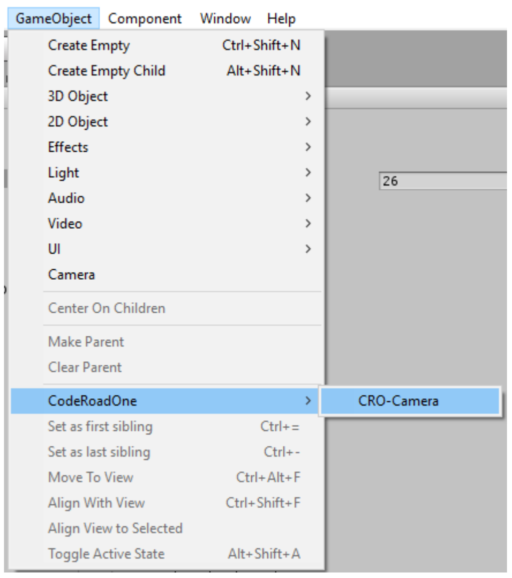
CODE ROAD ONE CAMERA

TABLE OF CONTENTS

ADDING CAMERA TO THE SCENE

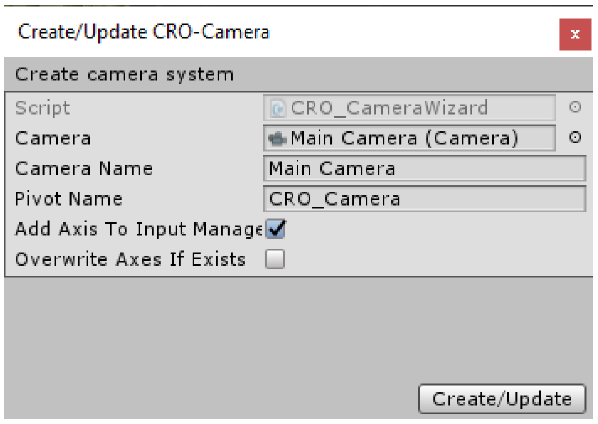
AUTOMATIC SETUP

As soon as you import the CRO Camera into your project, a new menu will be available:



From that menu you have access to **Camera Creation Wizard**. You can find it in: **GameObject->CodeRoadOne-> Camera**.

This will open this window:



Script	CRO_CameraWizard
Camera	Main Camera (Camera)
Camera Name	Main Camera
Pivot Name	CRO_Camera
Add Axis To Input Manage	✓
Overwrite Axes If Exists	☐

The wizard will try to find the first camera in the scene and add it to the '**Camera**' field. In case this is not the right camera, drag and drop from your hierarchy the correct camera that you want the script to control.

If there is no camera in the scene, a new camera will be created with the default settings.

You can rename camera and pivot in the respective fields. The pivot is another game object that has the script CRO_Camera attached, and is also the parent of the camera.

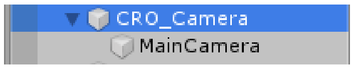
If “Add Axis To Input Manager” is selected the axis used by the CRO Camera will be automatically added to the Input Manager. If you don’t select the “Overwrite Axes If Exists” the wizard will not change the existing setup. This could cause some problems with the setup and manual fix will be needed. In general, the input axes names are really specific to the camera so not overwrite will happen if you just imported the camera. In case you click second time, overwrite will be unavoidable.

A note here: If you rename an axis after you added them to the input manager and decide to re-add axis to the input manager the old axes are still going to be there.

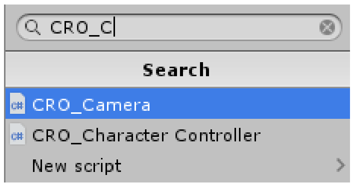
After that, you just need to click the Create button and everything will be set up for you.

MANUAL SETUP

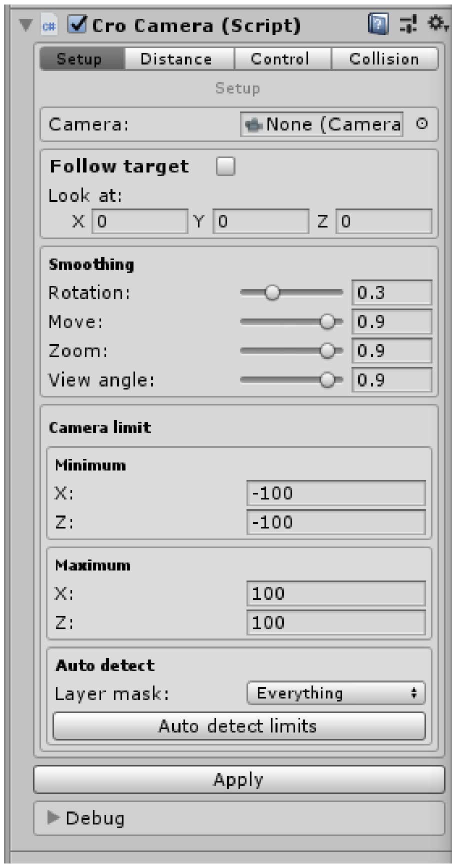
1. Create an empty game object in the hierarchy (the pivot). Name it as you want (for example: CRO_Camera).
2. Add as a child the scene camera (either you create a new camera, or you drag and drop the scene camera). In this moment you should have something like this:



3. Select the pivot.
4. Click on Add Component
5. Start typing CRO_Camera



6. Select CRO_Camera from the list.
7. Now you will have this:



Cro Camera (Script)

Setup | Distance | Control | Collision

Setup

Camera: None (Camera

Follow target ☐

Look at: X 0 Y 0 Z 0

Smoothing
Rotation: 0.3
Move: 0.9
Zoom: 0.9
View angle: 0.9

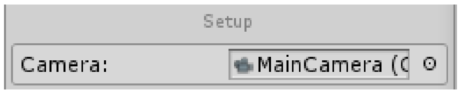
Camera limit

Minimum
X: -100
Z: -100

Maximum
X: 100
Z: 100

Auto detect
Layer mask: Everything

8. In the field camera, choose the camera that you added as child to the pivot:



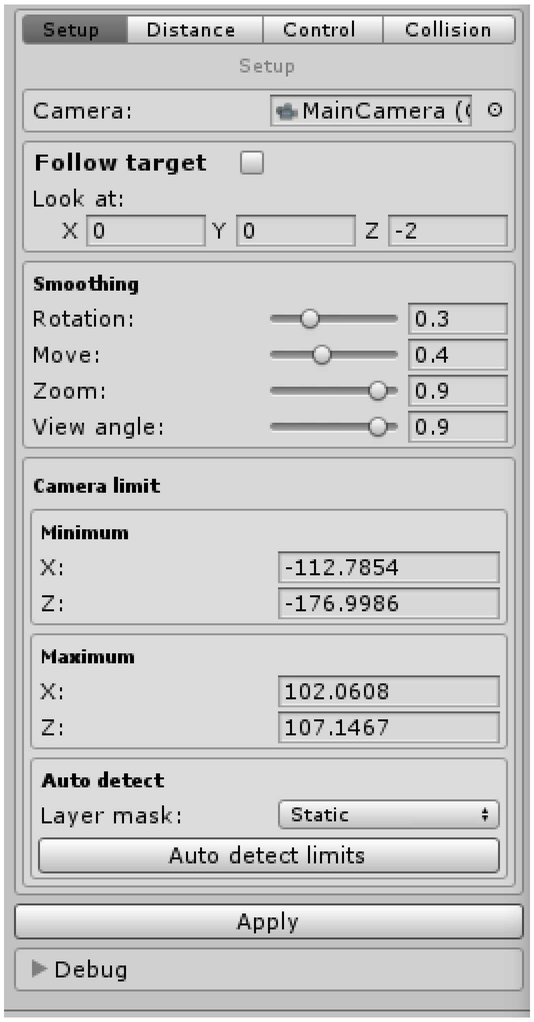
Setup

Camera: MainCamera (C

9. Now you can move to the next step on how to setup the camera.

SETTING UP THE CAMERA

When you select the CRO_Camera, you will see our custom editor:



Camera: MainCamera (

Follow target ☐

Look at:

X 0 Y 0 Z -2

Smoothing

Rotation:	0.3
Move:	0.4
Zoom:	0.9
View angle:	0.9

Camera limit

Minimum

X:	-112.7854
Z:	-176.9986

Maximum

X:	102.0608
Z:	107.1467

Auto detect

Layer mask: Static

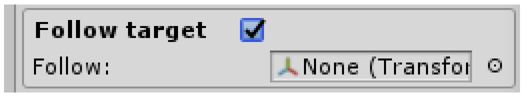
Let's check out all options in detail:

SETUP

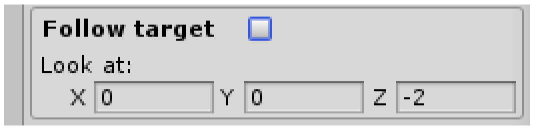
The first menu displayed is the setup of the camera. You can see the camera that is being used in the Camera field. Then you have the first option:

FOLLOW TARGET

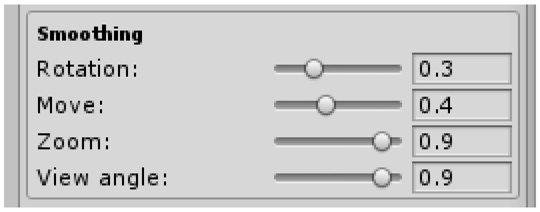
When selected, you will have the option to select which game object to follow


Follow target ✓
Follow: None (Transfor

When not selected, you can choose what position the camera should look at

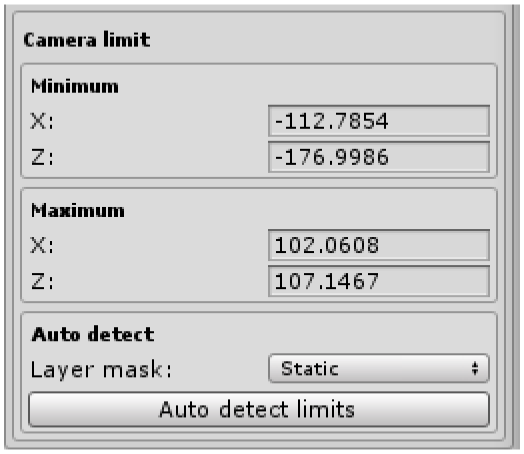

Follow target
Look at:
X 0 Y 0 Z -2

SMOOTHING


Smoothing
Rotation: 0.3
Move: 0.4
Zoom: 0.9
View angle: 0.9

This section allows you to set up how fast the camera will rotate to the targeted rotation, how fast it will get to the targeted position, how fast it will zoom in and out and how fast it will rotate to see the target position in case it needs to move up if a collision occurs.

CAMERA LIMIT


Camera limit

Minimum

X: -112.7854

Z: -176.9986

Maximum

X: 102.0608

Z: 107.1467

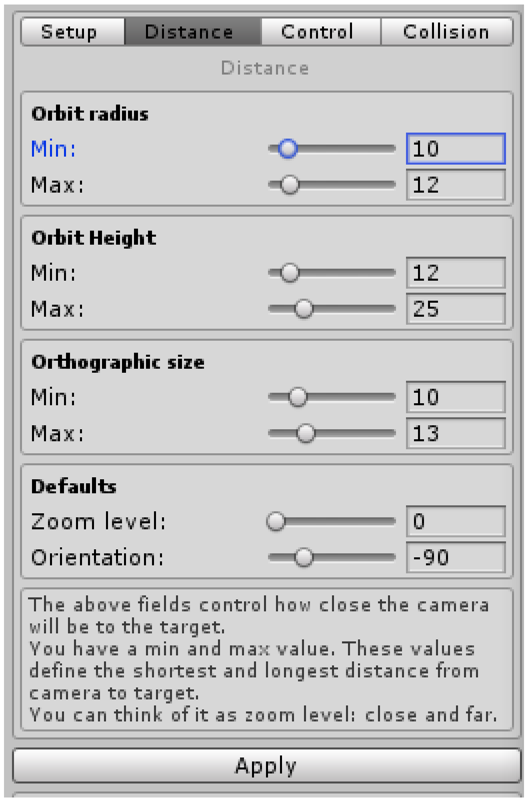
Auto detect

Layer mask: Static

In this section you can specify in what area the camera is allowed to move, the collision bounds. You can manually set it up or let it autodetect the world bounds.

For autodetected limits, you should select the layer mask that will be used when checking for objects. For example, you could have a collision layer that is used by the game play, that will dictate the actual world bounds for the camera move, and another layer that is defining an outer area for cosmetics or far away objects only, area that the player cannot in fact reach.

DISTANCE



Orbit radius
Min: 10
Max: 12

Orbit Height
Min: 12
Max: 25

Orthographic size
Min: 10
Max: 13

Defaults
Zoom level: 0
Orientation: -90

The above fields control how close the camera will be to the target.
You have a min and max value. These values define the shortest and longest distance from camera to target.
You can think of it as zoom level: close and far.

In this section you control how the camera is positioned in relation to the targeted position or object.

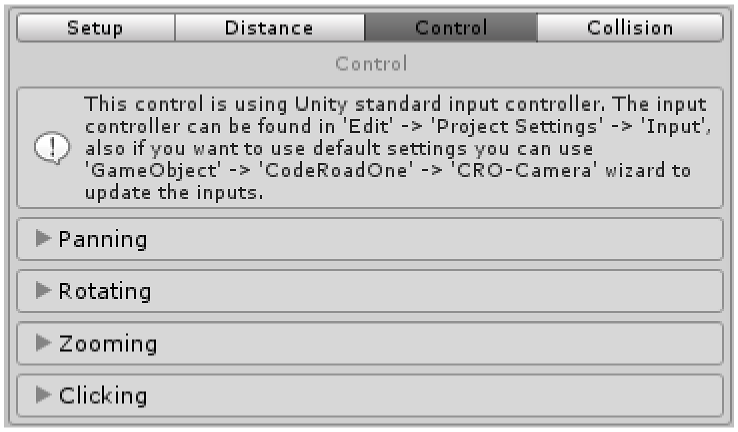
For each of the **Orbit radius**, **Orbit height** and **Orthographic size** you have two values. The two values indicate where the camera is located when zoomed in (min) and where the camera is located when zoomed out (max).

The **Orthographic size** is used only when you have an orthographic camera, which requires a specific way to give the illusion of zooming in and out.

The **Defaults** fields specify what is the zoom level and from what direction the camera is looking to the target when you press play.

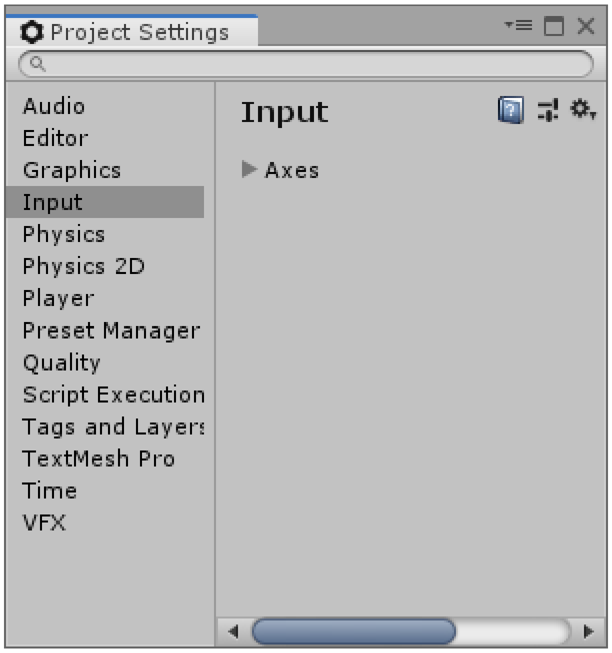
The **Apply** button will take the information from the target, distances and defaults and adjust the camera to look at that position immediately.

CONTROL



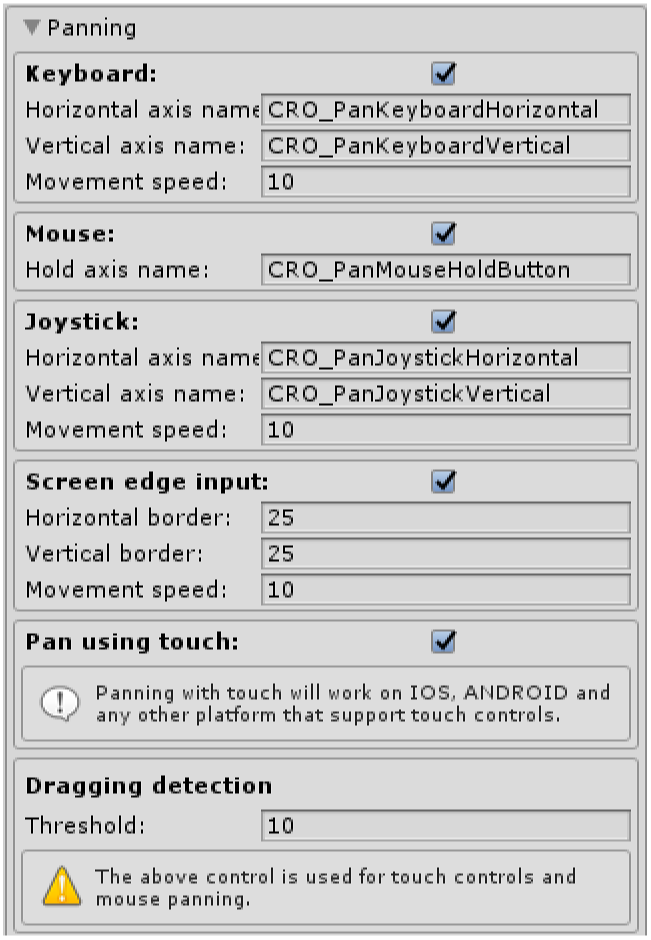
This control is using Unity standard input controller. The input controller can be found in 'Edit' -> 'Project Settings' -> 'Input', also if you want to use default settings you can use 'GameObject' -> 'CodeRoadOne' -> 'CRO-Camera' wizard to update the inputs.

In this section you define how the camera will be controlled by panning, rotating and zooming. The default mapping is using Unity Standard Input. This can be accessed from the Edit -> Project Settings -> Input Tab:



We also provide default mapping for touch control. The touch control should work on all devices that supports touch control.

PANNING



▼ Panning

Keyboard: ✓
Horizontal axis name: CRO_PanKeyboardHorizontal
Vertical axis name: CRO_PanKeyboardVertical
Movement speed: 10

Mouse: ✓
Hold axis name: CRO_PanMouseHoldButton

Joystick: ✓
Horizontal axis name: CRO_PanJoystickHorizontal
Vertical axis name: CRO_PanJoystickVertical
Movement speed: 10

Screen edge input: ✓
Horizontal border: 25
Vertical border: 25
Movement speed: 10

Pan using touch: ✓
Panning with touch will work on IOS, ANDROID and any other platform that support touch controls.

Dragging detection
Threshold: 10
The above control is used for touch controls and mouse panning.

In this section you control how or if you allow to pan the camera.

The panning only works if the camera is *NOT* following a target.

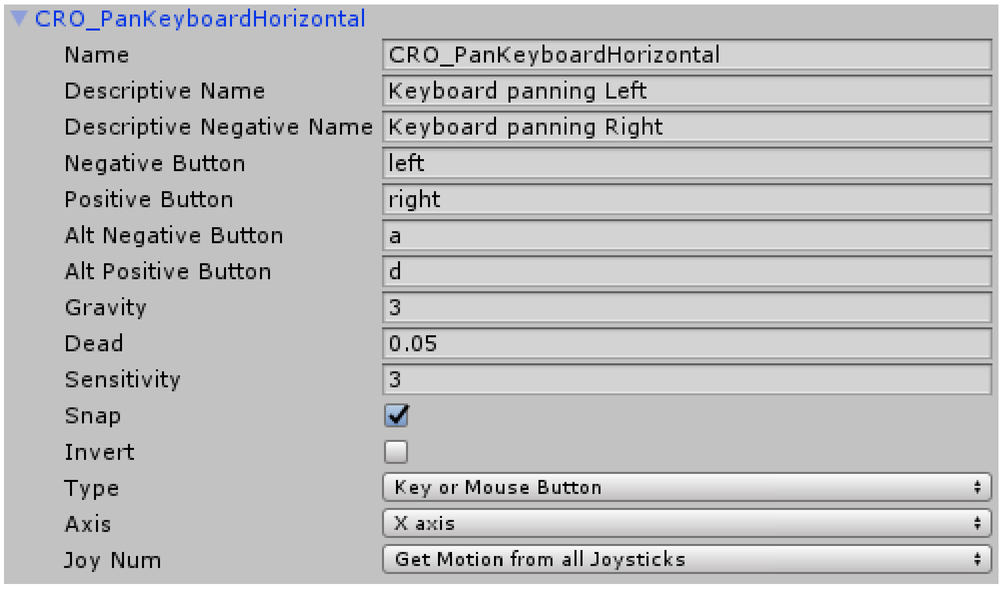
PAN WITH KEYBOARD

If you enable panning by keyboard you will need to set up the axis names for horizontal and vertical. The axis can be added inside the Input Tab.

When panning with the keyboard you need to specify how fast the camera will move when you press the keys. This can be set by modifying the value in the Movement speed.

In the demo scene you will find the following settings in the Input Tab:

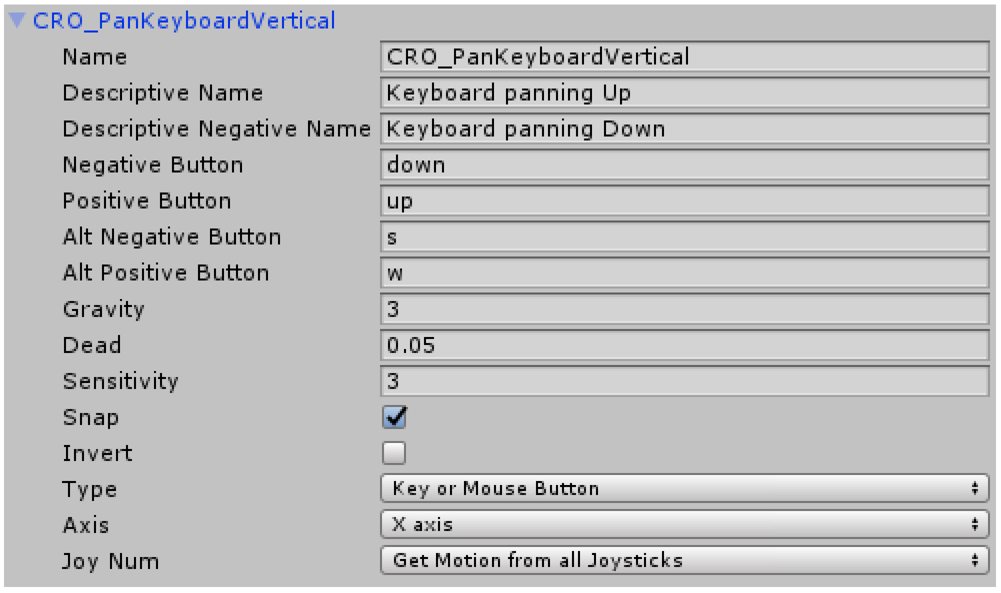
PAN KEYBOARD HORIZONTAL



CRO_PanKeyboardHorizontal

Name	CRO_PanKeyboardHorizontal
Descriptive Name	Keyboard panning Left
Descriptive Negative Name	Keyboard panning Right
Negative Button	left
Positive Button	right
Alt Negative Button	a
Alt Positive Button	d
Gravity	3
Dead	0.05
Sensitivity	3
Snap	✓
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

PAN KEYBOARD VERTICAL

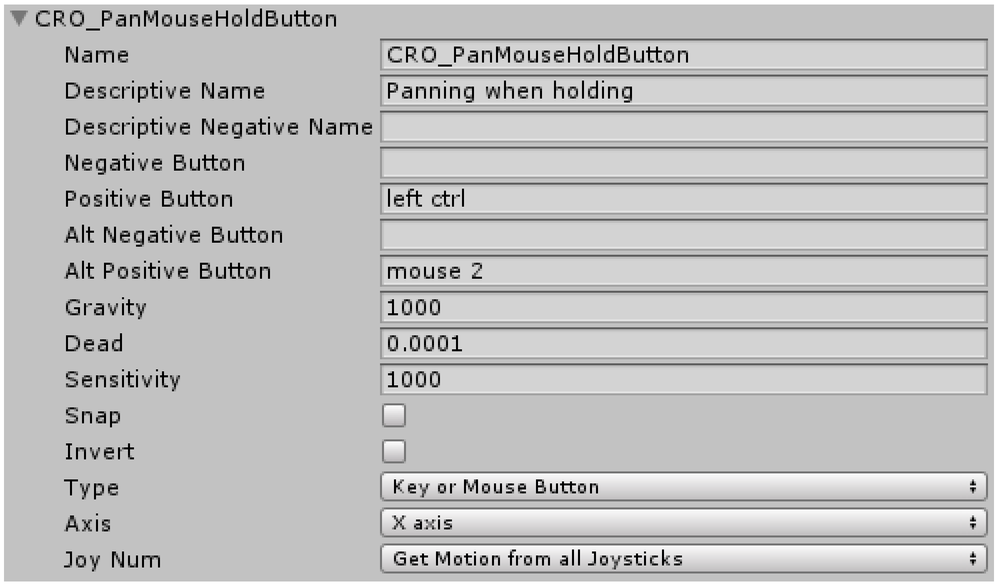


CRO_PanKeyboardVertical

Name	CRO_PanKeyboardVertical
Descriptive Name	Keyboard panning Up
Descriptive Negative Name	Keyboard panning Down
Negative Button	down
Positive Button	up
Alt Negative Button	s
Alt Positive Button	w
Gravity	3
Dead	0.05
Sensitivity	3
Snap	✓
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

As you can see from the above screenshots, the default movement keys are the arrows and A, S, D, W.

PAN WITH MOUSE



▼ CRO_PanMouseHoldButton

Name	CRO_PanMouseHoldButton
Descriptive Name	Panning when holding
Descriptive Negative Name	
Negative Button	
Positive Button	left ctrl
Alt Negative Button	
Alt Positive Button	mouse 2
Gravity	1000
Dead	0.0001
Sensitivity	1000
Snap	☐
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

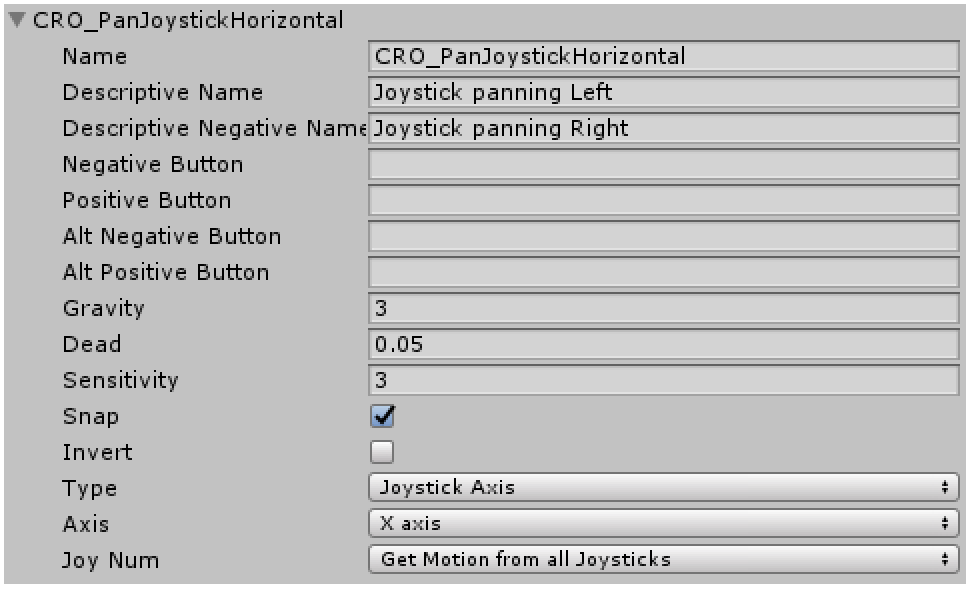
To pan with the mouse, you need to specify when this will happen. By default, you will start to pan with the mouse when you hold down the middle mouse button or left ctrl.

PAN WITH JOYSTICK

We also provide a way to allow you to pan the camera with the joystick. As you did for the set up of the keyboard, you will need to set up the names for the axis for horizontal and vertical movement, and set up the speed of movement as well.

This is how the axis are setup (in the Input Tab) in the demo scene, to use the left stick in the case of dual controller. **Please note that the joystick input controller was tested only on Windows.**

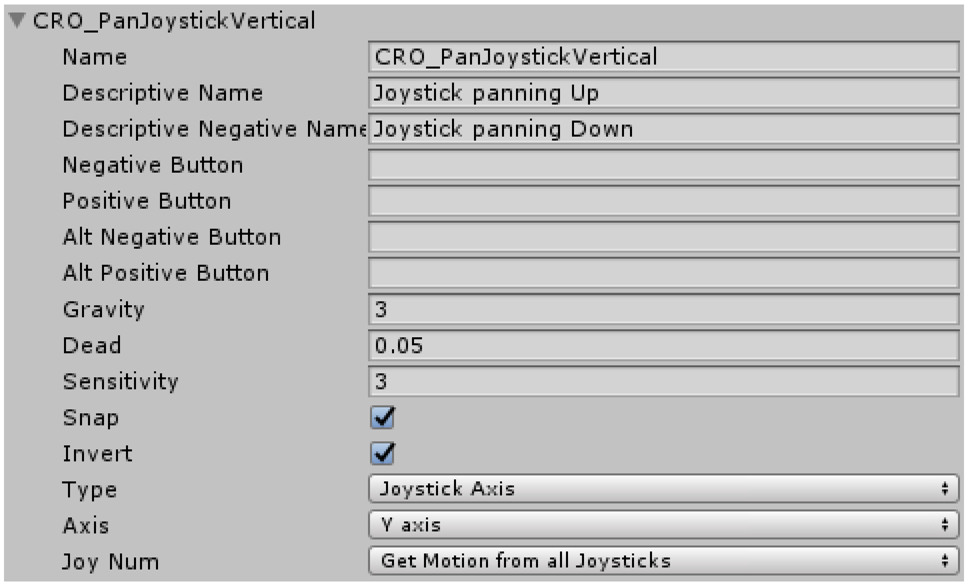
PAN JOYSTICK HORIZONTAL



▼ CRO_PanJoystickHorizontal

Name	CRO_PanJoystickHorizontal
Descriptive Name	Joystick panning Left
Descriptive Negative Name	Joystick panning Right
Negative Button	
Positive Button	
Alt Negative Button	
Alt Positive Button	
Gravity	3
Dead	0.05
Sensitivity	3
Snap	✓
Invert	☐
Type	Joystick Axis
Axis	X axis
Joy Num	Get Motion from all Joysticks

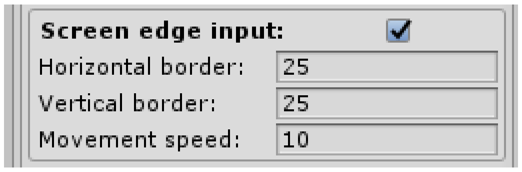
PAN JOYSTICK VERTICAL



CRO_PanJoystickVertical

Name	CRO_PanJoystickVertical
Descriptive Name	Joystick panning Up
Descriptive Negative Name	Joystick panning Down
Negative Button	
Positive Button	
Alt Negative Button	
Alt Positive Button	
Gravity	3
Dead	0.05
Sensitivity	3
Snap	✓
Invert	✓
Type	Joystick Axis
Axis	Y axis
Joy Num	Get Motion from all Joysticks

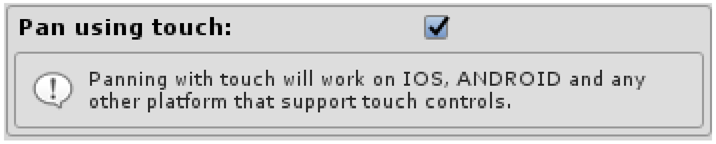
SCREEN EDGE INPUT



Screen edge input:	✓
Horizontal border:	25
Vertical border:	25
Movement speed:	10

Panning by screen edge input refers to panning by moving the mouse to the edge of the screen. If enabled, you can specify the thickness of the border at top and bottom (vertical) and left and right (horizontal). Also, you need to specify how fast the camera will move if the mouse gets in the specified area, close to the edge of the screen.

PAN USING TOUCH

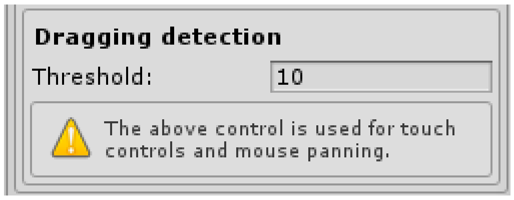


Pan using touch: ✓

Panning with touch will work on IOS, ANDROID and any other platform that support touch controls.

If you enable it, you will be able to pan on IOS and Android and any other platform that supports touch controls. To pan with touch, you will need to use one finger and drag around.

DRAG DETECTION



Dragging detection

Threshold:	10

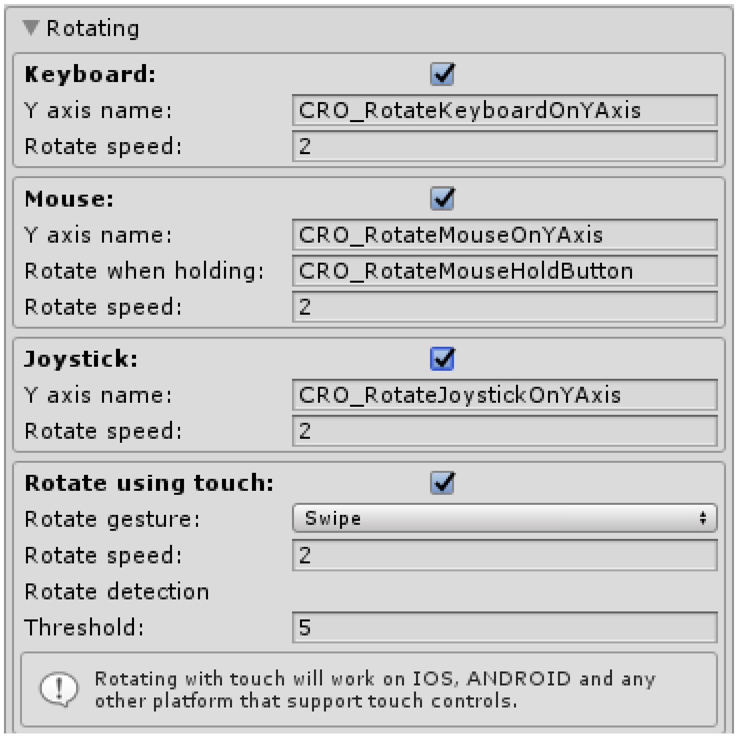
The above control is used for touch controls and mouse panning.

This is a very important parameter. This is used when panning with the mouse on a desktop, and on mobiles using touch controls.

Threshold is defining the distance in pixels you must move the mouse on desktop or finger on touch screen, before we consider that you are actually trying to drag. This is useful in case you want to add support to detect click, or some other useful controls.

If you don't want the user to be able to move the camera by panning, you can disable the panning by deselecting the Keyboard, Mouse, Joystick, Screen edge and Touch controls.

ROTATING



Rotating

Keyboard:	✓
Y axis name:	CRO_RotateKeyboardOnYAxis
Rotate speed:	2
Mouse:	✓
Y axis name:	CRO_RotateMouseOnYAxis
Rotate when holding:	CRO_RotateMouseHoldButton
Rotate speed:	2
Joystick:	✓
Y axis name:	CRO_RotateJoystickOnYAxis
Rotate speed:	2
Rotate using touch:	✓
Rotate gesture:	Swipe
Rotate speed:	2
Rotate detection Threshold:	5

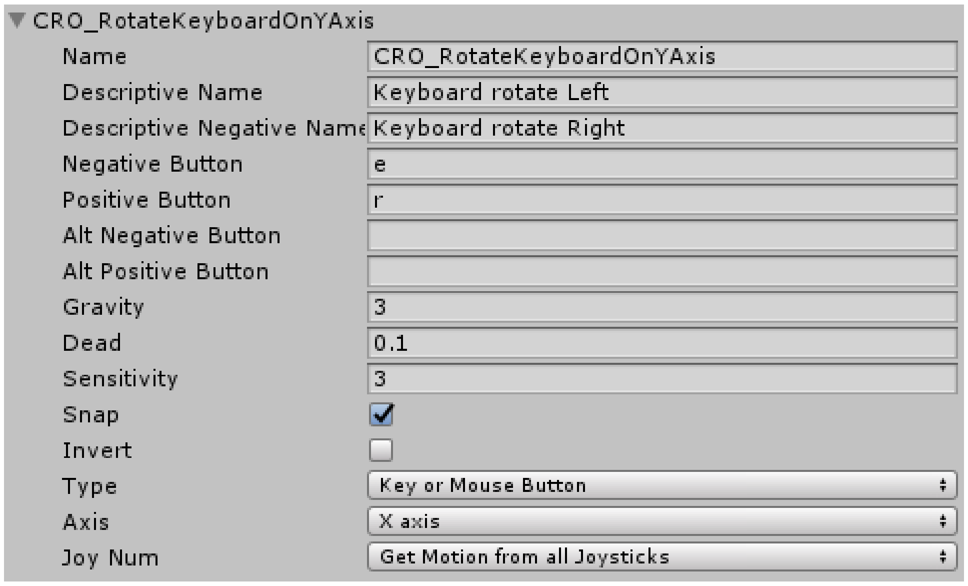
Rotating with touch will work on IOS, ANDROID and any other platform that support touch controls.

Here you have similar options as you had for panning. You can enable rotating around the target on the Y axis using the keyboard, mouse, joystick or touch controls. Same as on panning, if you don't want to allow the user to rotate around the target, you can just disable the keyboard, mouse, joystick and touch controls.

ROTATE WITH KEYBOARD

Here you need to setup the axis name for the keyboard rotation, and how fast you will rotate by using the keys. Bellow is how the rotation by keys is set up in the demo scene

ROTATE KEYBOARD ON Y AXIS

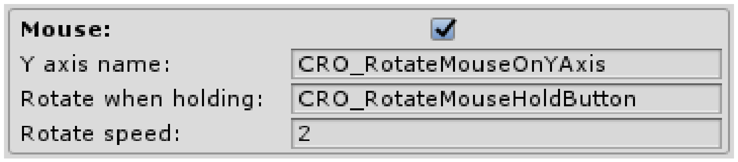


CRO_RotateKeyboardOnYAxis

Name	CRO_RotateKeyboardOnYAxis
Descriptive Name	Keyboard rotate Left
Descriptive Negative Name	Keyboard rotate Right
Negative Button	e
Positive Button	r
Alt Negative Button	
Alt Positive Button	
Gravity	3
Dead	0.1
Sensitivity	3
Snap	✓
Invert	
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

ROTATE WITH MOUSE

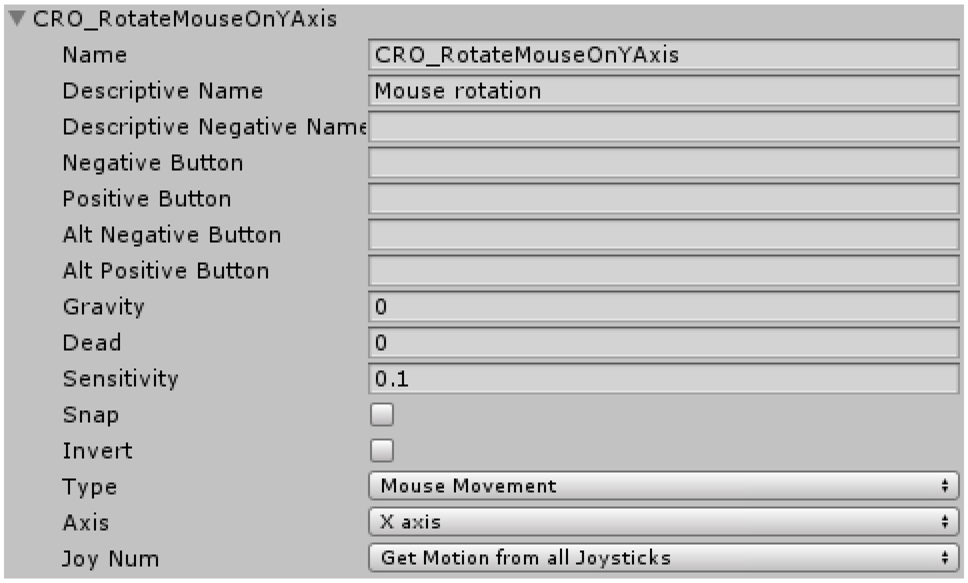
As in the case of panning, you need to define the keys used to rotate. By default, this is set up as right click and left alt button.



Mouse:	✓
Y axis name:	CRO_RotateMouseOnYAxis
Rotate when holding:	CRO_RotateMouseHoldButton
Rotate speed:	2

ROTATE MOUSE ON Y AXIS

In the demo scene, the rotation with the mouse is set up as following

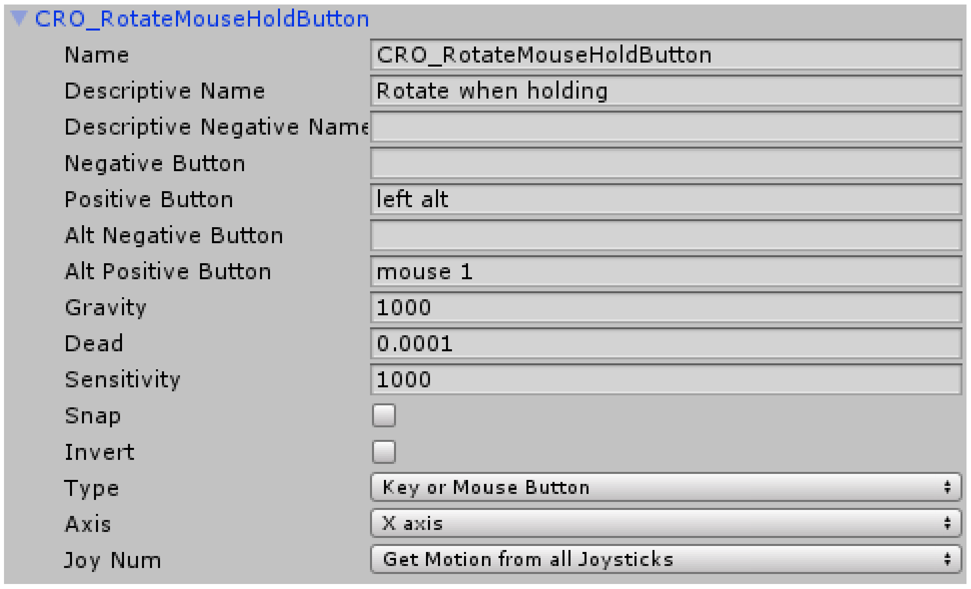


CRO_RotateMouseOnYAxis

Name	CRO_RotateMouseOnYAxis
Descriptive Name	Mouse rotation
Descriptive Negative Name	
Negative Button	
Positive Button	
Alt Negative Button	
Alt Positive Button	
Gravity	0
Dead	0
Sensitivity	0.1
Snap	☐
Invert	☐
Type	Mouse Movement
Axis	X axis
Joy Num	Get Motion from all Joysticks

ROTATE MOUSE HOLD BUTTON

Here are the default settings for mouse rotation. If you click left alt and move the mouse you will start to rotate the camera. As an alternative you can click right mouse button and move the mouse to rotate the camera.



CRO_RotateMouseHoldButton

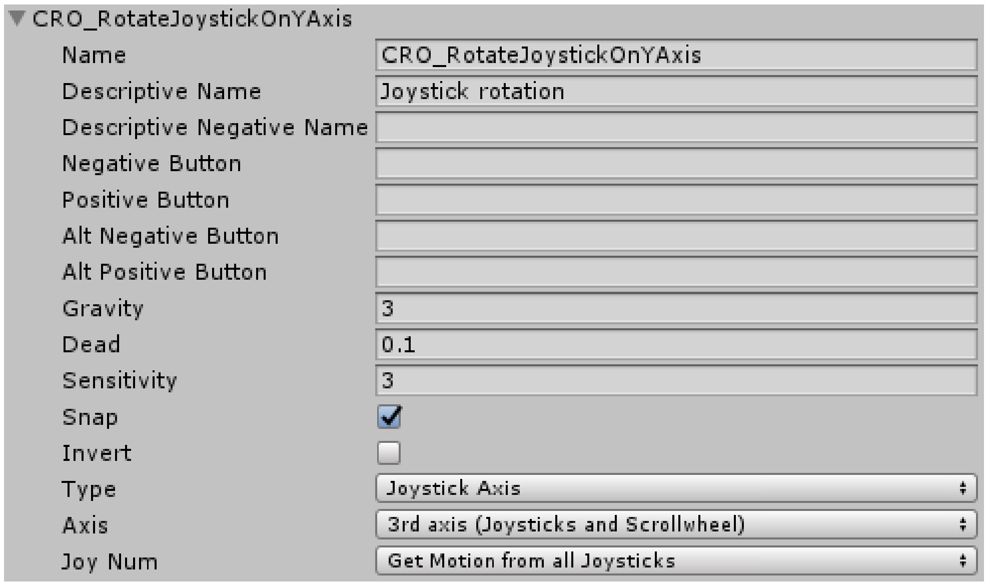
Name	CRO_RotateMouseHoldButton
Descriptive Name	Rotate when holding
Descriptive Negative Name	
Negative Button	
Positive Button	left alt
Alt Negative Button	
Alt Positive Button	mouse 1
Gravity	1000
Dead	0.0001
Sensitivity	1000
Snap	☐
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

ROTATE WITH JOYSTICK

In the demo scene, the rotation with the joystick is set up for using dual joystick, and rotation is done using the right stick. **Please note that the joystick controller was tested only on Windows.**

ROTATE JOYSTICK ON Y AXIS

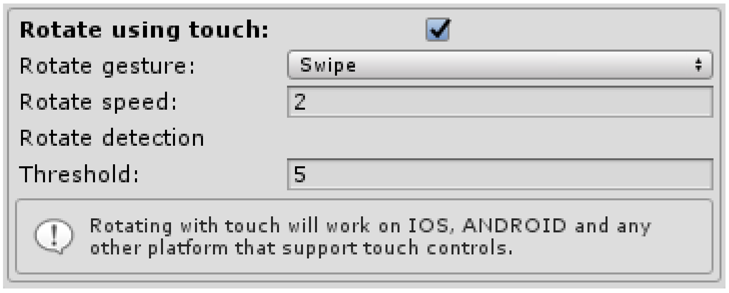
Here is the joystick setup in the demo scene



▼ CRO_RotateJoystickOnYAxis

Name	CRO_RotateJoystickOnYAxis
Descriptive Name	Joystick rotation
Descriptive Negative Name	
Negative Button	
Positive Button	
Alt Negative Button	
Alt Positive Button	
Gravity	3
Dead	0.1
Sensitivity	3
Snap	✓
Invert	☐
Type	Joystick Axis
Axis	3rd axis (Joysticks and Scrollwheel)
Joy Num	Get Motion from all Joysticks

ROTATE USING TOUCH



Rotate using touch: ✓
Rotate gesture: Swipe
Rotate speed: 2
Rotate detection Threshold: 5

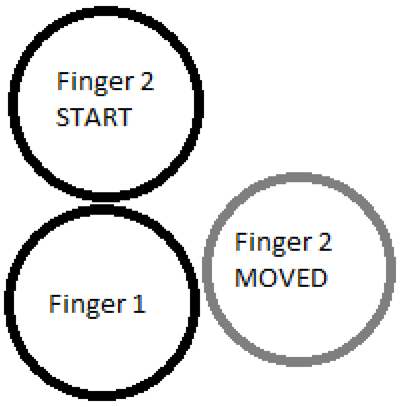
Rotating with touch will work on IOS, ANDROID and any other platform that support touch controls.

As in the case on panning, using touch controls you need to define the angle threshold that will indicate that you are trying to rotate.

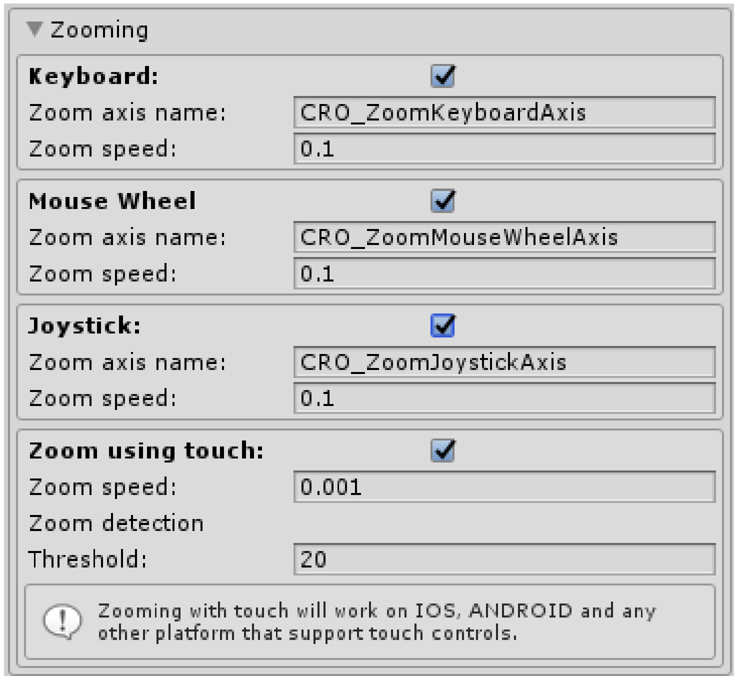
You have two option for rotation Swipe or Rotation gesture. Both of them are using two fingers.

For the **swipe gesture** place two fingers close to each other and move them left or right to rotate the camera.

For the **rotation gesture** the easiest method of rotation is to place two fingers close to each other and move one of them around the other.



ZOOMING



Zooming

Keyboard:	✓
Zoom axis name:	CRO_ZoomKeyboardAxis
Zoom speed:	0.1
Mouse Wheel	✓
Zoom axis name:	CRO_ZoomMouseWheelAxis
Zoom speed:	0.1
Joystick:	✓
Zoom axis name:	CRO_ZoomJoystickAxis
Zoom speed:	0.1
Zoom using touch:	✓
Zoom speed:	0.001
Zoom detection Threshold:	20

Zooming with touch will work on IOS, ANDROID and any other platform that support touch controls.

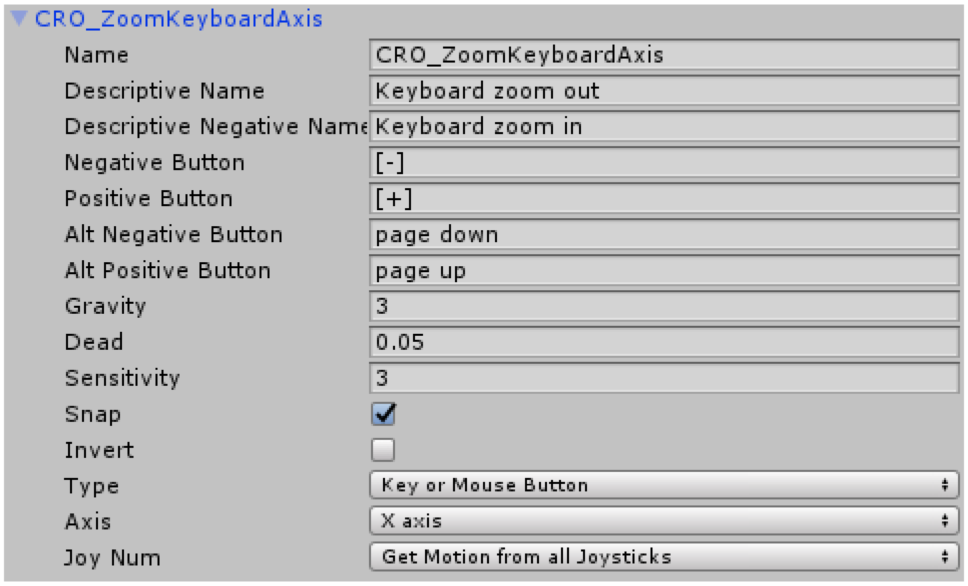
As already stated above, you can set up the controls for zooming by keyboard, mouse wheel, joystick and touch controls. The previous values are adjusted for the demo scene, so if you have different scaling or wider range for zooming you will need to adjust the speed for zooming.

ZOOMING WITH KEYBOARD

Again, you will need to specify the axis name for zooming and the speed.

Below we can see how the axis are defined in the Input tab for the demo scene.

ZOOM KEYBOARD AXIS



CRO_ZoomKeyboardAxis

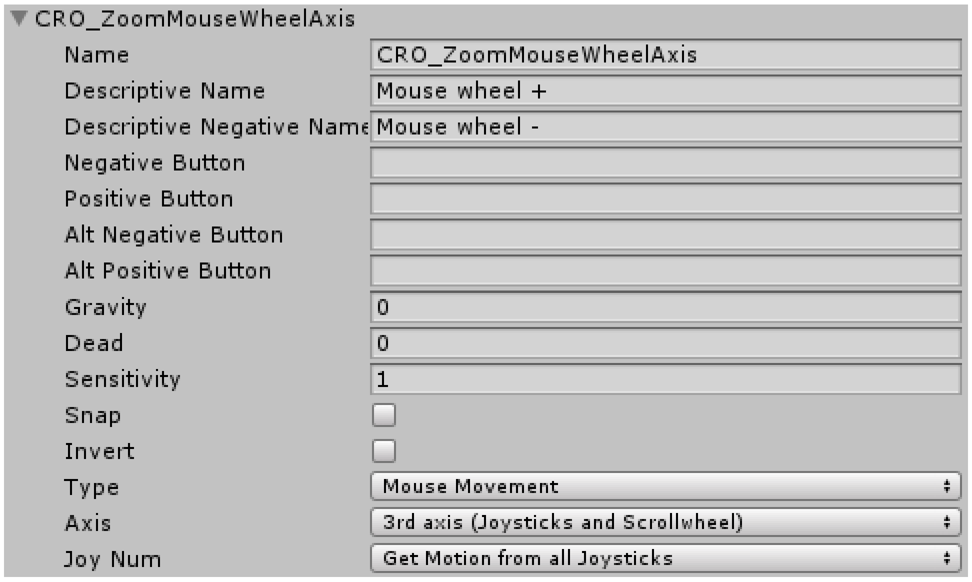
Name	CRO_ZoomKeyboardAxis
Descriptive Name	Keyboard zoom out
Descriptive Negative Name	Keyboard zoom in
Negative Button	[-]
Positive Button	[+]
Alt Negative Button	page down
Alt Positive Button	page up
Gravity	3
Dead	0.05
Sensitivity	3
Snap	✓
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

To zoom with keyboard, we are using the keypad + and -, or page up and page down.

ZOOM WITH MOUSE WHEEL

For this you just need to specify the speed of zooming.

ZOOM MOUSE WHEEL AXIS



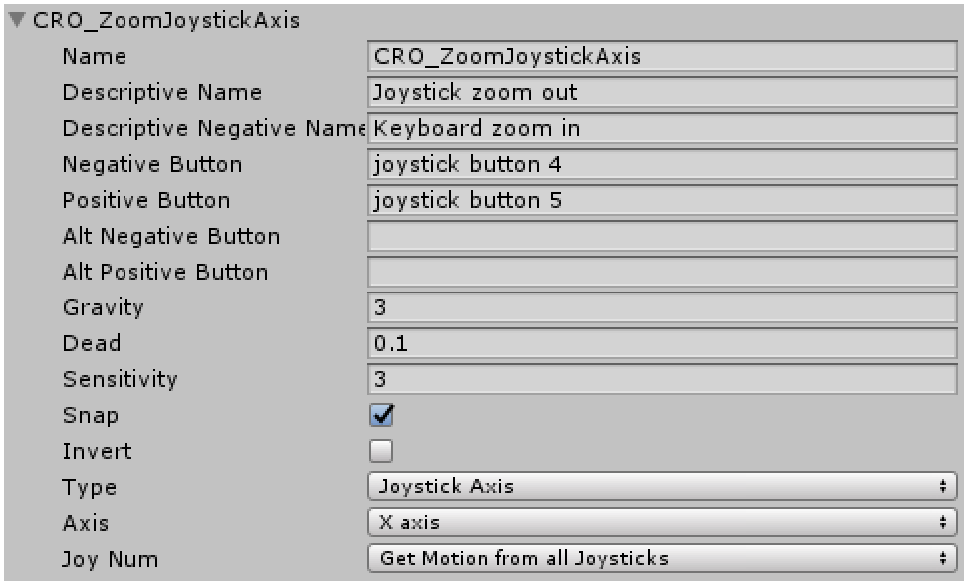
CRO_ZoomMouseWheelAxis

Name	CRO_ZoomMouseWheelAxis
Descriptive Name	Mouse wheel +
Descriptive Negative Name	Mouse wheel -
Negative Button	
Positive Button	
Alt Negative Button	
Alt Positive Button	
Gravity	0
Dead	0
Sensitivity	1
Snap	☐
Invert	☐
Type	Mouse Movement
Axis	3rd axis (Joysticks and Scrollwheel)
Joy Num	Get Motion from all Joysticks

ZOOM WITH JOYSTICK

Same as for the keyboard, you need to specify the axis name and the speed. Below we see how it is defined in the demo scene.

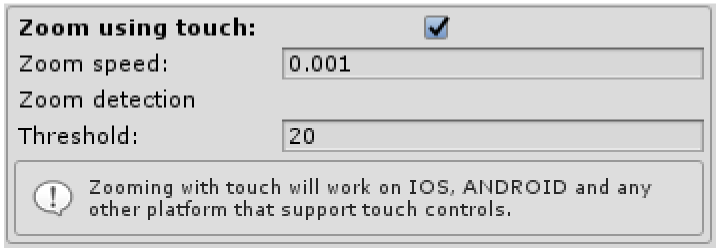
ZOOM JOYSTICK AXIS



CRO_ZoomJoystickAxis	
Name	CRO_ZoomJoystickAxis
Descriptive Name	Joystick zoom out
Descriptive Negative Name	Keyboard zoom in
Negative Button	joystick button 4
Positive Button	joystick button 5
Alt Negative Button	
Alt Positive Button	
Gravity	3
Dead	0.1
Sensitivity	3
Snap	✓
Invert	☐
Type	Joystick Axis
Axis	X axis
Joy Num	Get Motion from all Joysticks

In the demo scene the buttons for zooming are: L1, R1 (on a PS4 controller) or LB, RB (on an Xbox controller).

ZOOMING USING TOUCH



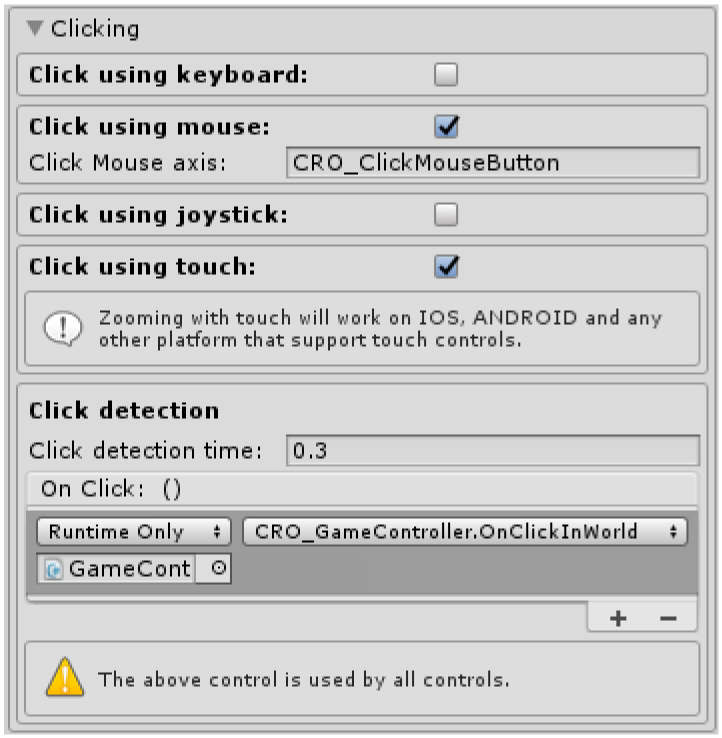
Zoom using touch:	✓
Zoom speed:	0.001
Zoom detection Threshold:	20

Zooming with touch will work on IOS, ANDROID and any other platform that support touch controls.

Zooming on mobile is done using two fingers. If you want to zoom in you need to move the fingers away from each other. If you want to zoom out you need to get your fingers close to each other.

You will need to setup the threshold for detecting that you are trying to zoom, and the speed of zooming.

CLICKING



Clicking

Click using keyboard:	☐
Click using mouse:	✓
Click Mouse axis:	CRO_ClickMouseButton
Click using joystick:	☐
Click using touch:	✓

Zooming with touch will work on IOS, ANDROID and any other platform that support touch controls.

Click detection

Click detection time: 0.3

On Click: ()

Runtime Only | CRO_GameController.OnClickInWorld

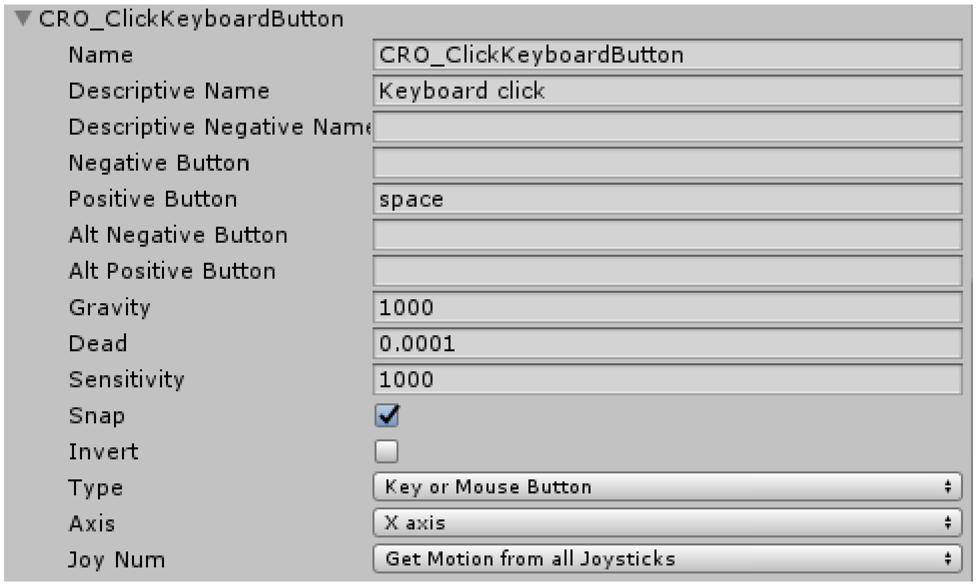
GameCont

The above control is used by all controls.

With clicking you can setup an event when a click is detected. You can customize what key, mouse button or joystick button to be used when detecting a click. This for example is useful to find out what object is under mouse if you click somewhere.

In the above image you have the default settings for the camera.

CLICK KEYBOARD BUTTON

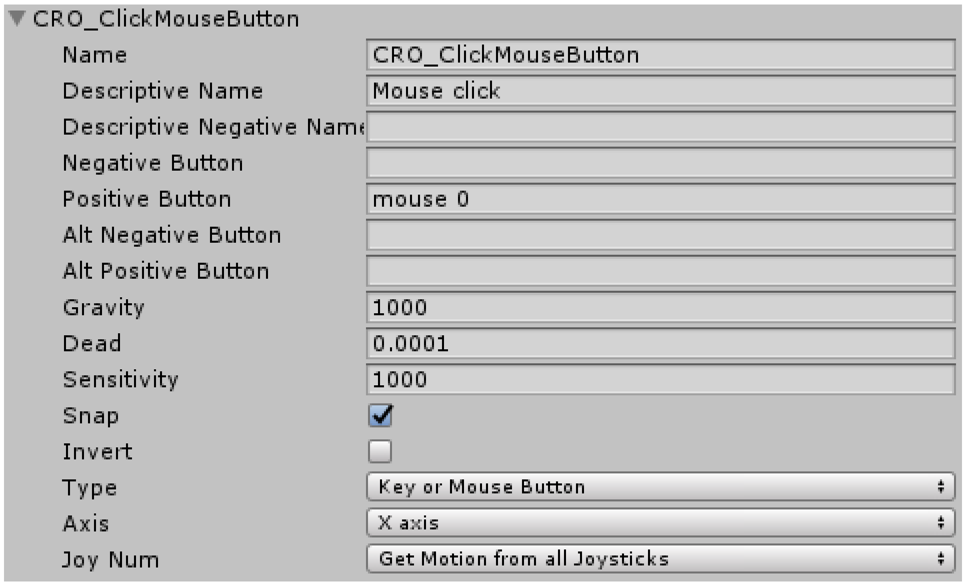


CRO_ClickKeyboardButton

Name	CRO_ClickKeyboardButton
Descriptive Name	Keyboard click
Descriptive Negative Name	
Negative Button	
Positive Button	space
Alt Negative Button	
Alt Positive Button	
Gravity	1000
Dead	0.0001
Sensitivity	1000
Snap	✓
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

In the demo scene the click keyboard button is ‘space’.

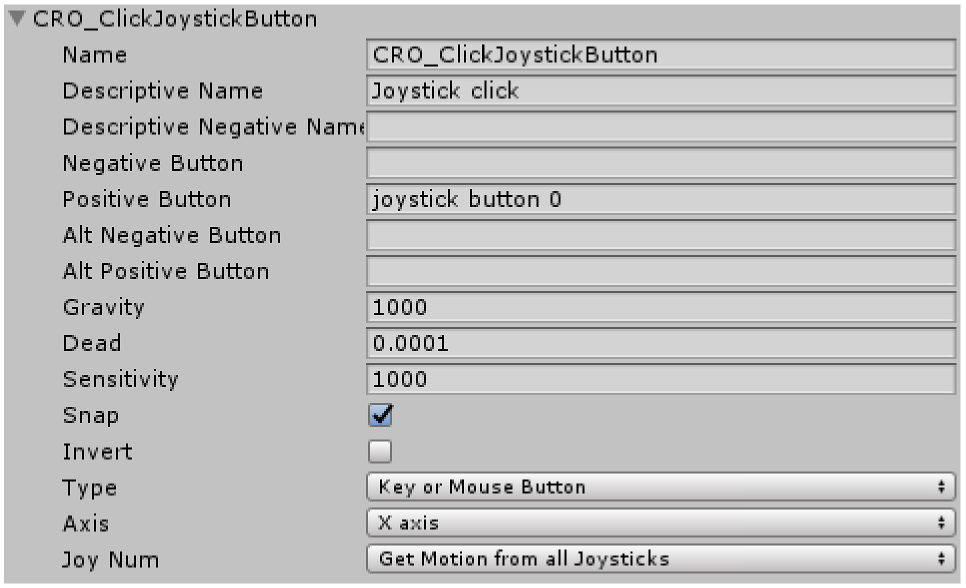
CLICK MOUSE BUTTON



▼ CRO_ClickMouseButton	
Name	CRO_ClickMouseButton
Descriptive Name	Mouse click
Descriptive Negative Name	
Negative Button	
Positive Button	mouse 0
Alt Negative Button	
Alt Positive Button	
Gravity	1000
Dead	0.0001
Sensitivity	1000
Snap	✓
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

By default, the click button in the scene is left mouse button.

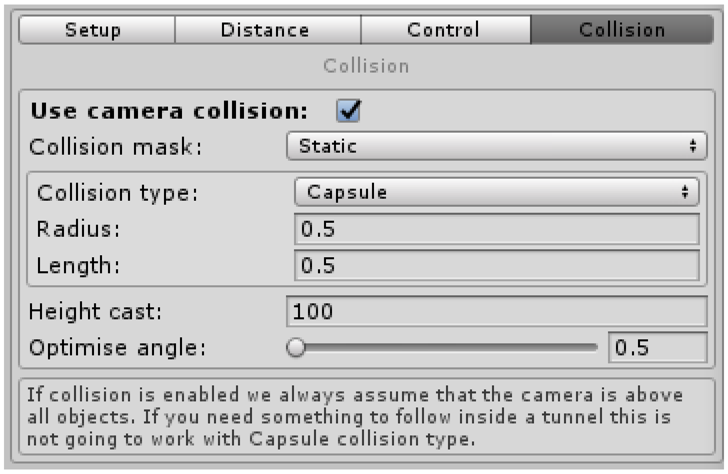
CLICK JOYSTICK BUTTON



▼ CRO_ClickJoystickButton	
Name	CRO_ClickJoystickButton
Descriptive Name	Joystick click
Descriptive Negative Name	
Negative Button	
Positive Button	joystick button 0
Alt Negative Button	
Alt Positive Button	
Gravity	1000
Dead	0.0001
Sensitivity	1000
Snap	✓
Invert	☐
Type	Key or Mouse Button
Axis	X axis
Joy Num	Get Motion from all Joysticks

In the demo scene the default joystick button for click is ‘joystick button 0’.

COLLISION



Use camera collision: ✓
Collision mask: Static
Collision type: Capsule
Radius: 0.5
Length: 0.5
Height cast: 100
Optimise angle: 0.5

If collision is enabled we always assume that the camera is above all objects. If you need something to follow inside a tunnel this is not going to work with Capsule collision type.

In this menu you can specify if you want the camera to react to the environment. Basically, to move up if there is no clear view of the target.

If enabled, you will need to setup a few parameters. First, there is the collision mask. What collision layers should we take into account when we try to detect the height of the environment. You can choose any combination of layers you want.

The height cast

This is defining from what height we should start to detect the collidable objects. **Be aware that if the height is too low and we start the cast from inside an object we will not be able to identify that object.** To be safe, you should set this value to be above all the objects in your world (Y axis).

Optimise angle

This is used for the case when the camera is moving up because of a collision with the environment, and defines how close the camera will get to the target. For every single unit that the camera is moving up, the angle set up is defining by how many units the camera will get closer to the target.

There are two types of collision types: **Capsule** and **Box**.

CAPSULE COLLISION

IF YOU ARE USING A PERSPECTIVE CAMERA THIS IS THE TYPE OF COLLISION THAT YOU NEED TO USE.

This is trying to create a capsule between the camera position and the target position, with the radius that you set up. The length is specifying how much the capsule will extend behind the camera. In this way we can try to detect if there is one object ready to enter the screen so we can start to move up the camera.

BOX COLLISION

IF YOU ARE USING ORTHOGRAPHIC CAMERA THIS IS THE TYPE OF COLLISION THAT YOU NEED TO USE.

The box collision was designed to be used with orthographic camera. For this method, you need to set up only the height of the Box that is used to detect the highest point of intersection of the environment with it. The box height should be small as there is no real influence on the result.

HOW TO USE THE CAMERA FROM CODE

You have access to all the parameters that you set up in the editor, by directly modifying them. The most useful parameters are “follow target” or “free camera”.

If you changed the settings at runtime you might need to call `m_CRO_Camera.SettingsChanged();` to readjust the camera position, orientation and zoom levels based on the new settings.

When switching from one target to another, the camera will smoothly go to the new target. If you want to just snap in one frame to the desired position, just call `m_CRO_Camera.SnapToDefault();`

FUNCTIONS AND HELPERS

You can access and set all the camera parameters also from code, not only from the editor. This allows you to create specific changes as reaction to gameplay.

In editor you have settings for different parts lets go throw all of them.

SETUP

For the setup you have getters and setters.

GETTERS

```
Camera GetCamera()
Transform GetTarget()
Vector3 GetTargetedPosition()
Vector2 GetMapLimitLow()
Vector2 GetMapLimitHigh()
float GetSmoothMoveSpeed()
float GetSmoothRotationSpeed()
float GetSmoothZoomSpeed()
float GetSmoothAdjDistance()
bool IsFollowingTarget()
```

SETTERS

```
void SetCamera(Camera camera)
void SetTarget(Transform target)
void SetFreeCam(bool keepCurrentPosition)
void SetTargetedPosition(Vector3 position)
void SetMapLimitLow(Vector2 limitLow)
void SetMapLimitHigh(Vector2 limitHigh)
void SetSmoothMoveSpeed(float value)
void SetSmoothRotationSpeed(float value)
void SetSmoothZoomSpeed(float value)
void SetSmoothAdjDistance(float value)
```

DISTANCE

Same system you have setters and getters

GETTERS

```
Vector2 GetOrbitRadius()
Vector2 GetOrbitHeight()
Vector2 GetOrthographicSize()
float GetDefaultOrientation()
float GetDefaultZoomLevel()
```

SETTERS

```
void SetOrbitRadius(Vector2 radius)
void SetOrbitHeight(Vector2 height)
void SetOrthographicSize(Vector2 size)
void SetDefaultOrientation(float orientation)
void SetDefaultZoomLevel(float zoom)
```

CONTROL

Control is a little different as all the controls are outside the camera in its own controller. You have similar options for getters and setters. We will look at the options for CameraStandardInput a little later.

PUBLIC ENUMS

```
enum TouchRotateGesture
{
    Swipe,
    Rotate
};
```

There are two types of Rotate Gestures supported, swipe and rotate. You can use this define to pass around the settings or checking for current state.

GETTERS

```
CRO_CameraStandardInput GetStandardInput()
```

SETTERS

```
void SetStandardInput(CRO_CameraStandardInput standardInput)
```

COLLISION

PUBLIC ENUMS

```
enum CollisionType
{
    Capsule,
    Box
};
```

There are two types of collision checks supported. Capsule that is recommended for perspective cameras, or Box that is recommended for orthographic cameras.

GETTERS

```
        bool GetUseCameraCollision()
        LayerMask GetCollisionMask()
        float GetCapsuleRadius()
        float GetCapsuleLength()
        float GetHeightCast()
        float GetPerUnitInHeightCloser()
        float GetBoxHeight()
        CollisionType GetCollisionTypeCheck()
```

SETTERS

```
        void SetUseCameraCollision(bool useCameraCollision)
        void SetCollisionMask(LayerMask collisionMask)
        void SetCapsuleRadius(float capsuleRadius)
        void SetCapsuleLength(float capsuleLength)
        void SetHeightCast(float heightCast)
        void SetPerUnitInHeightCloser(float perUnitInHeightCloser)
        void SetBoxHeight(float boxHeight)
        void SetCollisionTypeCheck(CollisionType collisionTypeCheck)
```

HELPERS METHODS

```
        //Get current camera orientation
        Quaternion GetCameraOrientation()
        //Get current camera forward
        Vector3 GetCameraForward()
        //Get current pivot position
        Vector3 GetPivotPosition()

        //Get the targeted zoom level. This is where the camera level should get. Is not
necessary where the camera zoom level is in this moment
        float GetTargetedZoomLevel()
        //Get the targeted rotation angle. Same as for the zoom, is where the camera
should rotate now where is the camera now
        float GetTargetedRotationAngle()

        //Set where the camera should get in terms of zoom. It will smoothly interpolate
towards that zoom level
        void SetTargetedZoomLevel(float zoom)
        //Set where the camera should rotate. It will smoothly interpolate towards that
rotation angle
        void SetTargetedRotationAngle(float angle)
        //Set where the pivot of the camera should be.
        void SetPivotPosition(Vector3 position)

        //Computes an offset taking into account the angle of rotation around Y on a
circle with a specific radius at specific height.
        Vector3 GetOffset(float angle, float radius, float height)

        //Using the current camera and a specific position, we re construct a plane at
the specified height (iY). Using the mouse position we will return where we intersect the
constructed plane. This helps not to have collision on all objects and still be able to
identify where we clicked.
        Vector3 MouseToWorld(Vector3 iCameraPosition, float iY, Vector3 iMousePosition)
        //Get the object under the mouse. It will return true in case there is valid
information inside hitInfo.
        bool GetObjectUnderMouse(out RaycastHit hitInfo)

        //In case you changed setting at runtime, you might need to call this function to
readjust the camera position and zoom levels based on the new settings
        void SettingsChanged()

        //If you want to disable camera controls just call DisableInputs(true). If you
need to re-enable camera controls call DisableInputs(false)
        void EnableInputs(bool enableInputs)

        //If you want the camera to jump to new settings and not smoothly interpolate
between current position/orientation/zoom call this function
        void SnapToDefault()

        //This is a helper function used to compute the map limits. It is used by the
editor to auto detect world bounds
        void ComputeMapLimits(LayerMask layerMask)
```

FOLLOW TARGET

Below is a simple example on how to make the camera follow a target by code. Just call the function FollowTarget with the target transform and the camera will start following that object.
In the example bellow you can also set the target in Unity Editor and when you press “play” it will automatically start to follow the target.

```
public class CRO_GameController : MonoBehaviour
{
    public Transform m_Target;
    public CRO_Camera m_CRO_Camera;

    public void FollowTarget(Transform target)
    {
        m_CRO_Camera.SetTarget(target);
    }

    void Start()
    {
        FollowTarget (m_Target);
        //The next line will snap the camera to the target object
        m_CRO_Camera.SnapToDefault();
    }
}
```

FREE CAMERA

To follow up on the example above, we will now show how to allow the user to switch to free camera.

```
public class CRO_GameController : MonoBehaviour
{
    public Vector3 m_TargetPosition;
    public CRO_Camera m_CRO_Camera;

    public void FreeCamera(Vector3 position)
    {
        m_CRO_Camera.m_TargetPosition = position;
        m_CRO_Camera.m_Target = null;
        m_CRO_Camera.m_FollowTarget = false;
    }

    void Start()
    {
        FreeCamera(TargetPosition);
        //The next line will snap the camera to the target position
        m_CRO_Camera.SnapToDefault();
    }
}
```

As you can see from the above example, you can change the camera usage easily. As an addon, we will demonstrate below a way to transition from the FollowTarget to FreeCamera by keeping the current position.

```
public class CRO_GameController : MonoBehaviour
{
    public CRO_Camera m_CRO_Camera;

    public void FollowTarget(Transform target)
    {
        m_CRO_Camera.m_Target = target;
        m_CRO_Camera.m_FollowTarget = true;
    }

    public void FreeCamera(Vector3 position)
    {
        m_CRO_Camera.m_TargetPosition = position;
        m_CRO_Camera.m_Target = null;
        m_CRO_Camera.m_FollowTarget = false;
    }

    public void SwitchToFreeCamera()
    {
        FreeCamera(m_CRO_Camera.m_Target ? m_CRO_Camera.m_Target.position :
m_CRO_Camera.m_TargetPosition);
    }
}
```

DEMO SCENE

When you import the camera into your own project and want to test the camera, make sure that you do the first step: Adding **Camera to Scene**. Even though the camera was added, you don't have the control scheme needed by the camera added to the Input Manager. For this it is a lot easier to use the Manager provided **GameObject -> CodeRoadOne -> CRO Camera** to update the control scheme. Just press the **Create/Update** and the new controls for the camera will be added to the **Input Manager**.

CONTACT US

You can contact us to request features or if you need support with our camera via the following:

Email: CodeRoadOne@gmail.com
Discord: Join server
Twitter: Follow us
YouTube: Follow us