Rooms

This page explains how the advanced rooms system works and how you can leverage it to link parts of your levels together.

The Rooms System

The TopDown Engine has, for a long time, offered the ability to connect different scenes together, or to use teleporters to link parts of a level together. With v1.8, the TopDown Engine now comes with a powerful and versatile **Rooms system**. This lets you **cut** a level into "rooms" (whether they're visually actual rooms or just zones is up to you), and **connect** these Rooms together using the **Teleporter** class (whether these Teleporters are represented visually as doors, portals, nothing, or anything else is also up to you). It leverages the power of Unity's Cinemachine camera system to handle camera transitions between rooms, as well as a few MoreMountains classes to handle fades and sprite masks. This page explains how to do that, and covers the other features and options of the **Rooms system**.

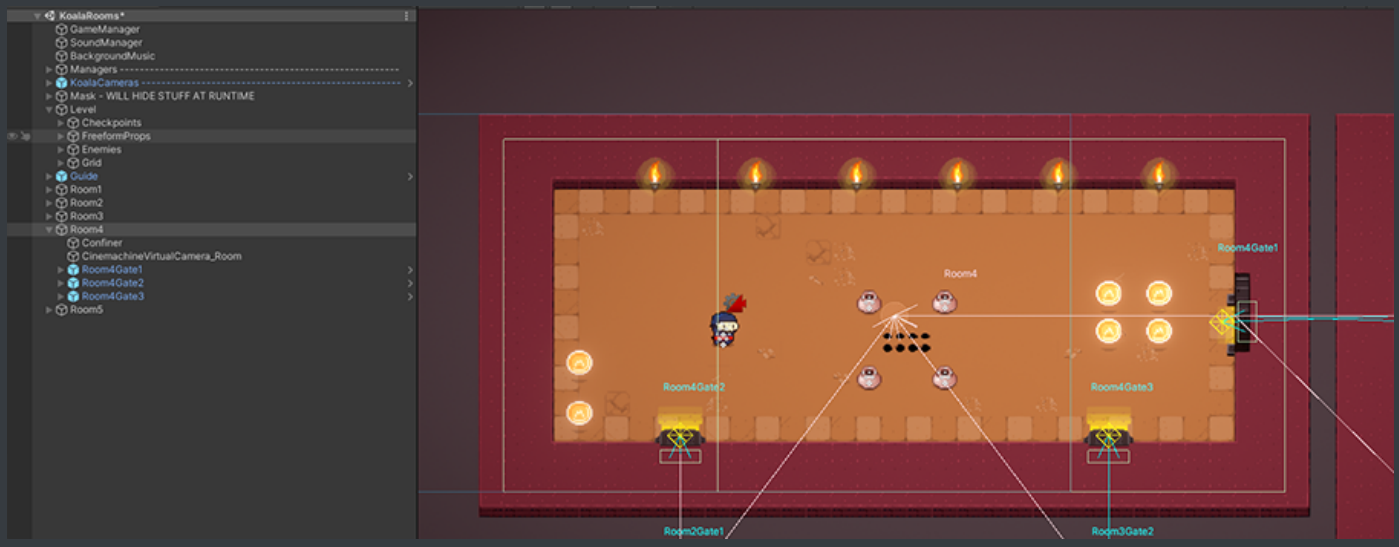
The **KoalaRooms demo scene** is a perfect example of that system, showcasing many Rooms, and different ways to setup teleporters. You can also check out the MinimalRooms3D demo scene for an example of how to setup rooms in 3D.

Rooms

To start things off, you'll want to create **Rooms** in your level. Whether your level is made of tiles, of 2D sprites or 3D models doesn't matter. To create a room, simply create a new empty object, add a **BoxCollider2D** to it if you're in 2D, or a **BoxCollider** if you're in 3D, and a **Room** component. Resize the collider so that it matches the size of your room.

Nested under the Room, as children, you'll want to add :

- a **confiner** : an empty game object, with a BoxCollider.
- a **Cinemachine Virtual Camera**. On this camera you'll likely want to have a CinemachineCameraController, and a CinemachineConfiner. On that CinemachineConfiner's inspector you'll want to set the Confine Mode to Confine3D, and drag the confiner you just created into the bounding volume slot. In 2D, you'll also probably gonna want to check Confine Screen Edges.
- one or more **Teleporters** (we'll get to that)



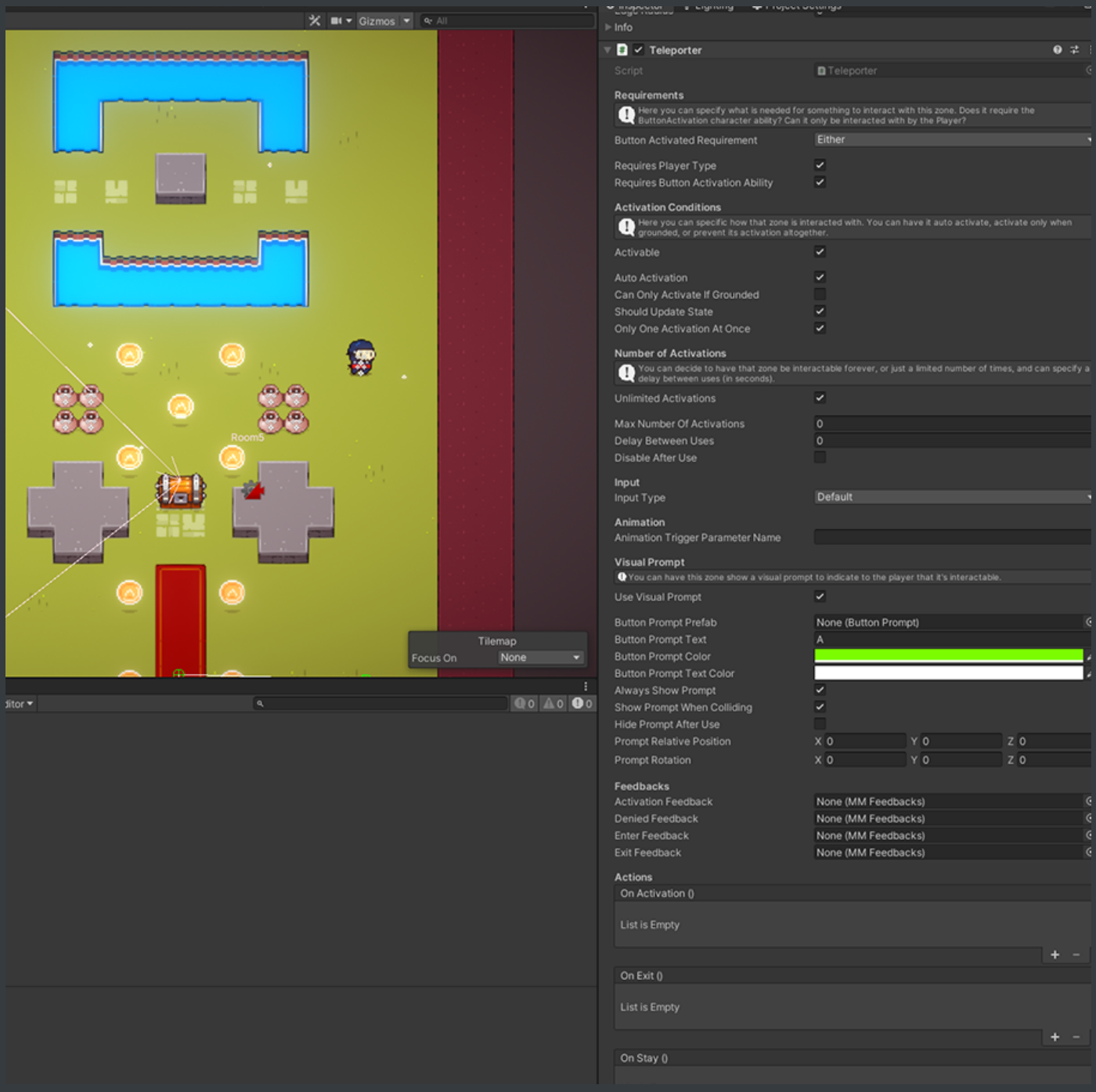
Room4 in the KoalaRooms demo scene

The Room class is responsible for **resizing the confiner** dynamically on Start to match the camera size (in 2D only), keeping track of its status (visited or not) and triggering events when the player enters the room, exits it, enters it for the first time.

Once you've created a room, the best way to go to add more is to **duplicate** it, reposition it, and resize its colliders to make more. Make sure to give them **unique names**. While not mandatory, this will make finding them later much easier.

Teleporters

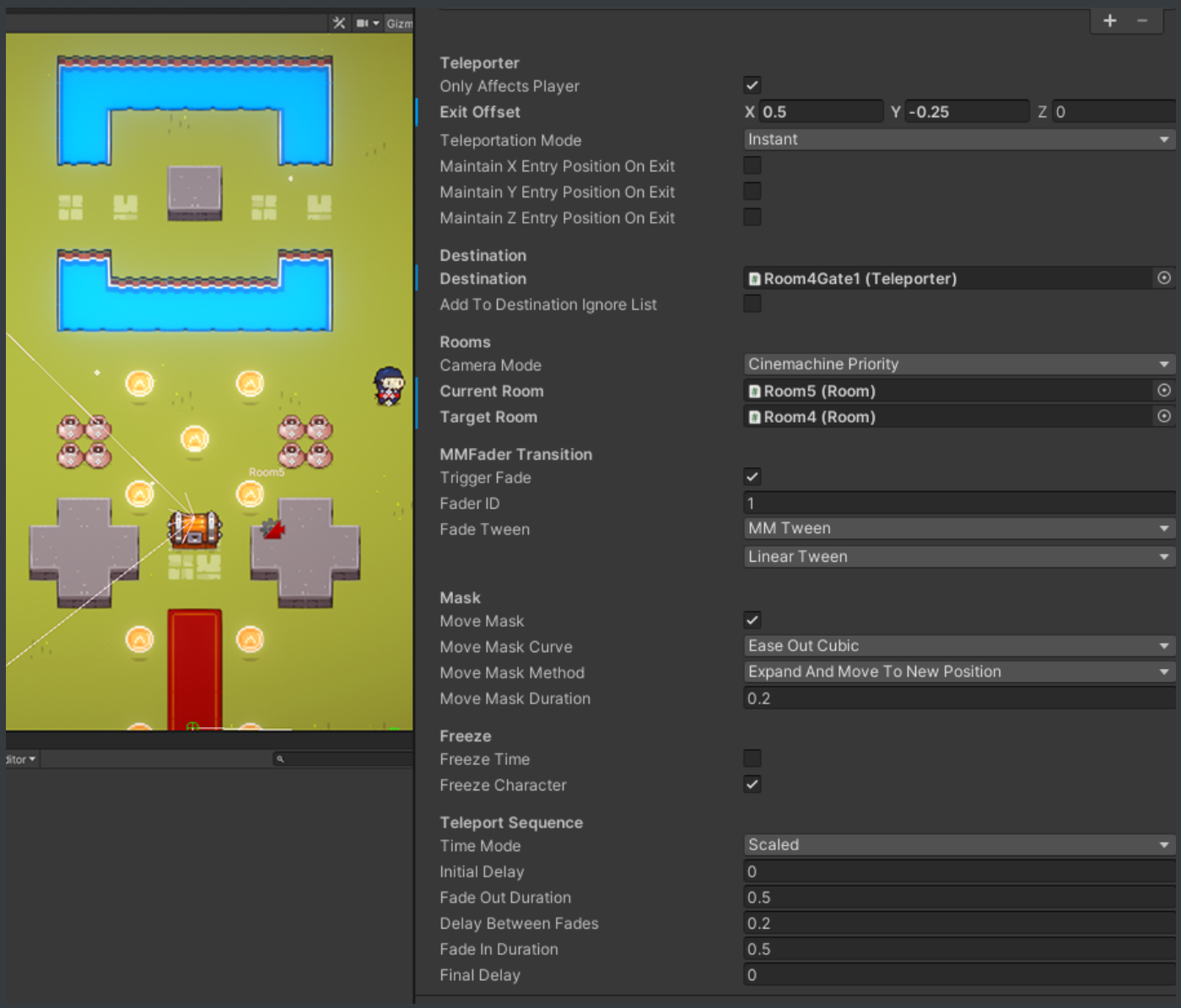
Now that you've got multiple rooms, it's time to add ways to go from one to the other. This is done using the **Teleporter** class, and its many options.



The Button Activation part of the Teleporter's inspector

To start things off, add an empty game object within your room, add a **BoxCollider2D** (in 2D) or **BoxCollider** (in 3D) to it (IsTrigger set to true), and add a **Teleporter** script to it. Teleporters have a ton of options. You can define activation requirements, activation conditions, amount of activations, prompts, input, feedbacks, and much more. For a door between rooms, maybe you'll just want to set **AutoActivation** to true, so that any Character that gets into the collider gets moved to the new room.

The **Teleporter** specific fields are at the bottom of the inspector (the top is common to all ButtonActivated zones).



The rest of the Teleporter's inspector

The only mandatory things to set will be the **Destination**, where you'll have to drag and drop another teleporter, as well as the **Target Room**, where you'll need to drag and drop a Destination Room.

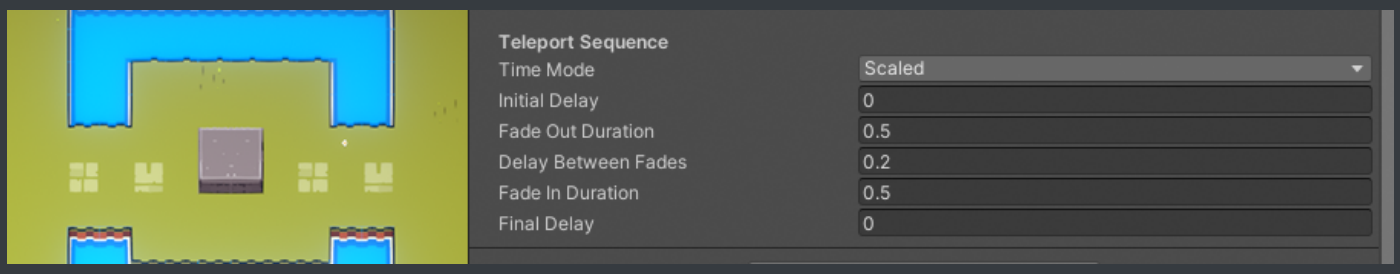
In the **Teleporter** section you can decide whether that teleporter only affects Player characters, an exit offset (added to the Teleporter's transform position when something exits it), whether the teleportation should be instant or a tween between entry and exit positions, and whether x or y positions should be maintained.

The **Freeze** section lets you choose whether you want the teleporter to freeze time and/or the Character when entering it.

In the **Rooms** section you don't have to set the **CurrentRoom** if your Teleporter is nested under a Room, it'll be automatically detected. You have different camera mode options, **Cinemachine priority** is the recommended one.

Teleporter Sequence

When getting into a Teleporter and activating it (by pressing an input, or by simply entering it, or via script, depending on your settings), you trigger a series of events known as the **Teleporter Sequence**. It's designed to be as customizable as possible, and easy to extend. It's made of several steps, whose duration (in seconds) you can customize from the bottom of the Teleporter's inspector.



The Teleporter Sequence's inspector

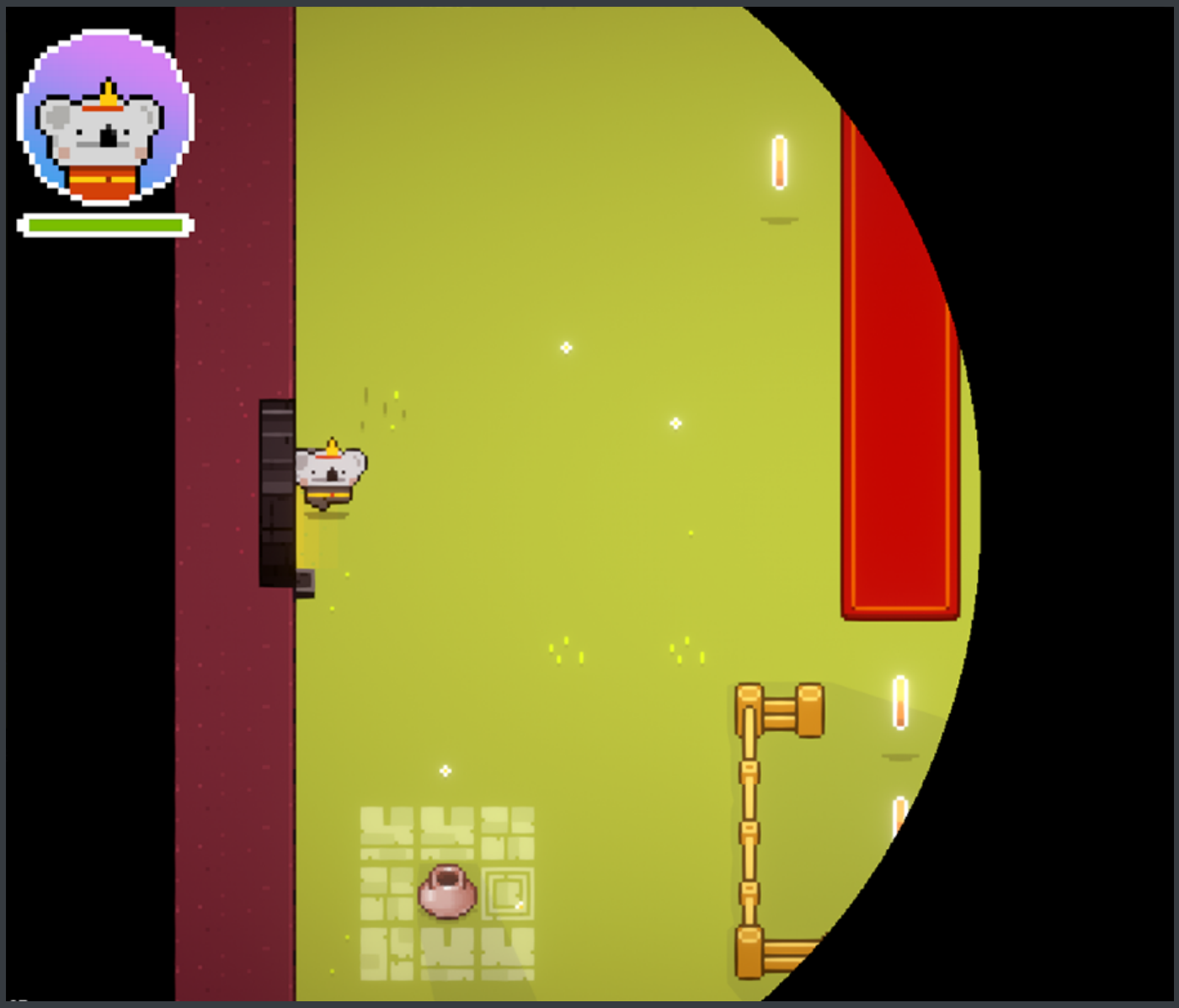
By default, the steps work as follows (note that not all of each step's events may happen depending on your settings) :

- **SequenceStart** : activates the zone (reducing the Use counter, triggering feedbacks, etc), prevents the camera from following, freezes time, freezes the character

- *wait for InitialDelay*
- **AfterInitialDelay** : fades the scene out via MMFader
- *wait for FadeOutDuration*
- **AfterFadeOut** : teleports the object, either instantly or by starting a smooth position tween, starts the camera transition, lets the current room know that the player has left, starts the mask transition
- *wait for DelayBetweenFades*
- **AfterDelayBetweenFades** : makes the camera follow again, fades the scene in via MMFader
- *wait for FadeInDuration*
- **AfterFadeIn** : nothing, for now
- *wait for FinalDelay*
- **SequenceEnd** : unfreezes time and the character

Each step is a separate method that you can easily override should you want to add more features.

Faders



The MMFaderRound in action in the KoalaRooms demo scene

The Teleporter's inspector lets you check whether or not you want to trigger a **Fade** when transitioning to your Destination. This in turn will trigger a MMFadeEvent, that will be caught by a MMFader object in your scene, if you have one. Most demo scenes do, usually within their UICamera object. By default the engine comes with two faders, the classic **MMFader** (which will tween a UI element's opacity - usually a black opaque sprite stretched across the whole screen, but not necessarily), and the **MMFaderRound**, which will apply a mask transition across a UI element.

2D Masks

Usually in 2D, when dealing with multiple rooms in a level you want to **obfuscate** the rooms the player isn't in at a specific time. Of course how you decide to do it is closely tied to your rendering choices. While it'd be a bit useless to try and provide a 3D solution for that, as it'd likely mess with your specific shaders, the engine comes with a 2D solution for that issue : the MMSpriteMask. You can create one from scratch, or simply copy the one in the **KoalaRooms** demo, it's ready to use.

On its inspector, you'll be able to have it automatically setup all renderers in the scene at start (which is useful otherwise it actually masks things in scene view, making level edition trickier). If you want an object to be ignored by the Mask at runtime, simply put a **NoMask** tag on it.