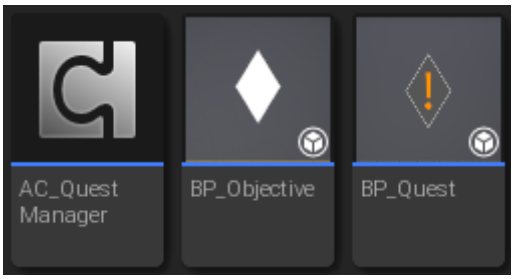
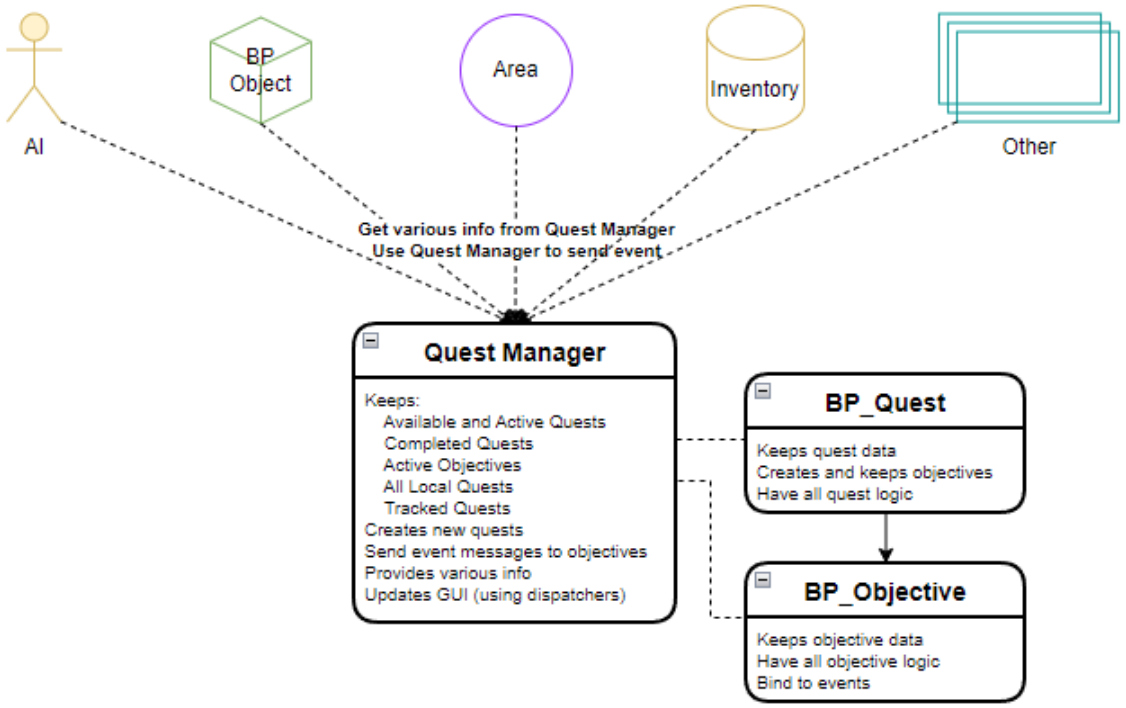
Quest System

Overview

Quest System includes 3 objects: Quest Manager, Quest and Objective.



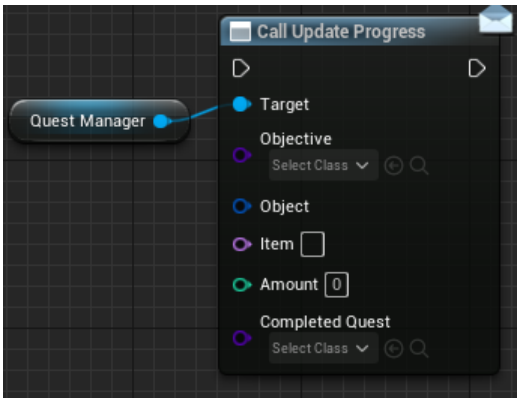
!!! Each Quest and Objective are unique object classes (children).



Quest Manager, Quest or Objective haven't hard references to world objects and react only to events.

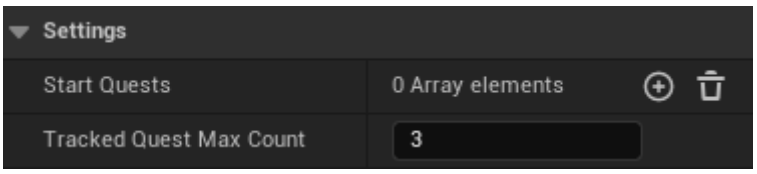
When you place objects on the scene, you can specify which quests and objectives they will be associated with.

When a player interacts with the world in different ways, game world objects send event message and provide the various data. All active objectives are bound to this event.



Quest Manager

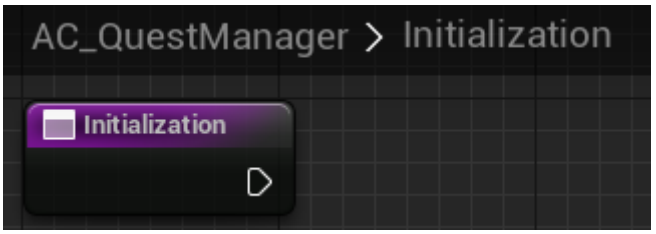
Quest Manager is added to the Player Controller. You can select it and set settings.



Initialization function is called in Player Controller (On BeginPlay), after HUD is created, and Track Manager is initialized.

I do this because Quest System hasn't any hard references to GUI, and uses dispatchers for updating.

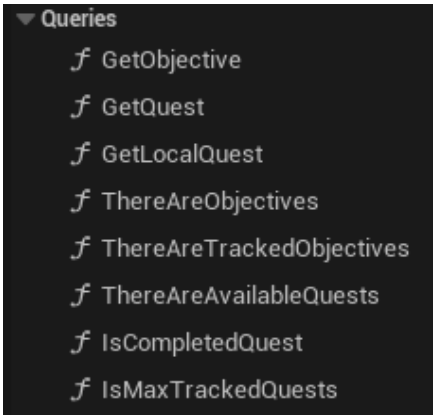
If Quest Manager is initialized earlier than GUI is created, there may be visual bugs. I don't want to write additional logic in order to handle this. It is simpler to do as I wrote above.



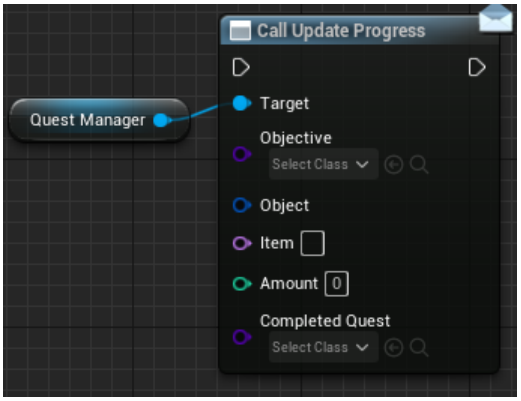
What is Quest Manager for?

Manager keeps all needed info.

During the game, world objects can get this info from the manager.



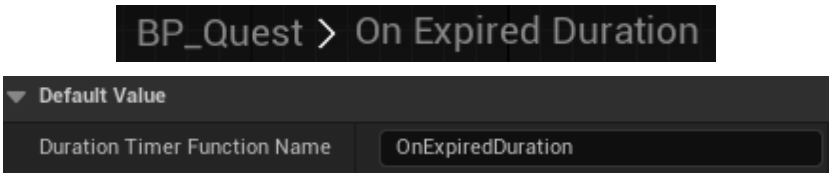
Using the manager we can **update objective progress**, **track/untrack** quests, **save/load** quests, **change** quest state etc.



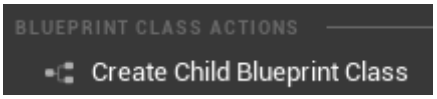
Quest

Quest is an object class that keeps quest data and logic.
BP_Quest is a template. All real quests are children of this class.

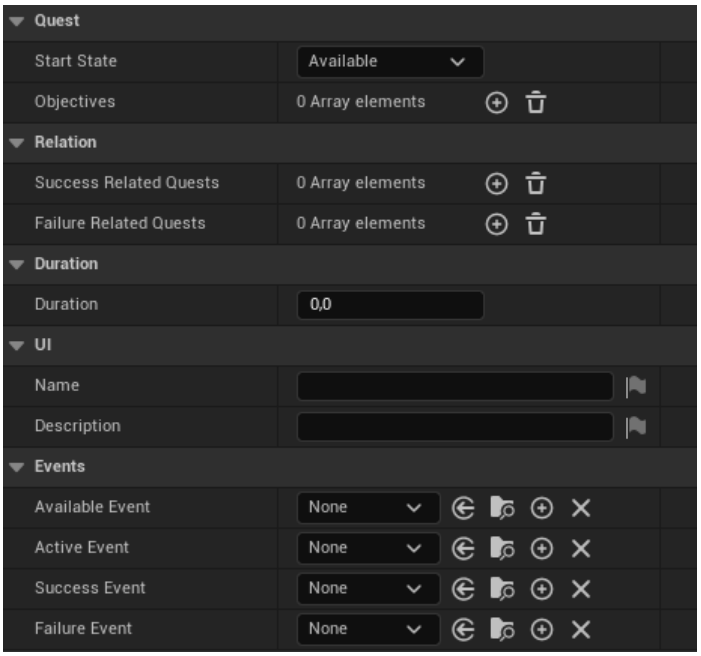
!!! Each Quest is a unique object class (child).
!!! If you changed this function name in BP_Quest, please, select the variable and set new default value.



Create new quest

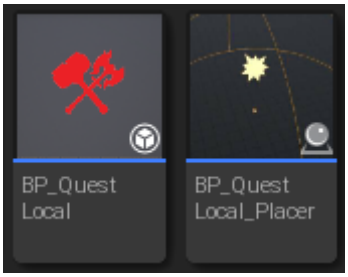


When you create a new quest, you need to open the class and set the data. Hover cursor on each parameter to see description.



Local Quest

Local Quest is a child of **BP_Quest**.



If you open it you will see that I have overridden **ChangeState** and **ActiveState** functions. It was necessary, because we don't need some functionality from **BP_Quest**.

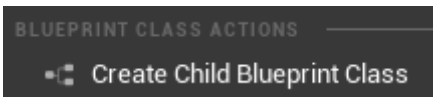
By design Local Quest has 2 states:

- Active (at start of the game)
- Success

Local Quest can be "discovered" when the player overlaps an area for the first time.

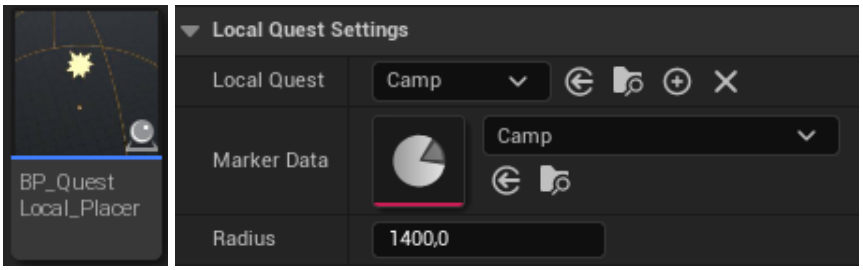
Create new local quest

The same for the quest, only need to create a child class from **BP_LocalQuest**.

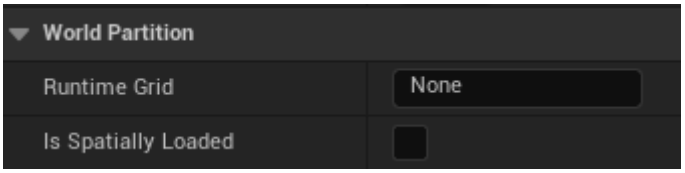


Place local Quest in the game world

Since quest is Object class, you cannot place it in the world. For this you need to use **BP_LocalQuest_Placer**. Placer creates the quest. Placer has sphere collision that is used for discovering area. Also placer is a target for track system. It can be added on Compass and World Map.

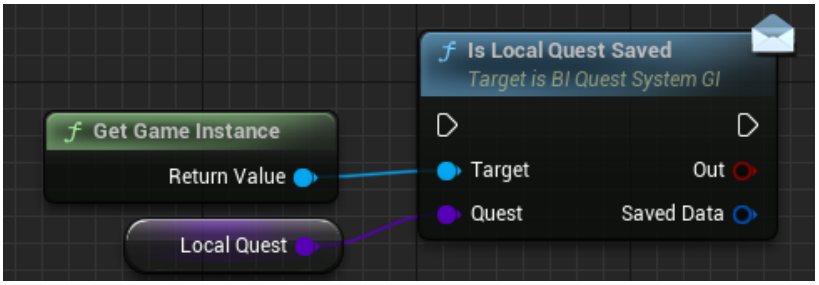


Placer should be always loaded. For WorldPartition, IsSpatiallyLoaded = False by default.



If you use Level Streaming. Place it on Persistent Level.

Saving and loading work in such a way that when the world is loaded, the placer asks the game instance if the quest is saved. For everything to work correctly, 1 unique quest should be assigned only to one place.

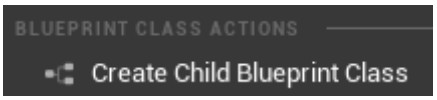


Objective

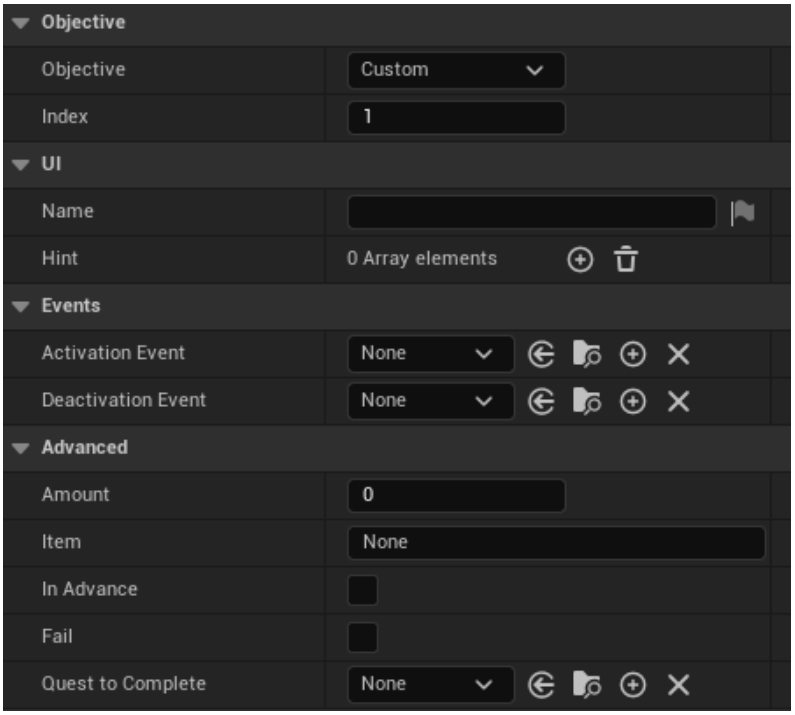
Objective is an object class that keeps objective data and logic.
BP_Objective is a template. All real objectives are children of this class.

!!! Each Objective is a unique object class (child).

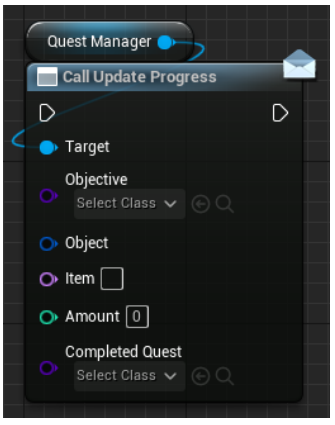
Create new Objective



When you create a new quest, you need to open the class and set the data. Hover cursor on each parameter to see description.



On **activation/deactivation** objective binds/unbinds to the **UpdateProgress** dispatcher event.

In order to update objective progress you need to call **UpdateProgress**. **<Item>** and **<Amount>** are used for **Gather**. **<Object>** is used for **Activate** and **Deactivate**. **<Objective>** is used for others.	

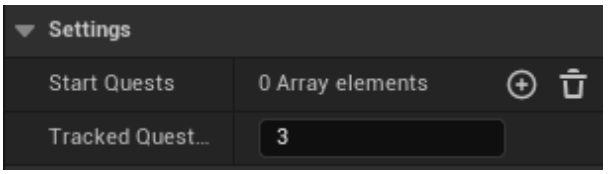
Add Quest To Game

Initialization

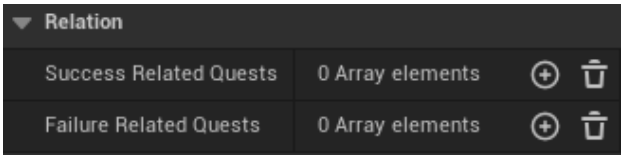
!!! Local Quest is just placed on the scene, see Local Quest section.

If Quest is accessible for Player at start game, it should be added to **StartQuests** parameter in Quest Manager settings.

Quest will be created, added to manager, initialized and tracked automatically.

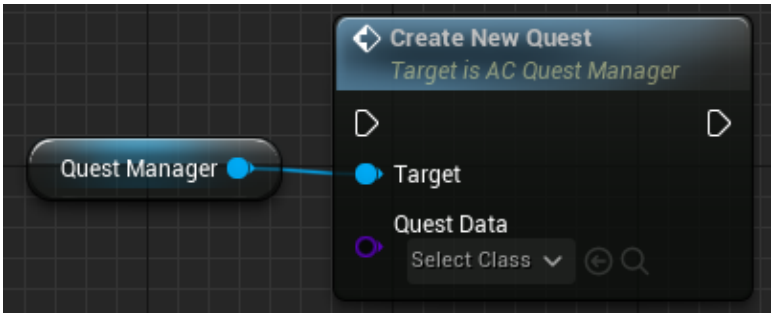


If Quest is accessible for Player when another quest is completed, it should be added to the appropriate parameters in Quest settings. When Quest is completed, quests will be created, added to manager, initialized and tracked automatically.



In both 1 and 2 cases CreateNewQuest function is called for Quests.

If you have another situation, for example, you want to initialize quest when player overlaps an area, you need to call CreateNewQuest from quest manager.



Assign with actors

As I wrote earlier, Quest Manager, Quest or Objective haven't hard references to world objects and react only to events.

Player has to kill a specific AI, interact with specific objects, reach specific place etc.

For this you need to assign Quests or Objectives to needed objects on the scene.

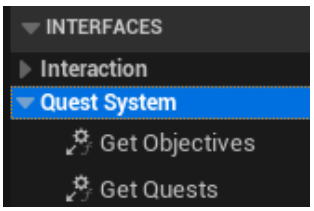
Let's look at **AI**.

If a quest is available or objective is activated, the player can interact with AI.

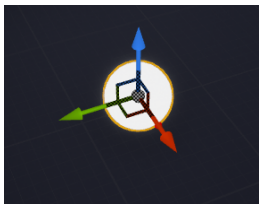
Quest Manager needs to know if the player interacts with the right object, without knowing about this object.

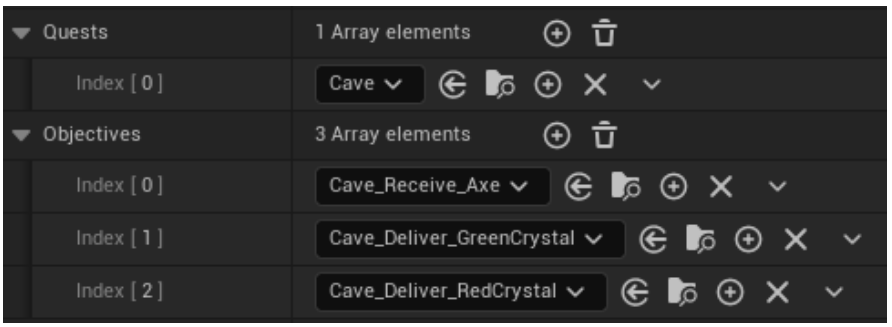
For this two variables are created in **BP_SpawnPoint.**

And interface is implemented in **BP_AI.**



When you place Spawn Point on the scene you can set quests and objectives.





The same implementation I use for BP_Marker.
Also **Objectives** parameter is used in the next cases:

- this info needs to be passed when we call **UpdateProgress** event
- when we need to show objective world marker (see AI or Object classes)

Also see BP_Place, BP_Region, BP_Object.
Objectives parameter is already added.

Events

Sometimes it may be necessary to execute additional logic at a certain time, for example, when quest or objective are completed.

It can be any additional logic you want:

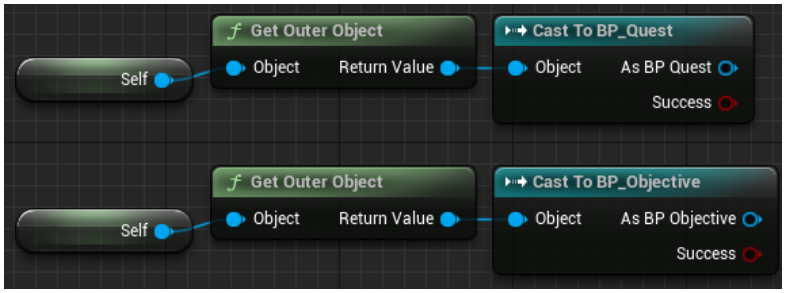
- change environment
- change AI state

etc.

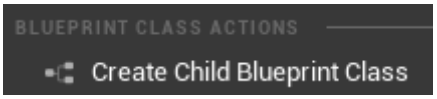
For this I created a special class **BP_QuestSystemEvent.**

It has a link to Player Controller. Using it you can get reference to any other managers like AI or Environment manager.

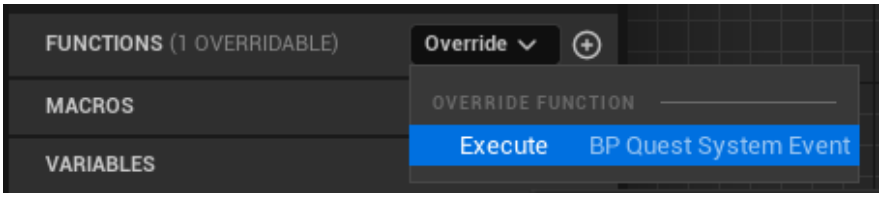
Using GetOuterObject you can get reference to Quest or Objective from what it was called.



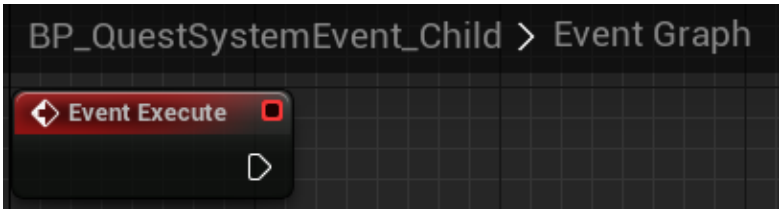
To create a new event, just create a child class from **BP_QuestSystemEvent.**



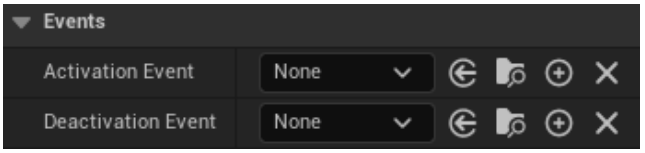
Override Execute function



And create logic you want



Finally, you can set event in Quest or Objective Settings.



!!! When loading the game, events are not triggered again. It is supposed that the state of AI or Environment static objects will be saved and controlled by other systems.

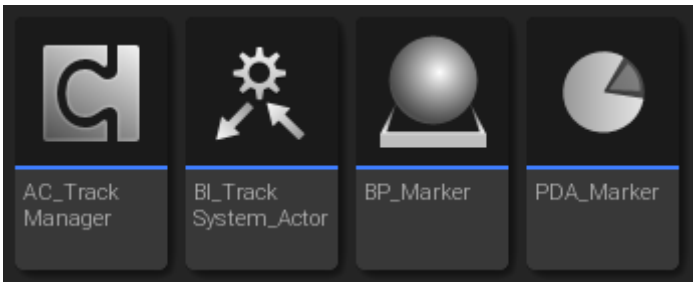
!!! Events are only created and executed. They are not saved. Execution flow is not controlled by outer objects.

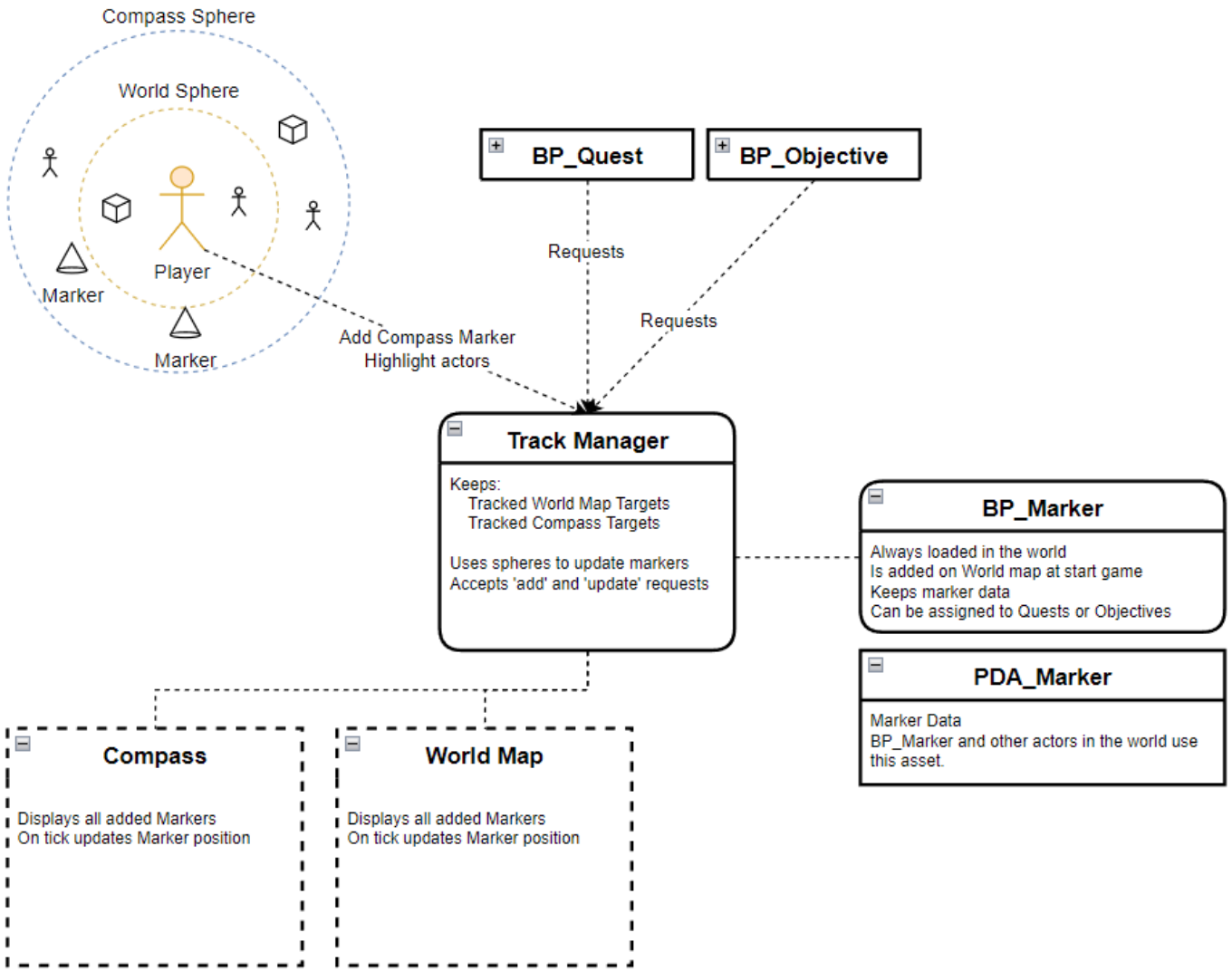
Track System

Overview

Track System is created to:

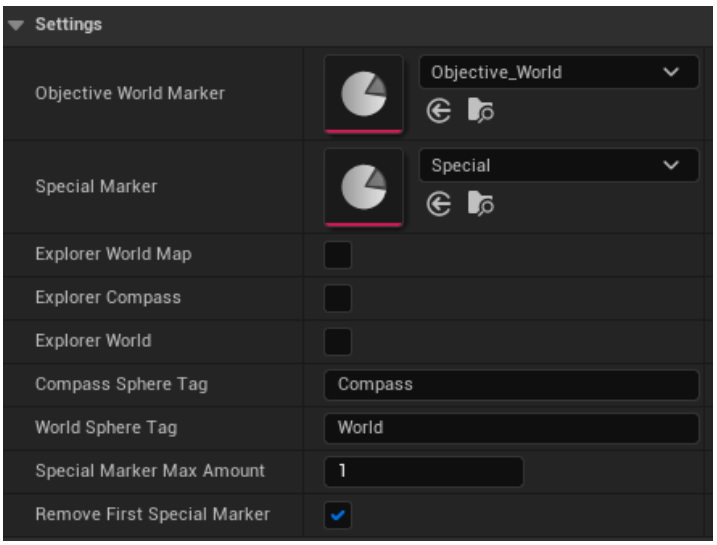
- display target location on Compass and World Map using markers
- highlight actors in the world





Track Manager

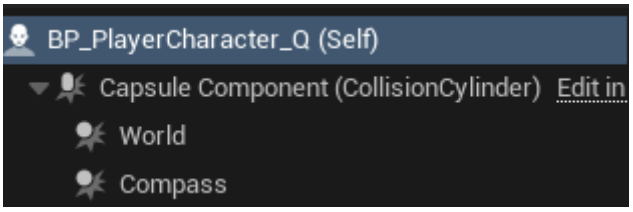
Track Manager is added to the Player Controller. You can select it and set settings.



Initialization function is called in Player Controller (On BeginPlay), after HUD is created, before Quest System initialization.

I do this because Track System receives requests from Quest System, and if it is initialized after Quest System, some targets won't be tracked at start game.

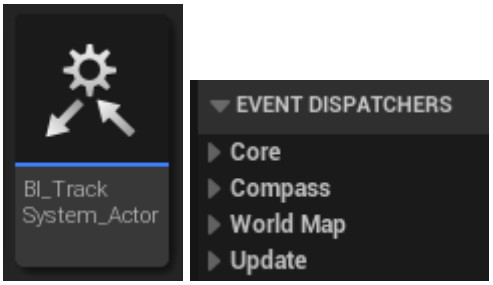
On Initialization, the manager gets references to spheres that are added to the player character. Later it will use these spheres to get overlapped targets and update markers.



What is Track Manager for?

Manager accepts requests to add, remove and update markers.

It knows nothing about targets (actor class) so it works with them using interface and dispatcher events.



You can implement the interface and dispatchers in any actor you want.

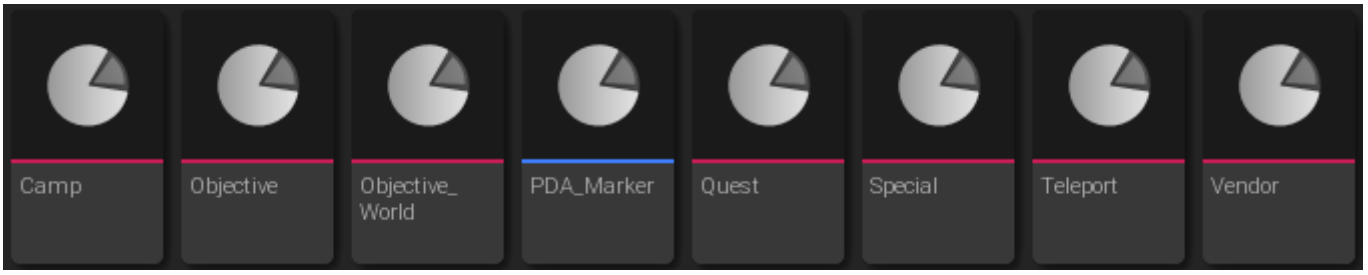
For example, see my **BP_LocalQuestPlacer**. I don't use BP_Marker to track local quest location. Local Quest Placer is a target for track system.

Marker Data

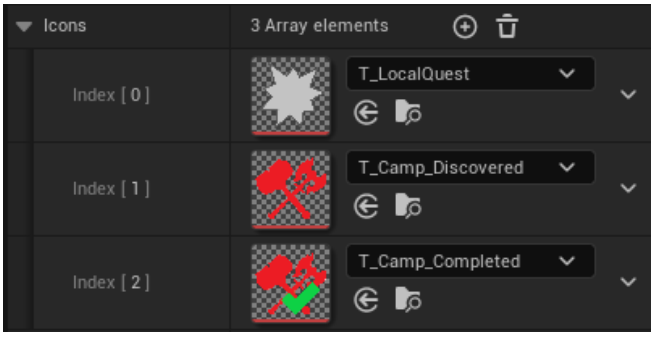
Data about the marker is stored in a primary data asset **PDA_Marker**. This is a template.

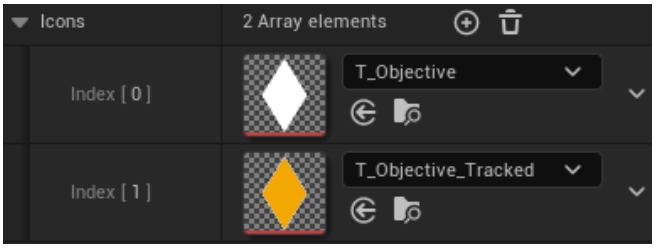
You can create in data assets not only parameters, and also supporting functions.

Real markers are created based on PDA_Marker.
Click RMB on Content Browser -> Miscellaneous -> Data Asset -> Select PDA_Marker.

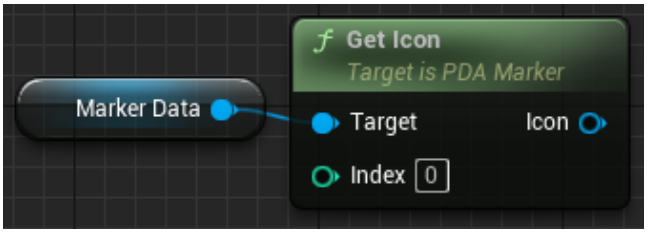


Let's look at **Icons** parameter.
For Camp I set 3 icons. For Objective - 2 icons.



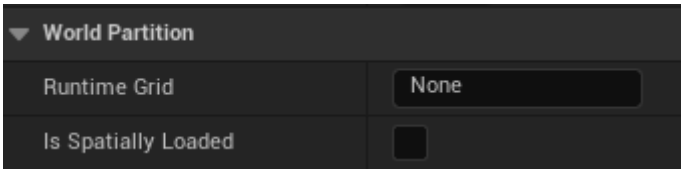


In widgets I can determine what icon I need to display.



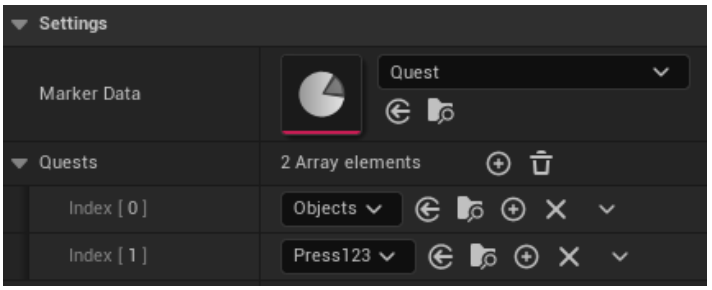
BP_Marker

BP_Marker should be always loaded. For WorldPartition, IsSpatiallyLoaded = False by default.

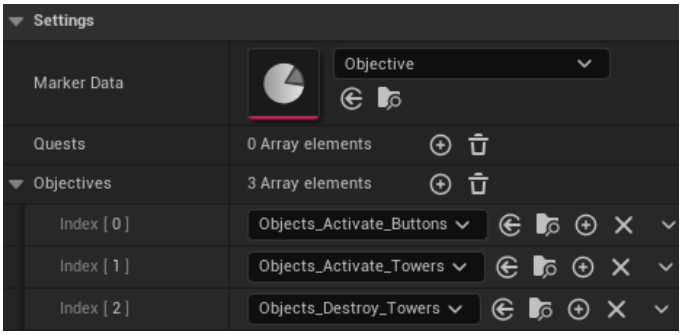


If you use Level Streaming. Place it on Persistent Level.

For Quest marker don't forget to set Quests. When these quests are available, target is tracked.



For Objective marker don't forget to set Objectives. When these objectives are activated, target is tracked.



A few words about tracking pawns - **Vendors**, **Quest Givers**.

On World Map we track BP_Marker. More about it in **Movable Target Tracking**.

On Compass we don't need to track BP_Marker, we need to track pawns, because pawns can move around the world.

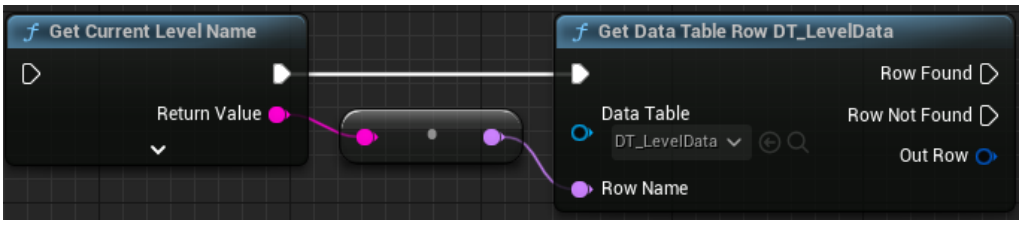
That's why BP_Marker is not added to compass for Vendor and Quest.

See Update Compass Marker in BP_Marker.

World Map

Level Data

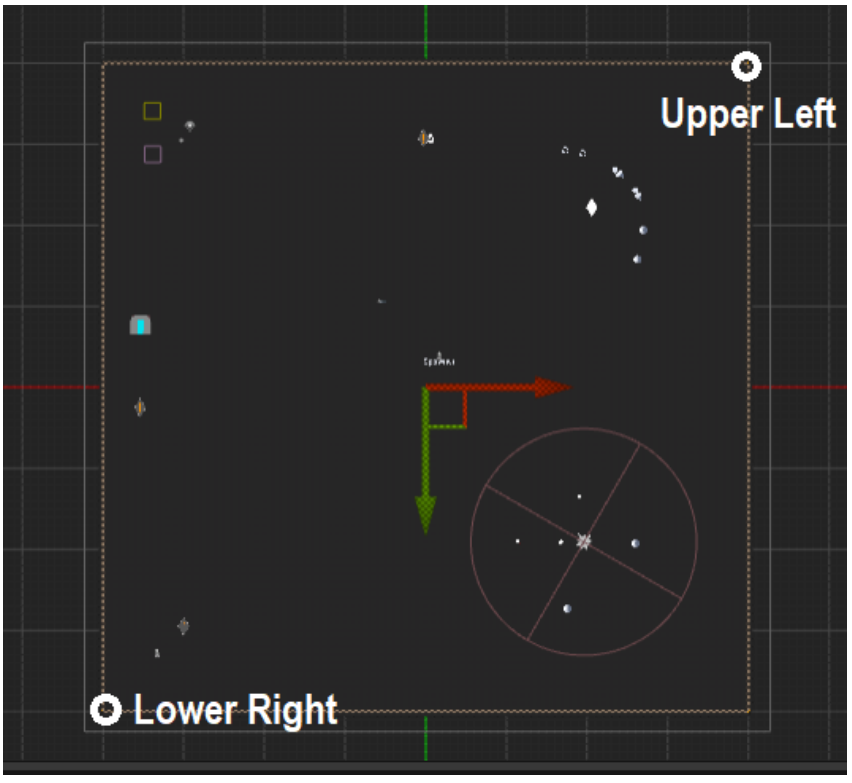
WorldMap gets level data using GetCurrentLevelName and Data Table.

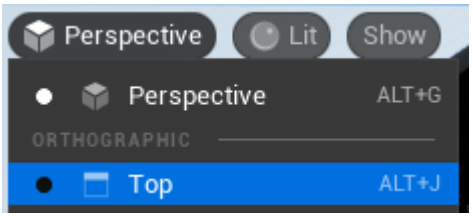


RowName = Level Name.

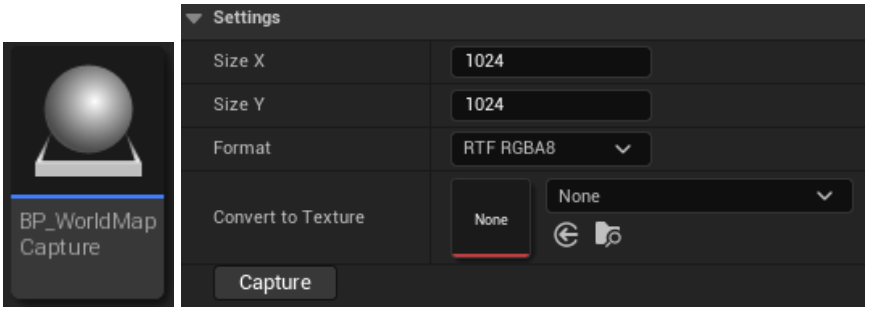
	Row Name	MapTexture	WorldUpperLeftCorner	WorldLowerRightCorner
1	Cave	Texture2D'/Game/QuestSystem/TrackSystem/T_Cave.T_Cave'	{ "X": 2500, "Y": -2500, "Z": 0 }	{ "X": -2500, "Y": 2500, "Z": 0 }
2	Main	Texture2D'/Game/QuestSystem/TrackSystem/T_Main.T_Main'	{ "X": 4000, "Y": -4000, "Z": 0 }	{ "X": -4000, "Y": 4000, "Z": 0 }

WorldUpperLeftCorner and **WorldLowerRightCorner** are needed to convert world position to map widget position.
Switch viewport to top view, and get coordinates of UpperLeft and LowerRight corners of your world.





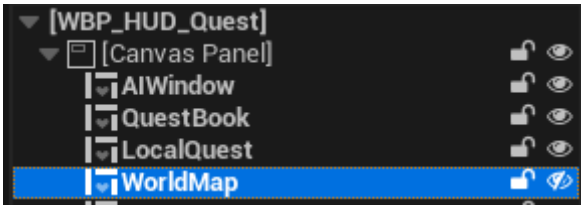
After you create a row in data table and set coordinates, you can create a map texture.
Place **BP_WorldMapCapture** on the scene. It is automatically aligned to the center using data from data table.
Set settings and press Capture button. Map texture will be created and saved in specified asset.



!!! Currently there is a limitation: world map works with **square** world.

GUI

Open **WBP_HUD_Quest** and select WorldMap widget. As you can see I have hidden it in editor.



There are settings for the map.

All UI logic is in this class - WBP_WorldMap.
There are dispatcher events in Track Manager in order to update UI. Some widgets are bound to these events.

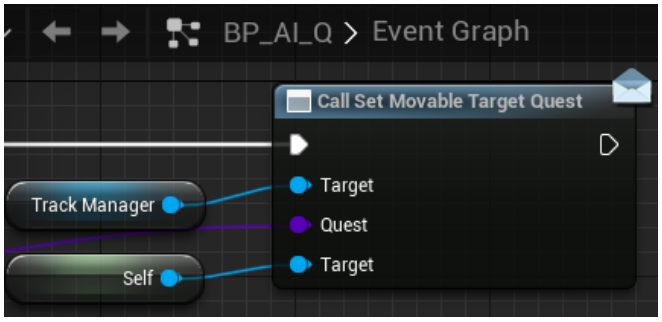
Track Movable Target

A few words about tracking pawns - **Quest Givers**.

BP_Marker for quest is added on World Map when quest is available.

If we are working with a large world, sometimes AI can be unloaded from the world. In this case we can track BP_Marker. When AI is loaded again, we need to track him on World Map, because he can move around the world.

As soon as AI appears in the world, if it has any quests, it calls the event.



BP_Marker is bound to this event.

If BP_Marker has this quest, it saves the target as movable.

WorldMap widget, in turn, when updating marker position, using a special interface function, checks if the main target has additional (movable) target.

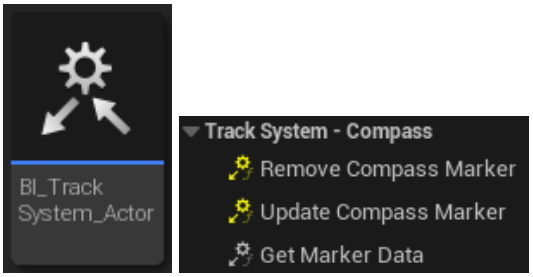
Out of the box, it works only for Quest Giver, because we can find a marker via quests.

Mandatory rule:1 BP_Marker for 1 Quest Giver.

Compass

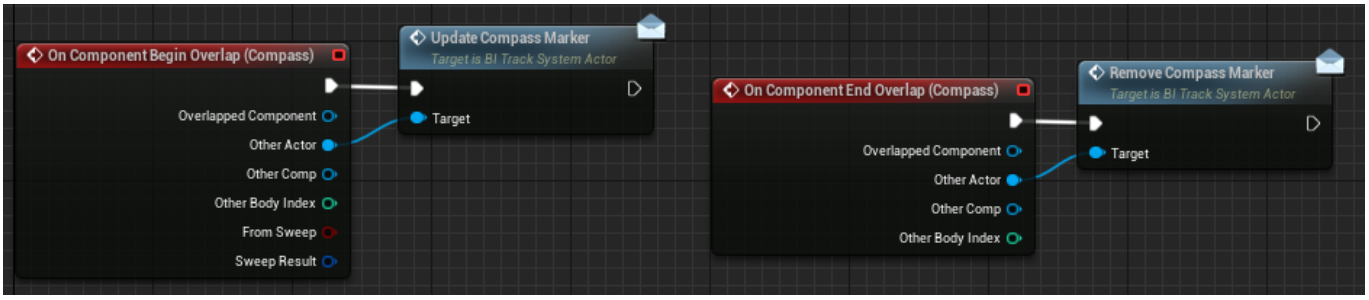
Spheres

If you want the actor to appear on Compass, add a special interface to this actor and implement the next functions. See BP_Marker or BP_AI.

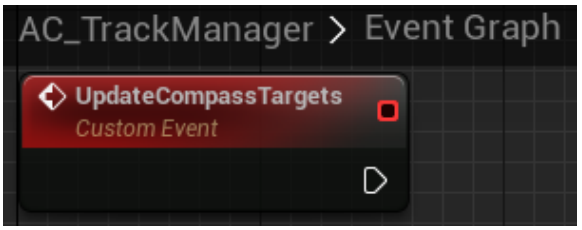


To determine what targets should be added to the compass, sphere collision is used.
Sphere Collision Component is added to Player Character. Component Tag = Compass.
If you want to change radius, select component and set new value for parameter SphereRadius.

When player moves, the sphere overlaps actors, and Overlap events are triggered.

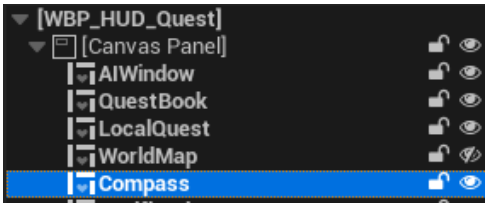


Also Track Manager uses the spheres to update all overlapped targets upon request from Quest System, for example, when quest becomes available or active.

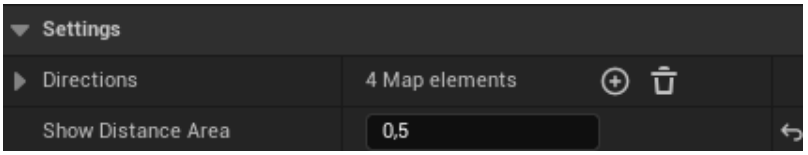


GUI

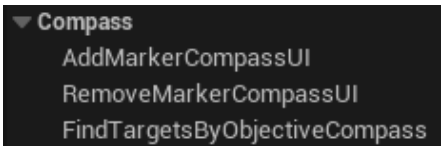
Open **WBP_HUD_Quest** and select Compass widget.



There are settings for the compass.



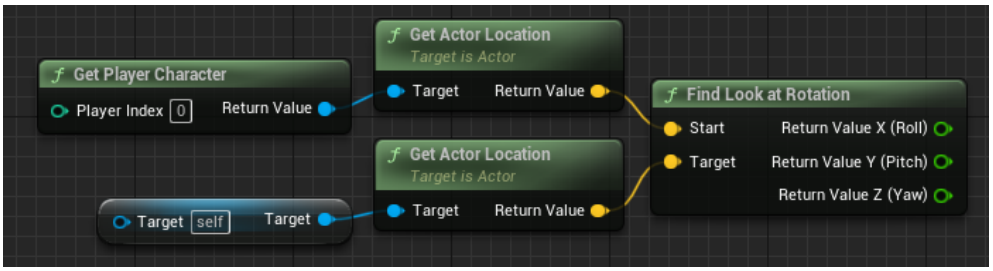
All UI logic is in this class - WBP_Compass.
There are dispatcher events in Track Manager in order to update UI. Some widgets are bound to these events.



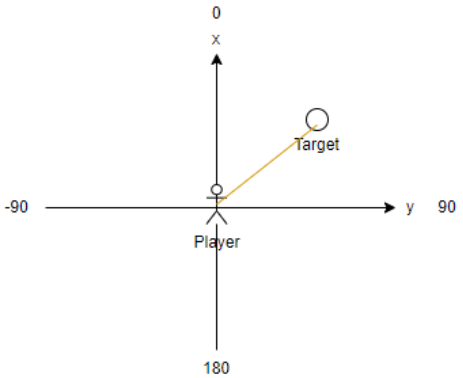
Translation Calculation

Here I describe How we get marker translation for Compass.

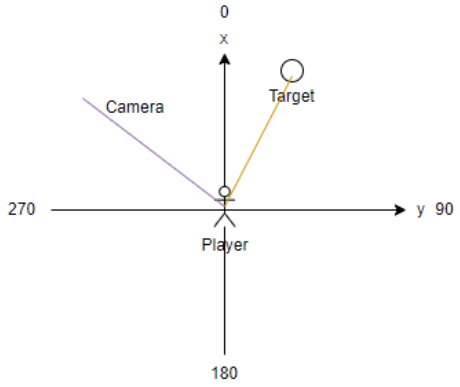
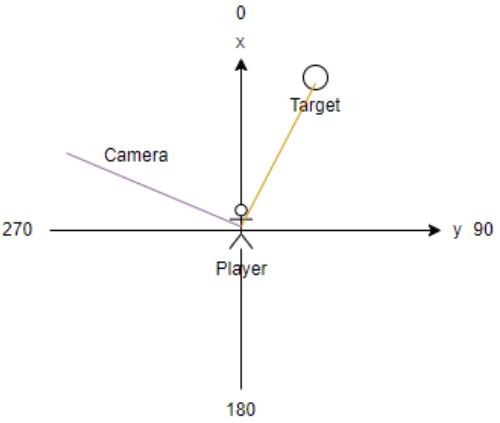
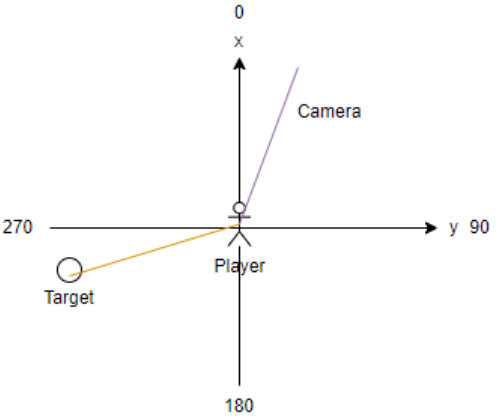
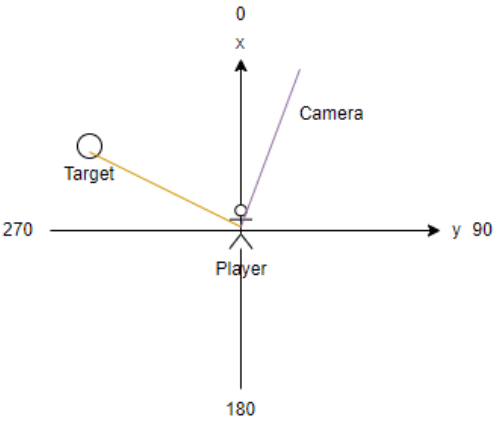
We get **Target Position** using **Find Look At Rotation**.



Here Player is a center of the coordinate system.

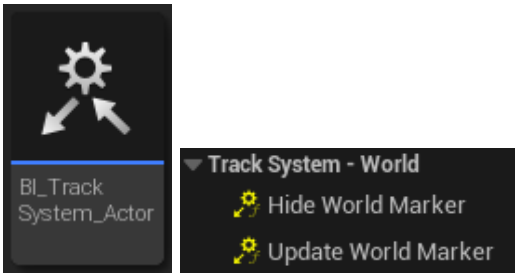


Then we need to get camera rotation and get angle between two vectors from 0 to 360.

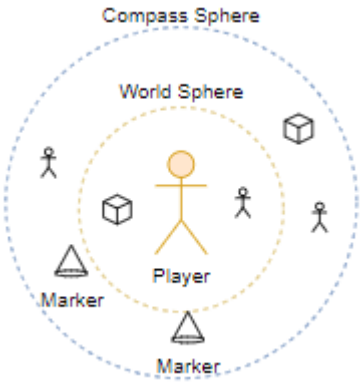
Camera(315) > PlayerToTarget(35). Angle = 315 - 35 = 280. 280 > 180. Target is on the right. Real Angle = 360 - 280 = 80. Marker is not out of FOV.	
Camera(300) > PlayerToTarget(35). Angle = 300 - 35 = 265. 265 > 180. Target is on the right. Real Angle = 360 - 265 = 95. Marker is out of FOV.	
PlayerToTarget(260) > Camera(15) Angle = 260 - 15 = 245. 245 > 180. Target is on the left. Real Angle = 360 - 245 = 115. Marker is out of FOV.	
PlayerToTarget(290) > Camera(15) Angle = 290 - 15 = 275. 275 > 180. Target is on the left. Real Angle = 360 - 275 = 85. Marker is not out of FOV.	

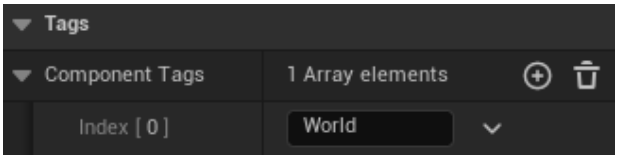
World

If you want the actor to be highlighted with a marker in the world, add a special interface to this actor and implement the next functions. See BP_Object or BP_AI.



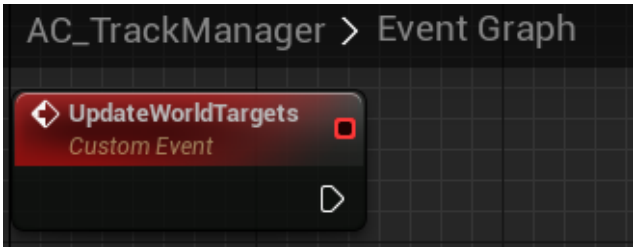
To determine what targets should be highlighted, sphere collision is used.
Sphere Collision Component is added to Player Character. Component Tag = World.
If you want to change radius, select component and set new value for parameter SphereRadius.



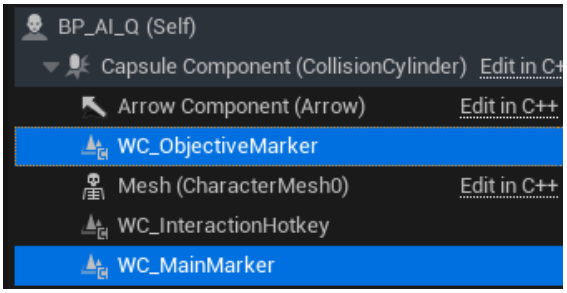


When player moves, the sphere overlaps actors, and Overlap events are triggered.

Also Track Manager uses the spheres to update all overlapped targets upon request from Quest System, for example, when objective is activated/deactivated.

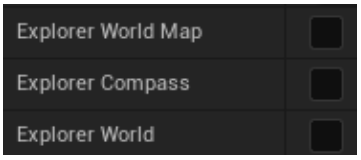


BP_AI has two markers. It can be a Vendor or Quest Giver, and also it can be highlighted as an objective for other quests.

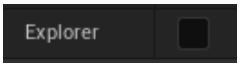


Explorer Mode

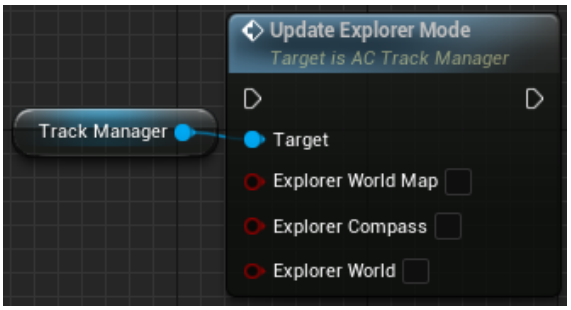
Select Track Manager in Player Controller.
If you want to hide markers on World Map, Compass or in World, you can enable these parameters.



There is a similar parameter in marker data (PDA_Marker).

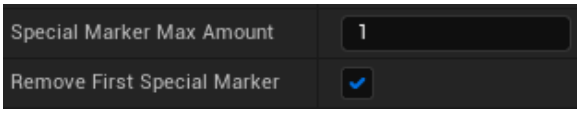


Global parameters don’t affect marker visibility, if the parameter in PDA_Marker is false.
It is useful if you want some markers to be left visible.
Use the next function to change the mode at runtime.

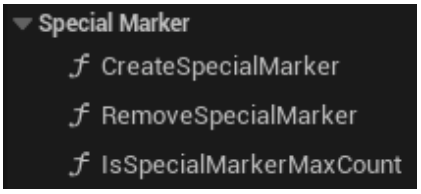


Special Marker

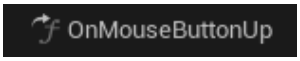
There are parameters in Track Manager settings. Hover cursor to see description.



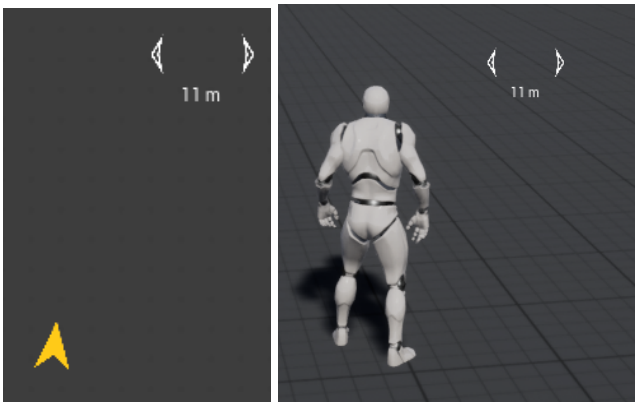
There are functions in Track Manager.



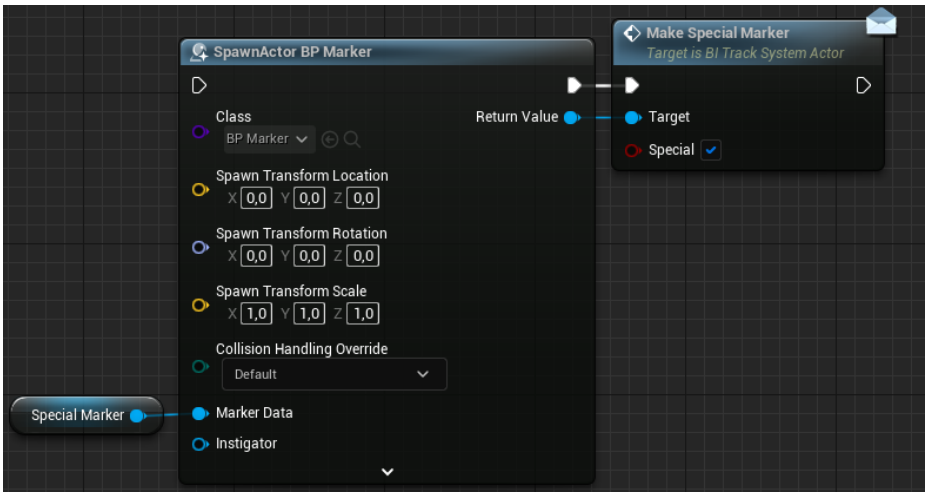
CreateSpecialMarker is called when player clicks LMB on World Map.



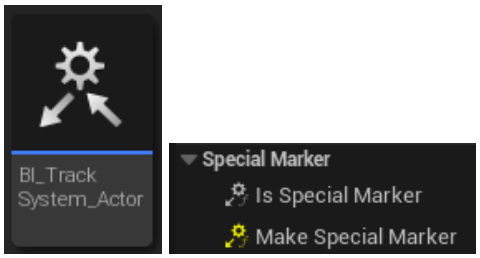
Marker is added to World Map and to the world.



When I spawn BP_Marker I call interface function in it.
In this function I add a widget and collision to the marker. **RemoveSpecialMarker** is called when the player has reached the location (handled in BP_Marker with this collision).



Out of the box, a special marker is implemented as a separate object.
But I use interface to determine if marker should be special, so theoretically you can make already existing marker special. See implementation in BP_Marker.

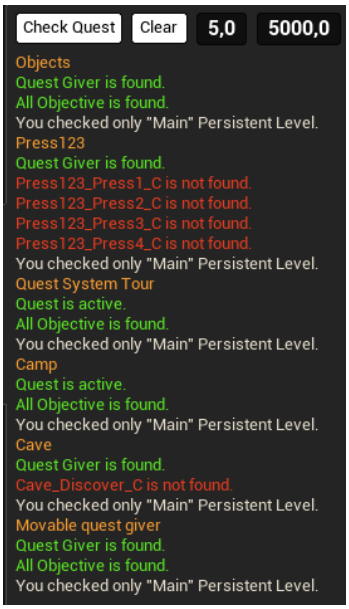


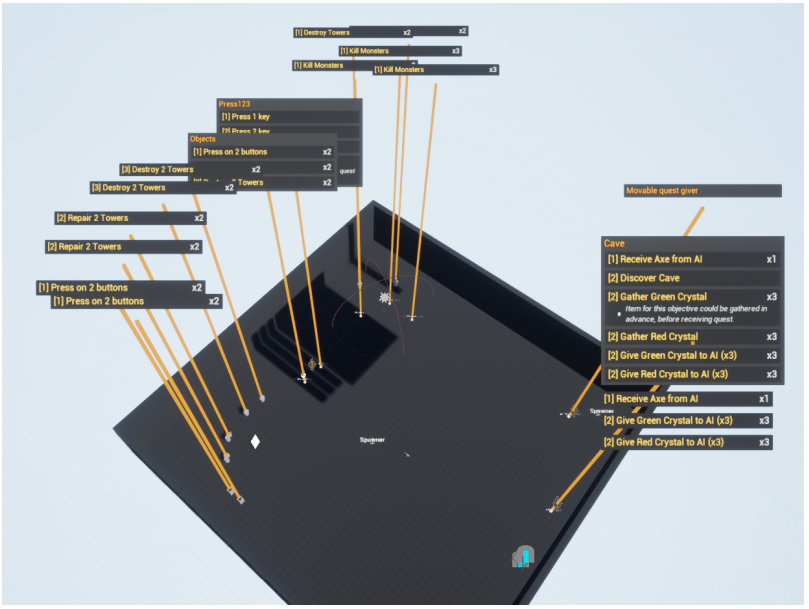
Quest Helper

First you need to run it. Click RMB on asset and select **Run Editor Utility Widget**.

You can place the widget in the editor as you want.

In the content browser, select one or more quest object classes (created from BP_Quest) and press the **Check Quest** button. **BP_EditorDataHolder** actors will be created on the scene.
Clear button removes all BP_EditorDataHolder actors from the editor.





How is Quest Helper useful?

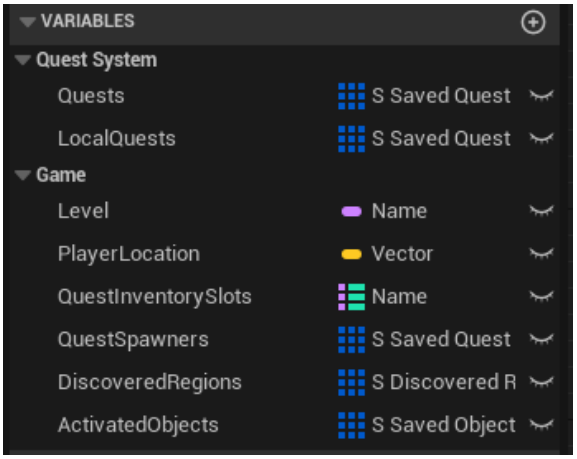
Helper checks if all quest elements (quest and objective classes) are assigned with actors in the world. In the future, new functions can be added.

Start State is “Available”. Quest is associated with AI (spawn point).	Quest Giver is found.
Start State is “Available”. Quest is not associated with AI (spawn point).	Quest Giver is not found.
Start State is “Active”. Player don’t need to receive the quest.	Quest is active.
In this case we check only Main level. These objectives are achieved in Cave level.	Cave Quest Giver is found. Cave_Discover_C is not found. You checked only "Main" Persistent Level.

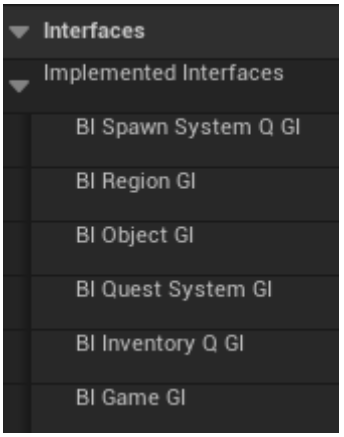
Save and Load

For saving and loading I use Game Instance and Save Game Classes.
GI_QuestSystem is always used (teleportation between levels or saving).
SG_QuestSystem is used to save data to a file.

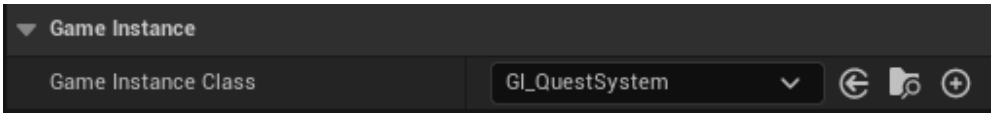
The following info is saved.



Each actor (Managers, Local Quest, Region, Static Object, Spawner) uses its own interface to contact the game instance class. Interfaces are implemented in Game Instance.



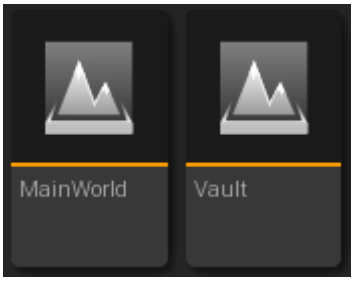
In order to use my Game Instance class in your game, open project settings -> Maps and Modes -> and select Game Instance class. If you are using your own class -> just copy logic.



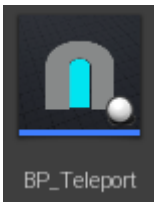
Other Actors (used for demo)

Teleport between levels

Let's create a teleport between two levels: **MainWorld** and **Vault**.

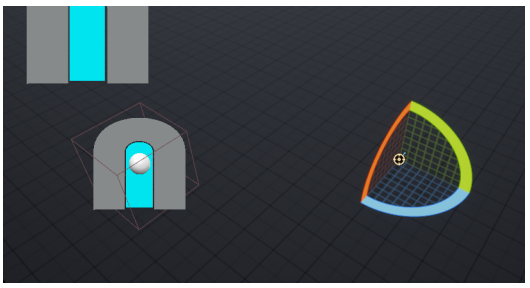


Open MainWorld and Place **BP_Teleport** on the scene.

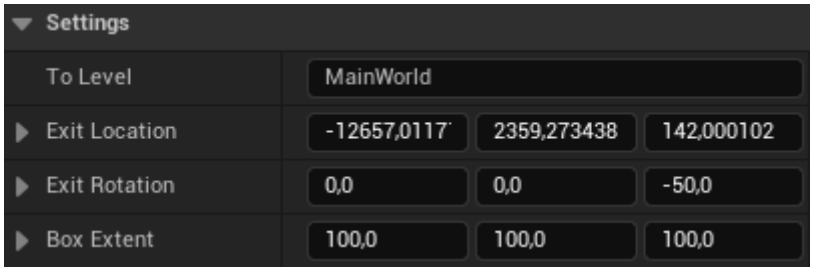


Select it and set settings.

In order to get **ExitLocation** and **ExitRotation**, open Vault, place any actor you want (for example,Target Point) in a place where you want the Player to spawn, and copy Location and Rotation(only Yaw).

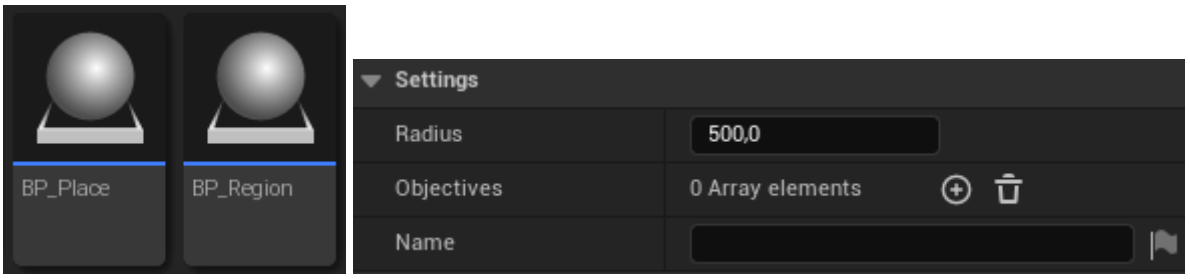


Do the same for BP_Teleport in Vault.



BI_Game_GI interface is implemented in **GameInstance**.
When the player teleports, **TeleportToLevel** interface function is called.
All game progress is saved, ExitLocation and ExitRotation are also saved.
Then a new level is opened, and in Player Controller (Begin Play), **LoadPlayerLocation** interface function is called.

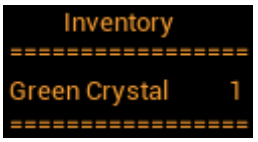
Area



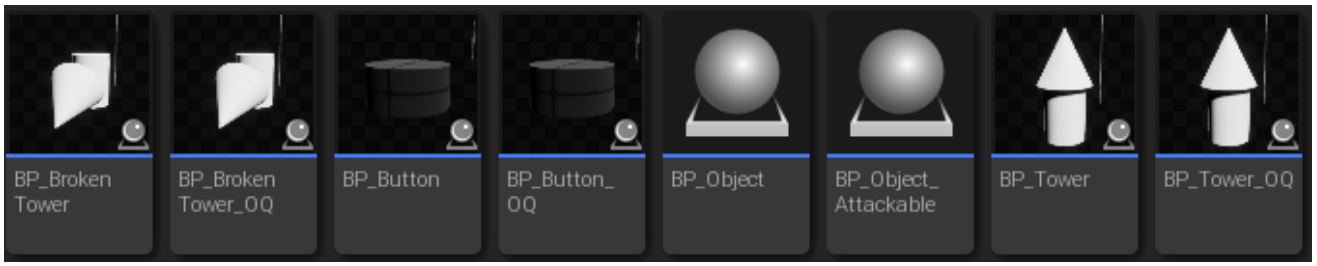
Place is used to complete Custom objective ‘Reach the place’.
Region is used to complete Discover objective.
Region can be discovered. All discovered regions are saved.
These actors are not tracked on Compass or World Map.

Item Giver

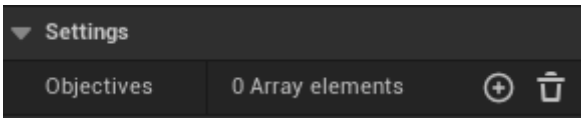
I use very simple inventory for demo. It keeps only the item name and amount.
When the player overlaps the area, the item is added to inventory and inventory data appears on screen.



BP_Object

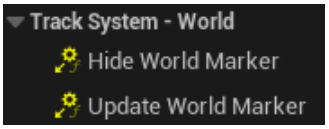


When you place these actors on the scene, set objectives with which the actor will be assigned.



_OQ is interactable or attackable only if the assigned objective is activated.

Interface functions for world markers are implemented. Objective marker appears if one or more assigned objectives are activated.



Activated or Destroyed objects are saved.

Objects variable is created in **BP_Objective**. This is necessary in order not to count the interaction with the same object twice.

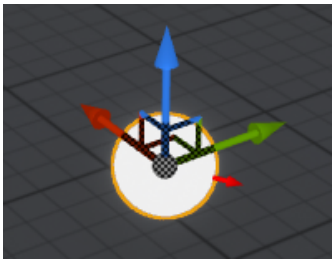
You can test it by interacting with BP_Button. Player can interact with it many times, but progress is counted only one time.

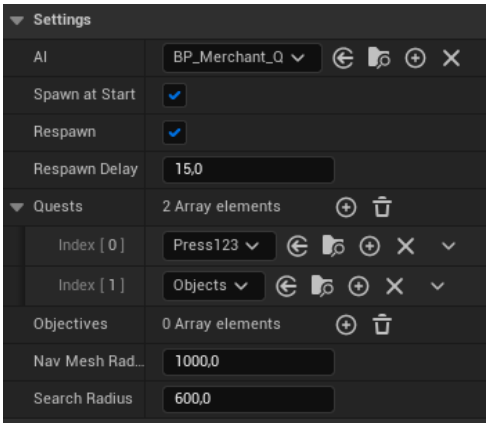
Spawner | Spawn Point | AI

Spawner and **Spawn Point** are taken from my **AI Tools** project.

BP_Spawner and **BP_SpawnPoint** are placed on Sub-Level, if you use level streaming.
When a sub-level is unloaded, Spawner, Spawn Points and AI are unloaded too.
If you use World Partition, these actors are automatically assigned to a grid cell.

If you need to spawn not movable AI, you can set the initial rotation for Spawn Point, using the arrow.

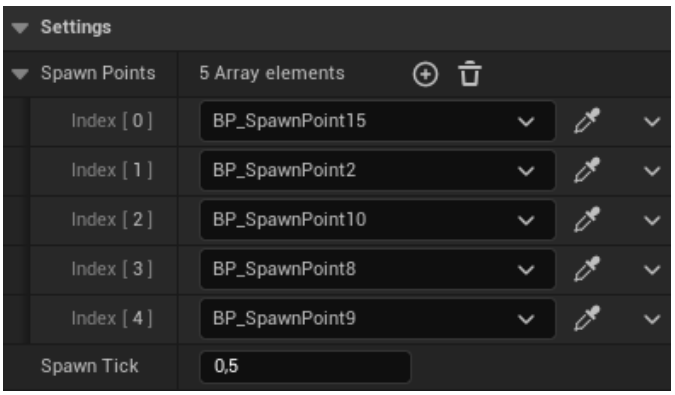




Respawn and **Spawn at Start**
For example, if you need to kill monsters that don't exist in the world, these parameters can be disabled.
When objective is activated, Quest Manager calls **SpawnAI** event. Spawn Point is bound to this event.

By design, AI uses Navigation Invoker features. So you can set the nav mesh radius.

Don't forget to assign the spawn point to the spawner. Also you can set Spawn Tick. It is useful for performance, because AIs will not be spawned in one frame.

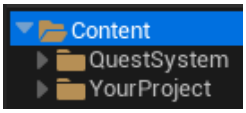


Connect to your Project

HUD

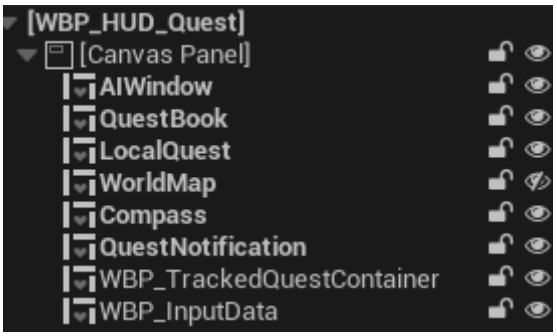
Quest System is asset pack. It can be added to your project from your library in the launcher.

After you add it you need to copy some logic from my main classes (Player Character, Player Controller etc.) to your own.

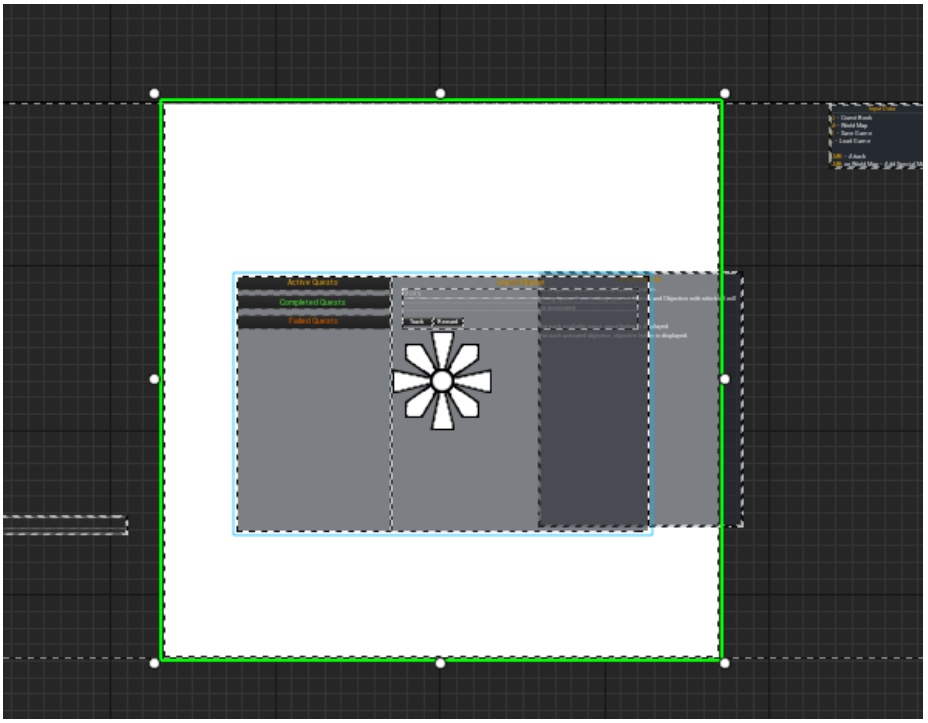


Copy all widgets from my **WBP_HUD_Quest** to your HUD.

If you haven't HUD, just use my one. HUD widget is created in the Player Controller at BeginPlay.

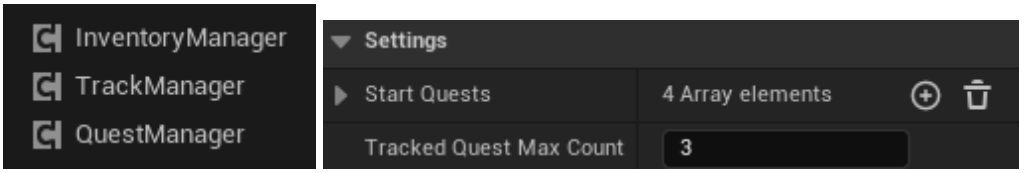


Make sure that World Map is aligned to the center of the widget.

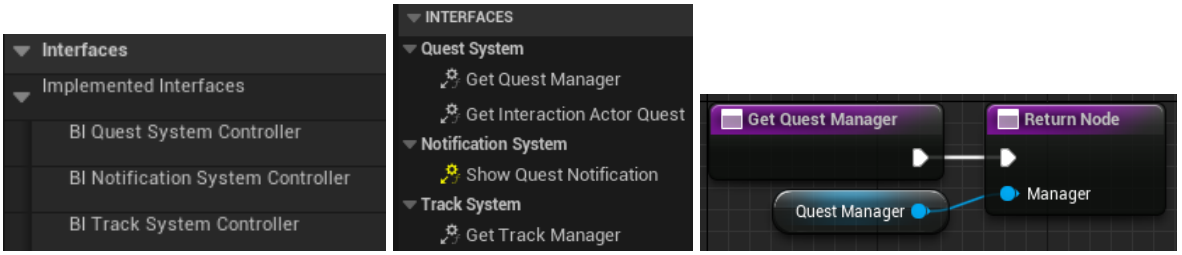


Player Controller

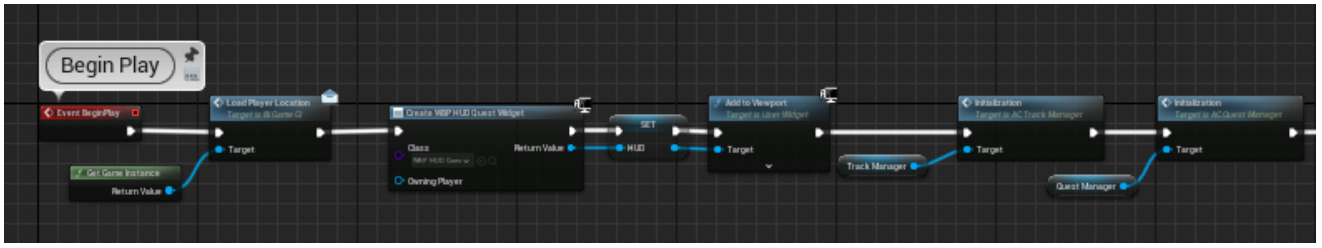
Copy managers with already set parameters. Make sure that StartQuests is not empty.



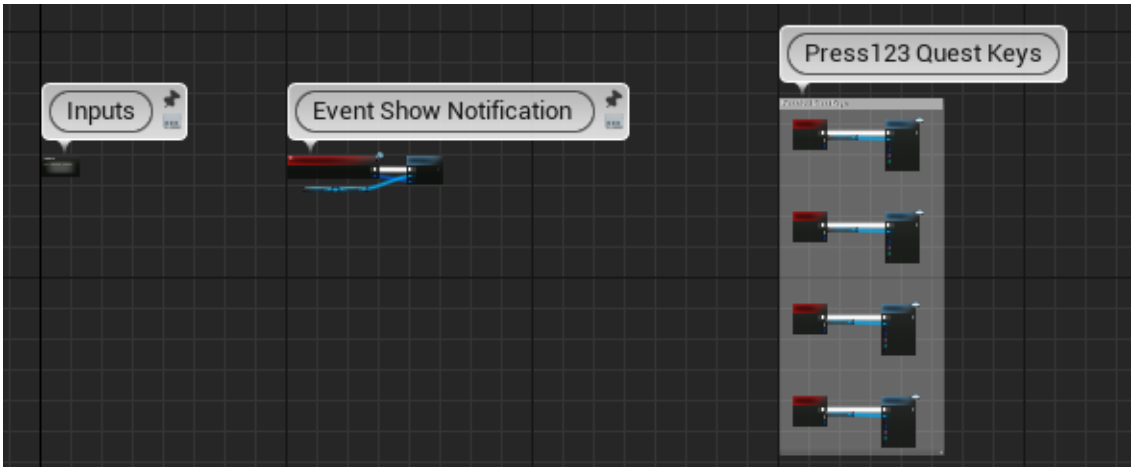
Class Defaults -> Implement Interfaces. Implement all functions as in my controller.

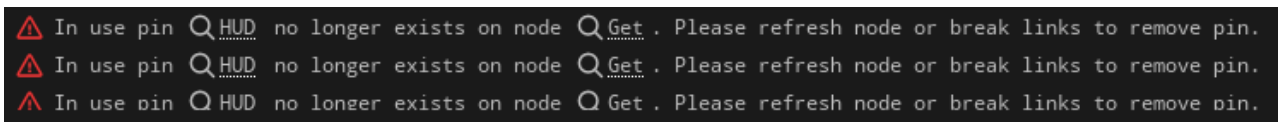


As for **GetInteractionActorQuest**, in my controller you can find **TargetToInteract** variable. In event graph you can find **Check Target In Front** block. That's how I get target.
Copy all logic from Event Graph.
For Begin Play. Managers should be initialized after HUD is created



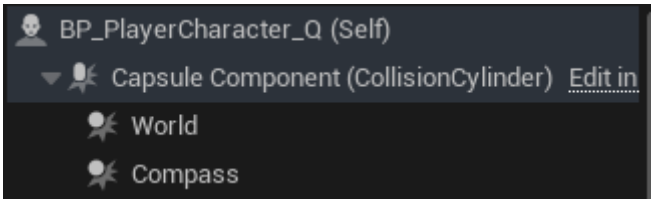
If you use your HUD, you will need to replace references for some events.



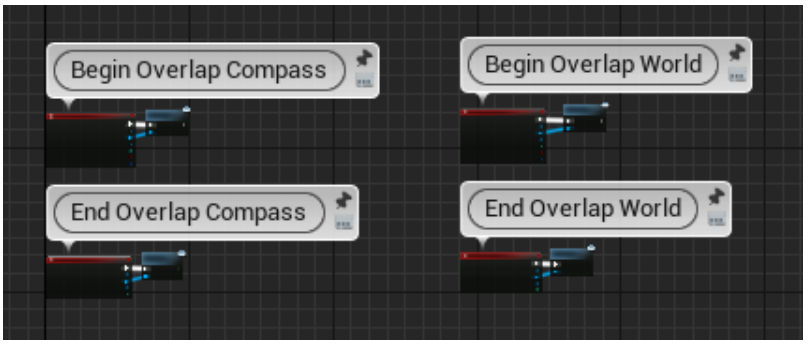


Player Character

Copy 2 sphere collision components (Compass and World) to your Player Character.



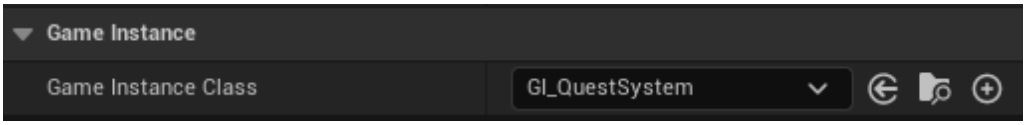
Copy all logic from my Event Graph.



Save Game

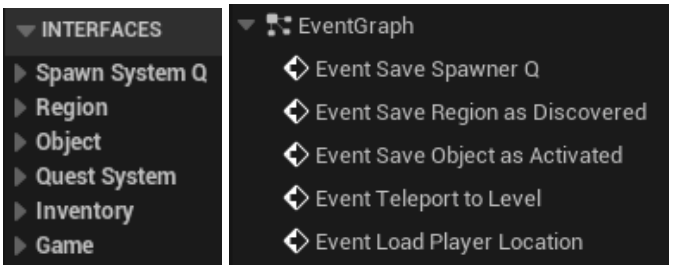
For saving and loading I use Game Instance and Save Game classes.

If you **don't have** your own classes, just open Project Settings -> Maps and Modes -> Set my Game Instance.



If you **have** your own classes, you need copy logic from my classes

Don't forget to implement interfaces in game instances. Each actor (managers, local quest, region, static object, spawner) are using their own interface to contact the game instance class.



Now, out of the box, if you will use my AI and other actors, everything will work.

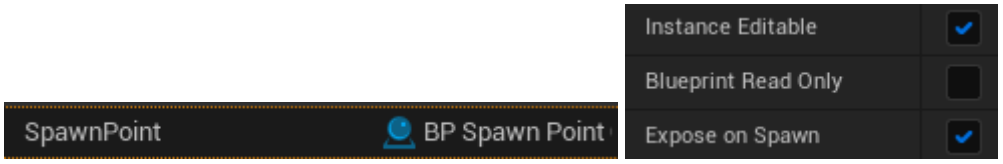
AI

In your project, most likely, you are using your own AI class.

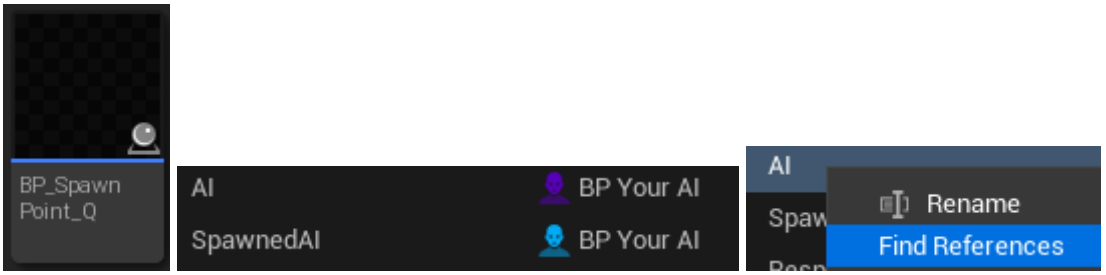
I don't know how exactly you spawn AI in the world. I do it via Spawners and Spawn Points. With them I can spawn AI when I really need it, and save all progress correctly.

You can use my spawn point.

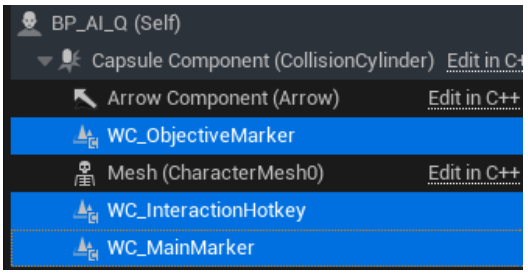
First create **SpawnPoint** variable in your AI.



Then you need to replace AI references in BP_SpawnPoint. Use **FindReferences** to change all references correctly.

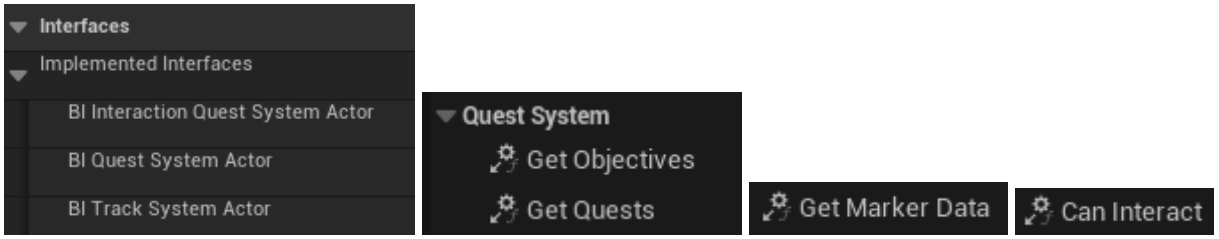


Copy widget components to your parent AI class.

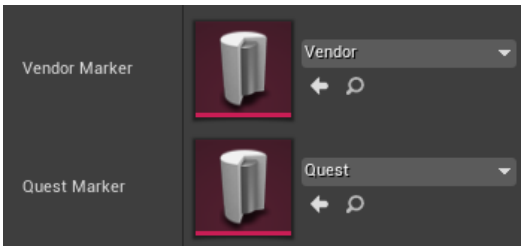


Add and implement interfaces. You can copy all logic for these interfaces from my BP_AI_Q from event graph (including Begin Play).

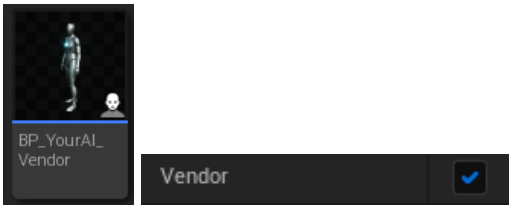
Don't forget to copy logic for functions.



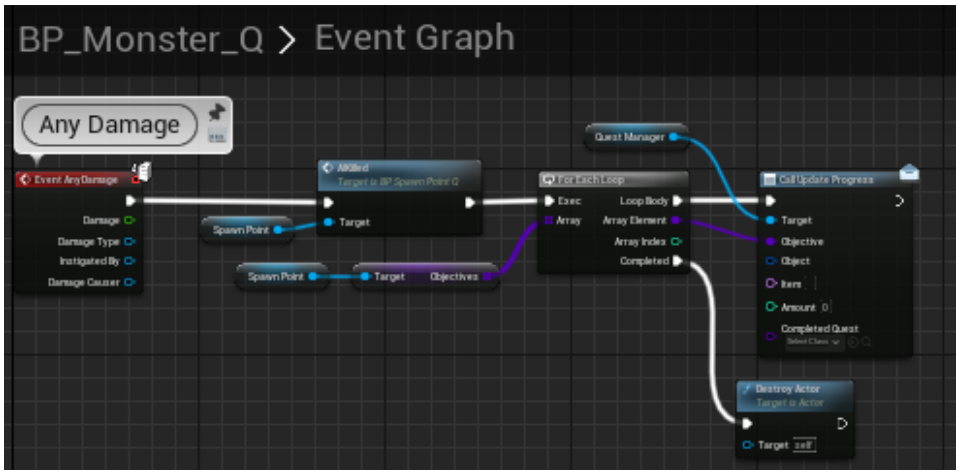
Set default values in settings for your AI.



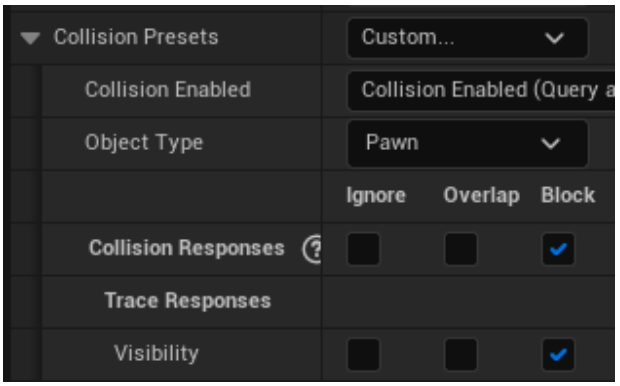
You can create a child class Vendor and set default value for Vendor variable.



For your monster copy logic from my BP_Monster_Q.
When AI is killed.



Make sure that AI capsule collision blocks trace responses by visibility channel. I use raycast by this channel in Player Controller, for checking targets in front of character.

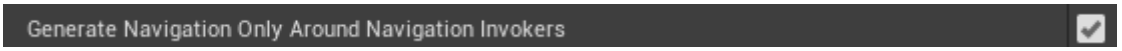


Navigation

If you are working with a large world and you don't want to generate static navmesh around the entire world you can use Navigation Invoker Feature.

Project Settings -> Navigation Mesh

Project Settings -> Navigation System



Don't forget to place nav mesh bound volume on the scene.

Static Objects

In my project I use the interaction interface to interact with any actor. In the Player Controller I get a target to interact and then, when I press R, I interact with the actor via interface.

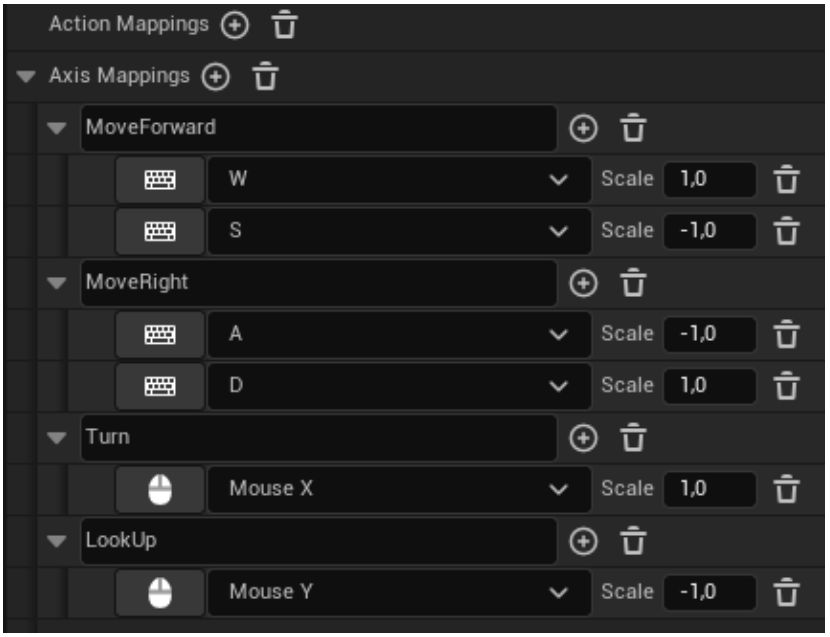
The interface is implemented in BP_Object and its child classes.

So if you use your own interface, just replace it.

These classes are used as examples. You can implement interfaces and copy logic to your own classes.

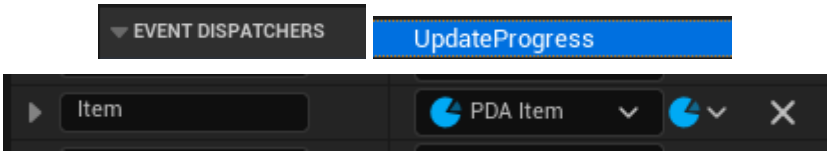
Inputs

Perhaps you connect the system to Blank project.
In this case make sure the inputs are created in project settings.

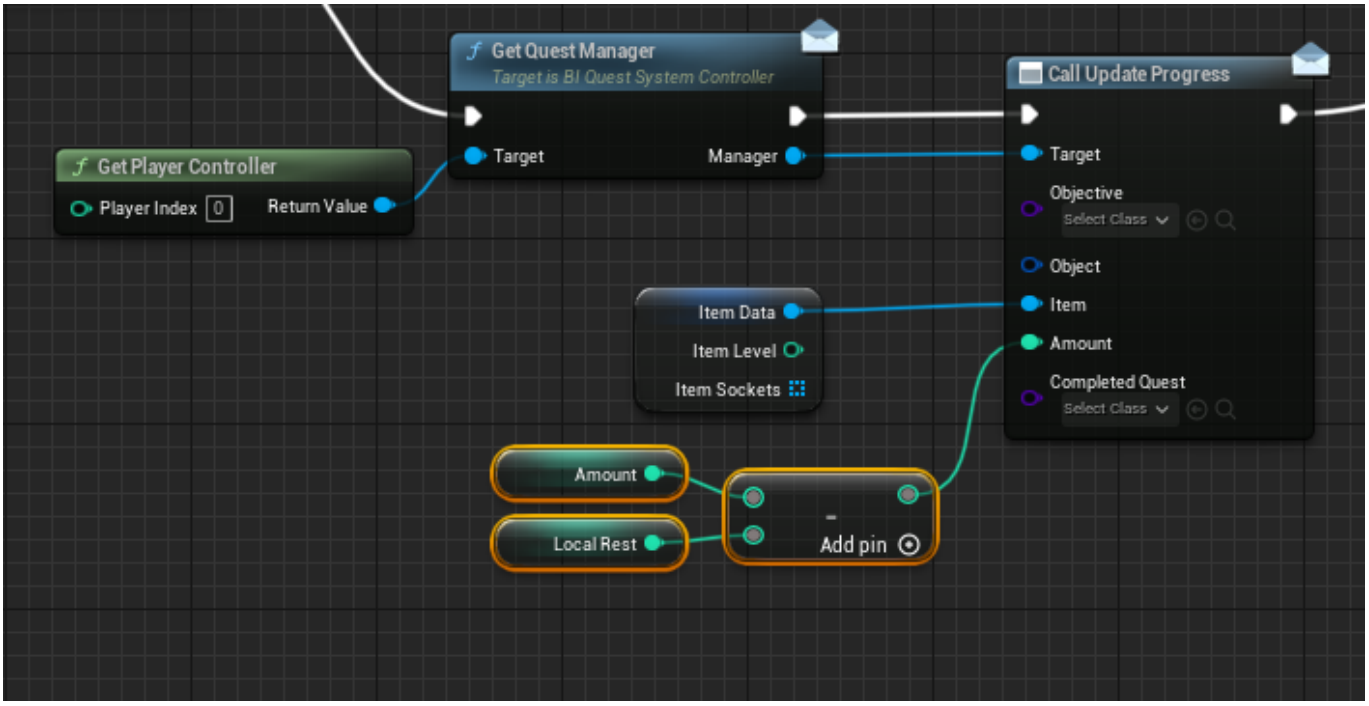


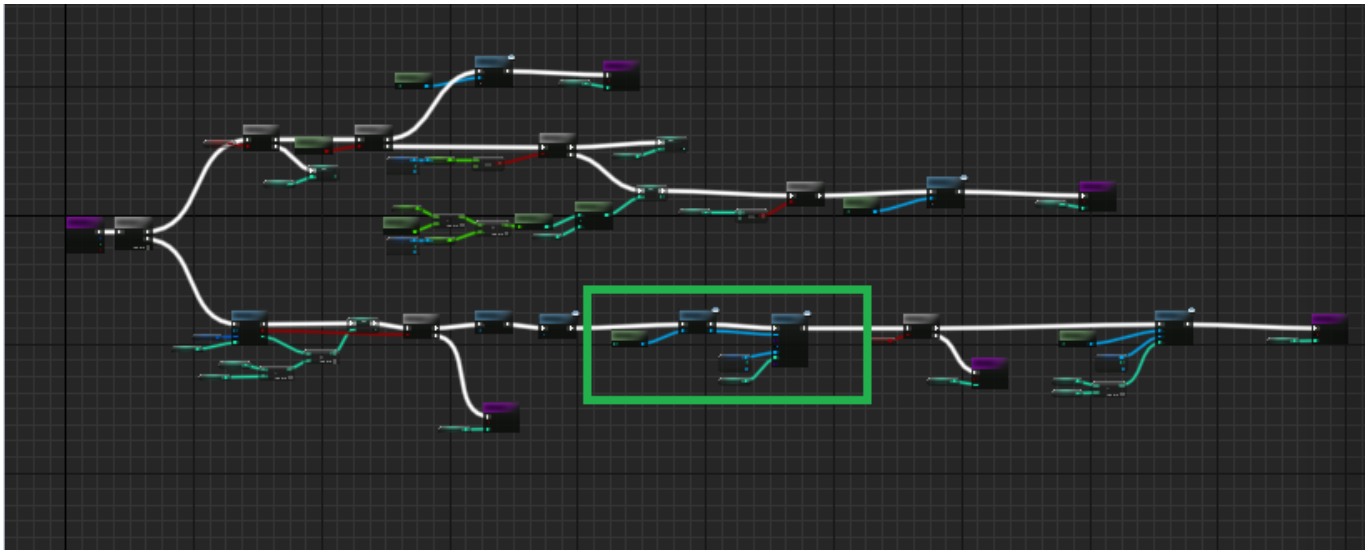
Simple Inventory

If you connect quest system to the project with my inventory system, you need to replace my simple inventory from quest system on powerful inventory.
At first open Quest Manager -> find **UpdateProgress** dispatcher event and change Item variable type to **PDA_Item**.



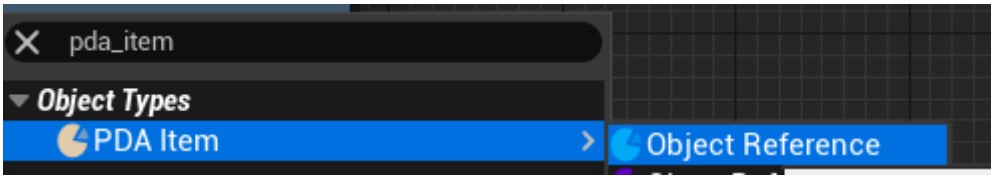
Open **Slot Manager -> TryAddItem** and add the next logic to here.



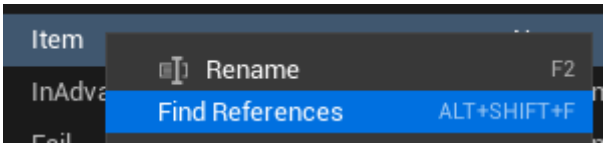


Open **BP_Objective** and find the variable **Item**.

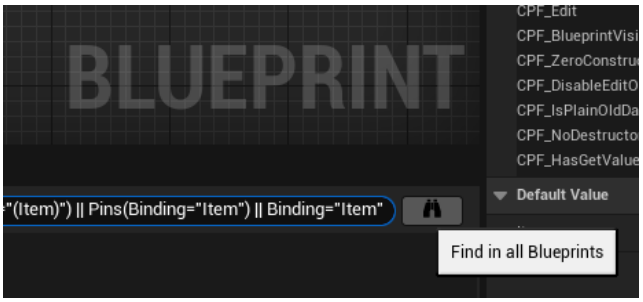
You will have to change variable type to PDA_Item.



Click RMB and select **FindReferences** option.



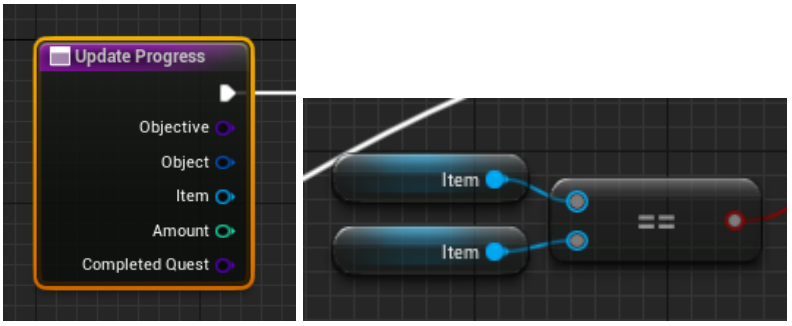
Click on “Binoculars” icon - Find in all Blueprints.



Then replace all found references.

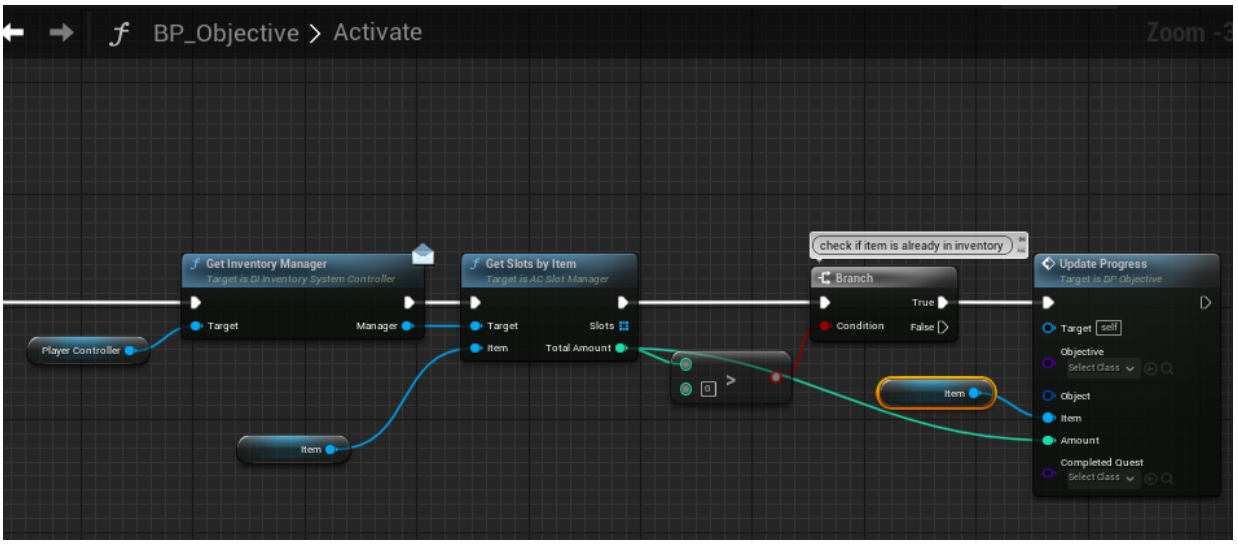
BP_Objective -> Update Progress.

Change variable type and the logic.



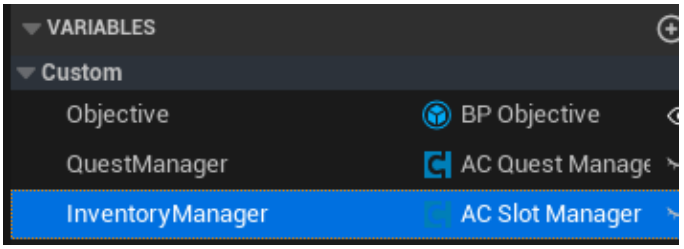
BP_Objective -> Activate.

Use **GetSlotsByItem** function from my powerful inventory system.

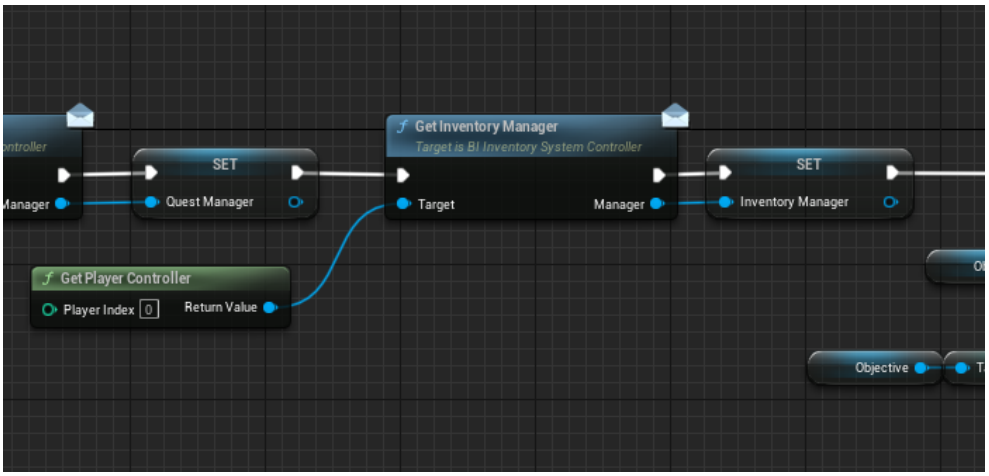


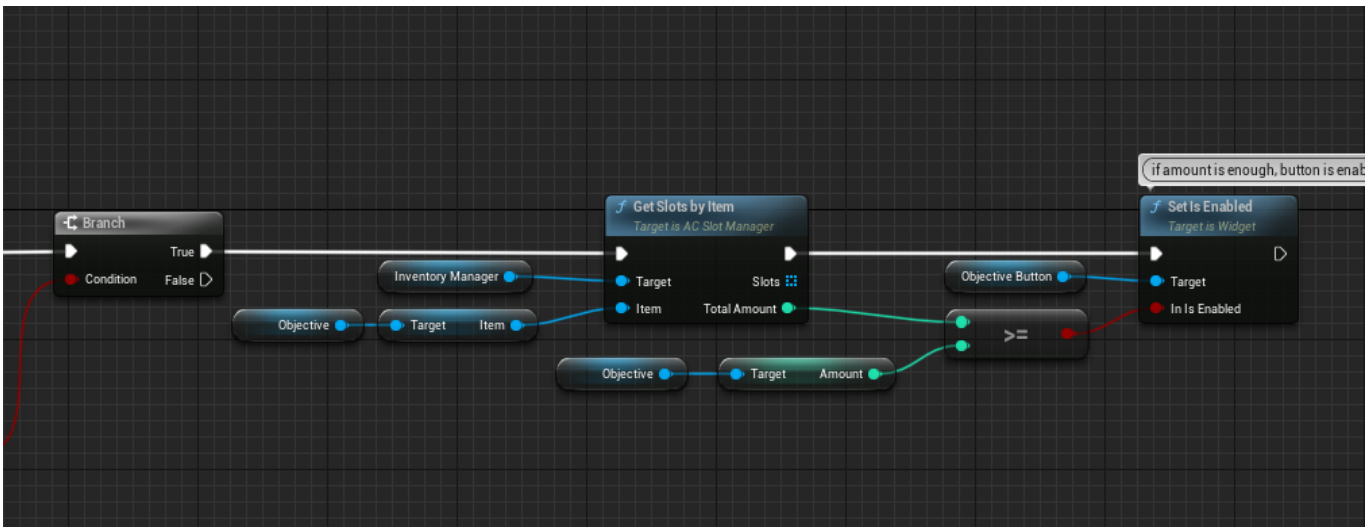
WBP_AIWindow_ObjectiveButton.

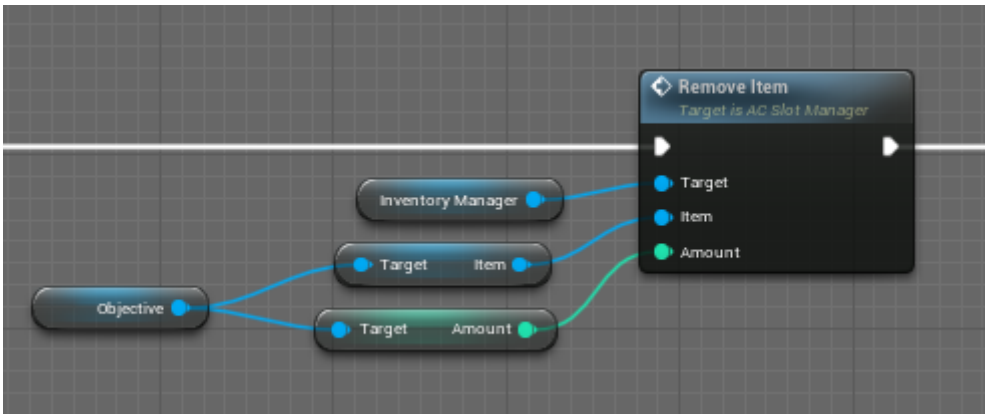
Select InventoryManager variable and change type to **AC_SlotManager**.

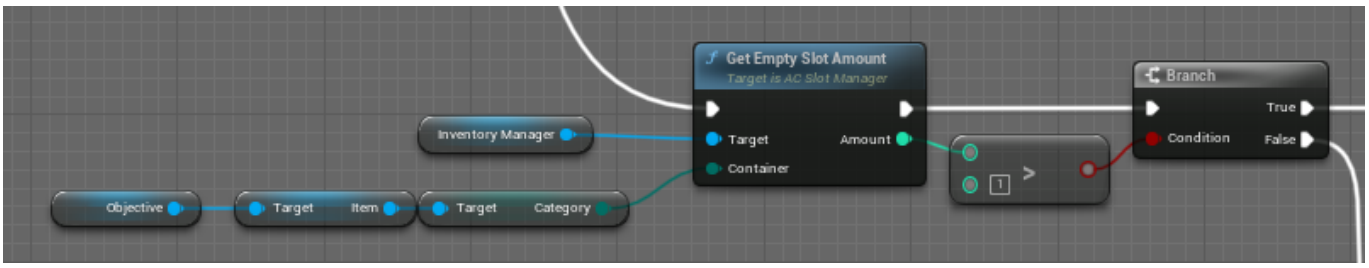


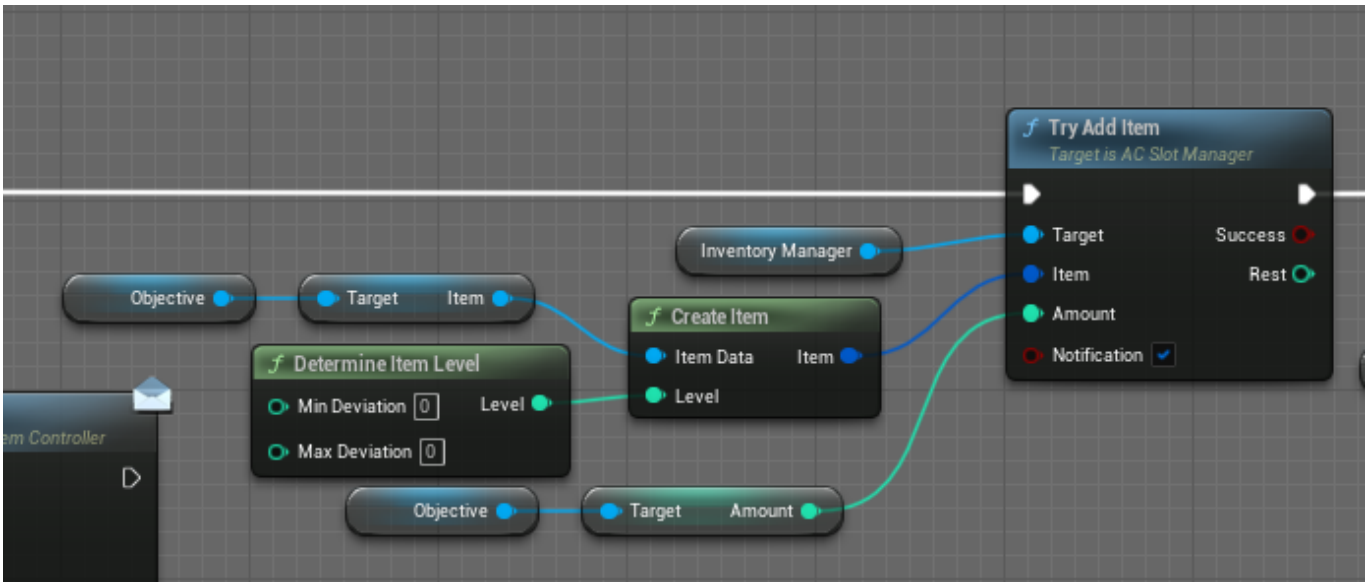
Change the logic.


SET
Quest Manager
Get Inventory Manager
Target is BI Inventory System Controller
Target
Manager
SET
Inventory Manager
Get Player Controller
Player Index
0
Return Value
Objective


if amount is enough, button is enab
Branch
True
Condition
False
Get Slots by Item
Target is AC Slot Manager
Target
Item
Slots
Total Amount
Inventory Manager
Objective
Target
Item
Objective
Target
Amount
Objective Button
>=
Set Is Enabled
Target is Widget
Target
In Is Enabled


Remove Item
Target is AC Slot Manager
Target
Item
Amount
Inventory Manager
Target
Item
Objective
Target
Amount


Get Empty Slot Amount
Target is AC Slot Manager
Target
Container
Amount
Inventory Manager
Objective
Target
Item
Target
Category
1
>
Branch
True
Condition
False

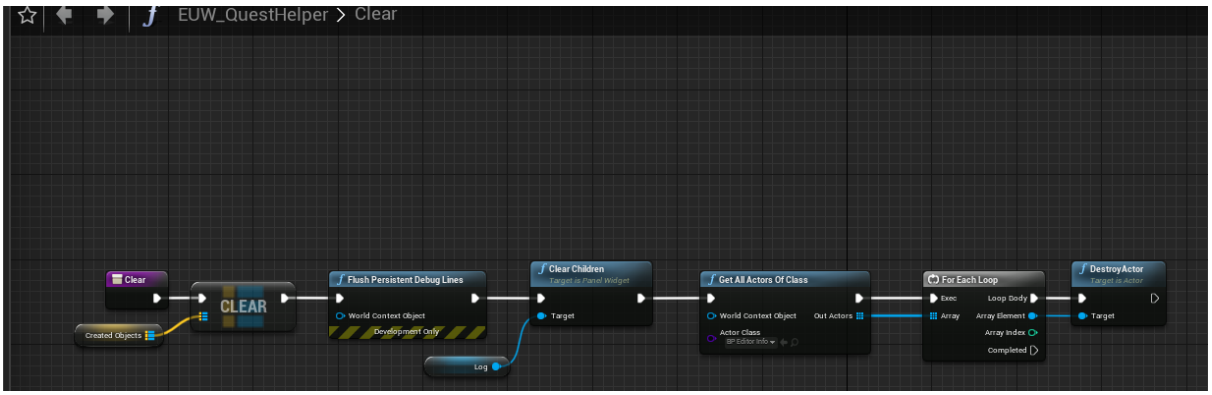

Try Add Item
Target is AC Slot Manager
Target
Item
Amount
Notification
Success
Rest
Inventory Manager
Objective
Target
Item
Create Item
Item Data
Item
Level
Determine Item Level
Min Deviation
0
Max Deviation
0
Level
Objective
Target
Amount

Change Log

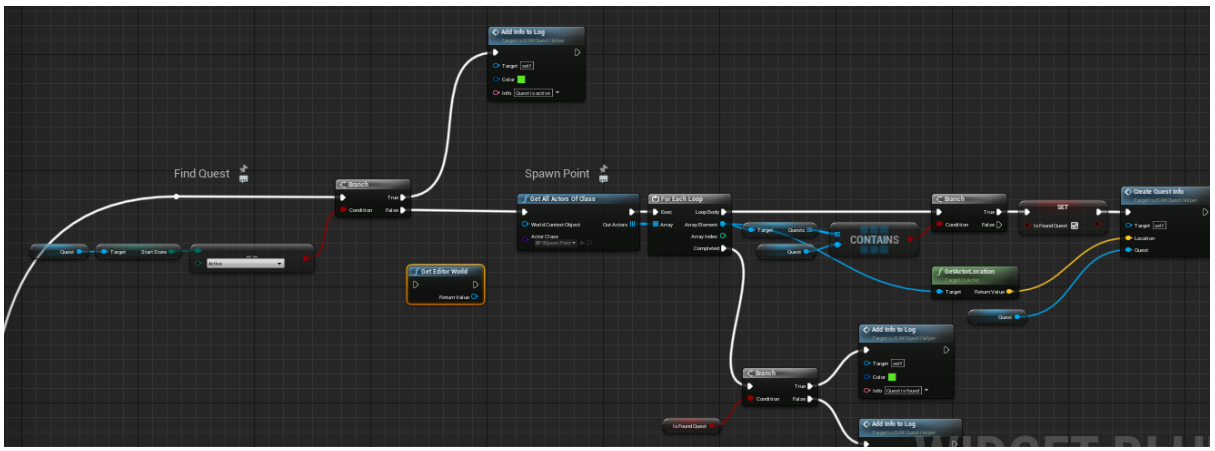
1.1

13 December 2020

- made minor changes in EUW_QuestHelper in order to open project correctly in 4.26



“Destroy actor” node is replaced



“Get Editor World” node is removed from “Check Quest” function

1.2

17 December 2020

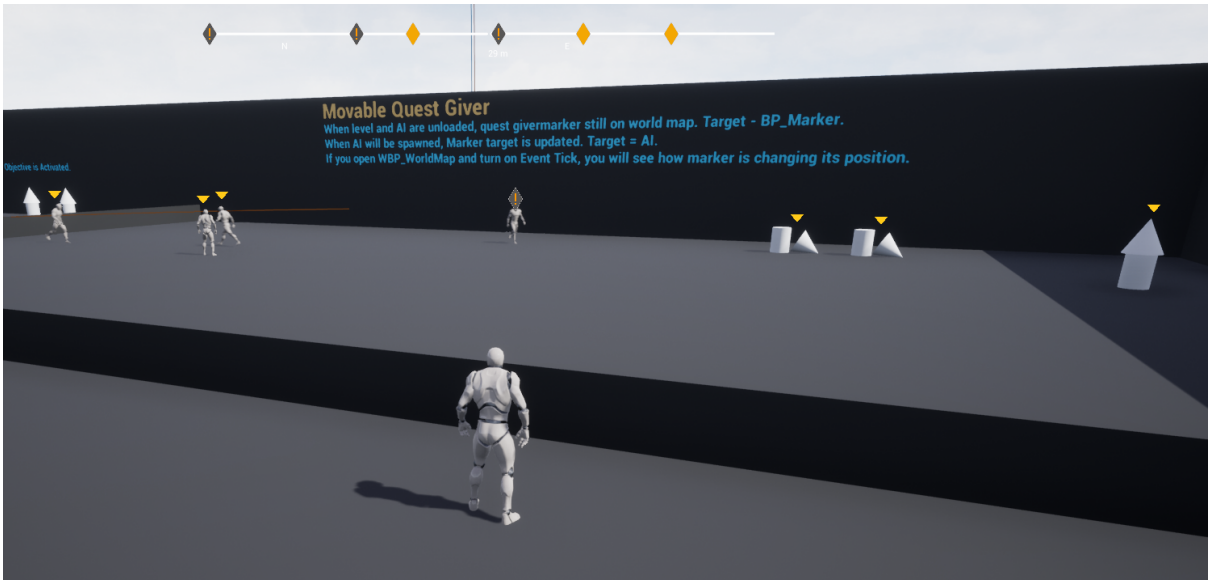
- I slightly improved the mechanic “reward”
    - now reward will always be automatically added
    - if any reward have not been received by Player, quest will not be removed from Quest Book, and “Reward” button appears
    - <reward manually> variable is removed from PDA_Item
- There was a visual bug: If you complete all objectives (discover, interact, destroy) and only after that you take a quest, the quest is completed, but not removed from the Quest Book. This is fixed -> quest will be removed from quest book.

Changes were made in BP_Quest (“Give Reward”, “Active”), WBP_QuestBookInfo (“Update”).

1.3

24 February 2021

- refactoring for track system
- Track system is improved
    - world markers are added
- save system is improved
    - player location is saved
    - killed AI will not spawned anymore, if Respawn = false

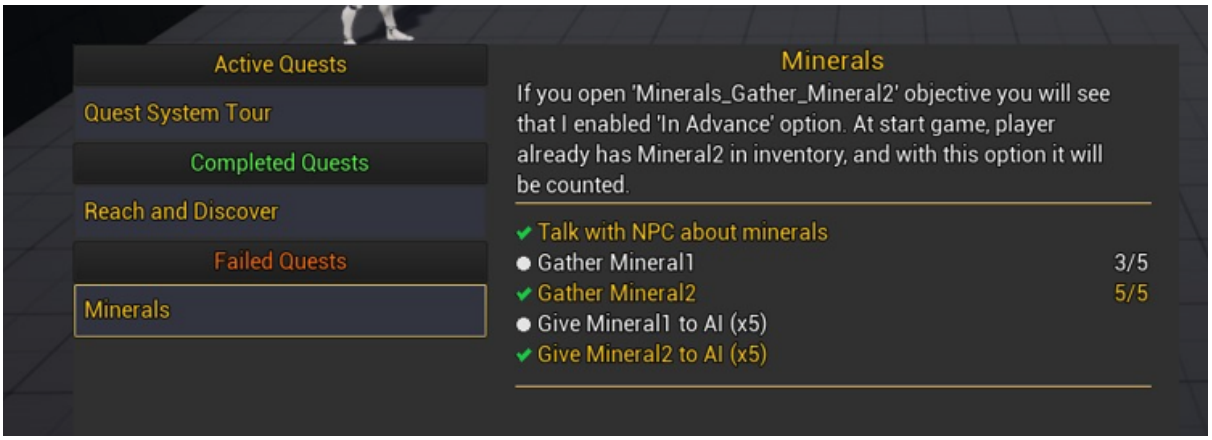


2.0

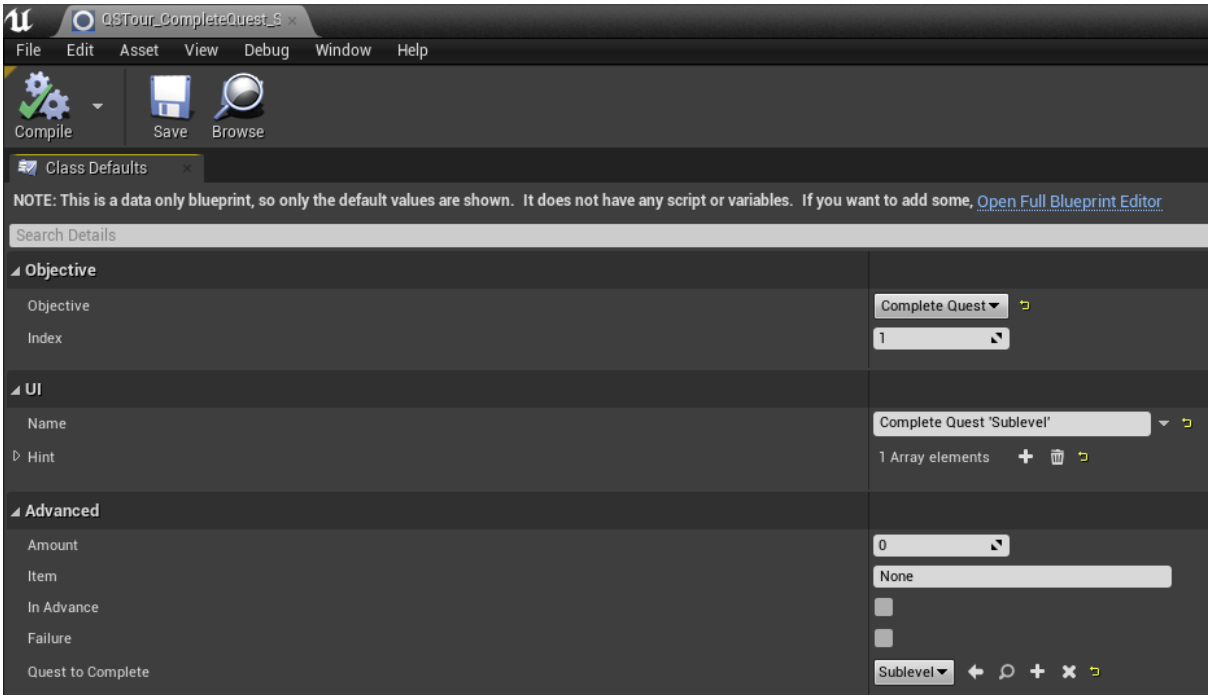
15 July 2021

- refactoring over the whole project
    - all logic is reworked
    - now Quest and Objective are object classes

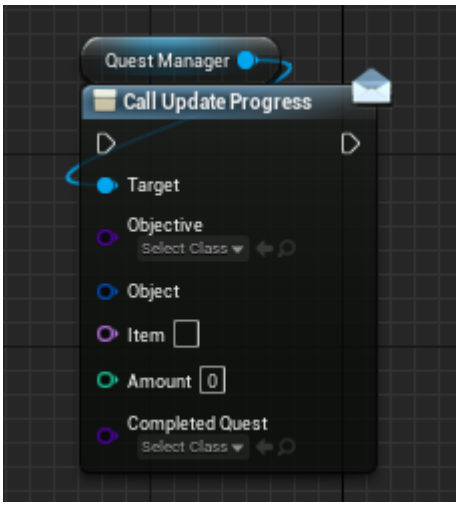
- documentation is updated
- now Quest Manager saves completed quests
    - completed quests are displayed in Quest Book

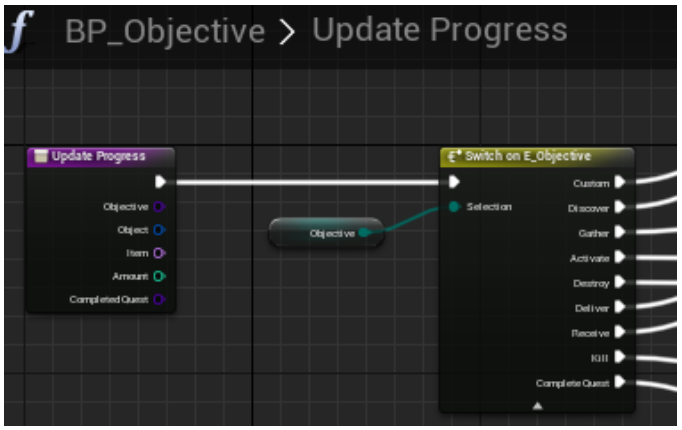


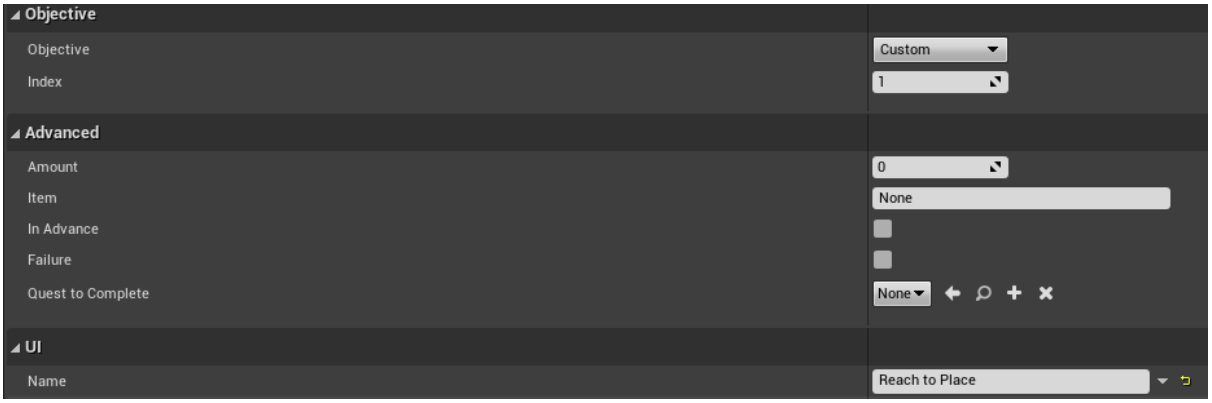
- new quest objective - ‘Complete Quest’



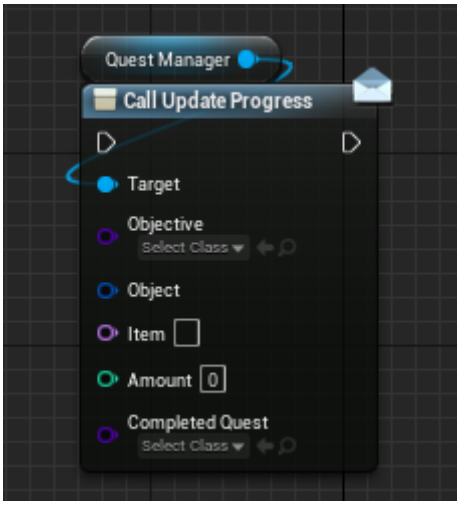
- new quest objective - ‘Custom’
    - objective will be completed automatically after you call UpdateProgress and pass Objective







- now in order to update progress you need to call Dispatcher Event, more in documentation in Objective section



2.1

15 July 2021

- Widget components **WC_MainMarker** and **WC_ObjectiveMarker** are created
  Event Graph is cleaner now, since logic for widgets is created in components.
- **ActiveState** function is overridden for Local Quest
  Function for this class is called when loading the game. We don't need to remove quest marker as we do for common quest.
- Minimal visual changes in widgets (background, text color, button style)

2.2

31 January 2022

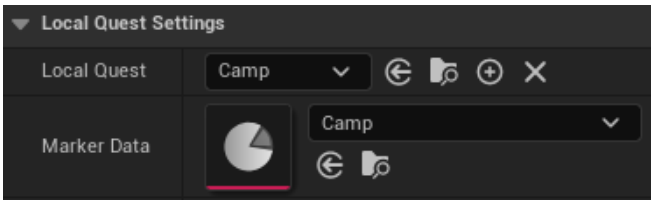
- new feature - **Special Markers**
  Markers can be added by clicking LMB on World Map. Marker is always visible on World Map and in World. Distance is shown. MarkerMaxAmount is specified in Track Manager.
  Video

2.3

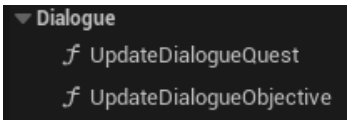
11 May 2022

- Now multiple quests can be **tracked** at the same time
  The appropriate parameter is added to Quest Manager. If current count is max, the first quest is untracked. ***IsTracked*** *variable is added to BP_Quest.*

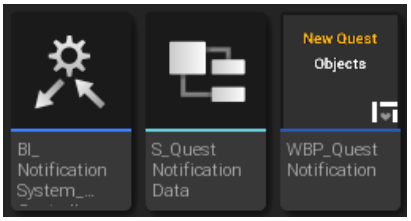
- Events for Quest and Objective has been added
  See ***Events*** *section.*
- Quest states ‘Success’ and ‘Failure’ have been changed to ‘Completed’ and ‘Failed’
- Now **BP_LocalQuest_Placer** is a target for track system
  You don’t need to use BP_Marker to track local quest location. Track System interface is implemented in BP_LocalQuest_Placer. Just place actor on the scene and set marker data.



- **Quest Manager** logic has been optimized
  All quests and objectives are keeped in Map variables now. It is done in order for faster finding needed entities. ForEachLoop is not good for optimization, if there are too many quests or objectives in the game.
- Two functions have been created in Quest Manager
  It can be useful when you use a Dialogue System. Since there can be only one Dialogue at a time, you can get current quest or objective that is assigned to dialogue.



- **Track Manager** logic has been optimized and improved
  Now Track Manager can work not only with BP_Marker, but with any actor class you want.
  All checks are moved and implemented in actor classes. Track Manager works with actors only with interface.
  Now if you want the actor to be a target for Track System you can implement interface BI_TrackSystem_Actor in this actor.
- Now **adding/removing Special Marker** works only with LMB
- Notification logic is slightly reworked
  Ofc currently you can send only Quest notifications. Just use interface BI_NotificationSystem_Controller.



- Teleport has been improved
  ExitLocation *and* ***Exit Rotation*** *parameters have been created. Now teleport hasn’t hard reference to Game Instance and uses an interface.*
- **BI_QuestSystem_Controller** and **BI_TrackSystem_Controller** have been added to Player Controller
  Now you don’t need to use GetComponentByClass. You can get a manager using the interface. This is for optimization.