Map System – Unreal Engine Plugin

Documentation

Map System is mainly designed for open world games where good navigation is required. Map System allows you to add a map and a minimap to your game quickly and easily. You'll be able to customize everything the way you want. All classes have many parameters and delegates for fine tuning. Any of your objects on the level can be displayed on the map, for this you just need to add a component to the actor and configure it. Map images are captured automatically (or you can use your own), with the help of a special actor at the level.

This plugin adds 10 classes to UE Editor. You can find here what each class contains, what variables, functions and delegates you can use in blueprints.

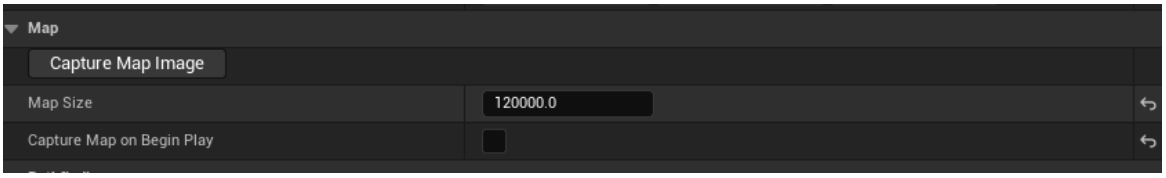
Video tutorial: https://youtu.be/iFtwCjECVec

1. Map Texture Capture

1.1

MapHelper is used to capture a map texture. To get started, follow these steps:

- Place a MapHelper actor on your level. Ensure it's positioned at the center of your map.
- Set the MapSize; this defines the area to be captured.
- Disable "CaptureOnBeginPlay."
- Press the "Capture" button.

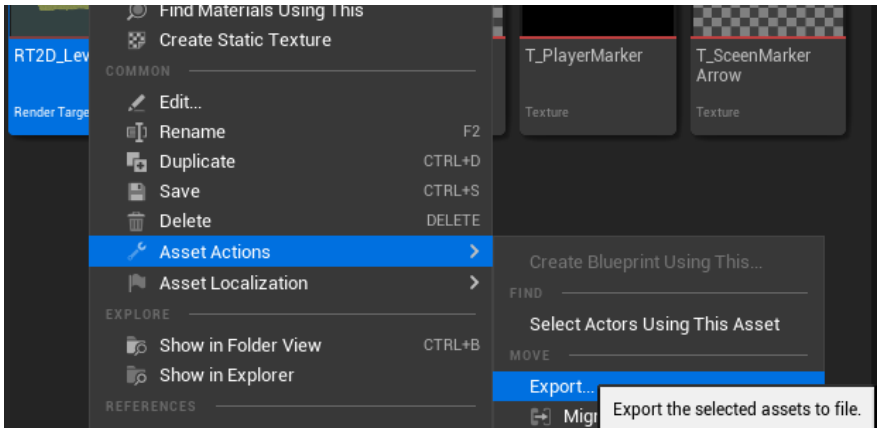


To view the captured map, navigate to MapSystem ->Textures ->RT2D_LevelMap.

1.2

You can customize the size of the Render Target Texture. It's recommended to use a size of **4096x4096 pixels**. To export it as an HDR file, follow these steps:

- Right-click on your Render Target.
- Choose Asset Actions -> Export.



1.3

Open the HDR file in a graphics program for editing or convert it directly to PNG or JPG format. This processed texture can now be imported into the Unreal Editor for use.

- Navigate to MapHelper->Materials.
- Open M_Map_Inst and M_Minimap_Inst.
- Assign the “MapTexture” as a new texture you’ve created.

1.4

If you're working with multiple levels, you'll need to repeat these steps for each level.

2. Adding Support for Multiple Levels

2.1

Capture map textures for each level in your game.

2.2

- Create a new Texture2D variable in BP_MapHelper, such as "MapTexture."
- Set up the following parameters:
    - Set "Instance Editable" to true.
    - Assign the "Category" as "Map".

You can create different Texture2D variables for the minimap and map if needed.

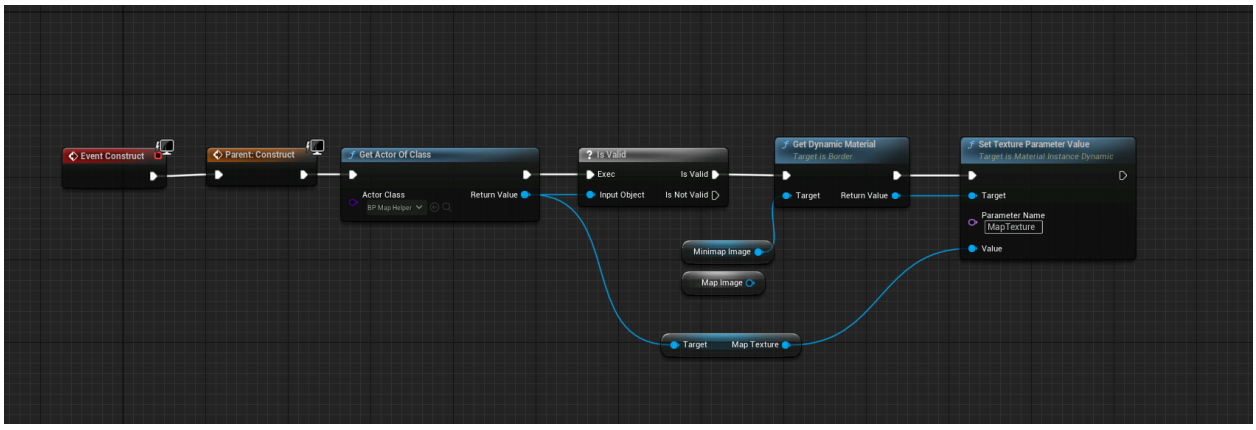
2.3

Assign these Texture2D variables in the MapHelpers at each level of your game.

2.4

In addition, you will need to assign the textures in the map and minimap widgets during the 'construct event.' To do this, follow these steps:

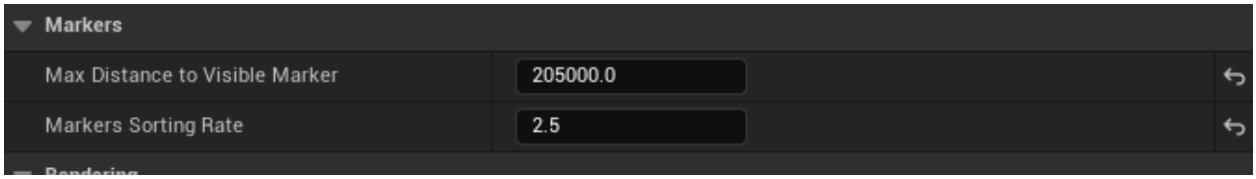
- Navigate to MapHelper->UI.
- Open WBP_Map and WBP_Minimap.
- Copy the code from the provided image



Code: https://blueprintue.com/render/x2humool/

3. Setting up the sorting of markers in MapHelper

MapHelper sorts markers by distance into hidden and visible groups. This allows for the avoidance of creating marker widgets if the marker is far away from the player. Here are two variable to configure this:



- MaxDistanceToVisibleMarker - If markers are further than this distance they'll be hidden
- MarkersSortingRate - It's how often markers will be sorted. Specify time in seconds.

Play with them to find the perfect value for your game. Additionally, you can change these values in the game dynamically.

4. Map System Classes:

1) AMapHelper : *AActor*

This actor is an important element for the operation of the map and minimap, it unites them. The main task is to capture the map image. It involves in creating and removing markers, pathfinding, creating custom markers, and updating the data in the MPC_Minimap

Variables:

```
USceneComponent* DefaultSceneComponent;

USceneCaptureComponent2D* MapCaptureComponent;

UBoxComponent* CaptureAreaVisualizer;

/**World map size */
float MapSize;

/**If true, then when the game starts, the map image will be
captured */
bool bCaptureMapOnBeginPlay;

/**It'll be created on right click on the map. USE ONLY BLUEPRINT
CLASSES*/
TSubclassOf<AMapCustomMarker> CustomMarkerClass;

/**Line tracing is used to find the surface on which the custom
marker will be placed */
          TEnumAsByte<ECollisionChannel> LineTraceCollisionChannel;

     /**The length of the Line Trace which is created when a custom
marker is spawning to search for a surface */
          float LineTraceLength;

     /**If true, it will not be possible to create more than one
custom marker */
          bool bOnlyOne;
```

```
     /**If true, then it will be possible to put a custom marker on
the map only in the place where there is the NavMesh */
          bool bProjectToNavigation;

     /**The size of the area to find the nearest point in the
NavMesh */
          FVector ExtentToProjectToNavigation;



/**If true, then the line between the start and the end will simply
be drawn. This doesn't require the NavMesh. */
bool bSimplePathfinding;

     /**If true, pathfinding will be performed asynchronously. */
          bool bFindPathAsync;

/**If true, then FindPath() will be called every frame.*/
          bool bFindPathEveryFrame;

/**If true, it'll look for the nearest point near the player to find
the path. It'll be useful if the player is outside the NavMesh.
Execute every frame */
bool bProjectPlayerLocationToNavigiation;

/**The size of the area to find the nearest point in the NavMesh */
FVector ExtentToProjPlayerLocToNav;

/**If true, it'll look for the nearest point near the custom marker
to find the path. It'll be useful if the custom marker is outside
the NavMesh. Execute every frame */
bool bProjectCustomMarkerLocationToNavigiation;

/**The size of the area to find the nearest point in the NavMesh */
FVector ExtentToProjCustomMarkLocToNav;

/**The position of the first pathpoint and the player may differ,
and if this difference exceeds this allowable value, then the
position of the player will be the first pathpoint. Too low a value
can result in path line artifacts on the map */
float MaxDifferenceBetweenPlayerAndFirstPoint;

/**If the path is lost during the player's movement, a line will be
drawn to the path that was found */
bool bOnlyUpdateFirstPointIfPathWasLost;
```

```
/**If there is no path, then a line will simply be drawn between the
start and end points */
bool bStartToEndIfNoPath;



     /**If markers are further than this distance they'll be hidden
**/
          float MaxDistanceToVisibleMarker;

     /**It's how often markers will be sorted. Specify time in a
second **/
          float MarkersSortingRate;
```

Functions:

```
/**Updates the player's position in the MPC_Minimap. By default,
every frame is called in EventTick */
void UpdateMCPMinimap();

/**Creates a map image */
void CaptureMapImage();

/**Returns size of the map */
float GetMapSize() const;

/**If true, then finding the path and position of the player on the
minimap will be suspended */
          void SetPauseMapHelper(bool bPause);

/**Converts the location to a position relative to the map helper */
FVector2D ConvertWorldLocToMapHelperPosition(FVector Location);
/**Converts the position relative to the map helper to world
location */
FVector ConvertMapHelperPosToWorldLocation(FVector2D Position);

/**Starts finding the path to the MarkerComponent every tick. Calls
OnFindingPath with true */
void StartFindingPathToMarker(UMapMarkerComponent* MarkerComponent);

/**Stops finding the path. Calls OnFindingPath with false */
void StopFindingPath();

/**Finds path points to a marker. By default, every frame is called
in EventTick */
void FindPath();
```

```
/**Returns an array of path points. These points were found while
finding a path */
const TArray<FVector>& GetPathPoints() const;

/**Returns true while finding the path */
bool IsFindingPath() const;

/**Returns the MarkerComponent to which it's finding a path */
UMapMarkerComponent* GetPathFindingComponent() const;


/**Converts the position to a world location and spawns a user
marker at that location. A Line Trace is made to scan the yaw axis
*/
void CreateCustomMarker(FVector2D MapHelperPos, bool
bStartFindingPath = true);

/**Returns only visible marker components. These markers are no
further than the DistanceToVisibleMarker from the player. */
     const TArray<UMapMarkerComponent*>&
GetVisibleMarkerComponents();

     /**Returns only hidden marker components. These markers are
further than the DistanceToVisibleMarker from the player. */
     const TArray<UMapMarkerComponent*>&
GetHiddenMarkerComponents();
```

Delegates:

```
/**Called when a visible marker is added to array of visible markers
*/
          FOnAddVisibleMarker OnAddVisibleMarker;

     /**Called when a visible marker is removed from array of
visible markers */
          FOnHideVisibleMarker OnHideVisibleMarker;

     /**Called when a new marker is created */
          FOnAddMarker OnAddMarker;

     /**Called when starts or stops finding path */
          FOnUpdateFindPath OnUpdateFindPath;
```

2) UMapMarkerComponent : *UActorComponent*

If add this component to any actor then it'll be displayed on the map and minimap.

Variables:

```
/**The marker widget of this class will be added to map and minimap. USE ONLY BLUEPRINT CLASSES*/
TSubclassOf<class UMapMarkerWidget> MarkerWidgetClass;

FText MarkerName;

FText MarkerDescription;

/**If true then it'll be displayed on map and minimap */
bool bVisible;

/**If true then it'll be able to find the path to this marker */
bool bPathable;

     /**If true then this market will only be displayed on the minimap */
          bool bOnlyMinimap;

     /**If true then this marker will be in the HiddenMarkers array in the MapHelper. It won't be displayed on the minimap and in the screen drawer but will be on the big map */
          bool bForceHiddenMarker;

     /**If true then the marker on the map will have the same rotation as its owner. This may be necessary for NPCs or other players */
          bool bRotateAsOwnerRotation;


     /** The marker widget of this class will be projected to the screen, only if ScreenMapDrawer exists. USE ONLY BLUEPRINT CLASSES*/
          TSubclassOf<class UScreenMarkerWidget> ScreenMarkerWidgetClass;

/**If true, the location of the marker can be projected to the screen */
          bool bCanBeProjectedToScreen;

/**If true, this marker will be projected to the screen but in the event that the hidden (ScreenMarkerDrawer) and visible (MapHelper) distance allow this **/
```

```
          bool bProjectToScreen;

     /**If true, this marker will always be projected to the screen,
ignoring the hidden(ScreenMarkerDrawer) and visible(MapHelper)
distance. */
          bool bAlwaysProjectToScreen;

     /**If true, this marker will always be on the screen */
          bool bStopAtScreenEdge;

     /**This value will be added to the owner location of this
component when projected to the screen. It may be useful in some
cases */
          float LocationOffsetZ;
```

Functions:

```
/**Starts finding the path to this marker */
void StartFindingPath();

/**Stops finding the path to this marker */
void StopFindingPath();

/**Returns true while finding the path to this marker */
bool IsFindingPath() const;

/**Returns true if bStopAtBorder, bFindingPath or
bAlwaysProjectToScreen is true */
          bool IsObjective() const;

     /**Returns true if bStopAtBorder, bFindingPath or
bAlwaysProjectToScreen is true */
bool CanBeProjectedToScreen() const;
```

Delegates:

```
/**Called when the right mouse button is pressed on the widget of
this marker on the map */
FOnRightMouseClick OnRightMouseClick;

/**Called when the left mouse button is pressed on the widget of
this marker on the map */
FOnLeftMouseClick OnLeftMouseClick;
```

```
/**Called when the mouse button is double-clicked on the widget of
this marker on the map */
FOnDoubleClick OnDoubleClick(bool);

/**Called when starts or stops finding path to this marker */
FOnFindPath OnFindPath(bool);
```

3) AMapMarker : *AActor*

This is an actor marker, this can be simply placed on the map, it already has a marker component and can be customized.

Variables:

```
UCapsuleComponent* Collision;

UMapMarkerComponent* MapMarkerComponent;
```

4) AMapCustomMarker : AMapMarker

This is set by right click on the map.

5) UMapWidget : *UUserWidget*

Variables:

```
/**If true, then the path can be drawn on the map */
bool bDrawPath;
```

```
     /**If true, marker positions and a path will be updated every
frame/
          bool bUpdateMapEveryFrame;

UCanvasPanel* MarkersCanvas;

UCanvasPanel* MapCanvas;

UBorder* MapImage;

UMapPathDrawerWidget* PathDrawerWidget;


/**This variable is responsible for the speed at which the map moves
when the left mouse button is pressed */
float MoveRate;

/**This is the map zoom step */
float ZoomStep;

/**This variable is used when interpolating the map size when
zooming. The smaller the value, the smoother the zoom */
float ZoomSpeed;

/**The maximum map size. Zooming changes the size of the map */
float MaxSize;

/**The minimum map size. Zooming changes the size of the map */
float MinSize;

     /**Multiply the size of the marker by this value */
          float MarkerSizeMultiplier;
```

Functions:

```
/**Completely updates the position of all objects on the map.
Removes all markers, updates the player's position, creates all
markers and finds a path */
void UpdateMap();


/**Remove markers and path */
void ClearMap();

/**Converts the map helper position to the map widget position */
FVector2D ConvertMapHelperToMapWidgetPosition(FVector2D
MapHelperPosition);
```

```
/**Converts the map widget position to the map helper position */
FVector2D ConvertMapWidgetToMapHelperPosition(FVector2D
MapWidgetPosition);
```

6) UMinimapWidget : *UUserWidget*

Variables:

```
/**Minimap scale depends on this value */
float Zoom;

/**If true, then the path can be drawn on the minimap */
bool bDrawPath;

UBorder* MinimapImage;

UOverlay* MarkerOverlay;

UOverlay* MaskedMarkerOverlay;

URetainerBox* MinimapMask;

UMapPathDrawerWidget* PathDrawerWidget;

/**When markers go out of view on the minimap, they stop updating.
This variable will be added to the minimap radius in order to
compare it with the distance to the marker */
float DisableMarkerRadiousOffset;

/**Markers may not leave the minimap even if they are far away, then
they remain on the border. With this variable you can adjust this
border */
float StopAtBorderOffset;

/**If the next pathpoint is outside the minimap, then it isn't drawn
*/
float StopDrawPathOffset;
```

Functions:

```
/**This function updates player icon rotation, markers positions and
path. By default, that's called every tick in NativeContruct */
void UpdateMinimap();
```

```
/**Sets the scale of the minimap */
void SetZoom(float Value);

/**Gets the scale of the minimap */
float GetZoom() const;
```

7) UMapMarkerWidget : *UUserWidget*

Variables:

```
UImage* Icon;
```

Functions:

```
/**Called when the left mouse button is clicked on this widget.
BlueprintNativeEvent */
void OnLeftMouseButtonDown();

/**Called when the right mouse button is clicked on this widget.
BlueprintNativeEvent*/
void OnRightMouseButtonDown();

/**This function is called when the marker is created in the map or
minimap widget. This is where variables are set and delegates are
bound. BlueprintNativeEvent */
void InitializeMarker(UMapMarkerComponent* MarkerComponent);

/**Returns the marker component of this marker widget */
UMapMarkerComponent* GetOwningComponent() const;

/**Returns true if this marker when out of the view of the minimap
will be displayed on the edge */
bool IsStopAtBorder() const;
```

Delegates:

```
/**Called when starts or stops finding path to this marker */
FOnFindPathToMarker OnFindPathToMarker(UMapMarkerWidget*, bool);
```

8) UMapPathDrawerWidget : *UUserWidget*

Variables:

```
FLinearColor Color;

float Thickness;

bool bAntiAlias;
```

9) UScreenMarkerDrawerWidget : *UUserWidget*

Variables:

```
          UOverlay* Overlay;

     /**Multiply the size of the marker by this value */
          float ScreenMarkerScale;

     /**How close will the marker be to the edge of the screen, 1.0
= at edge, 0.0 = at center of screen */
          float MarkerEdgePercent;

     /**If true, the arrow will indicate a direction to the marker
*/
          bool bIndicateDirection;

     /**if true, the distance to the marker will always be shown */
          bool bAlwaysShowDistance;

     /**Show distance to the marker if a path is being found to it
*/
          bool bShowDistanceIfPathfinding;

     /**If true, this marker will be on the screen when finding a
path */
          bool bStopAtScreenEdgeIfPathfinding;

     /**If true, screen markers will be hidden if distance from the
player is more than HideDistance **/
```

```
        bool bHideByDistance;

    /**If a screen marker is further away from the player than this
value then it'll be hidden **/
        float HideDistance;


    /**If no class is specified for the marker, this will be used
by default. USE ONLY BLUEPRINT CLASSES /
        TSubclassOf<UScreenMarkerWidget>
DefaultScreenMarkerWidgetClass;
```

Functions:

```
    /**Converts a world location to screen position for HUD
drawing. */
void ConvertWorldLocationToScreenLocation(const FVector& InLocation,
const float EdgePercent, const bool bStopCalcIfOff, const FVector2D
ViewportCenterLoc, FVector2D& OutScreenPosition, float&
OutRotationAngleDegrees, bool& bIsOnScreen);
```

10) UScreenMarkerWidget : UUserWidget

Variables:

```
        UOverlay* Overlay;

        UImage* DirectionArrow;

        UTextBlock* DistanceText;

    /**If true then a MapMarkerWidget will be created and added to
Overlay automatically. You must set it to false if you want to use
another widget instead of the default marker widget **/
        bool bAutoCreateMarkerWidget;
```