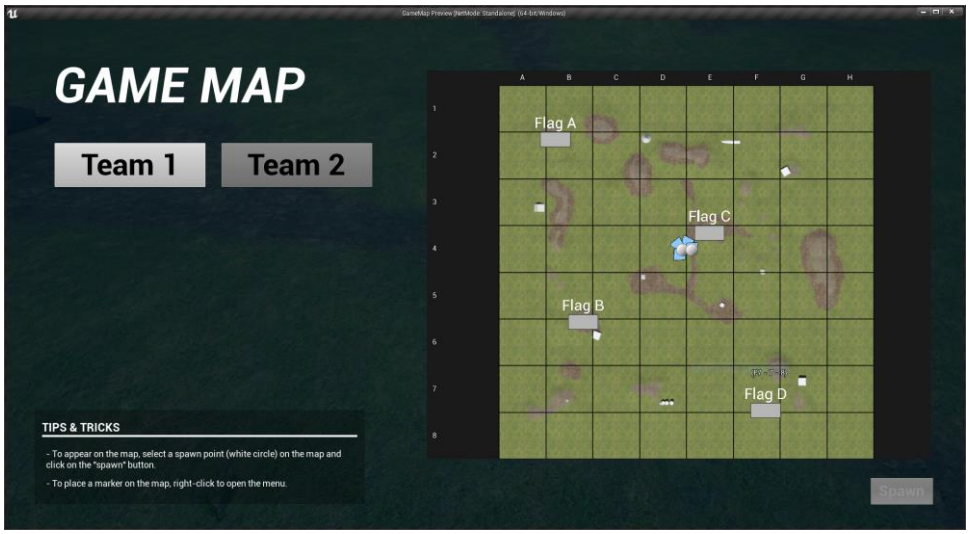
Game Map


GAME MAP
Team 1
Team 2
A B C D E F G H
1 2 3 4 5 6 7 8
Flag A
Flag C
Flag B
Flag D
TIPS & TRICKS
- To appear on the map, select a spawn point (white circle) on the map and click on the "spawn" button.
- To place a marker on the map, right-click to open the menu.
Spawn

Documentation

This document contains all the information needed to use our map in a project based on the Unreal Engine 4. For any questions, we invite you to join our discord at the following address: https://devkit.fr

Presentation

This package provides tools to display an interactive map in your game. To work, this map uses an image file as a background on which are added the icons of the actors present in the scene.

In addition to the map tools, the package comes with additional elements used in a demo environment. These elements will be used to understand how to implement your own indicators and are not intended to be kept in the final version of your game.

Features

The package provides tools for displaying a complete map as well as examples of uses. The main features of the map are:

- Advanced configuration allowing the creation of different maps such as:
    - Map from which the player can choose where to appear
    - Mini-map displayed in the HUD giving real time information
    - Map summarizing the game world with indications added by the players.
- Full compatibility to work with networked games.
- Display of flag icons that can change according to the gameplay rules of the game.
- The player can choose where to start the game by selecting the reappearance point of his choice on the map.
- The information display on the map has been designed to distinguish between two teams of players.
- A system for pointing at enemies makes them appear to allies for a few seconds on the map.
- Display grid dividing the map into zones allowing the exchange of coordinates between players.
- Menu to add a marker on the map visible on the map of the other players of the team.

Package content

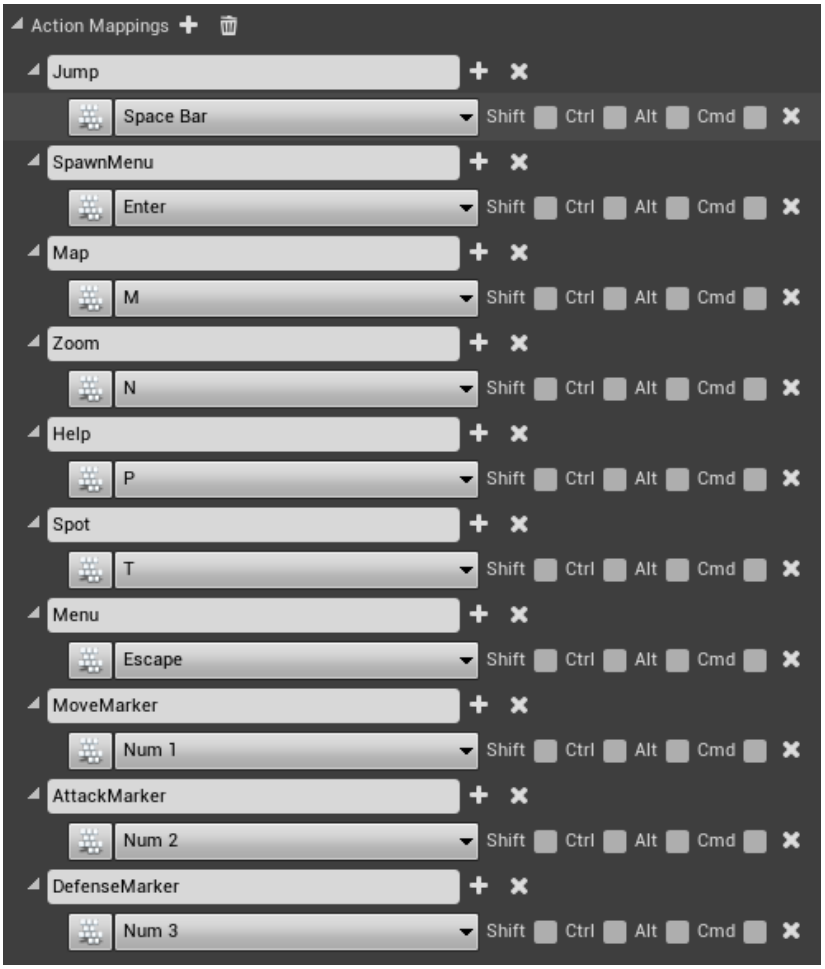
The package consists of six folders containing the following items:

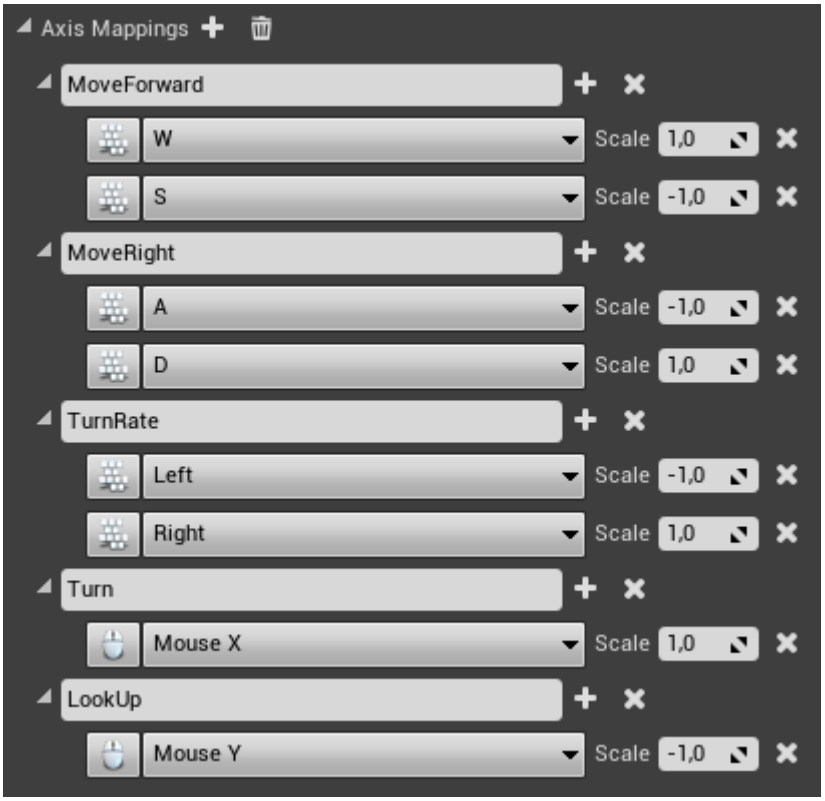
- **Assets**: all visual content related to the demo environment.
- **Blueprints**: the character actors, *GameMode* blueprints, spawn points and flags used for the demo scenes.
- **Capture**: the tools for taking screenshots of the map in orbital view.
- **Levels**: the level files provided for the demo environment.
- **Map**: all the important components used to display the maps.
- **Widgets**: the widgets for the purposes of the demonstration.

To facilitate the integration of the map in your project, only the "Map" folder is necessary, the other folders being intended to contain the elements of the demonstration.

Input parameters

Below is the list of configured inputs for the keyboard and mouse of the project. This information is given as an indication and is not useful if you use only the essential components of the board.


Action Mappings
Jump
Space Bar
Shift Ctrl Alt Cmd
SpawnMenu
Enter
Map
M
Zoom
N
Help
P
Spot
T
Menu
Escape
MoveMarker
Num 1
AttackMarker
Num 2
DefenseMarker
Num 3


Axis Mappings
MoveForward
W
Scale 1,0
S
Scale -1,0
MoveRight
A
Scale -1,0
D
Scale 1,0
TurnRate
Left
Scale -1,0
Right
Scale 1,0
Turn
Mouse X
Scale 1,0
LookUp
Mouse Y
Scale -1,0

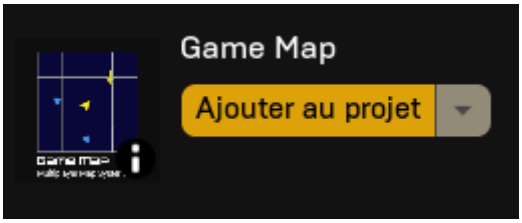
Integration of the package in your project

This first step will explain how to import the package into your project. At the end of this process, you will have the necessary tools to start displaying your maps.

Import from Epic Games

In the Epic Games launcher, go to the Unreal Engine library under "Vault" and enter "Game Map" in the search field.

You will see the package appear in your list. Click on the "Add to project" button, select the project you are going to work in and confirm the selection.

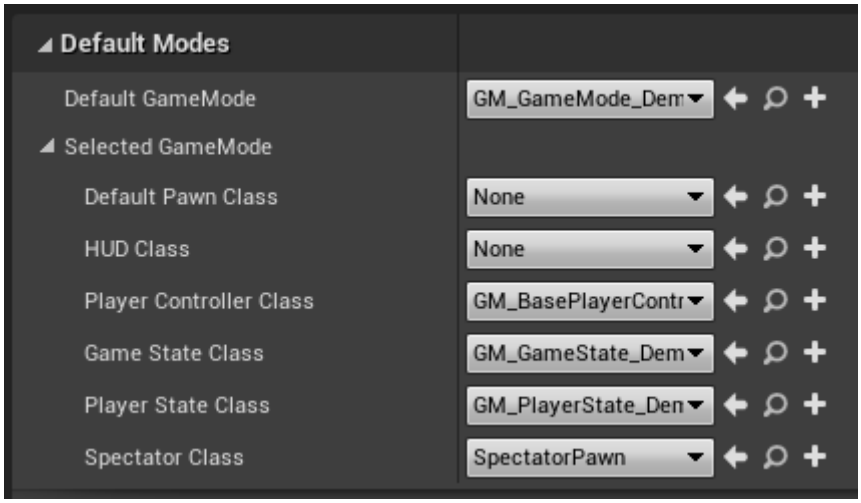


After a few seconds of waiting, you will see the "GameMap" folder appear in your Unreal Engine project.

Configuration to make the demo scenes functional

To be able to use the demo scenes, a configuration of the project is necessary. Note that this modification is not useful if you want to use the map in your project.

To modify the project, open "Edit > Project Settings > Maps & Modes" and fill in the information as below:



At the bottom of the form, you must also define the Game Instance as below:

Removal of demonstration items

If you do not want to keep the demo items with the usage examples, open the "GameMap" folder and delete the following folders:

- Assets
- Blueprints
- Levels
- Widgets

Also delete the contents of the following two folders in which we will later add the items related to your game:

- Capture/RenderTarget
- Capture/Textures

You are now ready to start integrating the card into your game, which we will detail without further ado.

Generate the satellite image of a scene

To start, we will generate a "satellite" image of a scene in your game. This image will be the one displayed by the map and on which the gameplay icons will be positioned.

Note that the following steps will have to be repeated for all the scenes that will require the display of a map.

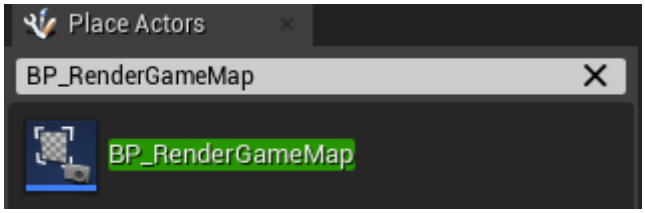
Use of an "orbital" camera

To generate an "orbital" image of your scene, we will use a special camera that we will place in your scene and that we will have to set up.

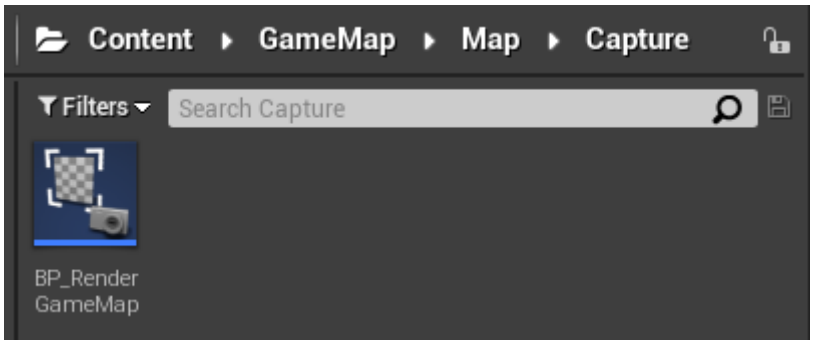
To start, open the scene for which you want to generate a map image.

1. Add the camera in the scene

Once loaded, look for a camera named *BP_RenderGameMap* in the list of actors and place the object in the scene.

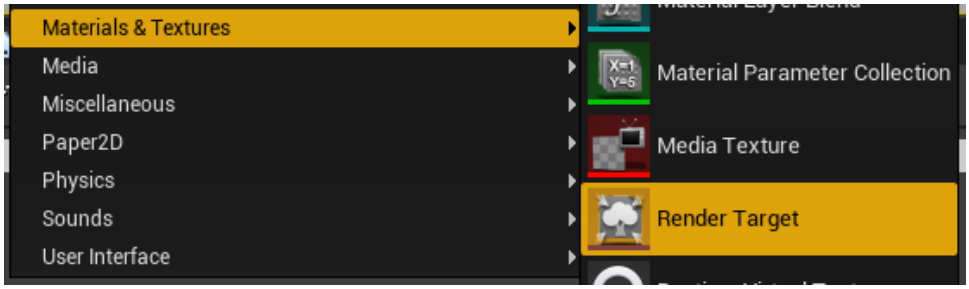


If the blueprint does not appear in the list, you can add it to your scene from the content browser in the *GameMap/Map/Capture* folder.



2. Creating a "RenderTarget" texture

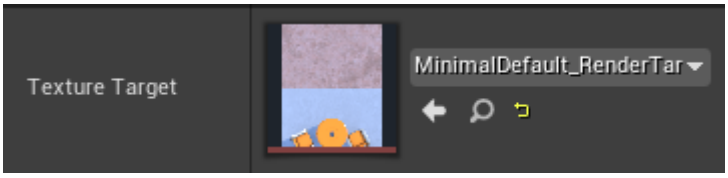
Go to the *GameMap/Capture/RenderTarget* folder, right click and create a new texture of type Render Target.



By convention, we will name this texture like this: *SceneName_RenderTarget*

3. Associate the texture with the camera

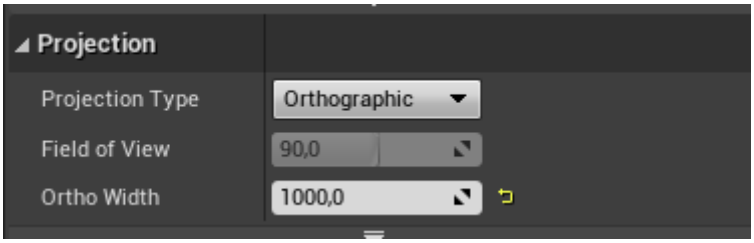
Now go back to the camera details under *Scene Capture* and configure the newly created texture in *Texture Target*.



As you can see, the texture displays in real time what the camera sees. It is thanks to this image that we will be able to position the camera.

4. Configure the camera resolution

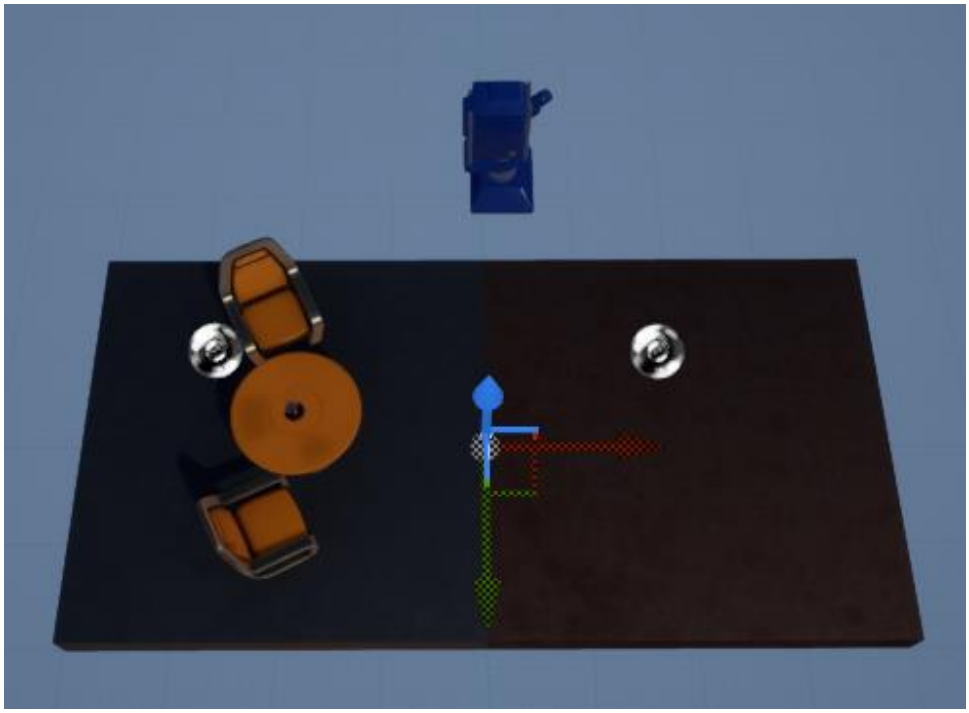
The setting of the visible area of the camera should be done with the *OrthoWith* parameter in the *Projection* section.



As indicated in the tooltip, the unit of measurement of the value is in "world unit". This same unit is used for the position of the actors in the scene which will help us to understand how to define its value efficiently.

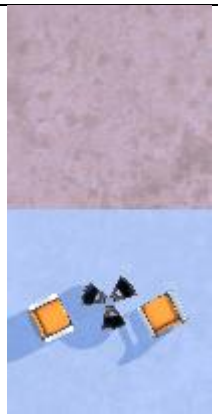
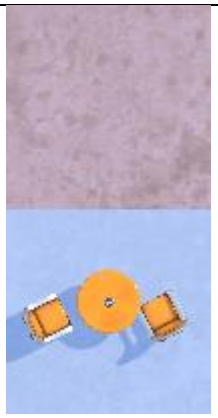
The image below shows a scene where the camera is in the center. The chairs are at about -200 wu on the X axis. In order for them to be fully visible in the camera, you will need to set the *Ortho Width* parameter value to at least "1000". The value entered is to be divided by two, so the camera displays the elements on each side up to 500 wu.

To find the value of Ortho Width in meters, simply divide the value by 100. Thus, when Ortho Width is 1000, the camera captures an image of ten meters by ten.



Note that the position of the orbital camera (on the X and Y axis) does not have to be in the center of the scene. You can place the camera wherever you want, as in some situations such as a game mode that would only require a small area of the game level.

The last parameter concerning the positioning of the camera is the height. It is not necessary to set a large value for the height, but if objects are higher than the camera, they will not be displayed in the generated image.

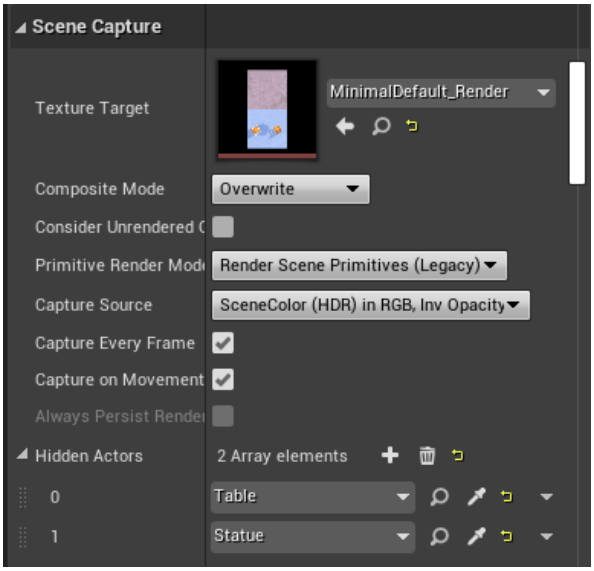
	
Z = 100	**Z = 250**

In the example above, a camera placed with a Z value of 100 will not display the table and chairs correctly. Whereas with a Z value of 250 all the elements of the scene appear correctly.

5. Exclude actors from the image

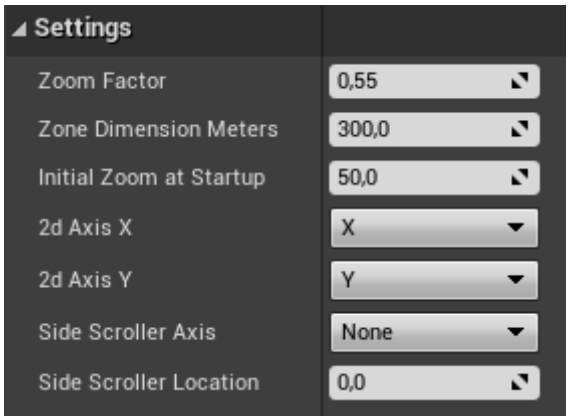
You may need to exclude elements from the scene so that they are not visible in the final image generated.

To configure these exclusions, go to the camera details under Scene Capture and you will find a list of actors to hide called Hidden Actors. Simply add the elements of the scene not to be displayed at the time of rendering like the table in the image below.



6. Configuring level options

In the details of *BP_RenderGameMap*, you can find in the *Settings* section the parameters specific to the scene which allow you to modify the display of the map.

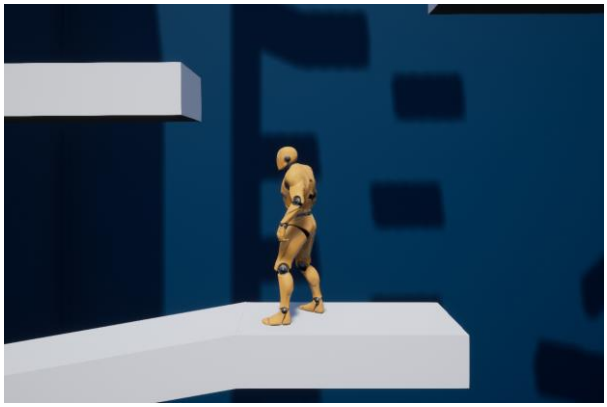


- **Zoom Factor:** this value, between 0 and 1, allows you to choose how strongly the zoom is applied to the map. The closer the value is to 1, the "weaker" the zoom force will be.
- **Zone Dimension Meters:** this parameter defines the size in meters of a grid cell.
- **Initial Zoom at Startup:** this value expressed in meters is used to display the map already zoomed when the widget is first loaded if the option is activated.

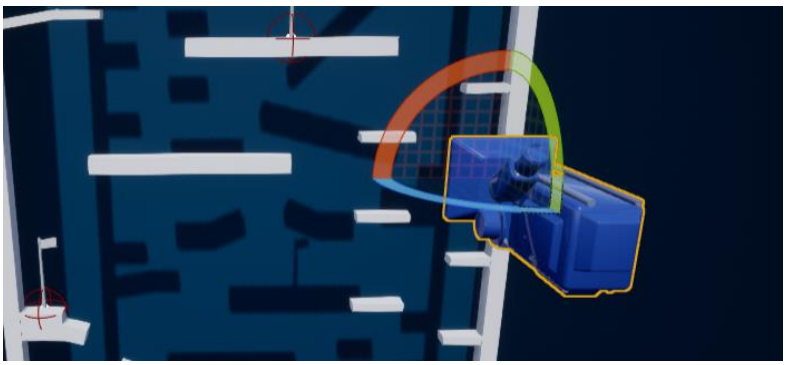
The last two parameters are only to be used in a "Side Scroller" type game and will be detailed in the dedicated chapter.

Side-scrolling video game

The card can be used with side scrolling games and requires only a few changes in the camera configuration to work.

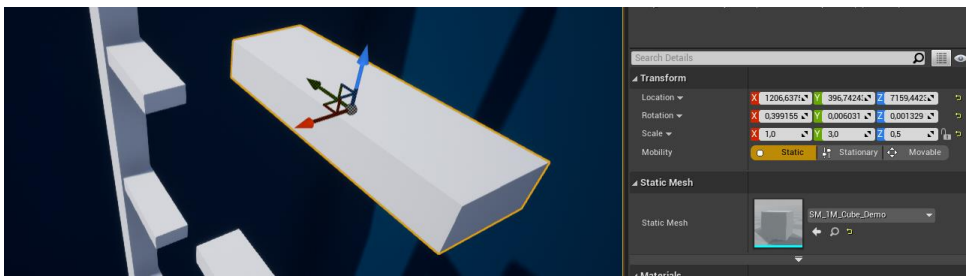


In addition to the camera rotation, you will have to define in the *BP_RenderGameMap* settings options in the "*Settings*" section the following property values:



Thanks to the rotation angle used for the camera, you can find the values to set in the *BP_RenderGameMap* settings options:

- **2d Axis X**: to change the axis of the 3D environment to be used in abscissa on the 2D map when displaying the actors.
- **2d Axis Y**: to change the axis of the 3D environment to be used in ordinate on the 2D map when displaying the actors.
- **Side Scroller Axis**: to define the scrolling axis of the scene.
- **Side Scroller Location**: position of the scroll axis used to place the markers.



In the image above, we can see from the orientation of the cube and its position the following values for the parameters:

- **2d Axis X**: Z
- **2d Axis Y**: -Y
- **Side Scroller Axis**: X
- **Side Scroller Location**: 1206,637085

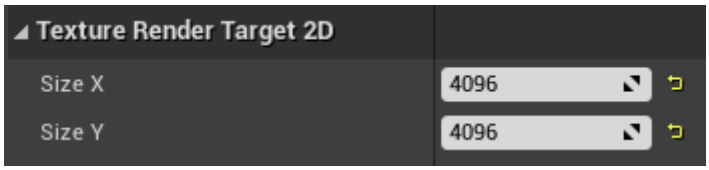
If you need more examples to find the values to set for your game, you can look at the levels in the *GameMap/Levels/Tests* folder to find different scene configurations and orientations.

Export the image generated by the camera

With the camera configured, we will export the image it captures. The exported file can be edited with external software to modify the rendering before reimporting the image into Unreal.

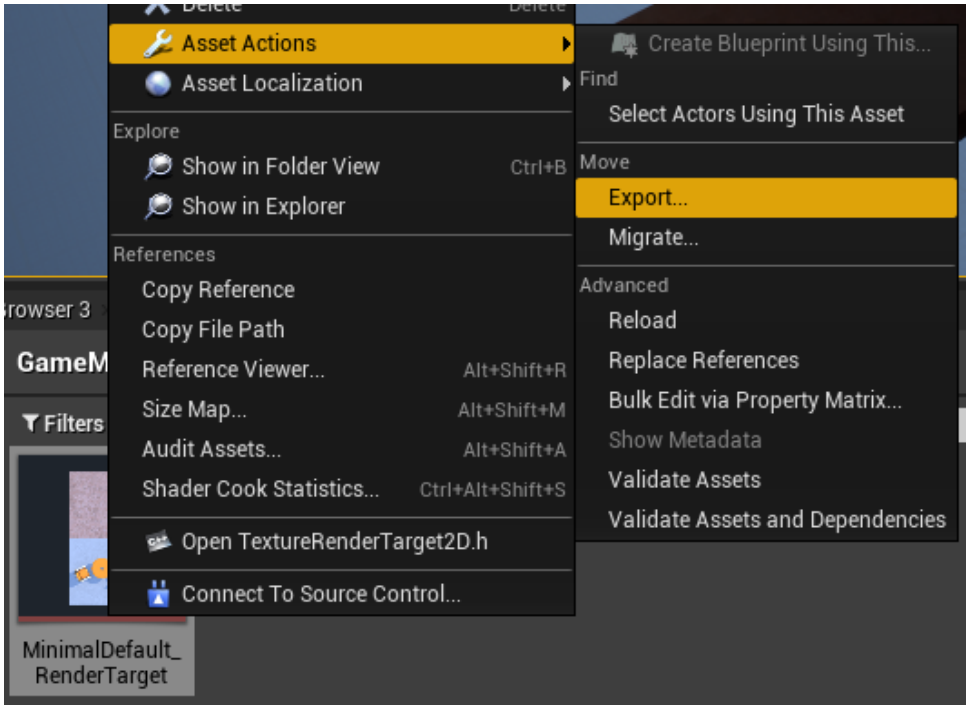
Before exporting the view taken by the camera, we will see how to change the resolution of the file that will be generated. To do this, double click on the Render Target created earlier to open its properties.

In the detail tab, increase the resolution to 4096x4096 like this:



Once you have made the change, click on the *Save* button and then close the tab.

Now create an image file by right-clicking on it and choosing the *Export* option as shown in the image below.



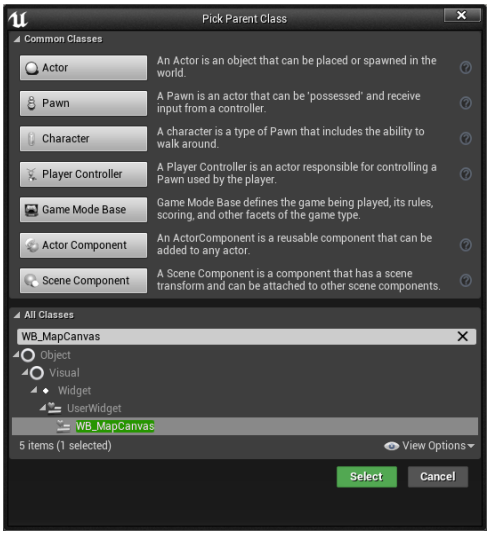
The generated file can then be opened with photo editing software such as *GIMP* or *Photoshop* to modify the rendering according to your wishes.

For the rest of this explanation, we will simply open the generated file (HDR type) and save it in *TGA* format in order to re-import the new file into *GameMap/Capture/Textures*. Once imported into Unreal, remember to rename it like this: *SceneName_Texture*.

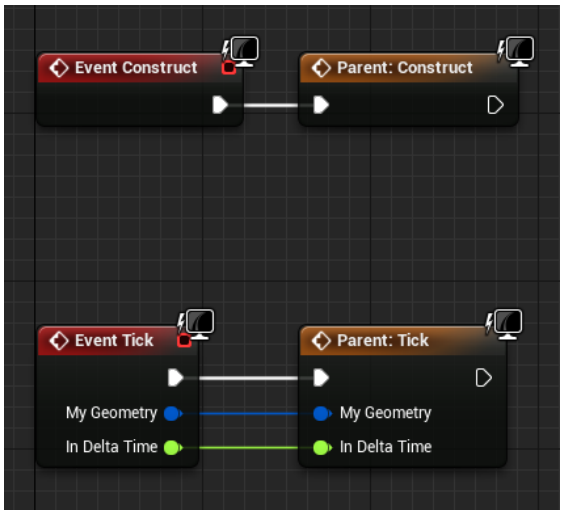
Repeat the operation described above for all scenes in which the map can be displayed.

Creation of the map canvas

The first step to integrate the map in your game will be to create a widget that we will name *WB_MapCanvas_Game*. This widget will inherit from *WB_MapCanvas* and will contain all the display rules specific to your game.

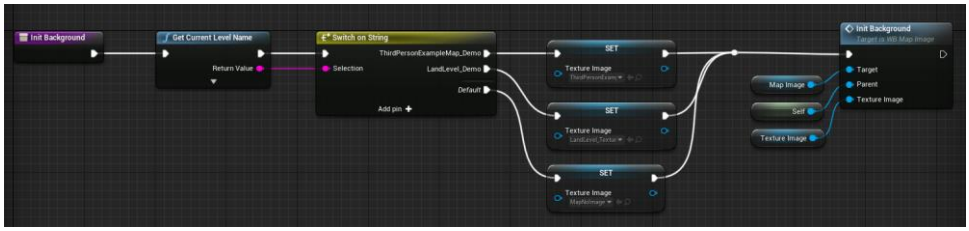


Once created, open this widget and go to EventGraph to activate the Construct and Tick events as in the image below. Without these two events enabled, the map will not work properly.



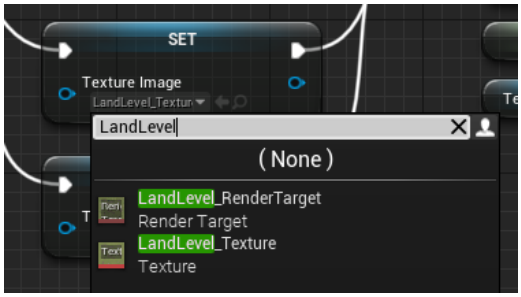
Map background

To begin, we will create a private function named *InitBackground* which will load the background image according to the scene name.



As shown in the code above, we initialize the local variable *TextureImage* following the name of the scene. Note that the variable *MapImage* is a variable inherited from the parent class.

Attention: when you initialize the value of *TextureImage*, make sure you take the *Texture* and not the *RenderTarget*.

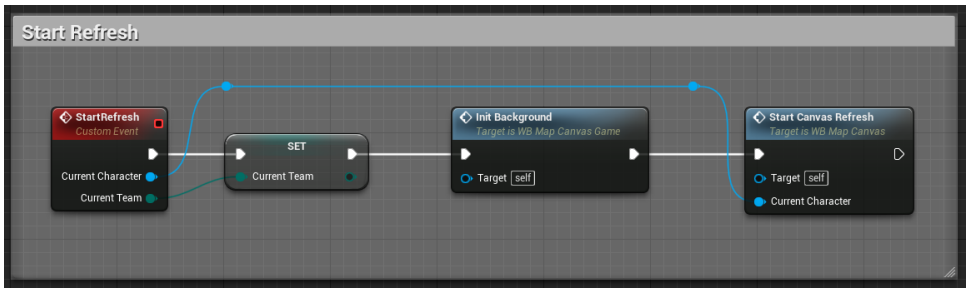


You must repeat all the steps described above for each scene in your game that requires a card to be displayed.

Events for display refreshment

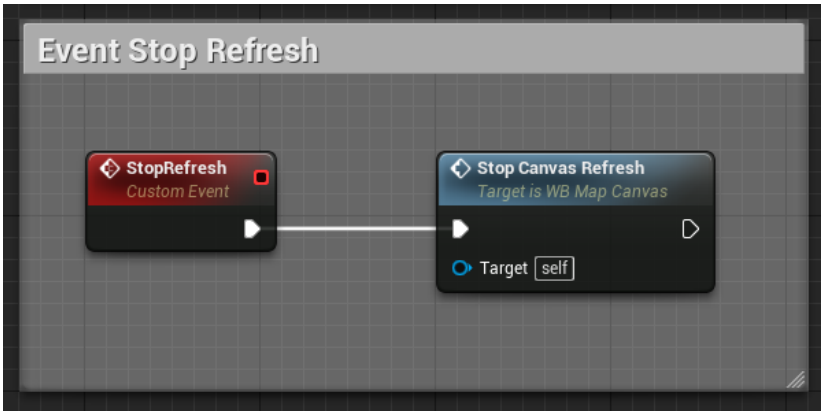
We are going to create two events that will be called when the map should be displayed or hidden. These two events will call functions defined in the parent. Finally, for the *StartRefresh* function, the *InitBackground* function will be executed to load the background image.

Event to activate the map refresh



This function allows to activate the refreshment of the information displayed on the map. The Current Character variable is mandatory and necessary for the proper functioning of the canvas. On the other hand, the Current Team variable is specific to the gameplay of your game and is not necessarily mandatory.

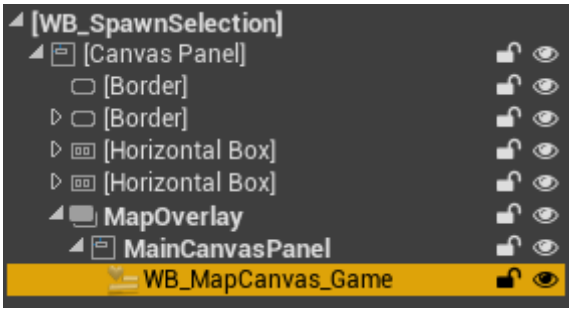
Event to stop the canvas refresh



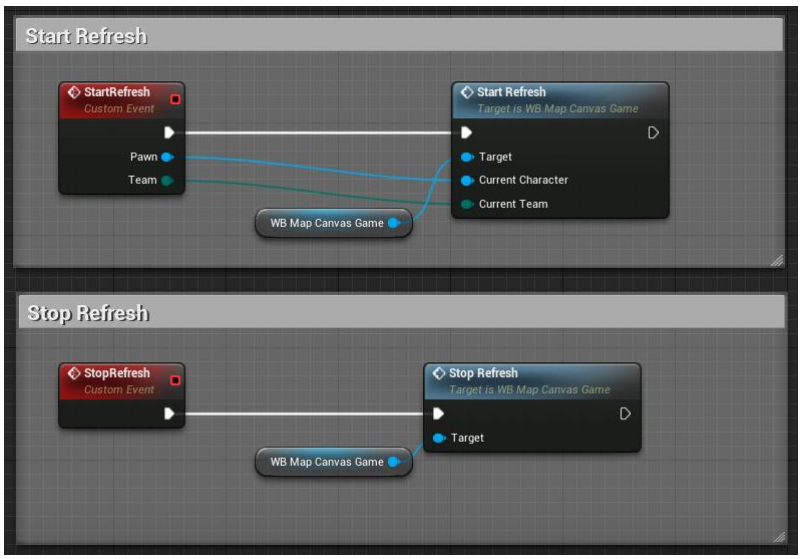
To stop the refresh, you must call the Stop Canvas Refresh function.

Add WB_MapCanvas_Game in an interface

Now that we have programmed the basic elements of the map, you can add the widget to an already existing interface of your game like the splash screen.



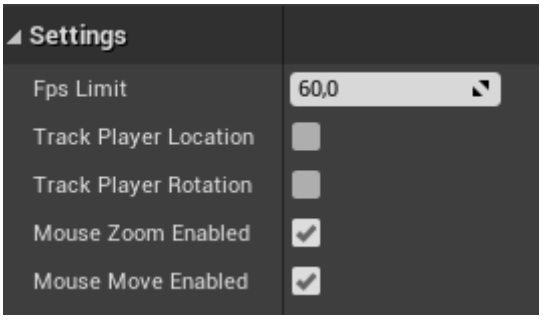
In your appearance screen, you will have to make changes to call two events when the widget is displayed and closed.



Setting up the map

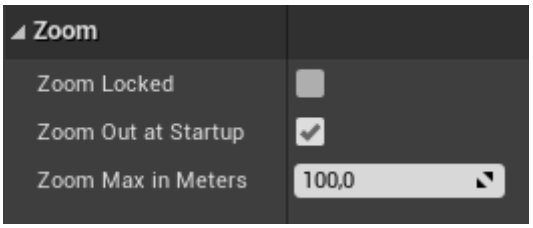
The last step for this integration will be to define the parameters according to your needs. There are several configuration blocks in the details part of *WB_MapCanvas_Game* and we will only see the usable elements now. The other parameters will be explained in due time.

General settings



- **Fps Limit:** limit the number of refreshes per second. If this value is zero, the FPS limit is disabled.
- **Track Player Location:** when the value is *true*, the player position will be used as the center of the map.
- **Track Player Rotation:** when the value is *true*, the rotation of the player's camera will be used to orient the map.
- **Mouse Zoom Enabled:** allows the player to zoom with the mouse wheel.
- **Mouse Move Enabled:** allows the player to move around the map by clicking and dragging.

Zoom settings

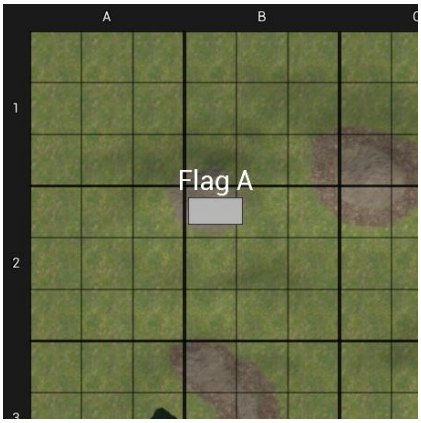


- **Zoom Locked:** when the value is *true*, the map zoom is disabled.
- **Zoom Out at Startup**: when the value is true, the "Initial Zoom at Startup" parameter of BP_RenderGameMap is used to automatically zoom the map at startup.
- **Zoom Max in Meters:** define the maximum zoom allowed with this map. The value is expressed in meters.

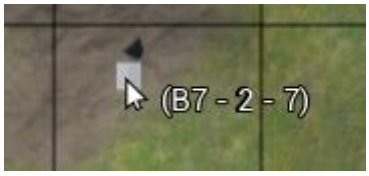
When you start your game, you will see that your map is displayed and that you can (if you have allowed it) zoom in and out with your mouse.

Coordinate grid

The map has an option to display a grid that divides the game space into zones, thus facilitating communication between players. In addition, another option is available to simplify the reading of coordinates, displaying next to the mouse an information bubble giving the exact location pointed.



Grid dividing the map into zones

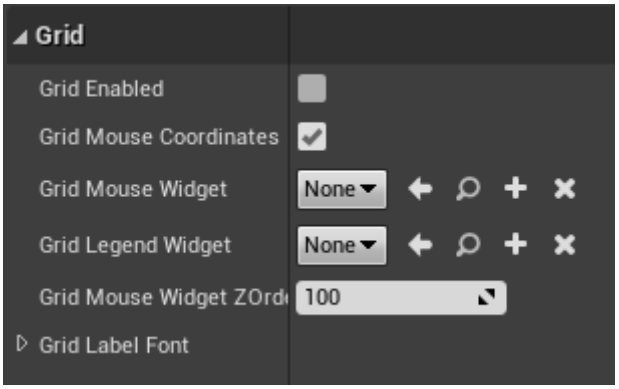


Coordinates tooltip

Grid configuration

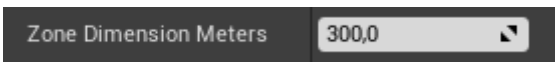
The configuration of the grid can be done from the *WB_MapCanvas_Game* widget for the global parameters and from *BP_RenderGameMap* for the scene specific parameters.

Grid parameters in WB_MapCanvas_Game



- **Grid Enabled:** when the value is true, the grid will be displayed on the map.
- **Grid Mouse Coordinates:** when the value is true, the coordinates tooltip is displayed.
- **Grid Mouse Widget:** widget class to use to customize the display of the coordinate's tooltip.
- **Grid Legend Widget**: class widget to be defined to display a legend on the map.
- **Grid Mouse Widget Z Order**: display height of the coordinate's tooltip.
- **Grid Label Font:** display parameters of the coordinate's texts present at the edge of the map.

Grid parameters in BP_RenderGameMap



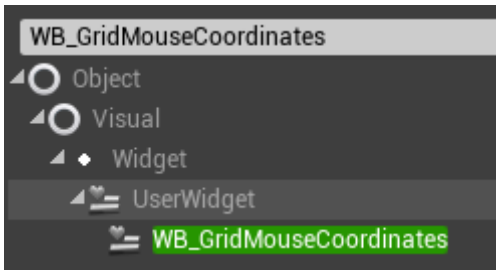
- **Zone Dimension Meters:** this parameter defines the size in meters of a grid cell.

The three displayed grids can have their style set thanks to the three variables named *Grid1Style*, *Grid2Style* and *Grid3Style*.

- **Enable**: activates the grid display.
- **Color**: options to set the color.
- **Thickness**: line thickness of the grid.
- **Gradual Opacity:** if activated, the display of the grid will be progressive.
- **Display Begin:** percentage of zoom from which to display the grid.
- **Display End:** zoom percentage from which the grid should be completely displayed (if the "*GradualOpacity*" parameter is set to *true*).

Customize the coordinates tooltip

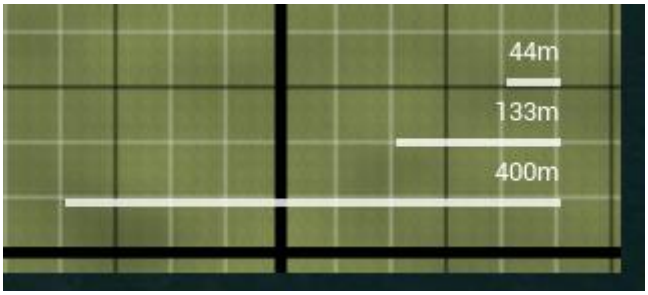
To customize the display of coordinates on the map, we need to create a new widget that we will call *WB_GridMouseCoordinates_Game* and that will inherit from *WB_GridMouseCoordinates*.



Once the widget is created you need to override the *UpdateText* event which is called when the coordinate text needs to be changed. To see an example of code for this event, please see the one defined in *GameMap/Map/Widgets/WB_GridMouseCoordinates*.

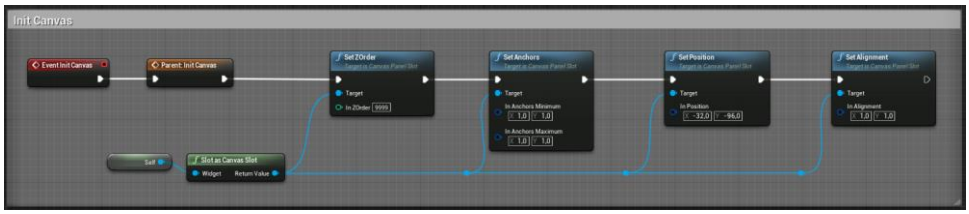
Display a legend on the map

A widget is provided to allow the display of the distances of the map. To customize the display, you have to create your own widget inheriting from WB_MapLegend.

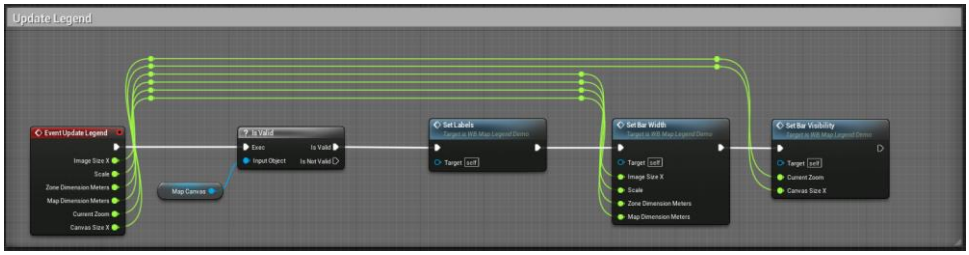


In this new widget, two events are to be overloaded:

- **InitCanvas**: which will allow you to define the position of the legend on the map.



- **UpdateLegend**: which contains all the useful information to display the distances in meters according to the zoom power.



See the *GameMap/Widgets/Map/WB_MapLegend_Demo* widget for full details on the various calculations.

Finally, you need to define the class of the newly created widget in the grid settings options of *WB_MapCanvas_Game* :

Show actors on the map

The map can display mobile or static actors present in the scene.

The addition of these actors on the map requires the creation of one or more widgets in which the gameplay display rules will be defined.

In the rest of this chapter, we will see all the steps to display some of the actors present in the scene. To see the detailed implementation of an actor (flags, player, teammates, opponents, reappearance point ...), we invite you to consult the elements available in the demo code.

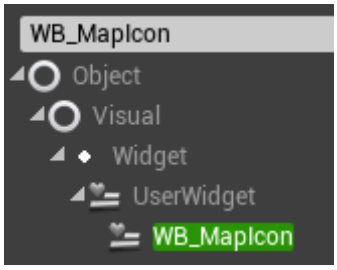
Types of icons

To limit the number of widgets needed to display, we will create an enumeration to distinguish between several actors using the same widget. The list below is extracted from the demo code available in *GameMap/Blueprints/Enums/E_MapIconTypes*.

Note that you may not follow this approach and prefer to create as many widgets as there are different actors to display. In the demo code, we have a generic widget (named *WB_MapIconGeneric_Demo*) that takes care of displaying all the actors except the flags which have a specific widget named *WB_MapIconFlag_Demo*.

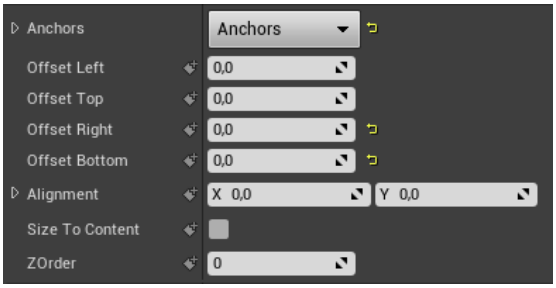
Creating an icon widget

To start, we will create a new widget named *WB_MapIconGeneric_Game* which inherits from *WB_MapIcon*.



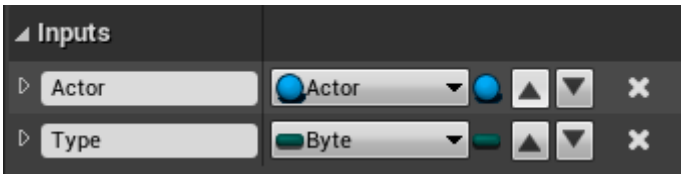
UI components

In this widget, start by adding a *Canvas Panel* and an image inside it by defining its anchor settings like this:



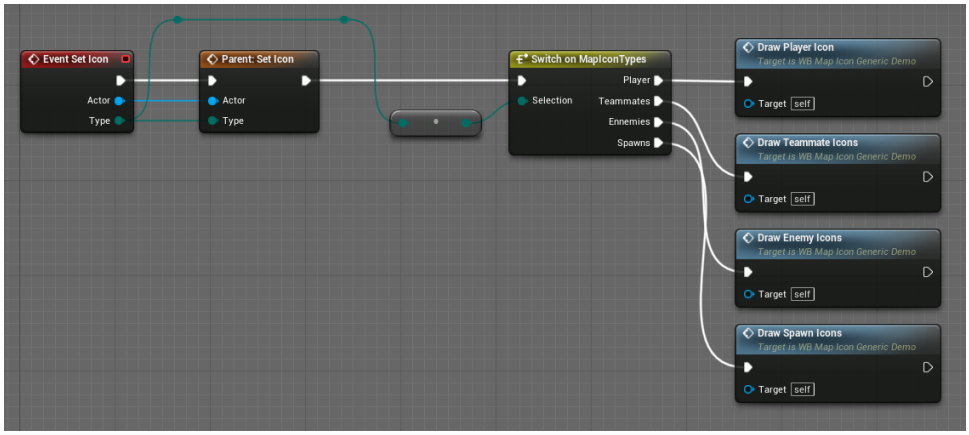
Initialize the image

In the *EventGraph*, we will override the *SetIcon* event which will be used to define the image to be displayed. This event has two parameters in input:

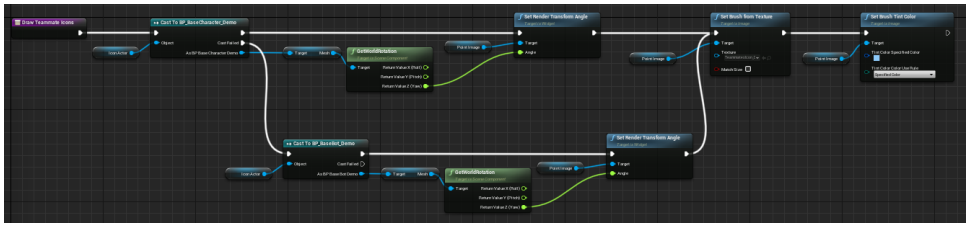


- **Actor:** corresponds to the actor associated with the widget.
- **Type:** can be converted to the enumeration created previously.

As shown in the image below, after calling the parent event, we use the Type variable to call the function needed to display the teammates icon.



The content of the *DrawTeammateIcons* function for displaying teammates can also be accessed from *GameMap/Widgets/Map/WB_MapIconGeneric_Demo*.



The code displayed above takes care of:

1. Retrieve the orientation of the actor to correctly orient the icon.
2. Define the texture of the image to use.
3. Define the color of the icon.

For more information on the other functions, we invite you to consult the demo code of this widget.

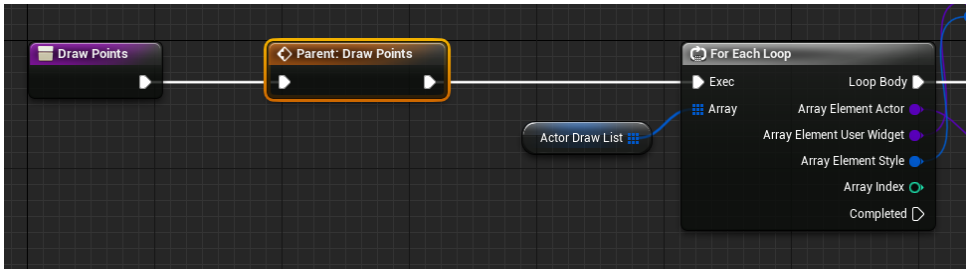
Change the display scale

We will not use in this chapter the *SetScale* event which can be used to resize internal elements of the icon such as texts. To understand its use, see the *GameMap/Widgets/Map/WB_MapIconFlag_Demo* widget.

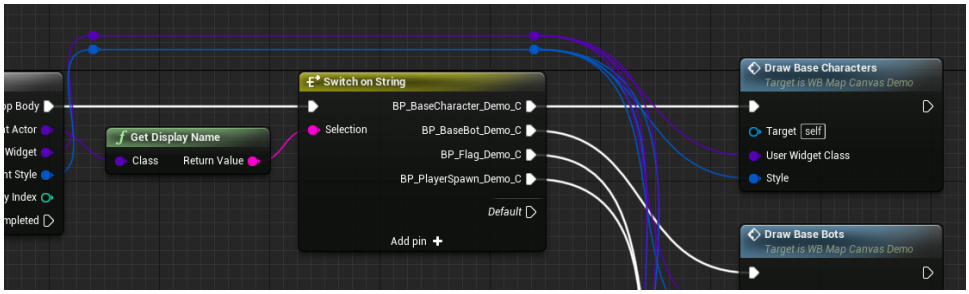
Implementation in WP_MapCanvas_Game

To finish in the implementation of the icons display on the map, we will see how to override the *DrawPoints* function in *WP_MapCanvas_Game*.

To start, we will call the parent function and then browse the list of actors to draw on the map.



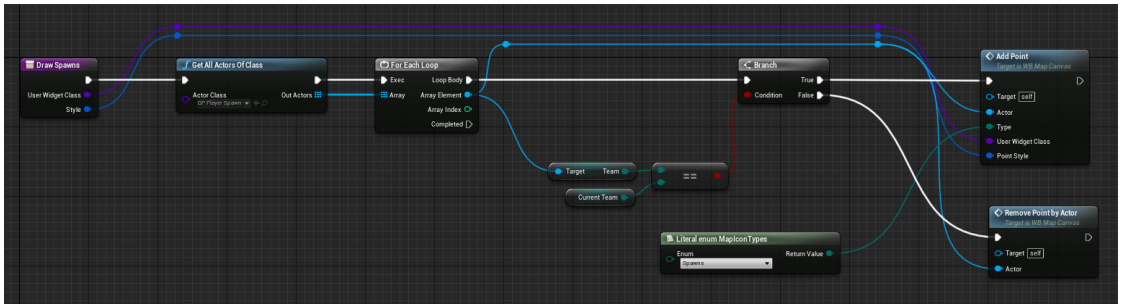
In the loop going through this list, we will make a *switch* and call a function for each actor to be drawn, taking care to pass the parameters to the function.



As in the picture above, you must add "_C" at the end of the names of your *blueprints* so that the *switch* redirects to the right place.

The input parameters to define for your functions are the following:

The function below is extracted from the demo code and takes care of displaying the appearance points whose defined team is the same as the player.

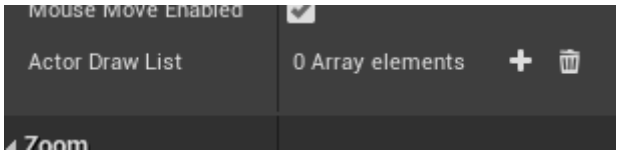


The *AddPoint* function will allow the icon to be displayed by handling the caching of the widget and the positioning of the icon on the map. The *Actor* parameter is important to define the link with the widget, the *User Widget Class* and Point Style parameters are also necessary to allow the creation of the icon. Finally, the Type parameter must be defined only if, as in our example, the *WB_MapIconGeneric_Game* widget uses this information for display.

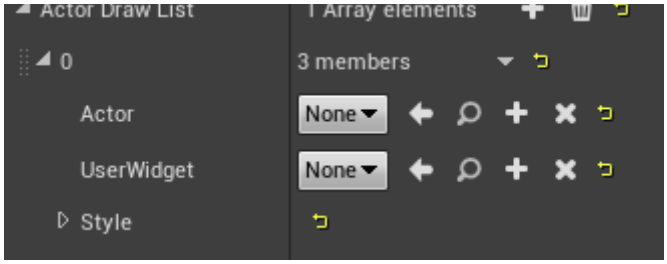
The *RemovePointByActor* function must be called in case the actor should not be visible on the map. This function will hide the icon or do nothing depending on its situation.

WB_MapCanvas_Game settings

Once the various widgets useful for displaying the icons are ready, we can go to the details tab of the widget that contains the map under Settings. You will notice the presence of the variable Actor Draw List.



This table is used to add the actors of the scene that will be displayed by associating them with the widget that will render the icon on the map.



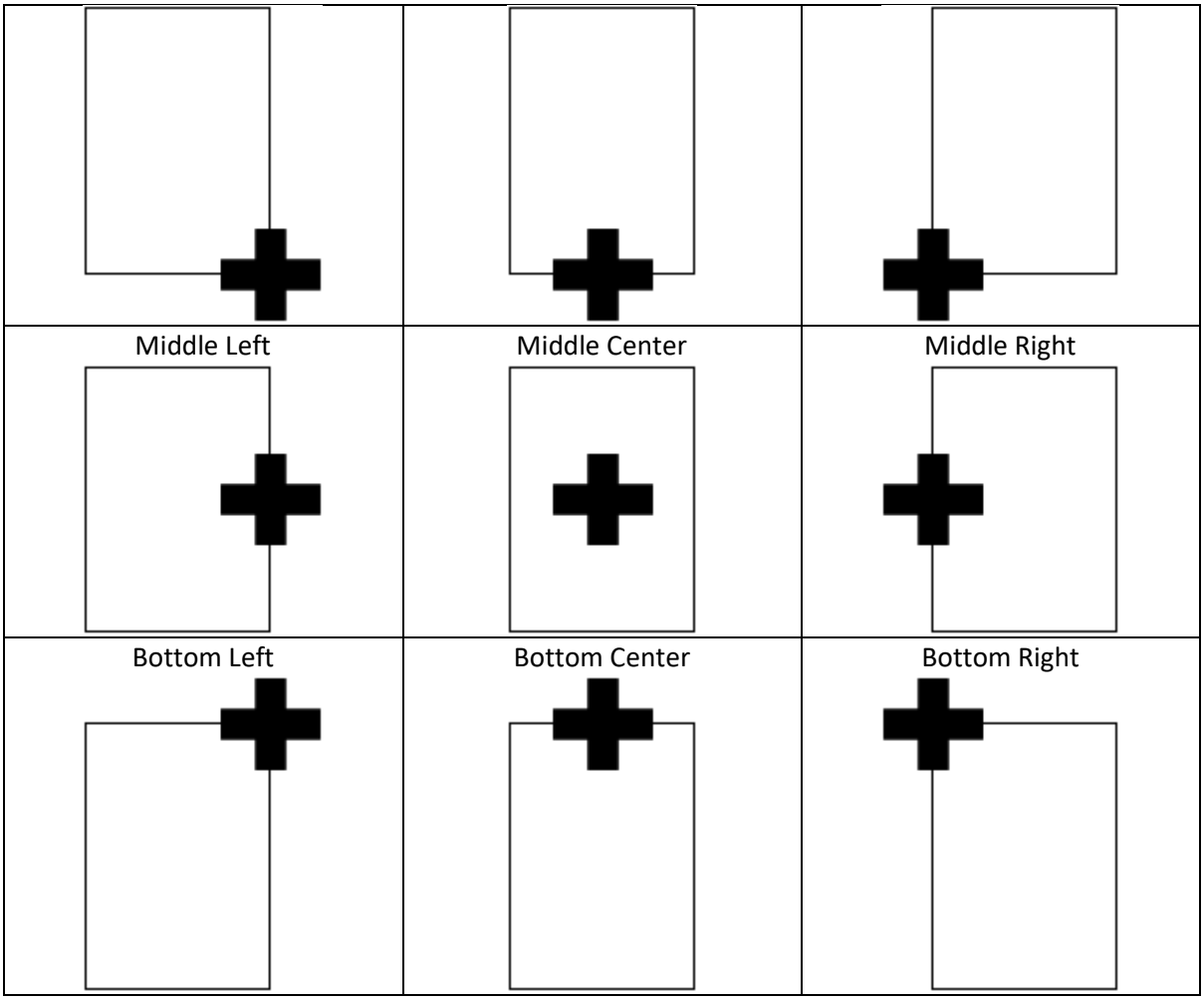
- **Actor:** class of actors of the scene that will be selected to be displayed.
- **Widget:** widget class to be used to display the actors on the map.

The *Style* variable is used to define constants to customize the display of the icon on the map.

- **Size:** the desired display size of the icon on the map (for a 1:1 size).

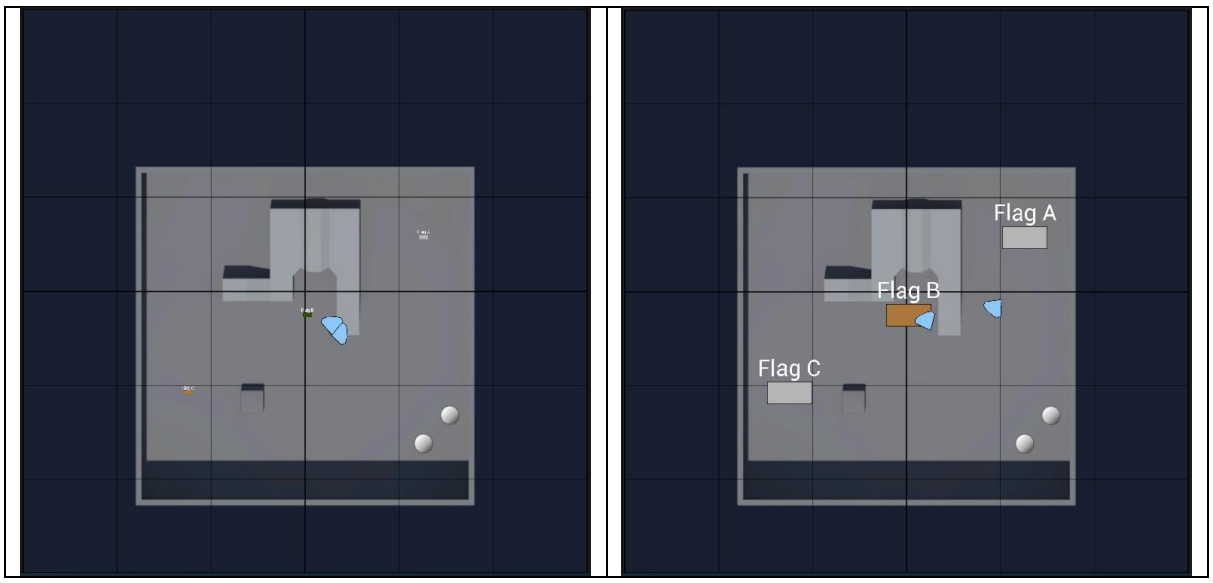
- **Position:** there are 9 possible values to place the icon in relation to the actor's position in the 3D environment.

Top Left	Top Center	Top Right



- **Opacity:** the transparency of the icon on the map with a value of 0 for totally transparent and 1 totally opaque.

- **Auto Scale:** activates the automatic resizing of the icons according to the zoom defined for the map display.



When the value is *false*, the flags have the size defined in the *Size* parameter.	When the value is *true*, the flags have a size that adapts to the zoom.

- **Z Order:** the display height allows you to prioritize the display order of the icons.

In this example, the flag has a *z-order* of 10
and the player has a *z-order* of 61.

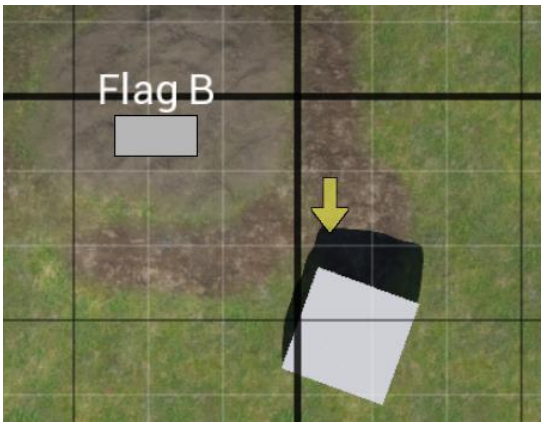
- **Ignore Rotation:** set this parameter to *true* to prevent the rotation of the element following the player's camera.

	
The value is *true*, the flag does not rotate with the map.	The value is *false*, the flag follows the rotation of the map.

Once you have completed all the steps described above, you can launch your game to check that the different actors are displayed correctly.

Markers on the map

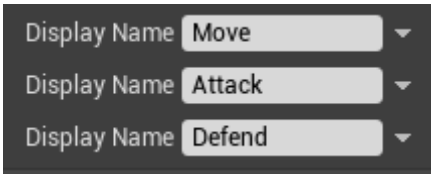
In addition to displaying automatic information related to the actors in the scene, players can position markers on the map to organize themselves during a game.



In this chapter, we will see the different widgets and actors to create to allow the display of a marker for the members of his team. However, we strongly invite you to consult the examples proposed in the demo code to understand in detail how it works in a multiplayer context.

Types of markers

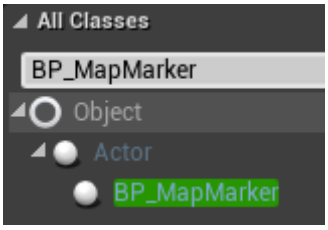
To start, we will create an *E_Markers* enumeration in which we will define different markers useful for the gameplay of the game.



In our example, this enumeration will be used to distinguish the display in the icon widget used on the map.

Creation of the marker actor

When the player wants to place a marker on the map, this is done by adding an actor in the game scene. So, we have to create a *BP_MapMarker_Game* object which will inherit from *BP_MapMarker* and in which we will define all the information related to the gameplay.



In the variables of this actor, we will define the type of marker using the previously created enumeration and the team to which the marker will be associated.

In the EventGraph of BP_MapMaker_Game, two events are to be used:

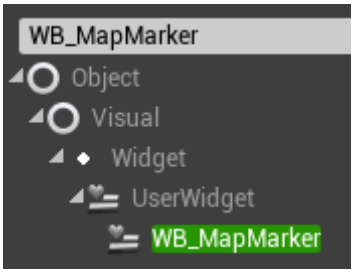
- **Event Begin Play:** allow to set the visibility of the object to the player according to the team he belongs to.
- **Event Tick:** update the visibility of the marker if the player changes team.

We invite you to consult the *GameMap/Blueprints/Map/BP_MapMarker_Demo* blueprint for more details.

Creating an icon widget

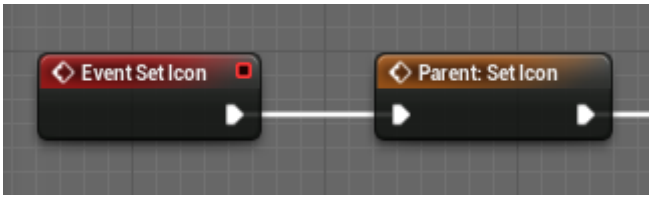
The display of markers on the map is done with a widget inheriting from *WB_MapMarker* and that we will name *WB_MapMarker_Game* in our example.

This widget will be used to be displayed on the map like the other information icons.

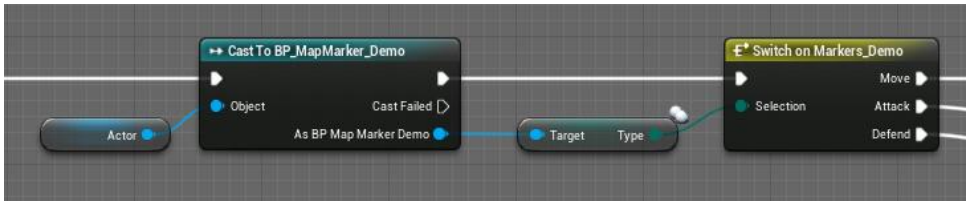


In this widget there is by default an image that will be used to display the image associated with the marker type. If the gameplay of your game requires more information in the markers, you can add them in this widget.

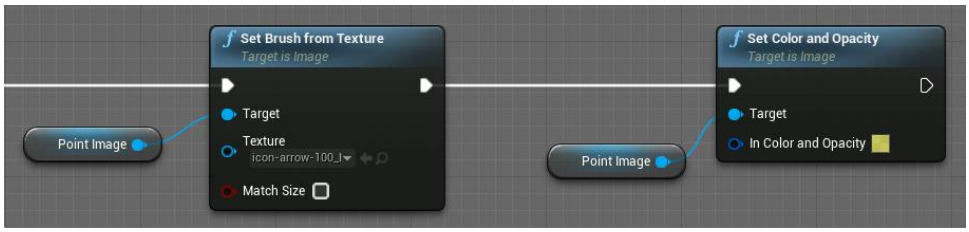
In the widget's *EventGraph*, you need to override the *SetIcon* function and start by calling the parent function.



In a second step, you must use the *Actor* variable available through inheritance. This variable allows you to have access to the marker actor associated with the widget. A *cast* is performed to get the type of icon to display.



Finally, the *switch* allows you to define the image and its color according to all the possibilities.



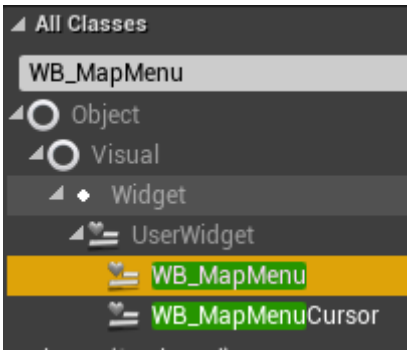
Note: you don't have to worry about the display conditions in this widget. This management is delegated to *BP_MapMaker_Game* which takes care of the visibility of the actors according to some conditions linked to the gameplay. The map takes care of displaying only the visible actors of the scene.

Contextual menu for markers

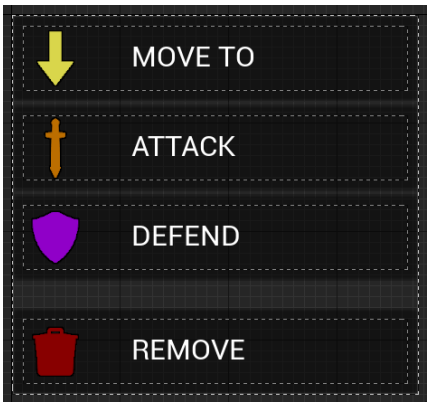
To place markers on the map, we need to create a widget that will list all the markers the player can place on the map.

Creating a menu widget

We will create a widget named *WB_MapMenu_Game* inheriting from *WB_MapMenu* in which we will add buttons to place markers.

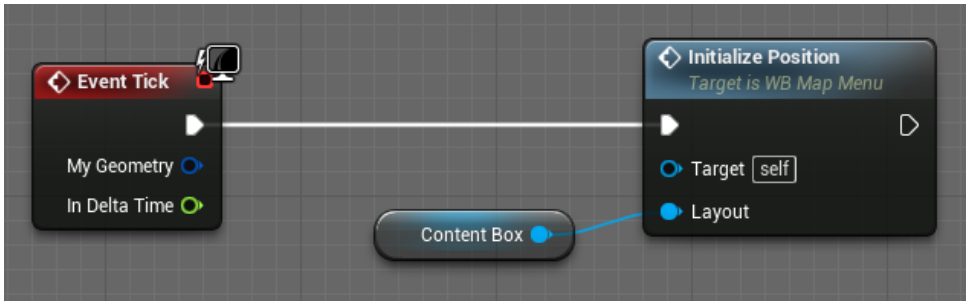


After opening the widget, start by adding a *Canvas Panel* and then the different elements that make up your menu.



Define the Tick event

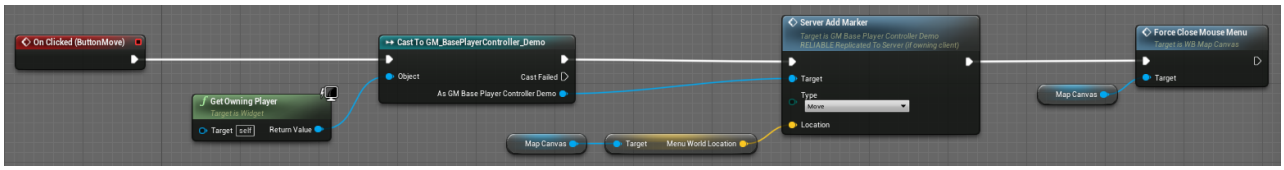
To allow the initialization and positioning of the widget when displayed on the map, we need to define the *Tick* event like the example below:



The variable *Content Box* is in our example a *Vertical Box* which contains all the buttons.

Functions for the click of the buttons

For each button, you need to add a function for the click event and reproduce the following code:

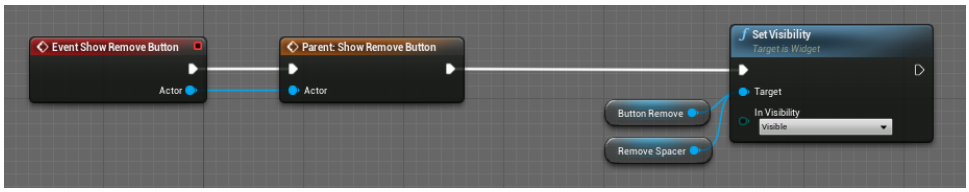


The *Menu World Location* variable of *Map Canvas* gives access to the position pointed by the mouse. In the example above, we call the *Server Add Marker* function which is defined in the game player controller. This function, designed to work in multiplayer, will allow the creation of an actor in the game scene. We invite you to consult the different blueprints proposed in the demonstration to understand and reproduce the proposed logic.

Show/Hide button to remove markers

To allow the display, if necessary, of the button for deleting a marker in the menu, the card will call an event named: Show Remove Button.

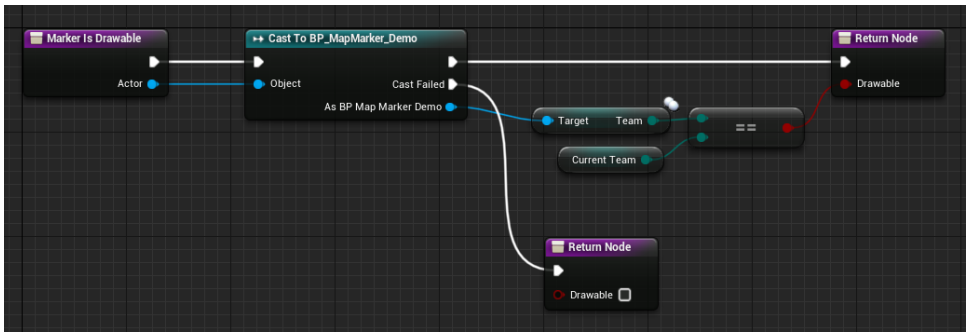
You must override the event to manage this information as you wish. In the example below, we display the remove button.



Conditions for displaying a marker

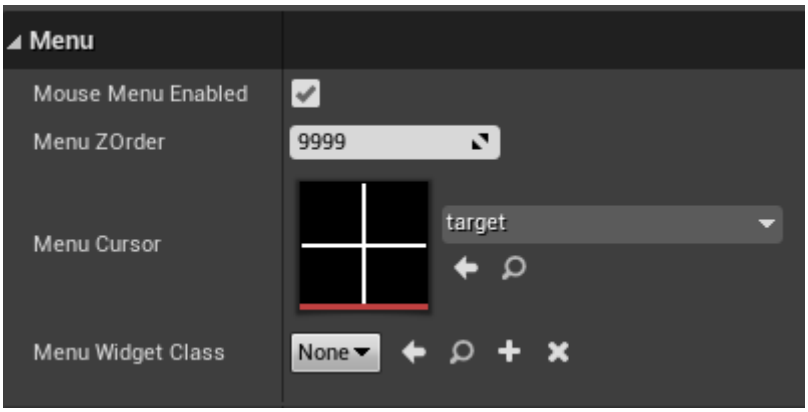
To define the conditions for displaying the markers, you need to override the *MarkerIsDrawable* function in the *WB_MapCanvas_Game* widget.

Using the actor passed as parameter, you can determine the display conditions. Below is an example that limits the display to the player's team.



Configuration of the contextual menu

The activation of the contextual menu is done in the widget containing the *WB_MapCanvas_Game* in the detail tab under Menu.



- **Mouse Menu Enabled:** by setting this value to *true*, the menu can be displayed with the right mouse click.
- **Mouse Z Order:** setting of the menu display height so that it is above the other elements on the map.

- **Mouse Cursor:** image used to indicate the precise location of the marker on the map and in the scene.
- **Mouse Widget Class:** indicates the widget to be used to display the menu.

To finalize the addition of the menu, define the class to use in the *Menu Widget Class* attribute and everything should be ready to add the marker on the map.

Version history

Each version of this package brings new features and bug fixes detected by our users.

Feel free to join the discord to suggest new features or seek help in case you encounter a problem when implementing the map in your game.

Version 1 (February 18, 2022)

This first version of the package contains the main components to allow the display of a map in a single or multiplayer game.

Features

- Texture-based map system that can be modified with the tool to take orbital pictures of your levels.
- Possibility to create different maps such as a map where the player can choose where to appear, a mini-map displayed in the HUD, a map to add markers for directions...
- The map has been developed for full compatibility with network games.
- Each player is associated with a team and displays information on the map accordingly.
- Example of displaying flags on the map with a color code indicating to which team it belongs.
- The player can choose where he will start the game by selecting the reappearance point of his choice on the map.
- System to signal the position of enemies on the map to teammates.
- Players can add markers on the map that will be visible only to their team.
- The player can place markers on the map as well as indications in the 3D game environment.
- Display a grid of coordinates on the map to divide it into zones.

Version 2 (April 15, 2022)

This version includes optimizations of existing elements and the addition of new features such as the ability to use the map in a side-scroller game.

List of changes

- Optimization to support very large maps (4x4 km).
- Added a demo level with a 4x4 km landscape.
- Support for side-scroller games by making some changes in the default settings.
- Use of the basic project “Side Scroller” proposed by UE4 for the creation of a demonstration level.
- Various modifications have been made to harmonize the use of distances in meters.
- Addition of a legend on the map to indicate distances in meters.
- Added a parameter to define the position (left/center/right and top/middle/bottom) of the icons according to the location of the actor in the 3D space.
- Improved the zoom animation and limited the maximum zoom with a parameter expressed in meters.
- Improvement of the grid settings by allowing the customization of the display threshold and the color used.