Planet Platformer Controller Manual

Table of contents

1 Getting started 2

1.1 Project setup 2

1.1.1 Installing and activating Unity's new Input System 2

1.1.2 Creating the "Ground" layer 2

1.1.3 Disabling the project's global gravity 2

1.2 Scene setup 2

1.2.1 Attaching the scripts 2

1.2.2 Adding animations 3

1.2.3 Tagging your character (optional) 3

1.2.4 Adding a planet to the scene 3

2 Engine explanation 3

2.1 The "Ground" layer 3

2.2 Gravitational fields 4

2.3 Strongest Gravitational fields 4

2.4 Character alignment 4

2.5 Planets 5

2.6 Camera settings 5

3 Helpful tools 6

3.1 Gizmos and debugging rays 6

3.1.1 Scene view 6

3.1.2 Play mode 6

3.2 PP Controller Tools 7

4 Version history 7

5 Got questions or problems? 7

1 Getting started

This chapter covers everything you need to do for running the provided demos or creating your own scene.

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

Find the asset in Unity's Package Manager (*Window > Package Manager*) and press "Import". A warning pops up stating that the asset has Package Manager dependencies. Press "Install/Upgrade" to install missing dependencies, including Unity's Input System. If you have not had the latter in your project activated already, another warning will show up asking you to activate it. Press "Yes" and let the Unity Editor restart. This enables the Input System in your project (under *Edit > Project Settings > Player > Active Input Handling*). After the restart, you are also prompted with the TextMesh Pro Importer where you can decide if you want to additionally import the TMP Essentials which are used inside the provided UI prefabs and used in the demo.

1.1.2 Creating the "Ground" layer

Navigate to *Edit > Project Settings > Tags and Layers* and add layer "Ground" to the list – preferably as "User Layer 8" so that they are automatically assigned to the provided prefabs. Otherwise, you would have to assign the layer manually to all ground objects. You can search for "Ground" in the provided demo scenes to find all relevant game objects.

1.1.3 Disabling the project's global gravity

Navigate to the Physics settings of the project (*Edit > Project Settings > Physics*) and set the global Gravity to 0, 0, 0. Otherwise, this will impact the gravity calculations of the asset's physics engine and cause undesired side effects.

Now you are already good to go to run the demo.

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. PPController: Adding this component additionally adds the PPMotor and a Rigidbody component. While the PPController listens for user input via Unity's Input System (Input Actions can be found inside *Scripts > Inputs*), the PPMotor acts on them to move the character and fire animations. The PPMotor is based on the PPEngineObject component that takes care of all physics calculations.
b. PPCharacter: Adding this component gives your object more character-like properties, e.g. the possibility to punch or shoot the equipped pistol to deal, but also

suffer damage. This component automatically spawns a Capsule Collider if none was already found on the object.

c. *Camera* > PPCamera: Component for managing the camera according to the planet the character is currently walking on.

Of course, if you want to animate your character object, an Animator component is needed as well.

1.2.2 Adding animations

The asset provides an Animator Controller called "Character" with dummy animations that should make it easier to maintain and override it. Therefore, two additional Animator Overrides are provided on top – one for the character and one for the NPC. I recommend creating an own Animator Override and assigning this to your character's Animator component. If you are planning to import the "Astronaut Cartoony" package by Rheedo Animations that is also used for the asset showcase, override "CharacterOverride" will be automatically filled with the corresponding animations.

1.2.3 Tagging your character (optional)

Tag your character object with "Player" if you plan to use the provided NPC AI (component "PPNpc"). Otherwise, NPCs would ignore the character and not interact with it.

1.2.4 Adding a planet to the scene

Go to the "Prefabs" folder and drag a prefab from "Planets" into the scene. The prefabs here already include a default ground surface, a gravitational field and camera settings. See section 2 for more information on how they work.

Now place your character object on the surface of the chosen planet and rotate it so that its feet point to the ground. Enter the play mode.

2 Engine explanation

In this chapter, the implemented physics engine and its mechanics are explained. Usually, this in-depth knowledge is not needed if you are just using the provided prefabs. The goal is to provide knowledge to everyone who is interested, and make scene creation easier despite the engine's complexity. If any aspect is not simple enough described or missing, please contact me so I can improve the documentation.

In the following, the term "character" is used instead of the more general term "engine object". A character is an engine object, but an engine object is not necessarily a character. See for example the prefab "Simple Object" with component "PP Simple Object that inherits from script "PP Engine Object".

2.1 The "Ground" layer

All game objects that are assigned to the "Ground" layer are considered as surfaces the character can walk on. The character's "Grounded Check" only considers objects of this layer. Based on Unity's physics engine, these objects need to have colliders for collision detection. Note that Unity's Terrains are only rudimentary supported due to their high

triangle count (leads to more surface variance and bumpiness). You can check the Terrain game object with disabled collider in demo scene “Level” if you are interested in an example.

2.2 Gravitational fields

Gravitational fields play a vital role in the physics computations. They are game objects that consist of a Unity collider component and a gravitational field script. The former is set up as a trigger, the latter is distinguished as spherical, uniform or capsule. As the name suggests, spherical fields have a gravity vector towards their center, whereas the uniform fields’ gravity vector always points into the same direction. The capsule field is a combination of both. You can enable gizmos for visualizing the field and its field lines per gravitational field.

Each gravitational field has a priority and a gravity value. Gravitational fields with higher priority always overrule other fields with lower priority. The stronger the gravity of a field, the faster the character is drawn towards the object of layer “Ground”.

At every time, there is at most one gravitational field physically affecting the character object – it is the *strongest gravitational field*. If the character is not affected by any gravitational field for the number of seconds stored in public variable “Lost In Space Time Threshold”, it is considered as lost in space.

2.3 Strongest Gravitational fields

The strongest gravitational field is picked from all the fields which could currently affect the character, i.e. all field trigger colliders which contain the character object. Generally, a field with a higher priority wins over the other. If the priorities are the same, for each field a ray is cast from the character along the field lines detecting only hits with objects which have the “Ground” layer assigned. The Ground object whose surface is closest to the character game object wins. Note that this means that grounds and gravitational fields can act independently. After a strongest gravitational field is found, the gravity vector of that field is applied to the character game object. The direction of the gravity vector results from the field lines multiplied by the field’s gravity.

Note: As the range and therefore the influence of a gravitational field depends on the attached collider size, it is recommended to leave the collider component scales to 1 and scale the whole gravitational field prefab instead for convenience.

2.4 Character alignment

If the character is not grounded and a strongest gravitational field is found, the field’s gravity vector at the position of the character is taken and a ray is cast in its direction. The character’s up vector is then aligned with the normal vector of the hit Ground collider.

If the character is grounded and variable “Stick To Surface” is checked, the character stays aligned with the ground surface it is currently walking on. As a result, it is possible to run up vertical ramps as long as they have a smooth ground transition.

2.5 Planets

Planets consist of ground objects, i.e. objects on layer “Ground”, and gravitational fields. A planet can have multiples of both. Planets can be simple objects like the provided prefabs inside folder “Planets” or complex with multiple ground objects and many gravitational fields. However, it is generally not recommended to overlap two different types of gravitational fields within one planet. If needed and depending on the use case, side effects can sometimes be overcome via assigning different field priorities.

To group ground and gravitational field objects into one planet, unite them under a single root/parent object and assign the “Planet” component to it. Then, all children are together considered as one planet. Why is this important? Planets should represent smoothly connected objects that form something bigger. No transition logic is applied when the character travels through the fields of one planet. On the other hand, if the new strongest gravitational field is from another planet, the “change planet” logic kicks in and the character is aligned and moved to the new planet ground object.

2.6 Camera settings

Each planet object can have its own set of Camera Settings that are represented by a single component on the same object as the planet script. This component stores the camera behavior, generally position and rotation, which is applied when the character lands onto this planet.

There are 6 different types of camera settings to choose from:

- Static/Static: Both, camera position and rotation, are fixed and do not change
- Static/LookAtCharacter: Camera position stays, but the camera always rotates towards the character position
- Follow/Static: The camera follows the character’s position but its rotation is fixed
- Follow/Top-Down: The camera follows the character’s position from a top-down perspective
- NearestWorldAxis/Static: The camera always positions itself along the global world axis that is nearest to the character, looking towards the world center/origin
- NearestWorldAxis/LookAtCharacter: Same as above, but the camera always rotates towards the character

I recommend checking out the corresponding demo scenes to get a feeling for them and possible use cases.

3 Helpful tools

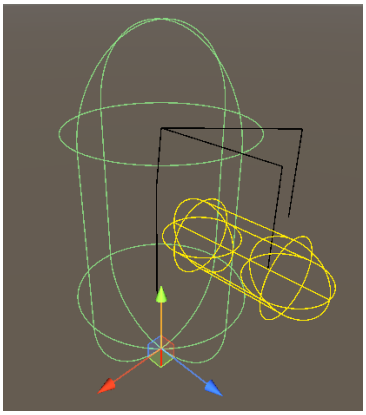
In this chapter, tools for easier scene setup and maintenance will be explained.

3.1 Gizmos and debugging rays

Almost every script implements multiple gizmos to make the setup of public variables easier. In the following, the most important ones are described.

3.1.1 Scene view

Gizmos related to the character:



- Red (PPMotor): Ray for the grounded check. Its length can be adjusted via public variable “Grounded Tolerance”. Turns green during play mode if the object is grounded.
- Black (PPMotor): Check rays for detecting ledges to hold on and climb up.
- Yellow (PPCharacter): Visualization of the punch height, radius and distance

Gizmos related to vector/gravitational fields:

- Check “Draw Field” to draw the whole field in the given color.
- Check “Draw Field Lines” to draw the given number of field lines in the given color. Field lines represent the “flow” of gravity inside the field

The different types of camera settings also have their own gizmos that are usually drawn at the scene’s origin/center as anchor.

3.1.2 Play mode

You can check public variable “Enable Debugging Rays” of the PPMotor to draw additional debugging rays during play mode:

- Magenta: Internally calculated movement direction
- Gray: Applied gravity vector
- Black: Velocity of the rigidbody
- Green: Internally calculated “forward” movement direction of the character (up/down input is mapped to this vector)
- Yellow: Internally calculated “right” movement direction of the character (left/right input is mapped to this vector)

Besides that, there is also the possibility to highlight the strongest gravitational field, i.e. the field that currently affects the character, via checking PPMotor parameter “Highlight Strongest Gravitational Field”. The total number of fields affecting the character, i.e. candidates for being the strongest gravitational field, can be displayed above the character’s head when “Show Affecting Fields Count” is enabled.

3.2 PP Controller Tools

In the editor menu “PP Controller Tools” at the top, you can find mass actions for drawing gizmos of all gravitational fields (and planets) in the scene. I recommend checking them out when building large, complex scenes to see what location is covered where by which planet.

4 Version history

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

5 Got questions or problems?

Feel free to send me an email to mail@johnstairs.com if your question is not covered by the FAQs on johnstairs.com/ppc. Please attach a screenshot showing your scene, the used variable values and occurring warning or error messages.

Best regards,

John Stairs