Character Creation

Wednesday, March 25, 2020 7:58 AM

Character Creation in Unity using Shape Keys(Blendshapes)

Introduction

If you are developing an RPG, you are quite possibly looking for ways to improve your efficiency when it comes to characters. (Especially humanoids in our case).

In this article, we will be focusing on how to easily improve character variation, without iterating characters in a modelling software. We will cover these aspects for the first part:

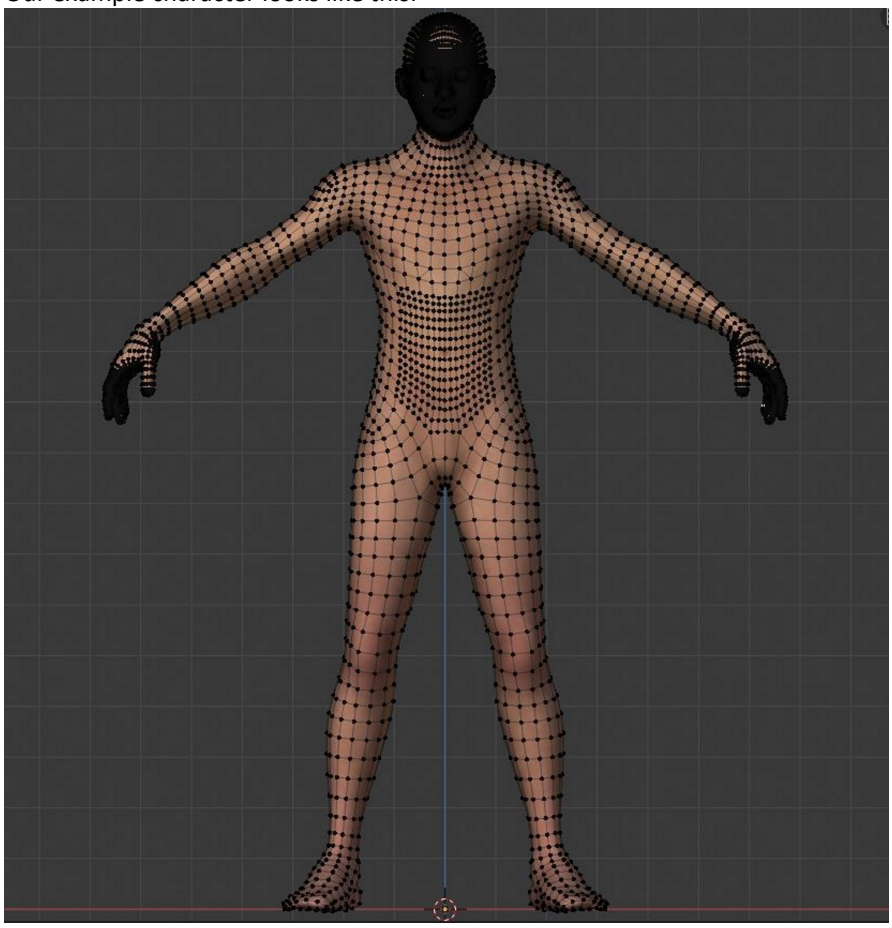
- Eyebrows
- Skin colors
- Eye colors
- And finally, body and face morphing with shape keys

We will be using:

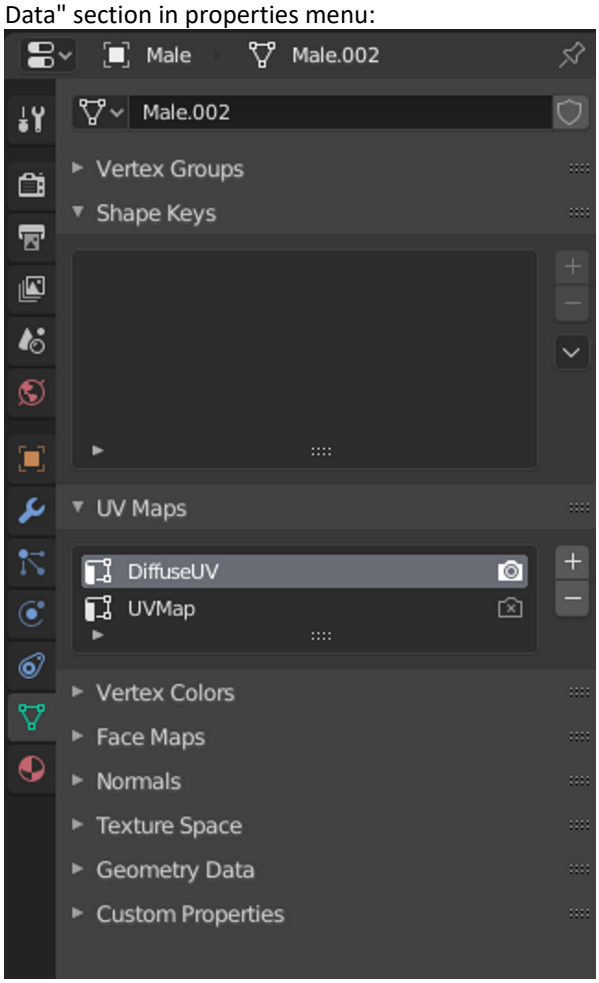
- Blender 3D for modelling and creating shape keys, hair styles etc.
- G.I.M.P or Photoshop to edit 2D textures we may use.

The blender version used in this article is 2.8, but everything that has been covered in this tutorial should work the same in older versions too.
This article will be assuming you have your rigged character model and have the basic modelling knowledge in blender.
(Burada karakterın lınkını verelım.)

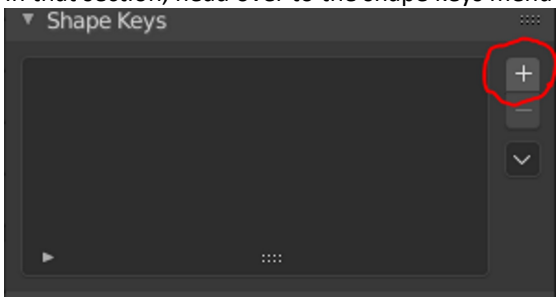
Our example character looks like this:



First of all, we need to create shape keys inside blender. To achieve that select your mesh in object mode and head for the "Object Data" section in properties menu:



In that section, head over to the shape keys menu and click the "+" button at right.

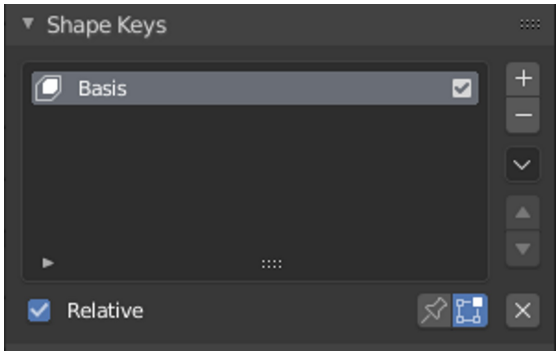


This will create a shape key named "Basis". As the name suggests, this will be the shape key blender will be using as a basis for your other shape keys.

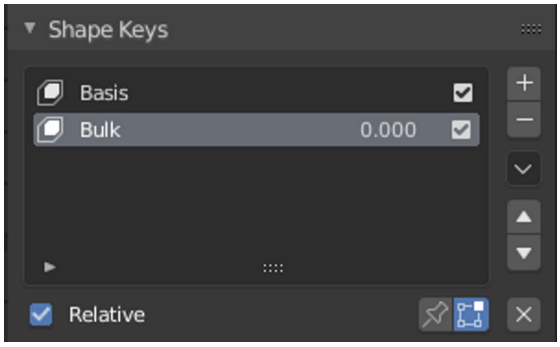
Please note that shape keys do not take topology into account. They only use vertex ID's and their relative movement from the basis. So if you add or delete shape keys after you have created the Basis, most likely you will have issues with lighting, UV's or the shape keys themselves.

There are 3 variable fields below the shape key list.

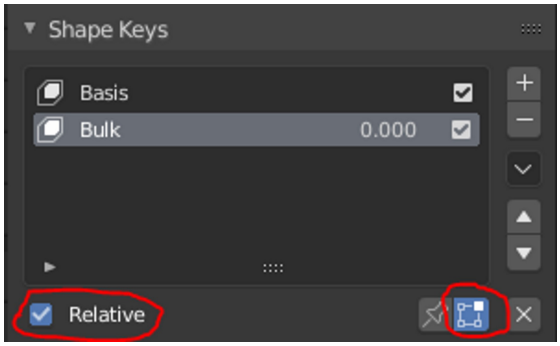
- ***Value:*** *This is the shape key multiplier that adjusts how much current shape key is effecting the "Basis".*
- ***Range Min:*** *This defines how low "Value" can be.*
- ***Range Max:*** *This defines how high "Value" can be.*
- *If "Range Min" is set to a negative value, the effect of the shape key can be reversed.*



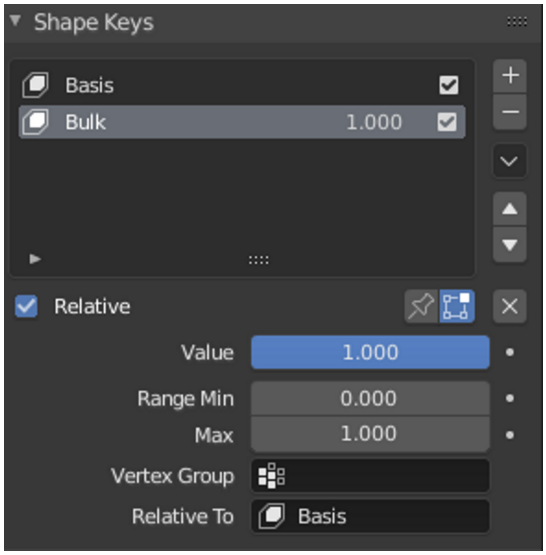
Now that we have created our "Basis", we can add our new shape key which will be holding the movement data relative to the "Basis" for each vertex. In order to do that, just click the "+" button again. Double click the shape key to rename it. For educational purposes I will call it "Bulk". With this key, we will be able to control our character's body size with a slider.



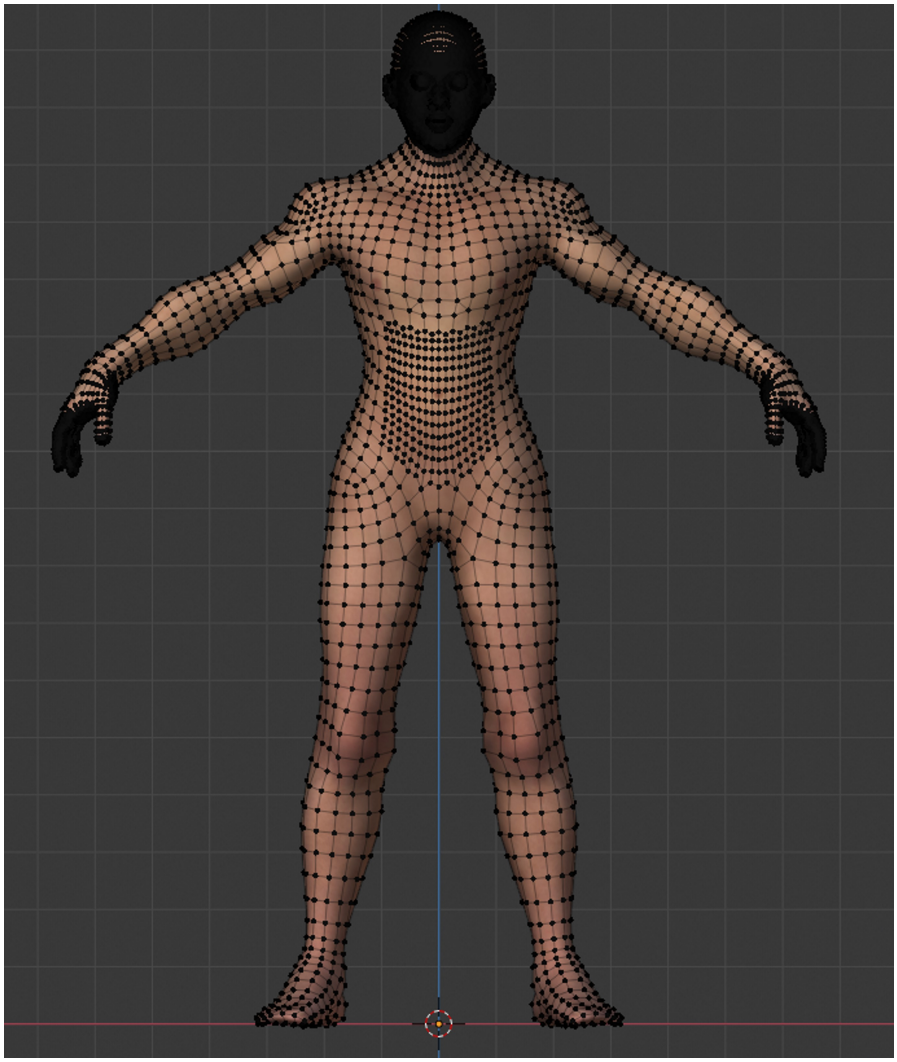
For an easier workflow I suggest activating "Shape Key Edit Mode" toggle. And keeping "Relative" toggle on. This way, any changes on a spesific "Value" setting can be observed realtime. (This means you can set how your mesh should look at, for example, -0.5 value).



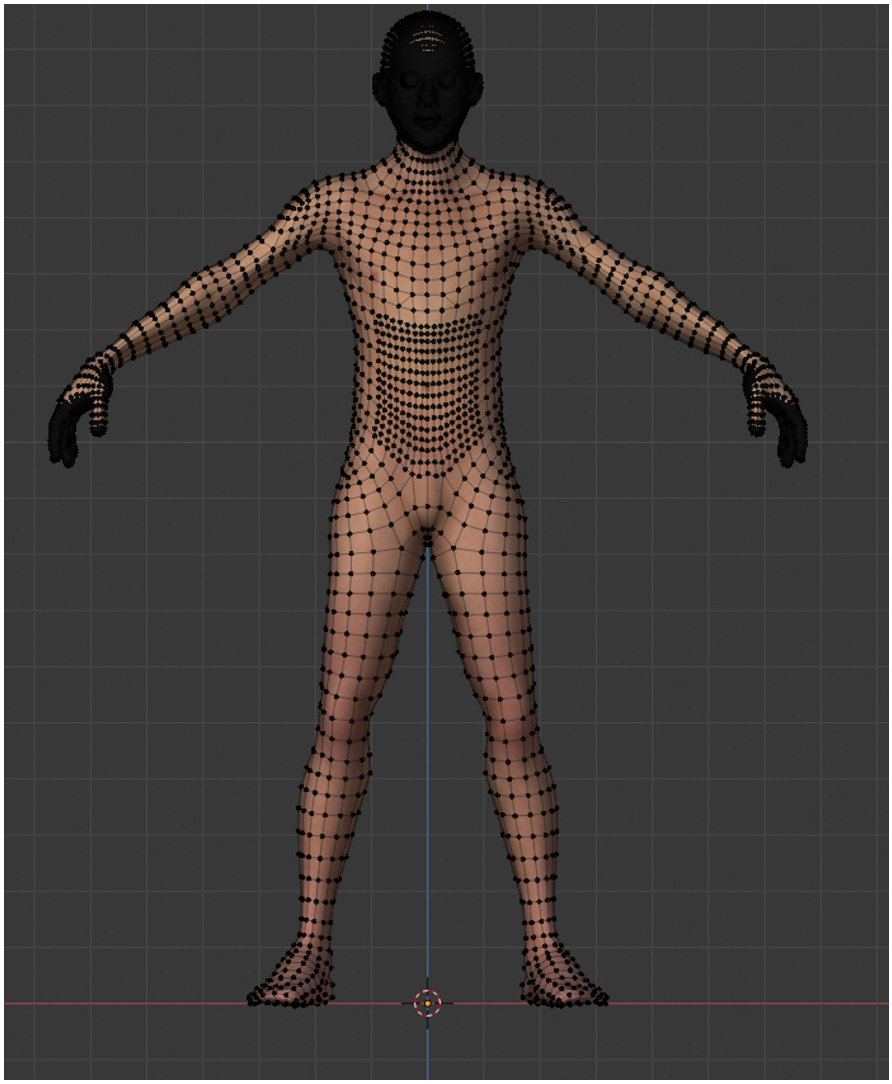
Now with the "Bulk" key selected, enter the edit or sculpt mode and set "Value" slider to 1. Now you can change your mesh however you like. This will define how your mesh will look like when shape key slider is maxed out.



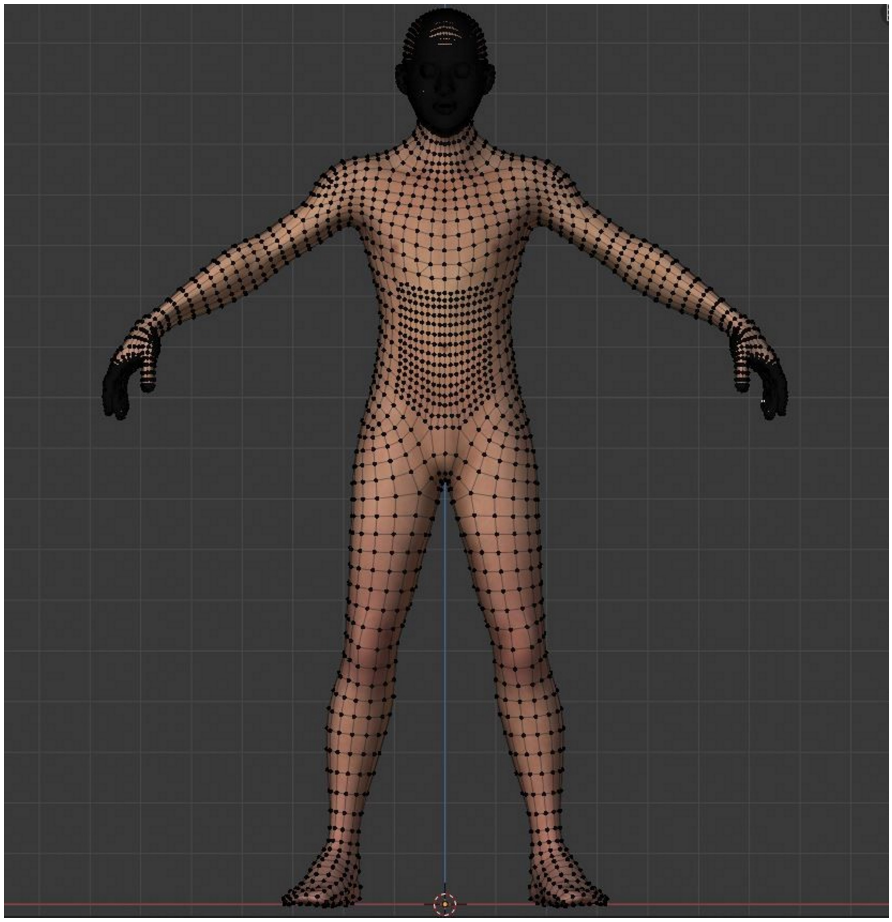
After final shape for the key is ready, we can now adjust the shape key value to morph between "Basis" and "Bulk".



Shape Key "Bulk" at Value=1



Shape Key "Bulk" at Value = -0.5



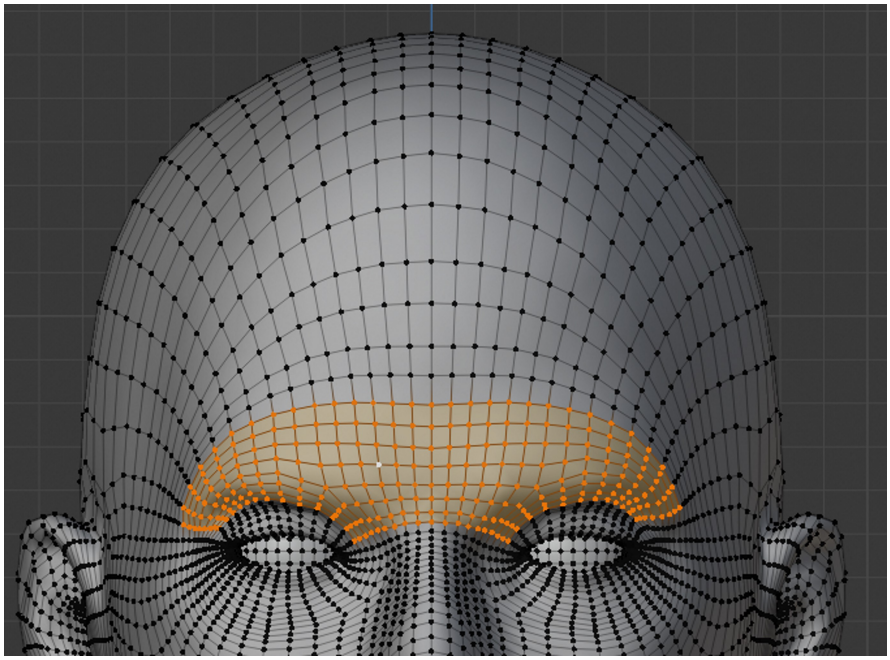
"Basis"

You can now repeat this process as much as you want to create different shape keys for many different purposes. At this point feel free to play around and get creative!

Replaceable Eyebrows

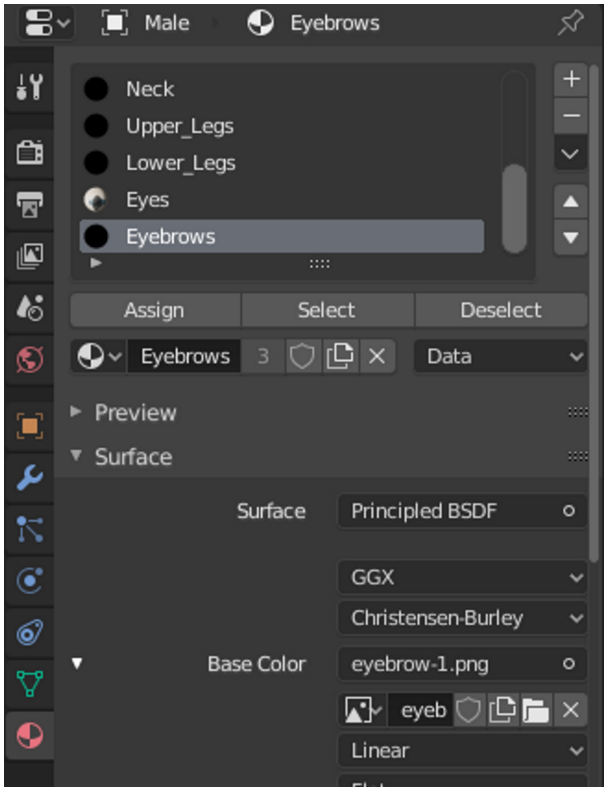
This is a tricky effect when it comes to character creation, and there are many ways to achieve it. This article will cover an easy and dirty solution.

Make sure you selected "Basis" before doing any changes to mesh. Because if you make any changes to basis, other shape keys will adjust to it since they are all relative to the "Basis", so you don’t have to make the same adjustmen on all of them. Switch to "Edit Mode", and select all the faces needed to paint a eyebrow on.



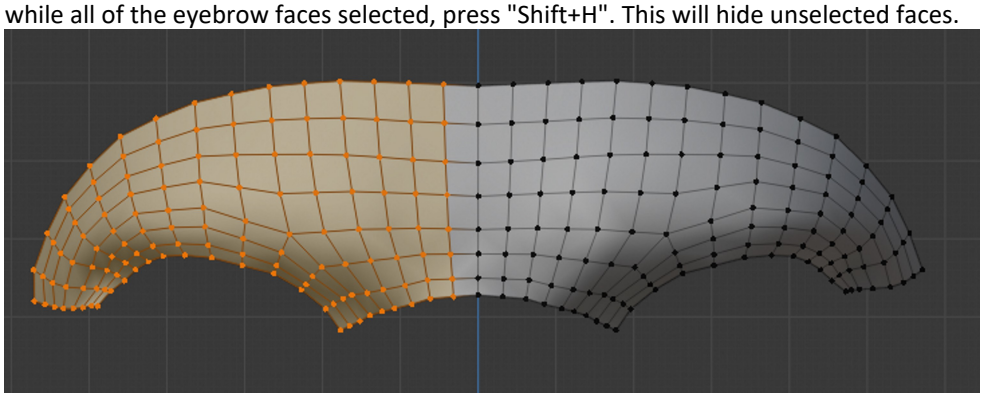
Press **Shift+D** to duplicate them. Make sure "**Pivot Point**" is set to "**Median Point**". Press **Alt+S** and set the value to 0.00001. This will deflate the vertices in a very small scale, in order to prevent "Z-Fight" between the face and eyebrow.

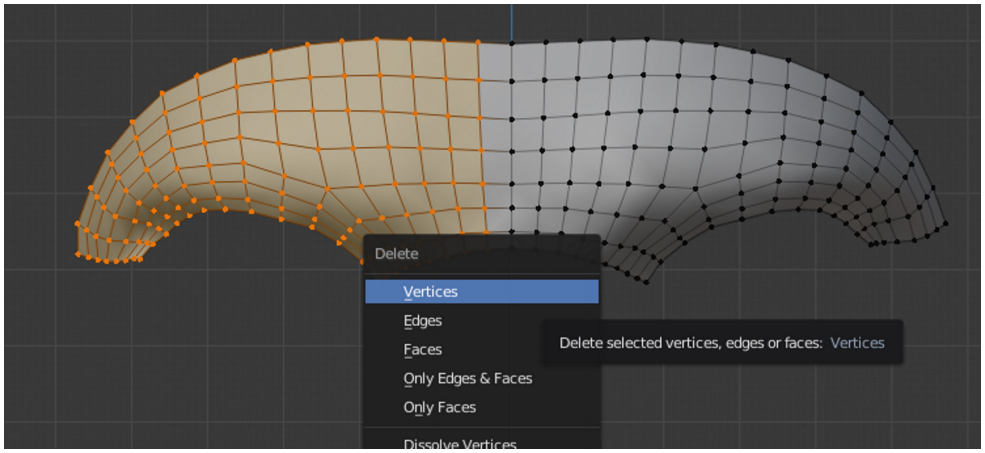
While newly created faces are still selected switch to **Object Mode** and go to the **Material** tab in **Properties** window. Create a new material using the "+" button at the right side of material list. Name the new Material "Eyebrows".

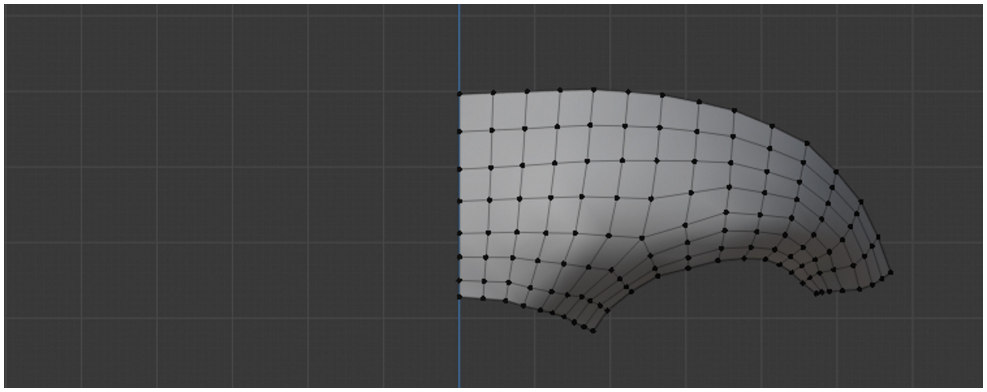


Switch to **Edit Mode** and press "Assign" to assign new faces to the "Eyebrow" material.

The UV's of the new faces are still from the original model. In order to increase UV island size, and achieve symmetry automatically, switch to **Edit Mode** and select only left half of the mesh and delete it**.** In order to isolate only the faces we work on, while all of the eyebrow faces selected, press "Shift+H". This will hide unselected faces.

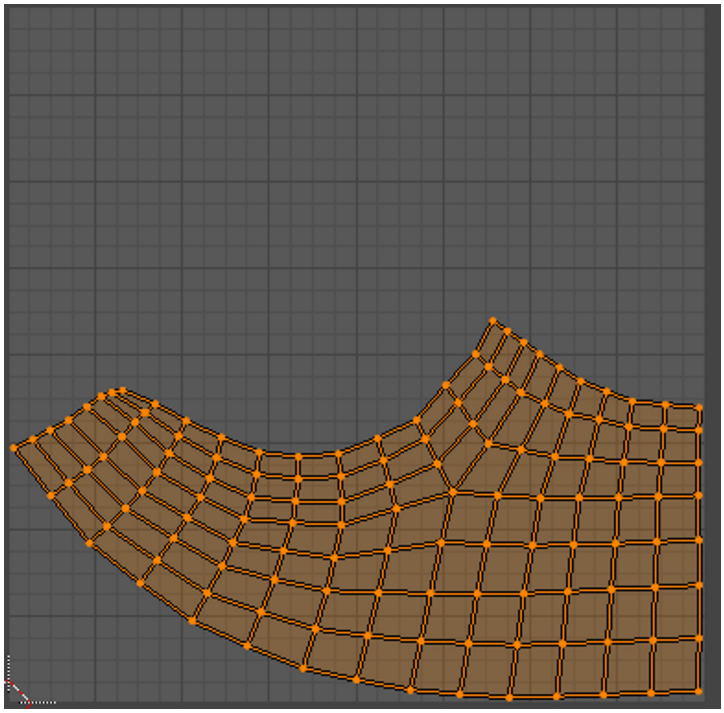




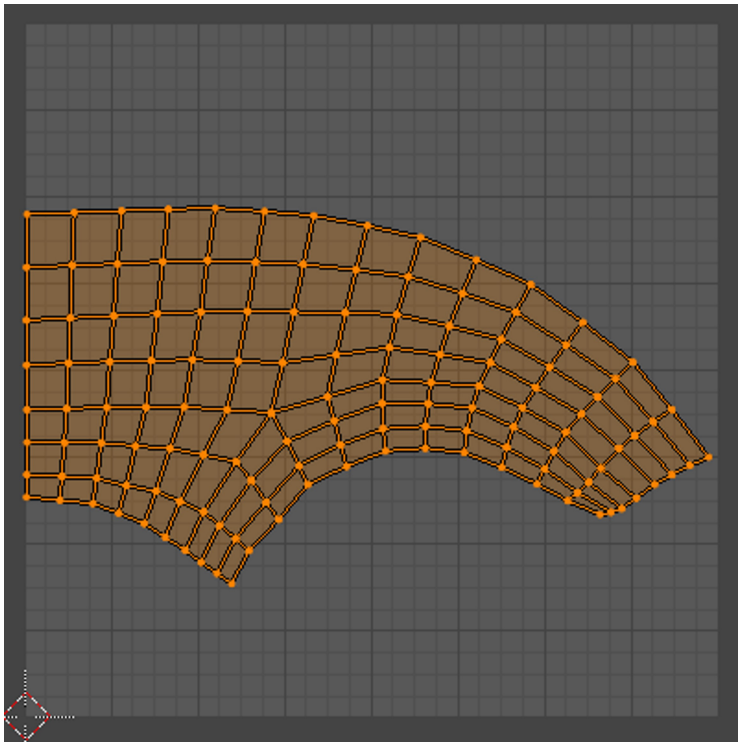


Select all of the remaining faces and unwrap the mesh by pressing "U" -> "Unwrap".

UV Editor window should look like this:



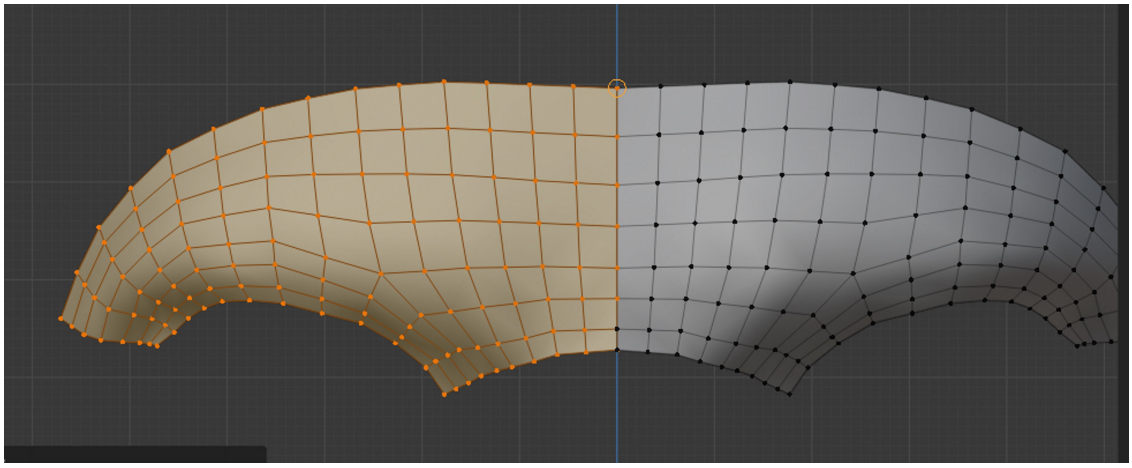
If the UV's are flipped like this, rotate them by 180 degrees and snap to the left-side of the UV grid.



Now return to 3D View and duplicate all faces using "Shift+D". Mirror them by using "Mirror" command (available commands are searchable by name, in 2.8 use F3 key, older versions use 2.79 key) and press "X" to mirror it by the X axis.

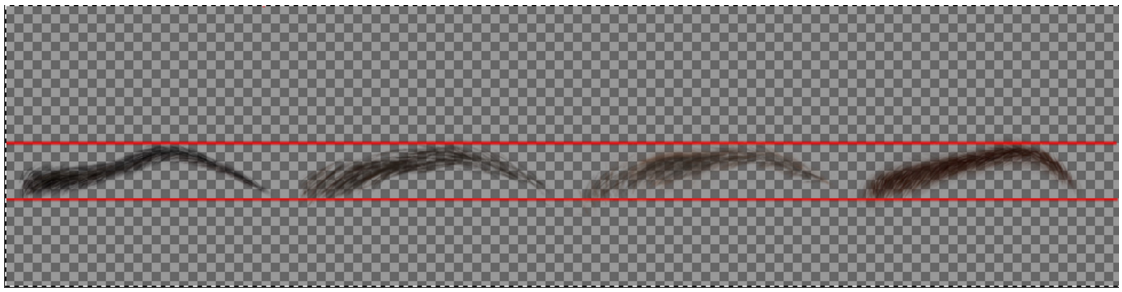
Normals of the mesh may be flipped, in that case use "Shift+N" to recalculate normals.

Set snapping mode to vertex and snap left half to right one.



Select all vertices and merge the middle by using "Merge by Distance"(2.8+) or "Remove Doubles" command. Now press "Alt+H" to reveal all hidden faces.

Eyebrow material is ready to be textured in Unity. Only thing you need to do from now is make sure the eyebrow positions in your textures are similar. And transparent.

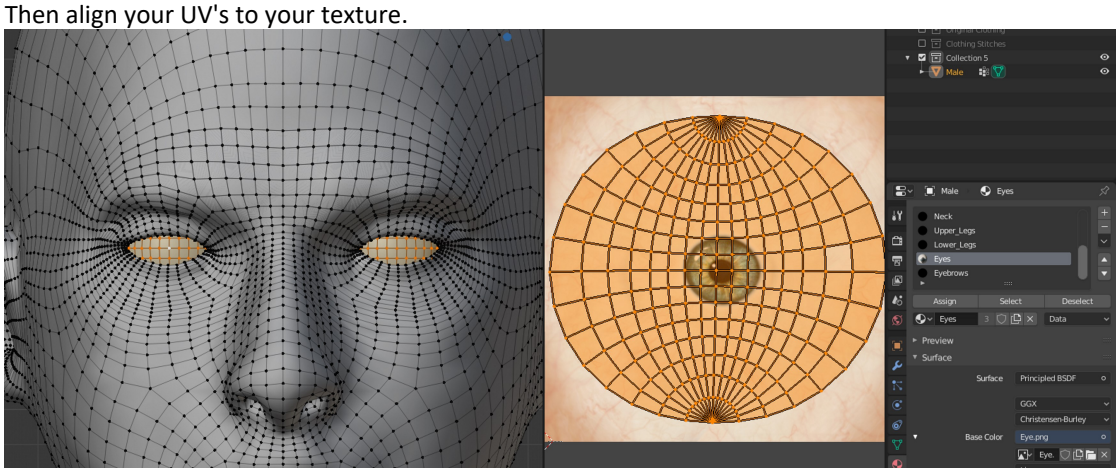


4 different eyebrow textures aligned.

Replaceable Eye Color

In order to easily replace eye colors, and easily adjusting the look of eyes in Unity, eyes should have their own material.

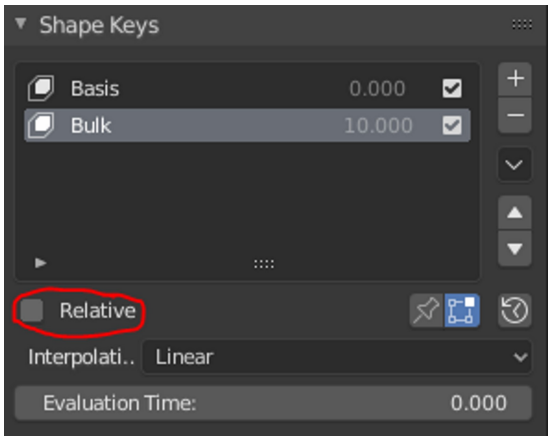
Select all faces on each eye, unwrap them, and assign them to "Eyes" material, same as the beginning of the Eyebrow section. Then align your UV's to your texture.



For eye colors, all we had to do in blender is these steps. Rest should be done in Unity.

Exporting The Character:

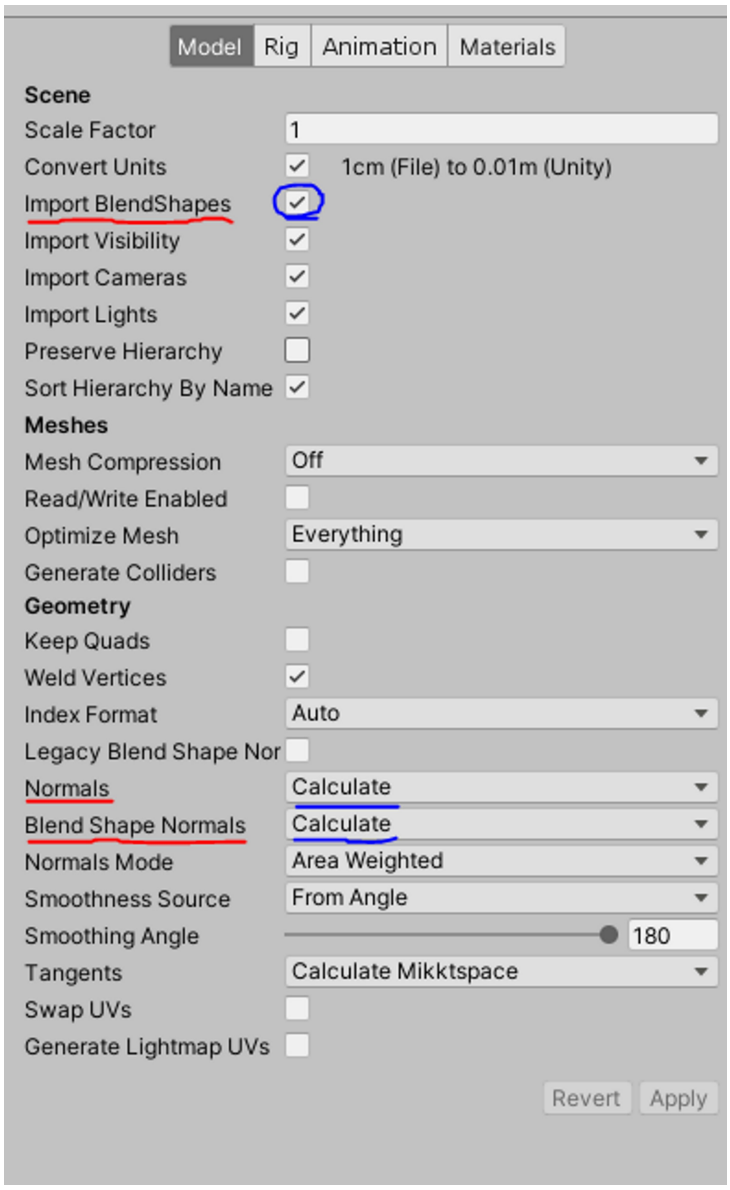
The character is now ready to export to Unity. If the character has skeletal animations, "Relative" toggle under the shape key list should be unchecked, in order to avoid animations overriding the shape key.



The character is now ready to export. For more information about exporting .fbx files for Unity, you can see: [Blender 2.8 Exporting FBXs to Unity 3D (In 60 Seconds!)]

In Unity:

Move your .fbx file to Unity. Click on the file in inspector and change these import settings:

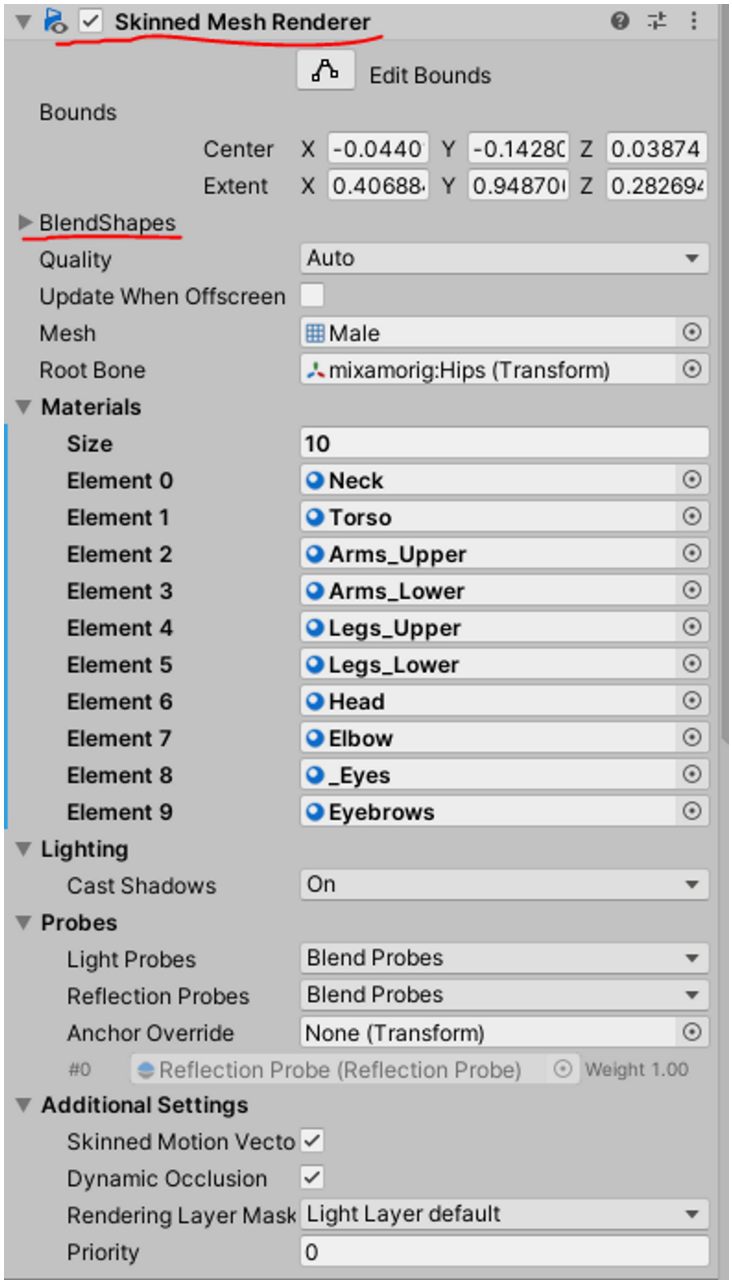


- **Import BlendShapes:** True
- **Normals:** Calculate
- **Blend Shape Normals:** Calculate

The rest is up to you, for high poly character models, setting "Smoothness Source" to "From Angle" and "Smoothing Angle" to 180 makes sure your smoothness and normals are consistent among the mesh.

Now drag your model to the scene, and assign materials.

Now select the gameobject that has the character model. Make sure it has a "Skinned Mesh Renderer" component, and in that component a "Blendshapes" foldout.



If it does, we are good to go. If it doesn't check the exporting step and try again.

Creation Menu

Now we need to create a UI for this job.
To make a simple UI, first switch to 2D mode in Unity.

Create a UI -> Panel and place that panel anywhere you like. Change its name to "Background".

Under "Background" create a new UI -> Scrollbar named "Scrollbar".

Under "Background" create a new UI -> Panel and name it to "Properties".

Under "Properties" create a new empty gameobject and name it to "List".
Under "List" create add "Vertical Layout Group" and "Content Size Fitter" components.
Set "Vertical Layout Group"s settings as:

- **Control Child Size:** (Width: true, Height: false)
- **Use Child Scale:** (Width:false, Height: true)

- **Child Force Expand:** (Width:true, Height: true)

Set "Content Size Fitter"s settings as:

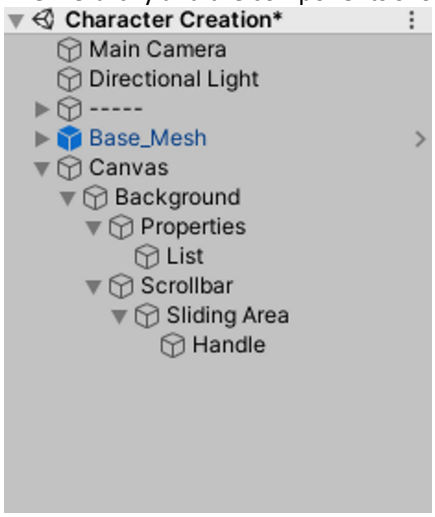
- **Horizontal Fit:** Unconstrained
- **Vertical Fit:** Preferred Size

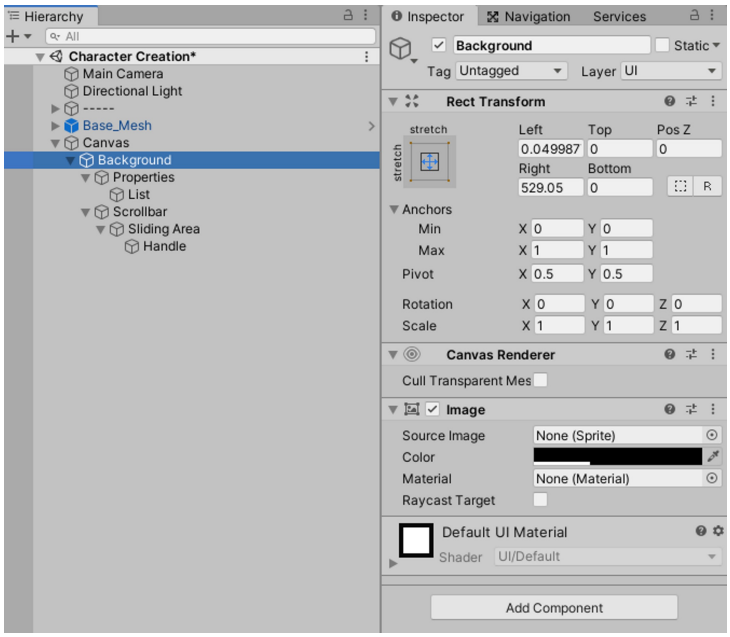
Click "Properties" gameobject and add "Scroll Rect" and "Mask" components.

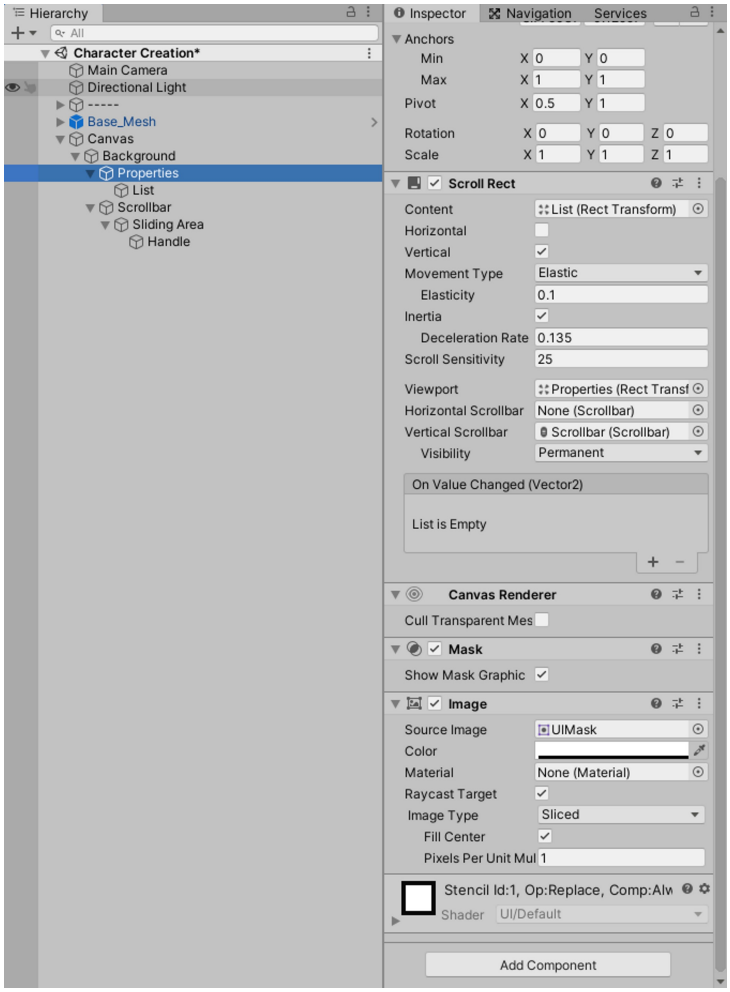
Under "Scroll Rect" component set "Content" to "List", "Viewport" to "Properties", "Vertical Scrollbar" to "Scrollbar".

Under "Scrollbar" gameobject set "Scrollbar" components "Direction" to "Bottom to Top".

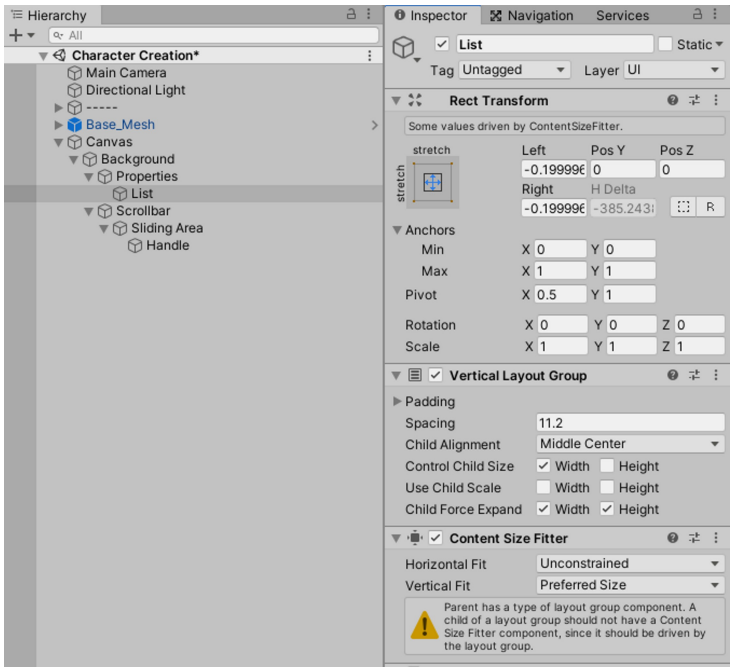
The hierarchy and the components should look like this:



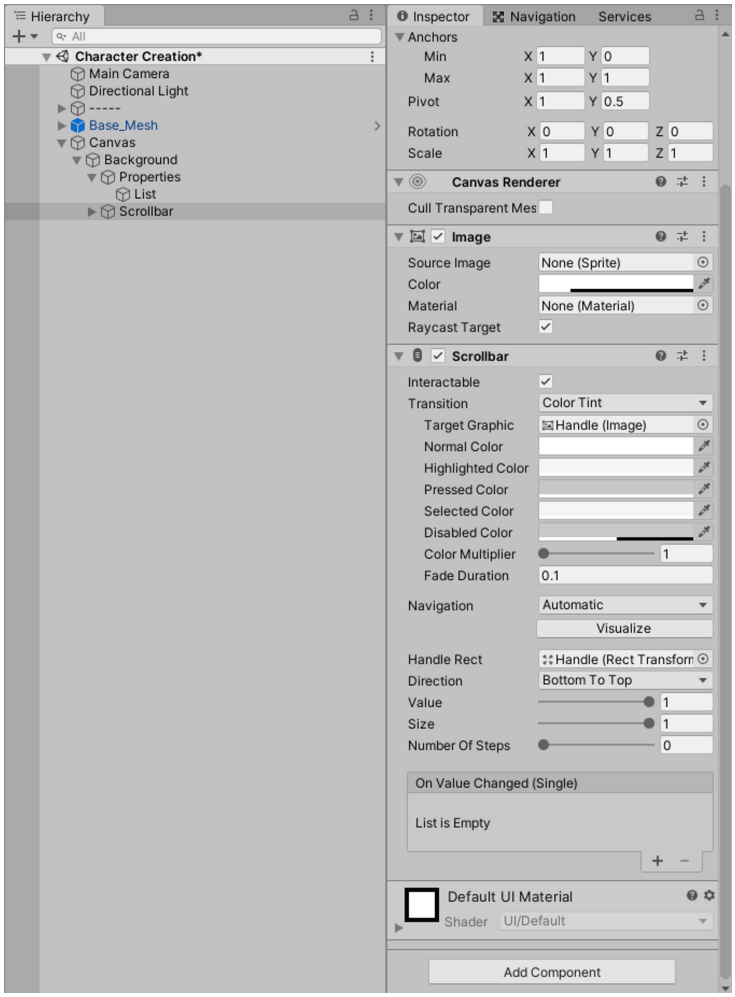



Hierarchy
All
Character Creation*
Main Camera
Directional Light

Base_Mesh
Canvas
Background
Properties
List
Scrollbar
Sliding Area
Handle
Inspector
Navigation
Services
Anchors
Min X 0 Y 0
Max X 1 Y 1
Pivot X 0.5 Y 1
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Scroll Rect
Content List (Rect Transform)
Horizontal
Vertical
Movement Type Elastic
Elasticity 0.1
Inertia
Deceleration Rate 0.135
Scroll Sensitivity 25
Viewport Properties (Rect Transf
Horizontal Scrollbar None (Scrollbar)
Vertical Scrollbar Scrollbar (Scrollbar)
Visibility Permanent
On Value Changed (Vector2)
List is Empty
Canvas Renderer
Cull Transparent Mes
Mask
Show Mask Graphic
Image
Source Image UIMask
Color
Material None (Material)
Raycast Target
Image Type Sliced
Fill Center
Pixels Per Unit Mul 1
Stencil Id:1, Op:Replace, Comp:Alw
Shader UI/Default
Add Component


Hierarchy
All
Character Creation*
Main Camera
Directional Light

Base_Mesh
Canvas
Background
Properties
List
Scrollbar
Sliding Area
Handle
Inspector
Navigation
Services
List
Static
Tag Untagged
Layer UI
Rect Transform
Some values driven by ContentSizeFitter.
stretch
Left
Pos Y
Pos Z
Right
H Delta
Anchors
Min X 0 Y 0
Max X 1 Y 1
Pivot X 0.5 Y 1
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Vertical Layout Group
Padding
Spacing 11.2
Child Alignment Middle Center
Control Child Size Width Height
Use Child Scale Width Height
Child Force Expand Width Height
Content Size Fitter
Horizontal Fit Unconstrained
Vertical Fit Preferred Size
Parent has a type of layout group component. A child of a layout group should not have a Content Size Fitter component, since it should be driven by the layout group.



Now its time to create a slider prefab to instantiate for each of our shape keys.

Shape Key Slider

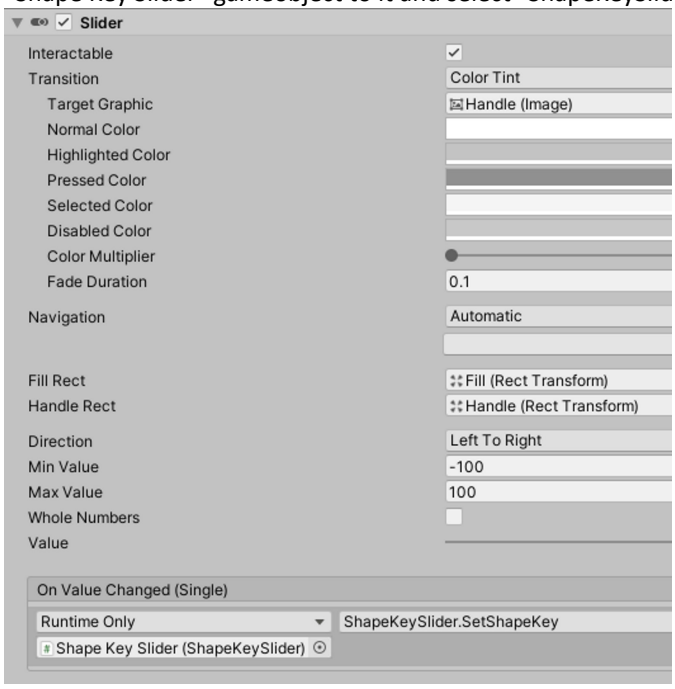
Create a new empty gameobject under canvas name it "Shape Key Slider". And under it;
Create a UI -> Text and name it "Shape Key Name".
Create a UI -> Slider.

Select "Shape Key Slider" and add a new script named "Shape Key Slider".
Open the script and paste this:

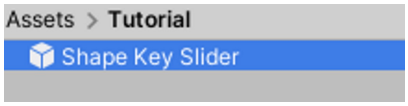
```
using UnityEngine;
using UnityEngine.UI;


public class ShapeKeySlider : MonoBehaviour
{
    private GameObject BaseCharacter;
    private SkinnedMeshRenderer meshToEdit;
    private int shapeID;

    private string _shapeName;

    public string shapeName
```

```
    {
        get
        {
            return _shapeName;
        }
        set
        {
            _shapeName = value;
            GetComponentInChildren<Text>().text = value;
        }
    }


    public void SetupShapeKey(GameObject baseChar,int shapeKeyID,string shapeKeyName)
    {

        BaseCharacter = baseChar;

        shapeID = shapeKeyID;

        meshToEdit = BaseCharacter.GetComponentInChildren<SkinnedMeshRenderer>();
        SetShapeKey(0);
    }

    public void SetShapeKey(float value)
    {
        meshToEdit.SetBlendShapeWeight(shapeID,value);
    }
}
```

Go to "Slider" and inside the "Slider" component create a new action for "On Value Changed" by clicking the "+" button. Drag the "Shape Key Slider" gameobject to it and select "ShapeKeySlider->SetShapeKey".



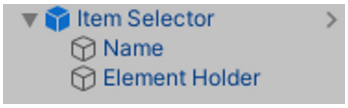
Now save the "Shape Key Slider" as a prefab. And now we can delete it from the scene.



This slider will be able to edit shape keys when enough information is given. To be able to change eye colors and eyebrows in realtime, we should have a UI just like the "Shape Key Slider".

Item Selector

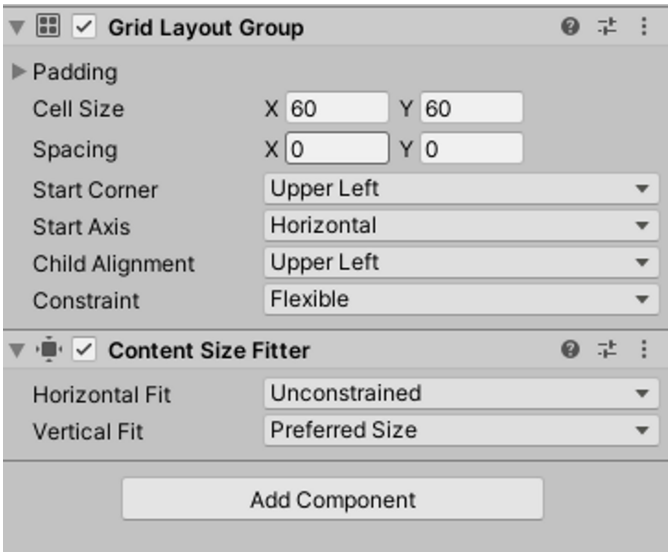
Create an empty gameobject and name it "Item Selector".
Under "Item Selector" create a UI->Text gameobject and name it "Name".
Under "Item Selector" create a new empty gameobject and name it "Element Holder".



Select "Element Holder" and add "Grid Layout Group" and "Content Size Fitter" components.

Settings for:

- **Grid Layout Component:**
  - Padding (0,0,0,0)
  - Cell Size (X: 60, Y:60)
  - Spacing (X:0, Y:0)
  - Start Corner: Upper Left
  - Start Axis: Horizontal
  - Child Alignment: Upper Left
  - Constraint: Flexible
- **Content Size Fitter:**
  - Horizontal Fit: Unconstrained
  - Vertical Fit: Preferred Size



Go to "Element Holder" and add a new script called "ItemSelector".

Copy this to the script:

```
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;
using System.Linq;

public class ItemSelector : MonoBehaviour
{
    [SerializeField] Transform itemHolder;
```

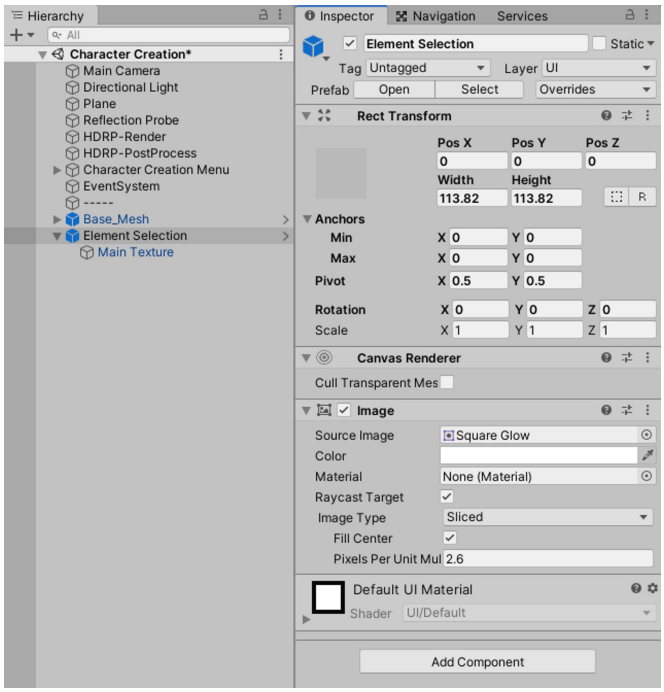
```
    [SerializeField] GameObject imagePrefab;
    public GameObject selection;

    private void Start()
    {
        if (itemHolder == null)
            itemHolder = GetComponentInChildren<HorizontalLayoutGroup>().transform;
    }

    public GameObject AddItem(Texture2D itemTexture, params UnityAction[] onClickActions )
    {
        var holderRect = itemHolder.GetComponent<RectTransform>().rect;

        var newElement = Instantiate(imagePrefab, itemHolder);
        newElement.GetComponent<RectTransform>().sizeDelta = new Vector2(holderRect.height, holderRect.height);
        newElement.GetComponent<Image>().color = Color.black;

        var button = newElement.GetComponentInChildren<Button>();
        button.onClick.AddListener(delegate { Select(newElement); });
        foreach(var action in onClickActions)
        {
            button.onClick.AddListener(action);
        }
        button.GetComponent<Image>().overrideSprite = Sprite.Create(itemTexture, new Rect(Vector2.zero, new
Vector2(itemTexture.width,itemTexture.height)), Vector2.zero);
        GetComponent<RectTransform>
().SetSizeWithCurrentAnchors(RectTransform.Axis.Vertical,GetComponentsInChildren<RectTransform>().Where(x=>
x.transform.parent == transform).Sum(x=> x.rect.size.y));

        return newElement;
    }

    public void Select(GameObject sel)
    {
        if (selection != null)
        {
            selection.GetComponent<Image>().color = Color.black;
        }

        selection = sel;

        selection.GetComponent<Image>().color = Color.white;
        Debug.Log("Click..");
    }
}
```

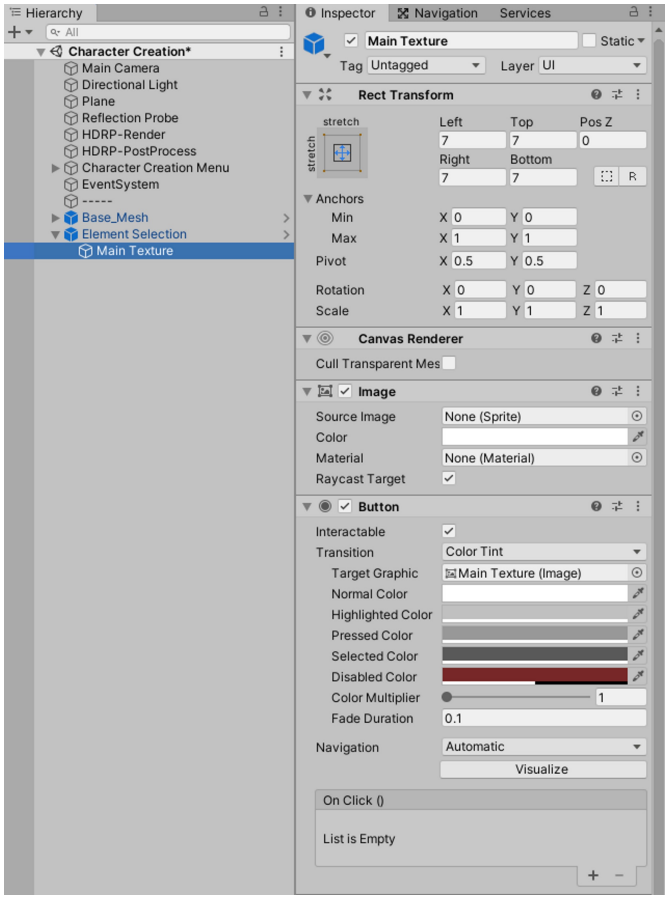
Save and exit.

Now its time to make an element viewer to hold our selection objects.

Element Selection

This part is up to your taste, but in order to make things consistent you can take this prefab as an example.

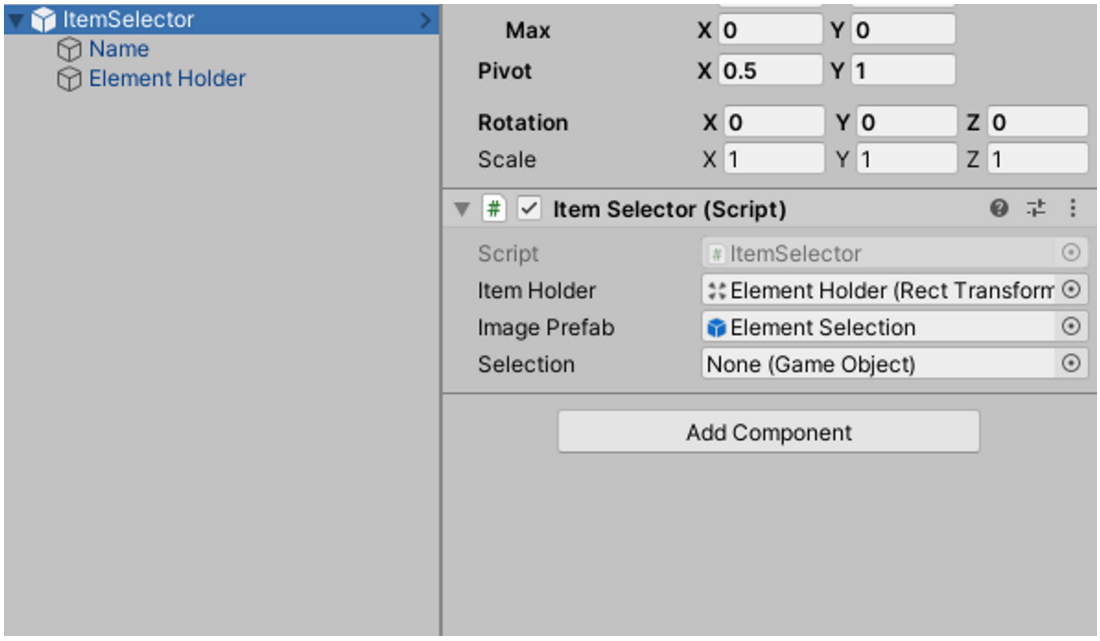




The main idea here is, we need a parent gameobject with an "Image" component and a child of it which has a "Image" and "Button" Component. Save your prefab and you can delete "Element Selection" from scene.

Now we can populate "ItemSelector" script under the gameobject "ItemSelector":

- **Item Holder: "**Element Holder" gameobject under the "ItemSelector".
- **Image Prefab:** "Element Selection" prefab we just created.
- **Selection:** This can be left empty, as it will be dynamic in runtime.

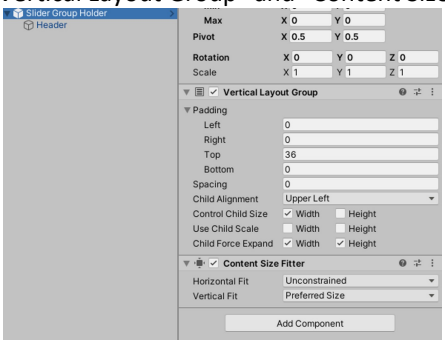


Item selector is done, now its time to make a seperator to organize the menu a little.

Slider Group Holder

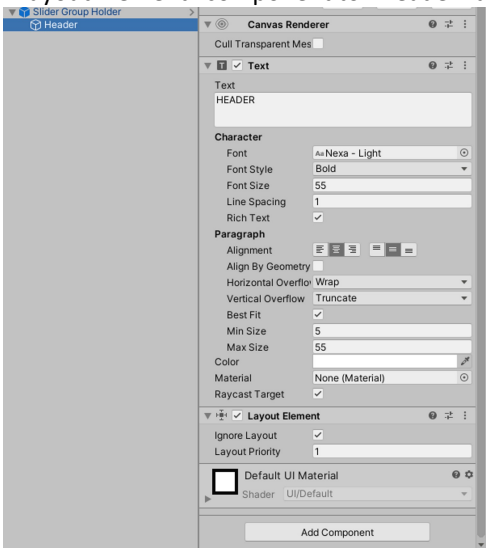
Create a new empty gameobject and name it "Slider Group Holder".

Add "Vertical Layout Group" and "Content Size Fitter" components to "Slider Group Holder"



Create a new UI-> Text under "Slider Group Holder" and name it "Header"

Add a "Layout Element" component to "Header" and set it to "Ignore Layout"



Now its time to create a data object that holds all the information we need about the character creation.

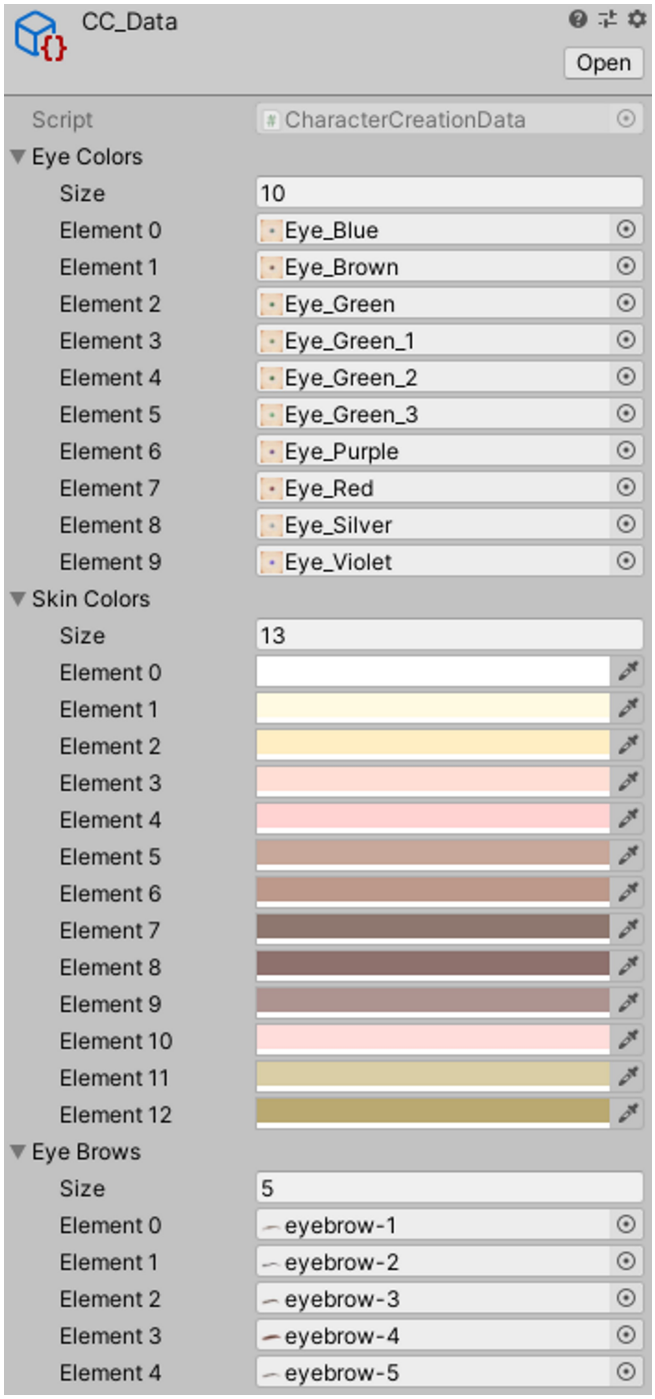
Character Creation Data

In assets folder create a new script, name it "CharacterCreationData".

Paste this into it:

```
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(fileName = "Character Creation Data", menuName = "Character Creation Data")]
public class CharacterCreationData : ScriptableObject
{
    public List<Texture2D> EyeColors = new List<Texture2D>();
    public List<Color> SkinColors = new List<Color>();
    public List<Texture2D> EyeBrows = new List<Texture2D>();
}
```

Right click in the project window and Create->Character Creation Data. Name it "CC_Data".
Now you can populate it with your own textures and colors. Here is a reference:



The skin color codes used in this preset are:

FFFAE1
FFEEC4
FFDFD5
FFD3D2
C8A89A
BC998B
8E776E
8E706D
AE9491
AE9491
FFDDDA
D9CEA5
BAA971

Connecting the dots - Property List

All the small parts are done and its time to move on to the connecting all the dots!

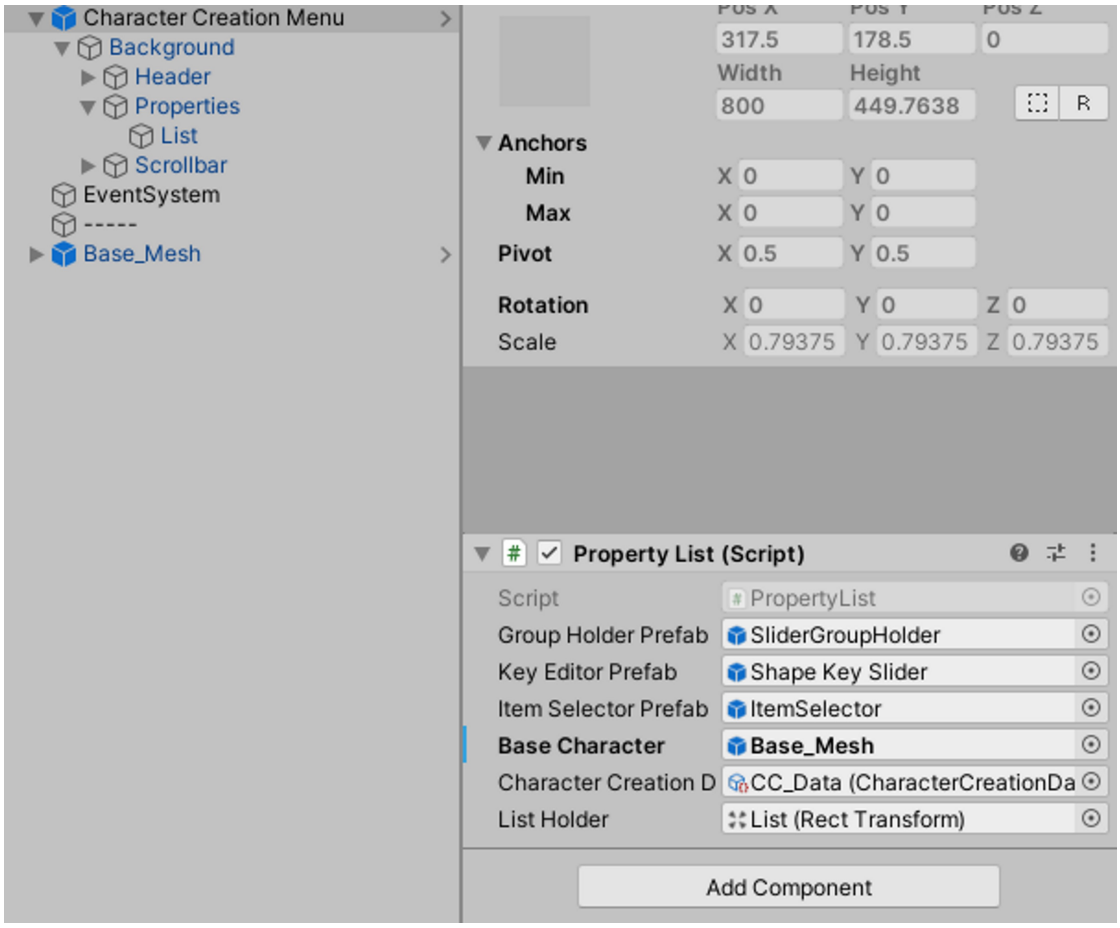
Back to the menu, click on the "Canvas" gameobject that holds all our menus and change its name to "Character Creation Menu". Now add a script to it, named "PropertyList".

Paste this into the it :

```
public class PropertyList : MonoBehaviour
{
    [SerializeField] private GameObject GroupHolderPrefab;
    [SerializeField] private GameObject KeyEditorPrefab;
    [SerializeField] private GameObject ItemSelectorPrefab;
    [SerializeField] private GameObject BaseCharacter;
    [SerializeField] private CharacterCreationData CharacterCreationData;

    [SerializeField] private Transform ListHolder;

    void Start()
    {
        if (CharacterCreationData.SkinColors.Count > 0)
        {
            var SkinSelector = Instantiate(ItemSelectorPrefab, ListHolder).GetComponent<ItemSelector>();
            SkinSelector.name = "Skin Selector";
            SkinSelector.GetComponentInChildren<Text>().text = "Skin Tone";

            foreach (var skinColor in CharacterCreationData.SkinColors)
            {
                var newTex = new Texture2D(32, 32);

                //set color of the texture
                var colors = newTex.GetPixels();
                for (int i = 0; i < colors.Length; i++)
                {
                    colors[i] = skinColor;
                }
                newTex.SetPixels(colors);
                var newObj = SkinSelector.AddItem(newTex, delegate { SetSkinTone(skinColor); });
            }
        }

        if(CharacterCreationData.EyeColors.Count > 0)
        {
            var EyeSelector = Instantiate(ItemSelectorPrefab, ListHolder).GetComponent<ItemSelector>();
            EyeSelector.name = "Eye Selector";
            EyeSelector.GetComponentInChildren<Text>().text = "Eye Color";

            foreach (var eyeColor in CharacterCreationData.EyeColors)
            {
                var newObj = EyeSelector.AddItem(eyeColor, delegate { SetEyeColor(eyeColor); });
            }
        }

        if (CharacterCreationData.EyeBrows.Count > 0)
        {
            var EyebrowSelector = Instantiate(ItemSelectorPrefab, ListHolder).GetComponent<ItemSelector>();
            EyebrowSelector.name = "Eyebrow Selector";
            EyebrowSelector.GetComponentInChildren<Text>().text = "Eyebrow";
```

```
            foreach (var eyebrow in CharacterCreationData.EyeBrows)
            {
                var newObj = EyebrowSelector.AddItem(eyebrow, delegate { SetEyebrows(eyebrow); });
            }
        }

        var mesh = BaseCharacter.GetComponentInChildren<SkinnedMeshRenderer>().sharedMesh;
        var keyCount = mesh.blendShapeCount;

        List<GameObject> parents = new List<GameObject>();
        GameObject activeParent = null;

        for (int i = 0; i < keyCount; i++)
        {
            string blendshapeName = mesh.GetBlendShapeName(i);
            string header = blendshapeName.Split('_')[0];
            if(parents.Where(x=> x.name == header).Count() == 0)
            {
                GameObject newgo = Instantiate(GroupHolderPrefab, ListHolder);
                newgo.name = header;
                newgo.GetComponentInChildren<Text>().text = header;
                parents.Add(newgo);
            }
            activeParent = parents.Where(x => x.name == header).ElementAt(0);
            var wordList = blendshapeName.Split('_').ToList();
            wordList.RemoveAt(0);
            blendshapeName = string.Join("_", wordList);

            var newObj = Instantiate(KeyEditorPrefab, activeParent.transform).GetComponent<ShapeKeySlider>();
            newObj.SetupShapeKey(BaseCharacter,i, blendshapeName);
            newObj.propertyType = ShapeKeySlider.PropertyType.ShapeKey;
        }

    }

    void SetEyeColor(Texture2D eyeColor)
    {
        BaseCharacter.GetComponent<CharacterMaterial>().Eyes.SetTexture("_BaseColorMap", eyeColor);
    }

    void SetSkinTone(Color skinTone)
    {
        foreach(var mat in BaseCharacter.GetComponent<CharacterMaterial>().bodyMats)
        {
            mat.SetColor("_BaseColor", skinTone);
        }
    }

    void SetEyebrows(Texture2D eyebrows)
    {
        BaseCharacter.GetComponent<CharacterMaterial>().Eyebrows.SetTexture("_BaseColorMap", eyebrows);
    }
}
```

It's time to assign our values in the inspector:



These values are:

- **Group Holder Prefab:** This is the "Slider Group Holder" prefab we have created.
- **Key Editor Prefab:** This is the "Shape Key Slider" prefab we have created.
- **Item Selector Prefab:** This is the "Item Selector" prefab we have created.
- **Base Character:** This is your character model in the scene.
- **Character Creation Data:** This is the "CC_Data" asset we have created.
- **List Holder:** This is the gameobject "Character Creation Menu" -> "Properties" -> "List"

That’s it! Now we can adjust our shape keys in play mode, realtime! And from now on, you can add new skin colors, new eye colors, new eyebrows with just editing CC_Data asset! And if you add more shape keys to your character, it will automatically appear in the creation menu!

Thanks for reading.

-Civelek Babacık