Table of contents

1 Getting started

This chapter covers everything you need to do to fully integrate this asset into your own project. The provided demo scenes can be run after the project is correctly set up. I have also uploaded a Getting Started video showing all necessary steps.

1.1 Project setup

The following steps describe the needed project settings for using my asset:

1.1.1 Installing and activating Unity's new Input System

After importing my asset, you will most likely get errors shown in the console. This is because you have not installed or enabled Unity's new Input System (*Window > Package Manager > Unity Registry > Input System*). After its installation, you are prompted with the question if you would like to enable the new input system and restart the Unity Editor. If you are planning to only use the new input system, you can click "Yes". Otherwise, I would recommend clicking "No", going to the *Edit > Project Settings > Player > Active Input Handling* and selecting "Both" from the drop-down list. This way, both input systems can be used in your project. Now you can restart the Unity Editor so that the changes take effect.

Note: This step is required even if you want to stick to Unity's legacy input system. In this case, you would disable "Use New Input System" on each of my input handling scripts, i.e. the RPG Controller and RPG Camera.

1.1.2 Creating the "Player" layer

Navigate to *Edit > Project Settings > Tags and Layers* and add layer "Player" to the list (preferably as "User Layer 8" so that they are automatically assigned to the provided prefabs).

1.1.3 Creating the needed inputs (required if you want to use the old input system)

Navigate to *Edit > Project Settings > Input Manager > Select Preset* (middle button at the top right) and select "RPGInputManager". Note that this will overwrite any existing adjustments to the currently set up input manager!

1.2 Scene setup

In the following, you find the steps for correctly setting up the scene with my asset in it:

1.2.1 Attaching the scripts

All scripts which are related to the character can be found inside *Scripts > Character*. Attach the scripts of your choice to your character game object, usually to its root game object:

a. *Camera* > RPGCamera: Attach this script if you would like to use my camera controls, i.e. everything which has to do with the behavior of the virtual camera. The required helper component RPGViewFrustum will automatically be attached as well.
b. *Motor* > RPGController: Attach this script if you would like to use my character controls, i.e. everything which deals with processing user input and propagating them to a motor. Therefore, having a controller without a motor does not have any effect. You additionally need to add a RPGMotor of your choice: ARPG or MMO.

c. *Motor* > RPGMotor[ARPG/MMO]: Choose how the player inputs should be processed, i.e. either in an ARPG or an MMO fashion. Attaching an RPGMotor component will automatically add Unity's Character Controller component if not already present.

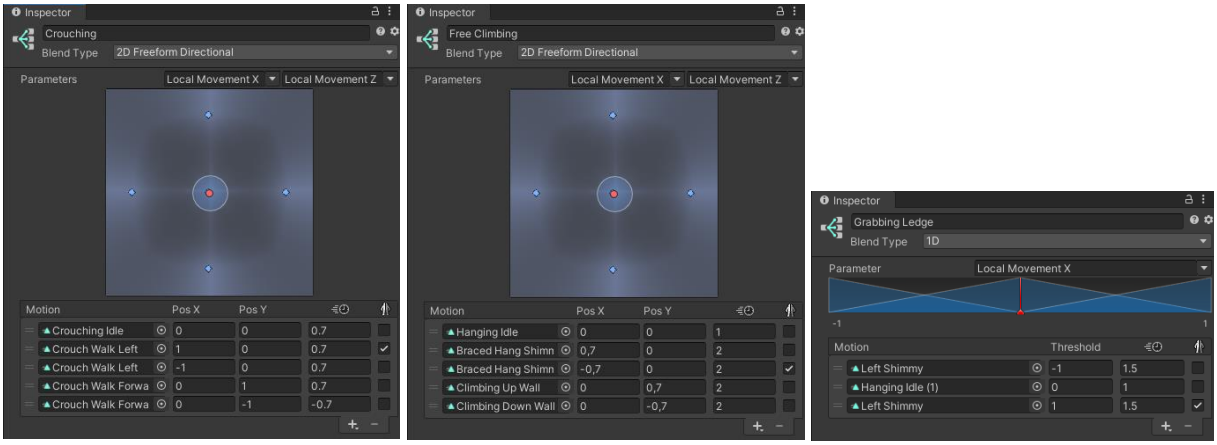
1.2.2 Adding animations

Attach Unity's Animator component to your character game object and assign the animator controller "Character" in folder *AnimatorControllers*.

Now, you can change the animations of the corresponding animation state or animation blend tree depending on your needs. The asset comes with animations from Unity's 3D Game Kit, user Naked Fighter and some from Blink Studio's "FREE - 32 RPG Animations" package. Please check out their great work here! Besides that, there are 8 animations from Mixamo in the provided animator I am unfortunately not allowed to ship with my asset. Nevertheless, it is possible to download them for free on their website if you want to use them in your project. Here are the links:

a. Crouch Idle 01
b. Crouch Walk Forward* (here the archer animation)
c. Hanging Idle
d. Braced Hang Shimmy
e. Climbing Up Wall
f. Climbing Down Wall
g. Hanging Idle (1)
h. Left Shimmy

And here are screenshots how they are properly set up in the Character animator:



* Animation "Crouch Walk Left" is just a copy of "Crouch Walk Forward" with 90° Root Transform Rotation offset.

1.2.3 Assigning the right layer

Make sure that your character game object and all of its children have the "**Player**" layer assigned. Otherwise, the provided motor scripts might not behave properly. However, if you definitely need to have child objects assigned to other layers, use the provided layer mask

"**Ignored Layers**" of the RPGMotor to rule them out. This way, they are not considered for the grounded check, sliding check, etc.

For setting up climbing or ledge grabbing, assign a layer of your choice to the climbable collider, e.g. an own "Climbable" layer, and make it one of the "**Climbable Layers**" in the RPGMotor component.

If you are using one of the RPGCamera scripts, make sure to check the "**Occluding Layers**" variable of the also attached RPGViewFrustum component. Every game object in the scene in/with one of these layers causes the camera to automatically zoom in on occlusion! Refer to the corresponding section 2.2 RPG View Frustum below for more information on the implemented logic.

1.2.4 Moving platforms (optional)

If you want the character to move and rotate with ground objects like platforms, you need to assign the MovingPlatform script to them. Furthermore, a box collider which acts as a trigger is required. The trigger is used for detecting new or left passengers. I recommend checking out the provided "Moving Platform" prefabs which are also used in the demo scenes. As you know, a picture is worth a thousand words.

1.2.5 Water and swimming (optional)

For leveraging the swimming feature of the RPGMotor, three things have to be considered:

1. You need a water game object which has the "Water" script/component assigned
2. This water game object must have a box collider that acts as a trigger attached
3. The variable "Swimming Start Height" of the RPGMotor controls at which local height the character should start to swim (visualized by a small blue plane when gizmos are enabled)

Check out the prefab "Water" in folder *Prefabs* for reference.

1.2.6 Using the correct shader (optional)

To make the fade out of game objects work, they need to have a material with a Fade/Transparent shader assigned, e.g. Unity's standard shader with render mode "Fade" or one of the provided custom shaders in folder *Misc > Shaders*. Just check out which shaders are used in the provided demo scenes.

1.3 Integrations to other assets

This asset features some integrations to other assets which are available in Unity's Asset Store. Find out in their corresponding sections below how integration can be accomplished.

1.3.1 RPGBuilder

This asset provides an integration to Blink Studios' award-winning RPGBuilder. For setup instructions, please browse to *Integration > RPGBuilder* and go through the contained README file. There is also an integration setup video which guides through the steps on YouTube.

1.3.2 Mirror (or similar networking libraries)

Steps for simple Mirror integration – or other networking libraries which work similarly – are also provided. Unfortunately, there are some manual code corrections which have to be done by you as these are hard to provide out-of-the-box due to maintainability. But there are not many and they are all documented in a PDF document inside the folder *Integration > Mirror*.

2 Variable overview and explanation

In this chapter, some public script variables and their functionality are explained as their names may not be self-explanatory. For the others, there are tooltips if you hover over their names in the inspector. Besides that, my code is completely commented, so you can jump into the scripts and check out all descriptions there.

2.1 RPG Camera

Camera To Use

Use this variable to set up which camera game object should be used in the scene:

- "Main Camera" will use the camera object which is tagged with "MainCamera"
- "Spawn Own Camera" will spawn a new camera object on start
- "Assigned Camera" will use the camera you assign to variable "Camera To Use"

Used Skybox

Skybox which is currently used or should be used by the camera object. The skybox can be changed at runtime by calling the RPGCamera script's method "SetUsedSkybox(material)". Direct assignments to this variable have no effect.

Camera Pivot Local Position

This is the local position of the camera pivot, the "anchor" of the virtual camera. Turn on the script's Gizmos to visualize it as a small cyan sphere (see on the right).

Internal and external Pivot

The RPGCamera also supports external pivots, i.e. pivots that are not within the character collider. When an external pivot is recognized, pivot occlusion and object fading between the character and the pivot is automatically turned on.

For internal pivots, there is additionally the possibility to enable the **Intelligent Pivot** that moves away from obstacles which the player could see through if zooming in enough, especially when using a wide screen.

Always Activate Orbiting

If set to true, you do not have to press any input to control/rotate the camera.

Align When Moving
If set to true, the camera view direction aligns with the character's view/walking direction when the character starts to move (forward/backwards/strafe). If additionally **Support Walking Backwards** is set to true, the camera faces the front of the character when walking backwards.

Cursor Behavior Orbiting
Sets the cursor behavior during camera rotation/orbiting.

- "Move" – The cursor continues to move as usual.
- "Move Confined" – The cursor continues to move but is confined to the game window. According to Unity's documentation, only this mode is only "supported on the standalone player platform on Windows and Linux".
- "Stay" – Before orbiting, the cursor position is memorized and restored after the orbiting ends. In combination with **HideCursor** = "When Orbiting", it looks like the cursor stayed where it was before starting the orbiting.
- "Lock in Center" – The cursor jumps to the center of the screen and stays there as long as orbiting is active. Good for games where cursor controls are not foreseen, e.g. on consoles.

Underwater Threshold Tuning
Gives the possibility to fine-tune the threshold where the camera is seen as underwater. The higher the value, the earlier the underwater effects kick in. The effects are enabled/disabled via methods "EnableUnderwaterEffects()" and "DisableUnderwaterEffects()", respectively. You can override these two methods to implement your own visual effects.

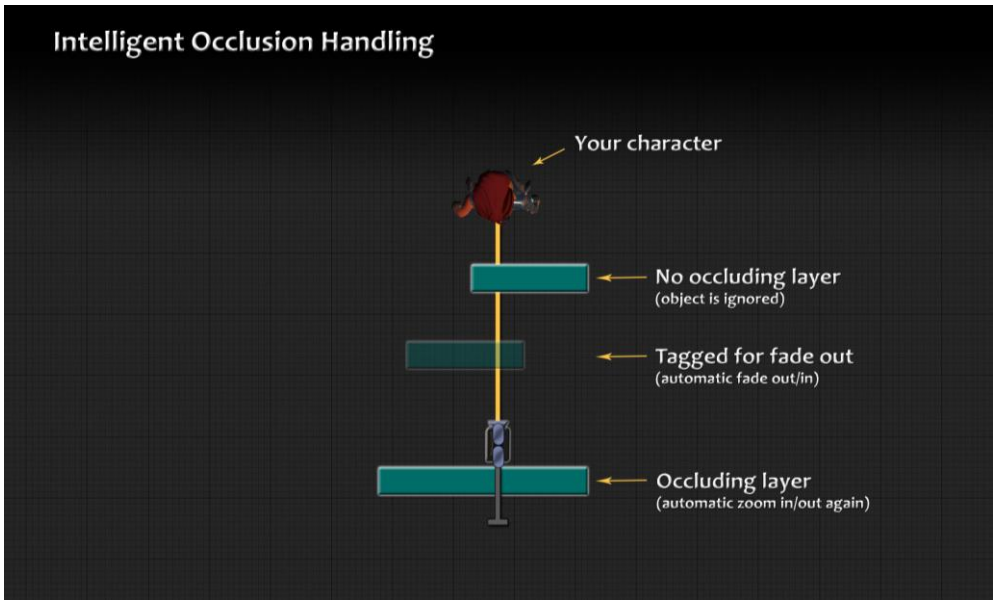
2.2 RPG View Frustum

Frustum Shape and Rays per Edge
Since v5.1, there is the possibility to change the shape of the view frustum, i.e. to influence the area which is tested for camera occlusion. While the cuboid shape checks for a consistent frustum plane from pivot to camera, the pyramid shape (default) has only a single frustum plane at the camera, forming a pyramid with its peak at the pivot. You can also see the difference in play mode if you have Gizmos turned on.

The pyramid shape is more forgiving but less performant (additionally influenced by the number of cast rays). Example: If you closely move the character around a pillar object, the cuboid frustum would cause a zoom into first person, whereas the pyramid frustum would not do anything.

Occluding Layers and Tags/Layers/Component For Fade Out
The occluding layers determine which game objects in the scene are processed by the camera view frustum. By default, every game object which has one of these layers assigned causes an automatic camera zoom in when it occludes the view to the character.

Objects which should be faded out in this case instead of causing a zoom in can either be tagged with one of the “Tags For Fading” or assigned to a layer of layer mask “Layers For Fading”. From v5.3 on, you can also choose the provided script/component “FadeOut” as a criterion for fading objects.



Tags/Layers/Component Causing Look Up
Game objects in the scene which are tagged with this tag (or assigned to this layer mask) are triggering the camera look up feature whenever the camera starts lying on them. From v5.3 on, you can also choose the provided script/component “FadeOut” as a criterion for fading objects.

Fade Out Alpha
The alpha to which objects fade out when they enter the view frustum.

Fade In Alpha
The alpha to which objects fade back in after they left the view frustum.

2.3 RPG Motor scripts

Camera Controlled 3D Movement (MMO only)
If set to true, character 3D movement, e.g. diving or surfacing, is only possible when the character should align/rotate with the camera. Otherwise, the character will always move into viewing direction irrespective of alignment input.

Align With Camera
Determines when the character’s view/walking direction should be aligned to the camera’s view direction.

Allowed Airborne Moves
The character is allowed to move slightly while airborne after performing a standing jump. This variable saves the maximum number of allowed airborne moves.

Allowed Midair Jumps
Only visible when **EnableMidairJumps** is set to true. Determines the number of additional jumps while in midair.

Climbing setup
In the RPGMotor, two different types of climbing are distinguished: **Ledge Grabbing** and **Free Climbing**. Both functionalities can be independently turned on and off. Ledge Grabbing enables the character to hold on ledges, move along them and climb them up if possible. Free Climbing basically covers the rest: climbing freely in all directions along a wall or generally any collider, without the possibility to automatically grab or climb up ledges.

For setting up one or both of them, I recommend turning on variable “Draw Check Gizmos”. As a result, gizmos are drawn that show which areas are checked for the hands and feet, i.e. their position (hands/feet height), the grab range between them, hands/feet size (check radius) and grab distance (how far away a ledge can be to grab it). Additionally, there are variables for setting the maximum angle a ledge or a collider can have to be considered climbable, and for specifying a cooldown after climbing was finished/canceled. Also do not forget to set assign climbable objects to a certain layer which is also part of the “Climbable Layers” layer mask in the inspector.

Falling Threshold
A value representing the degree at which the character starts to fall. The default value is “6” to let the character be grounded when walking down small hills.

Grounded Tolerance
Tolerance height used for the grounded check. The larger the value, the larger the distance to the ground which enables grounded character behavior to true. Useful for tweaking character movement on debris.

Sliding Timeout
Generally, sliding occurs whenever the terrain/ground is steeper than the “Slope Limit” given in the Character Controller component. This variable gives the time in seconds which has to pass before the sliding logic kicks in.

Swimming Start Height
Defines the local water height at which the character should start to swim. Enable Gizmos for easier setup.

3 Useful interfaces
Below you find interfaces that you can implement with your own scripts. Note that these implementing classes/components have to be assigned to the same object as my RPG camera or controller scripts which should use them.

3.1 IPointerInfo
This interface is used by the RPG Camera. It provides the method “IsPointerOverGUI” for checking if the cursor is over an UI element and – as a result – deactivates all camera input

logic. Since this asset does not provide an own UI, feel free to implement this interface in your own code to seamlessly integrate my camera logic.

3.2 IPlayer

This interface is mainly used by the RPG Motor. It provides methods such as “CanMove”, “CanRotate” or “CanFly”. As the names suggest, implement these methods for applying movement or rotation impairing effects on the character or enabling/disabling flying or a target lock. Refer to the provided “RPGPlayerExample” script inside the *Character* folder for an example interface implementation.

3.3 ITransportable

Implement this interface to make an object transportable for the Moving Platform (see the RPGMotor script for an example implementation).

3.4 IRPGCamera, IRPGViewFrustum, IRPGController, IRPGMotor

For completeness, there are also interfaces provided for each of the character scripts. So feel free to easily replace one of my components by implementing the corresponding interface yourself.

4 Version history

If you are interested in the release notes, please refer to my website johnstairs.com/rcc.

5 Got feedback, questions or problems?

Feedback, feature suggestions and asset reviews are highly appreciated!

In case of question which are not covered by the FAQs on johnstairs.com/rcc, feel free to send me an email to mail@johnstairs.com. If you are facing issues, please attach at least a screenshot showing your scene, the used variable values and error messages which might occur in the console.

Best regards,

John Stairs