2D Camera Boundaries and Transitions

By Stephan Zuidam

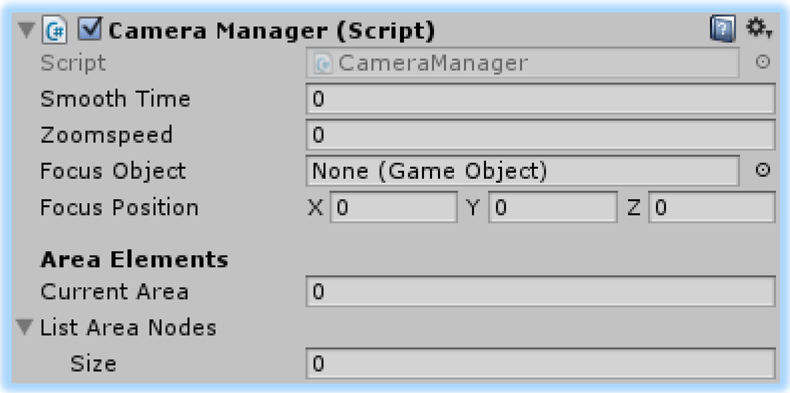
Introduction

While working on a simple 2D rogue lite I wanted a camera that followed the player (or any other object) around the room but within given boundaries. Whenever the object of interest would move outside of these boundaries the camera would transition to the newly entered area. To my surprise this proved more difficult than anticipated and any information about creating something like this hard to find. With a set goal I went ahead and figured a working camera with boundaries myself and, given the difficulty of finding similar projects, decided to work on the usability of the project for others to use.

This document will explain in a detailed manner how to set up this project with your own project and enjoy the ease of the camera boundaries.

Setting up the project

The only required assets to import to your own project are `CameraManager.cs` and the `AreaNodeStart` prefab. Once imported add the CameraManager component onto the Camera object in the scene. The added component will appear as seen in the image (img.1) in the inspector.



Camera Manager (Script)

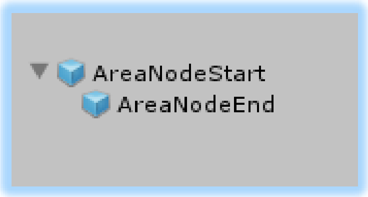
Script	CameraManager
Smooth Time	0
Zoomspeed	0
Focus Object	None (Game Object)
Focus Position	X 0 Y 0 Z 0
Area Elements	
Current Area	0
List Area Nodes	
Size	0

Img. 1 Camera Manager

The following variables can be found in the inspector:

- **Smooth Time**: The speed at which the camera follows its set target (working through the Vector.Lerp function). Can be adjusted in runtime to get the best fit value.
- **Zoomspeed**: Is the size at which the camera zooms in and out each frame when one of the zoom-functions is called. The size is taken from the camera’s *orthographic size*.
- **Focus Object**: Refers to the object the camera should follow. If there is a focus object, the focus position is set to 0, 0, 0.
- **Focus Position**: Is the position to which the camera moves if given. If there is a focus position, the focus object will be set to **null**.
- **Current Area**: Shows the current active area should multiple areas be defined.
- **List Area Nodes**: Hold all the AreaNodes which the user sets in the scene. The nodes have to be manually dragged to the inspector to add them to the list.

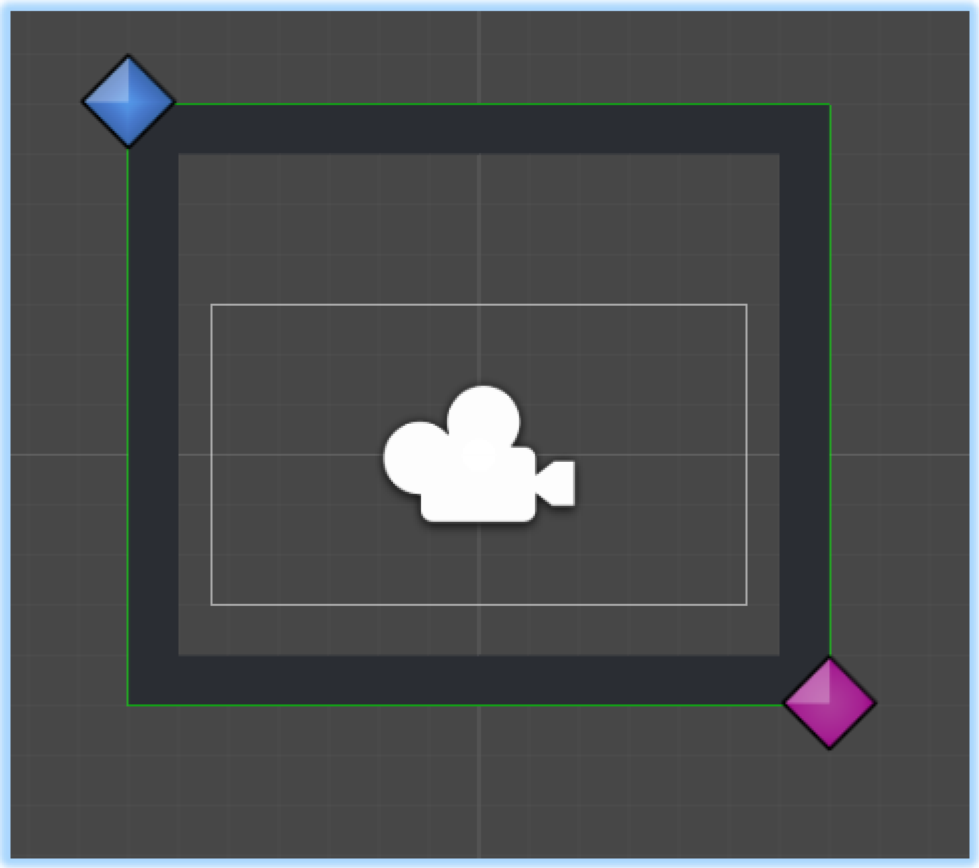
The **`AreaNodeStart`** prefab contains another gameobject named **`AreaNodeEnd`** (fig.2). The prefab has to be placed inside the scene after which the user can place the two gameobjects in to mark an area.



Img. 2 AreaNode prefab

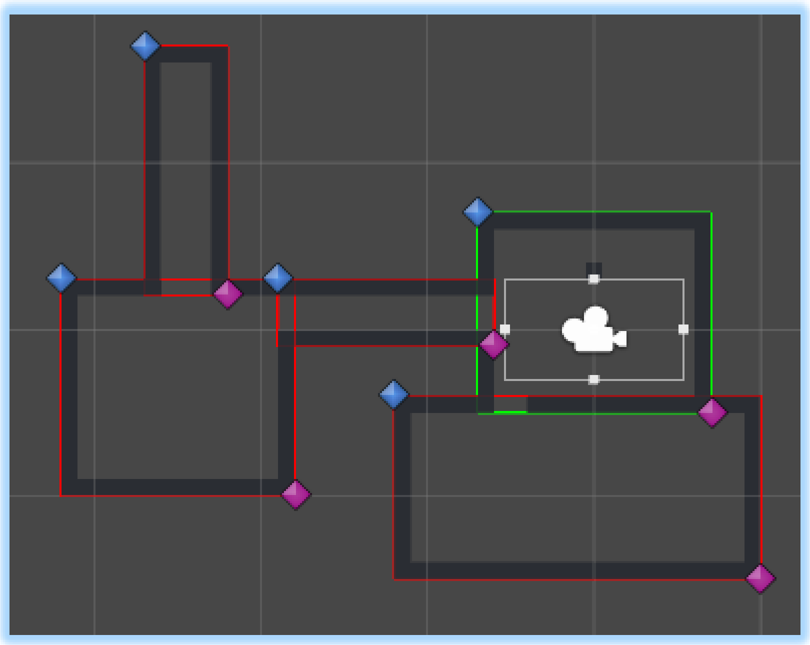
Once placed drag the gameobject to the inspector to add to the **`ListAreaNodes`** in the **`CameraManager`**. You'll notice right away a rectangle appearing between the set nodes, marking the boundaries. The first area you add will always appear with green boundaries (fig.3), meaning this is the starting area.

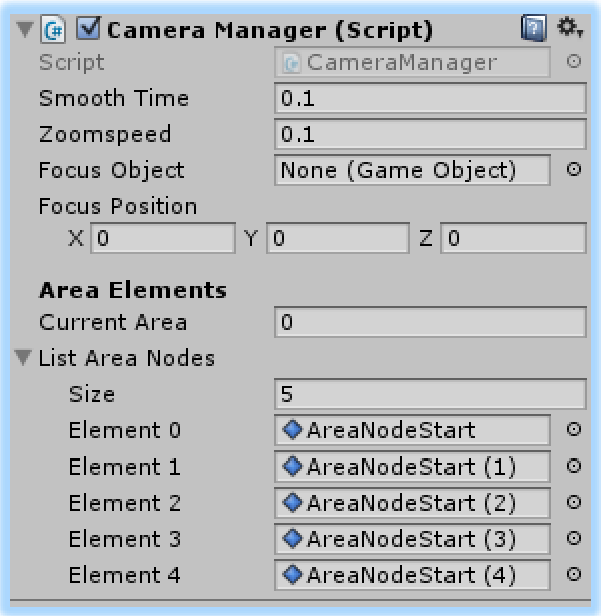
NOTE: you'll **only** have to add the **`AreaNodeStart`** to the list.



Img. 3 Marked starting area

With multiple areas marked your game scene might look something like below (fig.4).





Camera Manager (Script)

Script	CameraManager
Smooth Time	0.1
Zoomspeed	0.1
Focus Object	None (Game Object)
Focus Position	X 0 Y 0 Z 0

Area Elements

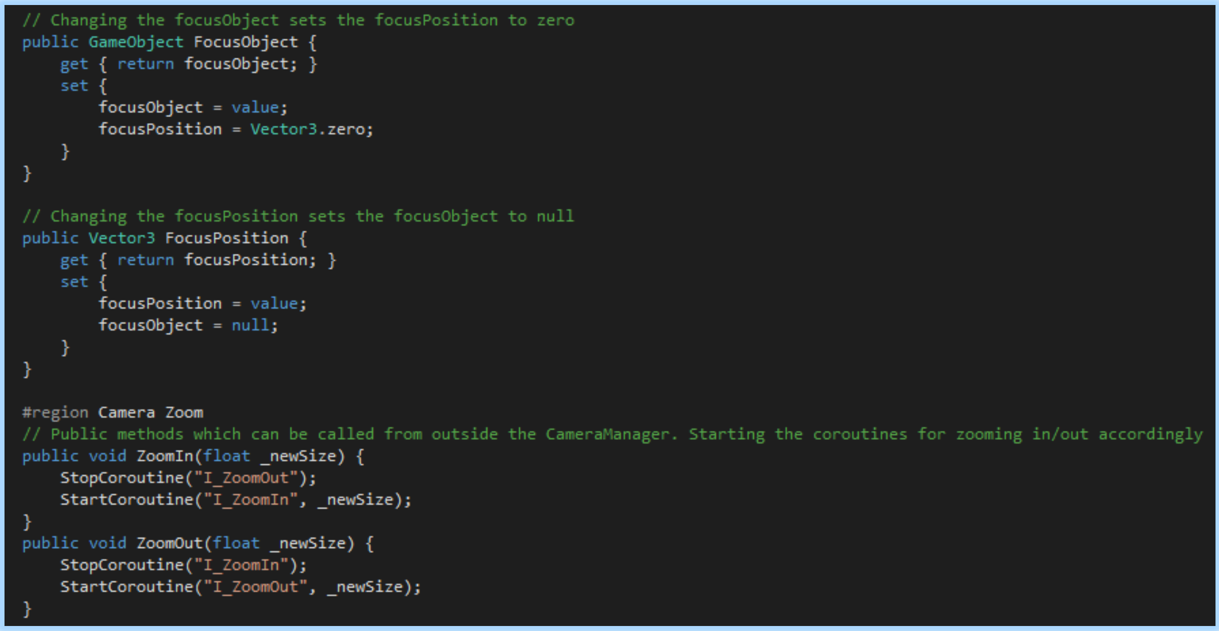
Current Area	0
List Area Nodes	
Size	5
Element 0	AreaNodeStart
Element 1	AreaNodeStart (1)
Element 2	AreaNodeStart (2)
Element 3	AreaNodeStart (3)
Element 4	AreaNodeStart (4)

Img. 4 Example gamescene with multiple areas

Utilizing the functions in your code

Several functions are made available in the CameraManager script (fig.5). Do take mind that to access these functions from other scripts you'll have to make a reference to the camera object first.

- **FocusObject(GameObject):** Accessor (Set/Get) to retrieve or change the current set Focus Object of the camera. Using the *set* of this accessor sets the Focus Position to **0, 0, 0**.
- **FocusPosition(Vector3):** Accessor (Set/Get) to retrieve or change the current set Focus Position of the camera. Using the *set* of this accessor sets the Focus Object to **null**.
- **ZoomIn(float):** Zooms in to the size of the value of the parameter. The size is taken from the camera's orthographic size.
- **ZoomOut(float):** Zooms out to the size of the value of the parameter. The size is taken from the camera's orthographic size. The **`CameraManager`** script has a variable named **`orthographicsize_base`** which takes the set size of the camera at the start, making it easier to always zoom in or out to the original size.

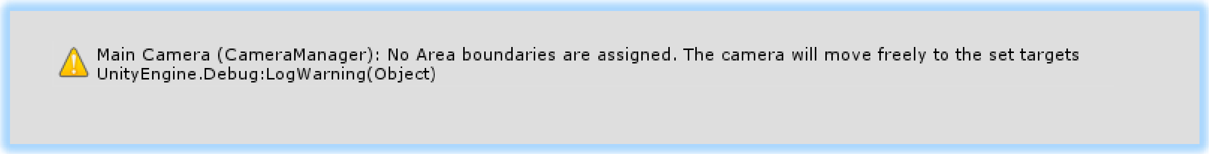


```
// Changing the focusObject sets the focusPosition to zero
public GameObject FocusObject {
    get { return focusObject; }
    set {
        focusObject = value;
        focusPosition = Vector3.zero;
    }
}

// Changing the focusPosition sets the focusObject to null
public Vector3 FocusPosition {
    get { return focusPosition; }
    set {
        focusPosition = value;
        focusObject = null;
    }
}

#region Camera Zoom
// Public methods which can be called from outside the CameraManager. Starting the coroutines for zooming in/out accordingly
public void ZoomIn(float _newSize) {
    StopCoroutine("I_ZoomOut");
    StartCoroutine("I_ZoomIn", _newSize);
}
public void ZoomOut(float _newSize) {
    StopCoroutine("I_ZoomIn");
    StartCoroutine("I_ZoomOut", _newSize);
}
```

Img. 5 Functions to influence the camera

A few things to note

When the **AreaNodeList** is empty and no boundaries have been assigned you may notice a 'warning' (fig.6) pop-up when running the project. This has no repercussions for the working except that the camera will move towards the set target freely.



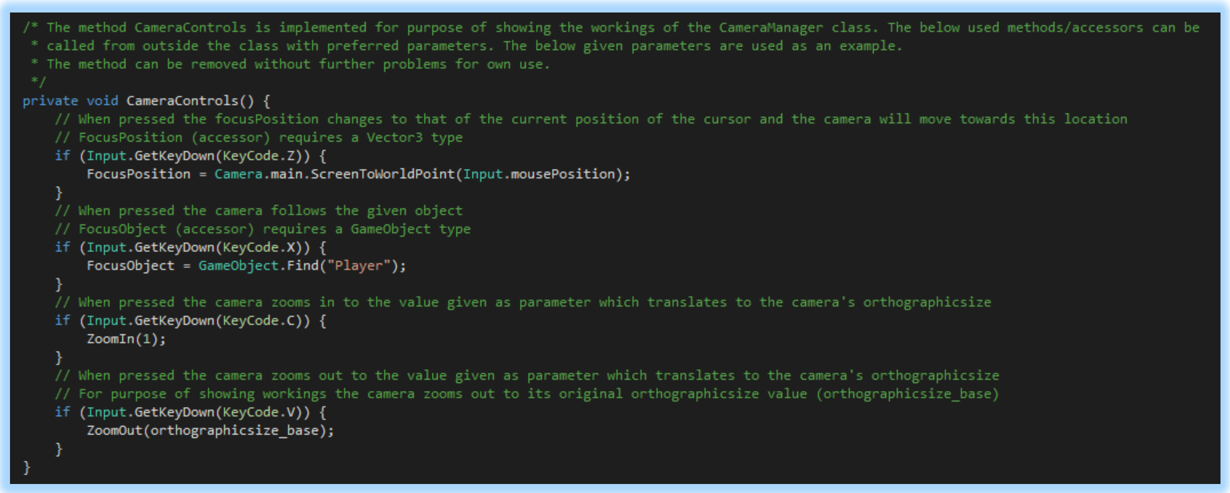
```
Main Camera (CameraManager): No Area boundaries are assigned. The camera will move freely to the set targets
UnityEngine.Debug:LogWarning(Object)
```

Img. 6 Warning when no boundaries are set

For purpose of showing the working of the project in the package's `Main_Scene` there have been added a few functions that trigger on key press.

- **Z:** Sets the Focus Position of the camera to the position of the cursor on the game screen and thus sets the Focus Object to **null**.
- **X:** Sets the Focus Object to that of the player in the example scene and thus disables the camera moving to a fixed point.
- **C:** Zooms the camera in to size '1'.
- **V:** Zooms the camera out to the starting size.

These functions have been added to show the effects in runtime but can be safely removed the code in your own project. The lines of code in matter are lines 185 through 205 (fig.7) as well as the function call to this method in line 76.



```
/* The method CameraControls is implemented for purpose of showing the workings of the CameraManager class. The below used methods/accessors can be
 * called from outside the class with preferred parameters. The below given parameters are used as an example.
 * The method can be removed without further problems for own use.
 */
private void CameraControls() {
    // When pressed the focusPosition changes to that of the current position of the cursor and the camera will move towards this location
    // FocusPosition (accessor) requires a Vector3 type
    if (Input.GetKeyDown(KeyCode.Z)) {
        FocusPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    }
    // When pressed the camera follows the given object
    // FocusObject (accessor) requires a GameObject type
    if (Input.GetKeyDown(KeyCode.X)) {
        FocusObject = GameObject.Find("Player");
    }
    // When pressed the camera zooms in to the value given as parameter which translates to the camera's orthographicsize
    if (Input.GetKeyDown(KeyCode.C)) {
        ZoomIn(1);
    }
    // When pressed the camera zooms out to the value given as parameter which translates to the camera's orthographicsize
    // For purpose of showing workings the camera zooms out to its original orthographicsize value (orthographicsize_base)
    if (Input.GetKeyDown(KeyCode.V)) {
        ZoomOut(orthographicsize_base);
    }
}
```

Img. 7 Functions for purpose of the example scene

And that is all!