Crispoly Characters Mini - Unity Asset

Version 1.0

Characters: 135

Animations 216

Overview

Crispoly Characters is a low poly character collection where all characters share the same armature making it easy to reuse animations between them. The low poly characters only have between 150-800 vertices and they all share the same material making them very performant for any game, including low-end mobile devices.

Table of Contents

Key Features

- All characters share the same armature = simple
- All characters can use all animations = flexible
- All characters use the same material = performance
- Texture sizes: 3 × 128×128 pixels, 64 KB each = low memory usage & performance
- Mesh vertex count: 150-800 per character* = performance & aesthetic
- URP (default), HDRP, and Built-in Render Pipeline Support

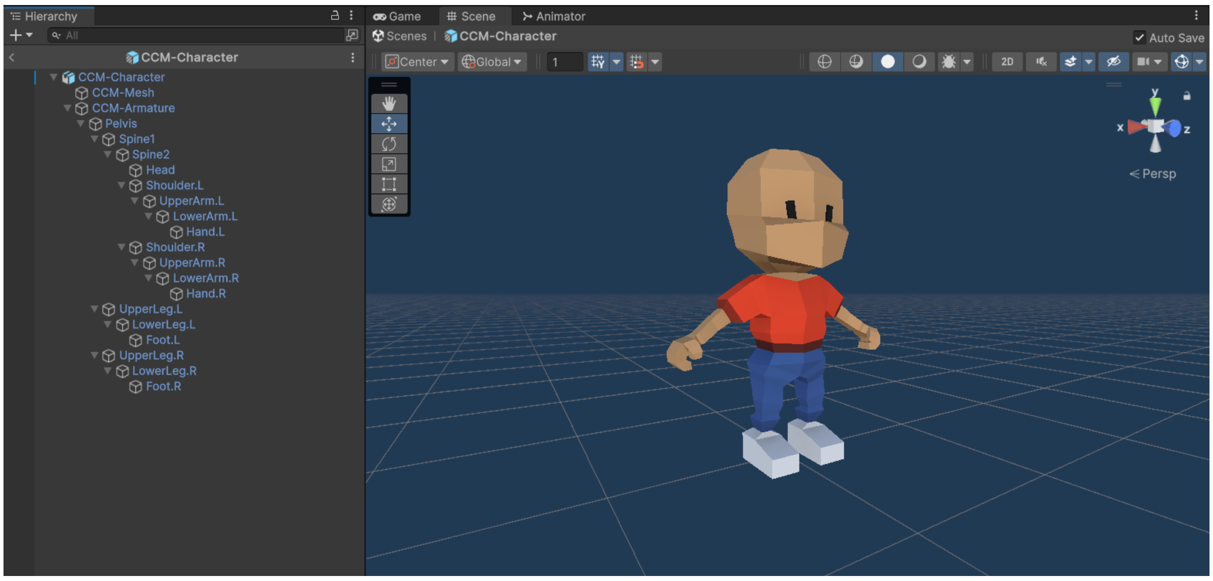
**) The game engine will report 1000-3000 vertices because the characters are flat-shaded and to render flat surfaces, the game engines need to duplicate each vertex multiple times to get different directional normals.*

Key Assets

CCM-Character_URP Prefab

Path: Imphenzia\CrispolyCharactersMini\Prefabs\CCM-Character_URP.prefab

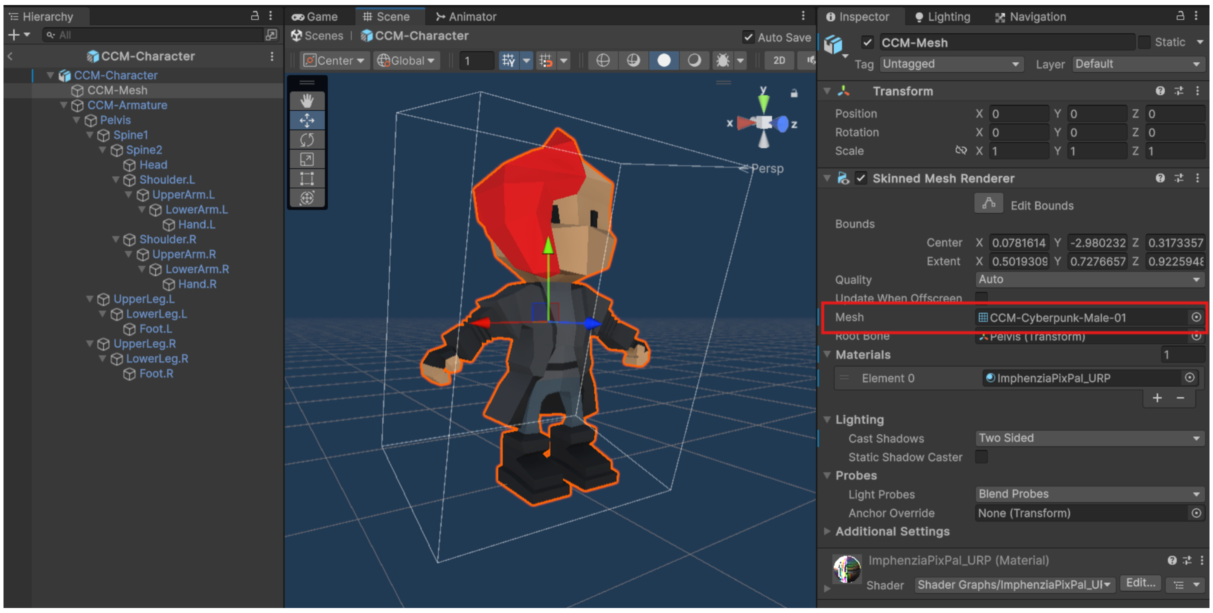
This is the main character prefab. It contains the master character mesh with material, the armature hierarchy, and an Animator component on the prefab root.



CCM-Character
- CCM-Character
  - CCM-Mesh
  - CCM-Armature
    - Pelvis
      - Spine1
        - Spine2
          - Head
          - Shoulder.L
            - UpperArm.L
              - LowerArm.L
                - Hand.L
          - Shoulder.R
            - UpperArm.R
              - LowerArm.R
                - Hand.R
      - UpperLeg.L
        - LowerLeg.L
          - Foot.L
      - UpperLeg.R
        - LowerLeg.R
          - Foot.R

Changing the character mesh

For an instantiated prefab, or if you create your own prefab, you can change the mesh by selecting the CCM-Mesh child object and changing the Mesh under Skinned Mesh Render in the inspector.



CCM-Character
- CCM-Character
  - CCM-Mesh
  - CCM-Armature
    - Pelvis
      - Spine1
        - Spine2
          - Head
          - Shoulder.L
            - UpperArm.L
              - LowerArm.L
                - Hand.L
          - Shoulder.R
            - UpperArm.R
              - LowerArm.R
                - Hand.R
      - UpperLeg.L
        - LowerLeg.L
          - Foot.L
      - UpperLeg.R
        - LowerLeg.R
          - Foot.R

CCM-Mesh Static
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Skinned Mesh Renderer
Bounds
Center X 0.0781614 Y -2.980232 Z 0.3173357
Extent X 0.5019309 Y 0.7276657 Z 0.9225948
Quality Auto
Update When Offscreen
Mesh CCM-Cyberpunk-Male-01
Root Bone Pelvis (Transform)
Materials 1
Element 0 ImphenziaPixPal_URP
Lighting
Cast Shadows Two Sided
Static Shadow Caster
Probes
Light Probes Blend Probes
Anchor Override None (Transform)
Additional Settings

ImphenziaPixPal_URP (Material)
Shader Shader Graphs/ImphenziaPixPal_UR

You can also change the character's skinned mesh renderer mesh from a script. For example, this randomizer script lets you specify character renderers in the scene and meshes from the project and when pressing space it randomizes all the meshes as seen in the "DemoActions" scene.

```
using UnityEngine;

namespace Imphenzia.CrispolyCharactersMini
{
    public class Randomizer : MonoBehaviour
    {
        [SerializeField] SkinnedMeshRenderer[] renderers;
        [SerializeField] Mesh[] meshes;

        void Update()
        {
            if (Input.GetKeyDown(KeyCode.Space))
            {
                Randomize();
            }
        }

        public void Randomize()
        {
            foreach (var renderer in renderers)
            {
                    // This is where the renderer gets
                    // assigned a new character mesh
                renderer.sharedMesh =
                    meshes[Random.Range(0, meshes.Length)];
            }
        }
    }
}
```

CCM-CharacterWithProps_URP Prefab

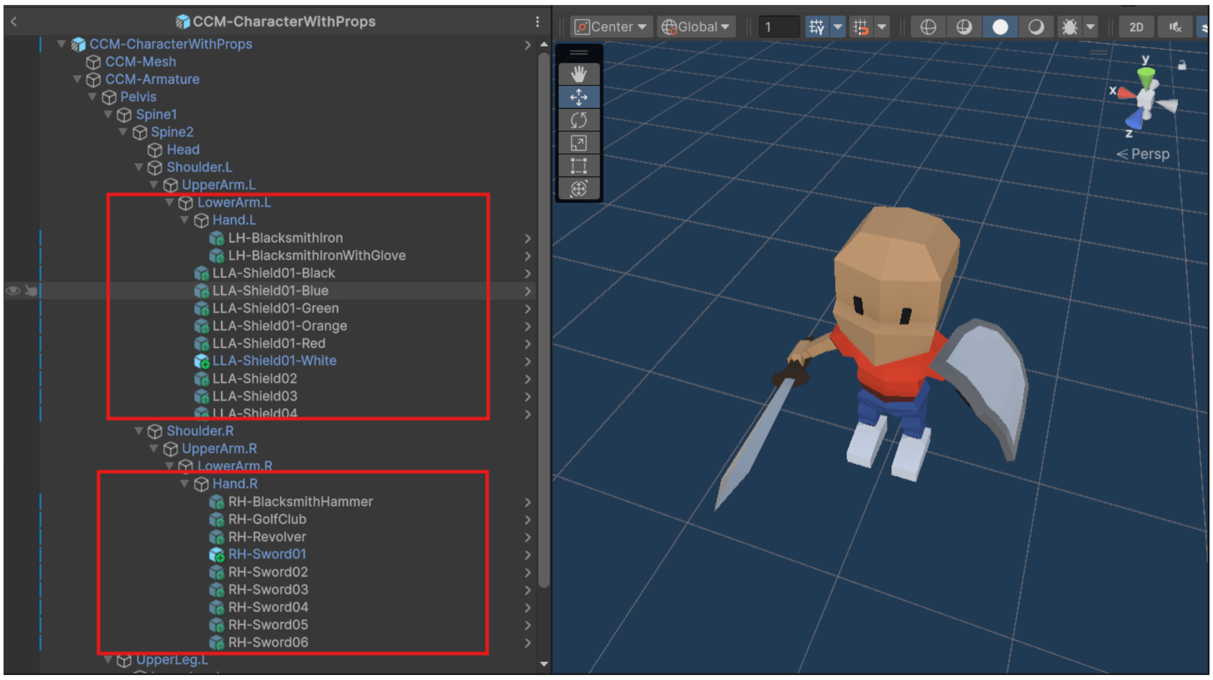
Path: Imphenziza\CrispolyCharactersMini\Prefabs\CCM-CharacterWithProps_URP.prefab

This prefab comes with some included props. The props are parented to various bones. Weapons are generally parented to Hand.R and shields are generally parented to ForeArm.L. When the props are parented to bones, they will automatically follow along when animations are played.

Working with props

You can add props to the character by dragging and positioning props to bones of the character prefab or new character prefab variants. You can also instantiate props through code to add or remove items from the characters.

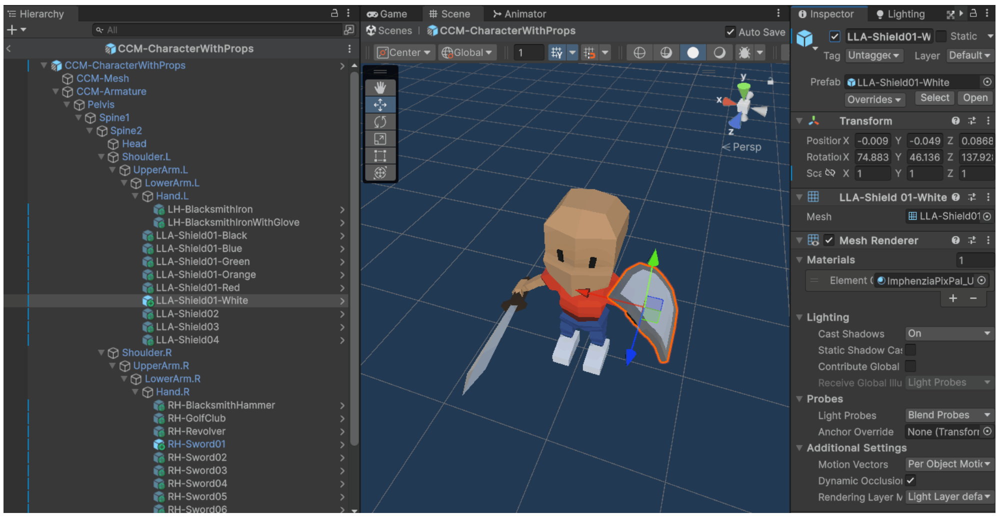
Commonly used props can be included in a deactivated game object state all the time and then enabled when necessary.



```
CCM-CharacterWithProps
  CCM-Mesh
  CCM-Armature
    Pelvis
      Spine1
        Spine2
          Head
          Shoulder.L
            UpperArm.L
              LowerArm.L
                Hand.L
                  LH-BlacksmithIron
                  LH-BlacksmithIronWithGlove
                LLA-Shield01-Black
                LLA-Shield01-Blue
                LLA-Shield01-Green
                LLA-Shield01-Orange
                LLA-Shield01-Red
                LLA-Shield01-White
                LLA-Shield02
                LLA-Shield03
                LLA-Shield04
          Shoulder.R
            UpperArm.R
              LowerArm.R
                Hand.R
                  RH-BlacksmithHammer
                  RH-GolfClub
                  RH-Revolver
                  RH-Sword01
                  RH-Sword02
                  RH-Sword03
                  RH-Sword04
                  RH-Sword05
                  RH-Sword06
      UpperLeg.L
```

The prefab, as seen in the screenshot above, has multiple props attached but set to inactive. By default the LLA-Shield01-White shield is activated under LowerArm.L, and the RH-Sword01 weapon is activated under Hand.R.

It is recommended that you create your own character prefabs and only including props that are appropriate for your game.



```
CCM-CharacterWithProps
  CCM-Mesh
  CCM-Armature
    Pelvis
      Spine1
        Spine2
          Head
          Shoulder.L
            UpperArm.L
              LowerArm.L
                Hand.L
                  LH-BlacksmithIron
                  LH-BlacksmithIronWithGlove
                LLA-Shield01-Black
                LLA-Shield01-Blue
                LLA-Shield01-Green
                LLA-Shield01-Orange
                LLA-Shield01-Red
                LLA-Shield01-White
                LLA-Shield02
                LLA-Shield03
                LLA-Shield04
          Shoulder.R
            UpperArm.R
              LowerArm.R
                Hand.R
                  RH-BlacksmithHammer
                  RH-GolfClub
                  RH-Revolver
                  RH-Sword01
                  RH-Sword02
                  RH-Sword03
                  RH-Sword04
                  RH-Sword05
                  RH-Sword06
```

```
[✓] LLA-Shield01-W   Static
Tag Untagged   Layer Default
Prefab LLA-Shield01-White
Overrides   Select   Open
Transform
  Position  X -0.009  Y -0.049  Z 0.0868
  Rotation  X 74.883  Y 46.136  Z 137.92
  Scale     X 1       Y 1       Z 1
LLA-Shield 01-White
  Mesh  LLA-Shield01
[✓] Mesh Renderer
  Materials  1
    Element  ImphenziaPixPal_U
  Lighting
    Cast Shadows  On
    Static Shadow Cas
    Contribute Global
    Receive Global Illu  Light Probes
  Probes
    Light Probes  Blend Probes
    Anchor Override  None (Transform
  Additional Settings
    Motion Vectors  Per Object Motio
    Dynamic Occlusion  ✓
    Rendering Layer M  Light Layer defa
```

This example shows how different props can be deactivated and activated. If you want to add your own props, import them and parent them to bones and position them as needed.

Props are prefixed with to easier identify which bone they may be best parented to, e.g. RH for Right Hand and LLA for Left Lower Arm.

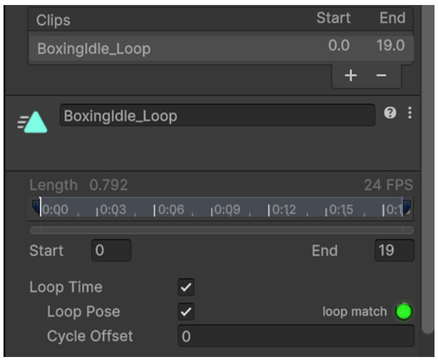
Character Animations

All character animations are stored in the **Imphenzia\CrispolyCharactersMini\Animations** folder. Animations are individually stored so you can remove unused animations in your project and to easily allow addition of more animations in the future.

Looping Animations

Looping animations have the suffix "**_Loop**" and they are configured to loop through the "Loop Time" and "Loop Pose" settings.

Mainly, all the idle animations, walking animations, and running animations are configured as loops.



Clips	Start	End
BoxingIdle_Loop	0.0	19.0

BoxingIdle_Loop

Length 0.792 24 FPS

Start 0 End 19

Loop Time ✓

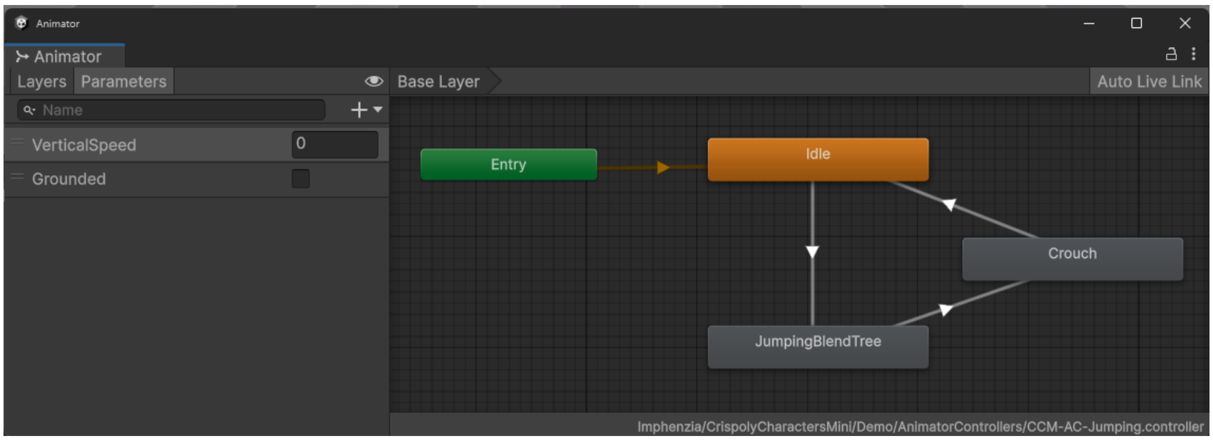
Loop Pose ✓ loop match ●

Cycle Offset 0

Pose Animations

Some animations have the suffix "**_Pose**" and they are included because it is common to blend between poses either using blend trees or through scripts.

The demo scene **DemoPoseBlending_URP** demonstrates how frame blending can be used. The scene has a demo character that uses the **CCM-AC-Jumping** AnimatorController which has an idle state, a blend tree to blend between jump up and fall down poses, and a crouch pose for landing.



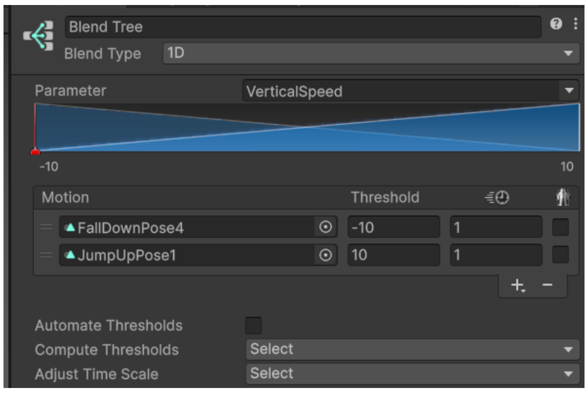
VerticalSpeed 0

Grounded

Imphenzia/CrispolyCharactersMini/Demo/AnimatorControllers/CCM-AC-Jumping.controller

The idle loop plays by default and when the Grounded bool is true. The demo script **jumping.cs** applies a jump force to the rigidbody component of the character every other second. As the character jumps up into the air, a physics check in the script detects that the character is not touching the floor and sets the Grounded bool to false which makes the Animator transition to the **JumpingBlendTree** state.

The JumpingBlendTree state has a BlendTree with two animations which in this case are poses rather than animations. The VerticalSpeed parameter is sent from jumping.cs to the animator and it contains the Y velocity of the rigidbody. The blend tree has a threshold of -10 for the falling down pose, and 10 for the jump up pose. As the character smoothly jumps up into the air and falls back down to the ground, the value of **VerticalSpeed** interpolates between the jumping and falling poses making it seem like an animation rather than key frames.



Blend Tree

Blend Type 1D

Parameter VerticalSpeed

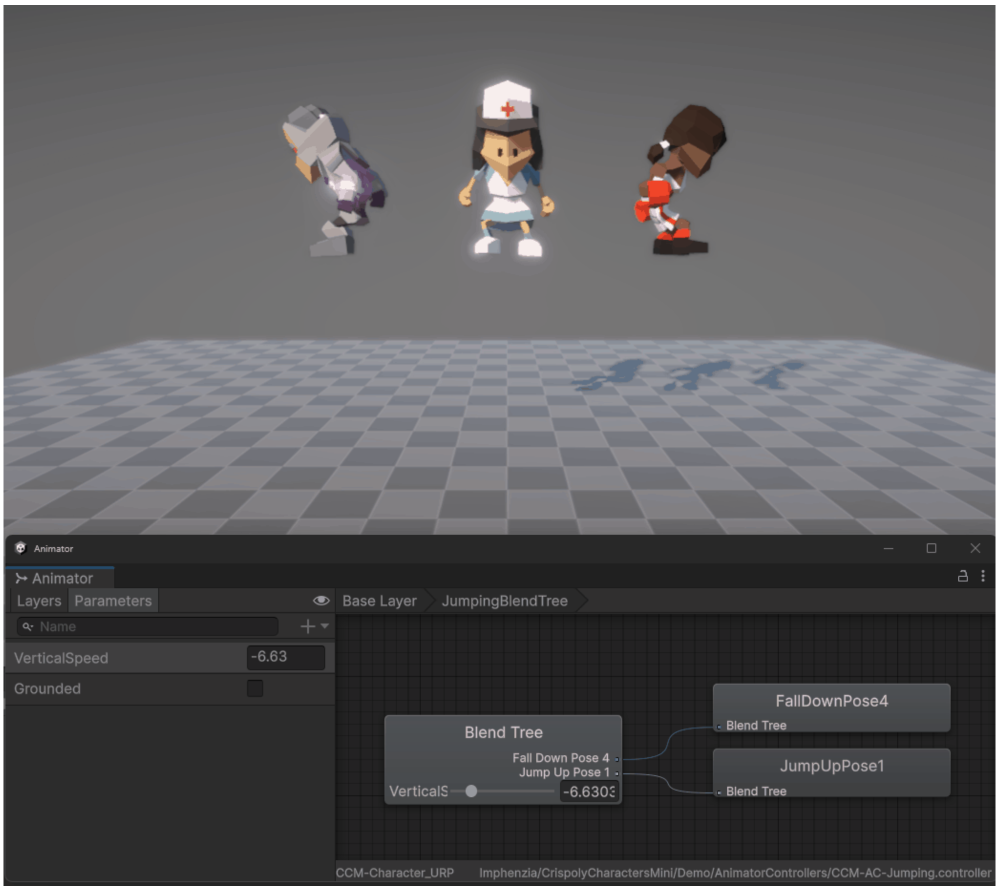
-10 10

Motion	Threshold		
FallDownPose4	-10	1	
JumpUpPose1	10	1	

Automate Thresholds

Compute Thresholds Select

Adjust Time Scale Select

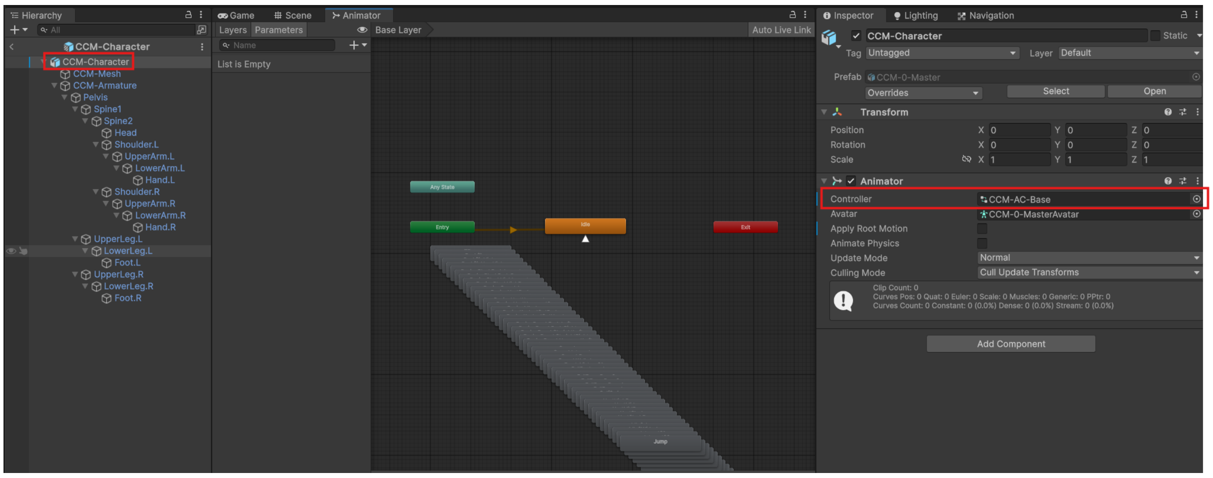


When the character lands again, it transitions to the Crouch pose, and then it blends back to the idle animation.

AnimatorController

To play Animations you need to configure an AnimatorController and set up the animations you need for your project. Each game will require their own AnimatorController since character movement and action needs vary from game to game.

This asset comes with some AnimatorController samples that can help you along the way and they can be located in the **Imphenzia\CrispolyCharactersMini\Demo\AnimatorControllers** folder.



CCM-Character
Tag: Untagged
Layer: Default
Prefab: CCM-0-Master

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

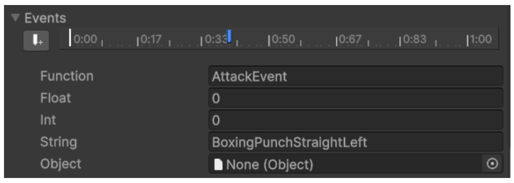
Animator
Controller: CCM-AC-Base
Avatar: CCM-0-MasterAvatar
Apply Root Motion
Animate Physics
Update Mode: Normal
Culling Mode: Cull Update Transforms

Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0 (0.0%)

The prefab comes configured **CCM-AC-Base** animation controller. This has a looping idle animation as the only transitional state and all other animations have been imported without transitions.

Animation Events

Some animations come configured with animation events. These events are triggered by the animation and can be useful to call a function, for example when a punch or kick is at its most extended state. In the example to the right, the animation BoxingPunchStraightLeft calls a function called AttackEvent() is called if there is one in a MonoBehaviour **on the same game object** as the AnimatorController component.


Events

Function	AttackEvent
Float	0
Int	0
String	BoxingPunchStraightLeft
Object	None (Object)

These animation events have been set up with the **SendMessageOptions.DontRequireReceived** so they won't generate an error if they are not found. The animation events have also been configured to send a string parameter as an identifier so you can play the appropriate sound effect or perform appropriate actions.

Other demo prefabs and scenes that use animation events to call script functions are boxers, blacksmith and martial artists.

You will most likely want to set up your own animation events for your game, so these are more included for demonstration purposes.

List of included animation events:

Animation	Time	Function Called	String Parameter
BlacksmithHammer	0.6666667	SparksEvent	
BoxingPunchBodyLeft	0.4	AttackEvent	BoxingPunchBodyLeft
BoxingPunchBodyRight	0.3333333	AttackEvent	BoxingPunchBodyLeft
BoxingPunchStraightLeft	0.4	AttackEvent	BoxingPunchStraightLeft
BoxingPunchStraightRight	0.4	AttackEvent	BoxingPunchStraightRight
BoxingUppercutLeft	0.3333333	AttackEvent	BoxingUppercutLeft
BoxingUppercutRight	0.2631579	AttackEvent	BoxingUppercutRight
GolfSwing	0.1538462	GolfSwingEvent	
KickFront	0.4	AttackEvent	KickFront
KickSide	0.32	AttackEvent	KickSide
KickSideAlt1	0.4444444	AttackEvent	KickSide
KickSideAlt2	0.3636364	AttackEvent	KickSide
KickSideAlt3	0.32	AttackEvent	KickSide
RevolverBodyHipFire	0.1428571	AttackEvent	RevolverBodyHipFire
RevolverDraw	0.1666667	RevolverAppearEvent	RevolverDraw
RevolverFire	0	AttackEvent	RevolverFire
RevolverHipFire	0	AttackEvent	RevolverHipFire
SwordHit1	0.59375	AttackEvent	SwordHit1
SwordHit2	0.5	AttackEvent	SwordHit2
SwordSlash1	0.46875	AttackEvent	SwordSlash1
SwordSlash2	0.4117647	AttackEvent	SwordSlash2
SwordStab1	0.5	AttackEvent	SwordStab1

Render Pipelines

The asset was designed for URP but materials are supplied for all three render pipelines. Prefabs, materials, shaders, and demo scenes have the suffix _URP for Universal Render Pipeline, _HDRP for High Definition Render Pipeline, and no suffix for Build-in Render pipeline.

Demo Content

Support is not offered for the demo content - it is included to showcase some of the feature and to act as guidance.

Scenes

Demo_URP.scene / Demo_HDRP.scene / Demo.scene

This scene shows all the characters in random positions on a grid along with sample animations. The demo scene is also available for HDRP (Demo_HDRP.scene) and Built-In Render Pipeline (Demo.scene).

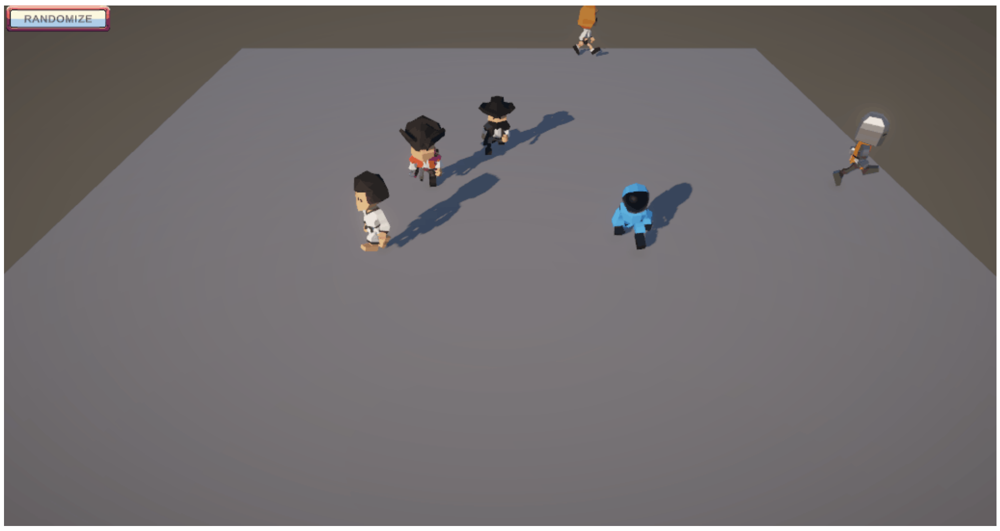
DemoActions_URP.scene

This scene shows how some of the characters and animations can be used. Demo prefabs have been set up in the scene to show boxers, martial artists, a blacksmith, fighting knights, a golfer, a backflipping ninja, and a cowboy.

If you look at the prefabs in the scene, you can see example animator controllers and scripts that are used to perform actions and in the case of the boxers, also detect punches and play animations when hit.

DemoRunning_URP.scene

This scene shows characters running to random positions on a NavMesh with NavAgent components. Three characters use a demo animation controller and script that makes them run forward in the direction of the destination. The other three characters use a different animator controller and script showing how the characters can use 9-directional strafing which is useful if you are creating a top-down shoot'em up where the character should always face forward.



DemoPoseBlending_URP.scene

This scene demonstrates how pose animation (containing just a single pose) can be blended into fluent animations using the AnimatorController and blend trees. See for more information and examples.

Thank you for buying Crispoly Characters Mini. Happy gamedev!