Update history:

- 2017-04-13: Initial release on Marketplace for UE4.15.
- 2017-05-09: Rendered background can show navigation mesh.
- 2017-05-19: Multi-level backgrounds, priority backgrounds and Z-ordering.
- 2017-05-30: Added support for UE4.16. Minimap widget can specify icon size multiplier.

JOURNEYMANS MINIMAP

This document describes how to use the **Journeyman's Minimap** code plugin for Unreal Engine 4.15 and up. If you have any questions, please reply to the support thread on the Unreal Engine forums.

Author: Zhi Kang Shao
E-mail: zhikang.shao@gmail.com
Forum: `https://forums.unrealengine.com/showthread.php?138793`
Video: `https://youtu.be/qqnaDa7nCbY`

Contents

1 Quick start guide

This is a compact guide on how to get started:

1. Add a **Minimap** or **Bordered Minimap** widget to your UMG HUD
2. Place a **Map Background** actor in the level to assign a background texture to an area
3. If minimap should follow the player: Add a **Map View Component** to your pawn class
4. Add **Map Icon Components** to actors that should appear on the minimap

Your minimap should now work correctly. To enable fog of war:

1. Place a **Map Fog** actor in the level
2. Add **Map Revealer Components** to actors that reveal fog
3. Configure **Map Icon Components** to show or hide in fog

The remainder of this document describes every step in more detail.

2 Detailed guide

2.1 Adding a minimap to the screen

The plugin includes a UMG widget to show the minimap. If you want to use a *borderless* minimap, add a **Minimap** widget to your HUD. You can set its size manually or let the size be controlled by a UMG layout widget, such as SizeBox, ScaleBox, VerticalBox, et cetera.

A *bordered* minimap can be achieved either by placing a graphic over the borderless Minimap yourself, or duplicating the included **Bordered Minimap** widget and making changes to its size and graphic. The Bordered Minimap also demonstrates zooming buttons and optional border rotation. To find the asset, you must enable **View Options: Show Engine Content** and **Show Plugin Content** in the Content Browser. By duplicating the asset to your project's Content folder this ensures that you can download future plugin updates without the risk of losing work. The Minimap and Bordered Minimap widgets are shown in Figure 1.

The minimap widget can be configured in various ways:

- **Auto Locate Map View**: Auto-detect an area to view. Discussed in Section 2.4.
- **Is Circular**: Enables a circle shaped minimap. By default it is rectangular.
- **Draw Camera**: Enables visualizing the player camera's view frustum in the minimap.
- **Floor Distance**: If Draw Camera is enabled, controls size of rendered trapezoid.
- **Fill Color**: Color to show beneath all background textures. Set alpha to 0 to disable.
- **Icon Scale**: A size scale applied to icons in this specific minimap.

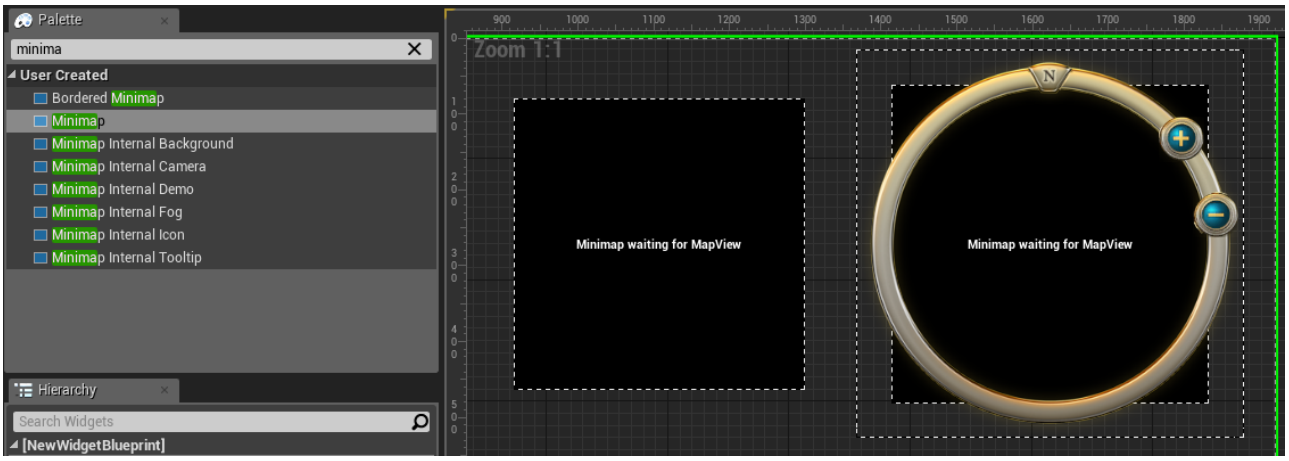


Figure 1: Add a Minimap or Borderless Minimap to your HUD.

2.2 Toggling between minimap and full screen map

Using standard UMG widgets, you can quite easily create a set-up that allows the player to switch between a minimap and a full screen map by holding a button. For example, create the following hierarchy of UMG widgets:

Switcher: Used to toggle between minimap and full screen map. Should span entire screen.

–**SizeBox**: Defines the minimap's size. Should anchor to a screen corner.

—**Minimap**: This is the minimap.

–**ScaleBox**: Stretch the childdren to full screen, while maintaining correct aspect ratio.

—**SizeBox**: Controls the minimap's aspect ratio by defining a base width and height.

——**Minimap**: This map will be full screen.

Whenever you want to show the minimap or the full screen map, you can call the Switcher widget's function **Set Active Widget Index**. Figure 2 shows how to implement showing the full screen map while Tab is pressed, and the minimap otherwise.

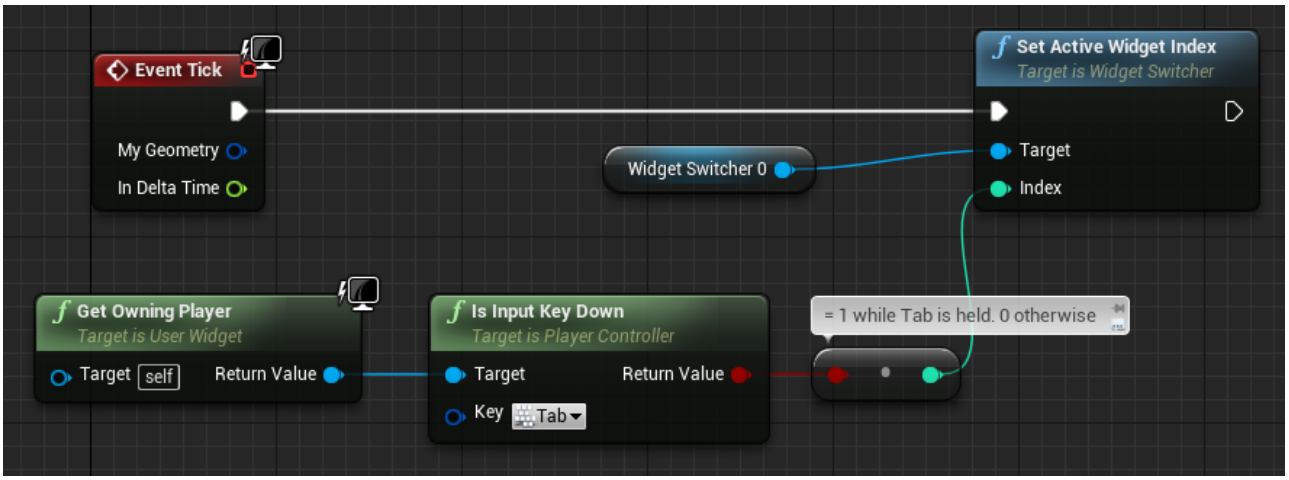


Figure 2: Player can hold Tab to show the full screen map.

When using multiple Minimap widgets, you may want icons to have a different size per minimap. Change the Minimap widget's **Icon Scale** for this. Set this to 3.0, for example, for a Minimap widget that is displayed full screen. This setting does not affect icons whose sizes are set in world space.

2.3 Minimap background set-up

You can assign background textures to one or multiple areas in the same level. Start by dragging a **Map Background** actor into the level. This actor has a box component. Use the top viewport to move, scale and rotate the actor so that the box covers the area that you want to assign a texture to.

Multiple floors

The background volume can have one or multiple floors, where the player's Z-position within the volume determines which background texture is shown. These are specified in the Map Background's list of **Background Levels**. If you don't want multiple floors, leave the list size as it is with one entry. Otherwise, add more levels by adding entries to the list. Per entry, you specify the **Level Height** in world units. The last level will use whatever space remains of the Map Background's box height. The Map Background actor's wireframe model in the side and frontal viewports will display the floor positions. Figure 3 shows a Map Background that is configured with multiple levels.

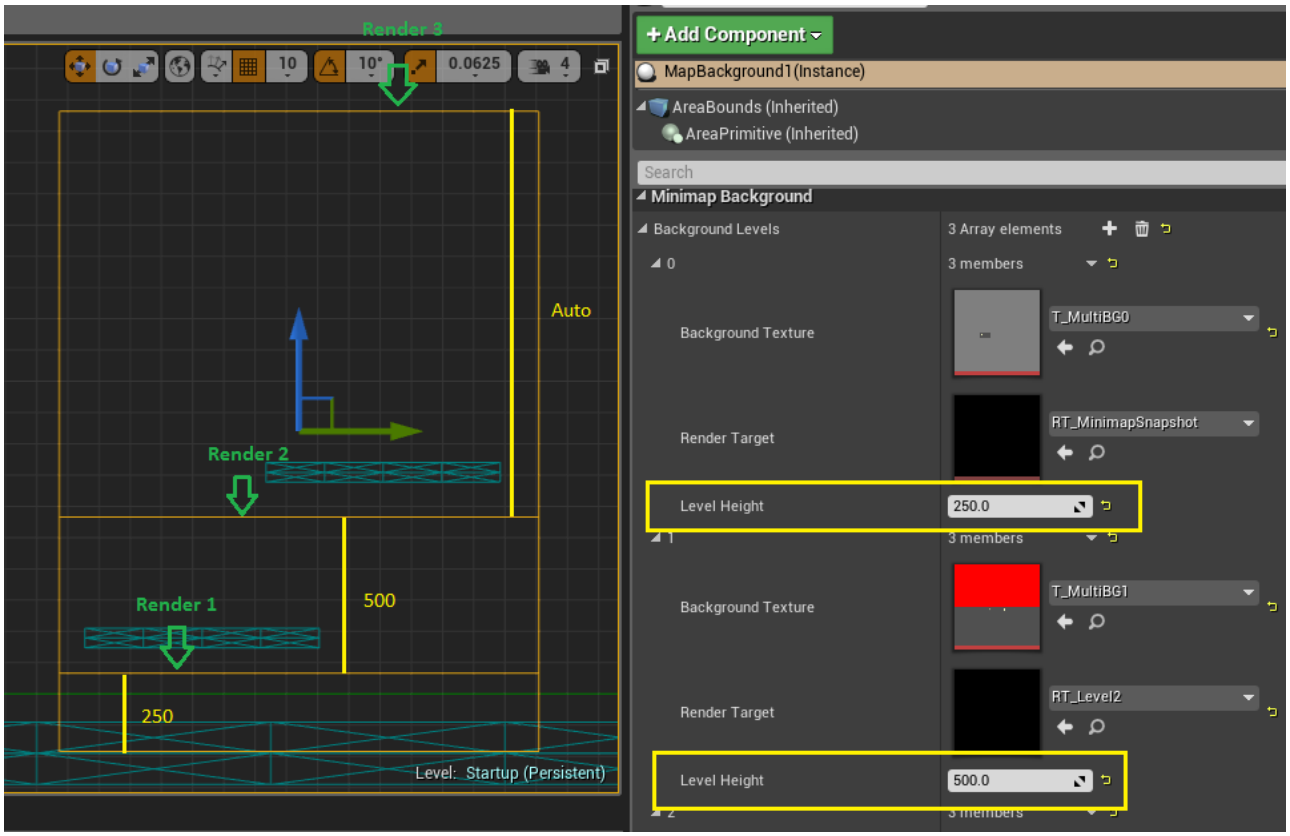


Figure 3: Backgrounds can be configured with multiple levels.

Painted backgrounds and background generation

Per background level you can choose whether to show a prepared background texture in-game or to generate a pixel perfect top-down render when the game starts. If a **Background Texture** is set, it will use that texture in-game. Transparent and semi-transparent textures are supported.

> ***Reminder:*** Be sure to set the Minimap widget's Fill Color alpha to 0, if you want the minimap to be see-through.

If you don't set the Background Texture, a render will be generated when the game starts and used as background texture. The plugin can also generate this render in the editor, so that you can export and paint over it in an image editor. To use this feature, create a render target asset and select this as

a background level's **Render Target**. Whenever you move the Map Background actor in the editor, the render is regenerated and stored in that asset. Any geometry above the level's ceiling is excluded from the render.

Since the render is pixel perfect, if you draw over it in an external image editor, import the result back in and select it as the floor's Background Texture, your drawing will line up perfectly. You can exclude actors from the render by adding actor classes to the Map Background's **Hidden Actor Classes** or adding level actor references to the **Hidden Actors** list. If you check the option **Render Navigation Mesh**, the navigation mesh will be included in the render. This is to make it easy for you to recognize the walkable area when preparing the painted texture. This feature is demonstrated in Figure 4.

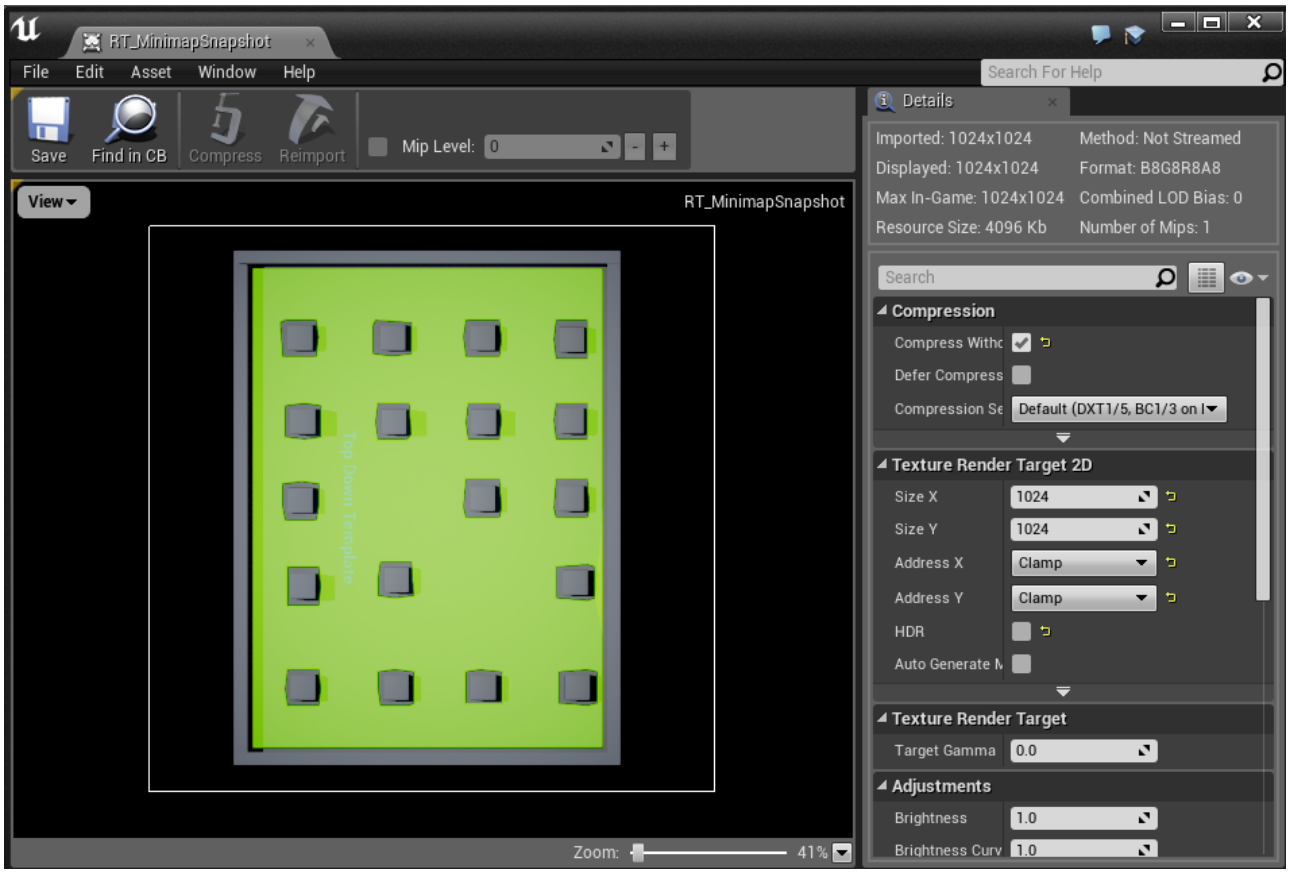


Figure 4: The navigation mesh can be included in a background render.

Overlapping backgrounds

When the volumes of two or more Map Backgrounds overlap, you can specify beforehand which background is rendered on top by setting the Map Background's **Z-Order** value: the one with higher Z-order is rendered on top. Map Backgrounds can also be configured so that when the player is inside multiple at once, only some of them are rendered. This is done by setting the **Priority** value: when the player is inside multiple Map Background volumes with differing Priority settings, only the one(s) with the highest priority are rendered, as well as nearby volumes with equal priority.

The intended use case for this is in-door environments: if you have a large outdoor level with some buildings that the player can enter, you can place a large, all-encompassing Map Background with the outdoor background texture and place smaller Map Backgrounds around buildings that specify in-door background textures. If you set the Priority of the in-door Map Backgrounds higher, while the player is inside a building it will only show the in-door texture. Icons can be configured to only show when the player is on the same floor or in the same priority volume.

Custom background rendering

You can change the **Background Material** to render the background texture in a custom way, for example to add animation. With the following blueprint exposed functions you can change the background appearance during gameplay:

- **Set Background Texture**: Changes the texture assigned to the area
- **Set Background Material**: Changes the material used to render the texture to the minimap
- **Reset Background Material**: Changes the material back to what it was initially

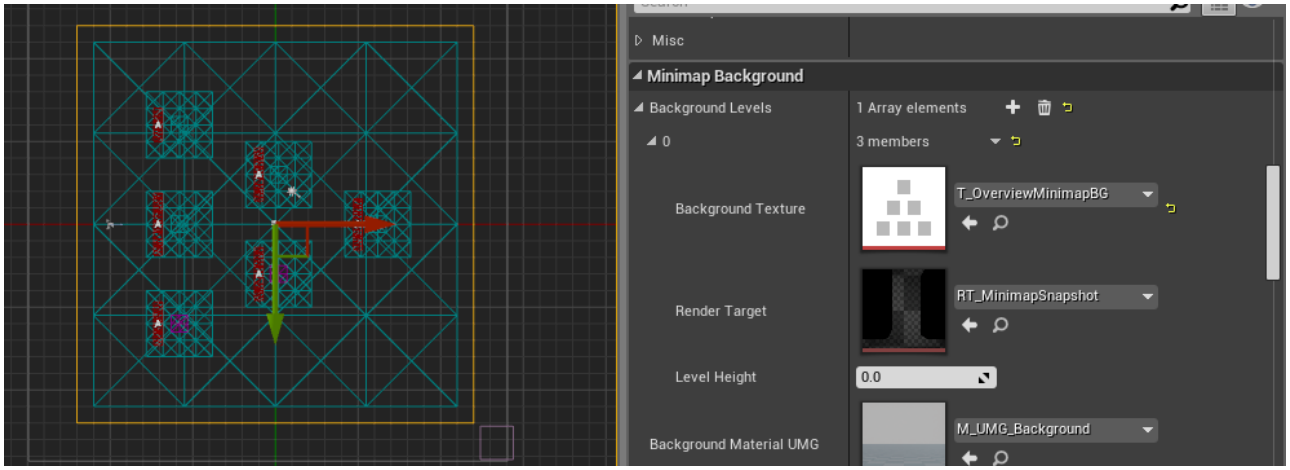


Figure 5: Place a Map Background in the level to assign a background texture

2.4 Minimap viewed area set-up

You control what part of the level is shown in the minimap. This is done by linking a **Map View Component** to the Minimap widget. A Map View Component is a special box component, as shown in Figure 6. The box shape defines what part of the level is rendered to the minimap. Linking a Map View Component to a Minimap widget can be done automatically or manually.

Map Backgrounds and Map Fogs are equipped with a Map View Component that matches the area that they cover. If the minimap should show the entire level at once and the level is covered by one Map Background or Fog actor, you can make the Minimap widget automatically use its Map View Component. In that case: set the Minimap widget's **Auto Locate Map View** setting to **On Map Background** (or Fog).

If the minimap should follow the player instead, add a Map View Component to the player pawn class. Set **Auto Locate Map View** to **On Player**. When the game starts, the minimap will automatically search for a Map View Component on the first player controller or its pawn. In multiplayer games it will only consider the locally controlled player. If at any time the active Map View Component is destroyed, for example the player pawn is removed from the world, the minimap will try to find a new target using its Auto Locate Map View setting. It will retry this with short intervals until a new Map View Component is found.

In all other cases, you can manually pass the Map View Component to a Minimap widget. Set Auto Locate Map View to Disabled. Then call the Minimap widget's **Set Map View** function after the game has started. Use this, for example, if the player can move the minimap separately from the player pawn: create a new actor with a Map View Component that can be moved through the level to pan the minimap.

Special case: If you have placed multi-level Map Backgrounds in the level, by default the minimap's active Map View Component's position is used to select the right floor to render. In some cases you may want another component's position to be used. In those cases, set the Map View Component's **Height Proxy** to another scene component. Do this for example, when a Map Background's view component is used to render the minimap, but the player's position should be used to select the floor to render: set the Map Background's Map View Component's Height Proxy to the player pawn's root component.

The Map View Component's size controls how far the minimap is zoomed in. You can change this during gameplay, either by calling the function **Set View Extent** to change the XY range that can be seen or **Set Zoom Scale** which acts as a multiplier to the view extent, i.e. a zoom scale of 2 doubles the area shown. The Bordered Minimap widget demonstrates zooming in and out with clickable buttons.

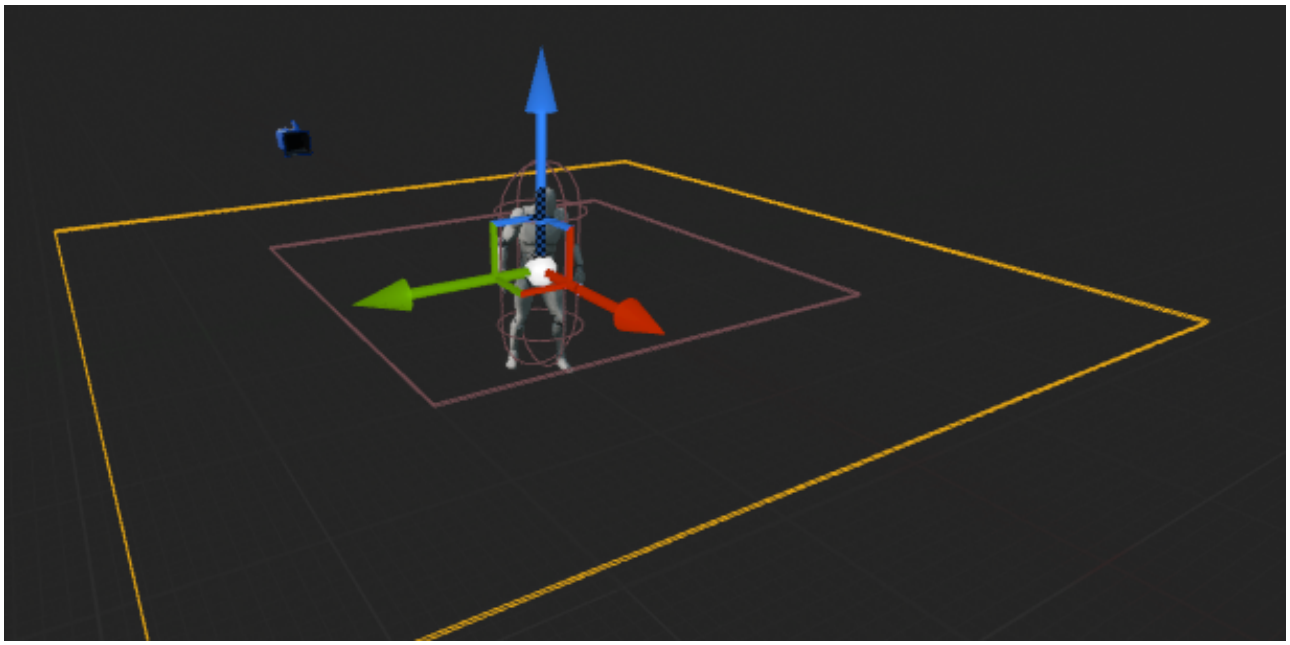


Figure 6: A Map View Component is a box that defines the area to show on the minimap.

2.5 Icon set-up for actors and areas

Make an actor appear on the minimap by adding a **Map Icon Component** to it. Configure the component to define how the actor appears:

- **Icon Texture**: The texture of the icon. Using textures with square dimensions is advised.
- **Icon Material**: The material used to render the texture. Animated materials are supported: a Time parameter is passed to the material, in seconds passed since the material was set. Animations can be gameplay controlled: this is discussed in Section 2.6.
- **Icon Visible**: Whether the icon is initially visible.
- **Icon Rotates**: Whether the icon rotates to represent the actor's yaw rotation.
- **Icon Size Unit and Size**: The icon's size, either in pixels or world units.
- **Icon Draw Color**: A color multiplier applied to the icon's texture.
- **Icon Z Order**: Icons with higher Z order are rendered on top and have priority when clicked.
- **Icon Interactable**: Enables mouse events on the icon.

- **Icon Background Interaction**: Icon visibility can be limited to only when on the same floor, or only when in a same priority Map Background.

Objective arrows

When an icon is outside the minimap, an arrow can be shown at the edge of the minimap:

- **Objective Arrow Enabled**: Whether the arrow is enabled
- **Objective Arrow Texture**: The arrow's texture
- **Objective Arrow Material**: The material used to render the arrow's texture
- **Objective Arrow Rotates**: Whether the icon rotates to point to the actor's location
- **Objective Arrow Size**: The arrow's size, always in pixels

All icon settings can be changed during gameplay via blueprint exposed functions such as **Set Icon Texture, Set Icon Material**, et cetera.

Highlighting areas

Highlight an area on the minimap by spawning an actor with a Map Icon Component in the center of the area. Set the icon's **Size Unit** to **World Space**. This means that an icon with size 500 will cover exactly an area of 500 x 500 world units on the minimap. When the minimap zooms in or out, the icon automatically resizes itself to ensure this. Consider using one of the included animations to highlight gameplay objective areas.

2.6 Icon animation set-up

Icons can be animated by applying an animated material. Three animated icon materials are included: *M_UMG_MapPulse*, *M_UMG_MapClock* or *M_UMG_MapFlashing*. To see these, you must enable **View Options: Show Engine Content** and **Show Plugin Content** in the Content Browser. The animations are shown in Figure 7. *Pulse* repeatedly animates an outward pulse from the icon's center to the edge. This is intended to mark areas or to make an actor stand out. *Clock* applies an animated clock mask. *Flashing* periodically makes the icon flash brightly.

To animate an icon based on gameplay values, an actor should call the Map Icon Component's function **Get Icon Material Instances**. This returns all active dynamic material instances (one per visible minimap). You can then update the material instances' parameters. This is demonstrated in Figure 8.

An icon's material can be changed during gameplay by calling the Map Icon Component function **Set Icon Material**. You can call the function **Reset Icon Material** at any time to reset the icon's material to its initial material.

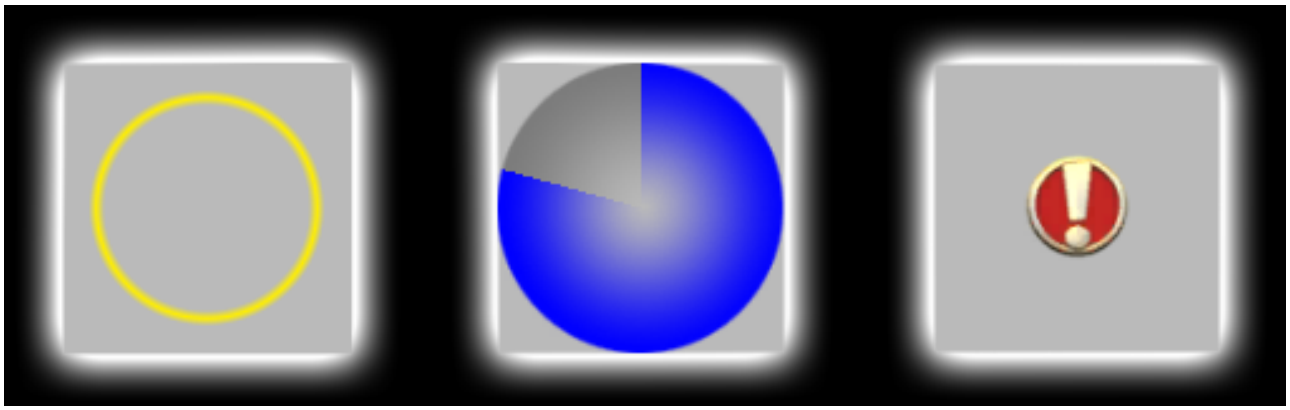


Figure 7: Three animated materials are included: pulse, clock and flashing.

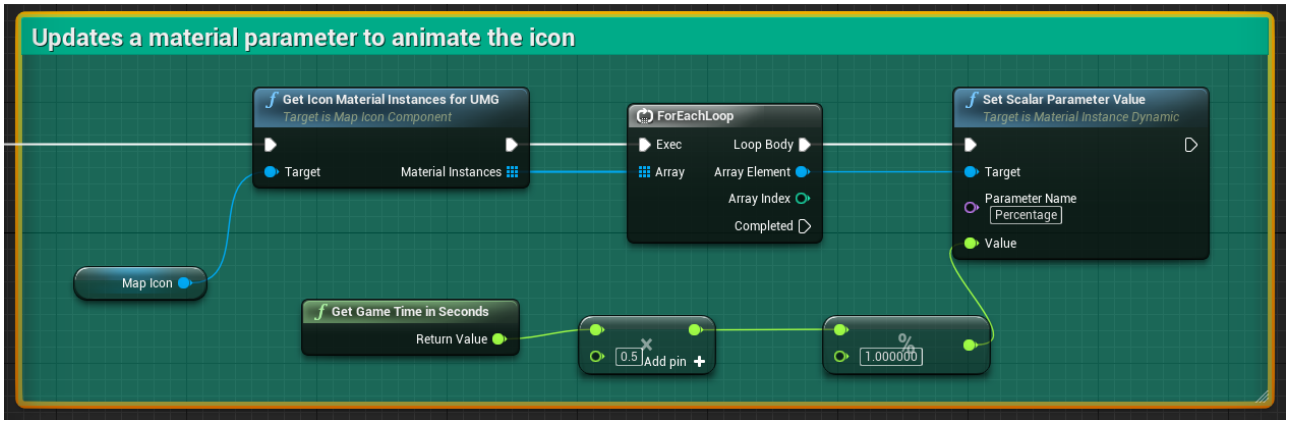


Figure 8: Actors can access their icon material instances to update animation parameters.

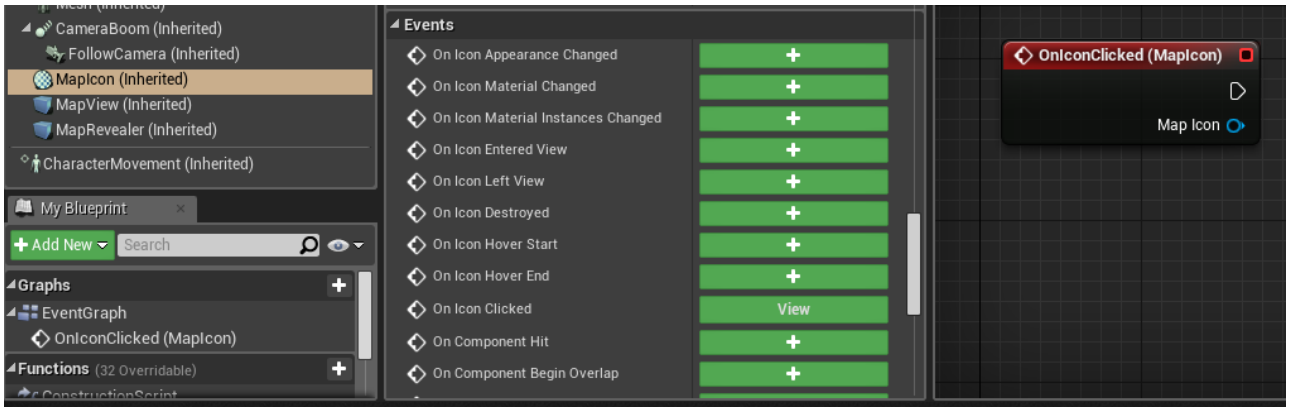
2.7 Reacting to minimap events

An actor can react to UI events that involve his icon. Map Icon Component has the following events:

- **On Icon Clicked**: Fires when the player clicks the icon.
- **On Icon Hover Start**: Fires when the player hovers over the icon with the mouse cursor.
- **On Icon Hover End**: Fires when the player stops hovering over the icon.
- **On Icon Entered View**: Fires when the icon becomes visible to a Map View Component. Takes icon position and size, minimap shape and size, and the icon's gameplay controlled visibility into account.
- **On Icon Left View**: Fires when the icon becomes hidden to a Map View Component.
- **On Icon Destroyed**: Fires when the icon component is destroyed.

You can create functions that are called when any of these events happen. Figure 9 shows how to do this in blueprint and C++. The icon must be interactable for the first three events: in the Map Icon Component, check the **Icon Interactable** option.

The Minimap and Bordered Minimap widget have an event **On Map Clicked** that fires when the map is clicked. When it fires, the event provides a world position calculated from the clicked position on the minimap and the Map View Component's Z-coordinate.



Blueprint: Select the Map Icon Component. Click on the + beside an event name.

```cpp
void AMinimapExampleCharacter::BeginPlay()
{
    Super::BeginPlay();
    MapIcon->OnIconClicked.AddDynamic(this, &AMinimapExampleCharacter::OnMyIconClicked);
}

// Be sure to mark this a UFUNCTION()
void AMinimapExampleCharacter::OnMyIconClicked(UMapIconComponent* Icon)
{
    UE_LOG(LogTemp, Display, TEXT("I was clicked!"));
}
```

C++: Bind a function to a Map Icon Component delegate.

Figure 9: Reacting to icon events in BP and C++

2.8 Fog of war set-up

If you want to add fog of war to the minimap, place a **Map Fog** actor in the level. Similar to when adding a background, use the top viewport to move, scale and rotate to make the actor's box component overlap an area. That area will be covered in fog on the minimap. The Map Fog actor has parameters to control the thickness of the fog in the minimap:

- **Minimap Opacity Hidden**: How visible areas inside fog are
- **Minimap Opacity Explored**: How visible areas are that have been revealed before
- **Minimap Opacity Revealing**: How visible areas are that are currently revealed

Showing fog in the world

The Map Fog actor's **Enable World Fog** setting controls whether the fog is visualized in the world. A post process material *M_WorldFog* is used to darken the world in places covered in fog: be aware that this effect is not physically based. The post process material must be added to a Post Process Volume. If the level already has an unbound Post Process Volume, you can assign it to the Map Fog's **Post Process Volume** setting. Otherwise, an unbound post process volume is automatically located or created when the game starts. The Map Fog actor has thickness settings for world fog, equivalent to the settings for minimap fog.

Revealing fog

Actors can reveal what is under the fog. To give an actor this ability, add a **Map Revealer Component** to it. For example, add it to your player pawn class. Map Revealer Components have a box shape. The size of this box controls the area around the actor that is revealed. The component can be configured to reveal a circular or rectangular area around the actor. The revealed area can have a soft edge, controlled by **Reveal Drop Off Distance**. The drop off distance is measured starting from the box extent.

If you want actors to reveal an area only while it is present, set its **Reveal Mode** to **Temporary**. This achieves fog of war like in top down RTS and MOBA games: when the actor leaves an area, the area becomes hidden again. If an actor should permanently reveal a visited area, set it to **Permanent**. This would result in fog like in action RPGs: the map is progressively revealed. You can set it to **Off** if you want to enable it only on actors that meet certain conditions during gameplay. For example: if only the player's allies should reveal fog, set a pawn's Reveal Mode to Off initially and change the setting during gameplay once teams are confirmed. To change the Reveal Mode during gameplay, call **Set Reveal Mode**.

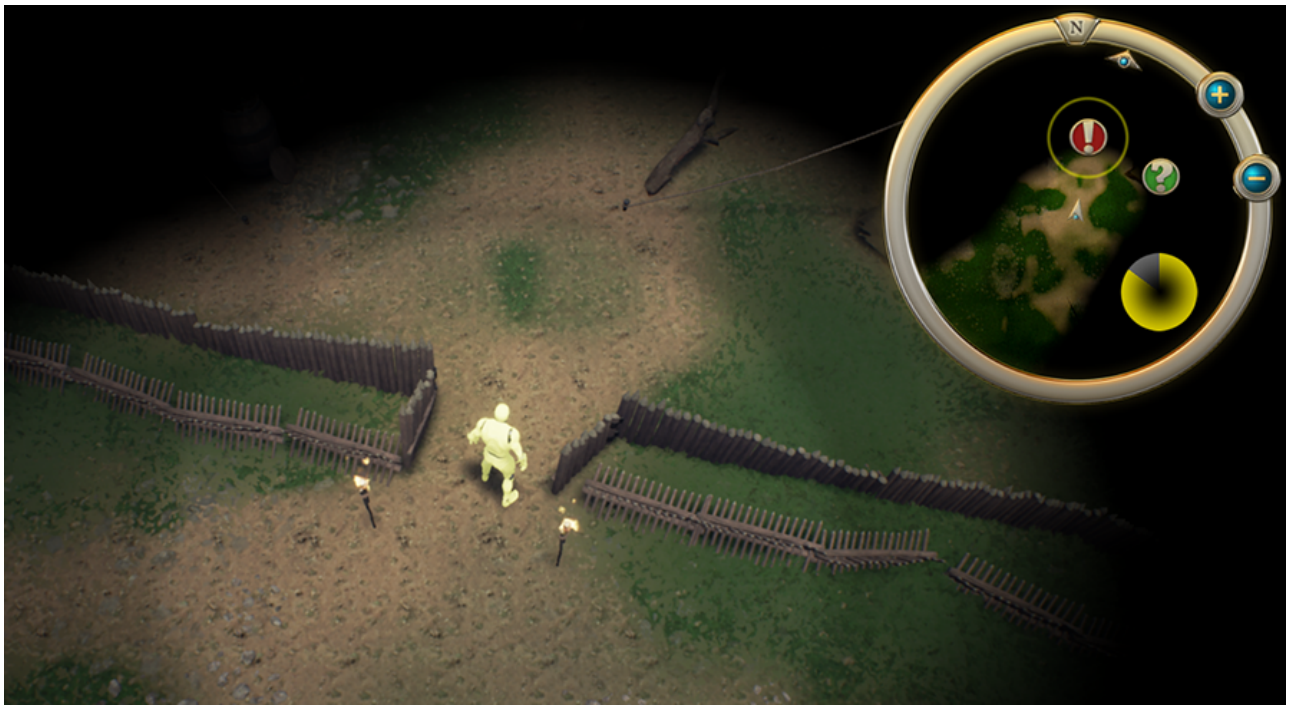


Figure 10: Add fog of war to your minimap and/or world.

Icons and actors inside fog

Map Icon Components can interact with fog in various ways. The following Map Icon Component settings control how the icon behaves when the actor is inside fog:

- **Fog Interaction**: Whether the icon is visible or invisible in fog. This setting can distinguish between locations that have been explored previously or are being revealed currently.
- **Fog Reveal Threshold**: If the icon hides in fog, how much of the fog must be revealed to make the icon appear. Recall that areas explored by Map Revealer Component have a soft edge, when the revealer's Reveal Drop Off Distance is positive. If the threshold is 0.7, the icon will appear when its world position is 70% revealed or more.
- **Hide Owner Inside Fog**: If enabled, the icon will hide its owning actor from the world while it is inside fog by calling the actor function **Set Actor Hidden In Game**.

3 Frequently asked questions

3.1 Where do I find the plugin's assets in the Content Browser?

To see the plugin's assets, you must enable **View Options: Show Engine Content** and **Show Plugin Content** in the Content Browser. The assets will then show in the folder *MinimapPlugin Content*.

3.2 Does the plugin support multiplayer?

Yes. Minimaps are simulated by the game client, so as long as actors with icons are replicated and their positions are too, the minimaps of all game clients will be synchronized. You can make icons appear differently per game client. Use this, for example, to hide icons of actors that aren't in the local player's team. Similarly with fog of war, you can achieve team based vision by enabling only the Map Revealer Components of actors of the local player's team.

3.3 Can I use the plugin with only blueprint?

Yes. The plugin's features are completely accessible to blueprint.

3.4 Can I use the plugin with only C++?

Yes. The plugin's features are completely accessible to C++.

3.5 Can I use the plugin without UMG?

Yes. The plugin includes a C++ rendering option. Add a **Map Renderer Component** to your HUD C++ class. Use the following functions to control to what part of the screen the minimap is rendered:

- **Set Horizontal Alignment**: Left, center, right or fill.
- **Set Vertical Alignment**: Top, center, bottom or fill.
- **Set Size**: Set's minimap size in pixels. An axis' value is ignored if its alignment is set to fill. For example, the Y size is ignored if the vertical alignment is set to fill.
- **Set Margin**: Distance in pixels from the screen's edge. A side's value is ignored if the alignment doesn't anchor to that side. For example, the right margin is ignored if the horizontal alignment is left.

Within your HUD C++ class' Draw HUD function, call the Map Renderer Component's **Draw to Canvas** function passing the HUD's **Canvas** to render the map. Pass mouse click positions to the Map Renderer Component's **Handle Click** function to support clicking of icons and the minimap background. To react to the background being clicked, bind a function to the Map Renderer Component's **On Map Clicked** delegate.