Quest Editor Plugin documentation

Contents

1. General information

1.1 What is the Quest Editor Plugin?

Quest Editor Plugin is a powerful system that is designed to create and manage quests and dialogs for your NPCs. It is very easy to customize and expand, making it suitable for any type of game.

Each Quest or Dialog consist of a network of interconnected nodes, called tasks, that you can create and customize for your particular game, making it very versatile. The tasks are easy to implement due to having many overridable functions designed for each state of the process. Some general tasks are provided such as the timer task, that allows to make timed quests.

The system is made entirely on C++ for a better performance, and exposed to blueprints to facilitate its use. Plenty of functions and event dispatchers are provided to make sure that the implementation of the system is as easy and powerful as possible and that it can be done entirely in blueprints.

The custom Quest Editor (Editor widget) is useful to create and debug quests and dialogs for NPCs. This tool is highly customizable making it incredibly valuable for faster iteration and debugging.

All the quests, dialogs and NPCs information are stored in datatables. The system supports any number of datatables, so you can organize the information to your needs.

The system supports single and multiplayer games. It has a prediction system to reduce the latency feel in the client side.

Quests and dialogs can have multiple starts and ends (even involving different NPCs) and there is no limit in the way the nodes are connected. Even when the system is designed to have “talk features” with NPCs integrated directly into the quests, it is possible to make quests without any form of dialog.

The plugin has a save game implementation, with several options, that allows to save the progress for players in game.

Back to top.

1.2 Technical details

More than 30 C++ classes and around 10k lines of code distributed in two modules, one runtime and one editor.

More than 100 blueprints classes.

Heavily exposed to blueprints, the system does not require C++ knowledge to implement or use.

Several premade tasks of general purpose, including Timer task, reaching location, conditions, etc.

Custom Quest Editor, to create quests, dialogs and manage NPCs.

Save game system for Quest Editor.

Save game system for Gameplay (Player progress). Supports multiple player saves.

Supports multiplayer.

Includes examples of Quests, Dialogs, Blueprints, Datatables, Game UI and Tasks, for easier setup and understanding of the system.

Back to top.

1.3 Creator's note

I will continue to develop, update and optimize this project as long as I can.

I am happy to answer any questions you might have about the system.

If you have features or functions you would like to have added to the system, or you find bugs (let's hope not) or you want to request tutorials on certain subjects, just let me know. Any feedback is appreciated.

If you acquired the system, don't forget to leave a review in the marketplace. This helps a lot. 😊

Thank you for your support.

Here are some links:

Discord: https://discordapp.com/invite/WmQ3pDR
Youtube: https://www.youtube.com/channel/UCkQVJmcV-szXTIySHz4CILw?
Twitch: https://www.twitch.tv/alfffffffffff

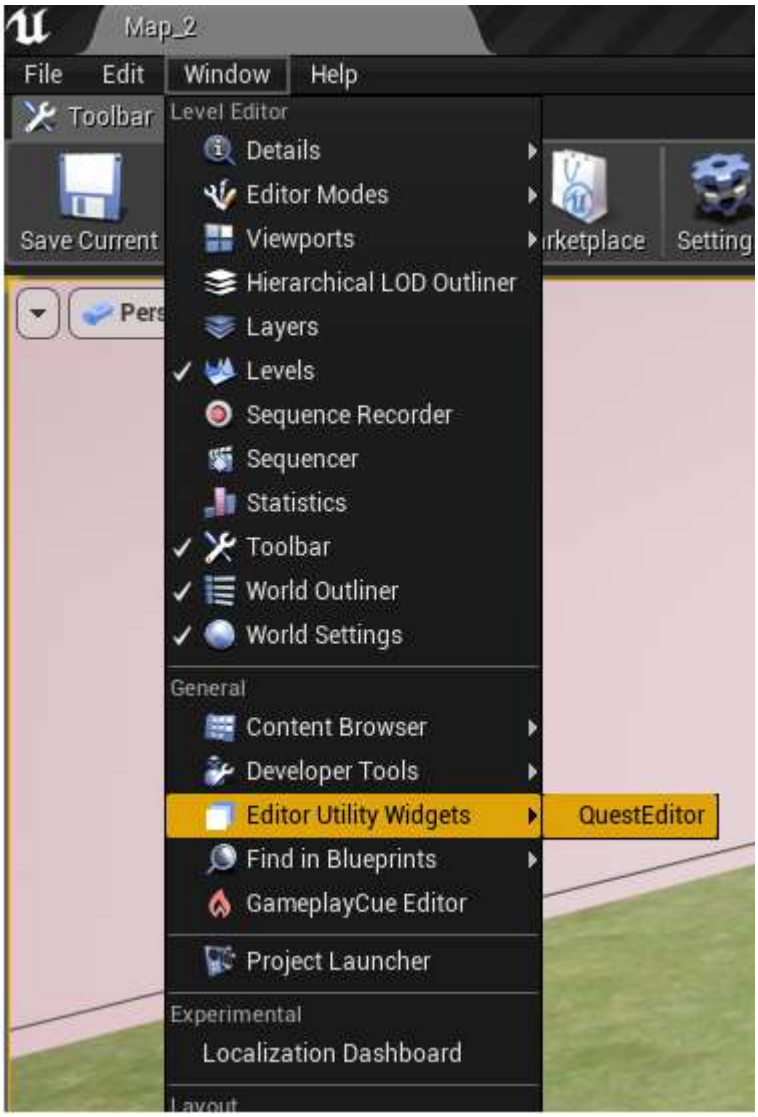
This documentation is subjected to changes. I will keep the link updated on the discord, marketplace and plugin. Make sure you always have the latest version.

Back to top.

2. Quest Editor

The Quest Editor is an editor widget designed for this particular plugin to facilitate the design and implementation of quests and dialogs for NPCs. It allows for fast iteration and debugging. It works directly with the datatables that we created, allowing to load data from them and also editing and saving data into the tables.

As any other editor utility widget, it can be accessed in the windows menu.



The editor is very easy to customize, designed intentionally this way, to adapt better to any system implementation and also to permit the users to adapt the tool to their workflow. We will abord the customization of the editor later in this documentation.

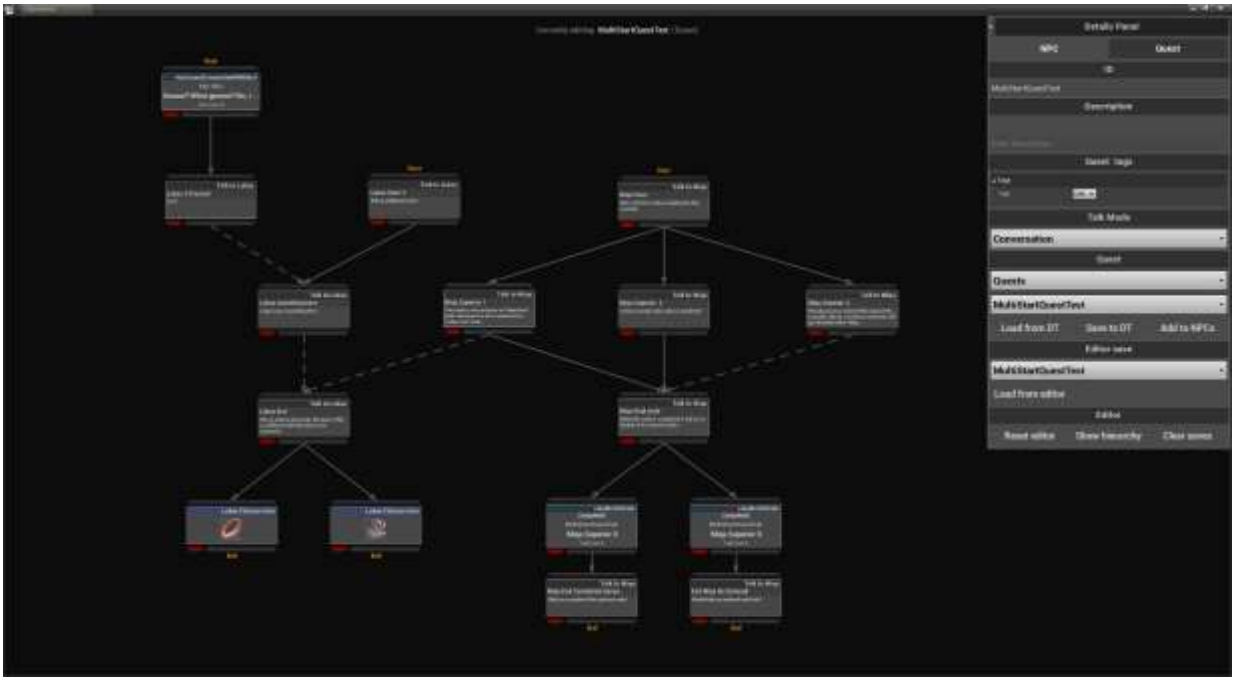
The Quest Editor has its own save system, to avoid any data loss or if you want to go back to an older version of a quest or dialog. The quest editor will also save its current state, so if closed or interrupted, you can continue from the point where you left. The Quest Editor system file will be saved along with all the save games files, in the “SaveGame” folder.

The editor allows to configure also the NPCs, setting their dialogs and quests directly inside the editor.

The editor will modify datatables, but it will never apply those changes. The save of the datatables must be done manually in the same way that is used to normally save any assets in the editor. This allows the user to cancel the saving at last moment, if he changes his mind.

Back to top.

2.1 Viewport



This is an image showing a quest being edited in the Quest Editor.

The viewport contains all the nodes widgets and is the most important part to design quests and dialogs.

It will allow you to place new nodes and connect them. It will allow you to edit the node connections and select nodes to edit.

It displays useful information such as the current editor mode and errors or messages when its needed.

Back to top.

2.1.1 Editing Links

The links can be edited in the viewport. You can interact with the links moving your mouse close to them.

Changing the type of the link or deleting links is performed with the mouse clicks. The node widgets have buttons for creating links between them. Superior buttons can only connect to inferior buttons and vice versa in normal connections

The superior and inferior buttons are for creating normal links and the red button on the bottom of the widget for creating disabling links. Red buttons can connect to any other red button as long as is not from the same node.

Back to top.

2.1.2 Key shortcuts and controls

Action	Key	Description
Add Task	Ctrl + Mouse left click	Add a task node in mouse position
Add Talk Task	Alt + Mouse right click	Add a talk task node in mouse position
Move up	Up arrow	Move selected nodes up. If no node is selected moves the viewport.
Move down	Down arrow	Move selected nodes down. If no node is selected moves the viewport.
Move left	Left arrow	Move selected nodes left. If no node is selected moves the viewport.
Move right	Right arrow	Move selected nodes right. If no node is selected moves the viewport.
Zoom in	+	Zooms in the viewport.
Zoom out	-	Zooms out the viewport
Align nodes up	W	Align selected nodes to higher node position
Align nodes down	S	Align selected nodes to lower node position
Align nodes left	A	Align selected nodes to the left
Align nodes right	D	Align selected nodes to the right
Align nodes up and even	Alt + W	Align selected nodes to higher node position and even positions on X axis.
Align nodes down and even	Alt +S	Align selected nodes to lower node position and even positions on X axis.
Align nodes left and even	Alt +A	Align selected nodes to the left and even positions on Y axis.
Align nodes right and even	Alt +D	Align selected nodes to the right and even positions on Y axis.
Align nodes to grid	Q	Align selected nodes to an invisible grid
Even X distribution	F	Distributes selected nodes evenly across X axis
Even Y distribution	R	Distributes selected nodes evenly across Y axis
Delete nodes	Delete	Deletes all selected nodes

Iterate start nodes	Home	Iterates through all start nodes
Iterate end nodes	End	Iterates through all end nodes
Iterate nodes forward	Page Up	Iterates through all nodes from lower to higher ID and from Tasks to Talk Tasks
Iterate nodes backward	Page Down	Iterates through all nodes from higher to lower ID and from Talk Tasks to Tasks
Find selected node	Space	Center viewport in the selected node. For multi node selection uses first one.
Center viewport	P	Resets viewport to the central (initial) position
Reset zoom	L	Resets the zoom of the viewport
Undo	Ctrl + Z	Undo last action
Redo	Ctrl + Y	Redo last action
Copy nodes	Ctrl + C	Copy selected nodes
Cut nodes	Ctrl + X	Cut selected nodes
Paste nodes	Ctrl + V	Paste selection
Connect Selected Nodes	C	Connects all selected nodes, like a chain, with the higher ones being superior nodes.
Connect Selected Nodes as Grid	V	Connects all selected nodes, based on the grid. All nodes with higher grid level will be superior nodes of immediate lower ones.
Toggle Selected Connections Types	Z	Toggles all connections types between selected nodes
Mutually Disable Nodes	M	Mutually disables all selected nodes
Remove Selected Nodes Connections	X	Remove all the connections of selected nodes.
Remove Selected Nodes Disable Connections	N	Remove all the disable connections of selected nodes.

Node selection

To select nodes in the viewport, you can click them or drag, while drawing a selecting rectangle over the nodes you want to select.

You can also use ctrl key while clicking to add or remove nodes from selection or use shift while drawing a selecting rectangle to add nodes to selection without losing any of the already selected ones.

Doble clicking a Talk Task will select all Talk Tasks of the same conversation that contains the clicked node.

Doble clicking a Task will select all Tasks of the same type of the one clicked.

Drag and drop nodes

You can drag and drop all selected nodes using the left click of the mouse to drag and moving the mouse around.

Moving the viewport

Holding right mouse click down and moving the mouse will displace the viewport, allowing you to navigate and reposition your quest or dialog for more comfortable inspection of particular nodes.

Creating links

The nodes have buttons designed for creating links between them. The yellow buttons are for creating normal links and the red ones for creating disabling links.

A link will be shown in the UI to make it clearer while connecting nodes. If right mouse click is pressed while linking nodes, the operation will be cancelled.

Changing link types

Left clicking a link will switch between its types. (Required and Optional), for normal links.

Swapping links

You can swap links and disabled links from a node to another, holding control while clicking the corresponding button.

Deleting links:

Right clicking any link will remove that connection.

Removing all links of a node connector

If you left click a button for creating nodes while holding down alt, all connections to that particular button will be destroyed.

Zoom in and out

You can zoom in and out using the mouse scroll wheel. This will also move the viewport towards the direction of the mouse.

Back to top.

2.2 Details Panel

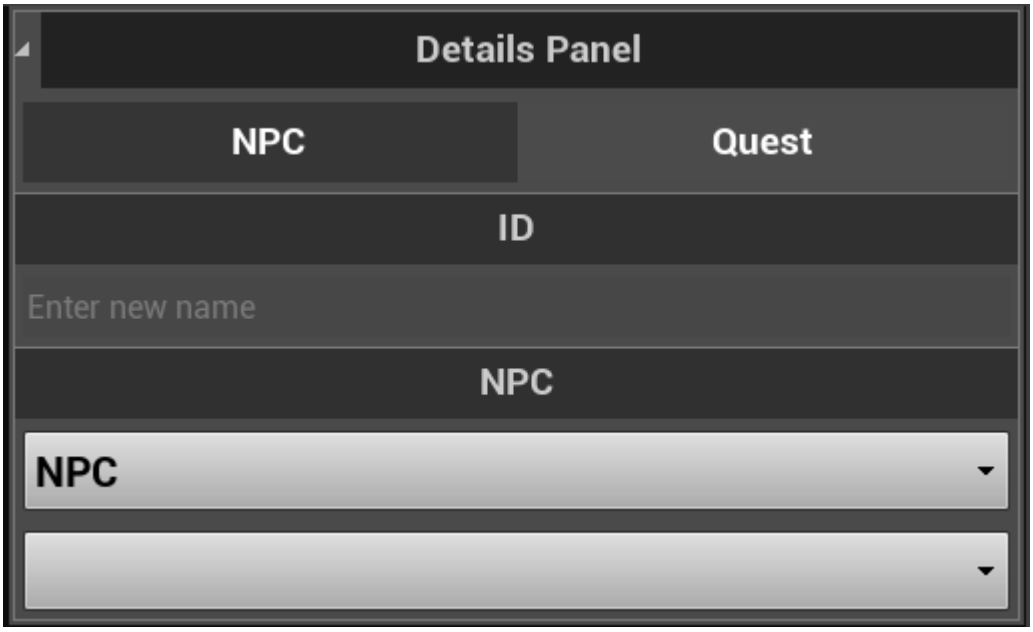
The Details Panel is located at the top right of the Quest Editor screen. This is the widget that will help you edit all the variables. This panel has two very different modes that are explained below.

Back to top.

2.2.1 Editing NPCs

The first mode of the Details Panel is the NPC. This section will help you manage and create NPCs, assigning them Names, Quest and Dialogs.

This is how the NPC mode looks like at first glance.

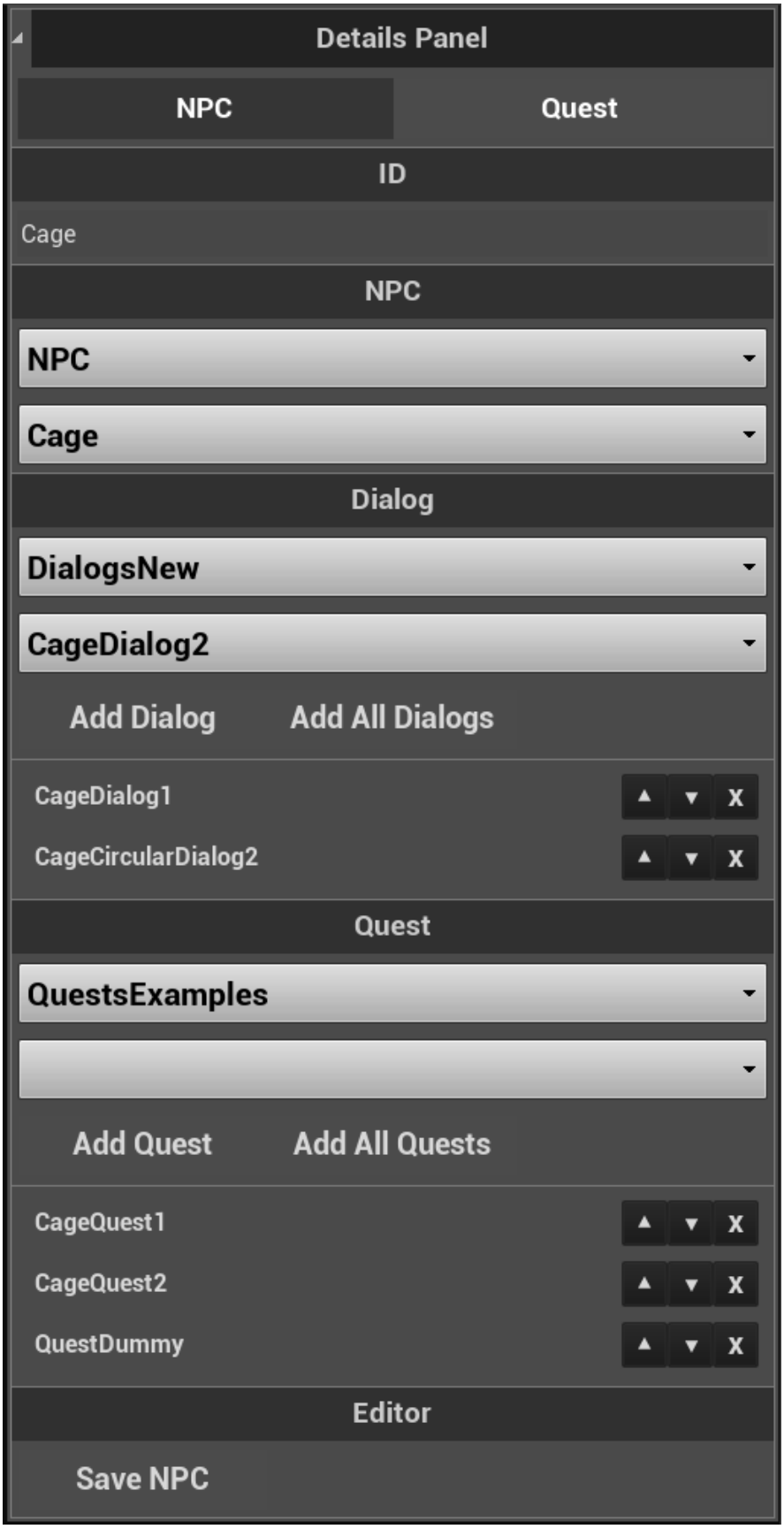


The first thing you have to do is create a new NPC. For this, you will need to fill a Name for it. This name must be unique to a Datatable. You can have same NPC names in different Datatables, but not on the same one.

You can use this name for NPCs in game if you want, or you could leave it as an internal ID.

Once the name of the NPC is selected, you will be able to configure all the dialogs and quests that the NPC has.

The following image shows an NPC with a name set and some quest and dialogs assigned to it.


Details Panel
NPC
Quest
ID
Cage
NPC
NPC
Cage
Dialog
DialogsNew
CageDialog2
Add Dialog
Add All Dialogs
CageDialog1
CageCircularDialog2
Quest
QuestsExamples
Add Quest
Add All Quests
CageQuest1
CageQuest2
QuestDummy
Editor
Save NPC

There is no limit to the number of quests or dialogs that an NPC can have assigned to, but it is important that at least one Start Talk Task for the selected NPC is in the quest/dialog. We will explain a bit more about this later on.

The slots for each quest and dialog allow for some operations such as removing or rearranging them.

To save an NPC, you need to click the button that is at the bottom. This will save the NPC to the datatable that is selected in the NPC tab and will use the name of the NPC set in the ID. Keep in mind that saving ab NPC will overwrite the previous one of same name on the datatable if it is one already and will create a new one if this is the first time you are saving it into this datatable.

Back to top.

2.2.2 Editing Quests and Dialogs

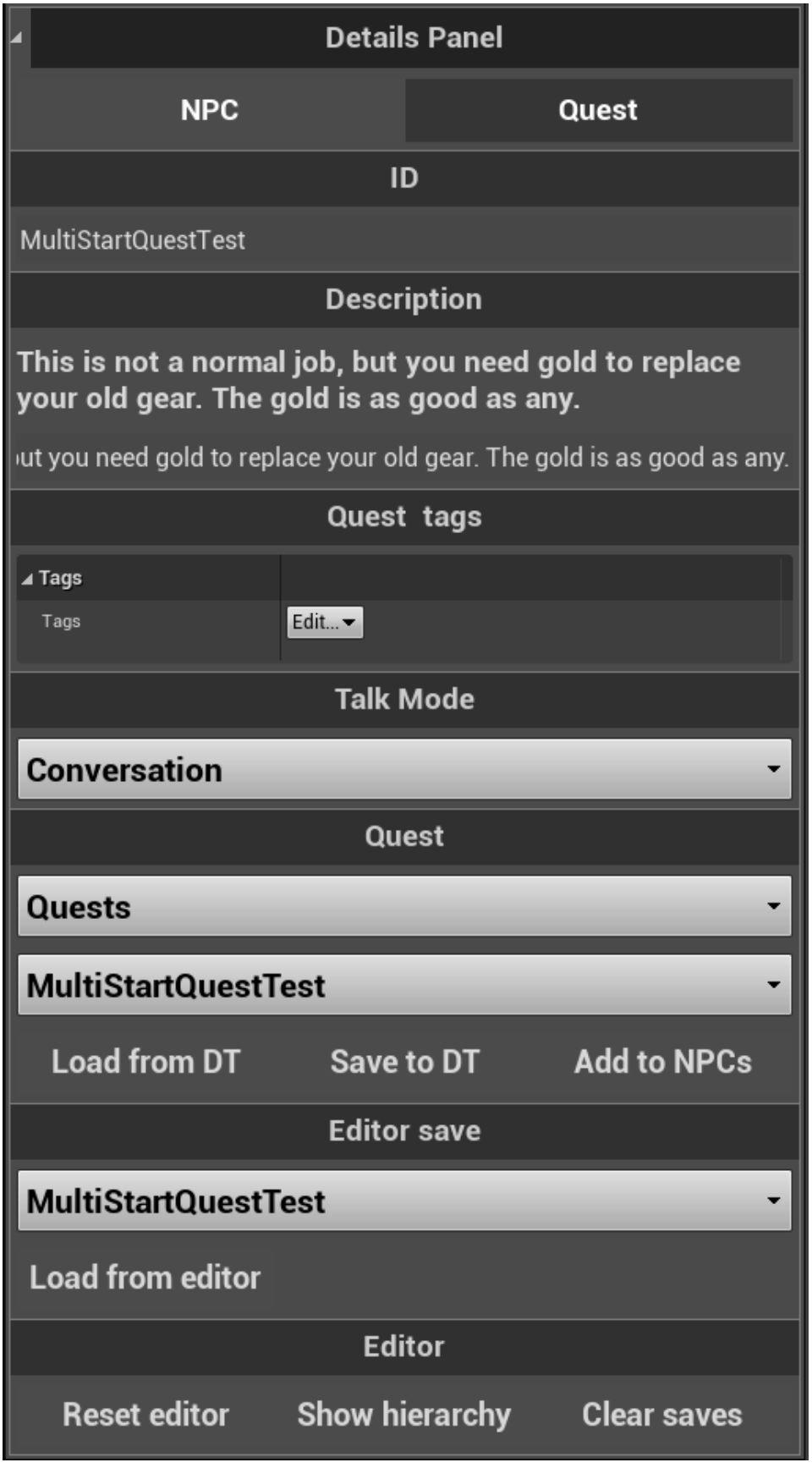
This is the second mode of the Quest Editor, and can be accessed clicking in the mode buttons, at the top of the details panel.

Quests and dialogs use the same structure in the system (Quest) and therefore there is no real difference between them other than the way they are handled in the system.

The only thing that makes a quest work as a quest is that is assigned to an NPC as a quest, and for a quest to work as a dialog, it has to be assigned to an NPC as a dialog.

I will refer to them as quests mostly during the documentation, but they could be used as dialogs as well and all things that apply to quests will apply to dialogs.

The following image shows the details panel for a quest.


Details Panel
NPC
Quest
ID
MultiStartQuestTest
Description
This is not a normal job, but you need gold to replace your old gear. The gold is as good as any.
ut you need gold to replace your old gear. The gold is as good as any.
Quest tags
Tags
Tags
Edit...
Talk Mode
Conversation
Quest
Quests
MultiStartQuestTest
Load from DT
Save to DT
Add to NPCs
Editor save
MultiStartQuestTest
Load from editor
Editor
Reset editor
Show hierarchy
Clear saves

A name must be specified for each quest. This name will be important inside the game, and the player might be able to see it, depending on how you setup your UI. It is also important to keep track of the quest in the editor.

Names should be unique inside the same datatable, otherwise you will overwrite the previous quest with the same name.

You can have quests with the same name in different datatables and this is fine gameplay wise, as the system will recognize that these are two different quests. But I advise against this for several reasons. First of all, it can lead to confusion and some features of the quest editor might not work properly, like the real time debugging, since this system will have problems to determine which of the quests is the one selected.

There is the option of adding tags to the quests, for further logic and organization. These tags are optional. You can use them to classify your quests, por passing information for UI behavior or for designing any extra logic that you want to have in your quests.

The Quest Tab allows us to save to and load from Datatables all the quests. To save a Quest, keep in mind that the selected Datatable in the Primary combo box of this tab will be used along with the name of the Quest.

The editor save will keep an automatic record of you quest to avoid losing progress, and you can use it to load too.

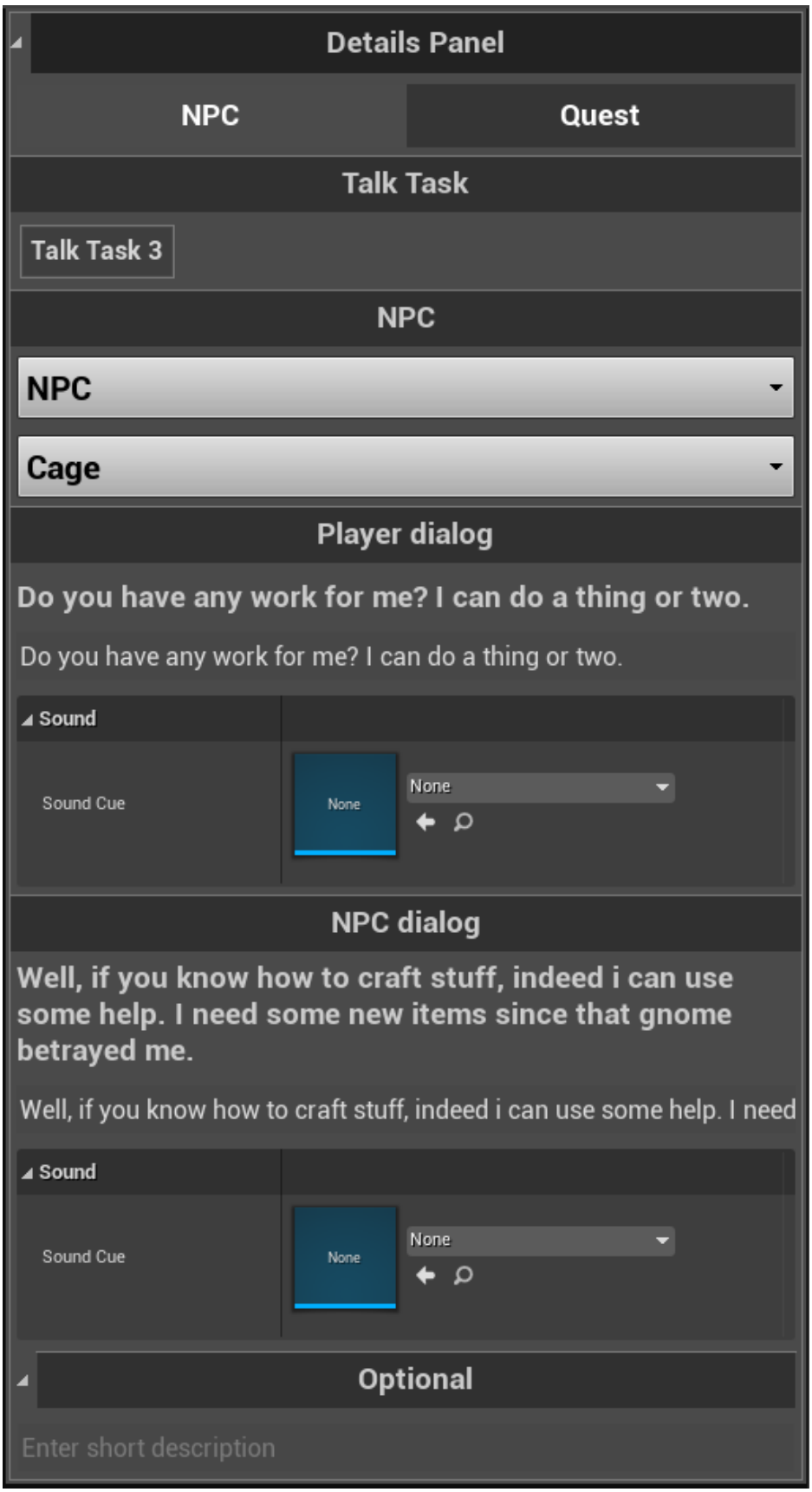
Back to top.

2.2.2.1 Editing Talk Tasks in quests.

Talk Tasks are an integral part of Quests and Dialogs, and you can't modify the variables that they hold, unlike normal tasks. This makes them less flexible, but more efficient. The UI for editing Talk Tasks is not customizable for now.

There are three important things to set when dealing with a Talk Task in a Quest. The NPC that the Talk Task is assigned to, the Player dialog and the NPC response. They represent only a "single exchange" of words and many will be required to make actual conversations. This makes them very powerful, since you can make different paths in a Quest base on which nodes where completed by the player.

Variables as sound and short descriptions are optional.

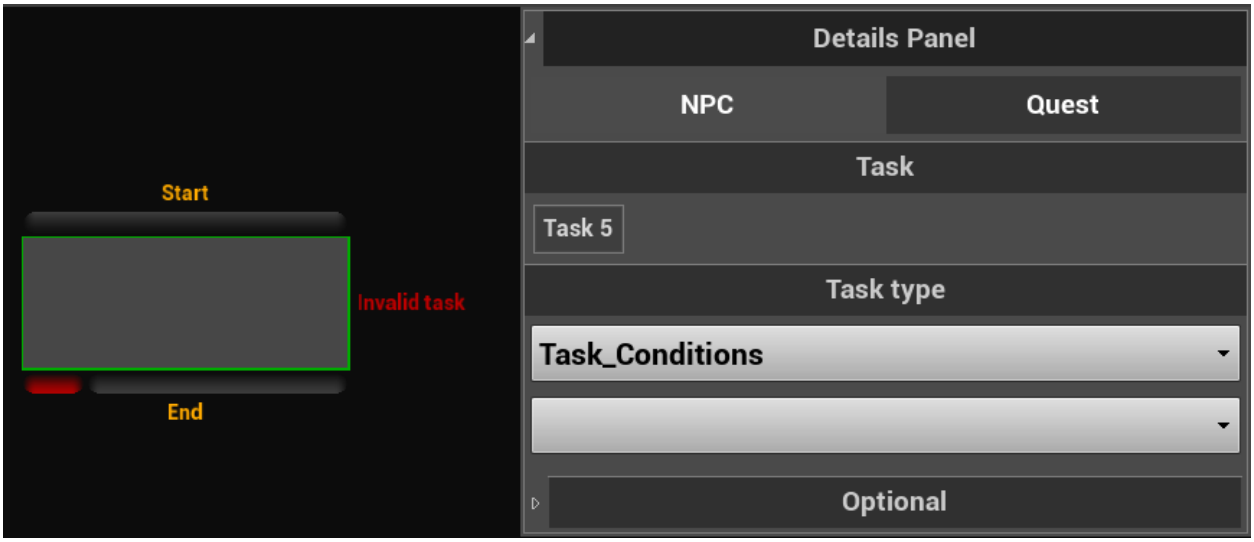

Details Panel
NPC
Quest
Talk Task
Talk Task 3
NPC
NPC
Cage
Player dialog
Do you have any work for me? I can do a thing or two.
Do you have any work for me? I can do a thing or two.
Sound
Sound Cue
None
None
NPC dialog
Well, if you know how to craft stuff, indeed i can use some help. I need some new items since that gnome betrayed me.
Well, if you know how to craft stuff, indeed i can use some help. I need
Sound
Sound Cue
None
None
Optional
Enter short description

Back to top.

2.2.2.2 Editing Tasks in quests.

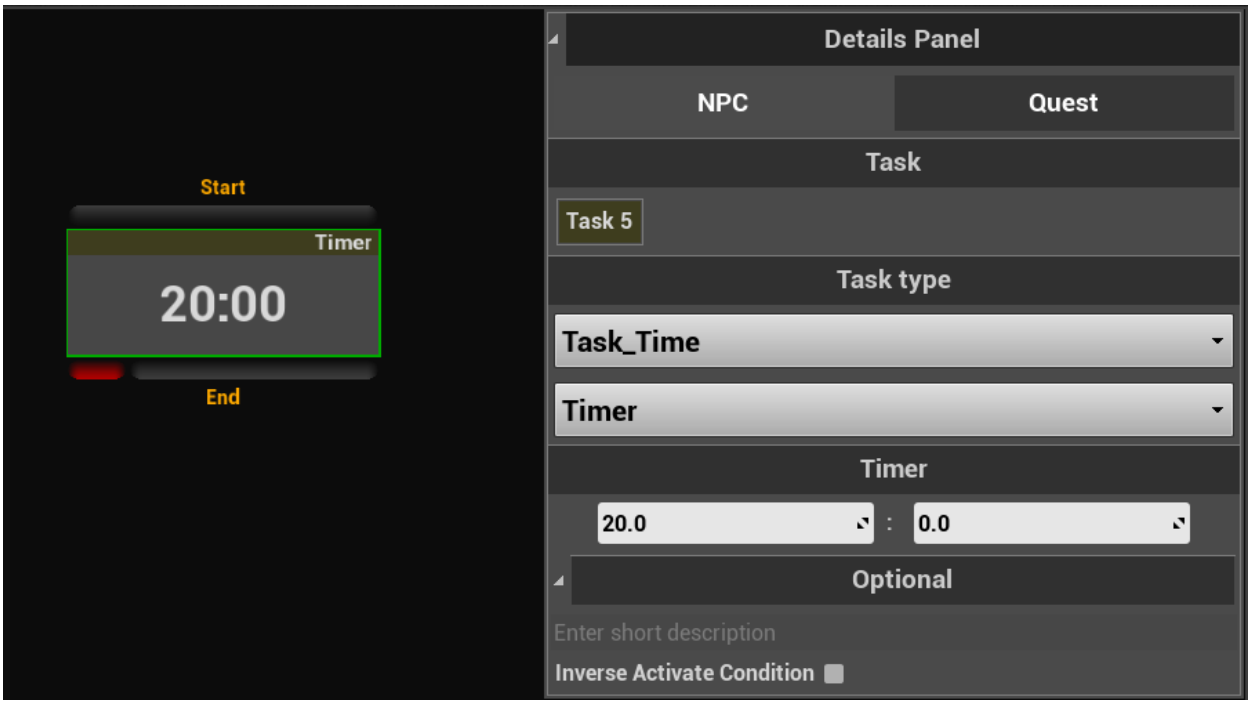
To edit a Task, you need to first select it.

When you are editing a Task that has no type assigned yet, you will need to fill this variable in order to continue. You can select between the datatables of “QuestTaskEditorInfo” and then the Task Type. You can use multiple datatables to organize your custom tasks. The selected task will be represented with a Task Slot, that shows the ID and the color of the task according to its type. In this picture, the task is white because it doesn’t have a type assigned yet.



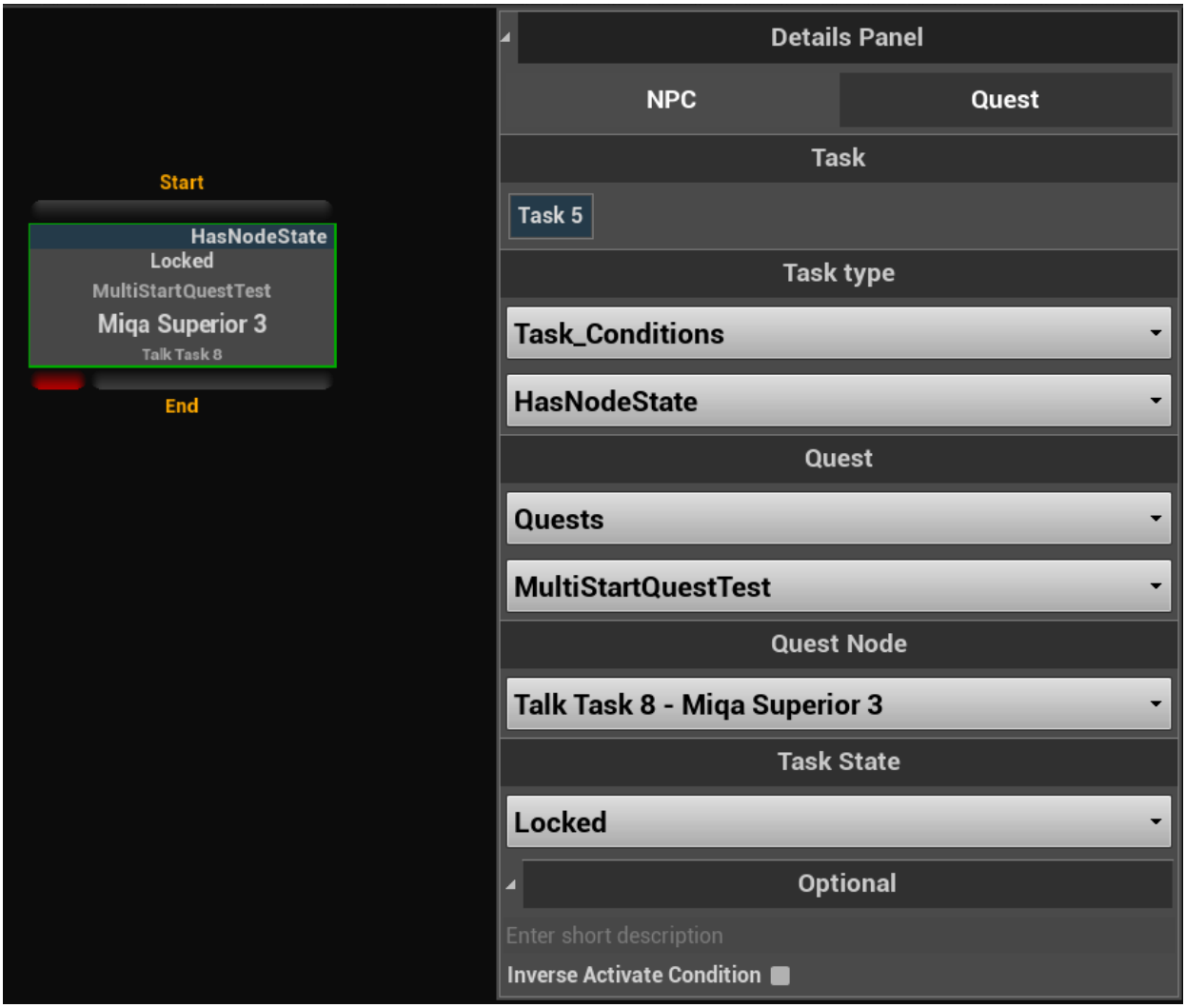
Once you select a Task type, the UI will be generated based on the information of the datatable for this task. Because you can design these widgets, this UI will vary a lot between Tasks. The next is an example for a Timer Task.

The Timer Task requires two values to be entered, the minutes and seconds of the timer. This is a fairly simple Task, that does not use many variables.



For more complicated Tasks, the UI can grow even more. Look at the following example for a “HasNodeState” Task. This Task determines if a Node of a Quest has a given State. All these variables must be entered using the details panel.

When changing the Task type, all the data that is set will be reset to default. This is a forced behavior to prevent saving extra data, that the next Task might not need, in datatables.



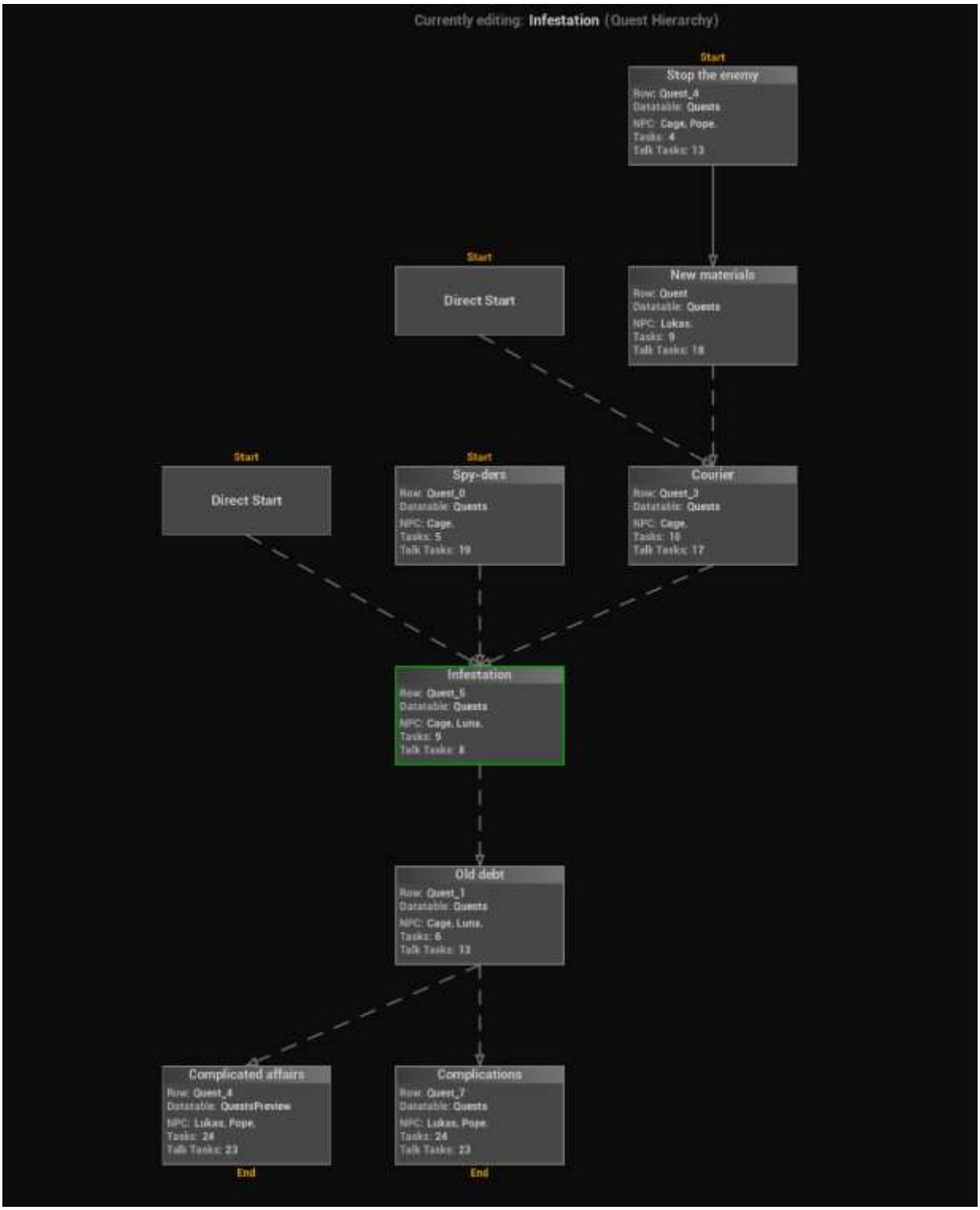
You can use the “Inverse Activate Condition” Boolean to have the opposite check (HasNotNodeState) using the same task node. This allows to use the same node for true and false condition and is particularly useful for creating branches inside the quests.

Back to top.

2.2.2.3 Quest Hierarchy

It is possible to visualize the selected quest relations with other quests that are connected to it via conditions. Using the quest hierarchy button (in the editor tab), you can see the quest chain flow. This is presented as a node graphic in the viewport.

The only allowed conditional nodes for chain hierarchy are “HasQuestCompleted” and “HasQuestCompletedWithEnd”. If other conditions are used for the quest, they will not stablish a hierarchy relation.



This is an example of a hierarchy visualization for the quest “Infestation”. Direct start nodes are added when the player can access that quest with a node that does not requires any condition.

The quest hierarchy is built upon the quest data from datatables and not from the temporal data stored in the editor. This is why it is recommended that the quests are saved to datatables before using this feature, so the information presented is updated.

Back to top.

2.2.4 Multi Node Editing

It is possible to edit multiple nodes at once. For this, all the selected nodes must be from the same type.

This means you can edit for example a group of Talk Tasks. You can edit multiple Tasks, but they must be of the same Task class.

When editing multiples nodes, if the variables are not the same for all the nodes, a null version of it will be displayed on the details panel.

For some variables this is not useful, but in some cases, this can save work.

A good example of this are Talk Tasks, you can easily set the NPC for a group of Talk Tasks using this feature.

Back to top.

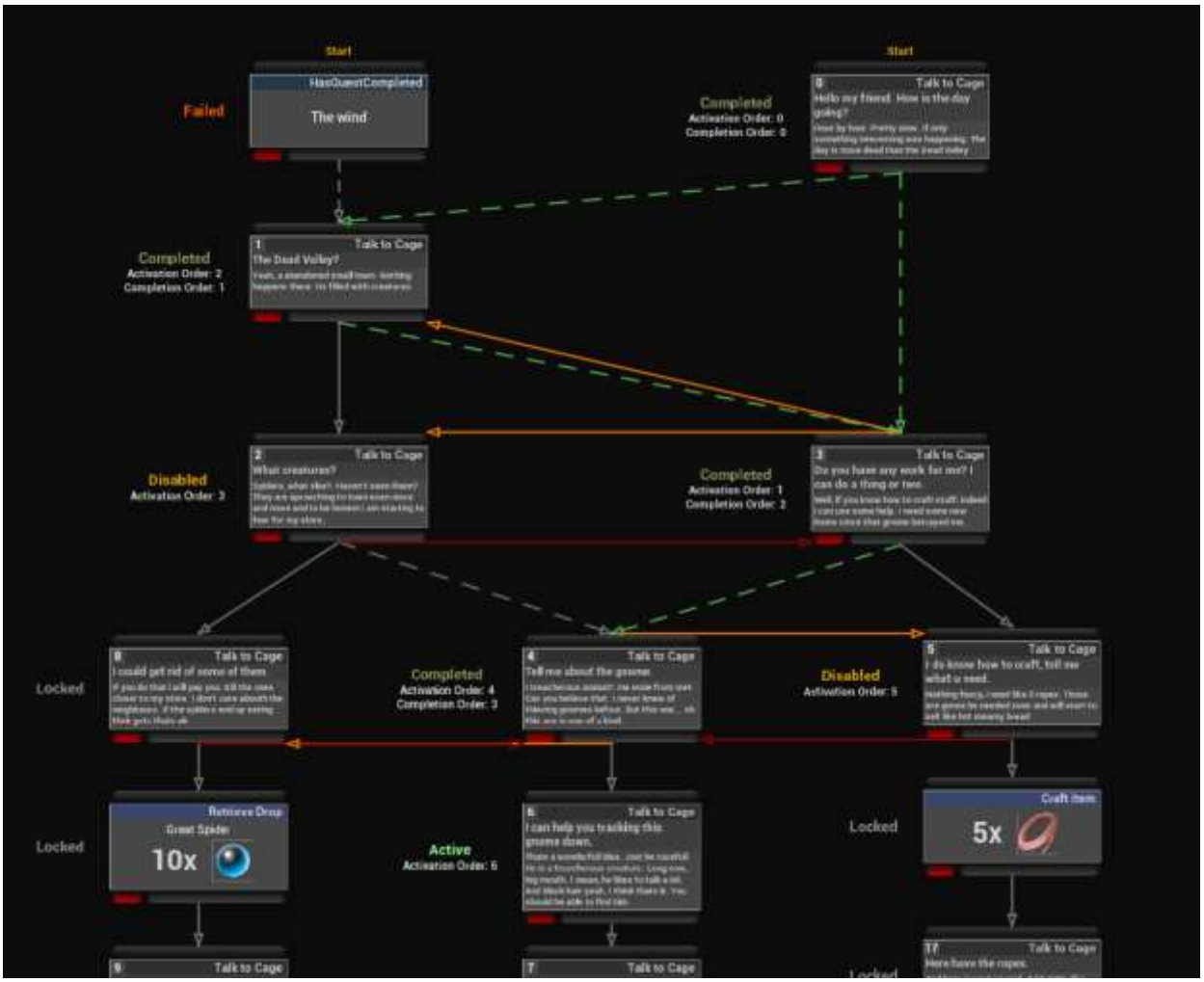
2.3 Debug

If the Quest Editor is open while playing inside the Unreal Engine Editor, some features for debugging become available. For this to work properly, the Quest Manager has to be placed inside the Player State, we will explain more about this on the setup section.

2.3.1 Debug in viewport

In the viewport, the nodes will show additional information related to the quest and the current progress of the player while playing in the editor, as long as the quest is active or completed. This will allow to see the path that the player took on the quest and how the nodes behave, in real time.

For dialogs, it will show information also if the selected NPC has the current dialog set.



This image shows how the nodes look while debugging.

In green you can follow the path that the player did and you can see some additional information for the nodes such as the Activation and Completion orders.

2.3.2 Debug Panel

The debug panel will be visible while playing in the editor, and will show additional information of the player progress, such as active quests or selected NPC information, depending on the editor mode.

It will also allow to switch between different players if playing with several sessions simultaneously.

Some information of the variables in the Quest Manager and the Current NPC Manager will be shown in the panel to help debug.

3. System Overview

3.1 Nodes

Each quest and dialog are represented by an interconnected network of nodes. This conforms a tree that is read from top to bottom. Tasks with lower IDs will be also prioritized if possible.

Each node is represented by the squared widget, and the connections between them by lines.

Each node is identified by their Type and ID that combined provide a unique identifier for the node.

Back to top.

3.1.1 Node types

There are two node types in this system and they are explained below.

Back to top.

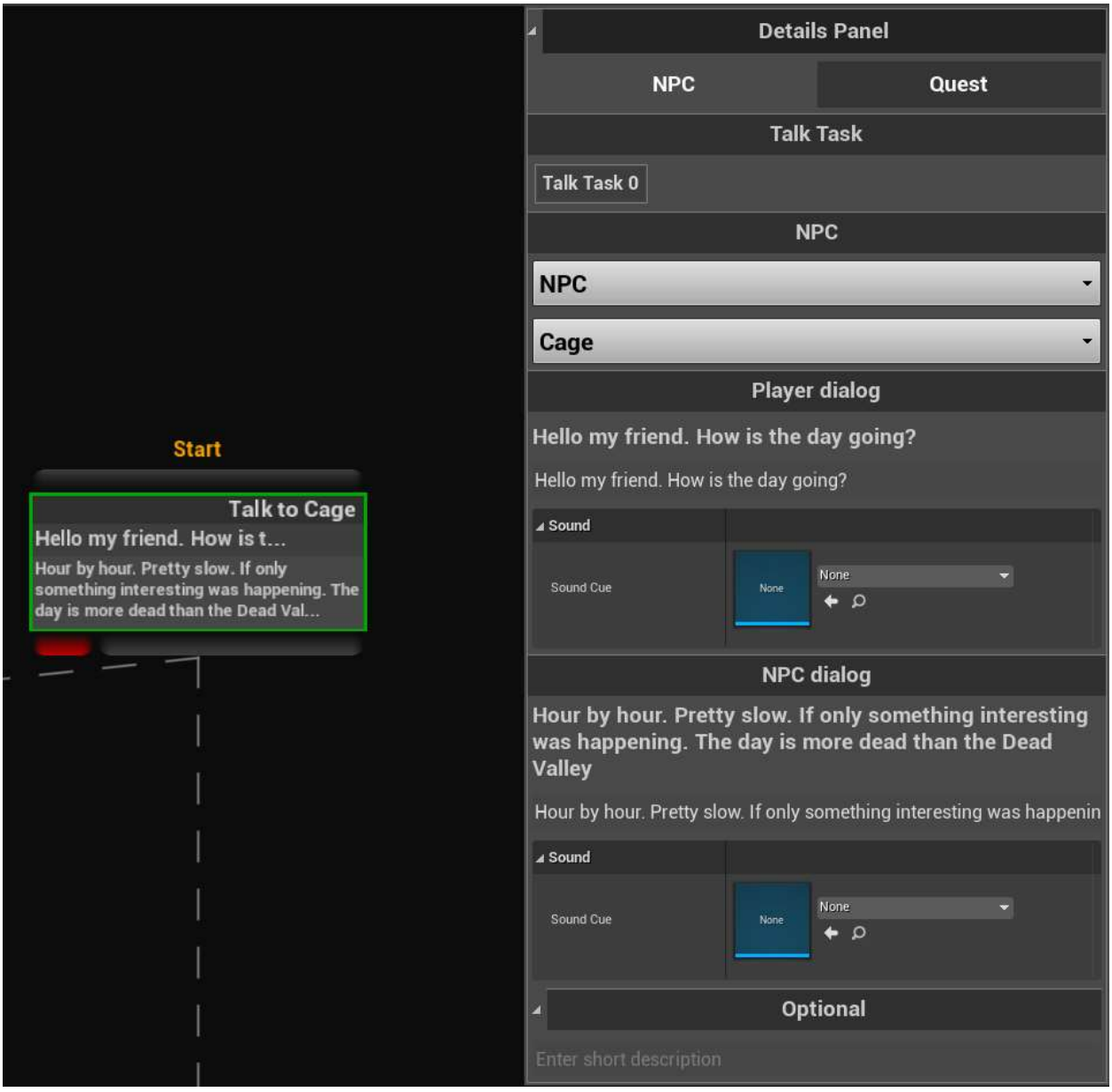
3.1.1.1 Talk tasks

Talk Tasks are the most essential part of quests and the ones that handle the NPC interaction with the player. Each of these nodes represent one exchange of “talk” between the player and the NPC.

They have a “Player say” and a “NPC response” and some other variables that need to be set, some of which are optional.

and can be connected to form conversations with the player.

To add a Talk task node in the editor, you need to press left mouse click while holding alt.



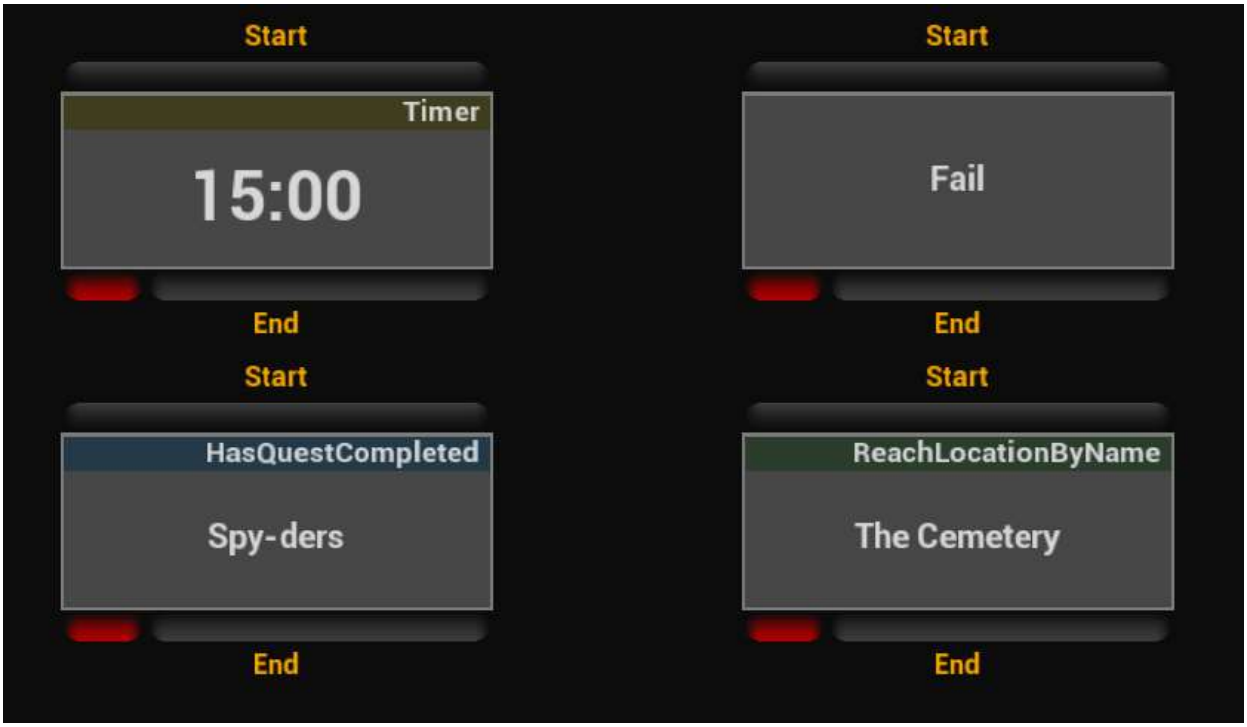
Talk tasks have the ability of inheriting short descriptions from previous talk tasks of same NPC. This allows for further customization of quests in the UI.

Back to top.

3.1.1.2. Tasks

The task nodes are the more versatile part of a quest functionality. They are where the real power of the system resides. These nodes will be mostly created by the user of the system and they will be able to perform any desired functionality.

They will be the ones that handle things such as counting the number of units of a type the player killed, the item they crafted or even simple things like just rotating the camera, making the character teleport or gain a level or basically anything that you feel should be integrated into your system.



These are some tasks that are in the plugin by default.

To add a task node in the editor, you need to press left mouse click while holding Ctrl.

Back to top.

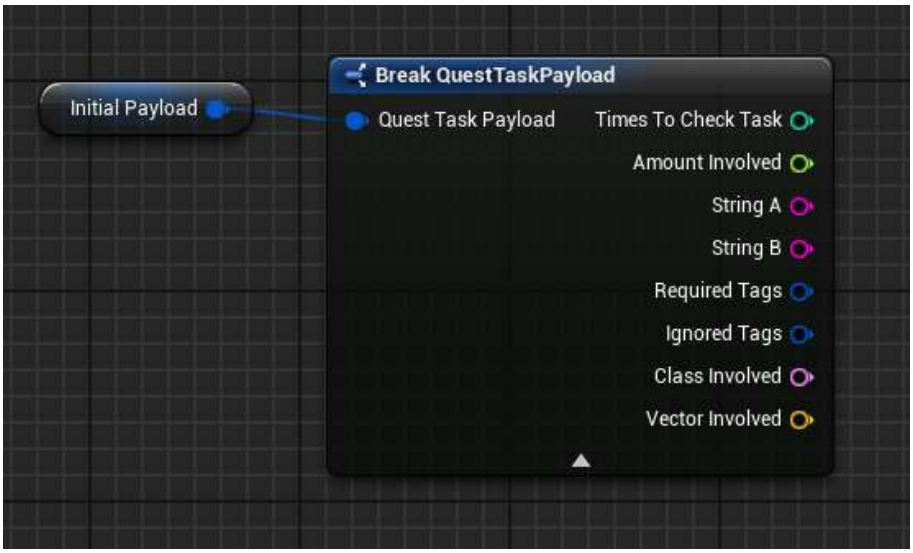
3.1.1.2.1 Task objects

The task objects will be spawned when their activation requirements are met and will perform the needed logic.

These objects are not replicated, but a copy of them exists in both client and server, to ensure there is correct prediction and to avoid cheating. Because of this, all the logic inside them does not need to contain any RPCs.

Back to top.

3.1.1.2.2 Task payload



The payload is a structure than can be used to push data into the task objects. Because of its general purpose, a group of common variables is what constitutes it.

You can use this payload for general variables or even for some custom solutions, where you convert structures to some of the variables from the payload. The string fields are very useful for this, since you can convert any struct to string and then reconvert it back to the initial struct once in the task. This is also useful if you want to push arrays of structures into the payload.

But this can be difficulty or inconvenient, and even when it can help in some cases, it is recommended that you use the Custom Payload for this.

One of the advantages of the task payload is that it can be modified inside the tasks, to update some of the variables, for example, for saving some of the task progress.

Back to top.

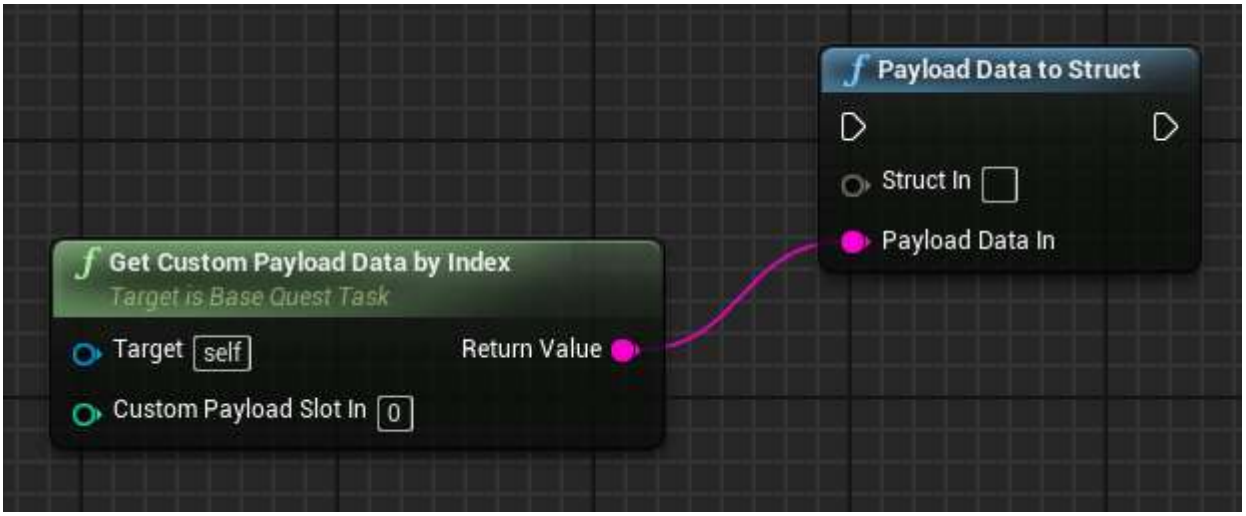
3.1.1.2.3 Task custom payload

The task custom payload is a specific type of payload that allows the user to store any type of structure in it, in the form of a string. It works in a similar way to the task payload.

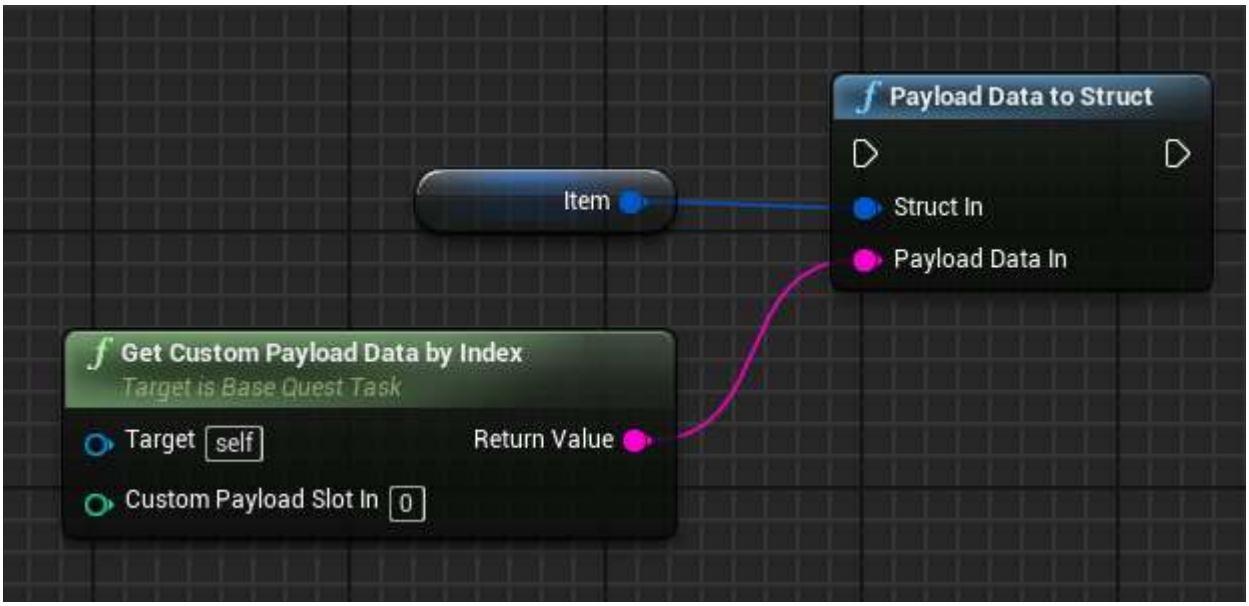
This payload cannot be modified later on, so only the initial values, that are set in the editor, will be available.

There is no limit for the number of custom payloads that a task can have, one of each capable of storing data for a single structure.

The custom payload string data must be reconverted in the task to be used. It is important to provide the correct structure type when doing this.



In this picture you can see how to obtain the payload data by index and how to convert this data to a structure. This structure must be the same provided in the quest editor.

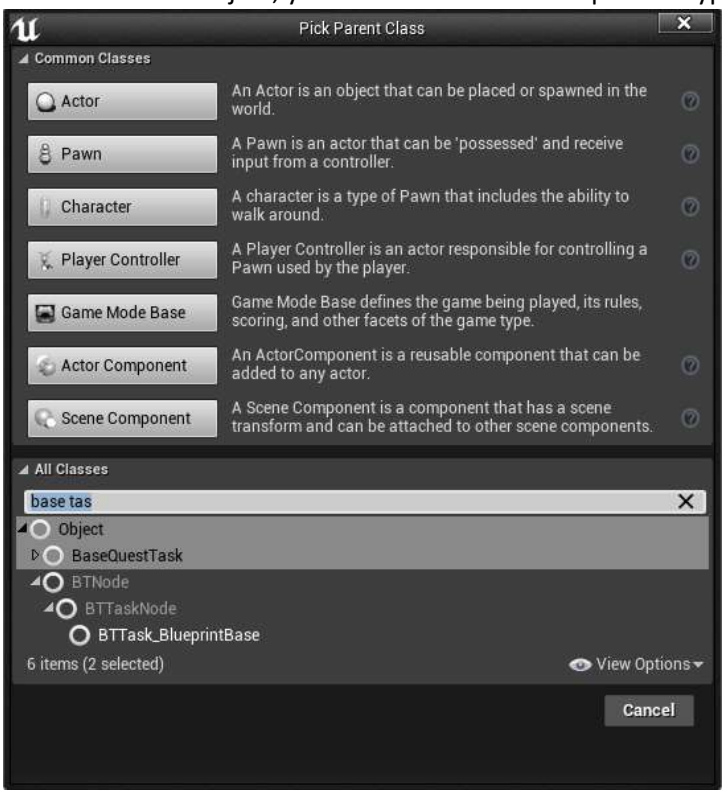


This is an example using an item structure. The item variable must be defined in the task and will be filled with the payload data after the “PayloadDataToStruct” function has been executed.

Back to top.

3.1.1.2.4 Creating task objects

To create a task object, you need to create a blueprint of type “BaseQuestTask”.



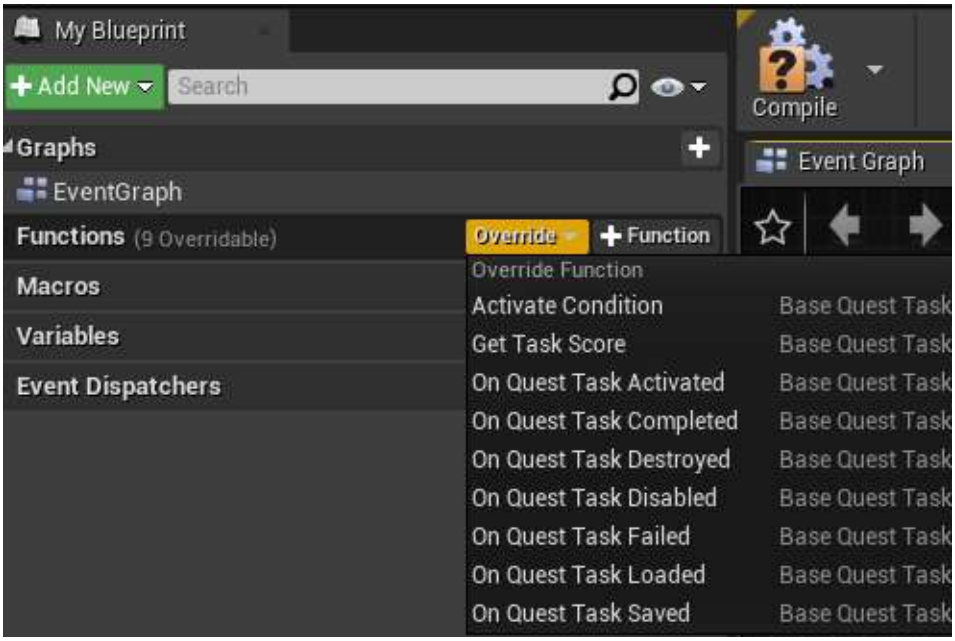
It is recommended that you create one BaseQuestTask and leave it as parent of all (or almost all) the task that you make later, to make it easier to implement interfaces or logic to all via modifying this parent task.

Some tasks that are suited for any project are included in the system.

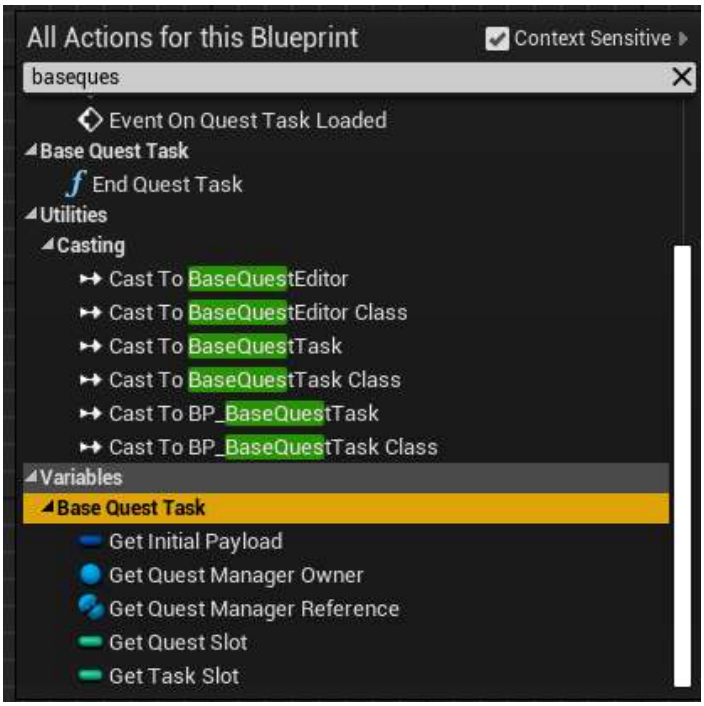
[Back to top.]

3.1.1.2.5 Task functions and basic concepts

The tasks objects have a variety of functions that can be overridden if needed to facilitate their implementation.



They also have access to several variables, under the “Base Quest Task” category, such as the payload and the QuestManagerReference, to facilitate their implementation.



Back to top.

3.1.1.2.5.1 Activate condition

This is a function that can (and will) be executed before spawning the task object. For the reason it can only use the variables provided and not variables from the task object itself.

If this function returns false, the task will fail and will not activate.

If it returns true, it will activate. In the Quest Editor, you can set the bool “ReverseCondition”, to invert the return value and be able to use the same task for the opposite check. For example, you can check if the player has a quest completed using the node presented earlier, but you can also check if the player does not have the quest completed using the same node and the ReverseCondition bool.

As a reminder, this is the function that will have to override the ActivateConditionOnly tasks.

Back to top.

3.1.1.2.5.2 OnQuestTaskActivated

This function will execute when the task is activated. It will only fire once and is perfect for doing initial checks and for handling the task main logic. Usually in this function, you will bind to event dispatchers to check for events, such as killing a monster or crafting an item.

Back to top.

3.1.1.2.5.3 OnQuestTaskLoaded

This will execute when the player loads the game from a previous save game. Usually it contains a very similar logic the OnQuestTaskActivated function. Note that on loading, the OnQuestTaskActivated function will not trigger again.

Back to top.

3.1.1.2.5.4 OnQuestTaskSaved

This is called when the task is saved. Depending on the manager settings, this can happen at the end of the session or when the “save functions” are called on runtime. This is useful to update the payload and save some variables in some cases. The timer task, for example, uses this functionality.

Back to top.

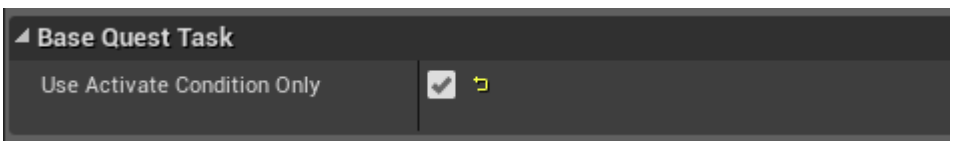
3.1.1.2.5.5 OnQuestTaskDestroyed

This is called when the task is destroyed. This occurs when the task is active and the player leaves the game or when the task ends, because of failing, completing or disabling.

Back to top.

3.1.1.2.6 Condition only tasks

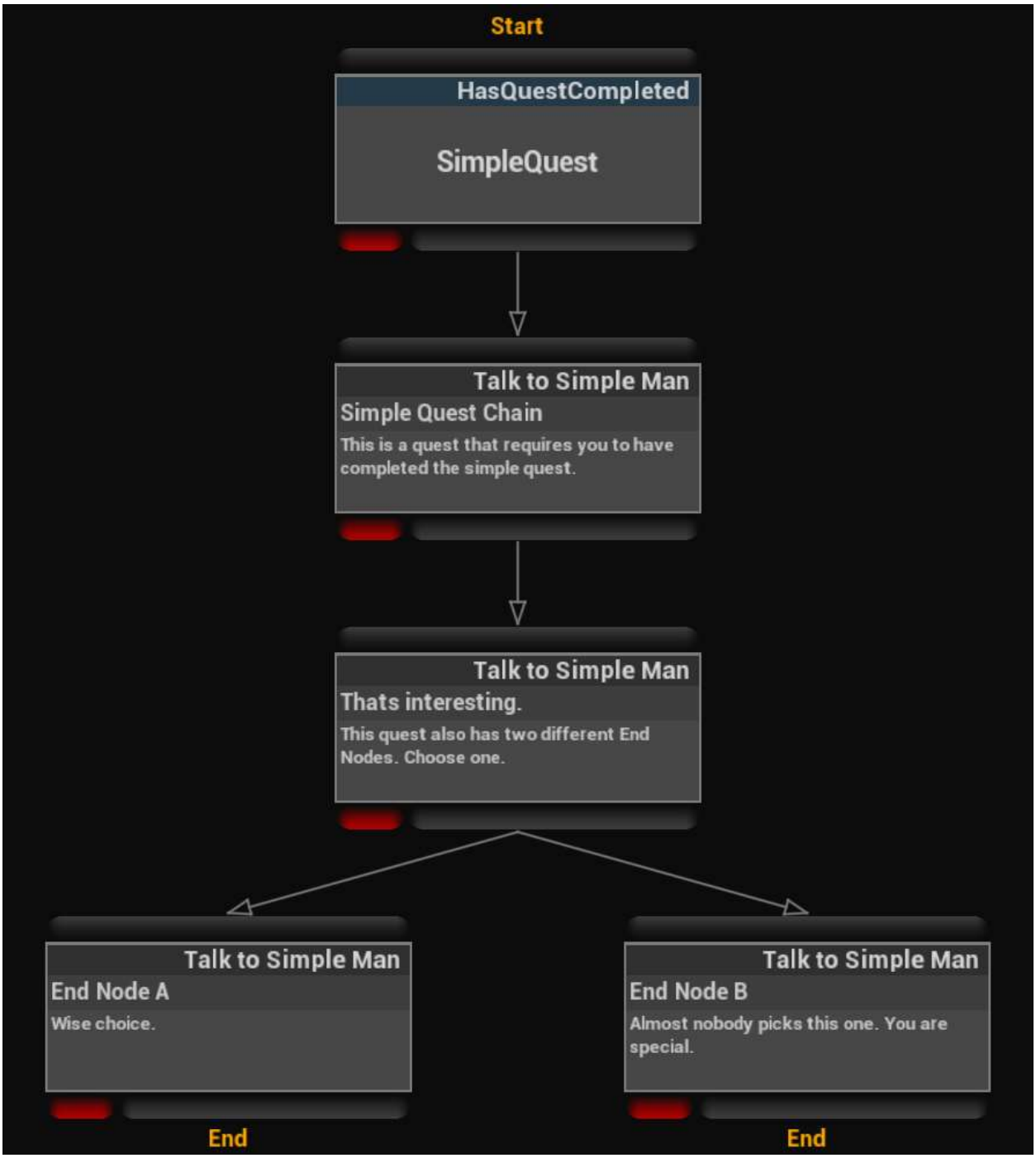
This are tasks that have the variable “UseActivationConditionOnly” set to true in the task defaults.



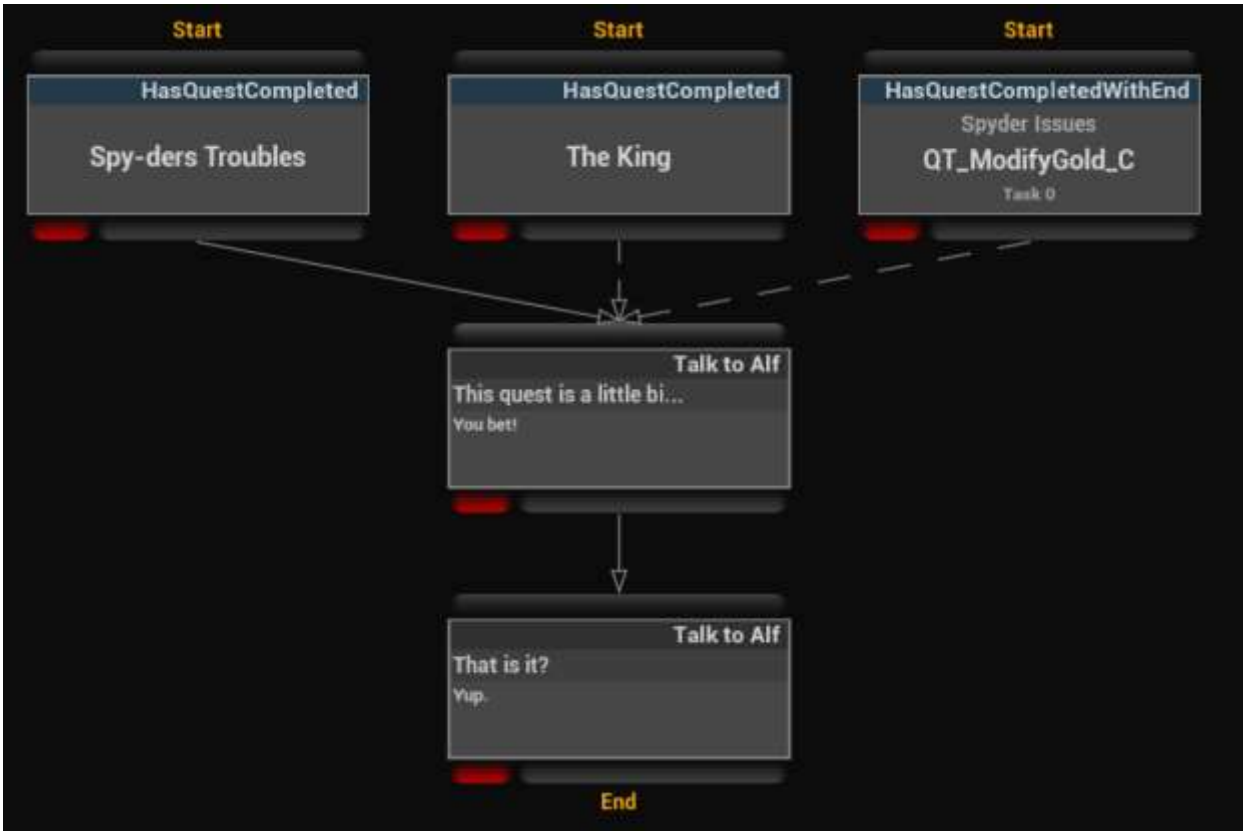
This task objects will not spawn and will not execute any logic outside the ActivateCondition function override. They will not execute any of the other overridable functions such as OnQuestTaskActivated, OnQuestTaskLoaded or OnQuestTaskFailed. If these functions are overridden, they will be ignored.

Because all of the above, they will gain in performance.

One of the particularities of these tasks is that they are also usable before quest start nodes and will serve as preconditions for quests.



In this picture, the first node is an ActivateConditionOnly. It will be checked before granting the quest to the player. Because this particular task checks if the quest is completed, in this case, only if the player has said quest completed it will allow him to see the quest in the dialog tab. If the condition is not met, the player will not know about the quest until its met.

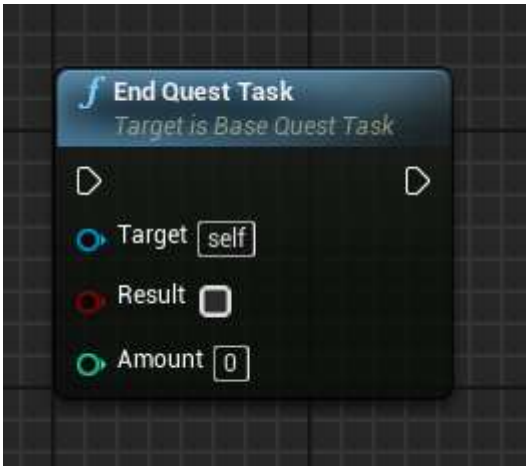


You can use also multiple conditions and even different link types to create any combination that you want. A very useful way is using all optional nodes, making the quest available if only one of those conditions are met.

Back to top.

3.1.1.2.7 Task completion

In order for the tasks to end they need to commit some sort of result. This is done calling the function EndQuestTask.



If the task ends with a positive result, it will use the amount to update the quest track and if the maximum number of desired repetitions is reached, it will complete the task. To decrement the quest track it is possible to enter negative amount values.

If the task ends with a negative result, independently of the amount provided, it will fail the task.

If the EndQuestTask function is never called during the OnQuestTaskActivated execution or the OnQuestTaskLoaded execution, then the task will never be able to complete and the quest most likely will be stuck in that task.

This example illustrates a bit more how the functionality of the tasks should be implemented.

You can see that this is a very simple task that is implemented in a third person project and that counts the number of times that the player jumped.

The events OnQuestTaskActivated and OnQuestTaskLoaded are overridden with the same logic. This is because I want to start counting the amount that the player jumped upon activation, but if I leave and loaded the save game, I want to continue counting the number of jumps.

Each time the player jumps, the delegate MovementModeChanged will trigger, and I will end the task with a positive result of one. This will force the system to check if the desired amounts of jumps have been reached in order to complete the task. If the amount is not reached, it will continue counting and if it is reached the task will complete and continue with the quest execution.

The number of times that this condition must be checked is a variable that is set in the payload of the task when the quest is designed in the Quest Editor.

Even when this example looks very simple, this is the way that a large number of tasks are implemented, just by checking the correct event dispatchers that we fire in some other part of our code.

Back to top.

3.1.1.2.8 Premade Tasks

The system includes several premade tasks, designed for specific functions. Some of them are listed below.

Back to top.

3.1.1.2.8.1 BP_BaseQuestTask

This is a Task that is used for parent of the other premade tasks. This is intended for setting some variables for the interfaces for UI. This task is not intended to use inside quests.

You should not use this task as a parent for your custom ones. Create a different one instead, so you can modify it without fear of losing the changes on the next update.

Back to top.

3.1.1.2.8.2 HasNodeStateInQuest

This is a task that is “ActivateConditionOnly” and is designed for determining the state of any node in any quest, either completed or active.

This task is especially useful for making different paths inside the quest based on previous nodes states, and is designed to use with optional links.

Using this task, you can check if optional nodes where completed or failed, for example, and enable special paths in your quest.

You could also use this task to check if other quests where completed with a specific ending node.

This condition does not affect the quest hierarchy, meaning that if it is used as a Pre-Activation condition, it is recommended that is replaced with “HasQuestCompletedWithEnd” if possible, so that the quest hierarchy will reflect the relation with the quest.

Back to top.

3.1.1.2.8.3 HasQuestCompleted

This is a task that is “ActivateConditionOnly” and is designed for determining if a quest was completed previously.

This task is specifically designed for making quests chains, enabling or disabling the entire quest depending on the player progress on previous quests.

You can also use it to control the flow of paths inside quests too.

This condition affects the quest hierarchy.

Back to top.

3.1.1.2.8.4 HasQuestCompletedWithEnd

This is a task that is “ActivateConditionOnly” and is designed for determining if a quest was completed previously and with a specific End Node.

It’s also designed mostly for quests chain, but with very specific requirements.

This condition affects the quest hierarchy.

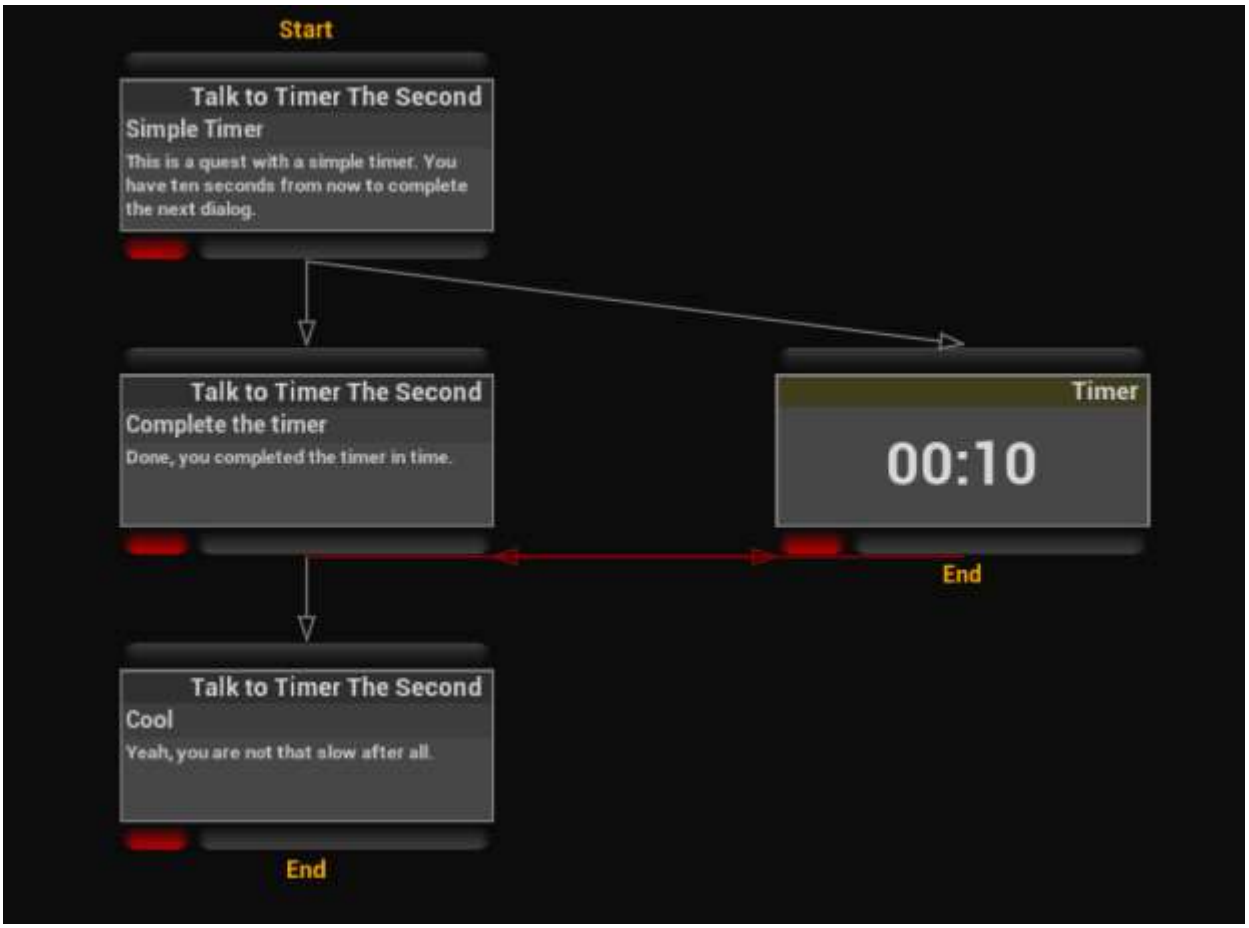
Back to top.

3.1.1.2.8.5 Task_Timer

The timer task is a particular implementation of a task that allows to add a time factor to other tasks in the quest.

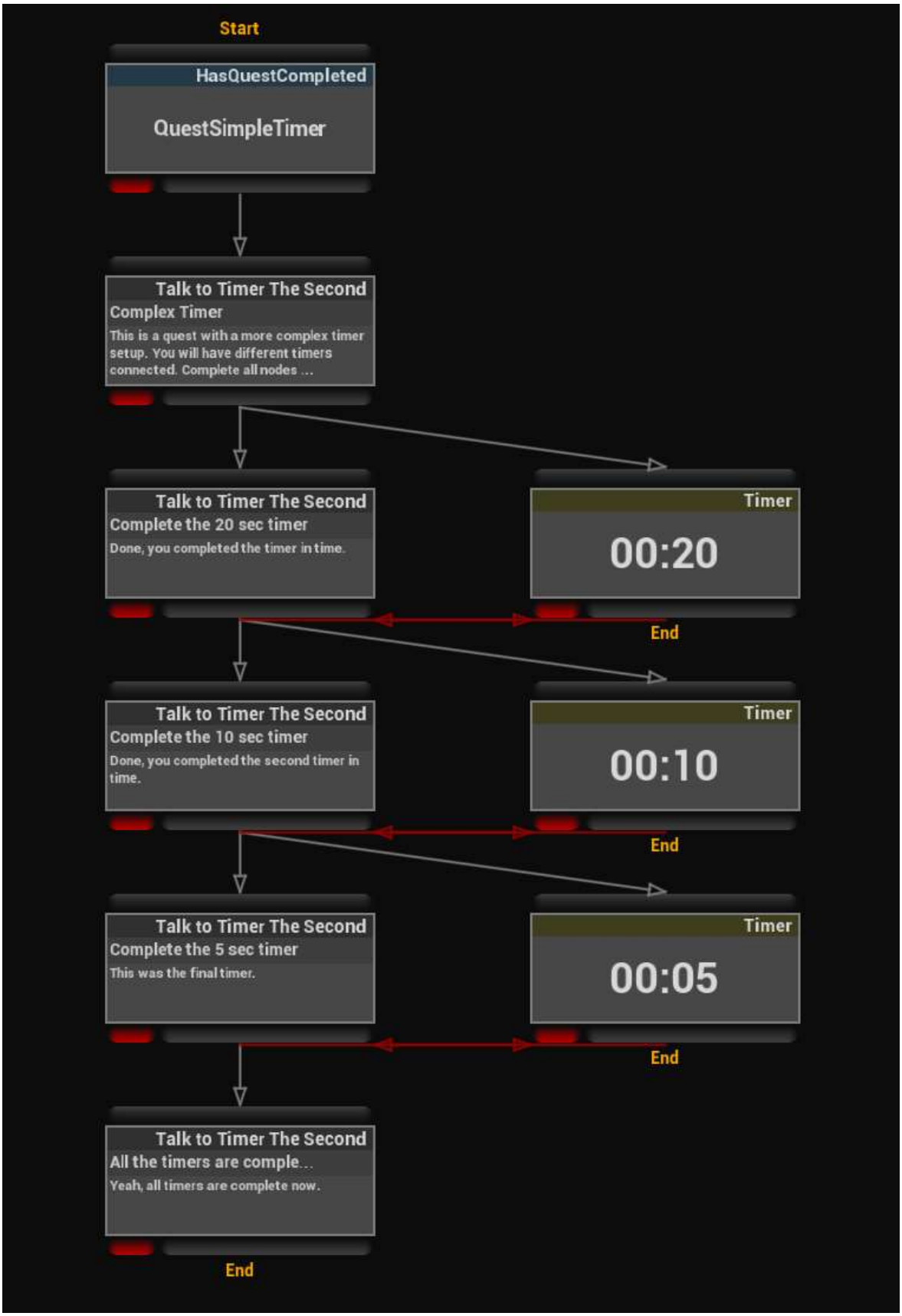
The timer must be linked via a disable connection to all the nodes that it controls. This way, if the task completes, meaning the timer ran out, it will make all those nodes disabled.

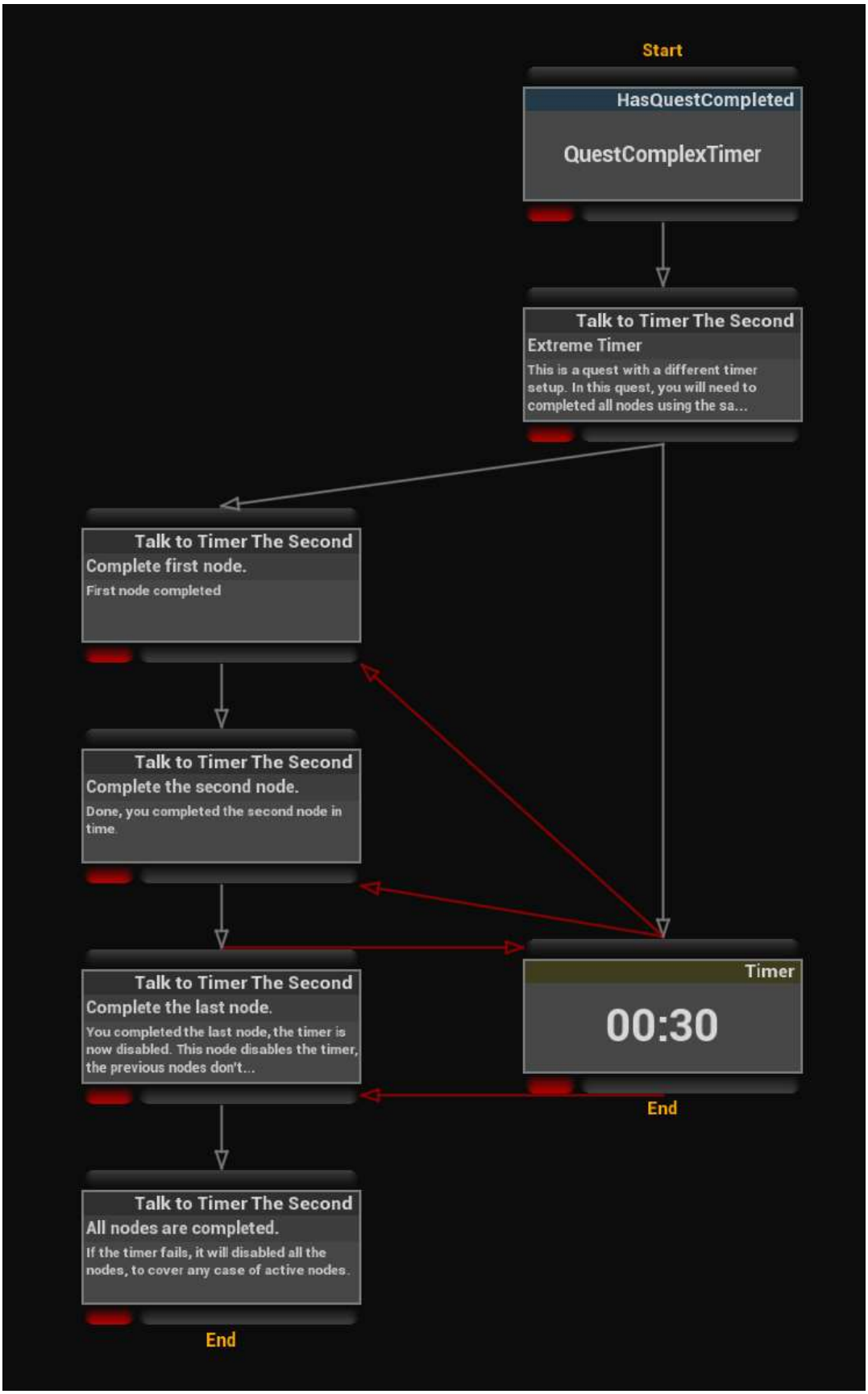
Some or all of the nodes that are linked to the timer must disable the timer task if they complete, depending on the logic you want to achieve, in order to disable the timer and preventing it from failing.



This is a very simple timer example.

The following picture has a more complicated one, using more than one timer task.


Start
HasQuestCompleted
QuestSimpleTimer
Talk to Timer The Second
Complex Timer
This is a quest with a more complex timer setup. You will have different timers connected. Complete all nodes ...
Talk to Timer The Second
Complete the 20 sec timer
Done, you completed the timer in time.
Timer
00:20
End
Talk to Timer The Second
Complete the 10 sec timer
Done, you completed the second timer in time.
Timer
00:10
End
Talk to Timer The Second
Complete the 5 sec timer
This was the final timer.
Timer
00:05
End
Talk to Timer The Second
All the timers are comple...
Yeah, all timers are complete now.
End


Start
HasQuestCompleted
QuestComplexTimer
Talk to Timer The Second
Extreme Timer
This is a quest with a different timer setup. In this quest, you will need to completed all nodes using the sa...
Talk to Timer The Second
Complete first node.
First node completed
Talk to Timer The Second
Complete the second node.
Done, you completed the second node in time.
Talk to Timer The Second
Complete the last node.
You completed the last node, the timer is now disabled. This node disables the timer, the previous nodes don't...
Timer
00:30
End
Talk to Timer The Second
All nodes are completed.
If the timer fails, it will disabled all the nodes, to cover any case of active nodes.
End

This is timer example where one timer is used to control three nodes. The last node is the one that disables the timer upon completion. If the timer runs out it will disable all three nodes.

Back to top.

3.1.1.2.8.6 Task_Delay

The delay timer is very similar to the timer task. Allows you to add delays between the execution of the nodes. It is useful when combined with other tasks, for example, if you have a task that spawns monsters, but you don't want that to happen so suddenly after a dialog with a NPC, you could use a delay in between the nodes.

Back to top.

3.1.1.2.8.7 Task_ResetTrackToNode

This task will stablish some sort of connection to node of the same quest that is placed, and all nodes in between will be locked, meaning they will return to their original state, as if they never had been activated, when this task is executed. This is useful for when you want to repeat dialogs for players. The node specified in the task will be activated.

It is recommended that is used when the paths between these nodes are clear, as it can try to lock nodes that might not be directly related if the nodes web is too intricated.

This node will never execute past it, this means that connecting anything to its inferior nodes will have no effect. The quest will not be able to be completed with this node either, so even when it appears to be an end quest node, it is not.

Back to top.

3.1.1.2.8.8 Task_ResetTrackToNodeWithDisableds

This task works in the same way that the "Task_ResetTrackToNode" node but it will also include any disabled paths of the nodes involved, even with exterior branches. This is useful when having options that disable mutually.

Back to top.

3.1.1.2.8.9 Task_ResetTrackBetweenNodes

This task executes like a normal task. It will take the nodes between the start and end node provided, including these ones, and it will reset them. After completing, it will continue with the normal flow of the quest execution. This node allows to control nodes from any part of the quest and is not tied to its own branch.

Back to top.

3.1.1.2.8.10 Task_FailQuest

This task will force all active nodes to fail and then will fail the quest. It should always be an End Node, since it will never continue executing past it. It is designed to facilitate the workflow.

Back to top.

3.1.1.2.8.11 Task_IsQuestActive.

This condition task will check if the player has this quest in the active quest inventory, meaning that the quest is in progress.

Back to top.

3.1.1.2.8.12 Task_AcceptQuestDirectly

This task will add a quest to the quest inventory directly. It will check if the quest can be accepted and otherwise will fail. It can add quests that starts with task instead of talk tasks and this is its primary use.

Back to top.

3.1.1.2.8.13 Task_RemoveQuestFromCompleted

This task will remove the selected quest from the completed track, as if the player would have never completed it. This allows the player to take and complete the quest again. If the quest is not completed, this task will fail.

Back to top.

3.1.1.2.8.14 Task_AbandonQuest

This task will force the player to abandon the quest, as long as the quest is in progress and will fail otherwise.

Back to top.

3.1.1.2.8.15 Task_IsQuestFailed

This task will determine if the player has a determined quest in fail state.

Back to top.

3.1.1.2.8.16 Task_IsQuestFailedForTime

This task will force the player has a determined quest in fail state and will also check if it has been in this state for at least the time specified.

Back to top.

3.1.1.2.8.17 Task_RemoveQuestFromFailedTrack

This task will remove a quest from the fail track, making it available again for the player, overriding the max fail time that has to be achieved in order for the quest to be removed automatically.

Back to top.

3.1.1.2.9 Reach Location Tasks

The system provides with functionality for creating tasks that check that the player reaches a place or location in the map. This is a very common task in many quest systems and very useful to lead the player during the quest.

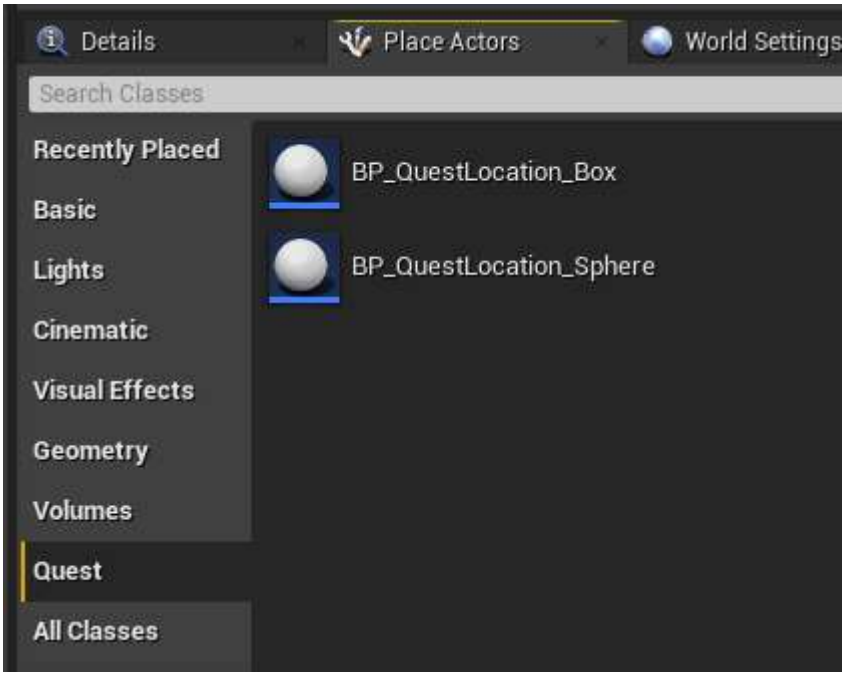
3.1.1.2.9.1 Location actors

These actors are the ones that represent locations in the map. They are invisible in-game and they can interact with the player to complete tasks that follow the reach location logic.

They have a name and tags to differentiate between them and to utilize inside the tasks. You can create children of these actors if you want to add extra variables and customization to them.

The ones provided are for sphere and box collisions, but you can create more of them for different shapes also.

You can always scale these actors to fit the space in the map better.



You can place the location actors directly from the “Place Actors” tab, under the “Quest” category.

You can also find them in the plugin content folder.

Back to top.

3.1.1.2.9.2 Reach Location Tasks

There are two types of premade “Reach Location” tasks. You can decide if you want to handle it via “LocationName” or using “LocationTags”.

You can create new ones too if you want to check different conditions on them. For that, there is a specific task object, “BaseQuestTask_Location”, from which you can derive your new objects.

This object has extra functionality that you can override, such as “GetInteractingCharacter” function, that provides the character that should be used to check on location and “OnQuestLocationReachedCondition”, that handles the check to determine if the player successfully reached the location.

If you have a normal setup for the character spawn, the premade tasks for location will work. If you do not have the character spawned via gamemode, or if you have not placed the QuestManager in the player state, you might have to override the “GetInteractingCharacter” function to provide a valid reference of the character in your game. You will have to create custom tasks for this, that are children of the premade ones.

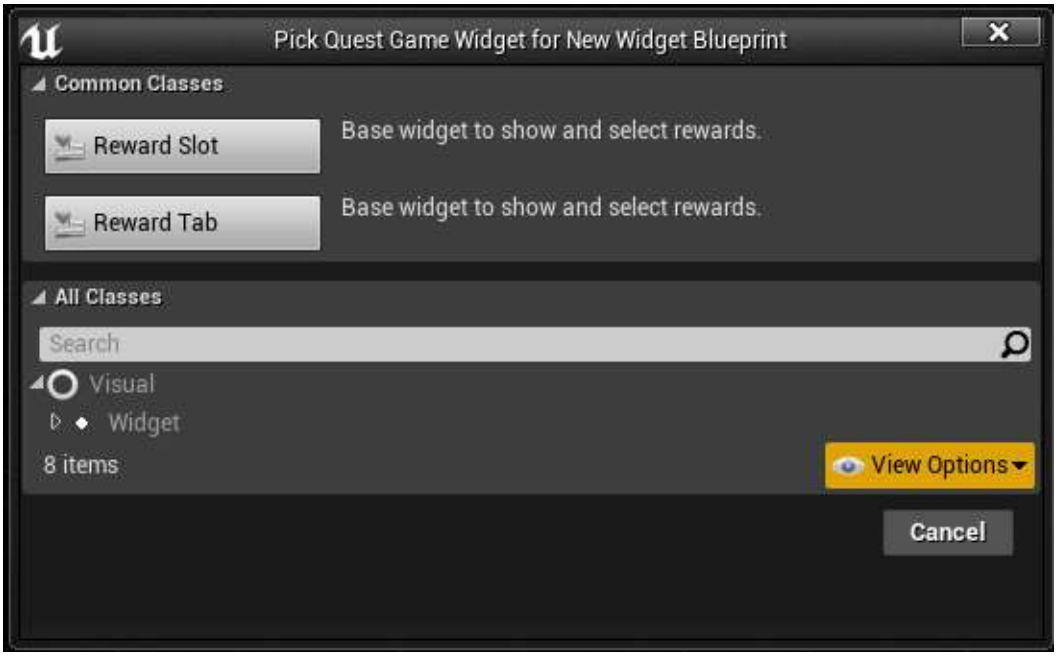
3.1.1.2.10 Reward Tasks

The system provides a template and specialized functionality to make tasks that give rewards to the players easily. You can make the player choose between rewards using this and even give the player mixed rewards options, like choosing between many items, or an item and a spell or gold and experience, etc.

Even when these tasks were designed for the reward system, that is not the only application they have and you could use them for anything that requires the player to select an option from the UI. Examples of this are the selection of levers in an artifact for a puzzle, or cutting wires in a console panel that can lead to different outcomes.

You can create these tasks using the “Quest” category, in the right click drop down menu, in the editor.

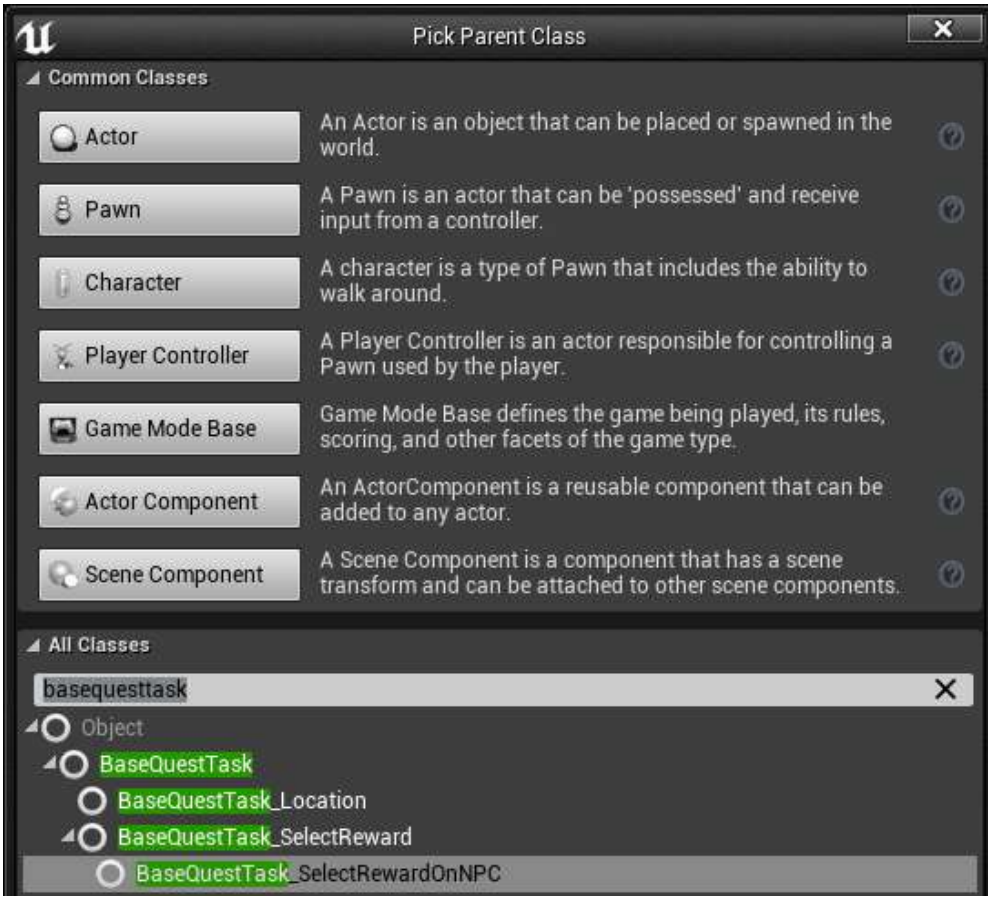
In the game widgets, you have access to the widgets designed as templates for in game rewards UI.



Back to top.

3.1.1.2.10.1 BaseQuestTask_SelectRewardOnNPC

This is a task object, child of the BaseQuestTask object, that has extra functionality for facilitating the creation of tasks that gives rewards on reaching an NPC.



This is how you create one of these objects. You can also create them using the Quest category that the plugin adds on the right click.

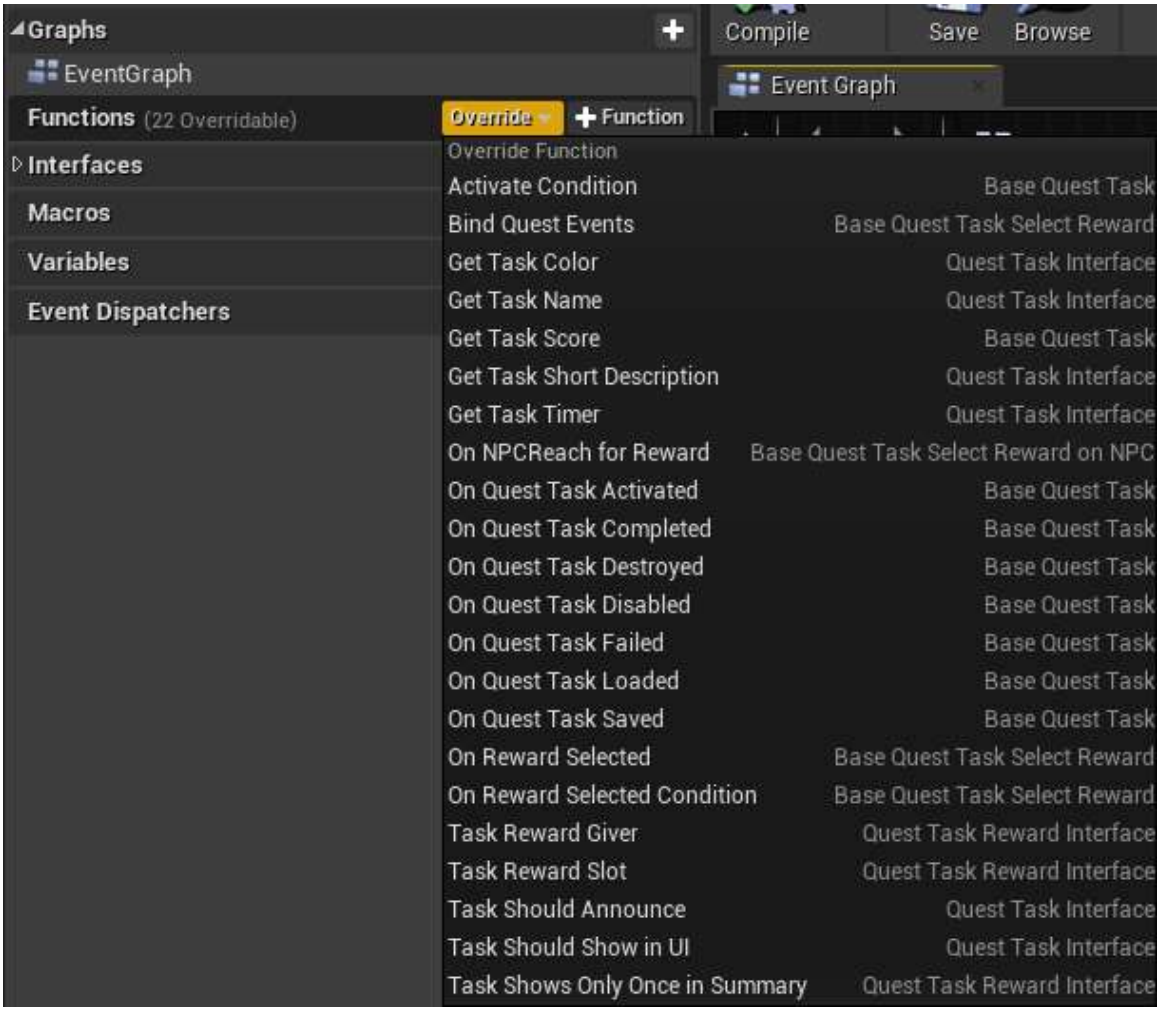
For the NPC check, the payload string A is used and that should not be changed. If you create tasks that are derived of these objects, you will have to use a “NPCSelector” in the string A of the payload.

These tasks will allow the player to select from all active rewards upon reaching an NPC that has the rewards active or on the activation of the rewards.

For creating a custom reward, you will some overridable functions that will facilitate the process. In particular, the function “OnRewardSelected” is designed to handle the reward delivering to the player and in most cases, is the only function that you should override.

You will have to specify the type of UI slot that you want to create also for the particular task, and you will do so by overriding the function “TaskRewardSlot”, from the Quest Reward Interface. This information will be used by the BaseQuestRewardTab to create the correct UI for this task in game.

These tasks have already the logic for activation and loading implemented, and you don’t need to modify them in that fields.



There are the functions that you can override in this object, in particular “OnRewardSelected” and “TaskRewardSlot” are the most important ones and the only ones that you must override with your custom logic.

Back to top.

3.1.1.2.10.2 BaseQuestTask_SelectRewardOnLocation

Similar to “BaseQuestTask_SelectRewardOnNPC”, but it activates when reaching locations of specified name. The Payload String A must always contain the name of the location.

Back to top.

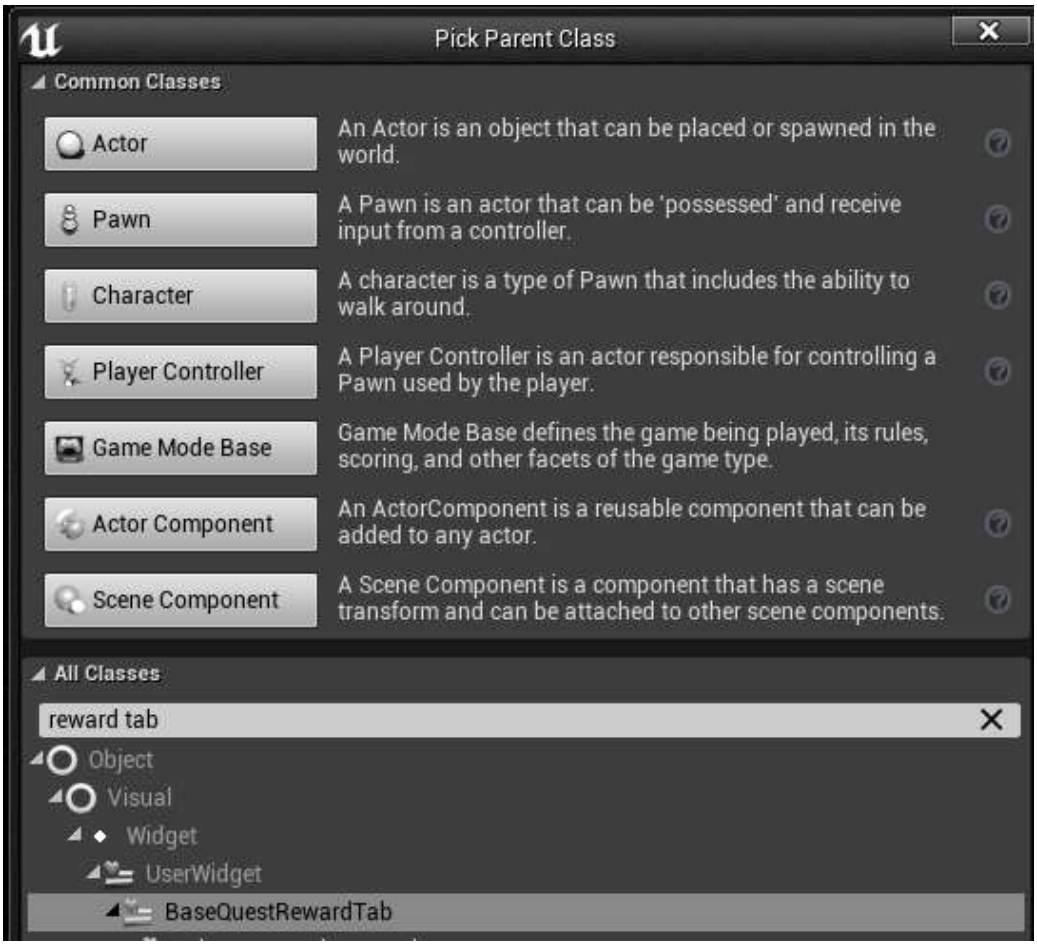
3.1.1.2.10.3 BaseQuestRewardTab

This is a widget from which you should derive the main tab for handling reward slots in your game.

This widget has a function, “UpdateRewardTab”, that you should override to update it. It provides all the information for creating the reward slots inside the tab.

It also has a function to get the QuestManager reference, that you should override in case your QuestManager is not in the player state.

An example of this UI is provided, that will work fine for most cases. You can use this widget as a base for your reward selecting, and change how it looks and adjust it to your game.

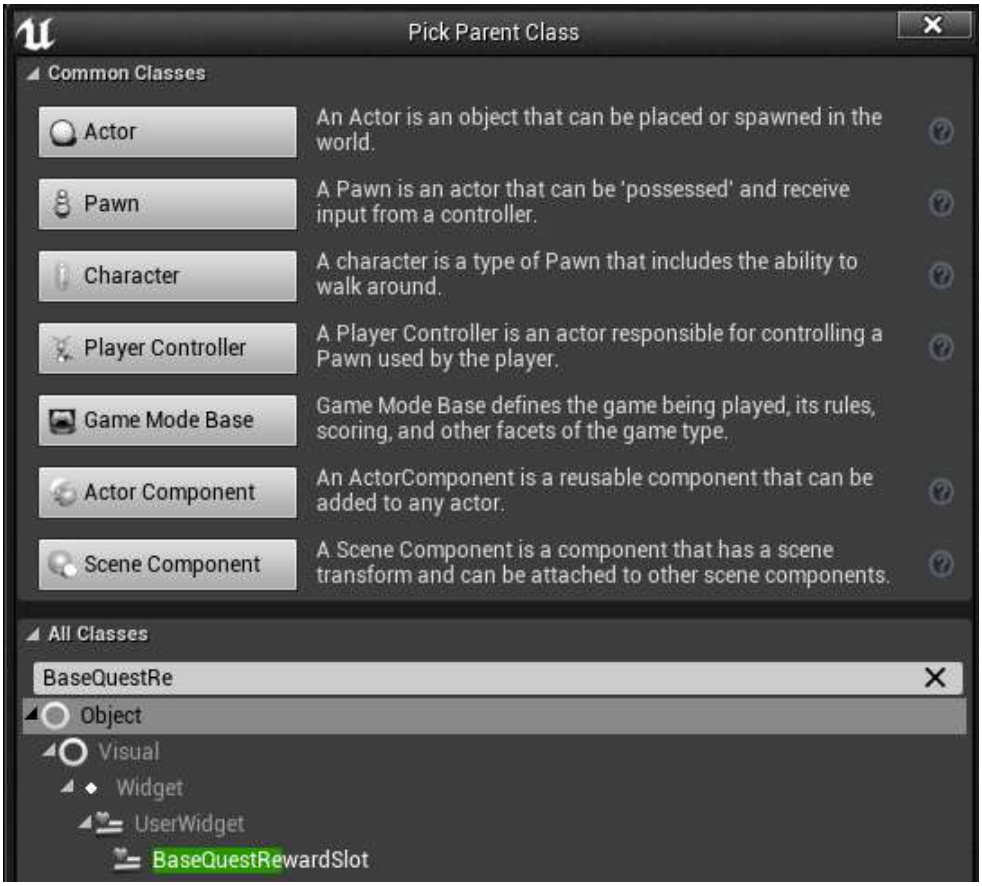


This is how you create one of these tabs. You can also use the example one as a starting point and modify it for your game.

Back to top.

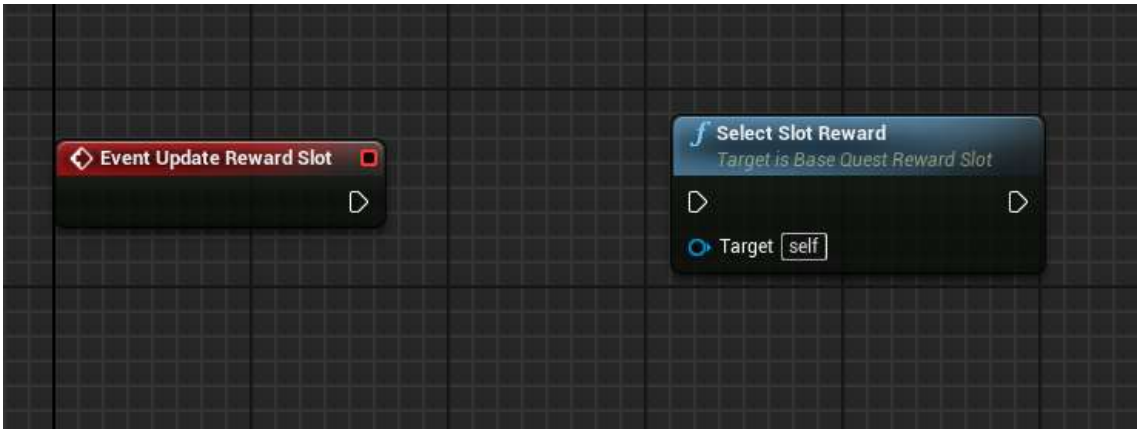
3.1.1.2.10.4 BaseQuestRewardSlot

This is a widget from which you should derive all the slots for reward selecting in your game UI. You can create with this any UI for displaying each reward in the reward tab, with very few and easy functions that you can call and that will handle the reward selection and implementation.



This widget has a function, “UpdateRewardSlot”, that you should override to update it. The task data will be available to be used on this event.

It also has a function to handle the reward selection that is called “SelectSlotReward”, and should be called, for example, on button click, to handle the task completion via selection of the reward that the widget represents.



These are the functions that you can use in the slot for handling update and reward selecting.

[Back to top.]

3.1.2 Node connections

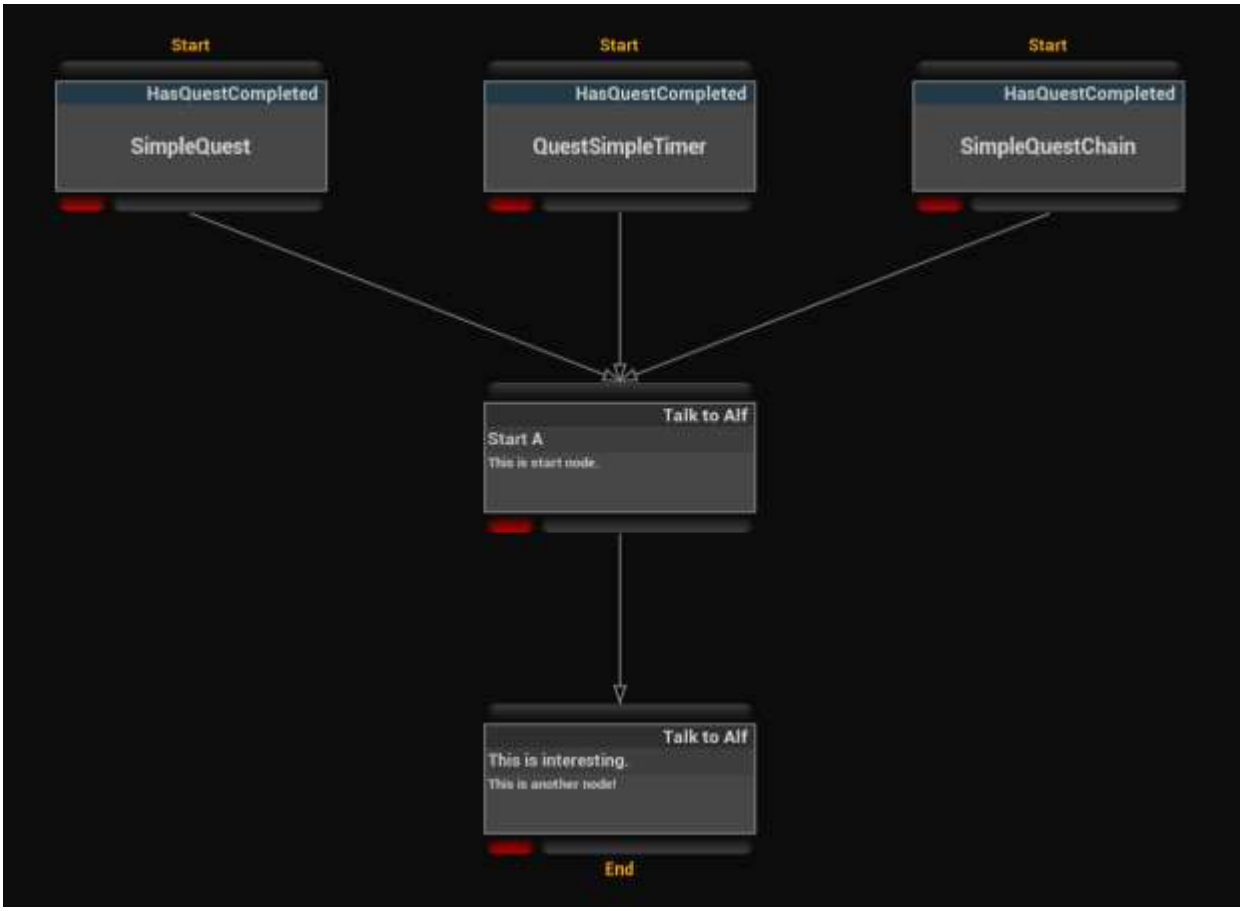
The nodes interact between them via connections o links. With this links it is possible to control and setup the flow of the quest or dialog.

Links can be of three main types.

3.1.2.1 Required links

The required links show what nodes are necessary to complete for the node connect to activate. A node cannot activate if any of the required nodes connected to it via a required link is not completed.

These links are shown as complete lines in the Quest Editor.



In this quest, all nodes are connected with Required links. In particular the three starting conditions are, which means that, in order to start this quest, you will need every single of these conditions to be true.

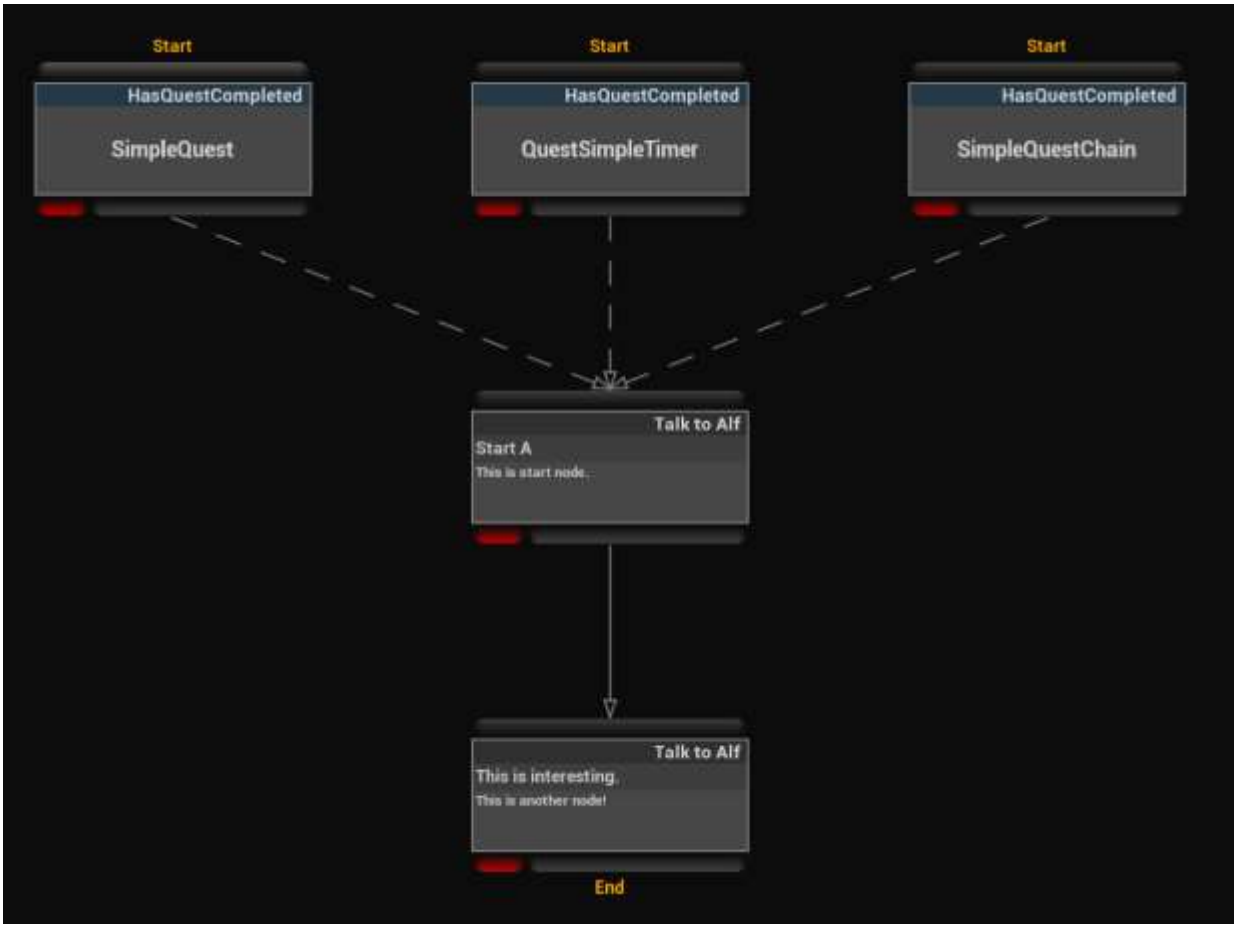
Back to top.

3.1.2.2 Optional links

These links make the relation between nodes more flexible, and nodes connected with this link are not need to be completed in order for the connected node to be able to activate. These nodes are useful for

creating tasks that are optional in the quest. The flow of the quest could be later by modified by the state of the optional node if needed.

These links are shown as segmented lines in the Quest Editor.



In this picture you can see that all starting conditions are using Optional links. This mean that for this quest to start, you need one of these conditions to be true, no matter which one.

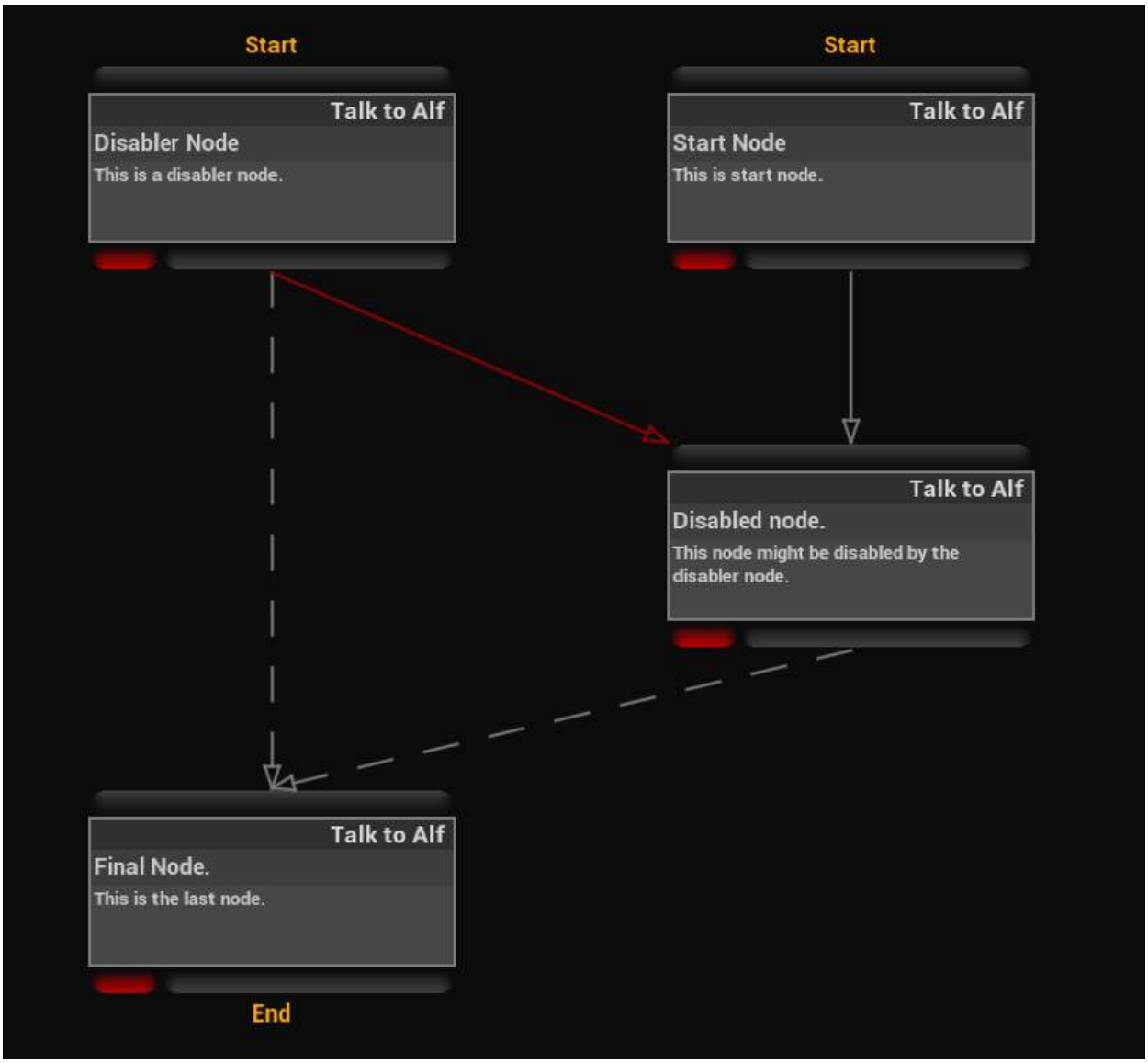
Back to top.

3.1.2.3 Disabled links

These links allow to disable nodes of a quest based on the completion of others. This way, they will affect how the nodes are presented to the player and will be useful for creating different paths inside the quests, having a type of “decision matters” system.

These links are shown as red in the Quest Editor.

The following is an example of a disable connection. The arrow shows which node is disabling the other upon completion.



Back to top.

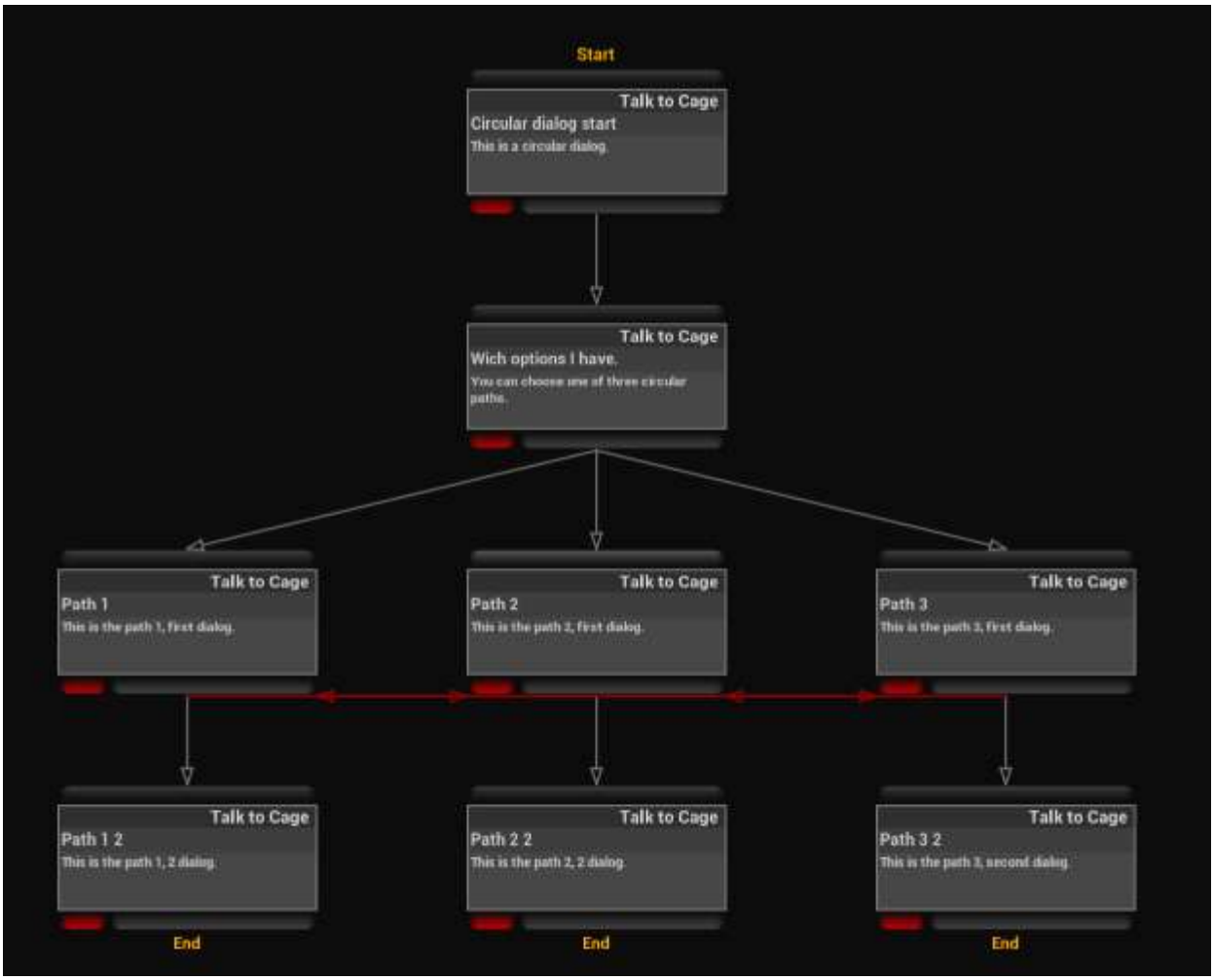
3.1.2.3 Circular links

You can create circular connections in the system. This means that a node can Re-Active nodes that have already been completed by the player.

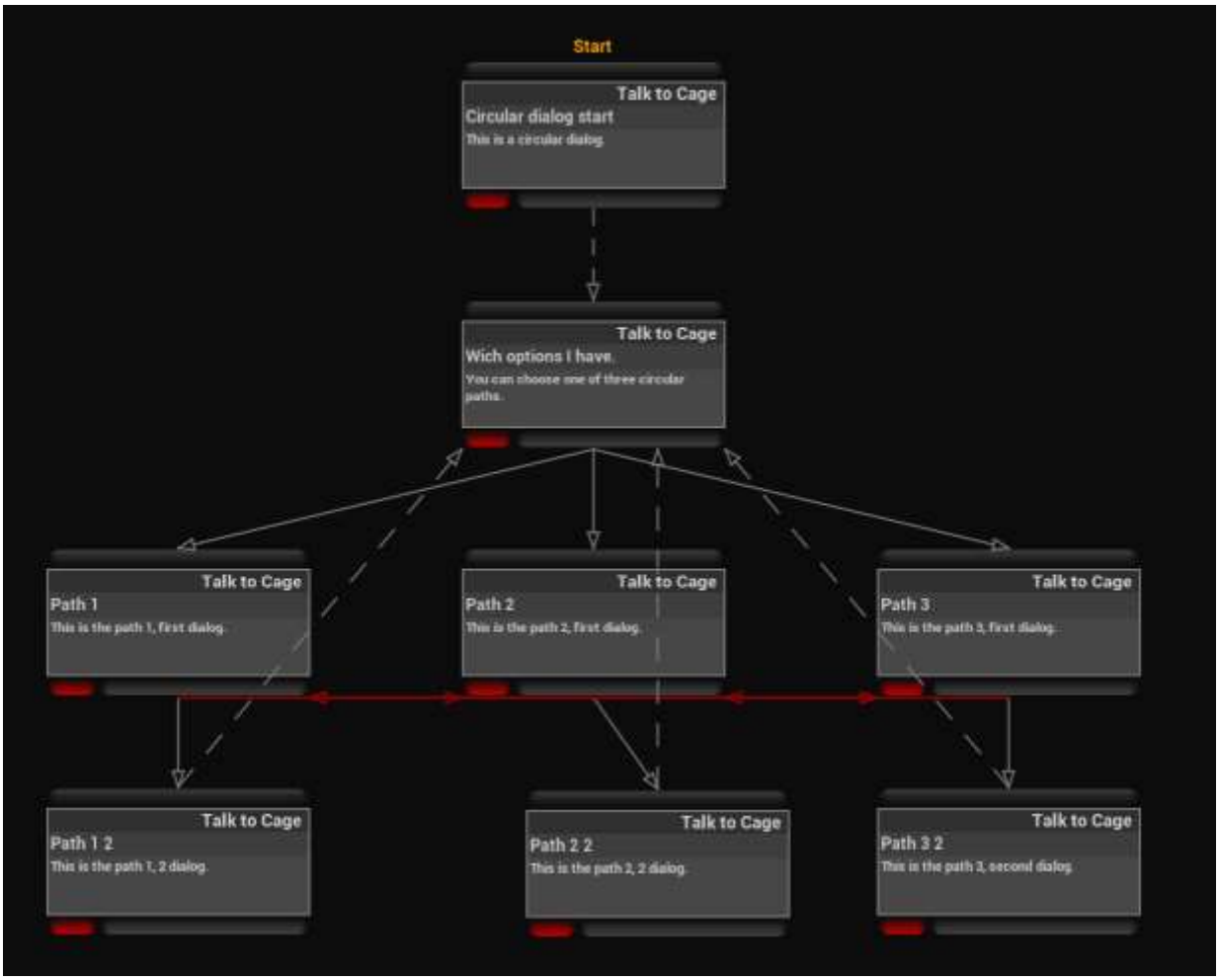
Using this is possible to repeat certain parts of a quest for the player, like a conversation where you explain something and have the chance to re-read it.

This is not a specific type of link, but more of a way of connecting nodes in a quest, and does not look any different than other connections.

You can create even quests or dialogs that cannot end and repeat forever. Of course, you have to be careful with this, specially if using it in quests (not dialogs).



Take a look at this image, this is a normal quest that have three options for the player. Once the player selects the path, the others are disabled and they cannot be seen or completed by the player anymore. After one path is completed, the quest ends.



This is the same quest, but now all end nodes had been connected to the core node, that gives the player all the different options. It is also important to note that all the connections to that node are optional now, otherwise you could prevent that node from activating in the first place. You can also see that this quest has no more end nodes.

This quest will never end, it will continuously give the player the paths to choose or the nodes of that path in particular.

This functionality can be very useful for dialogs or for side paths in quests.

The quest must always have a start node, so keep that in mind when making this type of connections. In this example, the core node is purposely separated from the start node for that reason.

This type of connections behaves like the task “ResetTrackToNodeWithDisableds”, using that task or this connection is the same thing, but this can be faster.

[Back to top.]

3.1.3 Nodes in connections.

The connections between the nodes determine also the type of node they are, relatively to other nodes. In a node connection, there are three types of different nodes.

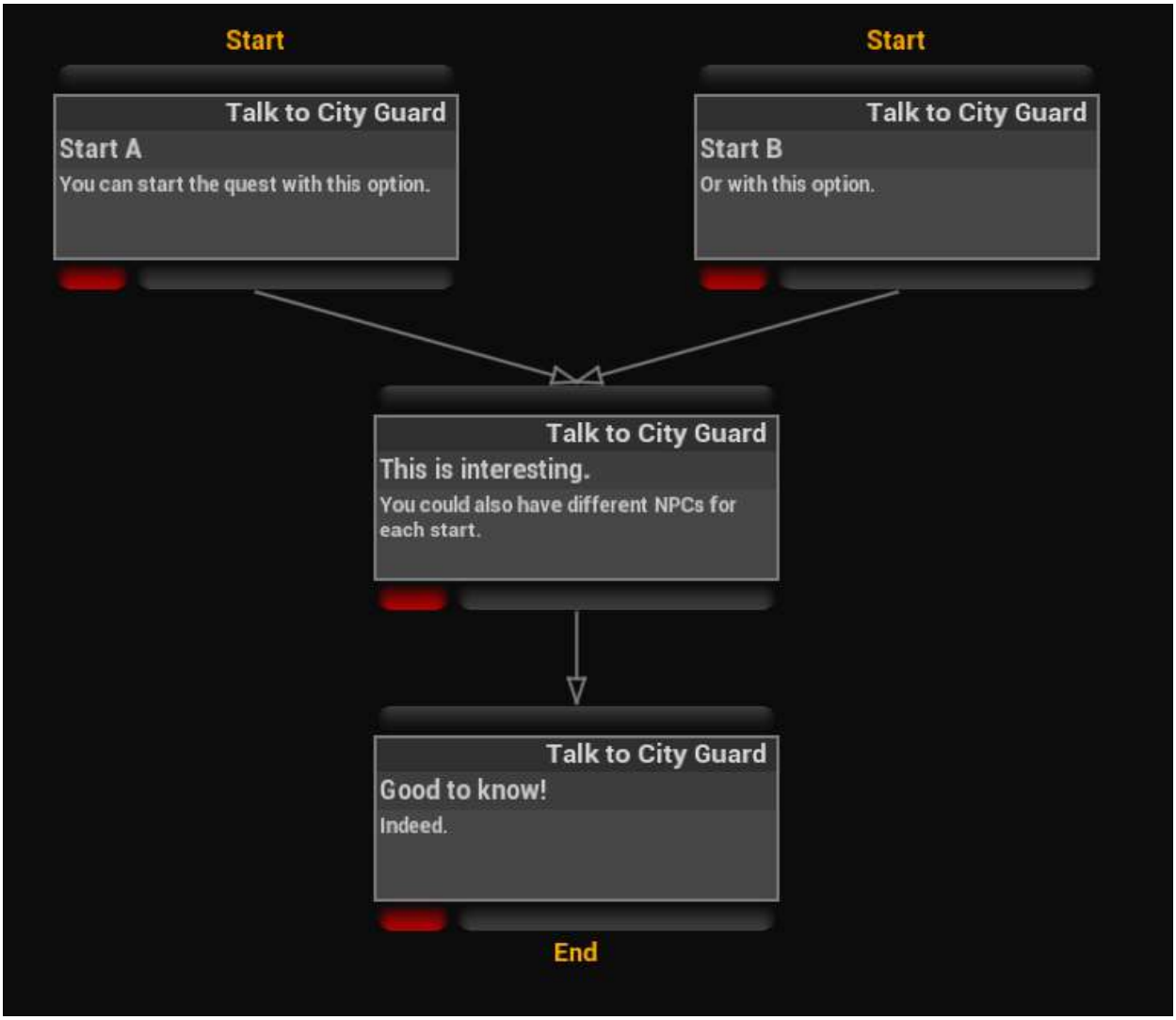
Back to top.

3.1.3.1 Superior Nodes

Given a node, those nodes that are connected to its superior slot are considered superior nodes of that initial node as shown in the picture.

In this picture, “Start A” and “Start B” are superior nodes of the middle node. The middle node is also superior node of the ending node.

The quest flows from superior nodes to inferior nodes and it uses the links to know what nodes must activate after one node is completed.

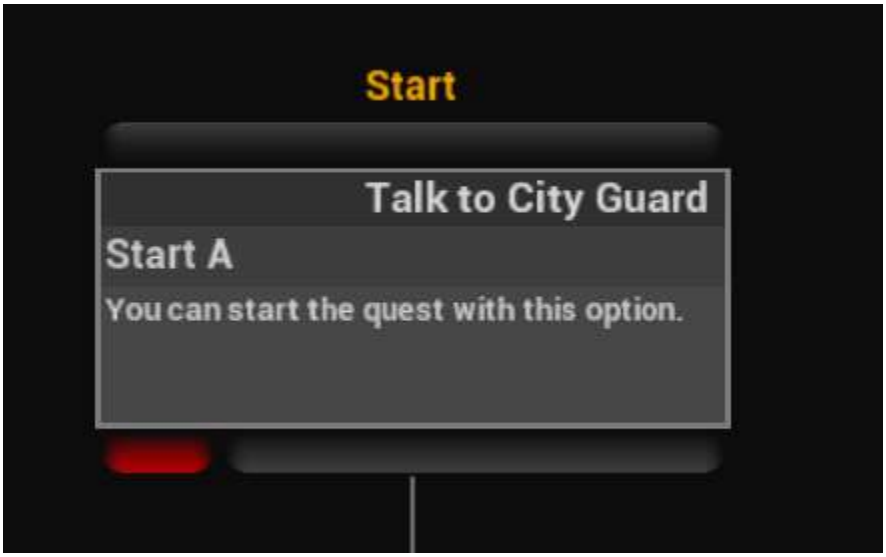


Those nodes that don’t have any superior node are considered as Start nodes for the quest.

It is important to know that when a quest is activated, all start nodes are activated as well.

To create a superior node for a particular node, you need to click the superior button of the node that is brown colored and then click the inferior button of the node that you want to be its superior node.

You can have any amount of start nodes in a quest.



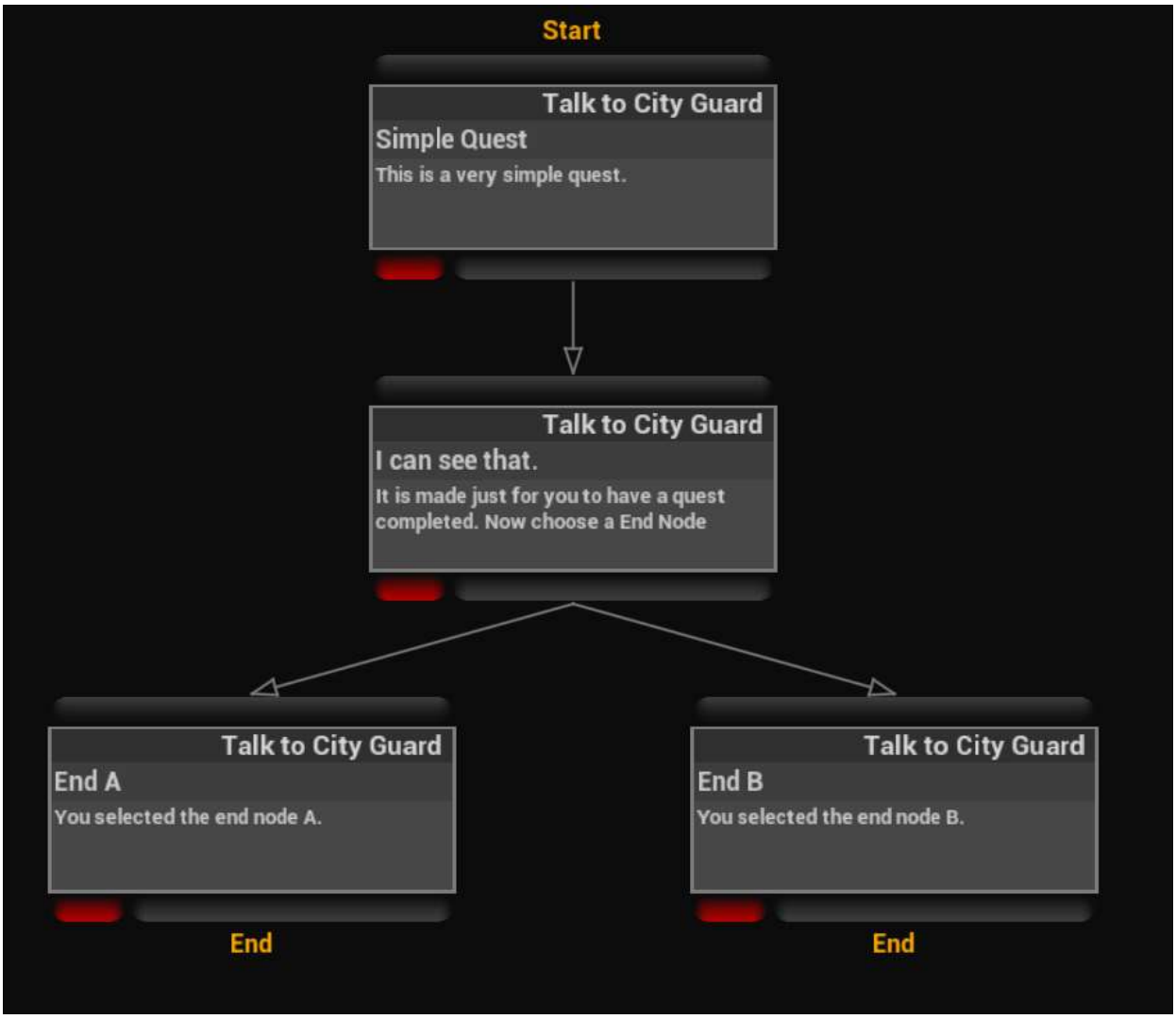
This is a single start node.

Back to top.

3.1.3.2 Inferior Nodes

Given a node, those nodes that are connected to its inferior slot are considered inferior nodes of that initial node as shown in the picture.

In the following picture, the “End A” and “End B” nodes are inferior nodes of the middle node, and this is also an inferior node of the Starting node.



Those nodes that don't have any inferior node are considered as End nodes for the quest.

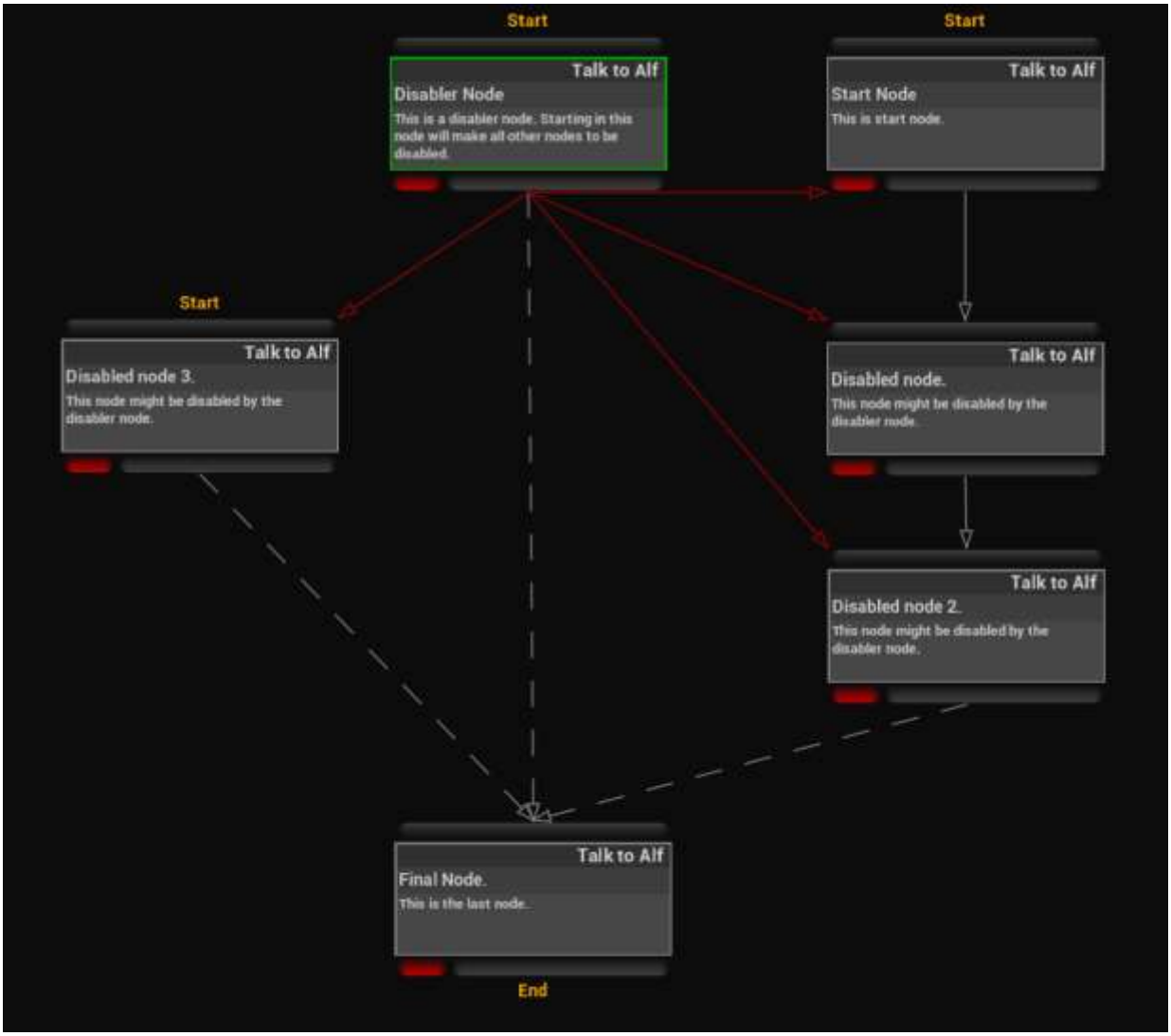
The way to create inferior nodes for a node is homologous to the explained for the superior nodes.

Back to top.

3.1.3.3 Disabled nodes

When a node is completed, it has the ability to disable other nodes. To select the nodes (disabled nodes) that this node (the disabler node) will disable on completion we will use the red buttons of the nodes. The first disable button clicked will determine the disabler node and the second one the disabled node.

This is an example of disabling connections. The arrow indicates which node is the disabler and which is the disabled one. The disabler node in the image is disabling almost all other nodes. If you start the quest in the disabler node, you will be only left with the final node as option in this quest.



The links adapt base on the relative position of the nodes involved, and is the same if the link points to the upper or the lower part of the node.

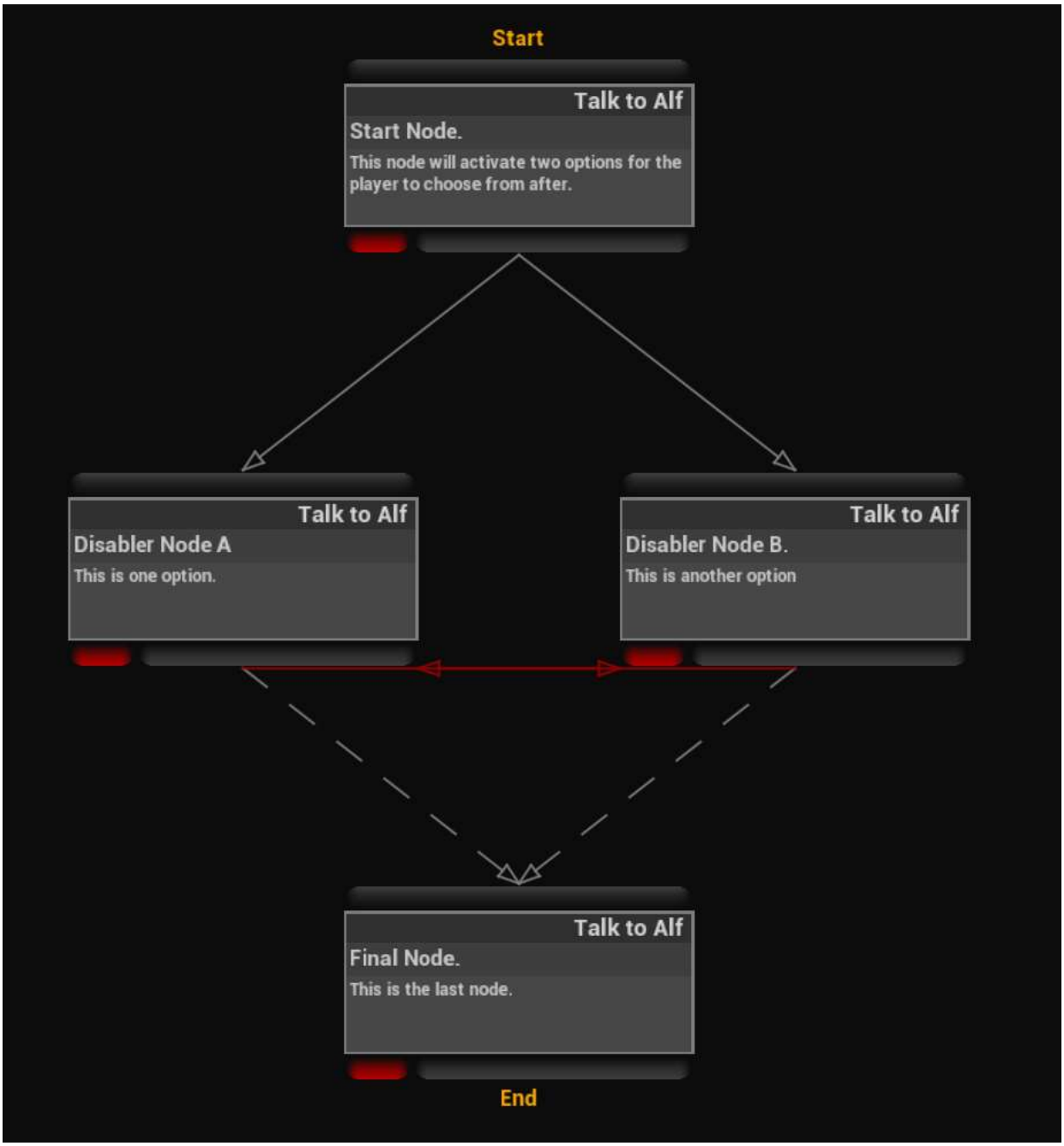
There is no limit in the number of nodes that a node can disable.

This type of connection is useful to block parts of the quest for the player base on the completion on certain tasks, shaping the path that the player takes inside the quest and allowing for more complex quests, with meaningful decision making and different endings.

Back to top.

3.1.3.3.1 Mutually disabled nodes

This is a particular case of the disable connection, that occurs when given two nodes, both fulfill the disabled and disabler roles.



The above picture shows a mutually disabled connection between nodes. Based on the decision of the player when selecting which talk option trigger first, a complete path of the quest will be “blocked”.

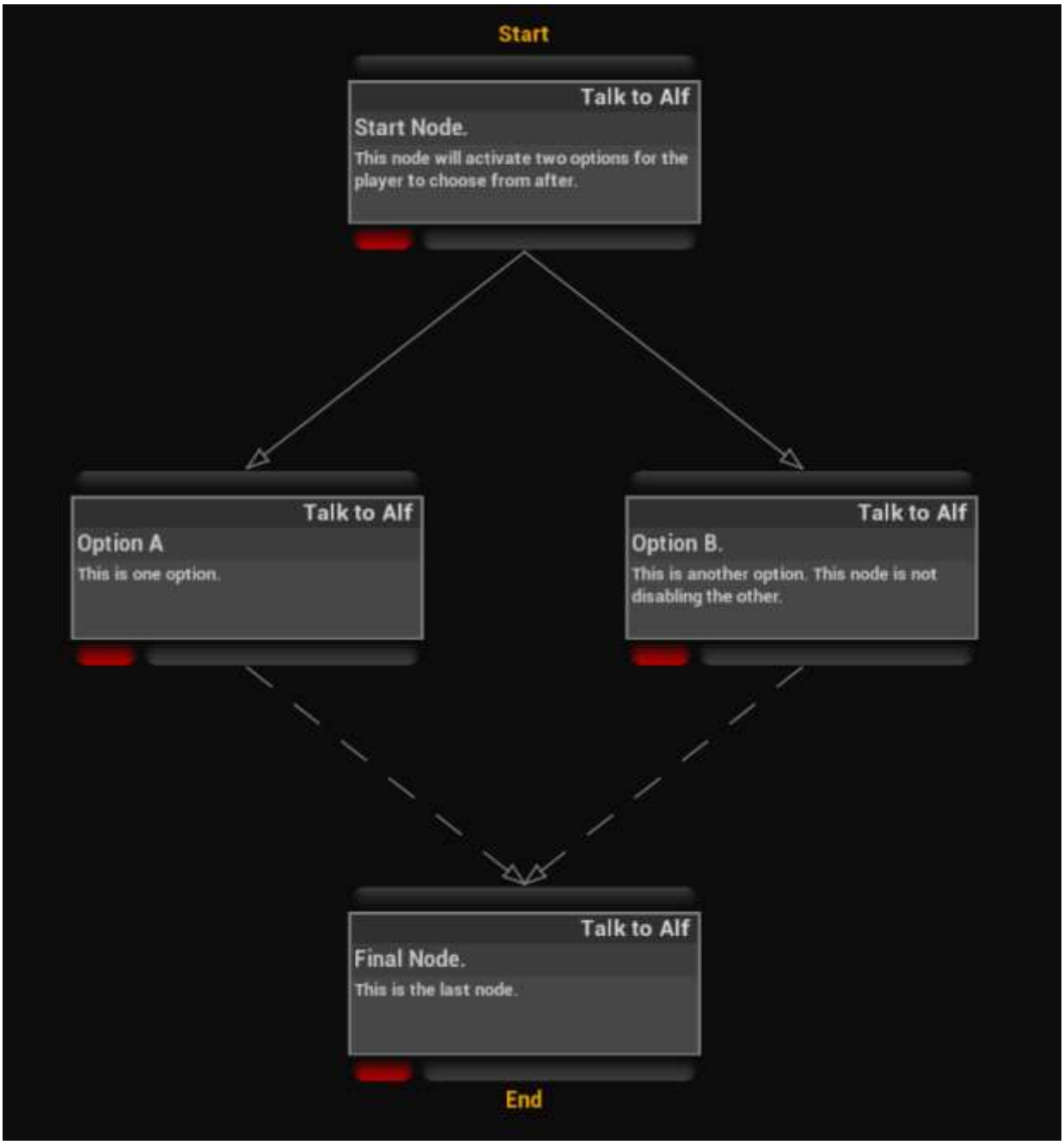
Note that the last node is connected with optional links. This is a must in this situation, since it would be impossible to have both nodes completed at the same time.

Sometimes, to show the connections more explicitly, you can move one of the nodes slightly up or down, so the arrows are separated.

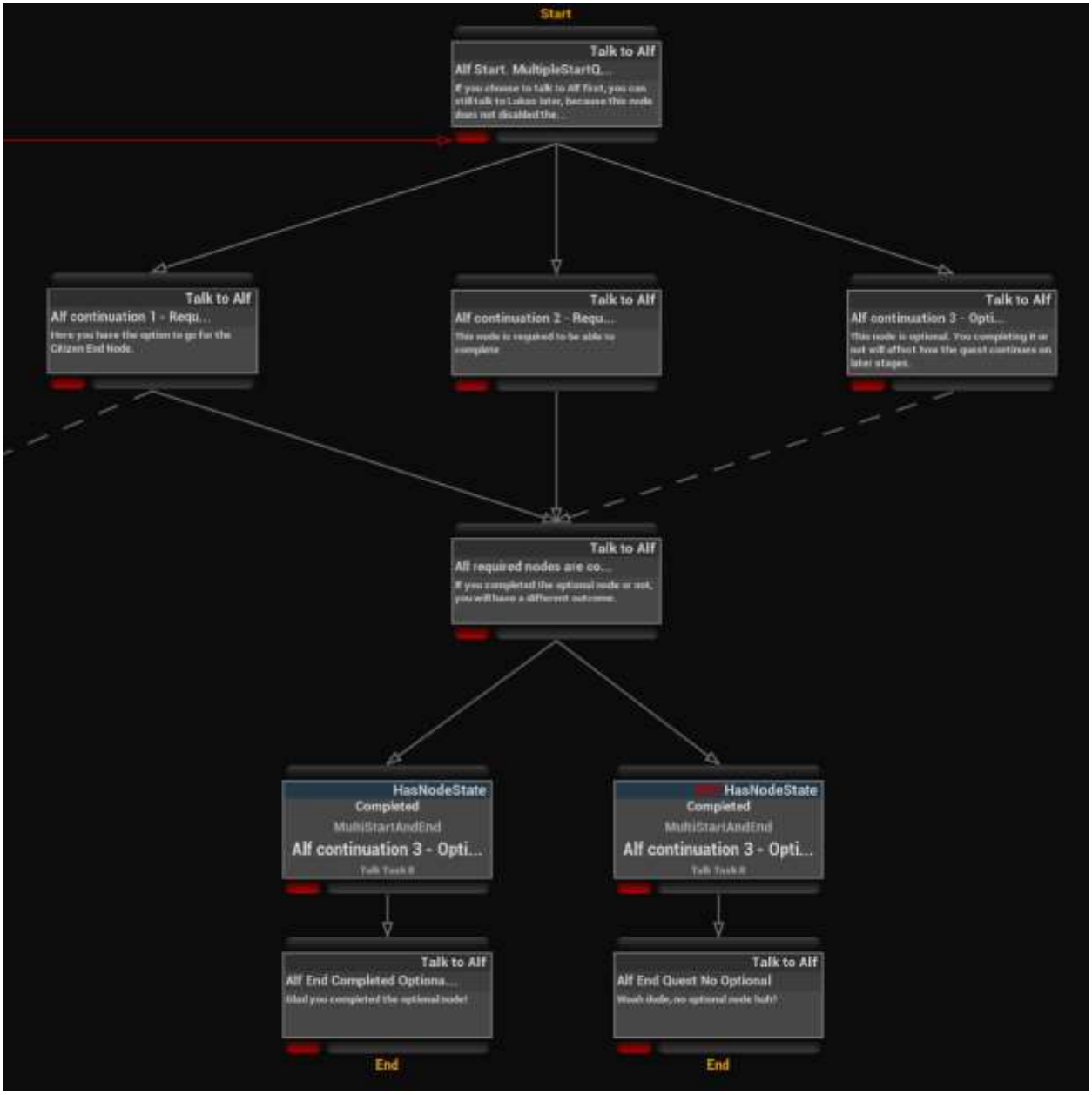
Back to top.

3.1.3.4 Exclusive connected nodes

These are nodes or groups of nodes that are connected exclusively to a node and don't share inferior or superior connections with other nodes. This constitutes a path that can lead to nowhere if said node is completed. These nodes will be automatically disabled if the node completes, since they don't add new paths to the quest if they are completed afterwards.



As you can see in this example, the left middle node and the right middle node are both exclusively connected to the lower node. If one of the nodes is completed and since they are connected with an optional link, the remaining node will be disabled by the system and will no longer be accessible for the player, for example, in the talk options.

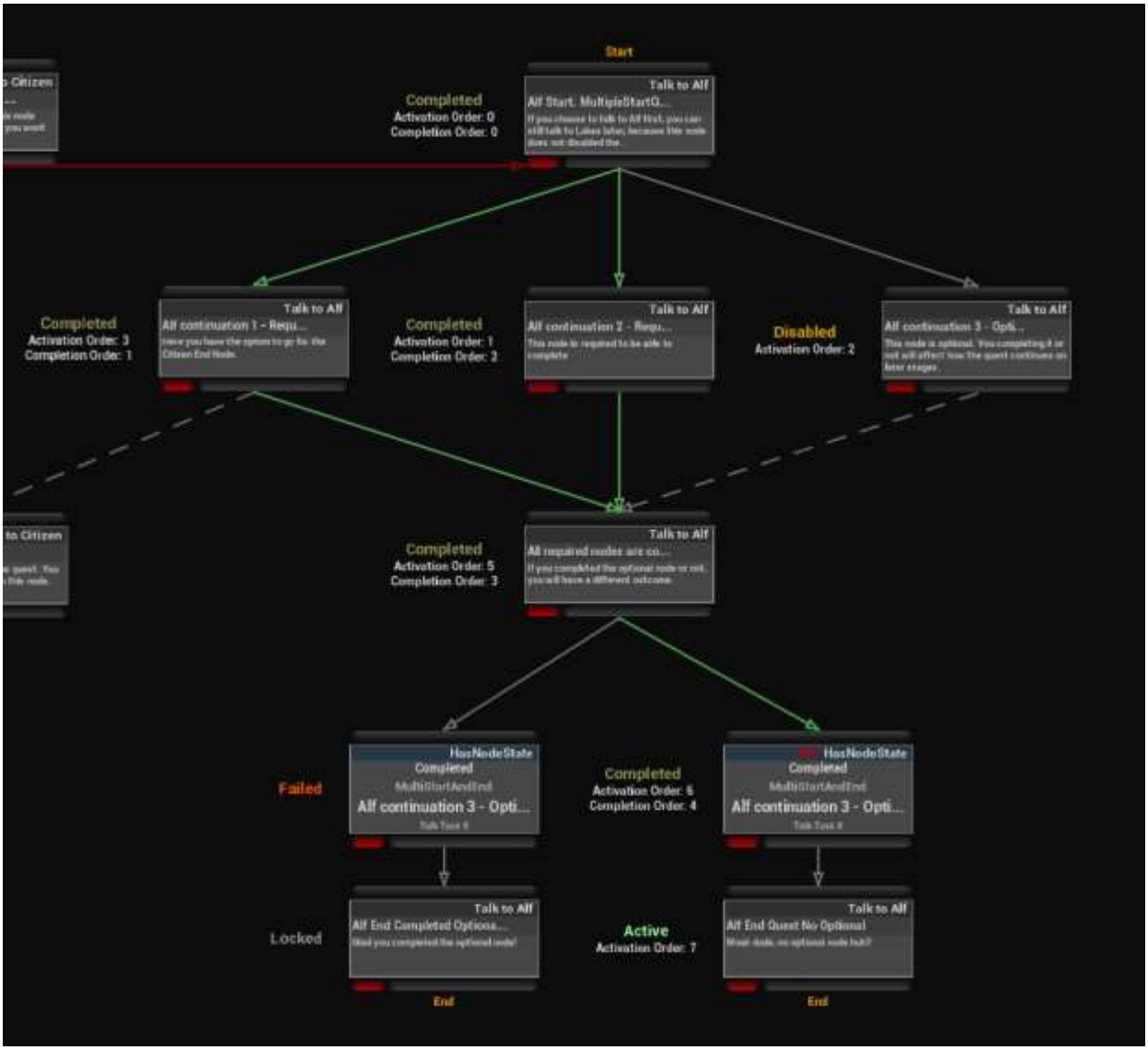


This is a more complex example for exclusive disabled nodes. Let's say the player start the quest with the "Alf Start" node. At this point all the inferior nodes "Alf Continuation 1", "Alf Continuation 2" and "Alf Continuation 3" are activated. Then the player completes "Alf Continuation 1" and "Alf Continuation 2", which both are required for the following node. The player does not complete the optional node "Alf Continuation 3".

If the player completes the last node "All required nodes are con…", three different things will happen to its superior nodes.

The nodes "Alf Continuation 1" and "Alf Continuation 2" have already been completed by the player and because of this they will remain completed. This state cannot be changed.

The node “Alf Continuation 3” is still active, since the player did not complete it, but is also only connected to the last node “All required nodes are con…”. Because this last node is already completed, keeping “Alf Continuation 3” active is no longer needed, as it is not useful anymore for the quest. This node is considered Exclusive Connected Node and it will be disabled.

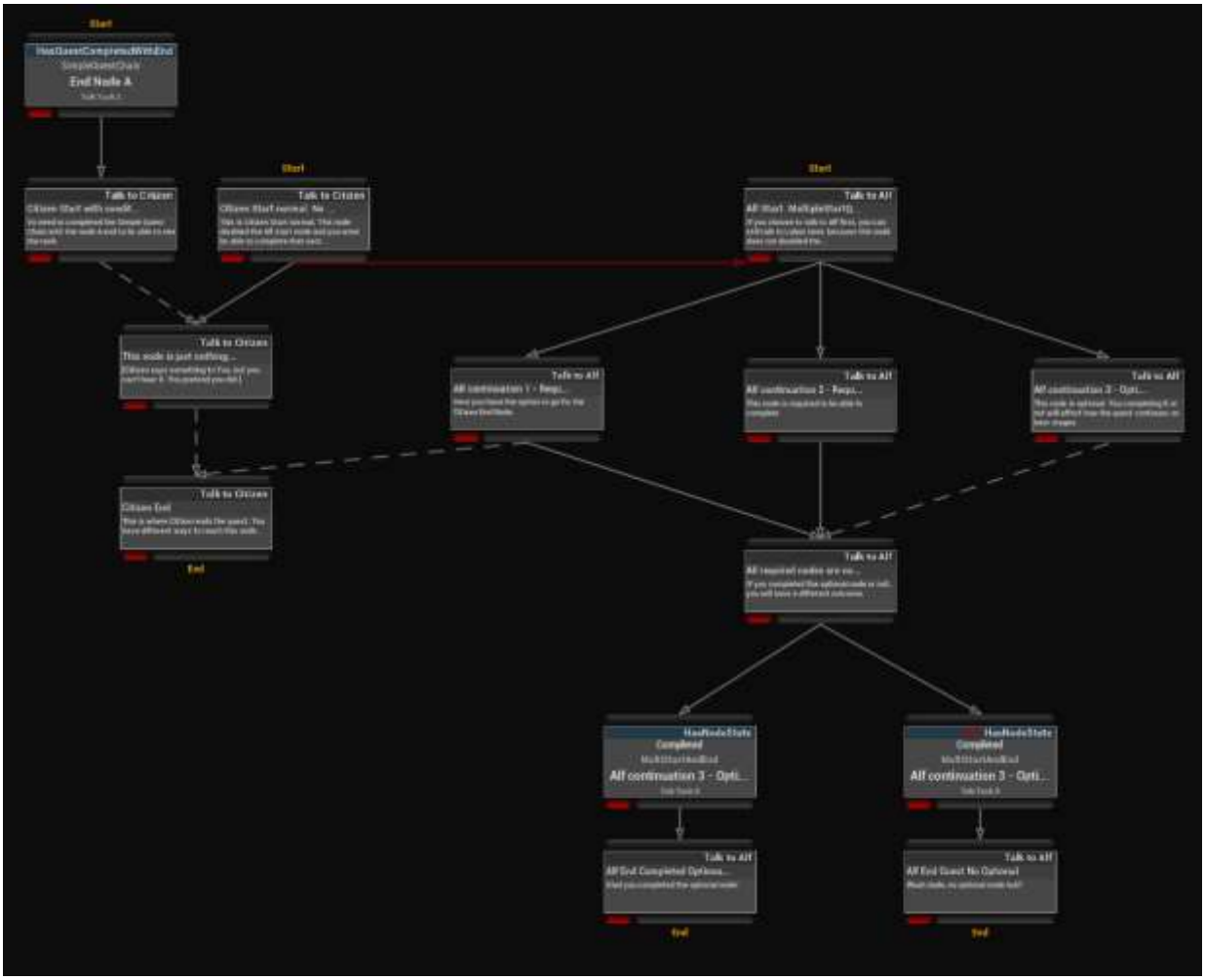


This picture shows the example using the debug tool.

[Back to top.]

3.1.3.5 Conversations

A connected group of talk tasks nodes (via inferior or superior connections) of the same NPC is called a conversation. Two talk tasks that are separated for a task that is activate condition only are also part of the same conversation. This allows to hide parts of a conversation to a player upon certain conditions.



In the image two conversations are shown, one for the NPC “Citizen” at the left and one for the NPC “Alf” at the right. This last one is selected and you can see that consist of seven nodes. The last two nodes at the bottom are separated by tasks from the rest of the conversation, but given these tasks are “Activate Condition Only” they are not splitting these nodes into two different conversations for the NPC “Alf”.

The conversations are identified with the talk task node of lesser ID.

Conversations events are only client side.

Back to top.

3.1.4 Node states

The nodes can have different states depending on the quest current state and the path that the player took inside it. The state of each node is saved in the Quest Manager for each quest inside the “QuestsTrack” variable.

Back to top.

3.1.4.1 Locked state

This is the initial state of all nodes. When a quest is activated, all the nodes that are Start nodes of the quest that can activate will change their state to Active.

Back to top.

3.1.4.2 Pre Active state

This is a state that only starting nodes can have. When a quest is activated, all starting nodes that can activate will go to the pre active state.

In this state, they can activate directly, for example if this is a talk task that was already committed by the player or remain in Pre Active State, waiting for the right moment to activate. This is mostly used for multiple starts of quests and is more of an internal state that is not really important for the gameplay, and could be considered as “Locked”.

Back to top.

3.1.4.3 Active state

This is the most important state of the node, during this state all the task objects will be spawned and the logic for tasks will be executed. These nodes are the ones that the player is interacting with at any given time. In this state, the nodes can change their state to several others.

Back to top.

3.1.4.4 Completed state

This state can only be achieved by an active node that ended with a successful result. When a node enters in this state, it can no longer change their state and will remain like this until the quest ends.

Back to top.

3.1.4.5 Failed state

This state can be achieved by an active node that ended with a not successful result.

A node can also reach this state when another node fails that its connected with a disable connection to this one. In other words, all nodes that fail will make all their disabled nodes to also fail.

A node will also fail if the activate condition returns false, in this case it will be changed from the Locked state directly to the Fail state.

Once on this state they will never change it again.

Back to top.

3.1.4.6 Disabled state

Nodes can be disabled by other nodes that are connected with them via a disable connection and they can change to this state from the Locked or the Active state.

Once on this state they will never change it again.

Back to top.

3.2 Quests and dialogs

A quest is a group of interconnected nodes, tasks and talk tasks. They can be assigned to NPCs as Quests but also as Dialogs. You could even assign the same quest as dialog and quest to the same NPC, although this would not make sense in most cases.

The only difference between them is how they are handled by the system. While quests are replicated, dialogs are not. This means that dialogs are executed client side only.

Dialogs also are considered “temporal” or “disposable” by the system, which means that the system can decide to stop tracking them if is needed. For example, if you reach the maximum number of active dialogs, and you talk to a new NPC that has a new dialog that needs to active, the system will remove the oldest dialog to make room, causing the player to lose progress in said dialog. In quests, this is handled differently, the player will have to complete or abandon a quest in order to accept a new one in similar circumstances.

Other than that, dialogs and quest share their functionality and all principles and rules apply to both.

Back to top.

3.2.1 Accepting quests

For a quest to be eligible for the player to be accepted, it has to have at least one initial task that can be activated.

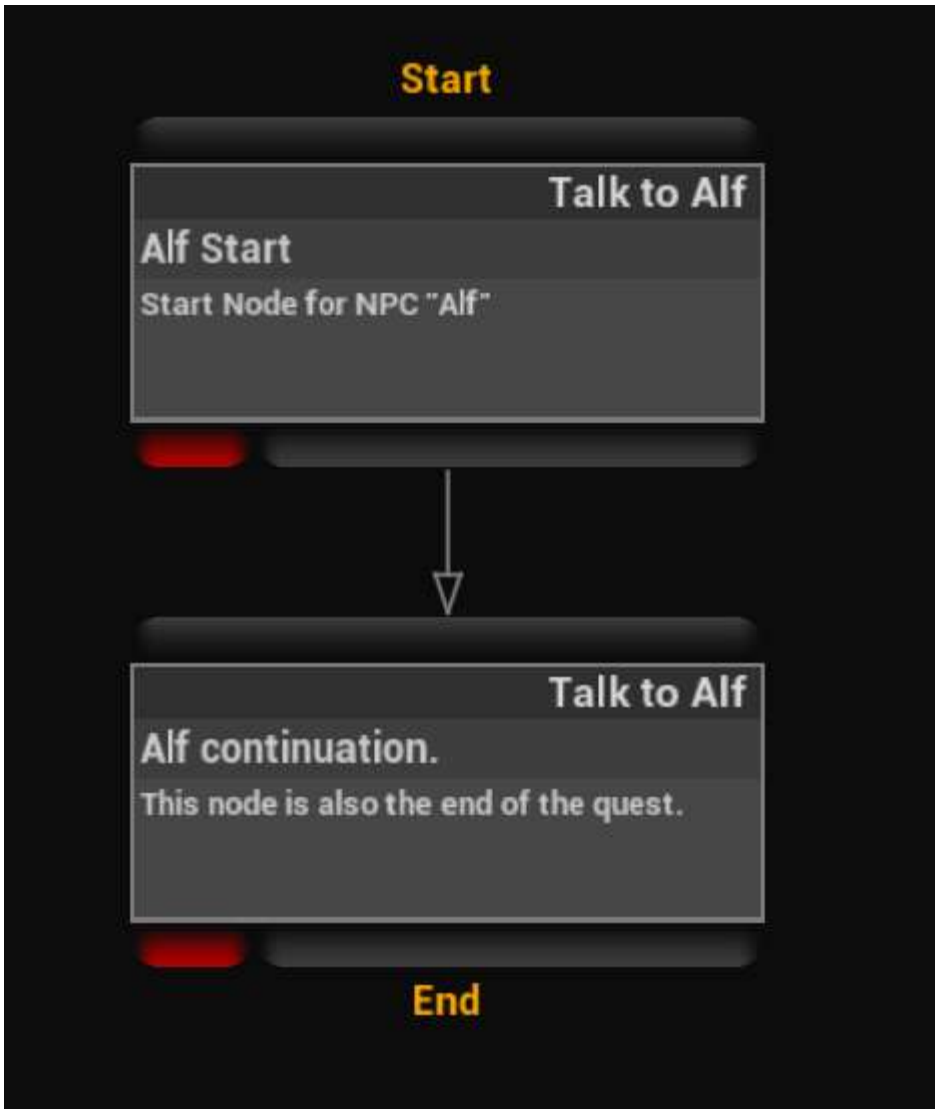
On accepting a quest, all initial tasks that can activate will do. There is no limit for the number of initial tasks that a quest can have.

Back to top.

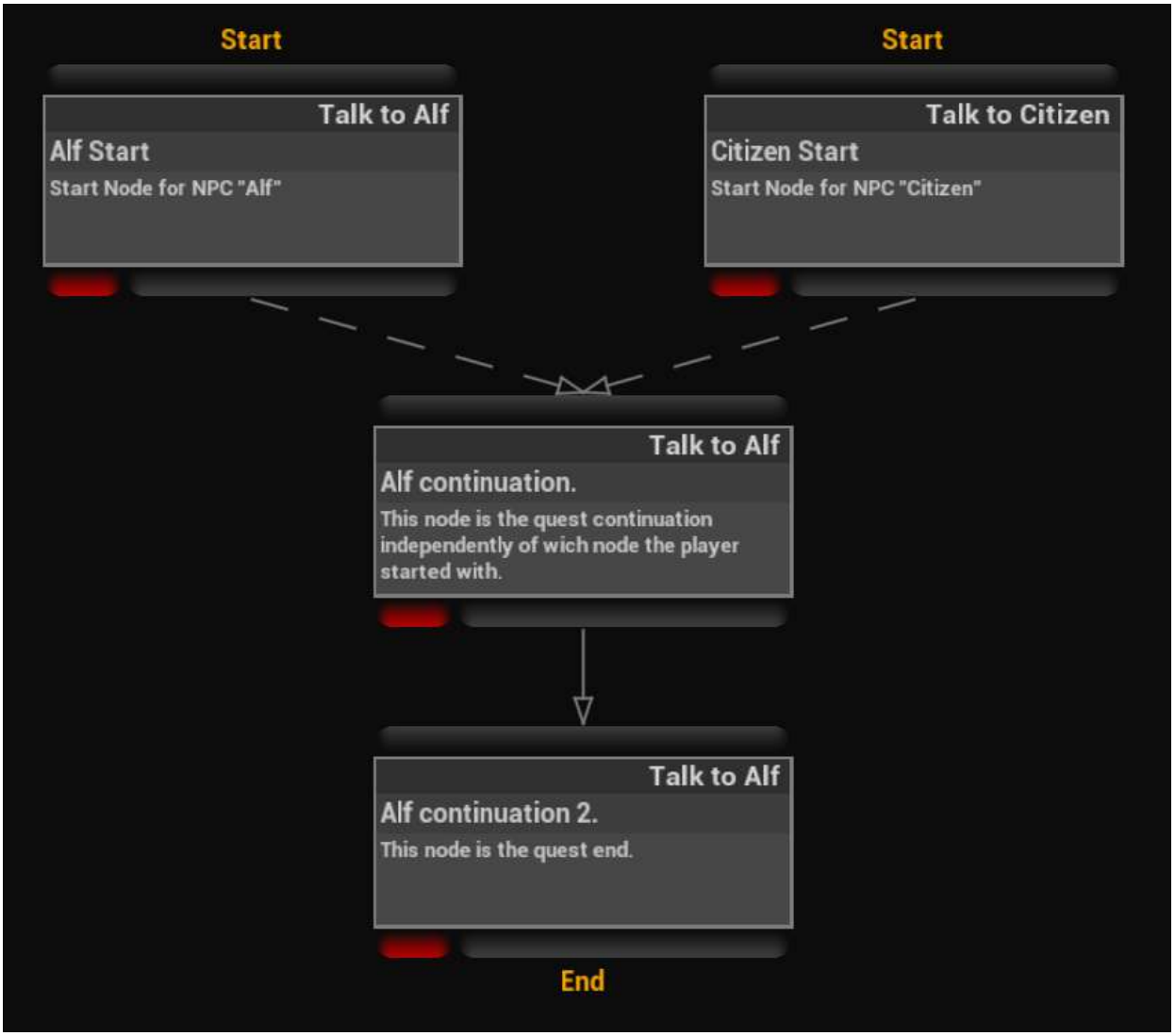
3.2.1.1 Accept quest via talking

This is the standard method for accepting quests. The player will accept the quests selecting the talk text options provided in the talk tab.

For a quest to be able to be accepted this way, it has to have at least one Start Talk Task node assigned to the NPC that we are talking to.



The quest shown in this picture for example is a quest designed for the NPC named “Alf”. Any other NPC can have this quest, but they will not show the start to player and therefore the quest will never be visible.

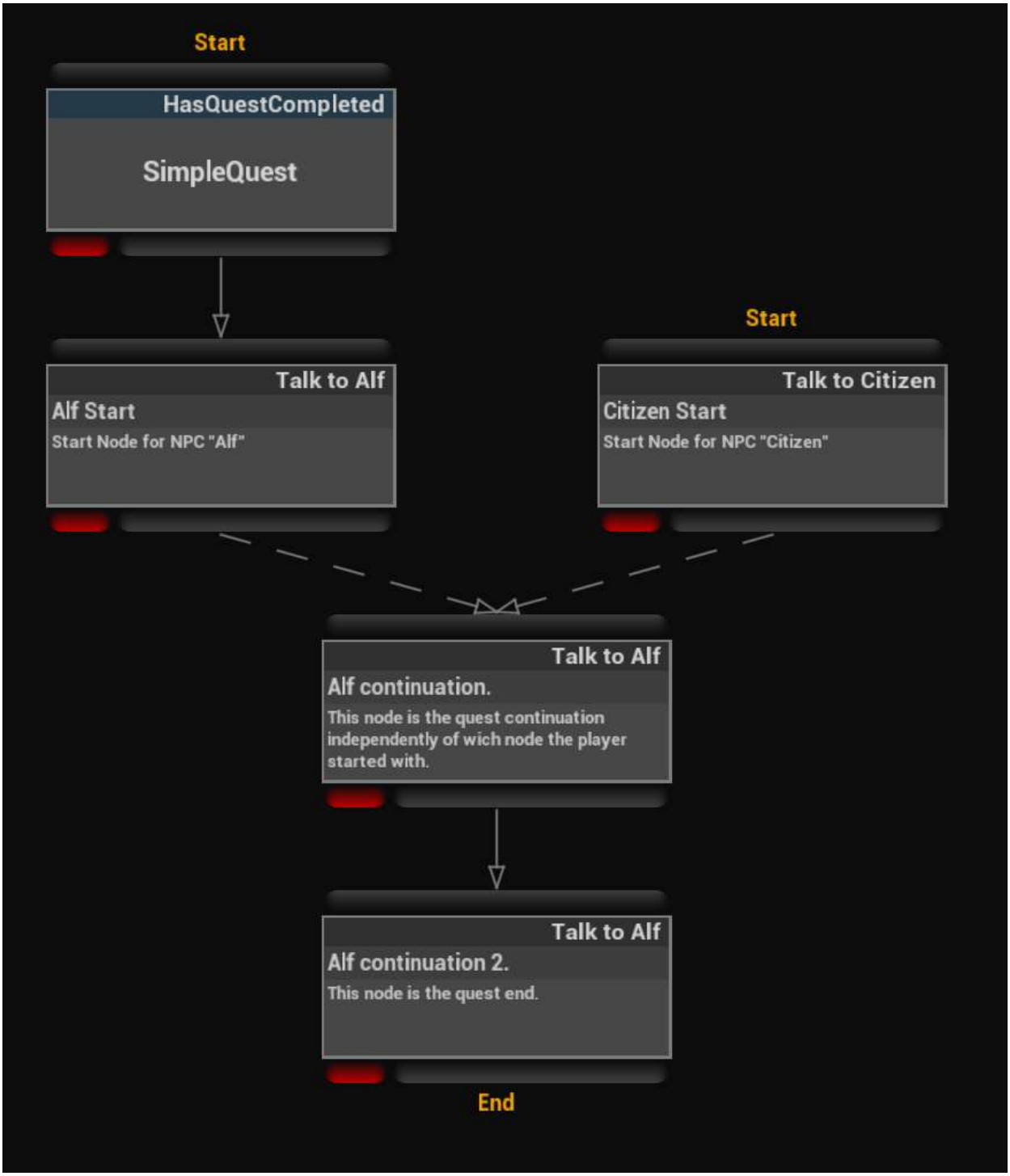


This quest has two different start nodes, for two different NPCs. Both NPCs can have this quest and they will be able to show their start node while talking to them.

You can even have multiple start nodes for the same NPC and all of those will show simultaneously in the talk tab.

As mentioned earlier, it is possible to add ActivateConditionOnly tasks before Start nodes and that will still work as shown in the following example.

Nodes that have conditions before them are considered “Pseudo Start Nodes”, since they have superior nodes, but they are “Activate Condition Only”. Nodes that are “Activated Condition Only” and starting nodes of a quest will be executed before the quest is accepted.



In this example, if the player has the “SimpleQuest” completed, he will not see any difference in the texts that are shown. If the player has not completed that quest yet, he will not be able to start the quest from the NPC “Alf”, but he will be able to do so from the other NPC.

Back to top.

3.2.1.2 Accept quest directly

Sometimes you will want to add quests to the player inventory directly, without having the player to talk first with an NPC. A good example of this is when the player is exploring a cave and suddenly, he or she decides on overlapping some weird looking voodoo altar, to find more information about it and potentially make some friends in the process.

You can accept any quest with this method, but usually you want to leave the quests that start with talk tasks to be started by NPCs via talking. If you start a quest that has only Talk Tasks as starting nodes, the quest will be added to the quest inventory, but you will have to still talk to the NPC to complete the starting task.

To accept a quest directly, you need to call the function “AcceptQuestDirectly”. The quest will have to meet the activation conditions in order to be added as an active quest.

You can use the Quest Start Location actors to give quests via this way.

Back to top.

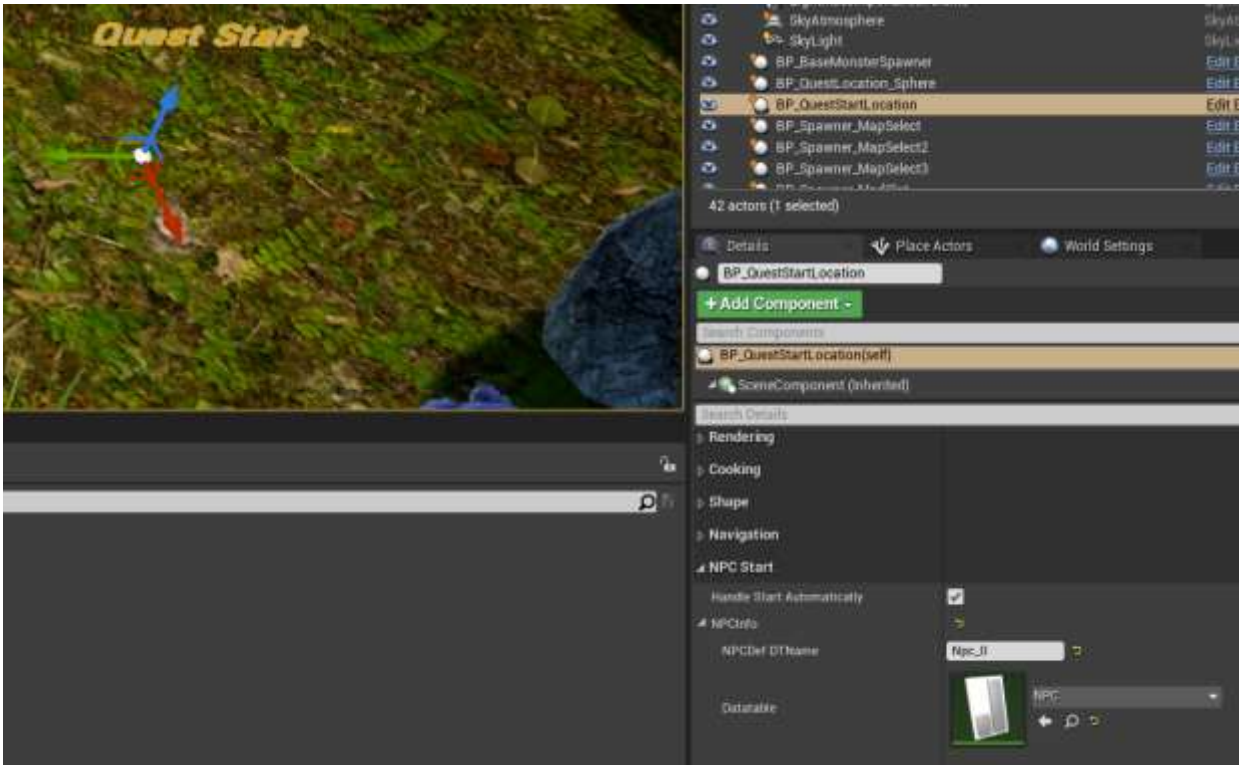
3.1.3.3.1 Quest Start Location Actors

These are actors that are designed to give quests to the player upon reaching them. They attempt to give all the quests assigned to them directly to the player without the need to talk to an NPC when they are overlapped.

They are designed especially for quests that start with tasks and not talk tasks. If you add quests that start with talk nodes, they will be ignored. These quests should be added to NPCs that can interact with the player via the talk tab.

They work in a similar way to the NPCs, in the sense that they use a NPCQuestDefRep structure and an NPC manager. Once you placed them on the map, you need to make sure to set the NPCInfo for the actor with some previously designed NPC in the Quest Editor, in the “NPC Start” category, in the same way that you set normal NPCs.

This allows to generate NPCs via the Quest Editor that are designed specifically for these purposes. It is recommended that you create the NPCs for Quest Start Location Actors in different datatables from the normal ones so is not confusing when wanting to add NPCs to talk tasks, since the NPCs designed for this purpose should not be used to interact with player via talk tasks.

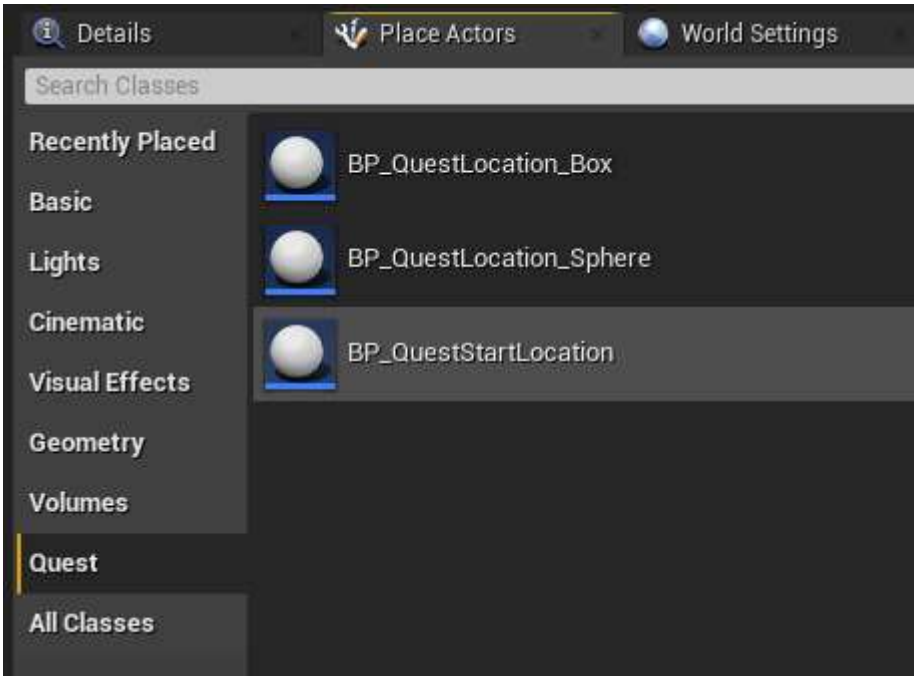


These actors provide some overridable functions in case your setup is not the standard one, such as a get Quest Manager of overlapping actor.

If you placed the Quest Manager in the player state and your character is a pawn, they should work by default.

There is also a condition that you can override in case that you want to filter the actors that can overlap the Quest Start Location.

You can place the location actors directly from the “Place Actors” tab, under the “Quest” category.



Back to top.

3.2.2 Ending quests

There are several ways for a player to end a quest.

Back to top.

3.2.2.1 End quest via completion

This is probably the method more desired for the player to end a quest. This happens when an End node is completed, and therefore the quest also completes.

All the quests that are completed in the system will not be able to be completed again for the player. A track of all completed quests is stored in the QuestManager. You can use this track to show information about completed quests in the UI.

To be able to be completed again by the player, the quest must be eliminated from that track via some provided functions.

Back to top.

3.2.2.1.1 Quest score

A quest score is calculated as the sum of all tasks scores, from tasks completed by the player when doing said quest. This allows to give rewards accordingly that will need to be implemented for your particular system, such as gold and experience.

The task score can be calculated inside the task objects, overriding the “GetTaskScore” function. This is purposely a function and not a variable, so you can dynamically calculate the score of the task based on the payload, for example.

The talk tasks also have a score that can be set in the QuestManager variables.

Back to top.

3.2.2.2 End quest via abandon

Sometimes the player is feeling lazy and decides that he or she no longer wants to complete the amazing quest that you created. If you are okay with this, you can allow this behavior implementing the “AbandonQuest” functions.

On abandoning a quest, the player will lose all the progress in it and will be able to accept the quest again in the future.

Back to top.

3.2.2.3 End quest via fail

When a task fails, all the disabled nodes of said task will also fail. It might happen that in this point, the player is left with no active tasks. When this happens, the quest fails. When a quest fails, similarly to what happens when the quest is abandoned, the player loses all the progress in said quest, which is added to a fail track.

The quest will remain in the fail state for a period of time that can be specified in the quest manager, that can be nonexistent or that can also be infinite. This allows to make permanent quest fails, and completely take away the quest from the player, or to add it right back to the NPC.

There area few tasks that aids in the control of the fail state inside the quests.

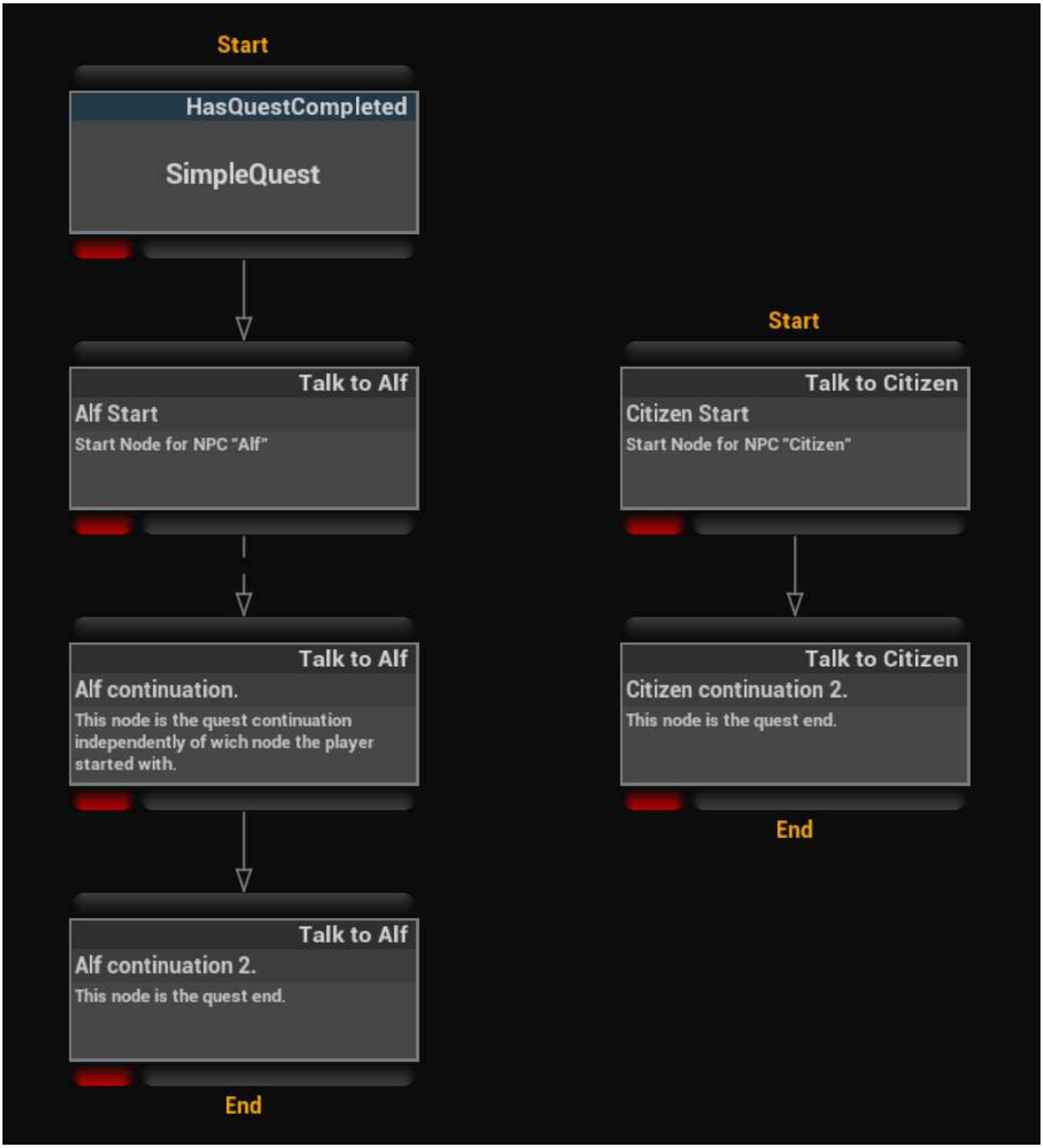
Back to top.

3.2.3 Parallel quests

Because of the flexibility of the system, it is possible to make two sets of paths completely independent from each other, like two quests in one.

You can totally do this and you will have your reasons to do this, but if you do, keep in mind that this will still be considered as one quest for the system and will behave like it.

The following is an example of parallel quests.



Back to top.

3.2.4 Task interfaces

A good way of adding custom functionality to your tasks is using interfaces. This is especially useful for working on the UI or for tasks that need to provide extra information.

The system includes a Quest Task Interface, that is optional and contains some general UI functions.

Back to top.

3.2.5 Talk texts

The TalkTextInfo structure is a way of providing all the “talk” options for a determined NPC and for the current player progression. This structure contains information about the NPC and player texts and some additional info.

This structure is designed to facilitate the implementation of the UI.

All the options for talk with an NPC, whether it comes from a dialog or some quest talk task, it is wrapped inside a structure called “TalkTextInfo”.

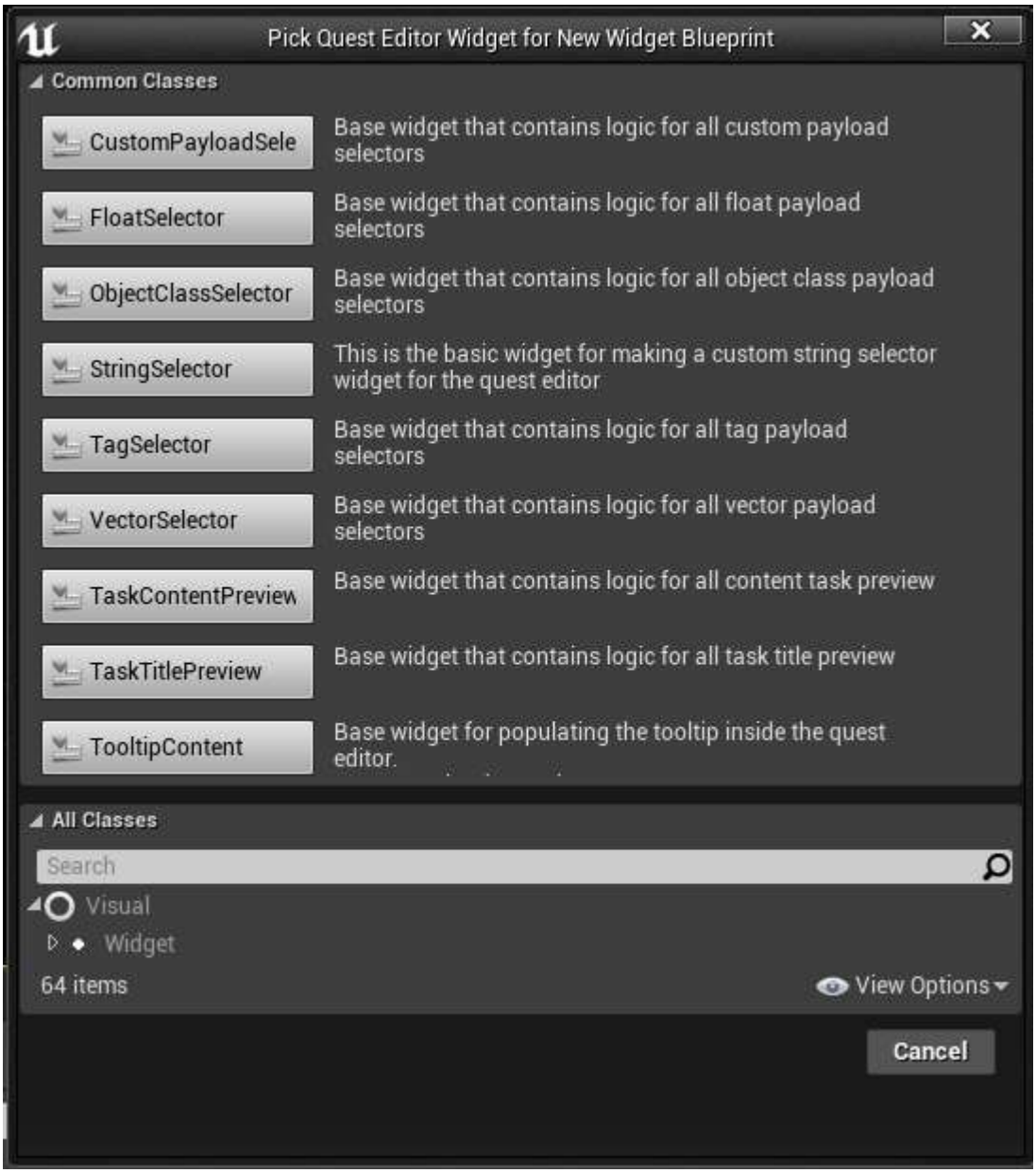
Back to top.

4. Quest Editor Customization

The Quest Editor is designed to be easy to customize to fit your needs and to speed up your workflow. This is done by making custom assets, mostly widgets, that are already predesigned.

If you right click in the editor and search for the quest category, you will find a lot of useful blueprints that you can create for this.

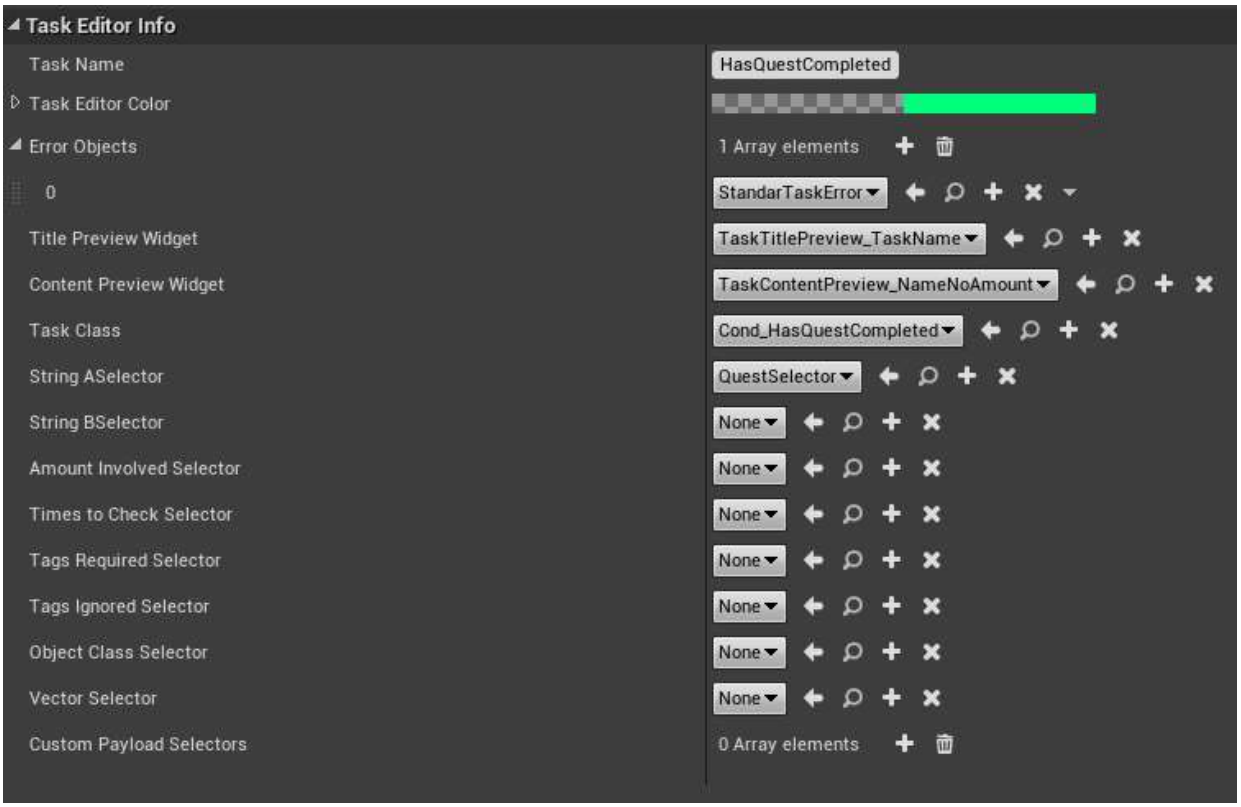
If you select the “Quest Editor Widgets” options, you will have access to all the important widgets for editor customization.



4.1 Tasks customization

The tasks will be configured using the “QuestTaskEditorInfo” datatables, where all the important variables for each task must be set.

It is important that after creating a new task, an entry for that task is added to one of these datatables, to ensure the Quest Editor correct performance.



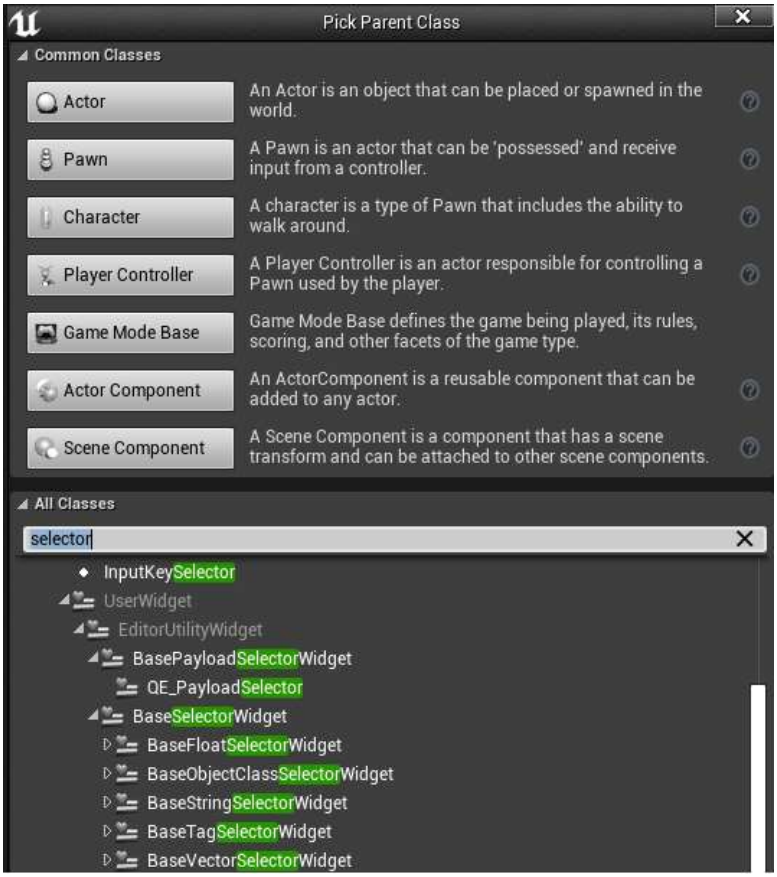
This picture shows all the variables that can be customized for a task. All the ones that are not set are not important or used for this particular task, but could be used for other task.

[Back to top.]

4.1.1 Selectors

The selectors are type of widgets that allow us to fill any variable in the payload while in the editor.

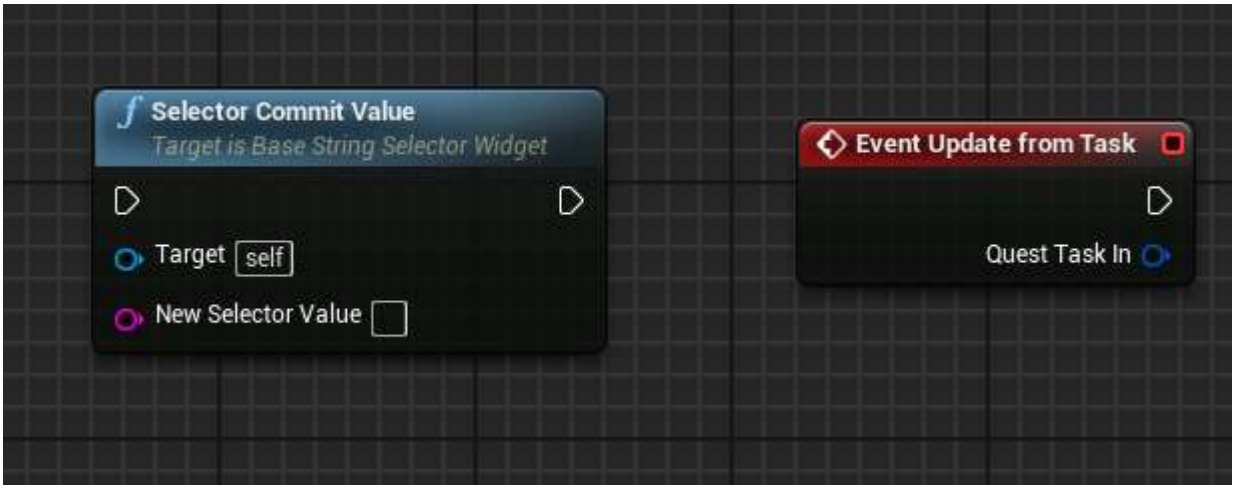
To create a custom selector, we need to know what is the final type of variable or the selector output that we want, and we will select the base selector type accordingly.



Inside the selectors, we will have to override the function “UpdateFromTask” to provide functionality for the selector updating. For those variables that repeat inside the payload, a “SelectorSlot” integer variable is provided to indicate to which variable the selector should be using on update.

The selector needs to be able to commit the variable value to the Quest Editor. This is done be calling the function “SelectorCommitValue”, that will have a different input depending on the selector type.

Some basic selectors are provided in the system. Its recommended that you check how they are implemented.



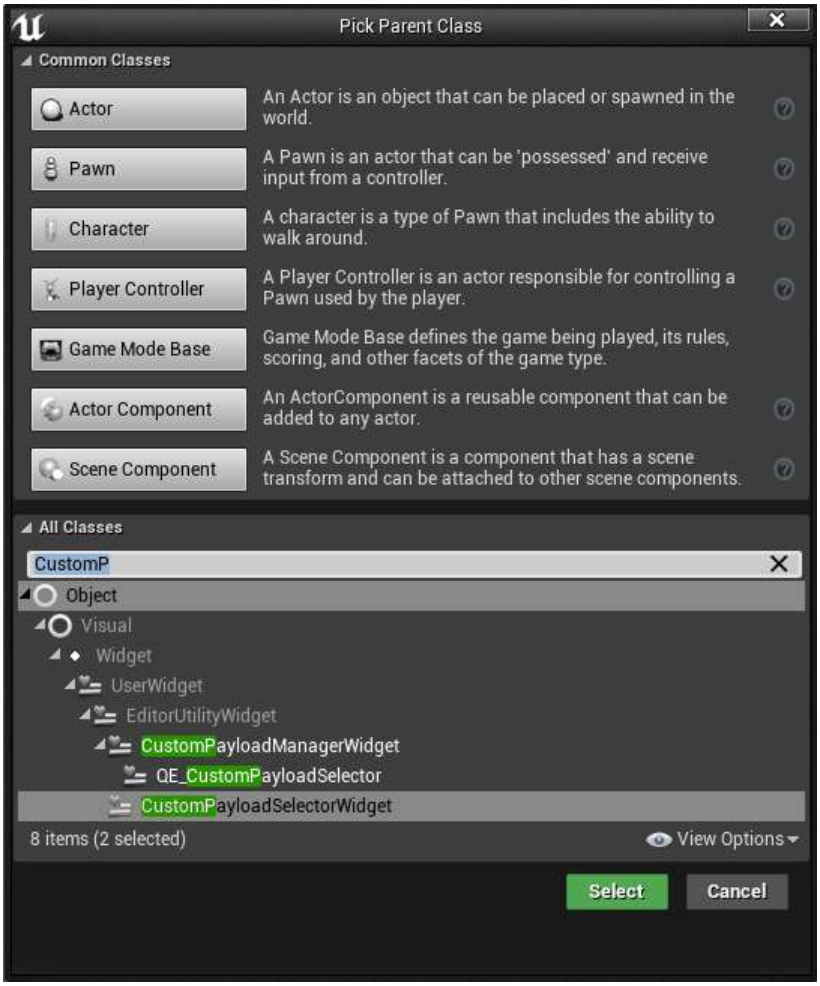
These are the two main functions or events that should be used inside a selector. In this case is a string selector. The first one is used to commit the string value. The second one should be used to update the string value from the quest task provided in the selector widget.

Back to top.

4.1.2 Custom Payload selectors

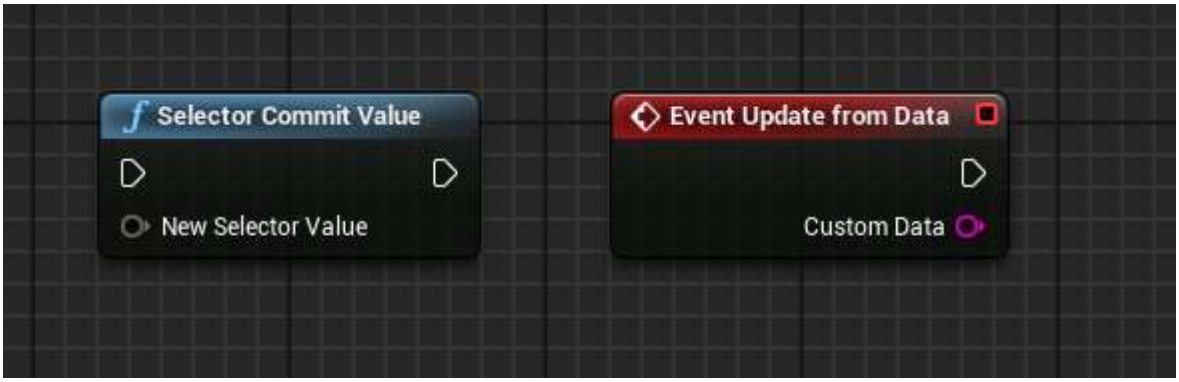
These selectors allow as to fill the custom payload data in the tasks. They can convert any type of structure into a Custom Payload Data String.

To create one, you need to select the “CustomPayloadSelectorWidget” as class for the new selector.

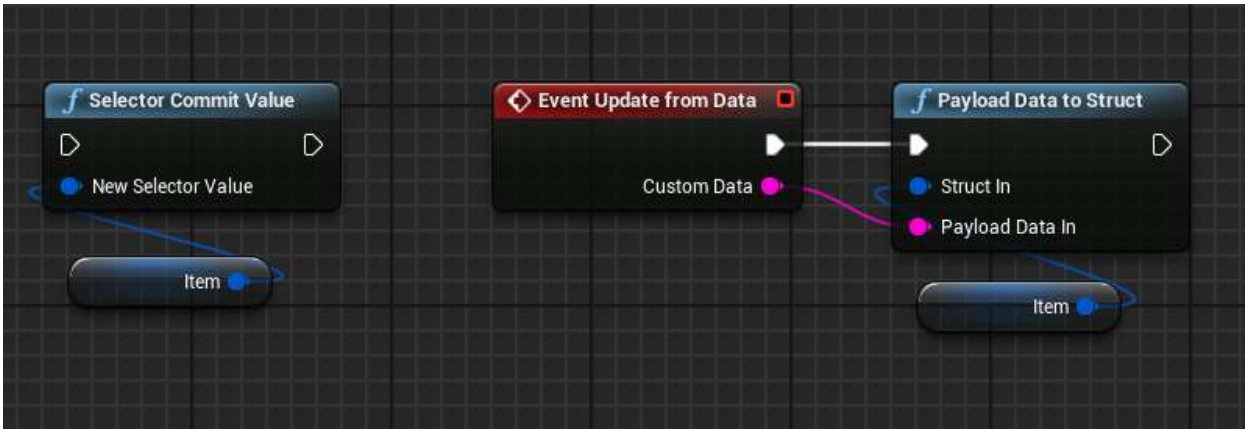


There is no limit to the number of selectors we can add to a task, and they will determine the amount of payload data strings generated later on the task.

The order that we give them in the array of the QuestTaskEditorInfo table will determine the index of the Custom Payload Data String. We will use this index later to retrieve the data inside the tasks.



There are the two main functions or events that should be used inside a custom payload selector. The commit value allows us to use as input any type of structure. The second one gives us the data to update. Since the data is presented as a string, it should be provided a structure of the input type to "recover" that data into something more useful.



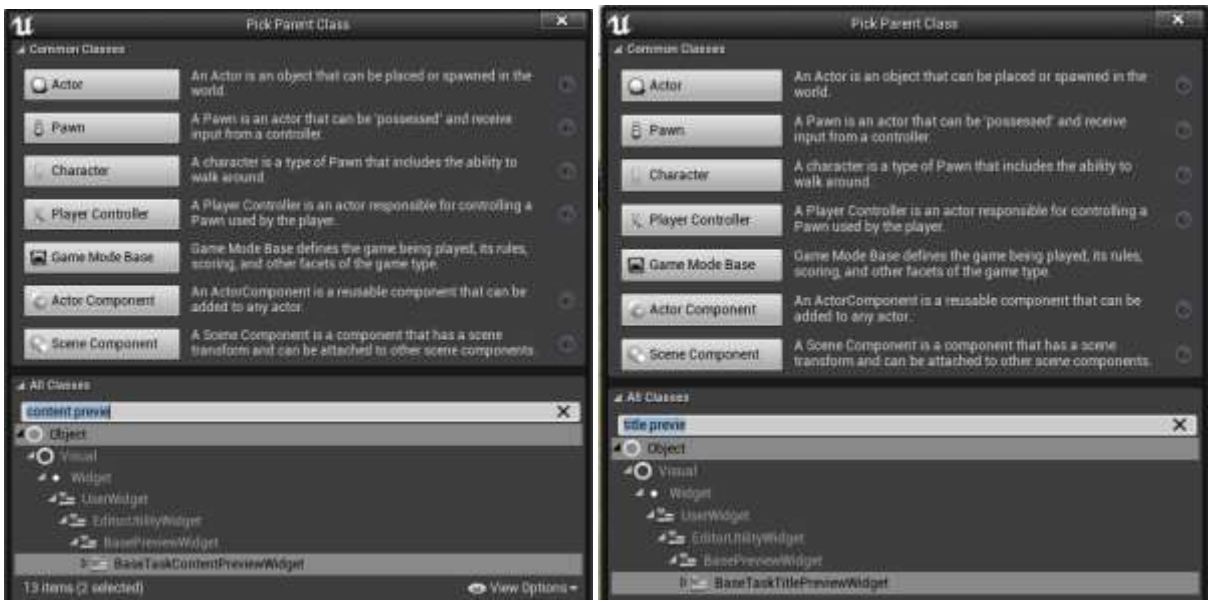
In this picture an example using a custom item structure is shown. The string data is reconverted and stored in the item variable of the custom payload selector.

4.1.3 Content Preview and Title Preview

These widgets allow to customize the way in that the task node is present inside the quest editor. They should show all the important information for the particular task.

To create these widgets, you will have to create "BaseTaskContentPreviewWidget" and

"BaseTaskTitlePreviewWidget" blueprints.



They both have a “UpdatePreviewWidget” event that should be overridden with the logic necessary to update the widget. These widgets have access to the task information and some specific functions.

Inside the node, they are aligned one on top of each other, the title is the first widget and below it the content will be created. They are very similar in term of functionality, and it could all be done with only one widget, but having two of them makes them more modular and easier to setup.

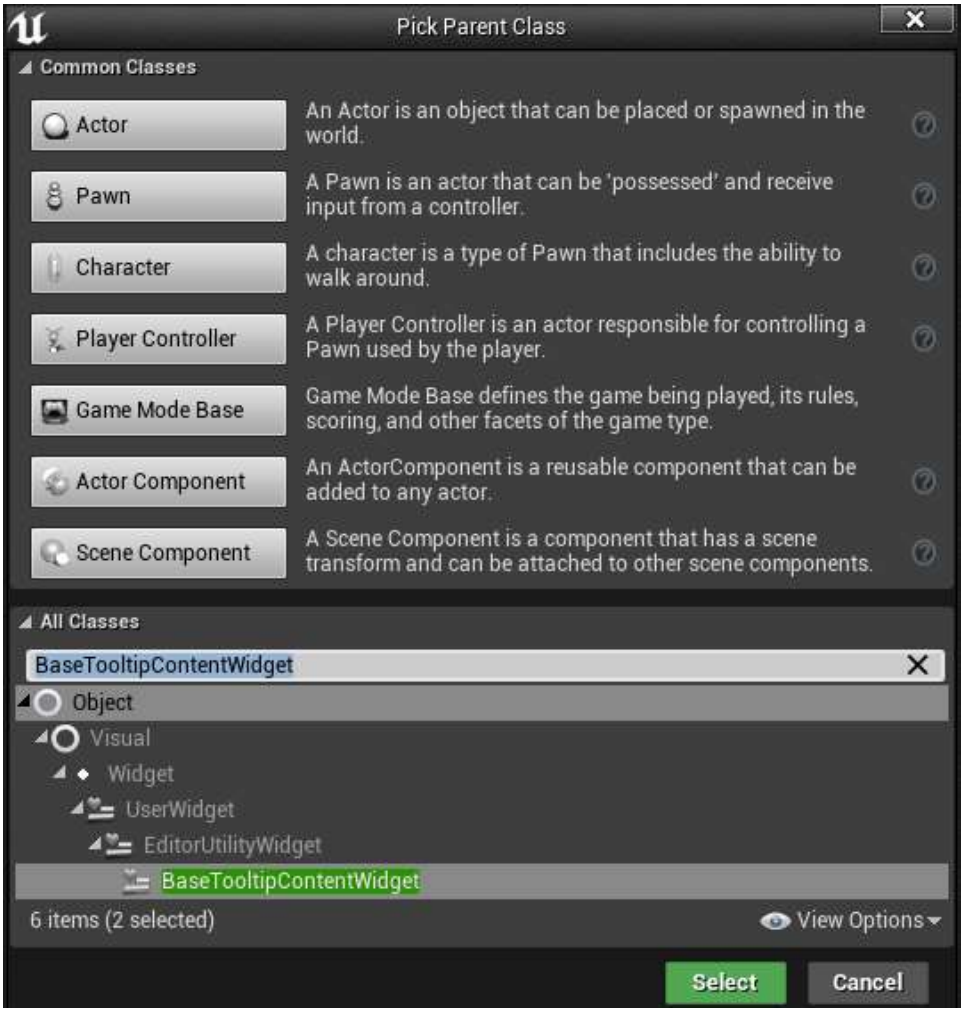
Back to top.

4.1.4 Payload Tooltip

It’s very simple to include tooltips inside the Quest Editor. For this you will need to create a “BaseTooltipContentWidget” blueprint.

You can use the tooltips you have already made for your game inside this widget or make a custom widget. I use for example the same tooltip that I have for items in game for showing the items in the editor.

Payload tooltips have the option to be shown at the top left of the screen or in the mouse position.



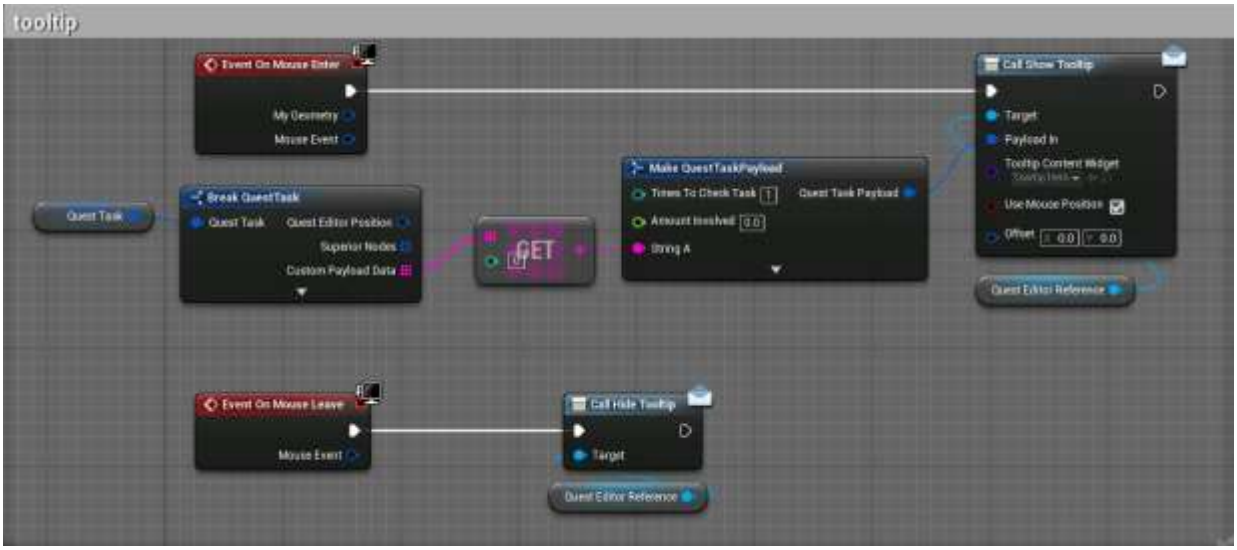
Inside the widget, you will need to override the function “UpdateTooltipWidget” and use it to update the tooltip. The information will be passed to the widget via a payload struct.

To control the tooltip, you will need a reference to the Quest Editor. Some widgets already have a reference to it. If these is not the case, you can use the function “FindQuestEditorReference” to get one.

With the Quest Editor reference, you can call the “ShowTooltip” and “HideTooltip” event dispatchers to control the visibility of the tooltip.

On showing the tooltip, a payload will need to be provided and also the class of the custom Payload Tooltip widget that we want to create.

The “UseMousePosition” and “Offset” variables allow to control the tooltip position in viewport.



This is an example of a custom tooltip visibility control for a custom item structure inside a content preview widget.

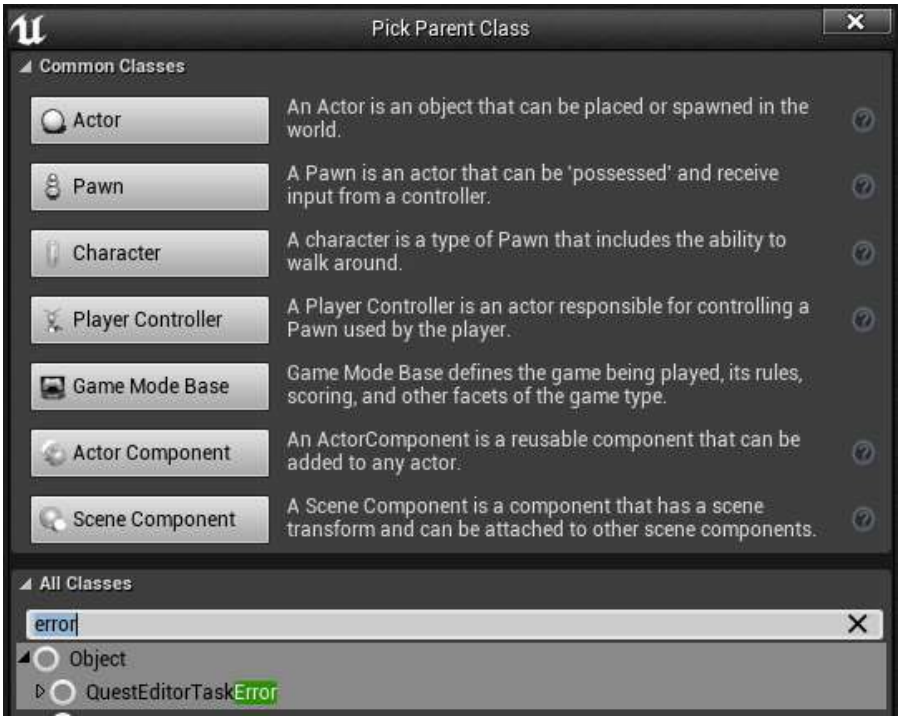
Back to top.

4.1.5 Task Errors

It is possible to customize the errors for each task that are shown inside the editor. This allows for faster debugging of the tasks.

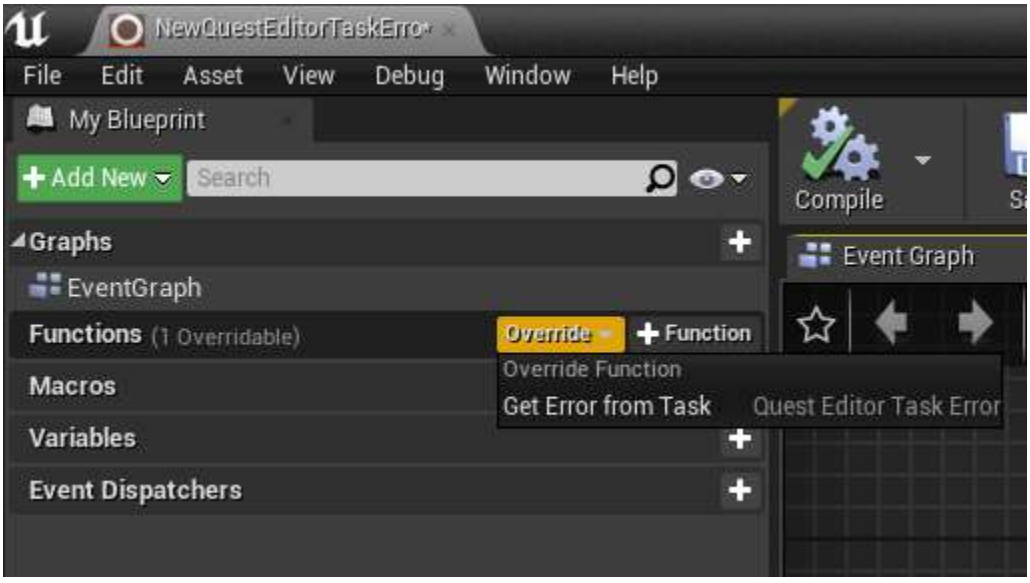
This logic will be handled with a “QuestEditorTaskError” object.

These objects are designed to be very modular and reusable, and therefore, tasks can have any number of these objects assigned to them. They will all execute simultaneously to provide all the errors to the editor.



There is a function that must be overridden in order to calculate the errors using these objects, as shown in the following picture.

In this function you can determine the errors and output an error message with a specific color.



The system has a general error object that can determine which variables are not set in the payload.

The function “GetErrorFromTask” should return a null string if there is no error.

[Back to top.]

4.1 Talk Tasks customization

In a similar way to the tasks, the talk tasks can also be customized to improve the workflow. There are several things that you can change, and you will by doing this via the talk modes.

Back to top.

4.1.1 Talk Modes

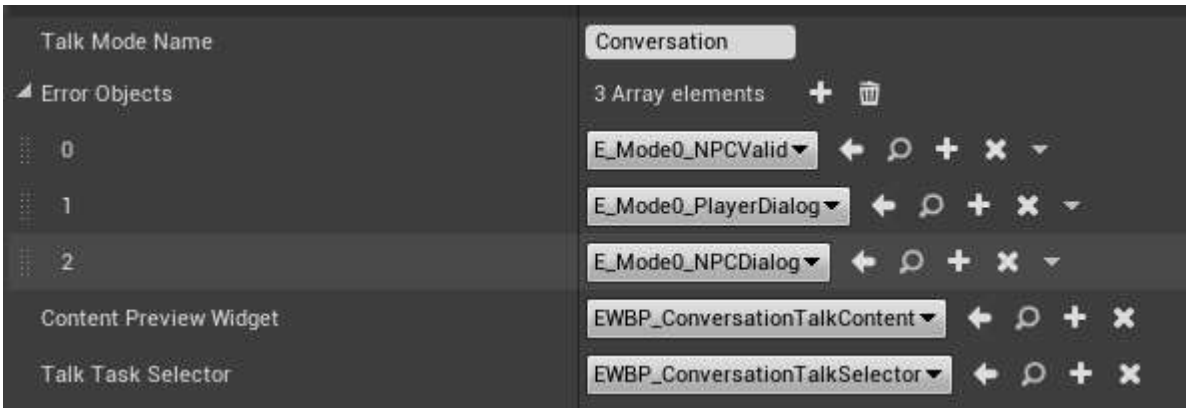
Talk modes allow to customize how the nodes look on the editor, the selectors that are used and even the errors that are displayed. These modes also allow to pass additional data that can also be customized.

They will work for all talk nodes, no matter if they are from quests or dialogs. Each quest or dialog can be assigned a talk mode, so it is even possible to have several modes at the same time. This is not usually recommended as it can complicate the in-game UI, but is possible.

In general, it is better to use one of the existing modes to handle all the talk tasks in your game, or to create a custom talk mode and stick to it.

The talk modes can or not be compatible between them, that depends entirely on their design.

To define talk modes, it is necessary to create a datatable of the type “QuestTalkTaskEditorInfo”. This datatable must be inside the designed datatables area for your project.



This is how a talk mode is defined.

You can change the default mode of the editor in the project settings, in the “QuestEditor” category.

There is also the option to add a custom payload selector to the mode for passing extra information.

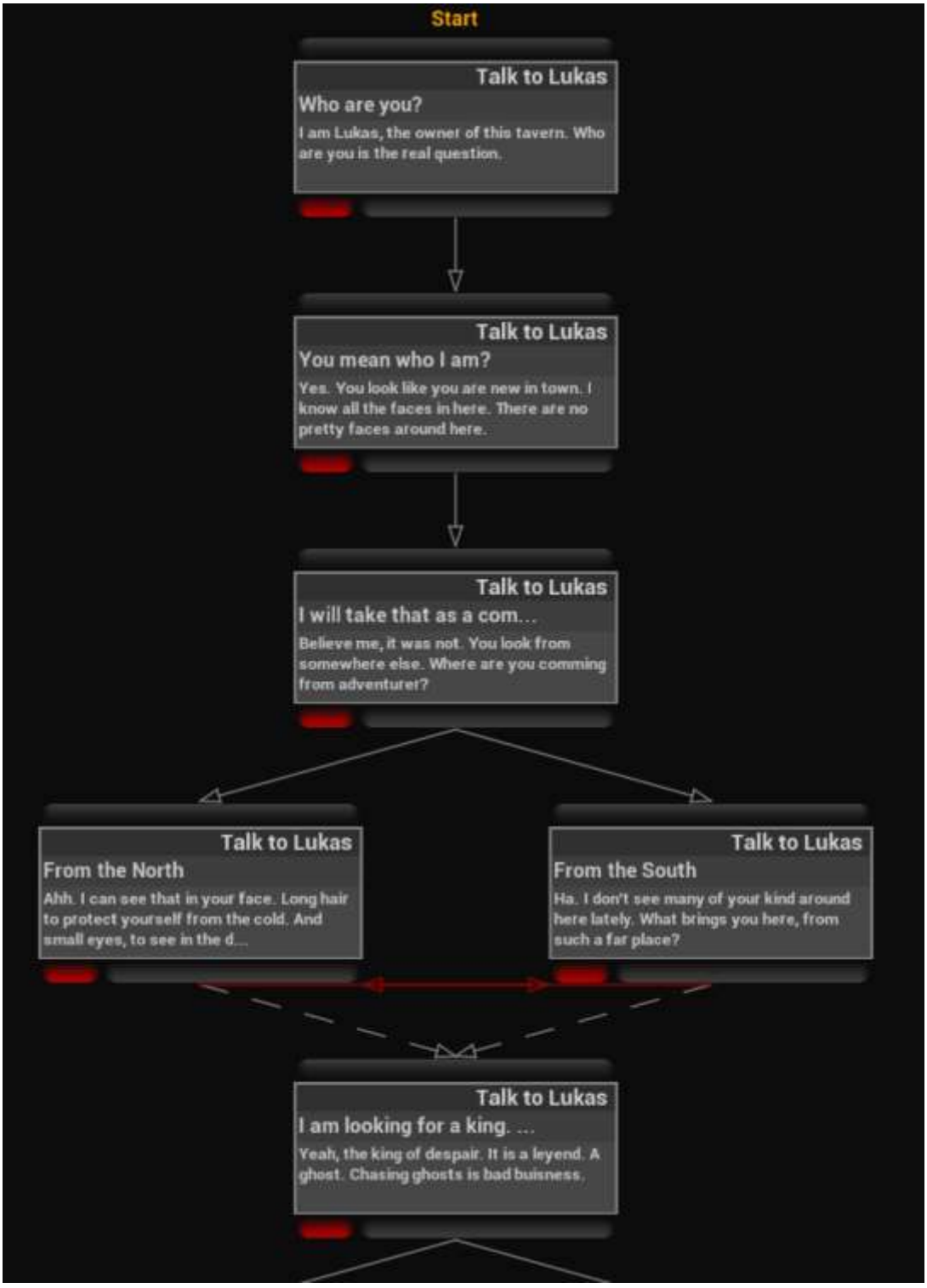
Back to top.

4.2.1.1 Conversation mode

This mode is the default mode of the editor. All the nodes are based on a set of Player Dialog and NPC response to that dialog. This mode allows any connection and is more suited for using in parallel. This

means that if the player has multiple conversations with a single NPC, they will be displayed all at the same time, at least if the base provided UI is being used.

The dialog is always used as an opener in this mode and is useful for when the NPC has no quests, for the player to still have some interactions.



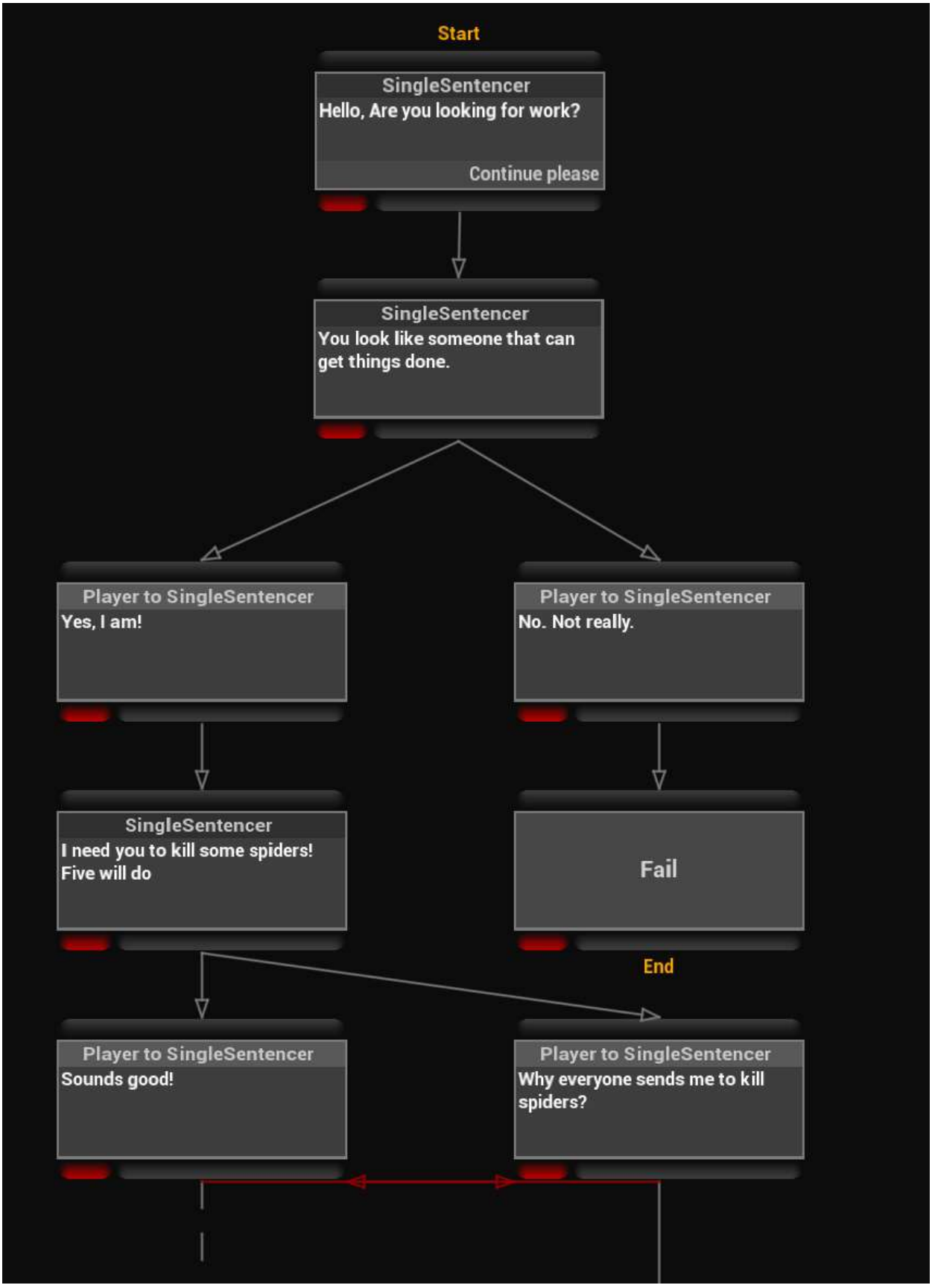
This is an example of a Conversation mode dialog.

Back to top.

4.2.1.1 Single sentence mode

This mode is based on only using one of the strings of the talk task as data. This means that the node can represent an NPC or a Player talk line, but not both (usually). This mode is better suited for a “serial” way of handling quests.

All the quests have priority over the dialogs in the provided UI, and they will be given to the player one after another and not all at the same time.


Start
SingleSentencer
Hello, Are you looking for work?
Continue please
SingleSentencer
You look like someone that can get things done.
Player to SingleSentencer
Yes, I am!
Player to SingleSentencer
No. Not really.
SingleSentencer
I need you to kill some spiders! Five will do
Fail
End
Player to SingleSentencer
Sounds good!
Player to SingleSentencer
Why everyone sends me to kill spiders?

This is an example of a Single Sentence mode dialog.

Back to top.

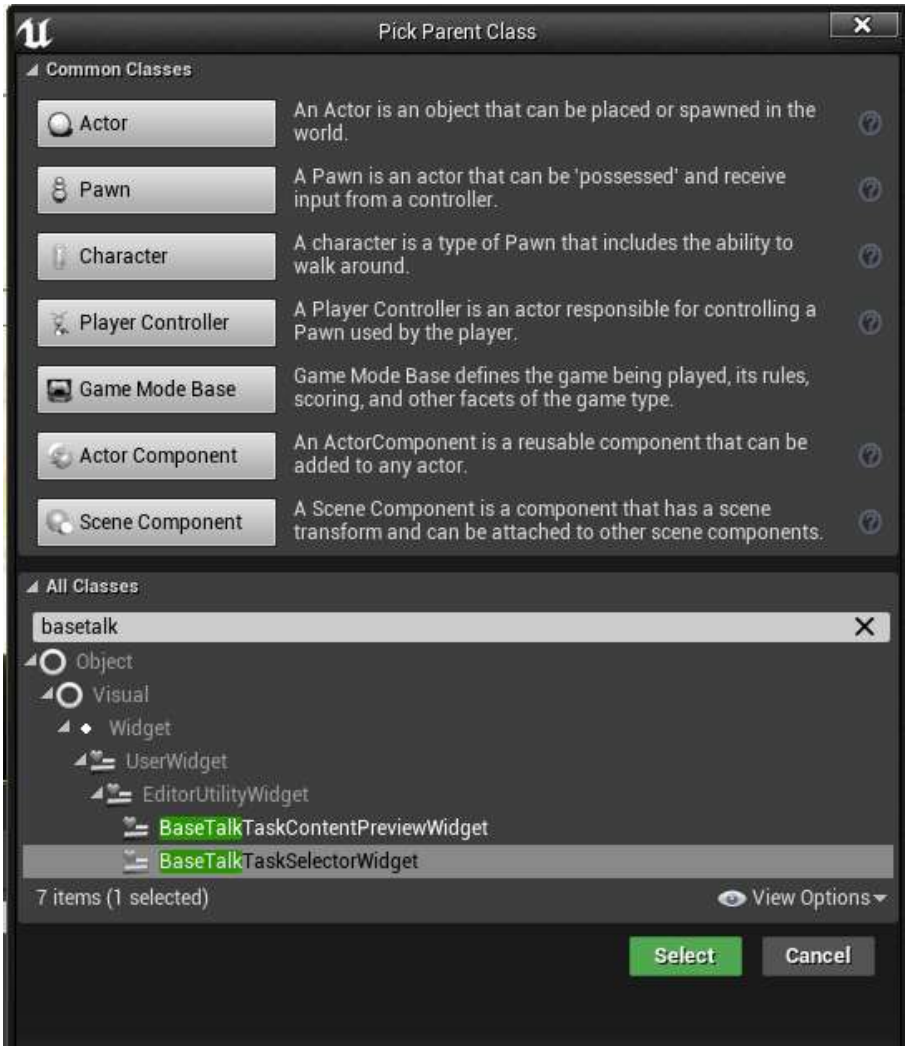
4.2.1 Talk Modes Components.

There are several components that can be customized in order to create a new mode that are specific to the talk modes. In talk modes you can also use tooltips and custom payload selectors to help customize even further.

Back to top.

4.2.1.1 Talk Task Selectors

This is the widget that will allow to fill the talk nodes with information. This can be customized to improve your workflow. A simple case for example, is the creation of presets that are selectable via enumerators.

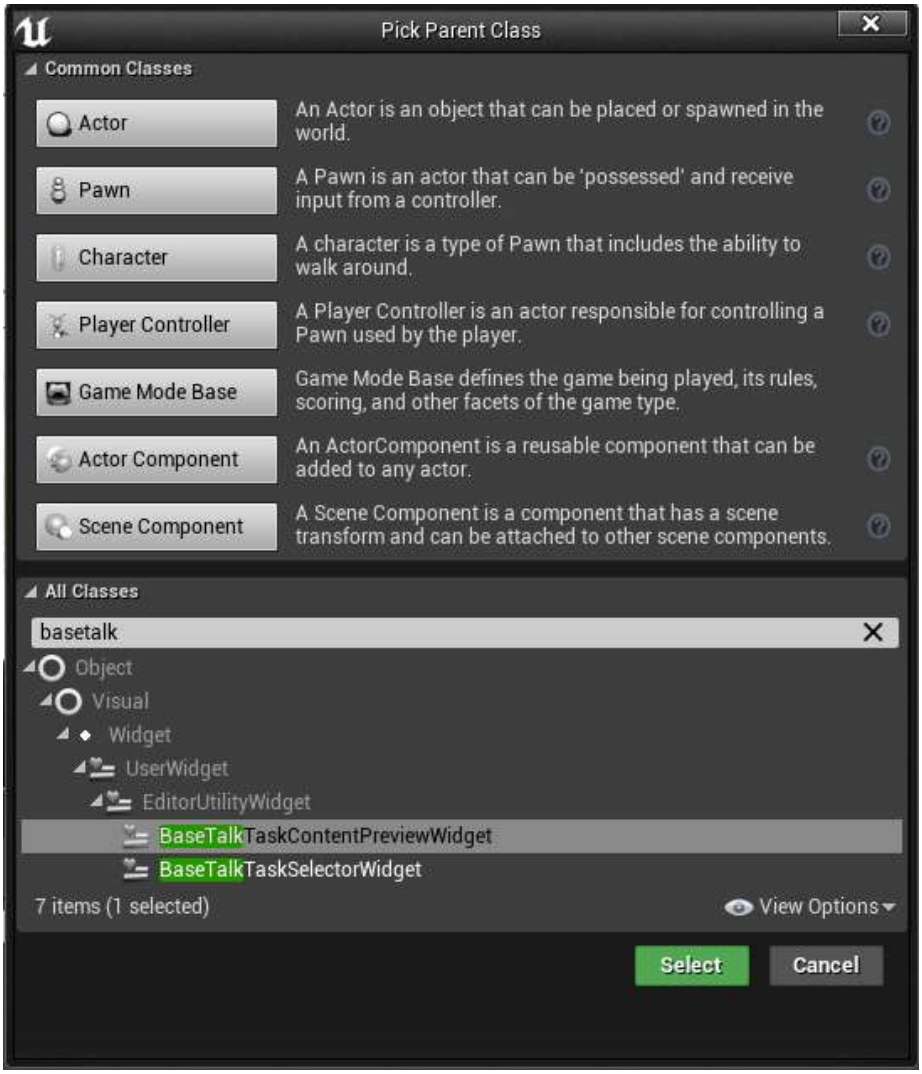


This is the widget you have to create for this. “BaseTalkTaskSelectorWidget”.

Back to top.

4.2.1.2 Talk Task Content Preview

This widget allows to customize how the nodes look in the editor. A simple use of this is the amount of text you want to show or how compact the nodes are. You could also show additional information depending on your needs. Or perhaps you want to make your talk nodes look cooler.

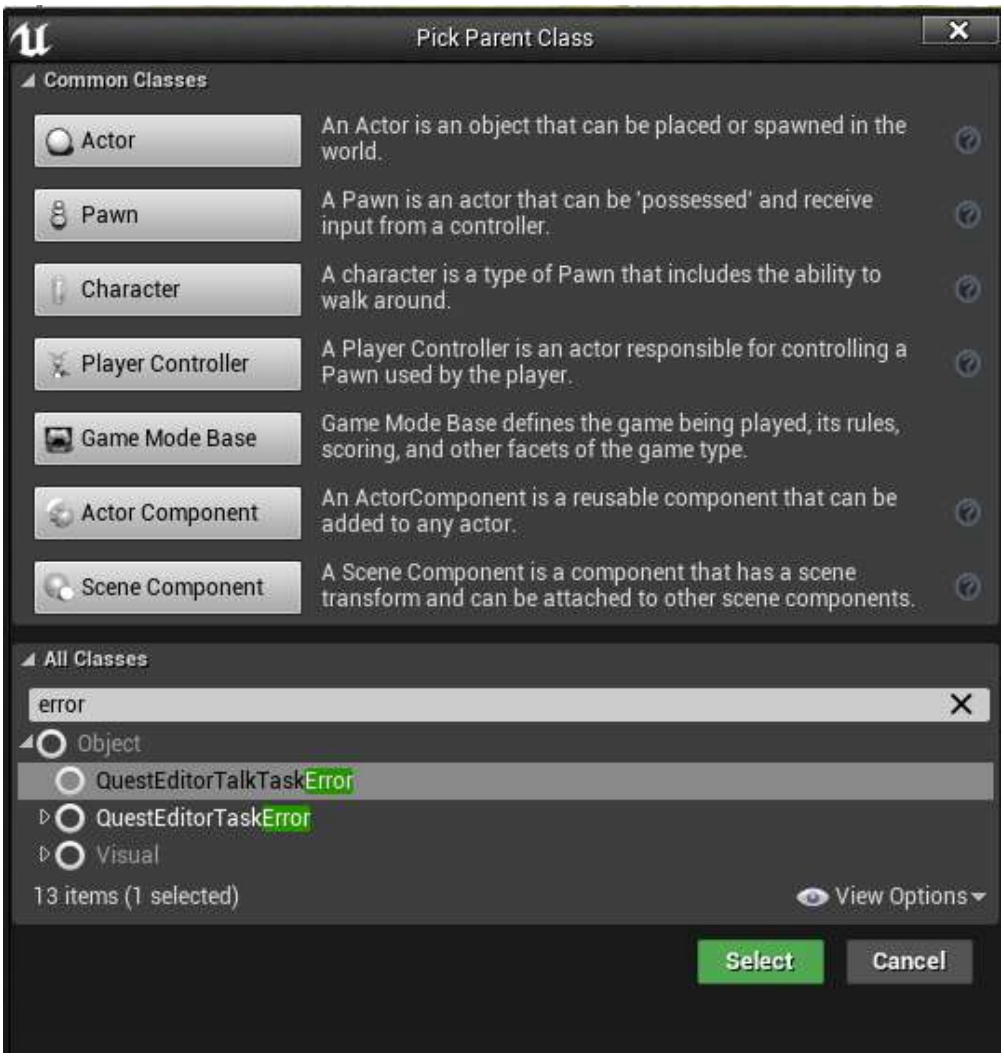


This is the widget you have to create for this. “BaseTalkTaskContentPreviewWidget”.

Back to top.

4.2.1.3 Talk Task Errors

This are objects that allow you to customize the errors you display for your particular mode. For example, you could show a limit of characters as an error, to avoid having too long dialogs. They work in a very similar way to the task errors and are fully modular.



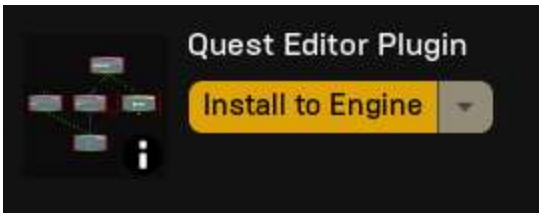
This is the object you have to create for this. “QuestEditorTalkTaskError”.

[Back to top.]

5. System Setup

5.1 Installing the plugin

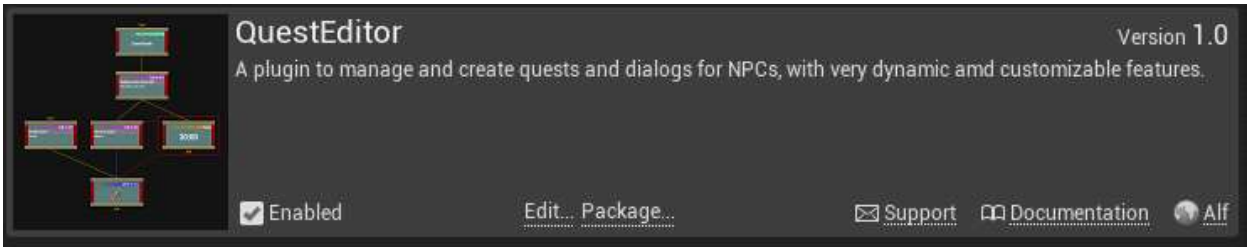
You will need to install the plugin into the engine using the Epic’s launcher. Go to your library and install the plugin.



Back to top.

5.2 Enabling the plugin

In order to start using the plugin, you will need to enable it in the Plugins section on the Edit tab of the engine. You will find it under the Quests Category. You will have links to the Discord, YouTube and to the last Documentation version there too. This will ask you to restart your project.



Back to top.

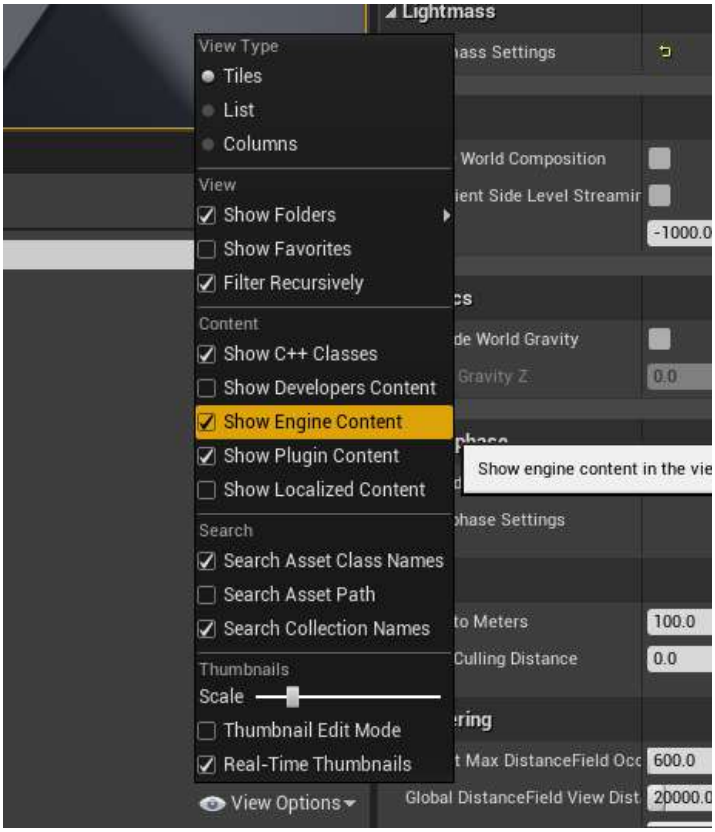
5.3 Quick Setup

In order to quickly review the system, you can use this form of setup. It will allow you to see the example blueprints working and some examples quests and dialogs.

First of all, we will need a project. I suggest you use the Top Down Template and create a new project.

You will need to enable the “Editor Scripting Utilities” and the “QuestEditor” plugins, as shown in the 5.1 and 5.2 sections of this documentation.

To start the setup, you will need to find the plugins content folder. For this you might have to “Show Engine Content” or “Show Plugin Content” in order to see it, as shown in the following picture.

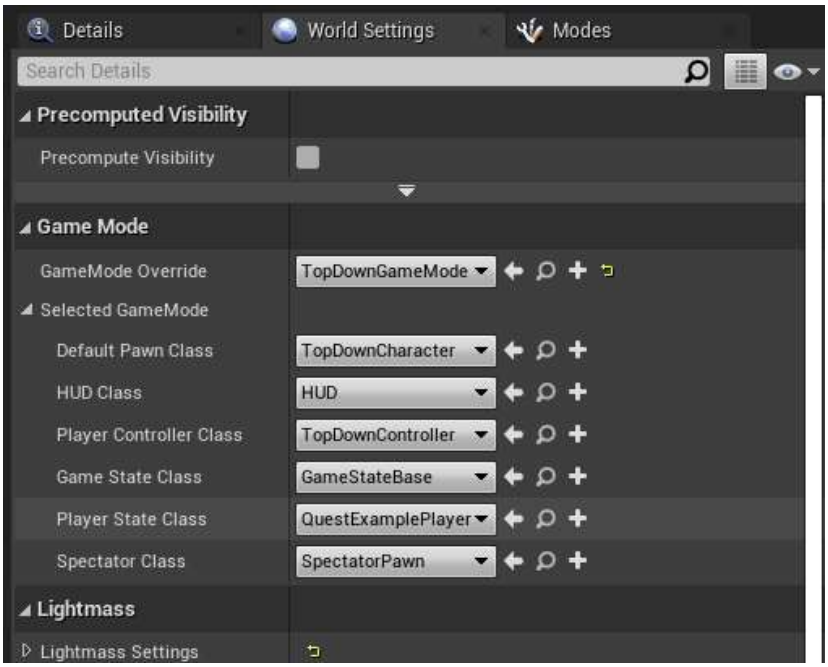


Once you found the Plugin content folder, in the content browser of the editor, go to "/QuestEditorContent/ Examples/Example_Blueprints/NPCs".

In this folder you will find several NPCs that are ready to use. Drag them and add them to the map example.

One of those NPCs is the "ExampleNPC". This NPC contains all the base logic and you don`t need to add this one to the map, because is not a "functional" one.

Finally, you will need to change the player state of the current map. You will go to the World Settings, and override the player state for the TopDownGameMode (or the current Game Mode, depending on which template you used for the project) with the "QuestExamplePlayerState".



Now you can hit play and test the NPCs, with their quests and dialogs. You will have to get close to them in order to interact, since this function is implemented on overlap actor events for this particular example.

Back to top.

5.4 Quest Editor Setup

There are a few and very simple steps that you need to follow in order to setup the Quest Editor. This will allow you to use the Quest Editor and access it in the “Window” tab. With the Quest Editor you will be able to create Quests, Dialogs and NPCs, edit and manage them.

Back to top.

5.4.1 Editor loading

To ensure the Quest Editor loads properly in engine startup, it is needed to modify the config file “DefaultEditorPerProjectUserSettings.ini” of your project. This file is located in the “Config” folder of your project. If there is no config file with that name, you will have to create one.

In said config file, you should add the following lines:

```
[/Script/Blutility.EditorUtilitySubsystem]

+LoadedUIs="/QuestEditor/QuestEditor/QuestEditor.QuestEditor"
```

This is done automatically by the plugin now. If you cannot see the Quest Editor widget loaded when starting the Unreal Engine editor, just close and restart your project and this should be updated.

Back to top.

5.4.2 Datatables Setup

To start working with the system, you need to define at least three datatables, one for each of the main structures.

“Quest” struct datatable: In this table we will save all the quests and dialogs data. It is probably better to have two different datatables at least, so you can be more organized and save dialogs separated from quests.

“NPCQuestDef” struct datatable: In this table we will save all the NPC information, the quest and dialogs that the NPC will have in their inventories. We will use this datatable to load the NPC data in the NPC manager.

“QuestTaskEditorInfo” struct datatable: This is the table that will hold all the information for the tasks inside the editor. All the configuration for each task, the type of payload and how to set it via selectors, the custom preview slots that will be used and the errors are some examples of the variables that we will be able configure here.

There is no limit for the amount of datatables of each type that we can have, this allows for a better organization of the information.

To improve the loading times of the Quest Editor, its recommended that there are no additional assets inside the folders that contains this datatables.

Back to top.

5.4.3 Quest Editor Settings

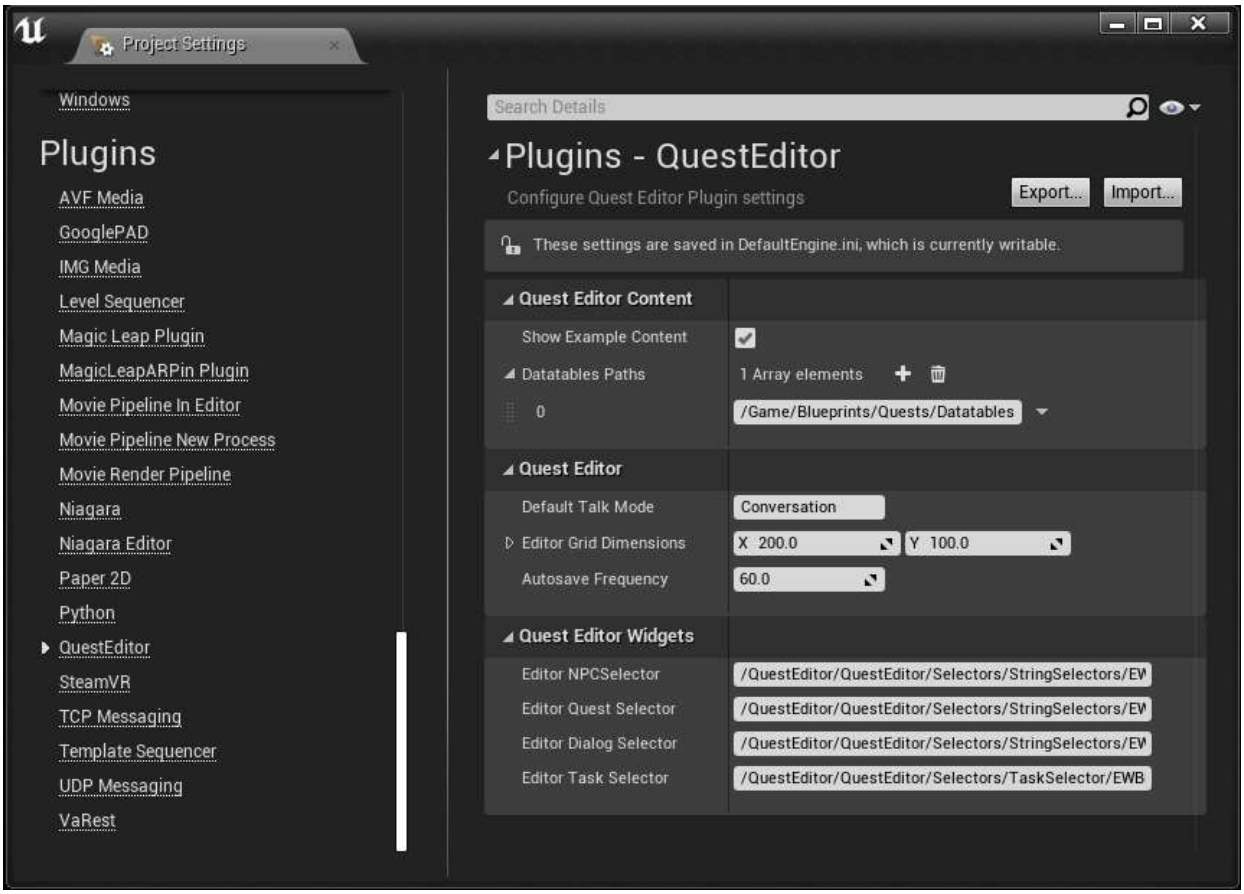
In Edit, Project Settings, in the Plugins category you will find the settings for the Quest Editor. In here you can configure some preferences.

It is recommended that the Datatables Paths are all added here, to improve the loading time of the Quest Editor.

The default talk mode can be useful to override to make it easier for you to start quests and dialogs in the mode that you selected for your workflow.

You can disable also the example content that is provided, so you can see clearer all the content you create.

Some of this settings will require to restart the Quest Editor in order to be applied.



Back to top.

5.5 Quest System Setup

In order to implement the system inside your game, you will need to setup a few different things, being the NPCs and the Quest Manager the most important ones.

Back to top.

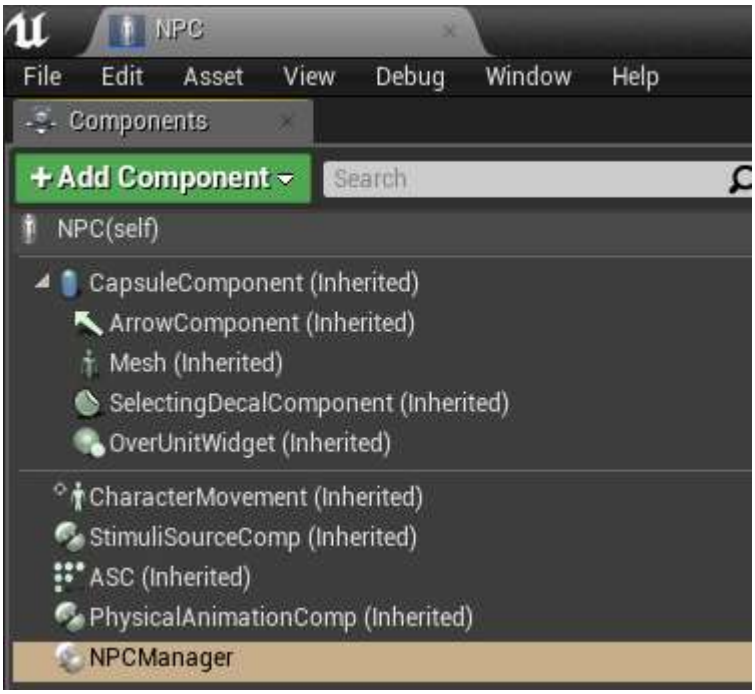
5.5.1 NPC setup

For the NPCs to be able to work properly, they will need to have a NPCManager, which is an actor component. The NPC itself is not provided by the system (since is not relevant) but rather is something that you will probably have set up already in your game.

For this reason, the NPC itself can be any actor.

The NPCManager handles loading dialogs and quests and their interaction with the player.

The NPCManager requires of an initialization using a “NPCQuestDef” structure (we made a datatable for this) and there are several ways of doing so. Here are some of them briefly explained.



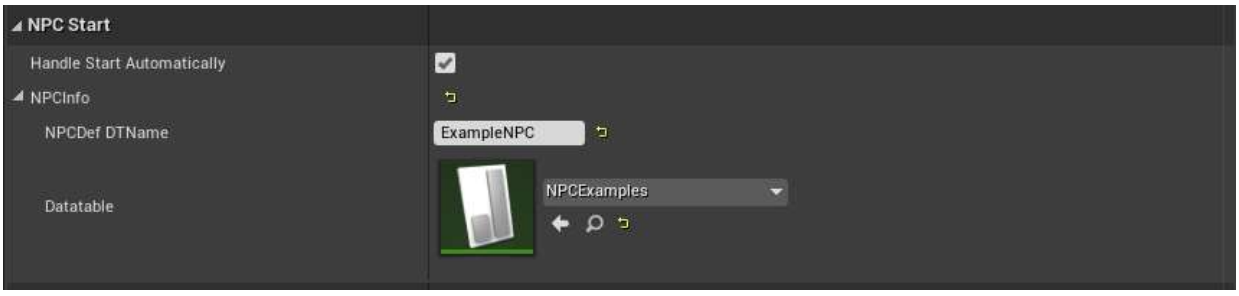
Back to top.

5.5.1.1 NPC Direct initialization

This is the simplest way of initializing the NPC. Consist of directly setting the variable that is exposed in the NPCManager called “NPCInfo”.

This struct contains the name of the datatable row for the NPCQuestDef and the datatable path in which this struct will be found.

The NPCManager will handle automatically the start of the component using that information. It is needed also that the variable “Handle Start Automatically” is set to true or otherwise the component will wait for the manual call of the initialization. This way is useful if you have organized your NPCs as different assets.

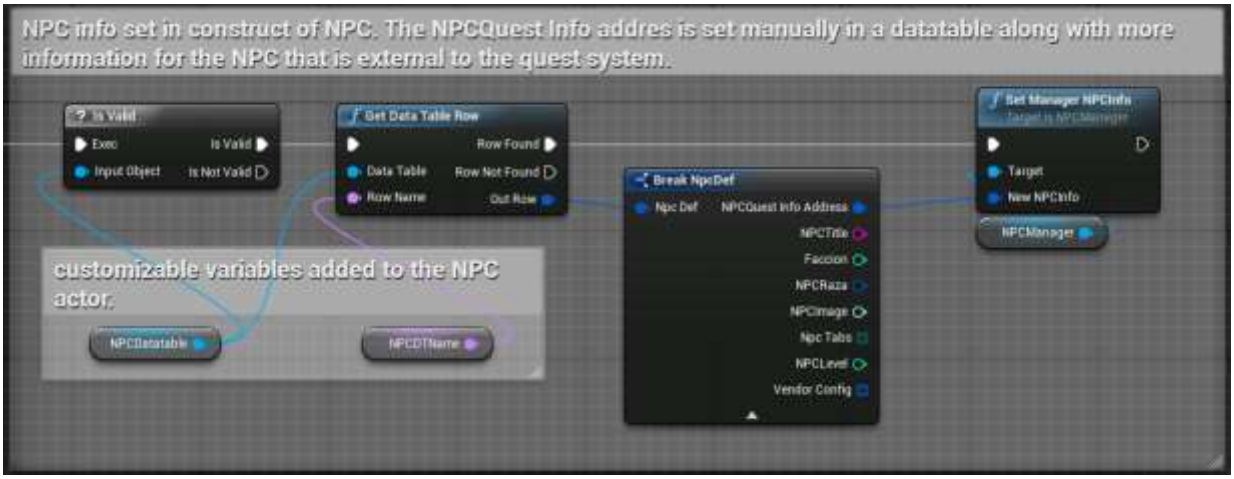


Back to top.

5.5.1.2 NPC Dynamic initialization

Instead of setting the NPCInfo directly, we can use the function “SetManagerNPCInfo”. This allows us to do it in the event graph and using logic or getting the NPC information from something like a datatable or a function. For this to have effect, the “Handle Start Automatically” must be set to true, just like in the previous method, the only difference is that we are now setting the NPCInfo through code.

Because the NPCManager will handle the initialization on begin play of the component, we need to make sure that the NPCInfo variable is set prior to this, for example, in the event construct of the NPC.



In this example, the customizable variables are set for the NPC actors once they are placed on the map and the NPCInfo is obtained from these datatables, that are previously set with the correct information.

This method is recommended for when we are not using different actors for each NPC and we are just using one base one and setting it with code.

Back to top.

5.5.1.3 NPC Manual initialization

In this case we will be setting the variable “Handle Start Automatically” to false, which means that the NPCManager start will not be executed on begin play or automatically by the NPCManager. Instead, the manager will wait for the manual initialization via the function “ROS_StartNPCManager”. This is similar to the previous method, but it allows this function to be called in any place, without worrying about the begin play timing in the execution. It’s important to keep in mind that the NPC will not be able to interact with the players successfully until is initialized and its recommended that this function is called as fast as possible in the execution to avoid any problems. This method is useful when the NPC initialization of your own game logic (not the Quest System one) requires a bit of time or a very specific way of handling and you need more control over the process.

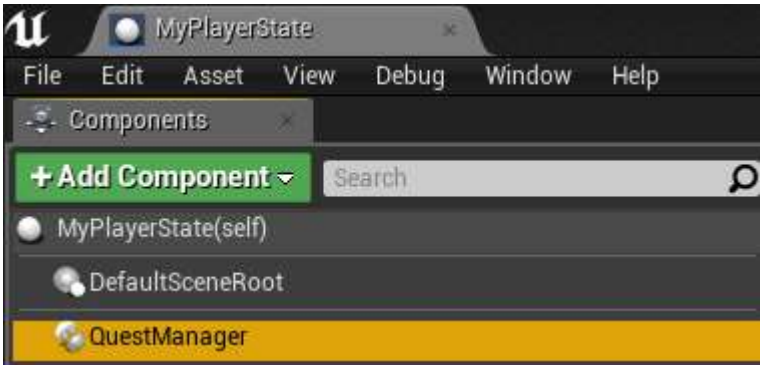
Back to top.

5.5.2 Quest Manager Setup

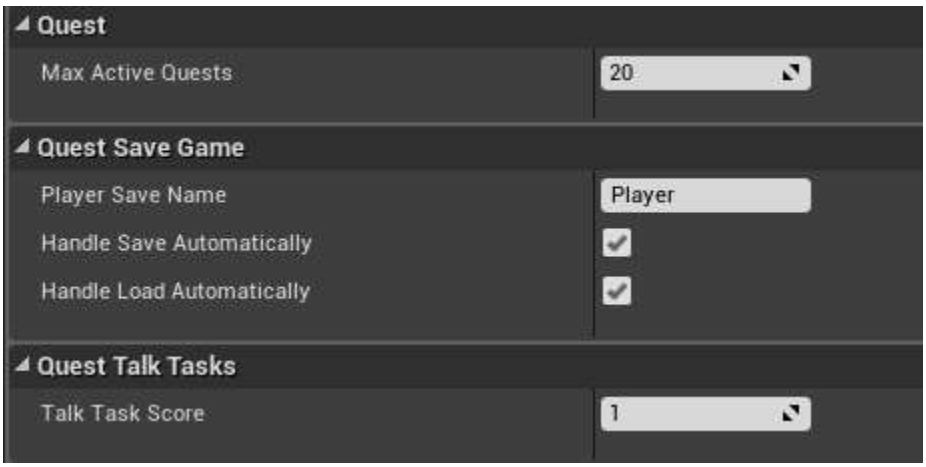
The QuestManager is an actor component that will handle all the quest and dialogs logic for the player as well as holding all the important variables for it, such as the active quests or completed quests.

This component must be placed in the player state of your game.

This component should be set as replicated and with autoactivate bool to true, these are the default settings for this component and should stay this way to ensure its properly functioning.



In the QuestManager there are some variables that can be configured such as the maximum number of active quests allowed for the player.



It also provides plenty of event dispatchers for all the different events that may occur, perfectly suited for extra logic and UI updating.

Back to top.

5.5.2.1 Player loading and saving

The QuestManager has built in logic to handle saving and loading the player progress for the quest system.

Player Save Name: It's the name that will be used for the save file that contains all the progress for the player that owns the QuestManager.

If multiple save games must be implemented, it can be set at runtime, in a similar manner to the one presented in the NPCManager for initialization.

Handle Save Automatically: Whether or not to save the progress to a file when the game is closed.

Handle Load Automatically: Whether or not to load the progress from the PlayerSaveName file when the QuestManager begins play.

You can also handle the save game on your own, if you already have a save game system. For this you will need to set to false the Booleans provided accordingly to your needs. There are functions to load and save the game manually and also functions that provides the variables that needs to be saved.

Back to top.

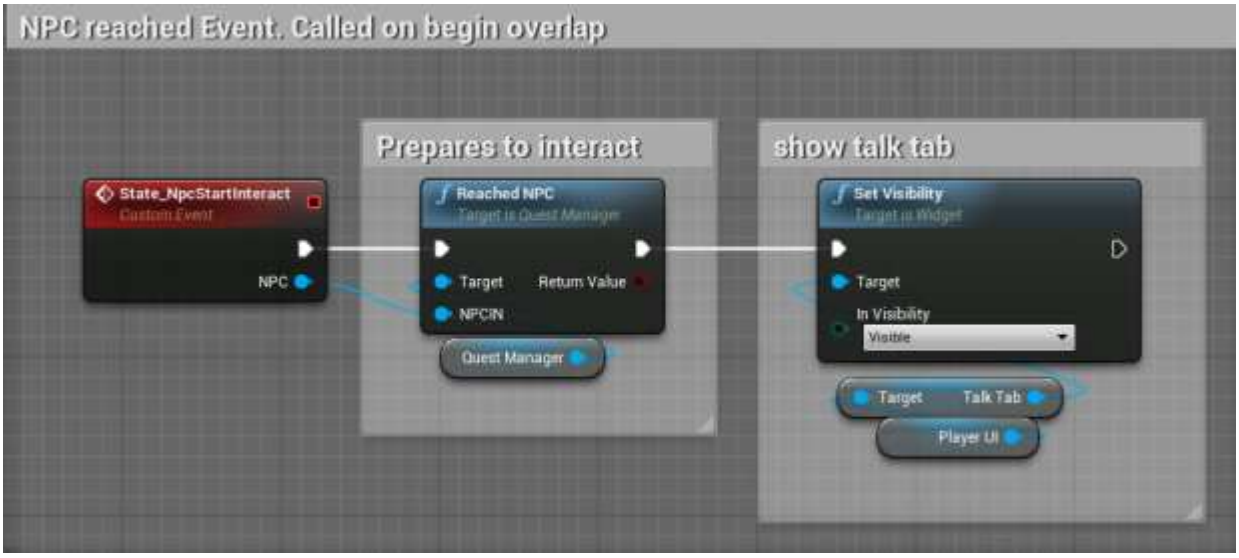
5.5.3 NPC talk and interaction

We need to tell the QuestManager which NPC we want to interact with using the ReachedNPC function.

We can do this on a mouse event, on overlap, just anything that suits your game. In this example I am using an overlap event.

Once we have called ReachedNPC we can safely show the UI that displays the NPCs Quest and Dialogs.

For convenience, all this logic is in my player state, which also holds the QuestManager. On reaching an NPC, my logic looks like this:



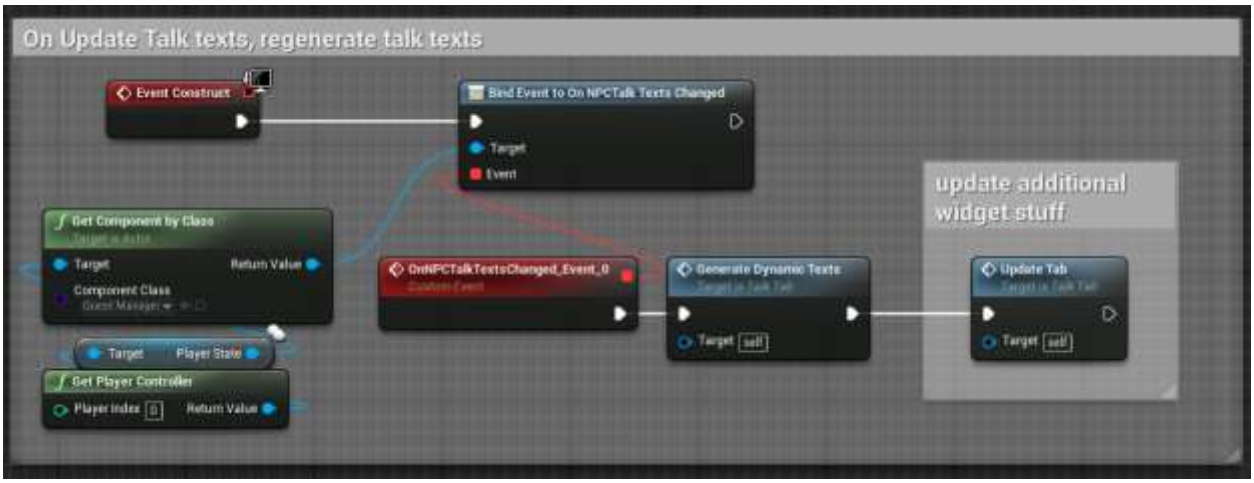
You can also select if you want to commit automatically the first dialog for the NPC or not. This helps with certain UIs design.

Back to top.

5.5.3.1 Talk tab and UI

For the player to be able to see the interaction with the NPC, you will need a UI that handles all the talk options that are currently available for him. I will call this UI talk tab. A very basic version of this UI is already provided in the plugin, to show the code and how the event is handled.

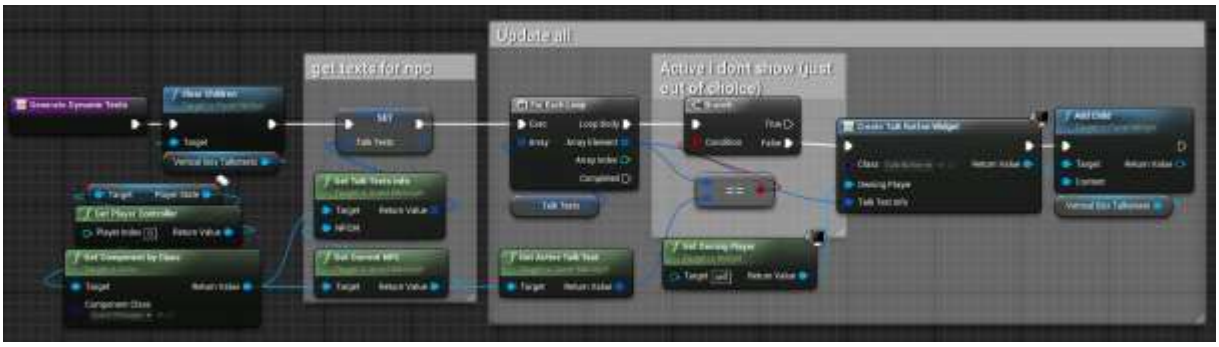
For generating the talking options in the widget, we need to use a dispatcher from the quest manager and create a custom widget that can hold the information for each of the talk options. I call these widgets Talk Buttons and simple version of them is also provided. This operation is handled in the talk tab and looks like follows in the image.



Using the “OnNPCTalkTextsChanged” event dispatcher is the simplest way of updating the talk UI. This event is called when the talk options for the player changes.

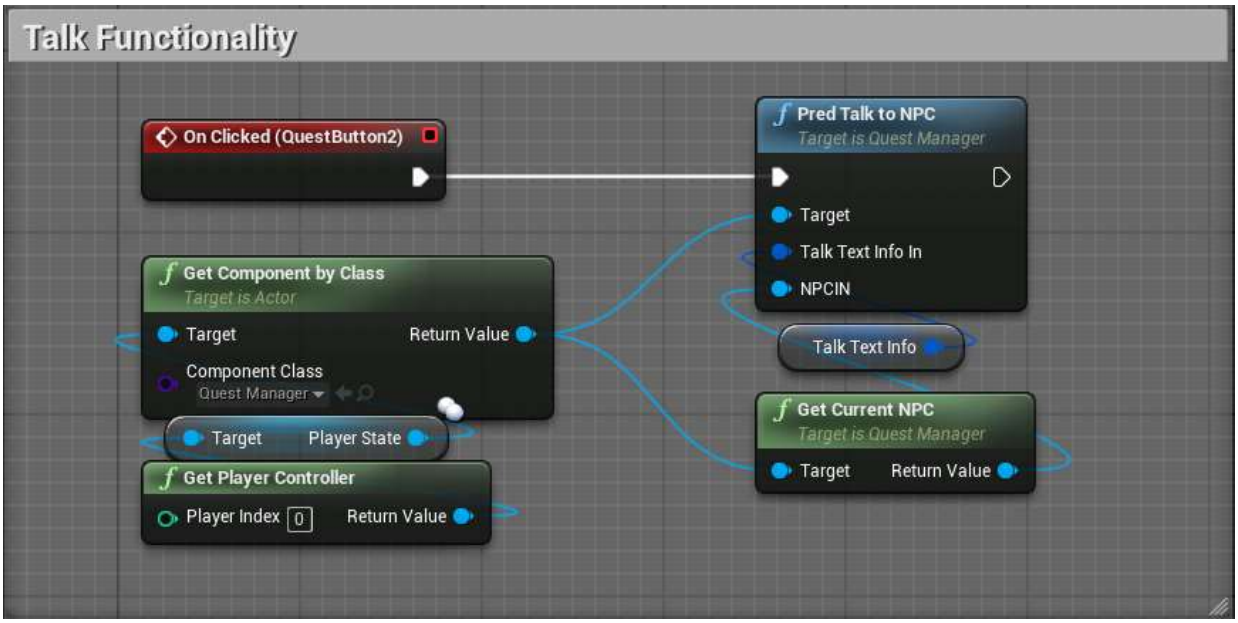
There are other ways of updating the talk UI, using different events of the quest manager but that will not be covered in this documentation for now.

The “GenerateDynamicTexts” function showed in the picture above contains the logic to obtain the current talk options, also called talk texts, and creating all the Talk Buttons, one per each option.



Each Talk Button will contain information about one single talk text, and will use that data to update itself. I will not go in details into this since this is not a UI tutorial, but you can check in the example widgets how this is done.

The importance of the Talk Buttons is that they allow the player to “talk” to the NPC when pressed. This logic is very simple and looks as follows.



This function will handle dialogs and quest talk tasks and will use an NPC reference that was set previously on calling the ReachedNPC function.

This is all you really need to do in order for the player to interact with NPCs for dialogs and quests. There is a lot more work on the UI that you can do to display all the information you believe is necessary for your game but that will run on your own.

Because there is an overwhelming amount of functions and events that you can use, I took the time to make a simple “Quest Summary” widget, to show one way of doing this. This summary displays only the active quests of the player and for these only show the active tasks. You can add this widget to your own project and also you can check all the logic inside it and how I am using the functions to update and get information from the system.

Back to top.

5.5.4 Localization

The system works with the localization dashboard, allowing to translate the NPC and player talk tasks as well as quest names and short descriptions.

You should create the quests and dialogs in the Quest Editor using the default or main language for your game and then use the Localization Dashboard to translate all the texts to the other supported languages.

Because the system generates the localization keys for the text variables, there are a few things to take into consideration. Do not modify the text variables directly in the datatable, this can reset the localization keys and lose the translation of the text, use the Quest Editor instead to modify them.

Renaming datatables can also lead to changes in the localization keys, is not recommended to do this after you have translated the texts in it.

Back to top.

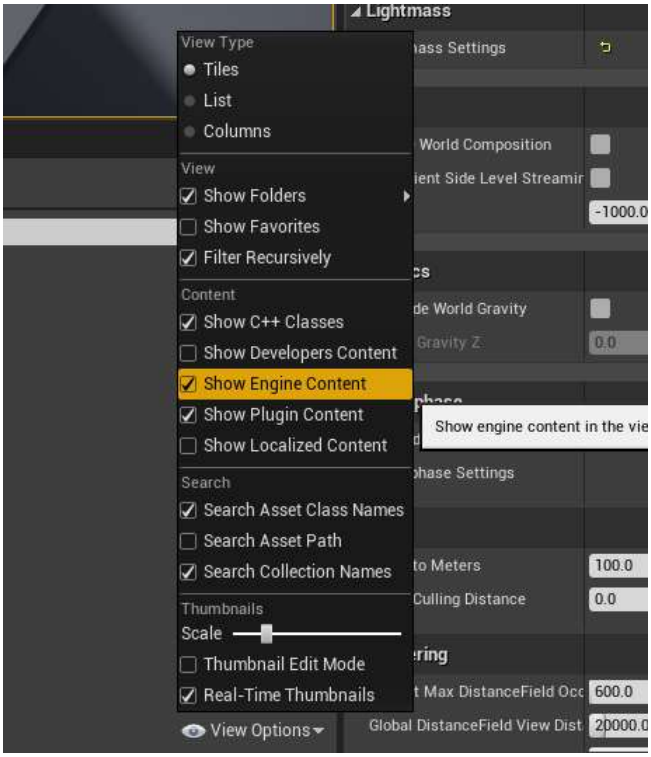
5.6 Examples

The plugin includes several blueprint examples that you can use as base for your implementation or just study for better understanding of the system.

To access these examples inside the editor, you need to add the path for these datatables on the config file.

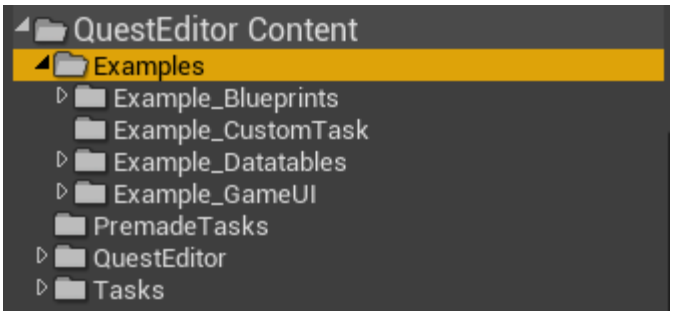
You can find these datatables in the plugin content folder.

If you are unable to see this folder, remember to check the “Show Engine Content” checkbox on the “View Options” section, normally at the bottom right in your editor.



All the examples are contained in the “Examples” folder.

You can use all the content in these examples to set up very quickly a project that has some NPCs, Quests and Dialogs set up.



Back to top.

5.6.1 Example Datatables

I made some datatables examples for some basic ideas of the system, so you can use them to test in game or to look at how they are set up.

Back to top.

5.6.2 Example Game UI

Some widget examples for NPC talk and quest summary are included in the “Example_GameUI” folder. You can use these blueprints to have an idea of how to set up or own widgets. Keep in mind this is a fairly simple implementation and you could make much more complex and prettier UI for your Quest System in game.

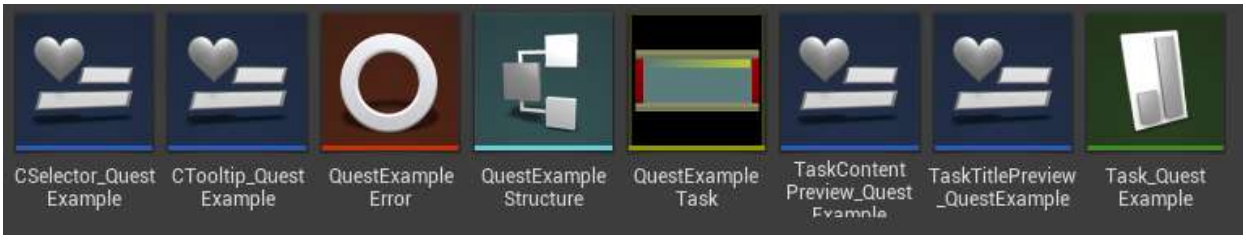
Because these are widgets, you will be able to access them in the widgets panel on UMG, under the “Quests” category. This way you can quickly add them to your project and test the system.

Back to top.

5.6.3 Example Custom Task

Because there are so many things you can do with this system, regarding task creation and editor customization, I took the time to make a custom structure with some random variables and make a task that uses this struct. I also made every custom widget possible, and even made a datatable with all the information that is required to use this task.

You can use this as a base to understand the whole process of making a custom task with a custom structure and using this structure to do some logic inside the task. Also, you can see how to customize every part of the Quest Editor regarding that task, to be able to set the variables properly inside the quest.



Back to top.

5.6.4 Example Blueprints

I provide a very basic implementation of the Player State and NPCs for the system. You can use these to test the system and understand the basic things you need to have for the system to work properly. Check the section 5.3 of this documentation for more information.

Back to top.