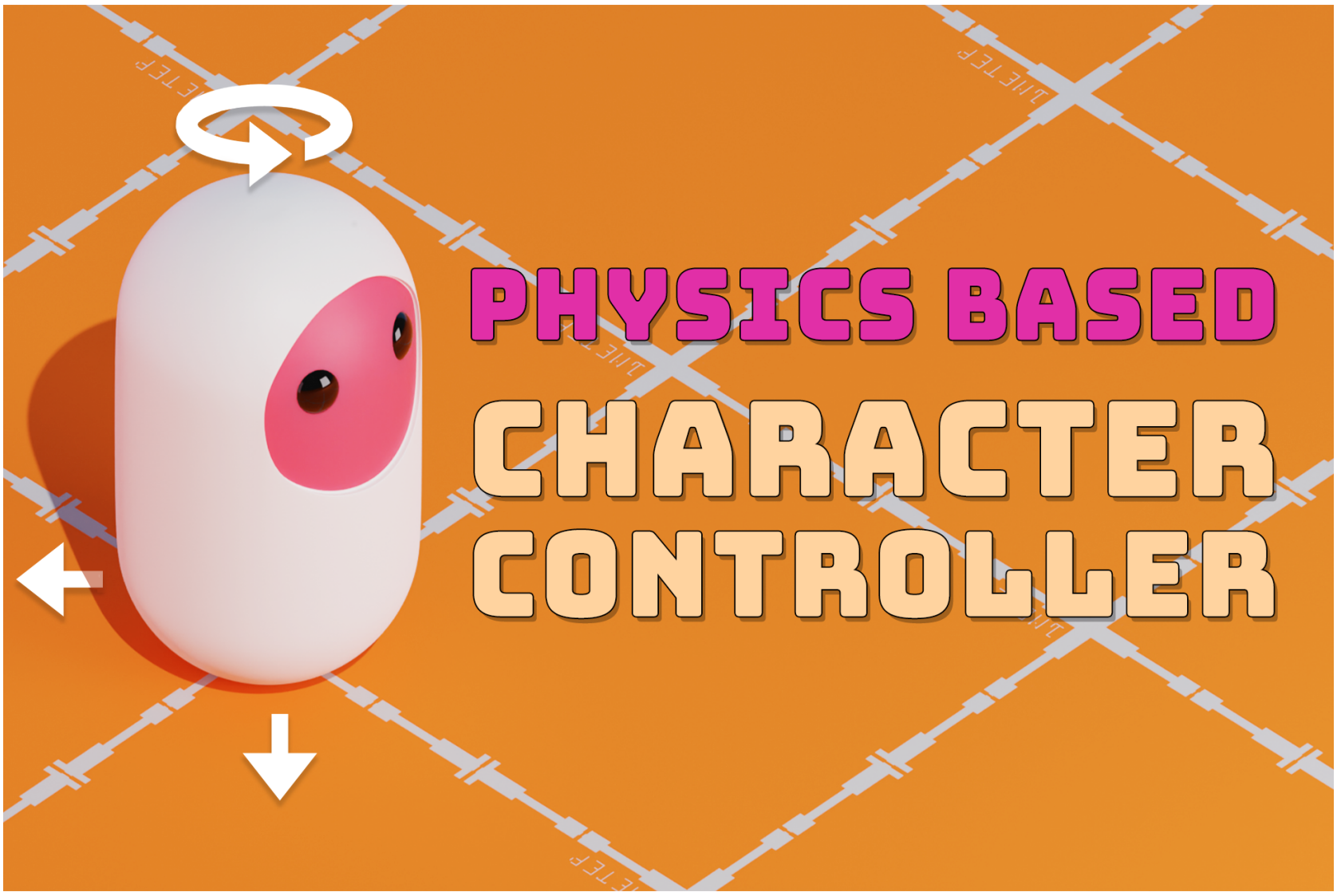


Physics Based Character Controller

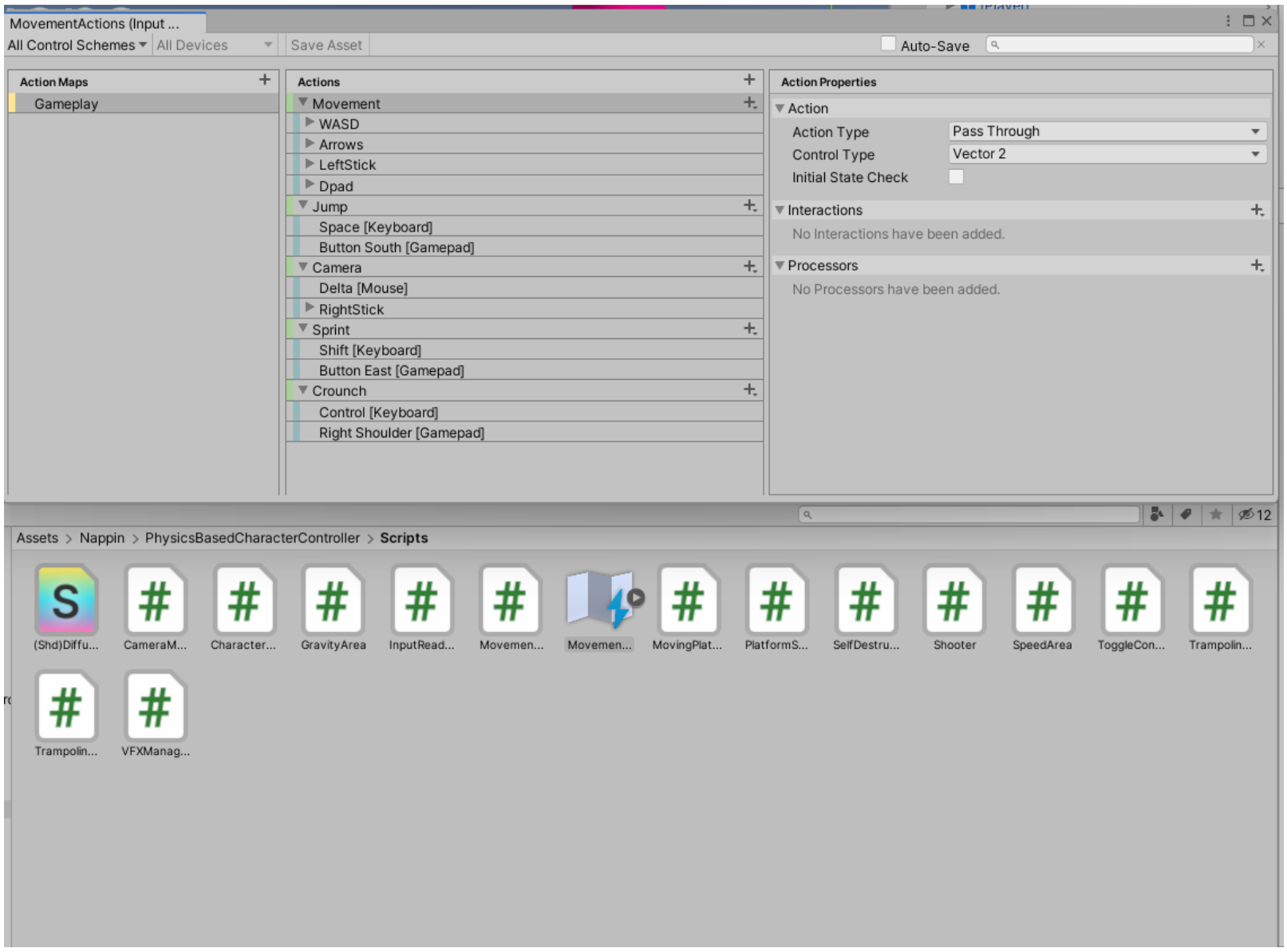
Support email: nappin.1bit@gmail.com

Table of Contents

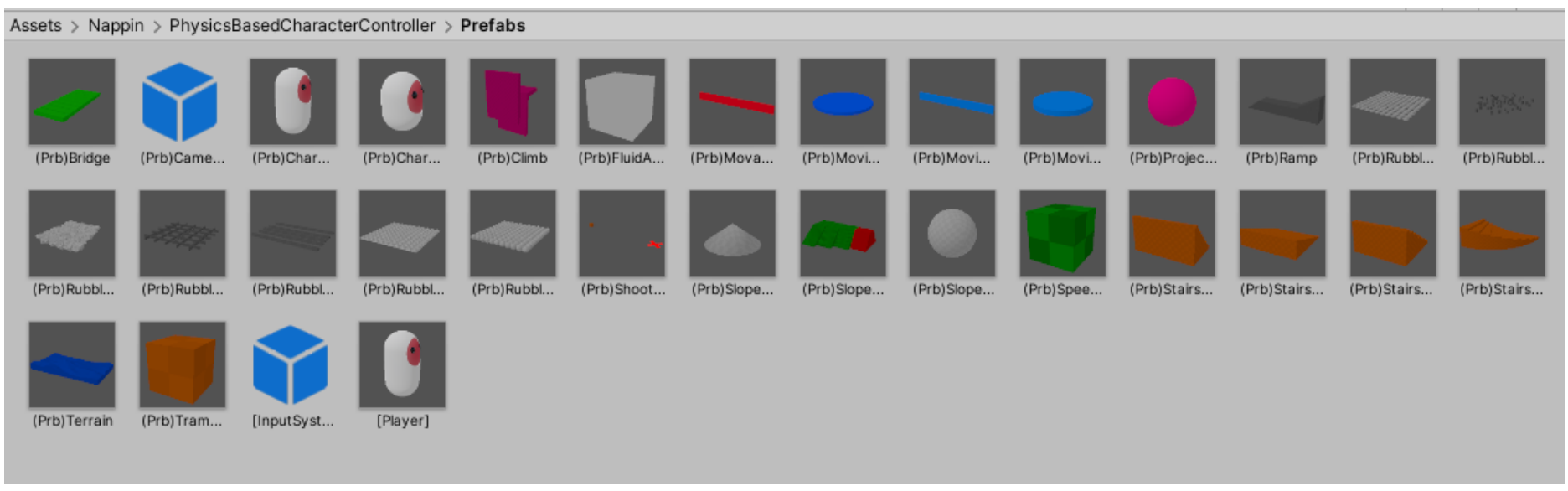
Asset content

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]**



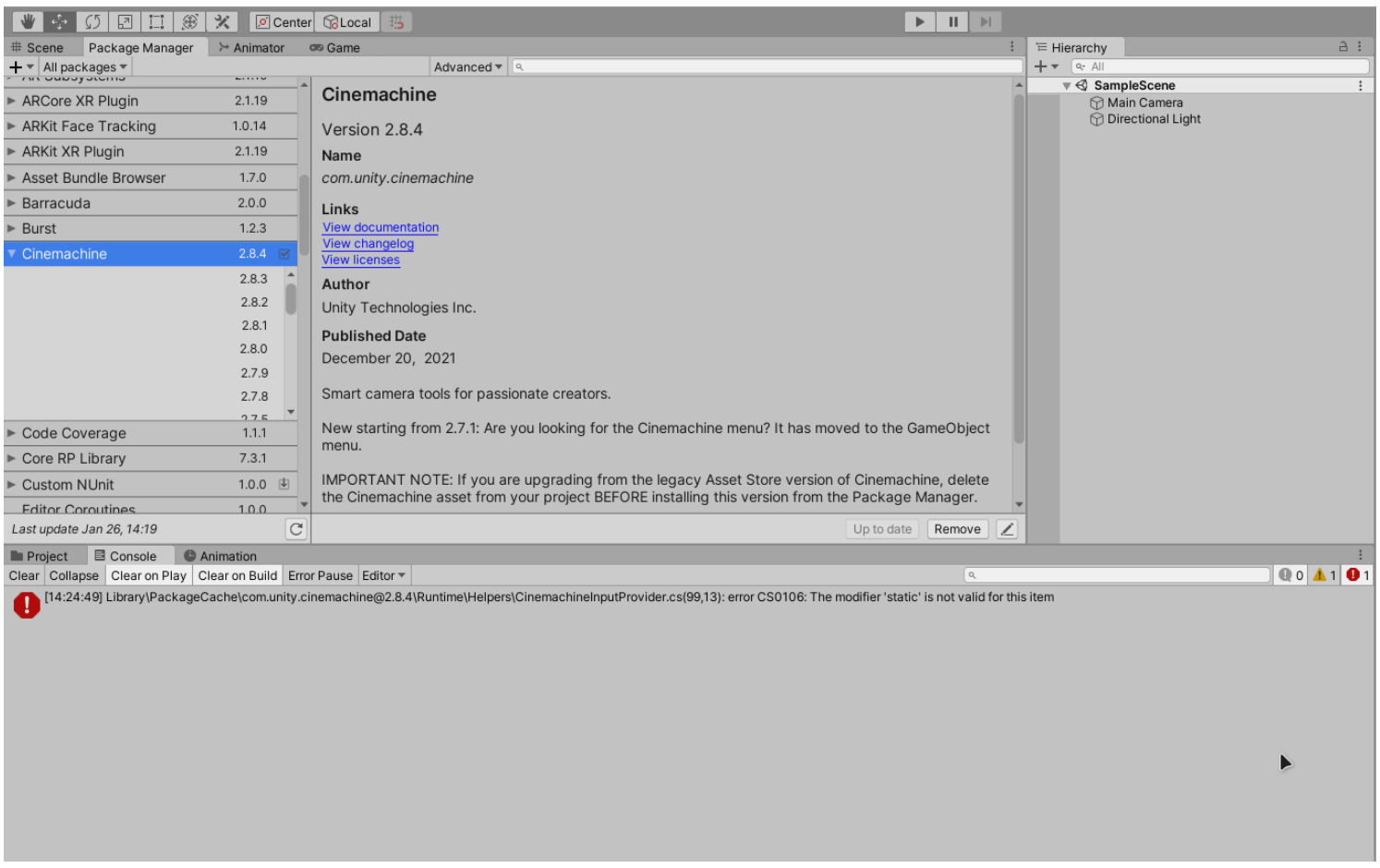
The **[InputSystem]** contains a script called **InputReader** that gets the input from the old or new input system and converts it into usable data for che **[Player]**.
The **[Player]** contains a script called **CharacterManager** that uses the input data and moves the character.

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.

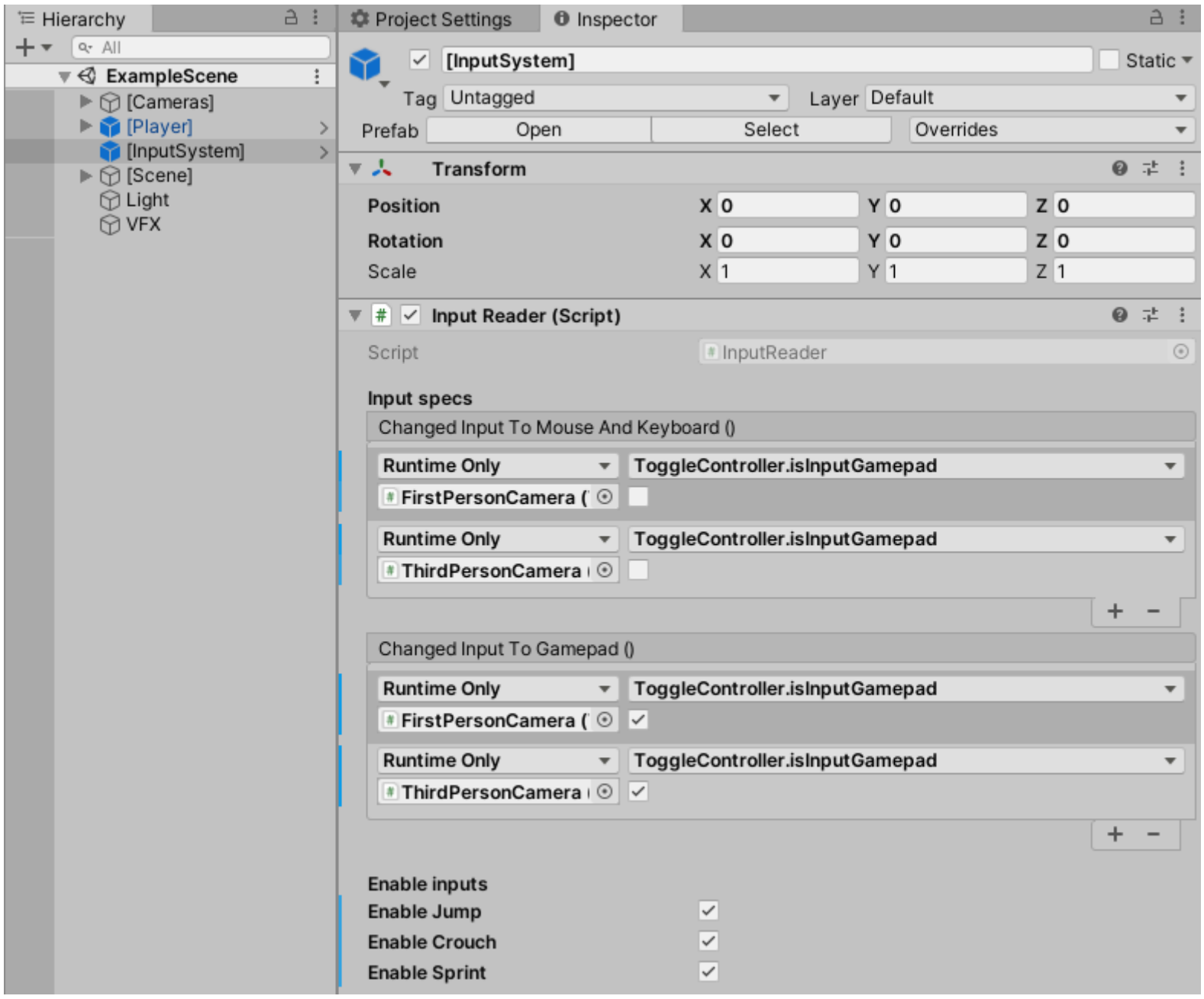
The example scene uses a simple **Cinemachine** setup to operate in first person and in third person. Even though the package is not required it's recommended especially if you want to explore the *ExampleScene*.



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.

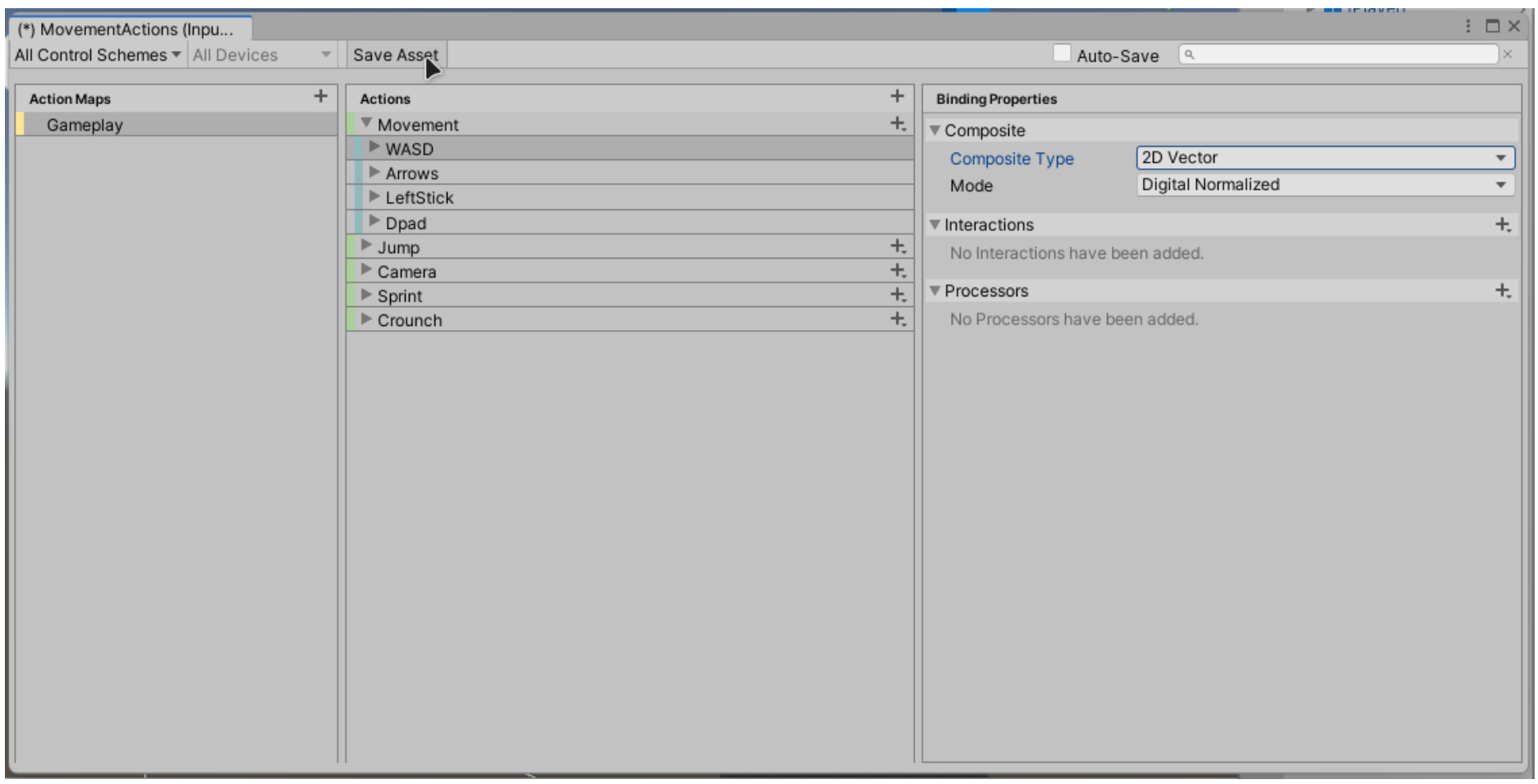
Setup the Input

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:

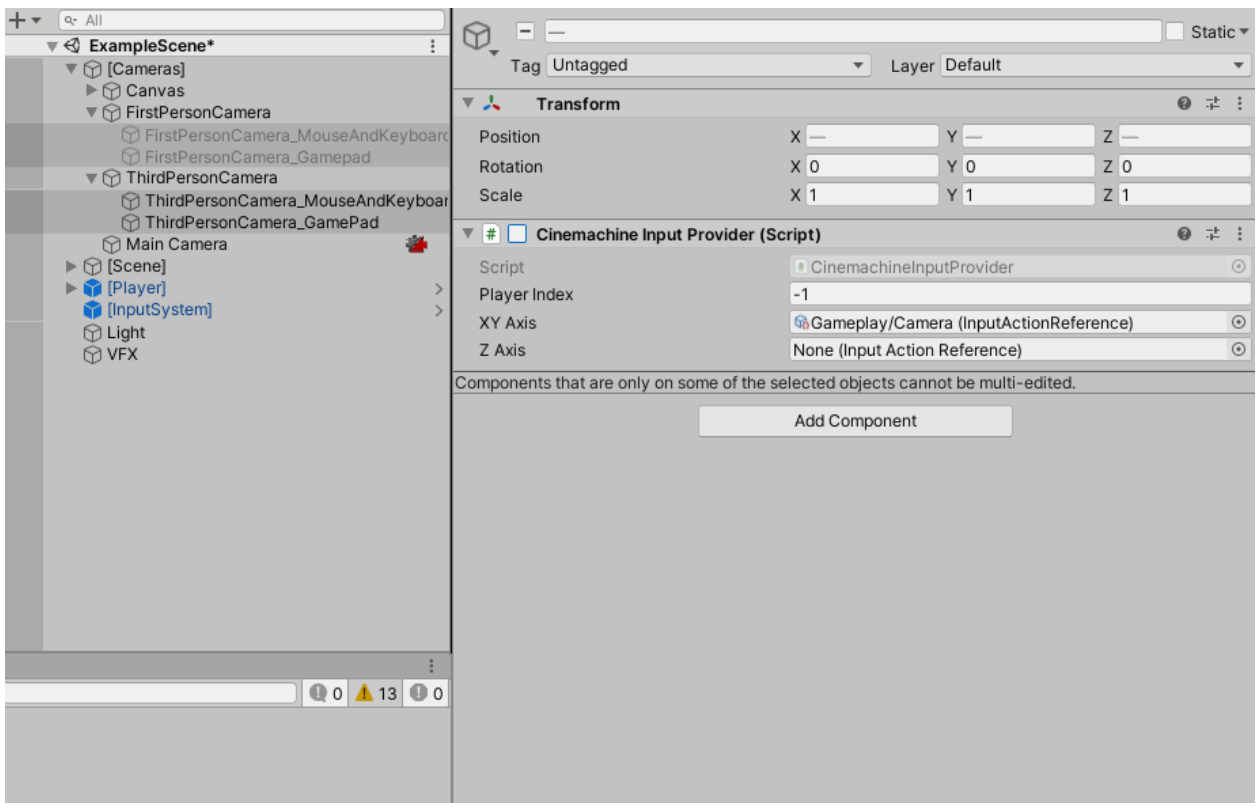
- Open the **InputReader** script, here you will find comments indicating to you what lines of code to ***disable*** and to ***enable.***

```
//DISABLE if using old input system
Unity Message | 0 references
private void Awake()
{
    movementActions = new MovementActions();

    movementActions.Gameplay.Movement.performed += ctx => OnMove(ctx);

    movementActions.Gameplay.Jump.performed += ctx => OnJump();
    movementActions.Gameplay.Jump.canceled += ctx => JumpEnded();

    movementActions.Gameplay.Camera.performed += ctx => OnCamera(ctx);

    movementActions.Gameplay.Sprint.performed += ctx => OnSprint(ctx);
    movementActions.Gameplay.Sprint.canceled += ctx => SprintEnded(ctx);

    movementActions.Gameplay.Crounch.performed += ctx => OnCrouch(ctx);
    movementActions.Gameplay.Crounch.canceled += ctx => CrouchEnded(ctx);
}


//ENABLE if using old input system
Unity Message | 0 references
private void Update()
{
    /*

    axisInput = new Vector3(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"), 0f).normalized;

    if (enableJump)
    {
        if (Input.GetButtonDown("Jump")) OnJump();
        if (Input.GetButtonUp("Jump")) JumpEnded();
    }

    if (enableSprint) sprint = Input.GetButton("Fire3");
    if (enableCrouch) crouch = Input.GetButton("Fire1");

    GetDeviceOld();

    */
}
```

- If you intend to use the camera setup provided in the sample scene then you need to edit the **CameraManager** script. Here you will find comments indicating to you what lines of code to ***disable*** and to ***enable.***

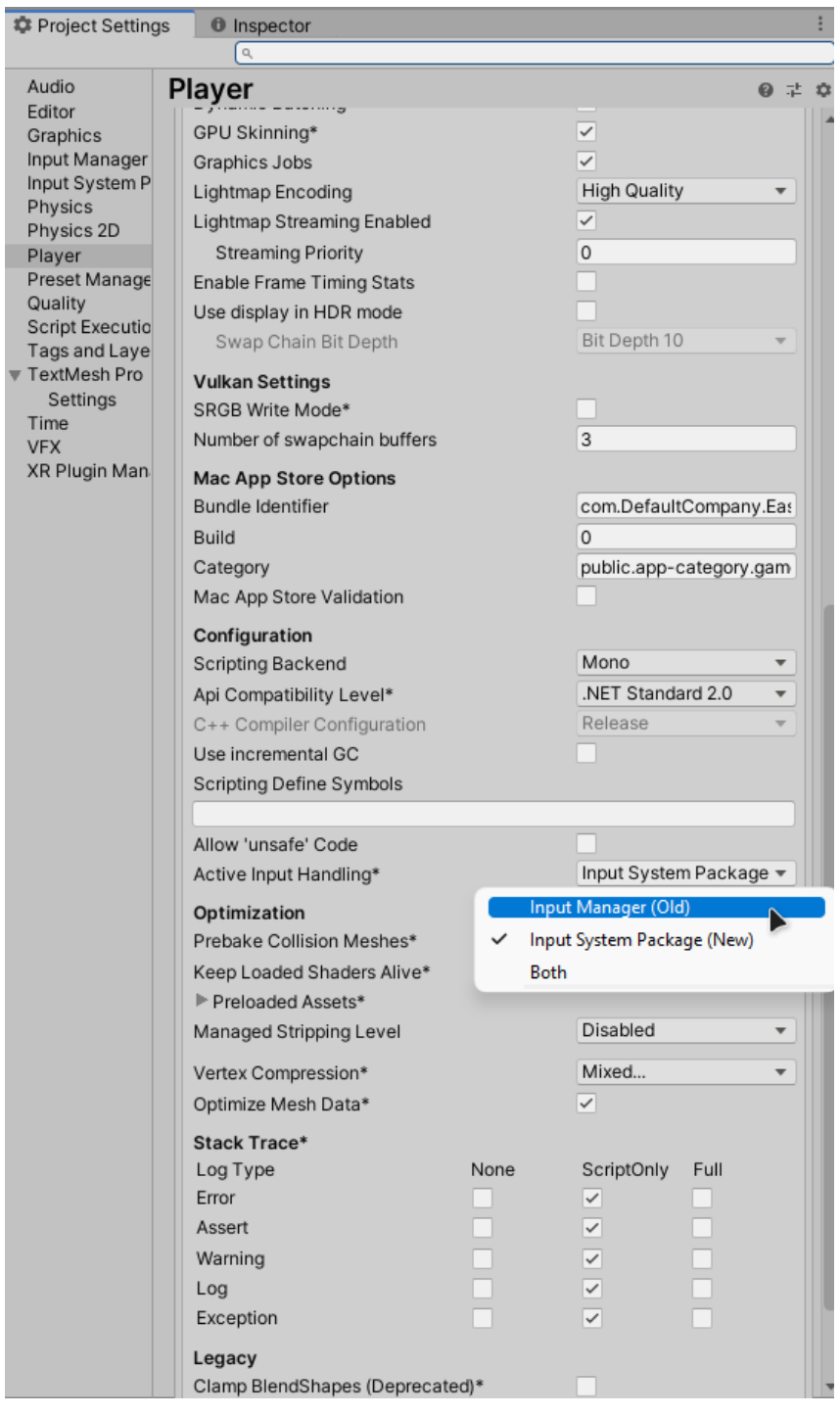
```
Unity Message | 0 references
private void Update()
{
    //DISABLE if using old input system

    if (Keyboard.current.mKey.wasPressedThisFrame)
    {
        activeThirdPerson = !activeThirdPerson;
        SetCamera();
    }

    //DISABLE if using old input system
    if (Keyboard.current.nKey.wasPressedThisFrame)
    {
        SetDebug();
    }

    //ENABLE if using old input system

    /*

    if (Input.GetKeyDown(KeyCode.M))
    {
        activeThirdPerson = !activeThirdPerson;
        SetCamera();
    }

    if (Input.GetKeyDown(KeyCode.N))
    {
        SetDebug();
    }

    */
}
```

- 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 **MovementActions** script and .inputaction files from the project or whatever **InputAction** you are using.

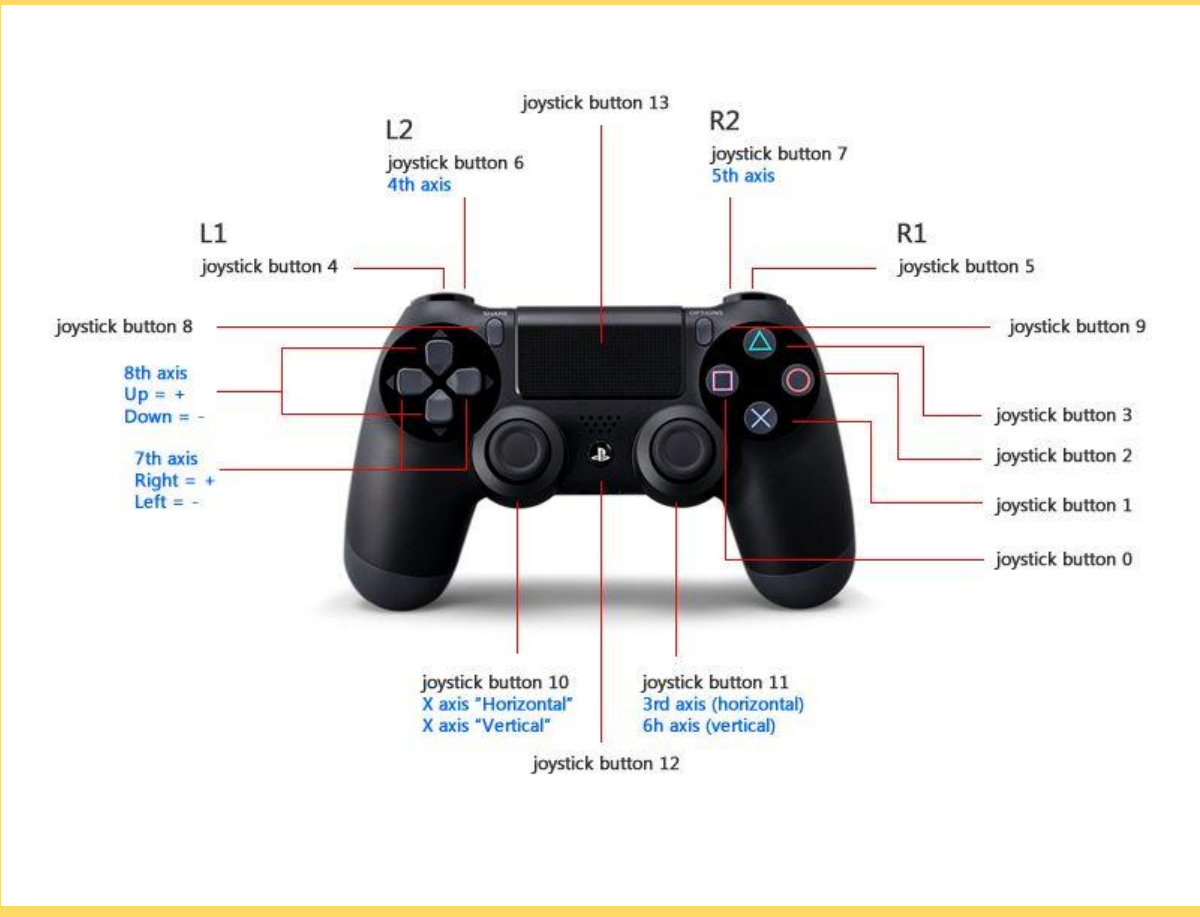
- Remove the New Input System from the project and enable the Old Input System in the **Project Settings**



- If they are reported as errors, remove all the instances of **UnityEngine.InputSystem** from the scripts

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

N.B. The controller inputs used for the controller when using the **old input system** are different from the ones used in the **new input system**. The reason is that in the old input system the inputs are handled by the *InputManager* and downloaded assets can't edit those settings so the default ones are used. If you want to edit them, you can work on Jump / Fire1 / Fire3 and edit the values “joystick button X” with whatever button you want. Here is a simple scheme to help you out:

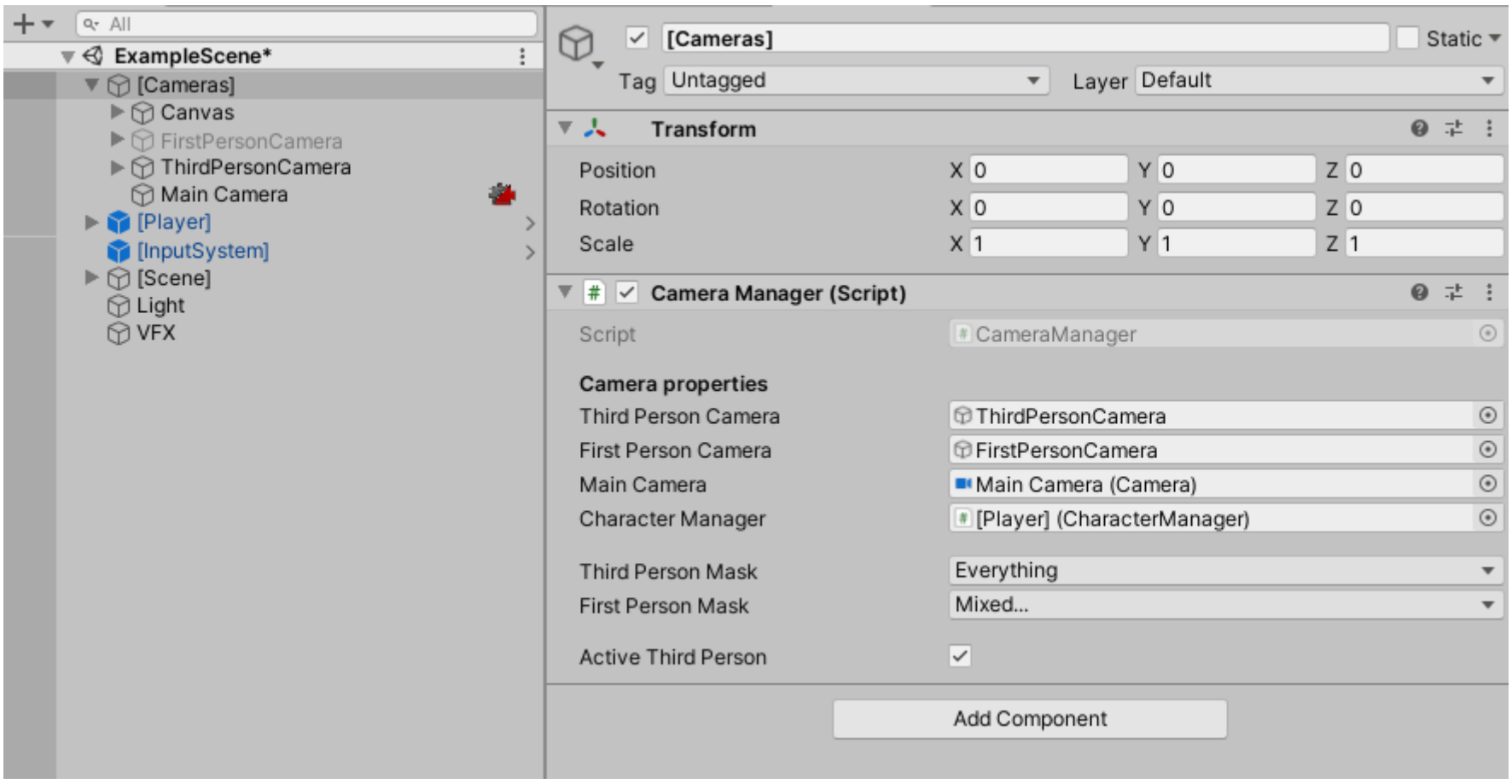


Setup the Camera

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

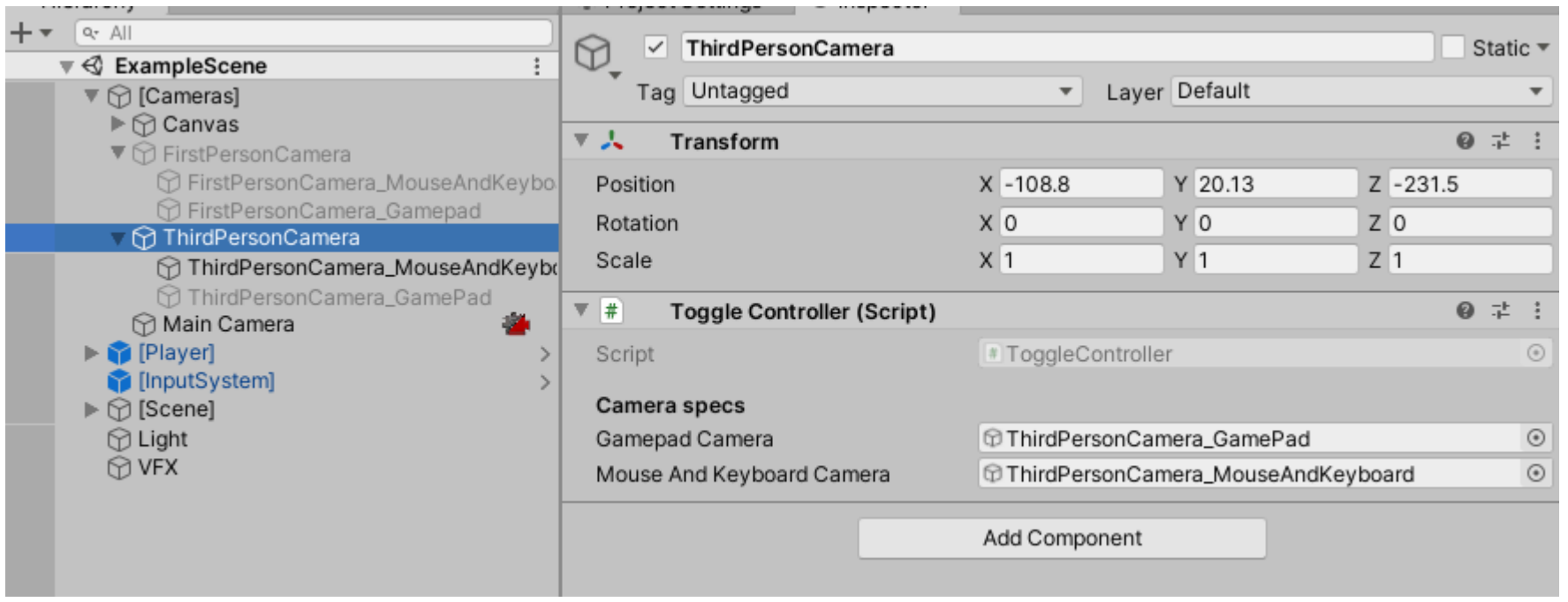
The brain of the setup is the **CameraManager**:



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



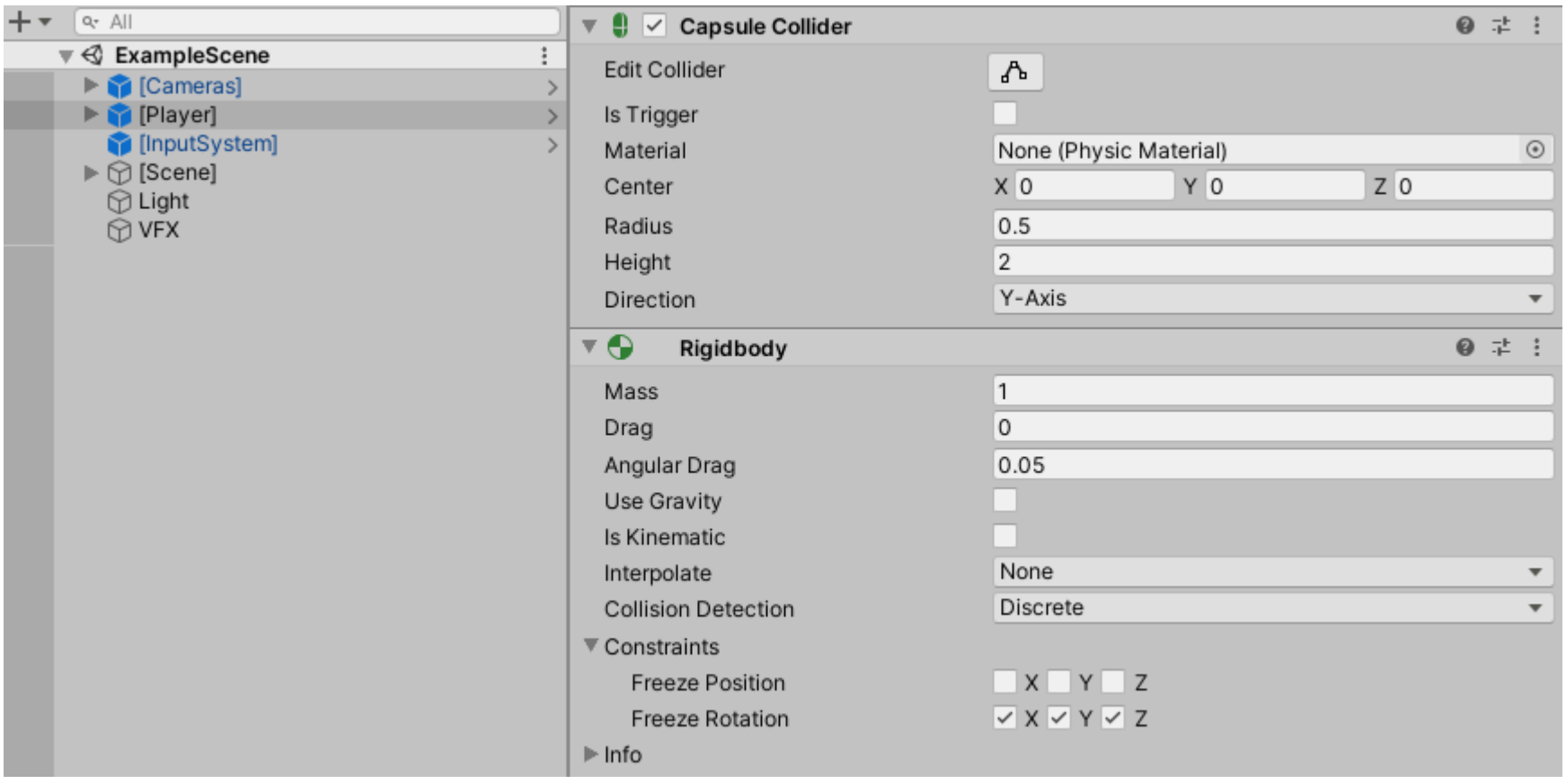
Setup the Player

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

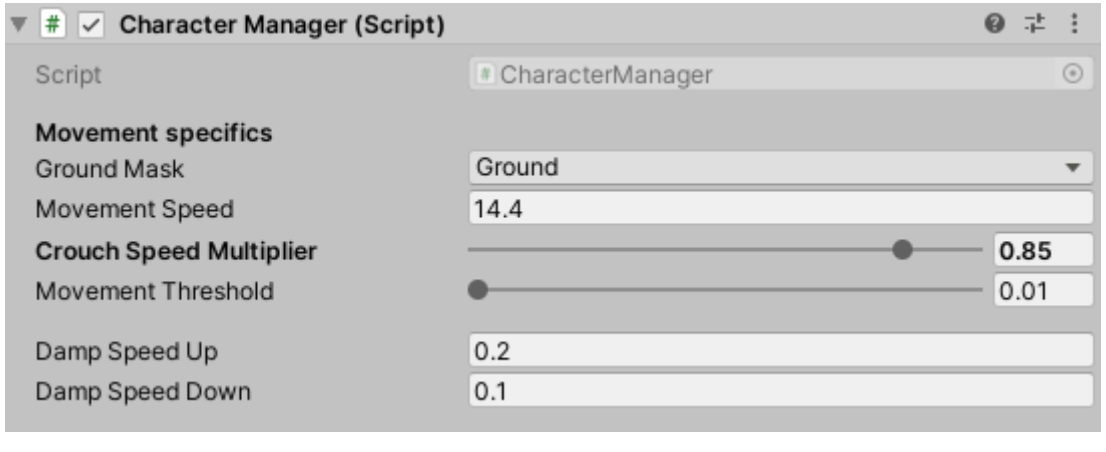
All the variables in the **CharacterManager** script have a description, you can read it just by hovering over the variable.

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:

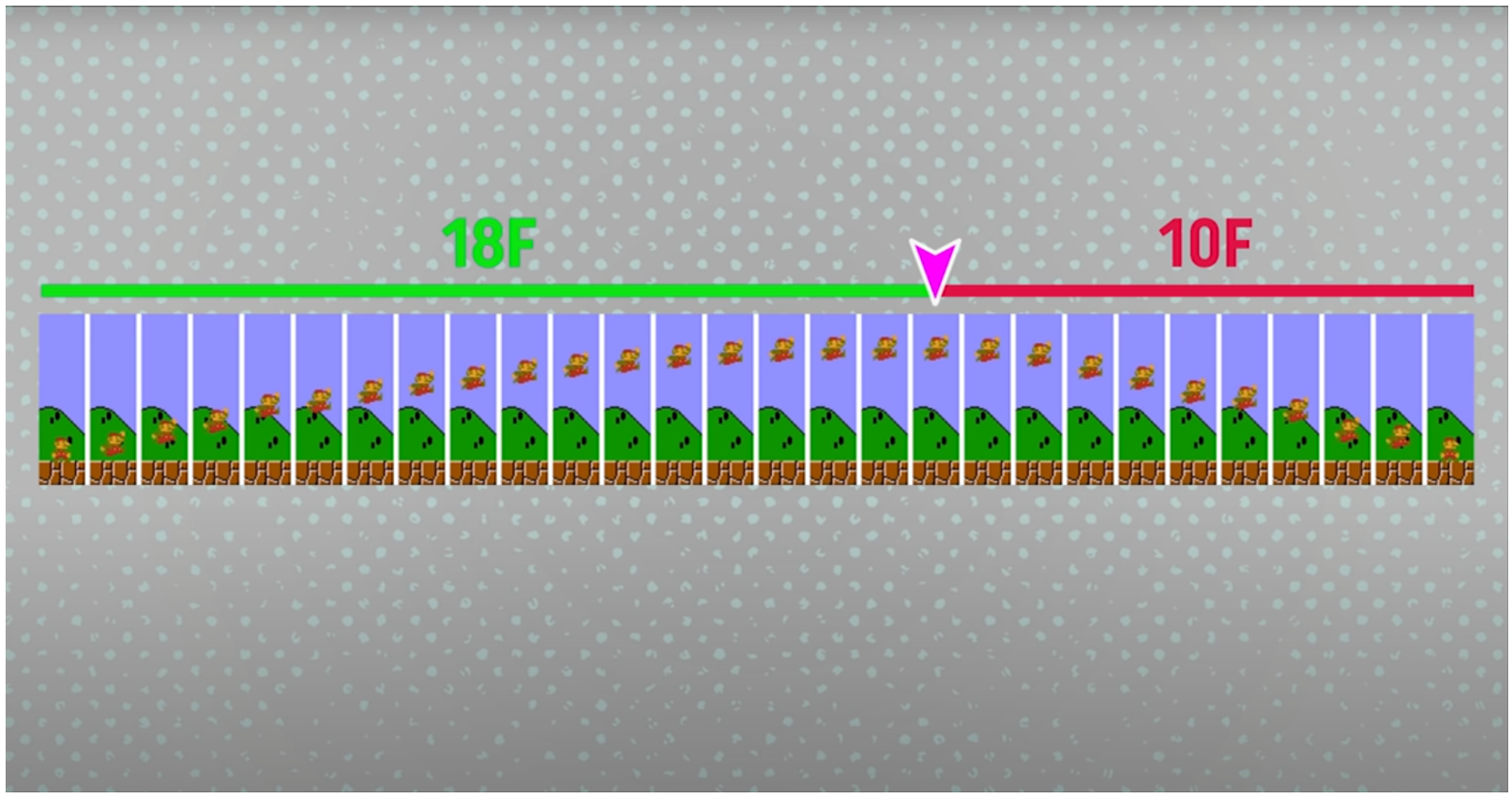
- ***Ground mask***: the layers that collide with the player
- ***Movement speed***
- ***Crouch speed multiplier***: if the crouch is enabled how slow does the character move (i.e. if the value is 0.5 the character moves at half the speed when crouching)
- ***Movement threshold***: what's the minimum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

Its variables are:

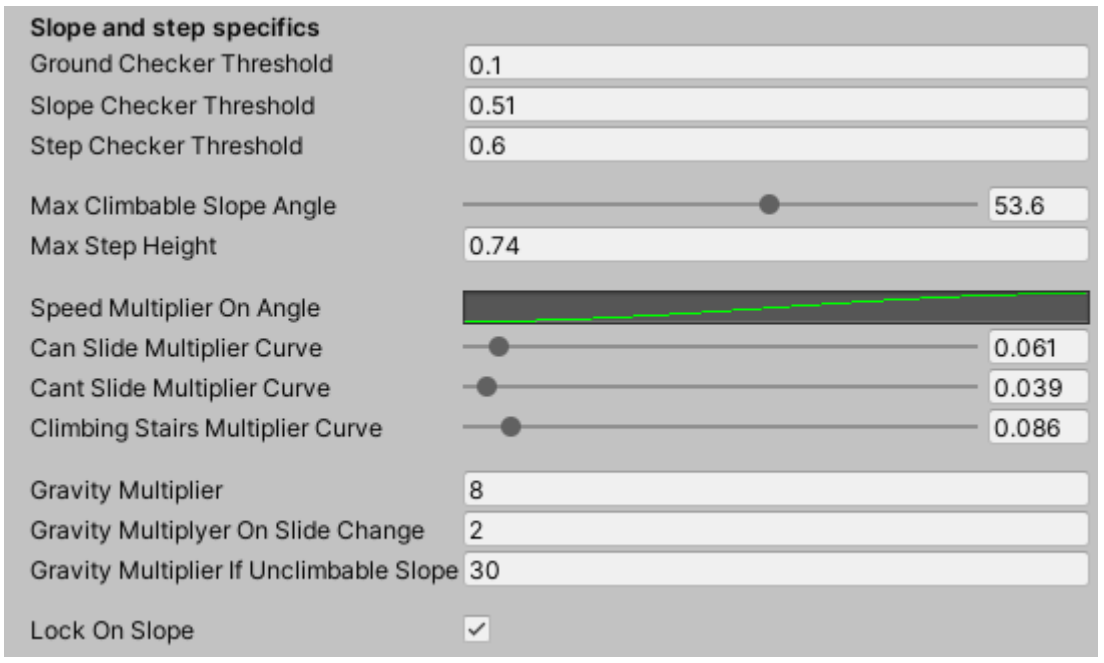
- ***Jump velocity***
- ***Fall multiplier***: if the value is left at 1 the gravity is always the same. This value allows you to achieve a more exaggerated and cartoony jump making the movement more intense when falling



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

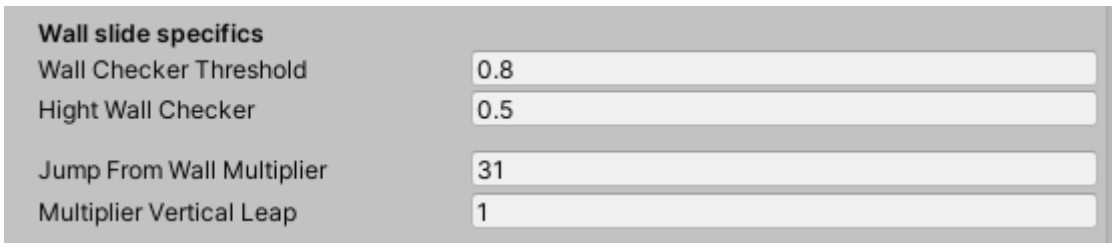
Slope and steps specifics



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

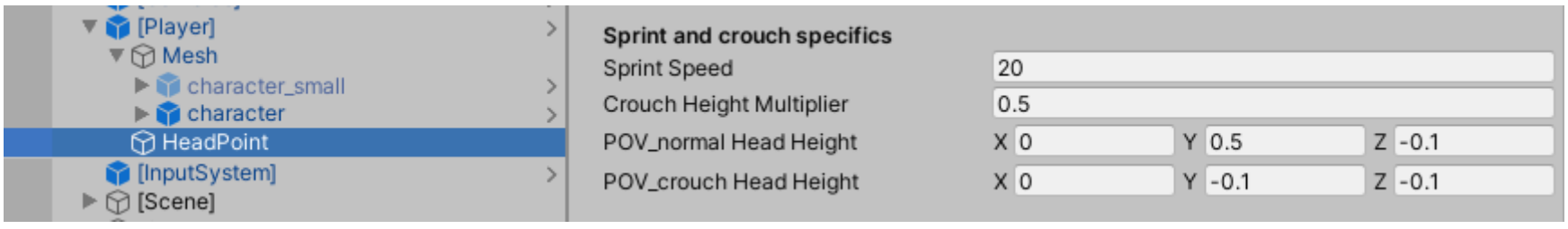
Wall slide specifics



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

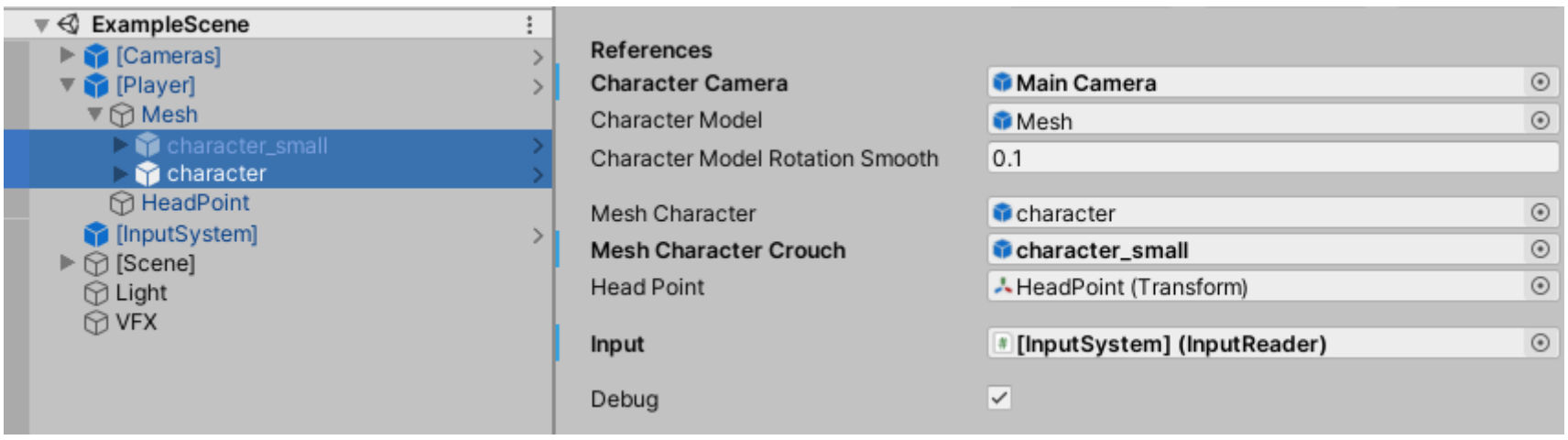
Sprint and crouch specifics



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***: the meshes used when moving and when crouching. This fields can be left empty to allow you to implement custom animation solutions
- ***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

Getters & Setters

If you want to use the **CharacterManager** script private variables for your own needs, you can access them using the following getters (or creating your own custom methods):

```
#region GettersSetters

1 reference
public bool GetGrounded() { return isGrounded; }
0 references
public bool GetTouchingSlope() { return isTouchingSlope; }
0 references
public bool GetTouchingStep() { return isTouchingStep; }
3 references
public bool GetTouchingWall() { return isTouchingWall; }
1 reference
public bool GetJumping() { return isJumping; }
1 reference
public bool GetCrouching() { return isCrouch; }
1 reference
public float GetOriginalColliderHeight() { return originalColliderHeight; }

1 reference
public void SetLockRotation(bool _lock) { lockRotation = _lock; }

#endregion
```

The available getters are:

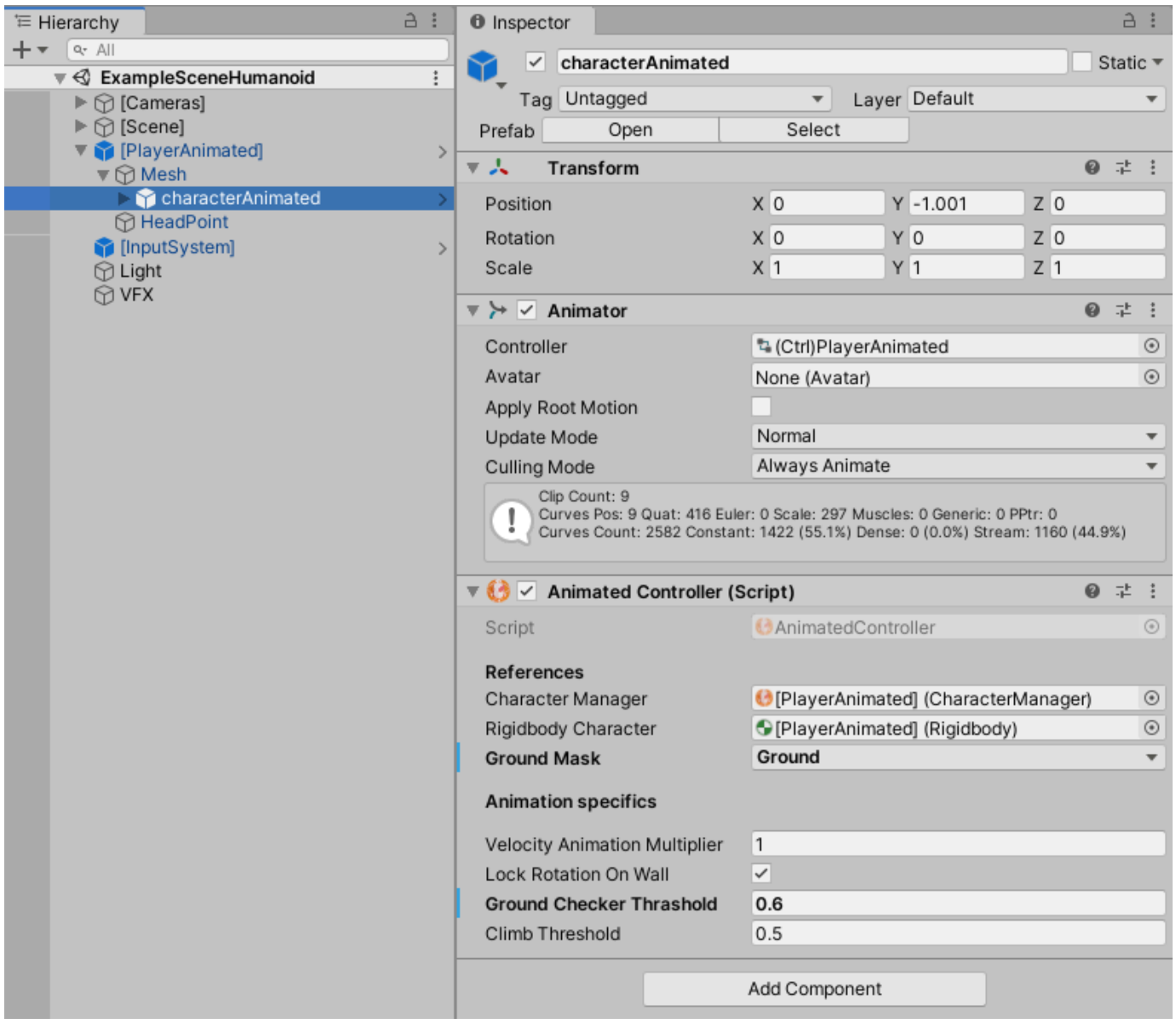
- ***GetGrounded():*** returns true if the player is touching the ground.
- ***GetTouchingSlope():*** returns true if the player is on a slope.
- ***GetTouchingStep():*** returns true if the player is touching a step.
- ***GetTouchingWall():*** returns true if the player is on a wall.
- ***GetJumping():*** returns true if the player is jumping.
- ***GetCrouching():*** returns true if the player is crouching.
- ***GetOriginalColliderHeight():*** returns the height of the capsule collider.
- ***SetLockRotation():*** allows you to enable or disable the character rotation.

Humanoid Character

The asset includes an animated **Humanoid Character** with script / animation controller and animations that can be used as a reference to create and animate your own characters. A sample scene including the **Humanoid Character** can be found in the project with the name *ExampleSceneHumanoid*.

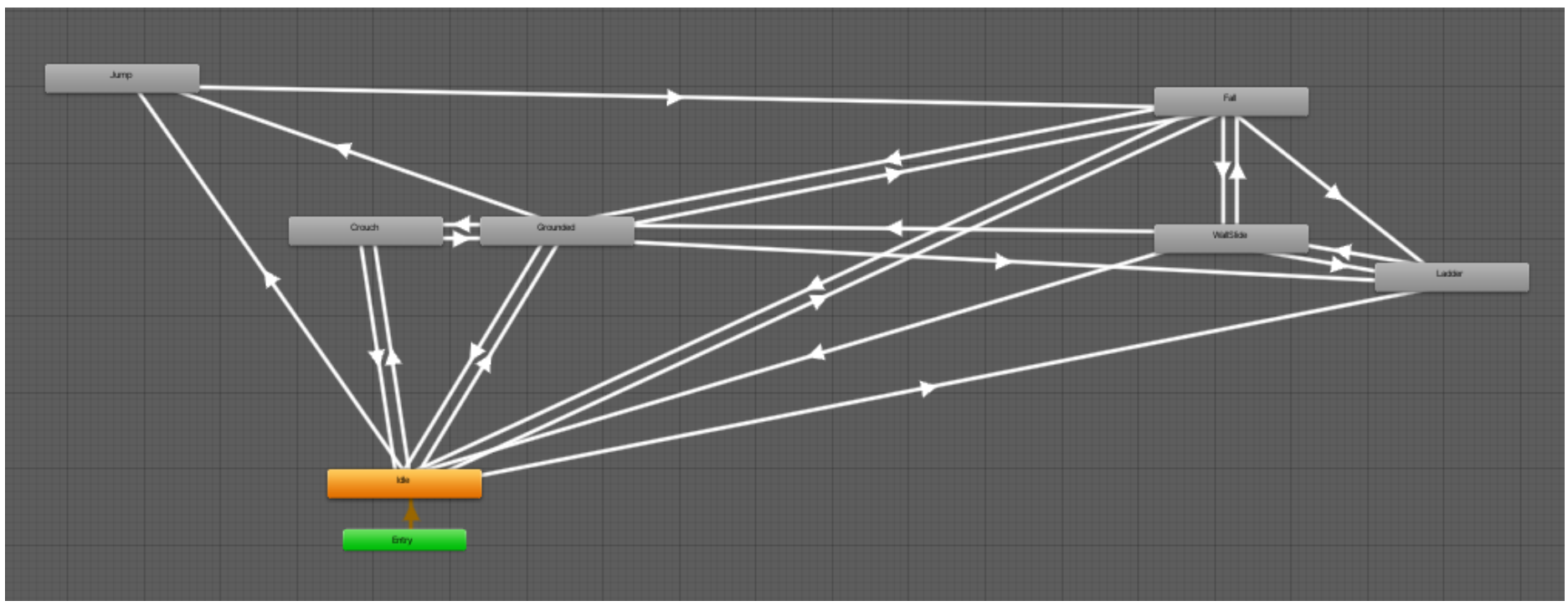
Hierarchy

The hierarchy of the Animated Player is the same as the capsule Player. The prefab of the [PlayerAnimated] and characterAnimated can be found inside a project. The characterAnimated contains 2 core component: **Animator** and the **AnimatedController**:

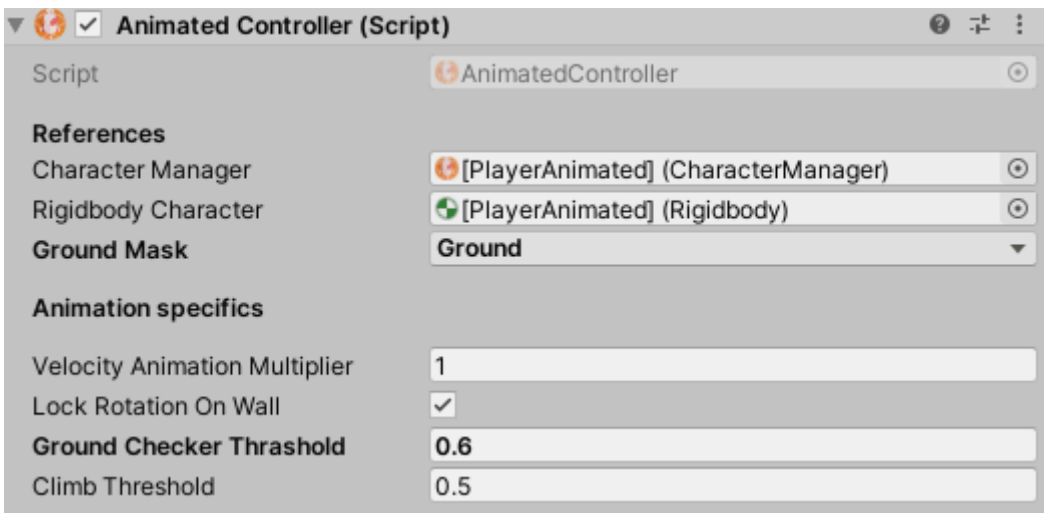


Animator

The Animator includes a variety of simple animation and blend trees. The animations can be found inside the *Animation* folder.



Animated Controller



Its variables are:

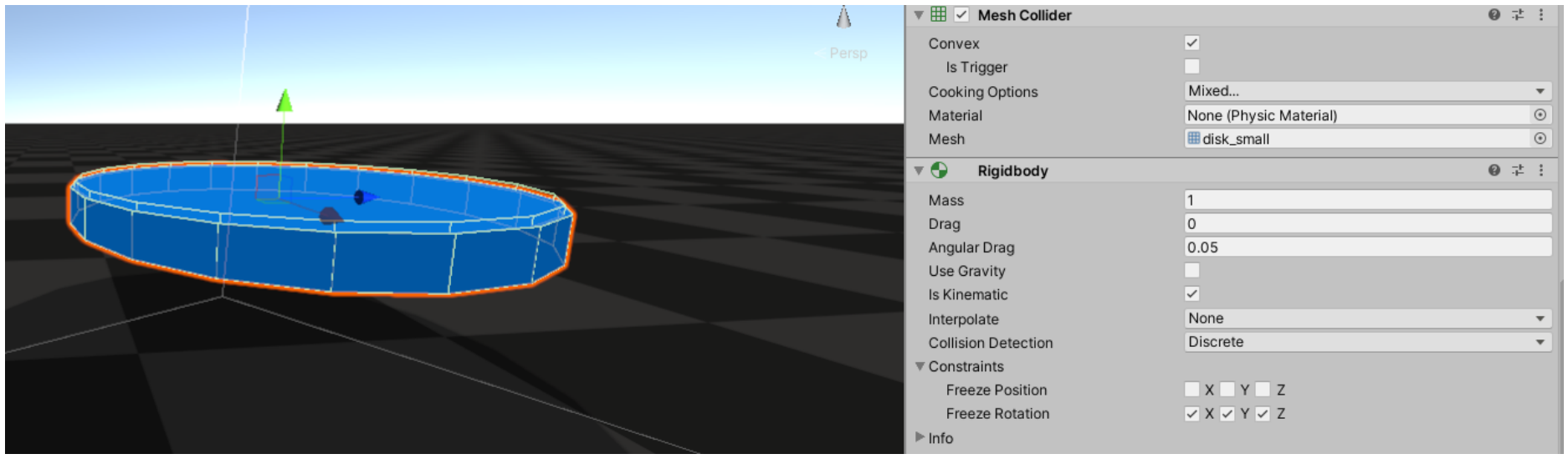
- ***CharacterManager***: reference to the player **CharacterManager** script.
- ***RigidbodyCharacter***: reference to the player Rigidbody.
- ***GroundMask:*** reference to the ground layer.
- ***VelocityAnimationMultiplier***: a multiplier value used in the velocity calculation. If the value is low the transition between walk and run state is more gradual.
- ***LockOnWallRotation***: if is turned on the Player can't rotate when touching a wall.
- ***GroundeCheckerThreshold:*** distance used to detect the floor. Differently from the **CharacterManager** script, the calculation here has only animation implications. If the value is high it will be more difficult for the character to reach the *Fall* animation state.
- ***ClimbThreshold***: the minimal value of vertical speed to assign the character the *Climb* animation state.

Platformer extensions

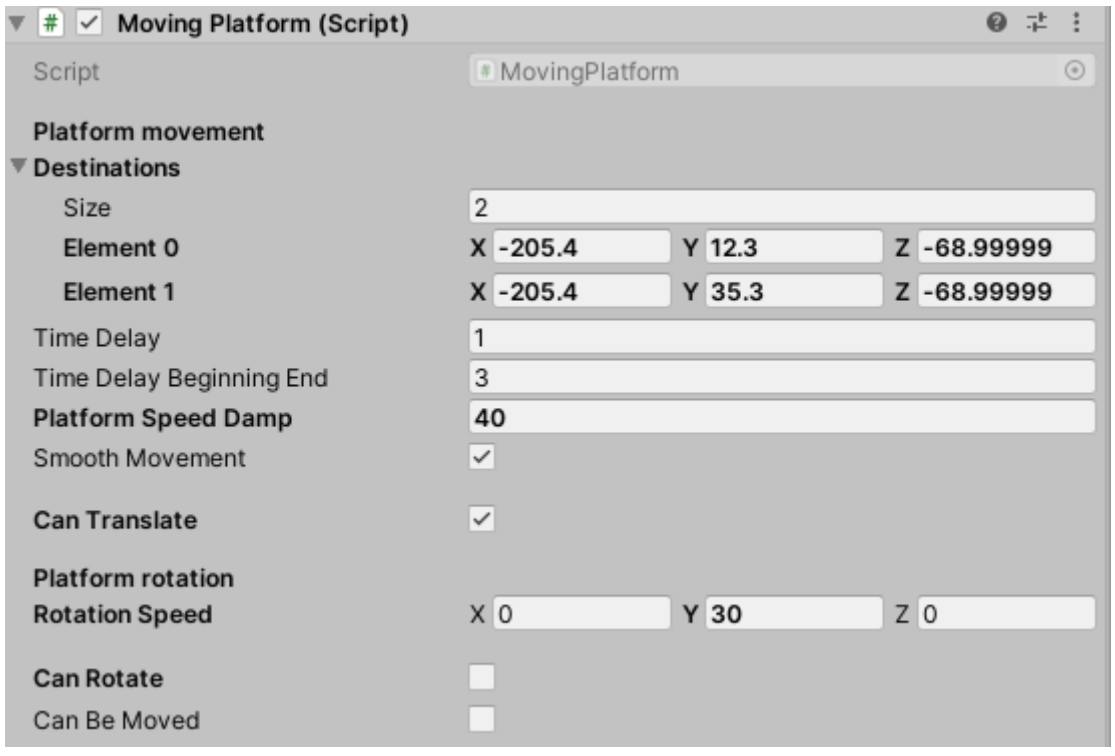
The asset has also a few common types of platforms and area effectors used in 3D platformers

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



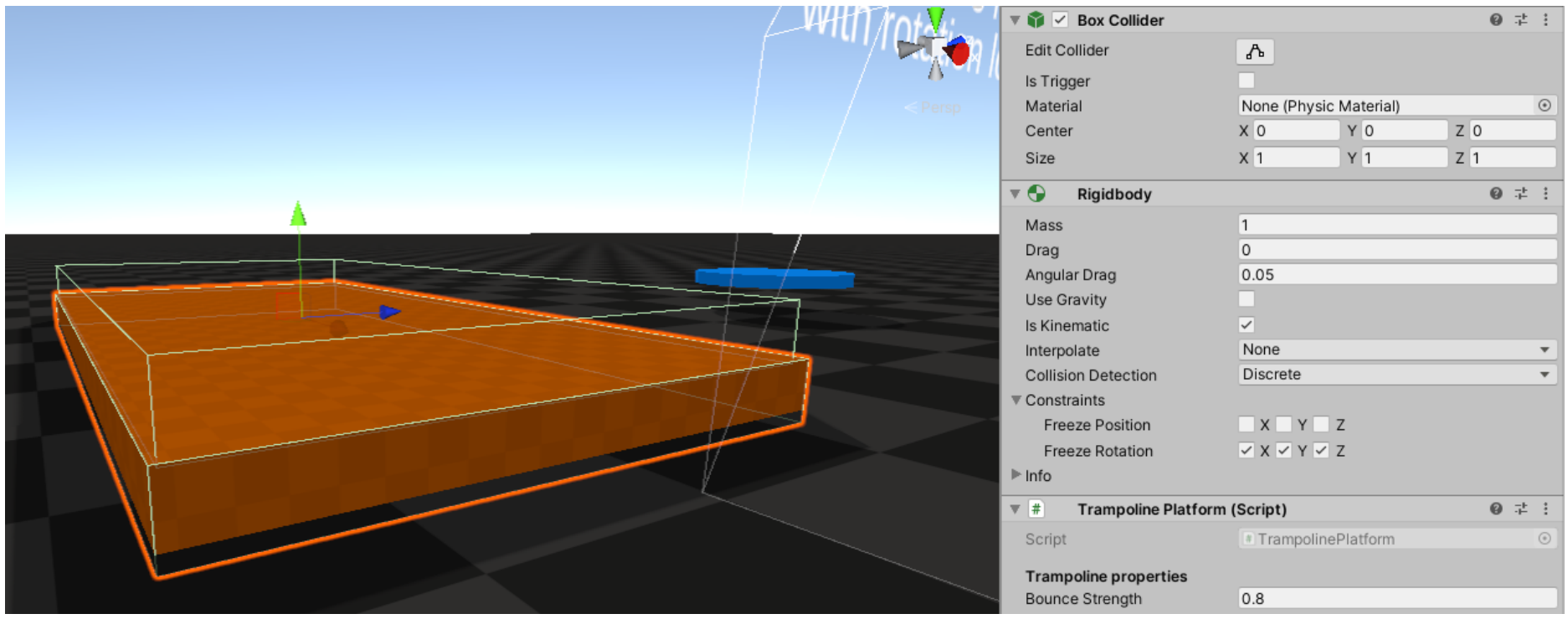
Their setup is quite simple, the platform requires a *Collider* that can interact with the player layer and a *Rigidbody*. The *Rigidbody* needs to be frozen in its rotation, its gravity needs to be disabled and it needs to be set to kinematic since its movement is managed by the script.



- ***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

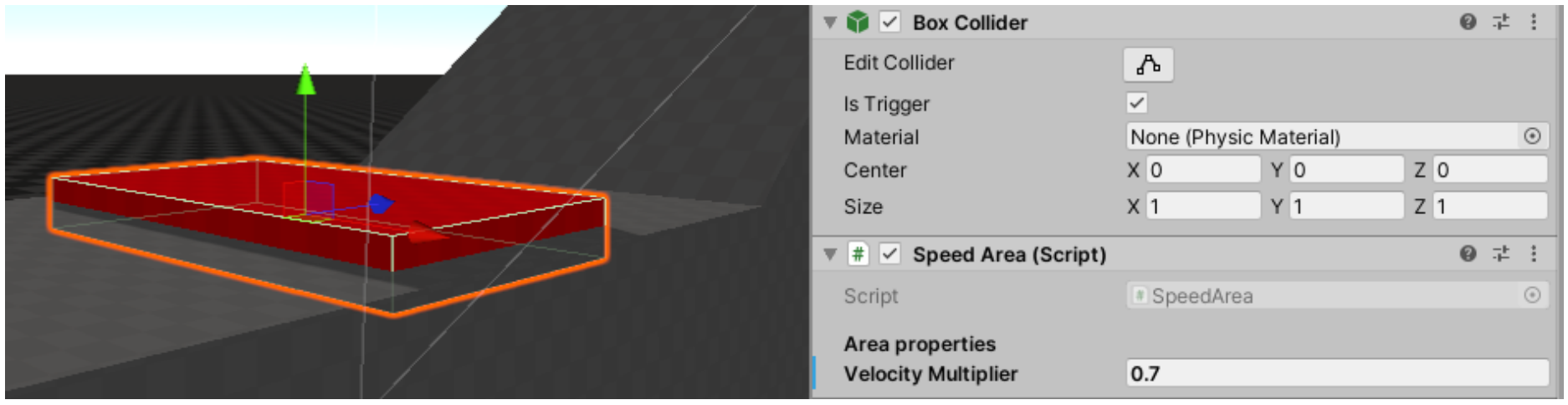
Trampoline Platforms

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.



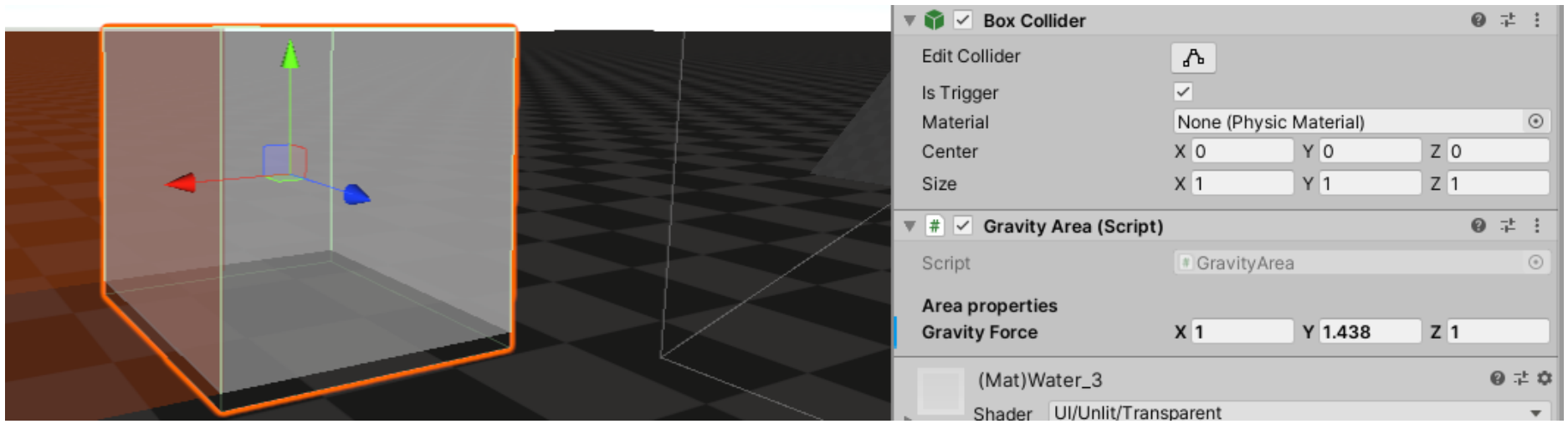
Speed Area

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



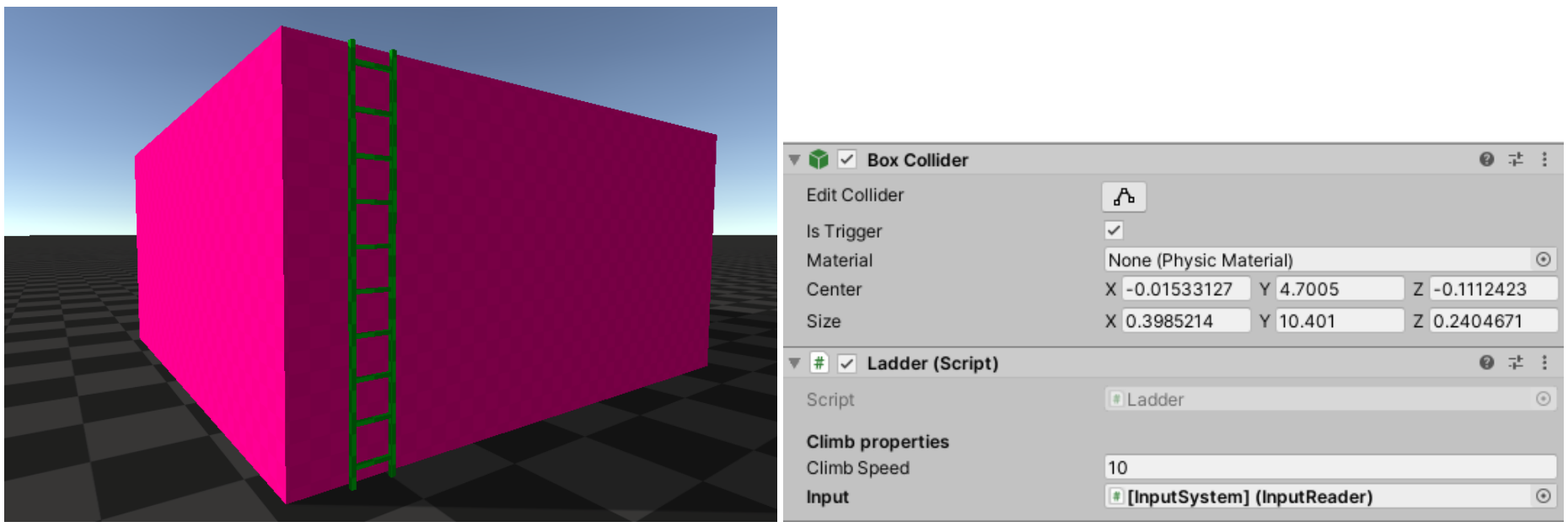
Gravity Area

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



Ladder

A variation of the Gravity Area that takes in the **InputReader** and translates the vertical and uses it to ascend vertically when inside the trigger area.



Contact

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

Cheers