Creation of 3D Characters and an Environment for a multiplayer fast-paced sports and shooter game

Bachelor's Degree Final Project

Adrià Pons Mensa

Director: Josep Serrano

Degree: Video Games Design and Development

Academic Year: 2024-25

University: Centre de la Imatge i Tecnologia Multimèdia (CITM-UPC)

To my thesis director, Josep Serrano, for his constant help and mentorship;

To Marc Ripoll and Pau Sánchez, for their valuable evaluation and academic insights;

To my parents, for their support and for always enabling me to pursue the path I chose;

Finally, to my partner, Laia, for her unconditional support;

To all of them, thank you.

Index

Figures Index

Tables Index

Summary [ENG]

The following report explains the whole process of creating three-dimensional art assets for a prototype of a game called *HyperStrike*, produced at a Vertical Slice level. The development has been focused on creating three different game-ready characters, as well as the environmental art for the game's main scenario. All the 3D assets found in the project -meshes, textures, materials, scenarios, lighting, particles- have been created from zero. The rest of the non-three-dimensional assets, such as audio or animations, have been extracted from third-party free-to-use tools.

The project proposes the challenge of producing the 3D assets of a game in all its stages, from the preliminary idea until the final development, combining two game famous game genres such as sports and hero shooters, with the final objective of creating an enjoyable gameplay experience from the hybrid of these two styles, as multiple games have been successful in their individual genres, but not in combining them together.

From the perspective of the author, the objectives set have been investigating and researching the processes of creation of three-dimensional digital art in the context of video games, getting to know which are the best workflows in the 3D industry, while developing a personal pipeline, and creating the 3D assets (both characters and environments) for a game prototype that results in a good-looking and attractive portfolio piece, without forgetting a realistic but effective planification schedule within the available time following a combination of the *Gantt Diagram* and *Agile* methodologies.

The development process of the project resulted in a nice-looking game prototype considered to be a Vertical Slice, in which the core idea, design, mechanics, and visuals of the game are conveyed into a single piece, notwithstanding, some parts of the it could have been improved in order to achieve an even better final product.

Resumen [ESP]

La siguiente memoria explica todo el proceso de creación de piezas artísticas tridimensionales para un prototipo de un juego llamado HyperStrike, producido a nivel de *Vertical Slice*. El desarrollo se ha centrado en la creación de tres personajes diferentes, así como el arte para el escenario principal del juego. Todos las piezas 3D del proyecto -mallas, texturas, materiales, escenarios, iluminación, partículas- han sido creadas desde cero. El resto de activos no tridimensionales, como el audio o las animaciones, se han extraído de herramientas de uso libre de terceros.

El proyecto propone el reto de producir los *assets* 3D de un juego en todas sus fases, desde la idea preliminar hasta el desarrollo final, combinando dos géneros de videojuegos tan conocidos como los deportes y los shooters de héroes, con el objetivo final de crear una experiencia de juego agradable a partir del híbrido de estos dos estilos, ya que múltiples juegos han tenido éxito en sus géneros individuales, pero no al combinarlos.

Desde la perspectiva del autor, los objetivos planteados han sido investigar y profundizar en los procesos de creación de arte digital tridimensional en el contexto de los videojuegos, conocer cuáles son los mejores flujos de trabajo en la industria 3D, a la vez que desarrollar un *pipeline* personal y crear los *assets* 3D (tanto personajes como entornos) para un prototipo de juego que resulte en una pieza de portfolio atractiva, sin olvidar un calendario de planificación realista pero efectivo dentro del tiempo disponible siguiendo una combinación del Diagrama de Gantt y metodologías ágiles.

El proceso de desarrollo del proyecto dio lugar a un prototipo de videojuego considerado como una *Vertical Slice*, en el que la idea central, el diseño, las mecánicas y los efectos visuales del juego se unen en una sola pieza, a pesar de que algunas partes del mismo podrían haberse mejorado con el fin de lograr un producto final aún mejor.

Resum [CAT]

La següent memòria explica tot el procés de creació de peces artístiques tridimensionals per a un prototip d'un joc anomenat HyperStrike, produït a nivell de *Vertical Slice*. El desenvolupament s'ha centrat en la creació de tres personatges diferents i l'art per a l'escenari principal del joc. Totes les peces 3D del projecte -malles, textures, materials, escenaris, il·luminació, partícules- han estat creades des de zero. La resta de peces no tridimensionals, com l'àudio o les animacions, s'han extret d'eines d'ús lliure de tercers.

El projecte proposa el repte de produir els *assets* 3D d'un joc en totes les seves fases, des de la idea preliminar fins al desenvolupament final, combinant dos gèneres de videojocs tan coneguts com els esports i els shooters d'herois, amb l'objectiu final de crear una experiència de joc agradable a partir de l'híbrid d'aquests dos estils, ja que múltiples jocs han tingut èxit en els seus gèneres, però no al combinar-los.

Des de la perspectiva de l'autor, els objectius plantejats han estat investigar i aprofundir en els processos de creació d'art digital tridimensional en el context dels videojocs, conèixer quins són els millors fluxos de treball a la indústria 3D, alhora que desenvolupar un *pipeline* personal i crear els *assets* 3D (tant personatges com entorns) per a un prototip de joc que resulti en una peça final realista però efectiva dins del temps disponible seguint una combinació del Diagrama de Gantt i metodologies àgils.

El procés de desenvolupament del projecte ha donat lloc a un prototip de videojoc considerat com una *Vertical Slice*, en què la idea central, el disseny, les mecàniques i els efectes visuals del joc s'uneixen en una sola peça, malgrat que algunes parts del mateix podrien haver millorat per tal d'aconseguir un producte final encara millor.

Keywords

Prototype, 3D Modeling, Assets, Character, Environment, Scenario, *Unity*, Stylized, Pipeline.

Links

- *HyperStrike* Latest Release: https://github.com/Historn/HyperStrike/releases
- *HyperStrike* GitHub Repository: https://github.com/Historn/HyperStrike
- Project Presentation Video: Bachelor's Final Degree Project - HyperStrike's 3D Art | Adria Pons [YouTube]

Glossary

Asset: Name given to any digital content that goes into a video game.

Baking: The process of saving information about a 3D mesh surface into a 2D texture file named "bitmap".

Blockout: Rough draft/initial version of level or character created with simple 3D shapes, without giving much details or polish.

Bug: Error or flaw that creates an unintended behavior or bad functioning in a video game.

Digital Sculpting: Process of creating a 3D model by moving, pushing, pulling, and smoothing the surface, as if it was digital clay, in specialized computer software.

Game Engine: A software program or environment designed for the creation of video games or interactive applications, thanks to its tools and features.

Hardware: Term used to describe the tangible components of a computer that are in charge of performing the instructions given by the computer's software.

High-Poly: Name given to 3D models that have a high level of detail due to having a high polygon count and complex geometry.

Low-Poly: Name given to 3D models that are optimized for using in real-time rendering applications, such as game engines, because of having a low polygon count.

Material: Term used to describe what is layered on top of the surface of a 3D model. It shows how the object is perceived when being rendered.

Mesh: It is the structural build of a 3D model consisting of polygons that are connected between them, giving the object a shape in the three-dimensional space.

Modularity: A technique used to create a large scenario by using simpler pieces that are perfectly connected to each other.

PBR: Stands for Physically Based Rendering, and it is the process of shading and rendering that tries to replicate how light interacts with the properties of a real-life material.

Polygon: It is a connected, closed array or sequence of vertices and edges that normally form a flat, planar surface.

Prototype: A Preliminary model of a product, which could be an app or video game, that allows its developers to test mechanics, art styles, identify problems, and the viability of the product before developing a final, polished version.

Retopology: Process of recreating or redefining the polygon topology of a 3D model to result in a much lower mesh resolution object that is optimized for being used in real-time rendering applications.

Shader: Term used to describe the mathematical calculations and algorithms for calculating the color of each pixel rendered on the screen, based on lighting and material configurations.

Software: Term used to describe the programs used to perform specific tasks.

Texturing: The process of creating and applying colored texture images to a 3D model surface.

UI: System of visual elements used in video games to display information to the player or to allow interaction with the application's elements.

UV Map: Type of vertex map that stores the horizontal (U) and vertical (V) positions of a 3D model's unfolded surface on a 2D texture.

Vertical Slice: Development of a portion of a video game that acts as a proof of concept for stakeholders or investors. Different from a prototype, a vertical slice is used to show the gameplay and basic visuals of the finished game.

1. Introduction

The gaming world keeps changing, with new types and mix-up play styles coming up to meet the growing need for new ideas. Games where players face each other, mixing sports and action parts, are now more liked because they have fast-paced mobility mechanics, and you can play them for hours and hours. Some examples of these are Rocket League or Overwatch, which show how fun skill-based matches in good arenas and scenarios can be, making the gameplay alive and fun.

The game proposed and developed in this project, *HyperStrike*, plans to mix sports and shooter styles. It creates a fast, group play where players try to score by hitting a ball into the other team's goal while they use guns, abilities, and cool moving mechanics like running on walls and jumping past things that block the way. The main part of the game is set in a big, but closed-off space, like a future soccer field, with things placed in a way to block paths and give power-ups to make play more fun and smart.

This will be the result of the bachelor's degree thesis project of three students of Video Games Design & Development. Each developer will focus on a key part of making a game: coding, game design, and 3D art. The scope of the project aims to be a vertical slice prototype, a demo that shows the basic ways to play, what it looks like, and if the idea can work well with the technology used during the whole development process.

This document will report the work done as an artist, focused on making 3D assets like characters and an environment. It focuses on producing the players' avatars they will be playing within the game, and the scenario they are in. The aim is to use game development's industry pipeline for 3D art modeling, sculpting, retopology, and texturing, to make sure the end work fits with the desired game art and feel, as well as do a market study of some studios and companies from the industry in order to follow a correct workflow in the creation of 3D.

The following sections of this introduction give a full look at how this project has been developed, what issues are faced to try to solve, and what has been planned to do.

1.1. Motivation

The motivation supporting the realization of this project is caused by different points. The first of them has been the idea of creating a new type of gameplay that could be innovative. In the last few years, some games have been released that combine a hybrid between different genres and elements, creating engaging video games. The key element of this is that they mix well-known elements that can be seen as simple, but once merged, they have a very enjoyable outcome. Some examples are Rocket League (2015), Overwatch 1/2 (2016, 2022), and Marvel Rivals (2024). This project tries to find players from this market niche[1] thanks to the combination of fast-paced mechanics from soccer sport and hero shooter abilities.

Moreover, this project also aims to make the developers grow in terms of professional skills and gain hands-on game development industry experience. It offers the opportunity to work simulating an indie studio workflow, applying practical elements learned during the bachelor's degree, resulting in a portfolio piece that can be showcased to the world.

1.2. Problem Formulation

Since the beginning of the bachelor's degree studies in video game design & development, the main challenge and ultimate goal has been gaining sufficient experience and knowledge to develop a preliminary game idea until the development of the whole video game.

Taking this into account, the project's primary objective has been to combine two game genres like sports and hero shooters, thanks to creating an effective game design, visuals, and prototyping with the available resources and team of developers. The objective is to create an enjoyable gameplay experience from the hybrid of these styles. Some games are successful in each of the genres individually, but there may be a lack of titles that integrate both of them and have the potential to triumph, and here is where the formulation and development of this project came to life.

[1] Market niche: Specific segments of consumers who share certain characteristics and qualities with one another. (https://www.coursera.org/articles/niche-market)

1.3. General Objectives

The general objectives of the project are:

- Investigate, learn, and apply the application and workflow pipeline of industry-standard 3D asset creation for characters and environments.
- Develop a playable prototype of *HyperStrike* at a vertical slice level, mixing sports and hero shooter elements, that can be presented in personal portfolio.
- Expand the already existing knowledge with the software used for the development of the project.
- Investigate and research the desired art style.
- Elaborate and find a personal workflow that balances quality and available resources.
- Document the study and process behind the creation of a game prototype from a simple game concept.

1.4. Specific Objectives

The specific objectives of this project are related to both the development and personal side of what it is intended to accomplish. These objectives are:

- Research the processes of 3D digital art creation and document it.
- Create a 3D Character with the traits of a DPS[2] from the initial concept research to its final implementation within Unity. This includes the alignment of the character with the game design, together with the sculpting, retopology, baking, and texturing of it from scratch.
- Create a 3D Character with the traits of a Tank[3] from the initial concept research to its final implementation within Unity. This includes the alignment of the character with the game design, together with the sculpting, retopology, baking, and texturing of it from scratch.
- Create a 3D Character with the traits of a Healer[4] from the initial concept research to its final implementation within Unity. This includes the alignment of

[2] DPS stands for 'Damage per second'. It is used when talking about a character class archetype that deals a great amount of damage to other players. (https://gametree.me/gaming-terms/dps/)
[3] Term used when talking about a character class archetype that has high armor or defense and health stats. (https://www.quora.com/What-is-a-tank-in-RPG)
[4] Term used when talking about a character class archetype that its main objective is to restore the health of its allies. (https://en.wikipedia.org/wiki/Healer_(video_games))

the character with the game design, together with the sculpting, retopology, baking, and texturing of it from scratch.

- Construct and compose a modular 3D Environment simulating a closed football stadium with objects and walls placed strategically from the initial concept research to its final implementation within Unity. This includes aligning the environment and scenario with the game design, together with the modeling, texturing, illumination, and FXs.
- Work and master the industry-standard pipeline to achieve the desired assets' quality without forgetting the performance within the game engine using baking techniques, correct retopology, rational texture sizes, etc.
- Implement a 3D main menu system using Unity's Cinemachine cameras.
- Create a custom shader using Unity's ShaderGraphs to enhance the overall game look and feel.
- Follow an appropriate naming convention for all the assets and their folder organization inside the Unity project.
- Learn and master the newest Unity version number 6.x.
- Plan a realistic work schedule and track each task using the selected project management tools and methodology.
- Establish effective communication with the rest of the development team during the whole project, providing constructive and useful feedback.

1.5. Scope of the Project

Once all the objectives have been set, a perspective view of the project can be seen. Two main parts can be distinguished, one is the development of the project itself, and the other one is the investigation, documentation, and elaboration of the report. It is important to break down each of them to organize all the tasks within the time available and use the project management methodology selected, in this case, the *Gantt Diagram*.

Taking into account that there are only 3 developers for this project. Each one is in charge of one entire area (programming, design, and art), the focus has been set on delivering a vertical slice demo that showcases the core gameplay mechanics, the characters, and scenario 3D art because the project is limited by academic boundaries of timeframe and available resources, so the emphasis has been set on the most impactful

elements. This means that aspects such as extended gameplay, advanced audio design, and full-set animation systems will rely on 3rd party free-to-use royalty assets.

Talking about the target audience of the project, the product is aimed at different segments. First, we have the academic evaluators and faculty teachers, who will be the ones who will be giving assessments during the process of development, as well as evaluating the evolution and final version of the prototype. Then there are the professionals and possible stakeholders from the industry, as the project can be seen as an important portfolio piece aimed at employers and game development studios, thanks to the replication of a realistic development pipeline, demonstrating the capability of delivering a product within a set of resources and limited budget. And last, there is the most important audience if we refer to the game itself, the players. Even though the project aims to be a prototype at a vertical slice level, it takes into account the design and experience of the end users, as they will be the ones using the final product delivered from this project.

Other beneficiaries could be future developers, artists, and academic students. The project could be useful for future game developers who might be interested in getting involved in a similar journey. So this project could be a guide for them to draw on the documentation of the process, problems solved, methodology used, etc. It can be used as a practical case study to know which are the best practices to follow. We also have those artists who might take it as an inspiration source and reference for the visuals, workflow pipeline, or any other aspect that could be relevant for their future projects. Finally, there are the students and the academic community of the *Polytechnic University of Catalonia (UPC)* who can profit from the project report and the documentation of the whole process, its insights, methodology, and more, thanks to the publication of the project to the open source repository of knowledge from the UPC School called *UPCommons*[5].

[5] UPCommons access: https://upcommons.upc.edu/handle/2099.1/18401

2. State of the Art

2.1. Digital 3D Art

Digital 3D art is the term used when talking about any digital artwork that showcases its shape within a three-dimensional space, with height, width, and depth. It can be seen from any angle that it will have a volume.

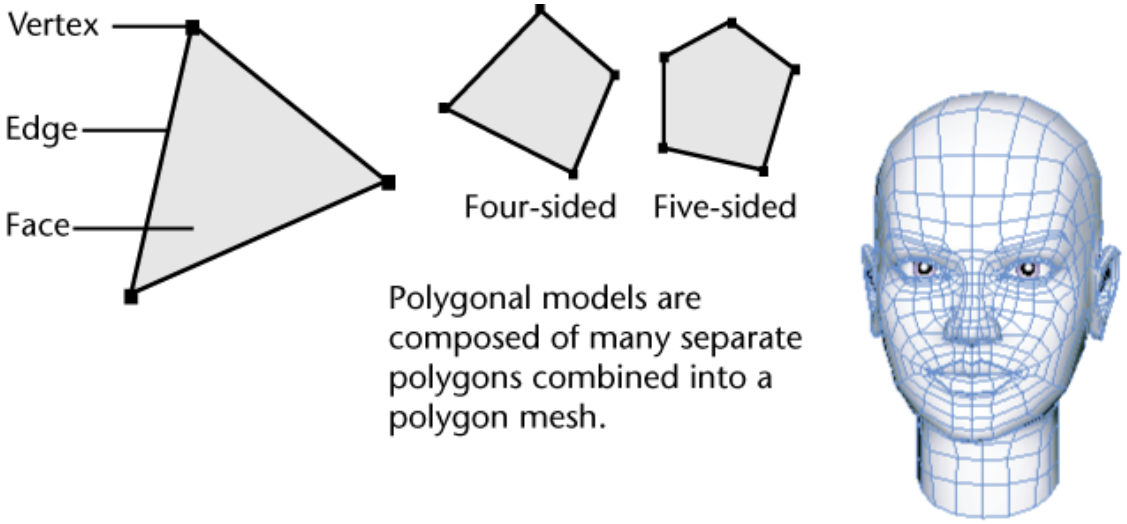
This type of art has been gaining more importance during the last decades, playing a crucial role when it comes to the creation of content for a diverse range of industries such as gaming, entertainment, architecture, etc. It is used for games, films, cartoons, digital sculpting, the creation of characters, and many other applications. To do so, there are different popular techniques in this art form, like modeling, sculpting, texturing, rendering, lighting, baking, etc. However, all of them have one thing in common: any piece of digital 3D art needs to go from the conceptualization of the idea to the final rendering of the art piece.

2.1.1. 3D Modeling

The concept of 3D modeling refers to the creation of computer-based images in a three-dimensional way. It is often thought that this process only implies the 3D object(s) created themselves, but a virtual space, lighting, and a camera or eye that renders the scene are also needed.

3D models are a representation of a set of polygons. These polygons are represented thanks to an array of vertices located in the virtual space mentioned before. Each of these vertices are dots located in a three-dimensional position in the X, Y, and Z axes. When vertices are connected, different shapes can be created. If two vertices are united, a line is created, and if continuing to connect more vertices irregularly, a curve is generated. If three vertices are united, a triangle (commonly known as “tri”) is created. If four vertices are united, a quad is generated. When more than three vertices are united, a polygon or face is generated. This concept refers to the creation of a flat surface because of the union of these dots, generating edges between the vertices. If there are polygons with more than four vertices or, as a consequence, five or more edges, an “n-gon” is created.

Figure 2.1.1.1: 3D Modeling Basics



(Source: Autodesk)

To sum up, 3D modeling consists of creating, manipulating, and shaping vertices, the most essential element. When multiple polygons are put together, they form what is known as a mesh, which can be manipulated using a large set of tools that are found within 3D modeling software programs.

2.1.2. 3D Digital Sculpting

3D Digital Sculpting is another technique to create 3D objects, but it is different from typical 3D modeling. As explained in the previous section, modeling normally implies the manipulation of vertices, edges, and polygons. This manipulation is often done with a low mesh resolution, meaning that the 3D object has the minimum amount of polygons needed to create the desired shape. This is because modeling is the easiest way to create hard-surface objects. These objects are precise, and they are called like that because they are commonly used when creating mechanical parts, architectural designs, etc. 3D Modeling is usually used to create inorganic[6] objects.

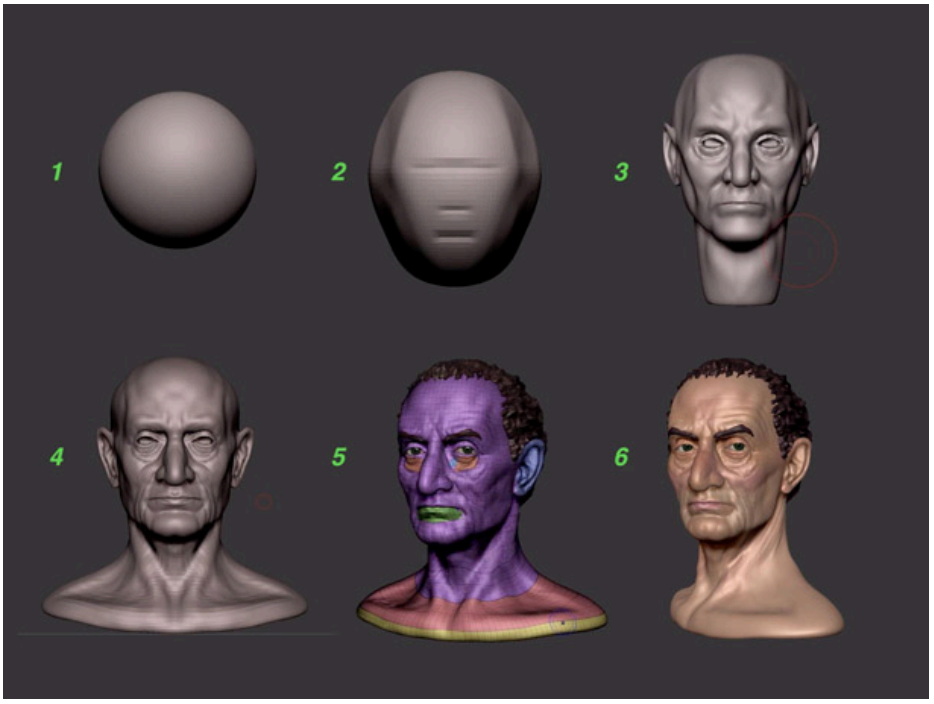
However, if you want to create organic objects, Digital Sculpting is the best option. This technique replicates clay sculpting but in a digital way. It allows you to freeform anything by moving, pushing, pulling, and smoothing the surface of objects. To make this possible, a very high resolution is needed, as at the end of the day, what is being

[6]Inorganic: Type of modeling used when creating geometric and structured shapes, such as buildings, vehicles, machinery, etc.

modified are polygons, and thousands or even millions of them are needed to achieve a natural result.

This kind of modeling is often used when making characters and creatures because it allows the creation of human-like expressions, forms, and silhouettes in a much faster way than manual 3D modeling. The level of detail and refinement that you can put in your models is enormous, and it can be done from a primitive shape like a sphere. It is a technique that can be more intuitive and easier to learn, but there are some drawbacks if the created 3D object needs to be used for real-time cinematics, video games, or applications.

Figure 2.1.2.1: Digital Sculpting Process



(Source: CGChannel.com)

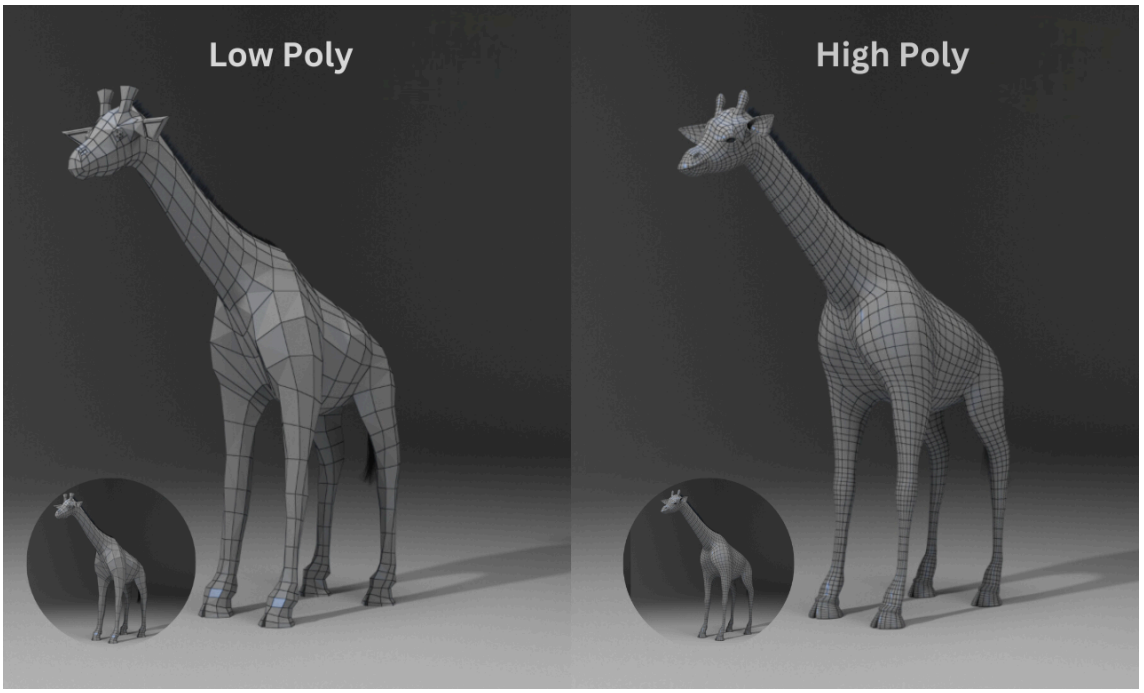
One of the main drawbacks of digitally sculpted 3D models is that, as the mesh resolution is very high and there are millions of polygons, the model is not optimal to use in real-time rendering programs such as game engines because it consumes a lot of resources in terms of computer performance, so working with them is not viable. The solution to this problem is to carry the sculpted model through the process of retopology.

2.1.3. Retopology: High Poly to Low Poly

As seen in the previous section, when using digital sculpting, the final sculpted object has a gigantic resolution, so it is not performance-optimized to be used in real-time applications. The solution to this problem is a technique known as retopology, a process of recreating or redefining the topology of a 3D model to result in a much lower mesh resolution. In other words, the objective is to eliminate the maximum number of polygons possible without losing the shape and quality of the 3D model, so it is extremely lightweight and easier to run.

Thanks to this process, you go from what is called a High Poly (a model with high mesh resolution, in this context, the sculpted object) to a Low Poly model (low mesh resolution). Performing this process makes sure that the model's topology is optimized and clean for later usage in fields like animation or UV Mapping while maintaining the details created during the sculpting process.

Figure 2.1.3.1: Low Poly vs High Poly

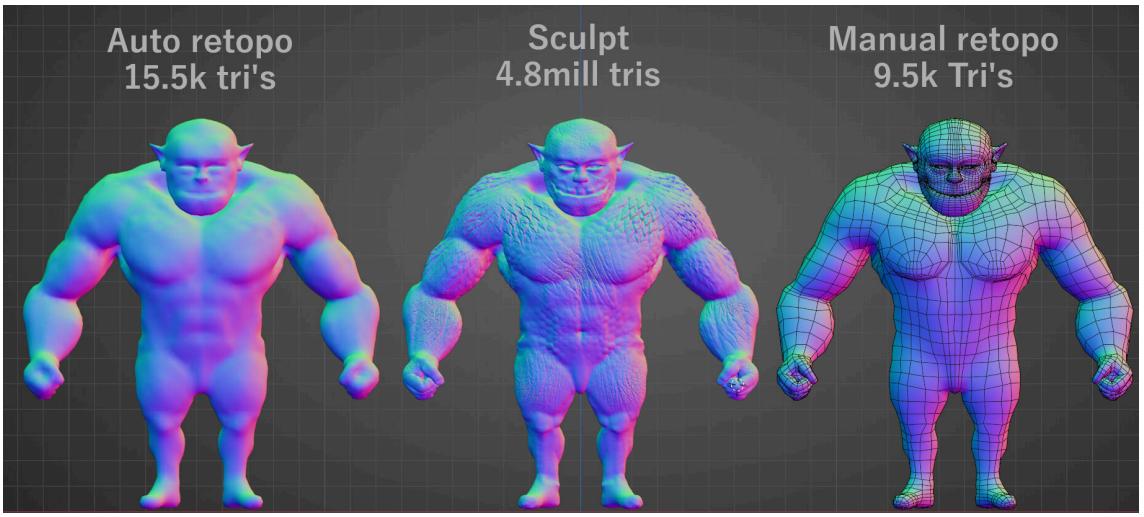


(Source: CuttingEdger.com)

There are two types of retopology, manual and automatic. Manual retopology, as its name indicates, is the process of restructuring the 3D model mesh by hand. This manual process allows a higher degree of precision and control on the newly created mesh, and it is ideal if the final objective is animation, as it will allow a better geometry

deformation. The main inconvenience is that it is a highly time-consuming task. On the other hand, automatic retopology uses computational algorithms and software to create the low-poly model. It is a fast process, but the control and details of the topology are lower. Nonetheless, it is useful for situations in which no time is available and speed is crucial.

Figure 2.1.3.2: Manual vs Automatic retopology



(Source: GameDev.tv)

The overall benefits of applying retopology to a 3D model are:

- Performance optimization.
- Improved mesh topology that is cleaner.
- Preservation of details without losing details.
- Easier UV mapping process.
- Reduction in the size of the 3D model's file.

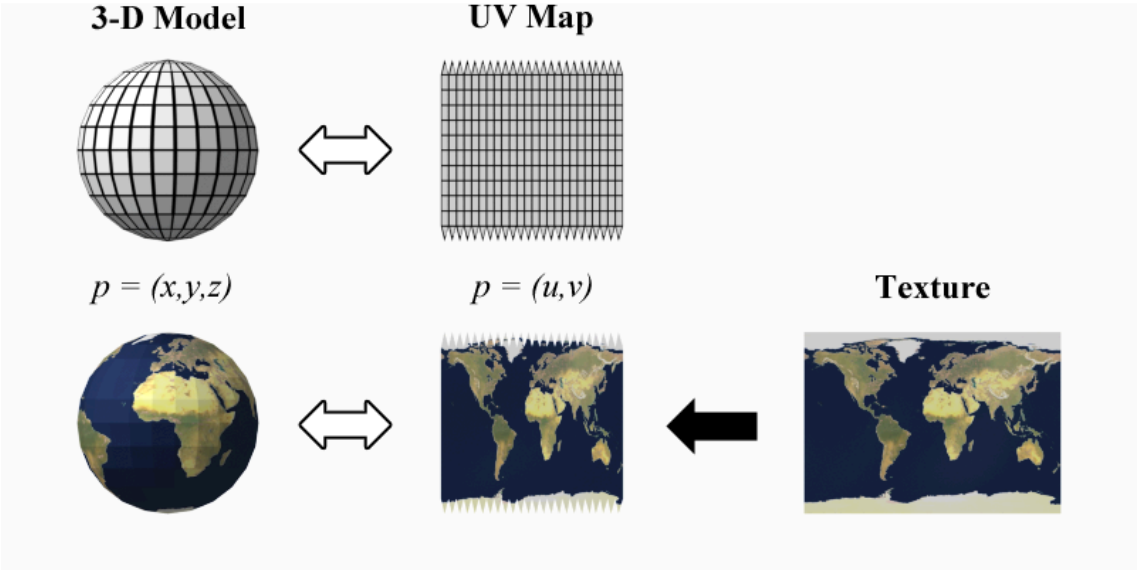
2.1.4. UV Mapping

UV mapping is the process of unfolding or unwrapping a 3D model surface in order to make it optimal for the texturing of it. It represents converting the three-dimensional surface of a mesh into a 2D projection. Its name, UV, represents each of the two-dimensional axes of the texture. "U" corresponds to the X-axis (horizontal) and "V" corresponds to the Y-axis (vertical).

In order to do the UV unwrapping, 3D models need to be unfolded by splitting the surface into different parts, thanks to the seams generated when cutting them. Once the

model has been cut correctly, these parts need to be unfolded and laid down in the UV space. These parts are known as UV shells or islands. When doing this, the texture pixels from the 2D layout correspond to the unfolded surfaces.

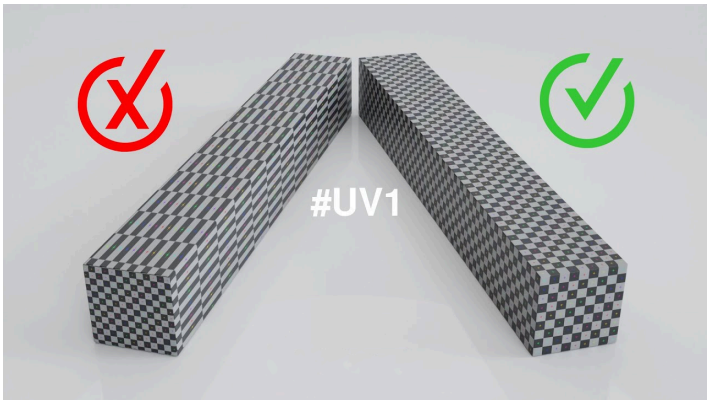
Figure 2.1.4.1: UV Mapping



(Source: 3dstudio.co)

Once the UVs have been mapped, there are some techniques to check if the unwrapped of the surface has been done correctly. Normally, a texture checker is used. This is just an image with usual squares from different colors, such as black and white. This texture is applied over the whole UV space, and the main objective is to check if the 3D object surface has the texture's squares deformed or not. If they are not deformed, it will indicate that the UV mapping has been done correctly. If not, the final texture will look stretched and deformed. That is the reason why UVs need to be done precisely.

Figure 2.1.4.2: Stretched vs Good UV mapping



(Source: CRNT Designers via Youtube)

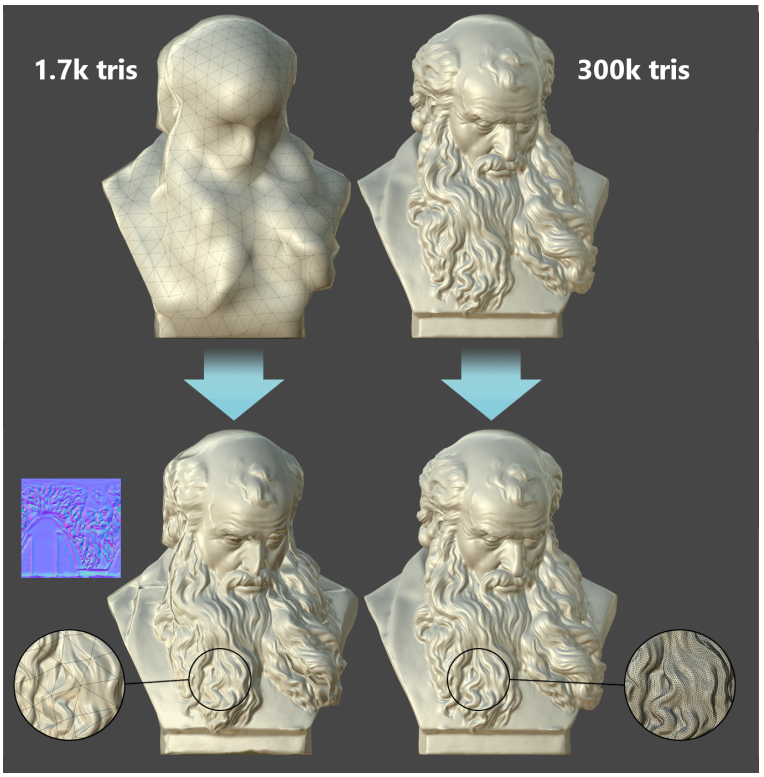
2.1.5. Baking

Baking is the name given to the process of generating 2D images called bitmaps that represent the rendered surface of the high-resolution mesh from a 3D object, so they can be applied to the low-resolution mesh surface. In other words, baking is the process of printing the details from a high-poly surface into a low-poly model surface.

These baked textures generated from the baking process can be applied to the low-poly model surface thanks to the coordinates of the UV maps, and thanks to this process, the low-resolution mesh surface can display the details by faking the colors computed due to the lighting influence on it.

This technique needs to be executed individually on every 3D object, and it can only be performed effectively if the low-poly mesh has been unwrapped into UV coordinates previously. This makes the overall process time-consuming and repetitive, as good baking needs a specific configuration for each object. However, it is still a really powerful method because of the large savings on rendering and loading times, as well as file sizes, especially in real-time applications such as video games.

Figure 2.1.5.1: Example of Bake



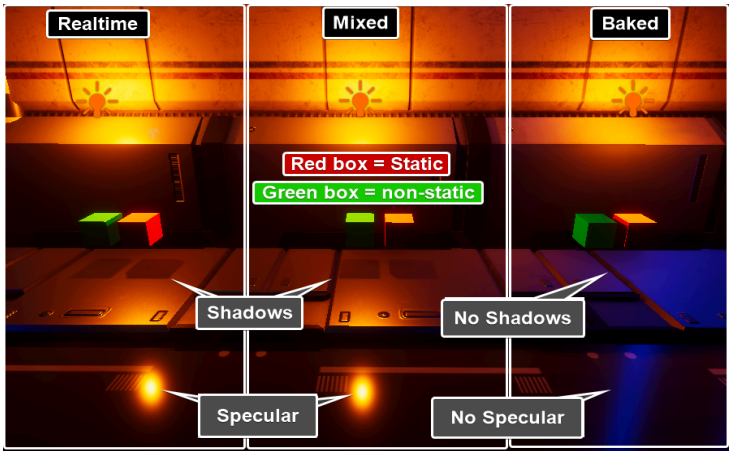
(Source: Total Baker via Unity Asset Store)

The usage of this technique is useful to precalculate surfaces on which lighting and shadows take place. Some examples are Ambient Occlusion or Soft Shadows, which are normally computed by real-time lighting within the game engines' rendering. However, if these are baked previously, there is no need for the rendering process to calculate them, and as a result, some computer performance will be saved up and there will be almost no differences. That is why in most real-time applications where there are scenarios with huge amounts of objects, their texture and light maps are baked.

The overall advantages and disadvantages of using the baking technique are the following.

- Advantages:
  - It reduces the rendering time extremely.
  - It reduces the number of polygons and the size of 3D models while preserving their initial aspect and details.
  - The texturing process can be easier as it is painted over the final result, thanks to the baked mesh maps information.
- Disadvantages:
  - The baked lights and objects from a real-time scenario can't be moved, as shadows are faked.
  - It is a highly time-consuming process, as the UV mapping and baking process can take several minutes or even hours for each object.
  - Sometimes, texture sizes need to be large, which can reduce the efficiency and benefits of the process.

Figure 2.1.5.2: Realtime vs Mixed vs Baked Lighting (Unity)



(Source: Vicent Taylor on *Medium*)

2.1.6. Texturing

Every 3D object has a flat, uncolored surface when it is created. This often gives the model a very simple look and shape. However, this is when 3D texturing takes place, covering the object with a set of texture layers that can completely transform it, giving it a new, different look. Texture layers can be from simple solid colors, repeating patterns, and unique images, to elaborated textures that simulate materials like stone, dirt, grass, etc. These textures contain different types of information, such as base color, glossiness, transparency, and more.

So, 3D texturing is the process of creating and applying texture images to any 3D model. In addition, it can also be considered within the texturing process the task of properly lighting the scenario where the object is placed, as well as adding final details to enhance the final rendering.

For the realization of this project, PBR materials will be used, with their respective maps. PBR stands for Physically Based Rendering, and it is the process of shading and rendering that tries to replicate how light interacts with the properties of a material. This can be done thanks to the different texture types that exist, as each one of them has specific properties. Below is the list of the PBR maps that will be used for the texturing process of the assets done for this project:

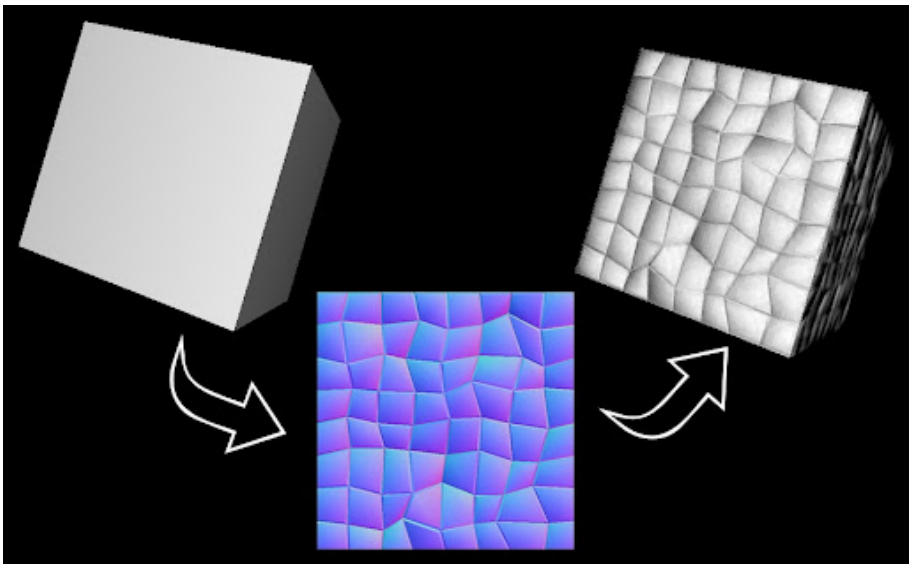
- **Albedo Map:** It is the essential map, and it is the one around which materials are built. It represents the flat base color and its illumination. There should be no shadows in it, as Albedo only represents the color of the texture. It is often confused with the Diffuse map, but they are not the same, as this last one tells how light is scattered across the texture surface of the model

Figure 2.1.6.1: Diffuse vs Albedo maps

(Source: A23D.co)

- **Normal Map:** The normal map gives information about the depth of the texture. It simulates how light hits the surface and creates a surface relief effect without modifying the model mesh. This map uses the RGB (Red, Green, Blue) color channels of images, and each one of them represents the X, Y, and Z axis components of the normal vector[7] orientation at each pixel.

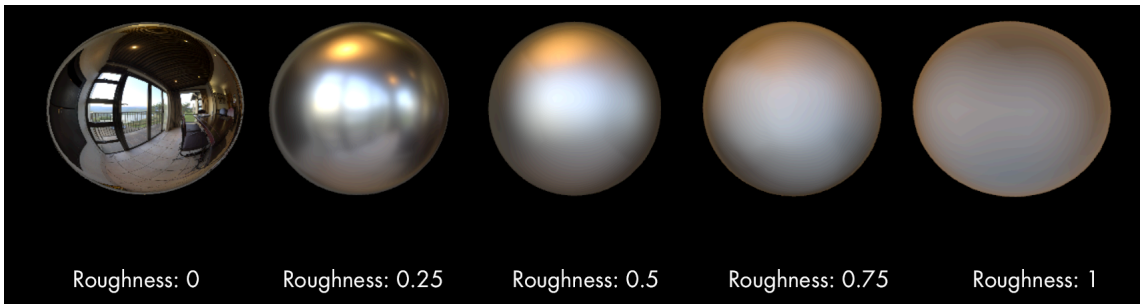
Figure 2.1.6.2: Normal Map application



(Source: AC3D Plugin via IndependentDeveloper.com)

- **Roughness Map:** It determines how rough or smooth a surface is, determining how light is reflected in the surface. This map is normally represented by a gray-scaled texture (white to black values only), and its value range goes from 0.0, where light is totally reflected (eg, Plastic), to 1.0, where light is dispersed throughout the surface of the material (eg, Rubber).

Figure 2.1.6.3: Roughness Map values

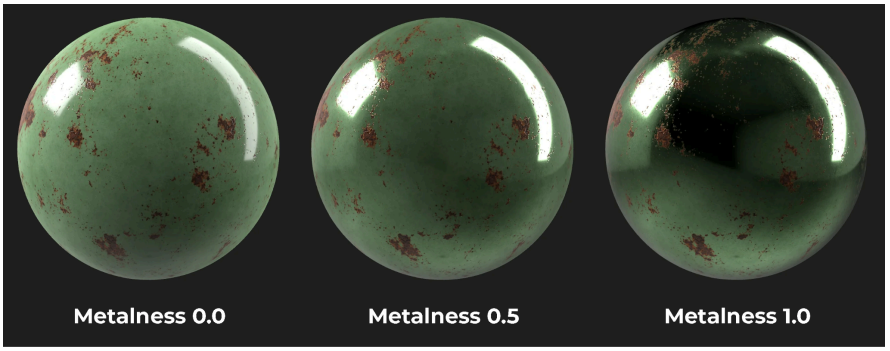


(Source: Aximmetry.com)

[7] Normal Vector: In the 3D modeling context, represents the vector that is perpendicular to the surface of a 3D object at a given point. It indicates the direction the surface is facing, which allows to calculate how light should reflect on the rendering process. (coohom.com/article/what-is-a-normal-in-3d-modeling)

- **Metalness Map:** It represents how much the model surface reflects what it is surrounded by. Again, it uses a grey-scaled texture, but it is recommended to use only the extreme colors (black or white) and fill in the in-between using the previously explained roughness map. Its value range goes from 0.0 to 1.0, and it uses the Albedo map to bridge this range. In the lowest value, the material seems to be plastic or ceramic, and the base color can be seen clearly, but in the highest value, the material simulates a real-world mirror, and the Albedo map color is almost indistinguishable.

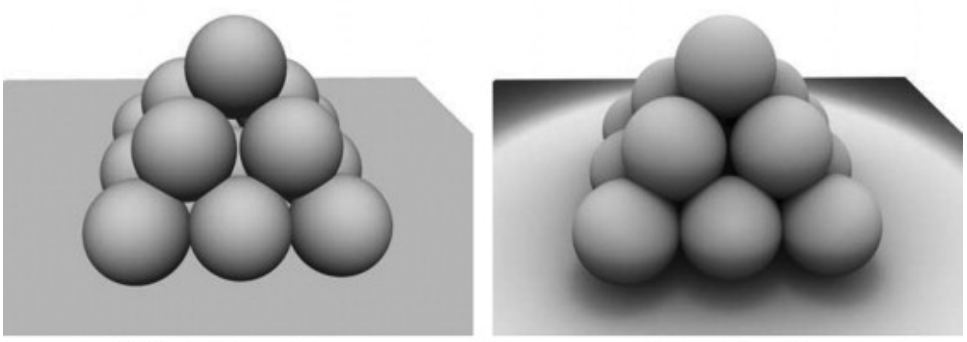
Figure 2.1.6.4: Metalness Map values



(Source: A23D.com)

- **Ambient Occlusion Map:** The main objective of the ambient occlusion map is to add shadows to the occluded parts of the objects, similar to how real-life sunlight affects them. When mixed with the Albedo map, it describes how light reacts with the 3D object. It is a grey-scaled map, with white meaning that it picks up light, and black meaning that it creates shadows. Its value ranges from 0.0 (dark) to 1.0 (light).

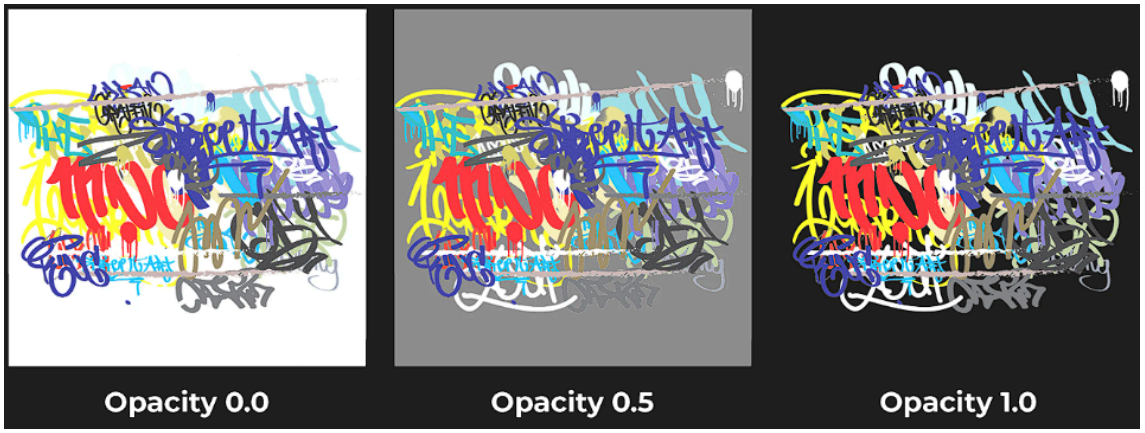
Figure 2.1.6.5: Ambient Occlusion Off/On



(Source: ScienceDirect.com)

- **Opacity Map:** The opacity map is in charge of making transparent the desired areas of a material. It is also a gray-scaled map, meaning white is opaque and black is transparent, and values range from 0.0 to 1.0. This map is commonly used when creating glass materials, tree foliage, and decals, as it allows for placing a detailed texture into a single plane and hides the excess thanks to transparency.

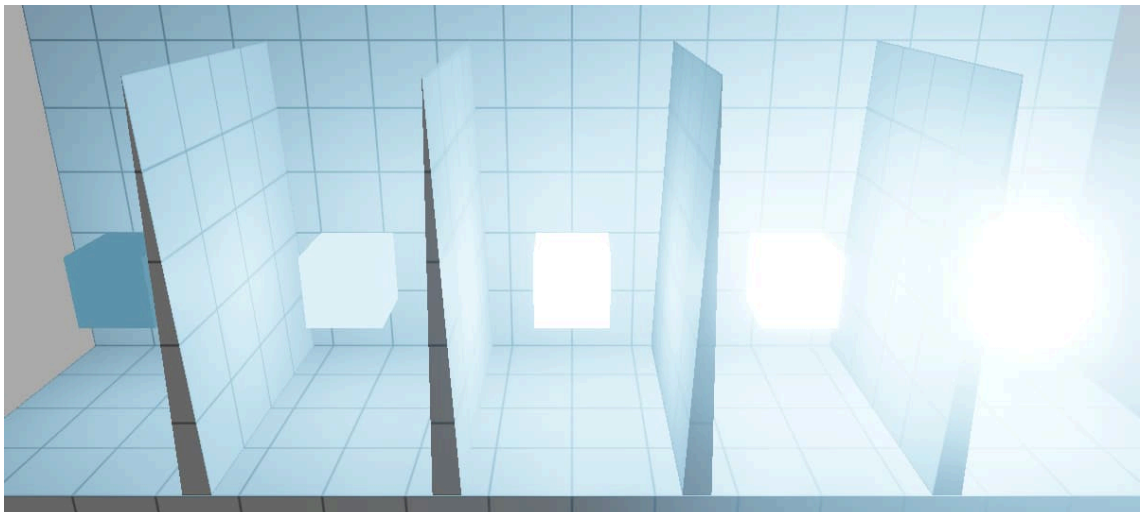
Figure 2.1.6.6: Opacity Map example



(Source: A23D.co)

- **Emissive Map:** The use of this map makes the surface of the material emit light from its interior. As a consequence, in a low-light scenario, it will be seen and stand out from the rest of the elements, but its intensity needs to be controlled, as it can ruin the overall appearance. It is an RGB map, similar to the Albedo one, but for radiating light.

Figure 2.1.6.7: Emissive Map with different intensities



(Source: glTF-transform.dev)

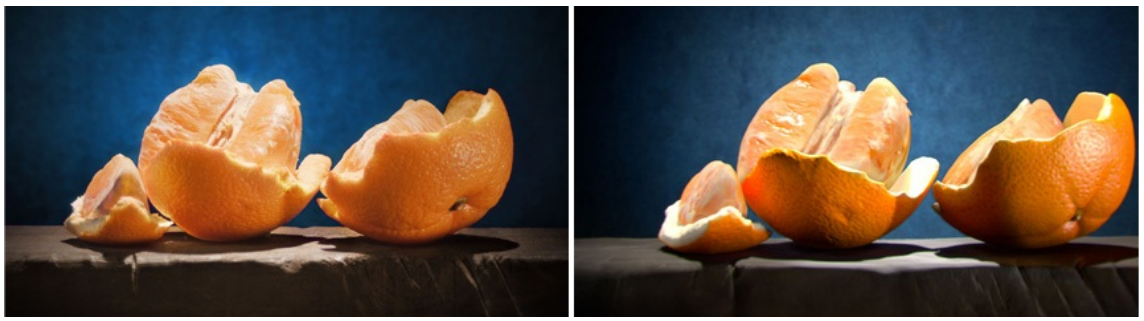
2.1.7. Rendering

3D rendering is a very commonly used process, and it can be found around you in many situations, from cartoon movies to nowadays commercials. It is frequently used due to its ability to preview a concept converted into a fake reality. Life without 3D visualization would not be the same as we know today. So, what is 3D rendering?

As Ricardo Ortiz explains in an article from *Chaos.com*[8] published on November 15, 2022, "*Put simply, 3D rendering is the process of using a computer to generate a 2D image from a digital three-dimensional scene.*" The concept is quite simple, generating a 2D image out of a 3D scenario. This image is known as a render.

The main usage of 3D rendering for the entertainment and media industries, such as video games, is to create sequences and animations for the creation of content. The evolution of it has been so big over the past few years that Ortiz mentions: *"In advertising, I would dare to say that 90% of automotive commercials are Computer Generated (CG) or even more. It accelerates processes, reducing its costs, and demanding better quality results."* Ortiz says that this has forced hardware technology to evolve, and because of it, Computer-Generated images are now the go-to option in these industries.

Figure 2.1.7.1: Real (left) vs Render (right)



(Source: WorldOfWaw.com)

To generate these 3D rendered images, there exist two main technologies: render engines and game engines. Render images use the ray tracing technique, while game engines use a technique known as rasterization, even though some engines mix both of them. As the final objective of this project is to produce assets used within the Unity game engine, it will only be explained the second technique.

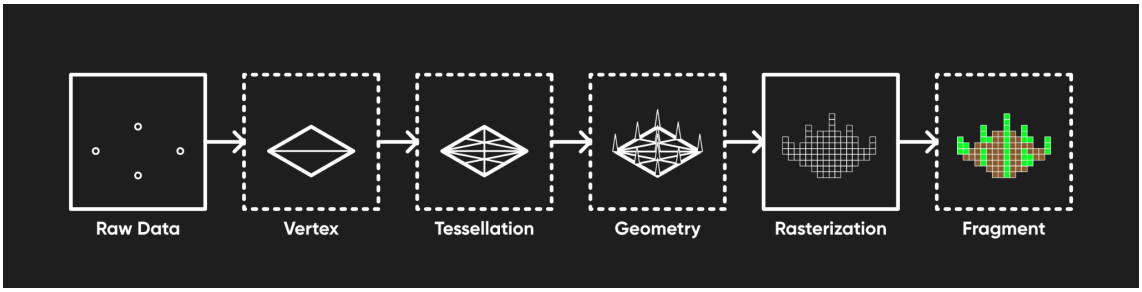
[8] Article mentioned: https://www.chaos.com/blog/what-is-3d-rendering-guide-to-3d-visualization

Rasterization is a methodology that can generate 3D-rendered images in an incredibly fast way. However, as a drawback, it may not be as realistic as pre-rendered images that use the ray tracing technique. It is often used in game engines, and the main advantage is that it is able to offer real-time experiences in which users can interact with the 3D scenario. A clear example of this is video games.

As Ortiz explains in the article about the rasterization technique, *"...objects on the screen are created from a mesh of virtual triangles, which create 3D models of objects. In this virtual mesh, the corners of each triangle, known as vertices, intersect with the vertices of other triangles with different shapes and sizes. Every single vertex provides specific information, and by gathering all this data, the shape of the object is created. Then, the computer converts the triangles of the 3D models into pixels on the 2D scree,n and we are presented with the final image."*

To be able for the computer to achieve this result, the main responsible is the GPU (Graphics Processing Unit), also known as the Graphics Card, designed for digital image processing and to accelerate computer graphics.

Figure 2.1.7.2: Example process of rendering pixels on the screen within Unity



(Source: danielilett.com)

2.2. Stylized Art Style

2.2.1. Overview

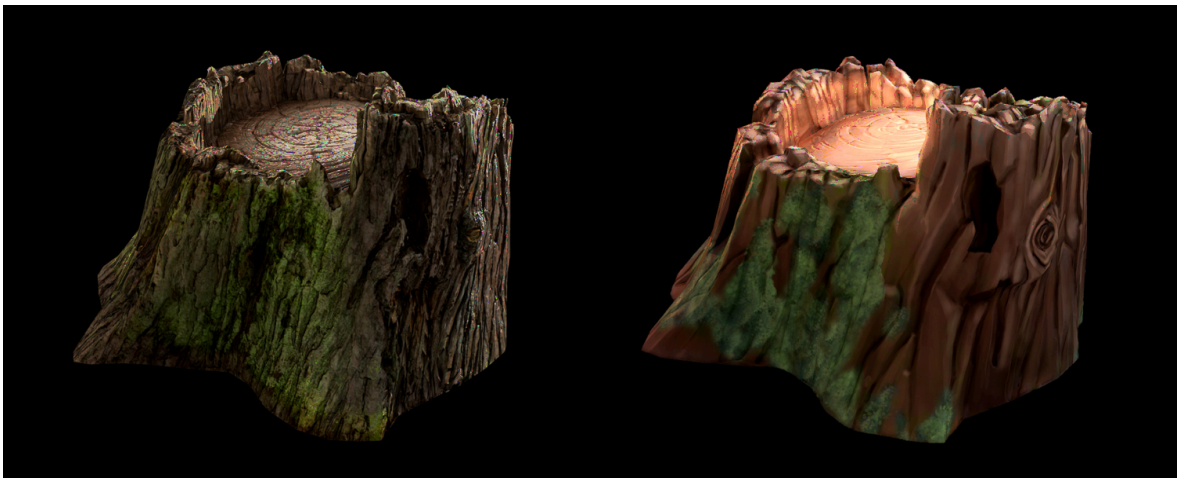
When talking about the different art styles from videogames, there exist two main ones: realism and stylization. These are the ones found in the most popular genres and with the biggest target audiences, and they are key to the development of a video game.

Different from other aspects of the game development general pipeline, the art style that is selected from the very beginning will influence the game feel as well as the aesthetic, and trying to change the art style halfway through the development will not be possible because of time and resource constraints. That is why choosing the right one from the start is essential.

For the development of this project, the Stylized Art style has been chosen. This kind of art uses exaggeration, non-realistic shapes, colors, and proportions as the main elements to create its unique style. It is an art that tries to find an appealing look by staying away from realism. When applied to video games, the Stylized art style often can include a mix of other techniques and visuals such as hand-drawing or cartoonish aesthetics. It usually uses minimalism, simplifying details, using outlines, and flat cell shading.

Even though it can be seen as simple, mastering this kind of art style is not an easy task. A basic understanding of how elements work in real-world situations is needed, but in addition to that, creativity is also necessary in order to apply the exaggeration traits that make this style so unique.

Figure 2.2.1.1: Realistic vs Stylized Art Style



(Source: 80Level)

2.2.2. Impacts of Stylized Art

The use of Stylized art in a video game can have a direct impact on the gameplay experience of the player because of its characteristics. Below are the most relevant elements.

2.2.2.1: Color Tones and Immersion

Stylized visuals normally tend to have a more inviting and unique color palette, mostly because of hand-painted details and imperfections given. Combining all the elements and their colors can engage the player with the atmosphere of the scenario they are in, even creating an emotional connection with these and a better immersion. But it can completely change if the chosen colors do not fit between them.

Figure 2.2.2.1.1: Color palette differences

(Source: UnrealEngine Reddit Community)

2.2.2.2: Character Design

Stylized art relies a lot on exaggeration. If it is applied during the design process of character (proportions, details, outline silhouettes, movements), it can help to make the player recognize and remember them easily, as well as connect. This can also be applied to the design of the gameplay mechanics and abilities of each character.

Figure 2.2.2.2.1: Stylized Character with exaggerated body proportions

(Source: 80Level)

As mentioned, the Character Design in the Stylized Art context often follows some characteristics depending on the role of the character itself, gameplay mechanics, abilities, etc. In the context of Hero Shooter games, such as *Overwatch*, characters will have a particular design depending on what is their function. They can be easily recognized thanks to their shape, silhouette, and color palette

Starting with DPS characters, when it comes to their design, they tend to be characterized by standard-sized characters with a similar proportion in their upper and lower body. Exaggeration is not present at all. However, the main reason for this is that the characters rely heavily on their in-game abilities, rather than their physical attributes.

Figure 2.2.2.2.2: Stylized Character Design: *Overwatch*'s DPS

(Source: DBLTAP)

Another type of character from Hero Shooters is the Healers. This type of character is known for its particular look and style. They tend to be slim characters. However, they are often complemented by a weapon or accessory that makes them recognizable, such as Mercy from *Overwatch* with its particular wings, which can be seen in the fourth place in the image below.

Figure 2.2.2.2.3: Stylized Character Design: *Overwatch*'s Healers

(Source: SportsKeeda)

Last, there are the Tank characters. These are known for being the ones who have the most exaggerated body proportions. They tend to be big and robust, as their main mission is to be able to deal damage received from the enemies while covering their teammates. That is the case in the Overwatch saga, where this type of character can be recognized in a very easy way.

Figure 2.2.2.2.4: Stylized Character Design: *Overwatch*'s Tanks

(Source: Overwatch Fandom Wiki)

With all these characteristics in mind, they will try to be applied during the Development phase of the 3D characters of this project, as 3 different characters will be created, each one of them taking one of the roles seen above: DPS, Healer, and Tank.

2.2.2.3: Accessibility

The contrast and legibility of stylized art can help players with visual impairment, such as color blindness, as the color design may not be the usual one. It also helps in the navigation and interpretation of environments and scenarios, because they tend to be simpler and characters' shapes are more distinguishable.

2.2.2.4: Art Lifespan

If it is paid attention to, there are some well-known games with stylized visuals that were released years and even decades ago, and their art style seems to be timeless. This is in part thanks to an emotional connection between the player and the game. However, games with realistic graphics tend to become outdated and age poorly because of technological progress in comparison with newer titles.

Figure 2.2.2.4.1: Realistic graphics: Before vs Now

(Source: gameranx via YouTube)

2.3. 3D Industry's Software

The first 3D modeling software that was commercialized was *Sketchpad*. Developed in 1960 by Ivan Sutherland, it is still considered the first drafting and 3D drawing software. It marked an important milestone at that time, as it opened the door to the commercialization of 3D models. The program allowed the user to draw, edit, and manipulate simple 3D shapes. It is also considered the first 3D rendering software that used object-oriented programming[9], and it was used as a foundation for later 3D modeling and CAD[10] software.

Figure 2.3.1: Ivan Sutherland using *Sketchpad* in 1963

(Source: ResearchGate.net)

Since then, multiple different software have been released, and at present, there is a wide variety of tools for each of the specific tasks that a project like this requires. Each of them has its advantages and drawbacks, due to the type and scale of the project, the necessity it covers, the level of expertise needed to master it, its price, etc.

[9] Object-oriented programming (OOP) is a computer programming model that organizes software design around data or object, rather than functions and logic. It is designed to use objects that interact with one another. (www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP)
[10] Computer-aided design (CAD) is a way to digitally crate 2D drawings and 3D models of future real-world products before they are manufactured. (www.ptc.com/en/technologies/cad)

2.3.1. Software Analysis

Below is a table analyzing the different software options from the market available for the realization of a project like this one, divided into each stage of the production.

Table 2.3.1.1: Software Analysis Table

Production Stage	Options Available	Advantages	Drawbacks
Concept Art	PureRef	- Easy to create moodboards and manipulate images. - Minimalistic design and available for all OS.	- Only supports images (no videos). - It can have a high memory usage, resulting in a bad performance on low-end PCs.
	Adobe Photoshop	- Industry standard tool. - Large number of tools & addons.	- Subscription-based software. - Resource-intensive.
3D Digital Sculpting	ZBrush	- It is the industry standard for sculpting because of its tools and capabilities. - It is able to handle 3D models with millions of polygons efficiently.	- The learning curve is steep and not ideal for new users. - It is more centered towards organic modelling only. - Its price is elevated.
	Blender	- Free and open-source software. - Integrated sculpting tools.	- Less specialization in sculpting tools compared with the rest of the options.
	Autodesk Mudbox	- Seamless integration with Autodesk software environment (Maya, 3ds Max). - Strong texture painting capabilities.	- Limited brush customization. - Few community support and resources.
3D Modelling	Autodesk Maya	- It is one of the industry standards for modeling and animation. - Several 3rd party tools can speed up the workflow.	- The interface & controls can be complex for new users. - It is expensive in both performance and economic ways if not have the educational license.
	Blender	- Free and open-source software. - Versatile modeling tools.	- The user interface may not be friendly for new users.
Retopology	Autodesk Maya	- "Quad-Draw" built-in tool, which allows precise control of manual retopology. - Integrated within the same Maya software. No changes needed.	- Time-consuming process. - No automation of the retopology process.

	TopoGun	- Dedicated software features centered towards retopology.	- Standalone application. - Additional cost.
	Blender	- Free and open-source software.	- Less refined tools and features for retopology.
UV Mapping	Autodesk Maya	- Integrated UV editor, which speeds up the workflow. - Automatic tools for unfolding and laying out UV islands.	- Some manual adjustments are almost always needed, as well as manual material assignment.
	RizomUV	- Advanced UV unwrapping algorithms.	- Standalone application. -Paid software.
Baking	Substance 3D Painter	- Integrated process within the same texturing program.	- Limited quality results compared with Marmoset Toolbag. - A powerful computer is needed.
	Marmoset Toolbag	- High-quality baking.	- Standalone application. -Paid software.
Texturing	Substance 3D Painter	- Interactive 3D texturing viewport. - Real-time PBR materials preview. - Tools like smart materials, advanced brushes, masks, etc.	- Hard to master PBR materials creation. - High subscription cost if not have an educational license.
Game Engine	Unity	- Large community support as well as an asset store. - Friendly and intuitive interface, with which prototyping can be done quickly.	- Performance demanding. - If commercializing a game, licenses need to be paid. - Few debugging tools.
	Unreal Engine	- High fidelity graphics. - Robust tools.	- Not centered towards mobile integration. - Steeper starting learning curve.
	Godot	- Open-source software. - Lightweight.	- Smaller community support. - Fewer features compared with the major game engines such as Unity or Unreal.

2.4. Asset Naming Convention

When developing a big project, where hundreds or even thousands of files are generated and stored in a workspace, it is crucial to establish a common naming convention. But what is it, and why is it so important that all members follow it?

An asset naming convention is a system of rules established by an organization, previously to the development process, which tells how assets should be named. By doing this, it is much easier for other members of the organization to search and find any asset in a fast way. It makes daily tasks more straightforward, and assets are easier to track, locate, and manage. Moreover, it reduces the risk of creating different assets with ambiguous or redundant names, such as a 3D model and a material with the same name. Additionally, a good naming convention for folder structure is also important, as assets are placed into categorized locations, and sometimes searching through folders might be the only way to find an asset.

For the realization of this project, it will be followed the recommended asset naming conventions published by the Epic Games developers community[11]. The main formula for the naming of assets to follow is:

[AssetTypePrefix]_ [AssetName]_ [Descriptor]_ [OptionalVariant]

- **AssetTypePrefix:** It identifies the type of asset.

Table 2.4.1: Asset Type Prefixes

Asset Type	Prefix
Animation Clip	Anim_
Animation Controller	AC_
Animation Sequence	AS_
Audio Mixer	AM_
Camera (Cinemachine)	CM_
Dynamic Mesh (moving object)	DM_
Light	(type)Light_

[11]Documentation:dev.epicgames.com/documentation/en-us/unreal-engine/recommended-asset-naming-conventions-in-unreal-engine-projects

Material	M_
Music	Music_
Particle System	PS_
Physics Material	PM_
Prefab	PF_
Rig	Rig_
Scene	Scene_
Shader	S_
Shader Graph	SG_
Skeletal Mesh	SK_
Sound Effect	SFX_
Static Mesh (Prop)	SM_
Texture	T_
Visual Effect	VFX_

- **AssetName:** It is the name of the asset itself.
- **Descriptor:** It gives additional information or context of the asset to identify which role it will have. Mainly used for Textures.

Table 2.4.2: Texture Type Suffixes

Texture Type	**Suffix**
Base Color (Albedo)	_BC
Normal Map	_NM
Roughness	_R
Metalness/Metallic	_M
Ambient Occlusion	_AO
Opacity	_O
Emissive	_E
Alpha	_A

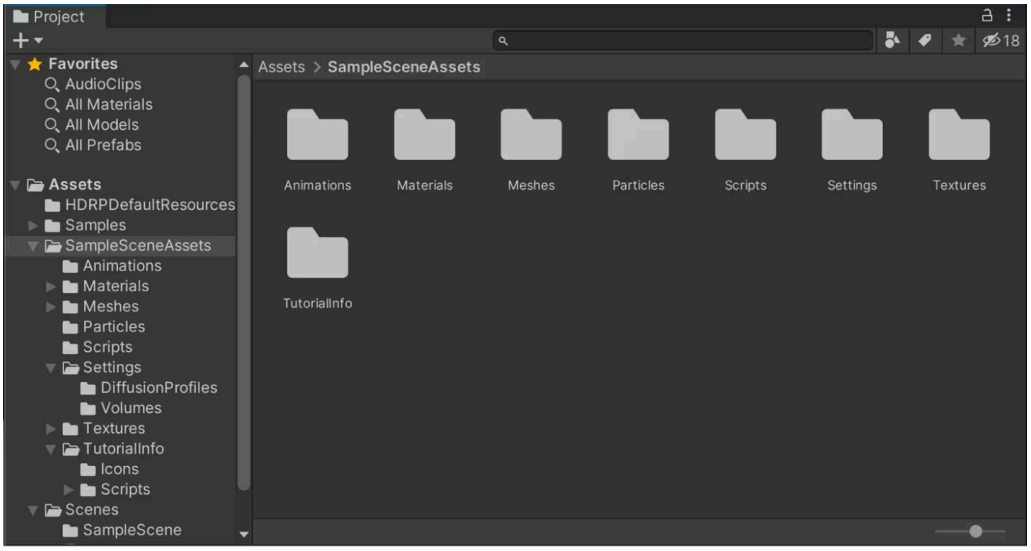
- **OptionalVariant:** It is an optional parameter used to differentiate between different versions or variants of the asset. For example: *"SM_Rock_01, SM_Rock_02, SM_Rock_03, ..."*

It is important to note that there are different cases in which assets are named. For the whole asset name (when applying the formula mentioned above), the Snake Capital Case is used, where words start in uppercase and are separated by an underscore. For example: "*PF_House"*. However, inside each word of the formula, the Pascal Case is used, where it is capitalized every word and all spaces are removed. For example: "*PS_CarSmoke"*.

Another important aspect to mention is the correct order in the naming of the assets depending on their traits. They should be named from a general to a specific classification. For example, if there is a tree that is considered a pine, it should be named "*SM_TreePine"* and not ~~*"SM_PineTree"*~~. By doing this, in case of having to select a tree to use, all of them will appear by just typing the parent word, and none of them will be excluded.

Lastly, as mentioned in the introduction of the section, a good folder organization is also needed. In this case, it will be followed by Unity's best practices recommendation for organizing a project.

Figure 2.4.1: Unity's Folder Structure Recommendation



(Source: Unity.com)

2.5. Market Study

The main objective of this project is to create 3D characters and an environment or scenario for them to be used in a video game prototype made with Unity 6.x. Based on the game's theme, mechanics, style desired, and more, the main games from the market are Overwatch 1/2 (2016/2023) and Rocket League (2015). Below is an analysis of the artistic elements that will be used as references for this project.

2.5.1. Overwatch 1/2 (2016/2023)

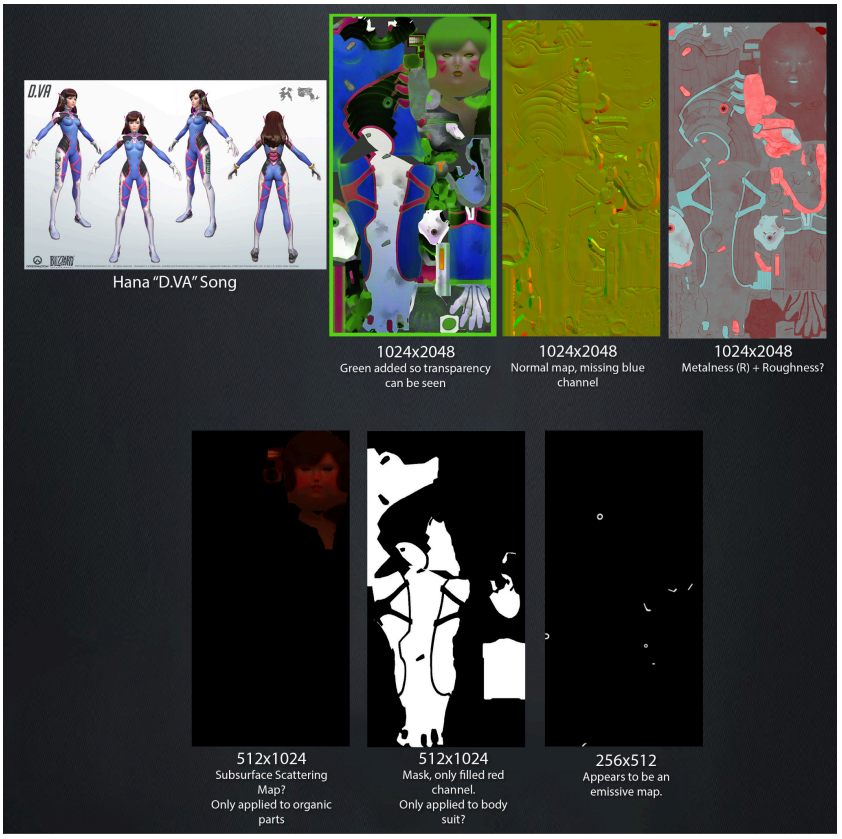
Overwatch is a video game saga with currently two game titles, the first one being released in 2016 and the second one in 2023. It is a multiplayer FPS[12] game developed and published by Blizzard Entertainment. It is considered one of the most famous hero shooters, in which two teams of five/six players are battling each other to reach the game mode objective. Each of these players has to choose a character (known as "Hero") from a large list. Within this characters list, there are three different types of heroes: the Tank, the DPS, and the Healer (explained in 1.4. Specific Objectives).

The main objective of this project is to try to recreate and get inspired by the style and the way Overwatch's characters are done. To do so, first, they need to be analyzed and see which techniques are being used. This study was already done in an *80Level* article[13] published in 2016 by *Kirill Tokarev*, in which the former co-founder of the indie studio Cardboard Keep, *Timothy Bermanseder*, studied the technical aspects of the beta version of Overwatch 1.

In this article, Bermanseder analyses the tank character called Hana Song, also known as "D.VA". It starts analyzing the texture maps and the techniques used to get the best-looking visuals without forgetting about performance. The first noticeable aspect is that the texture map sizes of the characters are not square. They are 1024x2048 px, even though they are still a power of two. Another element that stands out is how textures are blended using the different RGB color channels, meaning that each color channel of the image (red, green, blue) contains the information of a different texture map. Lastly, it can be seen that Normal Maps do not contain the blue channel information, but this can be because of the restrictions of using their own game engine.

[12] FPS (First Person Shooter): Video games genre in which the main characteristic is that player experiences the game through the eyes of the in-game character they are controlling, simulating its first person viewport. (https://eloking.com/glossary/general/first-person-shooter)
[13] Article (*80Level*): *https://80.lv/articles/overwatch-technical-overview/*

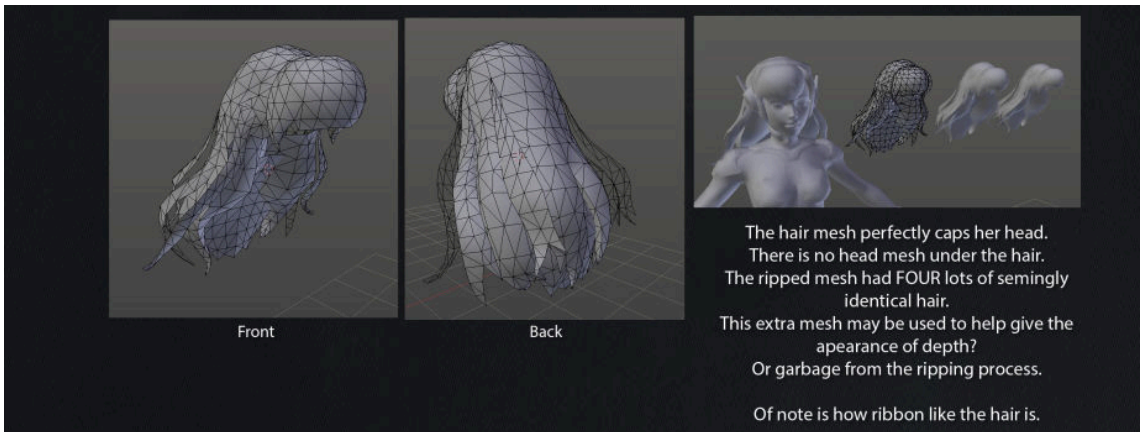
Figure 2.5.1.1: "D.VA" Texture Maps Analysis



(Source: 80Level)

Another element commented on in the article is that every character has some elements, such as accessories, and the mesh topology is targeted to have higher density in the most seen places (face, hands, weapons) than in the parts not that seen (legs, back, etc). Additionally, for those parts that are not seen, the mesh is deleted with the objective of winning performance.

Figure 2.5.1.2: "D.VA" Hair Mesh



(Source: 80Level)

Lastly, one of the most important elements when creating a character is the total number of polygons that the character contains. This value is crucial to maintain a correct performance and a good experience when playing the game on low-end computers. Below is a table of the polygon budget of each character from Overwatch (2016).

Table 2.5.1.1: Overwatch (2016) Polygon Budget

Character	Body Tri count	Weapon Tri C	Extras Tri count	Total Tri Count	Extra notes
Bastion	25,580	11,592	2,810 (Bird)	39,982	Body count does not include weapons
DVA	21,903 (Mech)	14,154	19,580 (Hana)	56,029	Mech count does not include weapons. In play Mech and Hana seem to be rendered at the same time
Genji	29,969	9,364	308 (Head cloth)	39,333	Head cloth is dynamic, double sided mesh
Hanzo	35,807	No weapon :(	2,050 (Fancy Arrow)	37,857	His bow was missing in the model rip :(
Junkrat	31,842	Missing	3,570 (Wheel)	35,412	Weapon missing
Lucio	30,355	Missing	1776 (Hair)	30,355	Weapon missing
McCree	37,222	12,118	80 (Cigar)	49,340	
Mei	45,228	Missing	2,310 (Robot)	47,538	Weapon missing
Mercy	30,736	12,079	224 (Glowy Wings)	43,039	Not the frame around the wings
Pharah	37,438	Missing		37,438	Weapon missing
Reaper	23,122	15,022		38,144	
Reinhardt	26,704	15,483		42,371	No projected shield, probably pretty low poly though
Roadhog	25,522	Missing		25,522	Weapon missing
Soldier76	20,898	10,708		31,606	
Symmetra	32,146	Missing		32,146	Weapon and turrets missing.
Torbjorn	35,908	6,804	2,360 (Facial hair)	42,712	No Rivit gun, no Turret
Tracer	35,006	Missing	2,404 (Hair)	35,006	Weapon missing
Widowmaker	33,498	14,194	2,209(Hair)	47,692	
Winston	25,919	Missing			Weapon missing
Zarya	36,662	16,951	1,907(Hair)	53,613	
Zenyatta	Missing	Missing	Missing	Missing	Zenyatta was unreadable
Average	31,556	12,588		40,270	

Note: These are rough, they were collected through ripped models hence some pieces are missing.

(Source: By Timothy Bermanseder on 80Level)

We can observe that the overall average of polygons per character in Overwatch is around 40.000 tris. Taking this into consideration, the objective to follow in this project is to try to get as close as possible to that number.

2.5.2. Rocket League (2015)

Another title that will be taken into consideration for the development of this project is Rocket League (2015), a soccer sports game that mixes cars with soccer mechanics. It was developed and published by Psyonix. The objective of the game is to hit a ball into the opponent's goal and score points by using the help of rocket power from vehicles.

One of the main inspirations and references that will be taken from this title are the scenarios where the game matches take place. These scenarios represent gigantic stadiums in which the football pitch is within a "cage", with large goals. Each of these stadiums or arenas is based in a unique and special location in the world.

Figure 2.5.2.1: Rocket League arenas

(Source: Rocket League Wiki)

The main objective in the creation of the scenario pitch for the game prototype developed in this project is to create a base symmetrical pitch, with its goals, and adding as well some obstacles and elements that the players can interact with when playing a match. Then, once the basic structure of the pitch is done, it will be time to set the elements of the environment surrounding the stadium, in order to create more immersion with the theme. This is an aspect that Rocket League does superbly, as it creates overall visuals that make the player feel inside the location and culture of the arena. Lighting, post-processing, and particles are fundamental elements to be used to achieve a similar result, as they can camouflage and boost the look of the whole scene without the need to create more assets.

3. Project Management

Management and planning are key elements in developing an idea for a final product of a certain quality. That is the reason why this project has created a plan to do so, thanks to the definition of realistic goals to be able to achieve them successfully.

3.1. Methodology and Tools

3.1.1. Gantt Diagram

The main project management methodology chosen to realize this project has been the *Gantt Diagram*[14]. It visually shows and represents the track of tasks in the timeline available. To create the chart, a general template[15] for Google Sheets was used, which was later personalized with the available time and stages of production, as well as other minor tweaks.

Figure 3.1.1.1: Gantt Chart Planning + Milestones

GANTT CHART PLANNING

PROJECT TITLE: HyperStrike
COMPANY NAME:
DATE: February 2025 - June 2025

February 2025
WEEK 1 | WEEK 2 | WEEK 3 | W
M TS W TH F | M TS W TH F | M TS W TH F | M TS

TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION (days)	% OF TASK COMPLETE
Pre-Production		**10/2/25**	**23/3/25**	**43**	
Delivery 1: Theory Framework Art	Adrià	10/2/25	23/3/25	43	15%
Delivery 1: Theory Framework Design	Joel	10/2/25	23/3/25	43	20%
Delivery 1: Theory Framework Code	Arnau	10/2/25	23/3/25	43	20%
Concept Discovery	Team	10/2/25	9/3/25	29	60%
Characters Design (Abilities)	Joel	10/3/25	16/3/25	6	33%
Level Design (Layout)	Joel	16/3/25	23/3/25	7	15%
Characters Concept Art + Moodboard	Adrià	10/3/25	16/3/25	6	33%
Environment Concept Art + Moodboard	Adrià	16/3/25	23/3/25	7	0%

In addition to creating the diagram, the project has been divided into four different stages of development: Pre-Production, Pre-Alpha, Alpha, and Beta. These stages represent the evolution of the game prototype and are aligned with the official deliveries of the thesis documentation from the University.

3.1.2. HacknPlan

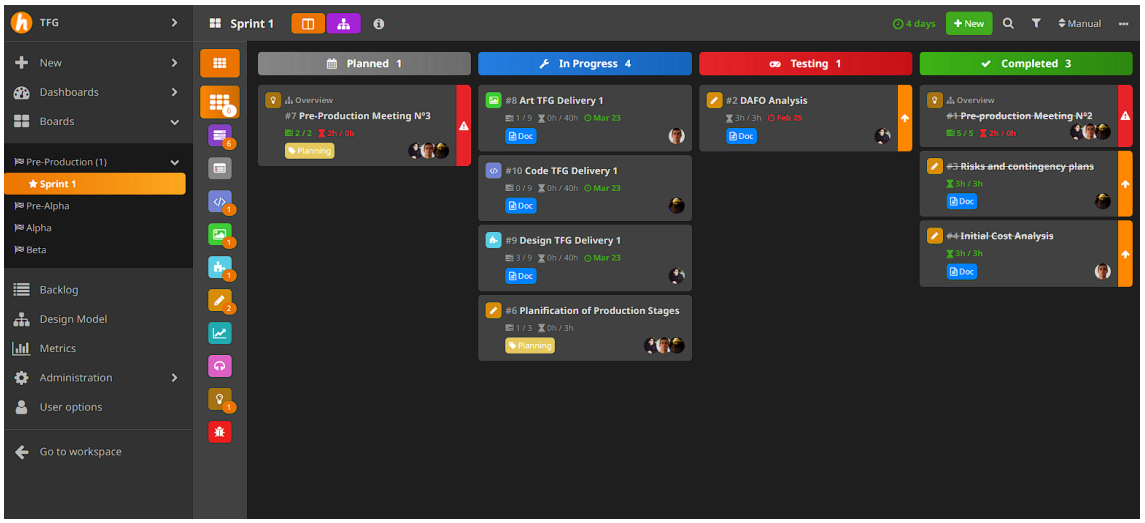
Another tool used to complement the management of the project has been *HacknPlan*, a tool that was created with the objective of managing Game Development projects using Agile[16] methodologies. It features Kanban boards for task management with different

[14] A *Gantt Diagram* is a visual project management tool that maps tasks, its durations, as well as dependencies over a period of time for an efficient planning and tracking. (https://www.gantt.com/)
[15] *Gantt* chart Google Slides template: GANTT CHART TEMPLATE
[16] The Agile methodology is a collection of management frameworks that break projects down into smaller phases. (wrike.com/project-management-guide/faq/)

disciplines and technical categories (design, art, code, marketing, etc.), advanced progress metrics for planning and organizing, and much more.

This tool has been useful when it comes to having a more specific track of all the tasks from the project, specifically those sub-tasks that can not be placed in the *Gantt* chart, as well as dealing with day-to-day tasks and having a visual overview of the weekly assignments. Furthermore, it has allowed the division of tasks in different boards depending on the current production stage, having a clear organization.

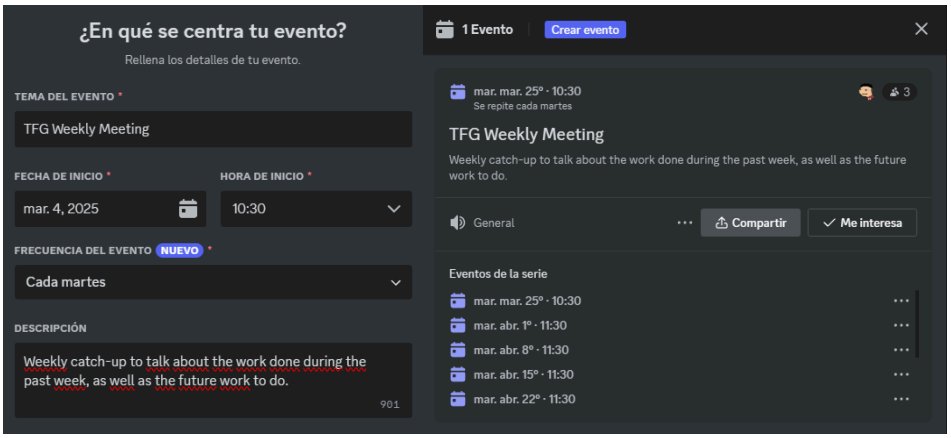
Figure 3.1.2.1: HacknPlan Kanban board



3.1.3. Discord

Discord has been the main communication channel during the project's development. It has been useful due to its versatility and ease of use, thanks to the possibility of creating infinite text channels for each project branch and voice chat channel, as well as weekly event meetings, useful to catch up with the team and work done.

Figure 3.1.3.1: Discord Weekly Meetings



3.2. SWOT Analysis

The SWOT analysis table is a very common tool used to break down the positives and negatives of an activity, from an internal and external point of view. It is a visual framework that allows the evaluation of the competitive position and also helps create a strategic plan for any business, evaluating what it is been doing well now, and what could be done better in the future.

In the following table, it has been investigated and listed the strengths, weaknesses, opportunities, and threats of the project, taking into consideration that the project was developed between February and June of 2025.

Table 3.2.1: SWOT Analysis

	Positives	**Negatives**
Internal Origin	**Strengths** ● Stylized Art style. ● Unique in its genre. ● Strong visual identity. ● Unity's multiplayer solutions.	**Weaknesses** ● Many FPS games in the market. ● It can become repetitive with only one game mode. ● Small development team. ● Non-balanced experience.
External Origin	**Opportunities** ● Unexplored genre, big potential market. ● A mix of the two most popular game genres. ● Streaming and content creation.	**Threats** ● Existing games could create similar game modes. ● Technical difficulties with servers. ● Multiplayer game market saturation. ● Low player engagement

3.3. Risk and Contingency Plan

To be able to develop the project in the best possible way and deliver a proper result, it is necessary to create a risk and contingency plan, as some problems might appear during the process, and it is important to know how to manage them.

Table 3.3.1: Risks & Contingencies

Risk	Solution
Running out of time for developing the project.	Implement a Gantt chart development approach using Hack'n Plan and Google Sheets with milestones and deadlines.
Team communication and coordination issues.	Set up a structured communication channel via a Discord server. Do regular meetings to check and align the team on progress.
Performance issues on lower-end PCs.	Optimize assets' textures and topology. Implement LOD (Level Of Detail) inside the Unity editor.
Bugs and exploits.	Do QA testing and track bugs and their fixing.
3D Character Modeling Issues.	Design and prototype the character concept before modeling it. Optimize character polycount for better performance.
3D Environment Issues	Balance visual quality and performance. Have clear artistic goals.

3.4. Project Validation Methodology

For the validation of this project, as it has been developed by 3 people, they will be the ones testing the first versions, within the Unity Game Engine. It will be evaluated if everything works as expected, if assets have a similar art style, if mechanics and game feel are the desired ones, etc. This validation process will take place during the last stages of the development: the Alpha and the Beta versions.

As mentioned before, there will be an initial evaluation without external users. It will mainly be done by the developers themselves. After that, once the project reaches its final stages, there will be some external playtesting sessions, in which all the feedback received will be used to analyze and further develop the Project Validation section of the project.

3.5. Initial Costs Analysis

Like any other commercial project, there is a large list of costs regarding its whole production. In this project, the costs have been calculated taking into account all the members of the team, so all costs will be grouped.

The level of developers has been considered Junior, and there are 3 people: 1 Game Designer, 1 3D Artist, and 1 Gameplay Programmer. The number of hours of work per week was 20, and as a consequence salary has been computed based on part-time dedication.

Table 3.5.1: Salaries Costs

SALARIES						
Job Role	**Annual Revenue (avg.)**	**Price/Hour**	**Weekly Work Hours**	**Weeks of Work**	**Total Work Hours**	**Total Cost**
Junior Game Designer	22.000€	11,22€	20	18	360	4.040,82€
Junior 3D Artist	20.000€	10,20€	20	18	360	3.673,47€
Junior Gameplay Programmer	24.000€	12,24€	20	18	360	4.408,16€
TOTAL						**12.122,45€**

The total cost of salaries for 3 developers during 5 months of development would be 12.122,45€. This estimation has been computed considering the average annual revenue[17] of each job role in Spain. Thanks to these numbers, it has been possible to calculate the price per hour, the total hours of work, and, as a consequence, the total cost.

Then there are the costs related to the Software used and its monthly fees. In the following table, it has been computed all the necessary costs regarding the different software used during the overall production. Most of the products have an educational or student license, and as the developers were eligible for these plans, the majority of the monthly fees were free of charge.

[17] Average annual revenues obtained from Glassdoor.com on February 20, 2025. See the salaries table for each salary consultancy link.

Table 3.5.2: Software Costs

SOFTWARE		
Name	**Monthly Fee**	**Total Cost (5 months)**
Unity Game Engine (Personal)	0,00€	0,00€
Autodesk Maya (Educational)	0,00€	0,00€
Adobe Photoshop (Students)	35,57€	177,85€
Adobe Substance 3D Suite (Education)	0,00€	0,00€
Maxon ZBrush (Monthly)	55,35€	276,75€
DaVinci Resolve	0,00€	0,00€
TOTAL		**454,60€**

It has also been taken into consideration all the hardware necessary to realize the project. In this case, it has been computed the cost per month based on the lifespan estimation[18] of each product for the total time of development (5 months).

Table 3.5.3: Hardware Costs

HARDWARE					
Product	**Price**	**Units**	**Lifespan Estimation (Months)**	**Monthly Amortization (per unit)**	**Cost (5 months)**
Computer	1.199,00€	3	72	16,65€	249,79€
Mouse	29,99€	3	24	1,25€	18,74€
Keyboard	49,99€	3	48	1,04€	15,62€
Monitor	149,00€	6	60	2,48€	74,50€
Digital Graphics Tablet	79,99€	1	36	2,22€	11,11€
Headphones	69,99€	3	36	1,94€	29,16€
TOTAL					**398,93€**

Last but not least, it has also considered some indirect costs like electricity consumption and Internet fees. In the electricity price[19], it has been taken the average price of January

[18] Lifespan estimation of office equipment (https://www.pdq.com/blog/equipment-lifecycle-management-guide)
[19] Average monthly electricity cost Spain on February 20, 2025 (https://www.costeenergia.es/historico/pvpc-anual)

2025 (0,1598 €/kWh). For the Internet monthly fee[20], it has been considered a 1 GB plan, with a price of 31€/month.

Table 3.5.4: Electricity Cost

ELECTRICITY					
Material	**Watts/h**	**Daily Work Hours**	**Price kWh (Avg.)**	**Daily Cost**	**Cost (5 months, 3 developers)**
Gaming Computer	300	4	0,1598	0,192 €	51,775 €
2 Monitors	44	4	0,1598	0,028 €	7,594 €
TOTAL					**59,369 €**

Table 3.5.5: Internet Cost

INTERNET		
Product	**Price (Monthly)**	**Cost (5 months)**
Internet Fee	31,00 €	155,00 €
TOTAL		**155,00 €**

Finally, after calculating all costs (direct and indirect), it can be said that the complete production of the project would have a price of 13.190,35€.

Table 3.5.5: Total Project Costs

TOTAL PROJECT COSTS	
Item	**Cost**
Salaries	12.122,45 €
Software	454,60 €
Hardware	398,93 €
Electricity	59,37 €
Internet	155,00 €
TOTAL	**13.190,35 €**

[20] Price from O2 company on February 20, 2025 (https://o2online.es/)

3.6. Environmental & Social Impact

The following section analyzes the environmental and social impact associated with the development process of the project. It examines the carbon footprint generated by all the activities done, and it also breaks them down into different categories. Thanks to this analysis, it has been able to identify which are the biggest threats to the environment in terms of CO2 emissions, as well as the potential social impact the final product can have on the end users.

Firstly, it has been computed the CO2 emissions in kilograms due to the usage of hardware equipment and its energy consumption. Note that in this case, it has only been considered those devices that are directly plugged into the power grid, as other devices like headsets or keyboards are powered by the computer.

Table 3.6.1: Hardware Usage CO2 Emissions

HARDWARE USAGE EMISSIONS							
Device	**Total Hours of Use (h)**	**Energy Consumption (W)**	**Energy Consumption (kWh)**	**Units**	**Total Energy Consumption (kWh)**	**Emission Factor (kg CO2/kWh) - Avg. Spain 2024**	**CO2 Emissions (kg)**
Computer	360	300	108	3	324	0,108	34,992
Monitor	360	40	14,4	6	86,4	0,108	9,3312
TOTAL							**44,3232**

For the hardware side, we also need to take into consideration the amortization carbon footprint due to its production, which normally represents the 80% of the total emissions of the devices. Here is the breakdown table:

Table 3.6.2: Hardware Amortization CO2 Emissions

HARDWARE AMORTIZATION EMISSIONS							
Device	**Production Carbon Footprint (kg CO2)**	**Lifespan (Years)**	**Working Days**	**Working Hours /Day**	**CO2 Emissions x Hour of Use (kg CO2)**	**Total Working Hours**	**CO2 Emissions (kg)**
Computer	778	6	90	4	0,360	360	129,67
Monitor	63	5	90	4	0,035	360	12,6
Keyboard	19,58	4	90	4	0,014	360	4,895

Mouse	5,86	2	90	4	0,008138888889	360	2,93
Digital Graphics Tablet	35[21]	3	90	4	0,032	360	11,67
Headphones	12,89	3	90	4	0,01193518519	360	4,296666667
Server	6000	5	60	24	0,83	1440	1200
TOTAL							**1366,06**

Then there are also the emissions related to the environment. This concept refers to the carbon footprint that the indirect elements from the work environment produce, such as lighting or air conditioning, due to energy consumption.

Table 3.6.3: Environment Usage CO2 Emissions

HOME OFFICE ENVIRONMENT USAGE EMISSIONS							
Device	**Total Hours of Use (h)**	**Energy Consumption (W)**	**Energy Consumption (kWh)**	**Units**	**Total Energy Consumption (kWh)**	**Emission Factor (kg CO2/kWh) - Avg. Spain 2024**	**CO2 Emissions (kg)**
LED Lighting	60	8	0,48	3	1,44	0,108	0,15552
A/C	20	3250	65	2	130	0,108	14,04
TOTAL							**14,20**

The usage of cloud storage services also produces emissions that are normally invisible. These are the network and cloud services that are used to host all the project's assets. They can be classified into different services, such as GitHub or Google Drive. Each one of the services might vary the quantities and the units in which they are measured, depending on what they are used for. The CO2 emission factor of each product has been calculated as a rough estimation. The following table creates an estimation of the kilograms of CO2 emitted as a consequence of the usage and subscription of the selected services for the development of the project:

[21] Estimation based on production carbon footprints from similar devices. As of March 11, 2025, there are no public official reports on the precise emission from these type of devices.

Table 3.6.4: Network & Cloud Services CO2 Emissions

NETWORK & CLOUD SERVICES USAGE EMISSIONS				
Service	**Quantity**	**CO2 Factor (kg CO2/unit)**	**Estimation Details**	**CO2 Emissions (kg)**
Github Commits	285 commits	0,0000024 kgCO2/commit	**Assumption:** A single commit transfers ~100 KB of data. **Energy Consumption:** ~0.06 kWh per GB translates to 0.06 kWh/GB × 0.0001 GB ≈ 0.000006 kWh per commit. **Emission Factor:** 0.4 kg CO2/kWh. **CO2/Commit:** 0.000006 kWh × 0.4 kg/kWh = 0.0000024 kg.	0,000684
Google Drive Storage	5 GB (annual storage)	0,08 kgCO2/GB/year	**Emission Factor:** Approximately 0.08 kg CO2/GB/year.	0,4
AI Queries	100 queries	0,00502 kgCO2/query	**Assumption:** A single query is estimated to produce around 0.00502 kg CO_2 (using online query energy estimates and model training).	0,502
Game Server (16GB)	288 kWh	0,4kgCO2/kWh	**Power:** Assumed constant power of 200 W (0.2 kW). **Total Hours:** 60 days × 24 h = 1440 h. **Energy Consumption:** 0.2 kW × 1440 h = 288 kWh. **Emission Factor:** 0.4 kg CO2/kWh.	115,2
TOTAL				**116,103**

On the other side, there is the usage by the end-users of the product developed in this project, and all the kilograms of CO2 emissions generated because of playing the game. In this case, as it is a fully digital product, players only need to use a computer (with its peripherals included) and a screen monitor. These emissions are produced due to the electricity consumption of the devices.

Table 3.6.5: Users CO2 Emissions

USERS USAGE EMISSIONS							
Activity	**Power (W)**	**Power (kW)**	**Time per User (h)**	**Units**	**Quantity**	**Emission Factor (kg CO2/kWh) - Avg. Spain 2024**	**CO2 Emissions (kg)**
Power Usage (PC)	250	0,25	1,5	100	37,5	0,108	4,05

Device Power Usage (Monitor)	40	0,04	1,5	100	6	0,108	0,648
TOTAL							**4,698**

Up until this point, it has been calculated an estimation of which is the carbon footprint this project produces during its development. However, it can also have other consequences. There are social impacts that the game can have on the users when playing the game, especially if it is played in long and repetitive sessions. These effects are described qualitatively, and they can be classified into positive and negative.

Table 3.6.6: Social Impact

Social Impact	**Positive Impacts**	**Community Building**	Games can be useful as a common way for players from diverse backgrounds to create connections and a sense of belonging between them. Cooperative/team gameplay and challenges can create friendships and collaborative relationships.
		Communication & Teamwork	Multiplayer games often require good communication and strategic plans. These interactions can boost interpersonal skills like communication.
		Stress Relief	For many people, games offer an engaging escape from everyday pressures. Positive social interactions within the game can provide emotional well-being and stress relief.
	Negative Impacts	**Social Isolation**	There is a risk that excessive play might lead to reduced face-to-face social interactions. This could potentially result in isolation if the online world begins to substitute for real life.
		Toxicity	Online competitive games sometimes generate aggressive behavior or toxic interactions between players. It can lead to a bigger and worse problem.

Lastly, once the project comes to an end, mainly because it has been called done or finished, some factors can make it generate a negative impact. Normally, it is because of unnecessary cloud hosting that creates a chain reaction of consequences that could be evitable. Here are some examples:

Table 3.6.7: End-of-Life Impact

Project's End-of-Life Negative Impacts	**Continuous Energy Consumption**	The centers where servers from GitHub are hosted need continuous power to be operational. Even though one single project might not use barely any resources, the sum of millions of repositories increases the energy consumption in the long term, and as a consequence, the carbon footprint, especially if energy isn't renewable.
	Utilization & Upgrades of Server's Hardware	The centers where the data is stored need to stay updated by replacing hardware to improve their performance. Keeping old projects that aren't being used anymore can contribute to the frequent update of the hardware, which generates electronic waste and potential recycling issues.
	Increase in Network Traffic & Maintenance	The fact of having an inactive project uploaded to the cloud services contributes to the network traffic infrastructure, which needs constant maintenance, energy, and all kinds of resources that contribute to a negative environmental impact.

After analyzing all the impact that the project produces, if a recount of the carbon footprint generated is done, these would be the total kilograms of CO2 emitted:

Table 3.6.7: Total CO2 Emissions

TOTAL CO2 EMISSIONS	
Activity	**Total CO2 (kg)**
Hardware Usage	44,32
Hardware Amortization	1366,06
Home Office Environment Usage	14,20
Network & Cloud Services Usage	116,10
User Usage	4,70
TOTAL	**1545,37**

It can be said that 1.545,37 kg is the amount of CO2 emitted to the environment for the overall production of this project. It is a big number, but how could we contribute to try to reverse this carbon footprint?

According to various studies made by *Encon*[22], an independent agency that supports companies through creative and innovative solutions in their transition to become more sustainable, it has been concluded that one single tree can offset between 21,77 and 31,5 kg CO2 annually. If we consider the lowest value in this range (21,77 kg CO2/tree/year), and calculate the equivalent for the 4 months of the development period of this project, we get that a tree would be able to offset 7,25 kg CO2. But if we go further, grab the total amount of CO2 emitted, and divide it by the offset capability of one tree, we obtain that 213 is the number of trees needed to offset the carbon footprint generated during this project.

$$Total\ Trees\ Needed\ = \frac{Total\ CO2\ Emissions}{Single\ Tree\ CO2\ Offset\ (4\ months)} = \frac{1.545{,}37\ kg\ CO2}{7{,}25\ kg\ CO2/tree} = 213\ trees$$

[22] Calculation of CO2 offsetting, by *Encon* (https://www.encon.eu/en/calculation-co2)

4. Methodology

As mentioned in the Project Management section, the project has been structured using the *Gantt Diagram*, and it has been divided into four different milestones or stages. Within each milestone, there is a list of tasks and subtasks to do. Thanks to the *Gantt* chart, a clear and structured vision of work versus time available can be created.

However, in addition to this technique, within each milestone, the available time has been divided into periods of one week. This has been possible thanks to the usage of the *Agile* methodology, also mentioned in the Project Management section. The value this methodology brings to the table is the possibility of structuring these one-week work periods into what is known as "Sprints", which allow planning, designing, developing, testing, and reviewing the work done during it in an efficient and structured manner. Sprints are normally complemented with weekly meetings with the rest of the team to catch up with the work that has been done during it, as well as organizing the next one. However, if the work assigned to each sprint is not finished on time, the following sprints will be affected, as previous work should be done before starting a new sprint. That is the reason why Gantt chart planning should be realistic and followed accordingly.

The different milestones of the project are Pre-Production, Pre-Alpha, Alpha, and Beta. These represent the division of work to be done in order to achieve all the desired initial objectives defined. Below is a sketch of the work in the production stages.

Figure 4.1: Project Milestones

The work done during each milestone of the project will be matched with the planning, but each task has an established timeline to be followed as strictly as possible. Any delay in one of the tasks could cause the rest of them to be delayed as well.

The different sections of the development have been classified by the expected workload and measured in days. Note that the project development is divided into the previously mentioned milestones, and within these timelines, the time is divided into 7-day Sprints.

Table 4.1: Major Tasks Workload

Task	Expected Workload	Milestone
Concept Discovery	14-21 days	Pre-Production
Character Sculpting (1 unit)	7-14 days	Pre-Alpha
Character Retopology + UV + Baking (1 unit)	2-5 days	Pre-Alpha
Character Texturing (1 unit)	4-7 days	Beta
Blocking of Scenario	5-12 days	Alpha
Modeling of Scenario Assets	7-14 days	Alpha
Texturing of Scenatio Assets	7-14 days	Beta
Integration of Assets within Unity	1-3 days	Beta
Creation of Materials, Illumination, Post-Processing, Shaders, etc.	7-14 days	Beta

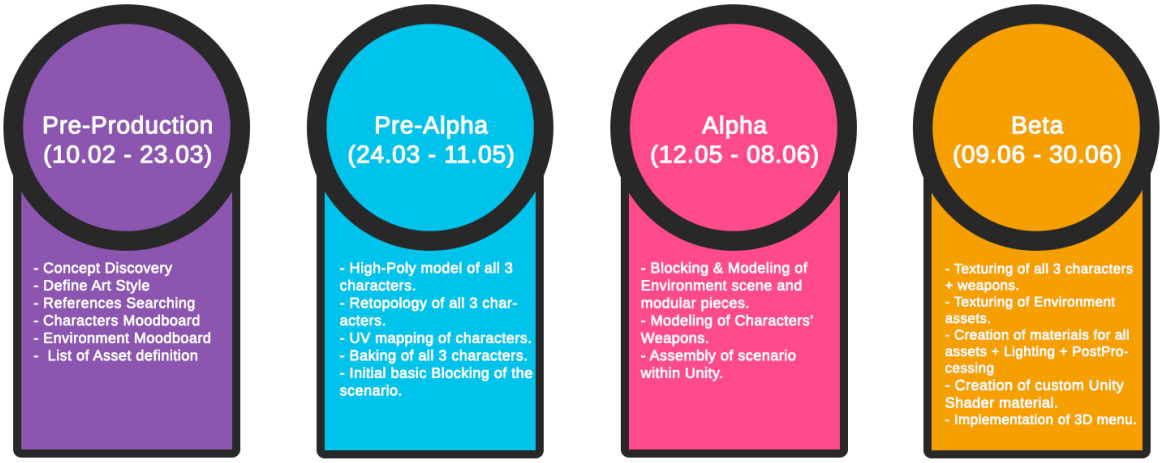
In case of delays in the development of the assets for the project, some sections could be penalized to be able to deliver the project within the deadline with the highest quality possible. These are mainly the secondary tasks that are not related directly to the creation of 3D Characters and an Environment, such as the creation of Materials, Illumination, Post-Processing, Shaders, and more inside the Unity project. The reason behind this is that the main objective of this Bachelor's Thesis project is to be able to investigate, learn, and apply the application and workflow pipeline of industry-standard 3D asset creation for characters and environments.

4.1: Reschedule during Production Phase

As in any other project, during the Pre-Production phase, an initial schedule of the work to be done is created. This schedule represents the ideal planning to be followed to achieve the best result within the available time. However, in projects like this one, the initial planning may change during the runtime of the Production phase due to a large number of reasons, both internal and external. In the case of *HyperStrike,* it hasn't been different.

Below is the initial planning based on each project milestone, which was first created at the start of the project (January 2025) and later updated during the Pre-Alpha production stage (May 2025).

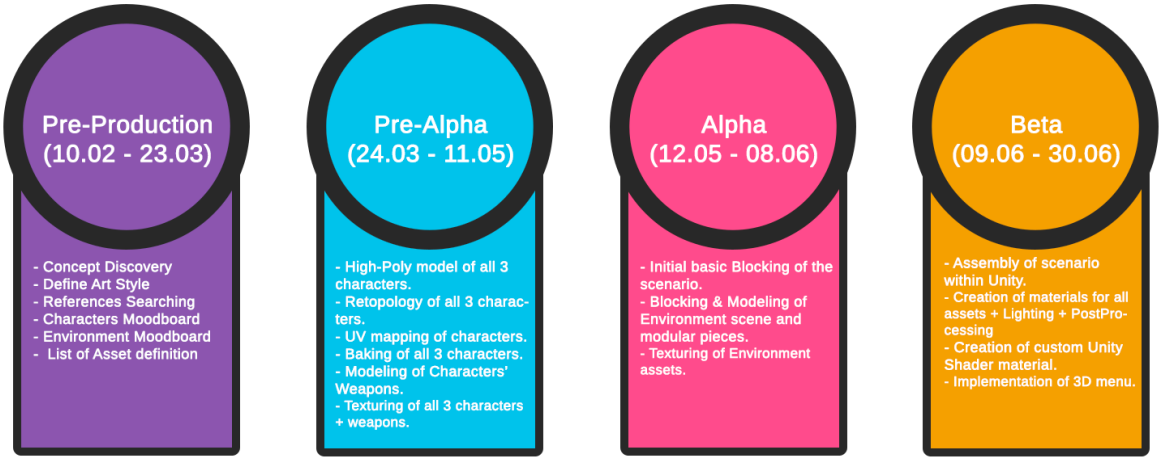
Figure 4.1.1: Project Milestones Updated on May 2025



If reviewing this schedule, it can be seen that the workflow of work was planned to start with the concept discovery phase, where the art style and the references of both characters and environment assets had to be defined. This phase was done and followed correctly, without any major change or delay.

Things started to change during the next phases. The initial plan was to start the production Pre-Alpha with modeling all the necessary 3D assets, for a later assembly of the environment in the Alpha stage, and finally the texturing process during the Beta. However, the planning changed when creating the 3D models of all the characters from the game. This whole process took longer than expected, and as a result, the workflow had to be modified, resulting in the following:

Figure 4.1.2: Project Milestones Final Reschedule



With the rescheduling of the work to be done, some tasks related to the Environment art of the scenario had to be re-planned, as time available was not enough. That is the reason why some tasks from it needed to rely on automatization of processes and 3rd party free royalty assets. By doing this, the project could be finished successfully as it was initially planned.

4.2. Software Chosen

Below is a table of the different software options from the market chosen for the realization of the project.

Table 4.2.1: Pipeline Software Choice

Production Stage	Description	Software Chosen	Advantages	Drawbacks
Concept Art	It is the initial stage of the production, where it needs to research, explore, and evaluate the different possible designs of the assets to be done. It is useful to determine the proportions, color, and traits.	PureRef	• Easy to create moodboards and manipulate images. • Minimalistic design and available for all OS.	• Only supports images (no videos). • It can have a high memory usage, resulting in a bad performance on low-end PCs.
3D Digital Sculpting	It is used to create the High-Poly version of Characters, Props, and other assets with a high detail level.	ZBRUSH	• It is the industry standard for sculpting because of its tools and capabilities. • It is able to handle 3D models with millions of polygons efficiently.	• The learning curve is steep and not ideal for new users. It is more centered •on organic modeling only. • Its price is elevated.
3D Modelling	It is the stage where blocking starts and meshes are manipulated, always having control of the polygons and vertices.	MAYA	• It is one of the industry standards for modeling and animation. • There are several 3rd party tools that can speed up the workflow.	• The interface & controls can result in complexity for new users. • It is expensive in both performance and economic terms if not have an educational license.
Retopology	It is time to reduce the number of polygons from the High-Poly sculpted model to a Low-Poly version, so it is performance-friendly.	MAYA	• "Quad-Draw" built-in tool, which allows precise control of manual retopology. • Integrated within the same Maya software. No changes are needed.	• Time-consuming process. • No automation of the retopology process.
UV Mapping	Once the model is finished, the 3D surface is cut and laid on the UV space as a 2D image.	MAYA	• Integrated UV editor, which speeds up the workflow. • Automatic tools for unfolding and laying out UV islands.	• Some manual adjustments are almost always needed, as well as manual material assignment.

Baking	Before texturing, the details of the High-Poly model are "stamped" into the Low-Poly model as a texture map so that polys are not increased.		• Integrated process within the same texturing program.	• Limited quality results compared with Marmoset Toolbag. • A powerful computer is needed.
Texturing	The last stage of the production of an asset. Its surface is painted to give the final look.		• Interactive 3D texturing viewport. • Real-time PBR materials preview. • Tools like smart materials, advanced brushes, masks, etc.	• Hard to master PBR materials creation. • High subscription cost if not have an educational license.
Game Engine	It is the stage where all assets are mixed, together with the creation of game mechanics, gameplay, and logic.		• Large community support as well as an asset store. • Friendly and intuitive interface, with which prototyping can be done quickly.	• Performance demanding. • If commercializing a game, licenses need to be paid. • Few debugging tools.

5. Project Development

As it has been described in section number 4 (Methodology), the development of the project followed different stages and milestones. The mission was to follow the scheduled Gantt Diagram chart during the weekly sprints, trying to meet the deadlines established as tied as possible.

Even with that, some tasks may require more time than others, and some problems and delays could appear during this process. That is why, in this section, the author tries to give a general overview of what the development process has been like, explaining what has been done, issues encountered, and more.

Note that the structure that has been followed to report this development process has been divided into the different tasks to be done. Each one of them has been broken down to explain the steps followed to achieve the final result.

5.1. Characters

As mentioned in the previous sections, the prototype *HyperStrike* will have 3 different characters. Each one is designed to represent each of the roles that Hero Shooters games have: DPS, Tank, and Healer. They have unique characteristics, abilities, and weapons that will work with the game mechanics and functionality within the game.

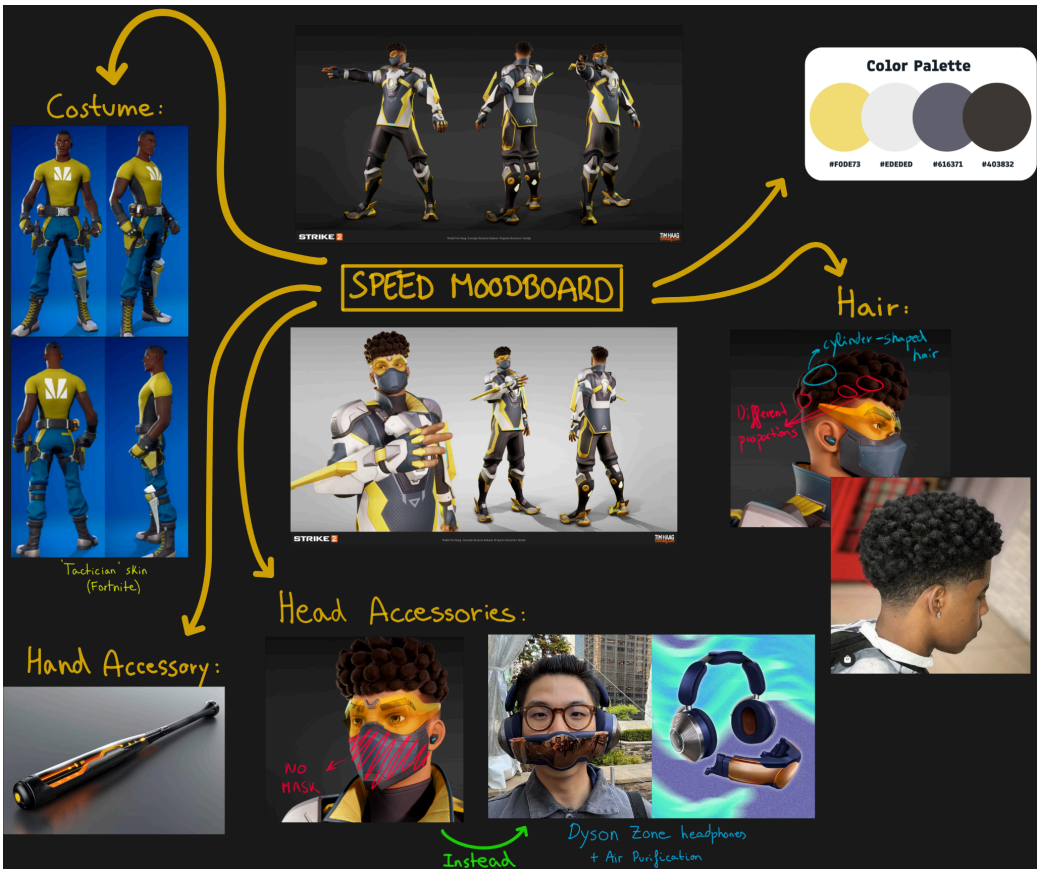
5.1.1: *Speed* (DPS)

5.1.1.1: Concept Art

Speed is a 21-year-old male who is a born competitor and self-assured. He is considered the Hero and 'Star of the Show'. He plays the role of 'DPS', being able to apply a high amount of damage while having a high degree of mobility.

From a very young age, *Speed* has always known his potential in athletics. From the United States, he started in sports playing baseball before going to athletics and football. He has always run in different sports clubs and gone to competitions where he has amassed an important number of awards and trophies. Football was another one of his passions, and playing as a winger, he reached his full potential for his childhood club. Below is a moodboard of the Character Design of *Speed*. It has been determined that all the elements that the character will be composed of, from its hair, clothing, and accessories to its color palette.

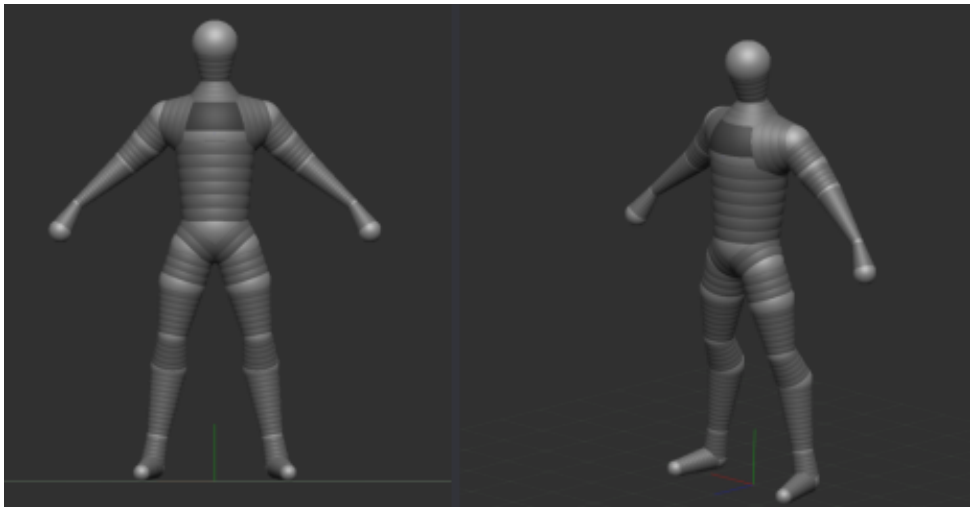
Figure 5.1.1.1.1: *Speed* Moodboard



5.1.1.2: High Poly Sculpting

When creating a 3D sculpted character, the best way to start from zero is to create an initial blockout with primitive-like shapes and then start giving more and more details to the character using different kinds of brushes that ZBrush offers. The initial blockout was created using the 'ZSpheres' tool. It allows to create the basic shapes and proportions by using 3D resizable spheres that are connected between them.

Figure 5.1.1.2.1: *Speed* ZSpheres Blockout



Once the basic shapes were created, the process of defining the body shapes and musculature began.

Figure 5.1.1.2.2: *Speed* Musculature

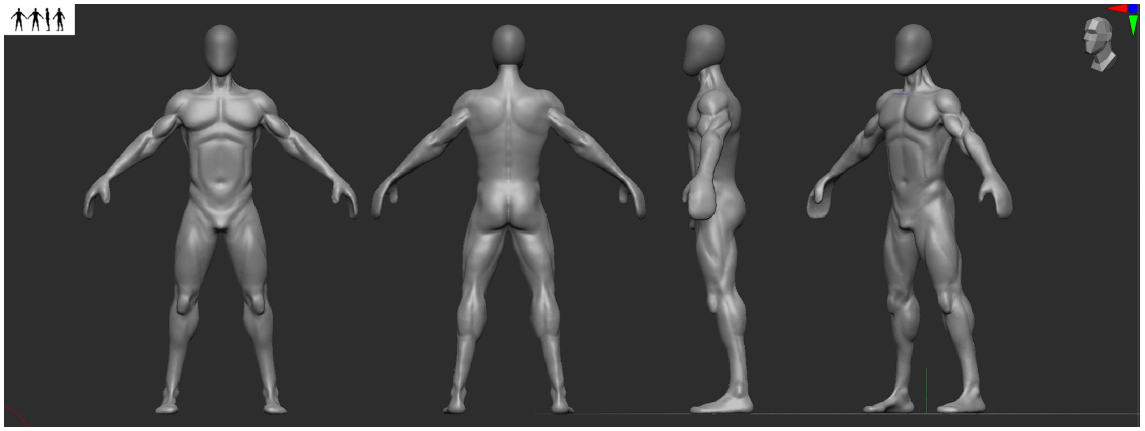
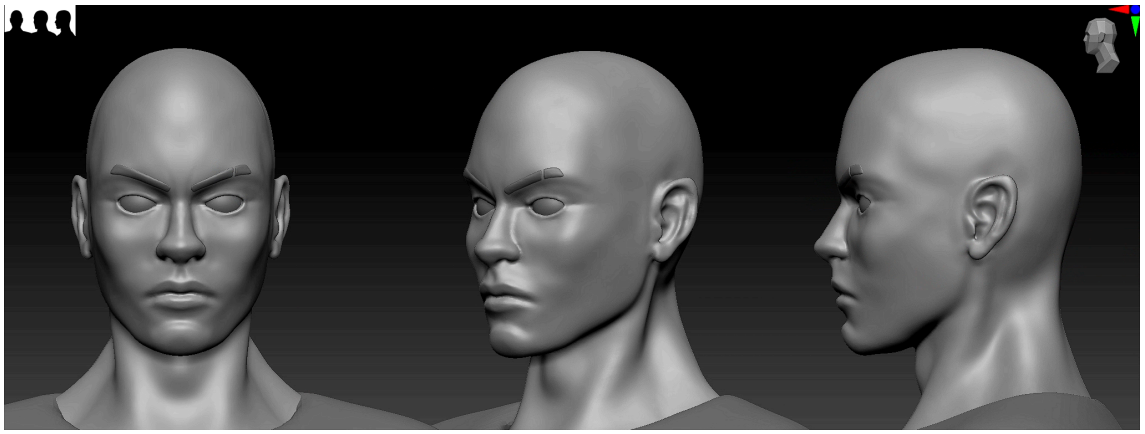


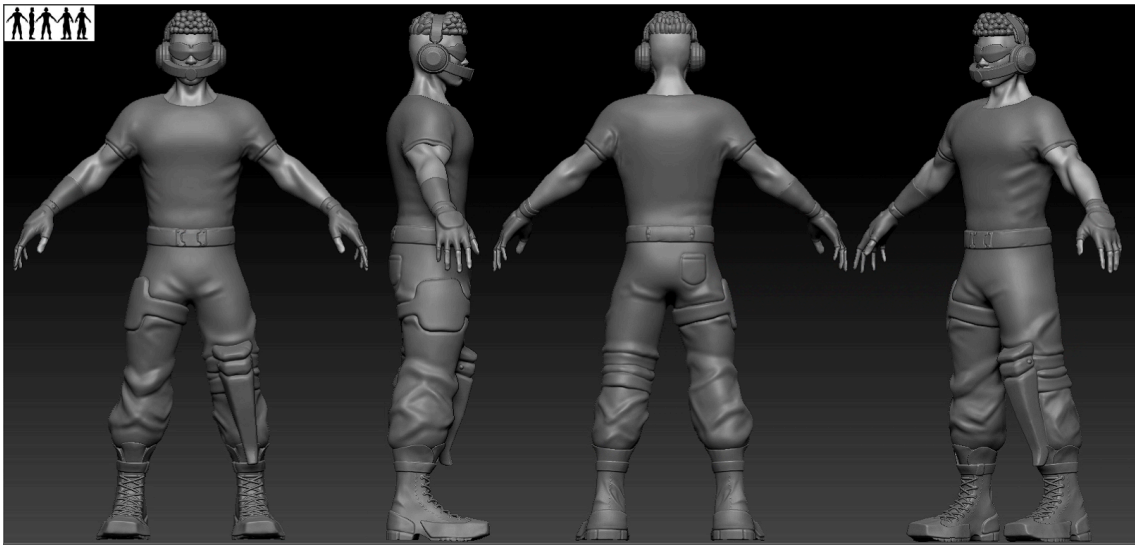
Figure 5.1.1.2.3: *Speed* Head & Face



Then it is time for the clothing of the character, together with the accessories. The main tool used for the creation of clothes is to Mask the desired shape onto the body and later Extract the masked shape, resulting in a new separate Subtool (object). After that, each piece is given details to enhance its look and match with the design of the character concept, resulting in the final version of the High Poly 3D model.

To be able to achieve a good-looking result, a high density of polygons is needed, so the mesh can be modified with precision and as needed. The total number of polygons can go up to some millions. However, the ZBrush software is capable of running these kinds of meshes thanks to its algorithms and tools, as long as your hardware is capable too.

Figure 5.1.1.2.4: *Speed* High Poly Model



5.1.1.3: Retopology

Once the final look of the 3D character has been achieved, the resulting mesh created in ZBrush has an enormous number of polygons (known as a High Poly object). However, the main objective of this character is to take it into a video game. Taking that into account, the number of polygons that a standard game engine can handle in an optimal way is much lower. That is why the Retopology process needs to be executed.

As explained in section 2.1.3 (Retopology: High Poly to Low Poly), there exist different types of Retopology: Manual and Automatic. For this project, a combination of both has been used. The main reason is that manual retopology is a highly time-consuming task, but it allows to have better control of topology. Taking this into consideration, the manual retopology has been done to the Head and Body Skin parts, while the rest of the elements have been done with Automatic retopology.

In order to do the manual retopology, the software that was used is Autodesk Maya. Maya offers a powerful tool called 'Quad-Draw'. What this tool does is enable the user to create polygons by joining vertices together into a desired surface. This surface needs to be the High Poly model of the character, and it has to be enabled as 'Alive', meaning that when creating polygons with the 'Quad-Draw' tool, they will automatically adapt to the surface of the character, resulting in a Low Poly version of the character.

Figure 5.1.1.3.1: *Speed* Head Manual Retopology

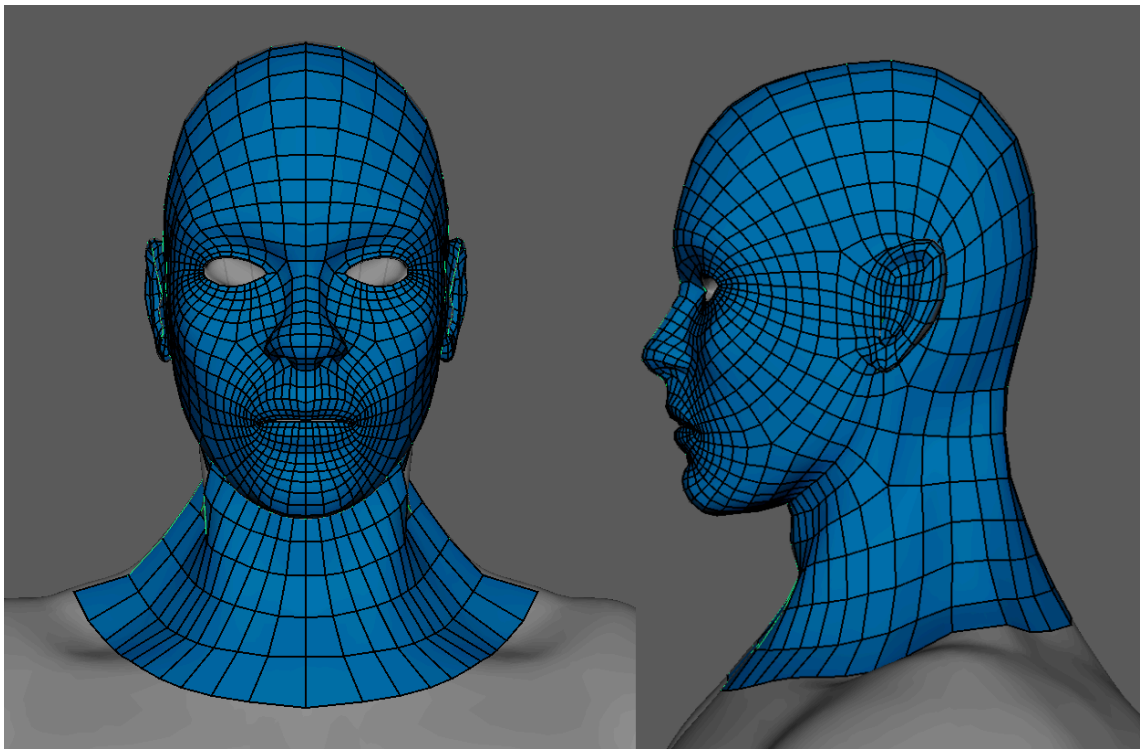
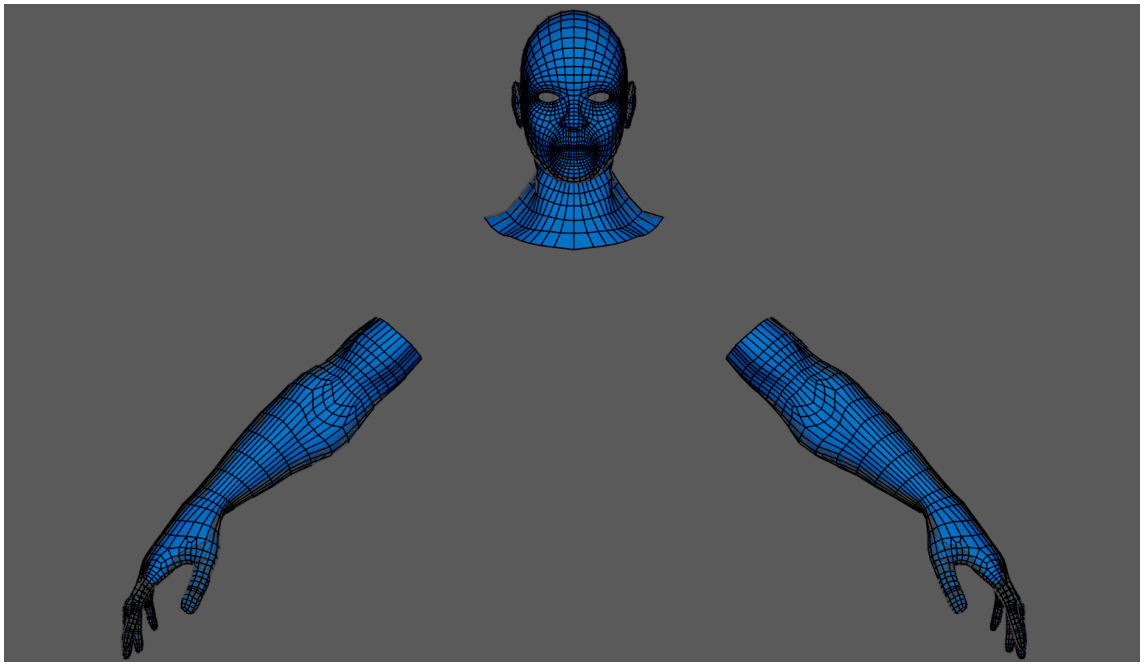


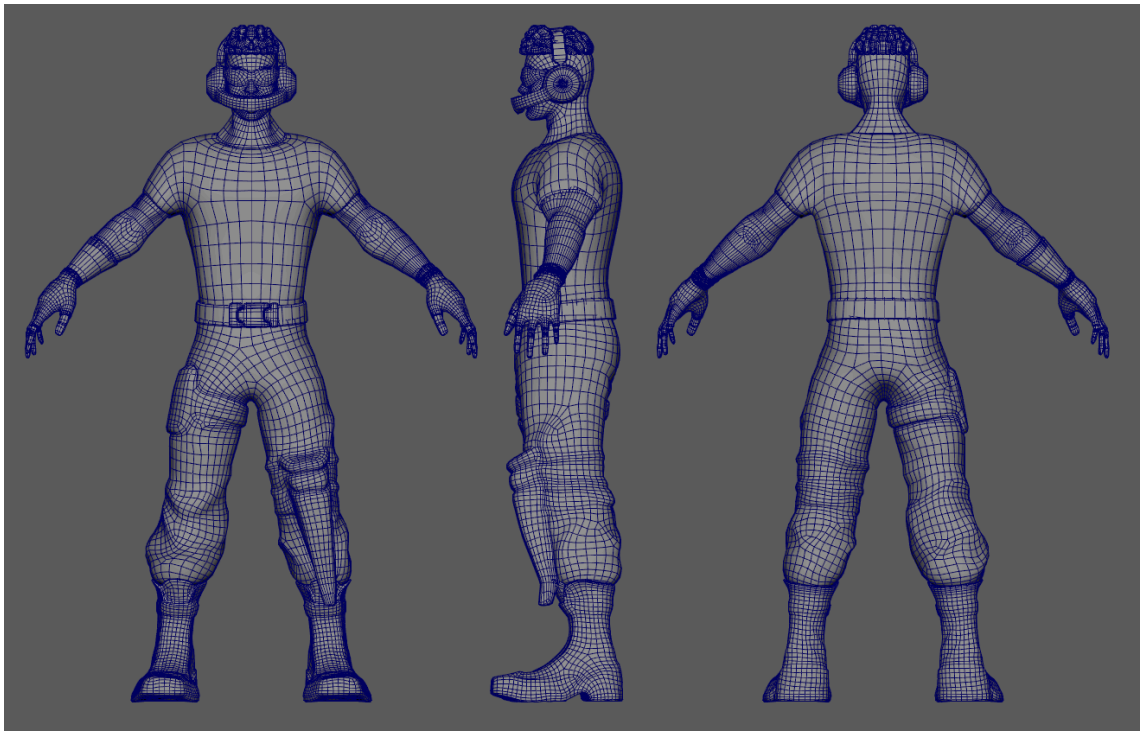
Figure 5.1.1.3.2: *Speed* Body Manual Retopology



Once the Head and Body Skin manual retopology was done, it was time for the retopology of the rest of the elements. To do so, automatic retopology was used. The main tool used was 'ZRemesher' from ZBrush software. It allows to re-construct the

mesh (re-mesh) of any object, with the option of reducing the number of polygons to a desired objective. Even though the final result may not be the best one possible, 'ZRemesher' is a very useful tool for reducing the workload in tight production timelines such as this project. Below is the result of the Low Poly version of the *Speed* 3D character model, which is considered to be "Game-Ready[23]", as it has 41.687 tris.

Figure 5.1.1.3.3: *Speed* Full Character Retopology



The last step of the retopology process consists of using a very useful tool that Maya offers. It is very likely that after creating this version of the character that has a very reduced amount of polygons in comparison to the High Poly one, it might look peculiar, as edges and vertices will look crispy. That is because of Normal Vectors from each face are not facing the correct direction. To fix this problem, all the character parts must be selected and apply the 'Soften Edge' tool, and as a consequence, the surface of the character will seem to be smoothed, but at the same time, keeping the same topology.

5.1.1.4: UV Mapping

Once the retopology has been done and there is a Low Poly version of the character, the UV Mapping of it needs to be done, as it will prepare the model to be textured later on.

[23] Game-Ready: 3D model created by being very aware of the performance impact of polygons and textures and use them as efficiently as possible. (https://blender.stackexchange.com)

During this stage, it has to be decided how many materials the model will have. Each material represents a UV Space, and the unwrapped and unfolded UV shells from the selected elements will be placed in it.

In the case of *Speed*, there have been used 5 different materials: Accessories, Body, Hair, Clothes, and Eyes.

Figure 5.1.1.4.1: *Speed* UV Mapping

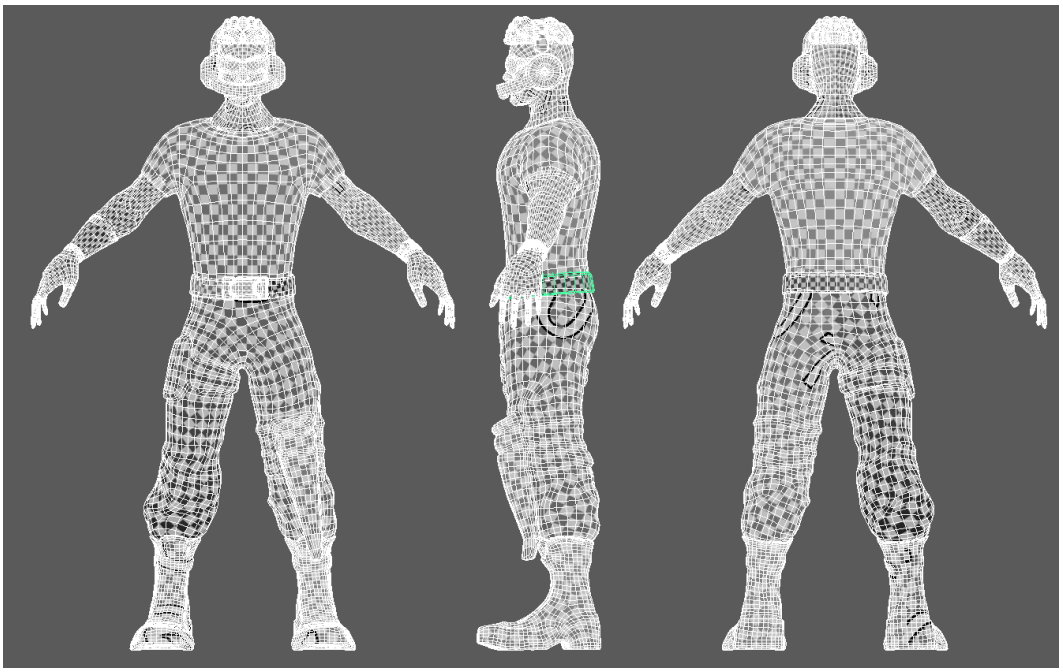
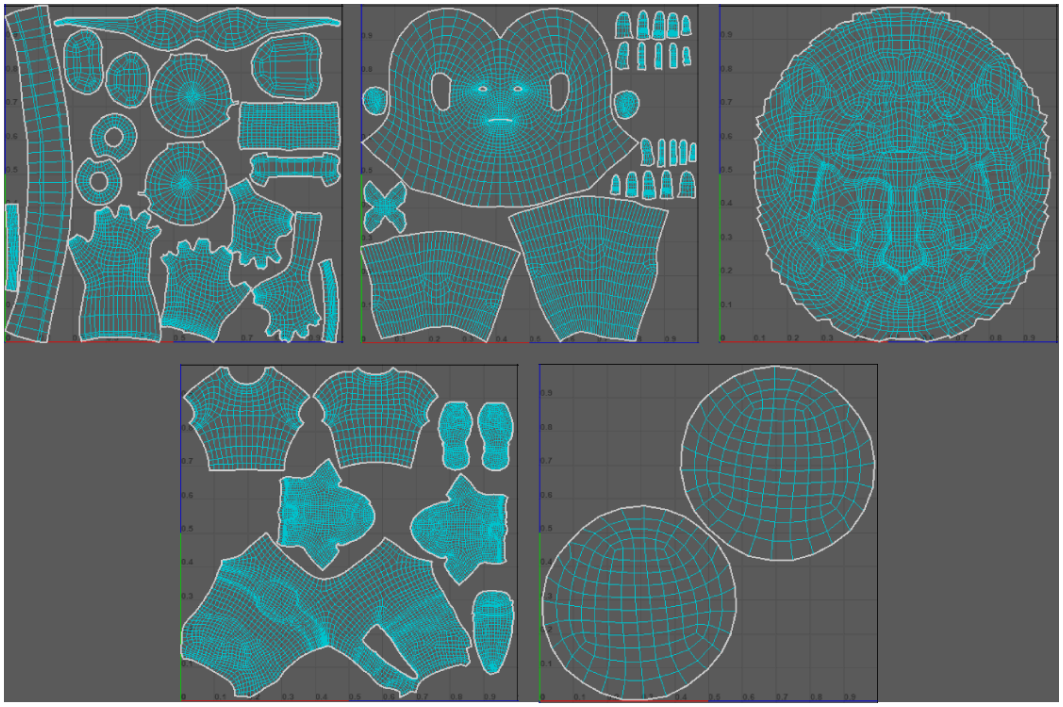


Figure 5.1.1.4.2: *Speed* UV Layout for each material



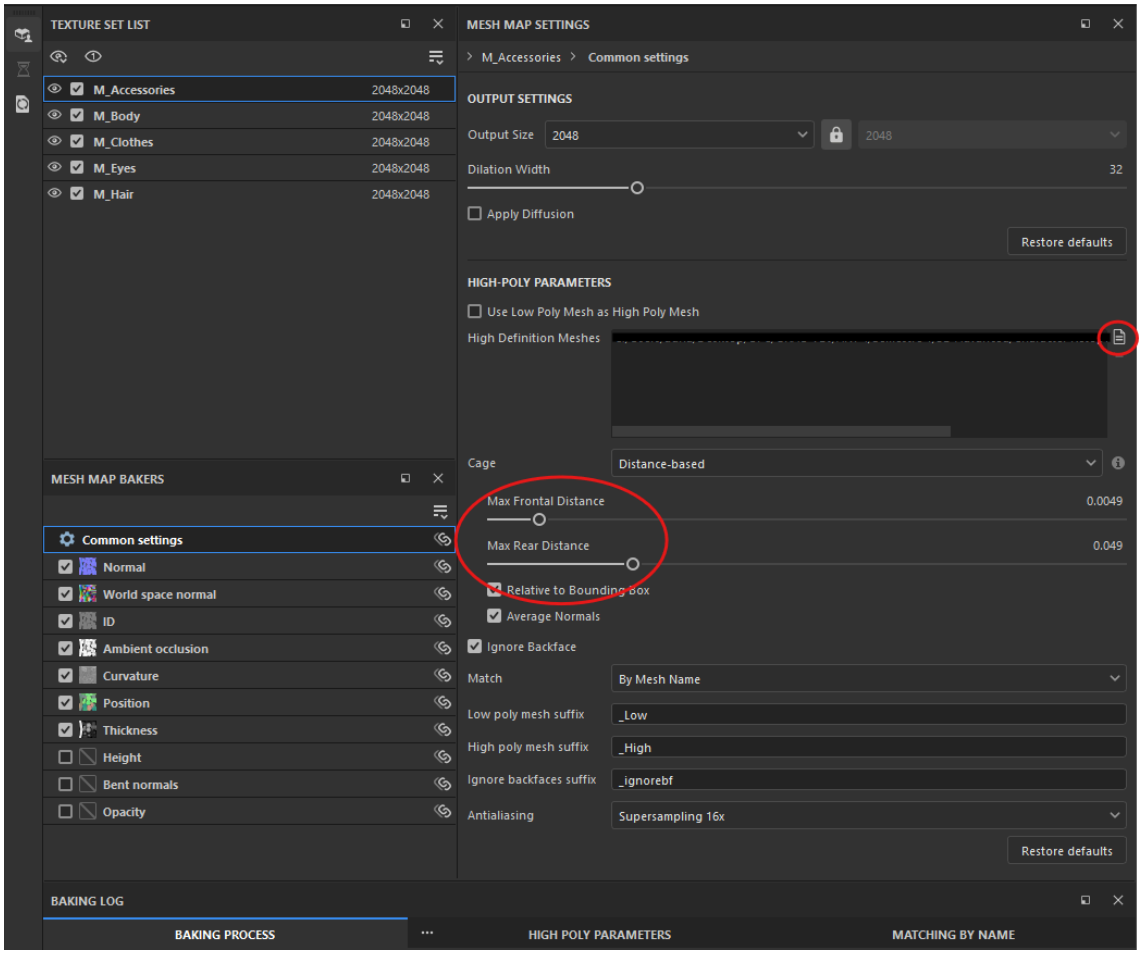
5.1.1.5: Baking

With the Low Poly model and UV mapping done, it is time for texturing. But before that, due to reducing the number of polygons in the Retopology process, lots of details are lost. However, Baking is the solution.

The software selected to do the Baking of the High Poly model details into the Low Poly mesh surface is Adobe Substance 3D Painter. It offers both Baking and Texturing tools in a single application, which speeds up the process.

Once Substance 3D Painter is opened, the Low Poly model needs to be opened into a new project. After that, open the 'Bake Mesh Maps' menu (F8), load the High Poly mesh, adjust the Max Frontal & Rear distances as needed together with Antialiasing, and Self-Occlusion values to 'Only Same Mesh Name'. Then bake the selected textures.

Figure 5.1.1.5.1: Bake Mesh Maps menu from Substance 3D Painter



Once the baking process is over, a set of different mesh maps will have been generated. Thanks to this process, the High Poly details will be seen on the Low Poly mesh as if they were stamped and with no negative performance impact.

Figure 5.1.1.5.2: *Speed* Bake

5.1.1.6: Texturing

As mentioned, the Texturing process has been done with Adobe Substance 3D Painter software, using PBR materials, Masking, and more, using 2048x2048 px textures.

Figure 5.1.1.6.1: *Speed* Texturing

5.1.1.7: Weapon

As a complement to *Speed* to use inside the gameplay of *HyperStrike*, he will be carrying a baseball bat with a futuristic look. The main usage will be the following. With the upper tip of the bat, the character will be able to shoot baseballs, but without the need of having to strike them as usual. Just by aiming at the desired target, the bat automatically triggers a ball.

Figure 5.1.1.7.1: *Speed* Weapon

In addition to the weapon, a projectile specific to the previous weapon was also created. It represents a baseball, which complements the overall design and theme.

Figure 5.1.1.7.2: *Speed* Weapon's Projectile

5.1.2: *Crashwall* (Tank)

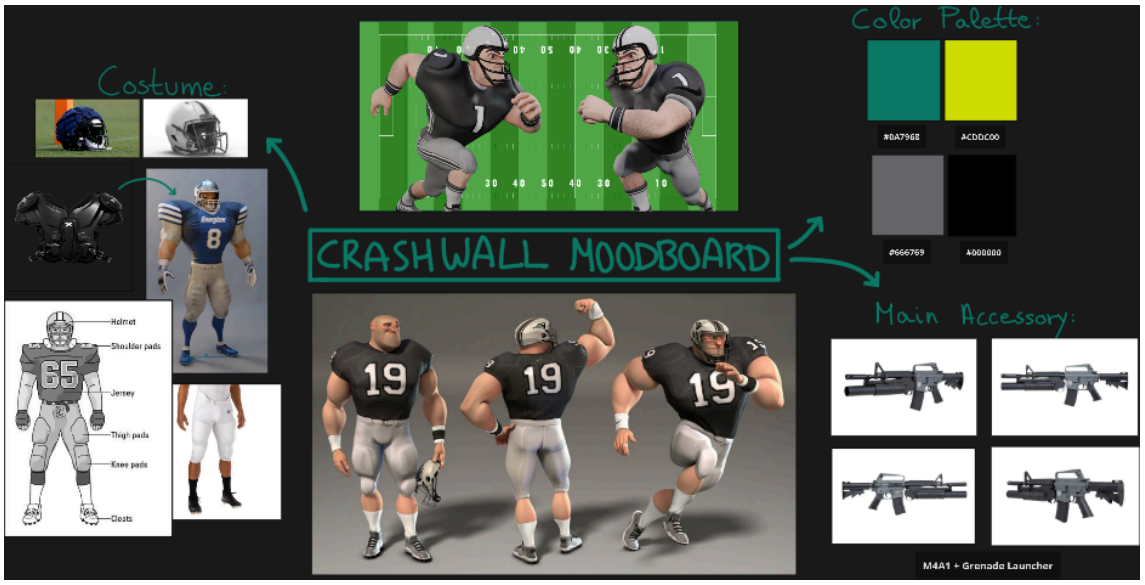
5.1.2.1. Concept Art

Crashwall is a 43-year-old male who is the veteran member of the game. He is known to be the 'Mentor'. Rookie players see him as an old, grumpy man who won't help anyone. On the contrary, *Crashwall* is just honest and willing to guide whoever needs it. He plays the role of 'Tank', being able to deal with receiving a lot of damage and protect his teammates while sacrificing mobility.

He started in the sport at a very young age and was the hope of a whole country. He fought for the championship against one of the greatest of all time and won in his fourth year in the competition. He repeated the same result the following year, earning that way two consecutive titles. He hasn't reached another championship since then, but he has been close. He still has a very solid following and is admired by all of his rivals. His longevity is astounding for such a harsh game, but he is still hungry, even after tasting victory in other categories of the sport.

Below is a moodboard of the Character Design of *Crashwall*. It has been determined that all the elements that the character will be composed of, from its hair, clothing, and accessories to its color palette.

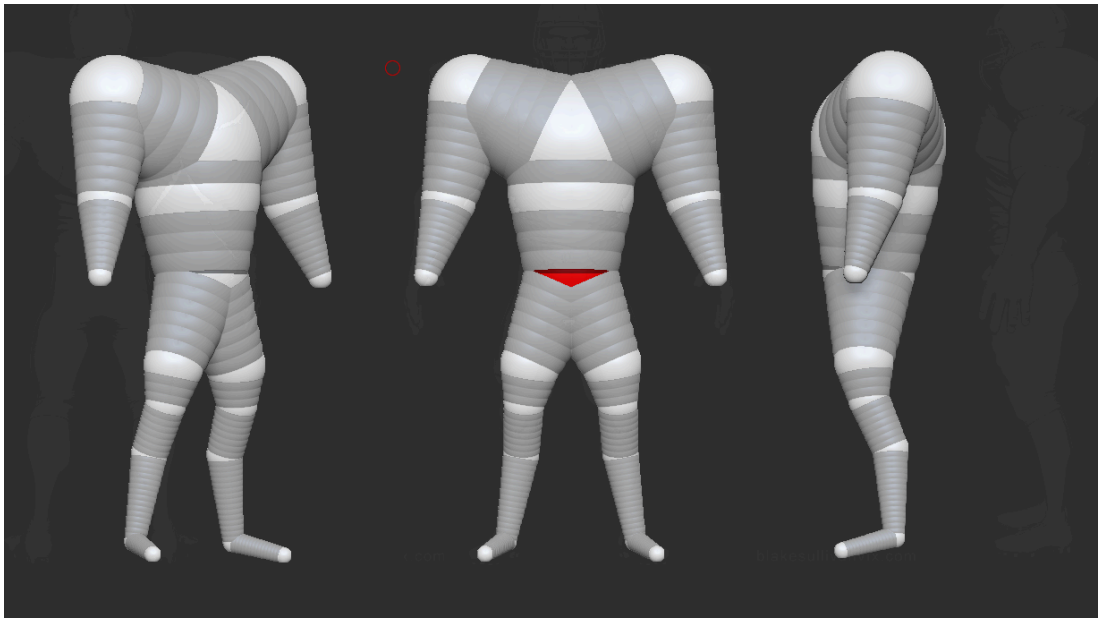
Figure 5.1.2.1.1: *Crashwall* Moodboard



5.1.2.2: High Poly Sculpting

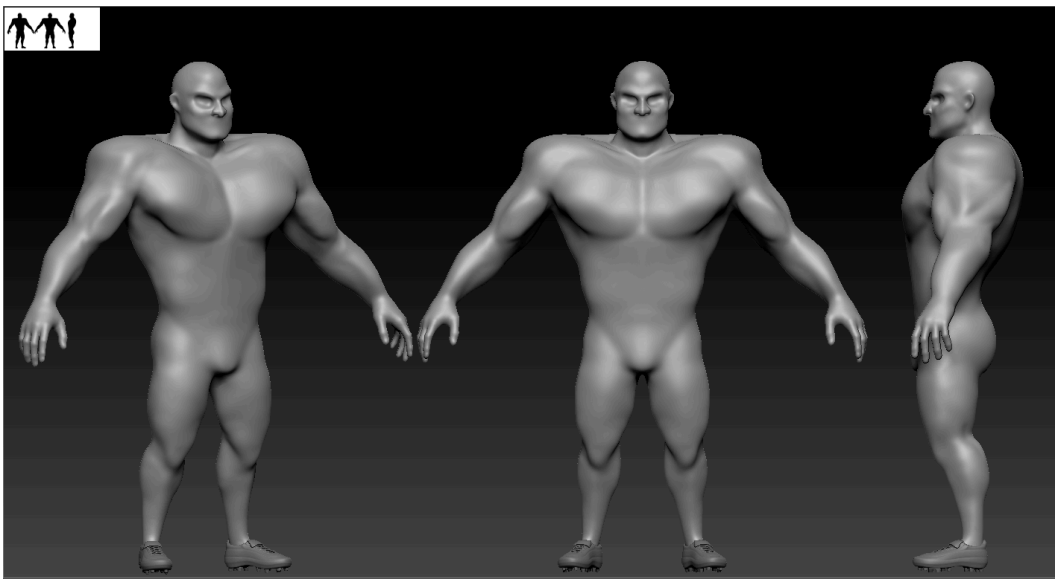
As mentioned in the previous character, to create a High Poly sculpt of the character, it might first start with a blockout using primitive shapes. Again, in this case, using the ZBrush tool called 'ZSpheres', a basic body shape can be created, giving the designed volumes.

Figure 5.1.2.2.1: *Crashwall* ZSpheres Blockout



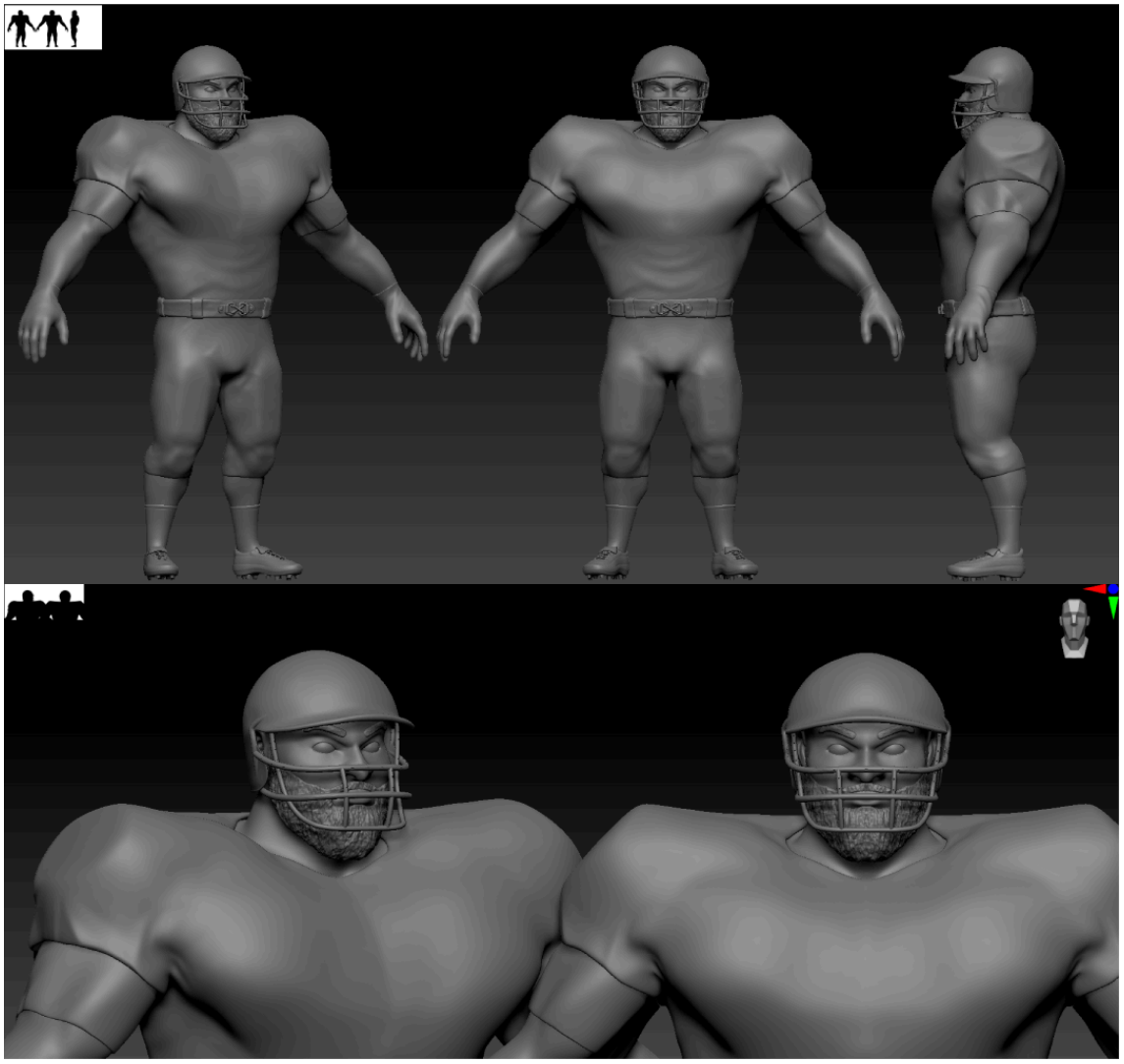
With this base mesh, it is possible to start building up the shape of the body with more details, such as musculature, head, hands, etc.

Figure 5.1.2.2.2: *Crashwall* Body



Once the body is defined, it is time for clothing and accessories. In this case, the character is wearing American football clothes. This type of clothing is characterized by being composed of tight fabric, but including hard protections underneath it, in parts such as the shoulders or legs. That is why it is important to exaggerate a bit these parts when sculpting them, otherwise they will be lost in future stages such as Retopology, Baking, or Texturing, as they won't be easily noticed.

Figure 5.1.2.2.3: *Crashwall* High Poly Model

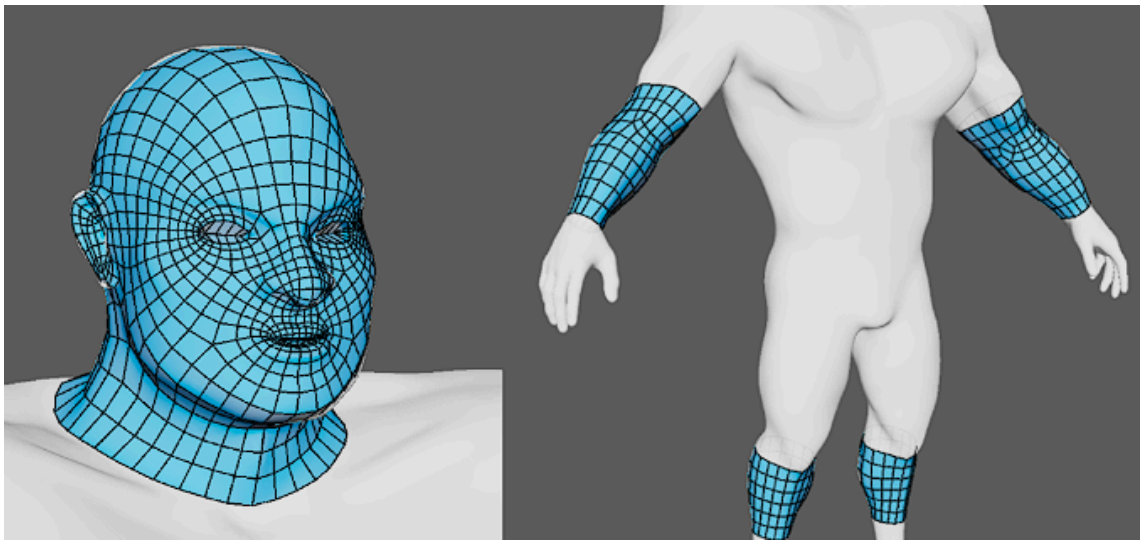


5.1.2.3: Retopology

The retopology process of the *Crashwall* was again very similar to the previous commented. It has been used both manual, for the head and body parts, and automatic retopology styles, for the rest of the parts and clothes of the character.

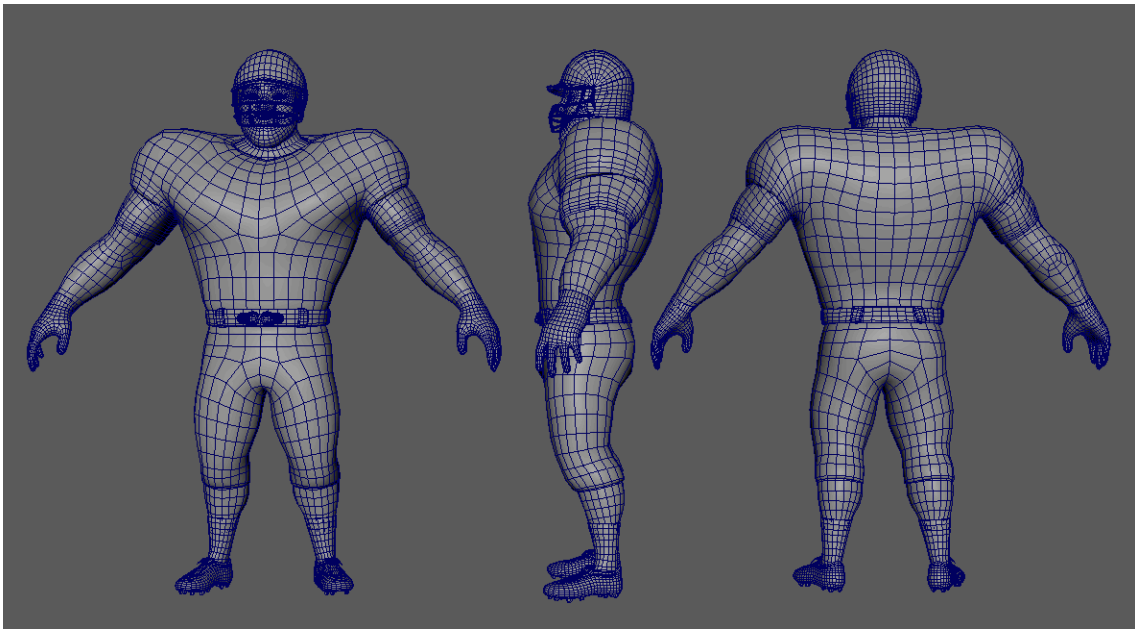
The workflow has been the same. Using Autodesk Maya's 'Quad Draw' tool, the Low Poly version of the character model can be created by connecting vertices and faces manually.

Figure 5.1.2.3.1: *Crashwall* Manual Retopology



For the rest of parts of the character, it has been used again the ZBrush tool called 'ZRemesher', as it helps to speed up the highly time-consuming task that manual retopology is.

Figure 5.1.2.3.2: *Crashwall* Full Character Retopology



The end result of the retopology of *Crashwall* is a 26.000 tris character with all its parts, clothing, and accessories.

5.1.2.4: UV Mapping

In the case of *Crashwall,* we have again 5 different materials assigned to the whole character: Helmet, Face, Head & Body, Clothes, and Accessories. Each material has its own UV space with the corresponding pieces laid out.

Figure 5.1.2.4.1: *Crashwall* UV Mapping

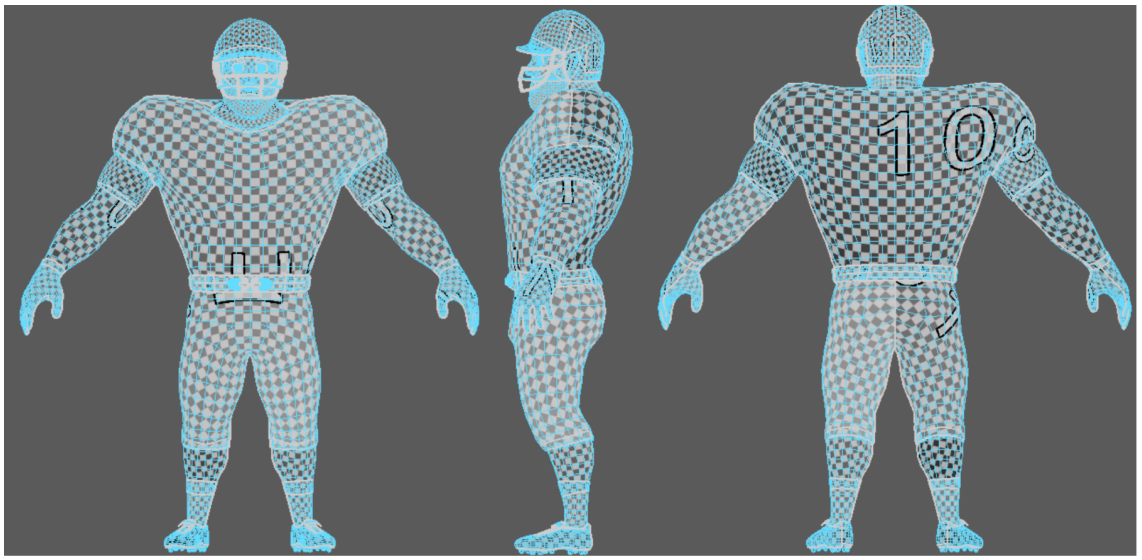
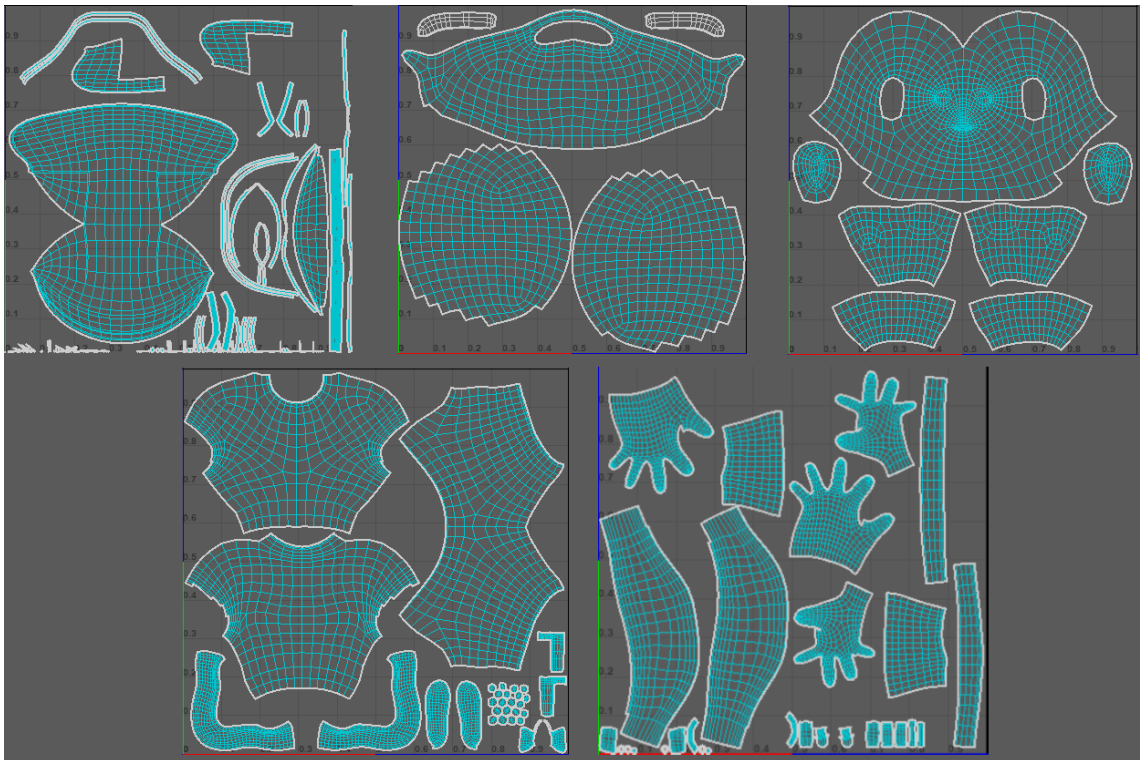


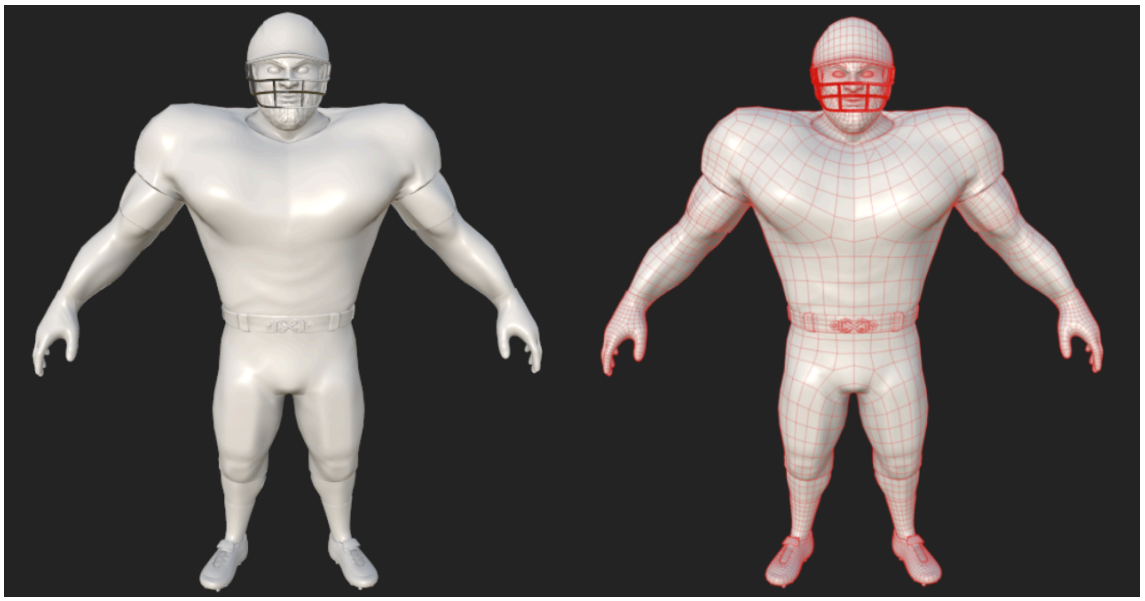
Figure 5.1.2.4.2: *Crashwall* UV Layout for each material



5.1.2.5: Baking

Once the Low Poly model of the character is prepared for Texturing, it is time to bake the High Poly details onto it. Again, to agilize the workflow, it has been used Adobe Substance 3D Painter to do the Baking process, and with the resulting bake mesh maps, texture the character in the same software.

Figure 5.1.2.5.1: *Crashwall* Bake



5.1.2.6: Texturing

The Texturing process has been done with Adobe Substance 3D Painter software, using PBR materials, Masking, and more, including the latest 'Stylization' filter released.

Figure 5.1.2.6.1: *Crashwall* Texturing

5.1.2.7: Weapon

As a complement for *Crashwall* to use inside the gameplay of *HyperStrike*, he will be carrying an M4A1 rifle with the Launcher Grenade accessory attached to it. The usage of this weapon is easy to understand. The character is able to shoot normal projectiles with the main rifle, as well as shoot grenade-like projectiles with the grenade launcher accessory.

Figure 5.1.2.7.1: *Crashwall* Weapon

5.1.3: *Nanoflow* (Healer)

5.1.3.1: Concept Art

Nanoflow is a 30-year-old female who is the healer of the team. She is known to be a selfless person, doing everything for the well-being of her teammates and to work towards victory. She plays the role of 'Healer', always staying in the front of the action, seeking injured teammates to heal and help them.

She has always liked medicine from a young age and knew that she wanted to attend medical school. She has also played football since she was almost a baby and still practiced it before entering the league. This footballing passion and the nature of Hyperstrike made her realize that she could exploit her capacities to the maximum, helping her teammates when needed to, and with the speed she carries, also be an integral part of the team. She has competed to the maximum level in football, earning several titles in the junior categories of her home team. Now she wants to achieve the maximum level in Hyperstrike and win everything that she can.

Below is a moodboard of the Character Design of *Nanoflow*. It has been determined that all the elements that the character will be composed of, from its hair, clothing, and accessories to its color palette.

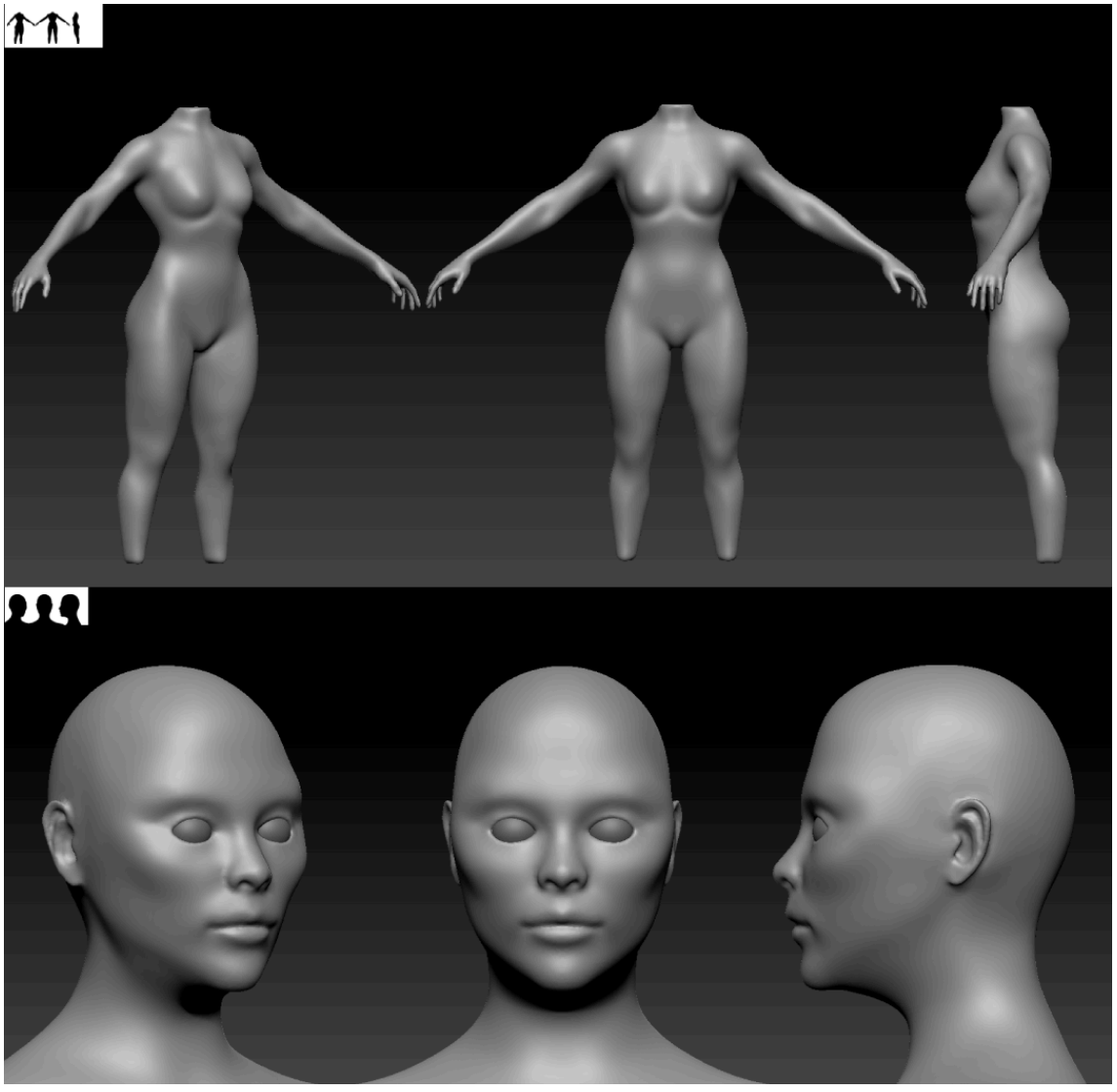
Figure 5.1.3.1.1: *Nanoflow* Concept Art

5.1.3.2: High Poly Sculpting

To start creating the High Poly version of the character *Nanoflow*, again it has been used ZBrush's 'ZSpheres' tool to create the initial shape. Later, it has been given the defined shapes of the body.

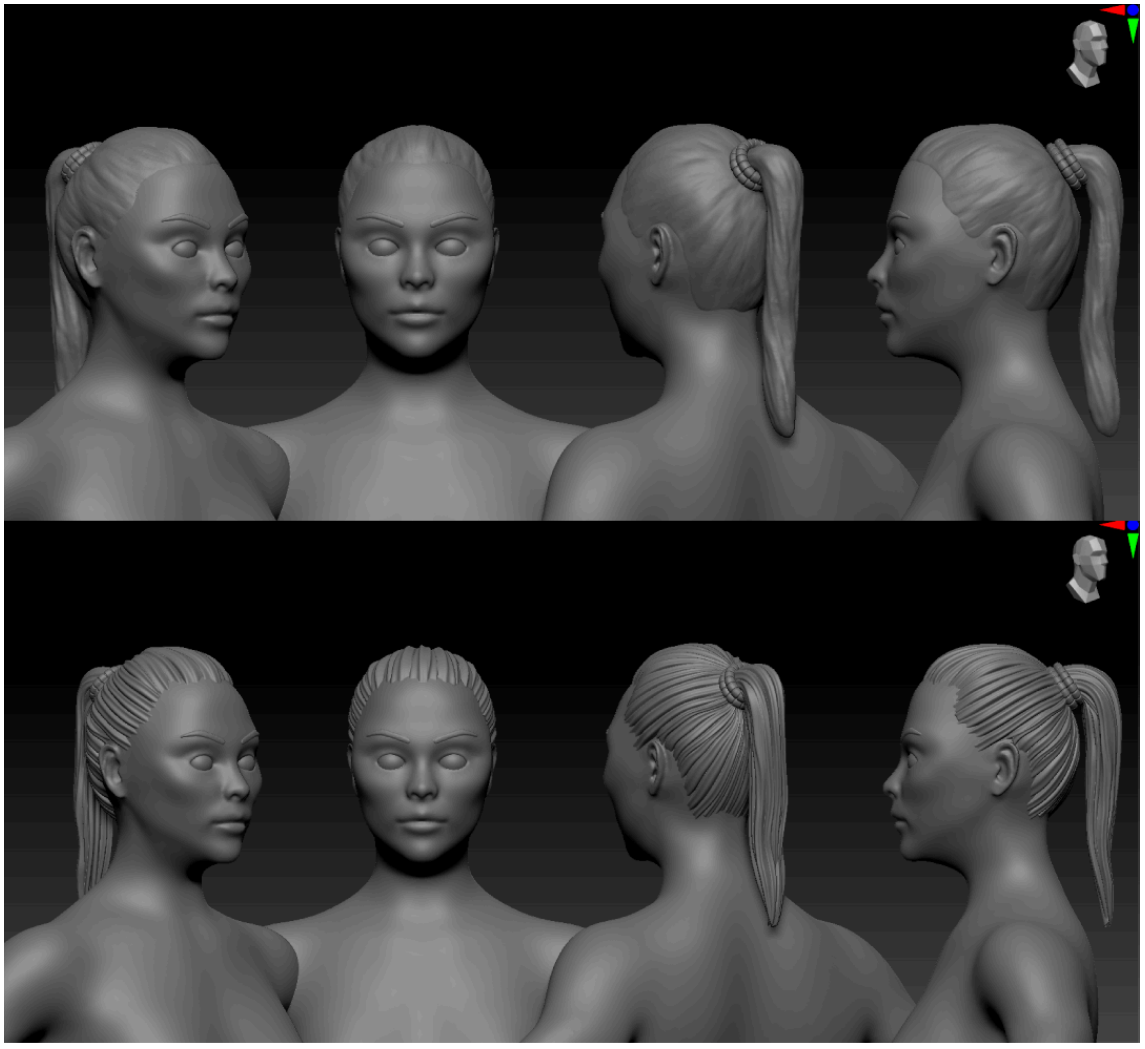
In this case, different from the other two characters created for *Hyperstrike*, it has been important to work and give importance to the definition of volumes, as females have different traits from males. The most evident differences are the amplitude of the hips and legs, which tend to be wider, as well as the shape of the chest, breast, and arms, being these last ones slimmer. Another factor to take into consideration is the shape of the lower jaw, which in women takes a sharper form.

Figure 5.1.3.2.1: *Nanoflow* Body & Head Blockout



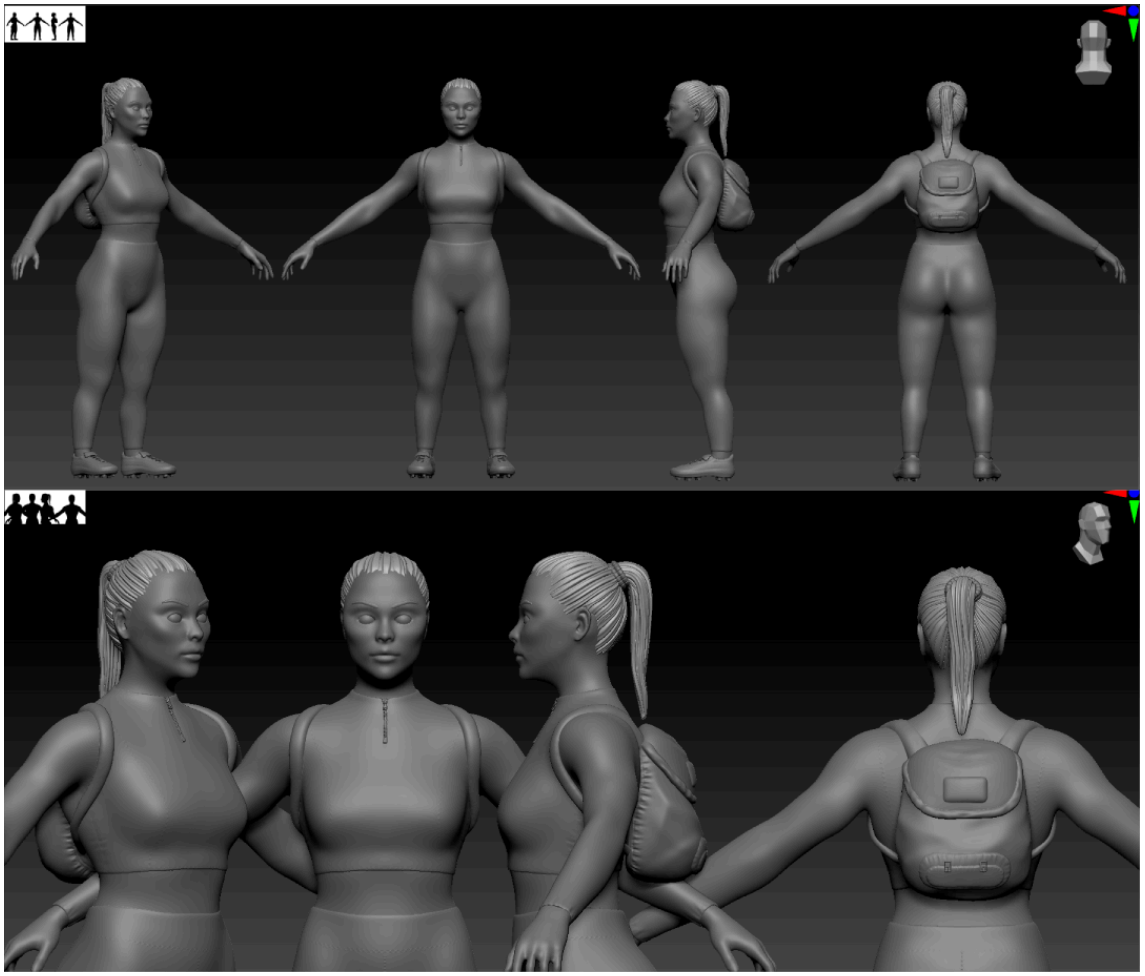
One of the major challenges when creating the *Nanoflow* character was the hair. The design of the character is set to have a ponytail. To do so, the process was to first create an extruded mesh from the head with the shape of the hairline, and then append a new sphere to this base and give it the form of the ponytail. When this base was created (see upper image below), it was time to use one of the hair brushes that ZBrush offers. It offers the possibility of creating a hair that follows a curve that snaps to the surface of any object, in this case, the base of the hair previously created. By repeating this process until covering all the hair base, it has been able to create the final hair, which can be seen in the lower image below.

Figure 5.1.3.2.2: *Nanoflow* Hair Creation Process



After having the body and head finished, it was time for the clothing and accessories of the character. The biggest challenge when it came to creating them was the medkit backpack, which was built from scratch using primitive shapes and then shaped until it got the final result.

Figure 5.1.3.2.3: *Nanoflow* High Poly model

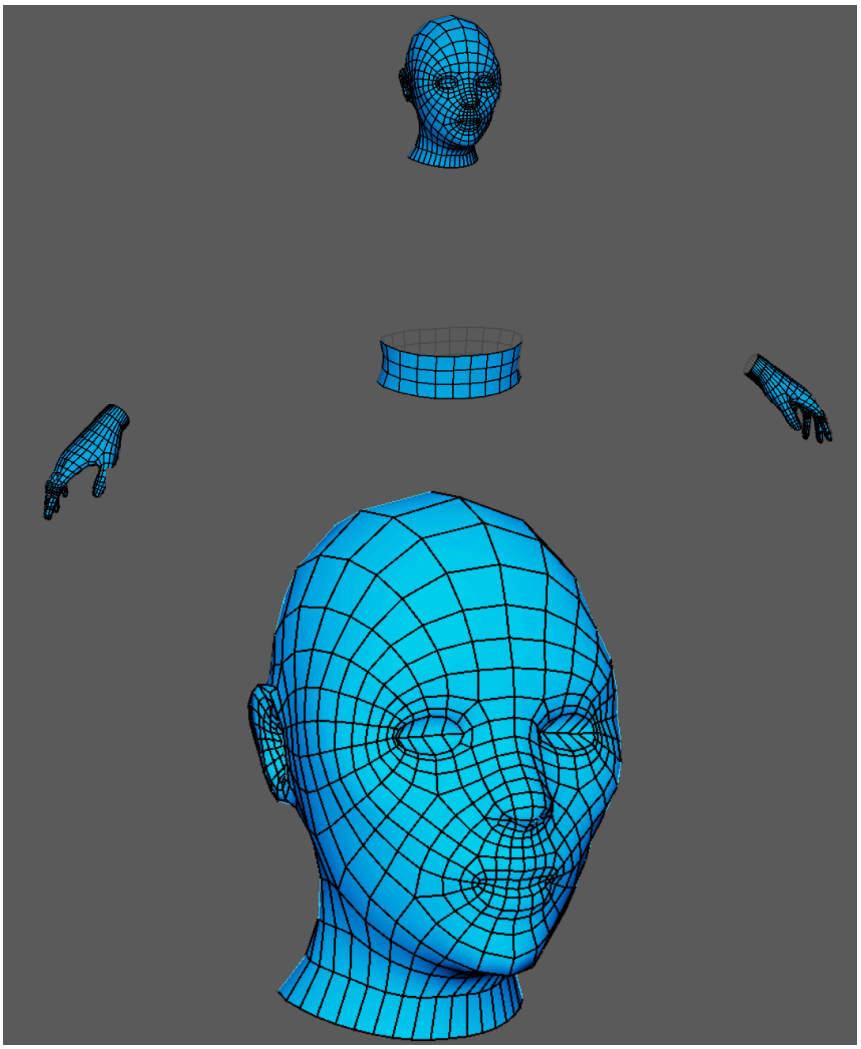


5.1.3.3: Retopology

With the High Poly model of the character finished, it was time for one last retopology process of it in order to make the character 'game-ready' to be used in the game engine.

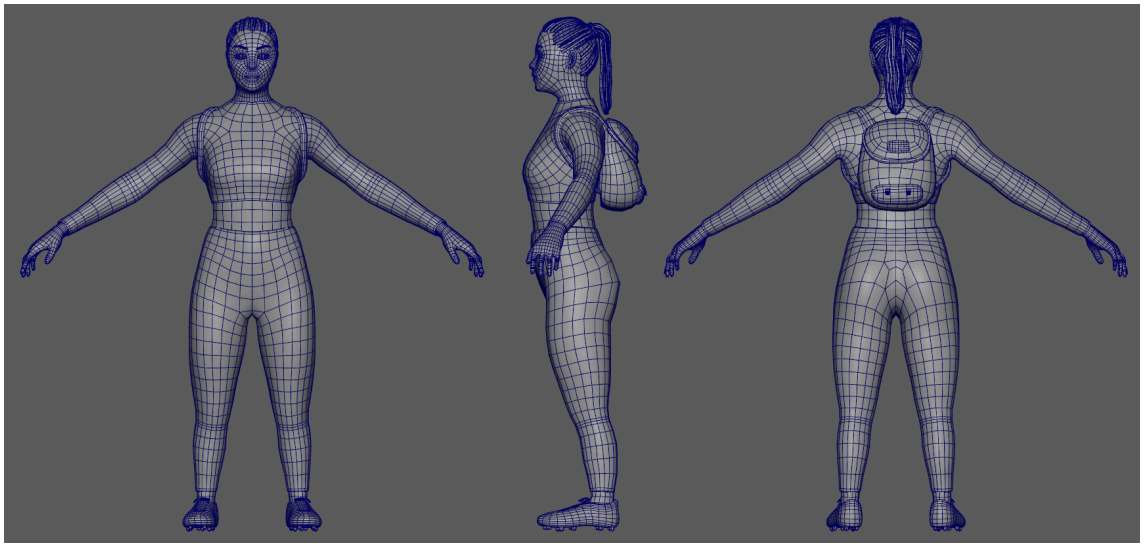
The software used was Autodesk Maya, thanks to its previously mentioned and explained 'Quad Draw' tool, for the manual retopology of the Head and Body parts. For the rest of elements, it was used the 'ZRemesher' tool from the ZBrush software, as it speeds up the process but at the same time getting a solid result. Below is an image of the manual retopology of the body parts obtained.

Figure 5.1.3.3.1: *Nanoflow* Manual Retopology



The rest of the elements that were remeshed, built the final Low Poly version with a result of a character with approximately 29.000 tris.

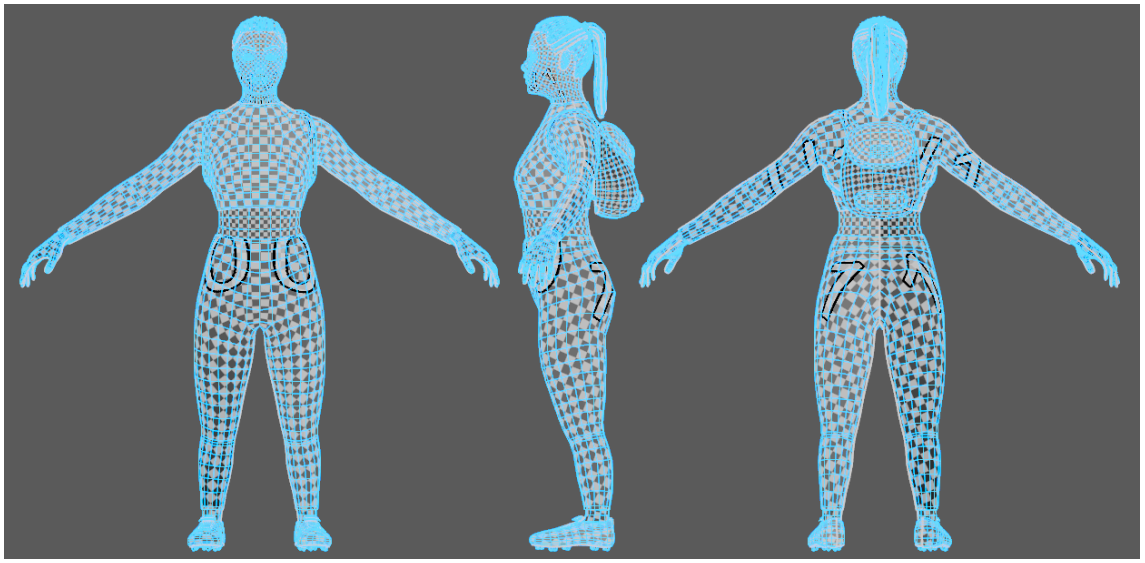
Figure 5.1.3.3.2: *Nanoflow* Full Character Retopology



5.1.3.4: UV Mapping

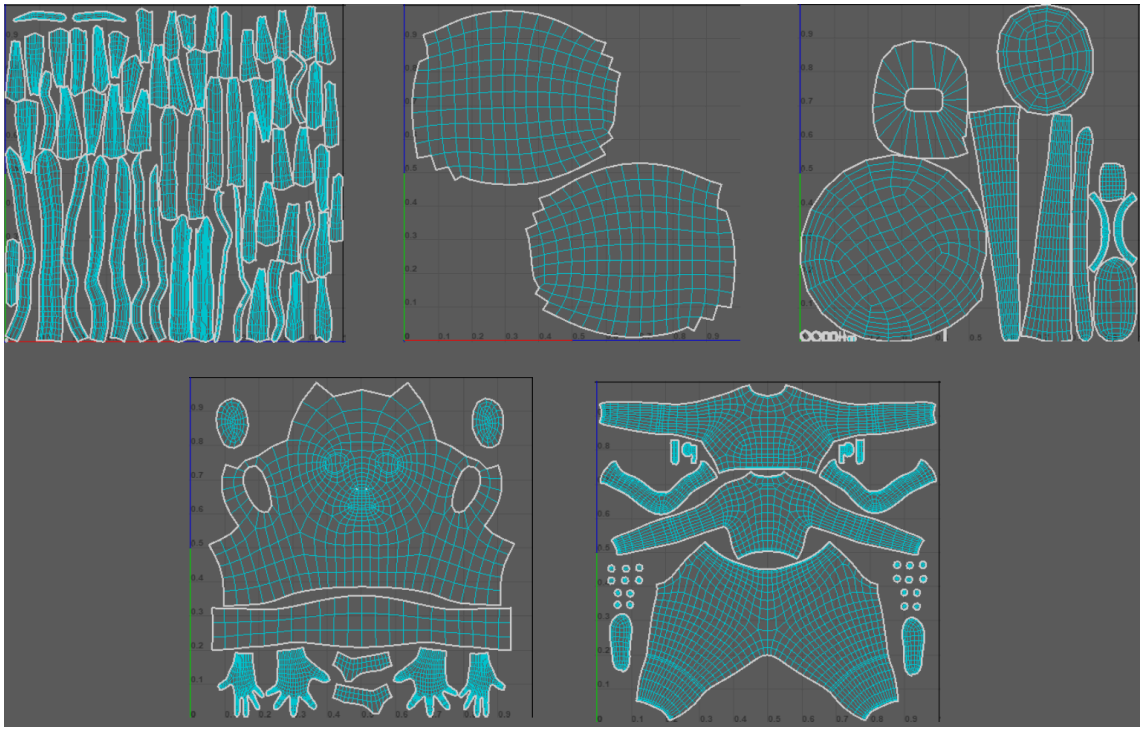
With the Low Poly model ready, it was time for the UV Mapping before exiting Autodesk Maya and opening Adobe Substance 3D Painter.

Figure 5.1.3.4.1: *Nanoflow* UV Mapping



The material assignment for the character *Nanoflow* was again composed of 5 different instances: Hair, Eyes, Accessories, Body, and Clothes, each one with its elements laid out in each corresponding UV space.

Figure 5.1.3.4.2: *Nanoflow* UV Layout for each material



5.1.3.5: Baking

With the UVs of the character done, it was time to bake the High Poly details from ZBrush to the Low Poly model. The software used was Adobe Substance 3D Painter, in which it was first opened a new project with the Low Poly mesh, and later the High Poly mesh was loaded in the 'Bake Mesh Map' menu. The result obtained can be seen in the image below.

Figure 5.1.3.5.1: *Nanoflow* Bake

5.1.3.6: Texturing

Using *Substance 3D Painter*, the texturing result achieved was the following. Again, using PBR materials, Masking, Height Maps, and more, including the latest 'Stylization' filter released.

Figure 5.1.3.6.1: *Nanoflow* Texturing

5.1.3.7: Weapon

As a complement for *Nanoflow* to use inside the gameplay of *HyperStrike*, she will be carrying a sci-fi pistol with a futuristic look. The usage of this weapon is easy to understand. The character is able to shoot projectiles with the pistol. However, these projectiles won't deal a great amount of damage.

Figure 5.1.3.7.1: *Nanoflow* Weapon

5.2: Environment

When it comes to workload in the game development context, environmental art might be one of the most time-consuming tasks, taking into account the whole process. That is the reason why, to set a realistic development objective for this project, it was agreed to only create one environment or arena for the game prototype, as it is normally a better decision to create a single product, but that it has the potential to be finished and polished as much as possible. The place where *Hyperstrike* games take place is the 'Pinball Arena'. In addition to that, a lobby waiting room has been created, which represents the locker room, where players will be waiting before the game starts.

5.2.1: Lobby Room

5.2.1.1: Level Design

Before starting to create any 3D asset, it was crucial to work together with the Design team member to determine what the Lobby Room would look like, in terms of structure, layout, illumination, etc.

Once this work was done, a Level Design Document (LDD)[24] was written with all the elements discussed, where the following was decided:

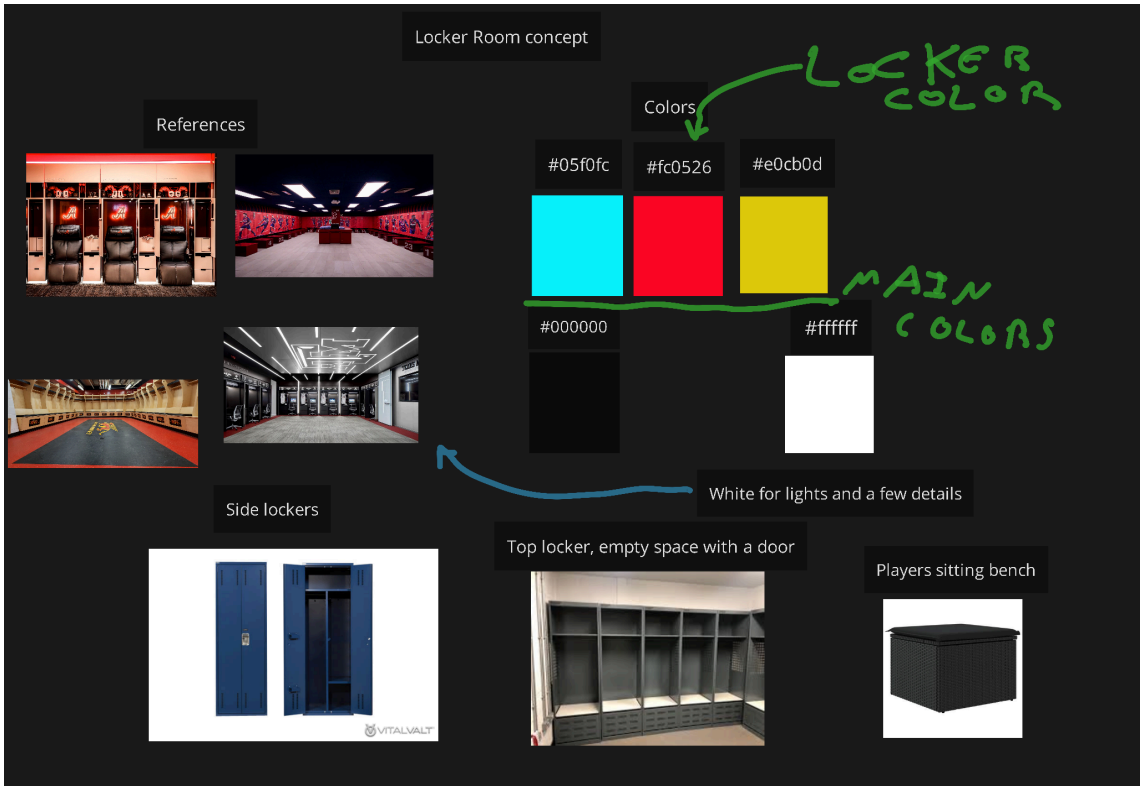
"In the Locker Room, there are two main areas: the open space where players move around and can kick a ball, and the lockers themself.

The open space big area has the Hyperstrike logo on the floor, since it is standardized for the competition, meanwhile the lockers have a black cushion for players to sit down and change, two lockers on the sides, and one up. These lockers have the colors of Hyperstrike, to make them exclusive for the league."

Before starting with the next phase, the Blockout, an initial moodboard was made, with some visual references of how the Locker Room structure should look, how individual lockers are supposed to be, as well as the color palette of the different elements and the lighting.

[24] LDD: A document that shows the plan for the level. LDDs shouldn't be treated as a rigid, firm guidebook, though. They're great for initial planning, figuring out difficult problems ahead of time, pitching to other team members, and looking back on to ensure the level is generally in alignment with the initial idea. (@JacobWMilld via Reddit/leveldesign)

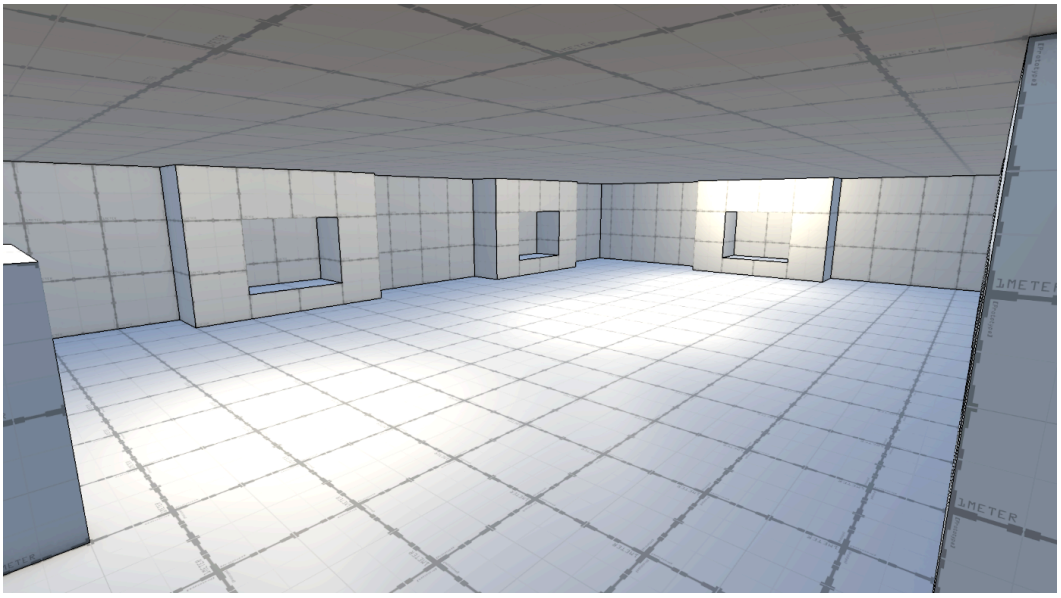
Figure 5.2.1.1.1: Locker Room Moodboard



5.2.1.2: Blockout

When developing a video game, the first instance of a level is the blockout of it, where the initial layout is designed and placed, trying to decide which volumes will form the 3D space, and see if they work well. This initial task is normally done by the Level Designers of the team, and it was no exception in the case of *HyperStrike*.

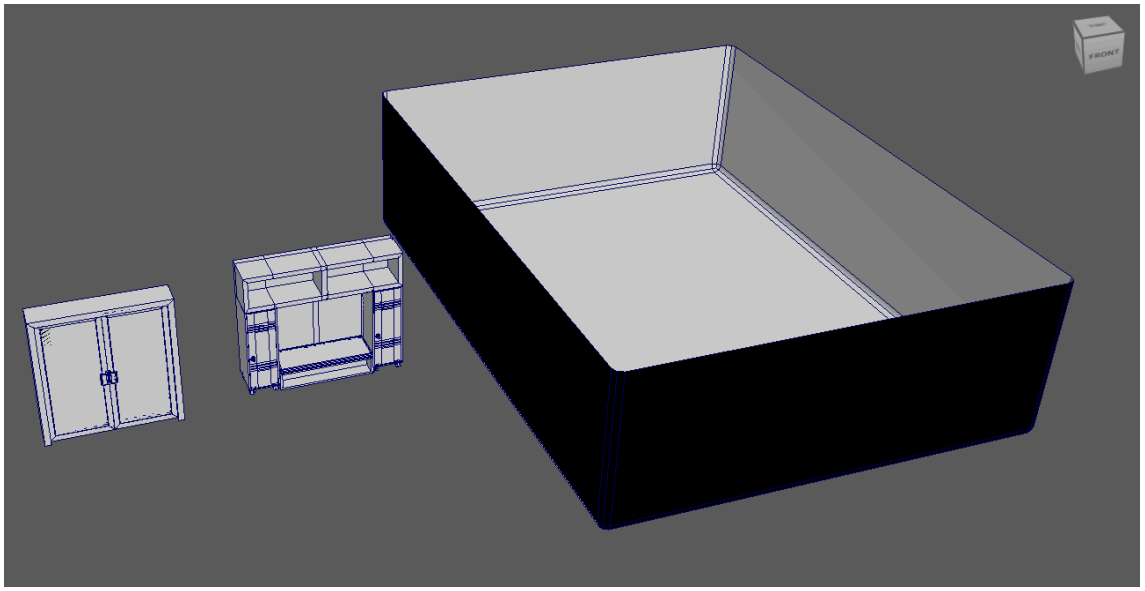
Figure 5.2.1.2.1: Lobby Room Blockout



5.2.1.3: 3D Modeling

With the blockout created, all the necessary assets to be modeled can be identified. In the case of the Lobby Room, there are 3 main pieces: the room structure, the lockers, and the entrance/exit door. In the case of the locker and door, it is only necessary to make one piece of each, as they can be later duplicated and placed where desired within the game engine scenario.

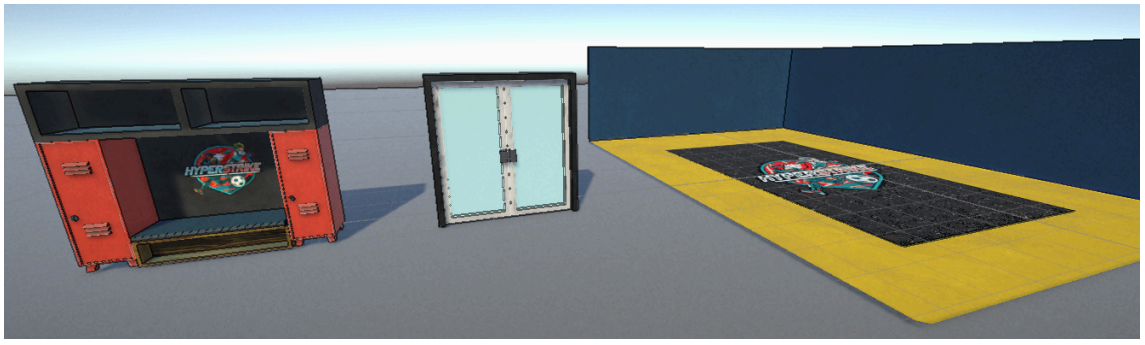
Figure 5.2.1.3.1: Lobby Room 3D Modelling



5.2.1.4: Texturing

With the pieces modeled and with the UVs unwrapped, it is time to texture them. To do so, the instructions and color palette from the design were followed, trying to convey the level design idea. To do so, Adobe Substance 3D Painter was used, adding the final touch using the 'Stylization' filter.

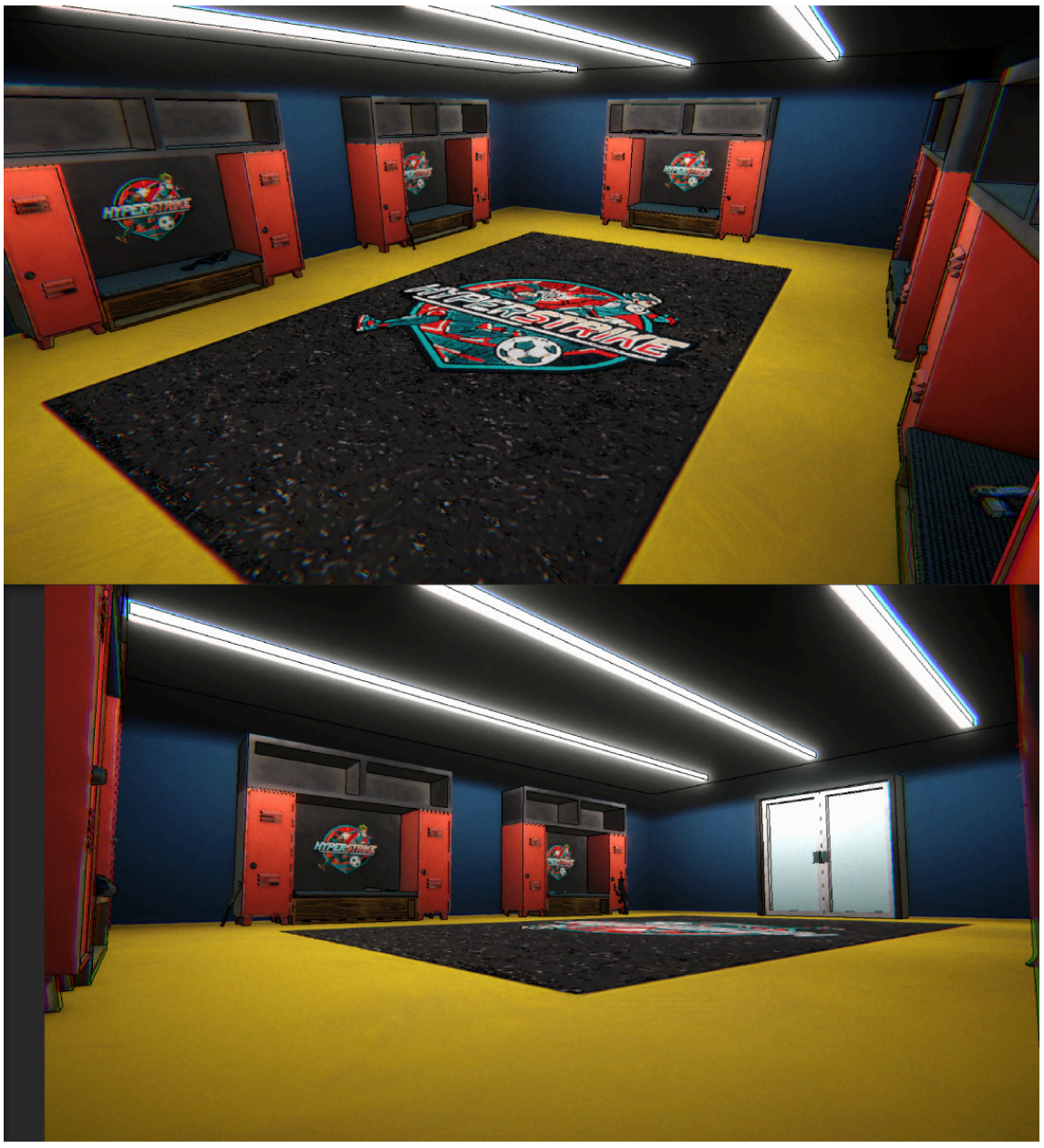
Figure 5.2.1.4.1: Lobby Room Texturing



5.2.1.5: Assembly in Unity

Once the final pieces were finished, it was time to assemble the scenario within the game engine in the Unity Editor, setting the 3D meshes in the correct place, materials, lighting, post-processing, and more.

Figure 5.2.1.5.1: Lobby Room Final Look



5.2.2: Pinball Arena Room

5.2.2.1: Level Design

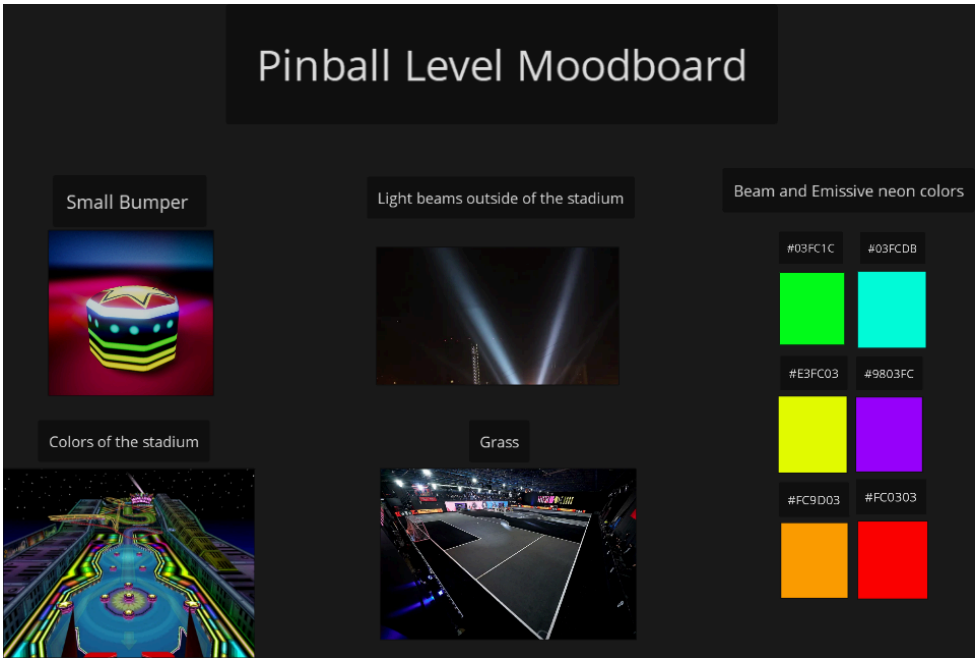
The Pinball Arena Room represents the main scenario where the gameplay takes place. Again, before starting the production of any asset, both the Design and Art members need to work together on determining what the arena would look like, in terms of structure, layout, illumination, etc.

Once this work was done, a Level Design Document (LDD) was written with all the elements discussed. Regarding the environment of the level, the following was decided: *"The arena has a structure similar to a pinball board with dark surroundings but bright colors on the elements that take place in the action of playing pinball.*

The outside of the dome will have an open-ceiling stadium with fans wearing colorful neon clothing. The stands will be dark with neon lights around them, differentiating each of the bleachers' heights. The bleachers have an industrial look to them, made out of a darkish metal, and the aforementioned neon lights separate one from another."

The LDD features more details and characteristics from the design of the Pinball Arena. However, in this report, they are not going to be discussed, as it is not the final objective of this project. However, an initial moodboard with a key reference of all the important aspects to take into consideration was made by the design team, in which it was taken into consideration the overall look of the lighting, color palette, etc.

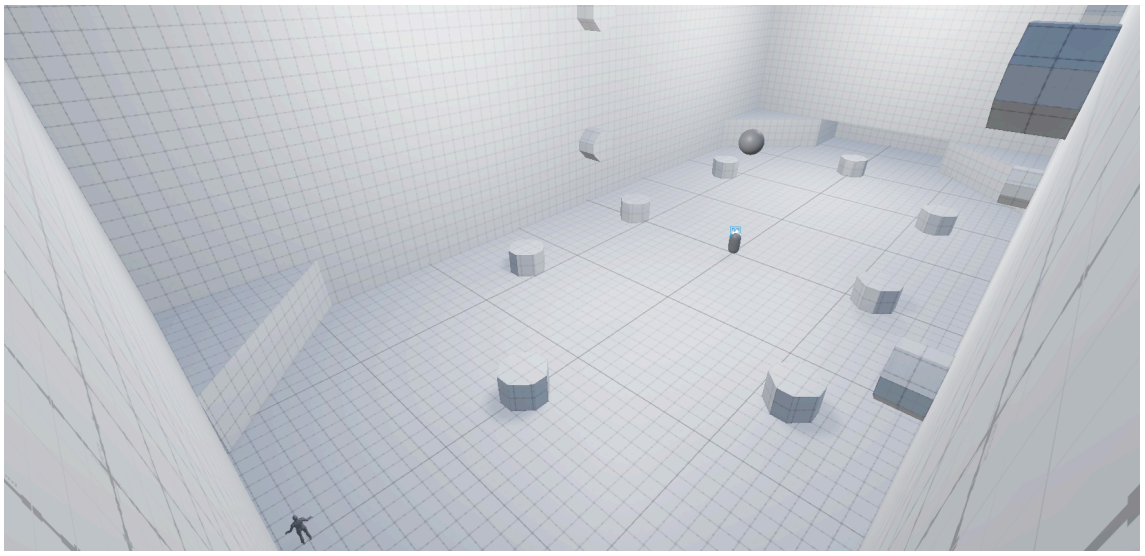
Figure 5.2.2.1.1: Pinball Arena Room Moodboard



5.2.3.2: Blockout

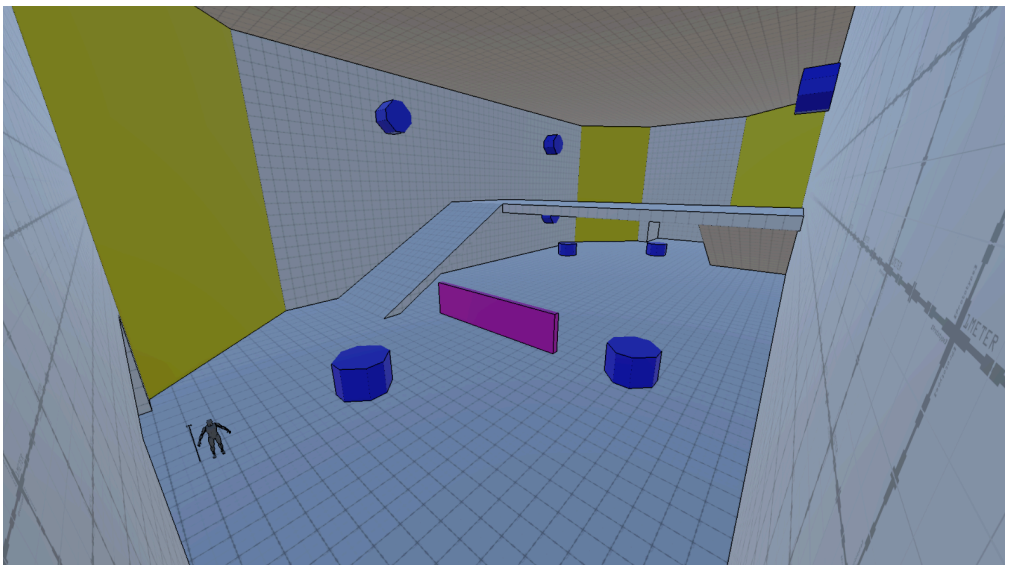
Once everything was decided, it was time to start the production of the stage. In the image below, it can be seen the first ever blockout elaborated within the Unity project of the Pinball Arena. Note that it is the first iteration, and it is taken as a base to start working on, meaning that it is guaranteed that it will change during its lifetime.

Figure 5.2.3.1: Pinball Arena's First Blockout Iteration



With the first iteration of a blockout, what it was looking for was to have an orientation of how the arena would look and to see what the proportions with respect to the players. As the days went by, this blockout went evolving, arriving at the final iteration seen below, in which new elements were added to it, in order to enhance the gameplay experience of the user.

Figure 5.2.3.2.2: Pinball Arena's Final Blockout Iteration



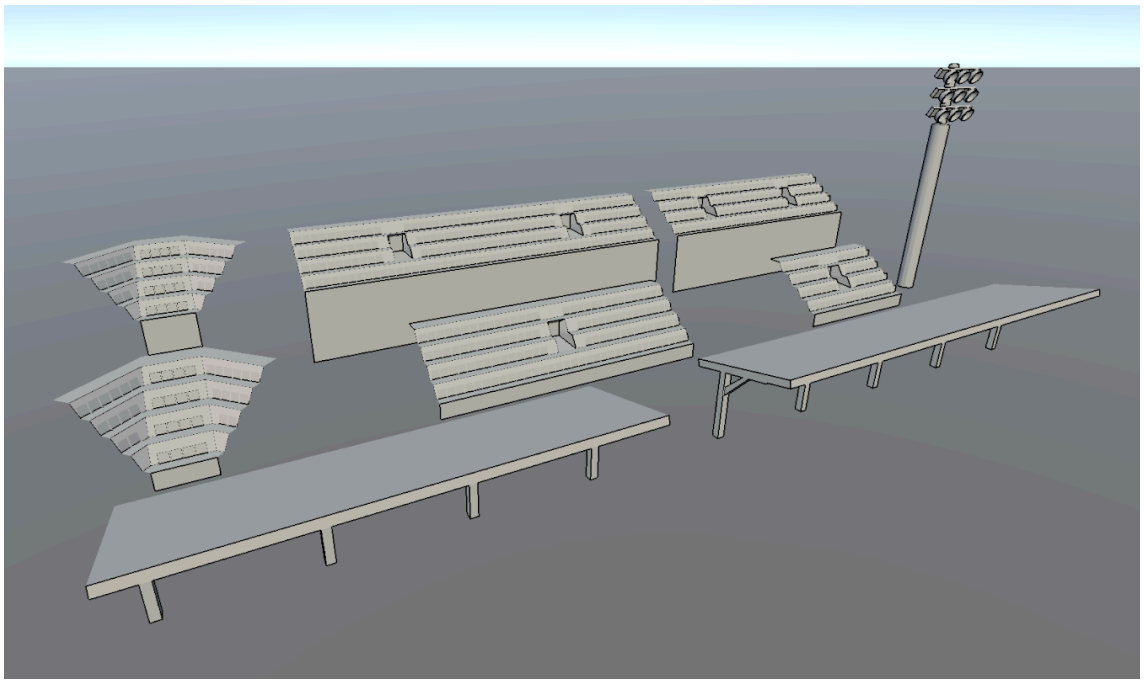
5.2.1.3: 3D Modeling

As it was decided during the initial steps of the design process, the Pinball Arena was set to have an open-ceiling stadium. To do so, one of the best options was to create a modular kit[25] of pieces of the stadium that, when placed together, would form the entire stadium. By doing it like this, it is only necessary to create one single unique piece of all the possible ones and later duplicate and snap them together to form a larger structure.

In the case of the stadium, the modular kit consisted of the following nine pieces:

- *MK_GrandStand_Corner_1*
- *MK_GrandStand_Corner_2*
- *MK_GrandStand_Lateral_1*
- *MK_GrandStand_Lateral_2*
- *MK_GrandStand_Goal_1*
- *MK_GrandStand_Goal_2*
- *MK_GrandStand_Roof_Short*
- *MK_GrandStand_Roof_Large*
- *MK_GrandStand_Light*

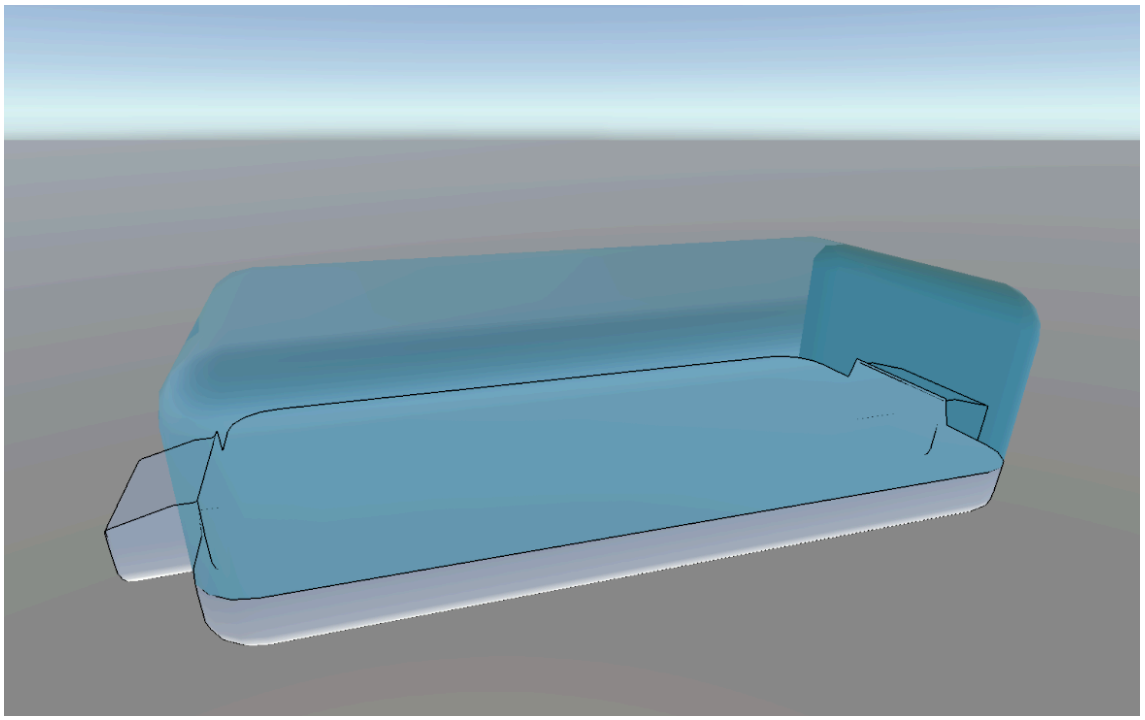
Figure 5.2.1.3.1: Stadium Grand Stand Modular Kit Pieces



[25] Modular Kit: A set or collection of reusable and grid-aligned assets created with the objective of snapping together and form effective but efficient environment structures, such as buildings, houses, etc.

With the modular kit of the stadium grandstand, it was time for the modelling of the Pinball Arena itself, together with the elements of the match field. To do so, the first thing done was the outer structure of the arena, which was given the name of 'cage'. The arena cage was made of two different pieces: the lower part, which defines the area of the match field as well as the goal sizes, and the upper part, which delimits the volume of the cage.

Figure 5.2.1.3.2: Pinball Arena Cage Mesh

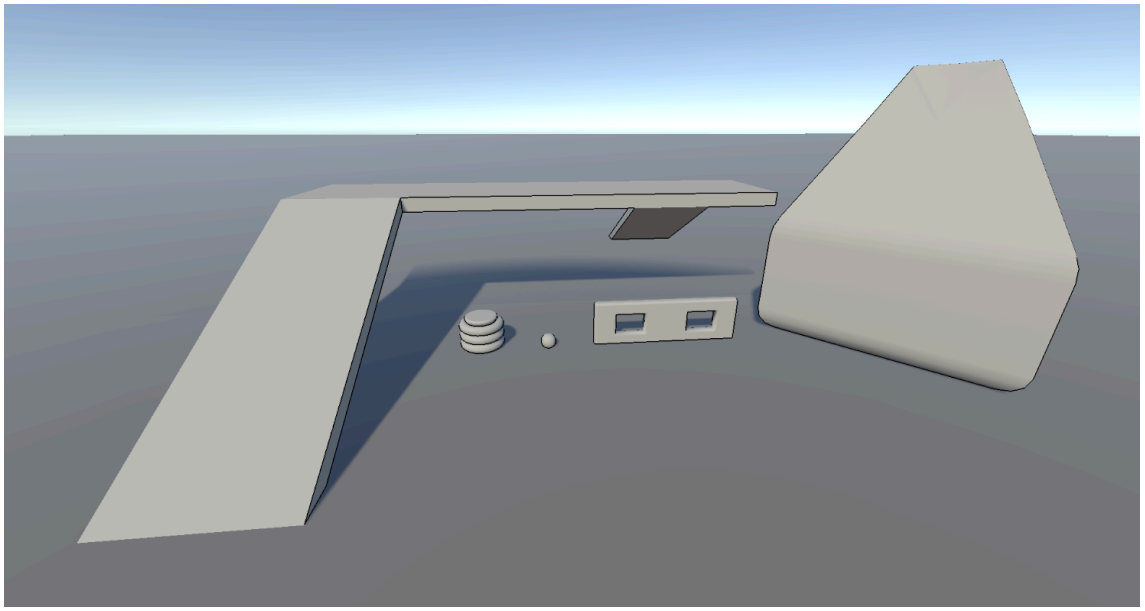


Up until this point, the assets missing from the Pinball Arena were the match field elements, which were directly related to the pinball game concept. The list of these elements was:

- *SM_Bouncer*
- *SM_BouncerCorner*
- *SM_SecondHeight*
- *DM_BoxBarrier*
- *DM_PinBall*

Both Static (SM) and Dynamic (DM) Meshes were included during the production of these assets, as all of them would be placed and play their role inside the arena cage structure during the gameplay of HyperStrike's matches.

Figure 5.2.1.3.3: Pinball Arena Elements' Meshes

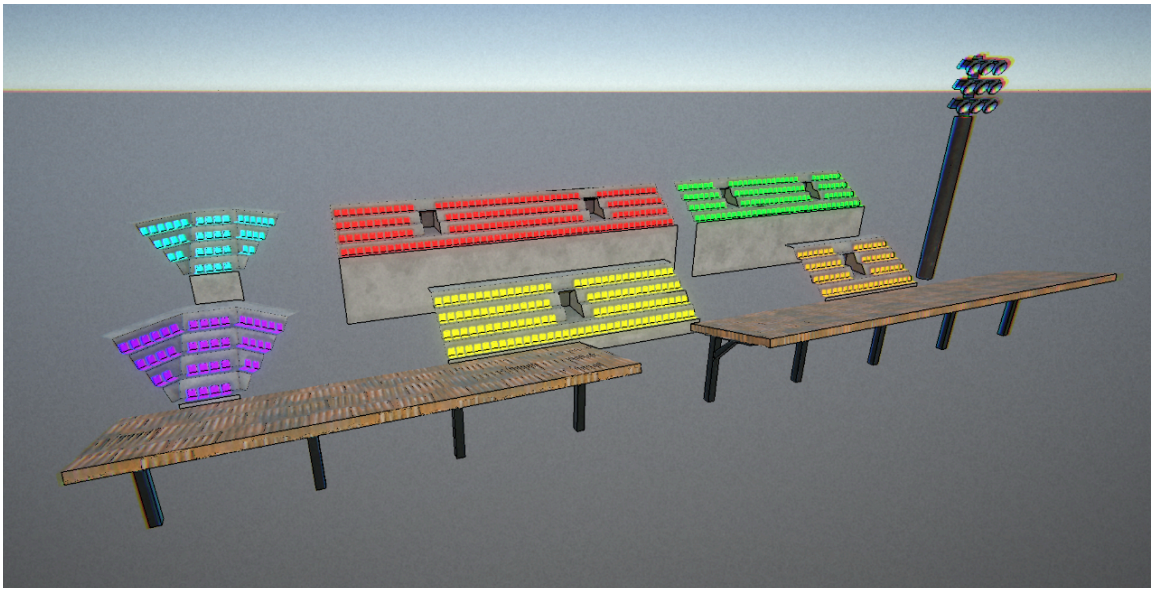


5.2.1.4: Texturing

When all the necessary assets were already modeled, it was time for the texturing. Following the level design decisions taken during the early stages of the production, the following results were achieved.

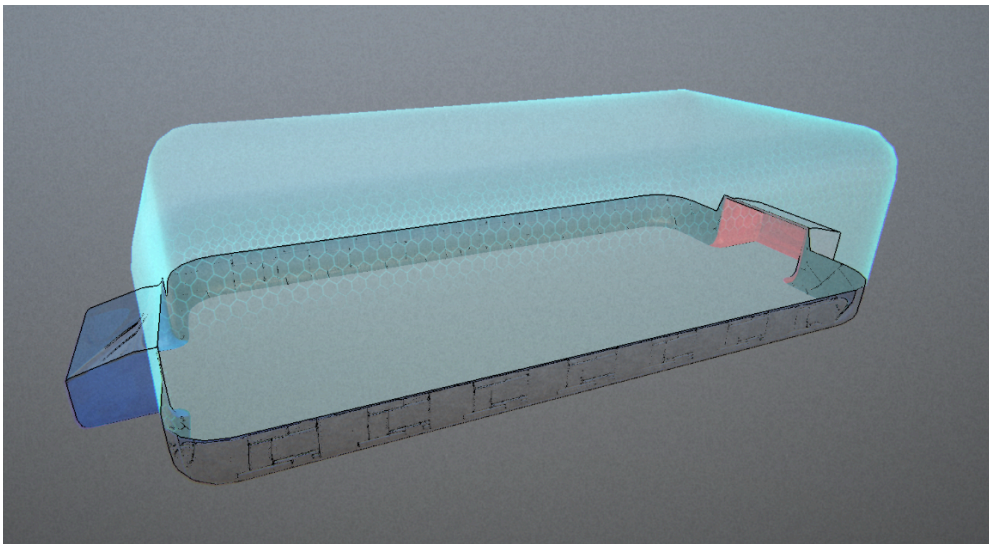
The first assets to be textured were the ones from the Stadium Modular Kit. It was done using Substance 3D Painter, and later on, Alpha Texture Masking and Emissive materials from Unity for the seats.

Figure 5.2.1.4.1: Stadium Grand Stand Modular Kit Texturing



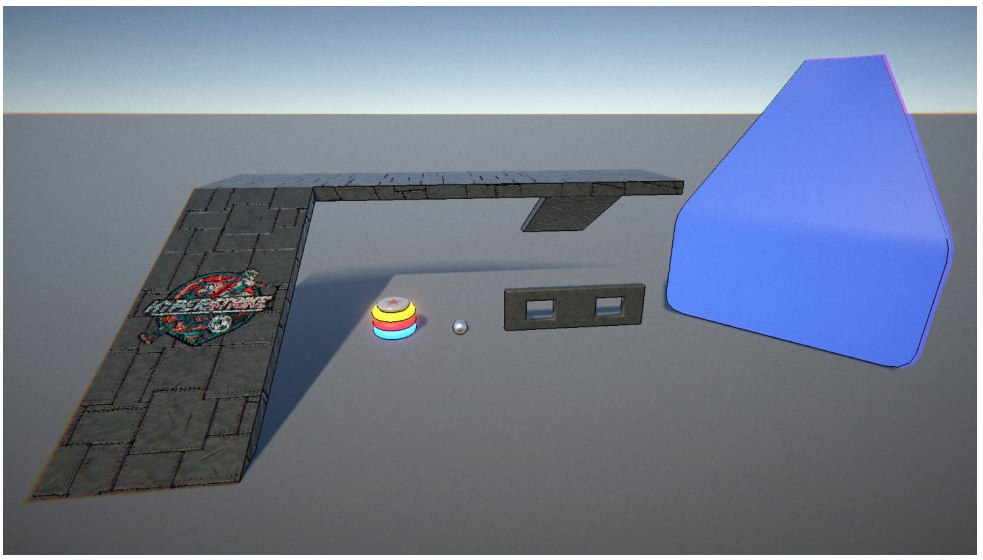
Then it was time for the Arena Cage structure. As mentioned in the previous section, this asset had two parts. The lower part was textured using Substance 3D Painter. However, in the upper part, a Shader Graph material[26] was used, which recreates a 'forcefield' effect, similar to a shield, that was a transparent surface with dynamic details.

Figure 5.2.1.4.2: Pinball Arena Cage Texturing



Last but not least, it was time for the texturing of the different elements of the Pinball Arena. Again, a combination of Substance 3D Painter and Unity's Emissive materials was used. By using these materials, it was possible to later update the emission colors if needed within the Unity Editor, without having to re-texture in another software.

Figure 5.2.1.4.3: Pinball Arena Elements' Texturing



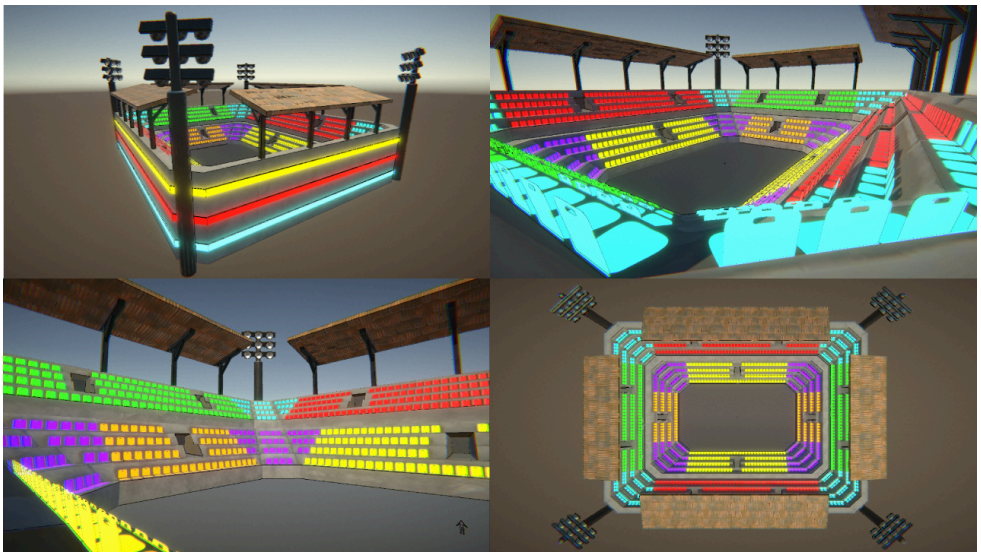
[26] Forcefield Shader Graph tutorial: **Shader Graph Forcefield: Update**

5.2.1.5: Assembly in Unity

The final phase of the production of the Pinball Arena scenario was to assemble everything together within Unity and make it look visually appealing to the player.

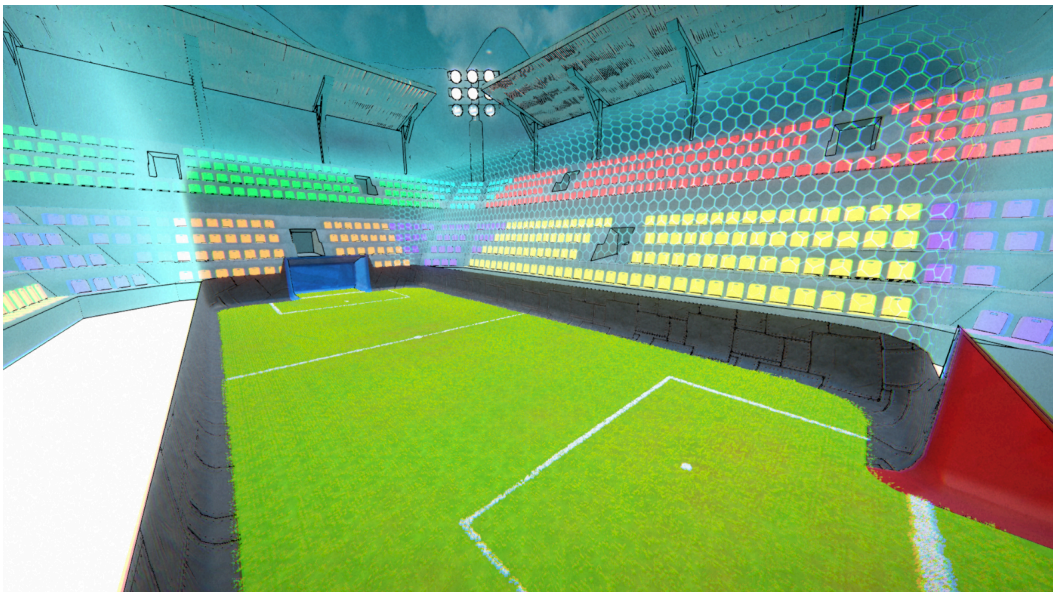
It all started with the setup of the modular kit pieces from the stadium grandstands, as well as the outer stadium lights. After setting them up, the final result looked like this:

Figure 5.2.1.5.1: Pinball Arena Stadium Assembly



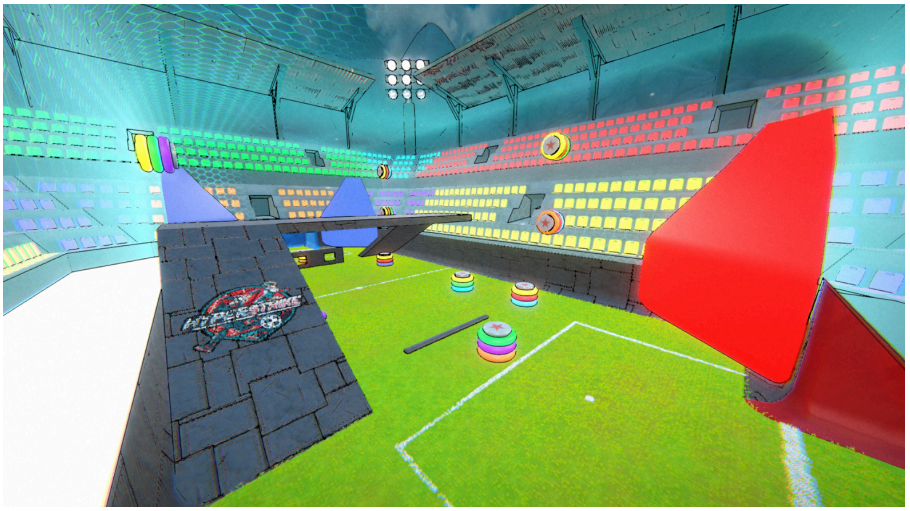
After that, the arena cage was added in the center of the stadium. Together with it, the grass of the pitch was created, thanks to the Unity Terrain Tool package. This tool enables the developer to easily create terrains and later sculpt, paint, and add grass/trees to them.

Figure 5.2.1.5.2: Pinball Arena Cage + Grass Terrain



With that, the only missing assets to be implemented were the elements from inside the cage. These elements are the ones the players directly interact with during the gameplay.

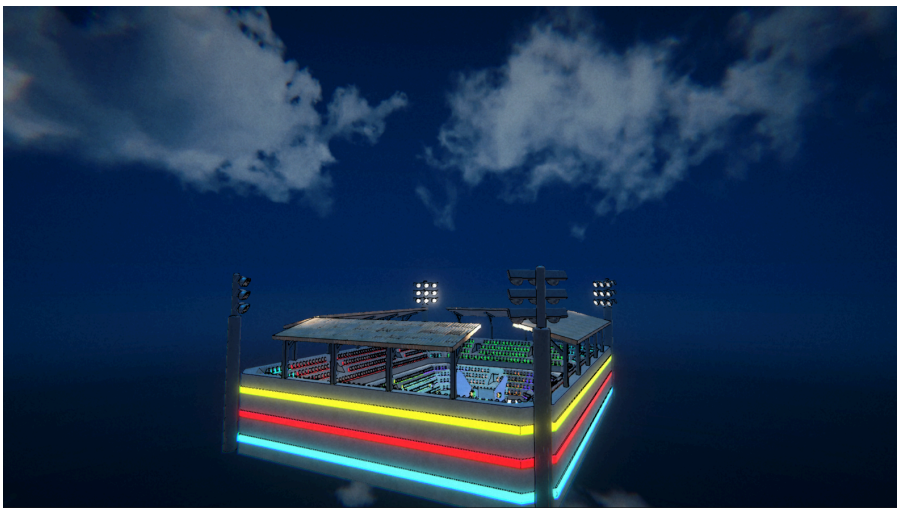
Figure 5.2.1.5.3: Pinball Arena Elements



After placing these elements, it was time to complement the rest of the scenario with different tweaks and settings, trying to give the game's visuals a final touch. To do so, these were the changes:

- **Skybox:** As decided in the level design document (LDD), the Piball Arena's surroundings are intended to be dark, meaning that the game takes place during nighttime. That is the reason why the skybox material was changed to a night-like HDRI map[27]. Due to time constraints, a free-to-use map from the Unity Asset Store was used.

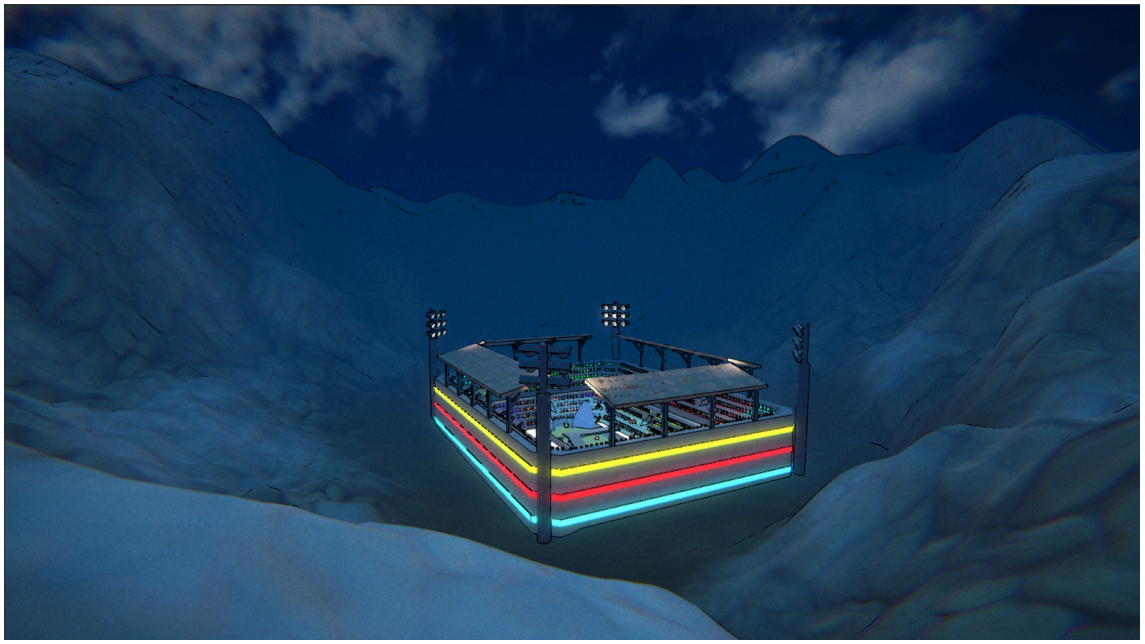
Figure 5.2.1.5.4: Pinball Arena Night Skybox



[27] A High Dynamic Range Image map is a digital image file commonly used in 3D computer graphics for image-based lighting. (www.lightmap.co.uk/blog/whatisanhdrimap/)

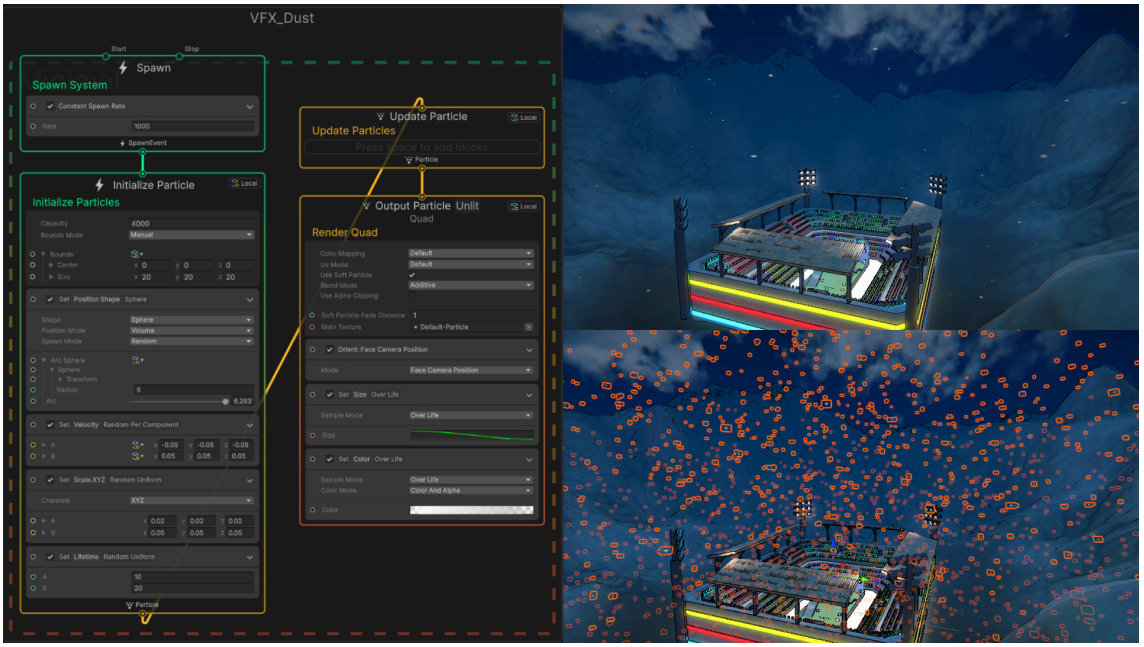
- **Surrounding Terrain:** Another element that was added to the scenario was the terrain that surrounded the whole stadium. It was made using Unity's Terrain Tool with the sculpting and painting functionalities, achieving the effect of having large mountains around the arena.

Figure 5.2.1.5.5: Pinball Arena Surrounding Terrain



- **Ambient VFX Particles:** One of the elements that are normally forgotten, even though they are very powerful, are the particle effects. They can enhance the visuals surprisingly. However, they can overload the scene if they are used a lot.

Figure 5.2.1.5.6: Ambient Dust VFX Particles



- **Stadium Crowd:** In order to make the stadium more alive, a system of spectator crowds was created. To do so, two simple *C#* scripts were created, one that was attached to the meshes of the spectators to create the behavior logic, and the other that acted as a manager of them.
    - *Spectator.cs*:

```CSharp
public class Spectator : MonoBehaviour
{
    private Crowd crowd;
    private float angle;
    private float startingYPos;
    private float yOffset;
    private float randomSpeed;

    void Start()
    {
        crowd = FindAnyObjectByType<Crowd>();

        startingYPos = transform.position.y;
                    randomSpeed = Random.Range(crowd.defaultSpeed - crowd.cheerRandomFactor, crowd.defaultSpeed + crowd.cheerRandomFactor);

        ChooseRandomColor();
    }

    private void FixedUpdate()
    {
        yOffset = startingYPos + crowd.maxHeight;
        angle += crowd.currentSpeedFactor * 0.1f * randomSpeed;

          Vector3 newPos = new Vector3(transform.position.x, yOffset + Mathf.Sin(angle) * crowd.maxHeight, transform.position.z);
        transform.position = newPos;
    }
```

```
    private void ChooseRandomColor()
    {
        Renderer renderer = GetComponent<Renderer>();
        Material newMaterial = renderer.material;

            newMaterial.color = new Color(Random.Range(0, 256) / 255f,
Random.Range(0, 256) / 255f, Random.Range(0, 256) / 255f);

        renderer.material = newMaterial;
    }
}
```

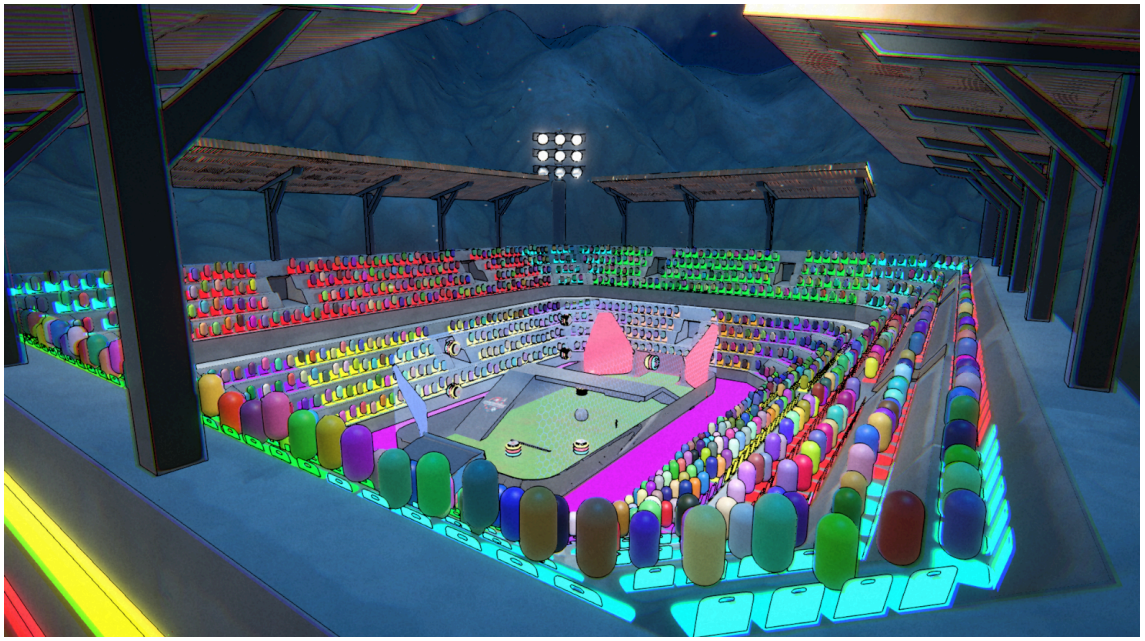
- *Crowd.cs*:

```
CSharp
public class Crowd : MonoBehaviour
{
    [Range(0, 5)] public float defaultSpeed;
    [Range(0, 5)] public float cheeringSpeed;
    [Range(0, 5)] public float superCheeringSpeed;
    [Range(0, 1)] public float cheerRandomFactor;
    [Range(0, 1)] public float maxHeight;
    [HideInInspector] public float currentSpeedFactor;
    public GameObject cheerTrigger;
    public GameObject superCheerTrigger;

    private void Awake()
    {
        currentSpeedFactor = defaultSpeed;
    }

    private void Update()
    {
        if (cheerTrigger.activeSelf)
        {
```

```
            UpdateState("Cheer");
        }
        else if (superCheerTrigger.activeSelf)
        {
            UpdateState("SuperCheer");
        }
        else
        {
            UpdateState("Idle");
        }
    }

    private void UpdateState(string state)
    {
        switch (state)
        {
            case "Idle":
                currentSpeedFactor = defaultSpeed;
                break;

            case "Cheer":
                currentSpeedFactor = cheeringSpeed;
                break;

            case "SuperCheer":
                currentSpeedFactor = superCheeringSpeed;
                break;
        }
    }
}
```

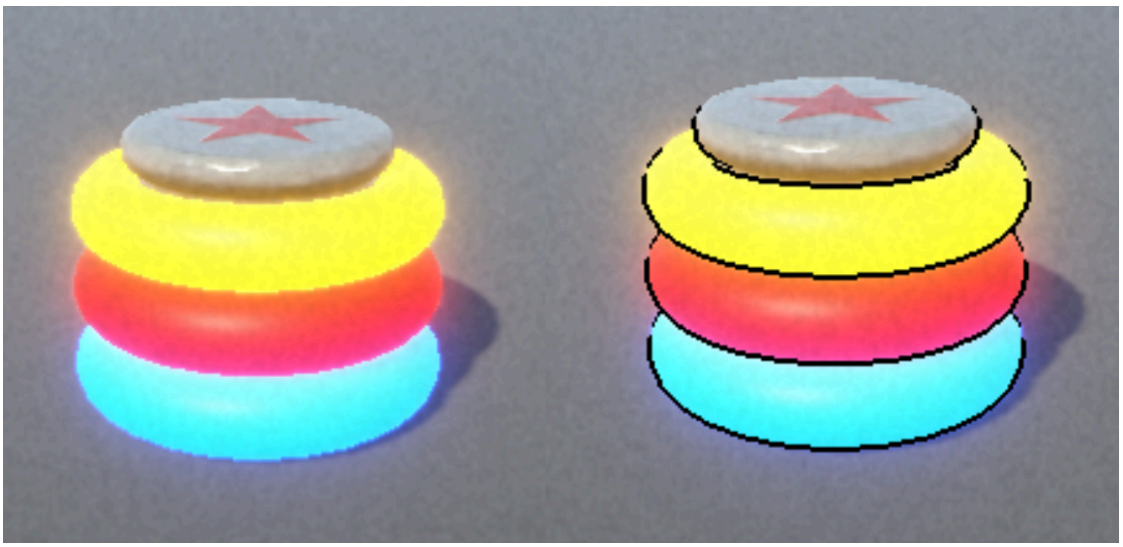
With this crowd system, the stadium could be felt as if it were alive, with simple spectators cheering in different intensities if needed.

Figure 5.2.1.5.7: Stadium Crowd



- **Outline Fullscreen Pass Renderer:** In order to give the assets of the game a more stylized touch, a fullscreen pass renderer shader graph was created. The main purpose of this was to add a black outline to the geometry of the scene, in order to make those elements that are closer to the camera stand out from the background.

Figure 5.2.1.5.8: Comparison Outline Fullscreen Pass Render



- **Post Processing:** Last but not least, to boost all the elements of the game and to enhance even more the visuals of it, some post processing was added to the camera renderer. Within all the different parameters available in the Unity Editor, the ones that were added and stand out the most are:

  - ***Bloom:*** It is used to differentiate elements of high intensity or bright light, such as the emissive materials of the Pinball Arena bouncers.

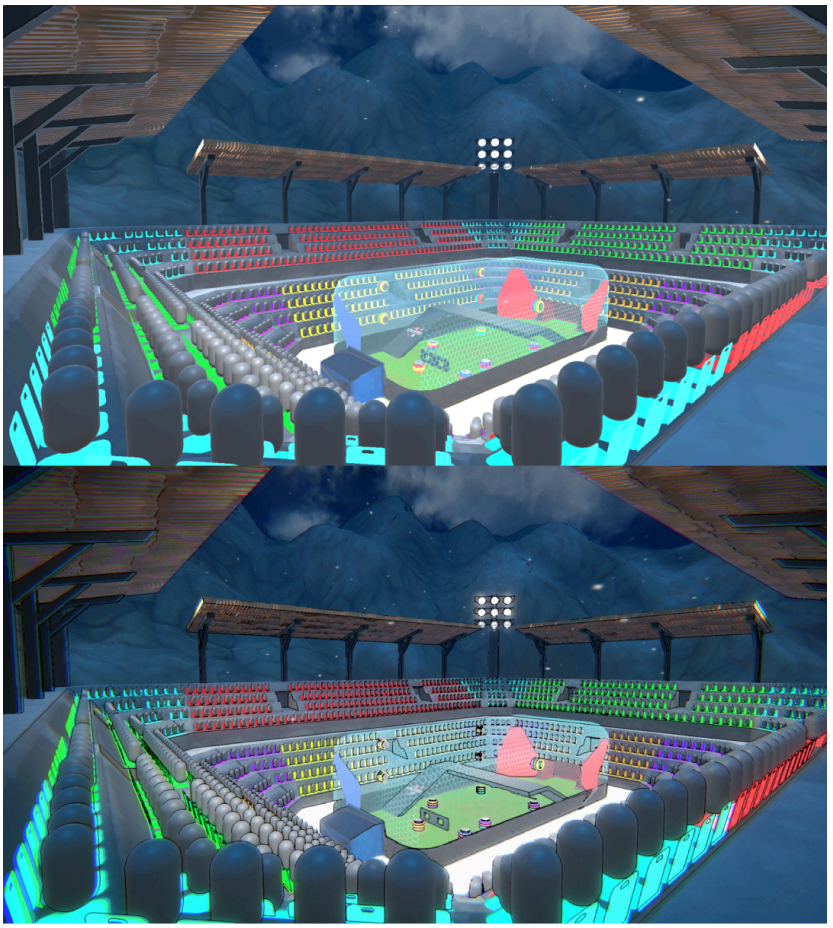
  - ***Chromatic Aberration:*** It is the dispersion of color that appears on an image if the lens of a camera fails to focus all colors to the same convergent point, used to give the sense of having a low-end quality camera.

  - ***Color Adjustments:*** It has been adjusted the post exposure, contrast, and saturation, resulting in more vivid and brighter colors.

  - ***Film Grain:*** It works as a way to simulate camera limitations, as if it was old.

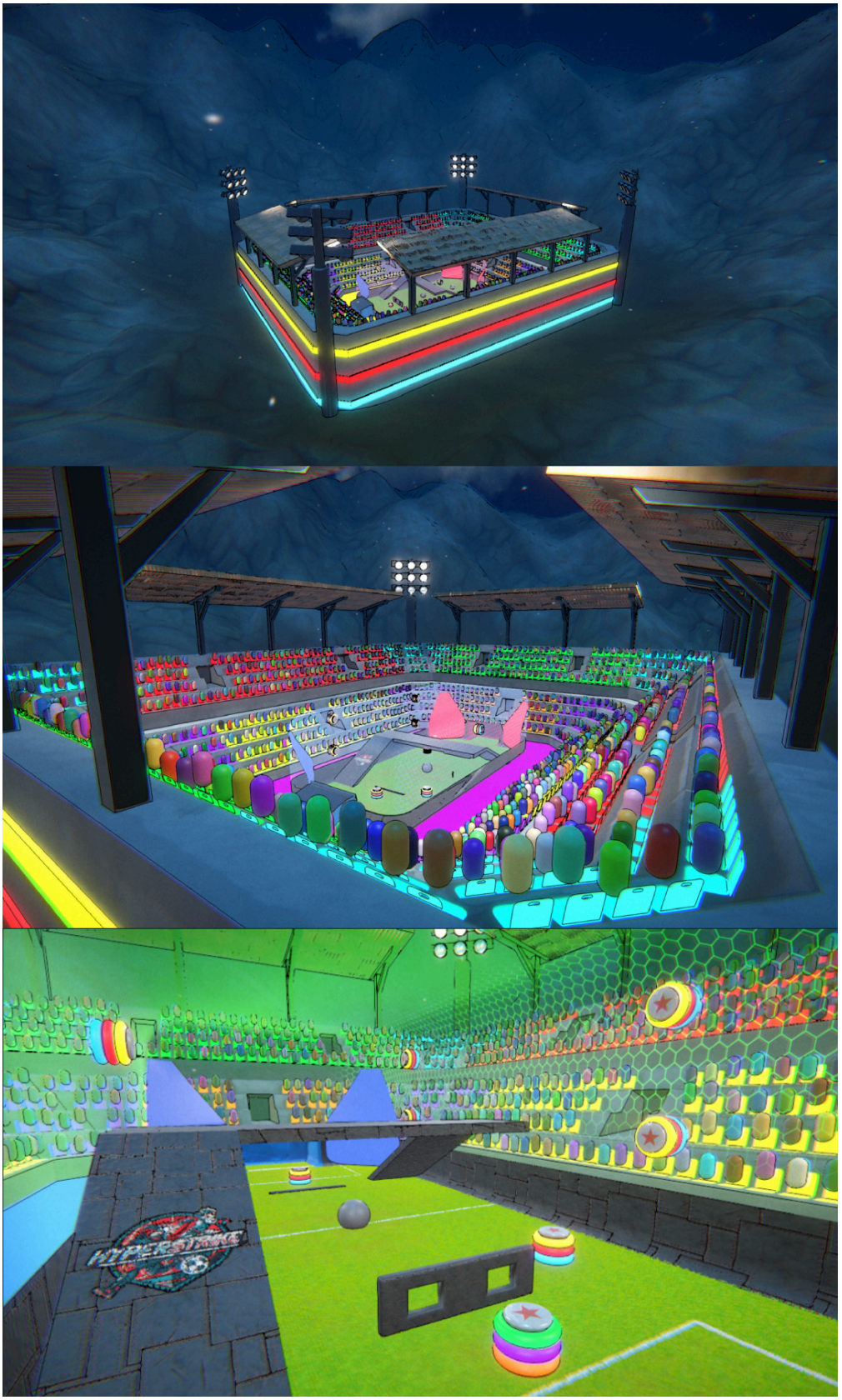
  - ***Vignette:*** It adds the effect of giving the sense of focus to a central point of the camera view, with rounded black color on the outer sides of the viewport.

Figure 5.2.1.5.9: Comparison Post Processing



After finishing the assembly of all the produced assets, the final Pinball Arena scenario had the following look:

Figure 5.2.1.5.10: Pinball Arena Final Environment Art



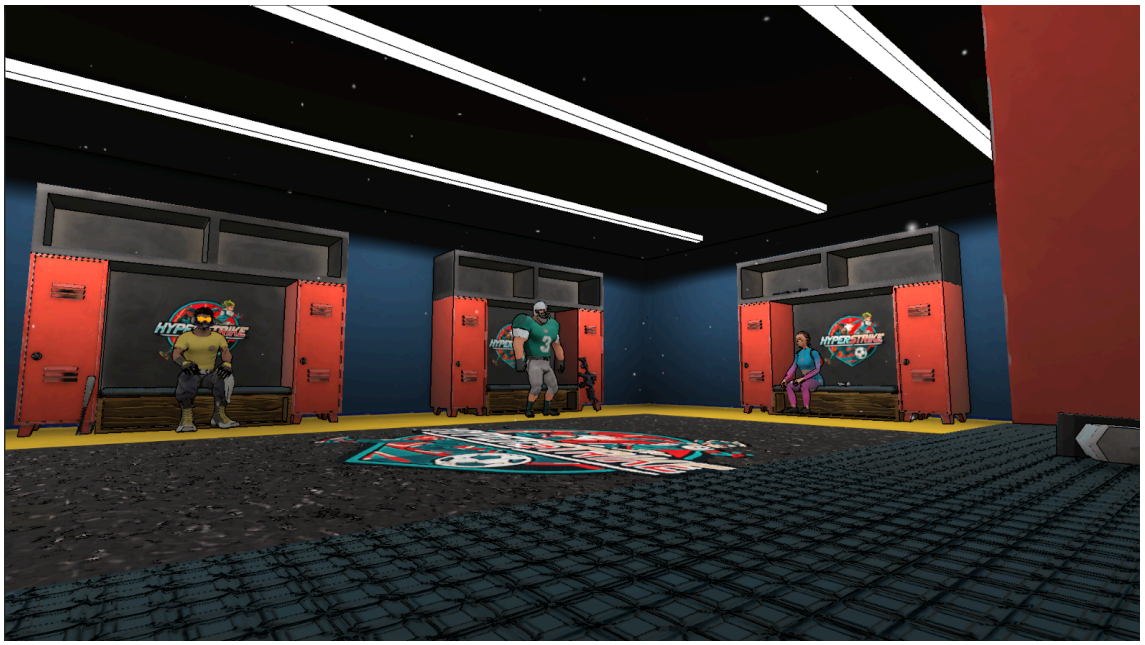
5.2.3: Main Menu (2D + 3D)

The last stage of the production of *HyperStrike* was the creation of the Main Menu. Normally, video games' menus tend to have a unique 2D implementation, meaning that all the elements on screen are two-dimensional images or sprites, such as images, buttons, etc. However, in the case of *HyperStrike,* it was a little different from what is usually seen. The objective was to blend the traditional 2D menu with the 3D space.

5.2.3.1: Scene Setup

The first steps to build the main menu were creating a new scene within Unity. In this scene, it was added the same base layout as in the Lobby Room, but with some minor changes. Firstly, as the design of the layout represented a locker room, in the case of the main menu, it was populated with the three different characters from *HyperStrike*. Each one of them was placed in a different locker in the room. In addition, each character was performing some animations, making them feel more alive.

Figure 5.2.3.1.1: Main Menu Scene Setup

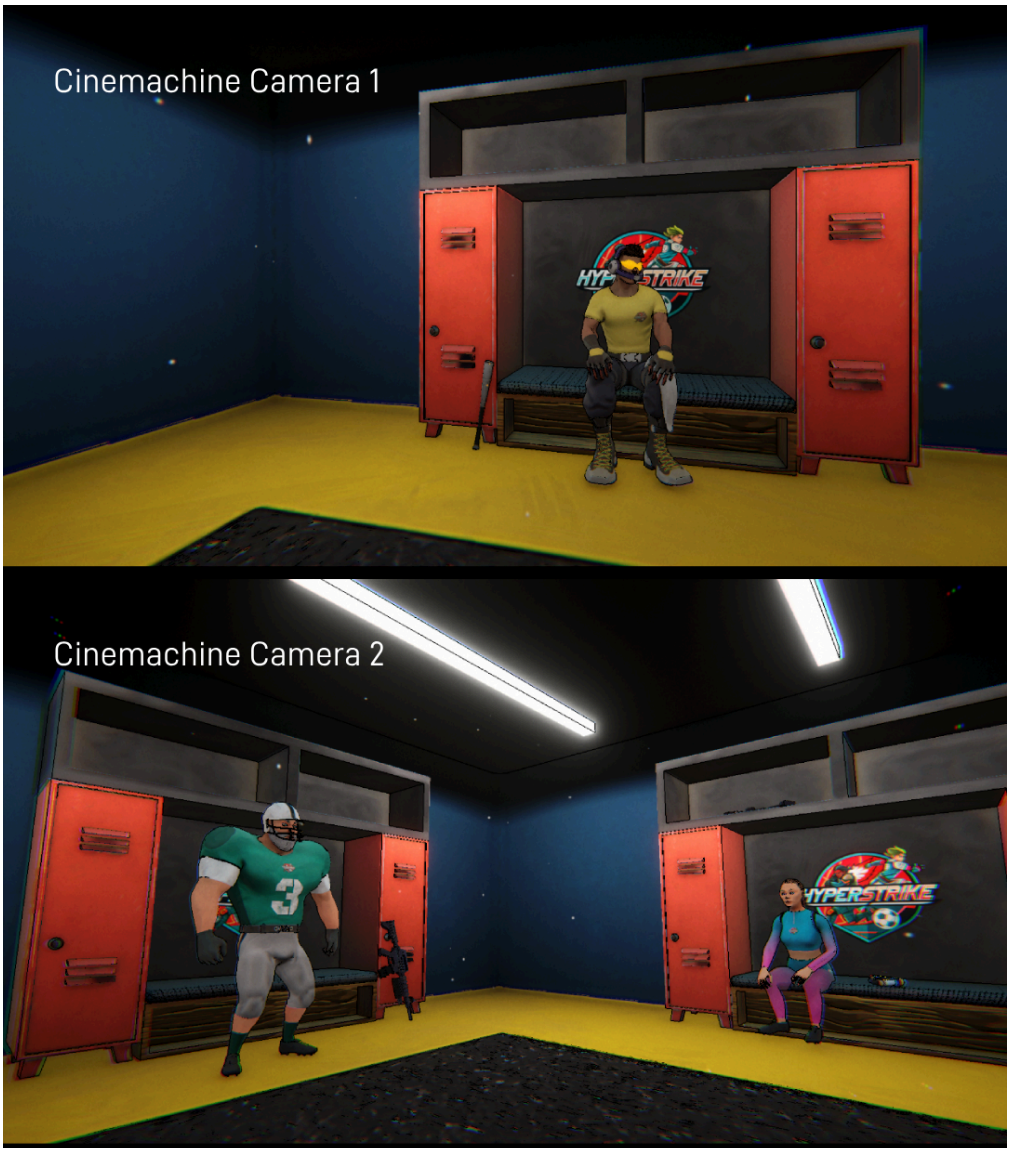


5.2.3.2: Cinemachine Camera Setup

With the scene setup, it was time to create the system of cameras that would react based on the user's actions with the menu element. To do that, the tool used was Unity's Cinemachine camera system. It allows to create a Cinemachine Brain Camera which can control multiple Cinemachine camera instances by enabling/disabling them,

creating animated transitions within different cameras, etc. In this case, two different Cinemachine camera instances were created, each with different shots and angles.

Figure 5.2.3.2.1: Different Cinemachine Cameras

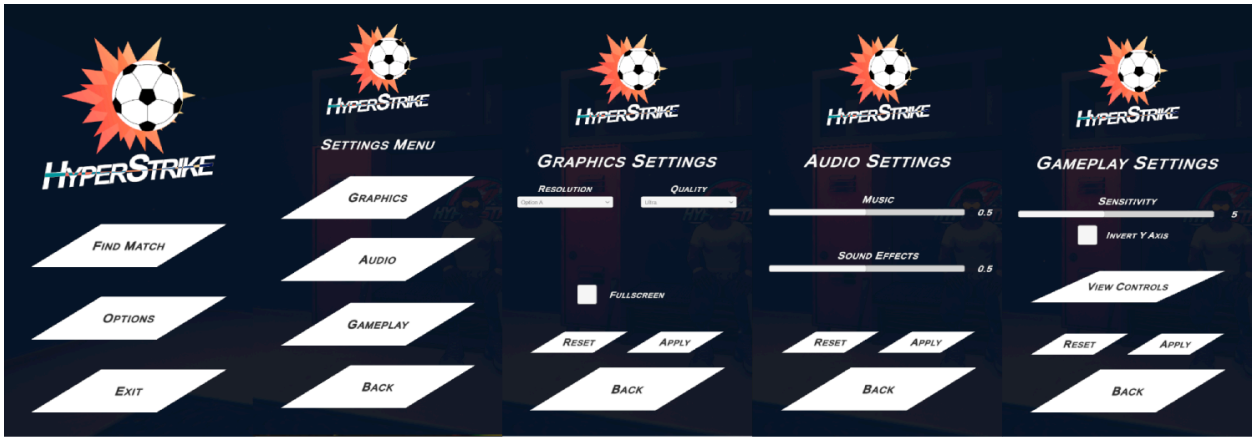


5.2.3.3: Creation of UI Canvas (2D)

As mentioned before, the objective of this main menu was to blend the traditional 2D menu with the 3D space. At this point, it was time to create a 2D menu system using Unity's UI Canvas and TextMeshPro's GUI elements.

The menu was designed to have 5 different screens: *Main Menu*, *Options Menu*, *Graphics Settings Menu*, *Audio Settings Menu*, and *Gameplay Settings Menu*.

Figure 5.2.3.3.1: 2D Menu Screens



5.2.3.4: Final Assembly

With the screens created, together with their functionalities working properly, as well as the navigation between them. It was time for one last thing: creating the logic between the previously set Cinemachine cameras in the scene and the 2D menu screens. The objective here was to make the Cinemachine Camera 1 active when the *Main Menu* screen was enabled, and later change to Cinemachine Camera 2 when any of the *Options Menu* screens were enabled.

To make this happen, Unity's Cinemachine cameras have one property called "Priority". This property is a numeric value, which can be either negative or positive, that tells the Cinemachine Brain Camera to make one camera active or another depending on the value of it. In this case, having a negative value would be translated into making the camera inactive, and conversely, having a positive value means that the camera should be activated.

Taking all this into consideration, in order to swap between cameras, a simple C# script was created:

```CSharp
public class MenuCameraSwitcher : MonoBehaviour
{
    [SerializeField] private GameObject mainMenuPanel;
    [SerializeField] private CinemachineCamera mainMenuCamera;
    [SerializeField] private int activePriority = 5;
```

```
    [SerializeField] private int inactivePriority = -5;

    private void Update()
    {
        if (mainMenuCamera != null && mainMenuPanel != null)
        {
            mainMenuCamera.Priority = mainMenuPanel.activeInHierarchy ?
activePriority : inactivePriority;
        }
    }
}
```

5.2.3.5: Final Result

Once all the elements were put together and working, the final look of *HyperStrike*'s main menu, which tried to blend the traditional 2D menu with the 3D space, was the following: HyperStrike - Main Menu [YouTube]

6. Project Validation

Once the project was finished and a playable build was created. After that, a private playtesting session was created. The objective of this session was to be able to obtain some external feedback from other players, as well as their opinion on what they liked, what could be improved, as well as report the possible bugs that could appear.

After the session was finished, a brief recap and a questionnaire were sent to the players, so they could express their thoughts. The answers have been categorized into three groups: what looked good, what could be improved, and what was missing.

Starting with the first answers, regarding what the players liked overall, these were the most repeated topics:

- "*The menu is cool. I like the fact that it is not static at all, and you can see the characters of the game as well as the locker room where you are later in before the match starts.*"
- "*Bouncers and the way they work are nice. They look integrated with the rest of the elements of the arena.*"
- "*The arena and the different characters are looking great.*"

Then, the answers related to what could be improved are:

- "*The explosion of the projectiles could be bigger, as it would be seen better, and it would make the player more aware of what is happening during the game.*"
- "*Main menu buttons could be a bit smaller in size.*"
- *"It would be great to have some kind of auditory feedback when scoring goals (like an air horn or explosion)."*
- "*I am not able to see my character body (arm, legs, etc), nor the animations.*"

Last, the answers regarding what was missing in the game:

- "*Characters do not feel alive at all when playing the game because of the animations."*
- "*It would be great to add some basic UI in-game in order to know which abilities the characters have and more.*"

- "*I am missing some kind of information of which is the role and stats of each character during the selection before the match.*"
- "*For future work, more arenas could be added and let the players vote which one they want to play before the match starts.*"

With all this feedback and answers, some deductions can be made. Starting with the comments on what looked good, it can be said that the overall visuals and the style of the art from the game were positive. Playtesters appreciated the design of the main menu and how it is connected to the scenario of the locker room in the 3D space. Also, the integration of the arena's bouncers together with the attention to detail in the characters and arena look.

On the other hand, for what could be improved, the majority of answers were focused on the need for polishing how the game feels and looks in terms of in-game character mechanics and their animations. Another comment was the auditory feedback, which, during the playtesting session, the game did not have audio for all elements of the game. However, regarding this comment, it has been able to improve the player experience by implementing sound effects during the in-game matches, creating a more engaging and immersive experience.

Lastly, regarding the elements that the player missed in the game, comments are focused towards receiving more information about what is happening during the game because of the lack of UI elements, especially having information about character abilities, their cooldown timers, what their stats are, etc. It was also suggested the suggestion of doing a poll before every match to vote on which arena the users would be playing in.

With all being analyzed, the overall feedback provided valuable information on what has been done well, and what the next steps on the development of the project. However, if the focus is put on the initially set main objectives of this project, these have been validated positively, as the work behind visuals of both 3D characters and environmental art of the match arena was favourable.

7. Conclusions

Taking a step back and looking at all the work that has been done during the whole production of the project (approximately 6 months), this bachelor's final project has been an incredible learning experience, not only when writing this document, but also when developing the project itself.

Overall, the project has achieved its main and specific objectives, but at the same time, identified areas for improvement for further development. The final result matches what the vision the whole team had for the project, and it turned out to be the way we wanted, except for some limitations due to time constraints, such as more levels, characters, audio, and UI feedback. But taking into account the final product, it can be said that from concept to completion, the game has grown into a functional prototype.

Reflecting on the entire production process, it is clear that some things could have been done differently, starting with the organization and initial schedule of the work to be done, which did not result in being accurate at all. Focusing on technical aspects, even though Unity 6 did serve us to deliver the desired result, trying and learning a new commercial game engine would have been an enriching experience, but to have been able to do that, the production time would have had to be longer than the available one, as some time should have been spent on learning the new tools.

To sum up, developing this project has taught not only the author, but the whole team, plenty about the game development field, more specifically when working in a small team. Even though the number of developers in the team was three (one person for each of the main departments inside an indie studio), the environment and experience working together have been great. We ended up delivering a product very similar to what was planned about a year ago, and it could not have been possible without the harmony and cooperation of all the members working together as a true team.

7.1 Future Lines

Looking ahead in the development of the project, several areas could be worked on to enhance the game. These have been possible to identify thanks to the feedback received during the playtesting. The main objective of them is to improve the overall gameplay experience, increase the engagement when playing, and add more elements to the game mechanics and visual presentation.

Starting with the list, the creation of more arenas should be considered, so there is more diversity in playing environments and not the same level every match. Together with that, some more characters could be added, so at least there are two characters to be chosen for each role.

Another point in which there is a lot of room for improvement is the animations of the characters. As the main purpose of this project was only to create the 3D art of both characters and the environment, currently, the game features free-to-use animations. However, as of future lines, working on creating custom animations for each character's actions should be highly considered.

Lastly, another area which was commented on during playtesting is the fact of working more on the visual effects part, by implementing more particles, explosions, etc. At the same time, the audio and UI feedback should be considered in the list, as it would give players more information about what is happening in the game when playing it and give them a more immersive experience.

8. References & Bibliography

- *Vertical slice*. (n.d.). What Games Are. www.whatgamesare.com/vertical-slice
- Kreisa, M. (2024, July 24). *When to retire what: Guide to the office equipment lifecycle | PDQ*. www.pdq.com/blog/equipment-lifecycle-management-guide
- *Evolución del precio de la luz para el año 2025 - CostEEnergía*. (n.d.). www.costeenergia.es/historico/pvpc-anual
- *Gantt.com*. (n.d.). Gantt.com. www.gantt.com
- Gurnov, A. (2025, February 18). *What is Agile methodology? | Wrike 2025*. Wrike. www.wrike.com/project-management-guide
- Staff, C. (2024, November 21). *What is a niche market? and how to reach one*. Coursera. www.coursera.org/articles/niche-market
- Uke, J. (2024, December 19). *DPS meaning in games | What does DPS stand for?* GameTree. https://gametree.me/gaming-terms/dps/
- *What is a tank in RPG?* (n.d.). Quora. www.quora.com/What-is-a-tank-in-RPG
- Wikipedia contributors. (2024, November 19). *Healer (video games)*. Wikipedia. https://en.wikipedia.org/wiki/Healer_(video_games)
- ArtsPainter. (2023, April 13). *History of 3d Art and How it Came About - Arts Painter*. Arts Painter. www.artspainter.com/blog/history-of-3d-art
- *Calculation of CO2 | EnCon*. (n.d.). https://www.encon.eu/en/calculation-co2
- *CO2 emissions per kWh in Spain - Nowtricity*. (n.d.). https://www.nowtricity.com/country/spain/
- Kilgore, G. (2024, October 3). *Carbon Footprint of a Laptop vs MacBook vs Desktop Computer vs iPhone*. 8 Billion Trees: Carbon Offset Projects & Ecological Footprint Calculators. https://8billiontrees.com/carbon-offsets-credits/carbon-footprint-of-a-laptop/

- *Carbon labeling and measuring carbon impact of products*. (n.d.). https://www.logitech.com/en-eu/sustainability/carbon-labeling-measuring.html
- Intikhab, S. (2023, May 26). What is 3D Digital Art? Digital Art Software and Process. *InvoGames*. https://invogames.com/blog/3d-digital-art/
- Badler, N. (2017, July 27). *3D object modeling*. ResearchGate. https://www.researchgate.net/publication/267561922_3D_Object_Modeling
- *Autodesk Mudbox Online Help: Sculpting overview*. (n.d.). https://download.autodesk.com/us/mudbox/help2010/index.html
- Shahbazi, N. (2024, December 14). 3D Modeling vs. Sculpting - What Are the Differences? *Pixune*. https://pixune.com/blog/3d-modeling-vs-sculpting/
- *Retopology | What is Retopology? | Autodesk*. (n.d.). https://www.autodesk.com/solutions/retopology
- Villanueva, N. (2022). *UV Mapping.* In: *Beginning 3D Game Assets Development Pipeline*. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-7196-4_5
- Tiigimägi, S. (2022, September 28). *Beginners' guide to UV mapping and unwrapping*. 3D Studio. https://3dstudio.co/uv-unwrapping-software/
- Lobo Aránguez, Ángel (2017). *Baking en Blender.* Proyecto Fin de Carrera / Trabajo Fin de Grado, E.T.S.I. y Sistemas de Telecomunicación (UPM) https://oa.upm.es/view/institution/ETSIS=5FTelecomunicacion/, Madrid.
- Taylor, V. (2022, June 21). Difference between realtime, mixed, and baked lighting in Unity. *Medium*. https://vintay.medium.com/difference-between-realtime-mixed-and-baked-lighting-in-unity-6bda1f24bfb

- *What is 3D texturing? | Adobe Substance 3D*. (n.d.). https://www.adobe.com/products/substance3d/discover/3d-texturing.html
- *What is PBR (physically based rendering)? - Adobe Substance 3D*. (n.d.). https://www.adobe.com/products/substance3d/discover/pbr.html
- A23d. (2022, January 6). *Different maps in PBR Textures*. A23D. https://www.a23d.co/blog/different-maps-in-pbr-textures
- A23d. (2023, February 10). *Difference between Albedo and Diffuse map*. A23D. https://www.a23d.co/blog/difference-between-albedo-and-diffuse-map
- *What is a normal in 3D modeling: Understanding the Concept of Normals in 3D Space*. (n.d.). www.coohom.com/article/what-is-a-normal-in-3d-modeling?hl=es
- Aximmetry. (2023, November 15). *PBR materials*. aximmetry.com. https://aximmetry.com/learn/virtual-production-workflow/obtaining-graphics-and-virtual-assets/creating-content-in-aximmetry-se/pbr-materials/
- McREYNOLDS, T., & Blythe, D. (2005). Lighting techniques. In *Elsevier eBooks* (pp. 317–359). https://doi.org/10.1016/b978-155860659-3.50017-2
- Ortiz, R. (2024, 14 agosto). What Is 3D Rendering? Complete Guide to 3D Visualization. *Chaos*. https://www.chaos.com/blog/what-is-3d-rendering-guide-to-3d-visualization
- Aminian, P. (2024, August 6). Stylized Visionaries: artistic interpretations of games. *Pixune*. https://pixune.com/blog/stylized-art-style/
- Gillis, A. S., & Lewis, S. (2024, June 14). *Object-oriented programming (OOP)*. Search App Architecture. https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

- *Recommended Asset Naming Conventions in Unreal Engine projects | Unreal Engine 5.5 Documentation | Epic Developer Community*. (n.d.). Epic Games Developer. https://dev.epicgames.com/documentation/en-us/unreal-engine/recommended-asset-naming-conventions-in-unreal-engine-projects
- *Asset Naming Conventions - A guide to best practices*. (n.d.). Zoidii. https://zoidii.com/blogpost/asset-naming-convention
- *Best practices for organizing your Unity project | Unity*. (n.d.). Unity. https://unity.com/how-to/organizing-your-project
- *First-Person Shooter*. (n.d.). eloking.com/glossary/general/first-person-shooter
- Bermanseder, T. (2019, July 19). *Technical and visual analysis of Overwatch*. 80Level. https://80.lv/articles/overwatch-technical-overview/
- Wiki, C. T. R. L. (n.d.). *Arenas*. Rocket League Wiki. https://rocketleague.fandom.com/wiki/Category:Arenas
- *Minionsart Tutorials*. (s. f.). https://minionsart.github.io/tutorials/
- Rubberduck. (2019, April 13). *Handpainted rock texture*. OpenGameArt.org. https://opengameart.org/content/handpainted-rock-texture
- *Pinball Sharpshooter Bumper FienUp 001 | SoundsNap*. (s. f.). https://www.soundsnap.com/pinball_sharpshooter_bumper_fienup_001
- Ramirez, M. (2022, June 14). Overwatch DPS Tier List: June 2022. *dbltap.com*. https://www.dbltap.com/posts/overwatch-dps-tier-list-june-2022-01g5hnqdjg9d
- Technologies, U. (s. f.). *Unity - Manual: Understanding post-processing*. https://docs.unity3d.com/2019.3/Documentation/Manual/BestPracticeMakingBelievableVisuals8.html

- CGSHOWCASE. (2020, September 14). *BEARD AND HAIR SCULPTING IN ZBRUSH | Hair Speed Sculpt* [Vídeo]. YouTube. https://www.youtube.com/watch?v=XE_ta4nE59w
- *Female tracksuit Marvelous Designer project | 3D model*. (s. f.). CGTrader. https://www.cgtrader.com/3d-models/character/clothing/3-piece-activewear-ensemble
- Ltd, D. (2023, February 23). *Medkit Pack 3D*. 3DExport. https://es.3dexport.com/3dmodel-medkit-pack-435268.htm
- *Win in the Video Game*. Enrico_Dering (s.f.). Pixabay https://pixabay.com/music/video-games-win-in-the-video-game-191820/
- *Air Horn*. SoundReality (s.f.). Pixabay https://pixabay.com/sound-effects/air-horn-186076/
- *Action rock cinematic trailer music*. LightStockMusic (s.f.). Pixabay https://pixabay.com/music/action-action-rock-cinematic-trailer-music-353156/
- Scribbr. (2025b, February 3). *Formato APA con el Generador APA de Scribbr*. https://www.scribbr.es/citar/generador/apa/

9. Annexes

- *Speed* Character Presentation. ArtStation - Speed 3D Character [Game-Ready]