Quest 8 - Steps

1) Import Assets

Import the zelda1.gif into your /Assets/Resources/Textures/ folder. Use the following settings:

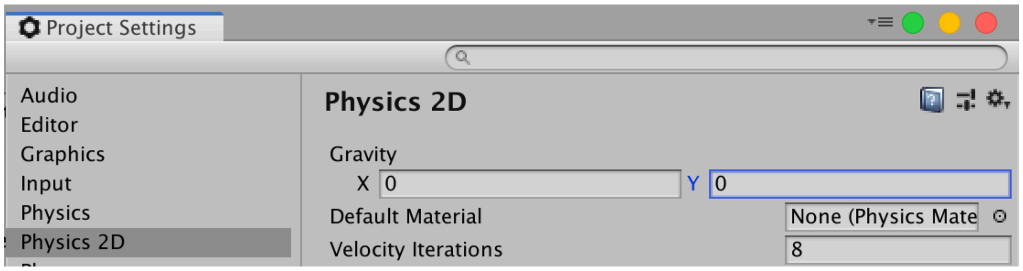
Sprite Mode = Multiple
PPU = 16
Filter Mode = Point (no filter)
Compression = None

Sprite Editor->Slice->Automatic

Apply all settings after changes and as prompted.

2) Turn off gravity

This will be a top-down perspective game that does not use gravity. Go to Edit->Project Settings->Physics 2D and set Gravity X and Y to 0.

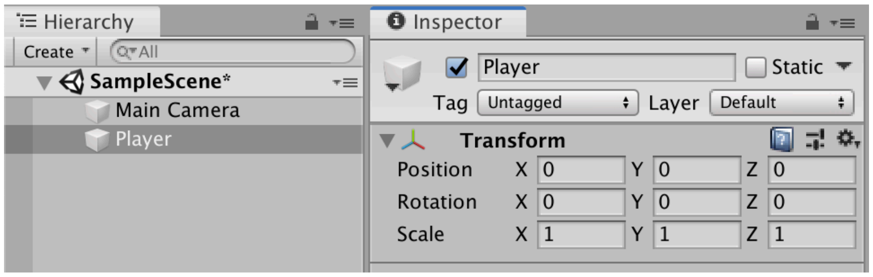


Physics 2D

Gravity
X 0 Y 0
Default Material None (Physics Mate
Velocity Iterations 8

3) Create a Player

Create a Player game object.

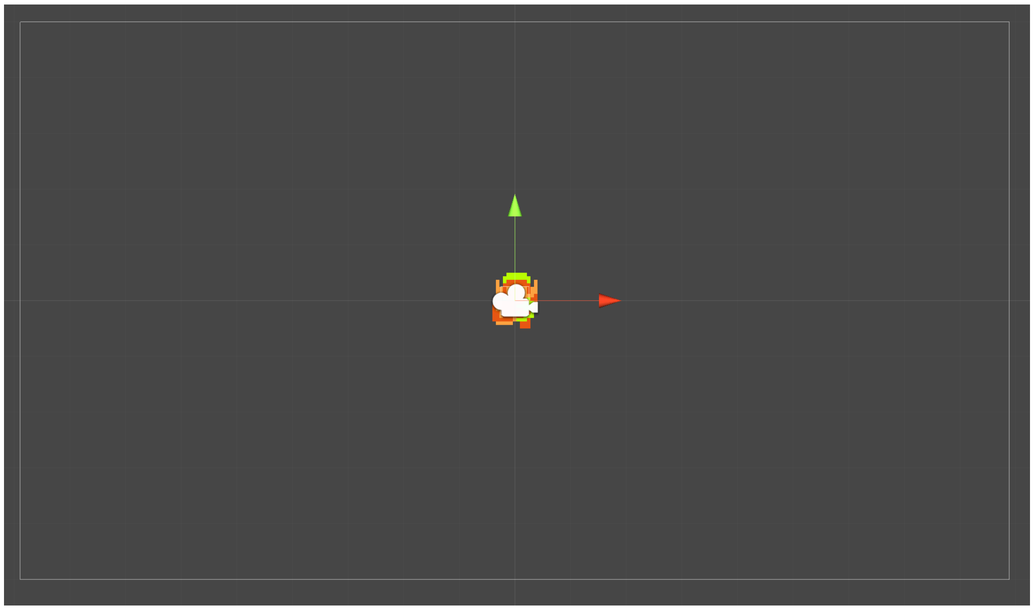


Player
Static
Tag Untagged Layer Default

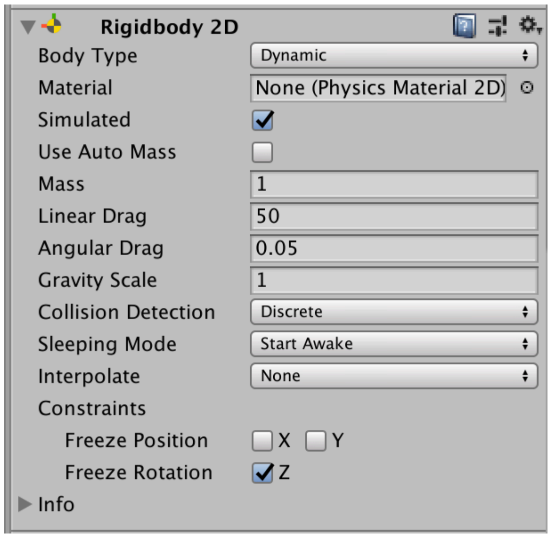
Transform

Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Attach a SpriteRenderer and assign the #4 graphic.



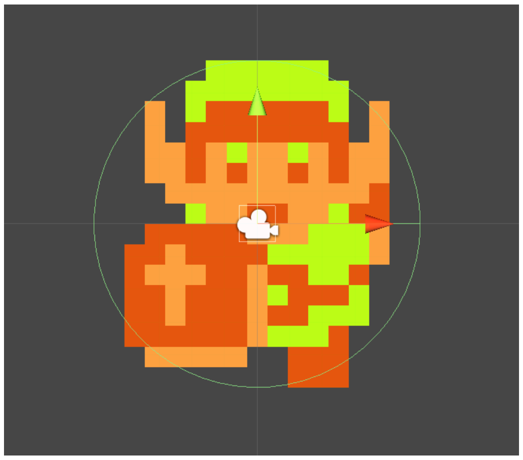
Add a Rigidbody2D component for physics and a CircleCollider2D component for collisions. Note the high Linear Drag and Freeze Rotation settings to keep our player from sliding away or spinning out of control.



Rigidbody 2D

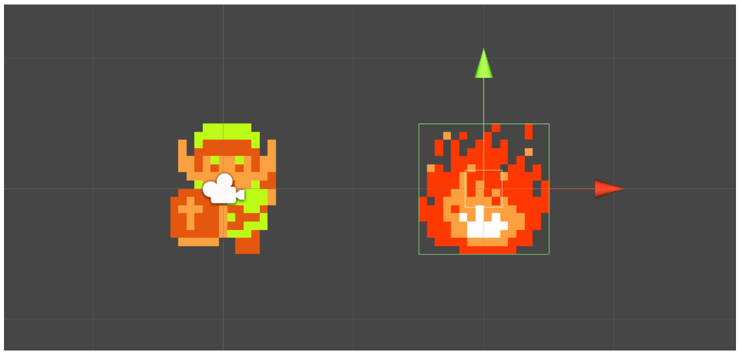
Body Type	Dynamic
Material	None (Physics Material 2D)
Simulated	✓
Use Auto Mass	☐
Mass	1
Linear Drag	50
Angular Drag	0.05
Gravity Scale	1
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	None
Constraints	
Freeze Position	☐ X ☐ Y
Freeze Rotation	✓ Z
Info	

We use a circle-shaped collider to help prevent the player from getting snagged on sharp boundaries.



4) Level Obstacle

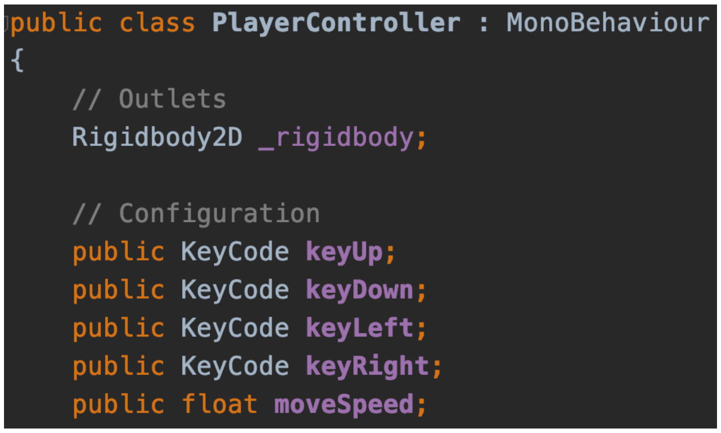
Create another object with a Sprite component. Use sprite number 48 which is the fire. Add a BoxCollider2D component. This obstacle will prevent passage by players.



5) Player Controller

Create a PlayerController.cs file in /Assets/Scripts/ and attach it to the Player game object.

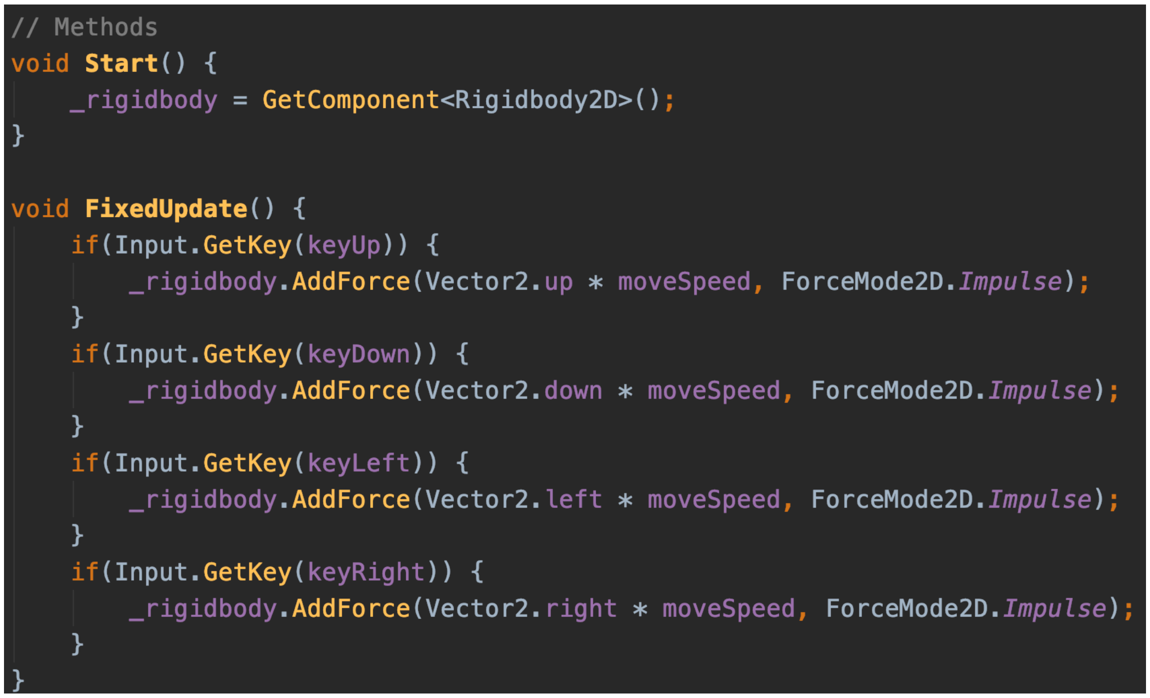
We need an Outlet to the Rigidbody2D so that this script can affect the object's physics. The configuration values let us set appropriate input keys and movement speed.



```
public class PlayerController : MonoBehaviour
{
    // Outlets
    Rigidbody2D _rigidbody;

    // Configuration
    public KeyCode keyUp;
    public KeyCode keyDown;
    public KeyCode keyLeft;
    public KeyCode keyRight;
    public float moveSpeed;
```

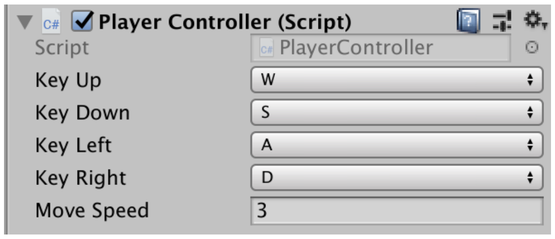
We fill the _rigidbody reference in the Start event and use the FixedUpdate loop to send physics forces. We use FixedUpdate instead of Update to help make movement more consistent if the game runs at a lower frame rate on some computers.



```
// Methods
void Start() {
    _rigidbody = GetComponent<Rigidbody2D>();
}

void FixedUpdate() {
    if(Input.GetKey(keyUp)) {
        _rigidbody.AddForce(Vector2.up * moveSpeed, ForceMode2D.Impulse);
    }
    if(Input.GetKey(keyDown)) {
        _rigidbody.AddForce(Vector2.down * moveSpeed, ForceMode2D.Impulse);
    }
    if(Input.GetKey(keyLeft)) {
        _rigidbody.AddForce(Vector2.left * moveSpeed, ForceMode2D.Impulse);
    }
    if(Input.GetKey(keyRight)) {
        _rigidbody.AddForce(Vector2.right * moveSpeed, ForceMode2D.Impulse);
    }
}
```

Fill in the public configuration values in the Unity inspector;

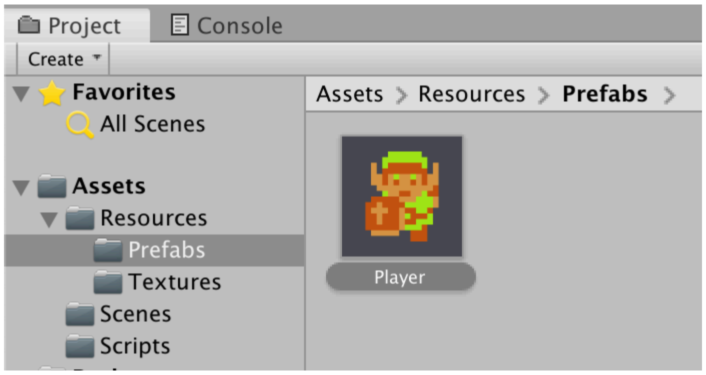


Player Controller (Script)

Script	PlayerController
Key Up	W
Key Down	S
Key Left	A
Key Right	D
Move Speed	3

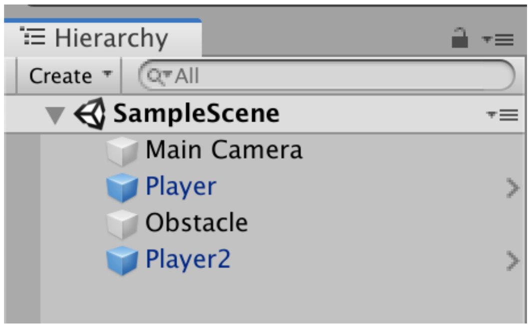
Playtest to ensure the character moves and cannot pass through the fire.

Prefab the Player game object in /Assets/Resources/Prefabs/

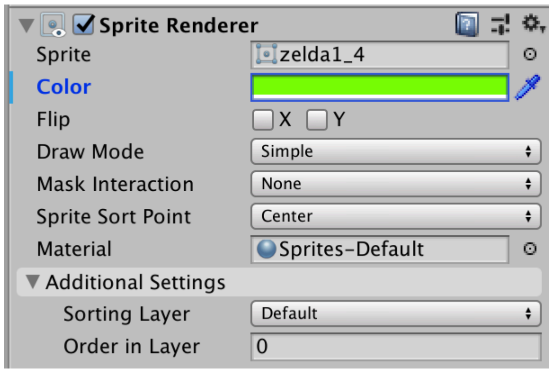


6) Player Two

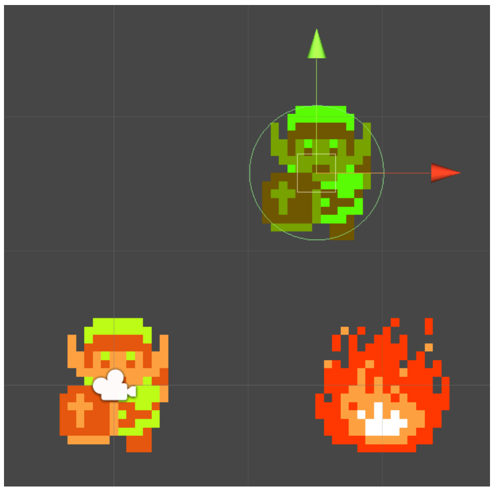
Duplicate the Player game object in the scene so that there are now Player and Player2 game objects. Both should be blue to signify that they originate from a prefab.



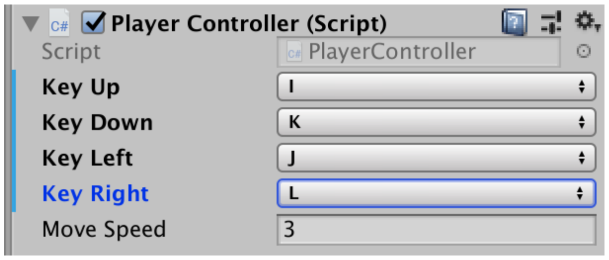
Tint Player2 slightly using the Color property on its SpriteRenderer, so that we can differentiate them.



Sprite Renderer	
Sprite	zelda1_4
Color	🟩
Flip	☐ X ☐ Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprites-Default
Additional Settings	
Sorting Layer	Default
Order in Layer	0



For Player2, use the IJKL keys for movement.



Player Controller (Script)

Script	PlayerController
Key Up	I
Key Down	K
Key Left	J
Key Right	L
Move Speed	3

Playtest to ensure both players can move independently using separate keyboard keys and that they cannot pass through the fire.

7) Follow Camera (Smoothdamp Approach)

There are many ways to setup a following camera using a mix of parenting, programmatic tracking, and cinematic packages. We will demo a few in class and implement a couple through this exercise.

Create a CameraController.cs file in /Assets/Scripts/ and attach it to your MainCamera game object.

```
public class CameraController : MonoBehaviour
{
    // Outlet
    public Transform target;

    // Configuration
    public Vector3 offset;
    public float smoothness;

    // State Tracking
    Vector3 _velocity;
```

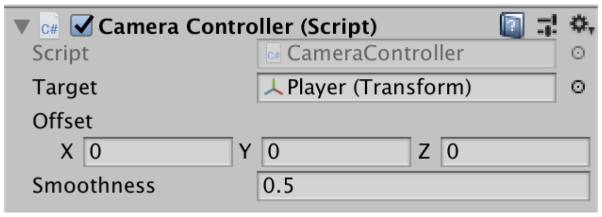
The target outlet specifies what object our camera will follow. The offset represents how far the camera will stay away from the target in all directions. (At a minimum you want the camera to stay away from the target in the z-axis so the camera has room to look at the target, rather than the camera following the target and being inside of it.) Smoothness specifies how long it should take the camera to catch up with the target (in seconds). The smoothing function we use needs a variable to output velocity into.

```
void Start() {
    if(target) {
        offset = transform.position - target.position;
    }
}
```

In the Start event, we compute the distance between the camera and the target (if there is one) and use this as the offset. This means the offset between the camera relative to the target in the scene is maintained even during gameplay.

```
void Update() {
    if(target) {
        transform.position = Vector3.SmoothDamp(transform.position, target.position + offset, ref _velocity, smoothness);
    }
}
```

In the Update loop, we run the camera's current position and target position (with offset) through a smoothing function. The smoothing function also takes a smoothness value and outputs velocity along with the final smoothed position for our camera as it follows the target.



Camera Controller (Script)

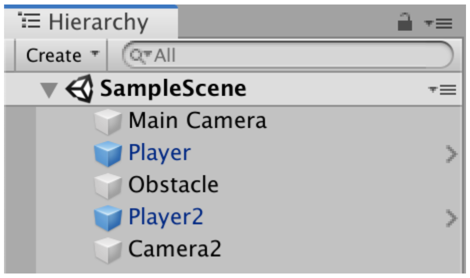
Script	CameraController
Target	Player (Transform)
Offset	X 0 Y 0 Z 0
Smoothness	0.5

With these settings the Camera will smoothly follow the player at a maximum of 0.5 seconds behind. Notice the offset is set to 0s until gameplay when it is automatically measured using the camera's starting position.

8) Split Screen

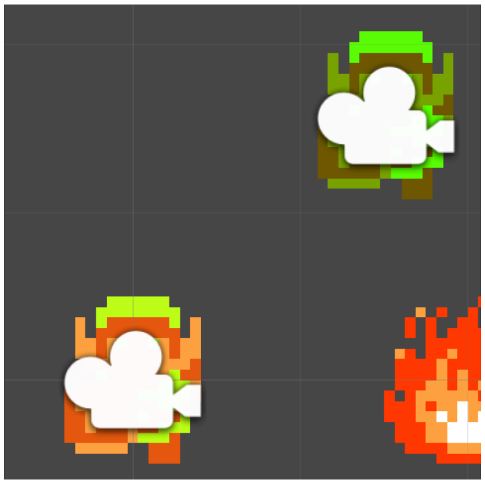
So far, all of our assignments have required only one Camera. It is possible to use multiple cameras. (Just remember to only have ONE AudioListener in the scene.)

Create a new Camera object. Name is Camera2 and delete its AudioListener component.

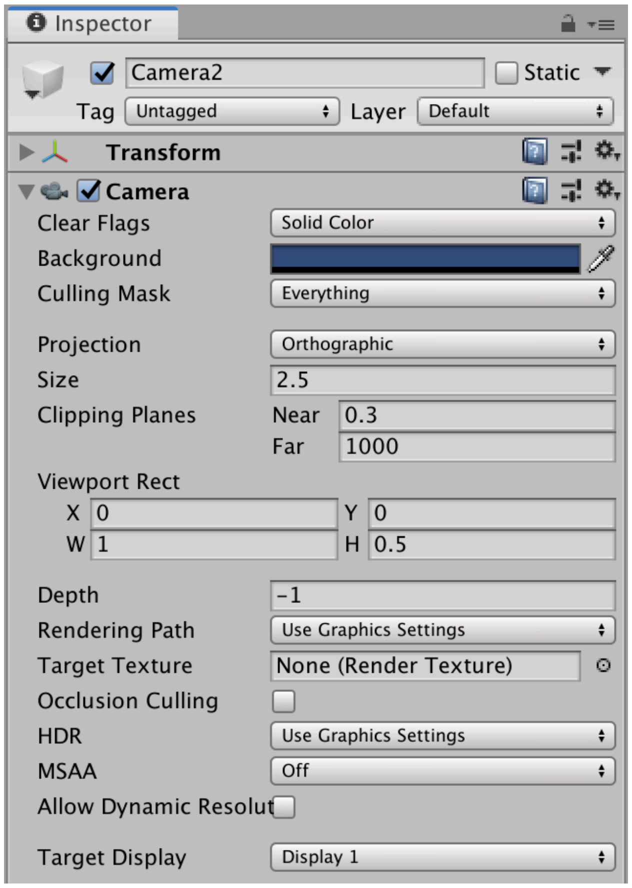


SampleScene
- Main Camera
- Player
- Obstacle
- Player2
- Camera2

Position Camera2 so that it is above Player2, while our prior Camera is positioned over Player. Note that we have two players and two cameras.



You can specify that Camera2 occupies only the bottom half of the game view by altering its Viewport Rectangle settings on its Camera component. Take note of all the other settings as well:


Inspector

Camera2 ✓ Static

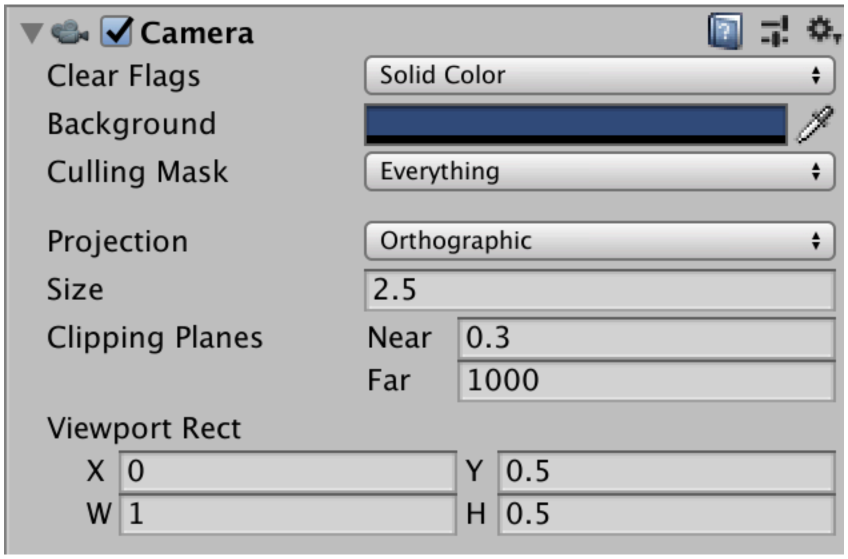
Tag Untagged Layer Default

Transform

Camera ✓

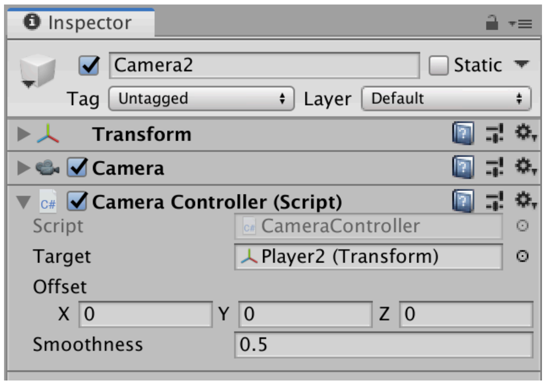
Clear Flags	Solid Color
Background	
Culling Mask	Everything
Projection	Orthographic
Size	2.5
Clipping Planes	Near 0.3
	Far 1000
Viewport Rect	
X 0	Y 0
W 1	H 0.5
Depth	-1
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	☐
HDR	Use Graphics Settings
MSAA	Off
Allow Dynamic Resolut	☐
Target Display	Display 1

Rename “Main Camera” to just “Camera” and alter its Viewport Rectangle settings on its Camera component so that it occupies the top half of the game view. Notice that we are also changing the size to 2.5:


Camera ✓

Clear Flags	Solid Color
Background	
Culling Mask	Everything
Projection	Orthographic
Size	2.5
Clipping Planes	Near 0.3
	Far 1000
Viewport Rect	
X 0	Y 0.5
W 1	H 0.5

Add a CameraController to Camera2 and set it to follow Player2:


Inspector

Camera2 | Static

Tag: Untagged | Layer: Default

Transform

Camera

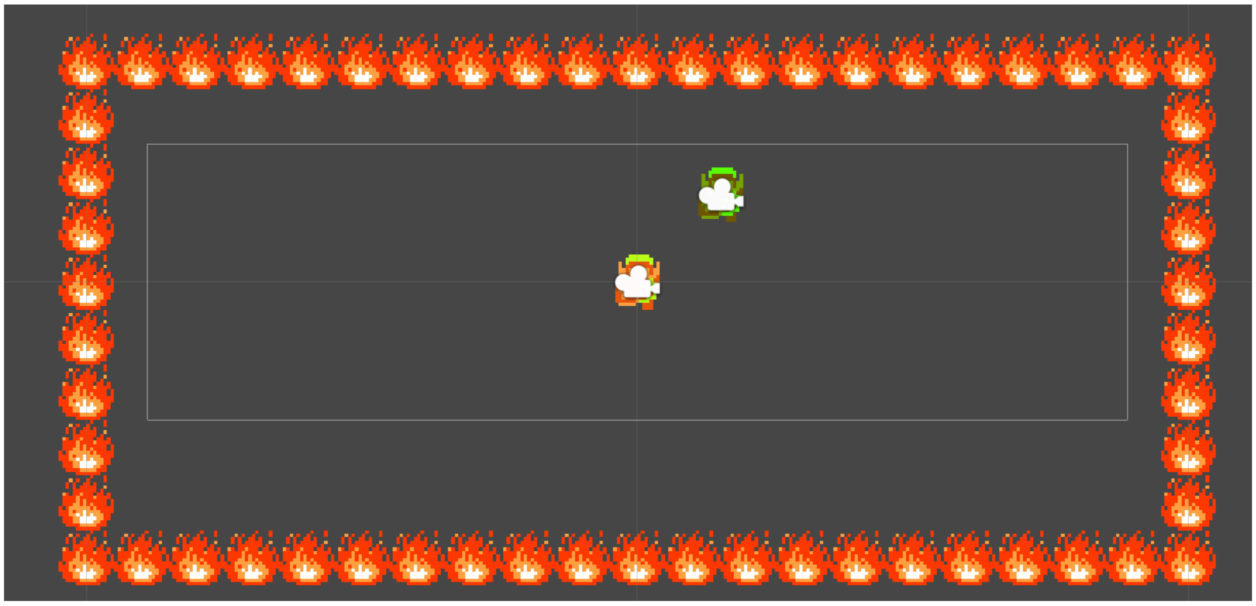
Camera Controller (Script)

Script	CameraController
Target	Player2 (Transform)
Offset	X 0 Y 0 Z 0
Smoothness	0.5

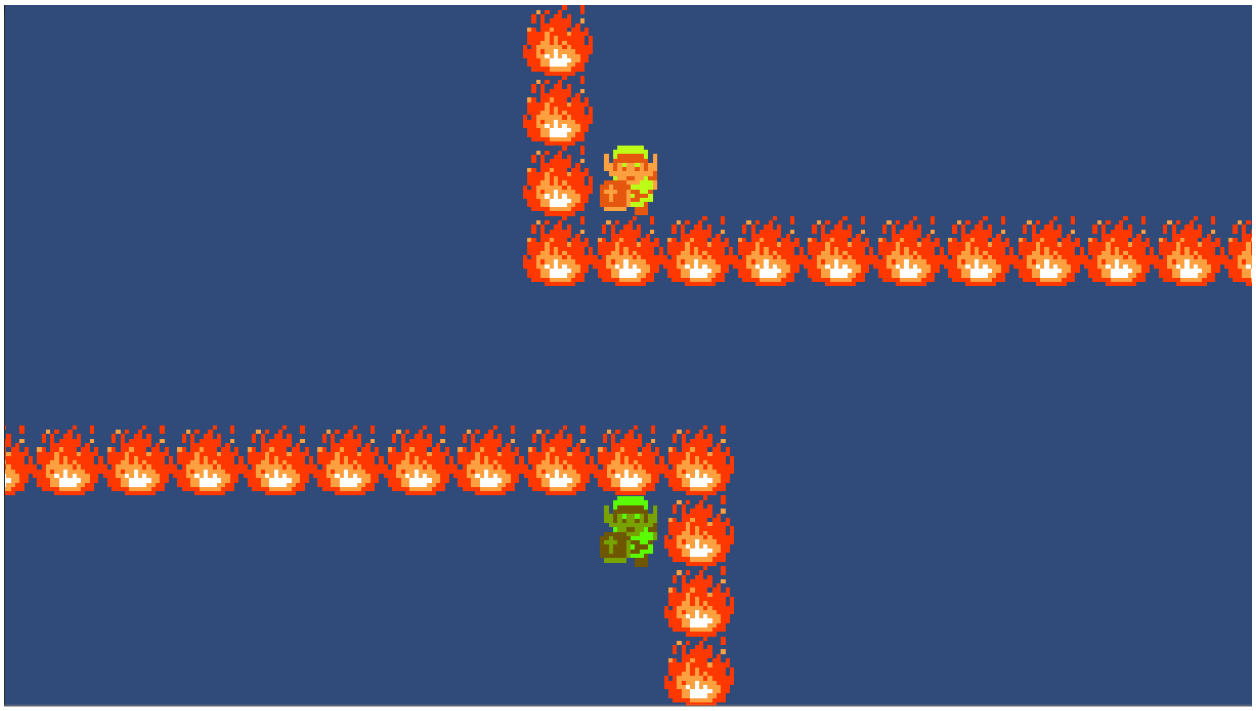
Playtest and ensure that the top half of the screen follows Player and the bottom half follows Player2. You made need to ensure the cameras are far enough back to see the character sprites.

9) Environment

Create an enclosed environment using level Obstacle objects. **Notice how the environment is larger than the camera view rectangle.**



Playtest and notice that the camera pans past level boundaries. This could be awkward for some level designs where you're in an enclosed room or dungeon, and don't want the camera to show a bunch of empty space.

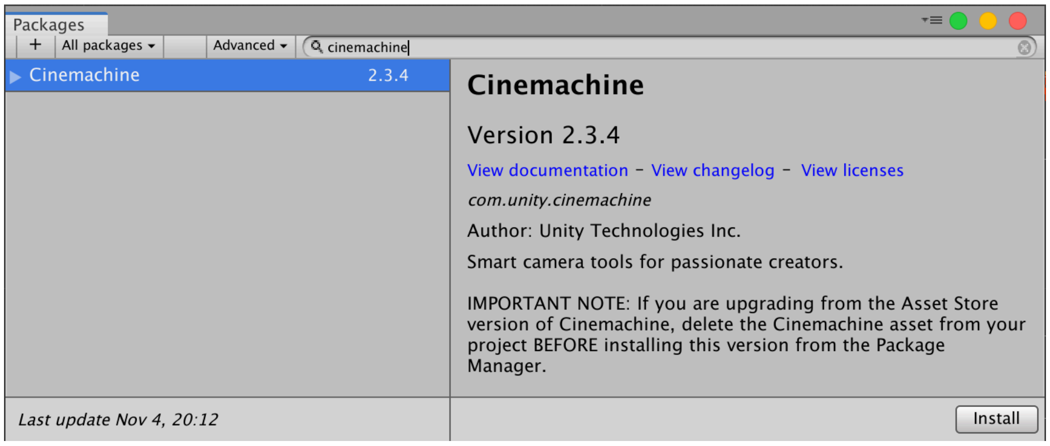


The Cinemachine camera has advanced logic for handling these situations.

10) Follow Camera (Cinemachine Approach)

We are going to switch Camera2 (bottom view) with a more advanced Camera. **You MUST leave the top half camera to the Smoothdamp technique so that it can be graded.**

To install Cinemachine, open the Package Manager from Window->Package Manager and search for Cinemachine.



Cinemachine 2.3.4

Cinemachine

Version 2.3.4

View documentation - View changelog - View licenses

com.unity.cinemachine

Author: Unity Technologies Inc.

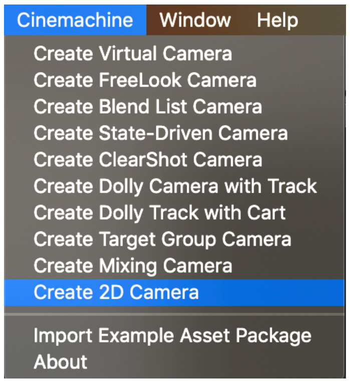
Smart camera tools for passionate creators.

IMPORTANT NOTE: If you are upgrading from the Asset Store version of Cinemachine, delete the Cinemachine asset from your project BEFORE installing this version from the Package Manager.

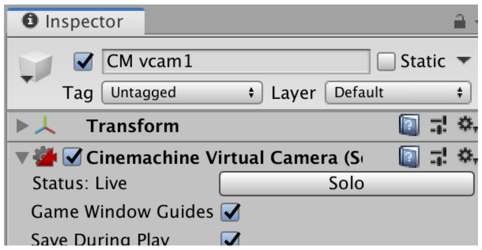
Last update Nov 4, 20:12

Cinemachine uses “Virtual Cameras” in conjunction with standard Cameras. The Cinemachine virtual camera can be used to specify the cinematic qualities, while the actual Camera component still renders the scene.

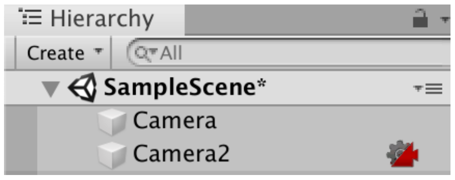
Once Cinemachine is installed, create a virtual 2D camera:



This creates a Cinemachine Virtual Camera object, which we will configure soon:

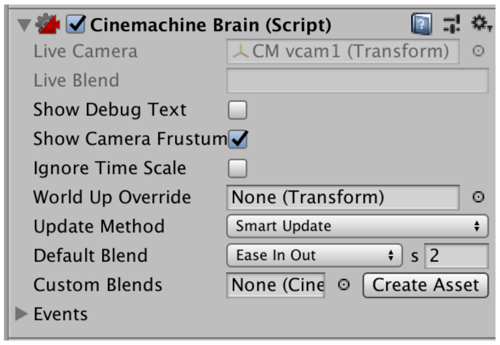


Check your Cameras. Creating a Virtual Camera will have also automatically added a Cinemachine Brain to one of your Cameras.



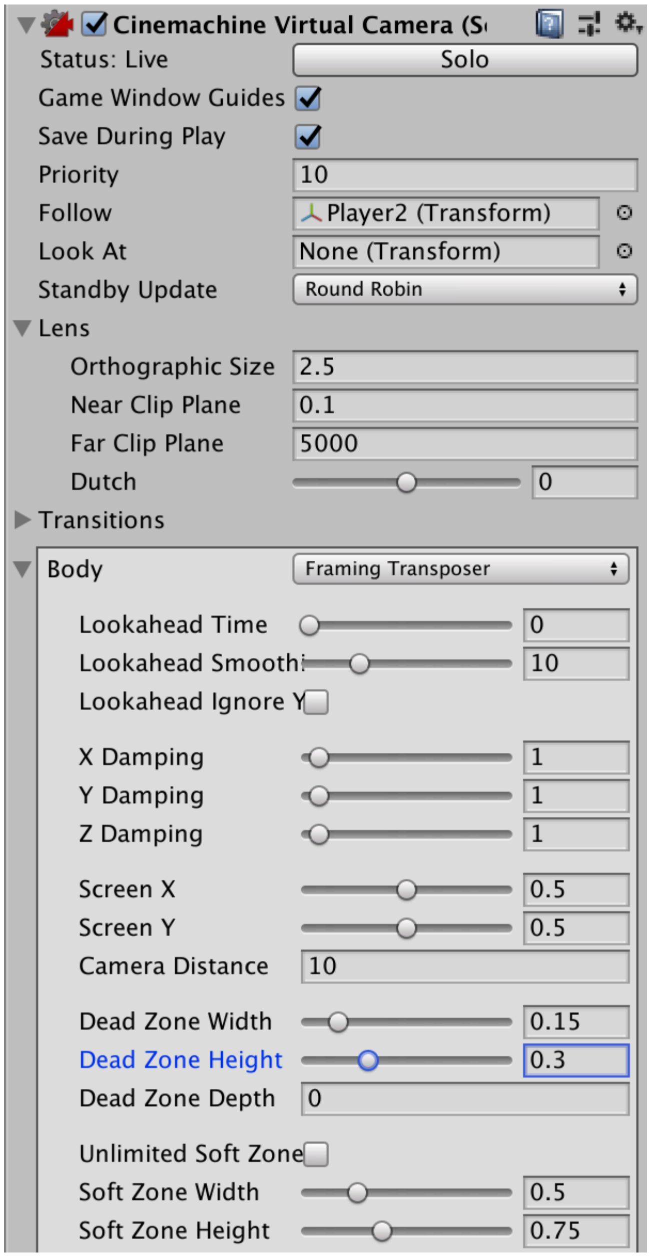
If the Cinemachine Brain was added to your top-half/player1 camera, remove the component and add it to Camera2 instead. The top camera will be graded for the smoothdamp technique. You will lose points if you can't properly grade both camera techniques in their appropriate halves.

Be sure to remove CameraController from Camera2 also. Cinemachine is a replacement.



With the Cinemachine Brain on Camera2, we can return to the Virtual Camera to begin configuring its options.

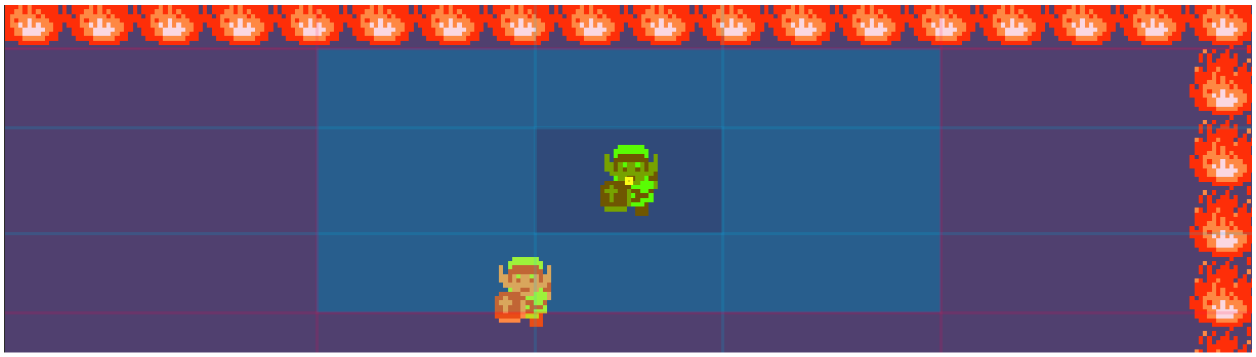
Similar to before, we use a Size of 2.5 and tell the Camera to follow Player2. We also specify Dead Zones and Soft Zones which define smoothing boundaries.



Cinemachine Virtual Camera (S

Status: Live	Solo
Game Window Guides	✓
Save During Play	✓
Priority	10
Follow	Player2 (Transform)
Look At	None (Transform)
Standby Update	Round Robin
Lens	
Orthographic Size	2.5
Near Clip Plane	0.1
Far Clip Plane	5000
Dutch	0
Transitions	
Body	Framing Transposer
Lookahead Time	0
Lookahead Smoothi	10
Lookahead Ignore Y	☐
X Damping	1
Y Damping	1
Z Damping	1
Screen X	0.5
Screen Y	0.5
Camera Distance	10
Dead Zone Width	0.15
Dead Zone Height	0.3
Dead Zone Depth	0
Unlimited Soft Zone	☐
Soft Zone Width	0.5
Soft Zone Height	0.75

In this diagram, the camera will follow the player such that the player never ends up in the RED portion of the frame. Meanwhile, the BLUE portion of the frame will trigger a smooth camera follow. The CENTER portion of the frame allows for free character movement without any effect on the camera.

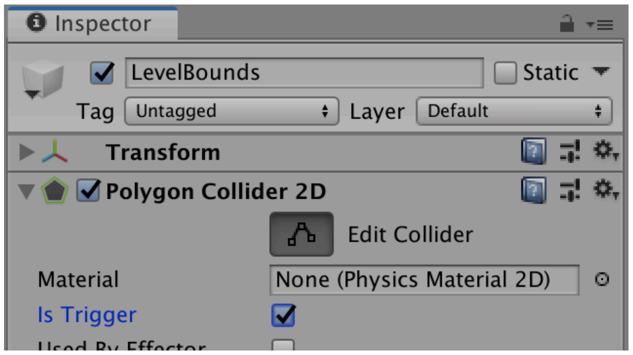


Playtest your game to ensure the camera actually follows these constraints.

11) Cinemachine Level Boundaries

While playtesting, you may have noticed the Cinemachine camera still goes passed the level bounds.

We need to create a collider to represent the valid camera coverage area. Create an empty game object called LevelBounds and attach a PolygonCollider2D to it.



Inspector

LevelBounds ✓ · Static ☐

Tag: Untagged · Layer: Default

Transform

Polygon Collider 2D ✓

Edit Collider

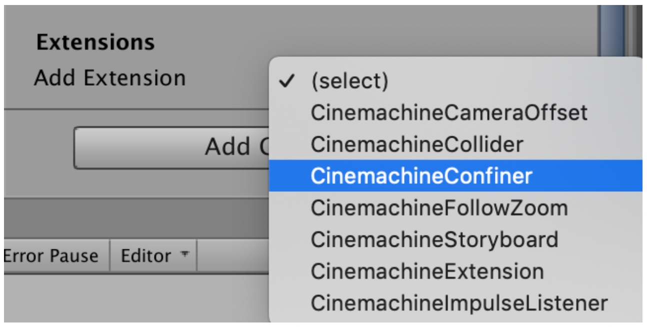
Material: None (Physics Material 2D)

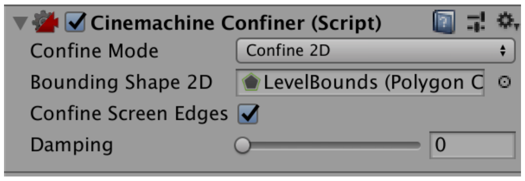
Is Trigger: ✓

Used By Effector: ☐

Set the collider to Trigger and Edit Collider so that the shape covers your level design. The polygon collider allows you to cover levels of any shape. Click anywhere on the lines to add a new point. Ctrl/Cmd-Click to remove points.

At the bottom of the Virtual Camera component, add Confiner extension, and set your LevelBounds object as the Bounding Shape.





Cinemachine Confiner (Script) ✓

Confine Mode: Confine 2D

Bounding Shape 2D: LevelBounds (Polygon C

Confine Screen Edges: ✓

Damping: 0

Playtest to see that Camera2 no longer moves beyond the level boundaries: