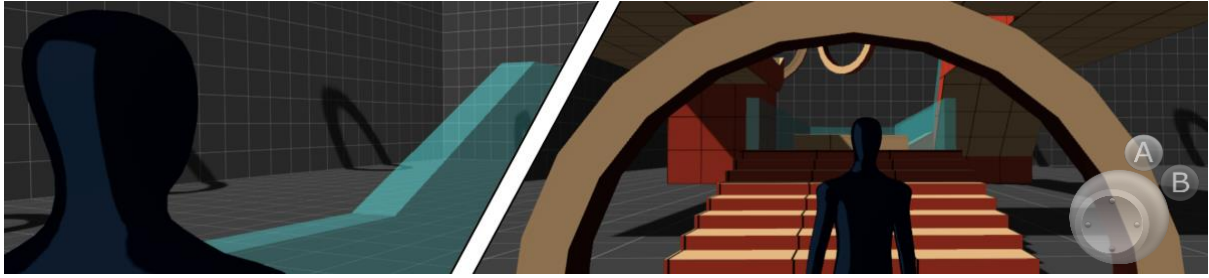


3RD PERSON CONTROLLER + FLY MODE (MOBILE)



Thank you for acquiring the **3rd Person Controller + Fly Mode (Mobile)** asset for the *Unity 3D* engine!

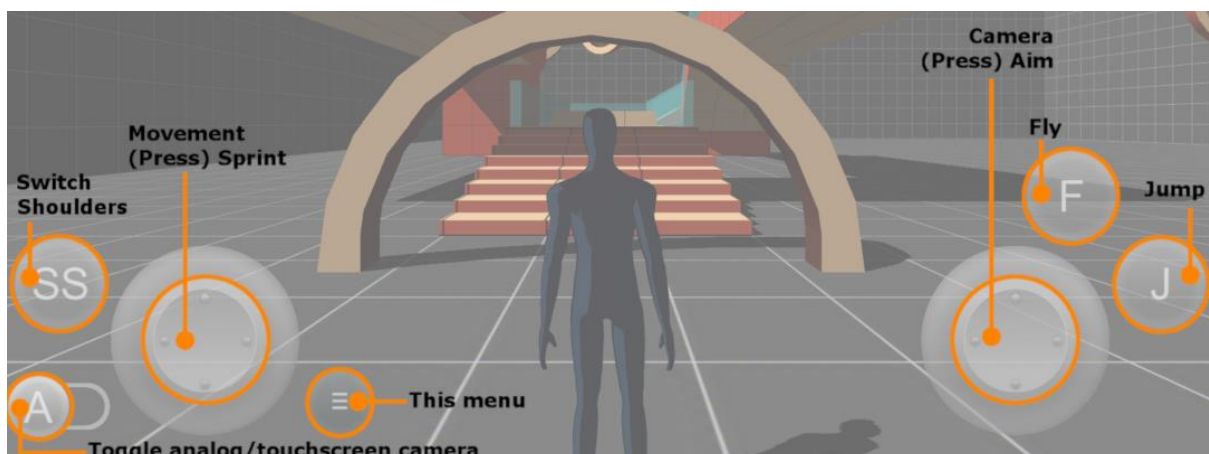
This asset provides a basic setup for a **3rd person** player controller on touchscreen devices. Includes scripts for the player basic movements, **camera orbit** and a Mecanim animator controller. Also includes ready to use virtual **buttons** and **analog**s.

The basic locomotion modes includes **walk, run, jump, sprint, aim & strafe**, and also an extra **fly** mode. Also comes with a dynamic **third person camera**, with full collision detection.

This asset features 2 types of touch controlled prefabs: **virtual buttons** and **virtual analogs**. The **VirtualInput** class also included, similar to Unity's standard *Input* class, contains all the basic static functions to control the player with the on screen virtual buttons and analogs (multi-touch supported).

SETUP NOTES

The demo scenes present two basic setups for the inputs used on the containing player behaviours. The default scene controls are mapped as follows:




This scheme uses a virtual analog to control the camera, similar to gamepads.

Also, there is an alternate scheme, presented on the other scene:



This alternate scheme uses the device touchscreen to control the camera.

You can press the  virtual button to see the input commands on the example scenes.

A detailed tutorial on how to setup the essential files on a custom project and how to properly configure the virtual input prefabs is provided on a video that can be accessed through the following link:

[Tutorial: How to setup on a custom project](#)

There are two types of prefabs in this asset: **virtual buttons** and **virtual analogs**. In order to setup custom controls, you can simply drag these prefabs inside a UI Canvas on the scene.

If you want to use a custom behaviour you acquired or created, or if you want to adapt any other script to properly work on a mobile device, you just need to remap the calls to the static functions of the Unity's default *Input* class for the equivalent static functions of the *VirtualInput* class, that comes with this asset:

<pre>h = Input.GetAxis("Horizontal"); v = Input.GetAxis("Vertical");</pre>	//	<pre>h = VirtualInput.GetAxis("Horizontal"); v = VirtualInput.GetAxis("Vertical");</pre>
--	----	--

More details upon the *VirtualInput* class functions, that is handled by the prefabs, are presented on a section below, which provides a detailed view of the class.

VIRTUAL BUTTON

A touch controlled button. After dragging this prefab to the scene, you need to define the following customization properties:

- **Source Image:** defines an image to represent the button on screen.
- **Color:** the color of the button when released.
- **Input Name:** the name of the input axis this button will handle.
- **Press Color:** the color this button will have when pressed.
- **Text (in Text):** defines the label to show on the virtual button.

VIRTUAL ANALOG

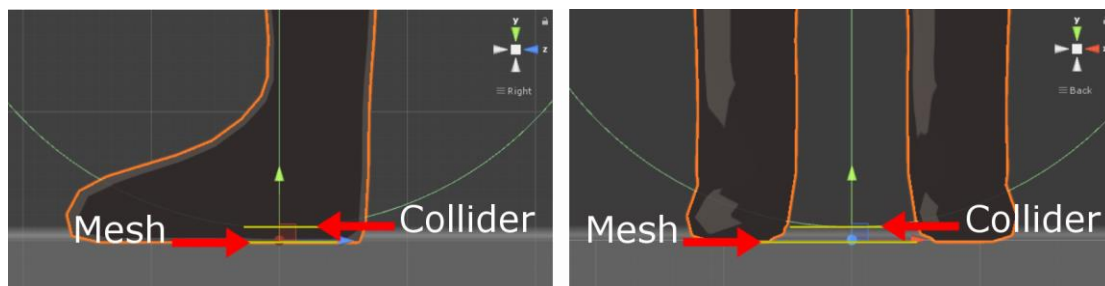
A touch controlled analog, similar to the present on modern gamepads. After dragging this prefab to the scene, you need to define the following customization properties:

- **Source Image (in analog):** defines an image to represent the analog base on screen.
- **Color (in analog):** defines the color of the analog base.
- **Source Image (in stick):** defines an image to represent the analog stick on screen.
- **Color (in stick):** defines the analog stick color when not pressed.
- **Input X Axis:** the input axis name controlled by the horizontal movement of the virtual analog.
- **Invert X:** whether or not to invert the Horizontal axis.
- **Input Y Axis:** the input axis name controlled by the vertical movement of the virtual analog.
- **Invert Y:** whether or not to invert the Vertical axis.
- **Dead:** size of the virtual analog dead zone.
- **Sensitivity:** the speed to move towards target value, considering the analog stick position.
- **Button Name:** name of the virtual button (activated by double touch, simulates the press analog action).
- **Press Color:** the virtual analog stick color when pressed.

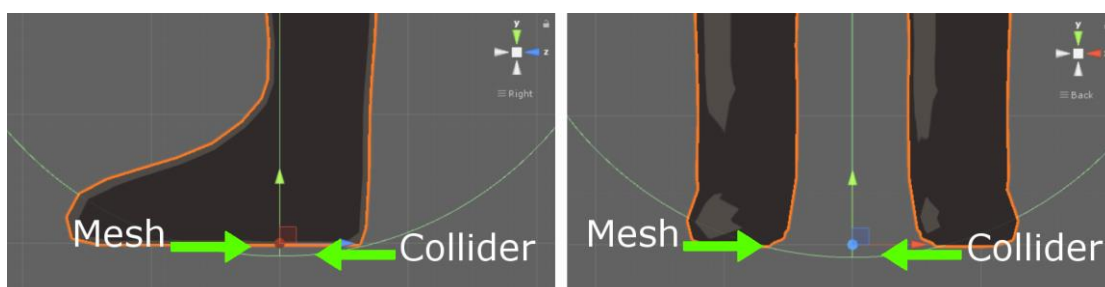
Warning: do not modify the *Anchor Preset* defined for a virtual analog, or it may not work as expected.

OTHER IMPORTANT NOTES

When configuring the *Capsule Collider* on your own character avatar, you may carefully avoid the following situation:



If the collider end is positioned above the mesh end, the character will get stuck when trying to move. Do not let the avatar mesh surpass the collider! The correct position is set when the bottom of the collider is at the same level or under the mesh end:



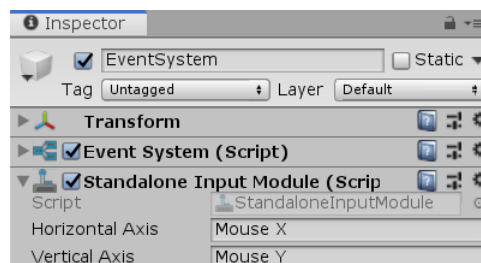
THIRD PERSON ORBIT CAM BASIC MOBILE

The **ThirdPersonOrbitCamBasicMobile** class manages all the third person camera features, such as orbiting around the player, FOV changes, and zooming in/out to avoid collision. This script has the following external parameters:

- **Player:** a reference to the player's game object.
- **Pivot Offset:** the default offset to point the camera when orbiting the player.
- **Cam Offset:** the default offset to relocate the camera when orbiting the player.
- **Smooth:** the default camera movement responsiveness factor.
- **Horizontal Aiming Speed:** the speed of camera orbit on the horizontal plane.
- **Vertical Aiming Speed:** the speed of camera orbit on the vertical plane.
- **Max Vertical Angle:** the maximum angle to limit camera orbit on the vertical plane.
- **Min Vertical Angle:** the minimum angle to limit camera orbit on the vertical plane.
- **X Axis:** the default horizontal axis input name (for virtual analogs).
- **Y Axis:** the default vertical axis input name (for virtual analogs).
- **Cam Touchscreen:** set this if you want to control the camera using the device touchscreen.

Hint: you can setup a *virtual analog*, containing input axes with the names defined in the *X Axis* and *Y Axis* parameters, to control the third person camera. Also remember to adjust the *virtual analog* axes *sensitivity* and *dead zone* to match your needs.

Alternatively, you can use the device touch to control the camera. Remember to set properly the horizontal and vertical axis parameters on the Unity *Event System's Input Module* component (the default ones used by devices with touchscreen is *Mouse X* and *Mouse Y*).



BASIC BEHAVIOUR MOBILE

The **BasicBehaviourMobile** class manages which player behaviour is currently active or overriding the active one, and call its local functions. This behaviour also contains a basic setup and common functions used by all the player behaviours. This script has the following external parameters:

- **Player Camera:** a reference to the player camera's game object.
- **Turn Smoothing:** the turn speed when moving to match camera facing.
- **Sprint FOV:** the camera *Field of View* to change when player is sprinting.
- **Sprint Button:** the default sprint action input name.

Hint: this behaviour uses the Unity's default *Horizontal* and *Vertical* input axes to control the player movement. You can configure a *virtual analog* containing input axes with these names, in order to move the player.

MOVE BEHAVIOUR MOBILE

The **MoveBehaviourMobile** class corresponds to walk, run and jump behaviour, it is generally the default behaviour. This script has the following external parameters:

- **Run Speed:** the player's default running speed (controlled by a Blend Tree).
- **Sprint Speed:** the player's default sprinting speed (controlled by a Blend Tree).
- **Speed Damp Time:** the delay to change between the locomotion animations when changing speed.
- **Jump Height:** the default jump height reached.
- **Jump Inertial Force:** the default inertial horizontal force applied to the player when on air, between jumping and landing. The higher it is, the longer the jump distance will be.

Hint: this behaviour uses Unity's default *Jump* input axis to handle the jumping action. You can configure a *virtual button* to control it.

AIM BEHAVIOUR BASIC MOBILE

The **AimBehaviourBasicMobile** class handles the aim and strafe movements. The following external parameters are present:

- **Aim Button:** the default aim action input name.
- **Shoulder Button:** the default switch shoulders action input name.
- **Crosshair:** the default aiming crosshair. If no parameter is passed, no crosshair is shown.
- **Aim Turn Smoothing:** the speed of turn response when aiming to match the player's orientation with camera facing.
- **Aim Pivot Offset:** the offset to repoint the camera when aiming.
- **Aim Cam Offset:** the offset to relocate the camera when aiming.

Hint: you can setup a *virtual button* to control the aiming action, using the *Aim Button* parameter as the controlled input.

FLY BEHAVIOUR

The **FlyBehaviour** class handles fly action. The following external parameters are present:

- **Fly Button:** the default fly toggle input name.
- **Fly Speed:** the default flying speed.
- **Sprint Factor:** how much sprinting affects flying speed.
- **Fly Max Vertical Angle:** custom angle to clamp the camera vertically when flying mode is active.

Hint: you can setup a *virtual button* to control the flying, using the *Fly Button* parameter as the controlled input.

VIRTUAL INPUT

The **VirtualInput** class is a simplified version of the Unity's standard *Input* class, containing the basic static functions that handles inputs by virtual buttons and axes. The following equivalent functions are present:

- **GetAxis:** returns the value of the virtual analog axis passed as parameter.
- **GetAxisRaw:** returns the value of the virtual analog axis passed as parameter, with no smoothing filtering applied.
- **GetButton:** returns true while the corresponding virtual button or virtual analog is held down.
- **GetButtonDown:** returns true during the frame the user pressed down the corresponding virtual button.
- **GetButtonUp:** returns true the first frame the user releases the corresponding virtual button.
- **GetKey:** returns true while the user holds down the corresponding key.
- **GetKeyDown:** returns true during the frame the user starts pressing down the corresponding key identified by *name*.
- **GetKeyUp:** returns true during the frame the user releases the corresponding key.

Hint: Setup a virtual key using the virtual button prefab

ACKNOWLEDGMENT

The author acknowledges some third-party assistance that facilitated the production of the demo examples within this asset. This includes the *Carnegie-Mellon University* mocap library and the fellows that contributes to the *Unity Answers* website.

DISCLAIMER

This document is part of the **3rd Person Controller + Fly Mode (Mobile)** asset, that can be acquired through official channels – such as the *Unity Asset Store*.

If you obtained this asset through unofficial ways, I kindly ask you to consider giving support for the great assets to come!

Did you enjoy the asset? Please consider leaving a review on the *Unity Asset Store*, it is really important and will be very appreciated!

[Support contact](#)

[Author's page](#)