Towards the Research and Development of Anthropomorphic Avatar Creation and Customization by using Unity3D Engine

Isiah Slater

Table of Contents

Abstract

In this project, we want to propose the research and development of an avatar creation and customization application that allows users to make 3D virtual characters.

We would incorporate the knowledge from a number of courses that I undertook in my Art major to create the character models and UI elements. Further, I would like to research and develop a software program that leverages computer graphics that I have completed as a computer science major student. Potentially, I am considering incorporating the Unity 3D game engine for this software development work.

The main concept that sets the project apart from many other such character creator apps, like fuse CC, is its focus on a cartoon style that revolves around anthropomorphic animals, rather than people. My current goal is to allow users to dynamically adjust the proportions and species of the character, with the ability to create species hybrids (like having a cat-dog for example). After they finish creating their character, they would be able to export it as an FBX format, for use in other applications like Blender, Unity, and Unreal engine.

Additionally, time permitting, we would attempt to allow users to upload their own custom textures, change to several different clothing and accessory options, and add a platformer demo stage to allow the users to view their characters in a game-like environment. This capstone project would test my knowledge on software development, 3D modeling, UI, and computer graphics.

Premise

This project is aimed in an attempt to give creators a very easy and accessible way to create a wide cast of characters for animation, video games, 3D printing; the list goes on. With the rise in popularity of anthropomorphic characters in the 3D scene, I felt it necessary to make a software revolving around them, as there is a hoard of softwares built around making human avatars already.

https://play.unity.com/mg/other/0-1-dpw37

Prototype

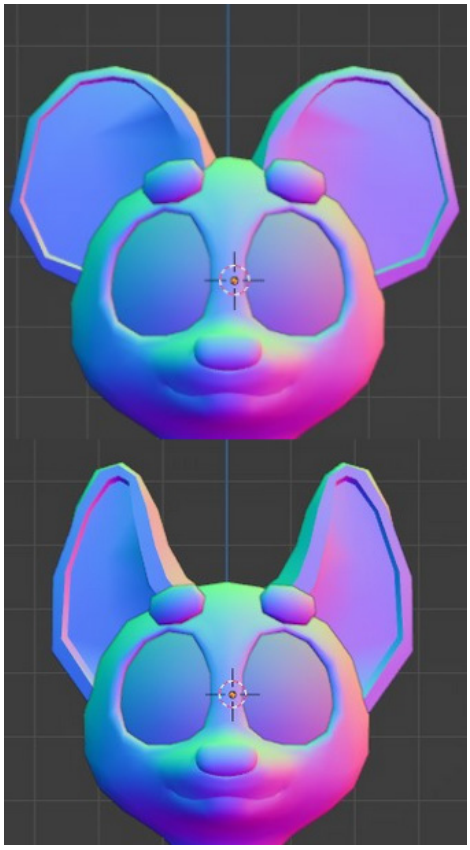
A working prototype can be found in the QR above (Desktop Only). This touches on the current concepts discussed later. Most specifically, it establishes the connection between the 3D model and Unity's Scripting. Using the model's skinMeshRenderer, I was able to manipulate mesh data based on blend shapes I made in Blender. By doing this, I was able to make the ears change shape based on the slider value of a unity UI element.

Softwares

There are several softwares that will be combined in intricate ways to make this project succeed. Below is a list of all of the softwares used:

- **Affinity Designer: Developing UI components**
- **Blender 3.Xv: 3D modeling and UV-unwrapping all assets**
- **Visual Studio 2019: Programming all scripts in Unity3D**
- **Unity 2020.3.25f1: Putting everything together into a cohesive experience**
- **Adobe Photoshop: Developing UI components**
- **Adobe InDesign: Creating this visual layout**
- **Substance Painter 3D: Texturing the 3D models**

Of these software, Blender and Unity will be the most frequented.



How I would use Blender

Blender would be used to create the base model, and hold information for changing different aspects of the character.

For example, the ears would be a part of the base model, and hold information for changing into bear ears, cat ears, or mouse ears (right). This process would occur over several parts of the model, too (ears, tail, jaw, nose, abdomen).

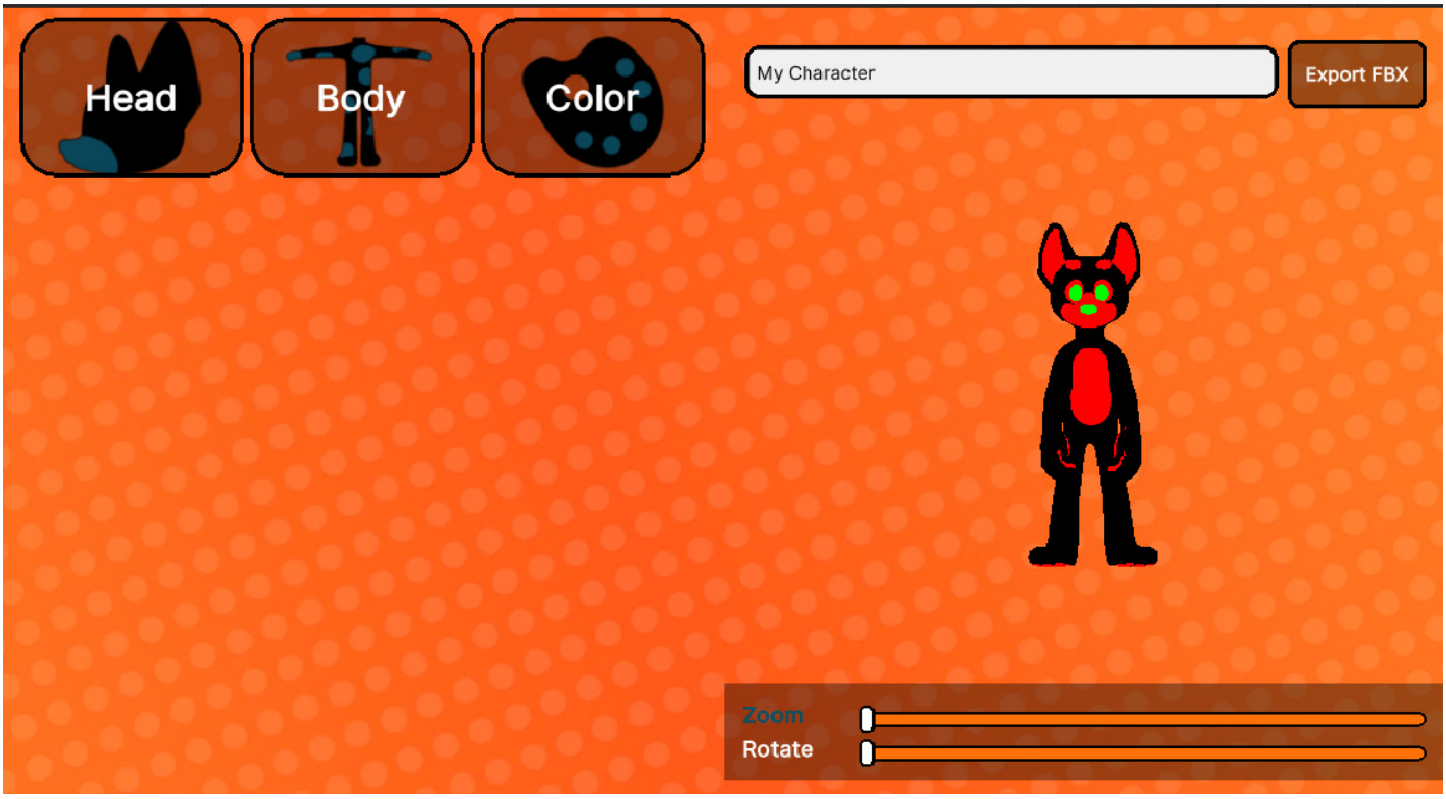
However, when these parts deform to change shape, the bones that control them do not, so using just Blender alone, I would be unable to adjust aspects like character height without running into issues with the character's rigging.

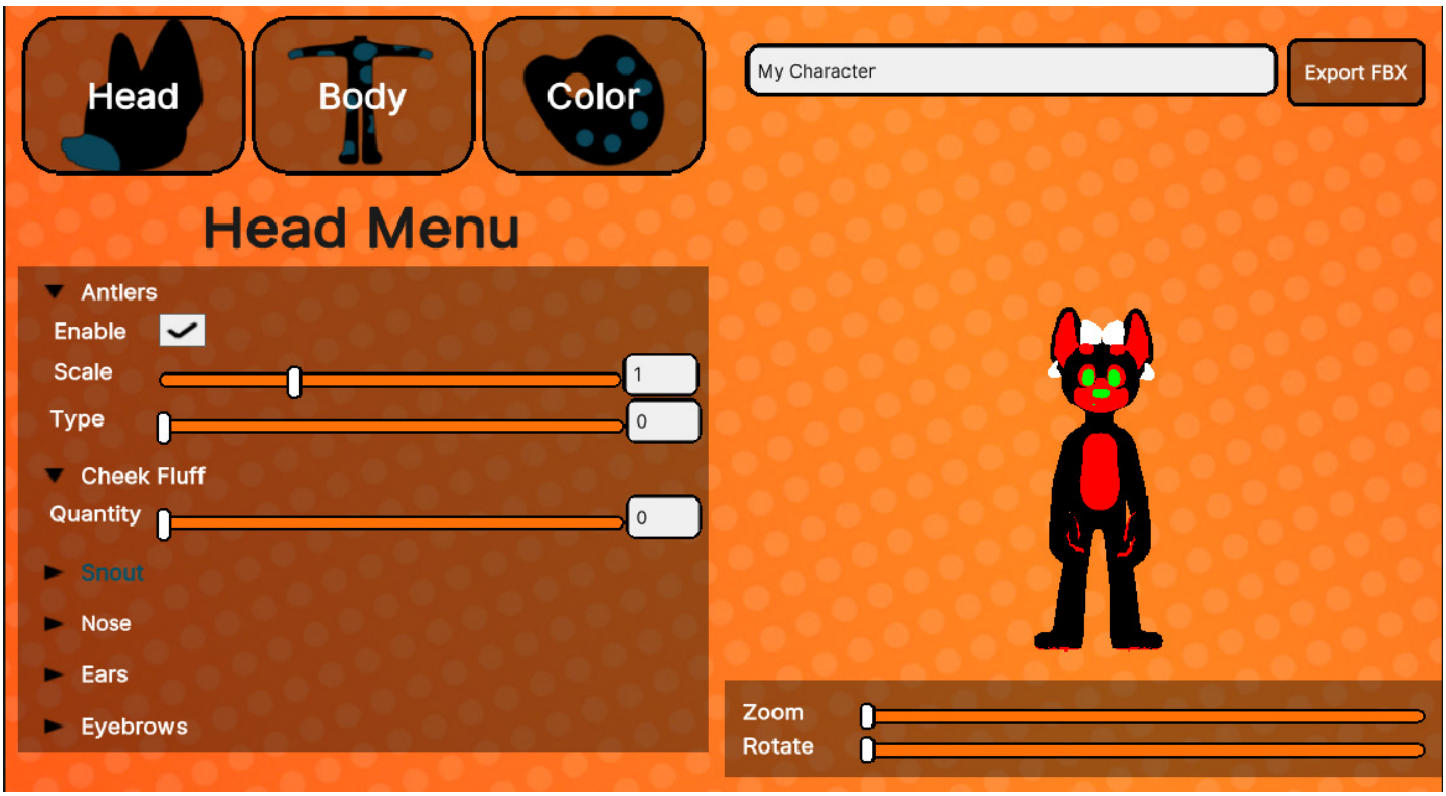
How I would use Unity

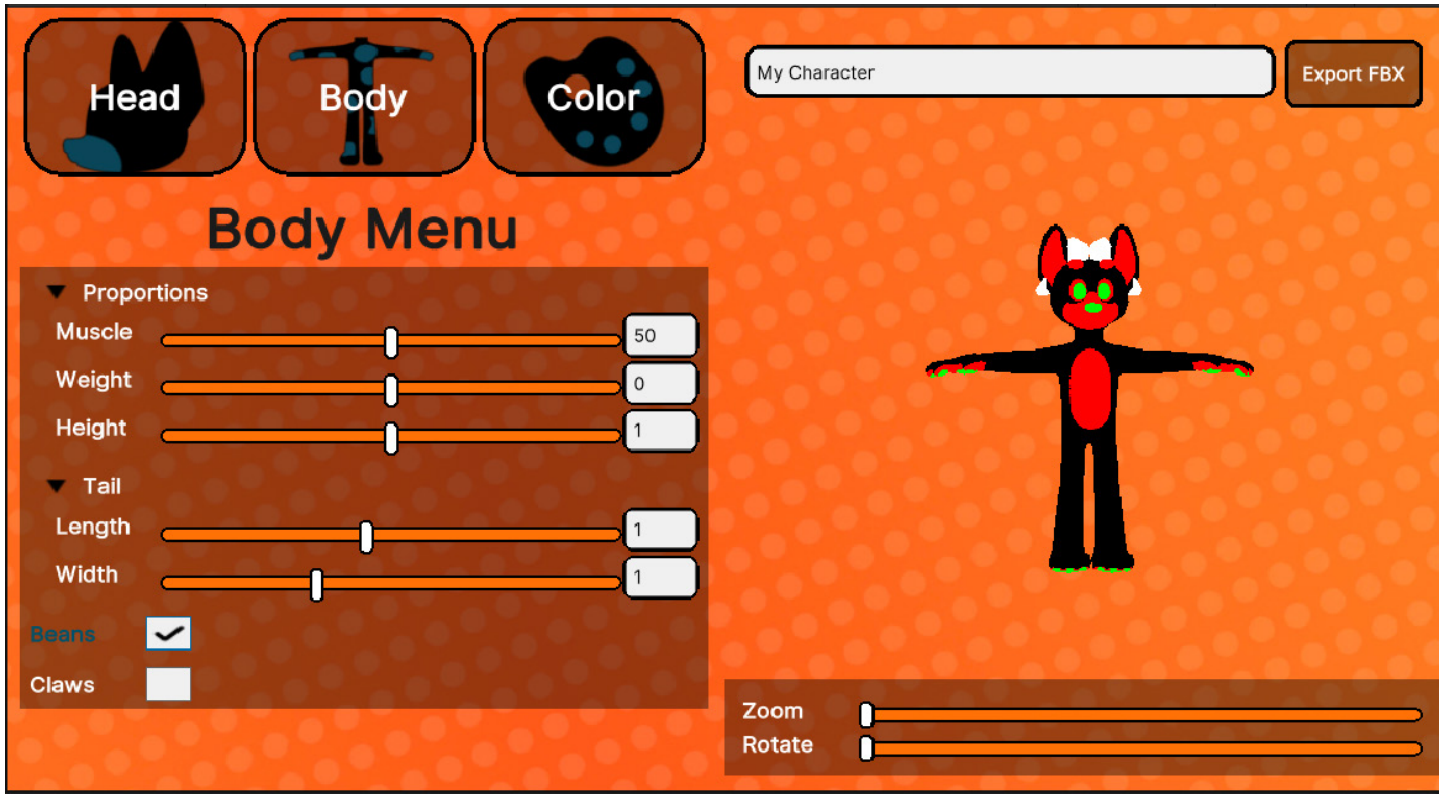
By utilizing Unity, I could allow the information within the model to change based on user input. With sliders, or a similar interface, I could allow the user to adjust aspects of the character, like ear shape, jaw width, etc.

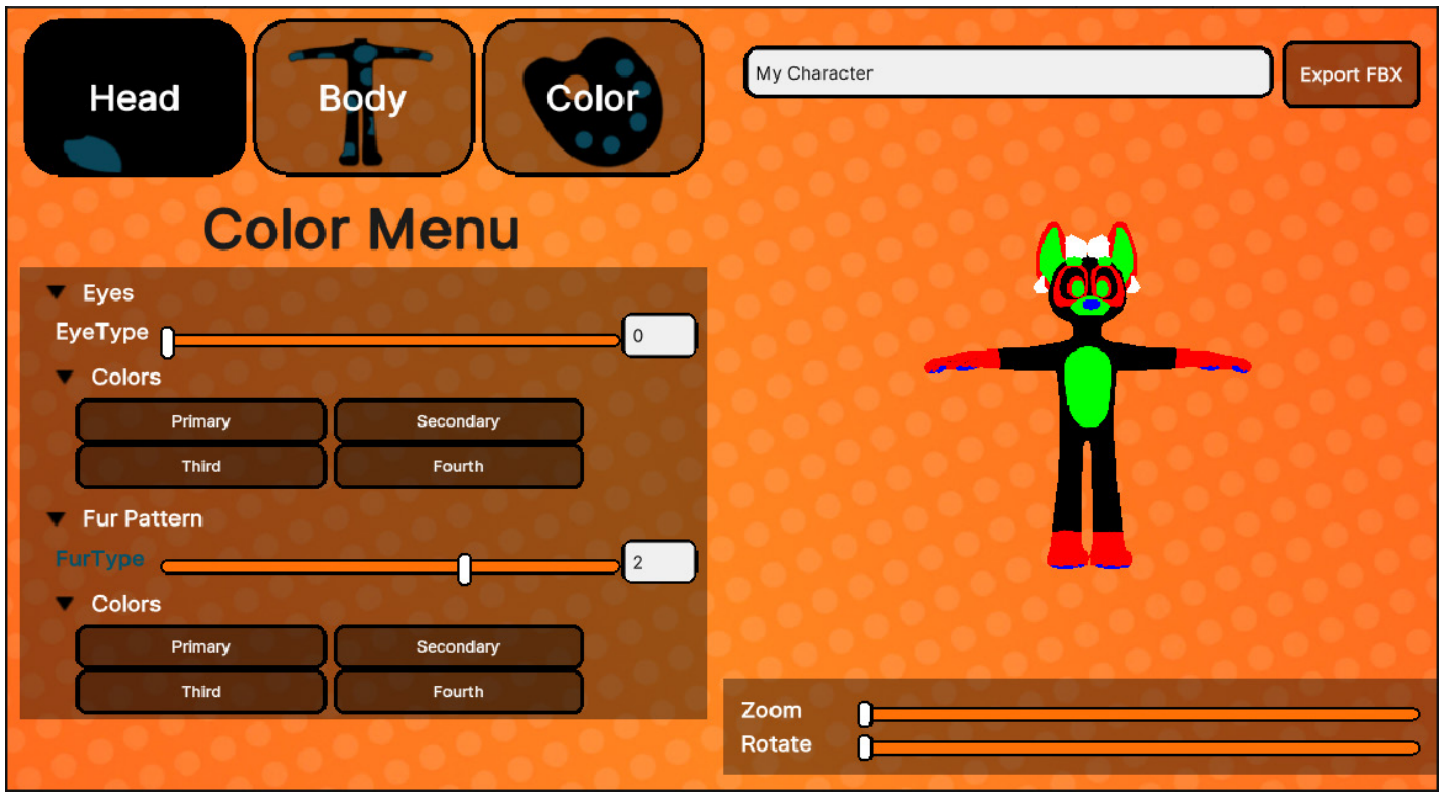
Blender, as previously mentioned, does not allow for easy scaling of limbs without causing rigging issues. Using Unity, I could also allow select bones to scale within the model, allowing for customization of the character's height, leg length, or tail length, all without causing rigging issues. Furthermore, if time allotted, I would like to see if I am able to program a color picker into the application. This would let users change the colors of a character, in addition to the previously mentioned features.

State of the Game



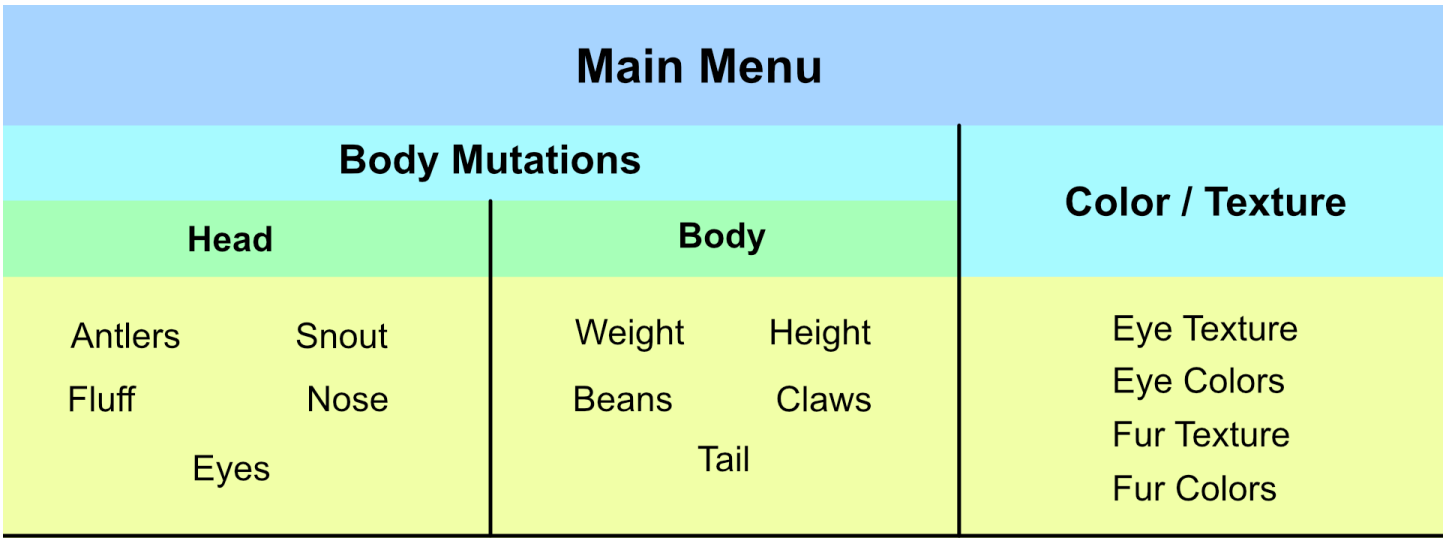



Head
Body
Color
My Character
Export FBX
Body Menu
Proportions
Muscle
50
Weight
0
Height
1
Tail
Length
1
Width
1
Beans
Claws
Zoom
Rotate


Head
Body
Color
My Character
Export FBX
Color Menu
Eyes
EyeType
0
Colors
Primary
Secondary
Third
Fourth
Fur Pattern
FurType
2
Colors
Primary
Secondary
Third
Fourth
Zoom
Rotate

UI

The Layout of UI will be very important for guiding users to change what they want from the model, without getting lost. I want the learning curve to be as small as possible. Thus, the interface will be structured by body part, and have a separate menu for color. Originally, the color menu was to be stuck on the screen at all times, to allow spontaneous changes during the character's development; however, the UI became more legible when coloring was placed into its own sub-menu. The interface relies on a combination of iconography and text. While using solely icons would be more intuitive for language barriers, there were certain locations that icons alone would struggle to explain. This methodology is elaborated in the research section.



Modeling

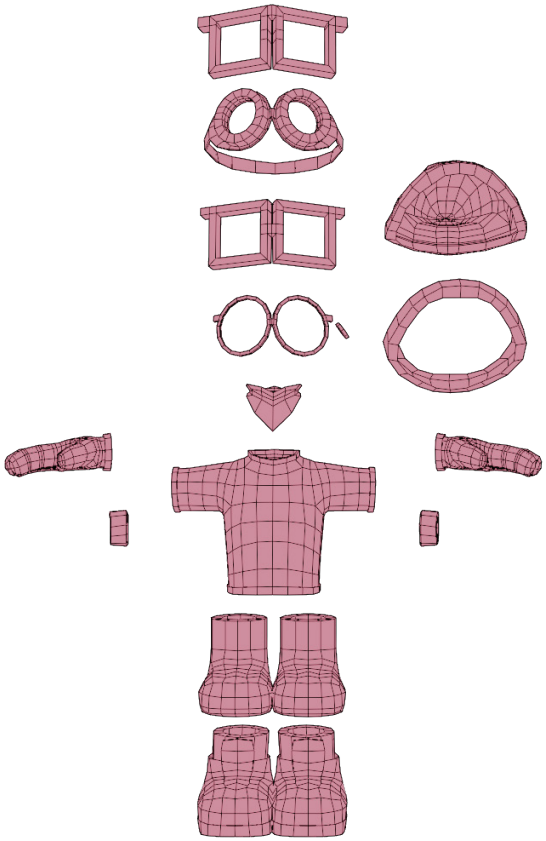
Character

An average completed character is composed of approximately 8000 tris. This number gives it compatibility for use in lower-powered devices, such as the oculus quest. Additionally, the character possesses all necessary visemes to function when placed into games like VR Chat and VSee Face.

has a lot of blendshapes, that allow for manipulation of specific aspects:

- **Ears can be rounded or pointed**
- **Weight can be added on or taken away**

- 7300 base tris
- 5 antler types
- Detachable Claws / Beans
- Visemes: PP / FF / TH / DD / kk / CH / SS / nn / RR / aa / e / I / O /U
- Symmetric Model



Clothing (Not Currently Implemented)

Clothing often deforms alongside the mesh as it bends and twists. Thus, the poly count on these are almost identital to the body part that they cover. Below are the clothing options that have been modeled:

- **Casual Tee**
- **Bandana**
- **Beanie**
- **Gloves**
- **Glasses**
- **Shoes**
- **Socks**
- **SweatBands**

Texturing

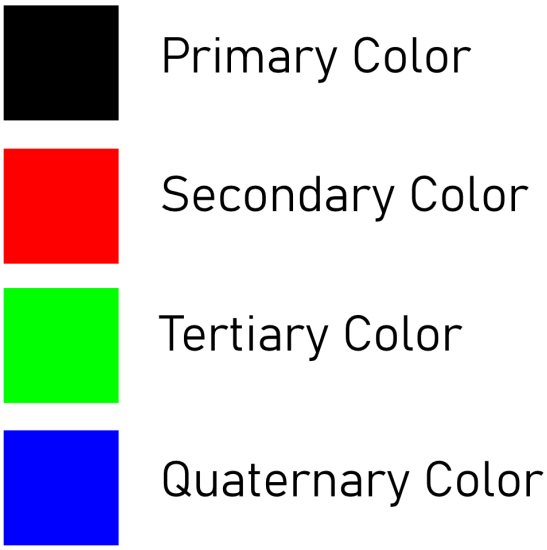
Colors

To texture the model from within game, I thought of a technique that utilizes four distinct colors to act as placeholders:

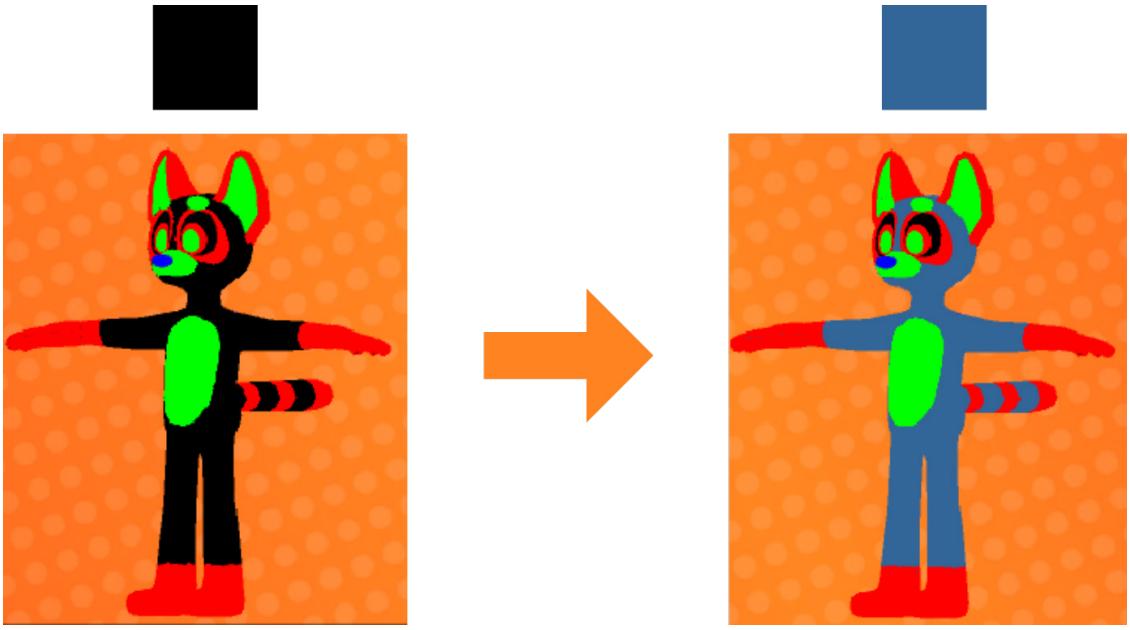
- **Black [RGBA: (0, 0, 0, 1)]**
- **Red [RGBA: (1, 0, 0, 1)]**
- **Green [RGBA: (0, 1, 0, 1)]**
- **Blue [RGBA: (0, 0, 1, 1)]**

When the user selects a color in-game, my script will search for the pixels that contain the value of the placeholder color (primary color, secondary, etc), and replace it with the user-provided color.

In total, there are currently three textures on the character: Fur Pattern, Eye Pattern, and antler color. Both fur and eye textures utilize this hot-swapping coloring method, and as such, the defuault textures for these are not natural looking (shown below).



Pictured: the four colors that will compose all character textures. This will allow for each texture file to have four unique colors.



For example, the user utilizes a raccoon texture. The primary color is black (left). If the user wants teal as the main color, they would select it in-game. The program will replace all black instances on the raccoon's original texture with the selected color, teal.

Fur Textures

The users have the ability to change the texture of the character to reflect the species of their choice. There are currently four implementations of fur textures:

- **Dog**
- **Calico Cat**
- **Raccoon**
- **Tanuki**

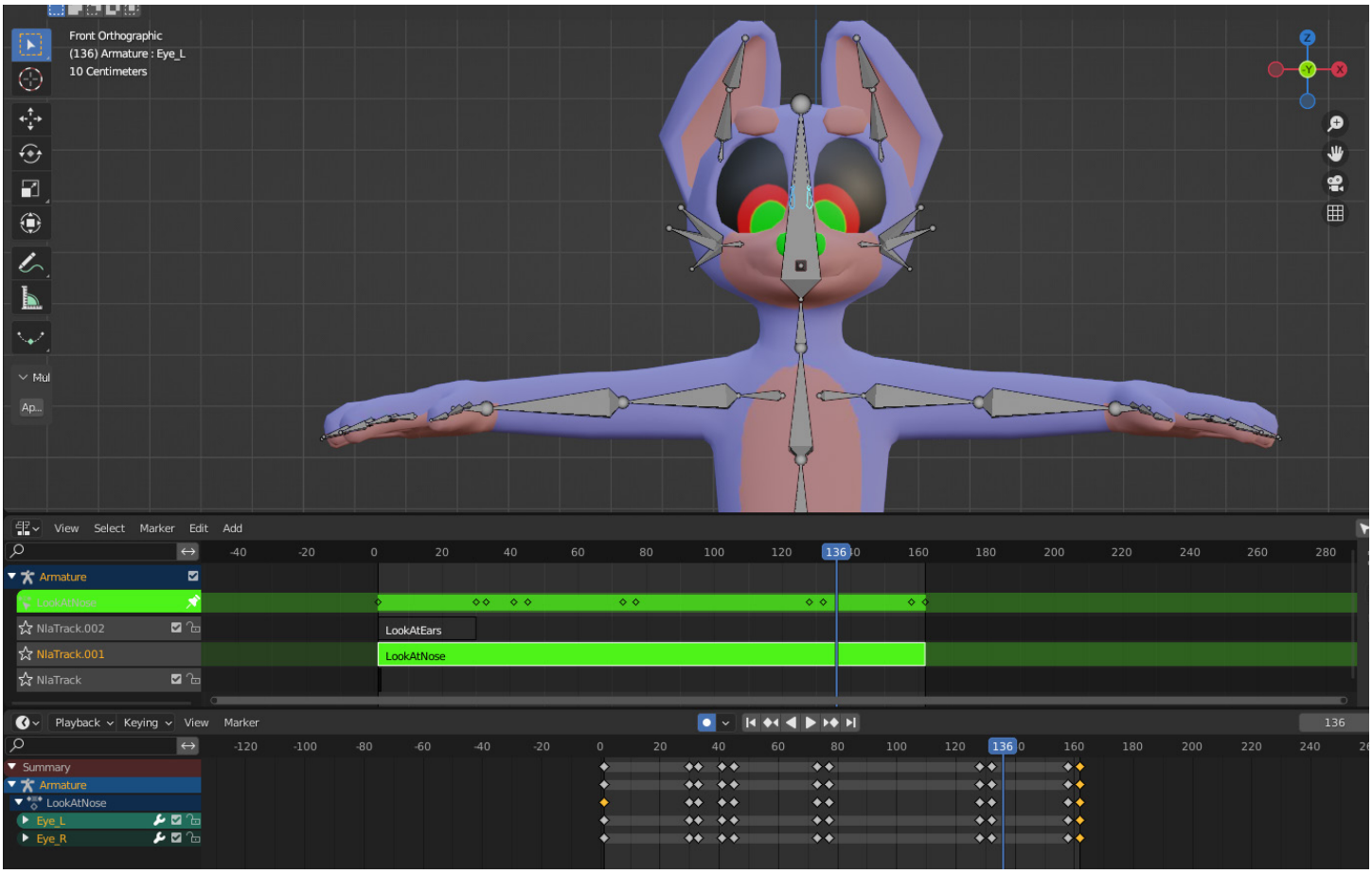
Eye Textures

With how diverse characters can be, the eyes are yet another avenue of character expression. Different eye shapes communicate different ideas or first impressions. If the character has big pupils, then a feeling of cuteness is given. Likewise, sharp cat eyes can create a feeling of intimidation or predation. There are currently only three eye shapes implemented:

- **Regular eyes**
- **Cat eyes**
- **Goat eyes**

Animation

This project combines hand-made animation with presets available on Adobe's Mixamo platform. There are five animations present in the current build of the project. When the application first starts, the character will wave to the user, both to welcome the player, and to ensure them that the project is running properly. When editing the character's body proportions, the character will pose in the default T-Pose, for the best field of vision for the player. Meanwhile, editing the values of the ears and snout will cause the character to look at these respective areas. An issue that was present initially was that the facial animations would override the body's animations while active. To prevent this, two layers were used: The Base (Body) animation layer, and the Facial animation layer. By setting the facial layer to be addative, the character would be able to look at the head piece being editing while maintaining the default idle animation everywhere else.

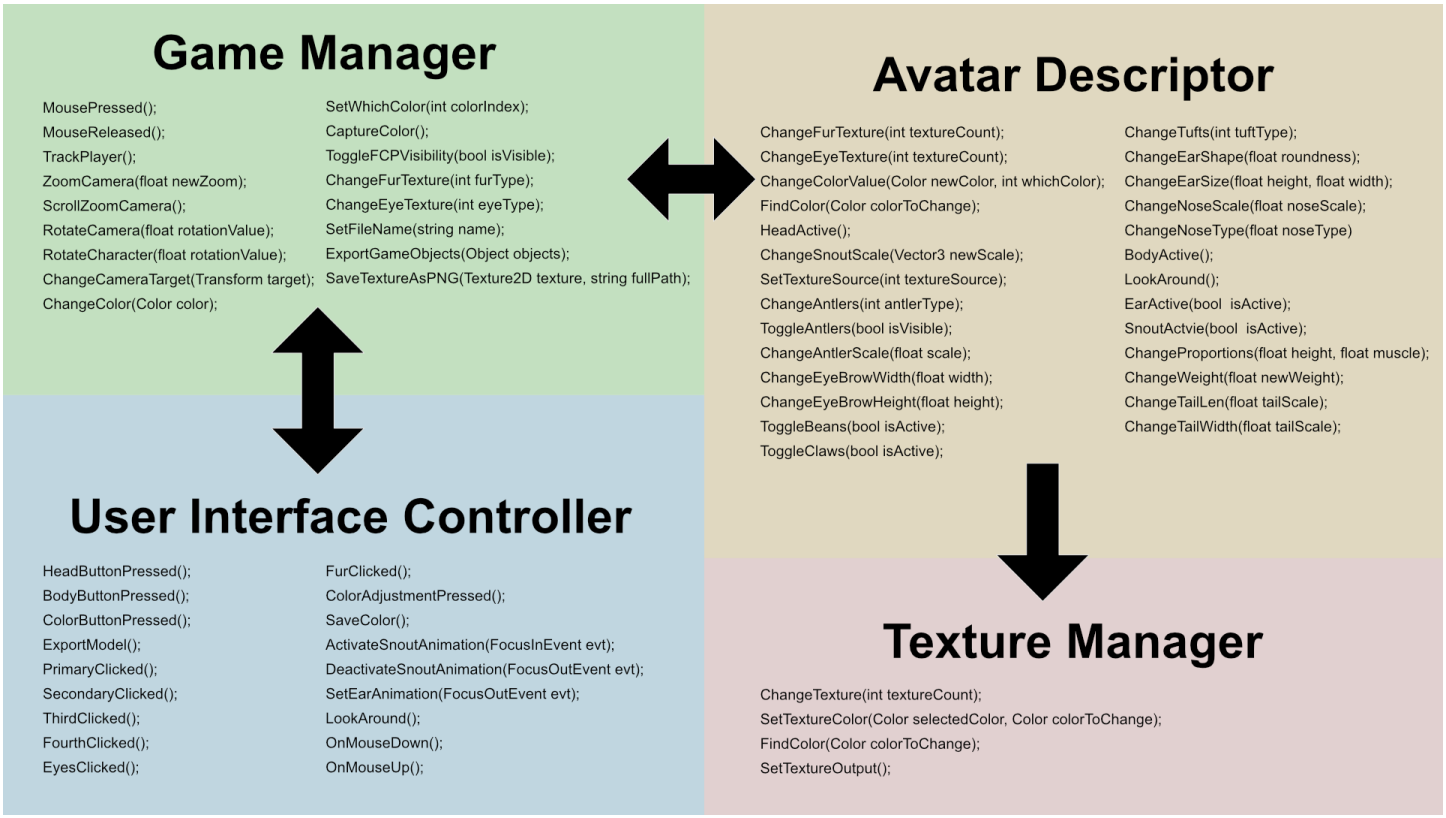


Pictured: The "Look At Nose" Animation, being made in Blender. Other animations, such as the "T-Pose" and "Look At Ears," were made purely in Blender, while the "Look Around" animation was sourced from Mixamo, and brought into Blender for further tweeking.

Coding

This project has four main scripts within it that describe the flow of interaction between the user and the software

- **Game Manager**
  Holds data on camera operations, as well as some communications between the UI and the avatar classes. Has one instance of the Avatar Descriptor and UI Controller
- **Avatar Descriptor**
  Holds all information about the character's blendshapes, bones, and how those are influenced by outside parameters. Has one instance of Texture Manager and Game Manager.
- **Texture Manager**
  Holds all information about reading textures, replacing colors on the texture, and swapping the textures on provided materials.
- **UI Controller**
  Holds all information about how the UI interacts with data, more specifically the avatar. Has one instance of Game manager to interact with the avatar class.

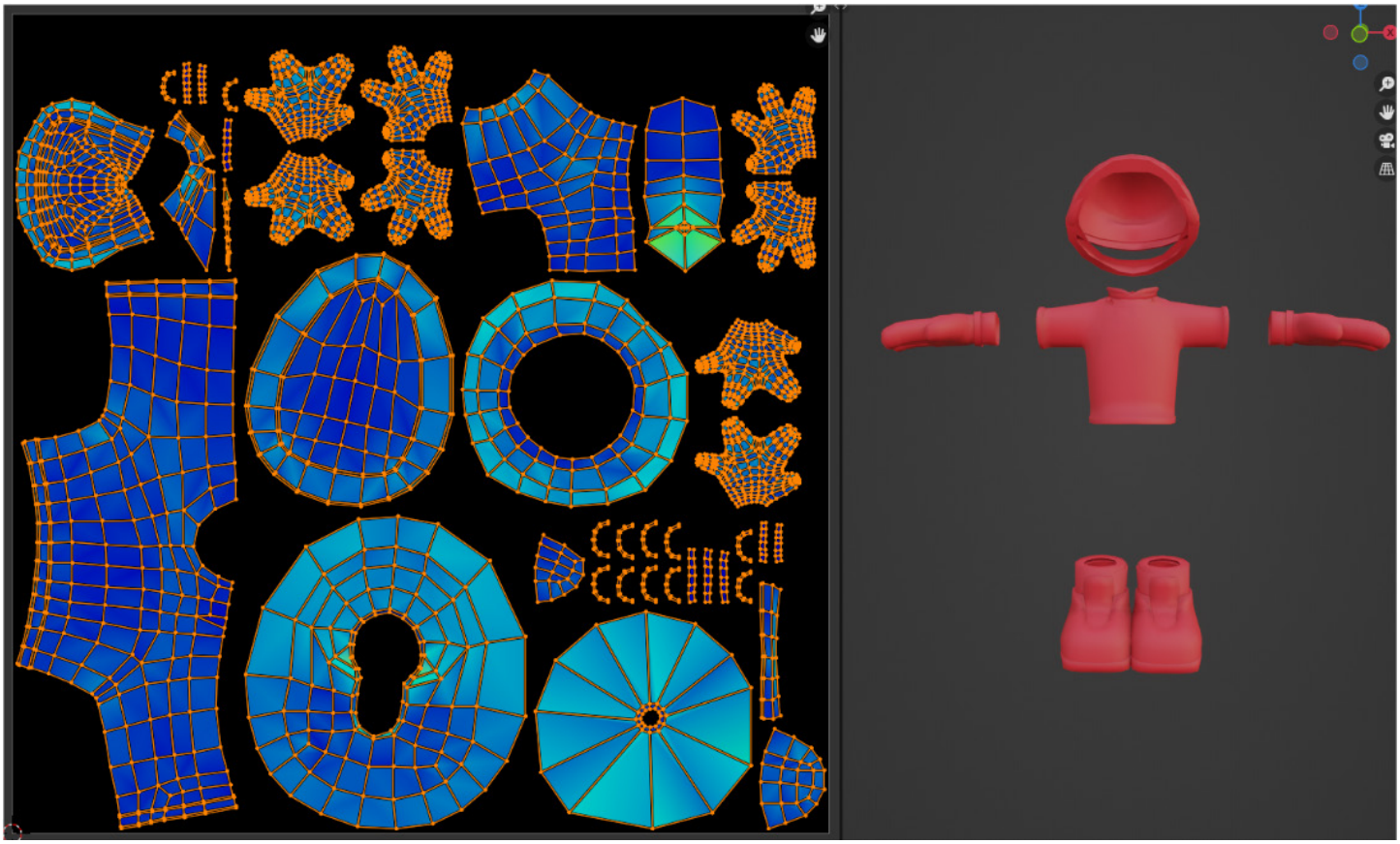


The functions of the program, and how they interact with one-another

Challenges

The most frequent challenges that I faced were a direct result of being new to the Unity and game development workspace. Below are a few examples, and more will be showcased within the Process to Completion montly logs:

- **Building a UI System in Unity. Originally I had created my own UI system from scratch, modeling it after a node tree system. In this scenario, I had buttons that would traverse the tree one depth down, and reparent the "back button" to the parent node. Pressing the "back button" would set the parent of it one node higher, and hide the menu that was just visible. The programming for this was very buggy, and luckily I found that Unity had a built-in UI Building toolkit. However, this was built in on later versions, so I had to import it by hand for the Unity version I used.**
- **Within this built-in UI toolkit, I had no idea initially on how to call functions from value changes. While I knew how to call functions on button presses the default slider event "OnValueChanged()" was missing for this toolkit's sliders, and was replaced with its very own function "RegisterValueChangedCallback()". After discovering this, many of the functions in the UI controller class came naturally.**
- **Clothing and accessories. Currently this functionality is not implemented, though I have taken steps of adding it into the program. An issue I struggled with was texture mapping all the clothes and accessories. Games like VRChat do not allow more than 4 materials being used at a time for the weaker device ports. The constraints were very tight considering that I was already using 3 materials (body, eyes, antlers). Mapping the fourth material to include all of the accessories and clothing options sounds like a logistical nightmare. The issue with placing all clothes onto a single texture would be that adding clothes in the future would require me to completely redo all clothing textures. In theory, the best option would be to assign the colors within the application, and once the character was exported, all of the materials for the clothing and accessories would get baked into a single texture. This would allow the texture to dynamically grow and shrink based on how many articles of clothing were selected by the user, and would require no overhaul every time a new accessory was added to the application.**

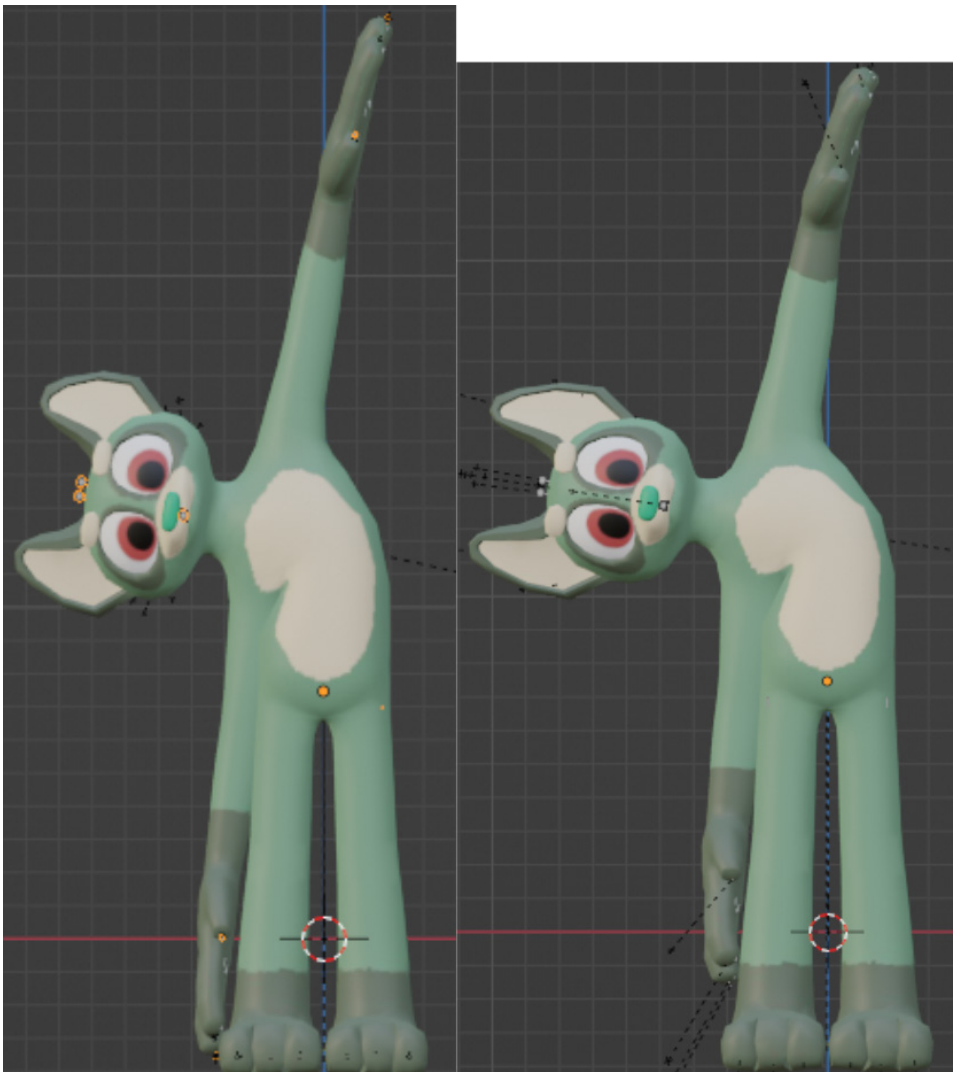


The UV texture space (left) of the clothes shown on the right. By creating new clothing, the UVs on the left would need to be redistributed to make room for the new clothing's texture space. This would

Additionally, there were challenges that I have not managed to solve yet. Researching answers for these brought no true resolve, and will remain so without making some form of breakthrough that the internet has yet to see, or that has been hidden behind a paywall (such as in the Unity asset store, for example):

- **FBX Exporter. Unity has a built-in FBX exporter, which I had intended to use for exporting the model after the user was finished with their creation. I programmed it to export to an export folder within the application, and to also include the textures that the users had modified via the program. Tests of this were very successful, but I was unable to build the game as a standalone application. This is because the built-in exporter only allowed use inside of Unity. This is what is known as an Editor-only application, as it cannot be used outside of the Unity game editor.**

  Luckily, there was a way to make an FBX exporter outside of Unity's built in program. This would be achieved through the Autodesk.fbx namespace. However, I have not yet seen a feasible way of exporting character models through this. In Unity, there are two types of meshes: regular meshes and skinned meshes. Skinned Meshes include anything which is manipulated by multiple bones on an armature, and contain blend shapes (like blinking or mouth shapes).
- **Non-Uniform Scaling. An issue that was unbeknownst to me until November was the game engine's inability to handle non-uniform scaling. This term refers to scaling the X, Y, and Z axis of an object non-proportionally. So if I made the torso longer by scaling the Y axis, then the torso would be non-uniformly scaled. While this achieves the desired look initially, any variance in the character from the T-pose results in a distorted model.**

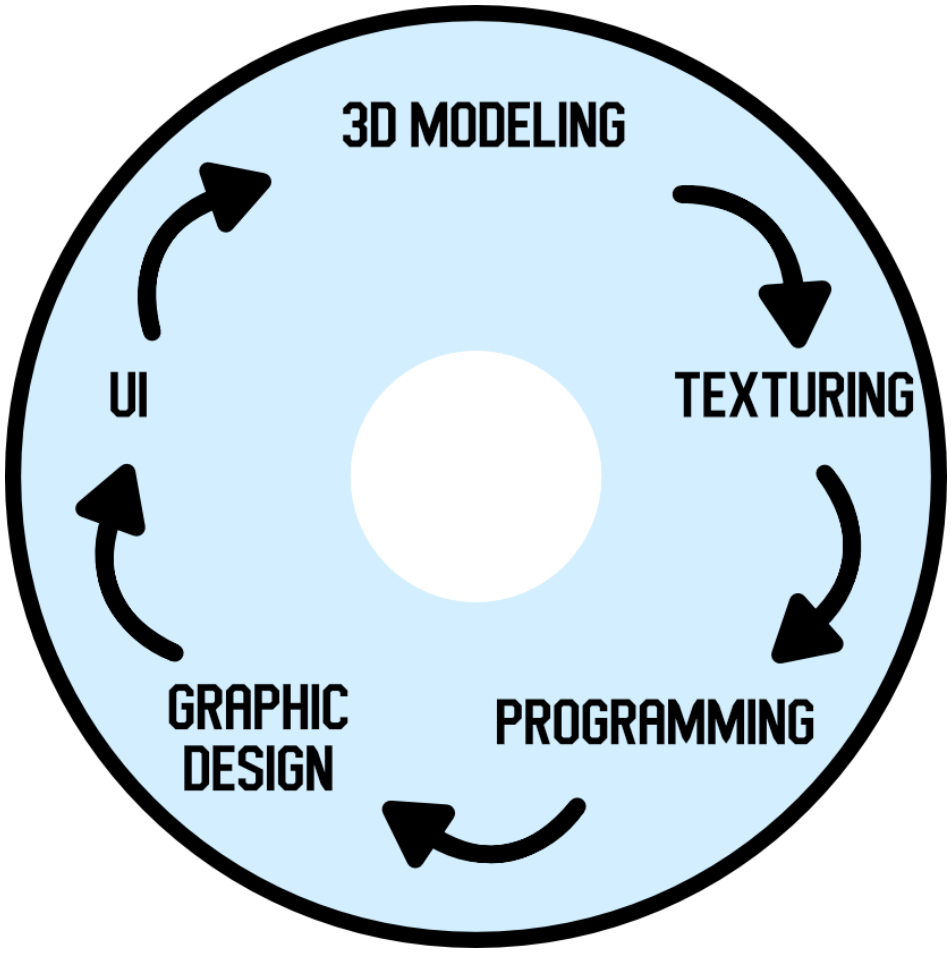


Local Scale distorts the rotation of the model (right). Many alterations outside of my application must be done for these models to be usable and rotate / scale properly (left).

Process to Completion

In order to complete this project, I need to take many, many facets into consideration. During this process, I will be a one-man team, wearing the following hats:

- **Graphic Designer: Making assets for the UI, choosing color and applying it accordingly**
- **Programmer: Combining user interaction with data to modify the character**
- **Game Developer: Putting everything together in the Unity Game Engine**
- **3D artist: Creating and rigging a 3D model to manipulate**
- **Animator: create animations for the characters for better user experience**
- **Texture artist: Painting a multitude of textures for the player model**
- **UI Designer / Developer: laying out the structure of menus for a simple learning curve and flawless user experience.**



The process of application development as a whole functions in a cycle. Everything gets covered on a general basis on the first pass through. This is to ensure that everything will work together? In this phase, nothing looks pretty. Instead, the goal is functionality over form. With much of the project left to do, getting into detail is a waste of time. However, planning for these details comes in handy when returning to a phase in the cycle. In hindsight, both programming and modeling were consistently revisited every step in the process, rather than being a stage in themselves.

June - July

Learning Unity

In the past I had used Unity to port character models into applications like VR Chat, LIV, and VSee Face. Suffice to say, I was only familiar with the interface of Unity.

Indeed, I had never used Unity to program or make games. Over the course of Summer, I took courses via Unity Learn. I was able to get through all of the Unity Essentials and Junior Programmer pathways. This gave me a greater depth of knowledge on how to program with game development in mind.

To continue learning into the semester, I have devised an action plan, seen below. This puts an emphasis on continuous learning, as there will always be issues and implementations that I cannot foresee. This is made clear in my timeline where I discuss all of the challenges I faced during my project

My Learning Action Plan

Milestones to reach my goal	Actions I will take to reach my goal	Timeline for each action	Outcome so I know I'm on the right path to reach my goal
1. Learn how to use Unity	Complete the Unity Essentials Pathway	1 weeks - I will spend 15 hours a week learning	I can open Unity, navigate and create projects
2. Learn how to program in C#	Complete the Create with Code course	2 weeks - I will spend 4 hours a week learning	I can create simple things in Unity using C#
3. Learn how to use Unity API	Something Something Tutorial	2 weeks - I will spend 4 hours a week learning	I will import and utilize my 3D model with scripts
3.5 Blendshape Manipulation	https://learn.unity.com/tutorial/setting-up-blendshapes-in-unity-2019-3	15 weeks, 2 hours per week	Manipulate blendshapes based on user input
5. Implement UI	https://learn.unity.com/tutorial/creating-ui-buttons https://learn.unity.com/tutorial/ui-components https://learn.unity.com/project/creative-core-ui	1 week start, 1 week return	Implement basic ui Return and Enhance UI with custom Sprites
4. Learn how to export	https://learn.unity.com/tutorial/baking-animation-for-fbx-export https://learn.unity.com/tutorial/working-with-the-fbx-exporter-setup-and-roundtrip-2019	1 week - 1 hour per day?	Be able to re-export the model as fbx
6. Integrate pre-made animations	https://learn.unity.com/project/re-targeting-and-re-using-animation https://www.mixamo.com/#/	1 week	Add reactive animations for user-interactivity
7. Animation Controllers	https://learn.unity.com/tutorial/may-18-animation	2 weeks, 5 hours per week	Link animations to actions to further drive user-interactivity
8. Export Project as Stand-alone	https://learn.unity.com/tutorial/publishing-for-pc-mac-2019-3	1 week, 2 hours per week	Export Project as a standalone software

Research

In order to capture an aesthetically pleasing and functional User Interface, I first did some research on other character creator apps. For this project, I wanted a design that would reflect the cute and cartoony nature of the characters. Something that I realized is that apps frequently use symbols instead of words to direct users. From retrospect, this is a fantastic work around language barrier, and would serve as a great accessibility path.

When thinking about the vision of this project, the following games come to mind:

The games, from left to right, top to bottom: Sims 4, Dark Souls III, Grand Theft Auto V, Elder Scrolls V: Skyrim

While each of the games had their own limitations and approaches on the character customization process, there were many foundational similarities between them. Most, for sake of clarity and compactness, sandwiched many options into general categories. This lets the developers minimize the number of decisions the player would have to make at any given time, reducing the amount of decision fatigue.

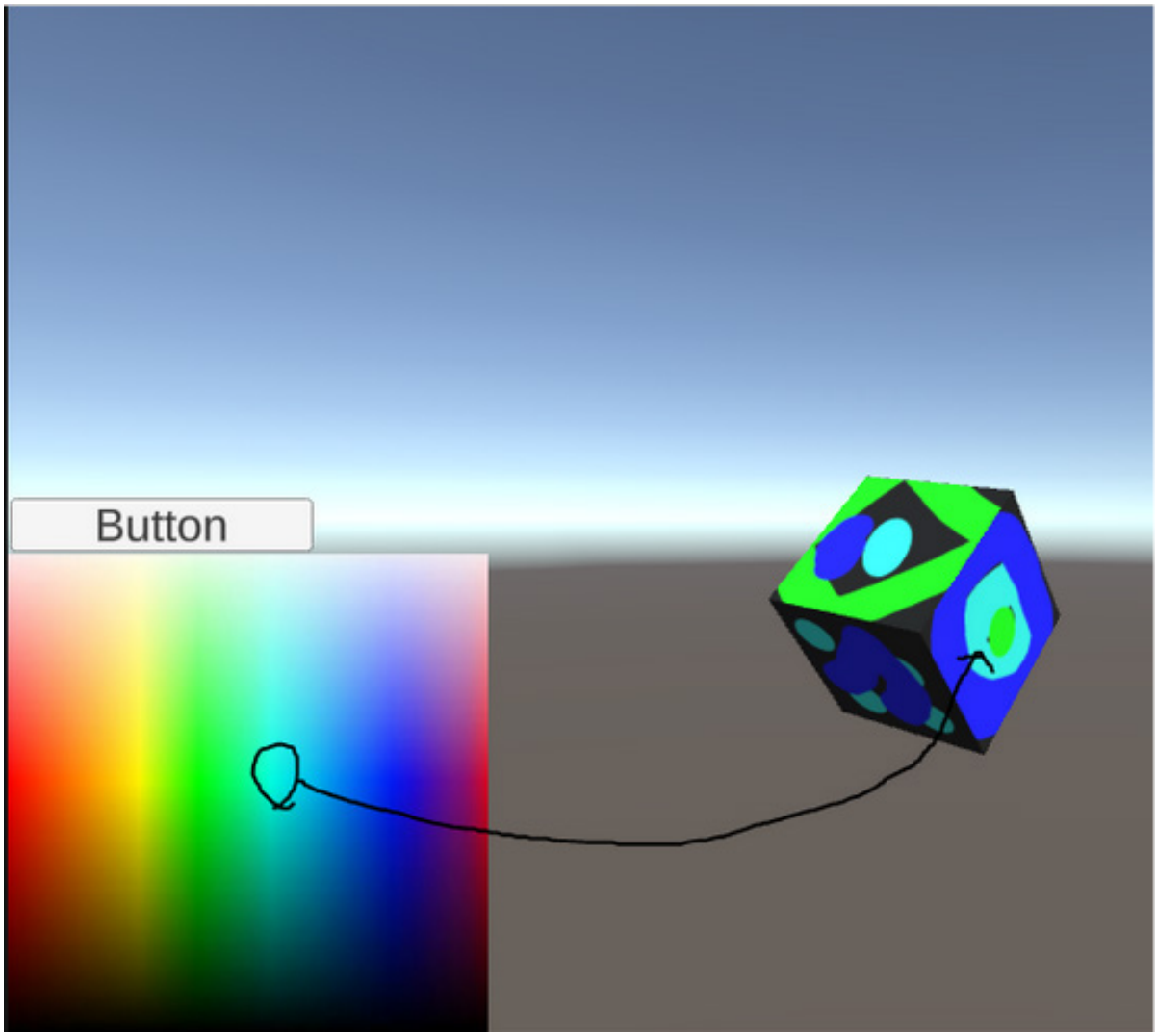
For facial details, the camera zooms in, often cutting off at the bust. Likewise, the camera would zoom out when (/if) there were any body parameters that could be altered.

Something I would like to try doing, if time permits, is to make a more visual menu. As seen in Dark Souls and The Sims, these would give the player a preview of how any given option would look on their character, before even having the item selected. Not only does this inform the user on what options are available, but allows the user to rapidly eliminate options and have a far more intuitive experience overall.

One more observation lies in the interactivity with the model the user is creating. In all of these titles, the user can spin around their creation to some degree. In The Sims and Dark Souls, they can do a full 360 degree rotation, and in Skyrim, the character can turn their head for a greater understanding of the profile.

Month of August

Recoloring Textures
UI from Scratch

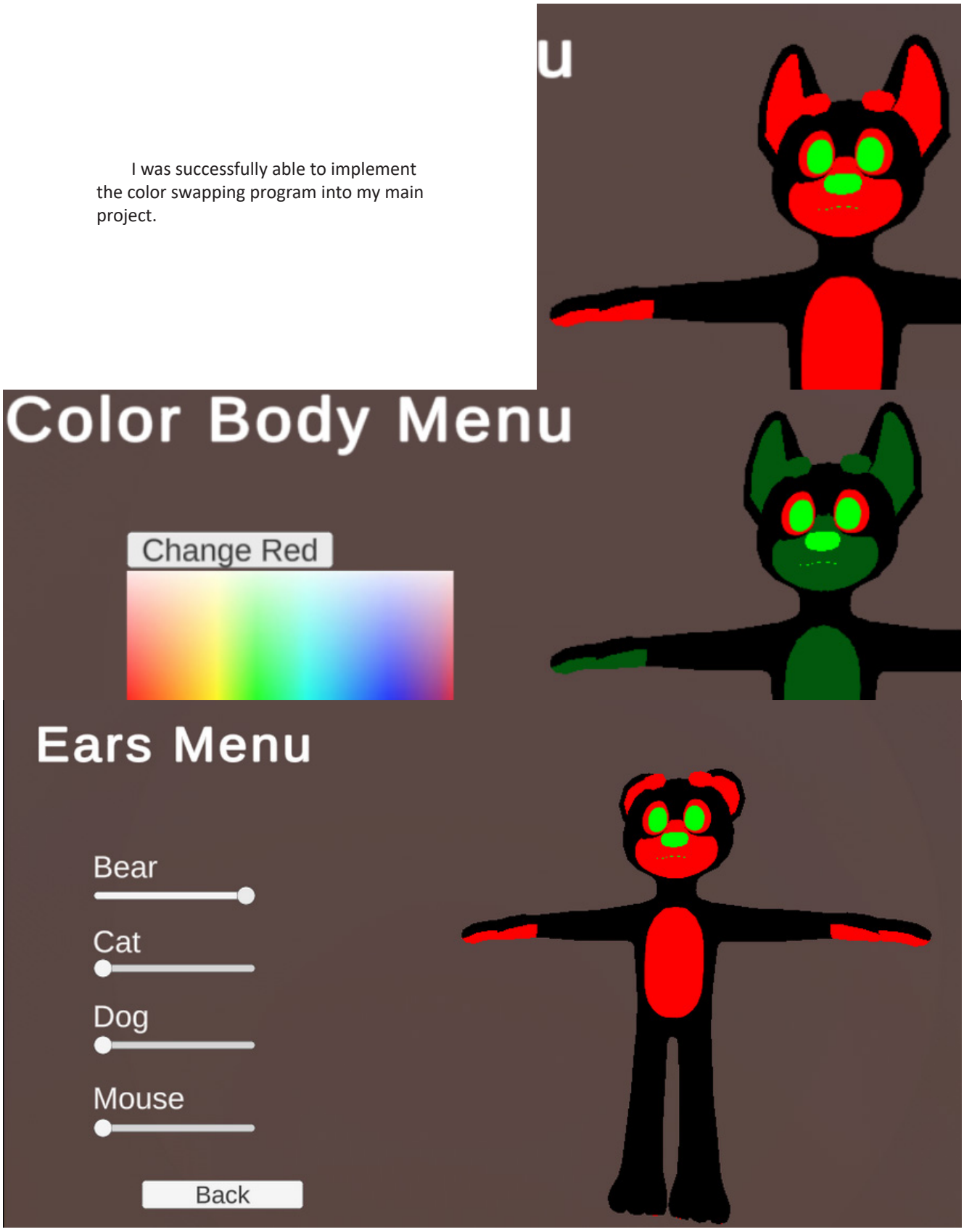


Demo in which selecting cyan changed the red values on the cube to cyan

This month I began work on creating an in-game recoloring mechanic. This would allow users to color the character to their preferred choice. A more detailed explanation of this process is outlined in the Texture section

In this picture, I was able to replace the red pixels on the texture image with a cyan color selected via the colorful image texture on the bottom-left. The downside I foresee with this process is that the textures on the character model must not have anti-aliasing, meaning I must either do blurring on the texture post-process, or not have any color blending at all.

I was successfully able to implement the color swapping program into my main project.

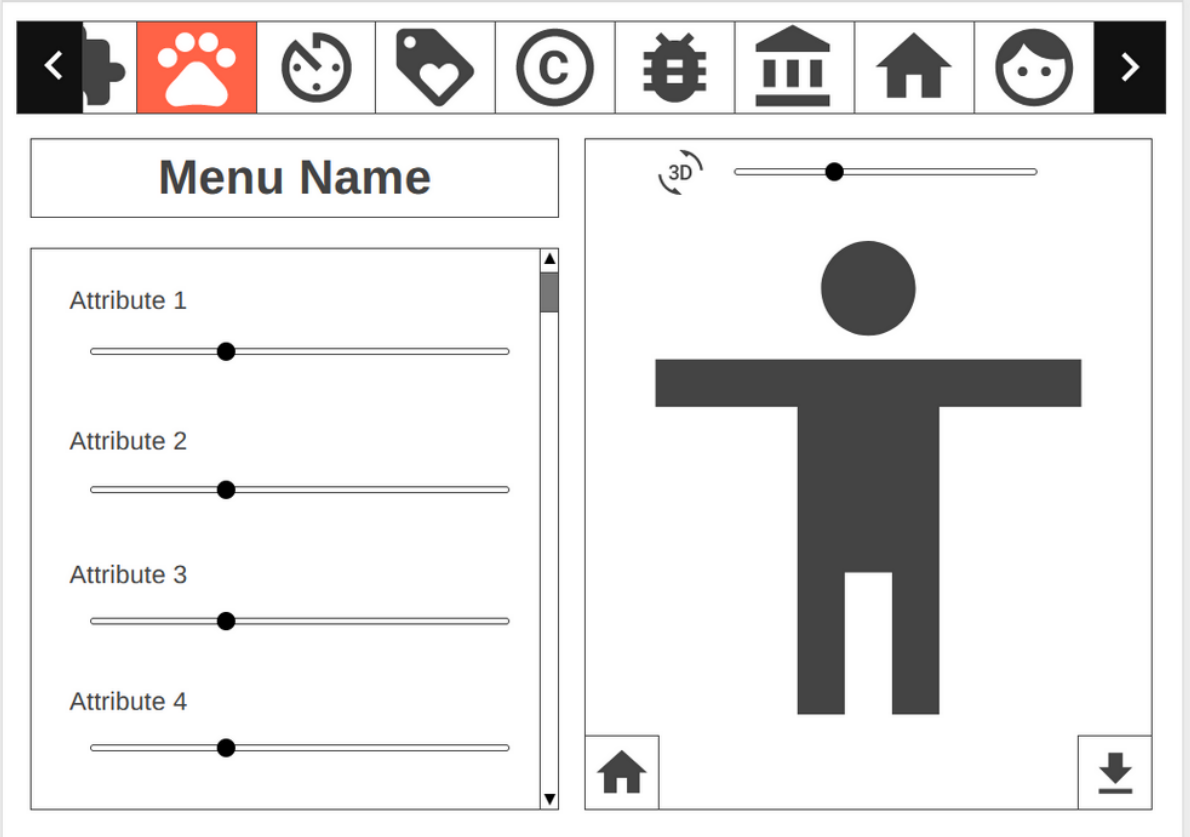


I also did a rapid prototyping of a basic UI system, building it from scratch. There are some errors with this interface, but it allows me to test the different slider interactions without having everything on the screen at the same time.

Month of September

Wireframing

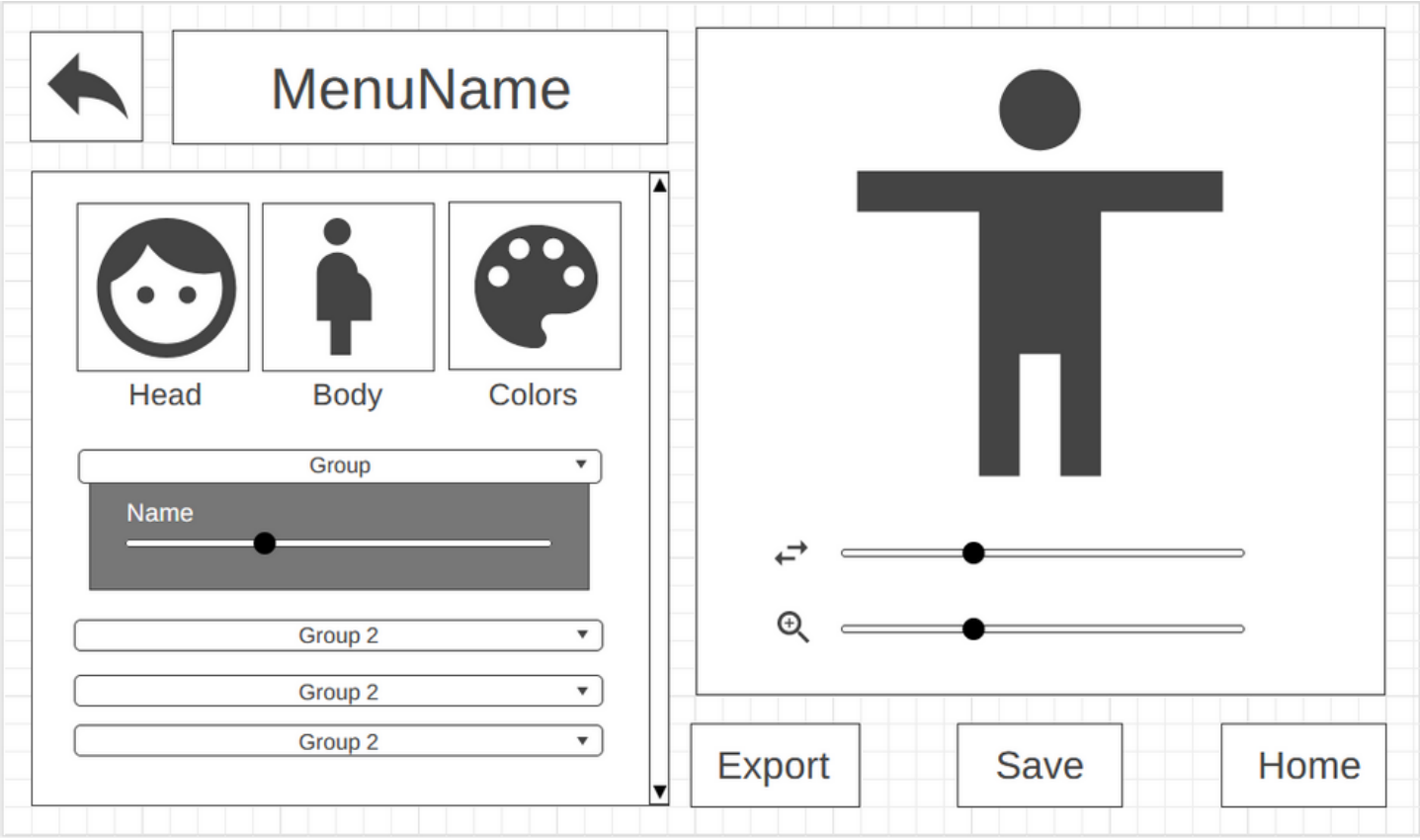
This week I began drafting my UI. This started with the low fidelity wireframe. I used wireframe.cc to layout a principal structure to the UI, so that my progress in Unity could be exponentially quicker. I realized that my last few weeks in Unity involved me struggling over creating a UI. So I referenced some text books and online tutorials to come up with this wireframe here: https://wireframe.cc/mpH6bu. In case this link does not work in the future, a screenshot of the UI is embedded below:



Across the top bar is the navigation menu for features. I believe this to be an unwise decision, as the navigation tree shown in the UI section is far more complex than the single level depth provided by this UI navigation.

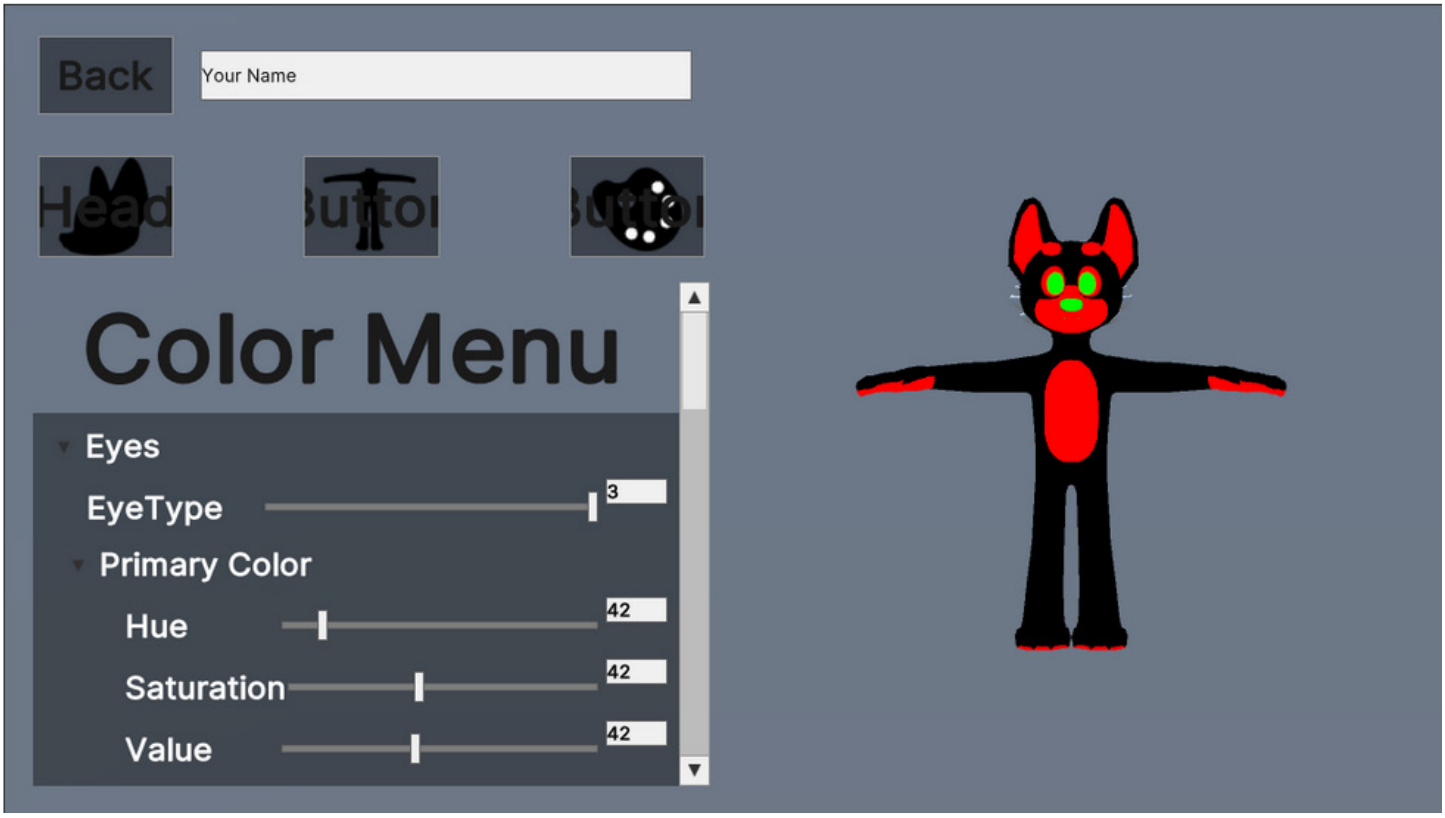
I once again refined the UI Concept Wireframe: https://wireframe.cc/xvLUgD

The three main categories for this planned event will be body, face, and colors. I will incorporate the other areas like Clothing if and only if time suffices. This would become its own section, and likewise have a navigation button next to the head, body, and color buttons



Month of October

Application of UI Wireframes
UI-Avatar Interactions

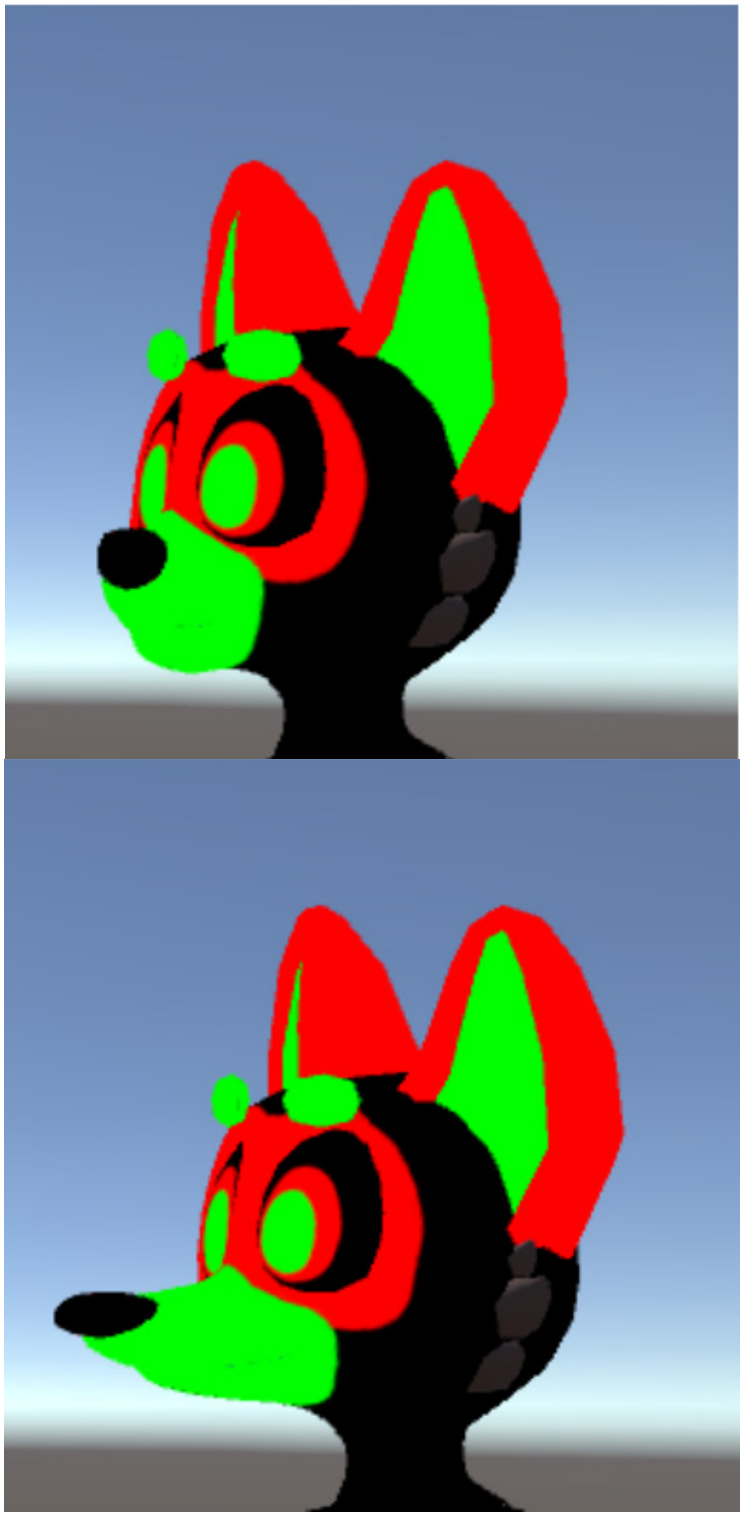


This month I integrated the new UI layout to a rough stage within Unity, Incorporated user interaction, and began programming the UI to manipulate the model. Prior to this month, I was hand-programming my own user-interface. This however was a very slow and tedious task, filled with many complications. For example, how was I to create dropdown menus that when opened revealed more menus from within while simultaneously pushing all previously beneath content further down to make room? This is when I came across Unity's UI Toolkit. While native to Unity, this package was not in the version of Unity that I was using, so I had to install it manually. The result of using this builder is the far more intuitive interface. This also allows for very quick and modular landscaping, in case the layout needs to rapidly change in the future.

The implementation of my UI from September. The implementation is in an early phase, as I have not devoted any time to make the ui scalable nor pretty.

Using this, I easily incorporated my aforementioned UI, and began to program some of the functionality alongside it, such as adjusting the dimensions of the snout of the player:

A current downside to my programming currently is that it runs on the Update() function. This method is called several times per second, and having every single one of my avatar-manipulation functions here (in order to read information from the UI) has caused a very noticeable hit on efficiency. This will be a challenge I will have to face soon.



The Snout length Slider when turned to minimum and maximum values.

Month of November

Function Calls on UI Changes
Weight Height Parameters
Beginning of clothes / Accessories
FBX Exporting

```
//change antler type on slider change
antlerType = root.Q<SliderInt>("Antlers");
antlerType.RegisterValueChangedCallback(v =>
{
    gameManager.avatar.ChangeAntlers(antlerType.value);
}
);
```

The function “RegisterValueChangedCallback()” allowed me to edit the avatar with the slider’s value

This month revolved around refactoring, polishing, and finishing everything I had set out to do. The first thing I tried doing this month was getting the UI - data relationship streamlined, with a far smaller Big O.

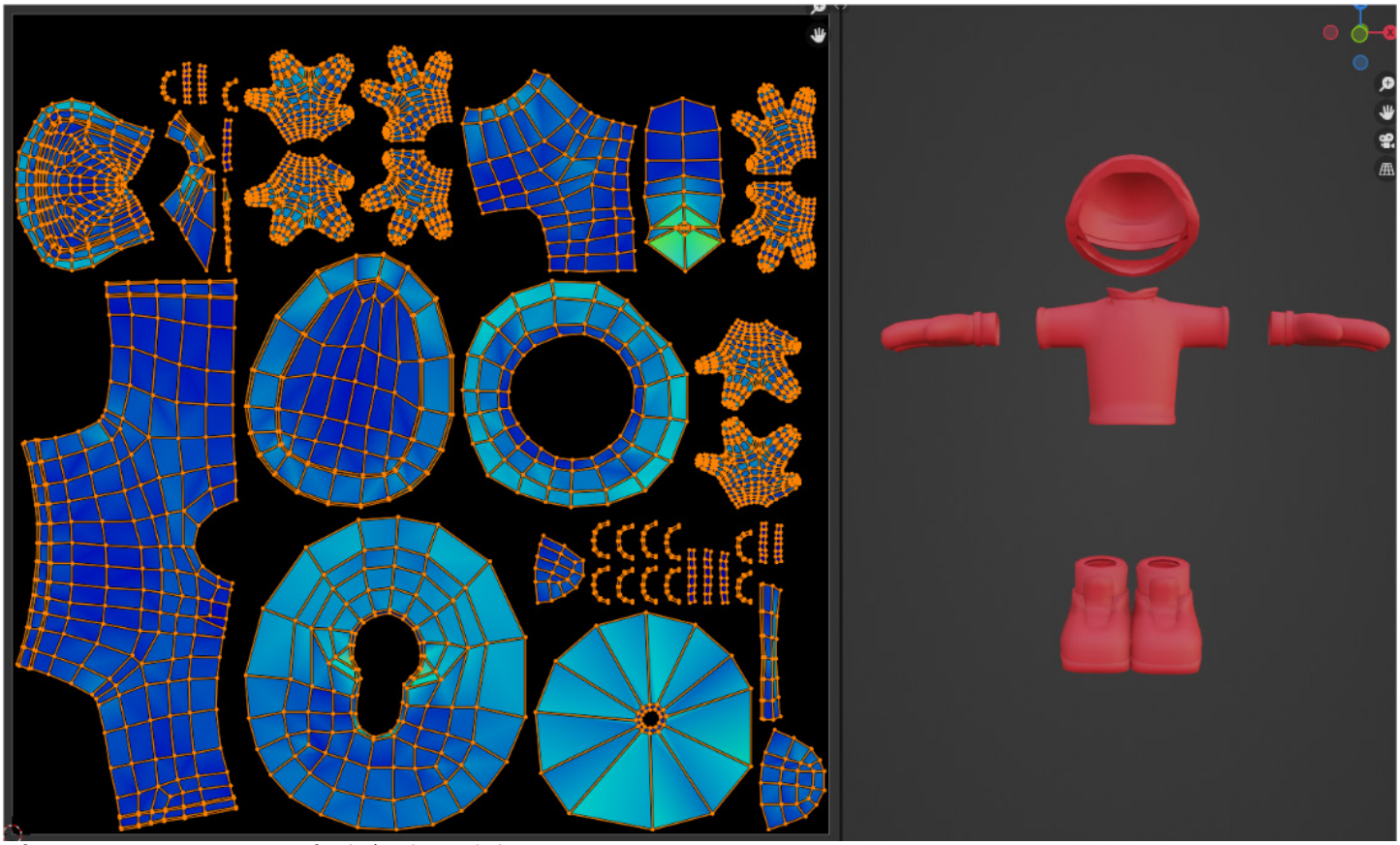
Previously, I had just used the built in update function to assign the data from the UI onto my variables, which would then change the features on the model. However, this meant checking every single slider multiple times a second, even if said slider was not touched at that time. To my knowledge, this was a linear Big O, but with so many function calls occurring every second, it was lagging even my computer. With my switch to Unity’s UI toolkit, I lost the ability to call the slider event OnValueChanged. Originally I thought it would be impossible to update my values without the update function running. However, I was able to find a very niche function within the Unity.Engine.UIElement namespace: RegisterValueChangedCallback. This allowed me to update variables whenever a slider changed values. Rather than updating all slider values several times a second, I was now able to update my data only when the slider for that particular variable was altered. This truncates the Big O to constant, as the function is now only called once. (https://docs.unity3d.com/2022.1/Documentation/Manual/UIE-Change-Events.html)

With this slider event finally figured out, I began work on the more complex methods, like the weight and height parameters. Scaling the height means scaling up the length on a lot of bones to ensure proper proportions are maintained. A challenge I faced here was integrating the height and weight event triggers. Initially I had each in their own function; the change height function would alter the length of bones, while change weight would alter the thickness of the bones. There were many issues with this approach, as I had soon discovered.

For context, bones in Unity operate in forward kinematics, meaning that any transform applied to a parent bone in the skeleton hierarchy would also apply to children bones. So rotating the shoulder would rotate the forearm, hand, fingers, etc. The same applies with scaling. The arm bones are children of the torso. Elongating the torso is done on the Y axis, while elongating the arms is done on the X axis (via the global space, not locally; both are elongated via their local Y axis). As a result of elongating the torso, the arms deform along the improper axis, and need to have the transform inverse applied on their own transform. This is easily accomplished; however, the X axis is also deformed in the weight function, so the X axis scale for the arms was constantly being overridden by whichever function was being triggered.

Different combinations of Weight and Height. All adjustments are calculated within the AvatarDescriptor Class.

My solution to this problem was to combine both functions into one and take two parameters (weight and height) as the parameters, instead of just one. This allowed my deformations to take both active variables into account when calculating the scale deformations.



Left: UV texture space, unwrapping of Right | Right: 3D clothing
The clothes dilemma, further described below

I just reached my third iteration of the cycle, and started on modeling the clothes and accessories. I am torn on how to do the texturing for several reasons. For one, with a long list of accessories, I would have to make everything small on the uv-texture space. This would heavily pixelate any detail that I put onto the clothing texture. Second, If all the clothes are unwrapped onto a single texture, then adding more clothes in the future would require me to completely redo all clothing textures of the past. If I added pants to the unwrap shown above, then everything else would need to move and scale down to make room for it. The result would be that everything which was drawn on top of the original spaces would no longer project onto the same faces, resulting in a very distorted texture. The easiest solution would be to give every clothing piece its own material, but importing this into games like VR Chat would no longer be possible since it would use too many materials to import properly

I then tackled texture recoloring again. In an earlier version, I had a working color picker that retextured each color flawlessly. However, completely reusing that code introduces two problems: First is that the flexible color picker is not a UI element that can be accessed in Unity's UI toolkit. The UI toolkit sits above Unity's 2D canvas (where the color picker rests), so any use of said color picker will have to be done after disabling the UI on part of the screen. This means I cannot use the toolkit's collapsible menu to display multiple color pickers at once.

Before I began on reworking this, however, I realized that the quality of the project would be much higher if I utilized an outside color picker, that already came in a visually pleasing and interactive package. Thus I went to the Unity asset store and grabbed the flexible color picker, by Ward Dehairs. This was a completely modular color picking asset, which allowed me to rid of the parts I did not need -- such as the saturation and hue sliders. Using this, I quickly crafted a color picker to suit my needs, and going forward, this is the color picker that is used in the project.

The new code I created works with Unity's UI toolkit. Rather than having the color picker be in a collapsible menu, it is in its very own menu. To access it, the user will have to select which of the four colors they wish to edit. The button, when clicked, will hide the original left menu and instead display a single color picker. By hiding the UI on the left side of the screen, the color picker will be able to show through and be interactable.

The second issue with the original script is that I am now using two texture spaces. Whereas in the past I only recolored the body, I am now also recoloring the eyes. I have not yet made the program capable of working with two texture spaces at once.

The aforementioned buttons solve this issue too. When pressed, these also indicate which texture source the user is working on. If the button is in the eye menu, it will call a function to set the texture workspace to the eyes, and vice-versa for the body. Whereas before the character had one RecolorTexture script attached to it, there are now two, both of which loaded into an array by the Avatar Descriptor. The texture space can be specified with an index. In this particular case, fur is the 0th index, and eyes are the 1st. As stated prior, there is only one color picker present here. I felt

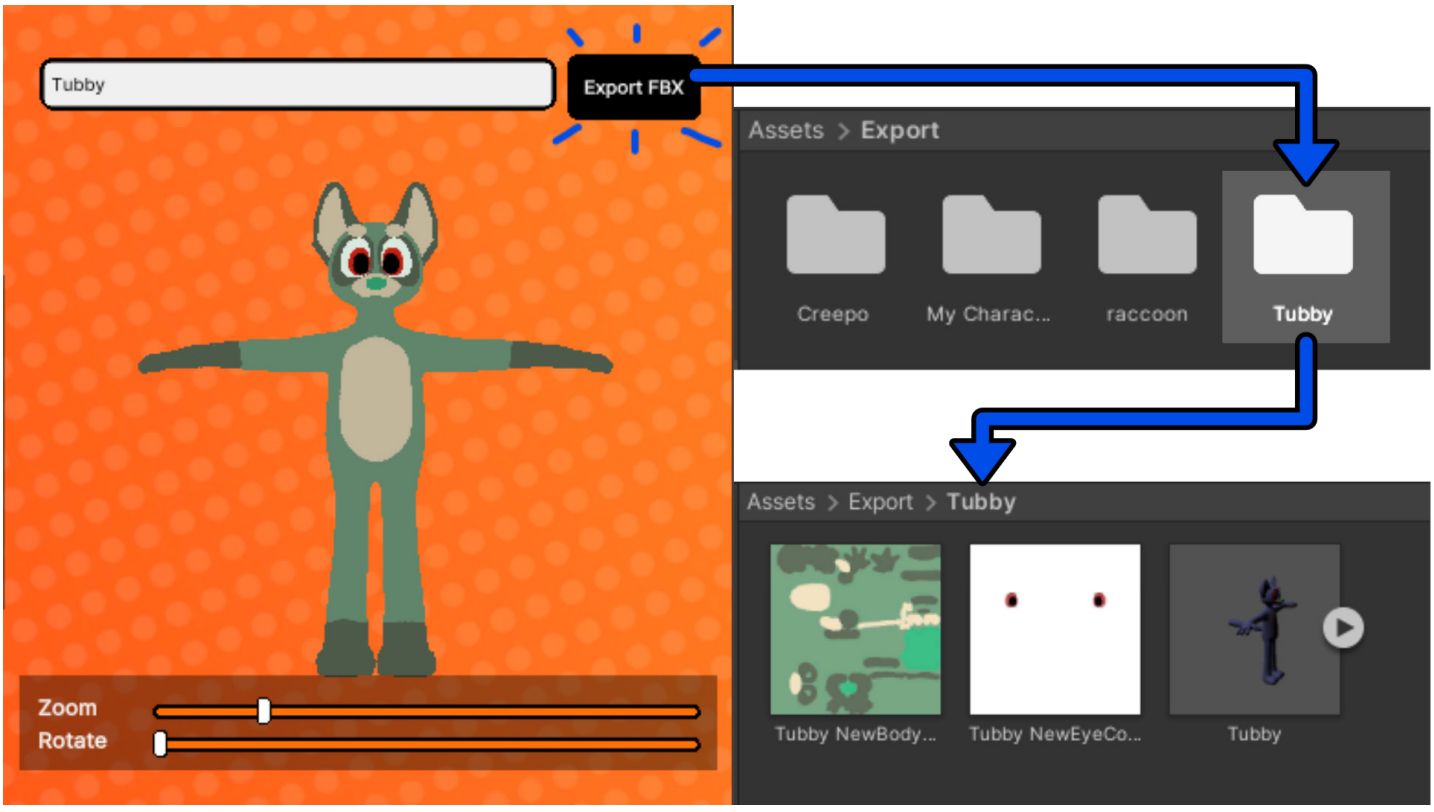
it more intuitive this way, but to make it fully work, I also had to program functions to save and load the colors for each recoloring on each texture, so that the color picker always displayed the proper color value that was being edited. This also prevented one color from recoloring another when the color picker's onColorChanged value was altered.

After this I moved my focus to the exporting functionality. This tool would be completely useless without this. To my surprise, Unity actually had a built-in package for this use-case, too! Unity FBX exporter, with some custom coding, allowed me to export my edited model and textures to an output folder, which I could then take into other programs.

Everything felt perfect, until I went to build the application. As it turns out, Unity FBX exporter is an editor-only package, meaning that it would not work on a standalone application. This has stumped me for quite a while, but I do know there is a possible way to export my model via the Autodesk.fbx namespace. From what I've read, this namespace works with standalone applications, but would require extensive programming from me to get it to work. Hopefully soon I will be able to add this into my program, but if not, I will make my application into a unitypackage instead of an exe. This would limit the program to being inside of Unity at all times, but would give me the power to export everything comfortably.

When exporting my model and importing it into a Blender, I notice a very weird event to occur. While the model imports as a T-pose, any rotations I apply to it will completely distort the model. Upon research, I have found this to be a result of Unity's non-uniform scaling. Whenever I programmed the height and weight functions, I had the character's bones scale disproportionately in the X,Y, and Z axis. This scale is globally applied upon export, meaning that if I rotate the elbow down, the scale applied to it shifts from the X axis to the Z axis, leading to an immense shrinking or growth of the limb (picture shown in the challenges section). There is a very round-about way to fix this method, but that is within the 3D application after the model has already been exported. For this, I cannot script anything within Unity to fix this. However, I will keep looking for an answer going forward.

One last detail I added to the program was overhauling the UI visually. Something I realized when working on the project was that the flat texture oh the character meant they could blend into the background if the correct color was selected. Thus, I created a background that combined gradients with a dot texture pattern to make the character stand out. I also added UI interaction, so that hovering over a button would switch its color to black, giving users much needed user feedback. Lastly, I overhauled the default mouse shape with some custom mouse designs I had created. This gave a cohesive atmosphere to the application, and it finally felt like it could stand apart from the unity editor.



The exporter, when used within the editor, allows for users to export their model and provided textures. It also removes any unrendered (unused) props / parts. This keeps file size small and gives the user exactly what they need for their projects. Currently there is no provided way to export models during runtime.

Conclusions

Given More Time

Going forward with this project, I hope to implement concepts that expand previously mentioned concepts, or add new ones entirely:

- **Custom texture uploading**
- **Mobile Support**
- **Purely iconographic navigation**
- **Clothing and Accessory implementation**
- **More camera controls (move camera, rotate on other axis)**
- **More thorough animation / interaction system**
- **Simple expressions on the avatar, like anger and sadness, for use in external softwares and animations.**
- **Creating a Run-time fbx exporter.**
- **Undoing the issues caused by Unity's local scaling system.**
- **An in-game simulation, where players could play as their character and preview all the animations in a platform freeplay environment.**
- **I also hope to start building a brand around this software, so that I may one day sell this as a real application to real clients. This will be hosted through itch.io. Next semester I am taking my art capstone, in which I could build the brand up for this application. This would include creating logos, banners, a cleaner UI system, advertisements, and motion graphics.**

To Answer the Questions

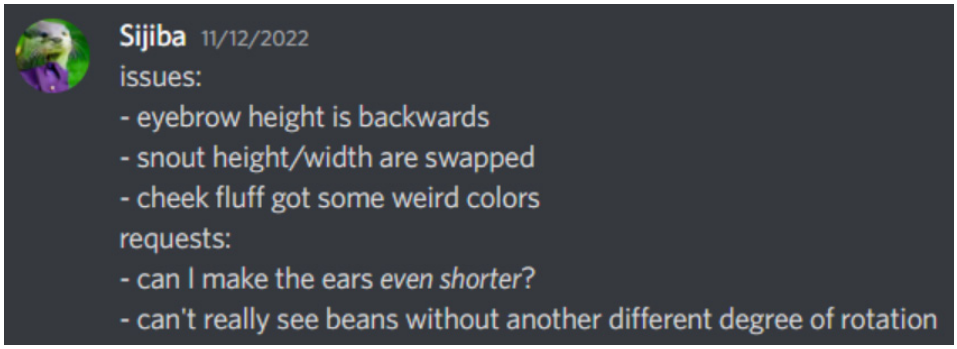
- **Describe the physical, social, and development environment of your project or Internship.**

This application was developed on my home computer. I had built the environment around me to be comfortable and inviting, with space to pace back and forth in case I got stuck on something. The social environment was completely online, which made communication efficient and non-time-specific. This relaxed the urgency for information and let me focus on my project more often, while giving people across the globe the time the needed to test my software. Development was done in Visual Studios and Unity, which developing on was completely new to me.

- **Characterize the extent to which you worked with other people. Describe the communications skills you used.**

I dug through countless online forums and tutorials to troubleshoot my issues. This communication was typically one-way, as the answer was already out there on the web, and I just needed to find it. I also had beta testers which I communicated with on the last stretch of the project.

I also communicated with beta testers, who tried out my software and gave feedback. They were able to weed out bugs and provide suggestions for future patches. Communication with beta testers was held through Discord.



- **Describe the computer hardware and software platform.**

The capstone project was created on my Windows 10 computer. The current specs of the device include an AMD RX 580 graphics card, Ryzen 5900x processor, 16gb of RAM, 2tb NVME SSD, and 500 gb SSD storage.

The softwares I used to create this application were Blender 3.X, Substance Painter, Affinity Designer, Photoshop, Unity 2020.3.25f1, and Visual Studio 2019. This project was coded in C#.

- **Describe the CS courses or topics which were particularly useful.**

Given the similarities between Java and C#, taking Object Oriented Programming and Advanced Java allowed me to very quickly get comfortable with the C# language. This was especially true for the declaration of variables and the class-oriented structure of these languages.

Besides these, Data Structures and Algorithms, as well as Computer Graphics, were probably the two most useful classes in regards to this project. Data Structures taught me the value of refactoring and minimizing the BigO of a program. I fought hard to reduce program times to constant or logarithmic. Computer Graphics I believe was done in Java, and gave me some base familiarity with coding in relation to manipulating shapes. While this was on an entirely different calipur, the foundations I grasped from that class allowed me to go the distance.

- **Describe the non-CS courses or topics which were particularly useful.**

I would not have been able to create this 3D model without the knowledge from my 3D modeling courses. Furthermore, these classes lent themselves for my understanding of 3D animation, too.

Within Unity's UI toolkit, there was a programming language called uss. This, of course, is a reference to the web's CSS programming language. Without first taking the web design class, I would have been very lost when creating the UI for this game.

The Unity bootcamp was very essencial for understanding how the Unity interface interacts with code, as well as the process of programming a game. The bootcamp taught me to use a central game manager script that communicated as the middle-man between classes.

- **List and briefly describe three things that you learned.**

Programming in relation to a game engine. Whereas programming before had a specific goal and a specific way of achieving said goal, game development felt like an art. There were many ways of achieving the result. Furthermore, relying on a game

engine API felt like learning an entirely different language. There was a lot of learning involved in this, despite my four years of programming at Truman.

Manipulating complex 3D objects with code. Wheras it is easy to make a primitive cube move across the screen, there are far more interactions that a character has with code. There are animations that can be activated on a trigger; there were materials that needed to dynamically alter based on user feedback; there unique bones that needed to be transformed in very specific ways.

Utilizing other people's assets. The color picker, as mentioned in this document, is an asset from the Unity asset store. This means that someone else had programmed its functionality and distributed it for others to use. Reading through this person's code gave me a newfound appreciation for pseudocode. I was able to save and load the user's selected colors on each texture by reading through the colorpicker's code and utilizing the necessary functions provided by said programmer

- **Indicate if there is anything you now realize that you should have known or studied before starting your capstone experience.**

Realizing the existence of Unity's UI toolkit before starting on this project would have saved mountains of time. Not only did I code my own UI from scratch when starting, but I also spent countless hours trying to absorb everything I could about Unity UI builder in the last two months, which ate away at time I could have spent adding more to the application.

In general I wish I had spent more time in Unity prior to this project. There was -- and still is -- a vast amount of knowledge to learn in this engine to be proficient with it. While the project result is great, there are probably many optimizations that can be made that I am simply unaware of.

- **Explain whether the experience turned out the way that you anticipated that it would**

When I first started, I thought that the process would go much smoother and quicker than it turned out to be. I had underestimated the amount of knowledge I would need to amass before endevoring on my first Unity project, though I managed to learn most of what I needed along the way. Overall, I am happy to finally put my coding skills to use on something that feels more real-world than the projects in university. I do wish that I knew how to export fbx in runtime, though. That in itself seems like a capstone project!