MMM

Instruction Manual v1.0.0

This document is an excerpt from the online documentation. We recommend you use the online documentation as it contains more details and is updated regularly.

Documentation is available at:

https://jnamobile.zendesk.com/hc/en-gb/categories/115000058848-MMM-Metroidvania-Mini-Map

You can also check out video tutorials at:

https://www.youtube.com/playlist?list=PLbnzW2Y4qytKsL2Fkv_k-cWimDCZS_0uj

Quick Start - How to Create a Map

MMM is strongly focused on being simple to use, you can get a map going very quickly with only a few steps, here's how...

Setting up the Map

Add the Map Manager

Start by adding a *MapManager* component. This is the key component that links the Map pieces to the Player, UI, and so on.

1. Create a new GameObject.

2. Add the *MapManager* component.

3. Set a grid size.

Grid size is a measure in world units. A room can, at its smallest, be 1x1 multiplied by the grid size. If your world was made up of rooms 5x5 world units in size then you could use a grid size of 5. If your rooms can be 2.5, 3.0, 4.5, 6.0 or 10.0 you might use (for example) a grid size of 0.5.

4. Set a *Player* GameObject or a *Find Player By Tag* tag.

Add the MapRooms

MapRooms are used to define the individual rooms. You will add one for each room in your level. MapRooms may be on the same GameObject as your level geometry (handy if you are moving things around) but can also be on a separate GameObject.

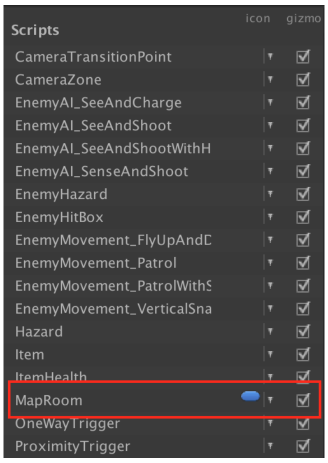
1. Add a new GameObject or select a GameObject from your exiting level geometry.

2. Add the *MapRoom* component to the new or selected GameObject.

3. Use the sliders to adjust the room *width* and *room* height to match your level geometry.

For example if your grid size is 2 and your room is 4.0 x 6.0 then your room size would be width 2 and height 3.

If you have the *MapRoom* gizmo turned on (see screenshot) then the rooms will be shown in the scene view as a translucent cube.



Scripts	icon	gizmo
CameraTransitionPoint		✓
CameraZone		✓
EnemyAI_SeeAndCharge		✓
EnemyAI_SeeAndShoot		✓
EnemyAI_SeeAndShootWithH		✓
EnemyAI_SenseAndShoot		✓
EnemyHazard		✓
EnemyHitBox		✓
EnemyMovement_FlyUpAndD		✓
EnemyMovement_Patrol		✓
EnemyMovement_PatrolWithS		✓
EnemyMovement_VerticalSna		✓
Hazard		✓
Item		✓
ItemHealth		✓
MapRoom	●	✓
OneWayTrigger		✓
ProximityTrigger		✓

4. You can optional pick an *override colour* and *override sprite* for your room. Depending on Map UI settings these will be shown on the map display.

Repeat steps 1-4 for each room in your game.

Setting up the UI

The UI for the map is created by the component *UIMapContent.*

1. Create or select your UI Canvas.

2. Create a child GameObject to use for your map.

3. Add the *UIMapContent* component to this GameObject.

4. Size and position the Map. You may want to add an *Image* component to use as the background for the map.

If you press Play you should see that the (rudimentary) map becomes visible as your Player GameObject enters each room. Color and sprites will be picked up from the MapRoom, and if none are set a blank image will be used.

5. **Use the *Map Scale* to control the size of the objects in the map and the *Room Offset* to control the relationship between world position 0,0 and UI position 0,0.**

For example if you map is appearing too far to the left of screen you may want to set a *Room Offset* of (10,0,0).

Structuring the UI

In order to provide more control over visibility and map layout we will generally create a GameObject to hold the visible content. The individual Map pieces will be created as children of the *Visible Content* GameObject.

1. Create an empty child GameObject underneath your UIMapContent GameObject.

2. Assign this new GameObject to the *Visible Content* field of the *UIMapContent* component.

How It Works

MapManager and MapRooms

Awake()

When your scene is loaded the *MapManager* instance in the scene assigns itself to the *MapManager.Instance* static variable. Thus making itself available to other components.

It then loads any saved map data from PlayerPrefs. Any *MapRoomData* is registered in the static dictionary *MapRoom.allMapRooms* via the method *MapRoom.AddMapRoom(data).*

The MapRoom.allMapRooms dictionary maps fully qualified room names to *MapRoomData.* A fully qualified room name is a combination of scene name and map name and should be unique.

Start()

On Start each room in the scene registers itself with the *MapRoom.allMapRooms* dictionary; if the room already exists the content is updated to associate the MapRoom in the scene with the already loaded *MapRoomData*. If it does not exist new *MapRoomData* is created.

Once data is loaded the game loop can begin.

Update()

Each tick of the game loop (update) the *MapRooms* check the location of the Player vs the rooms boundaries. If the player is in the room (and was not previously in the room) the *MapRoom* informs the *MapManager* that the player has entered the room (*MapManager.Instance.EnterRoom*).

Conversely if the player was previously in the room and is no longer in the room then the *MapRoom* informs the *MapManager* that the player is no longer in the room (*MapManager.Instance.LeaveRoom*).

When a Player enters a room that has not yet been revealed, additional behaviour is triggered such as revealing Points of Interest (see Points of Interest below) and sending a Reveal event (see Events below).

Points of Interest

PointsOfInterest are loaded and registered with the *MapManager* in a similar way to *MapRooms.* The only major difference is that during load *MapRooms* scan *PointsOfInterest* to see if they are within the *MapRooms* boundaries. If the Point of Interest is within the boundaries of a MapRoom it is associated with that MapRoom.

When a *MapRoom* is revealed any *PointsOfInterest* associated with that room are also revealed.

NOTE: A *PointOfInterest* can be marked as *hidden* which means it is not revealed when the room is revealed.

NOTE: A *PointOfInterest* can be marked as *revealed* which means it is always revealed regardless of the Player having revealed the *MapRoom* the point of interest occurs in.

MapManager Events

Whenever the state of the map changes the *MapManager* sends an event. The following events are sent:

Event	Description
EnteredRoom	Sent when the player enters a room.
LeftRoom	Sent when the player leaves a room.
RevealedRoom	Sent when a room is revealed (generally when the player entered the room for the first time, although rooms can be revealed by calling RevealRoom directly for example to show a target destination on the map).

PointOfInterestAdded	Sent when a point of interest is revealed.
PointOfInterestRemoved	Sent when a point of interest is removed from the scene (e.g. when an item is collected or an enemy dies).

How it Works - UI

UIMapContent

The main UI component is the *UIMapContent* component. This component listens to *MapManager* events and draws and updates the UI in response to these events.

The *UIMapContent* should be added to a GameObject that is a child of your UI canvas. You can add multiple *UIMapContent* objects to support multiple map views (for example a mini-map and a full map).

UIRoom

When a *MapRoom* is revealed a *UIRoom* component is created. You can provide your own *UIRoom* component by creating a prefab which contains a *UIRoom* and assigning it to the *roomPrefab* variable of the *UIMapContent* component. If no prefab is present a GameObject with a *UIRoom* component and an *Image* component will be generated automatically.

UIPointOfInterest

When a Point of Interest is revealed a UIPointOfInterest component is created.

NOTE: You can turn off *PointOfInterest* creation by setting the *PointRenderType* variable to *DONT_SHOW* in the *UIMapContent* settings.

You can provide your own UIPointOfInterest component by creating a prefab which contains a UIRoom and assigning it to the pointPrefab variable of the *UIMapContent* component. If no prefab is present a GameObject with a *UIPointOfInterest* component and an *Image* component will be generated automatically.

CurrentRoomIndicator

To indicate the room the player is currently in a current room indicator component is transformed to the position of the current room whenever the current room changes. The Current Room Indicator should be created in your scene as a child of the *UIMapContent visibleContent* (if *visibleContent* is *null* place the Current Room Indicator GameObject as a child of the *UIMapContent*). You can use any combination of GameObjects and UI components for your Current Room Indicator.

NOTE: The current room indicators size is scaled along with the room size, if you wish to have a room indicator that does not scale see: Creating a fixed size Current Room Indicator.

As an alternative to, or in addition to, the Current Room Indicator you may wish to show the players exact position on the map. To do this you can create a GameObject in your UI canvas and attach the *UIPlayerIndicator* component. This component will be transformed (and optionally rotated) to match the players position on the map.

Supporting Multiple Scenes

If you wish to have the same map counting rooms that spread across multiple scenes you will need to create a *SpriteDictionary*. This stores references to sprite overrides used by one scenes so that they can be loaded in to the map shown on another scene.

To create a Sprite Dictionary

1. Create a new *GameObject* and name it *SpriteDictionary*.

2. Add the *SpriteDictionary* component to this *GameObject*

3. Make a prefab from your *SpriteDictionary* by dragging to the project hierarchy.

4. Add this prefab to all scenes using the map.

Maintaining the SpriteDictionary

The *SpriteDictionary* will be automatically updated when you assign a sprite to a *MapRoom* or *PointOfInterest* however its important that you click the **Apply** button on the *SpriteDictionary* GameObject to ensure that the prefab is updated and thus making updates available to all scenes.

Alternatively you can manually drag the sprites you want to use on your map in to the list on the *SpriteDictionary* component and then hit **Apply** to apply the changes to the prefab.

TIP: If the sprite dictionary is not correctly updated you will see that the rooms in the UI appear but have the incorrect sprite.

REMEMBER: To support multiple scenes and the SpriteDictionary all of your map and point of interest sprites must have unique names.