

Testing Environment. All of the tests were run in the same room temperature. The only programs running were the game and the software used to measure the average FPS.

Testing Hardware. The tests were done Using an ASUS ROG STRIX G17 laptop with Ryzen 7 5800h for the CPU, a RTX 3060 for the GPU, 32Gb of DDR4 RAM, and 2.5 TB of M.2 NVMe SSD. The resolution of the screen is 1980x1080 and the refresh rate is 144Hz.

Steps to do the tests.

1. The game is built and run.
2. No measuring is done until after 10 minutes to stimulate the GPU.
3. After the 10 minutes are finished, the measuring software is used to start measuring performance for 15 minutes.
4. After the measuring, let the computer cooldown before doing the next test.

Assets Used.

Figure 15 shows the asset that will be used for the first test related to the GPU. The asset has 50,000 polygons and 35,000 varices. The asset was available to download for free at [51].

Figure 14 shows the asset that will be used for the second test related to the GPU. The asset contains 52,100 polygons and 25,600 vertices. The asset is available to download for free at [52].

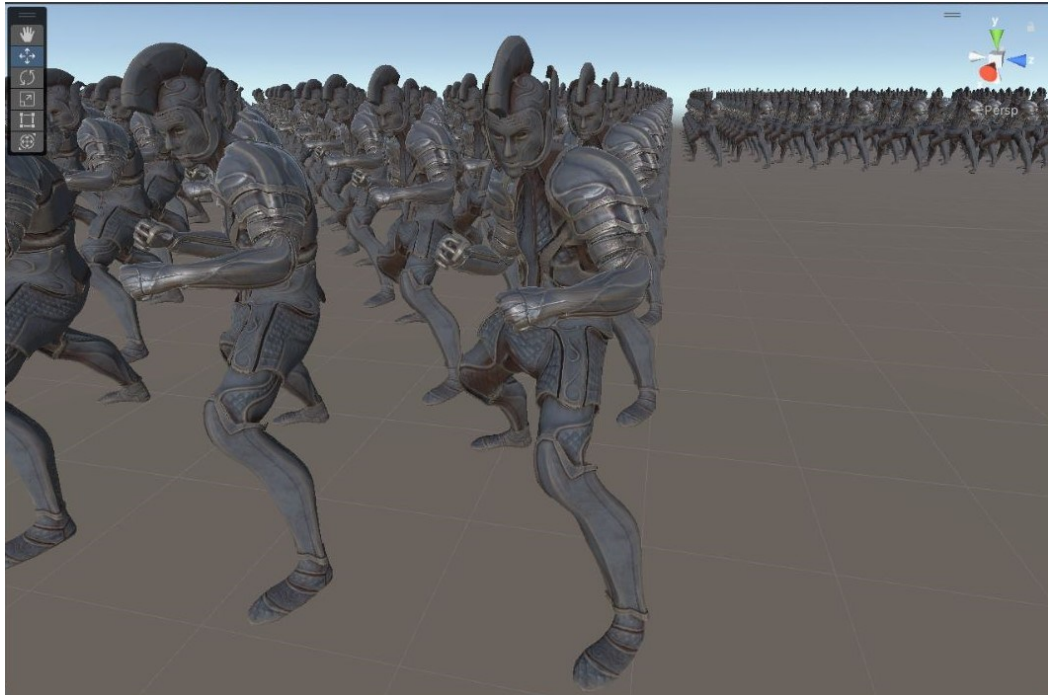


Figure 14 Image of the asset with animation of an ancient fighter.

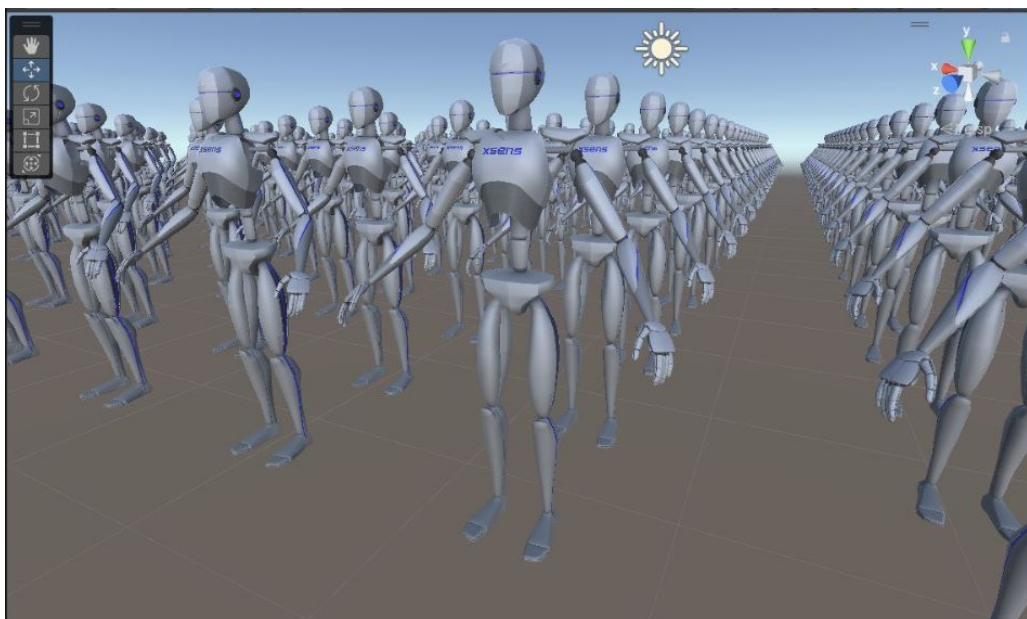


Figure 15 Image of the static asset of a mannequin.

5.2.1 Unity

Figure 16 shows the results of the first test in Unity for the mannequin asset that does not play any animation.

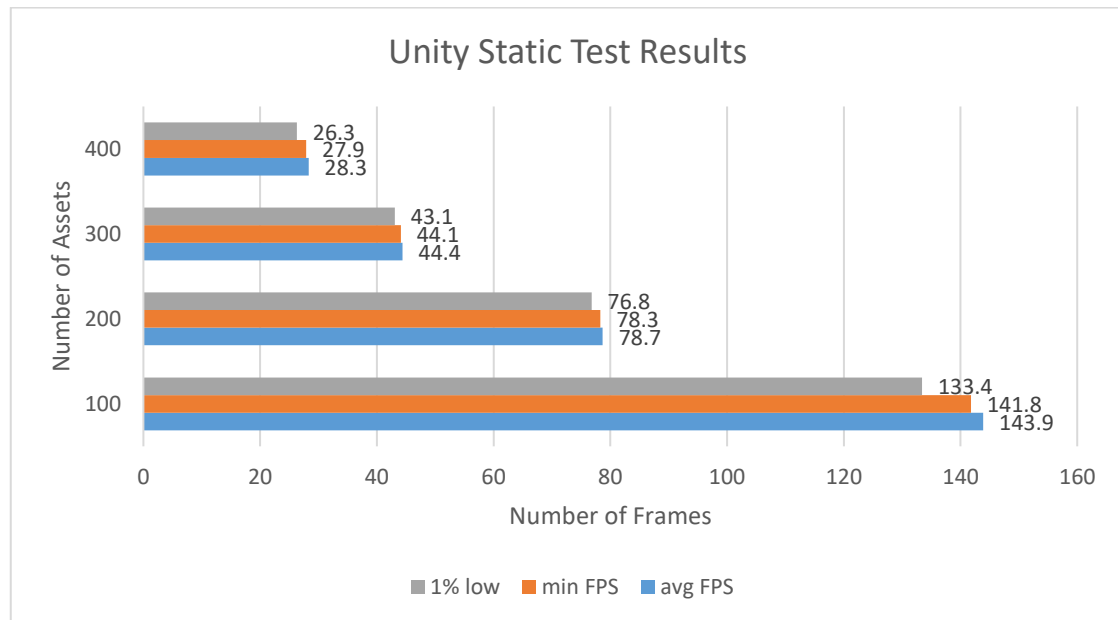


Figure 16 Results of the static asset tests in Unity.

Figure 17 shows the results of the second test in Unity for the asset with animation of the ancient fighter.

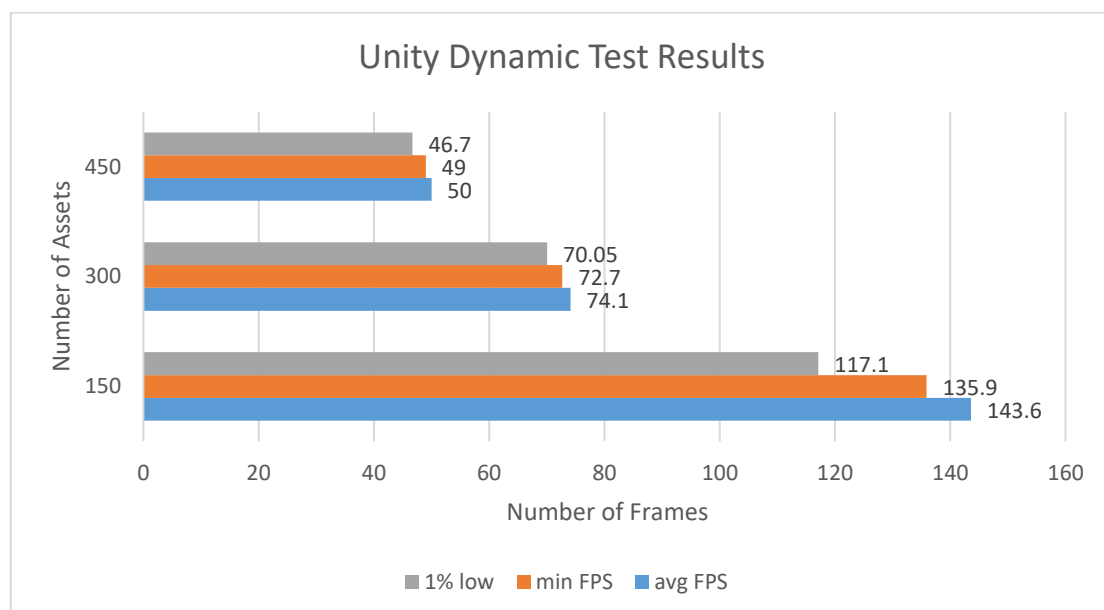


Figure 17 Results of the dynamic asset tests in Unity.

The first test shows that at 100 assets, it was just below the maximum refresh rate of the monitor, and it gradually decreased with the number of assets increasing. It is also possible to see the decrease in performance when increasing the number of assets in Figure 18.

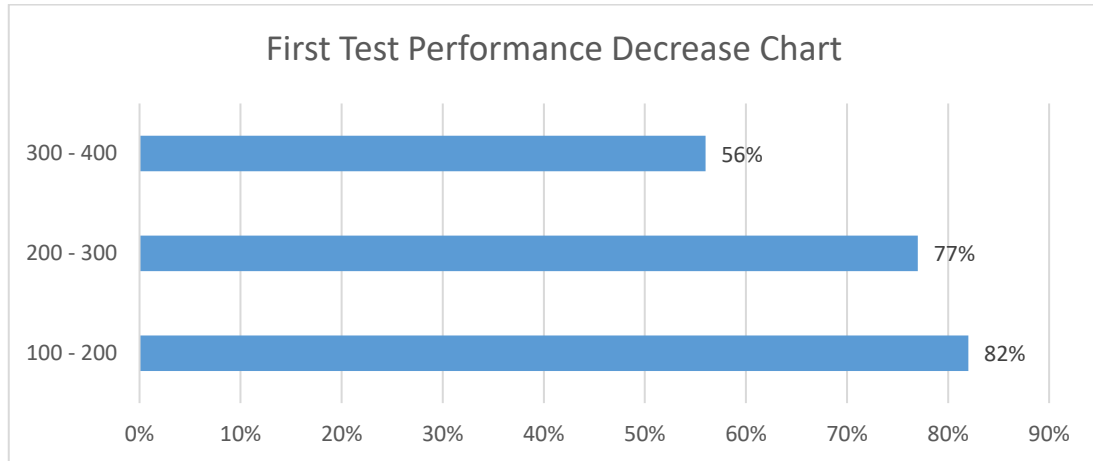


Figure 18 Result of static asset performance decrease chart in Unity.

At 400 assets, the average FPS reaches below 30FPS which going further does not give good enough information regarding the performance.

The second test shows results very similar to the first test. At 150 assets, the framerate is very close to the maximum refresh rate of the monitor. The performance decrease going from 150 assets to 300 is 93% and going from 300 to 450 assets is 48%. Doing the test for more assets would make it go below 30FPS and at that point the test's significance would most likely decrease.

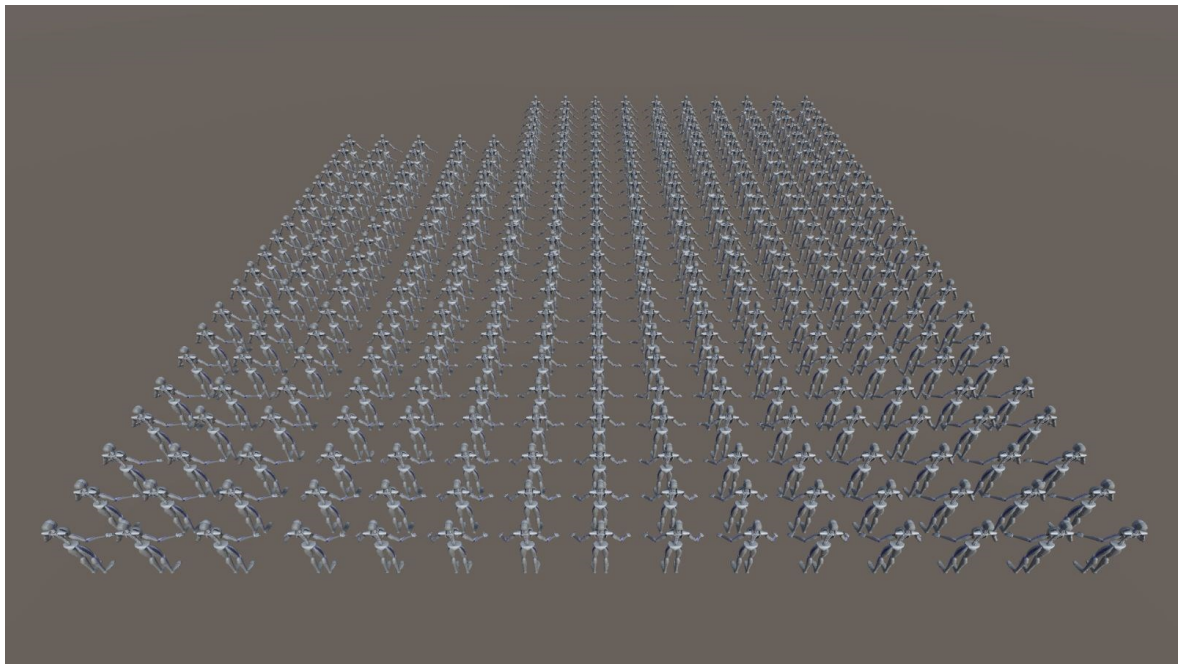


Figure 19 Image of the static asset test running in Unity.

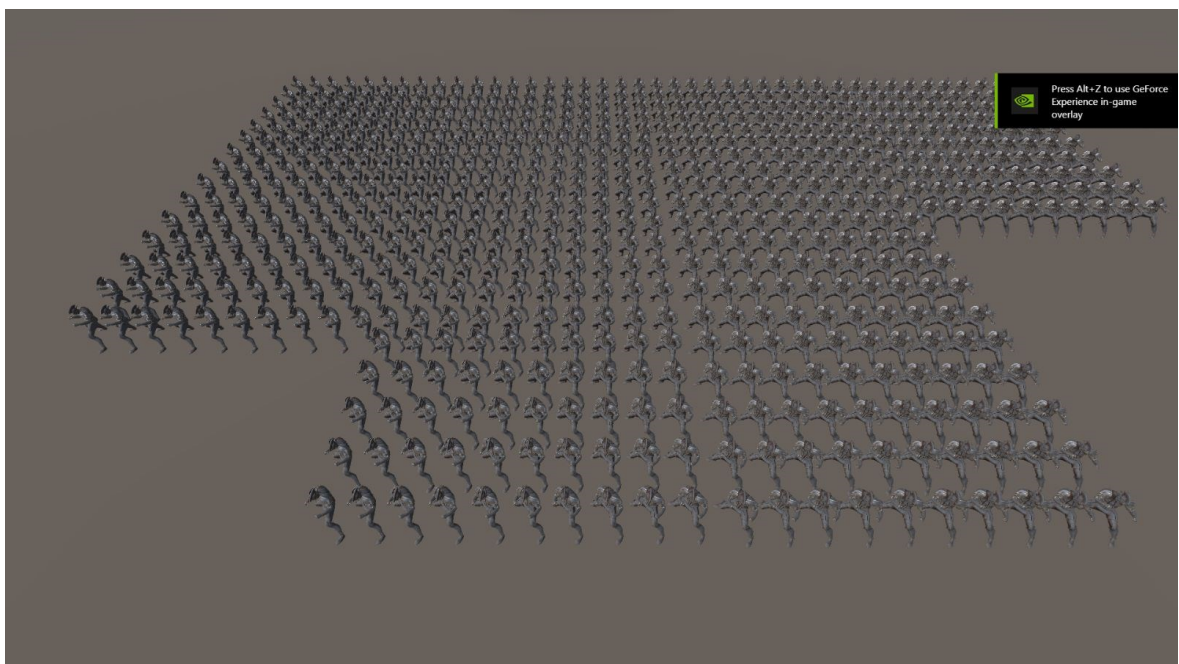


Figure 20 Image of the dynamic asset with animation running in Unity.

Figure 19 and Figure 20 show the assets for both static and dynamic with animation lined up and are all viewable. The number of assets for the static test shown is 350, while the number of assets for the dynamic test is 650.

5.2.2 Unreal engine 5

Figure 21 shows the results of the first test in Unreal Engine 5 for the mannequin asset that does not play any animation.

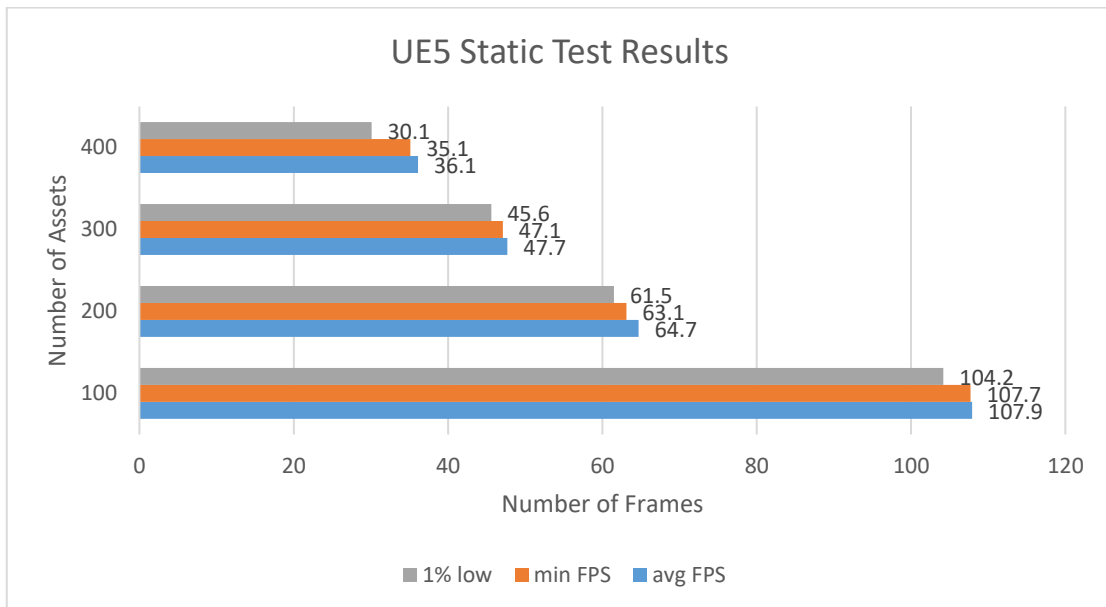


Figure 21 Results of the static asset tests in Unreal Engine 5.

Figure 22 shows the results of the second test in Unreal Engine 5 for the asset with animation of the ancient fighter.

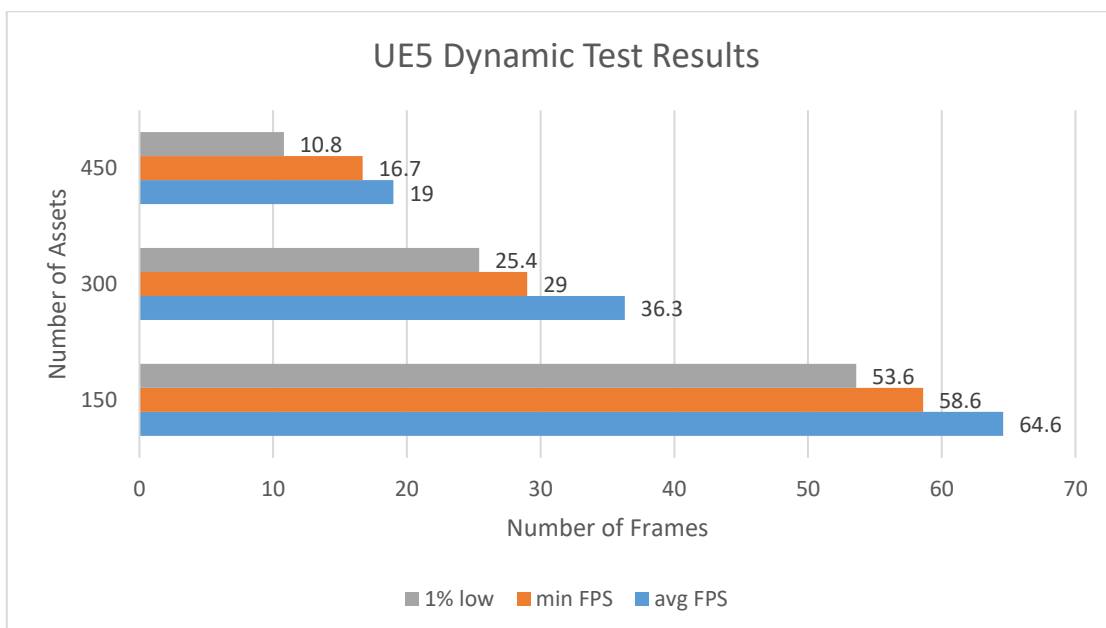


Figure 22 shows the result of the dynamic asset test in Unreal Engine 5.

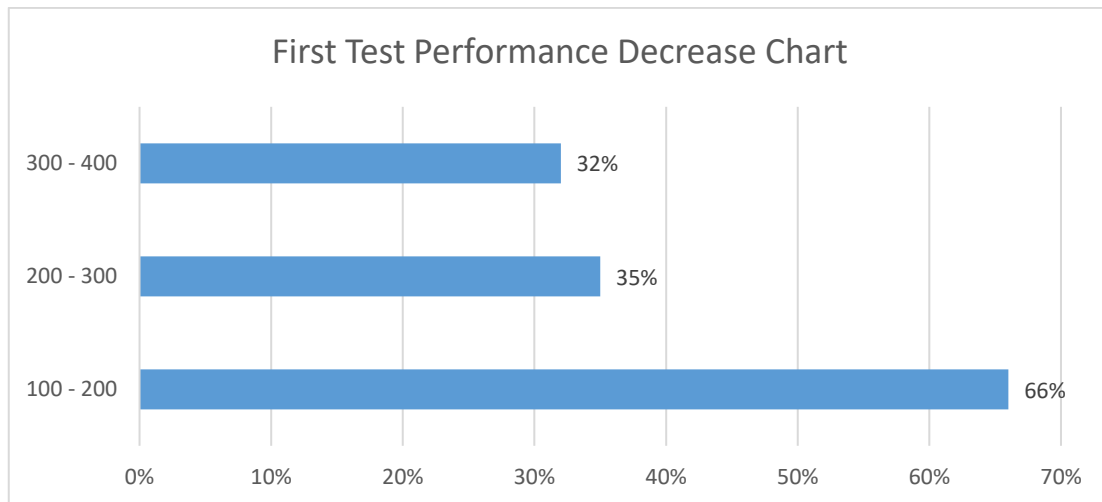


Figure 23 Result of static asset performance decrease chart in Unreal Engine 5.

The first test shows the gradual decrease of performance when increasing the number of assets in view. This result is consistent with the results of the tests from the previous game engine. Figure 23 shows the performance decrease chart of the first test.

The second test shows similar results. However, the average FPS is very close to 30FPS already at 300 assets and the minimum and 1% lows dropping below 30FPS certainly does not help. Furthermore, at 450 assets the performance drops below 30FPS too much which for a game would certainly not be playable.

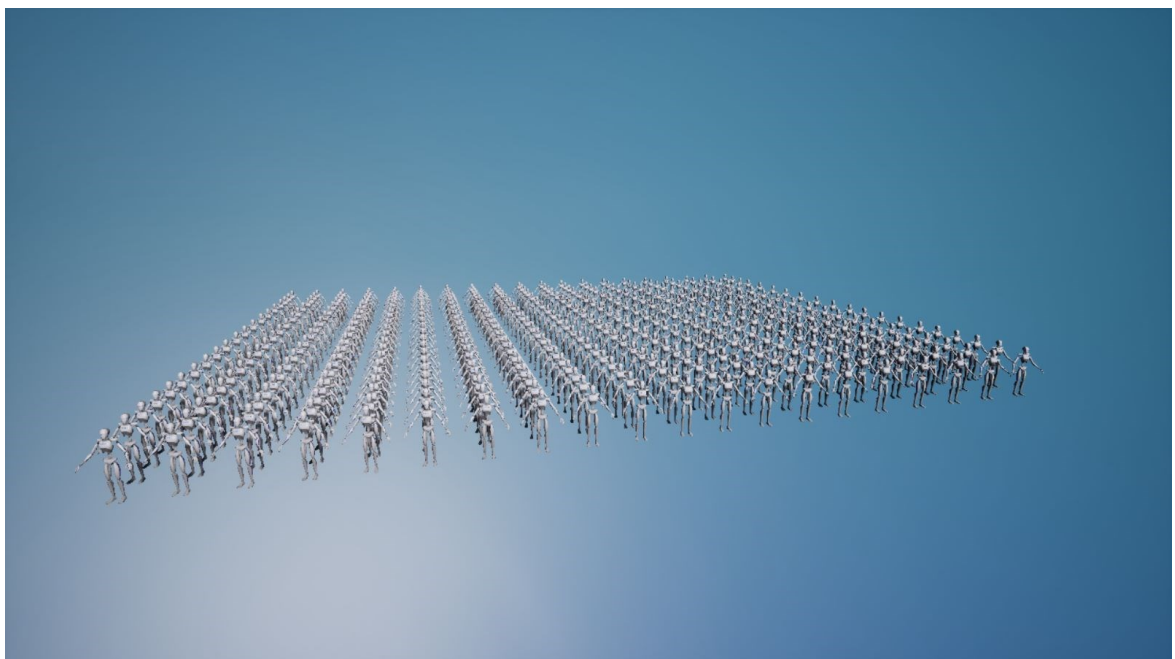


Figure 24 Image of the static assets test running in Unreal Engine 5.

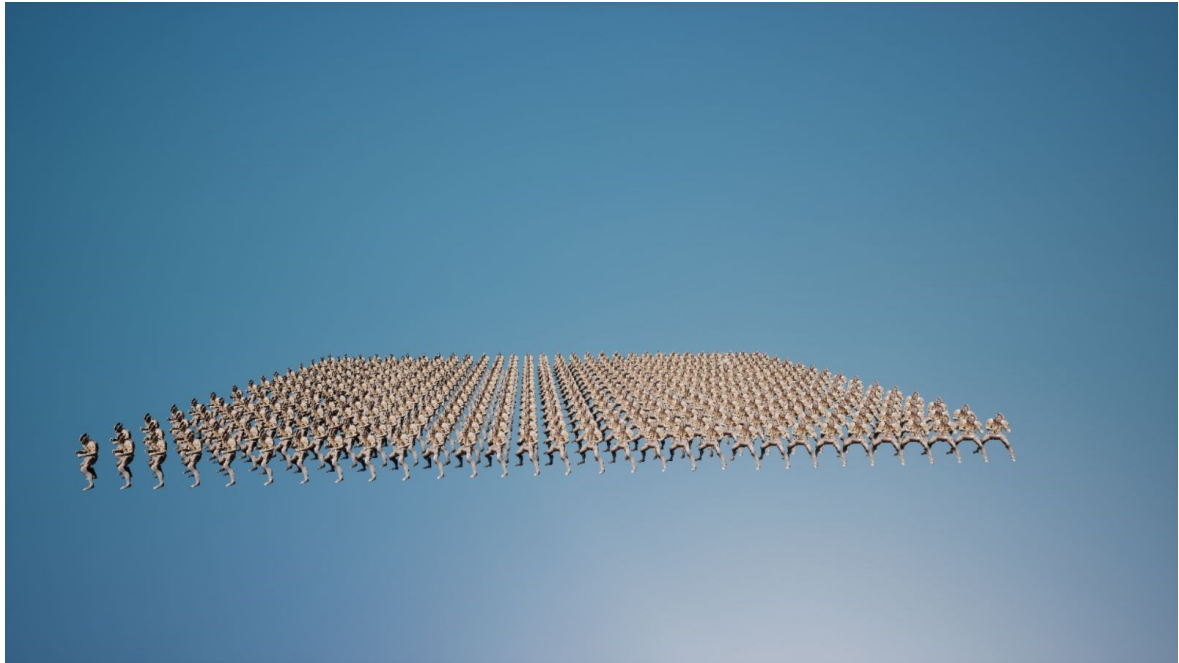


Figure 25 Image of the dynamic assts test running in Unreal Engine 5.

5.2.3 Results Analysis

The results of the first test show that the performances difference between Unity and Unreal Engine 5 were closer this time. Unity was performing better with a lower number of assets at 100 and 200 assets. However, Unreal Engine 5 was performing better when the number of assets were at 300 and 400 assets.

It must be said that this specific asset was chosen so it would be difficult for Nanite to influence the results, which is a new technology that increases the rendering performance for Unreal Engine 5 by providing dynamic LOD. When using assets that Nanite can take advantage of, Unreal Engine 5 wouldn't go lower than the maximum refresh rate of the screen even at 500 assets. The use of Nanite would give an unfair advantage for UE5 in this specific test.

The results of the second test tell a different story. Unity was performing better than Unreal Engine 5 for all number of assets. The performance difference between Unity and Unreal Engine 5 were on average around 110% while reaching 160% for the test with 450 assets.

It must be noted that Unreal Engine 5 has much better lighting and shadow system. Unreal Engine 5 also has the advantages of Lumen and Nanite which improve the rendering and the performance by a big margin. It is however not possible to put a number on how much better the rendering looks on Unreal Engine 5, but this is for the developers to choose if they want

to trade graphical quality for performance. It is possible to see the difference in quality of the rendering in Figure 2 and Figure 3.

5.3 Memory usage testing

This chapter should measure and show how much memory is used when running a project or game made with the game engines. The test should show how well the engines handle packaging the assets in the game and not use too much memory when the game is running. The less memory the game made with the engines uses the better the engine is performing.

Methodology. In each of the game engines, the projects that have been created in the previous test are running with the same number of cubes and assets. Then it is measured how much RAM the game is using Windows's Task Manager. No engine specific optimization is done regarding the memory usage, everything is left as default.

Testing Environment. All of the tests were run in the same room temperature. The only programs running were the game and Windows's Task Manager for measuring RAM usage.

Testing Hardware. The tests were done Using an ASUS ROG STRIX G17 laptop with Ryzen 7 5800h for the CPU, a RTX 3060 for the GPU, 32Gb of DDR4 RAM, and 2.5 TB of M.2 NVMe SSD. The resolution of the screen is 1980x1080 and the refresh rate is 144Hz.

Steps to do the tests.

1. The game is built and run.
2. The game is left to run for 10 minutes.
3. After the 10 minutes are finished, the RAM usage is measured in 10 seconds intervals and the average is calculated.
4. After the measuring, let the computer cooldown before doing the next test.

Projects Used for Testing. The projects used for measuring the RAM usage are:

1. Cubes collision test with 4000 cubes.
2. Static assets test with 400 assets.
3. Dynamic assets test with 450 assets.
4. The imported scene running in the highest quality.

5.3.1 Results and Analysis

Figure 26 shows the results of the test for measuring memory usage.

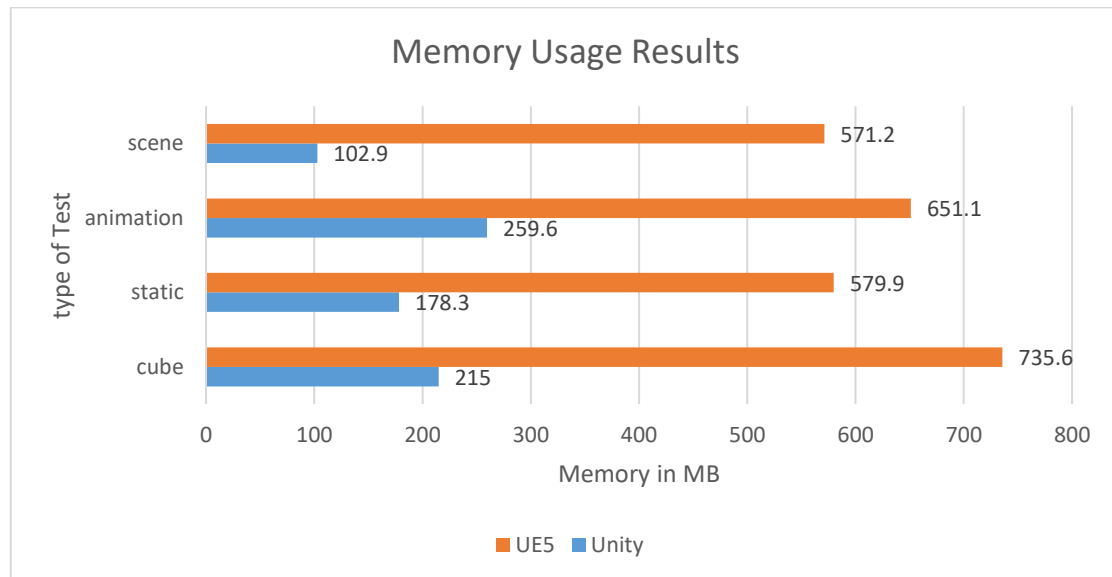


Figure 26 Results of memory usage tests in Unity and Unreal Engine 5.

The results of the test for measuring memory usage show that in all the scenarios used, Unity was using less memory than Unreal Engine 5. Figure 27 shows the difference in usage between the two engines.

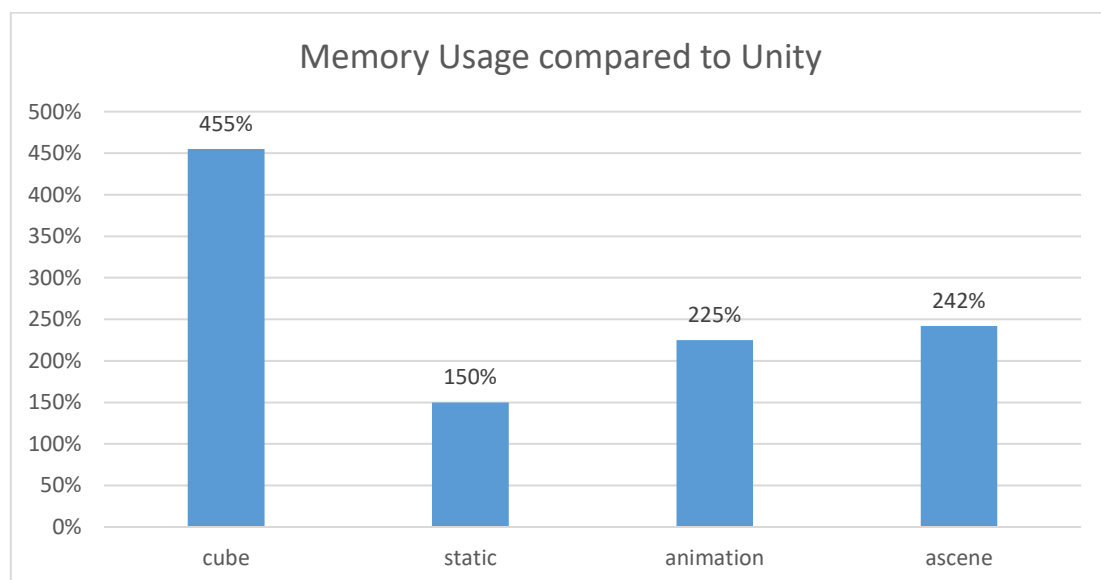


Figure 27 Chart of Unreal Engine 5 memory usage compared to Unity.

Unreal Engine 5 consistently uses more memory compared to Unity. Furthermore, the biggest difference is at the cubes collisions test where Unreal Engine 5 uses around 445% more memory compared to Unity. This difference is massive, and the difference in memory usage overall could be because of many factors. Some of the factors could be that Unreal Engine 5's rendering pipeline inherently uses more memory since it prioritizes visual quality, Unreal Engine 5 loading more assets into memory for faster access, and the engine's underlying architecture having way more overhead causing it to use more memory. None the less, in terms of memory usage comparison between the game engines, Unity is the clear winner here.

6 SOFT DATA COMPARISONS

There are aspects of game engines where it's easier to give them a numerical evaluation based on performance testing like how many FPS the engine can put out or how much memory the engine is using. However, there are also aspects of game engines where it's difficult to give a numerical value since it can be opinion based and influenced by the author's experience and personal preferences. Some examples of non-quantitative game engine aspects are accessibility, fidelity, ease of use, etc. These aspects can be considered just as important when choosing a game engine.

This chapter will go through each of these aspects and give an evaluation of them for each of the chosen game engines. As the author of this thesis, some of those evaluations are influenced by my view on what is considered good and bad, but some of the evaluations can also be considered as the overall game development community's opinion on these matters through my personal research and talking to some game developers.

Since one of the aims of the thesis is to give recommendations for developers who are just starting out with game development, and my experience as a beginner game developer reflecting that. My opinions based on my experience with game development could be valued as significant.

I have ~80 hours' worth of learning and development time with game engines. These hours don't include time away from the computer or time taken away by distractions. Of those 80 hours, I have spent ~35 hours with Unity, ~25 hours with Unreal Engine 5, ~15 hours with CryEngine 5.7 LTS, and ~5 hours with Godot.

6.1 Ease of Use

This sub-chapter will go over the user interface of the game engines, workflow, and the developer experience in general when using the gam engines' interface. This is a very important aspect. Some engines have complicated UI and make working with the engines a challenging experience and extend the time needed to learn the engine and get comfortable with it, while some engines have easy-to-use and intuitive interfaces that makes it easy for beginners to get up to speed with the interface and start developing games comfortably.

Unity. Unity has one of the best user interfaces compared to the other gam engines. The UI is uncluttered and easy to navigate. It was easy to understand which parts of the UI do what and didn't need extra time to learn how to navigate the UI, which made it easy to start developing right away. It was also easy to find here things are and find them quickly when they were needed.

Unreal Engine 5. Unreal Engine has updated its UI and made big changes with version 5 compared to Unreal Engine 4. The UI is now much better and intuitive. However, UE5 inherently being more complicated than other game engines regardless of the UI, made things a bit more difficult. Some time was needed to learn how the UI works, where everything is, and start developing. However, I would count the extra time needed to learn the UI as insignificant enough that would not brand it as complicated. Although, the extra time needed to learn the UI must still be mentioned.

The comparison between the engines in this area was really close. It would be difficult to choose one over the other as the winner in this area. The result of easy of use being a draw.

6.2 Accessibility

This sub-chapter will talk about how Accessible the engines are for someone new to game development. Going through the download and installation process, availability and quality of learning materials, and the minimum and recommended hardware requirements for running the engine (with focus for Windows). It is very important for the game engine to be accessible and for someone to easily be able to go from trying to download the engine, to learning it and start developing with it.

Unity. Unity has a very simple download and installation process. The process is further made easier by using its management software called Unity Hub. Unity also has a

comprehensive library of free learning materials made by Unity themselves, and by independent creators ease the initial learning curve. Its inclusive hardware requirements ensure it runs on a variety of systems making it possible to run on low-end hardware, but if the developer desires to create more than simple 2D games, they will have to use decently powerful and modern hardware. It is possible to get a comfortable developer experience with hardware worth ~2000\$. [41][42]

Unreal Engine 5. UE5 has a very simple and straight forward download and installation process with using the Epic Games Launcher. It can be said that UE5 has less online, free learning material compared to Unity, but it's still possible to find many online resources for learning, and with the recent boom in popularity that the engine is having, many more creators are starting to make online tutorials about it. UE5 has much more expensive hardware requirements compared to Unity. The recommended hardware requirements stated by the publishers would cost ~2200\$, but to have a comfortable experience, it will cost more. [43]

Unity should be the winner in this area of comparison. Although Unreal Engine 5 very good in terms of accessibility, Unity simply ticks all of the right marks and performs excellently in all of them.

6.3 Fidelity

This sub-chapter will talk about fidelity, graphical quality, and in general how good assets look when rendered in the engine. Each game engine has its own rendering engine that does things differently in terms of lighting, shadows, reflections, etc. How the engine handles these things gives assets rendered in the engine a certain look and feel that would be difficult to replicate in another game engine. The evaluation of the fidelity and rendering quality of an engine can be influenced by personal preferences. However, it would be difficult to deny when an engine does a good job with rendering scenes.

Unity. Unity has a very good rendering engine and with enough work, it's possible to make things look as good as any other engine, but the same could be said for other game engines as well. The rendering engine in Unity is good, but it is hard for it to compete with other AAA game engines.

Unreal Engine 5. UE5 has currently one of the best rendering engines in the market. The way the engine handles lighting, illuminations, shadows, reflections, and materials is

exceptional. Even though the engine is free to download and use, it can easily compete with any AAA game engine in terms of fidelity. From the pictures provided in the tests and the scene, it can be seen that assets look considerably better in UE5 compared to Unity.

Unreal Engine 5 would be the winner here without any doubt. The rendering quality and fidelity of Unreal Engine 5 is a notch better than the best Unity could provide.

6.4 In Engine Performance

This sub-chapter's main talking points will be how the engine is performing when using it, how heavy it is on the hardware, fan noise, etc. It's important for the developer to have a comfortable time when developing games with the engine so all of their focus would be on development, and not get distracted by things that are faults of the engine.

As the author of this thesis, I only have experience with running the game on hardware I had access to and can only speak for my own experience regarding this chapter. The specifications of the hardware are mentioned in chapter 5.1.

Unity. The development experience with Unity was very comfortable. It did not take too long when opening a project or when starting a new project. While developing games with Unity, the laptop remained quiet and was not a distraction, and except for one instance, no error/stutters/bugs were encountered. Overall, the time developing games with Unity and learning the engine was enjoyable, and it was possible to focus solely on developing and learning without facing any major problems.

Unreal Engine 5. The development experience with UE5 was more difficult compared to Unity. The time it took to open existing projects and starting projects was very similar and no errors/bugs were encountered throughout the time using the engine, but the same cannot be said for the other areas. The laptop's fans were continuously spinning at full speed even when just opening UE5 to the default scene when creating an empty project. The surface of the keyboard was also warm to the touch that made it uncomfortable after some time. The scene would sometimes stutter when moving the camera around, which is something that was not encountered when using Unity. To summarize, the time spent developing with UE5 was not as comfortable as with Unity. However, this can certainly be solved with more expensive hardware, but that for sure is a disadvantage.

The winner of this part without no doubt should be Unity. Unreal Engine 5 performs very poorly while Unity doing everything right.

6.5 Scripting Language

This sub-chapter will talk about the scripting languages used when programming in the game engines. Although it's possible to argue that the engine should not be held accountable if the language used is considered difficult, it is the language recommended by the publishers themselves to have the best intended experience. Furthermore, regardless of the language, there are still systems in the engine like making a variable available in the editor, updating on frame, at what point will the script run, etc. that make learning the engine take extra time and causing considerable confusions when just starting out.

As the author, I have had considerable prior experience with C# enough to find a job with it before starting with Unity, so I can't speak of how the development and learning experience would be for someone without experience, but I can describe my experience with C++, which although I had prior experience with, I still needed some relearning to do in order to be comfortably using it.

Unity. Unity uses C# which is easier to use compared to some other languages. The overall developer community's opinion on C# is that it is a very good language to learn as a first programming language when starting with game development. Moreover, the systems in place complement the language very well and the language made the time spent programming in the engine enjoyable overall.

Unreal Engine 5. UE5 uses C++ as its main programming language. The language is more complicated than C# and is easier to write bad code with it. However, coming from C#, learning C++ was a fulfilling experience and is easy enough to use if the developer knows what they're doing. Although still, it was difficult to get a grasp of how the systems work in the engine and how they interact with C++. With the limited development time with UE5, I still have not figured everything out and still need more experience with writing C++ in UE5 to get more comfortable with using it.

Unity and C# should be the clear winners of this part for me. C# is simply much easier to use compared to C++ while still being performant.

6.6 Regular Updates and Future Roadmap

This sub-chapter will talk about how regularly the engines receive new updates and on average, how impactful are those updates. It is very important for an engine to get updates regularly for it to stay competitive with the other game engines. It would cause a big

inconvenience for studios that use an engine, if the engine becomes very uncompetitive due to lack of regular updates. Good communication on the side of the developers of the engine is also very important. Since the game developers that use the engine are equivalent to being the customers of the engine, it is a good practice to take the feedback of the customers into account and keep them informed with clear roadmaps of what updates and new features will be coming in the future.

Unity. Unity's commitment to regular updates and transparency in its future roadmap speaks volumes about its devotion to continuous improvement and long-term planning. The team used to put out 4 major updates in a year and provide small bug fixes in between these big releases, but since 2020 the number of major updates decreased to 3 updates per year and since 2022, it was further decreased to 2 updates per year. The main reasons for this were to improve stability and quality of each update. There is also always a roadmap tab available on their website that talks about future improvements planned for the engine. [47][23]

Unreal Engine 5. Epic Games has a proven track record for delivering regular updates to Unreal Engine, and this tradition continues with Unreal Engine 5. The team does not have a strict schedule when it comes to releasing new updates. They usually release major updates when significant new changes and improvements have been implemented, and they release minor updates for bug fixes in between of those major updates. In general, the team has a very good track record of releasing updates for the engine and the pace with which they put out new features is excellent. This can be seen with the last 3 major updates, the updates being the release of version 5, 5.1, and 5.2, that each introduced many impressive features. The developers also release a new roadmap whenever they get close to releasing a new major update. [46]

Both Engines are very good in terms of releasing updates regularly, but it must be said that Unreal Engine on average has better updates compared to Unity, making it the winner.

6.7 Tools and Features

The tools and features provided by a game engine are integral to its functionality, defining its potential to create diverse and immersive gaming experiences. Those tools and features can be minor, or they can be major tools that are detrimental to the engine and are one of highlights of the engine. Ultimately, the quality and impact of these tools and features in a game engine not only impact the quality of the final game but also influence the development process, shaping the engine's ease of use, flexibility, and efficiency.

Unity. Unity has all of the necessary and needed tools that come usually with game engines, but it is lacking in the quality of some of these tools. Unity has always been known for having a mediocre terrain and foliage editor, importing fbx files usually need to be edited and updated to show properly in the engine. However, at the same time Unity has some amazing tools and features for developers who want to develop games for AR/VR. [45]

Unreal Engine 5. UE5 recently made big headlines with some of the new tools and features they have introduced. Features like Nanite and Lumen has been detrimental to its recent rise in popularity, and they continuously introduce new features like the new material framework called Substrate, procedural environment generation, new tool that lets developers record facial animations only using mobile phones, etc. Epic Games tend to update their engine when they introduce a new major feature, being a reason why the features they release are usually influential. [48]

Unreal Engine 5 is by far the clear winner in this area. The tools and features that Unreal Engine provides are much more impressive and influential to the development process.

CONCLUSION

Game development is an attractive carrier for many developers. New game engines are being announced every now and then, small game engines are getting major updates and are starting to compete with popular game engines, and the popular game engines keep getting major updates with headline features that attract everyone from indie developers to major game development studios. The aim of this thesis was to give a general introduction to game development and game engines in general, Identify the current popular game engines, make a comparison between them and give a recommendation based on the results of the comparisons.

Doing the practical comparison between the two game engines, it was clear that Unity was the better performer in most of the tests between the two game engines. Unreal Engine 5 did manage to outperform Unity in limited scenarios but overall, in those specific tests and limited testing that was done, Unity was the clear winner. However, the comparison was not limited to only these performance tests. The two engines were also compared in other areas that were not performance based. These comparisons were much closer compared to the performance comparisons. UE5 could be considered the winner of these comparisons, but the results were close. Overall, both engines excel in different areas, yet they couldn't be any closer.

There were however some limitations that were encountered while doing this thesis. Game engines being the complex pieces of software that they are, makes it difficult to study and learn multiple game engines comprehensively. Not having powerful hardware also makes the development process unnecessarily tedious with needing machines that have good cooling, big screens, powerful CPUs and GPUs, etc.

Given more powerful hardware and a greater amount of time for each of the game engines would make it possible to create better and clearer tests. Adding more game engines to the comparisons would also be a huge plus.

None the less, this thesis should give a good introduction to the world of game development, introduce the reader to certain popular game engines in the market, show them comparison between the two most popular game engines, explain the results of those tests and give a recommendation.

Finally, to give a proper recommendation. Both engines performed very well and are truly a notch above all the current publicly available game engines.

Unity showed that it is a robust and feature complete game engine. It performs very well in all areas while still being able to run well enough on low-end hardware. It is easy to user, very accessible, had decent fidelity, uses a comparatively easy scripting language, and has Adequate tools and features.

Unreal Engine 5 on the other hand showed that it was not as performant as Unity when it came to the performance tests. The engine was very heavy on the hardware, was decent in terms of ease of use, and uses a difficult scripting language. However, the engine has excellent accessibility, best-in-class fidelity, headline grabbing features, and gets updated regularly.

However, my final recommendation for developers who are just stepping into the world of game development is to choose Unity as the first engine they should learn. Unity is very welcoming towards new developers and is a great engine to become familiar with the game development process and game development in general.

BIBLIOGRAPHY

- [1] What Is Game Development?, 2019. freeCodeCamp.org [online]. Available from: <https://www.freecodecamp.org/news/what-is-game-development/>
- [2] How to Become a Game Developer in 2023? (Step-by-Step Guide), 2022. Simplilearn.com [online]. Available from: <https://www.simplilearn.com/how-to-become-a-game-developer-article>
- [3] The Ultimate Guide to Game Development, [no date]. The Ultimate Guide to Game Development [online]. Available from: <https://www.nuclino.com/articles/game-development>
- [4] Video Game Developer Job Description, [no date]. Betterteam [online]. Available from: <https://www.betterteam.com/video-game-developer-job-description>
- [5] 5 Essential Stages of Game Development Process, 2022. Innovecs [online]. Available from: <https://innovecs.com/blog/game-development-process/>
- [6] What are the stages of game development?, 2022. What are the stages of game development? | Pingle Studio [online]. Available from: <https://pinglestudio.com/blog/full-cycle-development/game-development-stages>
- [7] 9 Games That Made a Major Comeback - IGN, 2020. 9 Games That Made a Major Comeback - IGN [online]. Available from: <https://www.ign.com/articles/biggest-video-game-comebacks>
- [8] LTD., Arm, [no date]. What is a Gaming Engine? Arm | The Architecture for the Digital World [online]. Available from: <https://www.arm.com/glossary/gaming-engines>
- [9] What is a Game Engine?, 2021. Full Scale [online]. Available from: <https://fullscale.io/blog/what-is-game-engine/>
- [10] Elements of a game engine | Outsource Accelerator, 2000. Outsource Accelerator [online]. Available from: <https://www.outsourceaccelerator.com/articles/game-engine/>
- [11] Best Programming Languages for Game Development, 2021. Best Programming Languages for Game Development [online]. Available from: <https://www.mooc.org/blog/best-programming-languages-for-game-development>

- [12] TOKIO, Redacción, 2023. The best programming language for games: the list. Tokio School [online]. 23 January 2023. Available from: <https://www.tokioschool.com/en/news/best-programming-language-for-games/>
- [13] Visual Scripting - How Noodl was inspired by the world of game engines, [no date]. Visual Scripting - How Noodl was inspired by the world of game engines [online]. Available from: <https://www.noodl.net/post/visual-scripting-how-noodl-was-inspired-by-the-world-of-game-engines#:~:text=Visual%20Scripting%20in%20game%20engines,visually%20using%20nodes%20and%20connections.>
- [14] ENGINE, Godot, 2022. Godot 4.0 will discontinue VisualScript. Godot Engine [online]. 23 August 2022. Available from: <https://godotengine.org/article/godot-4-will-discontinue-visual-scripting/>
- [15] @techlearning, 2021. What Is Scratch And How Does It Work? TechLearningMagazine [online]. 24 February 2021. Available from: <https://www.techlearning.com/how-to/what-is-scratch-and-how-does-it-work>
- [16] Best Gaming Engines for Your Game Project | Kevuru Games, 2022. Kevuru Games [online]. Available from: <https://kevurugames.com/blog/best-game-engines-2022-pros-cons-and-top-picks-for-different-types-of-games/>
- [17] SOFTWARE, Arnia, [no date]. What Makes Unity So Popular in Game Development? Arnia Software [online]. Available from: <https://www.arnia.com/what-makes-unity-so-popular-in-game-development/>
- [18] SCHARDON, Lindsay, 2023. What is Unity? – A Guide for One of the Top Game Engines. GameDev Academy [online]. 13 January 2023. Available from: <https://gamedevacademy.org/what-is-unity/>
- [19] Unity Pros and Cons, [no date]. Unity Pros and Cons [online]. Available from: <https://ventionteams.com/blog/unity-pros-and-cons>
- [20] SUTEVSKI, Dragan, 2021. Disadvantages of Unity Game Development. Entrepreneurship in a Box [online]. 1 June 2021. Available from: <https://www.entrepreneurshipinabox.com/25226/disadvantages-of-unity-game-development/>

- [21] TECHNOLOGIES, Unity, [no date]. Plans and pricing. Unity [online]. Available from: <https://unity.com/pricing>
- [22] Top Programming Languages 2022, 2022. IEEE Spectrum [online]. Available from: <https://spectrum.ieee.org/top-programming-languages-2022>
- [23] TECHNOLOGIES, Unity, [no date]. Unity Platform Roadmap. Unity [online]. Available from: <https://unity.com/roadmap/unity-platform>
- [24] Top 10 Games Made With Unreal Engine | Vagon, 2022. vagon [online]. Available from: <https://vagon.io/blog/top-10-games-made-with-unreal-engine/>
- [25] Unity vs Unreal Engine: Differences and Performance Comparison, 2023. Kevuru Games [online]. Available from: <https://kevurugames.com/blog/unity-vs-unreal-engine-pros-and-cons/>
- [26] Nanite Virtualized Geometry, [no date]. Nanite Virtualized Geometry in Unreal Engine | Unreal Engine 5.0 Documentation [online]. Available from: <https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/>
- [27] Unreal Engine (UE5) licensing options, [no date]. Unreal Engine [online]. Available from: <https://www.unrealengine.com/en-US/license>
- [28] Unreal Engine History: Founding, Timeline, and Milestones - Zippia, 2020. Unreal Engine History: Founding, Timeline, and Milestones - Zippia [online]. Available from: <https://www.zippia.com/unreal-engine-careers-2149448/history/>
- [29] 10 Best Games Made With CryEngine, 2022. 10 Best Games Made With CryEngine [online]. Available from: <https://gamerant.com/best-games-made-with-cryengine/>
- [30] CRYENGINE | Roadmap, [no date]. CRYENGINE [online]. Available from: <https://www.cryengine.com/roadmap>
- [31] CRYENGINE | Support: Licensing, [no date]. CRYENGINE [online]. Available from: <https://www.cryengine.com/support/view/licensing>
- [32] CRYENGINE V Manual - CRYENGINE V Manual - Documentation, [no date]. CRYENGINE V Manual - CRYENGINE V Manual - Documentation [online]. Available from: <https://docs.cryengine.com/>

- [33] CRYENGINE | Support: General, [no date]. CRYENGINE [online]. Available from: <https://www.cryengine.com/support/view/general>
- [34] PATEL, Niel, 2023. Top 10 Popular Games Made With Godot Engine. Make An App Like [online]. 6 January 2023. Available from: <https://makeanapplike.com/best-games-made-with-godot/>
- [35] ENGINE, Godot, 2014. Godot history in images! Godot Engine [online]. 15 January 2014. Available from: <https://godotengine.org/article/godot-history-images/>
- [36] Unity vs Godot - 2022 Complete Game Engine Comparison, 2019. Video Game Design and Development [online]. Available from: <https://www.gamedesigning.org/engines/unity-vs-godot/>
- [37] ENGINE, Godot, 2023. Dev snapshot: Godot 4.1 dev 1. Godot Engine [online]. 21 April 2023. Available from: <https://godotengine.org/article/dev-snapshot-godot-4-1-dev-1/>
- [38] Vionix, 2022. URP vs HDRP in Unity - VionixStudio. VionixStudio [online]. 31 May 2022. Available from: <https://vionixstudio.com/2022/02/12/urp-vs-hdrp-in-unity/>
- [39] Lumen Global Illumination and Reflections, [no date]. Lumen Global Illumination and Reflections in Unreal Engine | Unreal Engine 5.0 Documentation [online]. Available from: <https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/>
- [40] Game Engines (Types & Purposes), 2013. Game Engines (Types & Purposes) | Jahmel Coleman [online]. Available from: <https://jahmelcoleman.wordpress.com/games-development/game-engines/>
- [41] TECHNOLOGIES, Unity, [no date]. Unity - Manual: System requirements for Unity 2021.1. Unity - Manual: System requirements for Unity 2021.1 [online]. Available from: <https://docs.unity.cn/2021.1/Documentation/Manual/system-requirements.html>
- [42] GLAWION, Alex, 2021. Unity 3D System Requirements & PC Recommendations. CG Director [online]. 18 October 2021. Available from: <https://www.cgdirector.com/unity-system-requirements/>

- [43] catness, 2023. Unreal Engine 5 Requirements - Catness Game Studios. Catness Game Studios [online]. 20 February 2023. Available from: <https://catnessgames.com/blog/unreal-engine-5-equirements/>
- [44] 7 Best Programming Languages for Game Development in 2023, [no date]. Hackr.io [online]. Available from: <https://hackr.io/blog/best-programming-language-for-games>
- [45] Unity Engine - is it bad & what is it good for?, 2022. NeuroSYS [online]. Available from: <https://neurosys.com/blog/why-people-say-unity-engine-is-bad>
- [46] <https://portal.productboard.com/epicgames/1-unreal-engine-public-roadmap/tabs/85-unreal-engine-5-2>, [no date]. [online]. Available from: <https://portal.productboard.com/epicgames/1-unreal-engine-public-roadmap/tabs/85-unreal-engine-5-2>
- [47] TECHNOLOGIES, Unity, 2023. Download Archive. Unity [online]. 18 May 2023. Available from: <https://unity.com/releases/editor/archive>
- [48] Unity vs Unreal: Which Game Engine Should You Choose?, [no date]. Hackr.io [online]. Available from: <https://hackr.io/blog/unity-vs-unreal-engine>
- [49] SCHARDON, Lindsay, 2023. What is Unity? – A Guide for One of the Top Game Engines. GameDev Academy [online]. 13 January 2023. Available from: <https://gamedevacademy.org/what-is-unity/>
- [50] Unity Gaming Report 2022: 5 insights on the gaming industry today | Unity Blog, [no date]. Unity Blog [online]. Available from: <https://blog.unity.com/games/unity-gaming-report-5-insights-on-the-gaming-industry-today>
- [51] XSENS Mocap Retarget Project for Unreal Engine | 3D model, [no date]. CGTrader [online]. Available from: <https://www.cgtrader.com/free-3d-models/character/sci-fi-character/xsens-mocap-retarget-project-for-unreal-engine>
- [52] Free Game Character - Ancient - Download Free 3D model by Kaan Tezcan (@kaanTezcan), [no date]. Sketchfab [online]. Available from: <https://sketchfab.com/models/19e18c8e8d5941deb3f9dd0a2a349b56/embed?autostart=1>

LIST OF ABBREVIATIONS

2D	Second dimension
3D	Third dimension
AI	Artificial Intelligence
API	Application Programming Interface
AR	Artificial Reality
CPU	Central Processing Unit
FPS	Frames per Second
Gb	Giga Byte
GDD	Game Design Document
GDScript	Godot Script
GPU	Graphical Processing Unit
HDRP	High Definition Rendering Pipeline
HTML5	HyperText Markup Language version 5
Hz	Hertz
LOD	Level of Detail
LTS	Long Term Support
MIT	Massachusetts Institute of Technology
NPC	Non Playable Character
NVMe	Non-Volatile Memory Express
OOP	Object Oriented Programming
QA	Quality Assurance
RAM	Random Access Memory
ROG	Republic of Gaming
SSD	Solid State Drive
TB	Tera Byte

UE5	Unreal Engine 5
UI	User Interface
URP	Universal Rendering Pipeline
US	United States
USA	United States of America
VR	Virtual Reality
VFX	Visual Effects

LIST OF FIGURES

Figure 1 Image of the scene rendered in Blender.	38
Figure 2 Image of the scene rendered in Unity.	39
Figure 3 Image of the scene rendered in Unreal Engine 5.	40
Figure 4 Results of the cube tests in Unity.	42
Figure 5 Image of the cubes test running.	43
Figure 6 Image of the cubes test open in the Unity editor.	43
Figure 7 Image of the code responsible for moving cubes to the middle.	44
Figure 8 Image of the code responsible for spawning the cubes.	44
Figure 9 Result of the cube tests in Unreal Engine 5.	45
Figure 10 Image of the cubes test running in Unreal Engine 5.	46
Figure 11 Image of the cubes test open in the Unreal Engine 5 editor.	46
Figure 12 Image of the code responsible for spawning the cubes.	47
Figure 13 Image of the code responsible for moving the cube towards the middle.	48
Figure 14 Image of the asset with animation of an ancient fighter.	50
Figure 15 Image of the static asset of a mannequin.	50
Figure 16 Results of the static asset tests in Unity.	51
Figure 17 Results of the dynamic asset tests in Unity.	51
Figure 18 Result of static asset performance decrease chart in Unity.	52
Figure 19 Image of the static asset test running in Unity.	53
Figure 20 Image of the dynamic asset with animation running in Unity.	53
Figure 21 Results of the static asset tests in Unreal Engine 5.	54
Figure 22 shows the result of the dynamic asset test in Unreal Engine 5.	54
Figure 23 Result of static asset performance decrease chart in Unreal Engine 5.	55
Figure 24 Image of the static assets test running in Unreal Engine 5.	55
Figure 25 Image of the dynamic assts test running in Unreal Engine 5.	56
Figure 26 Results of memory usage tests in Unity and Unreal Engine 5.	58
Figure 27 Chart of Unreal Engine 5 memory usage compared to Unity.	58

LIST OF TABLES

No table of figures entries found.

APPENDICES

Appendix P I: Appendix title

Appendix P II: Appendix title

