Designing and Customizing Character Sprites - Instructions & Reference Manual

by William Chilcote

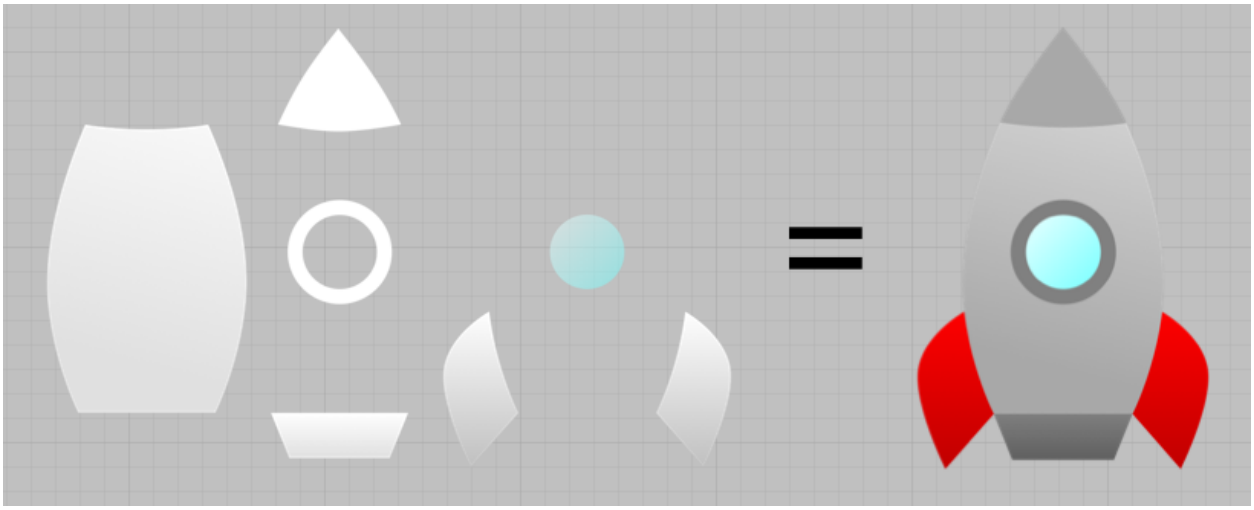
Contents

Sprite Composition Guidelines

Sprite Component Basics

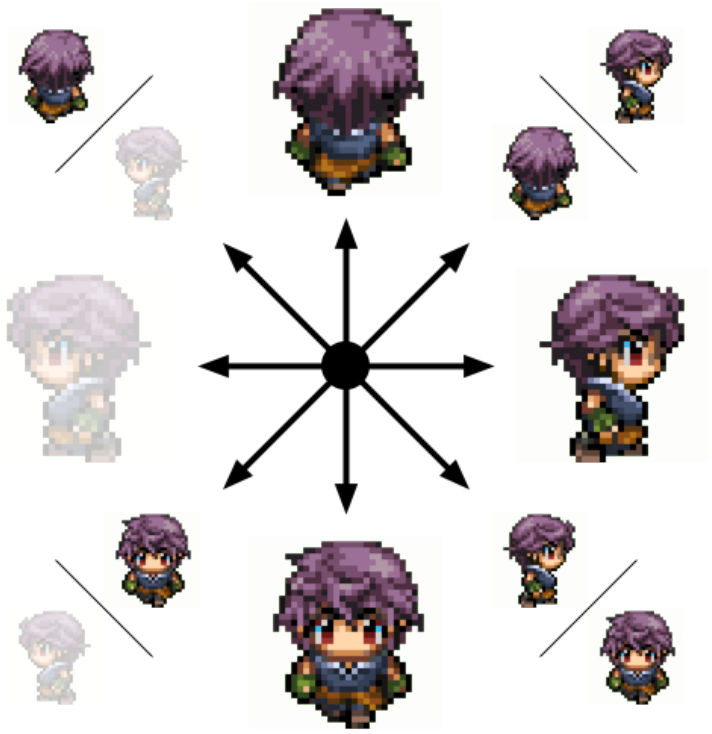
Characters, including the player and most NPCs in the game, are composed of multiple grayscale sprite components to allow for greater customization with a minimal amount of assets. All characters consist of a base sprite, representing their body. Using the body sprite as a reference point, other sprites, such as clothing and accessories, can be overlaid. As the sprites do not contain any associated positional information, the center point of each sprite must match the center point of each additional sprite on the character. For sprites that represent things such as shoes or hair, that exist primarily on the top or bottom of the character, this means there must be empty space extending just as far from the center in the opposite direction for the sprite's center to align correctly.

All character sprites, including the body sprite, should be created or at least exported in grayscale, so the color can be procedurally set in game, either by the level designers, or by the player during character customization. The brightest pixel in most sprites should generally be white (#FFFFFF), or very close to it, to allow for the greatest range of brightness that can be set in the editor. Unity can change the color of sprites to be darker, but it cannot change their color to be lighter.



8-Directional Movement

Character movement is 8-directional. However, the sprites used by player animation system are 4-directional. For diagonal directions, the sprites representing the most recent cardinal direction are used. This applies both to the body sprite, as well as to all attached component sprites.



As long as a sprite is symmetric, it can be drawn first for either left or right, and then automatically flipped.

Animation

Because characters often are required to walk, they need additional frames to represent multiple steps in the walk cycle. The minimum number of frames in a walk cycle should be four - neutral, first foot forward, back to neutral, and other foot forward. However, adding more frames will make the animation appear smoother.

Because character navigation is 8-directional, there must be an additional version of each frame for each direction. A single sprite can also be used for both neutral positions in the walk cycle.This means that for a symmetric sprite, with a 4 frame animation cycle using a single neutral sprite for each direction, there will be 3 x 3 = 9 unique frames.

While it is not necessary for many non-body sprites to be animated, such as for hair that doesn't need to move, other sprites, such as pants will have to be animated to match the movement of the legs for each frame in the walk cycle.

Wardrobe Editor

The Wardrobe is a custom asset that contains a hierarchy of all potential character customizations available to the player and to NPCs at any point. The asset has a custom editor that is only available within the Unity Editor, designed to load, populate, and save the asset between runtimes. The Editor is opened by selecting Window > Wardrobe from within Unity.

It includes several different levels in which sprites are organized, many with additional information attached, as well as a list of color palettes that the sprite selections can make use of. The hierarchy is organized as follows: Part > Version > Layer > Direction > Frame.

Part

The first primary component of the Wardrobe Editor is “All Sprites.” This foldout contains listed items for each different spatial part of the character that can be customized. It includes the Body, Hair, Eyes, Top, Bottom, and Shoes, as specified by their “Name” properties. The number on top represents how many different versions of each part are available to be switched between when customizing. For example, the number could be 2 for Hair if there are 2 different hairstyles to choose from. The “Palette” property is a dropdown that determines which color palette will be used for this part of the player in the character customization interface. The process for creating these palettes can be found below. The last listed item for each part is a foldout for each of the versions specified by the number above.

Version

Within each version foldout, the first item is the “Name” property. This contains a string that will be seen by the user when previewing this version in the character customization interface. The second attribute is “Frames.” This contains an integer that specifies how many frames are to be played for this customization before cycling back to the beginning. The standard number is 4. The third attribute is “Layers,” which works similarly to the way layers work in other image editing software (except for the render order which can be stored as additional metadata as explained below).

Layer

An additional foldout is shown for each layer specified by the “Layers” property. Within each layer, the first property is “Apply tint.” This determines if the sprites on this layer will have their color modified by `SpriteRenderer`, set by a color chosen by the user from a palette defined by this part’s “Palette” property. If the layer is intended to contain any white pixels, this property should not be selected. The second property is “Render order,” which specifies the order in which the sprites in this layer are rendered. It is used to set the “Order in Layer” property on the Sprite’s `SpriteRenderer`. It is important to note that while the sprites’ render orders may be changed, their

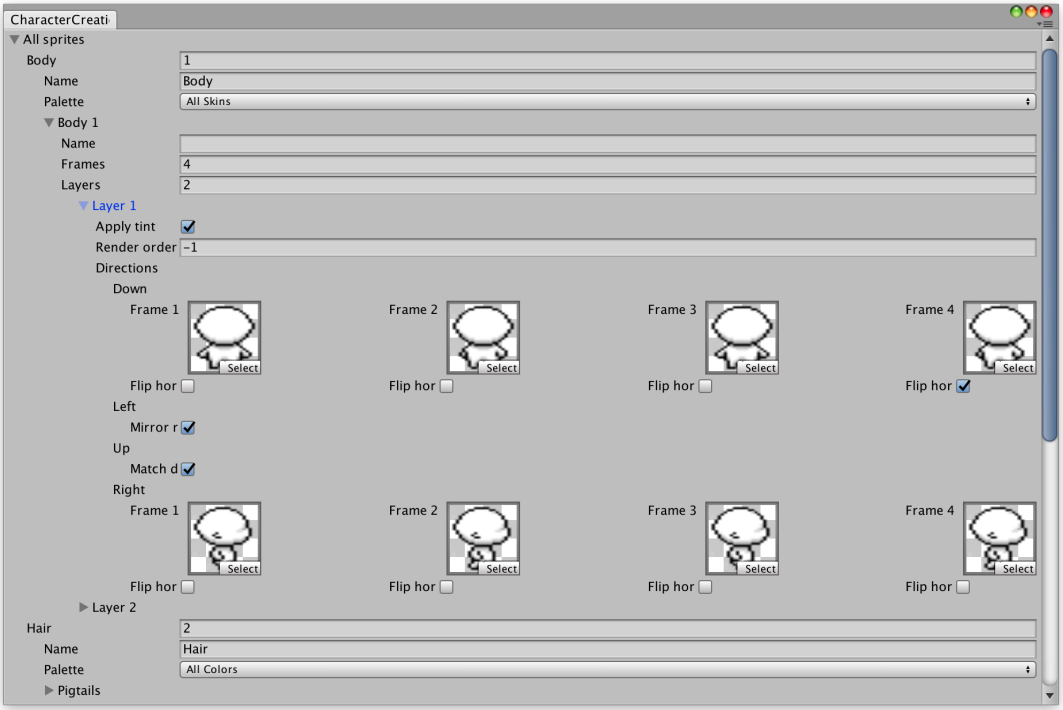
SpriteRenderers are all set to the same Sorting Layer. The information toward the bottom of each layer foldout is specific to each direction.

Direction

Four directions are displayed for each layer - down, left, up, and right. The “left” direction contains a checkbox property “Mirror right.” If this is checked, all sprites specified under the “right” direction will be horizontally flipped and used for the “left” direction as well. The “up” direction contains a similar checkbox property titled “Match down.” If this is selected, the sprites used for the “down” direction will also be used for the “up” direction with no modifications. This is useful for sprite sets such as the base body that are visually the same in the front and the back.

Frame

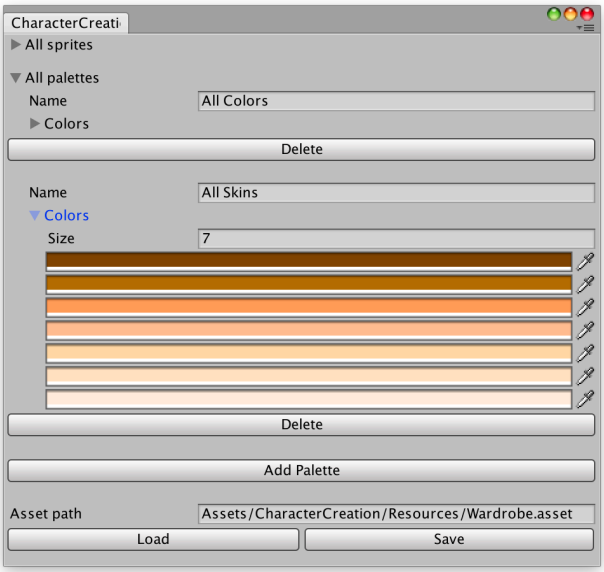
The amount of frames available to bet set within each direction is determined by the “Frames” property in the enclosing version. Each frame contains a sprite field, labelled with its index in the animation cycle, and a checkbox property “Flip horizontal.” If this is checked, the given sprite will be flipped when displayed. This is useful for sprites that display horizontal symmetry, such as mid-step frames for the character to move up or down.



Palettes

Below the “All Sprites” foldout is the “All palettes” foldout. This contains a list of palettes that are available for each character part to choose from. Each palette contains a “Name” property which contains the string that will be displayed in the dropdown field

for each character part above. Below is a foldout for the list of colors that make up the palette. This foldout works similar to the way that Unity serializes the type `List<Color>` in the Inspector by default. As expected, new palettes can be created with the “Add Palette” button, and individual palettes can be deleted with the “Delete” button beneath their colors.



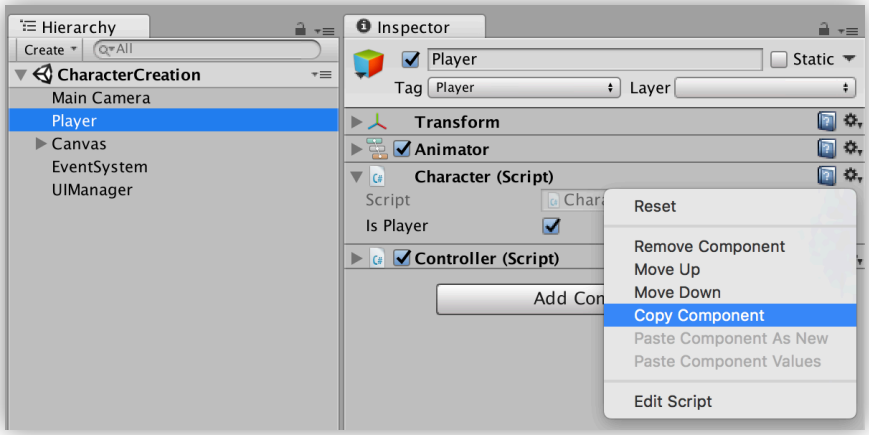
Wardrobe Asset Serialization

At the bottom of the Wardrobe Editor is a set of serialization options. When the editor window is opened, the “Asset path” property is populated with the path of the first found asset of type `Wardrobe` in the project’s Assets folder, and that asset is loaded into editor’s properties above. Although this is generally not needed, a path to an alternative `Wardrobe` asset can be specified and loaded into the editor using the “Load” button. When any changes have been made to the asset through the editor, they will be deleted when the Unity Editor is closed if the “Save” button is not pressed first. Character Creation-related scripts locate the `Wardrobe` asset using the `Resources.Load()` method, so it is always necessary for the `Wardrobe.asset` file to be in the top level of a “Resources” directory.

Interface & Serialization

Character design information is serialized so it can be saved between instances of the game. Character information for NPCs is determined by the developers, and fixed at compile-time, so it is serialized internally. The player has the option of customizing their avatar at runtime however, which necessitates the use of external serialization for the player character. The file containing the user's customizations is stored as part of the game's save file, in Unity's `Application.persistentDataPath` directory, named `SaveData.ucd`. All character information is deserialized at runtime.

While there is not a custom interface for designing NPC characters, this can be achieved easily through the existing player customization interface and simple built-in Unity Editor functionality. NPCs can be designed the same way as the player, by using the in-game character customization interface. However, instead of pressing "Start" to save the character information to the player file and be sent into the game, pause Play mode, find the `CharacterCreation.Character` component attached to the Player GameObject, and select "Copy Component," using the gear button on the top right of the component cell in the Inspector.



Then, when viewing whatever scene contains the desired target NPC character, select it's attached `CharacterCreation.Character` component in the GameObject's Inspector window, and paste the component while not in Play mode. Be sure that the boolean property `isPlayer` is not selected, or the NPC will load the character customizations for the player at runtime.

If a `CharacterCreation.Character` component has it's property of type `CharacterCreation.CharacterSetting` set to null at when `Monobehaviour.Awake()` is called on it, a random `CharacterSetting` instance will be generated. This applies to both NPC and Player characters. Because `CharacterCreation.Character` utilizes the `ExecuteInEditMode` attribute,

`Monobehaviour.Awake()` is called when the scene is loaded. This is so the character's final form can be seen from Edit mode without having to press Play. To reset the character settings for an NPC, select "Reset" from the gear menu on the Inspector cell for the `CharacterCreation.Character` component. Reloading the scene will cause a new NPC customization to be generated.