

## Physics Based Character Controller

Support email: [nappin.1bit@gmail.com](mailto:nappin.1bit@gmail.com)

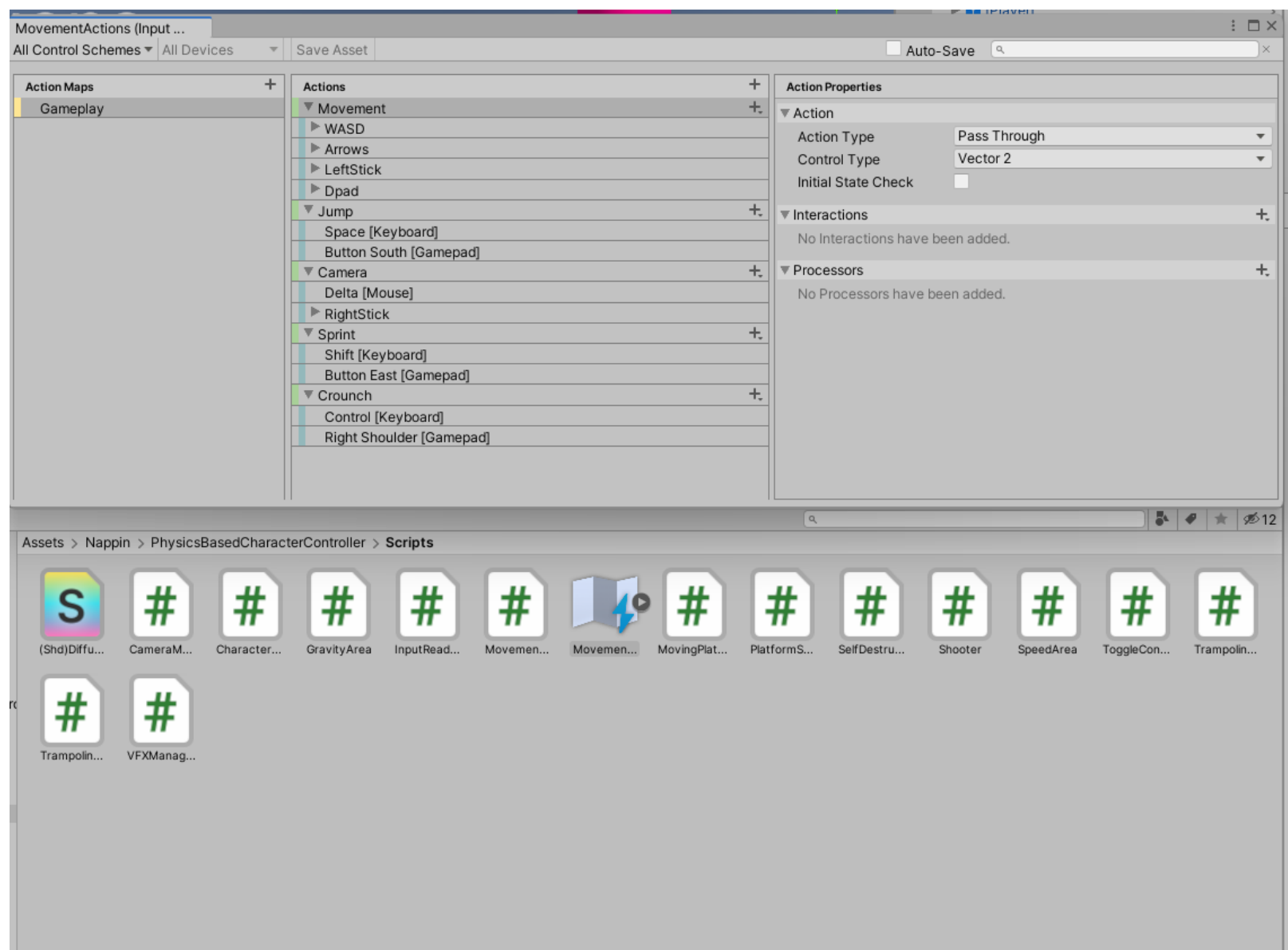
# Table of Contents

Asset Content	<a href="#">3</a>
Inputs and Scripts	<a href="#">3</a>
Prefabs and Essentials	<a href="#">3</a>
External Packages	<a href="#">4</a>
Setup the Input	<a href="#">4</a>
What is the [Input System]?	<a href="#">4</a>
Setup the NewInputSystem	<a href="#">5</a>
Setup the OldInputSystem	<a href="#">6</a>
Example Camera Setup	<a href="#">8</a>
Do I need it?	<a href="#">8</a>
Setup the Camera Manager	<a href="#">9</a>
Setup the Toggle Controller	<a href="#">9</a>
Setup the Player	<a href="#">10</a>
Collisions	<a href="#">10</a>
Movement	<a href="#">10</a>
Jump and gravity	<a href="#">11</a>
Slope and steps	<a href="#">12</a>
Wall Slide	<a href="#">12</a>
Sprint and crouch	<a href="#">13</a>
References	<a href="#">13</a>
Platform extensions	<a href="#">14</a>
Moving platforms	<a href="#">15</a>
Trampoline platform	<a href="#">15</a>
Speed area	<a href="#">15</a>
Gravity area	
Contact	<a href="#">16</a>

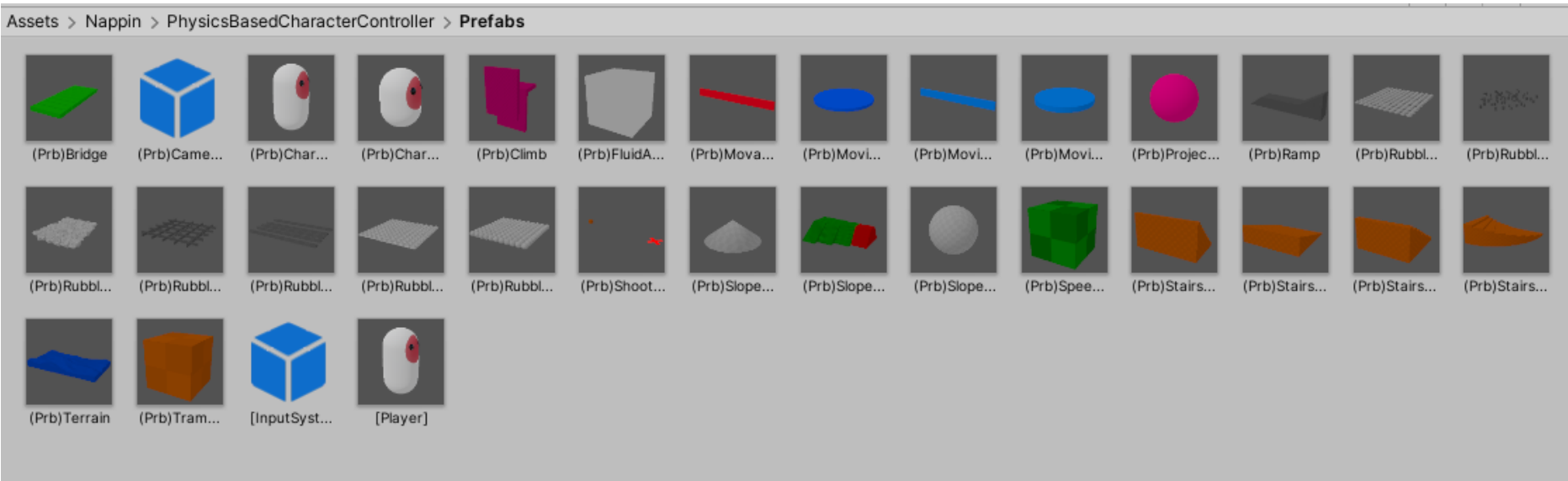
# Asset content

The asset contains multiple prefabs of platforms / fluids / areas to kickstart your project and to experiment with the character mechanics. The core content of the asset can be found in the *Prefabs* folder and in the *Scripts* folder.

In the *Scripts* folder you can find the example **InputAction** used to move the character called *MovementAction* and inside it a setup to operate the asset with both mouse and keyboard and a gamepad. If you intend to use the **OldInputSystem** click [here](#) to learn how you can set it up



In the prefab folder you can find platforms, fluids, areas and the default character model with the prefix (*Prb*). Here you will find the 2 **essential components** to use the asset inside the square brackets: **[InputSystem]** and **[Player]**



The **[InputSystem]** contains a script called **InputReader** that gets the input from the old or new input system and converts it into usable data for the **[Player]**.

The **[Player]** contains a script called **CharacterManager** that uses the input data and moves the character.

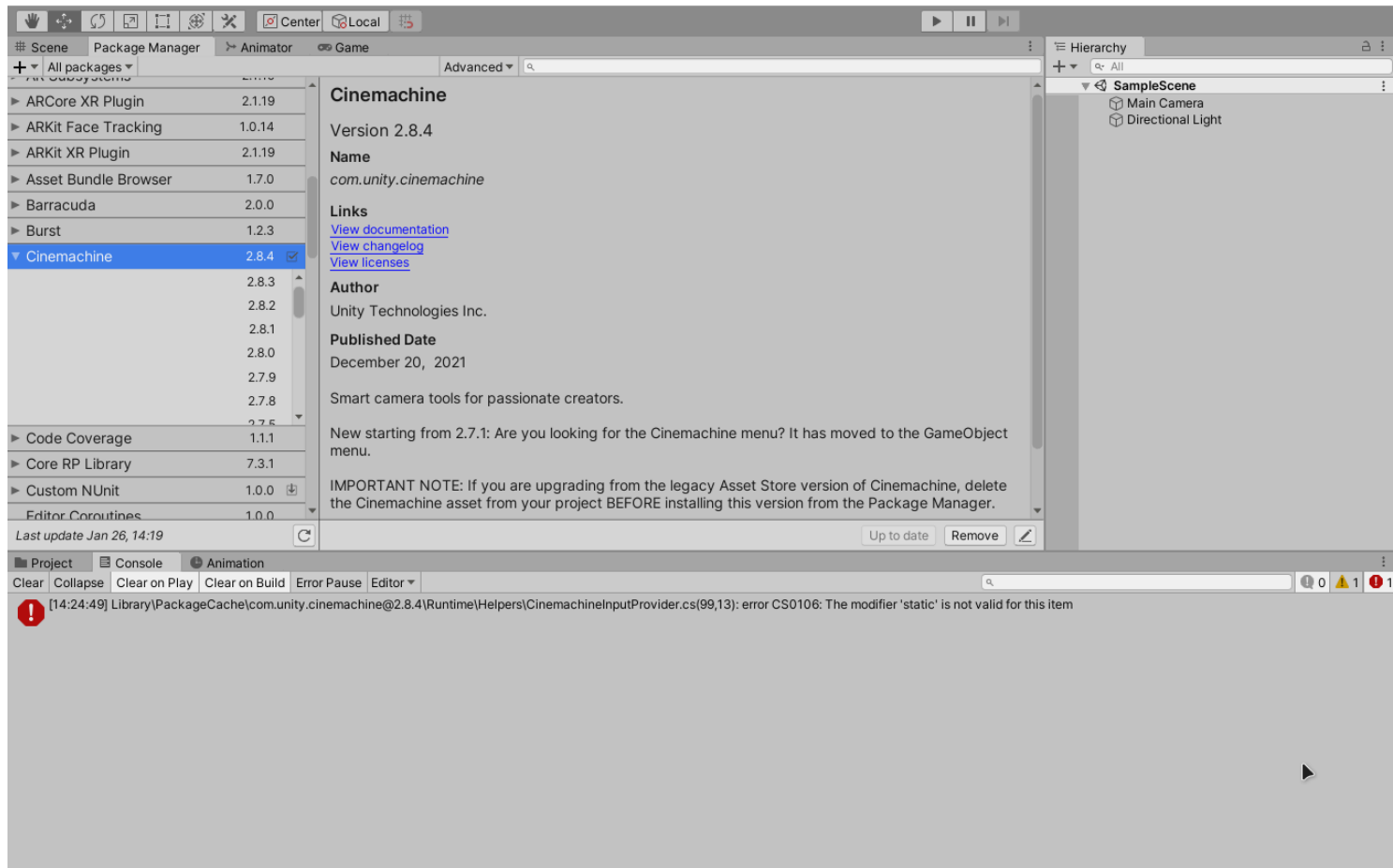
This two prefabs are essential to operate the asset and are required to move the **[Player]**

## External Packages

The asset doesn't require any packages if you intend to use the **OldInputSystem**.

The only asset **required** if you intend to use the **NewInputSystem** is the *NewInputSystem Package* that you can find in the package manager.

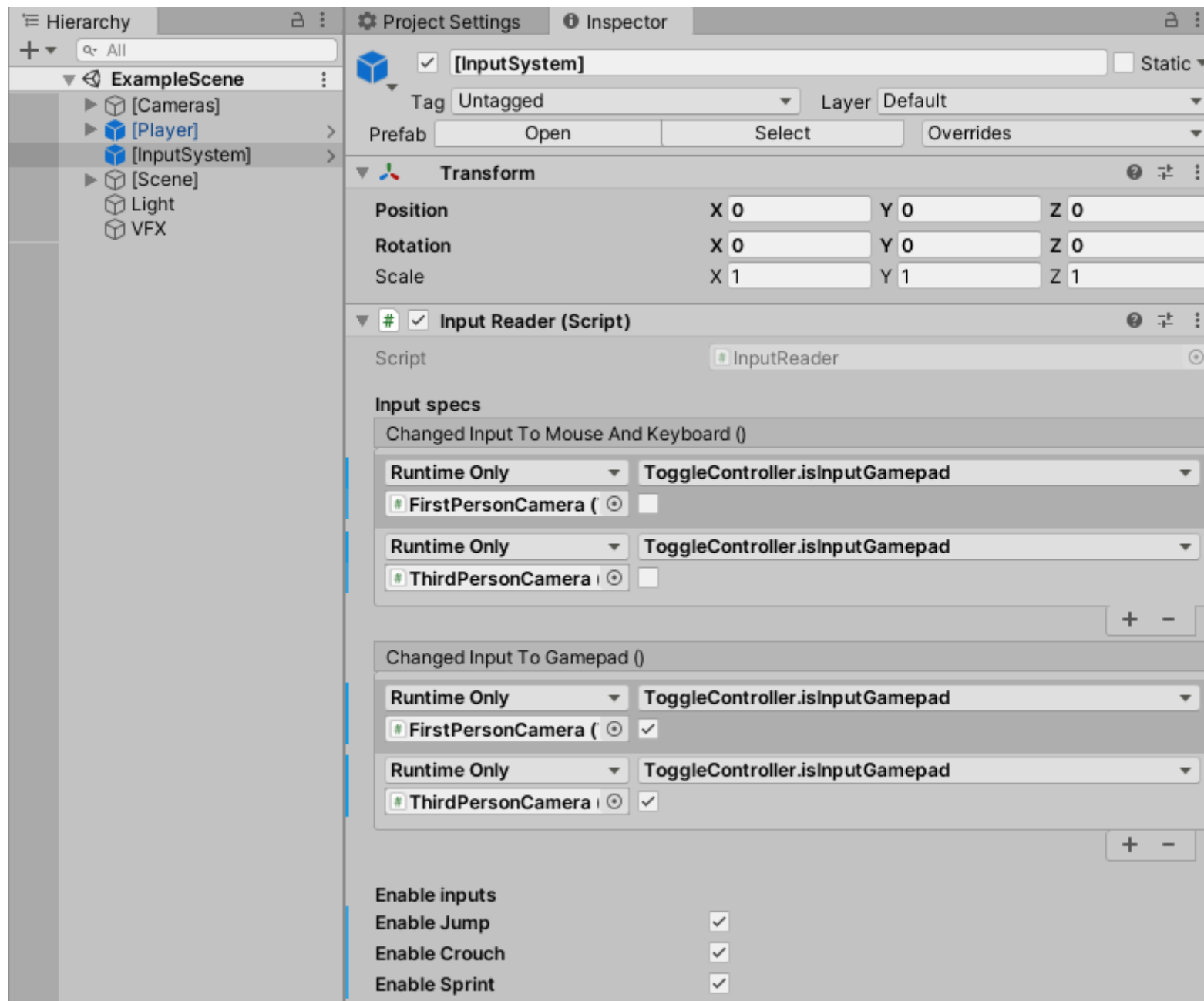
The example scene uses a simple **Cinemachine** setup to operate in first person and in third person. Even though the package is not required it's recommended especially if you want to explore the *ExampleScene*.



N.B. Cinemachine has some issues when it comes to using the NewInputSystem in the unity version 2019.4.2. If you get the error of the image above in the script CinemachineInputProvider.cs it's recommended to use Cinemachine version 2.3.3. This has nothing to do with this asset specifically but with Cinemachine integration of the NewInputSystem.

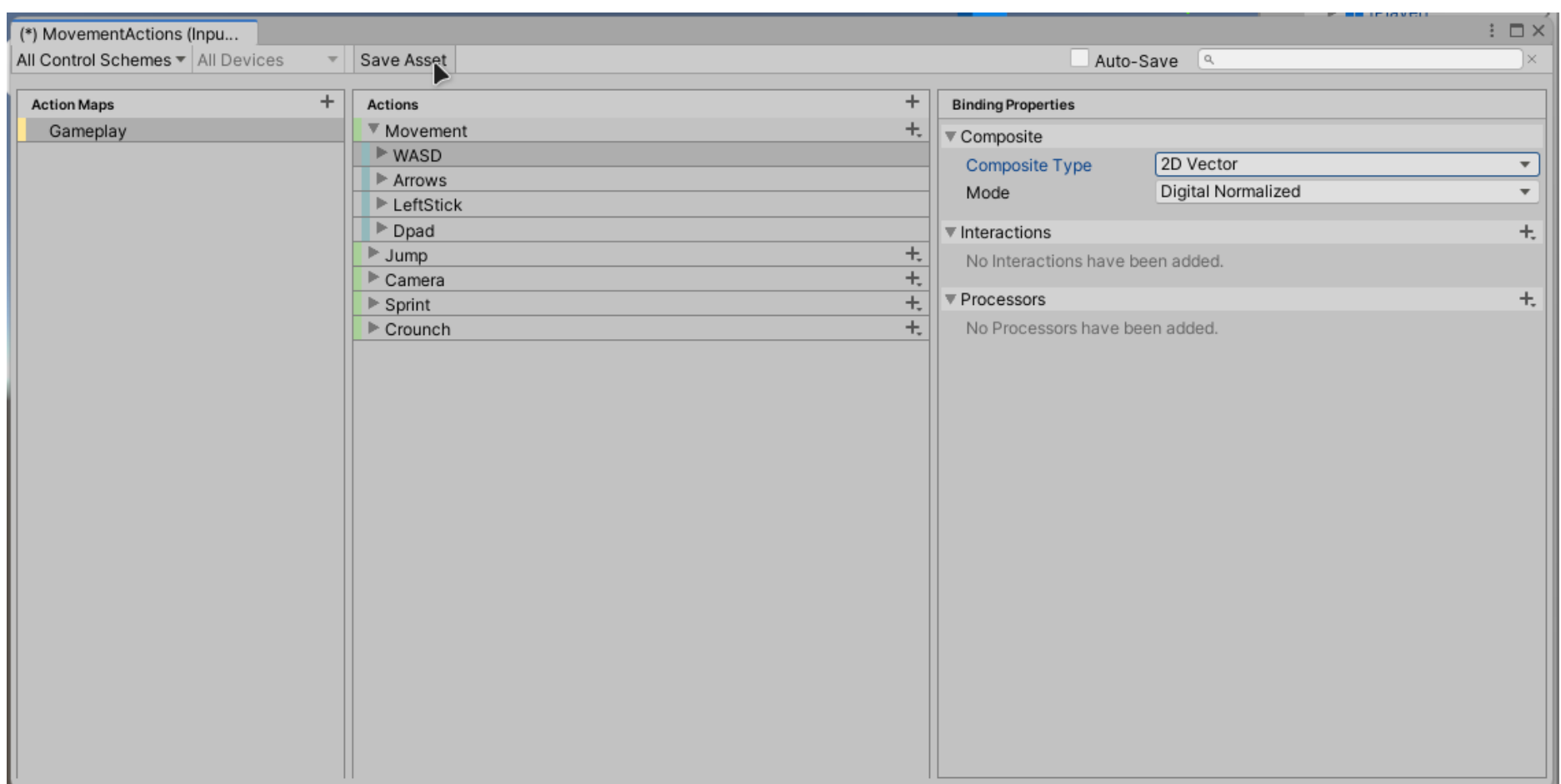
## Setup the Input

The script **InputReader** allows you to enable / disable jump, crouch and sprint before launching the editor and at runtime. It also allows you to call specific events when the input changes from mouse and keyboard to gamepad or from gamepad to mouse and keyboard. This feature is really useful when you want to change the input icons of your UI based on the device that your player is using.



## Using the New Input System

You can easily change the type of input by adding content to the **MovementAction**. Remember that the **InputReader** script uses the action names so if you want to add new actions or rename the current one, you need to update the script as well.



P.S. This might be obvious but remember to save your InputActions since the auto-save option is disabled by default

## Using the Old Input System

The asset uses the new input system by default but you can easily edit it to use the old one, here is how you do it:

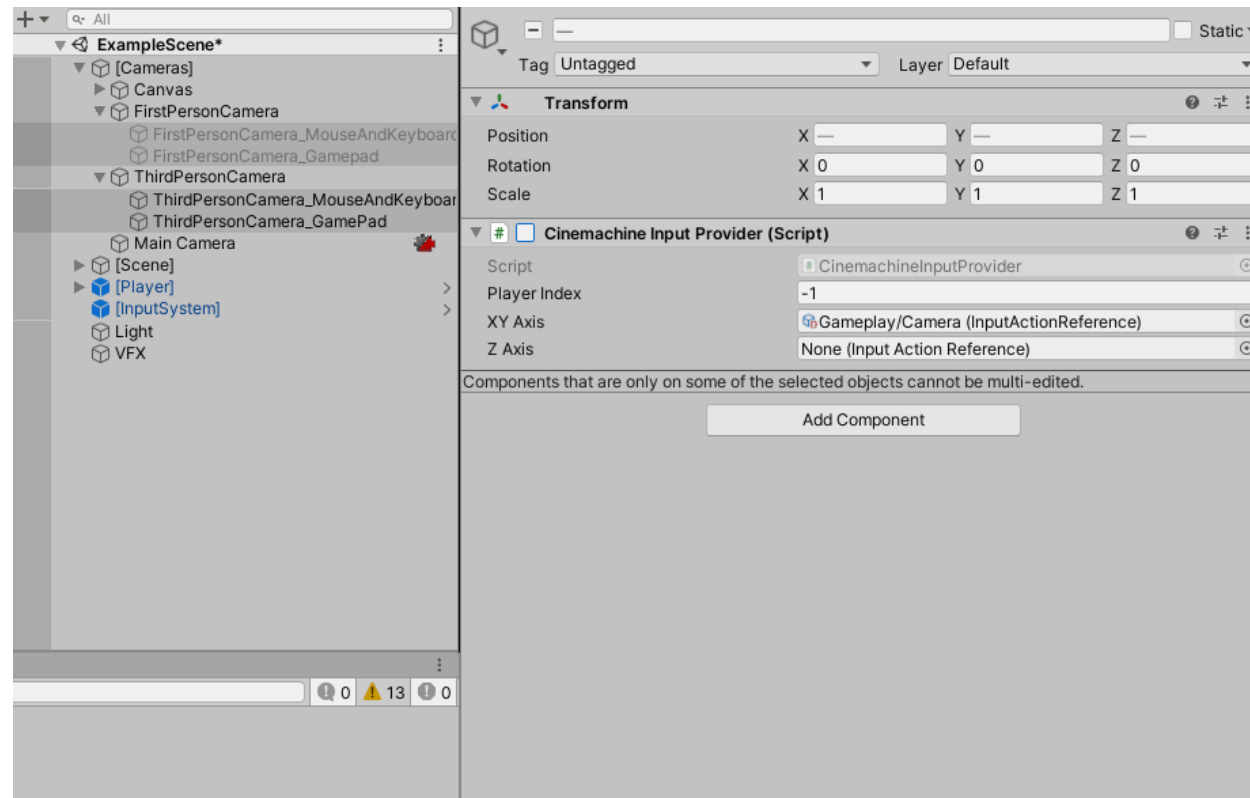
- Open the **InputReader** script, activate the collapsed script and remove everything else. The result is the following:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Events;
5
6
7  public class InputReader : MonoBehaviour
8  {
9      [Header("Input specs")]
10     public UnityEvent changedInputToMouseAndKeyboard;
11     public UnityEvent changedInputToGamepad;
12
13     [HideInInspector]
14     public Vector2 axisInput;
15     [HideInInspector]
16     public Vector2 cameraInput;
17     [HideInInspector]
18     public bool jump;
19     [HideInInspector]
20     public bool jumpHold;
21     [HideInInspector]
22     public float zoom;
23     [HideInInspector]
24     public bool sprint;
25     [HideInInspector]
26     public bool crouch;
27
28     private bool hasJumped = false;
29     private bool skippedFrame = false;
30     private bool isMouseAndKeyboard = true;
31     private bool oldInput = true;
32
33     /**/
34
35     //old input system
36     @ Unity Message | 0 references
37     private void Update()
38     {
39         axisInput = new Vector3(Input.GetAxisRaw("Horizontal"), 0f, Input.GetAxisRaw("Vertical")).normalized;
40         jump = Input.GetButtonDown("Jump");
41         jumpHold = Input.GetButton("Jump");
42         sprint = Input.GetKeyDown(KeyCode.LeftShift);
43         crouch = Input.GetKeyDown(KeyCode.LeftControl);
44
45         GetDeviceOld();
46     }
47
48     1 reference
49     private void GetDeviceOld()
50     {
51         //get device input
52         oldInput = isMouseAndKeyboard;
53
54         if (Input.GetJoystickNames().Length > 0) isMouseAndKeyboard = false;
55         else isMouseAndKeyboard = true;
56
57         if (oldInput != isMouseAndKeyboard && isMouseAndKeyboard) changedInputToMouseAndKeyboard.Invoke();
58         else if (oldInput != isMouseAndKeyboard && !isMouseAndKeyboard) changedInputToGamepad.Invoke();
59     }
60 }
```

- If you intend to use the camera setup provided in the sample scene then you need to edit the **CameraManager** script. Here activate the collapsed text in the update function. The result is the following in the update function:

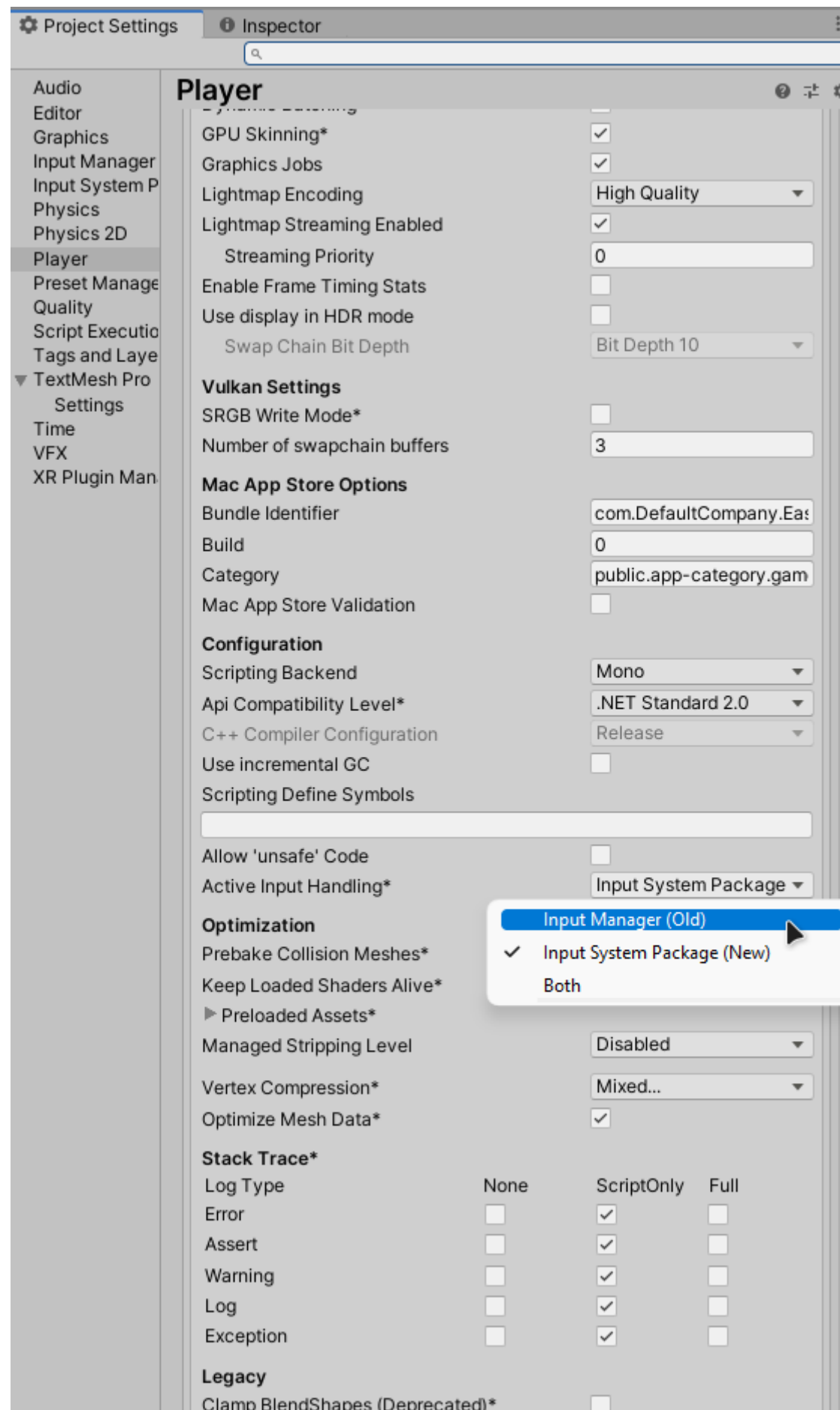
```
31
32
33  @ Unity Message | 0 references
34  private void Update()
35  {
36      if (Input.GetKeyDown(KeyCode.M))
37      {
38          activeThirdPerson = !activeThirdPerson;
39          SetCamera();
40      }
41
42      if (Input.GetKeyDown(KeyCode.N))
43      {
44          SetDebug();
45      }
46  }
```

- If you intend to use the camera setup provided in the sample scene then you need to disable the **CinemachineInputProvider** in the **Cinemachine** cameras.



- Remove the New Input System from the project and enable the Old Input System in the **Project Settings** and remove the **MovementActions** script from the project or whatever **InputAction** you created





You don't need to execute this steps in order but they are all needed if you want to change the InputSystem used

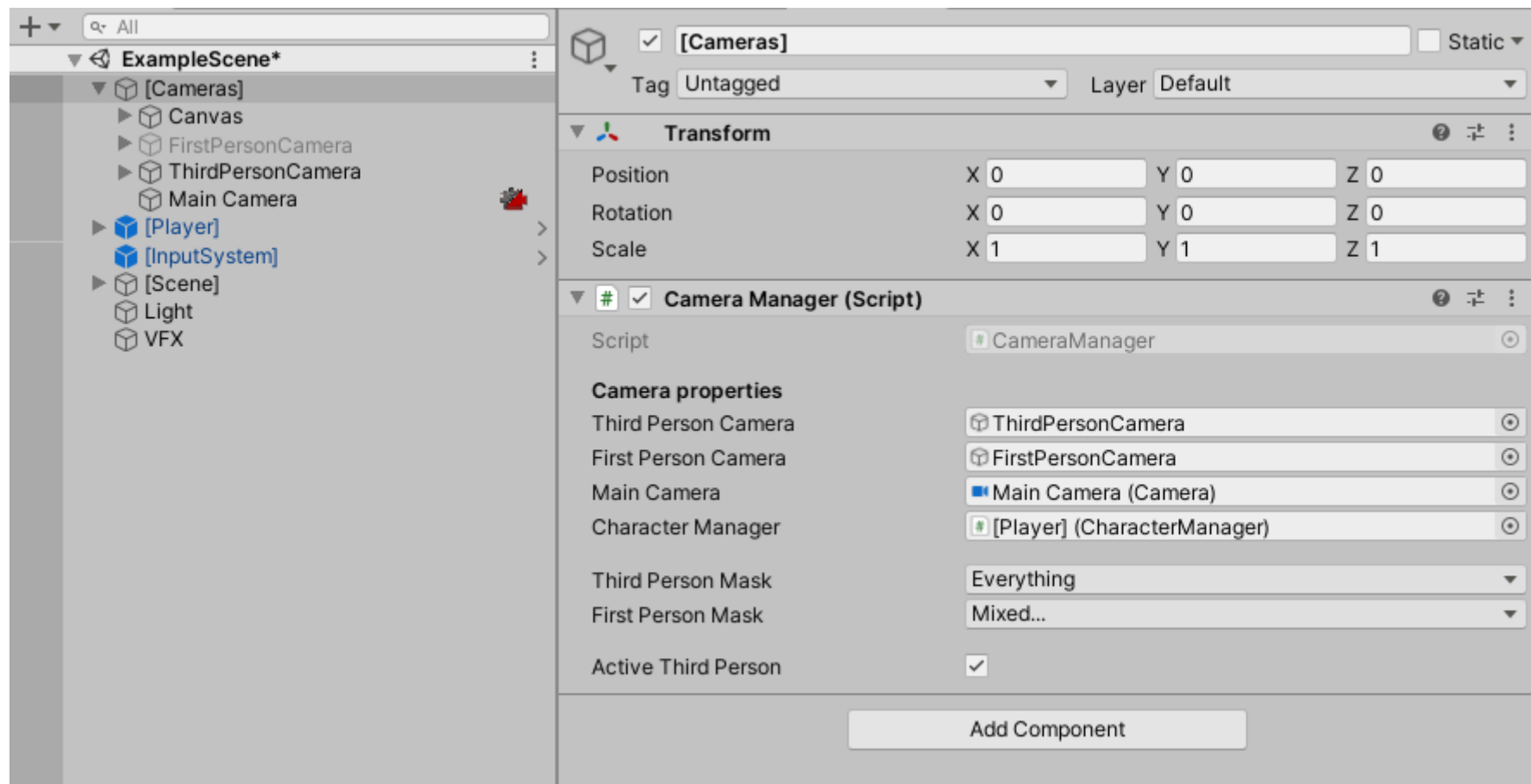
## Setup the Camera

The asset works simply by using the default MainCamera without any change. You can also use your own custom camera system. If you intend to use the camera setup included in the asset here are its features:

- It uses **Cinemachine** so it's highly editable
- It uses 4 different cameras: *FirstPerson* gamepad camera, *FirstPerson* keyboard and mouse camera, *ThirdPerson* gamepad camera, *ThirdPerson* keyboard and mouse camera
- Functions can be called from the **[InputSystem]** events to swap the camera used



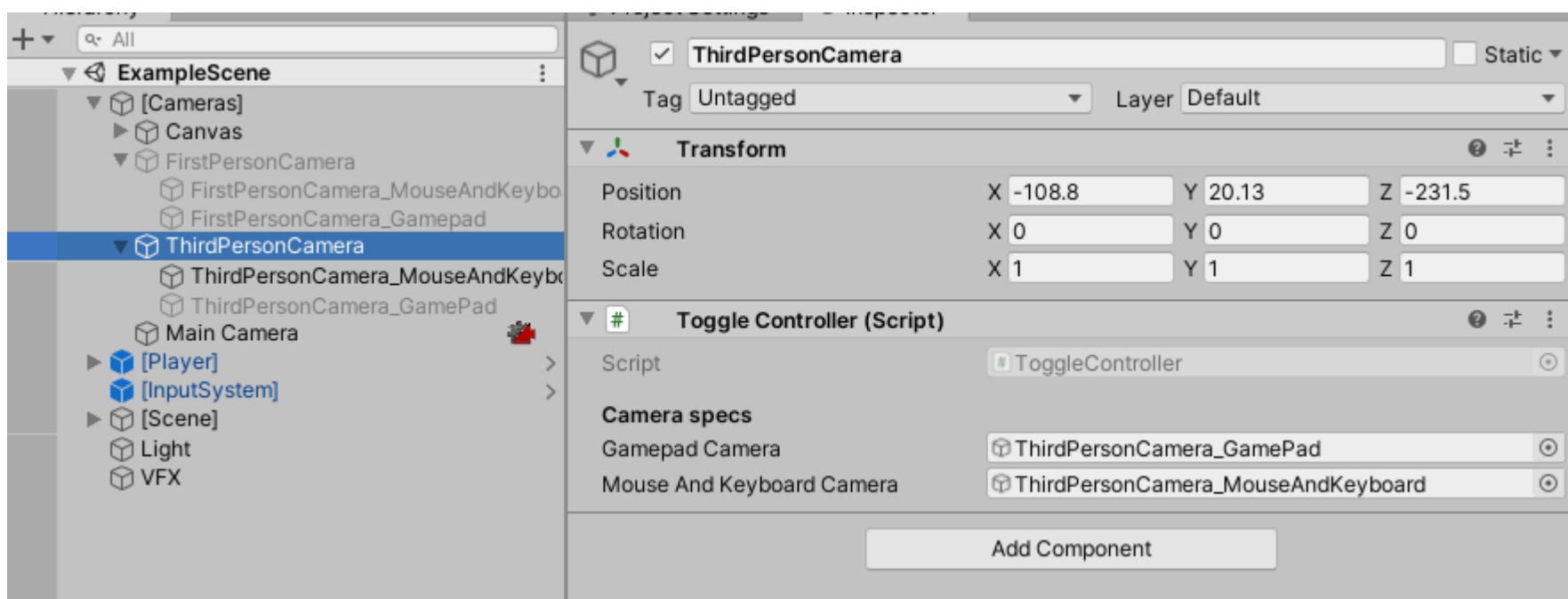
The brain of the setup is the **CameraManager**:



It's variables are

- The *ThirdPersonCamera* gameobject parent
- The *FirstPersonCamera* gameobject parent
- The *MainCamera*
- The **[Player]**
- The mask used when in third person (the layers that the camera can render)
- The mask used when in first person (the layers that the camera can render)
- If the default camera when the game is launched is the third person

The *FirstPersonCamera* gameobject parent and the *ThirdPersonCamera* gameobject parent contain a simple toggle script called by the **[InputSystem]** that enables or disables specific cameras



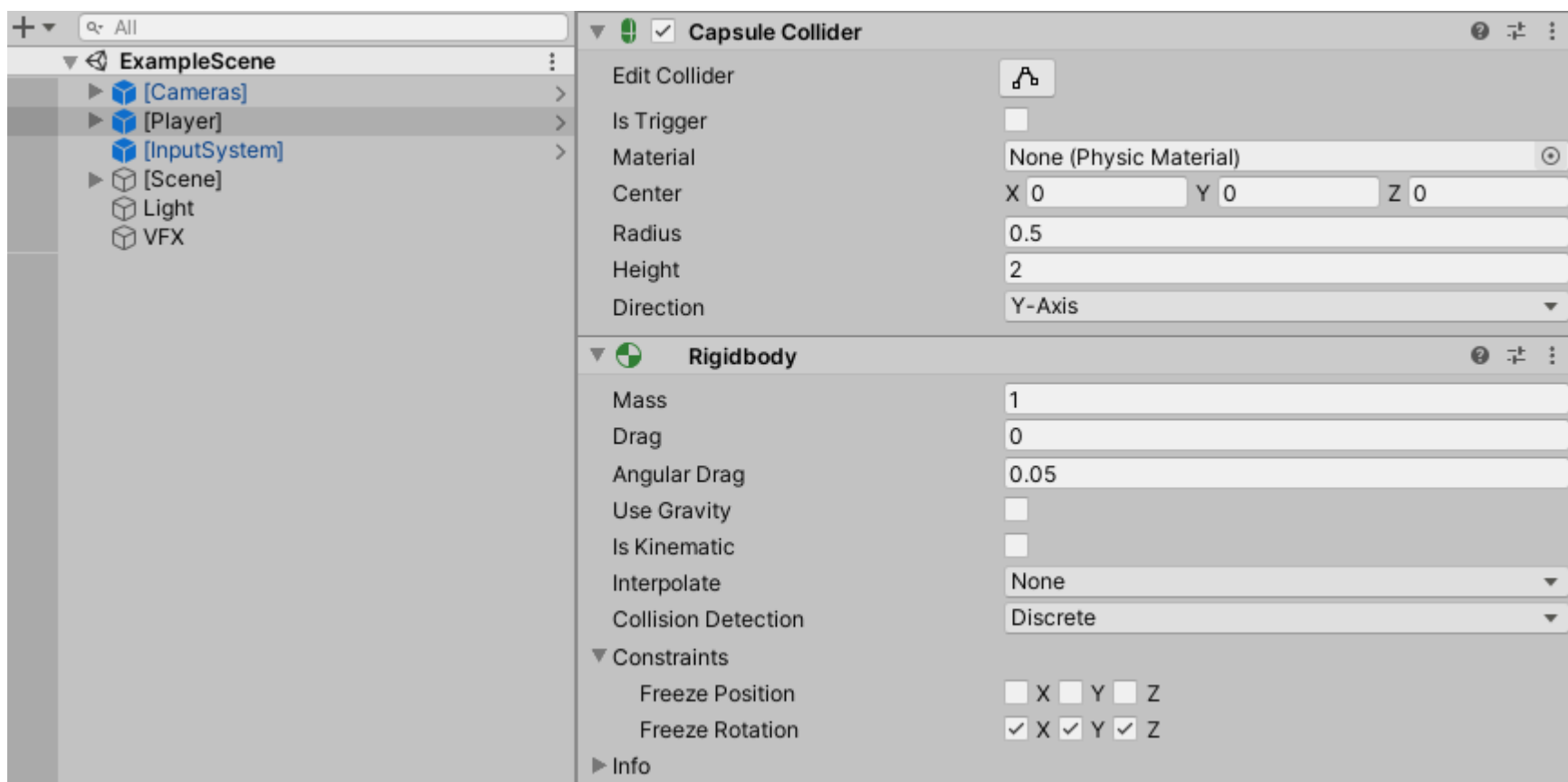
## Setup the Player

Let's dive into how to setup the [Player] and how to handle its collisions. The most important part of the setup is [referencing](#) the corrects scripts / gameobjects

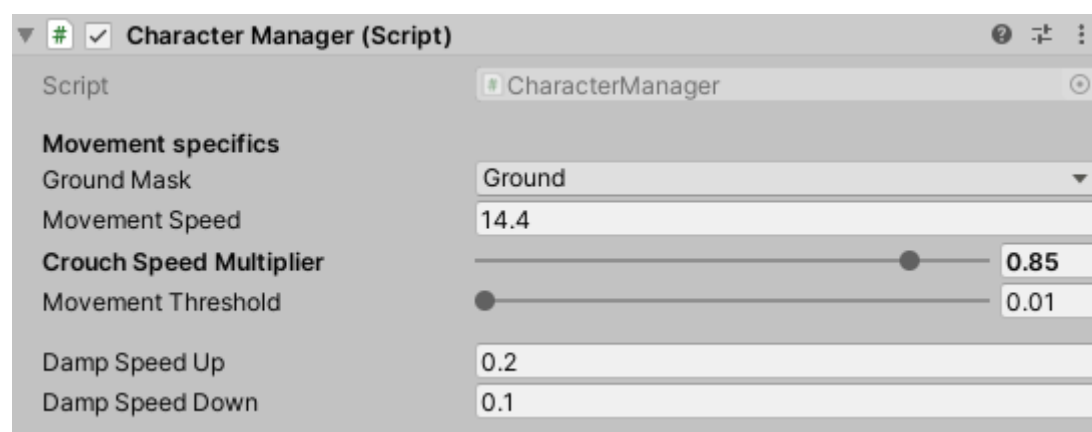
All the variables in the **CharacterManager** script have a description, you can read it just by hovering over the variable.

## Setup the collisions

The collision setup is quite simple, it **requires** a capsule collider and a rigidbody. The rigidbody doesn't have gravity enabled (it's applied via code by the **CharacterManager**) and its rotation is frozen



## Movement specifics



Its variables are:

- **Ground mask:** the layers that collide with the player
- **Movement speed**
- **Crouch speed multiplier:** if the crouch is enabled how slow does the character move (i.e. if the value is 0.5 the character moves at half the speed when crouching)
- **Movement threshold:** what's the minimum input that allows the character to move
- **Damp speed up / Damp speed down:** how fast does the player slow down or speed up

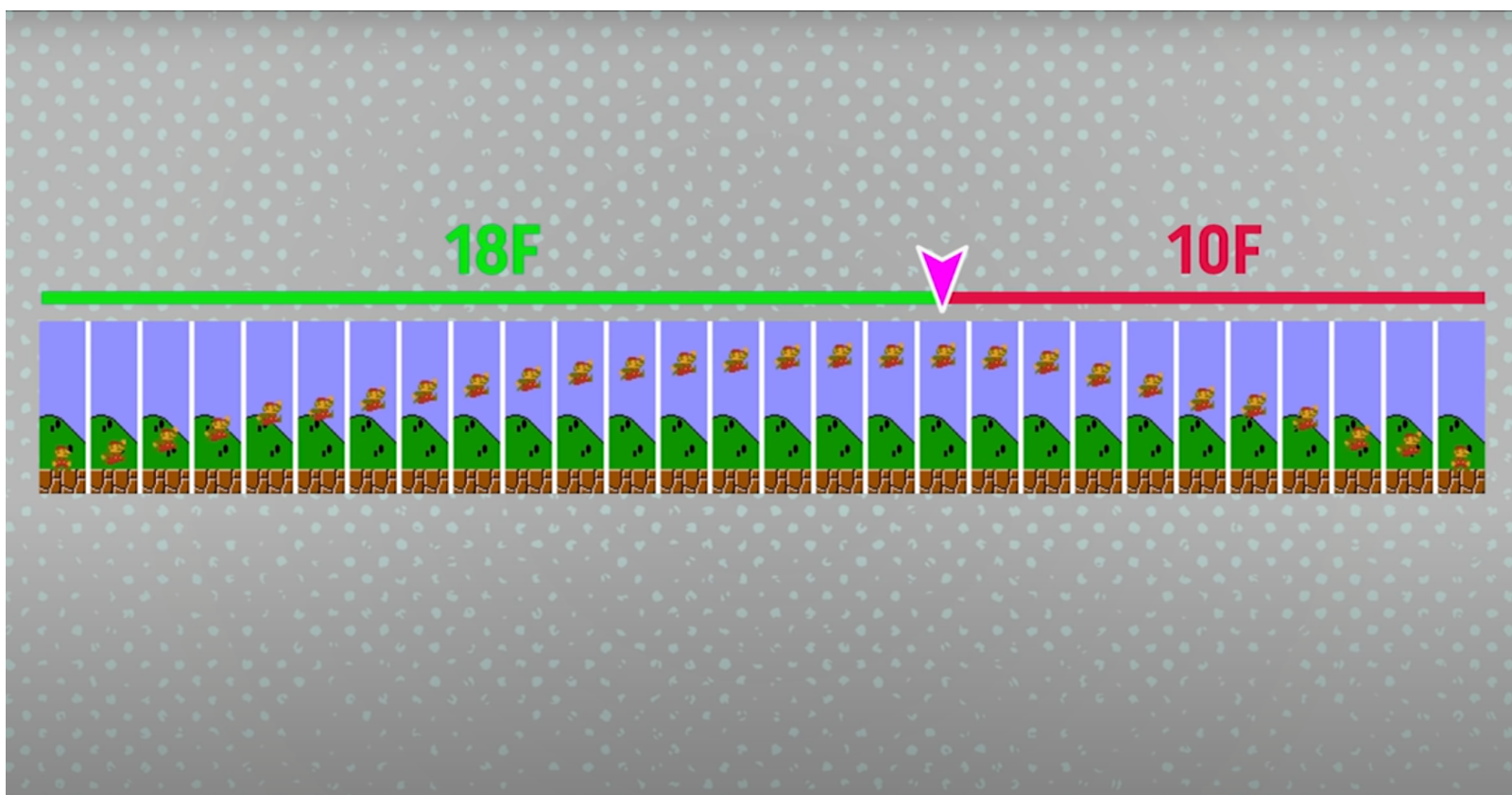
N.B. The GroundMask is essential to allow the player to be grounded. It's also recommended to setup the "Physics" to enable the capsule to collide with whatever you want

## ***Jump and gravity specifics***

Jump and gravity specifics	
Jump Velocity	20
Fall Multiplier	1.7
Hold Jump Multiplier	5
Friction Against Floor	<input type="range"/> 0.189
Friction Against Wall	<input type="range"/> 0.082
Can Long Jump	<input checked="" type="checkbox"/>

Its variables are:

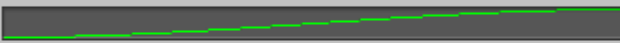
- **Jump velocity**
- **Fall multiplier:** if the value is left at 1 the gravity is always the same. This value allows you to achieve a more exaggerated and cartoony jump making the movement more intense when falling



Source: BoardToBits

- **Friction against floor**
- **Friction against wall**
- **Can long Jump:** if enabled allows the player to stay longer in the air and traverse a bigger distance when holding the jump button

## ***Slope and steps specifics***

<b>Slope and step specifics</b>	
Ground Checker Threshold	0.1
Slope Checker Threshold	0.51
Step Checker Threshold	0.6
Max Climbable Slope Angle	53.6
Max Step Height	0.74
Speed Multiplier On Angle	
Can Slide Multiplier Curve	0.061
Can't Slide Multiplier Curve	0.039
Climbing Stairs Multiplier Curve	0.086
Gravity Multiplier	8
Gravity Multiplier On Slide Change	2
Gravity Multiplier If Unclimbable Slope	30
Lock On Slope	<input checked="" type="checkbox"/>

Its variables are:

- **Ground checker threshold:** the minimum distance from the bottom to the player to the floor to be considered grounded
- **Slope checker threshold:** the distance used from the center of the player to check if is on a slope
- **Step checker threshold:** the distance used from the center of the player to check if is on a step
- **Max climbable slope angle:** the maximum angle that the player can climb
- **Max step height:** the maximum step height that the player can climb over
- **Speed multiplier on angle:** an additional speed that can be applied at specific angles. It can be used for example to increase the speed artificially on ramps
  - **Can slide multiplier curve:** impact of the curve on a slide
  - **Can't slide multiplier curve:** impact of the curve on a slide that can't be climbed
  - **Climbing stairs multiplier curve:** impact of the curve on a step
- **Gravity multiplier:** gravity applied to the player
- **Gravity multiplier on slide:** gravity applied to the player when on a slide
- **Gravity multiplier on unclimbable slide:** gravity applied to the player when on an unclimbable slide
- **Lock on slope:** if enabled the player sticks to the slide and doesn't move down due to gravity

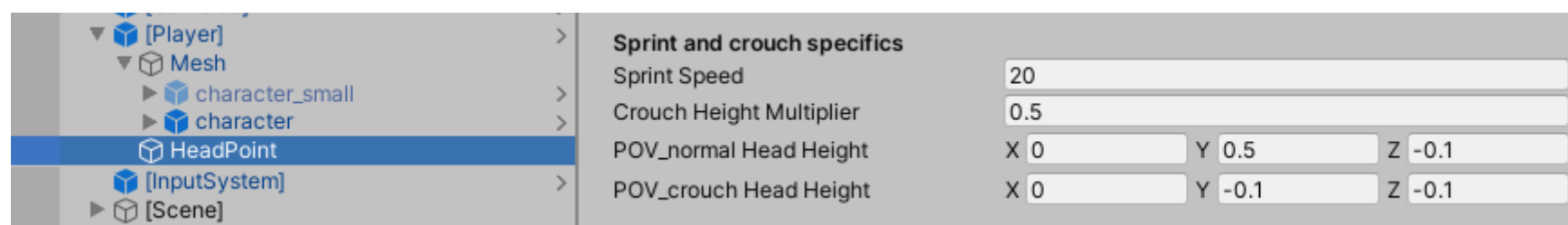
## Wall slide specifics

<b>Wall slide specifics</b>	
Wall Checker Threshold	0.8
Hight Wall Checker	0.5
Jump From Wall Multiplier	31
Multiplier Vertical Leap	1

Its variables are:

- **Wall checker threshold:** the minimum distance of a gameobject from the center of the player to be considered in contact with the player
  - **High wall checker:** height of the wall checker threshold
- **Jump from wall multiplier:** speed added to the player on both the Y and X axis when jumping from a wall
- **Multiplier vertical leap:** a multiplier applied only on the Y axis when jumping from a wall. This additional multiplier can be useful in case of extreme gravities to artificially boost the wall jump

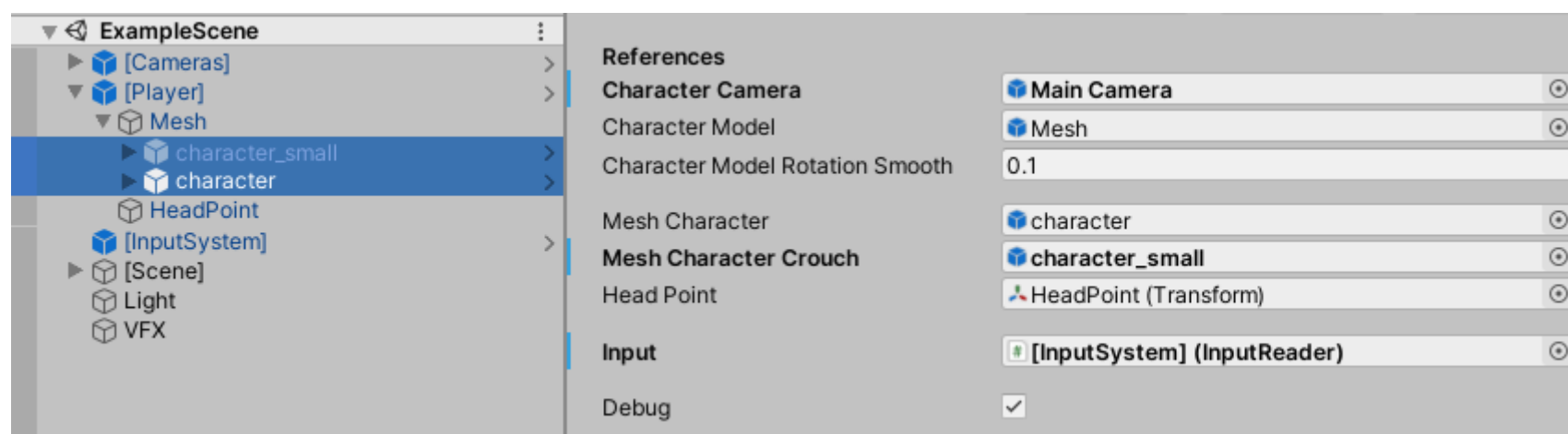
## Sprint and crouch specifics



Its variables are:

- **Sprint speed**
- **Crouch height multiplier**: the multiplier height of the collider when crouching (i.e. if the collider is tall 2 and the crouch height multiplier is 0.5, then the height of the collider when crouching is  $2 * 0.5 = 1$ )
- **POV\_normal head height**: the position of the head point (useful to be referenced by cameras)
- **POV\_crouch head height**: the position of the head point when crouching (useful to be referenced by cameras)

## References



Its variables are:

- **Character camera**: the camera reference is essential to move the character relative to the camera position
- **Character model**: the gameobject parent of the mesh, it's used to handle the rotation
  - **Character model rotation smooth**: the speed of the rotation
- **Mesh character / Mesh character crouch**: if your character has two different meshes when moving and when crouching then fill in these variables and the script will do it automatically. **If they are left empty only the collider will scale**
- **Head point**: reference to the head position
- **Input**: Reference to the input system
- **Debug**: allows you to see the debug infos and gizmos

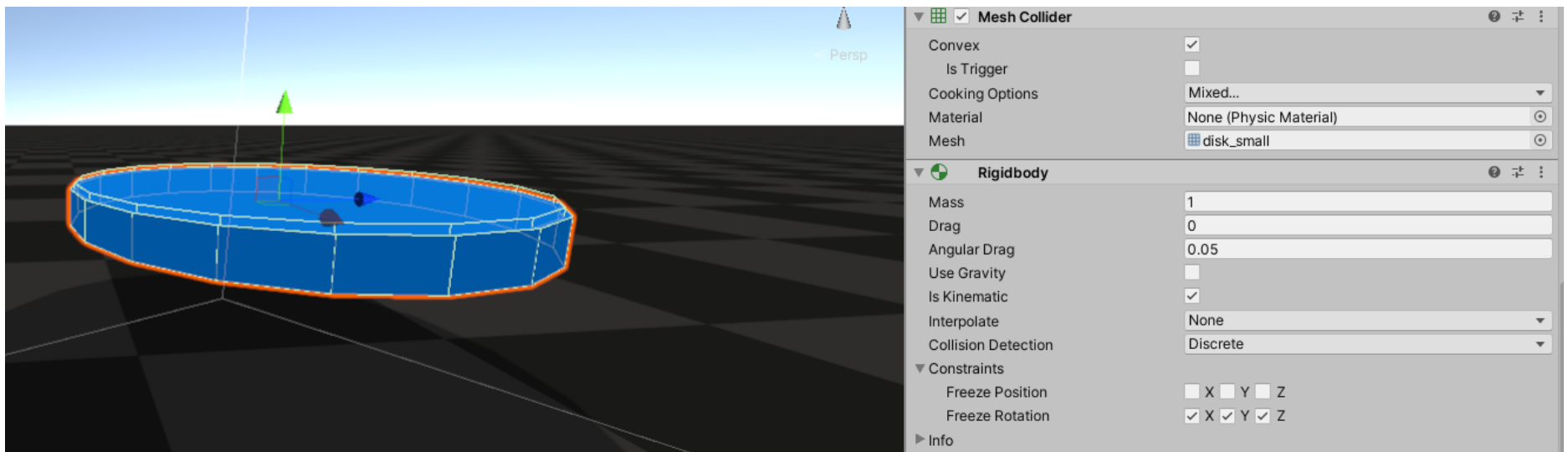
N.B. all this references except for the mesh character and mesh character crouch are essential to use the character manager properly

## Platformer extensions

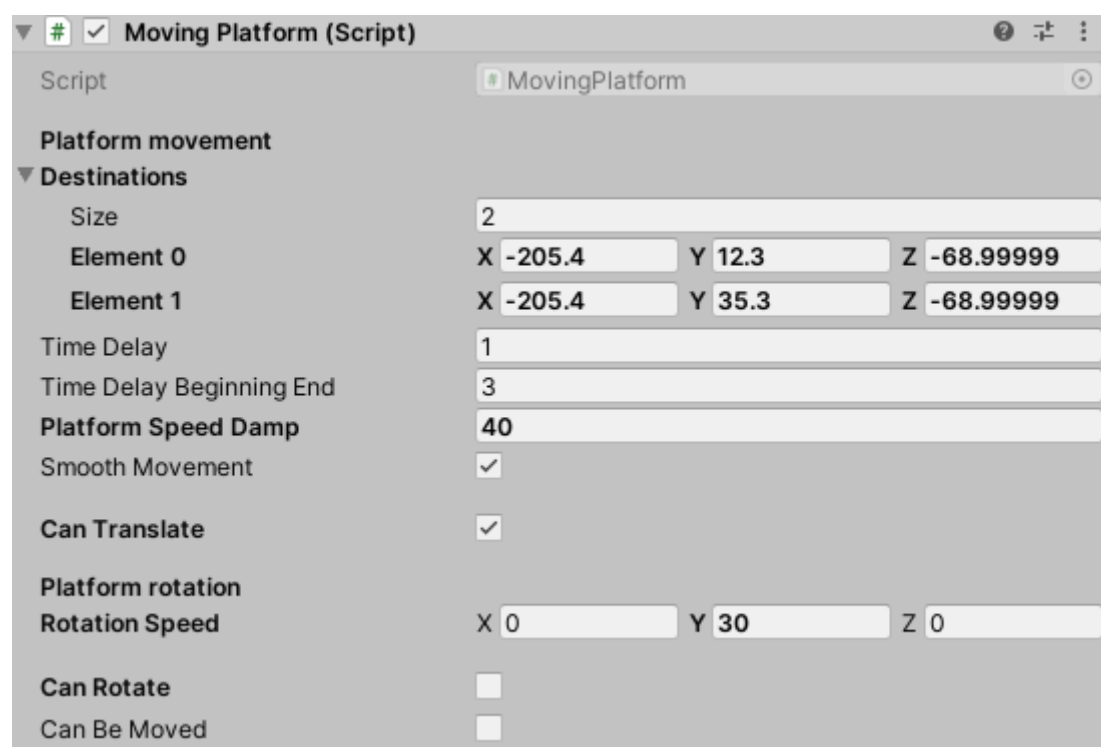
The asset has also a few common types of platforms and area effectors used in 3D platformers

### Moving platforms

It is a script that allows gameobjs to translate and / or rotate. The platform has a *Trigger* with the script **PlatformSensor** to detect interaction with the player



Their setup is quite simple, the platform requires a *Collider* that can interact with the player layer and a *Rigidbody*. The *Rigidbody* needs to be frozen in its rotation, its gravity needs to be disabled and it needs to be set to kinematic since its movement is managed by the script.

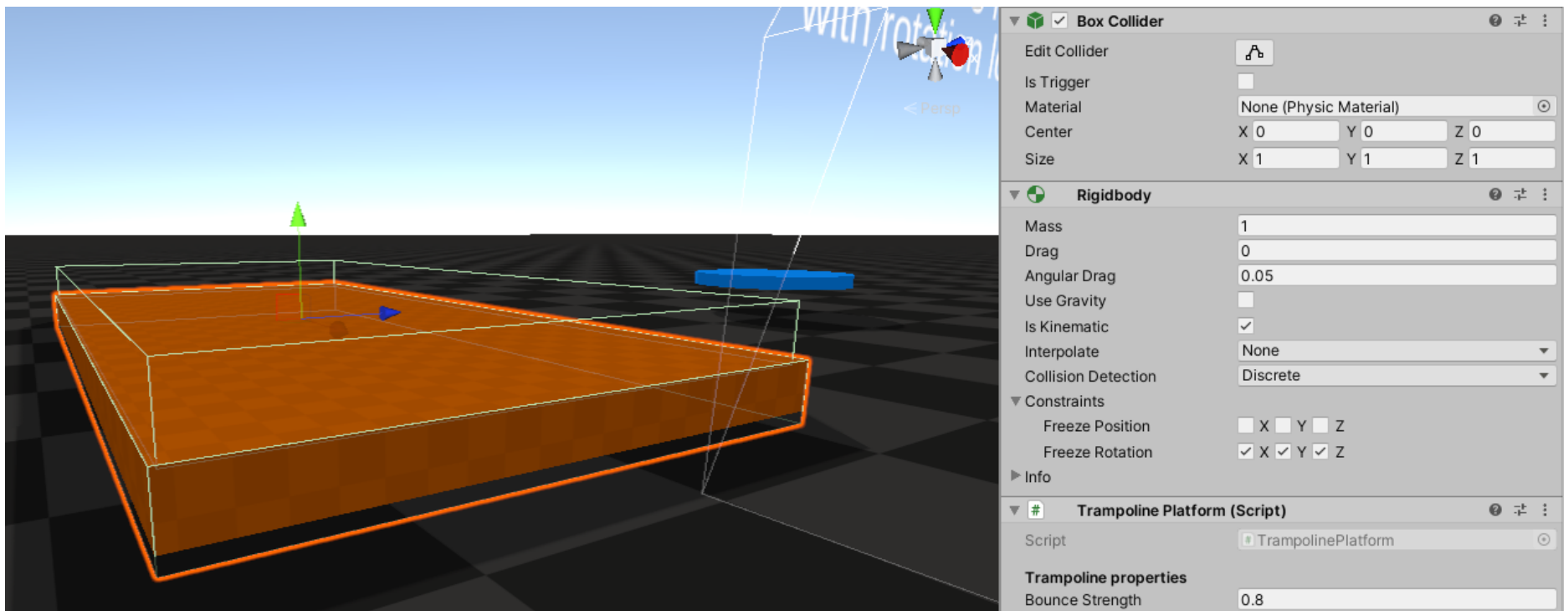


- **Destinations:** the positions that the platform can move to (in local space if it has a parent)
- **Time delay:** after how much time it changes position
- **Time delay beginning end:** how much delay it takes before starting a new cycle
- **Platform speed Damps:** smoothness of the movement
  - **Smooth movement:** enables / disables dampness
- **Can translate**
- **Rotation Speed:** how fast it spins on one axis
- **Can rotate**
- **Can be moved:** can interact and its motion changed by the player



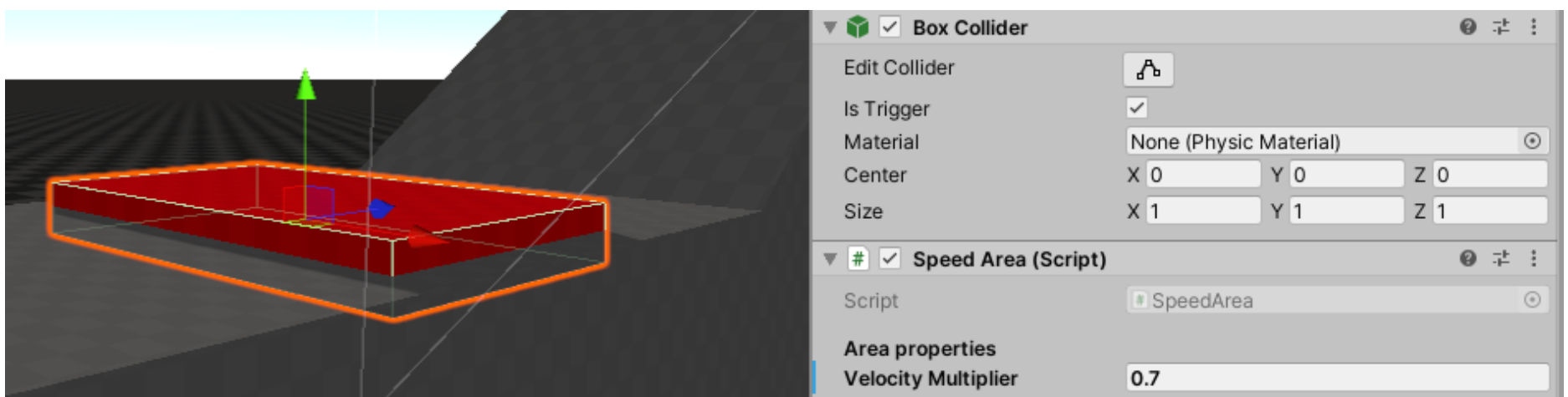
## Trampoline Platforms

It is a script that applies a vertical force to the player and whatever object interacts with it. Just like the **MovingPlatform** it has a child *Trigger* with the script **TrampoLineSensor** and its physics is handled kinematically.



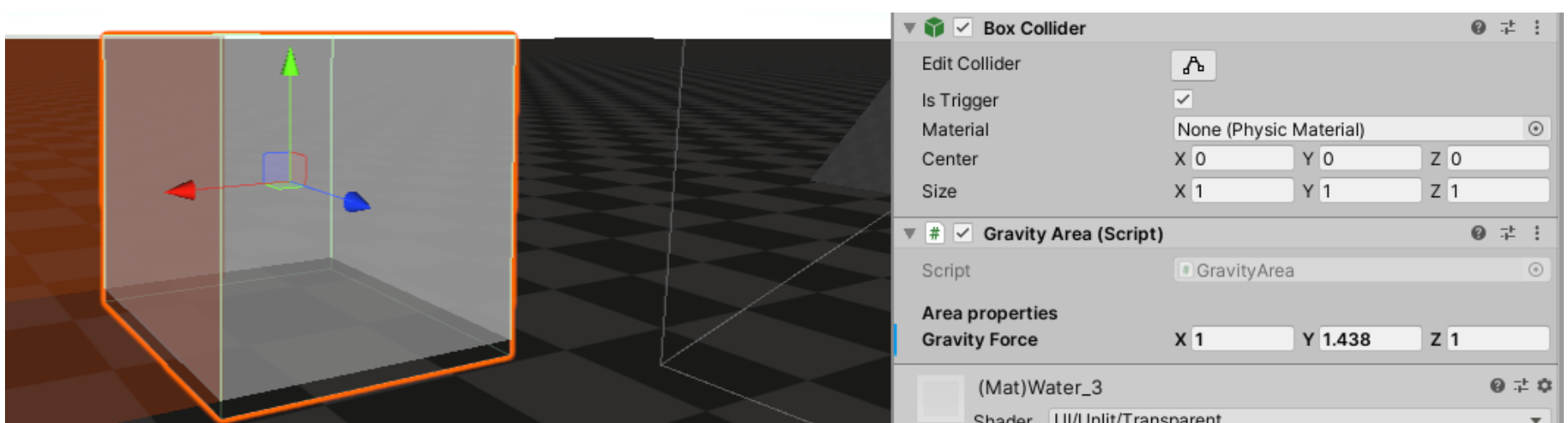
## Speed Area

A simple script applied to a trigger area that multiplies the character speed to whatever value is set. If the **Velocity Multiplier** is set to 1 the character speed is left to unchanged



## Gravity Area

A simple script applied to a trigger area that applies gravity the character speed to whatever value is set. If the **Gravity Multiplier** is set to (1,1,1) the character gravity is left to unchanged





## ***Contact***

If you found this guide useful but need further help feel free to contact me at the email [nappin.1bit@gmail.com](mailto:nappin.1bit@gmail.com)  
P.S. A positive review of the asset would help a lot!

Cheers