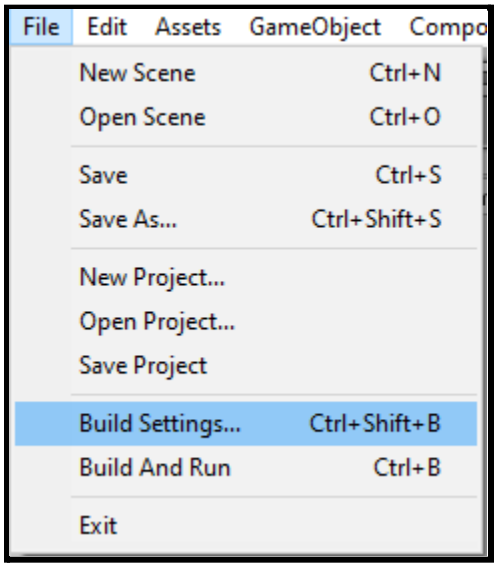
Welcome

Welcome to the **Character Customization** asset for Unity! **(Version 1.4)**
The purpose of this asset is to provide you with the ability to change the look of characters in your games similar to what you would find in role playing games, battle royale games, dress up games etc.
This asset lets you take compatible characters from the assets store that have separate **SkinnedMeshRenderers** and mix and match their parts for your prototypes until you are ready to use custom characters. And it will work with your custom characters that have separate **SkinnedMeshRenderers**.
Even if you are not a programmer you can still use this system for your game. It is advised that you have some prior familiarity with the unity game engine. If you have prior coding experience you should be able to extend the system in some of the recommended ways depending on any custom needs you may have for your game.
The asset works with unity 2018.4.0f1 LTS or any newer version unless something drastically changes in the engine's subsequent updates.
Please note that this asset is constantly evolving. The latest documentation can be found on the website.

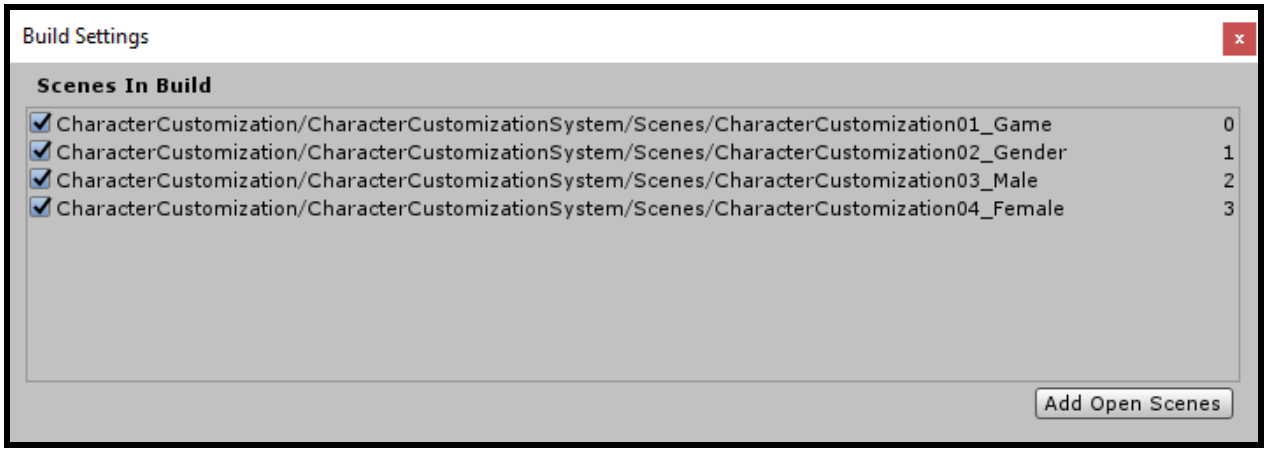
COMFORTGAMES.CA

Getting Started

Once you import the package you need to add the demo scenes into the build settings.

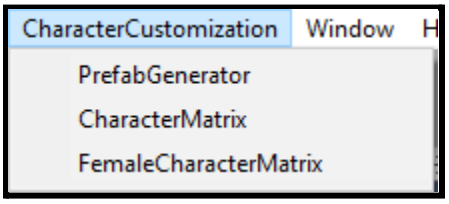


1) "CharacterCustomization01_Game"
2) "CharacterCustomization02_Gender"
3) "CharacterCustomization03_Male"
4) "CharacterCustomization04_Female"

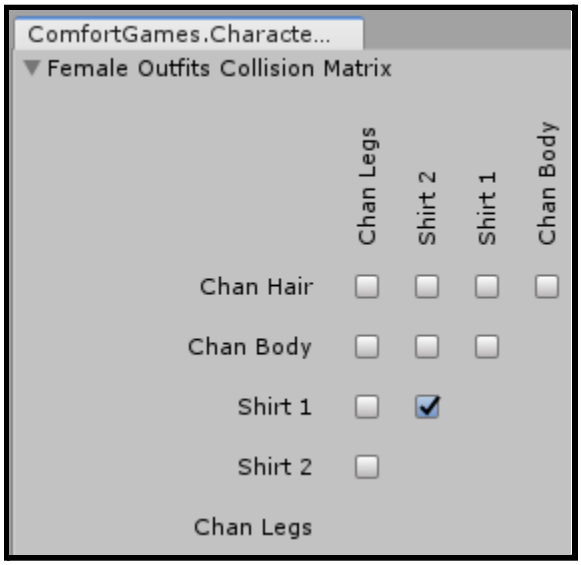


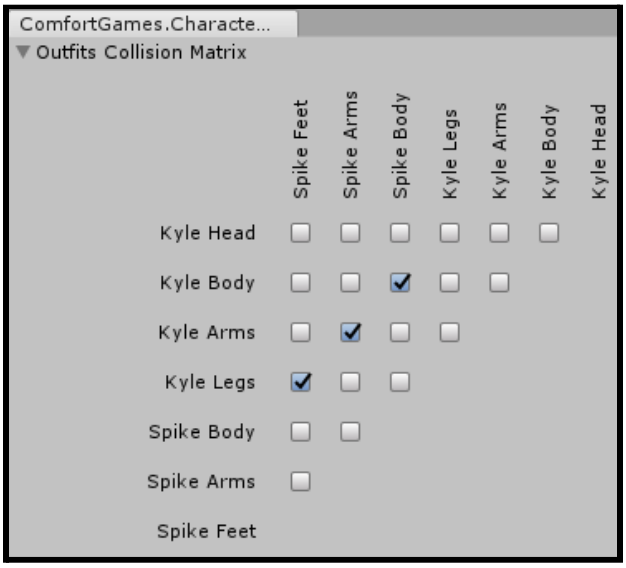
Open the unity scene "CharacterCustomization01_Game". This scene represents your game.
Hit the Play button in the unity editor.
When the game starts you will see a "Customize Character" button. This button represents taking you to the character customization portion of your game.
If you have previously done customizations and any saving has occurred it will take you to your last selected gender. If it is a new setup then it will take you to the gender selection scene "CharacterCustomization02_Gender".
Press the "Male" button and this will take you to the scene "CharacterCustomization03_Male". (Behind the scenes your gender selection has been saved automatically).
You will see a partial character with some missing parts. This is intentional to demonstrate the different functionality.
You can select a **Category** by pressing “Head”, "Body", "Arms", "Legs".
With the "Head" **Category** Selected **Equip** the head "Kyle Head" by pressing "Change".
Now the character doesn't have any holes.
Hit the "Save" button so that you retain your changes the next time you load.
Press the "Done" button, it will take you back to the scene "CharacterCustomization01_Game".
This time the character will be there because you saved a customization! And restarting unity will do the same and still show you your character.
Let's go back to "character customization".
Now let's go to the "Body" **Category**.
You will see “Kyle Body” **Equipped** already. This is because it is configured as the **Category Default** in the body **Category**. If nothing is **Equipped** the **Category Default** is automatically **Equipped**. Also there is no “change” option for the “Kyle Body” because a **Category Default** cannot be **Unequipped** unless you **Equip** something else in the category first. We want this kind of functionality to make sure our character is never naked or worse never has holes.

Usually the **Category Defaults** will be parts of the character skin and underwear or your default outfit parts. For example you might have a default pair of shoes if you didn't want your character to be able to be barefoot.
See "Category Default" for more info.
Let's go to "Spike Body" and press change. Notice that the "Kyle Body" was automatically **Unequipped** when the "Spike Body" was **Equipped**. This is because "Kyle Body" and "Spike Body" are configured to **Collide** in the **Collision Matrix**.**Outfits** that **Collide** cannot be **Equipped** at the same time. For example if your character is wearing jeans and you want to put some shorts on you need to remove the jeans first.
This is important to make sure your character **Outfits** do not clip.
You can change the **Collisions** by going into The Editor Window "CharacterCustomization" and "CharacterMatrix" or "FemaleCharacterMatrix".



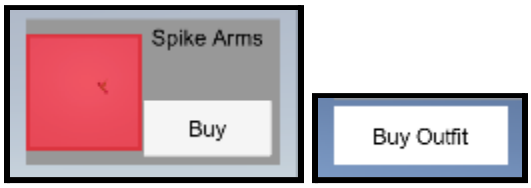
This opens an Editor Window with an Outfit Collision Matrix.
It is similar to how Unity displays and edits their Layer Collisions.



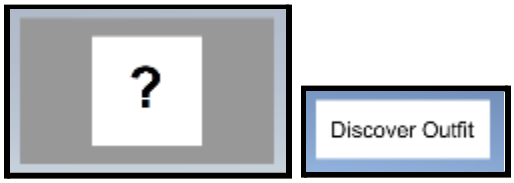


You can read it as Outfit at Row Y will Collide with Outfit at Column X if it is checked off.
An outfit automatically collides with itself. That is why if you try to Equip the same outfit you already have **Equipped** it will be **Unequipped**.
If you modify the matrix checkmarks click out of the collision matrix editor window and then select the window again to save your matrix changes. See "Collision Matrix" for more info.

Now back in the game window select the "Arms" **Category**.
"Spike Arms" is **Visible** but we can't **Equip** it because we do not **Own** it.



Press the "Buy Outfit" button and you will notice the "Buy" button on the outfit turn to "Change".
We now **Own** the outfit and can **Equip** it by pressing the "Change" button like the other **Outfits**.
You can extend this functionality and make additional restrictions. For example by charging your players some virtual currency as they press the "Buy" button but in our example it is free!
Notice how changing "Spike Arms" will **Equip** it but **Unequip** "Kyle Arms" and vice versa.
Once again these outfits are configured to **Collide** with each other.
Now let's go to the "Legs" **Category**.
You will also see that there is an **Invisible** Outfit that cannot be added ("Outfit6_Legs02" also known as “Spike Feet”).
This is because it needs to be **Discovered**.



Press the "Discover Outfit" button.
Now we can see ""“Spike Feet". Press the "Change" button on ""“Spike Feet".
We have now **Equipped** ""“Spike Feet" and automatically **Unequipped** "Kyle Legs".
They are also configured to Collide.
Press the "Save" button.
Press the "Done" button.
Now our character looks cooler with spikes that we **Equipped** using **Buying** and **Discovering**.
We can configure if an outfit needs to be **Bought** or **Discovered** in the Scriptable Object of that Outfit. More is explained about that later but you can see “Outfit Scriptable Object” for more details.
Next from our game scene go to "Select Gender" and "Female".
(Behind the scenes your gender selection has been saved automatically).
In our example Unity Chan is not a very good model because her face Skinned Mesh Renderers and Mesh Renderers are in her bones hierarchy, not separate. For our example her face is always on. But this demonstrates an alternative way to have non removable parts. To do this keep your Face or any other permanent part inside your **Character Base**

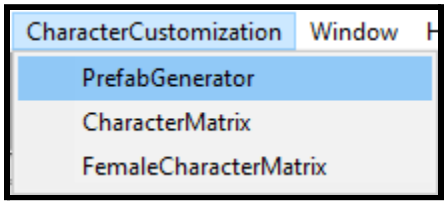
“CharacterBuilder_Male” or “CharacterBuilder_Female”. For more information see “Character Base”.
Next press "Change" on "Chan Hair” to **Equip** it and go to the "Body" Category and **Equip** "Chan Body" and "Shirt 1" with their "Change" buttons.
“Shirt 2” is configured to **Collide** with “Shirt 1”.
Go to the “Legs” **Category** and **Equip** “Chan Legs”.
Press the "Save" button.
Press the "Done" button.
We now have a female character in the game!
In this setup we can alway change the gender at any time using the “Select Gender” button from in our game scene. If you only want the option to select the gender at the start of the game you can hide the “Select Gender” button in the game and in the male scene and in the female scene.
If you make changes that you don't like and you haven't **Saved** them yet you can press the "Load" button to **Load** your last saved version of your character and this will discard your current unsaved changes. Note that your last selected gender is not discarded since it is automatically saved at the time of your selection.
If you want to **Delete** your character customization you can press the "Delete Save" inside your male or female scene. The next time you go into the game or restart unity you will need to do a new "Character Customization".
This functionality may be desired if your players want to start a brand new version of your game after beating it.

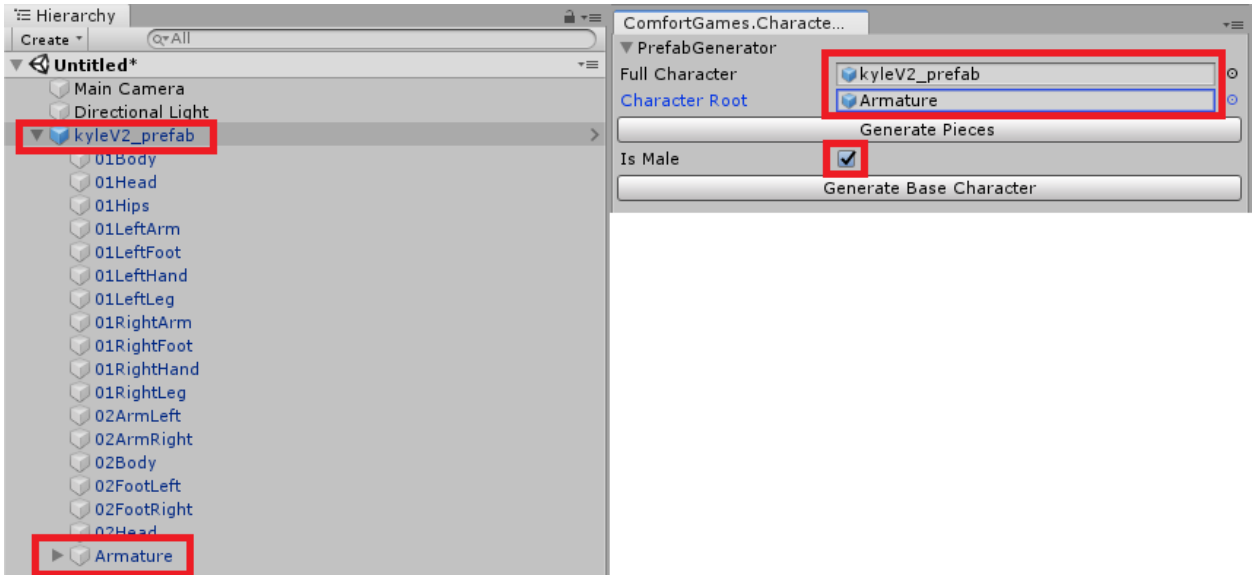
Parts of the System

“Character Base”

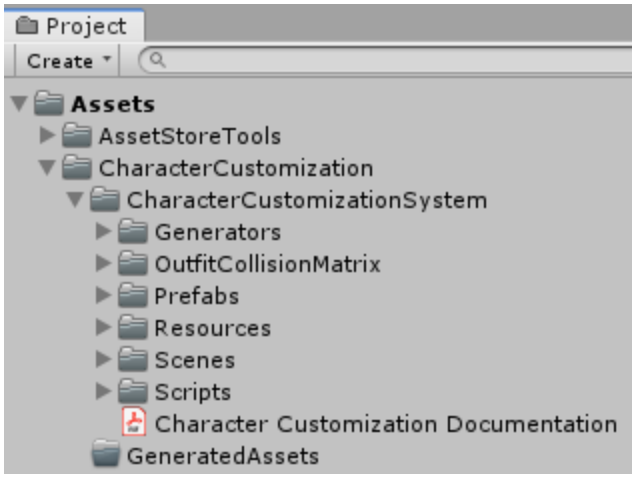
A **Character** is what appears in your game and what you can customize **Outfits** for.
This example features two character prefabs "CharacterBuilder_Male" and "CharacterBuilder_Female".
To create a new **Character** let’s open a new scene so that we don’t have confusion with anything else.
Drag a **Model** into your scene e.g "kyleV2_prefab". You can also follow these steps with compatible models from the Asset Store or Custom ones you or your team made.
Note that in this example every outfit you make will need to be something that is a **SkinnedMeshRenderer** of Kyle. This depends on your 3D model so the same applies for assets from the asset store or what your 3D artist makes.
An asset from the asset store will only work with this system if the model has multiple **SkinnedMeshRenderers** for different parts. Luckily the asset store has a lot of these.

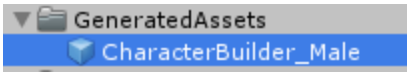
The "Kyle" **Model** from unity technologies was not compatible but was edited with blender to create multiple skinned mesh renderers for this demonstration.
We can generate a Character Base with our tool. Go to CharacterCustomization -> PrefabGenerator.





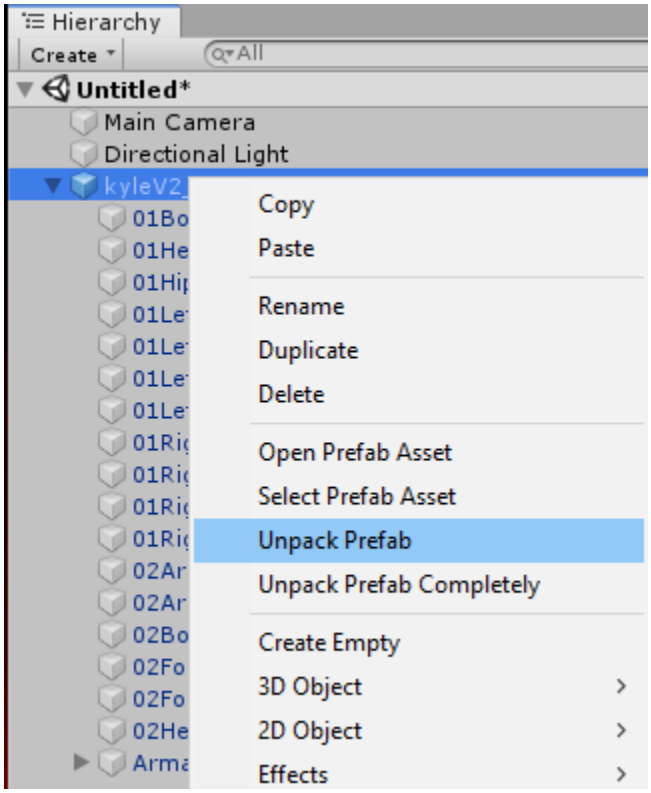
Drag the character model into “Full Character” and the Root of the Bones into Character Root. The Root may be called differently for different characters you buy or that the D artist on your team makes.
You need to toggle IsMale to be On or Off depending on the gender of the character base you are making. Press “Generate Base Character”.



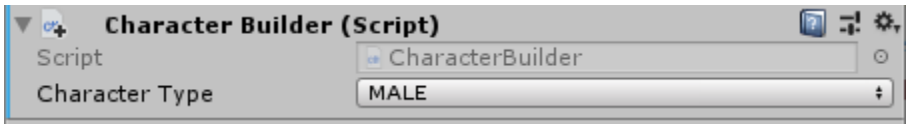


The Base will be Created in your “GeneratedAssets” folder in your projects.
It is important that you do not move that folder or rename it!
You can then move your generated **Character Base** Prefab into another folder of your choice.
If you need to generate into another folder you will need to modify the path in CharacterCustomizationAssetManager.

Sometimes the tool doesn’t create the base for a custom character correctly or you may want more control over how your base is created so you can create the **Character Base** manually.
Right click and unpack the "kyleV2_prefab" prefab.

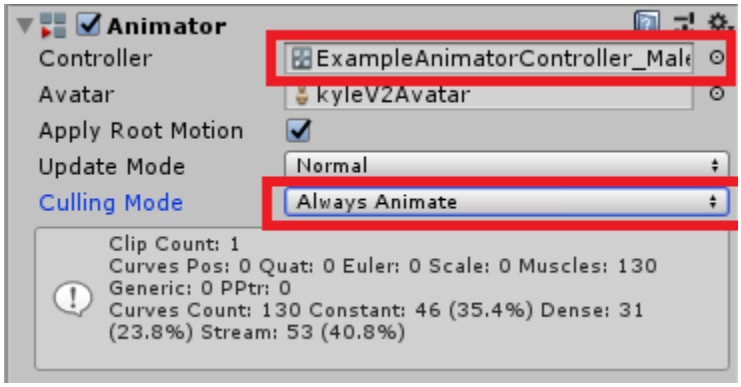


Rename it to whatever you want your character base to be called e.g. “CharacterBuilder_Male”.
Add the "CharacterBuiler" component.
Set the Character Type to MALE or FEMALE depending on the type of base you are making.



You can see that there is already an animator component, you can keep it and reference whatever animator controller you want.
This example has “ExampleAnimatorController_Female” and “ExampleAnimatorController_Male” with just an idle animation but you can extend it.

You will want to set the animator component "Culling Mode" to "Always Animate" so that your character can start with no Mesh Renderer, and have them get added and still animate the way we want.

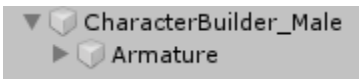


You can also add your own character controllers, rigid bodies, colliders and any other unique components you might have.

Since this character is both in your game and in the Character Customization you may want to prevent the character from moving while they are being customized or you can use two different prefabs (one for your game and one in the male/female scenes). These **Character Base** prefabs are referenced by our CharacterSpawning scripts.

For our **Character Base** we don't want to have any initial SkinnedMeshRenderers since this is driven by our **Outfits** and what we **Equip**.

Delete everything that isn't our bones. So only keep the root "Armature".

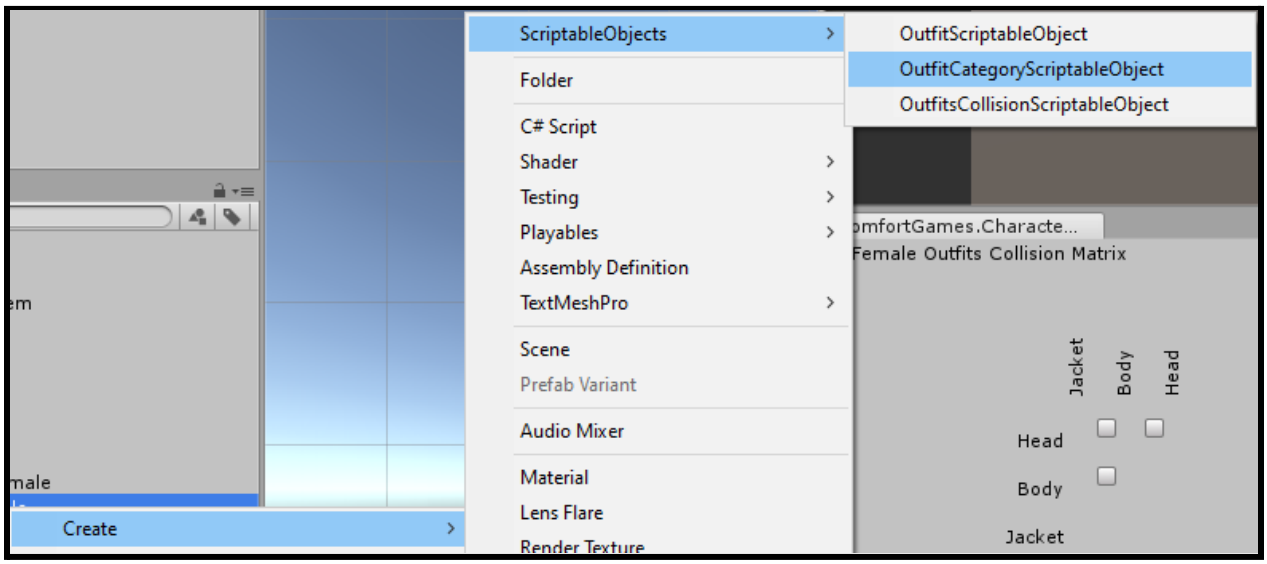


It is very important that you keep "Armature" and everything under it.

This is because “Armature" is the bone structure of your character. Depending on what models you purchase, create or use it might be called slightly differently and have more or less children but it conforms to a humanoid skeleton.
With Character Customization V1.4 we now have a Root variable. It is assigned automatically when you generate the base. If it is not you can manually drag the root reference into it.
Other characters may have some “Skinned Mesh Renderers” or “Mesh Renderers” that are in the bones. This is typically bad. We can hide them by disabling those Skinned Mesh Renderer or Mesh Renderer Components. Not the GameObjects. WIth Character Customization V1.4 we can now enable or disable renderers in the bones at runtime. For example Unity Chan’s face.
You can drag this character into the Project Window to create a prefab for it.
This character needs to be referenced by the "CharacterSpawning" in 3 scenes. "CharacterCustomization01_Game" and "CharacterCustomization03_Male" and "CharacterCustomization04_Female" since they have CharacterSpawning.

“Category Scriptable Object”

A **Category** is a way to group **Outfits** together. For example we might want a "Shoes" Category. Then every pair of shoes/boots/sandals can all show in the same view and you can pick between them when deciding what to **Equip**.
To create a new **Category** we can right click in the hierarchy and do "Create" -> "ScriptableObject" -> "OutfitCategoryScriptableObject".



This scriptable object needs to be placed inside the "OutfitsCategoryData_Male" or "OutfitsCategoryData_Female" folders in order for the system to recognize it.
With this scriptable object selected we can modify the values in the inspector.
Specify a Name which will be what you see in the Category View.

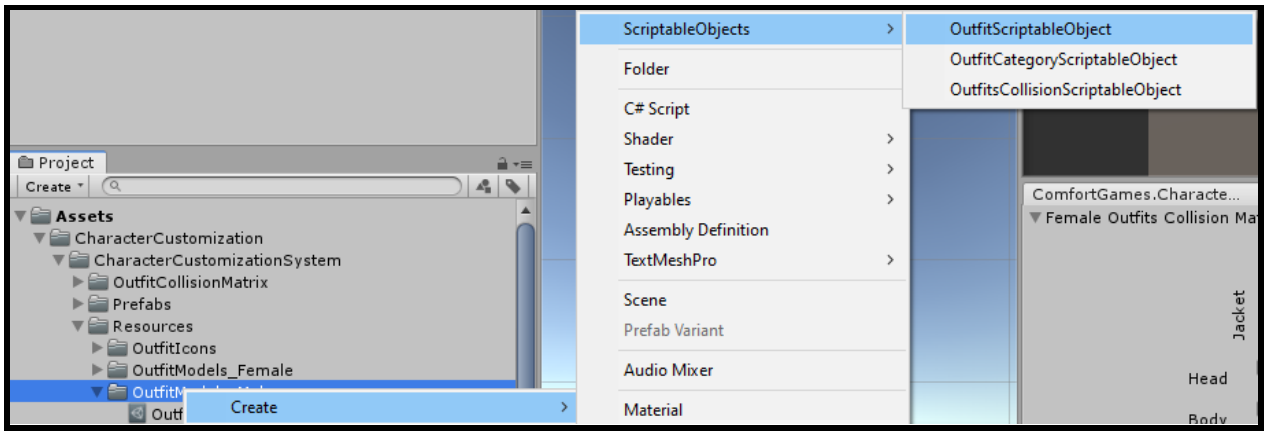
The Sorting Order determines which **Category** appears above or below another one. The higher the sorting order the higher up the **Category** will be.
IsInvisible is optional and used to prevent the **Category** from being seen.
If you wanted you could extend the system and set up a way to discover an entire **Category** similar to how you discover an **Outfit**. Or just keep the category invisible while you are building it until you are ready to show it to your players.
The DefaultOutfitScriptableObject is optional and determines what **Outfit** is equipped if nothing in the category is **Equipped**. See "Category Default" for details.
Version 1.2 Introduced the option to zoom into your character when you select a **Category**.
TargetBoneName is an optional string matching the name of a bone in your **Character Base**. If you have it set then the CharacterCustomizationCameraView in your male or female scene will position itself at the X,Y position of the bone.
CategoryZoom is an optional float that sets the FieldOfView of the camera when the **Category** is selected (if there is a TargetBoneName).

“Category Default”

For a **Category** such as Shoes we might want a default **Outfit** to be bare feet.
To specify a **Category Default** select your **Category** ScriptableObject and drag a OutfitScriptableObject into the "DefaultOutfitScriptableObject” in your inspector. See **Outfit Scriptable Object** for details after you finish reading this section.
Note that **Category Defaults** are optional and not every Category needs to have one.
If a Category Default is set then when there are no **Equipped Outfits** in a **Category** the Default **Outfit** will automatically be **Equipped**.
And you will not be able to remove that **Outfit** unless you **Equip** a different **Outfit** in that **Category** first.
As an example if we have default bare feet **Equipped** then unless you **Equip** some shoes the skin of the feet cannot be **Unequipped** or your character will have invisible feet.
You might have a **Category** like accessories with different glasses where your **Character** can have the option of not having any glasses at all, therefore no **Category Default**.

“Outfit Scriptable Object”

An **Outfit** is an object that represents one or more pieces of clothing. A T-Shirt is a piece of clothing that can be an **Outfit**. But a T-Shirt and Shorts can be two pieces of clothing that can also be an **Outfit**.
In the case of multiple pieces that are part of the same Outfit this is often referred to as a **Set**.
To create a new **Outfit** in the hierarchy right click "Create" -> "ScriptableObject" -> "OutfitScriptableObject".



This new scriptable object must be placed in the folder "OutfitModels_Male" or "OutfitModels_Female".

The Name will be what you see when you look through your **Outfits** in the view.

The OutfitIcon is a spite that you will see when you look through your outfits in the view. It needs to be set to "TextureType" Sprite(2D and UI). You can place these sprites anywhere in your project. We can generate Outfit Icons with a tool but it is best to have an artist on your team create them.

If you turn on "Is Invisible" then the **Outfit** will appear as a question mark and you will need to **Discover** the **Outfit** before it can be **Equipped**.

If you turn on "Is Locked" then your player will not own the **Outfit** and they will need to **Buy** it before the **Outfit** can be equipped.

If you turn on both "Is Invisible" and "Is Locked" you will first need to **Discover** the **Outfit** and then **Buy** it.

Sorting order determines which **Outfit** appears above or below the others in the same **Category** in the view. The higher the sorting order the higher up the **Outfit** will be.

The OutfitCategoryScriptableObject determines which **Category** this **Outfit** belongs to and also determines which **Category** the **Outfit** is displayed in.

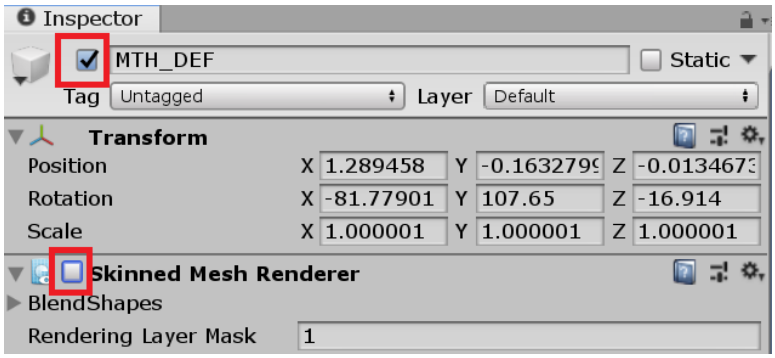
Every **Outfit** must be part of a **Category**!

The “Outfit Pieces” is a list of one or more prefabs that represent the actual 3d art of your **Outfit**. If the list has no pieces then no pieces are instantiated or destroyed. The list count must equal the number of pieces and the list cannot have empty references.

The **Outfit Piece** prefab needs to have a complete matching skeleton of your **Character** for reference and at least one object with a Skinned Mesh Renderer which is what you will be seeing.

See **Outfit Piece** for details.

The “Renderer Object Names” is a list of names of objects that are in the Root and have a Renderer (Skinned Mesh Renderer or Mesh Renderer). If no names are specified then this is ignored. Renderers inside the Root are not created or destroyed. Instead we need to enable or disable the Renderer Component. The object itself must be active so that it can be found by name. But when creating a Character Base that base should have the Renderer components inside the root disabled (this is done manually after generating the base). If they are enabled then they will be visible by default. For more information look at the example OutfitScriptableObject for Unity chan Outfit5_Face01.
It references “BLW_DEF”, “eye_base_old”, “EYE_DEF”, “EL_DEF”, “eye_L_old”, “eye_R_old”, “head_back”, “MTH_DEF”.
In “CharacterBuilder_Female”, those objects are active in her root “Character1_Reference” but their renderers are disabled and at run time they get enabled/disabled to show or hide her face.

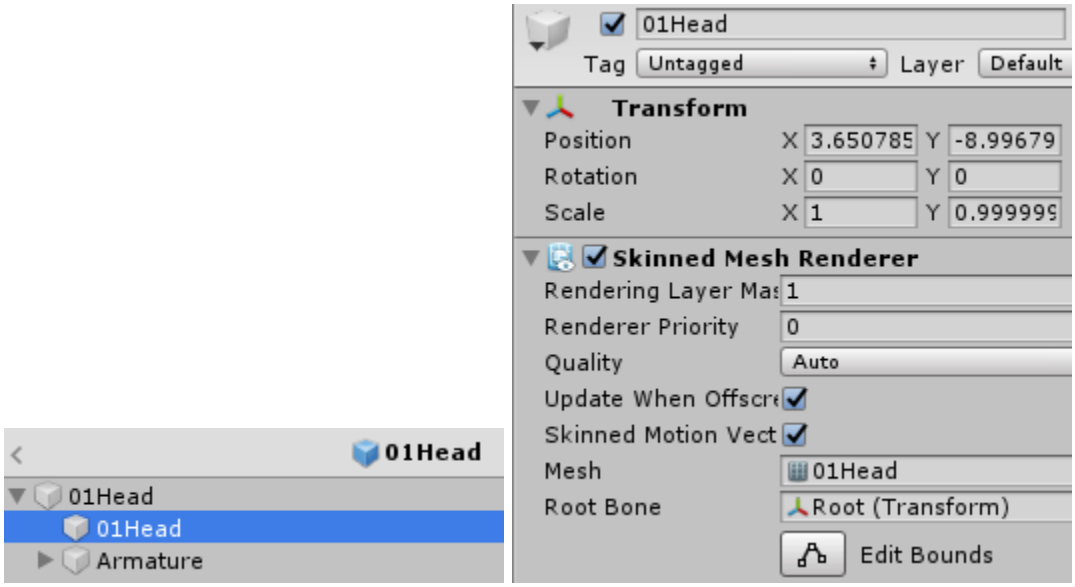


The “checkOutsideRoot” is off by default so we only look for renderers inside the root with names specified in “Renderer Object Names”. But if we want to search outside the root too we can turn this on and search for everything under Character Builder. This is only needed if you wanted to have all your renderers enabling/disabling instead of instantiating/destroying Outfit Pieces.

“Outfit Piece”

An **Outfit Piece** is a 3D model that corresponds with your **Outfit**. E.g. your T-Shirt.
To create an **Outfit Piece** from an asset in the asset store that asset must be compatible. To be compatible the **Model** needs to have multiple objects that are each a different Skinned Mesh Renderer. Then those can be separate outfit pieces.
If you have a T-Shirt that is one Skinned Mesh Renderer and Shorts that are a different Skinned Mesh Renderer then that will work.
You cannot mix and match **Outfit Pieces** from **Models** with different skeletons and bones.
In our example using UnityTechnologies Characters the "01Head" prefab is a single Skinned Mesh Renderer independent of other prefabs.

If you look inside the Outfit Piece prefab you will see 01Head -> 01Head and this object has the Skinned Mesh Renderer.



The skinned mesh renderer has “Update When Offscreen” set to true.

What is very important is that this object also contains "Armature" and all the bones under it. These can be called differently for different Models but correspond with a humanoid character and are necessary to map your **Outfit Piece** Skinned Mesh Renderer to your **Character Base**.

Your **Character** bones and your **Outfit Piece** bones must be the same and must have the same names!

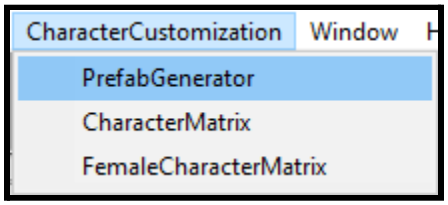
You should not move or delete any of these bones!

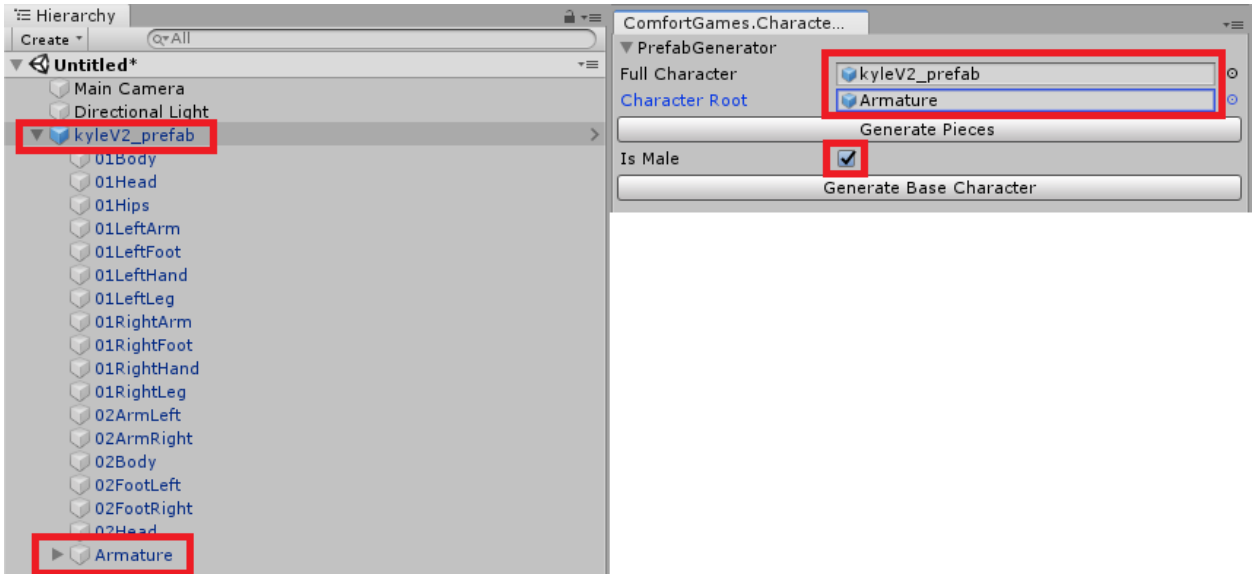
To create an **Outfit Piece** in our example for the male **Character** we can do so from the model “kyleV2_prefab”.

Let's make a brand new scene that we will not save so that we do not confuse it with anything.

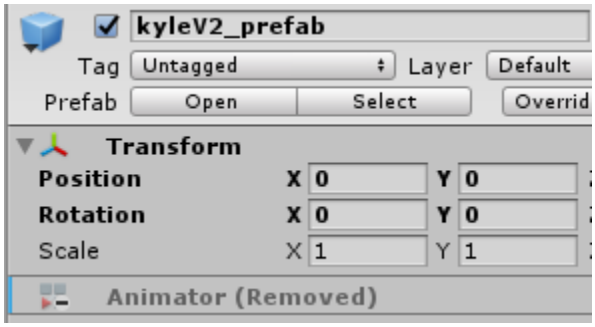
Let's take "kyleV2_prefab" and drag him into the blank scene.

We can generate Outfit Pieces with our tool. Go to CharacterCustomization -> PrefabGenerator.





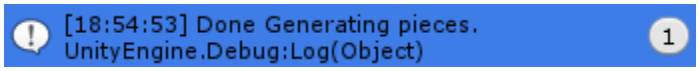
This time we want to first remove our “Animator” component and any other components from our model except for the transform. This is because our **Outfit Pieces** should not have anything else.



Don’t save or override your kyleV2_prefab when you remove the animator.

In the Prefab Generator toggle IsMale depending on the gender that these outfit pieces are for and click “Generate Pieces”.

This can take a couple seconds depending on how many Skinned Mesh Renderers your model has. When it is done you will see “Done Generating pieces” in the console output.

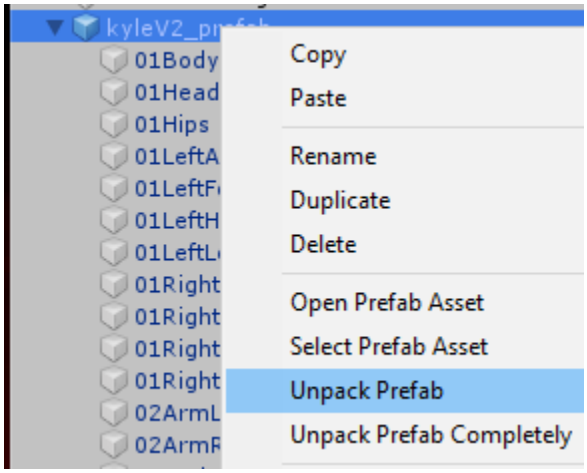


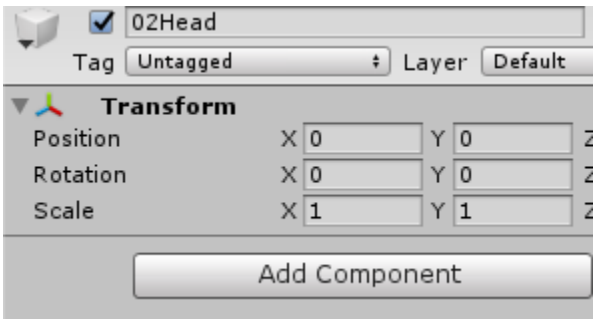
When it is done you will see the Generated Prefabs in the “GeneratedAssets” folder in your projects.

It is important that you do not move that folder or rename it!

You can then move your generated **Outfit Piece** prefabs into another folder of your choice. Sometimes the tool will not generate your pieces correctly or you might want to have more control over how the pieces are made so you can do it manually.

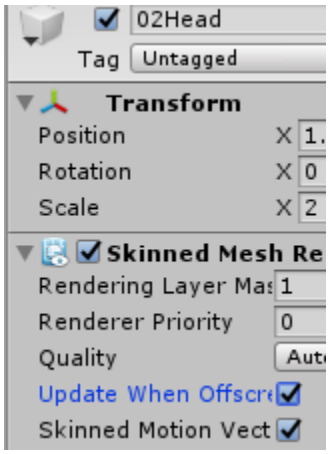
Let's manually make the Spike Head for Kyle.
In your Hierarchy right click on "kyleV2_prefab"and Unpack the Prefab.





Rename it to "02Head".
Remove the Animator Component.
Delete all the objects with Skinned Mesh Renderers and only keep the 02Head and the Armature.

The "02Head" has a SkinnedMeshRendrer component and you may want to set "Update When Offscreen" to True so that we always see it.



Again, it is very important that we Keep "Armature" and everything inside of it.

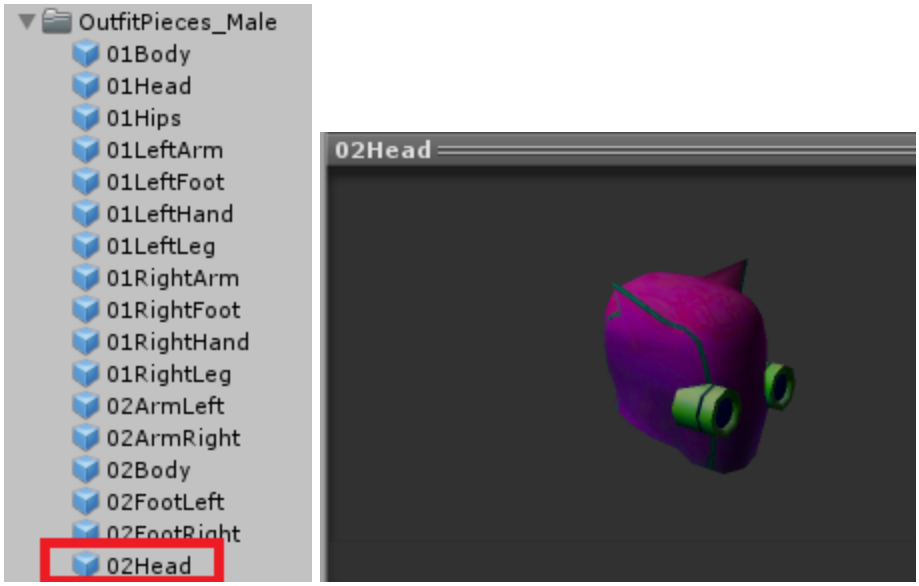
Sometimes models will have parts with mesh renderers in the bones structure, this is often the case for weapons like guns and swords.

For these cases we can disable the Skinned Mesh Renderer or the Mesh Renderer for those objects in the bones. But don't disable the GameObject itself. If you choose to keep them they will always be shown like Unity Chan's face.

Now we can take 02Head and drag it from the scene into the Project to create a Prefab for it.

The **Outfit Pieces** in our example have been placed in "OutfitPrefabs_Male" and "OutfitPrefabs_Female" but can be placed anywhere.

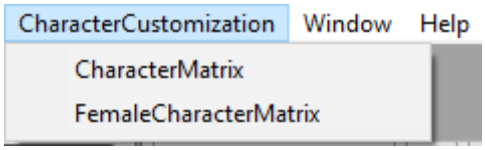
We actually already have a "02Head" in "OutfitPieces_Male".



Ideally, you may want to place them somewhere where your outfit models are since they are referenced by your OutfitScriptableObject in the OutfitPieces list.
Right now you can have the same **Outfit Piece** be referenced by two different **Outfits**.
For example a T-Shirt **Outfit Piece** can be in **Outfit1**. And a different **Outfit2** can have that same T-Shirt **Outfit Piece** with a Shorts **Outfit Piece** as a **Set**. Generally you probably want to have each **Outfit Piece** only referenced by one **Outfit**. You can add the **Outfit Piece** to your Outfit by selecting an Outfit Scriptable Object and dragging it to the Outfit Pieces list in the inspector. See **Outfit Scriptable Object** for details.
If you have a server driven game you might use asset bundles to load all of these into your game. Then your Outfit Model, Outfit Icon and Outfit Pieces could be part of the same asset bundle.

“Collision Matrix”

Currently there is a female and male **Outfit Collision Matrix**.
They are called “OutfitsCollision_Female” and “OutfitsCollision_Male”.
They are located in the “OutfitsCollisionData”.
You should not change their names or the folder name or move them anywhere else because then the system will not find them.
The matrix determines which **Outfits** collide with each other. Two **Outfits** that collide with each other cannot both be **Equipped** at the same time. So if you have one **Outfit Equipped** then when you try to **Equip** the second **Outfit** the first **Outfit** will automatically be **Unequipped**.
You can view and edit your **Outfit Collision Matrix** by opening their editor windows.
Go into the Editor Window "CharacterCustomization" and "CharacterMatrix" or "FemaleCharacterMatrix".



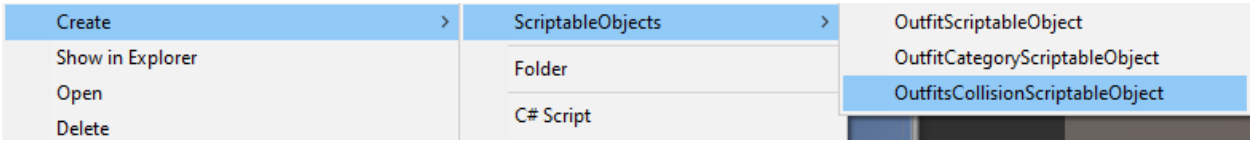
These automatically update based on the **Outfit Scriptable Objects** that you have created for each gender.

If you create or delete an **Outfit Scriptable Object** then this will shift all your collisions in the **Collision Matrix** and your collisions will need to be reconfigured.

It is best to create all your **Outfit Scriptable Objects** (models) first. Then, when everything is done you can configure your **Collision Matrix**.

When you make changes to a **Collision Matrix** you need to click out of the window of the matrix e.g. into the inspector window and then click back into the matrix window to make sure that your changes have been saved. This is because the matrix saves when it loses/gains focus.

If you accidentally deleted a **Collision Matrix** or needed to create a brand new one you can do so by right clicking in your project window and doing Create -> ScriptableObject -> OutfitsCollisionScritableObject.



The system currently only supports two **Collision Matrix** scriptable objects (one for each gender).

You can extend the system to support more in case you wanted to swap between them or if you had more character types that have distinct skeletons. For example instead of genders you might have fantasy races Elves, Orcs, Goblins that each have different bones so they would each need to also have their own set of models and their own collision matrix.

If you wish to modify the collision matrix editor or extend it you can do so using the “CharacterMatrix” and “FemaleCharacterMatrix” scripts.

“Outfit Icons”

Outfit icons are what appear in your view to represent your Outfit.

Normally the artist on your team will create them but while you are trying things out you can generate temporary ones with a tool.

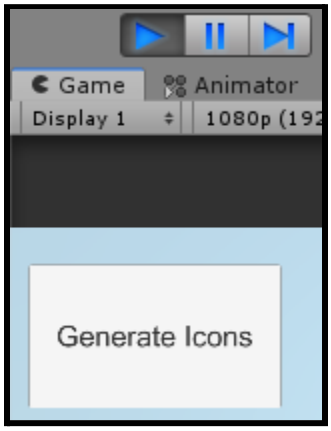
Go to the “CharacterCustomization05_GenerateAssets” scene.

In this example we will try to generate some Outfit Icons for the kyle character.

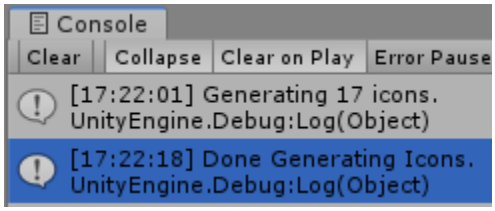
The scene has a “IconGenerator” with an “IconGenerator” script on it.

We need to give it a reference or a Full Character and that Character’s bone root, similar to what we did for the Prefab Generator tool. We also need to have a reference for the Icon Camera which uses a render texture.

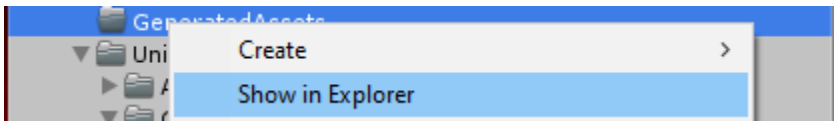
Our Model needs to be off to the side like how kyleV2_prefab has a transform X of -10 so that he doesn't appear in the picture while we generate icons.
Press play on the scene.
Then press "Generate Icons".



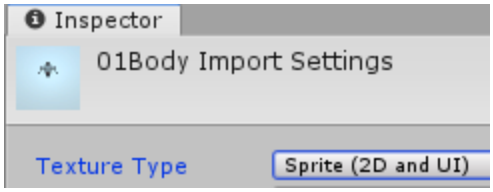
This will take several seconds and there is a delay between the snapshots that gets taken while the textures get saved.
When the process is done you will get a console output telling you "Done Generating Icons".



When it is done you will see the Generated Textures in the "GeneratedAssets" folder in your projects. It might take a moment before you see it or you might need to go into the folder.
If you still don't see the generated icons open the folder in windows or mac and they will be there.



It is important that you do not move that folder or rename it!
You need to change the generated icons type to TextureType Sprite (2D and UI).



You can extend this tool by making the camera zoom in on each outfit piece as it takes a snapshot and you can adjust the lighting in your scene so that the results are better.
Generated Icons should not be used for production and should be replaced with art made by the artist on your team.

“Saving”

The system saves the following information:

-The **Character’s** current **Gender**.

-**Outfits** the player **Equipped** per gender.

-**Outfits** the player owns by **Buying**.

-**Outfits** the player discovered by **Discovering**.

The save file is written to the Application.persistentDataPath and is called characterCustomizer.txt

It contains information for both genders.

It is in the binary format to make it slightly more difficult to tamper with.

You can extend this and obfuscate the data or persist your data on a server depending on your game.

The system automatically saves the player’s gender selection each time after they press the “male” or “female” button in the gender selection scene.

When a player **Equips** an **Outfit** this is not automatically saved because the player might not want to keep this change.

The system doesn’t automatically save when you **Buy** or **Discover** an **Outfit** but this is something you may want to do.

To save the changed **Equipped Outfits** the player needs to press the Save button.

If they want to discard their unsaved changes they can press the Load button or back out of the character editor scene or the player might close the game.

You can delete the save manually or whenever you want to start a brand new game by using the “Delete Save” button. You can also call the corresponding function from your own code..

You can extend the system in case you want to automatically save the equipment that the player **Equips** and then you would not show the option to manually save and load in your views.

“Views”

This system uses the Model View Controller pattern (MVC) to decouple your data from the logic and the display.
The OutfitSelectionViewPrefab is the main view that acts as a container for everything displayed for the particular selected gender.
Inside you will find children gameObjects with examples of navigating between the scenes of the system, and interacting with the system’s save file.
For every **Category** that you have created there will be a corresponding view instantiated.
It creates an OutfitCategoryToggleViewPrefab and OutfitCategoryContainerScrollViewPrefab.
These are separated because many character customization systems in AAA games like to have a visual separation for them.
The OutfitCategoryToggleViewPrefab is a toggle and is how you select which **Category** you are currently viewing.
The OutfitCategoryContainerScrollViewPrefab is a container that holds all the **Outfits** of that particular **Category**.
For each **Outfit** there is an OutfitViewPrefab created that is placed as a child into the corresponding **Category** parent.
You can modify all these views to match the visual style of your game or take the scripts from them and place them on your own view prefabs.

Note:
The folder structure, names and files names are very important for this system to function so be careful when renaming and moving things!
The CharacterCustomizationSceneManagementView contains scene references. In version 1.2 you can now modify them in the inspector instead of having to change the code.
CharacterCustomizationCameraView will zoom into a bone when a category is selected if that category has a targetBoneName. The FieldOfView is determined by the categoryZoom.
The CharacterCustomizationAssetManager contains your paths for resources.
The CharacterCustomizationFinderManager gets access to your managers and controllers.

“Adapters”

Adapters are a way for the system to work with other packages that have specific requirements.
For Example the Dias Climbing system uses Cinemachine.
The CinemachineAdapter script can be added to a CharacterSpawning object.
By default it has a cinemachineStateDrivenCameraName and cameraTargetName but you can change these if you rename the object in the scene.
This will allow the camera to know about the characters we spawn.

Note:
If you delete an outfit model this will require you to reconfigure the collision matrix.
Close the Collision Matrix editor window. Then re-open it and change the flags of the Matrix.
In general the best approach is to make all your models and then configure your collision matrix for that gender.

Note:
You will need to lose focus and regain focus of a character collision matrix editor window to save your changes. You can do so by clicking out of the window and back in it.

Note:
Animators need to have Culling Mode set to Cull Update Transforms (since initially they have no SkinnedMeshRenderers prior to generating your character).

Note:
It is best to set your outfit SkinnedMeshRenderers to Update When Offscreen (true) in the inspector so that they are always visible.

Note:
Every Outfit must be part of a Category!

Thank you for your support and for purchasing the **Character Customization** asset.
If you have any questions or comments you can find more information on the website.

COMFORTGAMES.CA

You can send an email for additional inquiries.

INFO@COMFORTGAMES.CA