UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

3D Character Editor

Bachelor's Thesis

Video Game Design and Development Degree

Surnames: Lakhwani Daswani **Name:** Manav

Scheme: 2014

Director: Ripoll Tarré, Marc

Index

Summary

When it comes to introducing new characters in your own designed videogame, it can be tricky to find that 3D model that fits the character as you want when searching for online assets. Specially in Indie Games, where their budget is way lower, some can't afford to hire an entire art team or spend that much time to work in the character models (making sure it fits the artistic style, represents all the elements the design team intended for the character, correct structure and amount of polygons to run in a videogame and such) and if the models are gotten online, it is very hard to find several models for all the different characters in the game that match the same artistic style and represent what designers want to in these esthetically.
It is for this reason that, to improve this situation, this thesis is dedicated to a research and solution of **character customization**, where developers with a low budget to make a game, would only have to download an app/software with a base character fully editable and with the ability to download the edited 3D model for its own game.
This app/software would include the base mesh, choosing between male or female, with the correct topology and number of polygons to run in console or PC. This base mesh will be fully editable to replicate any type of human character the designer thought of, as it would include customization inside the own app/software of facial features, hair length and style, general body types such as muscular or thin, height and weight, skin shade and general types clothing controlled by scrollers for a more accurate result. This edited version of the mesh can be exported as an .fbx (3D model) to introduce directly in the game or to add some final touches.
To create this app/software, the base model joined with the hair and clothing will be modelled and sculpted in ZBrush and textured in Substance Painter or searched online making sure its topology is correct and matches the idea and style. For the app/software itself, Unity will be the game engine used, where all the scrolls for the customization will be programmed inside Unity using blend shapes from the original model created in Maya.

This document will focus on the whole process of the prior investigation on how all this is possible to create, concepts of the app/software and artistic style of the models, the whole implementation step by step of the creation of the models and the app/software till showing final results.

Keywords

Character edition, blend shapes, 3D model, Unity, polygons, customization, topology, human features, export, videogames.

Links

Link to the latest build releases:

https://github.com/manavld/CharacterEditor/releases

Link to the trailer/tutorial:

https://youtu.be/6f47Yuog1m4

Link to all the texture files:

https://drive.google.com/drive/folders/12w4PUQDtfKb6dRQpHG4sVKMMa70wlxYP?usp=sharing

Tables Index

Figures Index

Glossary

Art Style: Definition of the type of artwork done, normally maintained in the whole game to not lose player's engagement (ex: realistic, cartoon).

Blend Shapes: The implementation of a technique that allows a single mesh to deform to achieve several predefined shapes and any combinations between them.

C#: Programming language used in Unity.

Facial Features: All the different elements that combine the structure of the human face.

Indie: A small independent team composed by one or few people that produce videogames with a very low budget.

Mesh / 3D Model: Representation of an object or living being by a combination of rectangles, triangles and vertices to create 3D surfaces.

Optimization: In terms of gaming, the process of making it the most effective to run with no issues such as frame drop and memory loss.

Pipeline: The step by step workflow to create a full 3D model.

Polycount: Total amount of polygons in a 3D model.

Rigging: The process of creating and attaching a skeleton to the 3D model.

Sliders: A button that is dragged along only horizontally or vertically to adjust a value more accurately.

Software: Computer programs.

Topology: The way polygons and vertices in a 3D model are arranged and distributed.

Triple A: Classification of videogames that are produced and distributed by a major publisher, and so having the highest quality in the market with typically higher development and marketing budgets.

Unity: Cross-platform game engine.

1. Introduction

1.1 Motivation

The **main motivation** I have had to do this project is to enlarge my knowledge in videogames involving 3D character models. Characters in videogames have a whole design investigation behind it and portraying all those ideas into the model itself has always been a challenge for me, this type of research project is going to help me understand and learn all the concepts to be able to create 3D characters the exact way the designers thought.

When playing videogames, something I have always spent way more time into than anyone should is your own character customization inside the game itself, as it could immerse the player into the videogame way more by relating himself to the main character through the similarity of the 3D model's looks.
I have always debated whether if I liked more the programming or the artistic side of videogames, and this more technical art project is a good way to combine both.

1.2 Problem Formulation

Something I have encountered in multiple occasions when developing videogames in university projects is the lack of correctly made free 3D models with the same artistic style around the internet. When students are developing a game for a programming project for example, they don't have the time to create a bunch of 3D characters for their game and have to search it online, but it is very hard to find characters that have a similar style and end up looking out of place in the final game. The same situation is encountered for indie games who don't have the budget or the time to dedicate to all of the 3D characters.
This might not be an issue for triple A companies that have a whole art team working in a single character, but indie teams and students face the **main problems** of: searching for the correct models online, character model not matching the art style of the rest of the game or other secondary characters, the number of polygons being way too high or way too low for a PC/console game, models being expensive.

There exists a huge amount of free and payable 3D character models online already, there even is a multiple amount of websites that are specifically dedicated to downloadable models such as sketchfab, artstation, cgtrader, free3D, turbosquid and more. In these websites you can find a lot of 3D human males and females models, but these are created how the artist who posted it wanted to, with the polygons, style and human features they decided, and you could look for ages and still not find the exact type of model you want for the game. Especially when wanting to introduce multiple characters, finding free similar models (in terms of art style, number of polygons and topology) from different creators that match your ideas (body type, facial features,

hair, clothing) is close to impossible to find.
In the case of indie companies, their solution to this is normally paying a freelance artist, where basically they pay the artist to create the model exactly how they want it just that one time. This would still involve investing an economic amount and is not an easy solution, especially for students.

There are some better **solutions** to this problem but that still have some missing features, like free packages inside game engines such as Unity that includes multiple 3D character models with the same style, polygons, topology and different human features/clothing but these still might not be the idea the designers had as it can't be customized.
A customizable 3D character software for designers does already exist, solving the main problem encountered, but this software by *Reallusion* called *Character Creator 3* is extremely expensive for these indie companies and students as it is meant to be for triple A companies.

Which is why the **problem** this thesis presents is the lack of a free fully customizable 3D character models for the development of videogames that maintains the same art style and has an optimized number of polygons and topology to run in main game engines, to assure an easier work for students and indie game companies.

1.3 General Objectives

The **main objective** would be to create a free PC software with a main base 3D human model with a realistic art style, where anyone could edit and customize all this model's features in real time to replicate their idea they designed and envisioned. For this reason the general objectives of this project will be to:

- Investigate all the possibilities to replicate this customizable character like using blend shapes or by a skeleton.
- Concept art of both the human base designs and the software UI.
- Search online two realistic 3D base models (male and female) all modelled, sculpted, painted and rigged to be fully optimized for PC/console videogames in polycount and topology to be configurable in real time in Unity.
- Create and configure a UI to modify the 3D base model in real time to emulate any type of person, videogame character or any design ideas thought of.
- Export the edited version of the character (3D model as an .fbx choosable if rigged or not rigged and all the maps such as normal, diffuse, displacement).

1.4 Specific Objectives

More **specific objectives** involve:

- Investigate and decide the number of polygons both male and female meshes will have, to be fully optimized for any game engine.
- Investigate and analyze all the features and anatomy that are composed in a human body to understand which features should be the most customizable to recreate any type of person.
- Decide and create a concept of a UI system that is simple and easy to understand for designers.
- 3D character model:
  The priority is to get the model online, but even base meshes are hard to find with a correct polycount, topology and fully textured so, if time permits it, there should be a development process for the creation of the models. When it comes to creating the base model it should follow a pipeline/workflow that consists in:
  - Concept art: the character drawn in 2D colored with its features defined.
  - Blueprints: front, side, back and top 2D line drawings with the correct sizes and measurements to use as a base when creating the 3D version.
  - Blockout: Create the 3D model with basic geometric shapes like spheres and cubes matching it with all the blueprints.
  - Low Poly: Using the blockout, create the 3D model with a correct topology for animating a character and with the number of polygons established as this is the mesh that will be used in the game and software.
  - High Poly: Subdividing this low poly into approximately a million polygons to define every little detail by sculpting and making the final result.
  - Texturing: Generate a UV map for the low poly, paint the high poly and bake meshes to export all the normal, diffuse and displacement maps.
  - Rigging: Create the skeleton for the mesh to animate the character.
- Creation of different type of 3D clothing for both male and female using the same pipeline except for rigging.
- Investigation of an optimized method for 3D hair specified for videogames and creation of different hair types and length for male and female.
- Investigation and execution on how to deform the base mesh in real time using blend shapes or skeleton to replicate every type of facial features, body type, hair/beard type and length, clothing and color (skin, hair, clothing) in Unity.
- Investigation and execution of exporting 3D meshes inside Unity joint with all maps and rig.

1.5 Project Range

This project can end up having a very big range as the it has a lot of potential to include more customizable options. It is for that reason that the **limits** set for this project are to be able to edit facial features (shape and length of nose, eyes, eyebrows, cheek, mouth, chin, face shape and ears), hair/beard (length, type, color), body type (thin, muscular, weight, height), clothing (different types of clothes and color) and color (eyes, skin tone).

This software is **aimed** for developers who can't afford to spend money on a freelance artist, art team or payable models online but that still want to be able to use an own character model with their own idea and/or don't have the time or knowledge to create their 3D model from scratch.
This is why designers or programmers would be able to **use** it without needing to have the knowledge of the whole 3D pipeline, as the software is meant to be able to use even for people who aren't artists. Designers would ideally use this as they are the ones who had the original idea for the character and could customize it exactly how they envisioned it.
Students and indie game companies would **benefit** from this software the most as their main issue is not being able to pay for a custom model or a whole art team. Any type of company, like a triple A, could also use it as testing while the main model is still in production. This software would help to keep that same style between characters and they would be able to add free environmental props that still match the same style (as realistic art style assets are the most available for free).

2. State of the art

When it comes to creating this software, there are a lot of things to have in mind and a lot of research on what technologies can be used to be able to complete it.
For instance, as this is dedicated to videogame development, the model and the extra clothing/hair need to be completely optimized for general game engines and should have a strict number of polygons.
In terms of customization, research of human features and anatomy have to be made in order to understand what esthetic elements defines a person to be able to make those customizable in real time. To maintain the number of polygons and the same 3d model, this would be performed by mesh deformation and there are some existing technologies that allows mesh deformation that can already be seen inside certain games.

2.1 Optimization

Developing a videogame involves not only designing, creating art and programming, but it also involves a lot of optimization for the game to run smoothly. This means that when a game is not optimized for some reason, such as code complexity, physics usage or the gaming device of use, it can damage the performance of the game. Having a **low performance** in a videogame would result in a low frame rate issues, memory loss, lag, and aspects that affect the visual and gameplay experience of the player in a negative way.
This is never what a developer wants as it ruins the experience of the player, which is why every company tries to optimize their games to the fullest. There are a lot of elements in videogame development that can be badly optimized, and the **number of polygons** of 3D models is one of them, which is why, in order for this software to be usable for videogames, it has to be correctly optimized to not affect any performance issues.
To understand the reason why 3D models can damage the performance we have to know that game engines draw each polygon of all the meshes in the screen in every single frame. When some models or the total of them consist on a very high amount of polygons, the rest of the game functions will have to wait for the models to fully load. This works with draw calls, which is the way polygons data are read and drawn in screen, and until this call ends, none of the other calls (like movement, animation) can start, which is why having to wait every frame for very high polygons to load and draw would affect in skipping and dropping frames and lag experience for players.

There are ways already integrated that attempt to solve graphic problems like these, such as **camera culling**, that detect the models that appear on the sight of the main camera and only draw those models, ignoring all the meshes that are out of the camera range. The main optimization for artists when creating a 3D model is to have a low poly mesh with the correct **polycount** and bake it with a higher polycount version

containing all the details (high poly), providing a 2D texture called normal map that can be applied inside the game engine to add those details to the low poly mesh.

But even with solutions like these, a low poly model with too many polygons could still affect in performance issues, which is why a **polycount budget** (maximum number of polygons for the full character including clothing and hair) is needed to set for the main game engines to load and drawn every frame without any frame issues knowing that other assets could be included in the scene.
High-end AAA companies can afford to have 3D character models of over 50k polygons for PC games as they normally have their own game engines and a whole programming team to optimize everything, but for game engines like Unity or CryEngine, when its dedicated to PC or console games, the ideal polycount would be around **10k to 20k** for just the base character model as the quality is still wanted to be preserved, and to not surpass **30k** with all the clothing and hair. This polycount would not delay the draw calls for games ran in PC or console (PS4/Xbox) and would ensure a good optimization in terms of graphics even when adding more meshes to the scene.

Unreal Engine is currently working on releasing its 5th version with a technology called *nanite*, which allows artists to not worry about polycount as it can contain millions and millions of polygons in all 3D models without any performance issues in runtime, aimed for next-gen console PS5.
As assured in Unreal Engine's website about its 5th edition preview:
"Nanite virtualized micropolygon geometry frees artists to create as much geometric detail as the eye can see. Nanite virtualized geometry means that film-quality source art comprising hundreds of millions or billions of polygons can be imported directly into Unreal Engine—anything from ZBrush sculpts to photogrammetry scans to CAD data—and it just works. Nanite geometry is streamed and scaled in real time so there are no more polygon count budgets, polygon memory budgets, or draw count budgets; there is no need to bake details to normal maps or manually author LODs; and there is no loss in quality."

2.2 Human anatomy and features

To understand what human elements and body parts are the ones that define the differences in people and characters, a study of the human anatomy and facial features is needed.

A psychologist, William Herbert Sheldon, developed a system called *somatotype* that classifies the different **types of body** shapes in all extremes. With Sheldon's system, it is possible to categorize any human being into a combination of 3 types of body builds. The 3 types of body shapes that marks the extremes are:

- **Endomorphic**: physically tending to a round build with body fat, round head, large and round abdomen, fat around upper arms and thighs with short arms and legs and thin wrists and ankles.
- **Mesomorphic**: muscular type, square body build, big head, wide chest and shoulders, very muscular arms and legs with low body fat.
- **Ectomorphic**: linear body build, thin face, big forehead, sharp chin, narrow chest and shoulders, long thin arms and legs with little muscle and little body fat.

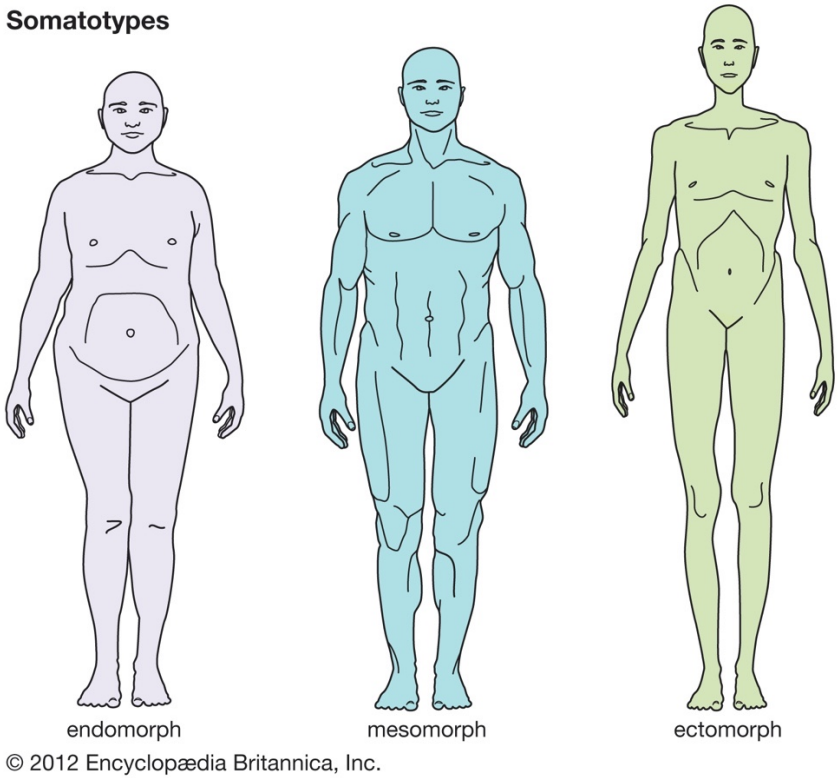

Fig 1. Somatotypes

A genetic reproductive company, California Cryobank, uses 8 different **facial features** to categorize human beings in which each of them have different elements like shape, size or positioning to identify:

- **Face**: shape (round, oval, square), forehead position (low/high), forehead size (narrow/wide), cheekbone position (low/high) and unique features (dimples, facial hair, freckles).
- **Eyes**: shape (round/almond), size (small/large), position (narrow/wide), lashes (short/long) and eyebrows (thin/thick).
- **Nose**: front shape (straight, round, wide), profile shape (straight, round, bump), width and length.
- **Ears**: size (small/large), lobes (attached, detached) and distance from head (close/far).
- **Mouth**: upper lip (thin/thick), lower lip (thin/thick).
- **Teeth**: size (small/large), front tooth gap.

- **Chin**: shape (square, round, point), prominence (slight/strong).
- **Hair**: texture (straight, wavy, curly), volume (thin/thick), color.

Using these two classifications between body types and facial features, it is possible to **replicate** any type of human being, which is why the software will have to let the user edit all these elements in order to correctly portray his original idea into the 3D model.

2.3 3D Modelling

The **art style** decision is made by the availability of free 3D environmental assets in the internet, as it is meant to fit the same artistic style for a whole game where designers can get all the models for free. Currently, in websites that have these free environmental assets like sketchfab, artstation, cgtrader, free3D or turbosquid, the general art style for these free 3D models are a realistic art style. This is why the software will be made for realistic art style games, as the realistic free 3D character editor will be easy to match with free assets found online for students or indie companies.

A base 3D character model has to be created or searched in order to start the customization. This base model/clothing/hair will need an optimal polycount and a good topology based for animating as it is made for videogame development. For this reason a **pipeline** (workflow to follow) should be made considering all the possibilities with the current technologies and softwares available:

- **Concept art**: Character drawn in 2D colored with its features defined. Photoshop is the main software professionals use when creating concept arts as it provides a huge amount of features to create any type of painting. These will start by being sketches of the idea of the model wanted to be modelled.

- **Blueprints**: Front, side, back and top 2D line drawings with the correct sizes and measurements to use as a base when creating the 3D version. Photoshop still is the main software to use in this case, but as it is a more simple type of drawing and focused in the correct volumes and lines, a software dedicated to blueprints like AutoCAD could be useful although it is more focused on structures like houses.

- **Blockout**: Create the 3D model with basic geometric shapes like spheres and cubes matching it with all the blueprints. There are a lot of 3D modelling softwares with very similar styles and very similar tools where you can get the same results on. The best softwares for 3D modelling currently are Autodesk Maya, Autodesk 3Ds Max, Blender and ZBrush. Generally, the usage of these is more opinion based as the tools in each software give you the ability to create the same results, although some have different procedures into doing so. 3Ds

Max includes a more technical perspective that can be more helpful in some aspects, which makes it a less appealing for a more artistic user. However, ZBrush is a more free and artistic version of a 3D modelling software, which makes it harder to understand at first, but many agree that this software had a better depth to it with more variation of tools, freedom and control of brushes.

- **Low Poly**: Using the blockout, create the 3D model with a correct topology for animating a character and with the number of polygons established as this is the mesh that will be used in the game and software. When talking about topology, it Is meant as all polygons being 4 sided if possible, joined correctly and the way the polygons are placed in order make the character animations more real. This involves in making loops and rings of polygons in certain areas that are more predominant to fold or move in animations. The topology of character's faces is the strictest, as it needs multiple polygon loops and rings around the eyes and mouth.

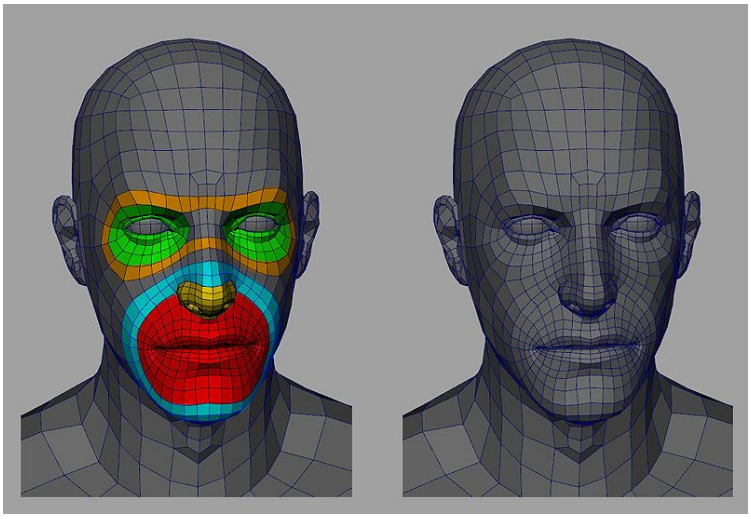

Fig 2. Facial Topology

- **High Poly**: Subdividing this low poly into approximately a million polygons, define every little detail by sculpting and making the final result. It is needed to have a bigger number of polygons to add the details and to paint the model. ZBrush is the main software companies use for sculpting as it involved the large use of different brushes, making it easier to add details and in a more visual way.

- **Texturing**: Generate a UV map for the low poly, paint the high poly and bake meshes to export all the normal, diffuse and displacement maps.
  A UV map is portraying all the polygons of the model in a 2D version. Currently, these softwares do have an automatic UV map generator but it can sometimes not work correctly. ZBrush included a way to paint zones where you want the mesh to separate in order for form the UV map, which helps the process and reduces the error.
  Painting is generally made in Substance Painter, a software to import models,

bake them to generate normal maps and paint them with extensive materials, brushes and even importing other materials from online stores. It also allows to export all types of maps for a much more accurate representation in-game. Other softwares like ZBrush aren't as extensive and can only export a minimum amount of maps, but this is because Substance Painter is fully dedicated in painting and rendering.

- **Rigging**: Create the skeleton for the mesh to animate the character. All the 3D modelling softwares contain a rigging tool, although Maya has a very consistent automatic rig tool called Quick Rig that generates the full skeleton for human models. This tool lacks the finger bones but they can still be added manually. Other automatic rigging tools include a software called Mixamo, that can be used directly in its website and facilitates all the rigging process by letting you upload your model and with just manually setting some points will create a full accurate skeleton. In addition, inside Mixamo, there are professional animations already created that can be applied in the same place after generating the skeleton and exporting it all to your computer.

Generating realistic 3D hair meshes is one of the main issues in videogames as it usually tends to have some problems such as performance, disappearance of parts of hair, transparent spots or visually unpleasing.
Hair generators such as fibermesh in ZBrush and XGen in Maya are tools to create realistic mesh hair in a set zone of an existing model. These are focused more for an accurate representation of real life, aimed to be rendered for film and animation where polycount is not as strict as it is not running real-time. These hair meshes are composed by a very high amount of polygons and are combined with alphas to allow transparency on the hair tips for a more realistic result. These tools also allow you to groom the hair, set its length, consistency, change the color of them and more.

As these polycounts are not optimized for in-game real-time, there are some existing workaround solutions.
The more optimized solution is to create singular strands of hair one by one, composed with just a few polygons, and to add as many as it needs for a realistic result without surpassing the polycount budget set. This solution allows you to have full control of the number of polygons but does require multiple days and weeks of work and a very advanced modelling and sculpting skill set.
There are ways to optimize the hair generators for real-time renders, they all have a similar method into doing so. The way both fibermesh, XGen and other hair generators can be optimized is generally focused on the textures used. The method involves using rectangular strands of hair called **hair cards** with a low polycount, and applying textures of very detailed versions of multiple strands of hair. These textures consist in giving the rectangles multiple strands of realistic hair by using maps affecting color,

light reflection and most importantly, alphas, which allow transparency making the end of the strands look pointy.

This is an example of a senior groom artist at *Airship Images*, *Jansen Turk*, that made a post of his implementation of XGen hair optimized for real-time games, where the different hair cards and textures can be noticed:

Fig 3. Jansen Turk's Game Hair Implementation

Fig 4. Jansen Turk's Game Hair Render

2.4 Customization

Character customizations can be performed in 2 different ways, by using multiple meshes and choosing from them or by deforming just that one mesh into different versions of it.

Multiple meshes would involve in the creation of a certain amount of 3D models following the pipeline, which would take a few amount of weeks to finalize in order to have a decent variety to choose from, depending on the complexity of the mesh. These could be useful for types of clothing or hair as they are unique and with less complexity as a whole human 3D model.

Mesh deformation, on the other hand, maintains the original model and applies controlled changes to it. This is a good way to ensure that the polycount is still going to be correct as it is the same model. Meshes are composed by polygons set by vertices that mark their limit points, mesh deformation is the result of moving these vertices around, and thus maintaining the original polycount of the model. This could be more useful for different human variations as there can be an infinite amount of them and modelling all of the possible variations is not a realistic achievable goal.

There are ways to deform meshes in real time, the most commonly known as animations, where the 3D model is able to rotate, translate, stretch and contract certain zones of the mesh in order to simulate a realistic movement. This is normally done by applying a **skeleton** to the 3D character model, where each bone has a weight influence of the polygons around them, making realistic bends of the mesh when this skeleton is moved for animations.
This method can also be used to deform the character's face, by simply creating more bones inside the head of the mesh to influence little amount of polygons from the face. Real-time control of those bones could portray different structural facial features in order to edit a character in an infinite variations when using a high amount of facial bones. Body types/builds work the same way, as it can increase size too. The optimal customization for these editions are be able to expand to the extremes, and then, using a slider, be able to combine multiple options to create accurate and infinite variations of human models.

Although using skeletons are very reliable, there is another option that animators tend to use for facial expressions and lip-synching, this other method is called **blend shapes**. Blend shapes have the same purpose of morphing the target, but are manually deformed previously from the original mesh to set the extremes of one facial feature (largest eyes and smallest eyes for example). This can be done with every facial feature, modifying them in multiple types, shapes or lengths, but only changing one aspect as blend shapes allows you to combine them with other deformations done. Creating these deformities by the human limits/extremes (facial features and somatotypes) would allow a slider to accurately modify each feature. By combining these multiple blend shapes, making infinite variations of human beings is now a more

realistic achievable goal.
3D softwares like Maya and ZBrush allow you to deform the duplicated original mesh by moving vertices using tools like brushes, translate and rotate and to assign these deformations to the original mesh. This way the original mesh will have a separate component called blend shapes with all the different deformations made, that can be set as a slider and have an easy exportation towards other programs.

Fig 5. Blend Shapes in Maya

Color variations such as skin tone, hair color or clothing color can be changed by each individual 3D model's diffuse map. The variation of coloring of these can be made when painting the mesh, exporting different variations, or by editing the existing diffuse map in an 2D art software like Photoshop and applying a different color tone.

2.5 Unity

The objective is to be able to customize and visualize the results in real time inside the game engine Unity.

Translating blend shapes into unity is not a complicated measure, in fact, the imported mesh itself contains the *BlendShapes* inside the component called *SkinnedMeshRenderer* shown in the inspector of the selected mesh, with blend shapes applied previously in Maya or ZBrush. Using and controlling these also have a simple application to it, as each individual blend shape contains a component *influence* with a range of 0 to 100 that defines the weight of that blend shape's deformation over the base mesh.

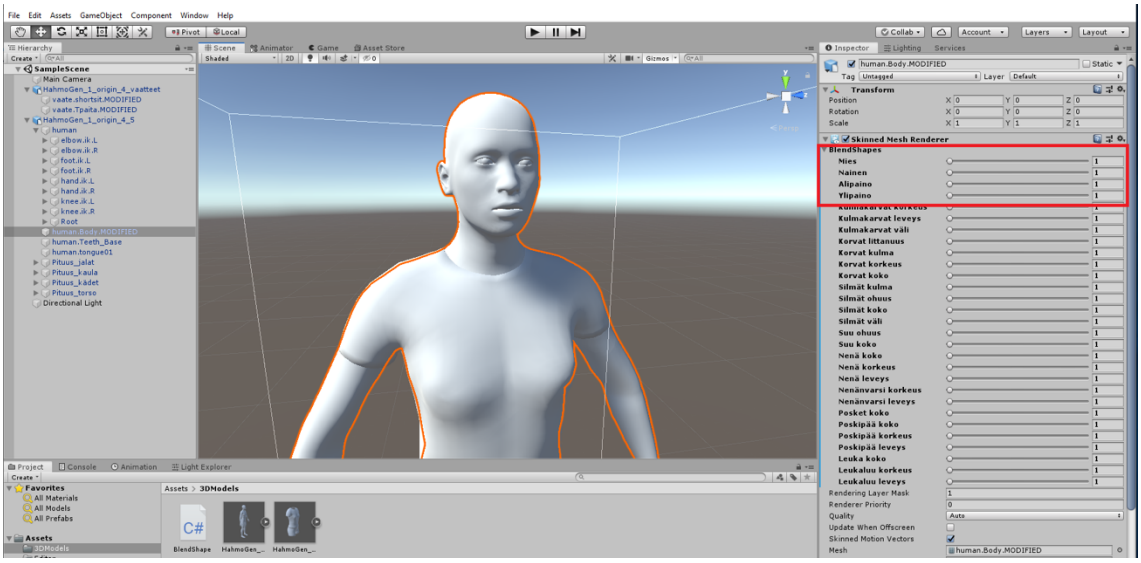

Fig 6. Blend Shapes in Unity

Unity contains multiple **UI elements** to apply into the scene composed in a *canvas,* that is portrayed when the game is ran into the size and vision of the camera. These elements include any possible UI found in any game, like buttons, images, text, sliders and more. Using these, along with scripts that access the mesh's blend shapes, it is possible to modify in screen the mesh in real time setting a slider, for example, to the influence value of a blend shape. This also works with skeletons where bone positions can be accessed in scripts and set to sliders for a similar result. This way we can accomplish real time customization.

In order to **export the final result** of the model, Unity contains a plug-in called FBX exporter that allows to export a mesh from current the running scene in real time. This method does have some issues for older versions, but there is another option involving scripting. Scripts can be created and found online, like *ObjExporter.cs*, that focus on exporting a whole mesh assigned to it manually with their texture maps appended as a material.
If maps are wanted in a separate export, it is known that texture maps are more easy to track in customization as it is not measure by a slider, instead it is selected through different options provided. The current in real time texture maps that are being used can be easily tracked in scripting In order to export the correct maps that correspond to the variation of mesh result.

If the exported model is wanted to be completely **rigged**, this is where certain issues can occur. When deforming the mesh, especially with body types and height, the skeleton could not match the resulting model, as blend shapes only change the mesh and not the skeleton itself. Unity has released an Animation Rigging package that allows developers to rig a character inside the game engine itself, but this does not work in real time as it is not an automatic rig but a manual rig.
The two main solutions to this, would be either to set the blend shapes body type

limits in a way that doesn't affect the skeleton composition (which would remove the height component), or to not simply not export the skeleton of the character and recommend the user to drag it in Mixamo for an automatic rig and animations.

2.6 Market Study

There are a few editors in the market currently, but these are tend to be really expensive or limited in certain aspects/softwares.

2.6.1 Character Creator 3

This is a **software** with a same purpose as the software this thesis is attempting to develop. It is aimed at designers and artists in triple A companies to be able to modify and existing 3D model to and edit and customize every type of feature, clothing and hair, and is not only restricted to a human anatomic mesh. This software created by *Reallusion* manages to customize in both stylized and realistic art styles, and is able to create the 3D model version in several different ways. As mentioned, the first way is to edit an existing one that is provided by the software, but you can also import own models to customize them as its advanced technology recognizes its features to be able to morph them in the way the user wants. It also includes realistic 3D scan recognition, not only for characters facial features but also for animations.
In conclusion, this software has a very advanced technology to create very high quality character models, clothing and multiple combinations of clothing with animations, realistic game hair with smooth animations, built in rigging, animations that can also be made by acting recognition and high quality rendering. A software with this type of technology and aimed for AAA companies obviously comes with a price and it is way out of range for students and indie companies, which in this case is between 200 and 1000 dollars depending on the license to purchase.

Fig 7. Character Creator 3 Screenshot

2.6.2 MakeHuman/MB-Lab

MakeHuman and MB-Lab are both **free open source add-ons** for blender with the same purpose. It loads a base 3D character model with a correct topology and polycount in blender with a full rig and blend shapes created for facial animations (expressions and lip-synch). This base model can be fully edited in terms of body type, age, fat/muscle, skin tone and multiple facial features. This is available for male and female characters in realistic, anime, stylized and fantasy art styles. Clothing and hair can be added on to this mesh and they are very extensive add-ons with multiple possibilities for artists, designers and even programmers as they have a very basic UI interpretation. As of now, both add-ons are currently still on the works for a better result but versions of it are already available for only blender, although MakeHuman released its own final **free software** version last year.

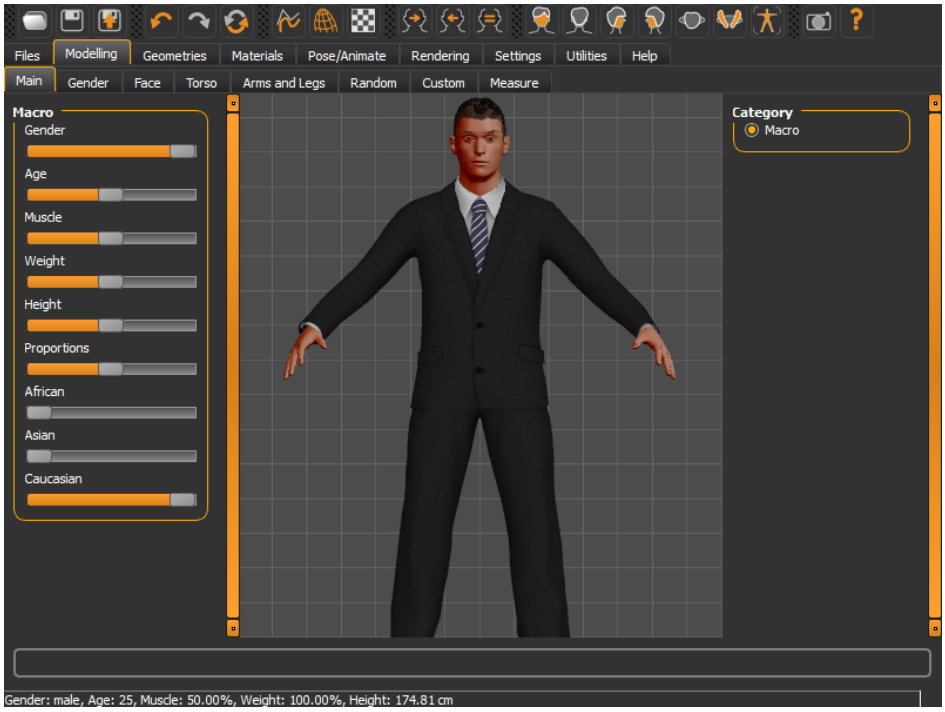

Fig 8. MakeHuman Screenshot

2.6.3 Autodesk Character Generator

An Autodesk **free website** lets signed in users generate a 3D character model with existing meshes and a full rig ready for animation. There a realistic, bulk and gorn artistic styles to choose from, where in each of these styles you have multiple base mesh options to customize. Once chosen, the method to modify its elements such as body types and facial features are done by letting you choose one other mesh from multiple options and dragging a slider to generate a mix between the base mesh and the selected option. It still allows you to choose between those meshes for each

feature but it still ends up being limited as the user has to select only one option from existing ones that are provided. Clothing, hair and skin tone can also be edited by premade options and an export configuration is also provided (polycount resolution, geometry, skeleton, textures and file type).

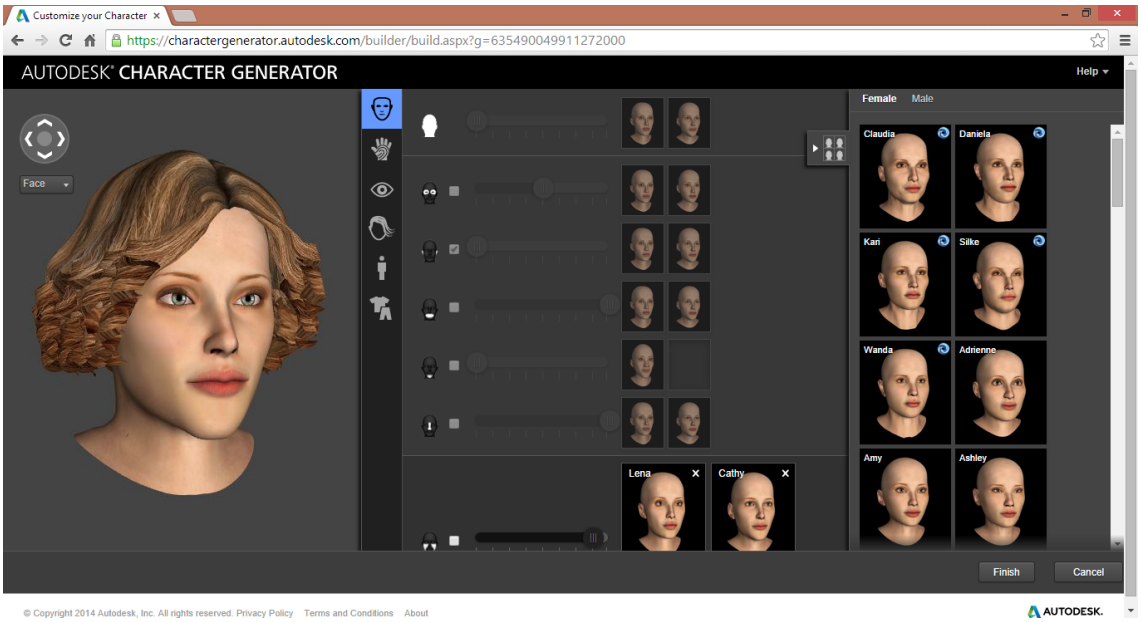

Fig 9. Autodesk Character Generator Screenshot

2.6.4 Others

Mixamo itself does not only have automatic rigging and professional animations, but it also contains a multiple choice of 3D models correctly rigged inside the own software. This is very practical as anyone can choose the character from all the choices and extract the animations from the same place, but the model choices are very limited.

Poser is a rendering and animation software but that also includes some customization. This software allows you to import your own model and apply different types of clothing to it and make slight morphs to the mesh. As it is focused more on animations and rendering, it does not have a full and extensive customization and requires a model to be imported into the software.

In-game editors let the player of the game customize his own 3D character that he/she will play with and control. This is not exportable as its already in the game, the purpose is slightly different as this is aimed for a better immersion of the player in the specific game, while the software to create in this thesis is intended for designers, students and indie companies to generate their own model or multiple models for the development of any realistic art style game.

Free character packs can be found online or even inside the own game engine, like in Unity asset store. These packs don't generally have a realistic art style and consist on different variations of already created 3D character models to choose from, but not to customize or edit in any way.

3. Project Planning

3.1 Planning and tracking tools

To ensure a correct development of this project's thesis, an extensive planning has to be made. This planning consists in setting all the tasks for the implementation of the software, previous research and investigation, the written documentation for the memory and, lastly, the testing involved.
After the tasks are listed, a realistic achievable Gantt Chart has been made where the starting and ending dates of each tasks are set, knowing that the full timeline of the project is around 4 to 5 months.

3.1.1 GANTT

This Gantt Chart represents all the timeline of all these 5 months from February to July, with all the tasks to complete in a timeframe, which have been analyzed and added a 1 day error margin. Each task has separate labels that indicate what type it is, such as research, art creation, documentation, coding or testing.
These are also divided in the methodology process that will be used to complete the project, which are pre-production, production and post-production.

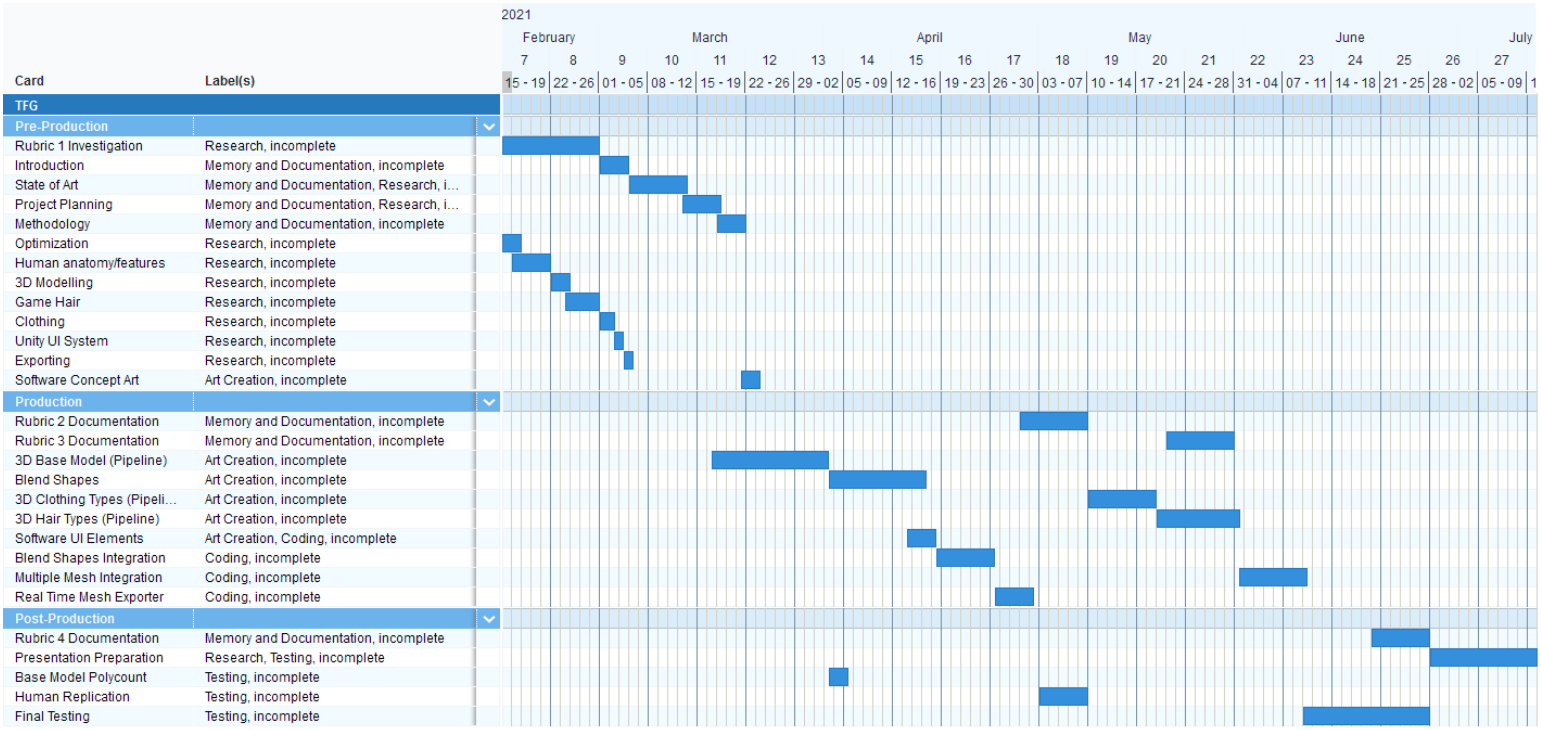

Fig 10. Gantt Chart

The **milestones** of this project are the following:

- March 19th: Rubric 1 Documentation.
- May 7th: Rubric 2 Documentation and beta version of the software (blend shapes can start to be controlled real time in Unity).
- May 28th: Rubric 3 Documentation and all extra meshes (clothes and hair) to be modelled.
- June 24th: Rubric 4 Documentation and final version of the software (addition of clothing and hair real time customization, full exportation and testing).
- July 12th: Project Presentation.

3.1.2 Trello

Trello is the task manager of choice to use during this project, where all these task will contain a detailed description of what exactly it consists of. This tool will help the organization of the project and task tracking during the development.
Each task is separated by the methodology system, but these will be dragged along to addition lists that were made to track during development:

- **TO DO:** Tasks for next milestone.
- **IN PROGRESS:** Tasks that are currently being produced.
- **Testing:** Tasks that are finished will always be set into testing first to assure a perfect completion.
- **DONE:** Tasks that have passed the testing process and have been finalized.

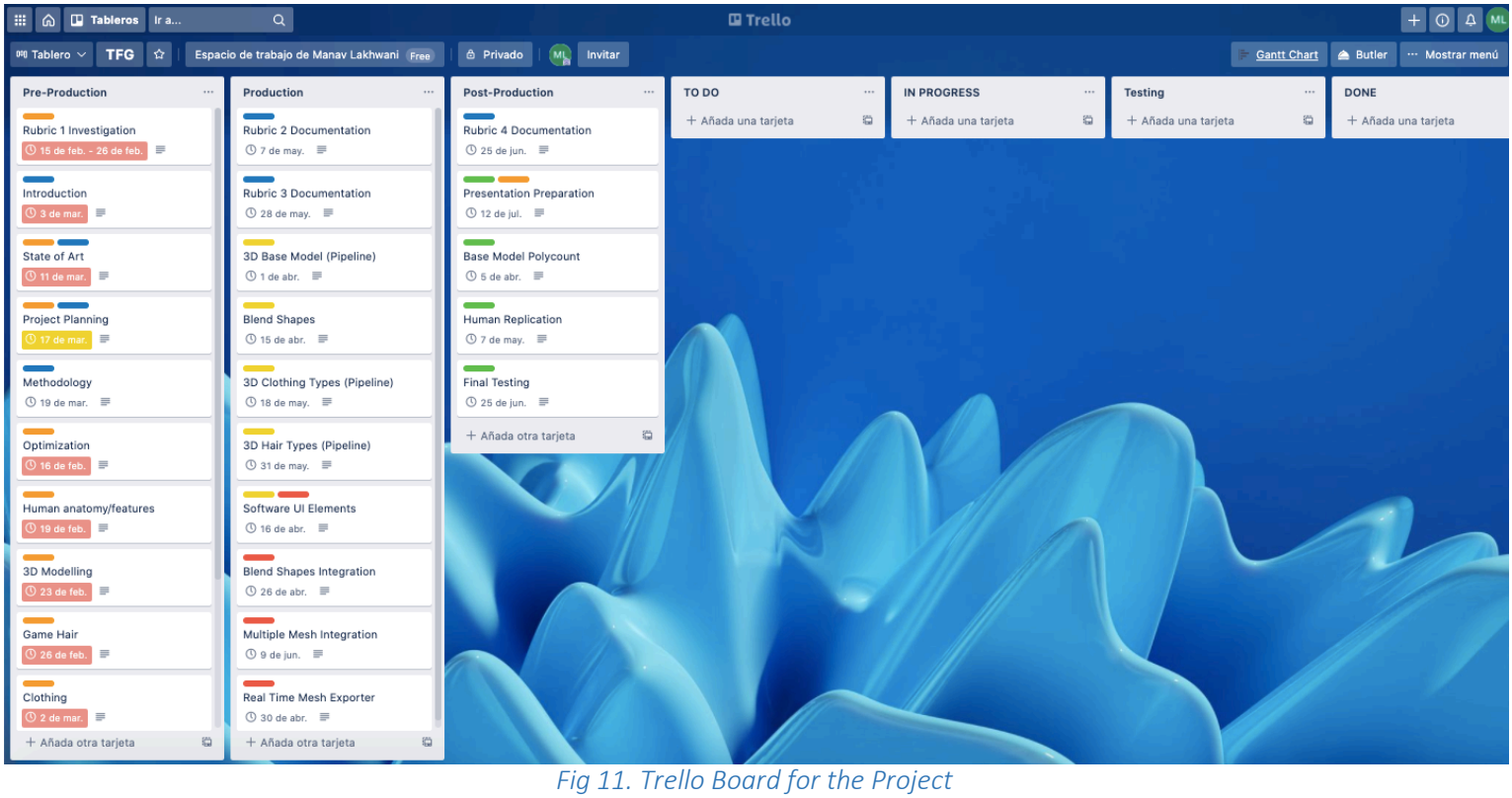

Fig 11. Trello Board for the Project

These tasks show their due date and are also set in different **tags/labels** like in the Gantt Chart, where green corresponds to testing, yellow to art creation, orange to research, red to coding and blue to memory and documentation:

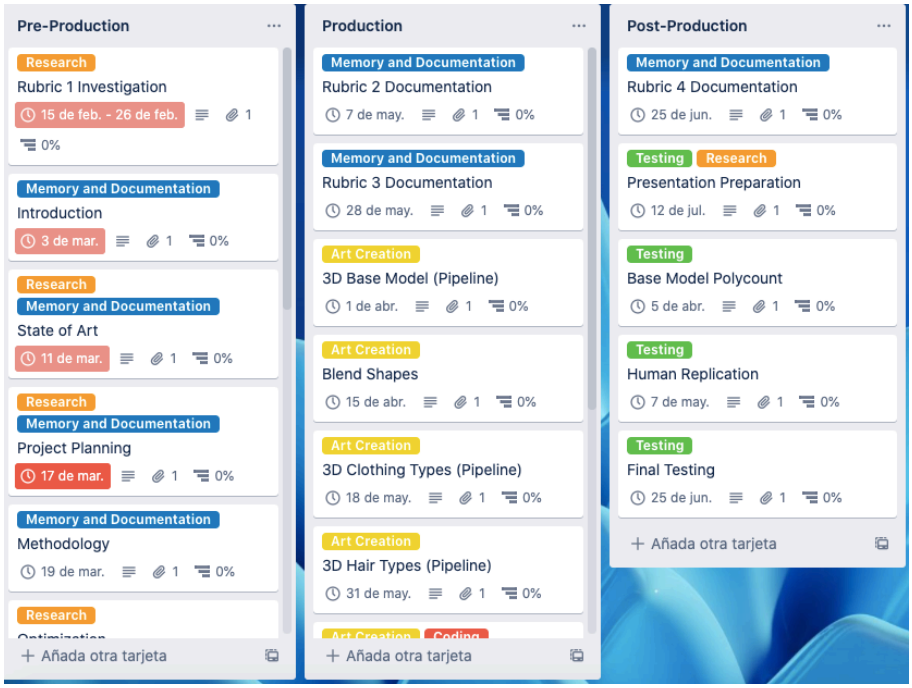

Fig 12. Trello Tasks and Labels

3.1.3 GitHub

GitHub will be the tool to use for the software development, where a repository will be made and updated during these 4/5 months. This will allow control over the versions and releases made for future testing, and will also contain a readme file that will have a documentation of the software for easy use.

3.2 Validation Tools

Some **tests** to confirm a correct and optimized final product will be done during the development of the project:

The first one will be after the second milestone, with the beta version. This version, joint with the Rubric 2, will be sent for **evaluation** by Marc Ripoll (3D art teacher in *Universitat Politècnica de Catalunya*) and several meeting will be set to receive feedback and improvements.

An **optimization test** will be made twice during development, one will be done during the beta version development and the second one will be made during the final version testing. This optimization test will consist in exporting the customized model (base model in the case of the first test as exportation won't be applied till later) and placing it in a basic Unity game. A Mixamo generated skeleton and animations will be

applied to the model to control the player in game and test the difference in frame drops and memory loss before and after importing the model.

Lastly, during the final version testing, this software will be sent to classmates inside and outside of the videogame industry. These will be informed to attempt to **replicate** themselves with all the customization tools in order to receive feedback for bugs, missing features or UI complexity.

3.3. SWOT

The following table corresponds to a SWOT analysis, where main positive and negative points are been made related to the thesis, project development and its competitors. These points are correspond to the strengths, weaknesses, opportunities and threats of the project:

	Positive	**Negative**
Internal Origin	**Strengths** • Previous experience with 3D character creation for videogames and blend shapes • Previous experience with Softwares (ZBrush, Autodesk Maya, Substance Painter, Unity) and the chosen pipeline for the project. • Previous experience in C# Unity programming videogames.	**Weaknesses** • Individual project and limited hours combined with other subjects • Low knowledge in human anatomy and human features • Difficulty modelling and sculpting human faces
External Origin	**Opportunities** • Low amount of competitors in the market, either too expensive or too simple/in progress • No possible financial loss as it is a student individual project	**Threats** • Available softwares with very high quality and optimization as they work with advanced technology, in bigger teams and bigger time frame

Table 1. SWOT Analysis

3.4. Risks and contingency plans

Developing a complete successful software with these advanced materials and programs within 5 months can be very difficult to accomplish. There are some possible risks that could appear in the duration of the project and these should have an already planned solution to minimize the errors of the development. These risks are ordered by lower to higher importance:

- **Technical Software Issues:** The softwares used for the creation of the project are very complex and have some technical issues, like crashing, from time to time. These are very common in Autodesk Maya (especially with hair creation using XGen), Substance Painter and occasionally Unity. Thankfully, I have previous experience with these issues and, when working with these softwares, the best solution is to save the project using fast shortcut keys in a constant basis.

- **Skeleton Exportation:** When modelling the base mesh using the pipeline, a skeleton will be made too. The problem occurs during customization, where the mesh is deformed and could not match the skeleton anymore (height would be the perfect example). If the solution to controlling the size and expansion of the skeleton at the same time as the blend shapes is not found, then 2 options would be established. The first one would be to remove certain customization features that could produce this issue, for example removing the height blend shape. And the second one would be to simply not export the skeleton and recommend the user to use Mixamo for the creation of the skeleton and future animations.

- **Poor Optimization:** The combination of the base mesh, clothing and hair can consist of a high amount of polygons and produce optimization issues when using it in a game. This is why extensive research on the subject is being made and a polycount budget for each mesh type is set in order to assure a correct optimization and use for videogame engines, joint with the optimization tests.

- **Bad Topology from Deformed Model:** The combination of modifying blend shapes could show some inconsistencies in the model itself. These could present some badly deformed parts of the models body and face when combining different blend shape values at the same time, as shown below.

Fig 13. Badly Deformed Mesh with Blend Shapes Combinations

This is a an example of two blend shapes acting in opposite ways, where the first (smile) is pulling the cheeks up but the second one is pushing the cheeks inwards, provoking this odd deformation. The solution to this problem is to target the shapes in harmony where none of them work as an opposite. To make sure that this is correctly targeted, all the blend shapes will have to be combined to its maximum value previously in Maya and correct all bad deformations shown before introducing to them to Unity.

- **Lack of Time:** As seen in the Gantt Chart, a lot of tasks have to be completed in just less then 5 months to develop the whole software. The estimation of longer tasks involve all the models to be created from scratch, such as both male and female base meshes, clothing and hair. In the case that time is an issue where models and implementations aren't possible to reach on any version established, these models will be purchased online (bought or free) with the correct polycount and topology. For example, if time is an issue for the beta version, the male and female base models will be got online and blend shapes will be done on them and implemented in the software to arrive to the beta version. For the final version, the same thing could be applied in clothing and hair meshes.
  If these contingencies manage to make too much extra time, poses and small animations will be applied to the base mesh in order to select in real time inside the software.

3.5. Cost analysis

This following table shows an initial cost analysis done for the development of the whole project for the duration of 5 months. These costs include the fact that an office with equipment is rented and that myself as an Artist/Programmer would have a 700€ monthly salary.

		Type	Cost	Amortization (years)	Total Cost
Salary	**Artist/Programmer**	Monthly	€700.00		€3,500
Equipment	**Computer**	Amortization	€800.00	3	€111
	Screen	Amortization	€150.00	3	€21
	Mouse	Amortization	€15.00	3	€2
	Keyboard	Amortization	€30.00	3	€4
	Chair	Amortization	€50.00	6	€3
	Desk	Amortization	€100.00	6	€7
Software	**Unity**	Unique	€0.00		€0
	ZBrush	Monthly	€33.39		€167
	Maya	Monthly	€171.35		€857
	Substance Painter	Monthly	€16.63		€83
	Visual Studio	Unique	€0.00		€0
	Mixamo	Unique	€0.00		€0
	Github	Unique	€0.00		€0
	Trello	Unique	€0.00		€0
	Photoshop	Monthly	€24.19		€121
Maintenance	**Rent**	Monthly	€500.00		€2,500
	Electricity	Monthly	€20.00		€100
	Water	Monthly	€10.00		€50
				Total Cost	€7,526

Table 2. Cost Analysis

All marketing and publishing costs are not shown as this is just the cost analysis for the actual development of the software. A total of **7526 euros** is the cost for the duration of 5 months, where rent and salary are the most predominant.

3.6 Planning changes

During the development of the software, some planning changes were made in order to accomplish the objectives in the time frame given. The changes are only focused on the production part of the project and are only targeted to the **creation of the models** (base meshes, clothing and hair).

Initially, the idea was to create all these models from scratch with the pipeline explained in the state of art in 2.3, but during the development of the project I realized that this requires multiple days of work that are not invested on the main objective of

this thesis, which is the **customization**. In order for models to be aesthetically pleasing with a same art style, correct topology and optimized polycount, it can take a few days to complete, and in a customization software this means it requires multiple models for the user to choose. On top of that, the method of using blendshapes for the base mesh in body types for a more accurate result implicates that all the clothing meshes will need to be deformed with additional blendshapes on each clothing model.
After that realization, it was obvious that a decision had to be made, where in less than 5 months there was only time to focus on either creating all models the models myself and providing a very limited customization to the user, or searching and exporting online models with the art style, topology and polycount desired and applying the blendshapes to them providing the user with a much more wide customization.

As the main objective is the character customization with as many options as possible, it is clear that the choice of searching all the models online (base meshes and clothing) had to be made and focus more on the customization part with the correct blendshapes.
It is for that reason that the only **planning change** is to not create all the models myself, but instead to get the base mesh, hair models and clothing models online. The tasks, order and duration of them will maintain the same except of the creation of the base mesh and 3D clothing models, where the duration will be reduced by 5 days on each task and those extra 10 days will be used to implement the blendshapes on all the base meshes, clothing and hair models.

4. Methodology

As seen in the Gantt Chart and Trello tasks, the methodology process for the thesis and the development of the project is preproduction, production and postproduction. This process is a chronological method that helps distribute the different stages of the full project development.

Preproduction is the first stage of the project, where everything that is needed to know and to be set and prepared is done, before the development of the actual product is started. This is also known as the planning stage and it involves extensive research, investigation, concepts and preparation for the correct implementation further on.
The tasks of the project in this stage, apart from the Rubric 1 documentation, involve an extensive research and investigation of optimization, 3D modelling, game hair, clothing, human anatomy and features, softwares, customization, mesh deformations, pipeline/workflow and concept arts of the software and 3D meshes.

The **Production** stage is the second stage where all the implementation comes to place. Using all the research and knowledge learned from the preproduction, the whole product is developed and completed for testing.
The tasks of this project in this production stage, apart from the Rubric 2 and 3 documentation, involve all the development of the software in a version that includes all the elements wanted, using Trello to track all the tasks and assuring a correct development in the small time frame (as explained in 3.1.2).
During this production stage, 2 versions will be developed, the **first version is the beta**, where the 3D base models (male and female) will be added to the Unity Scene with all the blend shape deformations, joined with the UI system programmed to control each of these blend shapes in real time.
The **second version** will be the completion of the software, where clothing, hair and color variations will be added after modelling them. The user will able to choose between them in the software and export the final customization.

The final stage, **Postproduction**, is the testing and perfecting of the product. This includes adding the final touches and modifying certain flaws in the product to be ready to use for the public.
The tasks of the project in this stage, apart from the Rubric 4 documentation and the presentation planning, involve the optimization tests of the exporting models, the testing of an accurate replication of any type of character (using the validation tests explained in 3.2), and the final testing of all the software elements itself with any modifications to be made to perfect it.

After this stage, the product will be completed and fully tested, and for that reason it will be ready for a public use.

5. Development

Following this methodology explained, the main focus in the development process is to be able to create a software where designers can fully customize a base mesh with a very straight forward UI that will be easily understandable to for all types of users.

5.1 Preproduction Phase

In order to complete this, there are some decisions to be made after the extensive research made in the state of art.

5.1.1 Solution options

The first decision to make is to figure out if the base **meshes** (male and female), all clothing models and hair meshes will be created or will be searched online. In order to make a software that is fully customizable, there needs to be as many meshes as possible to give the user a huge range of options.
Blendshapes or a skeleton applied to the base meshes controlled by sliders manage to accomplish the objective of infinite options, as sliders can be set to infinite amounts. But these cannot represent different types of clothing and hair, so in this aspect a very large amount of models are needed for the user to choose between them.
For that reason, creating all these models instead of focusing on the customization would end up giving an end result of very limited options for the user to choose from, so the base meshes and clothing models will be searched online. This will give more time to focus on the customization and the different hair and beard types for a complete character editor and a big amount of options for the user to choose from.

Hair mesh creation is a complicated method, as explained in the state of art in 2.3, but there are still multiple ways to create it. The XGen generator in Maya or the fibermesh generator in ZBrush are the most advanced examples for videogames. XGen is a very extensive generator with a huge variation of options to edit, but it can be a complex process to turn it into hair cards for it to be optimal in videogames. Which is why the creation of hair type meshes will be made with **fibermesh** in ZBrush, that already has an easy option to select the amount of polygons to use and apply the textures for an easy visualization of a correctly optimized game hair.

When it comes to **customization**, as seen in the state of art in 2.4, there are a few solutions to pick from to create this software. During the research, it was seen that models could be edited in real time by mesh deformation (blendshapes, a skeleton) and by multiple meshes activating and deactivating.
It is clear that the only option for clothing and hair types are needed to be made in separate meshes, as these types vary a lot from each other. For example, to convert a t-shirt mesh into an armor suit in the same model by mesh deformation would be

close to impossible, apart from damaging the topology and the textures applied to it (normal map, height map and ambient occlusion). Apart from this not making sense, the point of mesh deformation is the usage of the intermediate values between the transition from the t-shirt to the armor suit, which would have no use in these situations. For that reason it only makes sense to use **multiple meshes** positioned on top of the base model that activate and deactivate when selected for these hair types and clothing types.
On the other hand, **mesh deformation** is the suitable option for the base mesh customization of facial and body features, as seen in the state of art in 2.4. But mesh deformation can be done by blendshapes or by applying a skeleton to the mesh, and controlling either the blendshape value or the bones of the skeleton in real time to modify the base mesh for customization. The upside of using a skeleton are that the skeleton is only needed to be done once, it can add as many bones as you want, and it can also modify the clothing models at the same time. The downside of using the skeleton are that they are not as accurate, the bone movement has to be configured for each movement and rotation inside of the engine for every deformation that is wanted to be made and would need an extra skeleton for animations. Using blendshapes can be more work, but it is definitely more accurate you modify the vertices of the mesh to make the deformation desired and will automatically create a transition between the original base mesh to the deformation made when the blendshape is applied. This is very useful for customization, as you can set the maximum and minimum of each of the features to create a full infinite optionality. The downside of blendshapes is that it needs more work, especially as it only applies to the individual mesh, which means that it is needed to redo the blendshapes for the clothing types to match the body types deformation of the base mesh.
Both methods are very acceptable, but in an customization software for characters, the more accurate representation the better, which is why **blendshapes** will be used for the deformation of the facial features and body types of the base mesh, for the deformation of the clothing models to match the body types and for the hair models of all types to represent the length of it.

It is also important in customization to have the ability to change the **color** in multiple places of the character. The user will have the option to change the skin tone of the character, the color of the eyes, the color of the clothes and the color of the hair/beard. There are a few ways to accomplish this in real time, as Unity allows you to access the materials that the models are using and change certain aspects through a script. The obvious way is to change the whole material with different textures, but as the only thing that changes is the color, it is more optimal to access and change solely the **albedo map** (color) of the material in use.

The colors of all the clothing, skin tones, eye color and hair color will be made by painting it myself in an external software, this way there is much more control of the different options and it maintains the same style as it should when only changing the color opposing to searching these color variations online.

5.1.2 Technology usage

Nowadays, there exists many **modelling softwares**, as seen in the state of art in 2.3, where Autodesk 3ds Max, Autodesk Maya, Blender and ZBrush are the most famous as it permits you to create anything with unlimited options with many tools to use. The choice of software depends on the personal preference, as all of these allow you to create the same type of model with the same options, but, as mentioned previously, 3ds Max and Maya do have a more technical view to 3D modelling. When it comes to creating new models, and in this case hair types, I personally find it more important to work in a more artistic view, which is the aim of ZBrush with its sculpting and visual tools. In addition, as the models created will only be hair meshes, the fibermesh generator in ZBrush is the best option to choose from for the game hair in this project, which is why for any 3D modelling (blockout, low poly, high poly) purposes during this development I will be using the **ZBrush** modelling software.

As explained before in the solution options in 5.1.1, the models will be gotten online but I will be painting them to complete the color variation for the user to pick from. In order to do so, a **texturing software** is needed to produce and export these albedo maps to integrate it to Unity.
Some softwares that focus in modelling still have painting tools to complete the whole pipeline in the same software, such a ZBrush. This software not only has an extensive amount of sculpting tools but it also gives the user the ability to paint the model in 3D when having a UV map. This method can be used with ease for this project as the main focus is to only change the base color, and what ZBrush lacks in painting tools are the other maps (metallic, ambient occlusion) as it has a limited amount of materials.
The most famous texturing software, Substance Painter, is mainly focused on producing these textures by also painting your imported 3D model with a UV map and applying different materials, alphas and brushes that allow to portray a very accurate representation of your ideas. It also allows you to import online smart materials created by other in a public forum to apply to your models and it has an easy access to export all the textures of your mesh at once.
As it has a much more wider range in painting tools and is a software dedicated to texturing, I will be using **Substance Painter** to paint and export the albedo maps from all the color variations as it can portray a much more accurate result, especially for a realistic art style.

The main focus of this thesis to accomplish this customization of infinite options is clearly the creation of **blendshapes**. As seen in the state of art in 2.4, the most famous 3D modelling softwares that allow the creation of blendshapes are Autodesk Maya and ZBrush. ZBrush is a very good option as it has an extensive option of sculpting tools, which is essential for mesh deformations as blendshapes must only be moving vertices and not adding any. It also allows you to view and export the model with the blendshapes applied already.
Even though this a very good option, testing combinations of blendshapes is a very important part of the method to correct badly deformed mesh combination, as seen in

Fig. 13 in 3.4. This is why a more technical software could come in handy, Maya has a very visual way of expressing blendshapes with the Shape Editor, where all the deformations are shown in sliders and can be controlled in real time modifying the original mesh. This way it is easy to combine multiple blendshapes and find any topology errors with the wireframe view, and additionally contains a method to create an extra blendshape to correct errors between the combination of two other deformations. Although Maya has a more limited amount of sculpting tools than ZBrush, it does also have a soft selection tool that allows to control a numerous amount vertices as a circular range where it has a lower effect in the outside areas, making blendshapes a much easier process to complete.
It is for this reason that **Autodesk Maya** will be the software of use for the creation of blendshapes, as it has a better focus on it than ZBrush despite its lack of sculpting tools.

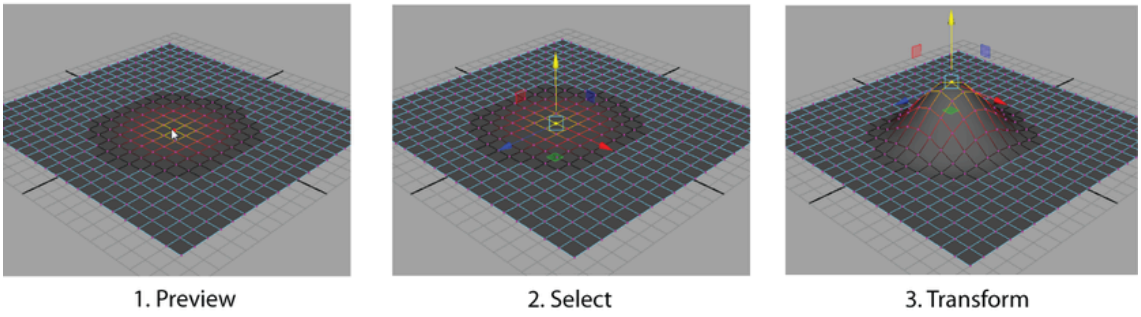

Fig 14. Soft selection tool in Maya

To **create the software** with the objectives of this thesis, another software is needed that has functionality, multiple 3D models (with textures) are needed to be imported and exported, it needs to be run at real time and, most importantly, this software needs to detect the blendshapes applied to the base meshes. The softwares that meet these requirements are game engines, which is also a good thing as the exported mesh is meant to be used for those exact engines as it is for videogame development. For that reason, with Unity and Unreal Engine 4 being the most complete free public game engines, the choice between them is not for technical or artistical reason as both have very similar options and availabilities. Unity has been a bit more advanced in technical and optimal aspects, but the decision of using **Unity** for the development of the software is simply from having a larger personal experience on it, and where UI functionality, blendshape integration and creating an executable is much more clear and straightforward process.

For the **optimization testing** part further on the project, as said in the validation tools in 3.2, the models will be placed in a game scene that has already been made and tested the runtime statistics of any FPS drops or memory loss to assure that the model is not affecting it negatively. This will have to be in a game engine and, for the same reason as before, **Unity** will be used for all of these testings. Apart from the previous experience in Unity, this game engine also has a lot of packaged to import to the engine, that also includes a full game scene from Unity Engine itself, which will be the

one that will be used for this testing. Unity also has a runtime statistics window that shows and updates immediately all the rendering statistics during play mode such as FPS, milliseconds per frame, polycount and more.

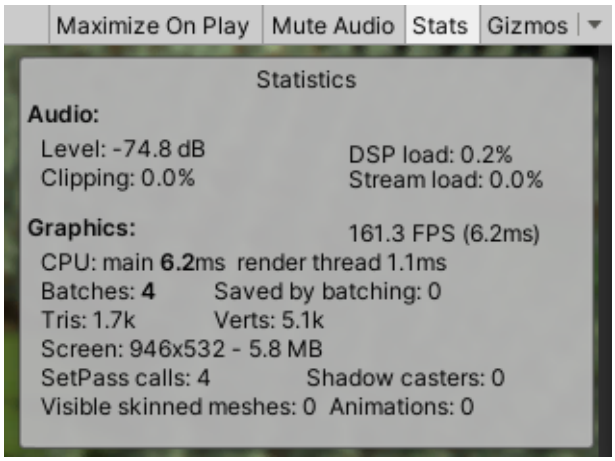

Fig 15. Example of runtime statistics window in Unity

5.1.3 UI Design

During the preproduction, a concept of the user interface design has to be made in order for everything to be clear when adding the UI elements in the engine. This way, if any changes have to be made, it can easily be adjusted as it is just a sketch instead of having to be changed after programming all the buttons inside the engine, having to do that process all again.

This software is aimed for **designers**, which is why all the elements have to be clear for users that do not understand a lot about 3D modelling. For that reason, all the UI has to be very simple, with the mesh being seen all the time for the user to identify what he/she is changing real time, and very basic nomenclature to understand everything clearly. It should not contain complex words in the artistic area that designers do not have a full understanding about.

Taking all into account, the final concept menus that were made are the following:

The **main menu** is a simple selection menu, a very straightforward option between the male and female base mesh, which can be previewed on screen for a better choice.

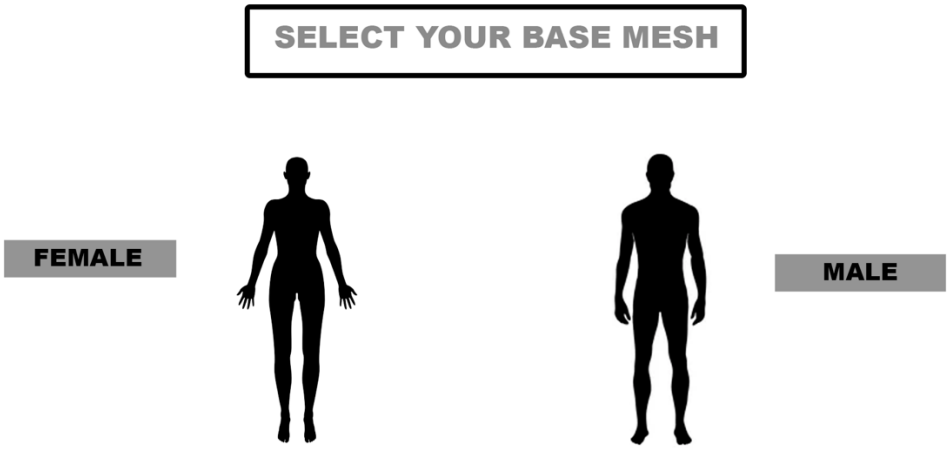

Fig 16. Main menu UI concept

After picking the base mesh, this **first mesh menu** will appear with the first customizable options. The first options will consist of all the facial features and body types done with blendshapes. The base model will be on screen at all times and dragging the mouse button on it, it will rotate for a better view of all of it. It can return to the main menu, reset the rotation of the model after rotating it, have a wireframe option to view the topology. In the customization part, it will contain buttons of all the body parts that can be edited, will have a randomize option to set every blendshape to random, and a next button to continue to the next customizable options in the second mesh menu.

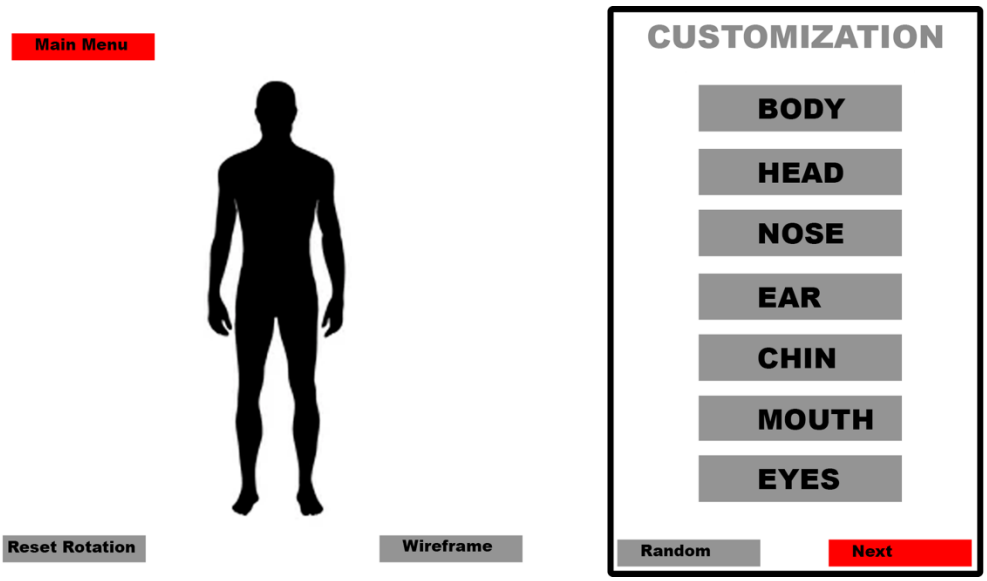

Fig 17. First mesh menu UI concept

When one of the buttons is pressed from the first mesh menu, the following **blendshapes sliders menu** will appear. This will maintain the same elements but will now have the title of the part of the body pressed, a back button to return to the first

mesh menu and all the multiple sliders that control the blendshapes from that part of the body. All these sliders will have a title informing what it does and the value of the slider on the right, which will range between 0 to 100, or -100 to 100 if the slider contains two opposite blendshapes (maximum and minimum).

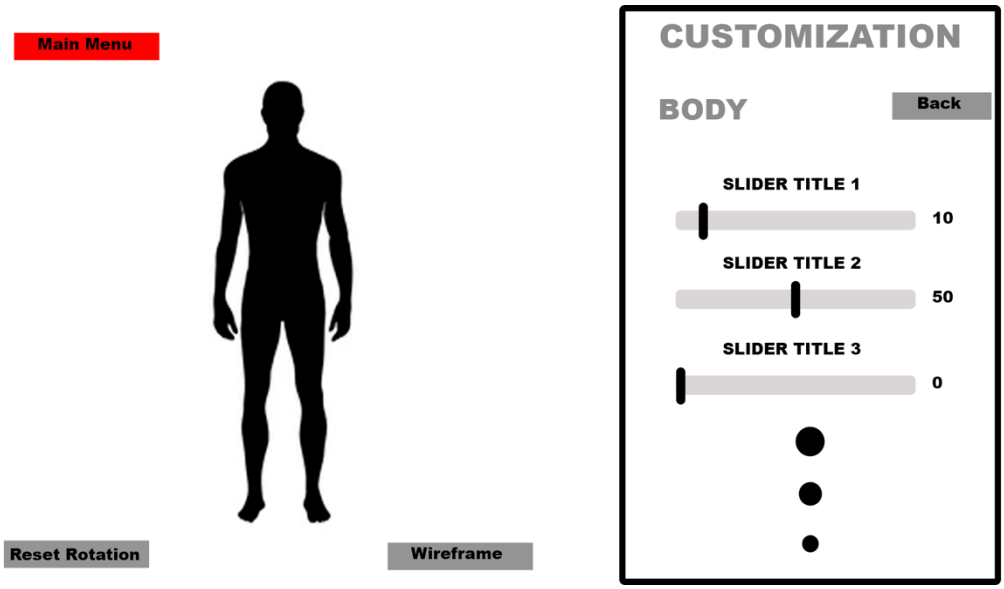

Fig 18. Blendshapes sliders menu UI concept

When clicked next in the first mesh menu, the **second mesh menu** will appear with the exact same UI structure but with the next customizable options and a back button to return to the first mesh menu if wanted to change a previous option.

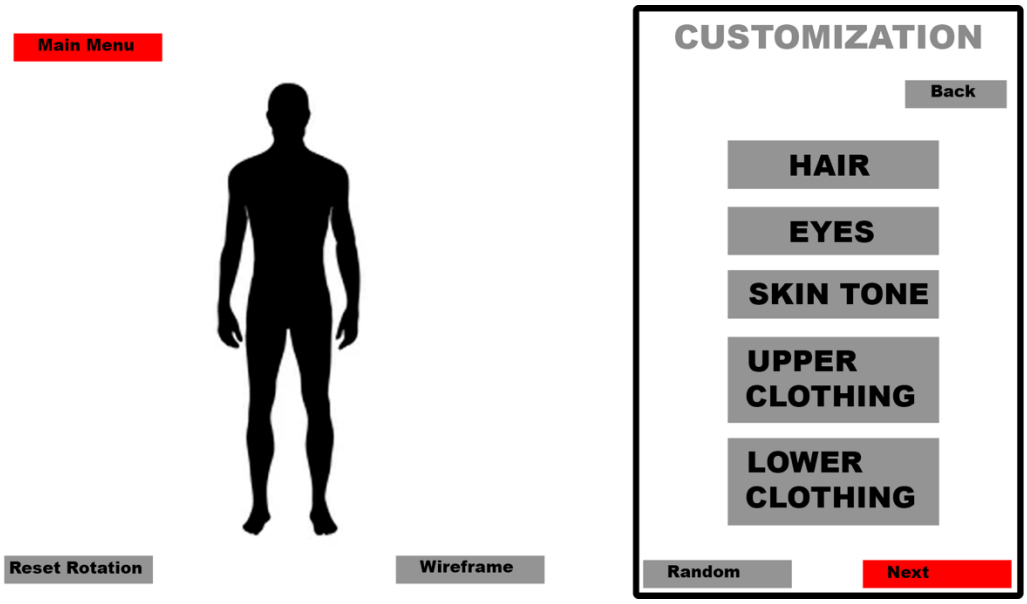

Fig 19. Second mesh menu UI concept

When selected one of the second mesh menu buttons, the following **menu displaying the multiple meshes and color options** will appear, where depending on what button was previously pressed, more or less options will appear, just like the blendshape

sliders menu. The different types of clothes, beard and hair will be selected by switching through the right and left arrows and will show a small icon on the middle apart from showing it on the model in real time. If a blendshape exists on any of these meshes, a slider will also appear, for example the length of the hair. And lastly, for color variations, multiple options will appear as circular buttons of the corresponding color, that will change in real time when pressed a different one.

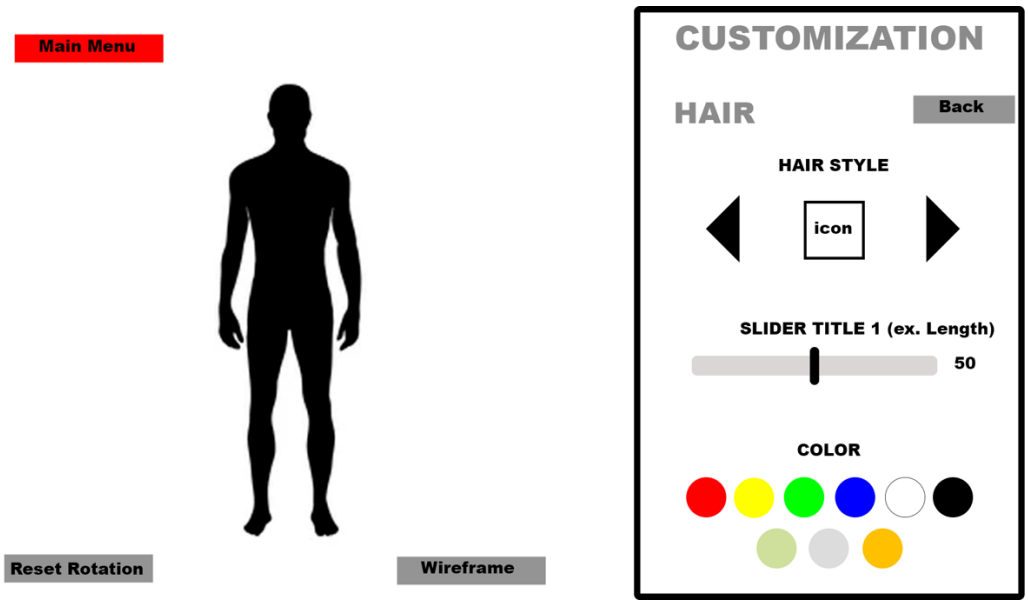

Fig 20. Multiple mesh and color variation menu UI concept

Lastly, when clicking next in the second mesh menu, all the customization will be finalized and the **export menu** will appear on screen. This will again maintain the same UI structure for the user to still see the model and decide if he/she is happy with the final result. If the user is not happy, the back button is there to return and change any aspect again, but if the user is happy, the export button will automatically export the model and textures to their computer. Some information text will appear explain the location of the exportation and any additional information needed to be told to the user, such as the follow steps in game development like setting the model in mixamo for the rigging and animation process.

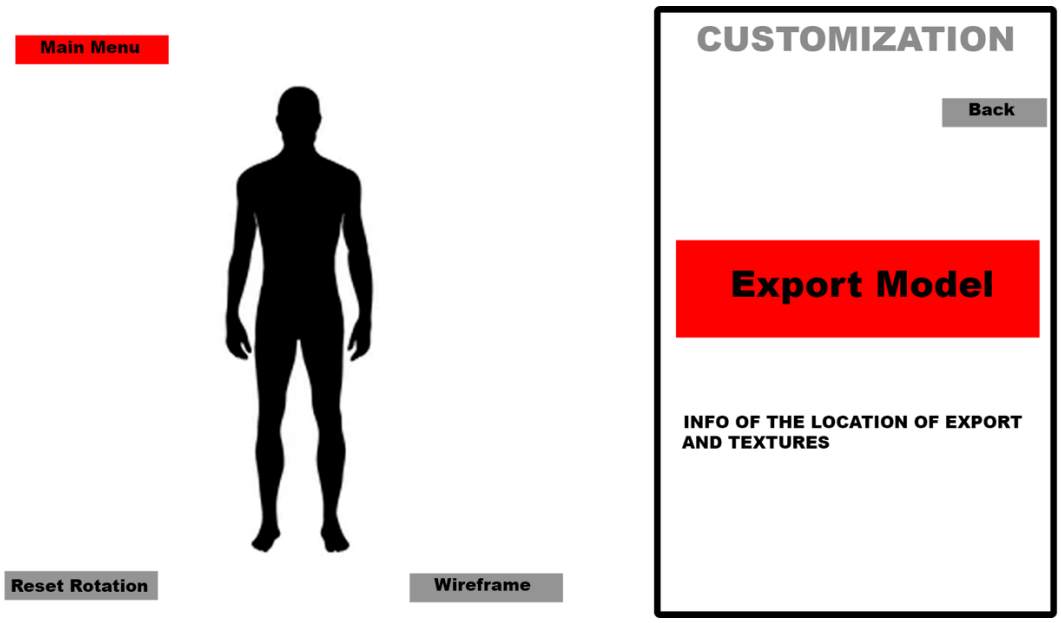

Fig 21. Export menu UI concept

5.2 Production Phase

5.2.1 Base Mesh

The base meshes, both male and female, need a very correct **topology** for videogames and a **polycount** high enough that does not surpass the 20k polygons, as we have seen in the research made in the state of art in 2.1 and 2.3.
In an ideal place where this was made by a company with months or a year to work on, these models would be created from scratch following a pipeline to ensure that the mesh has that correct topology with loops around areas that vertices move in animations and a set polycount number. But due to the time frame and being an individual project, these base meshes are needed to find online.

There are multiple websites that contain free and/or paying models to download instantly to your computer, such as sketchfab, artstation market, cgtrader, free3D or turbosquid, which are the websites used to find both male and female base meshes for this project. The **main requirements** set when searching for these are the following: low poly mesh polycount around 10k/20k with UVs done, a correct topology made for videogames/animation, a realistic art style, a good aesthetical condition, without any clothes or hair (or at least separate) and that contains either a high poly version or a normal map applied (this way the only missing part to finalize the base meshes would be the painting).

After an extensive search to find male and female models that meet these requirements, a few options were found, some of which very expensive, but the final models to be used for the project were made by *Jellostain* (female mesh) and *Pter Jzsa Jr* (male mesh), both found in turbosquid for free.

The downloaded **male mesh** met all the requirements and even had a full rig with all textures (color included) applied to it, which meant that there not many changes were needed to be made as it was a completed model. The hair and skeleton will have to be removed as animations will not be used for this project.
On the other hand, the downloaded **female mesh** is a ZBrush project that contains the low poly model with its multiple subdivisions where the high poly is being sculpted. The model does already have UVs, so in theory the last step would be to export the normal map from the highest subdivision and export the model in one of the lowest subdivision (around the polycount that we set). But some changes were needed to be made to this mesh, as the eyes in this model had a extremely high polycount. For this reason, a new set of eyes was needed to be made in the same position following the pipeline (low poly, UVs, high poly) and extract the normal map of all these elements joint with the low poly.

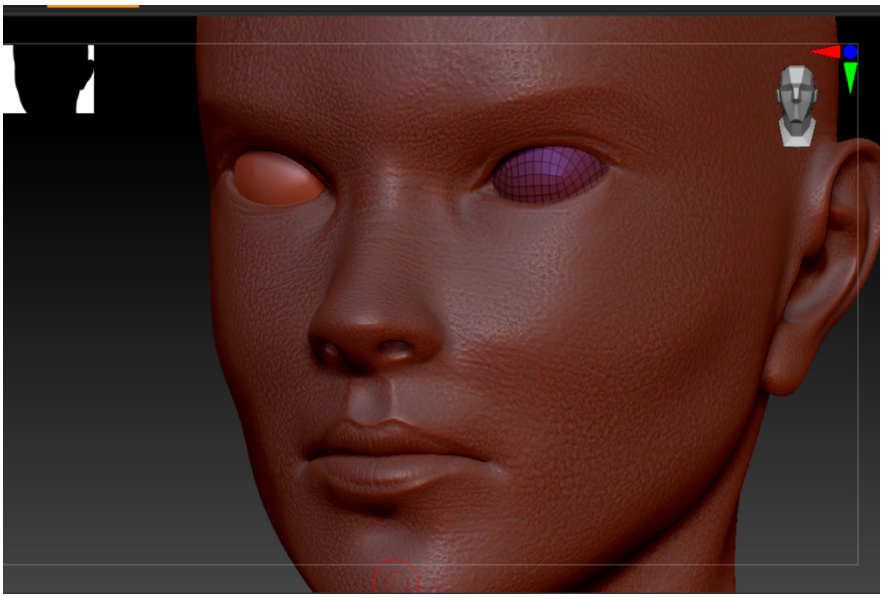

Fig 22. ZBrush final eye remake

After the final adaptations for both these models, the male mesh contains a total of **16322 polygons** and the female mesh contains a total of **8186 polygons**.

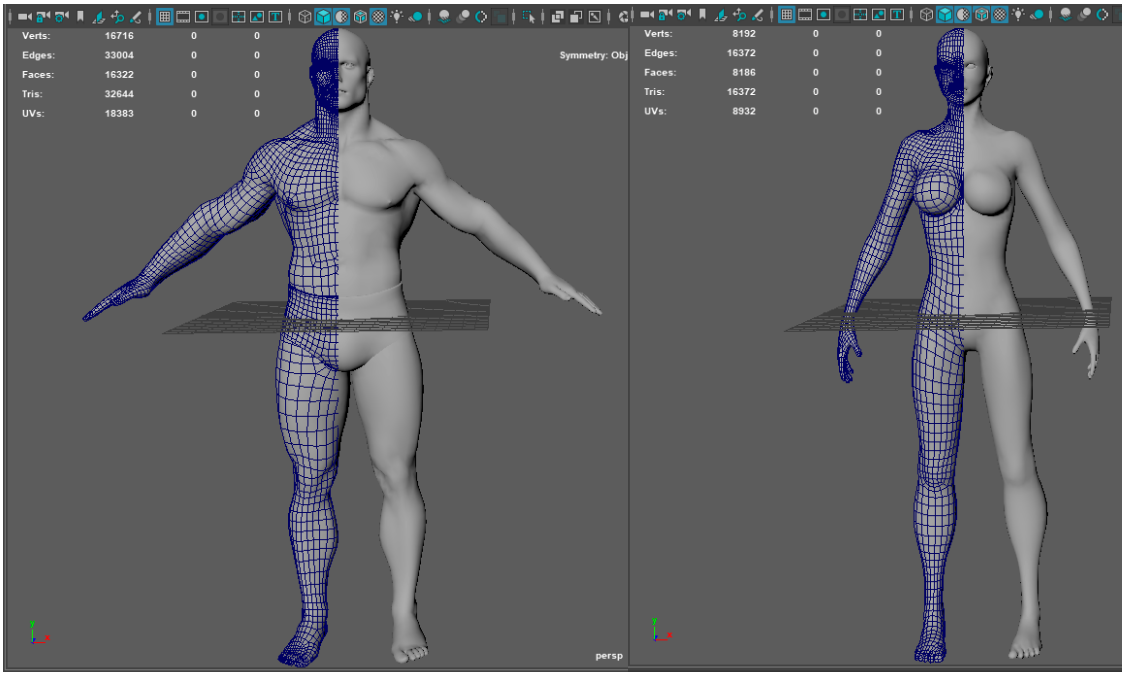

Fig 23. Male and female mesh in shaded wireframe in Maya

When deforming these meshes using blendshapes, it is important to have all textures applied in the model to make sure that the deformations do not have visual errors involving the textures too. Which is why the **creation of textures** is needed to be done before starting the blendshapes process.
Using Substance Painter, the low poly can be imported containing the normal map, and a skin material can easily be applied to the base mesh (which will also facilitate the color variations that are needed to be made later on by changing the color of this skin material). The female eyes will also have to be painted as it was done again, and the best way to paint realistic eyes in this case is to get an eye image online and use it as a projection to paint it on top of the mesh in 3D view.

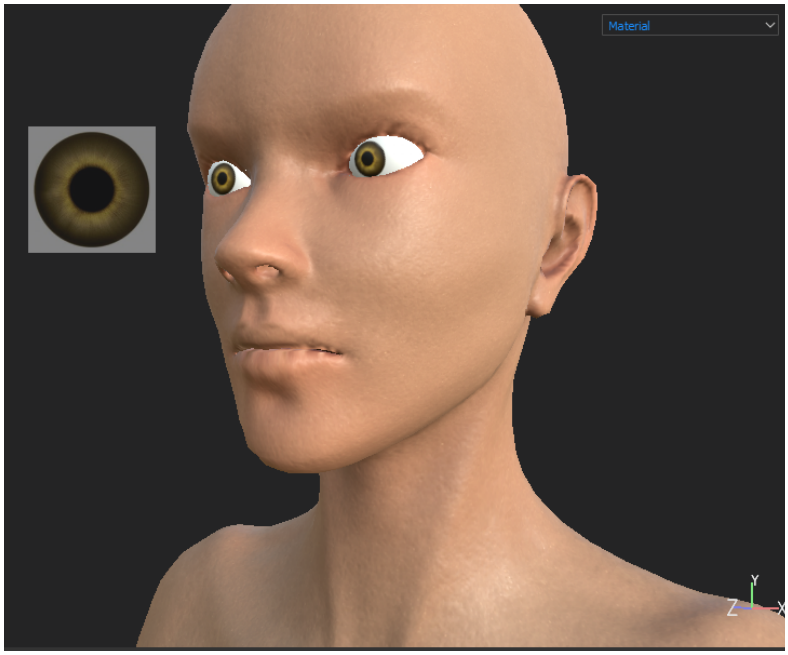

Fig 24. Substance Painter skin material and eye projection

In order to assure that these models are optimal for videogames, there needs to be an **optimization test**, where these models are placed in a Unity game scene and studied the difference in performance before and after the integration of these meshes. Unity contains a videogame package in its asset store called *3D Game Kit*, which is a good choice to use for optimization tests.
The models should be place in a scene of a level of this package with all its textures applied and, not only ensure that the models look correct with their respective material, but also that the they do not cause any performance issues when playing the game.

Fig 25. Models in game scene with materials

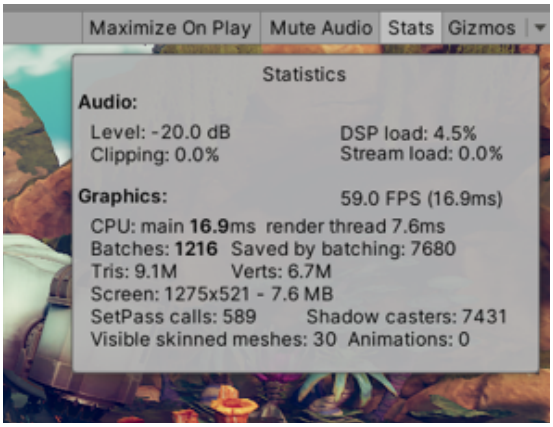

Fig 26. Runtime statistics by default

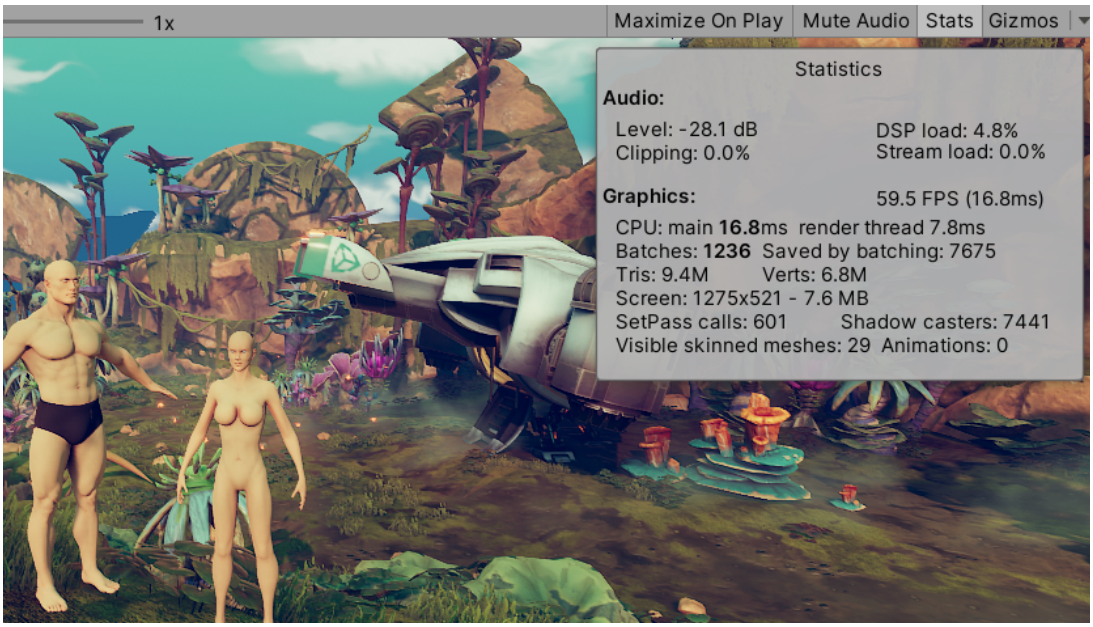

Fig 27. Runtime statistics with the mesh integrated

As seen in these screenshots, the runtime statistics stay in a very similar value in the same camera position and angle, which means that the meshes do not affect negatively in the performance of the game, approving the mesh as optimal for game development.

In order to also ensure that animations look correct with these models, a simple test has to be done of putting the mesh into the **Mixamo** website for an automatic rigging and testing it with some of their sample animation for any odd deformities and stretches in the mesh.

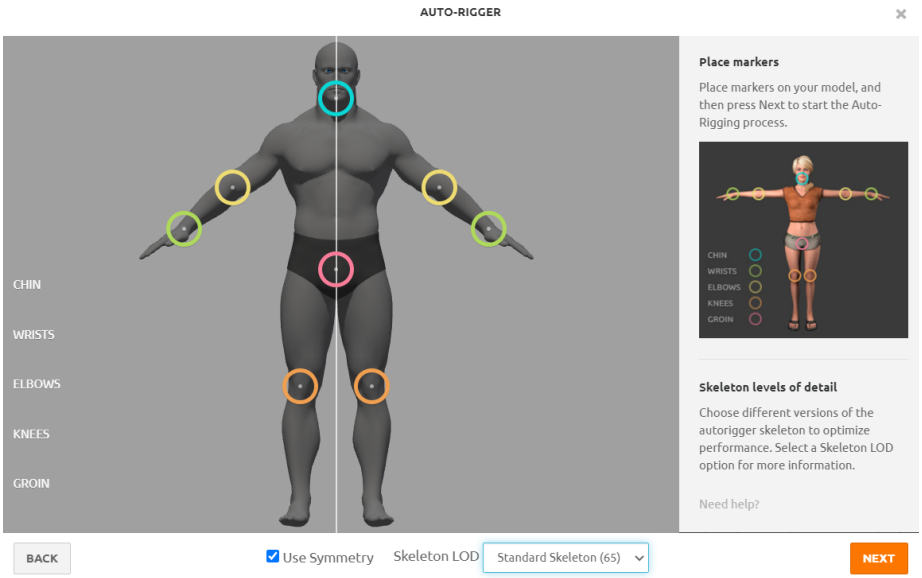

Fig 28. Autorig in Mixamo

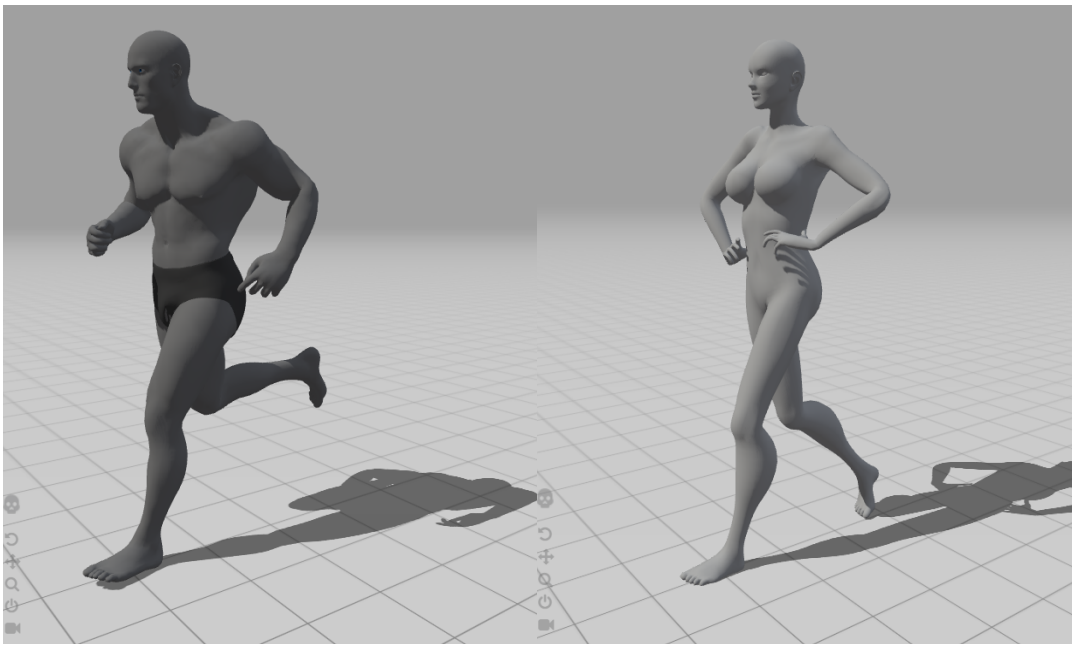

Fig 29. Sample poses/animations in Mixamo

If these tests are successful like the examples shown, the model is now surely optimized and usable for videogames, which is why it is now ready to start the creation of blendshapes.

5.2.2 Blendshapes creation

As explained before in the technology usage in 5.1.2, Autodesk Maya will be the software of use to create the blendshapes for the models.
The first step is to **import** and set the Maya scene correctly, which is why both low poly meshes will have to be imported and it is important to place both meshes in the

middle of the grid with a similar scale to each other. When both of these models are in the desired scale, Maya permits to *freeze transformations* of the mesh, setting the object position, rotation and scale to its identity by default, allowing the meshes to export at that same translations for the integrity to Unity.
Textures are needed to be applied to both models too, which can be done by applying a new **material** to the object. There are a selective amount of materials to choose from, but the important part is to make sure that the material of choice allows you to apply all the textures that were exported for the mesh. In this case, as the meshes have a normal map, albedo map, height map, ambient occlusion map and a metallic map, the Ai Standard Surface material will be used, although there are more options such as Phong or Lambert.

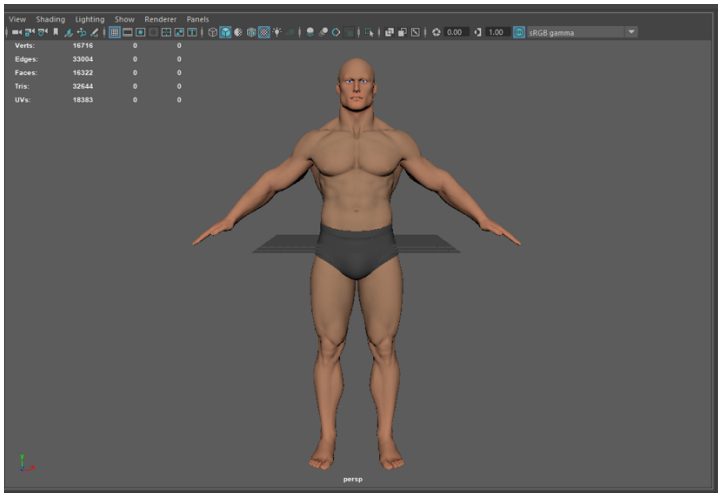

Fig 30. Textured Male Mesh in Maya

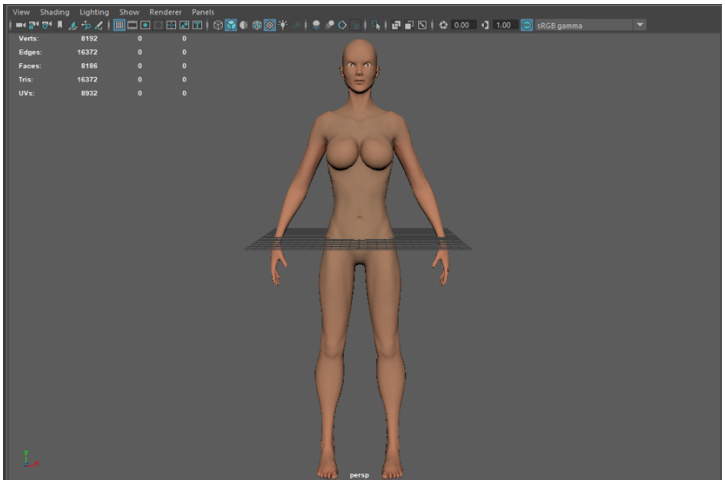

Fig 31. Textured Female Mesh in Maya

Once the meshes are imported and set correctly, the blendshapes can be created. Maya includes a *Shape Editor* window aimed for these deformations, where new blendshapes can be created and edited modifying the original model. Although this is a plausible option, when having multiple blendshapes it can be a bit messy if wanted to go back and change some of the deformations.
For that reason, a better and more structured way is to first **duplicate** the mesh as

many times as you want for each deformation. This will also reduce complications when working in one only model with multiple variations and will have a more structured scene with easy access to change any deformation further on if any modifications are wanted to be done.

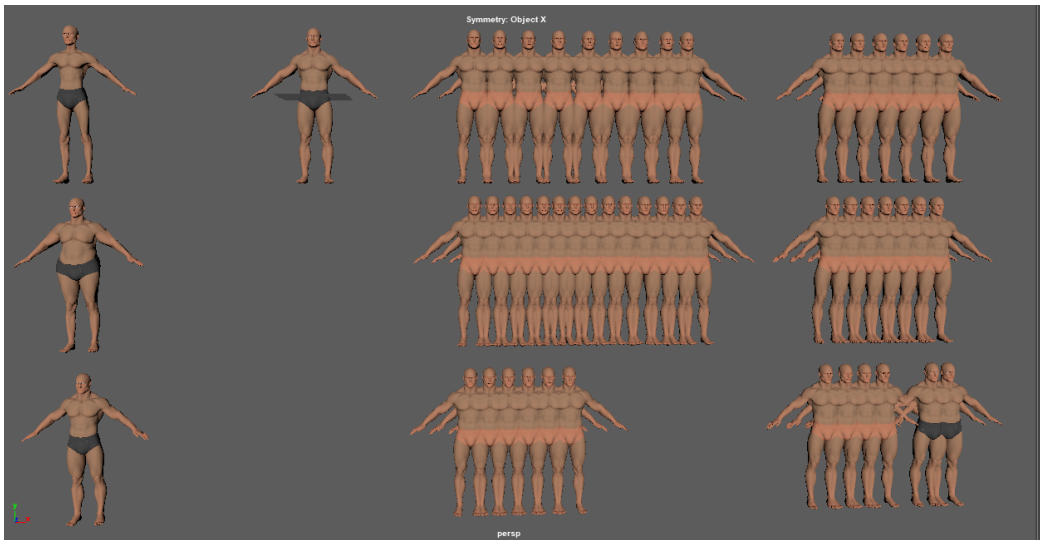

Fig 32. Maya scene with all duplicated meshes

All the deformations will be worked on the duplicated meshes and left the original untouched, as changing the original would cause problems when applying the blendshapes. In order to deform these duplicates, it is important to only use sculpting tools and the soft selection tool, as it is mandatory to only move and/or rotate vertices and never add any additional ones (this would not work as blendshapes are simple transitions moving the original vertices into its new position of the deformation created).
Using the vertex mode, the best tool to use for blendshapes is the **soft selection**. This tool allows you to select a vertex and will also select the closer vertices in a circle range with a lower value when getting closer to the edge of the circle. This range can be augmented and diminished depending what is needed to sculpt and it allows you to move, rotate and scale the vertices in a smooth way as the value decreases as it gets wider into the circle. Maya also includes multiple **sculpting tools** that move the vertices in a circular range too. These tools include the smooth tool (levels the vertices positions in relation with the closer ones), sculpt tool (builds up the vertices), grab tool (drags the vertices selected), flatten tool (levels the vertices positions towards a common place) and some more that are all found in the sculpting menu, including some blendshapes ones that allow you to smooth the deformations in relation to the original mesh or even delete the deformations of some vertices to return them to the original position.

Using these tools it is now possible to create the deformations of the **different types of facial features and body types** in all of these different duplicates. It is important establish all the deformations that are going to be made and to have references on each case, which is why the extensive research done of human anatomy and features

in 2.2 will be put to use for these blendshapes.
All of these deformations can be separated into 7 different parts of the body, which is the way that the duplicated meshes are separated in the scene in the first place:

- **Body Types**: To create the different combinations of body types, the study of *somatotypes* will be considered, which says that any type of human can be categorized combining 3 different body type extremes. For that reason, 3 different blendshapes will have to be done, one for each extreme using the descriptions that William Herbert Sheldon qualified as these.

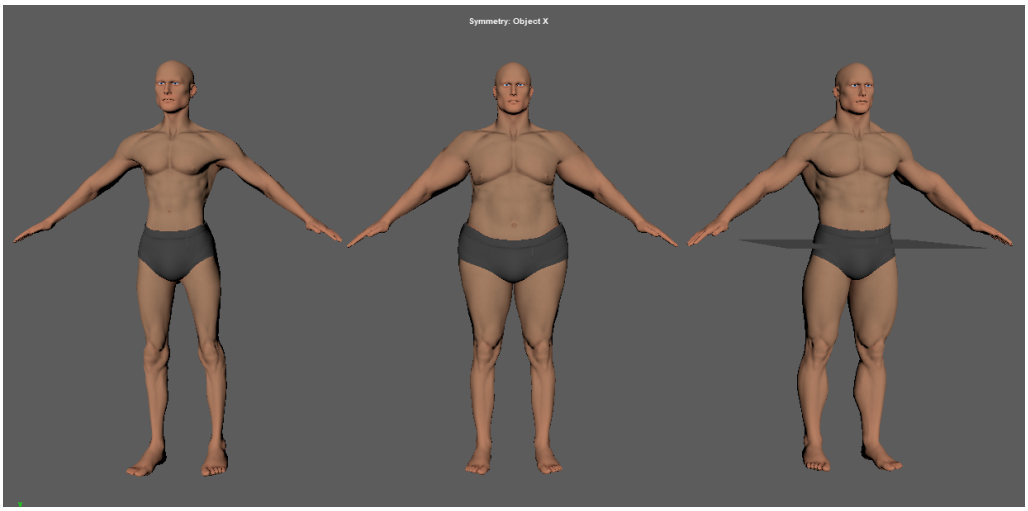

Fig 33. Body Types Blendshapes

For the next parts of the body, the *California Cryobank* facial features categorization seen in 2.2 will be used as reference to create infinite possibilities:

- **Head Shape**: The head has multiple features defining the shape of it, the general head, the forehead and even the jawline, which is why all of these need to be editable. For that reason, the blendshapes that will be created for this will be 3 general head shapes (oval, square and round), forehead size (big and small), forehead width (wide and narrow) and cheekbones position (high and low).

Fig 34. Head Shape Blendshapes

- **Nose**: The nose has the most unique features, as the front shape, profile shape, width, length and nostrils have to be considered. Which is why the blendshapes to create will be the nose base width (narrow and wide), the nose base length (long and short), the nostrils size (big and small), the front shape (straight and round),

the profile shape (straight and round), the width of the point of the nose, the bump on the start of the nose and the nose point position (high and low).

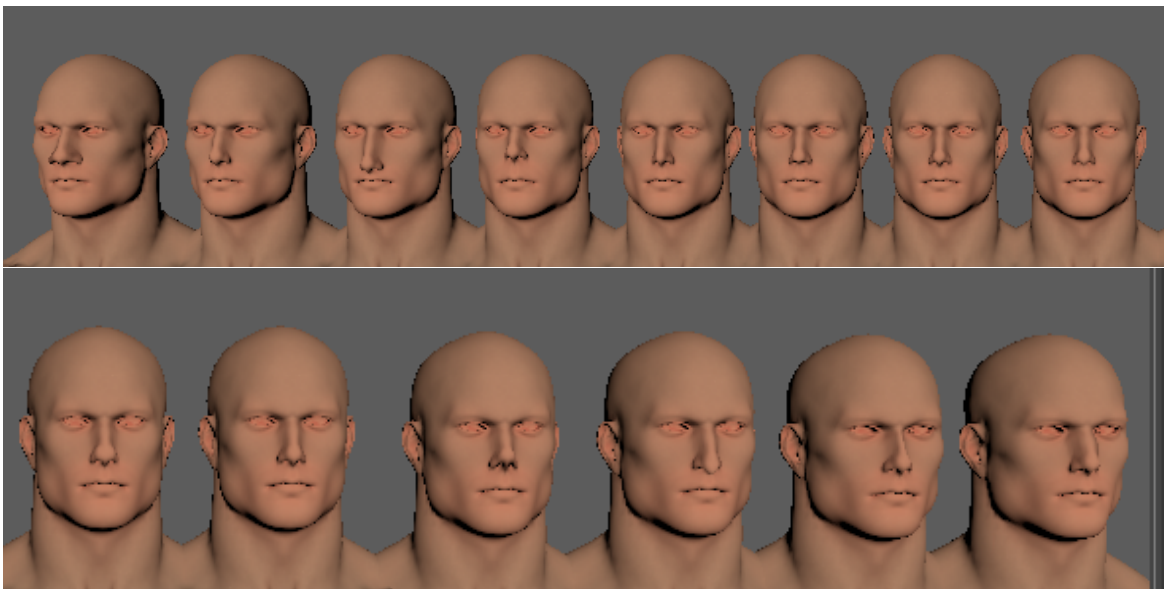

Fig 35. Nose Blendshapes

- **Ear**: The ear does not have that much complexity, but it does have some general features involving size, distance and lobes. Which is why the blendshapes to create will be the ear size (big and small), the ear lobes (attached and detached) and the distance to the head (close and far).

Fig 36. Ear Blendshapes

- **Chin**: The chin can be a very prominent feature of the face and even the shape of it can define the last elements of the head shape. Which is why the blendshapes to create are the chin profile prominence (high and low), the front position (high and low) and the shape (square, round and pointy).

Fig 37. Chin Blendshapes

- **Mouth**: For the mouth, not only the lips have to be considered, but the general size of the whole mouth itself has to be taken into account too. Which is why the blendshapes to create are the general mouth width (wide and narrow), the upper lip size (thick and thin) and the lower lips size (thick and thin).

Fig 38. Mouth Blendshapes

- **Eyes**: The eyes of a person can really define the overall face between its position, size and shape. Which is why the blendshapes to create are the eyes width position (wide and narrow), the eye size (open and closed), and the eye shape (round and almond).

Fig 39. Eyes Blendshapes

Once all of these duplicates are deformed, these can be selected and added to the original mesh by **applying** the blendshapes. When the blendshapes are applied, the original mesh will contain an extra value names blendshapes shown in the inspector with all of the deformations with the name of the duplicated meshes. Even though the blendshapes are now applied, if the duplicate is changed it will automatically update the blendshape applied to the original mesh, making modifications extremely easy. These blendshapes are shown as a value from 0 to 1, that can be activated by putting a 1 in the inspector of the original mesh or by using the *Shape Editor*. The shape editor shows all the blendshapes of the mesh that can be controlled by sliders, and can activate multiple blendshapes together and the result will be shown in real time on the original mesh in the scene.

This way, all the blendshapes combinations can be tested to find any topology errors or vertex overlapping. When one of these badly deformed combinations occur, there is a simple solution that can be done in order to fix these errors. Inside the same shape editor, with both combinations that produce the error put to the maximum value (1), a **combination target** can be added. A combination target is an additional blendshape that consists on fixing a combination of two other blendshapes that cause errors, this works by activating this new deformation when the other two blendshapes have positive values. This is a simple calculation that sets the new blendshape value by multiplying the values of the other two that cause the problem. Using that same shape editor, the blendshape fix can be edited on top of the original mesh to fix any topology error and overlapping that was produced by the other two and will be automatically calculated when both of them are active.

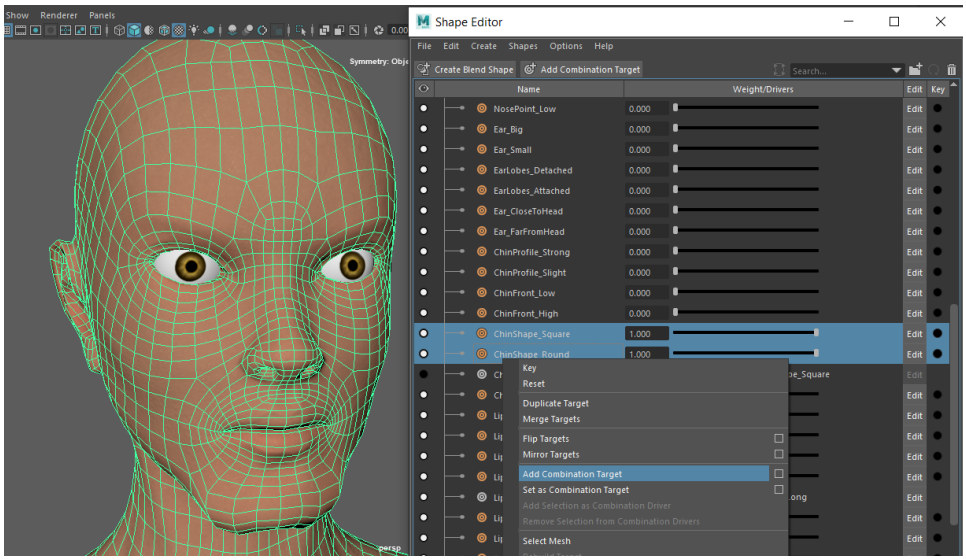

Fig 40. Odd combined deformation with chin shape round and square

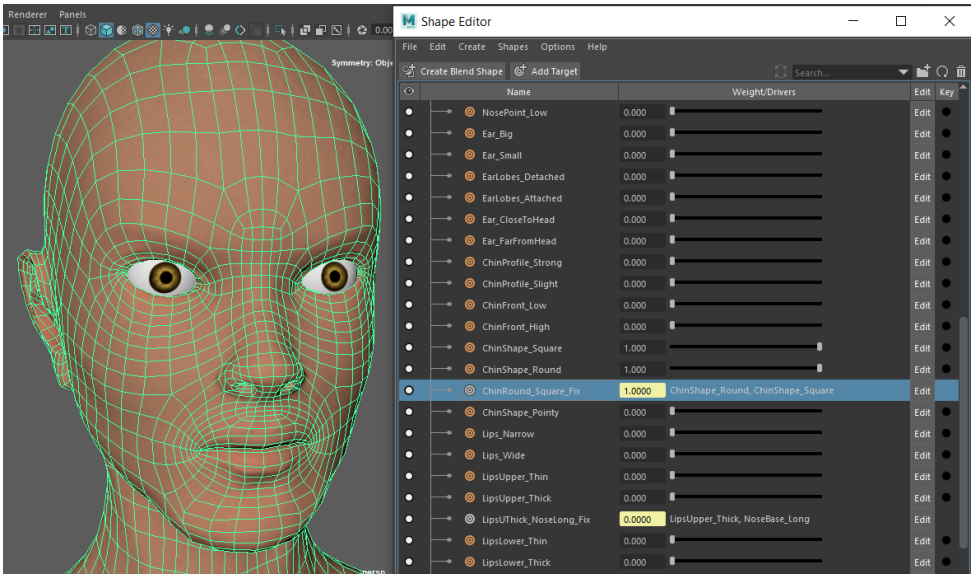

Fig 41. Combination target blendshape to fix round and square chin shape error

Blendshapes are focused on changing only one of the meshes at a time, which is why if the model contains **more than one mesh**, this process will have to be done again for

the additional meshes. A clear example is the case of the eyes or the boxers that are separate meshes from the body.

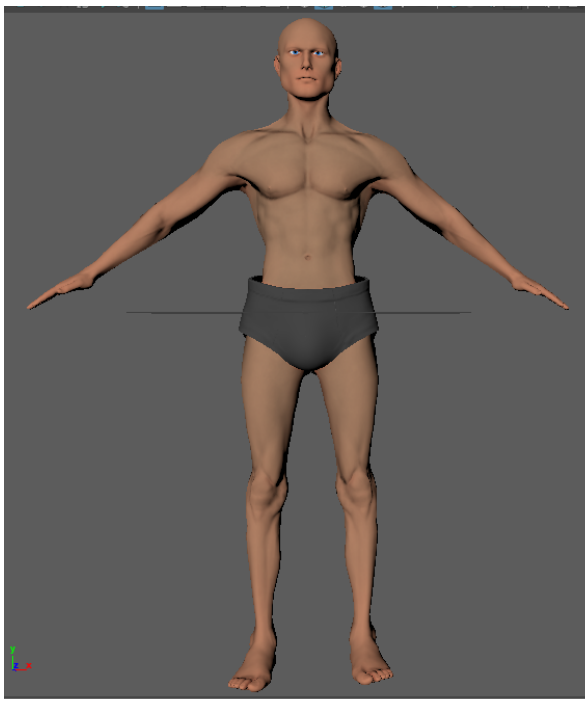

Fig 42. Blendshape only applying to the body mesh and not the boxers mesh

Once all of these blendshapes are applied and fixed all combination errors or final modifications, the model is ready to export and integrate to Unity. The **export process** is not any different as others, simply because the blendshapes are applied inside of the mesh itself, which means that, by exporting the original mesh normally, the blendshapes will export too as an extra characteristic of the model.

5.2.3 Unity UI

The first steps in the creation of the software is to prepare the **scenes** and UI elements inside the game engine, which in this case is Unity. All these elements have to be placed and applied its functionality considering all the previous research and the established UI design in 5.1.3. It is for that reason that the software will contain 3 different Unity scenes, the first one will be the main menu containing the selection between the male or female mesh, and the other 2 will be the male scene and female scene which will be focused on the customization of them.

Inside these scenes, using the UI concept art of each menu, the UI buttons and texts should be placed in the same position as the concepts. These buttons do not have to have the final art style, it can be set as the default art that comes with the engine, as this first step is simply to use them as **placeholders** and assure the functionality of all of them. The final art will be done during the final polishing phase and is not the current focus, the focus is for the UI to be understandable for the target audience and

that the elements can be seen correctly with a full functionality and game loop around menus.

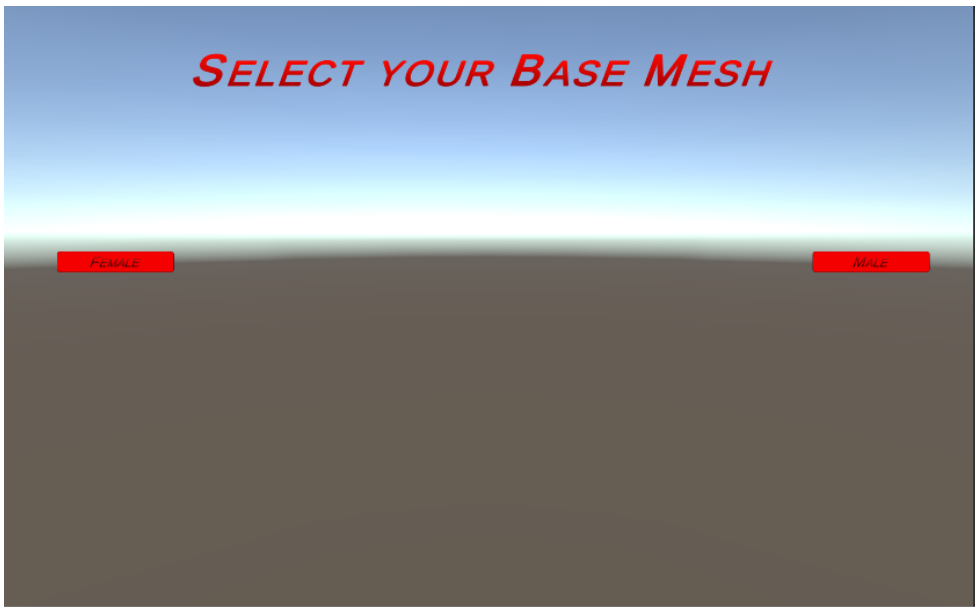

Fig 43. Main menu UI elements

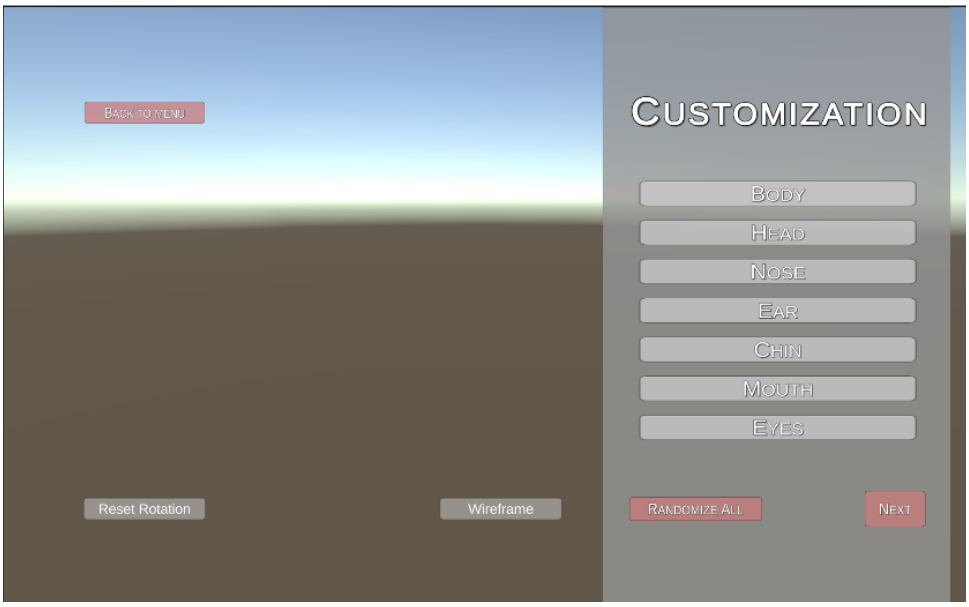

Fig 44. Mesh menu UI elements

Once these elements are placed in their respective position, the scripts are needed to be made to make these buttons functional and loop around the scenes. It is for this reason that a *SceneSelection.cs* script and a *FeatureSelection.cs* script has to be created with these objectives.

The **SceneSelection** script contains the functions to load the other scenes, and it is applied in all scenes in an empty gameobject called SceneController, where the corresponding buttons is given an onClick instruction to access one of the functions of the script to load the next scene. The buttons that have these instructions will be the *male* and *female* buttons from the main menu, and the *back to main menu* in the other two scenes.

The **FeatureSelection** script contains the functions to activate and deactivate the windows of the individual body parts and customization options from the mesh scenes. This script is applied to an empty gameobject in the male and female scene called ButtonController. The functions activate the UI elements of the body part window and deactivate the buttons from the mesh menu, and so on for every button as an onClick instruction. This script also contains the function to skip to the next *customization options* buttons and to skip to the next *export model* window, using the same method of activating and deactivating different UI elements.

The body parts customization windows include multiple **sliders** that also have to be placed, joint with a text title for each slider and a number next to it indicating the value of the slider. When there are a high amount of sliders and do not fit the in the window, a scrollview element has to be applied so that the user can scroll down the different options. Both these sliders and scrollview UI elements are also predetermined already by the engine, which means that it can added very easily just like the buttons and texts elements.

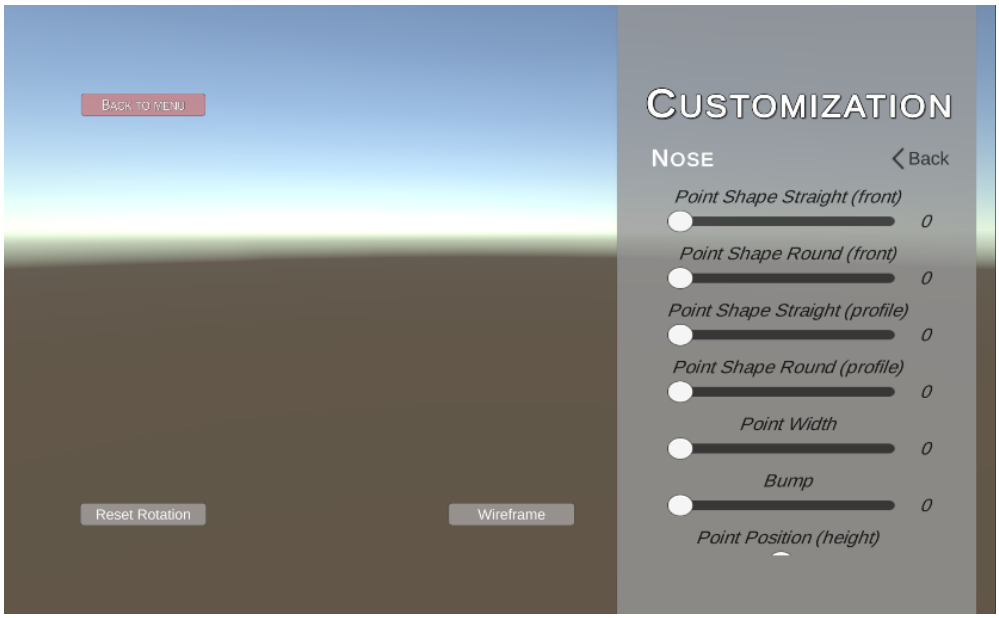

Fig 45. Body parts window with sliders

5.2.4 Blendshapes Integration in Unity

With all the UI elements set and functional, the male and female models have to be integrated to the scene and set the sliders to control the blendshape values of the meshes.

The way to do this is to **import** both the low poly and all the textures to the asset folder. This way it can be used in any scene by just dragging it, but using the textures is a bit more complex. For that, a material has to be created in the asset folder, and all those textures (albedo map, normal map, height map, metallic map and ambient occlusion map) have to be applied to the material in its respective position. With this

done, the material can be dragged to the mesh it corresponds to and it will automatically be applied.

The most important part is to be able to access the blendshapes of the mesh and to be able to edit it in real time, Unity has a very straightforward way to do this. When the mesh of the model is selected, the component of *Skinned Mesh Renderer* is shown in the **inspector** as applied to the mesh. This component contains a section of *Blendshapes* when the mesh has these deformations integrated to it, and inside this Blendshape option there is a list of all the blendshapes applied to the mesh. Each of these blendshape elements in the list contain a slider and a value next to it (that always ranges from 0 to 100) to be able to apply it to the mesh in real time.

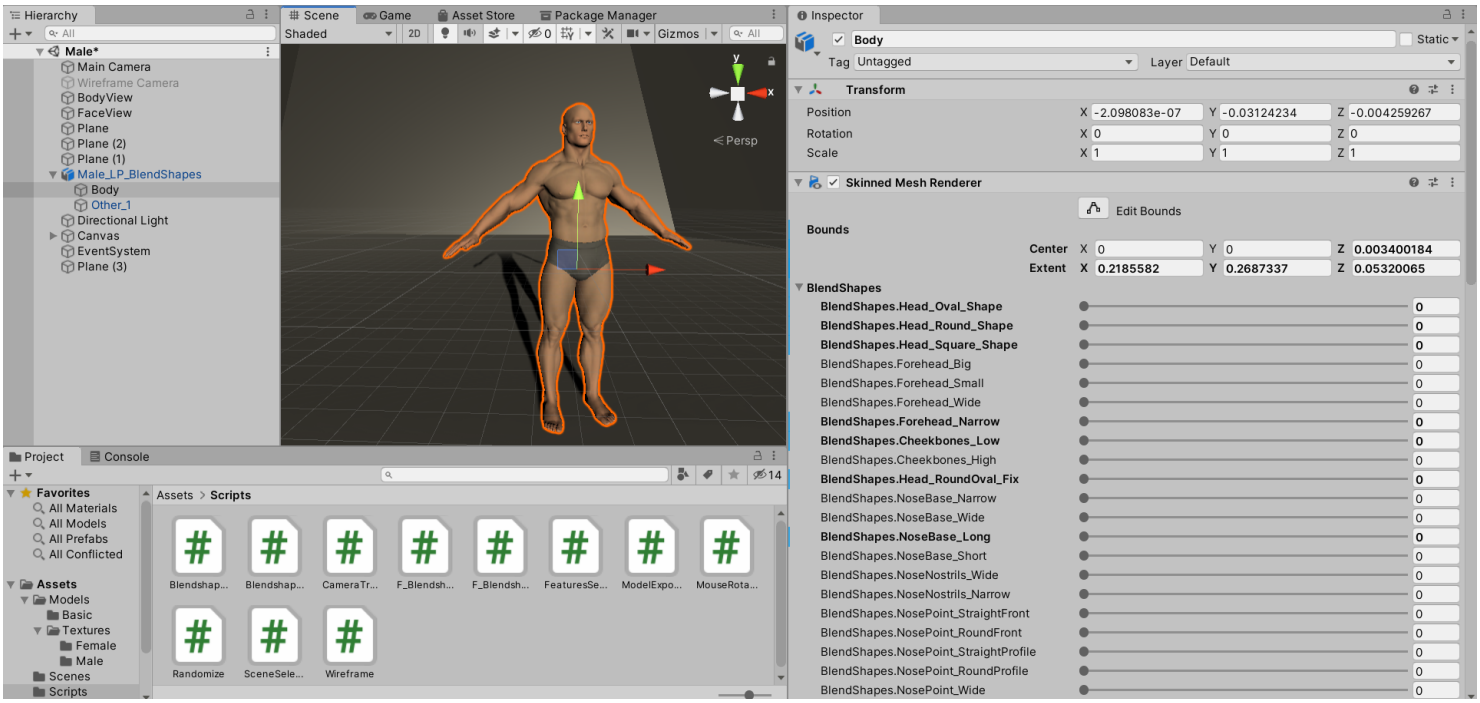

Fig 46. Blendshapes component in Unity inspector

As the mesh already has a component that contains all the blendshapes, it has an easy access through scripts to be able to edit the value of them when a slider is modified. In order to do this, a *Blendshape.cs* script is needed to be applied to the mesh where it is initialized its Skinned Mesh Renderer component. To **connect it to the sliders**, some extra float variables have to be created, which are set to equal to the slider value each time the slider is dragged. The only missing thing is to use those float variable to update the blendshape, which is why this is done in the update function that calculates every frame. To change the blendshape value, the Skinned Mesh Renderer component contains a function called *SetBlendShapesWeight* that receives 2 variables: an integer as the number of the list of all blendshapes and a float between 0 and 100 that will change the weight value of that blendshape. Using this function with the float variables we created and connected to the sliders value, the blendshapes can be controlled in real time by just dragging the sliders placed in the body parts customization window.

This is not only needed to do for each slider, but a separate script is also needed for each separate mesh in order to maintain the structure and coherence of the scripts.

When **blendshapes are opposed** to each other, like a maximum and minimum size of a body part, using two separate sliders does not make much sense. In order to use only one slider to control both ends of the 2 different blendshapes, those sliders will need a minimum value of -100 instead of 0 and a maximum value of 100. With this done, the script would simply equal the negative slider values to the float variables as a positive number (so it can be read in the SetBlendShapesWeight function) to change one blendshape, and the positive slider values to the float variable that changes the opposite blendshape. This way, 0 would be none of the blendshapes, a positive value would only activate the maximum size blendshape and a negative value would only activate the minimum size blendshape.

Something that is not integrated already in Unity is the **combination targets** that were automatically applied in Maya. Unity does not read this as a blendshape to fix combination of others, but it simply reads it as an extra blendshape of the mesh. As explained before, a combination target is an extra blendshape that activates when the other two blendshapes that cause an odd deformation have positive values. The value of this combination target blendshape is simply the multiplication of other blendshape values. That makes it very easy to calculate, as the SetBlendShapesWeight function for the combination target will simply receive the float value of the multiplication between both of the variables that correlate to the blendshapes that are meant to be fixed.

```
5   public class BlendshapesOther : MonoBehaviour
6   {
7       //all blendshapes
8       private float Eyes_Narrow_Value = .0f;
9       private float Eyes_Wide_Value = .0f;
10      private float BodyTypeEctomorph_Value = .0f;
11      private float BodyTypeEndomorph_Value = .0f;
12
13      private SkinnedMeshRenderer SMR;
14      // Start is called before the first frame update
15      void Start()
16      {
17          SMR = GetComponent<SkinnedMeshRenderer>();
18      }
19
20      // Update is called once per frame
21      void Update()
22      {
23          SMR.SetBlendShapeWeight(0, Eyes_Narrow_Value);
24          SMR.SetBlendShapeWeight(1, Eyes_Wide_Value);
25          SMR.SetBlendShapeWeight(2, BodyTypeEctomorph_Value);
26          SMR.SetBlendShapeWeight(3, BodyTypeEndomorph_Value);
27          //Combination Fix
28          SMR.SetBlendShapeWeight(4, (BodyTypeEctomorph_Value * BodyTypeEndomorph_Value) / 100);
29      }
30
31      //UPDATE ALL SLIDERS
32      public void EyesWidthSlider(float new_value)
33      {
34          if (new_value >= 0)
35          {
36              Eyes_Wide_Value = new_value;
37              Eyes_Narrow_Value = 0;
38          }
39          else if(new_value <= 0)
40          {
41              Eyes_Narrow_Value = -new_value;
42              Eyes_Wide_Value = 0;
43          }
44      }
45
46      public void EctomorphSlider(float new_value)
47      {
48          BodyTypeEctomorph_Value = new_value;
49      }
50      public void EndomorphSlider(float new_value)
51      {
52          BodyTypeEndomorph_Value = new_value;
53      }
54  }
55
```

Fig 47. Blendshapes script example

Considering and applying all of these, it will complete all the real time functionality of editing the blendshapes with sliders using all of the elements worked on in Maya.

5.2.5 Beta build

With all the UI elements functional and controlling all the mesh blendshapes through sliders in real time, the beta version of the software is almost ready, but it still has some **missing features** to complete it.

Even though the model is placed on the left side of the scene and the user can see it clearly, some additional options should be added in order to visualize all the parts of the model. Using 3D modelling softwares as reference, the most common user interactions for the **different views of the model** are being able to rotate it, zoom into it and have different material options including a topology view called wireframe.

For that reason, a script *ModelRotation.cs* is needed to be attached to the whole model (not individual meshes), that accesses the model's *transform* in order to **change its rotation** when clicking on the model and dragging the mouse to its desired rotation. This script contains all the programming involving the model's rotation, which is why it should also have the *reset rotation* function that sets the rotation of the model to its initial one, this is ran when pressing the reset rotation button in the mesh menu.

In a customization software, it is important to transmit to the user what he/she is going to change before it is even done. Which is why automatically **zooming** into the part of the body that is going to be affected by the blendshapes is an ideal representation. In order to do this, a *CameraTransitions.cs* script is needed, which accesses the main camera's *transform* and changes its position and rotation to a different view of the model. To make smooth transitions between these views, a small coding animation can be used called *Lerp* and *LerpAngle* functions inside the *Vector3* class, which linearly interpolates the position between two different vectors points in a given velocity.
There are two main different body parts that affect the blendshapes, the whole body anatomy and the facial features. Which is why these two different view transforms (position and rotation) will have to be set to change the main camera to look at the whole body or zoom into the face of the model automatically. When clicking the body types button in the customization mesh menu, the camera transform will be set to the body view of the mesh, and when clicking any of the facial feature buttons (nose, head, eyes, mouth, ears and chin), the camera transform will be set to the face view of the mesh. This should also be able to control manually with keys for the user to change the view when desired, using the '1' and '2' keys, for example, to change from body view and face view respectively.

Fig 48. Body view of the mesh

Fig 49. Face view of the mesh

The **wireframe** view of a model consists on showing all the polygons of the object marked by lines. This should appear on top of the mesh in real time and updating when the model is rotated or zoomed into, showing all the topology of the object. Unity has a render function that changes the camera rendering into a wireframe view of the objects called *GL.wireframe*. In order to apply this, a *Wireframe.cs* script is needed joint with a new camera with the same position and rotation as the main camera. This way the new camera will contain the wireframe render view and the script should also contain a function to swap between the main and new camera (activating and deactivating). This function will have to be ran only when pressing the wireframe button on the object, and will have to return to the main camera when pressed again.

After these scripts, all the user interactions for the different views of the models will be completed, making the customization much more understandable and faster.

Some additional final touches should be applied to fulfill the user's experience, such as including a simple temporary **background** to the scene and being able to randomize all the sliders to produce a completely new mesh every time.
In order to **randomize an update all sliders**, a simple *Randomize.cs* script is needed to be applied to an empty *SliderController* gameobject inside the UI elements group called canvas. The objective of this script is not only to randomize the value of all these sliders, but it also will update the text that appears next to the slider with the value of its corresponding slider. For that reason, the *update* function should equal the value of each slider to its corresponding text, and it should also contain a separate *randomize* function that sets the value of each slider to a random number between 0 and 100 (or -100 and 100 if the slider contains two blendshapes). This last randomize function will only be ran when pressing the randomize all button in the mesh menu.

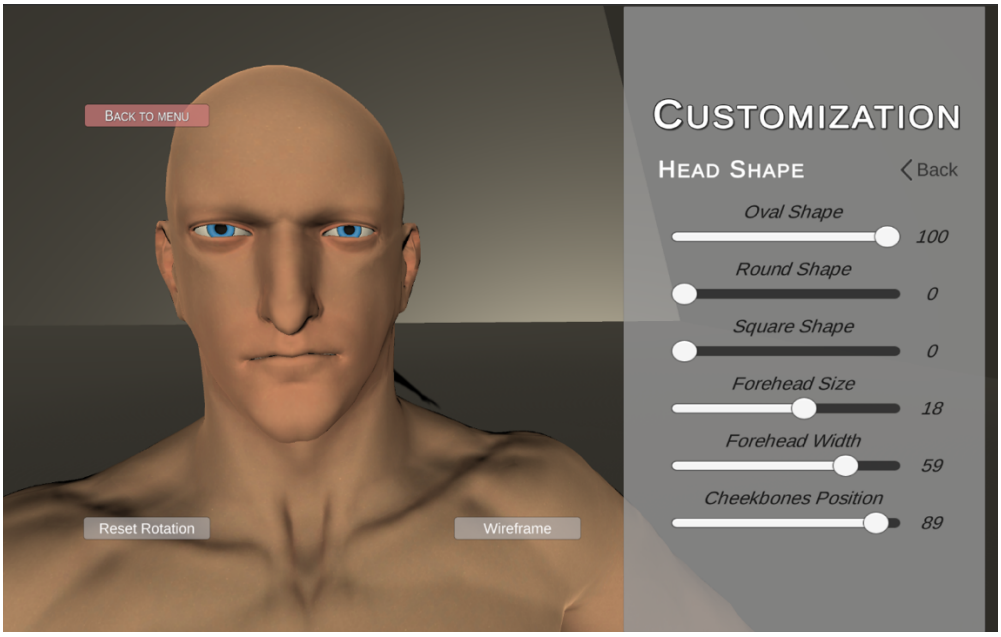

Fig 50. Updated blendshapes and slider values with temporary background

The final missing functionality for the beta version to be ready is to be able to **export the customized model**. Unity contains its own package called *FBX Exporter* that allows you to export models and textures applied to it through the editor with just a click for easy FBX transitions between different softwares. But this exporter is not usable in this case as it does not export the model with the updated blendshape values, instead it exports the model with all blendshapes set to 0.
Kellan Higgins has created a free Unity package with the objective of exporting models in an FBX format too in his own GitHub repository, called *UnityFBXExporter*. This package has a very similar functionality as the one Unity provides, but, with Higgins' exporter, the updated blendshape values are taken into account, making it possible to export the fully customized mesh.
In order to apply this to the model itself after importing the package, a *ModelExport.cs*

script is needed to be applied to the whole model containing all separate meshes. The reason why this has to be done is because the package is meant to be for a Unity editor usage (exporting through the hierarchy instead of runtime). But as the objective is to exporting in runtime, the package scripts functions can be used in this Model Export script to accomplish this. The function *ExportGameObjToFBX* in this package allows you to export a given gameobject as an FBX into a given path file, where it will appear in your computer. This function will only be ran when pressing the export model button in the final menu.

Fig 51. Export mesh menu

Now that the customizable models can be exported, an **optimization test** should be redone to assure that the model can be used for videogame development. This test will should be the same as the one done previously for the base mesh, shown in 5.2.1.

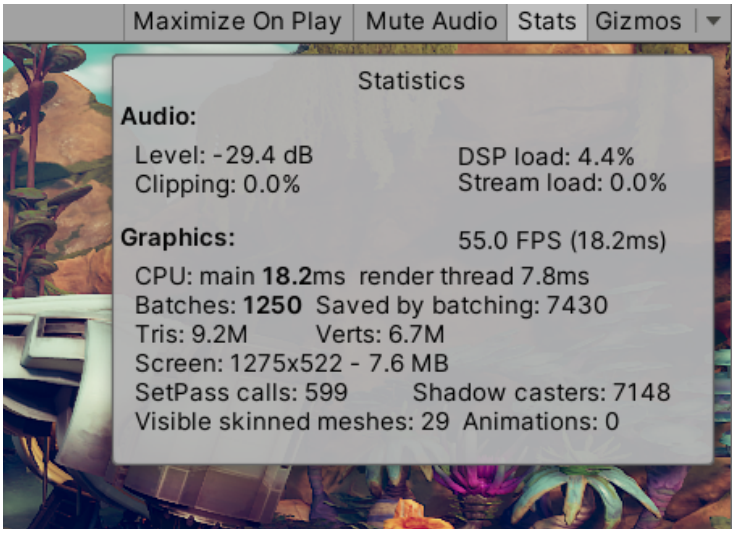

Fig 52. Runtime statistics before importing custom mesh

Fig 53. Runtime statistics after importing the custom meshes on the scene

As seen in the previous two images, there is barely any difference in the runtime statistics before and after including the customized models, meaning that no performance issues are found, so it is optimized for a videogame use.

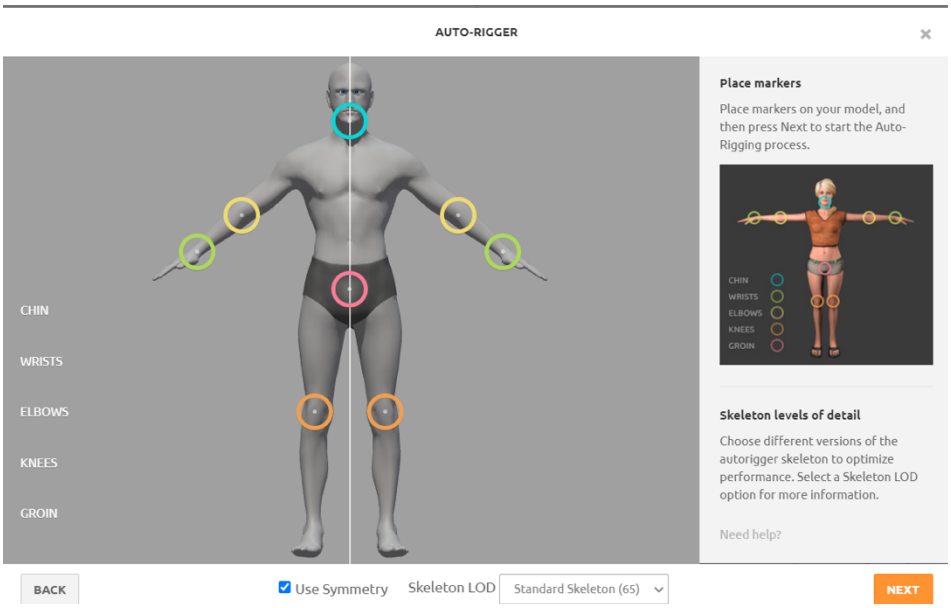

Fig 54. Mixamo autorig in exported mesh

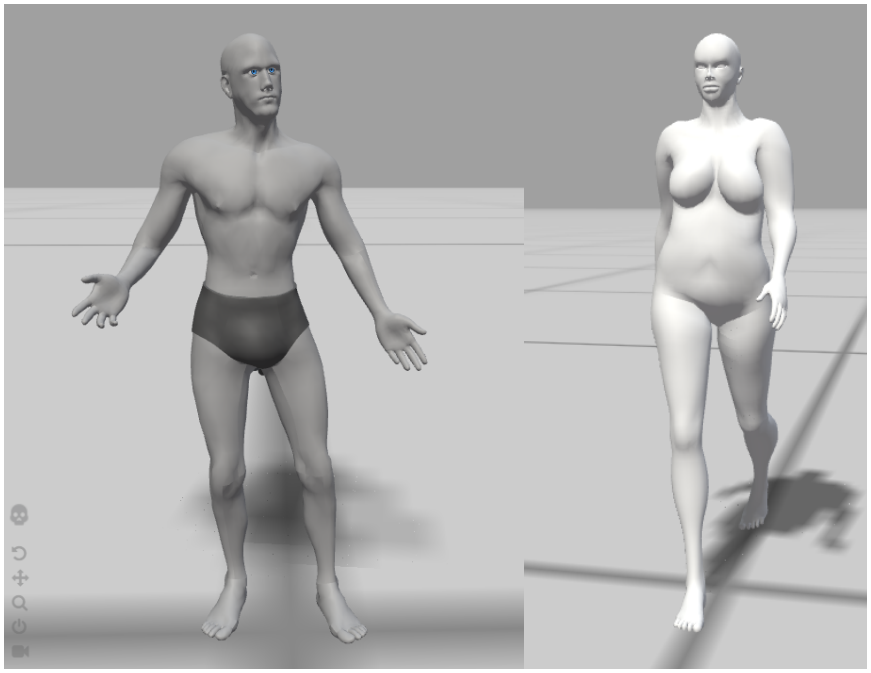

Fig 55. Sample poses/animations in Mixamo for the exported meshes

As the mesh was deformed by using blendshapes and the **topology** was controlled with those combination targets, this was maintained correct and organized in the model, making rigging and animations completely optimized as shown in the example above in Mixamo.

Now that everything is tested and corrected, the beta version is ready to create, which is simply done by creating a build inside Unity of all the scenes. This will create a folder with an executable file that will run the beta version of the software, and the custom mesh will be exported in that same folder.

5.2.6 Issues encountered during the beta build development

During all this development, there are several issues that can be encountered and other alternatives have to be found in order to advance. Some of these issues have already been explained in previous points, such as solving blendshape combination errors by using combination targets or solving the Unity FBX exporter not considering the blendshape values by using an external package.
But other issues that have not been mentioned can also be found:

During the blendshape integration to Unity, if the model is not imported with the correct settings, it can cause some **visual artifacts** when combining multiple deformations together. This artifact shows a damaged texture that is caused when the normals of the mesh are not the same as the blendshape normals.

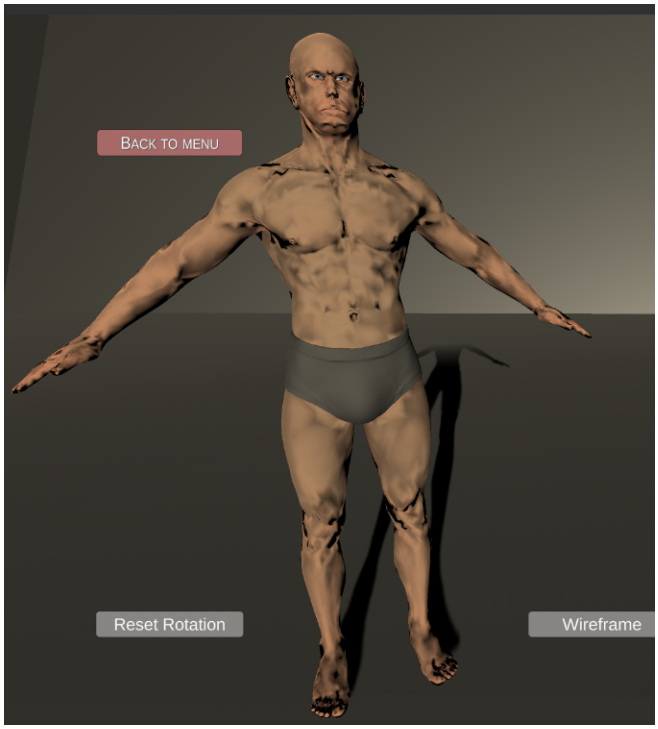

Fig 56. Visual artifacts caused by incorrect normals

This is a representation of this visual artifact when the mesh and the blendshapes do not have the same normal values, messing up the mesh and textures of the object. Depending on your default Unity import settings, this could be set to *import* the normals of the mesh and *calculate* the normals of the blendshapes. As these are different, this is what causes this error as the normals of the mesh are being imported from the model values and the blendshape normals are being calculated automatically by the engine. In order to fix this, both of these normals should be changed to the same method, either both calculated or both imported, although it is recommended to **import both normals** as it has already been tested previously.

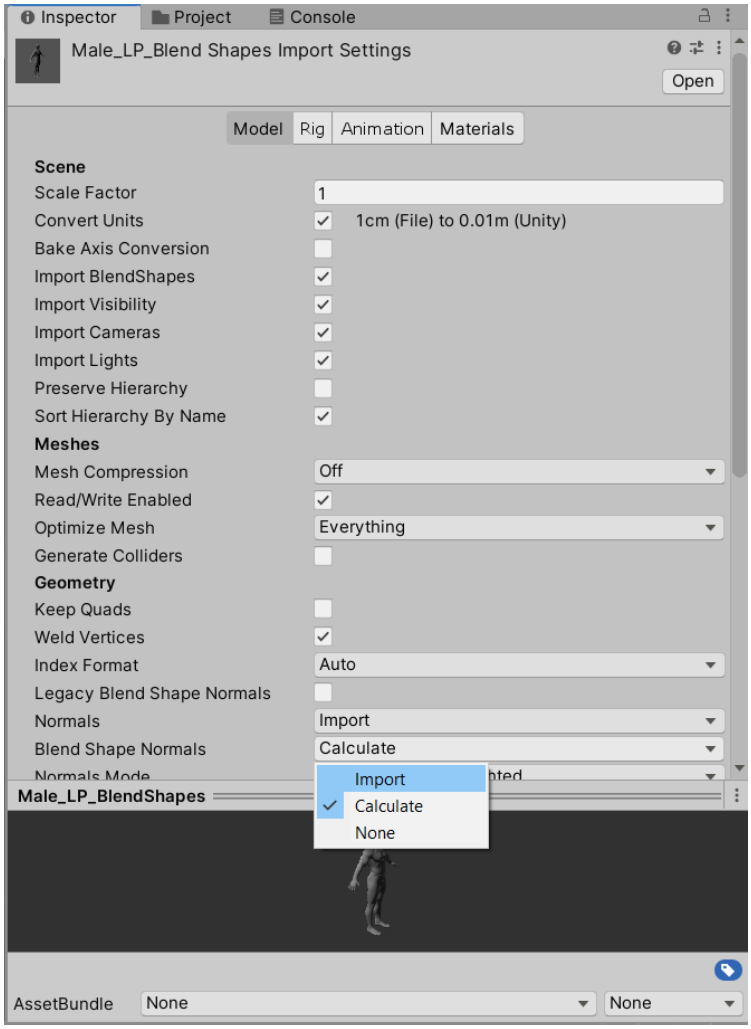

Fig 57. Mesh import settings window containing the mesh and blendshape normal information

The external Kellan Higgins **FBX Exporter** package is useful as it considers the blendshapes values, but it does not export the **textures** through a runtime function. This can easily be solved by simply exporting the textures separately in the same *ModelExport.cs* script.
The function used to export in runtime with this package needs a given path to place the FBX file. If this **path is not accessible** or incorrect, the model will simply not export at all. Which is why, in order to assure that it will export in the build, the path can be set by using the *Application* class. This class can access the folder where the executable is in, so to be completely sure that the path works in all computers, it can be set by using *Application.dataPath* as the given path to the function.

Lastly, the **wireframe camera** using the *GL.wireframe* function can cause some errors and also show different results between the Unity runtime and the Unity build. Another alternative can be used, where instead of rendering the whole scene as wireframe, a new material can be applied to the mesh that displays in wireframe.

5.2.7 Clothing Models

It is established that in order to make customizable clothing for the character, multiple separate types of clothing have to be placed on top of the base mesh. This way the user can switch through the different clothes to choose the preferred one. All of these clothing meshes should individually have their own different color options to add more customizability.

This is why the first step is to **search all the separate clothing models** in all 3D modelling websites such as sketchfab, cgtrader, free3d and artstation. These models have to meet a criteria in order to be optimized and useable for the software and for videogames. The most important is to consider the **optimization** aspect of the model, knowing that the base model is around 10-15k polygons, and that the aim is to not surpass 30k for the whole exporting model as explained previously, it is easy to deduce the polycount of the clothing models. A maximum polycount must be needed to set when searching for these models, which concludes in 5k as the maximum for each top (ex: shirt) and bottom clothing (ex: pants). As for the shoes and other complements, it should not pass from 2k polygons as it does not need that many to have a good shape with a correct normal map because of its size. This would give the maximum polycount of the whole model between 22-27k, and would have the margin to add the only remaining mesh, the hair, to contain around 3k polygons.
The other criteria that are needed to be met for these models are that it needs to be a correct **art style**, which is realism, in order to match with the base models and any props that designers would use in a further development of their videogames.
When getting models online, in order to save as much time in this software development, it is important to check that the models are fully done and, for that reason, contains its different maps such as normal map. This would mean that it is not only a low poly of the model and that the high poly and details of the mesh have been created and put into the normal map, which would save a lot of time by not having to do the high poly mesh. This is why the las criteria is that all models should need to include, at least, their **normal map** on top of the mesh itself. If it also contains a base color map with all the painting done it would be preferable, but it is not a criteria as all the different color options will have to be done in Substance Painter either way.

Clothing can be very extensive as there are a million different types of clothing, styles and shapes for each individual garment. Making or finding all these garments in the amount of time given is impossible to do, which is why there are limits that need to be set and choose the most **basic** and **important types of clothing**.
The clothing will have to be slightly different for the male and female meshes, but the **male** will include these following garments to choose from:

- *Top part*: Male Jacket, Shirt, Tank Top and Male T-Shirt.

Fig 58. Top Part Garments for male

- *Bottom part*: Joggers, Shorts and Jeans.

Fig 59. Bottom Part Garments for male

- *Shoes*: Formal, Skater (DC) and Sneakers (Converse style).

Fig 60. Shoes for male

And, for the **female** model, these will be the garments available:

- *Top part*: Female Jacket, Dress, Tank Top and Female T-Shirt.

Fig 61. Top Part Garments for female

- *Bottom part:* Jeans, Shorts and Joggers.

Fig 62. Bottom Part Garments for female

- *Shoes*: Crocs, Sneakers (Converse style) and Skater (DC).

Fig 63. Shoes for female

These models represent the most basic clothing both male and female use, and will give the option to choose from and change from multiple colors and prints.

Even though theses garment meshes are fully done when downloaded, some tweaks have to be made in order to **fit on the base mesh.** All the polygons and vertices of the clothing meshes have to be on top of the base mesh, which should be completely covered by it in its respective position. This is very important especially for further on when the model is being rigged, where animations can show the base mesh poking out of the clothing in random areas, giving a very bad visual experience to the player. These tweaks can be done in the same way as modifying meshes during the creation of blendshapes in Maya. The garment mesh has to be placed in the same project as the blendshapes Maya project, and rescaled and transformed on to the top of the base model. Once this is done, using the soft selection with symmetry on, the vertices can be moved to ensure that all of them are visible and covering all the base mesh polygons beneath them. It is also important to do the garments that are underneath the rest first (bottoms like pants and shorts) so that the vertices and polygons of the clothing that goes over it can be placed just on top of them.

Fig 64. Modifying original clothing to fit to the base mesh

This will work for the clothing when the body type chosen is the one of the base mesh, but if the user decides to change the body type, the clothing will not fit correctly. This can be fixed by simply doing **blendshapes on each garment mesh** corresponding to the body types blendshapes of the base mesh. The garment mesh can be placed on top of the different body types blendshapes and can be modified in order to correctly fit these base mesh blendshapes. This is needed to be done for each piece of clothing and this way all the clothing can be updated to the body type chosen in the initial process.

Fig 65. Clothing Blendshapes in Maya

The last step to be done with the clothing is **painting** each garment into multiple colors to provide different options to the user. Using Substance Painter, is it a very easy process, where the individual garment can be imported with its corresponding maps and a simple cloth material can be applied to it. These smart materials can be modified by visualizing all the different layers that produce the material, and, for that reason, can be changed its color. Once a color is completed, only the base color map is needed to be exported as it is the only one that will be modified in the color selection process inside the software. Multiple base color map variations have to be exported for the garment mesh with a significant difference in color or with additional prints using the projection tool. This process has to be done for each single garment used in the software in order to complete the whole clothing modelling process.
The colors that created for each garment are black, white, red, blue, pink, green, brown and yellow, although each garment will have different options between the mentioned ones depending on what fits the garment the best (most used or visually attractive).

5.2.8 Clothing Importation

In order to import all the clothes with its blendshapes in the correct position in Unity, the **export** process has to be taken into account. Autodesk Maya will export the center position of the models selected, which means that the initial position of the exported mesh will be the center of the selected objects. If a singular garment like a shirt is exported by itself, the initial position will not be the same as the base mesh, which would mean that the shirt would have to be transformed inside Unity to fit to the base model. This can produce some inaccuracies that cannot be afforded where the base mesh can poke out on top of the clothing. The **solution** to this is to select the base mesh and al the clothing at the same time and exporting it all at once, this will mark the initial position to the same one as the base mesh and will fit perfectly inside Unity. These clothing meshes will appear in a group as it has all been exported at once, and this group will have to be unpacked and placed as a child of the base model, in order for the clothing to rotate at the same time as the base mesh with the script that was already created in 5.2.5 called *ModelRotation.cs*. Finally, the extra base mesh that was exported can simply be deleted from the scene so that only the clothing is showing.

Once all the clothing are set in both male and female scenes and the blendshapes are checked to be correct, all the maps (including the base color map variations) have to be imported too. Using these, the **materials** can be created with one of the base color map variations that will be set as the initial color. Further on, using a script, this can be changed by accessing the material albedo map. These materials have to be assigned to its respective garment by dragging them into them, the same way the base mesh materials were set and assigned.

Fig 66. Clothing in Unity

In order for the **clothing blendshapes** to update when the body types are changed, two scripts *ClothingBlendshapesFemale.cs* and *ClothingBlendshapesMale.cs* have to be created. These scripts will access the body types sliders and all the clothing meshes, and will change each garment blendshape when the slider value is changed, the same way that was done in 5.2.4 with the base mesh blendshapes but for multiple garment meshes at once. This way, when the clothing types are chosen, the garment will fit the base mesh correctly and will not have any visual artifacts of polygons poking out on top of the clothing.

Fig 67. Clothing Blendshapes in Unity

5.2.9 Hair Models

As mentioned previously in 3.6 in the planning changes, all the different hair style models were initially going to be created using fibermesh in ZBrush, but to increase the customization options, these models are needed to be found online.

Although the majority of the hair models were found online, two hair meshes were created using this **fibermesh** tool in ZBrush.
Fibermesh permits the user to select an area of the mesh where hair is wanted to be added, which then created multiple rectangles to simulate hair in that area. This generally created a very high amount of rectangles that are also divided in multiple segments each, meaning that the polycount is very high and not optimal for videogames. Thankfully, fibermesh contains a modifier menu that can allow you to optimize this polycount and preview it in real time. Using certain parameters of this modifier, the objective of creating hair cards and a maximum of 3k polygons can be easily completed.
The *fibers*, as shown in the modifier, refer to these rectangles that produce the hair mesh, so lowering the *MaxFibers* sliders, will simply lower the amount of rectangles which is in the interest of lowering the polycount. This can produce certain gaps as the rectangles might be too thin, which is why the length, width and coverage parameters can be tweaked in order modify the hair cards to fit and fill up the base mesh. The last important parameter in order to complete the optimization is the segments slider. The segments are the amount of polygons in each fiber (or rectangle), this is what produces the hair shape, simulating a curve depending on its gravity, which is why this can be lowered to a minimum of 3 segments in order to maintain the shape but still achieve the polycount desired.
Once the polycount and hair cards are finished, these fibers can be combed with the groom brushes that ZBrush provides to accomplish the hair style desired.

Fig 68. Fibermesh Modifier Menu

The rest of the hair meshes have to found **online**, but these will need some requirements in order to match the other meshes and be optimized. The **requirements** consist on: a maximum of 3k polygons, hair cards (rectangles) to be able to apply an alpha texture, realistic art style and visually pleasing.

When these models are found, the integration and **fitting to the base mesh** are not that complicated, as it can simply be modified by using the *soft selection* in Maya to fit the shape of the head. As the shape is different for the male and female models, the step will be needed to be done for both meshes.
It is also important to take into account the different blendshape and clothing possibilities, meaning that a long hair style could overlap with the shirt the model is wearing. For this reason, all the blendshape sliders should be checked in order that it does not produce visual artifacts on the hair meshes and all the clothing should be active. The hair meshes will need some tweaking that can be done with the soft selection in the same way to minimize the overlapping.

The main idea was to be able to increase the customization by applying **length blendshapes** to these hair styles, but as all the hair cards are placed on top of each other due to its creation by hair generators, the end vertices for each hair card was close to impossible to select to simulate the length of the hair. Using the soft selection would select the vertices of the area close to them and moving them produces visual artifacts of overlapping hair cards, and using the surface option in the soft selection still made it impossible to access certain vertices and still produced visual uncertainties in the blendshapes. Due to this, the hair length blendshapes were not created and more hair styles were needed to be found in order to provide more customization.

The final hair models for the software were these:

- Male: short, shaved side, ponytail, middle comb, back comb, messy, mustache and beard.

Fig 69. Male hair styles

- Female: long, double ponytail, short, shaved side, ponytail, middle comb, back comb and messy.

Fig 70. Female Hair styles

As these are hair cards (simple rectangles), the **painting** is very important, where each rectangle should have a transparency that simulated very small hair fibers and accomplish a realistic hair style. These can be made in two basic ways, separating the color of the hair (albedo map) and the alpha map or simple having a base color alpha map. This base color alpha map is a transparent *.png* that contains the functions of an alpha by having a transparent part in the map, and functions of the albedo by having the color in the non-transparent part of the map.
The base color alpha is the one that is going to be used simply because Unity has an easier integration to it, in contrary to the other option.
This hair strand base color alpha map can be found online, which simulates the multiple fibers of hair to create that realistic effect and the color of the hair. Once this is found, the different variations of colors can be easily made in any 2D painting software such as Photoshop accessing the color curves of the image.

The different **colors** created are black, blonde, grey, brown (brunette), blue, red, pink and green:

Fig 71. Hair color variation maps

5.2.10 Hair Importation

The hair styles meshes exportation from Maya and **importation** to Unity is the same as the clothing meshes, explained in 5.2.8. All the hair meshes are exported at once joined with its corresponding base mesh to maintain the position when imported in Unity. This is then placed inside the models gameobject in Unity's scene hierarchy for the rotation to work with the existing base mesh and the extra base mesh exported is deleted. This way, the hair models are placed exactly in the same position as it was in the Maya scene.

Setting the **material** is a little bit more complicated than the ones that were already created. The hair needs a material that supports alpha textures and that applies the transparency to the hair cards in order to simulate all the hair fibers. This can be done by changing the shader of the material created, where a list of different Unity supported shaders are shown and a few of them can actually allow this type of hair transparency with the base color alpha like the *Tree Creator Leaves* shader. Although this is a valid shader to use, an external package asset with an specific hair shader can be downloaded for more accuracy and more control in the hair meshes as it is specifically for hairs. This asset is called *Genesis Hair Shader* and provides more hair

options in the material like a base color alpha texture, richness of hair, wetness of hair and an alpha cutoff to control the amount of transparency inflicted in the hair cards.

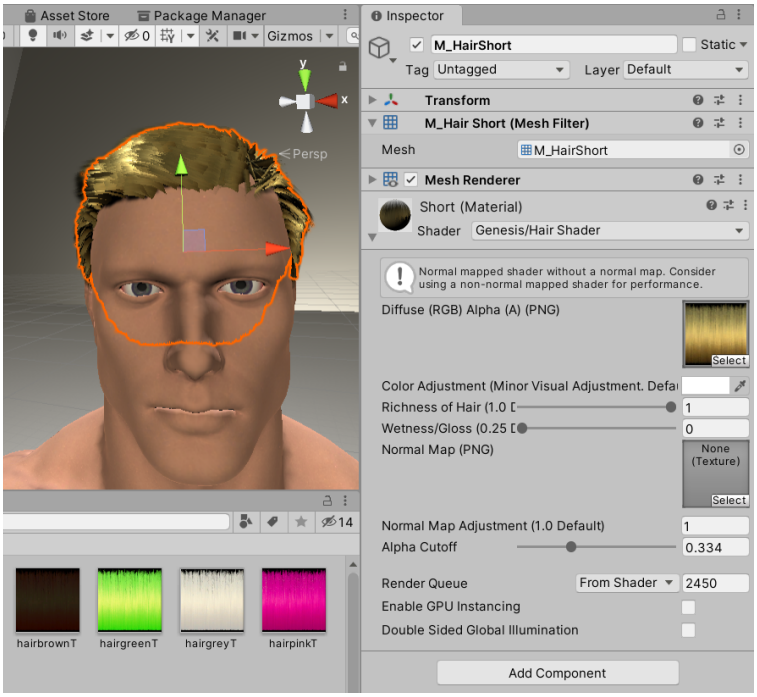

Fig 72. Genesis hair shader

5.2.11 Skin Tones and Eye Color

The initial **skin tone** and eye color was already made and explained in the creation of the base mesh in 5.2.1. All the textures (normal, base color, height and metallic) were already created and exported using Substance Painter, but the different variations of base colors are still needed to be painted and exported.

For this reason, using that same Substance Painter file for the base mesh, **variations** of different skin tones and eye colors have to be painted and only the base color map of these have to be exported (as the other maps do not change).

For the skin tones, an image of different skin tone qualifications was found in order to reproduce the color with a more accurate result.

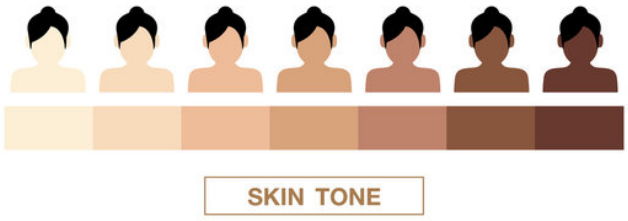

Fig 73. Skin tones reference image

Accessing the skin material applied to the mesh, the color component can be changed and even pick a color that is showed anywhere on the screen with the eyedropper tool. Using this tool and the reference image, all the different skin tone can be applied in that exact reference color and the base color map for each skin tone can be exported, completing a total of 7 different variations.

Although the initial **eye painting** was done by projecting an existing image, this can end up being in different positions of the eye ball when done with multiple different images. For that reason, it is better to use one of many existing eye materials that can be found in substance share, an website dedicated to sharing free materials for Substance Painter. The **material** selected for this project is the eye material created by *Ken Jiang* as it allows the user to change the coloring of the color and more other parameters. This material will have to be imported to the substance project, added to the mesh and scaled down to the correct position of the eye ball.
Once this is done, the color variations are created and exported the same way as the skin tones. The reference image of different eye color that was gotten is the following and it contains several colors to pick from to get the best visual color for the material.

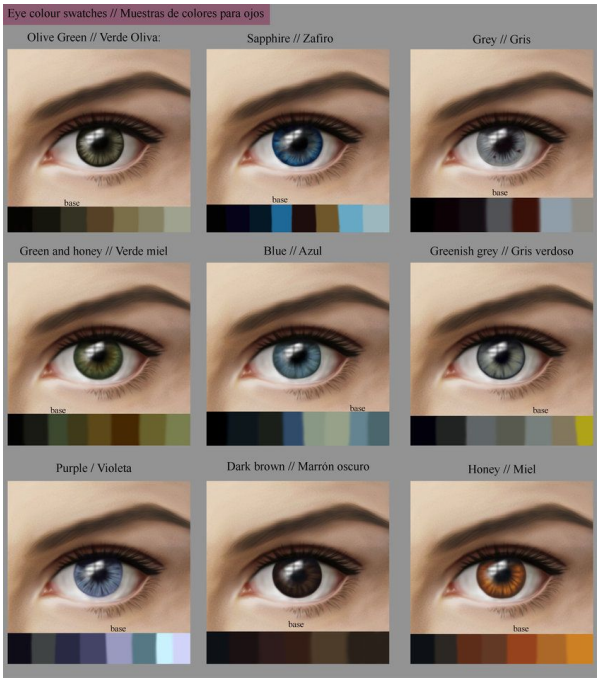

Fig 74. Color variations reference image

The **final colors** chosen from this image for the different variations are dark brown, light brown, dark blue, light blue, dark green, light green and hazel.
Changing the materials color with these reference images, all these different eye color variations can be created and exported each base color map.

Once all of these base color maps are exported for both skin tones and eye colors, these can easily be imported to Unity with different names and this will conclude all the modelling and painting that was needed to be done for the whole software,

meaning that the only thing left to finalize the software is the implementation of these.

5.2.12 Multiple mesh and Color options Implementation

In order to implement all these clothing attires, hair styles and different color variations, all the **UI** inside of Unity is needed to be done.

This means that a **second customization menu** has to be created with all its respective choices: hairstyle, eye color, skin tone, upper clothing, bottom clothing and shoes as shown in *Fig 19*. This has to be based on the UI design that was made in the preproduction phase and explained in 5.1.3.
When pressing each of these buttons, another menu should be show to customize the respective part of the body in terms of selection of meshes and color picking, just as the UI concept done in *Fig 20*. This menu has to be done for each button with its respective options that are available for that part of the body.

Fig 75. Customization part 2 UI

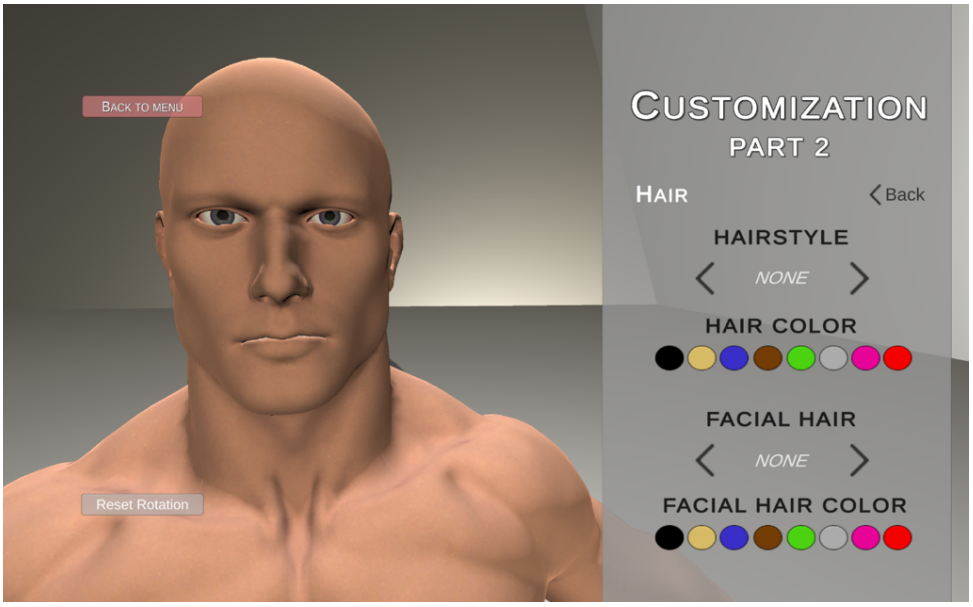

Fig 76. Hair customization UI menu

Once these are created, the continuity and **functionality** across menus can be done the same way as the first part of the customization, using the FeatureSeleciton script explained in the Unity UI 5.2.3. This should be allow the user to go across all menus and buttons by activating and deactivated the respective menus.

In order to select the different available meshes such as hair and clothing in real time, a complex MultipleMesh script should be created for each base mesh (male and female). The functionality of this script is not only to allow the user to select and see the selected mesh in real time, but to also allow the user to pick the different available colors and apply it to the mesh in real time.
All the available hair and clothing meshes are inside the base mesh gameobject (as seen in 5.2.8 and 5.2.10 in its implementation) which makes it easier to find using the script and searching its gameobjects child's. All of these are needed to be initialized and set to inactive to not show all the meshes until they are selected. Which is why, in order to **implement the mesh selection**, the script basically activates and deactivates the gameobjects when the next and previous buttons are pressed. This is done by using a simple counter, starting from 0 and going up or down when clicked next or previous, with the maximum being the number of mesh options that part of the body provides. And using a switch regarding this counter, one of the meshes will be set to active while all the others are set to inactive, making the selection in real time be possible and allowing the user to select all of the available options. Inside this switch, the UI text showing the name of current mesh can be changed too to its respective name when setting it active.
This process has to be done for all the different mesh selections in the customization menus, which are the hair, beard (only male), upper clothing, bottom clothing and shoes.

As mentioned, the **color picking** and real time color change is done in the same script as it already has the meshes initialized. Some of the meshes have different available colors than the rest, which means that the color buttons are needed to be changed depending on the mesh that the user selected. For that reason, all the color buttons have to be initialized in the script and, using the previous switches to detect the mesh selected, the color buttons will also have to be set active or inactive in each respective mesh depending on its available color options. This means that, for example, it will show 8 color buttons on a t-shirt when selected, but when switched to a jacket it will only show 6 of them as only 6 different color variations were made.
Once this is done, the only thing left is to **apply** the different albedo files to the material of the mesh in real time when a color button is selected. A simple Unity function in the renderer component of the mesh can allow you to access and edit the material of the mesh, which makes it optimal for the objective that is needed to accomplish as it also works in real time. Once all the different albedo textures are initialized in the script, multiple on click functions have to be created for each color button on all menus. This will mean that the function will only run when the button is pressed. Inside these functions all the different color changes have to be made for all the meshes available in that selection option even if they are inactive at the time, this way the user will not have to press the blue color every time he/she changes the selected mesh. To change the color of the material's object, the simple Unity function mentioned before, called *material.SetTexture()* inside the renderer component of the object is used. This function allows you to choose what texture it is you are trying to change from the material (which in this case is only the albedo file) and asks for the new texture you want to apply to it. This way the object's material will change the albedo file to its new one and will simulate the color change when pressed that respective color button in real time.
This has to be done for all the different color options that all the menus provide, which are the hair, beard (only male), eye color, skin tone, upper clothing, bottom clothing and shoes.

All in all, this will leave the model's gameobject with its respective childs set to active if they are the ones that are selected and will be inactive if they are not the selected ones. On top of that, all of the meshes materials will be updated to the last color button the user pressed.

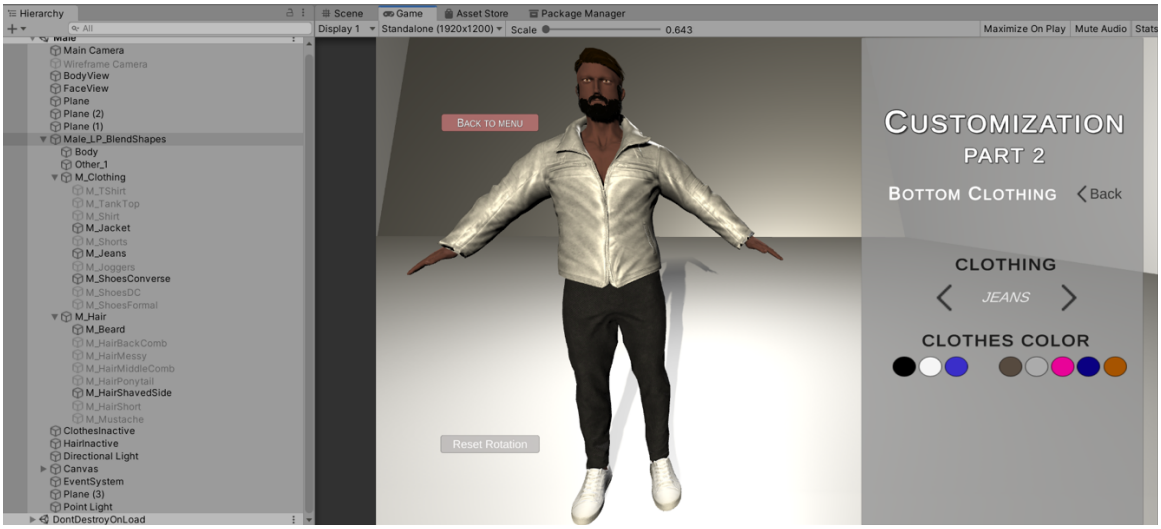

Fig 77. Inactive/active gameobjects in hierarchy after selecting meshes

5.2.13 Export Model with Textures

The **exportation** of the model, although it was already implemented and explained in 5.2.5, it now needs some changes and additions to the ModelExport script in order to export the base mesh, the selected meshes and all its textures.

The exporter used to get the fbx is an external package that exports the selected gameobject including all of its childs, whether they are active or not. This is a problem as all the inactive gameobjects that the user did not select will still be exported and will conclude in an fbx with every single hair style and clothing meshes options. The ideal way to fix this is to access the fbx exporter script and add a for loop that checks all the inactive childs in the gameobject and simply not add them to the exportation process, but, as this script was imported, it is hard to understand its process and meddling with it could produce more errors than fixes. For that reason, it is easier and safer to delete or **move the inactive childs** out of the model's gameobject before calling the export function as those meshes are the ones that the user did not choose to use.
In order to do this, an empty gameobject outside of the model's one is created called InactiveMeshes where all the meshes that are not going to be exported will be placed temporarily during the exportation. When the export button is clicked, the script should check all of the optional meshes and changed its parent to the InactiveMeshes parent when it is shown as inactive. **Changing the parent** of a gameobject can be done by accessing the gameobjects *transform* component and setting it to a new gameobject of choice. Using this, it is possible to move the inactive meshes inside of the model's gameobject outside of it to its new parent, InactiveMeshes. This way, only the active gameobjects inside the model's gameobject are present and will allow the user to only export the clothing and hair meshes that were chosen by them.

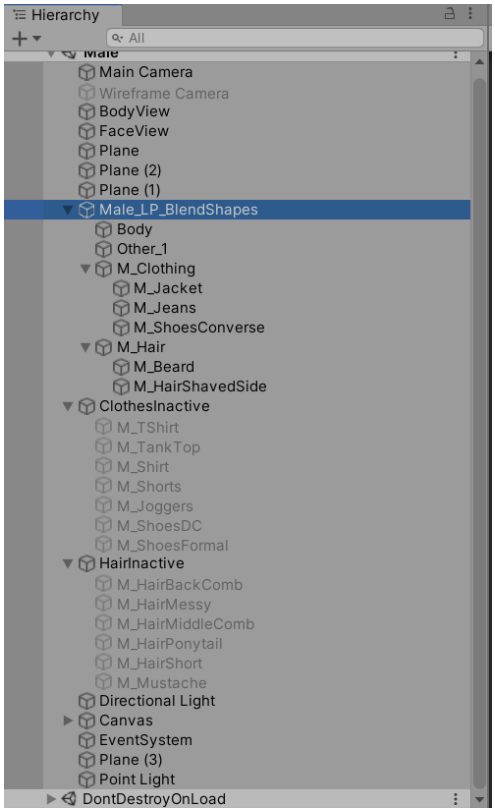

Fig 78. Result of changing the parent of inactive gameobjects

Now that the fbx exportation is ready, the only thing left to finalize the software is **exporting the textures** of the selected meshes and selected colors.
The albedo files of each meshes material can change during real time, which is why it can be done by either tracking the colors selected inside the script to know the albedo file that is being used or by accessing the material and getting the albedo file from it. The second option is clearly more ideal and can be accessed the same way as the albedo textures were set, in the renderer component. On the other hand, all the other textures (height, normal, roughness, ambient occlusion and metallic) of the meshes do not change, which makes it easier to simply export the original texture.
It is for that reason that, inside the export button function, more things are needed to be added.
The first thing that is needed to be done is to know what meshes are active, which can be done with simple *if*s. Inside these, we can simply export the original textures that are not going to be changed, which are the height, normal, roughness, ambient occlusion and metallic of that respective mesh. But as the albedo texture can change during real time, the meshes material has to be accessed in order to get the correct texture used. The renderer component can allow you to **access the material** of the mesh, and using *material.mainTexture* this will return the albedo texture that is currently being used in the mesh. This way all the textures that are being used in real time in the selected meshes are exported using the *System.IO.File* of Unity and now allows the user to have both the fbx and all the textures.

The **path** of these textures exported will be the same as the fbx, inside the application file, and will have their original name. This will export all the textures and fbx at once in the application file location as shown in the picture below and all the software's implementation of the production phase will now be concluded.

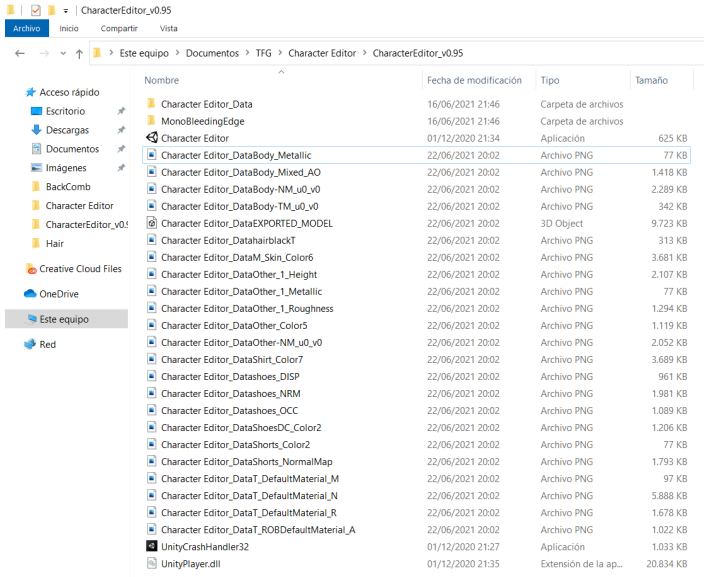

Fig 79. FBX and Textures exported in application path

Although the textures can be exported, a link to all the available textures from all the meshes is also provided in the readme file in case of any errors encountered or any changes are wanted to be made after exporting.

5.3 Postproduction Phase

After all the implementation of the software done, the final phase is the postproduction which involves the **testing** of multiple aspects of the software and final changes to **improve** the quality and objective in retrospect of these testing. These tests are focused on making sure the software reaches its goals proposed, that it is a helpful tool for students or indie companies for their videogame development, as well as the replication of any type of character, that it is easy and simple to use for any type of public and that it is useable in videogame development in terms of optimization.

5.3.1 Optimization Testing

The **optimization test** that has been performed is the same the ones before, explained in the validation tools in 3.2 and also done in 5.2.1 for the base mesh and in 5.2.5 in the beta build.
This test involves in placing the fully exported model with all its hair, clothing and textures into a Unity game scene and ran the game to check for any **performance issues** using the runtime statistics. This has to be compared between the statistics

shown before the models are placed and make sure that the addition of these do not inflict in any frame drops or memory loss.

Fig 80. Runtime statistics before importing the fully exported models

Fig 81. Runtime statistics with the fully exported models in the scene

As seen in these images, the difference in the stats is barely noticeable, which assures that the exported models are fully **optimized** and ready for game development without any need to do any changes.

The second part of the optimization testing, also explained and done previously the mentioned points, is the assurance of **correct animations** with no visual errors such as overlapping vertices, odd deformities or abnormal stretches in the mesh. This is tested

by creating a simple autorig using Mixamo, and checked if the sample animations that the website provides does not show any of these errors mentioned.

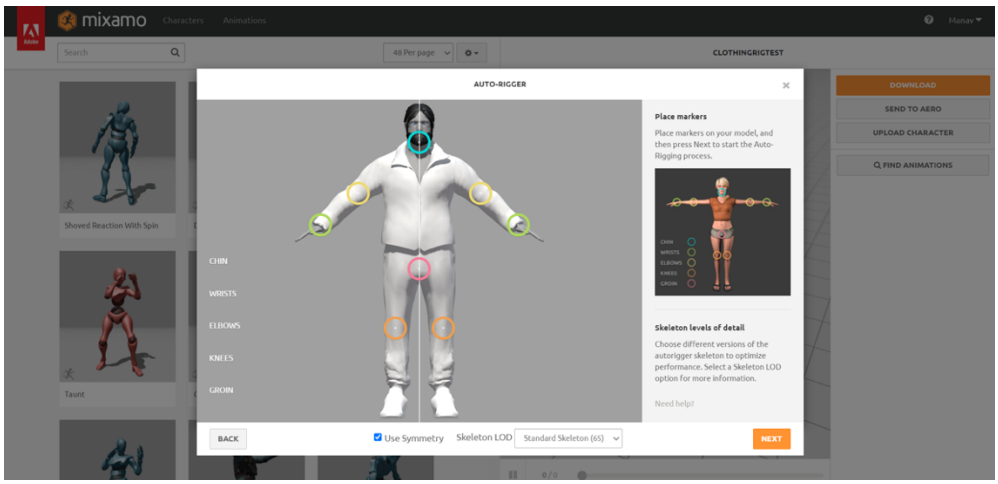

Fig 82. Mixamo autorigging finished model

Fig 83. Sample Mixamo animations in finished models

As seen in these images where 3 different models were used with different clothing and hair styles, the animations do not show any clear visual artifacts, which makes the animations very possible in these exported models.
For this reason, it can be concluded that the models the user can export are **fully optimized** for videogame development.

5.3.2 Replication Testing

The most important testing is to assure that the software **meets the objectives** of its goals, which is the usage of this character editor tool in videogame development for students from all knowledge levels and for videogame companies, whether they are used by artists or game designers, to recreate any human character they want for their games.

The way to make sure that these objectives are met, a testing has to be done by multiple people where they try the software and respond some certain questions, as explained in 3.2. This test is focused to be done by not only videogame students, but also people that do not have any involvement in it to certify the easy use of all the software aspects, as well as both male and female testers.
All of the testers had the same instructions, to download the software in their computer and use the software to attempt to **recreate** themselves, followed by answering a simple **form** of questions regarding it.
The first questions were based on a more **general** area: their involvement in videogames, if they found this software useful for videogame development students or companies and why, and if they would have used it as a student.
The following questions were regarding the **replication** of themselves: their gender, if they were able to replicate themselves and what was missing in case they could not.
And the last questions were about the **software** itself: any UI problems (visual, functional or unclear), if they would preferred to have a different order of customization, anything they liked about the software and anything they disliked about it.

After getting responses from this form, a small analysis and conclusion has to be made in order to do any future changes to the software.

Regarding the **first general questions**, the aim to use testers of all areas was accomplished as seen in this graph.

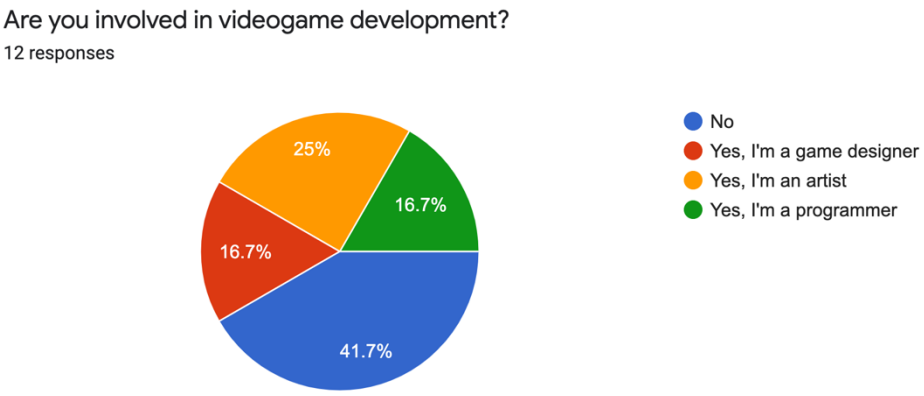

Fig 84. Testers videogame involvement graph

The two important questions from the first ones are if they found it useful for game development, if they would use it as a student and why.

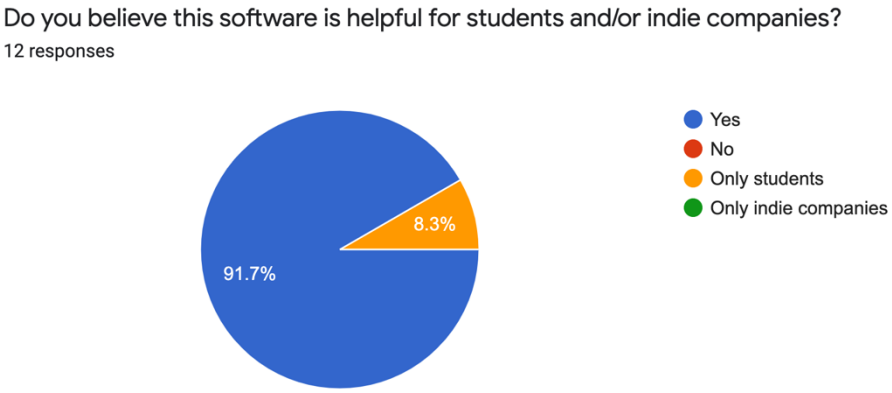

Fig 85. Testers response to if it is helpful for videogame development

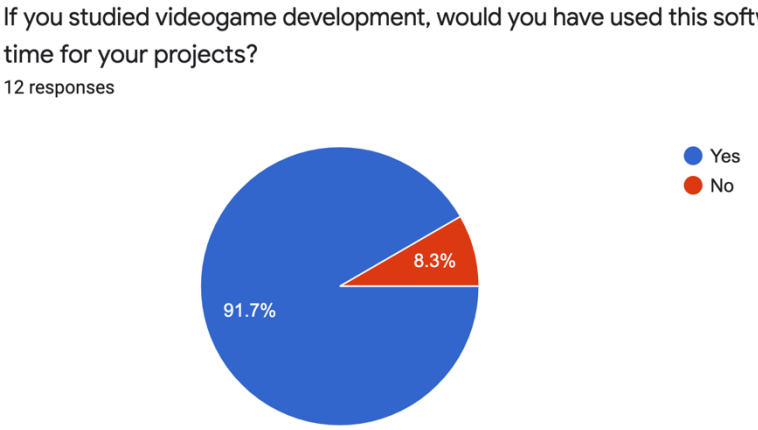

Fig 86. Testers response to if they would use it as a student

As seen in the graphs, these responses are very positive and over 90 percent believe that the software is useful for both students and indie companies, as well as that they would use it for their projects as a videogame student. The main reasoning of these testers are that they reduce time and has enough customizable options to create either a full model for the game or even use it as a prototype. This shows that the software is meeting very closely its objective of being a useful tool for videogame development between students and indie companies.

The **second part of the questions** involving the replication results turned out to be a bit different.
As this is a male and female software, a variety of gender between testers is needed, although this was not a 50/50 as desired, some female testers did try the software and attempted to replicate themselves.

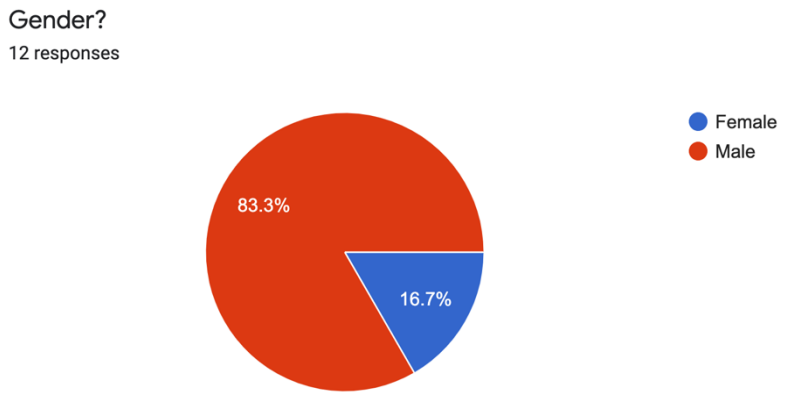

Fig 87. Testers gender graph

The successful replication of themselves is an important part of this testing, which is why the tester will answer if they succeeded completely, if they succeeded partially, if they did not succeed and why.

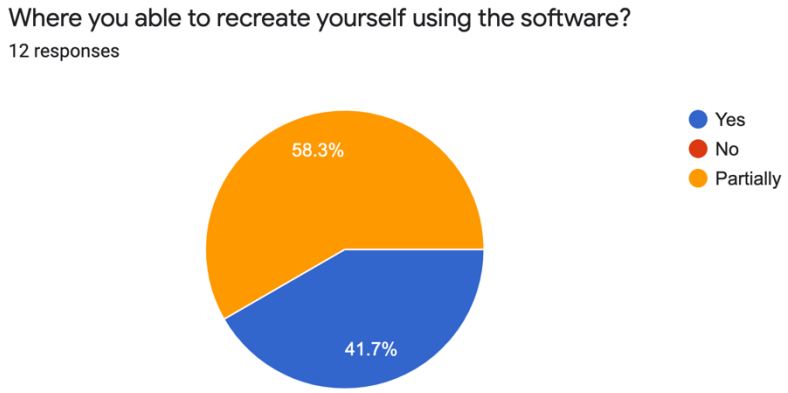

Fig 88. Testers response to the success of replicating themselves

As seen in the graph, half of the testers were able to fully replicate themselves, the other half did succeed only partially and **none** of them failed to produce a clone of themselves. This result, although it might seem low, is actually an **expected** result as this project was needed to be limited in terms of customization options, explained in the project range in 1.5. Since the start of the project, it was established that some options such as accessories, unique features like freckles, scars or tattoos, eyebrows and eyelashes were not going to be customizable options due to the time frame given. And so, the **limits** were set on only facial features, body types, hair, beard, clothing and colors to be editable in this software. For that reason, it is understandable that half of the testers did not find these options to complete a full replication, but the very good sign is that none of the testers could not replicate themselves at all.
The response to what these testers found missing were generally these things mentions, which are the accessories, but some testers do also mention a limited hair style choices.
All in all, the replication within its limits set in the project did reach to the aim to accomplish the possibility of creating a variety of any human using this software, except for maybe needing more hair style options.

Continuing to the **last questions** regarding the functionality of the software itself, the first question asked was about the UI, whether if the testers found any difficulty understanding anything or found any functional or visual errors. Although the all the responses were extremely positive, mentioning the very easy use of all the UI, the clarity of the buttons and the correct functionality, one tester did provide a helpful feedback about the nose menu that does not include a sidebar to scroll down to access all the parameters.
When asked if any tester wanted a different order of customization from the offered ones, none of them found the order misjudged and affirmed that all the order of customization options provided would be the ones they would have like to do.
This means that both the UI and order of customization buttons and menus achieved their goals, especially as there are non-videogame students who took this form, making it clear that the UI has a very easy usage.

Lastly, the testers were asked to express any of the software features they liked and/or disliked, for some further improvements.
The general things testers **liked** about the software were the clarity and easy usage of the UI, the high amount of customization options, the facial features sliders making it possible to create an infinite amount of faces and head shapes, the randomize button and the option to be able to export for game development.
On the other hand, the main **dislikes** from the software were the rotation system being hard to control, having more options involving accessories or unique features and a limited amount of hairstyles

This feedback provided really shows that the software **does meet the objectives** that were set initially in the start of the project, and a few improvement can still be made to make the software ready for the final build. In conclusion, reflecting over the results

of the form, the software is a success inside the limits set at the start and can be seen as a useful tool for videogame development.

5.3.3 Final Touches

As seen in the replication testing and the **feedback** gotten in the responses of the form, there are still some improvements to be done in order to make this software ready for use. These improvements correspond to the responses gotten about limited hair styles, the nose menu not having a sidebar, the rotation of the model being hard to control and some other final touches that were initially meant to be done. The accessories and unique features feedback gotten from the form will not be added as it was already established in the start of the project that those aspects would be off limits due to the time frame given.

The first thing that is needed to be done is to **add more options** in hair styles, which is why two more hair styles were added, for each male and female, to the software in order to make give more choice to the user.

Inside the nose menu, there are too many sliders to customize the nose that they do not fit all in the screen at once, which is why a **scrollbar** method was added, for the user to be able to access the rest of parameters by scrolling their mouse wheel or by pressing and dragging down the menu. For laptop users that do not have a mouse, this can be hard to understand and may not be able to access the other sliders, which is why a sidebar is added to the nose menu in order for laptop users to have a simple way of dragging down through the options, and also makes the whole nose menu more clear.

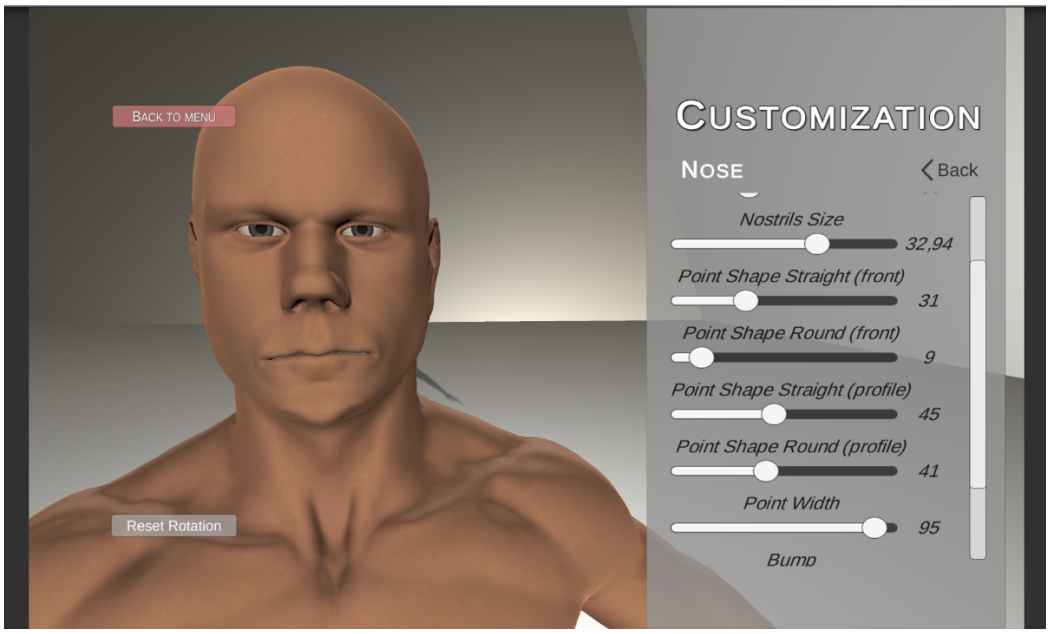

Fig 89. Scroll bar added in nose menu

The **rotation** script of the model is also needed to be modified in order to make it a more smooth, easy to control and simple rotation for the users to have more control of them. Although the reset rotation button is already available if the rotation goes off the user's hands, a better rotation system helps the user to visualize all the aspects of the model.

Lastly, some final improvements that do not involve the responses of the form has to be done.
The **UI art** style does not match with the main menu, which is why the main menu buttons were modified to the same color palette as the other scenes, giving a greyscale sensation to the user in terms of UI throughout all menus.
The **wireframe** button has to be deleted as the functionality of it does not work, and making it possible would inflict in change the objects material, which would mess up with exporting the textures of the objects. For that reason, the wireframe option will not end up being a part of this software.
A small **text information** after the model is exported was added. This text informs the user of the location of the fbx and the textures, and also provides a recommendation to the user to use Mixamo for the auto rig as it has been tested and proven it works correctly.

6. Conclusions

When the project was **initially** pitched and started, I personally did not know what tools could be used and how all the implementation worked in order to accomplish a software like this. During the research of the project I learned a huge amount of new information that I did not know it even existed, finding multiple options to create the customization, and even implementing the customization into a game engine with a fully functional exportation system. Even though this research was based more in programming and creating multiple variations and deformations, I also learned a lot about the human body itself, something that I initially did not think that would happen.

Looking at the finalized software, I believe the **objectives** that were set at the start ended up being **accomplished**, taking into account the limits of the project range of customization options. A very simple, easy and functional UI that really allows any person to use the software without any understanding issues, with large customization options that allows the user to create any human, a body and facial customization that does provide infinite possibilities and a fully functional exportation system for the model to be used in videogame development.
Although there are always things that can be improved, looking at the form and by personal opinion too, the software does seem to be **useful for students** and maybe even companies that want to reduce time, that do not have the resources to provide an artist or even to use it as a prototype or testing for any aspect of videogame development (animation practice, rigging practice, for programming testing and more).

The good thing about exporting the model for **videogame companies** is that all the individual meshes from the model are separated in different groups (childs of the model). This means that, any artist in any company could essentially export the model they like, import it in any 3D modelling software and do any final touches to the model, even combine it with clothing, hair or textures they made or found online and end up with a desired character while still saving a lot of time in the pipeline production.

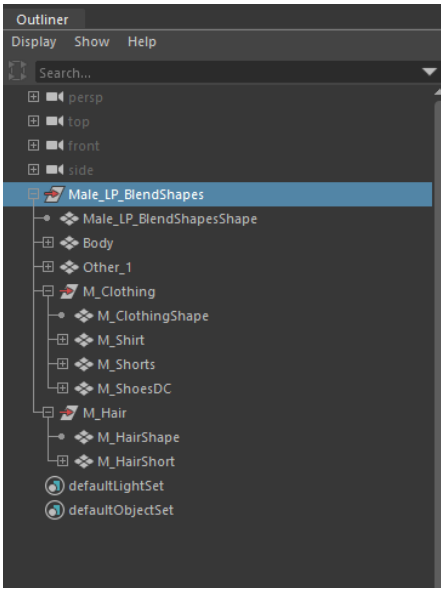

Fig 90. Hierarchy of exported model in Maya

During the development of this project I learned way more than I thought I would, and I can say that I am proud with the final result. Blendshapes were something I practiced once before but after this project I feel like I have full control of it now, and creating an entire functional software in Unity really helped me learn even more about the game engine that I never thought was even a possibility.

The **software** itself, inside the limits that were set, has a very elaborate first part of customization involving the blendshapes controlled by sliders of the body types and facial features. This first part was very liked and provided the user to make an infinite type of variations, and even recreate any type of human character. But the second part of the customization involving the hair and clothing ended up being more limited due to having to use preset models and choose between them, instead of editing them how the user would like.
Which leaves me thinking of what I would do in the **future** to improve this software.

Firstly, there are some thing I would **improve** about the elements that are **already in the software**, that were not able to be made due to the time frame. I would make the hair meshes myself finding the balance between a visually good realistic hair style and a very low amount of polygons where I can easily select the vertices of all the hair cards at once, this way I could create some length blendshapes to allow the user to control the length with a slider from that particular hair style. I would definitely also add more meshes and more color options involving the hair, beard and all the clothing (not only more garments but also more color variations like using prints on a t-shirt).

For **future work** in the project that does not involve what already is implemented, I would definitely add more customization parts of important things that were missing due to the limits set initially. These are the customization of eyebrows, eyelashes, body hair, all different unique features (like freckles, scars, tattoos, makeup, etc.) and also multiple accessory options (hats, earrings, glasses, rings, watch, etc.). If these

customization options were added, the software would become an extremely good tool for videogame development.
Apart from customization values, there some improvements regarding the software itself that can be made in the future. These involve the ability to show the polycount, change to wireframe, export the model already rigged, have the model with a slight idle animation in real time during the customization, exporting in the location the user provides, make a photography studio as the background scene with better lighting and improve the UI art to make it more unique.

All in all, this project was very interesting to make and I personally think that the software ended up being a **success** making future students to use it. Although some improvement could have been made, I am fully proud of the outcome of the project, not only with the software itself, but also with all the knowledge I am taking with me after that development process.

7. Bibliography

- Somatotypes Article. *https://www.britannica.com/science/somatotype*. [Consulted March 5, 2021]

- Cryobank Facial Features PDF Sample. *https://www.cryobank.com/_resources/pdf/sampleinformation/facialfeaturesample.pdf.* [Consulted March 6, 2021]

- Unreal Engine 5 Preview. *https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5.* [Consulted March 7, 2021]

- Jansen Turk's Game Hair Implementation. *https://www.artstation.com/artwork/XEyeY.* [Consulted March 9, 2021]

- Blend Shapes in Unity. *https://docs.unity3d.com/Manual/BlendShapes.html.* [Consulted March 12, 2021]

- Unity Obj Script Exporter. *http://wiki.unity3d.com/index.php/ObjExporter.* [Consulted March 12, 2021]

- Reallusion's Character Creator 3. *https://www.reallusion.com/character-creator/.* [Consulted March 14, 2021]

- MakeHuman. *http://www.makehumancommunity.org/.* [Consulted March 14, 2021]

- MB-Lab. *https://mblab.dev/about/.* [Consulted March 14, 2021]

- Autodesk Character Generator. *https://charactergenerator.autodesk.com/.* [Consulted March 14, 2021]

- Substance Painter online material sharing: https://share.substance3d.com/ [Consulted March 23, 2021]

- Unity rendering statistics window: *https://docs.unity3d.com/Manual/RenderingStatistics.html* [Consulted March 28, 2021]

- Jellostain Female model: *https://www.turbosquid.com/3d-models/free-simple-female-basemesh-3d-model/714970* [Consulted March 19, 2021]

- Pter Jzsa Jr. Male model: *https://www.turbosquid.com/3d-models/rigged-mike-freeman-model-1191338* [Consulted March 16, 2021]

- Unity's game package "3D Game Kit": *https://assetstore.unity.com/packages/templates/tutorials/3d-game-kit-115747* [Consulted March 25, 2021]

- Maya's Soft selection tool: *https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/Maya-Modeling/files/GUID-FF7C8670-97C7-4C13-9A6F-3B0A8F881EC9-htm.html* [Consulted April 2, 2021]

- Maya's Sculpting tools: *https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/Maya-Modeling/files/GUID-820F611F-3380-4D33-AC69-EEC62A59109B-htm.html* [Consulted April 2, 2021]

- Combination target blendshapes: *https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-CharacterAnimation/files/GUID-2BB492B9-4056-4877-A667-8ED43621CFB2-htm.html* [Consulted April 10, 2021]

- Lerp function *https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html* [Consulted April 16, 2021]

- Kellan Higgins FBX Exporter: *https://github.com/KellanHiggins/UnityFBXExporter* [Consulted April 28, 2021]

- Unity's Application.dataPath function: *https://docs.unity3d.com/ScriptReference/Application-dataPath.html* [Consulted April 28, 2021]

- Unity's GL.wireframe rendering: *https://docs.unity3d.com/ScriptReference/GL-wireframe.html* [Consulted April 25, 2021]

- Male Jacket Model: *https://sketchfab.com/3d-models/jacket-final-73237e73f8c34eaab22194b4c594df34* [Consulted May 10, 2021]

- Male T-Shirt Model: *https://sketchfab.com/3d-models/t-shirt-00daf5abf6d44db9a310b8264ad0e834* [Consulted May 5, 2021]

- Female T-Shirt Model: *https://sketchfab.com/3d-models/t-shirt-t-pose-20fbb246bd19402c82e01d7541a72f8e* [Consulted May 5, 2021]

- Jeans Model: *https://sketchfab.com/3d-models/blue-jeans-pants-6fe45842b1924f6fbf9e8bb021c034c6* [Consulted May 6, 2021]

- Joggers Model: *https://sketchfab.com/3d-models/dillard-darren-joggers-1f6f7318f9364456b8d58c1eaf71762b* [Consulted May 7, 2021]

- Shorts Model: *https://done3d.com/workout-shorts/* [Consulted May 6, 2021]

- Shirt Model: *https://sketchfab.com/3d-models/pilot-shirt-clothing-prop-game-resolution-eba426d142724fb9985897fe57eca977* [Consulted May 12, 2021]

- Tank Top Model: *https://www.cgtrader.com/free-3d-models/character/clothing/t-shirt-1f56abaa-989d-4c29-9b3b-583c1140e88e* [Consulted May 5, 2021]

- Female Jacket Model: *https://www.cgtrader.com/free-3d-models/character/clothing/female-jacket* [Consulted May 9, 2021]

- Dress Model: *https://www.cgtrader.com/free-3d-models/character/clothing/shirt-dress-with-multiple-textures* [Consulted May 8, 2021]

- Skater Shoes (DC): *https://www.cgtrader.com/free-3d-models/character/clothing/dc-shoe-c4d-16* [Consulted May 14, 2021]

- Sneaker Shoes (Converse): *https://www.cgtrader.com/free-3d-models/character/clothing/sneakers-8e81cc05-3c54-4e60-bc2e-e5ecb5ce9890* [Consulted May 14, 2021]

- Crocs Shoes: *https://www.cgtrader.com/free-3d-models/character/clothing/pink-e24a0ef1-9b62-46eb-9371-7e813a82d8b8* [Consulted May 14, 2021]

- Formal Shoes: *https://www.cgtrader.com/free-3d-models/character/clothing/2-pairs-of-shoes* [Consulted May 14, 2021]

- *Ken Jiang's* Eye Material: *https://share.substance3d.com/libraries/328* [Consulted May 24, 2021]

- Renderer set texture material Unity: *https://docs.unity3d.com/ScriptReference/Material.SetTexture.html* [Consulted June 7, 2021]

- Set Parent Unity: *https://docs.unity3d.com/ScriptReference/Transform-parent.html* [Consulted June 10, 2021]

8. Annexes

Replication Testing responses:

https://docs.google.com/spreadsheets/d/12LMYkI0snKH2JVZiqAqejm_LAziPhWJ2oUQfw1r_i-U/edit?usp=sharing