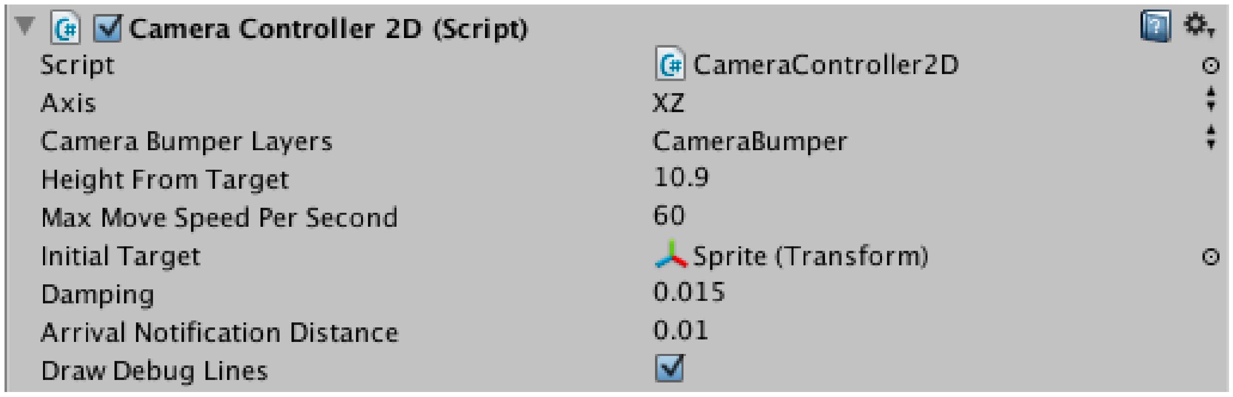
Camera For 2D Documentation

1. Overview

Camera For 2D is comprised of a primary CameraController2D script that you attach to one or more cameras within your scene, as well as a collection of scripts that interact with the CameraController2D.



Camera Controller 2D (Script)	
Script	CameraController2D
Axis	XZ
Camera Bumper Layers	CameraBumper
Height From Target	10.9
Max Move Speed Per Second	60
Initial Target	Sprite (Transform)
Damping	0.015
Arrival Notification Distance	0.01
Draw Debug Lines	✓

The CameraController2D compoenent

The provided demo scene named “All In One Examples” which is located in the Example/ Scenes directory shows off many ways of interacting with the CameraController2D component. The script used in this scene can be located in the Example/Scripts directory.

1.1 Resources

A video walkthrough of the example scene is available at: http://bit.ly/Wilbi5
The demo scene can be found at: http://bit.ly/XuELmg
The project page is located at: http://bit.ly/Xqo8Lz
The Unity form thread is located at: http://bit.ly/WZXyYC
You can get support by contacting David Koontz at david@happycamperstudios.com

2. CameraController2D Component

The CameraController2D component itself is fairly simple. The available parameters are:

- Axis - The axis to be used for horizontal and vertical movement. By default this is the XZ axis, with X being the horizontal and Z being the vertical. Other options are XY and YZ.
- Camera Bumper Layers - This defines the layers that should be considered blocking for the camera’s movement. You use objects on this layer to prevent the camera from smoothly scrolling into an area you do not want revealed. Common uses for this include hidden areas, the bounds of the world, and one way passages. If the target of the

camera moves through a camera bumper the camera will move through it as well. This is how a “one screen at a time” camera system works.

- Height From Target - The height along the “up” axis, (Y in the case of an XZ axis), that the camera should be positioned. This is primarily useful for perspective cameras.
- Max Move Speed Per Second - The maximum speed the camera should be able to obtain.
- Initial Target - Equivalent to calling the AddTarget method on the CameraController2D.
- Damping - A factor by which the camera’s acceleration should be slowed. More damping means a softer, “squishier” camera feel that lags behind the target and smoothly arrives at the target’s position.
- Arrival Notification Distance - The distance at which callback relating to arrival should be sent. This is different from the distance at which the camera stops moving and allows you to trigger a callback at a distance from your desired target.
- Push Box - A Rect defining an area of the screen the camera should not track the target inside. This is very useful for games that have allow the player to move back and forth (or jump) without the screen moving. When the target moves outside of the push box, the camera will move just enough to put the target back inside, it will not move completely to the target. If you do not wish to use this behavior simply leave the default width and height of 0.
- Camera To Use - If you need to place your camera on a separate game object from where the CameraController2D script is placed.
- Use Fixed Update - Executes all camera movement code inside FixedUpdate instead of LateUpdate. This is needed if you are tracking a physics controlled object.
- Draw Debug Lines - Turns on debug rendering in the editor. This does impose extra calculations on the CameraController2D and as such you may see performance degrade.

2.1 Camera seek behavior

The CameraController2D is constantly seeking towards a GameObject that it instantiates in Awake, named _CameraTarget. This GameObject, by default is positioned each frame at the current target’s position (or at the midpoint of the targets if there are multiple). It will not always be possible for the camera to move to this point, however, due to the presence of camera bumpers. Additionally, the camera target can be offset via the presence of influences, which are vectors set on the camera by objects in the scene.

2.2 Camera targets

2.2.1 Adding targets

If you need to switch targets from the initial target, you can accomplish this via the AddTarget method on the CameraController2D. There are three main variants.

public void AddTarget(Transform target)

The first simply takes a Transform as the new target. The camera will pan to the new target as necessary using the default “Max Move Speed Per Second” value.

public void AddTarget(Transform target, float moveSpeed)
This version takes a Transform for the new target, but then overrides the move speed of the camera while panning to that new target. Upon reaching the new target the original move speed will be restored.

public void AddTarget(Transform target, float moveSpeed,
float revertAfterDuration,
float revertMoveSpeed)
This is similar to the previous version, with a new target Transform, a move speed while panning to the new target but adds the ability to auto-revert to the previous target after a specified duration. The final parameter is for the move speed while returning to the original target. This is the version that is useful for a momentary reveal in some part of the scene without leaving the target of the camera changed.

In addition to these three methods, there are three more that take an IEnumerable<Transform> instead of single transform. You use these variations when you want to focus multiple objects at once.

2.2.2 Switching targets

Switching targets is similar to adding a target, except that the current target is removed first. There are only two variations of the SetTarget method.

public void SetTarget(Transform target)
Removes the current target, then sets a new target as specified by the Transform.

public void SetTarget(Transform target, float moveSpeed)
Removes the current target, then sets a new target as specified by the Transform but pans using the provided move speed.

As with AddTarget, you can also pass an IEnumerable<Transform> instead of a Transform if you wish to set a group of targets.

2.2.3 Removing targets

Removing a target is achieved via the RemoveCurrentTarget() method. It takes no parameters as it always just removes the current target or IEnumerable<Target> if multiple targets were set with AddTarget or SetTarget. This is called implicitly as part of SetTarget.

2.2.4 Callbacks

Each variant of AddTarget and SetTarget can optionally be given a callback to be run when arriving at the target for the first time. The provided method should be a System.Action, that is a method that takes zero parameters and has a return type of void. A simplified example of this is given below:

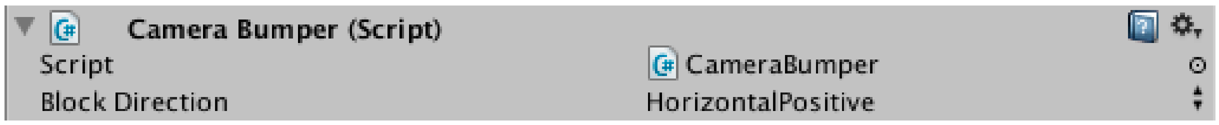
```
void Update() {
      if(Input.GetKeyDown(KeyCode.A)) {
            var controller = camera.GetComponent<CameraController2D>();
            controller.AddTarget(someTransform, MyCallback);
      }
}

void MyCallback() {
      Debug.Log(“Arrived at someTransform”);
}
```

This callback will only be run once, the first time the camera arrives at the new target. If the target is subsequently removed or is added again and the callback parameter is provided, it will be called again.

2.3 Camera Bumpers

Camera bumpers are nothing more than a collider set to one of the layers set on the CameraController2D’s “Camera Bumper Layers” property. By default, these colliders are considered blocking from any direction. If you wish to have finer grained control over this, you can attach the CameraBumper script to the object.


Camera Bumper (Script)
Script CameraBumper
Block Direction HorizontalPositive

This will allow you to set which directions the bumper should block the camera’s movement in. The directions that you can set are:

- None
- Horizontal (blocks both horizontal directions)
- Positive Horizontal
- Negative Horizontal
- Vertical (blocks both vertical directions)
- Positive Vertical
- Negative Vertical
- All Directions

In the case of the XZ axis, X would be the horizontal axis and positive X would correspond to Positive Horizontal, while positive Z would correspond to positive Vertical.

2.4 Influences

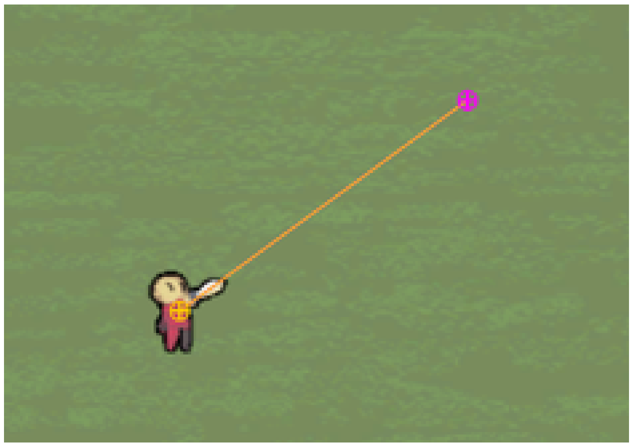
In many cases, you will need the camera to follow the target, but be displaced by some amount. Examples of this include a “look ahead” feature, a “point of interest” that draws the camera towards something in the scene, and even a simple offset such as for a boss character that is partially offscreen. All of these are accomplished via the same mechanism.

public void AddInfluence(Vector3 influence)

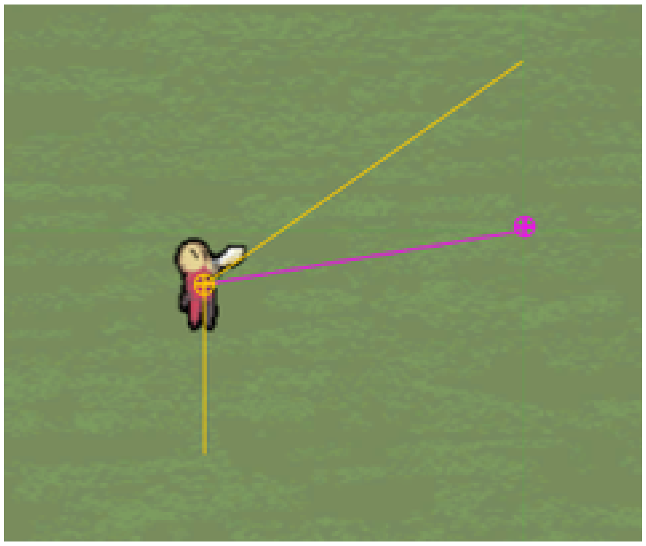
Simply call the AddInfluence method, passing in your desired offset for the camera. Any number of scripts can add an influence to the camera, they will be combined into one final offset for the camera.

NOTE: Influences are cleared in LateUpdate every frame. You will want to calculate and add your influence each frame.

If you enable Debug Lines you will be able to see the influences as pictured below.



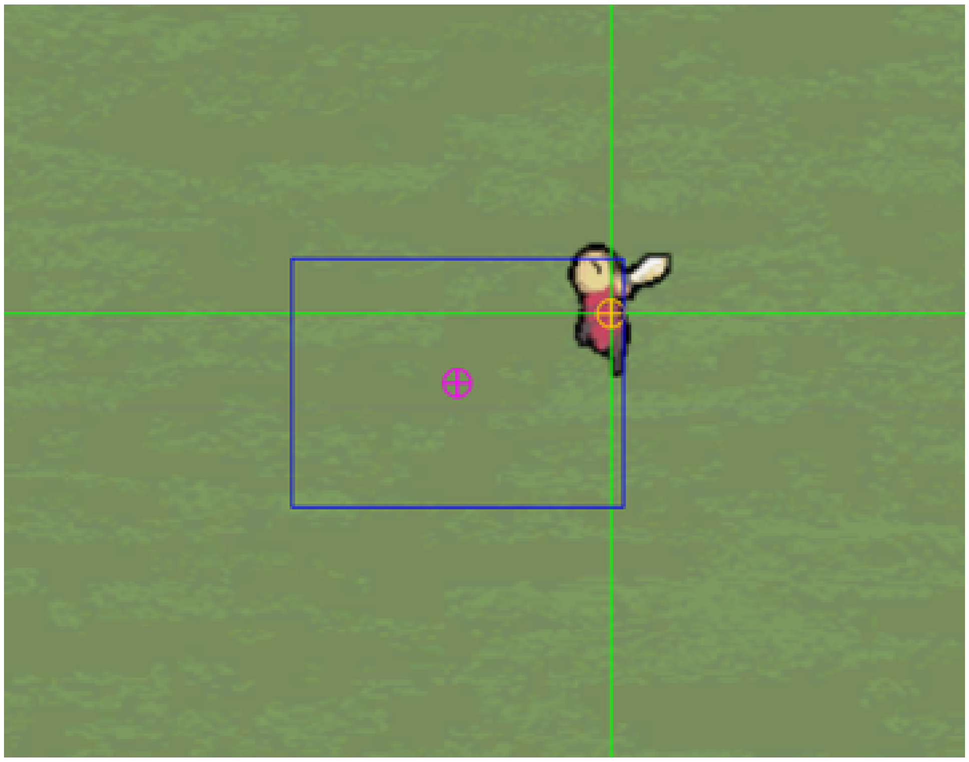
Single influence (orange line) moving the camera’s target point (magenta circle)



Multiple influences (orange lines) moving the camera's target point (magenta circle)

2.5 Push Box

In situations where you wish the target to be able to move around in a limited area on the screen without the camera traking its position, a Push Box can be defined. The default values are to center the box at the middle of the screen (0.5, 0.5) with a width and height of 0. In the image below, the Push Box is set up at (0.5, 0.5) with a width and height of 0.15. These values are all in viewport space, that is they go from 0 to 1 independently of aspect ratio or screen resolution. When the target moves beyond the Push Box, the camera will move the minimum amount needed to move the target back into the Push Box.


A Push Box at (0.5, 0.5) with a width and height of 0.15

2.6 Zoom

Although not visible as a property on the CameraController2D component in Unity, the DistanceMultiplier property can be used to “zoom” the camera. This works for both ortho and perspective cameras. The DistanceMultiplier starts at 1 and changing this will act as a multipler either the ortho size of the camera if in ortho mode, or the “Height From Target” value of the CameraController2D if in perspective mode. You can use values both greater and smaller than 1 as your multiplier. Smaller than one zooms in, while greater than 1 zooms out. So if you are in perspective mode, and have a height of 6 and set your DistanceMultiplier to 1.5 your camera would move to a height of 9.

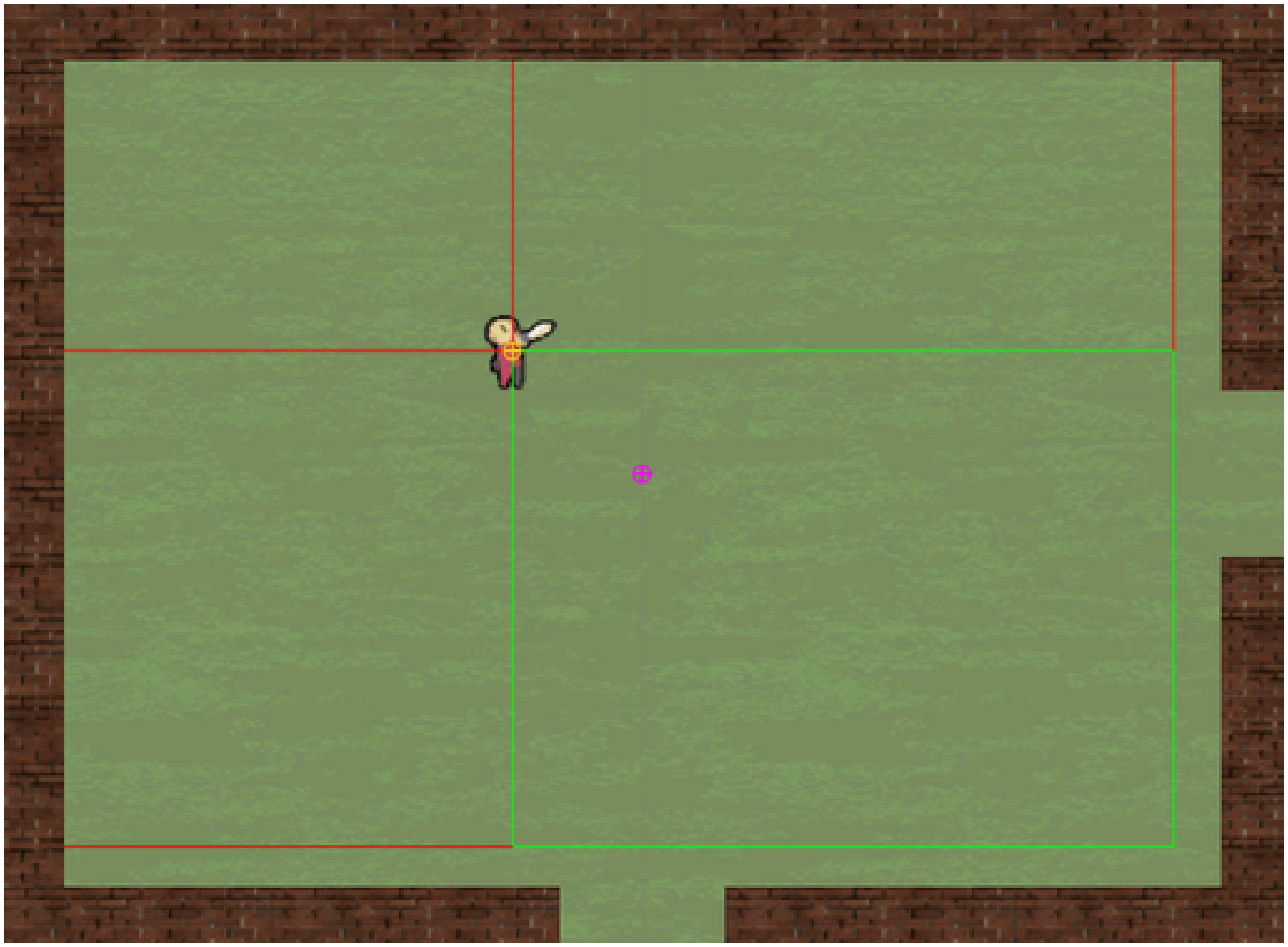
2.7 Debug Lines

Debug Lines are a useful way to see how some of the internal workings of the CameraController2D.

The primary elements that are drawn are:

- Magenta circle - Camera’s final seek position
- Orange circle - Camera’s desired position
- Red / green lines - Raycasts that hit or didn’t hit a camera bumper

- Orange/magenta lines - Individual and total camera influence



Here we see the target (orange circle) is above and to the left of the camera (magenta circle). The camera cannot move to the desired point as the camera's raycasts are colliding with camera bumpers at the wall to the left and above (red lines). On the right and below there are no camera bumpers so the raycasts show green.

As stated above, when Debug Lines are enabled, some extra calculations are done, so be aware that you may see some performance degredation.