



Manual

By Zetan Studio

(I'm not an English speaker, so there're many grammatical errors and some contents are translated by machine,
thanks for your understanding)

Contents

Part I: Dialogue System	4
Section 1: Concepts	4
1.1 Dialogue	4
1.2 Dialogue Node	4
1.3 Dialogue Option	4
Section 2: Dialogue	4
2.1 Dialogue Type	4
2.2 Dialogue Editor	5
Section 3: Basic Node Types	10
3.1 Node Base Type	10
3.2 Sentence Node.....	11
3.3 Option Decoration Node.....	13
3.4 Conditional Display Node.....	14
3.5 Block Node	15
3.6 Suffix Node.....	16
3.7 External Options Node.....	16
3.8 Bifurcation Node	16
3.9 Portrait-Voice Override Node	16
Section 4: Built-In Common Nodes.....	17
4.1 Root Node.....	17
4.2 Exit Node.....	18
4.3 Bold Option Node	19
4.4 Colorful Option Node.....	19
4.5 Italic Option Node.....	20
4.6 Display by Conditions Node	20
4.7 Display Before Done Node	21
4.8 Conditional Block Node	21
4.9 Recursion Node.....	21
4.10 Random Order Options Node	22
4.11 Revert Options Node	23
4.12 Branch Node	24
4.13 Event Node.....	24
4.14 Other-Dialogue Node.....	25
4.15 Bifurcate by Conditions Node	26
4.16 Override Portrait by Condition Node.....	26
4.17 Override Voice by Condition Node	27
Section 5: Interfaces for Node	27
5.1 Solo Main Option Node Interface	27
5.2 Event Node Interface	27
5.3 End Node Interface	27
5.4 Process Manually Node Interface	28
Section 6: Other Important Types.....	28

6.1 Dialogue Option	28
6.2 Dialogue Data	28
6.3 Dialogue Event	30
6.4 Dialogue Group	30
6.5 Dialogue Manager	32
6.6 Dialogue Handler	32
6.7 Interlocutor Interface	35
Part II: Condition System	36
Section 1: Concepts	36
Section 2: Components	36
2.1 Condition	36
2.2 Condition Group	37
Part III: Keyword System	38
Section 1: Concepts	38
Section 2: Components	38
2.1 Keyword Interface	38
2.2 Keyword Type	39
Section 3: Attributes	40
3.1 Keyword Set Attribute	40
3.2 Get Keywords Method Attribute	40
3.3 Runtime Get Keywords Method Attribute	40
Part IV: Multilingual Text System	41
Section 1: Concepts	41
Section 2: Components	41
2.1 Translation Mapping	41
2.2 Translation Set	42
2.3 Localization File	43
2.4 Localization Group	44
Section 3: Edit Language Set	45
3.1 Translation Set Editor	45
3.2 Edit Translation Mappings with Excel	46
Section 5: Translator	47
Section 6: Multilingual Text Component	48
6.1 Static Multilingual Text	48
6.2 Multilingual Text	49
Part V: Save System	50
Section 1: Concepts	50
Section 2: Components	50
2.1 Save Data	50
2.2 Save Manager	50
Section 3: Attributes	50
3.1 Save Method Attribute	50
3.2 Load Method Attribute	51
Part VI: Game UI	52

Section 1: Concepts	52
Section 2: Window System	52
2.1 Window Base Type.....	52
2.2 Window Prefab Collection	53
2.3 Window Manager	53
Section 3: Generic Window	56
Section 4: Dialogue Window.....	56
Part VII: Interaction System	59
Section 1: Concepts	59
Section 2: Components.....	59
2.1 Interactive Interface.....	59
2.2 Base Interactive Object.....	61
Section 3: Interaction UI	62
3.1 Interaction Panel.....	62
3.2 Interaction Window	63
Part VII: Quick Start.....	65
Section 1: Using Interaction System for Conversation.....	65
1.1 Building UI.....	65
1.2 Adding Characters.....	68
1.3 Creating and Editing Dialogue.....	71
Section 2: Trigger Conversation in Other Way.....	73
2.1 Building UI.....	73
2.2 Creating and Editing Dialogue.....	75
Section 3: Beautify Dialogue Window	76
Appendix.....	80
Editor Settings.....	80
Classes.....	80
1. Lightweight Generic Data.....	80
2. Singleton Asset.....	82
3. Singleton Window	82
Interfaces	83
1. Player Name Holder Interface.....	83
2. Copiable Interface.....	83
3. Fade Able Interface	83
4. Scene Loader Interface	84
5. Message Displayer Interface	84
Attributes	84
1. Initialization Attribute	84
2. Initialization Method Attribute	84

Part I: Dialogue System

Section 1: Concepts

1.1 Dialogue

[Dialogue](#) is a kind of ScriptableObject asset, which consists of dialogue nodes and dialogue options.

1.2 Dialogue Node

Dialogue node is the basic element of a dialogue. The type of dialogue node is [DialogueNode](#), it's an abstract class, your custom node types need to inherit this type, then you can implement their function, there are already built-in basic function node types in this plugin, such as [sentence node](#), which be used to write sentence, and [condition node](#), which be used to decide should an option display, and etc. They will be expanded in detail below, and only mentioned here.

1.3 Dialogue Option

Now that the dialogue options are mentioned above, here's an overview: Dialogue options, like their name, are used to show options when player have conversation with NPC. Dialogue options are divided into main options and normal options. The main options are not actual options (will not be displayed on UI), they are used to connect next sentence, hypothesis that there's no options to display in current sentence, but we need a way to switch to next sentence, so we need an agent to connect current sentence node and next sentence node, that's the main option. The main options are also used to connect special nodes, such as [exit node](#), which be used to mark where should the dialogue finish and close the [dialogue window](#). The normal options are actual options that display on UI, and their visibility can be determined by condition node that is mentioned above.

Section 2: Dialogue

2.1 Dialogue Type

The type of dialogue is Dialogue, inherit from ScriptableObject, its members are as follows:

Members For Runtime (Non-method members only show their names)	Explanation
Field: ID	The unique identifier of dialogue, which references the ID of root node of dialogue
Field: nodes	All nodes of dialogue, include nodes that cannot traverse to
Property: Nodes	Related read only property of field "nodes"
Property: Entry	The root node of dialogue
Property: Exitable	Dose this dialogue have any exit node
Constructor: DialogueNode()	Create an root node automatically, and add it to "nodes"
Method: bool Reachable(DialogueNode)	Check if the root node of this dialogue can traverse to given node

Static Method: bool Reachable (DialogueNode, DialogueNode)	Check if the specified node can traverse to the target node
Static Method: void Traverse (DialogueNode, Action<DialogueNode>)	Traverse through the specified node and its child nodes from the node itself
Static Method: void Traverse (DialogueNode, Func<DialogueNode, bool>)	Traverse through the specified node and its child nodes from the node itself, this can be aborted, and the return value represents that did any aborting happen while traversing

Its Editor-Use members are as follows:

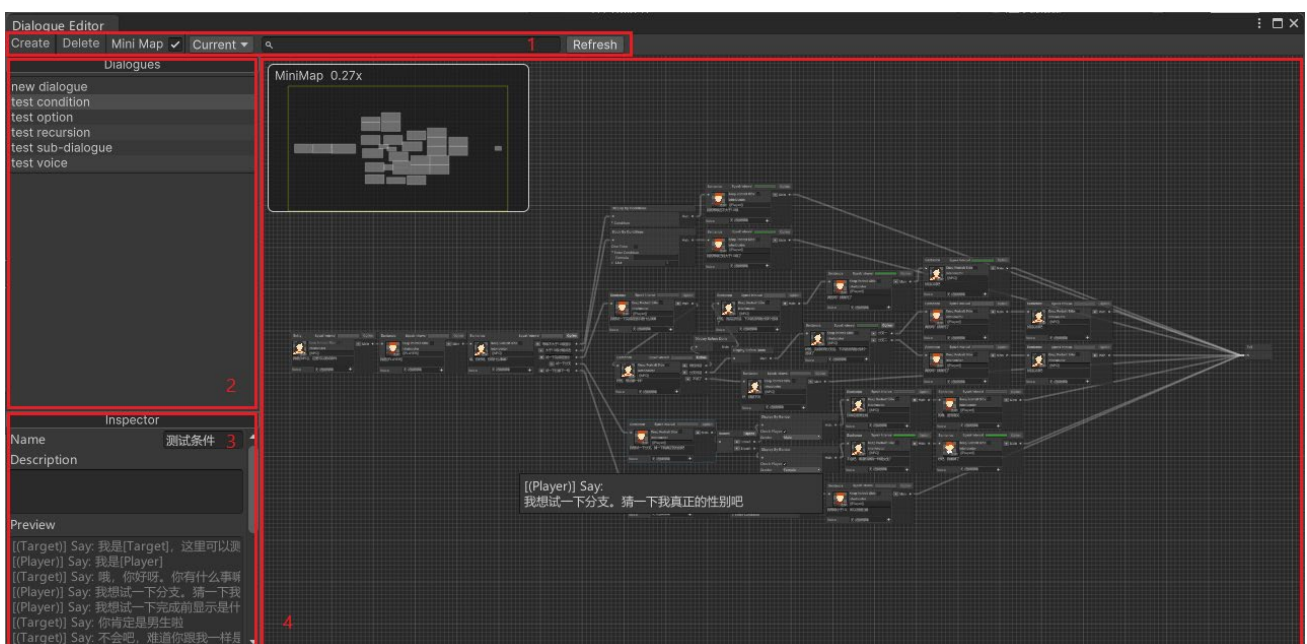
Members For Editor	Explanation
Field: _exit	The exit node of the dialogue, use for setting exit-here node in editor
Field: _name	Name of this dialogue
Field: _description	Description of this dialogue, use for noting in editor
Field: _groups	Node groups of this dialogue, use for setting up node groups in editor

The Dialogue type also has an embedded static class "Editor" for editor use. Its members are as follows:

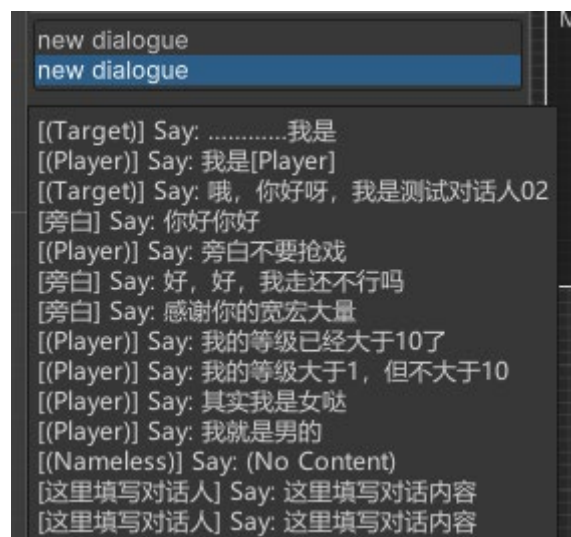
Members For Editor	Explanation
Static Method: DialogueNode AddNode (Dialogue, Type)	Add a node of the specified type to the dialogue
Static Method: void AddCopiedNode (Dialogue, DialogueNode)	Add a copied node to the dialogue
Static Method: void RemoveNode (Dialogue, DialogueNode)	Remove a node from the dialogue

2.2 Dialogue Editor

The dialogue editor is an independent editor window. It abandons the traditional inspector-mode and performs graphical node-to-node connecting editing for the dialogue asset files, as shown in the figure:

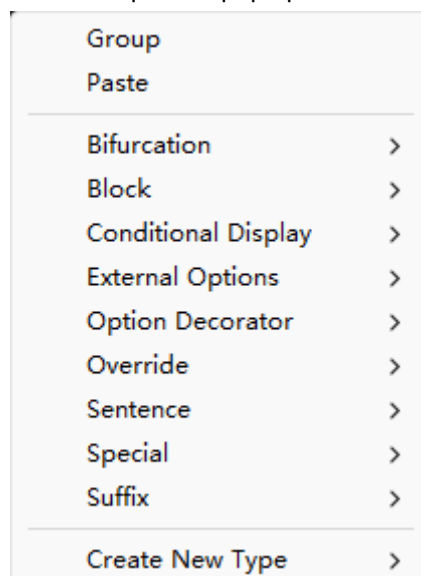


Area 1 is the toolbar area. Click the “Create” button to create a new dialogue in the specified path. Click the “Delete” button to delete the selected dialogues. The “Mini Map” check box can switch the display and hide of the mini map in the upper left corner of Area 4. The search box can select global and current local searches. When performing a global search, it will search for the dialogue ID, name, description, and the node's ID, interlocutor, speech content, option title and type (by adding a “t:” prefix to the front of the keyword), when performing a local search, only the latter five will be searched; a global search for the first three will locate dialogue, and the latter five or a local search will locate node. You should note that you cannot search rich-text tags as they will be ignored when searching. Area 2 is the list of all dialogue asset files (.asset) in the project. Left-click the list item to select the dialogue, and refresh Area 4 as the view of the selected dialogue at the same time. Right-click the list item to delete, locate and rename the dialogue asset file; hover the mouse pointer over the list item to preview the sentence nodes in the dialogue, as shown in the figure:

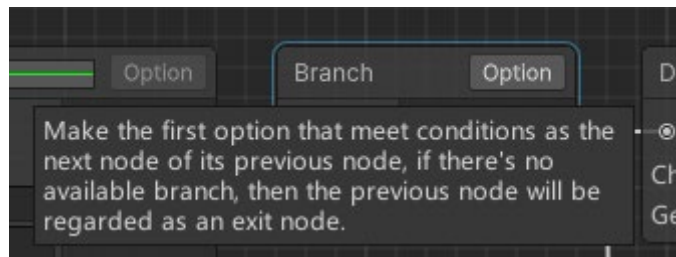


Area 3 is the inspector area. When there's not node is selected, it's the inspector of selected dialogue; when a node is selected, it's the inspector of that node, then you can set the node more deeply (some fields of the node will not be displayed in the graph node).

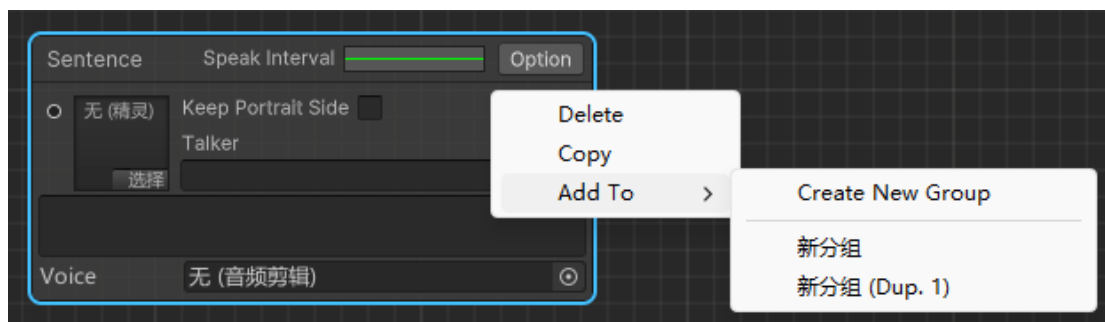
Area 4 is the dialogue editing view, which constructs the dialogue by connecting nodes directly. Press and hold the left mouse button and move the mouse to make box selection, press and hold middle mouse button to drag view, and mouse wheel to zoom. Right-click the blank space to pop up the node creation menu, as shown in the figure:



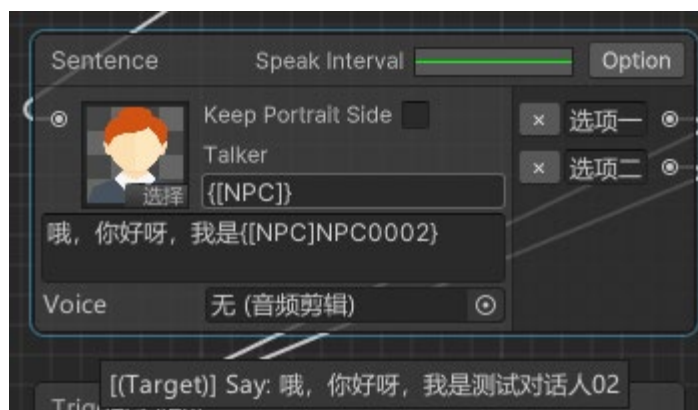
Click menu item to create the specified content. The “Paste” menu item will not appear until there’re nodes get copied. Hover the mouse pointer over the name label in the upper left corner of the node to view the function description of that node, as shown in the figure:




Right-click a node to group, copy, or delete it, as shown in the figure:



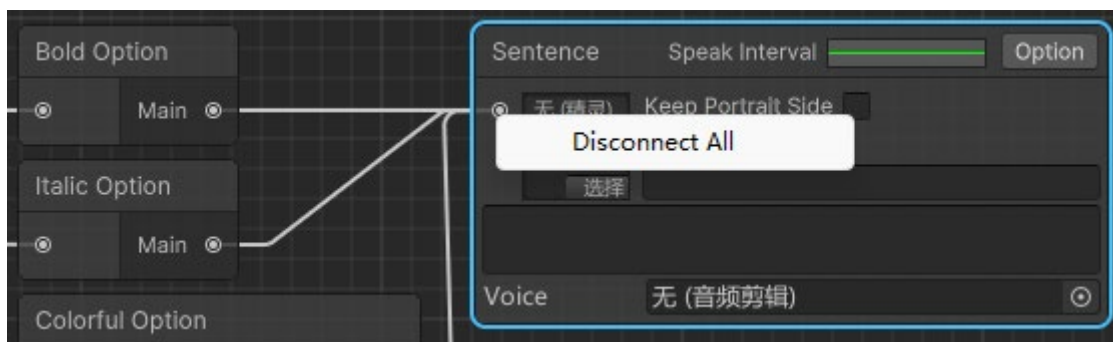
For sentence nodes, hover the mouse pointer over the blank space of them to preview the text of them below themselves, and the [keyword](#) ID string will be converted to the related name, as shown in the figure:




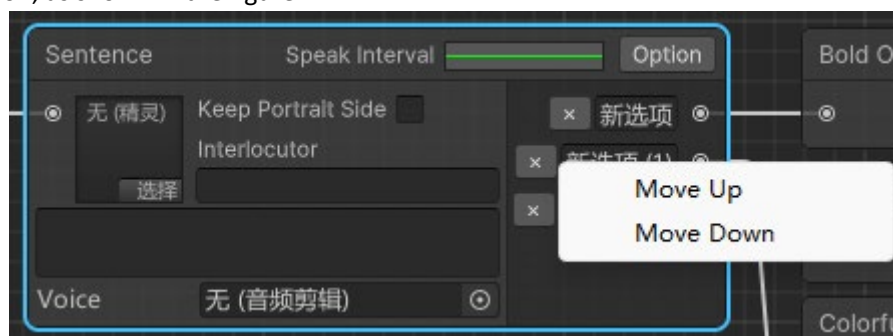
You can use the shortcut keys “Ctrl + C”, “Ctrl + V”, “Ctrl + X”, and “Delete” to copy, paste, and delete nodes. You can also use “Ctrl + Z”, “Ctrl + Y”, or the  button in the upper right corner of Unity to undo and redo the changes.

When pasting nodes, the pasted nodes will be placed centered on the position of the mouse pointer;

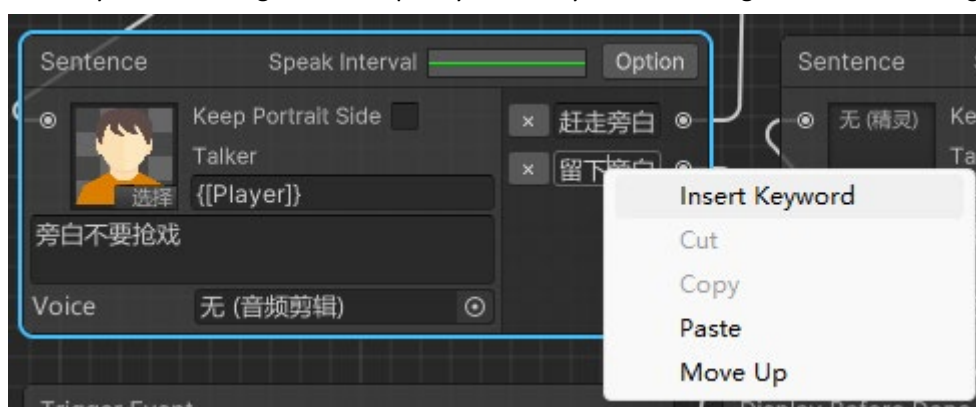
Right-click the entry end of the node to quickly disconnect all connections connected to it, as shown in the figure:



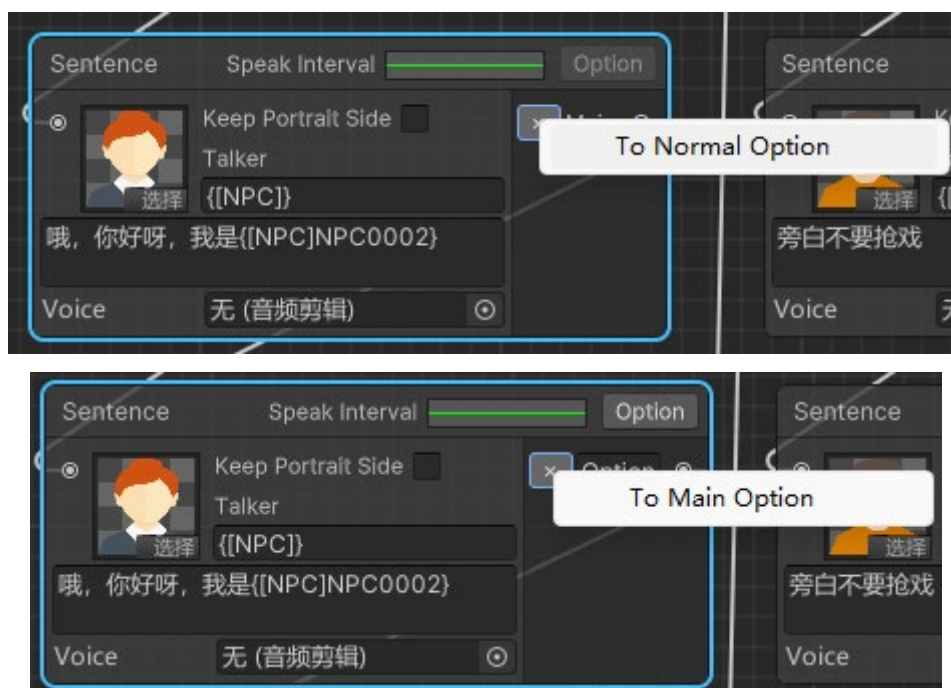
Click the  button on the left of the node's option (that is, the output port) to delete it, and right-click the option to move its position, as shown in the figure:



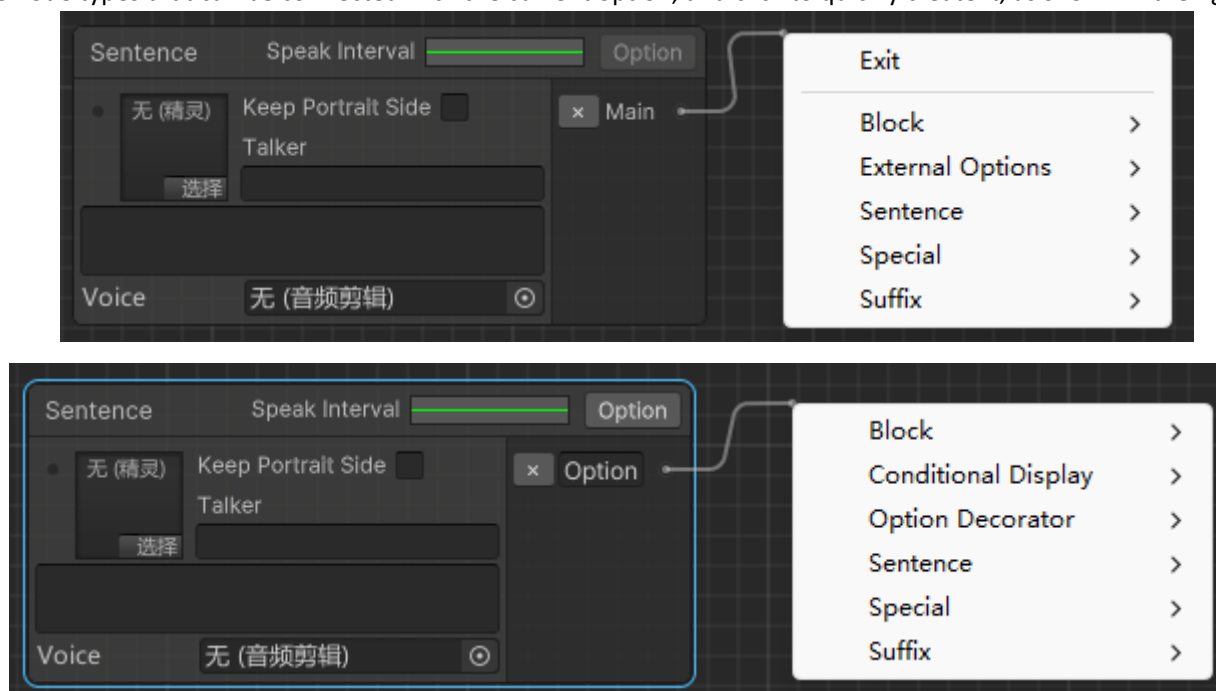
The difference between the main option and the normal option is that the title of the main option is “main” and cannot be edited, while the title of the normal option is an input box that can be modified, if there is no selected text in this input box, you can also right-click to quickly insert keywords ID string, as shown in the figure:



For sentence nodes and [other-dialogue nodes](#), if their main option can be replaced by a normal option, right-click the main option to turn it into a normal option. Similarly, if there is only a single normal option, you can also turn it into a main option, as shown in the figure:

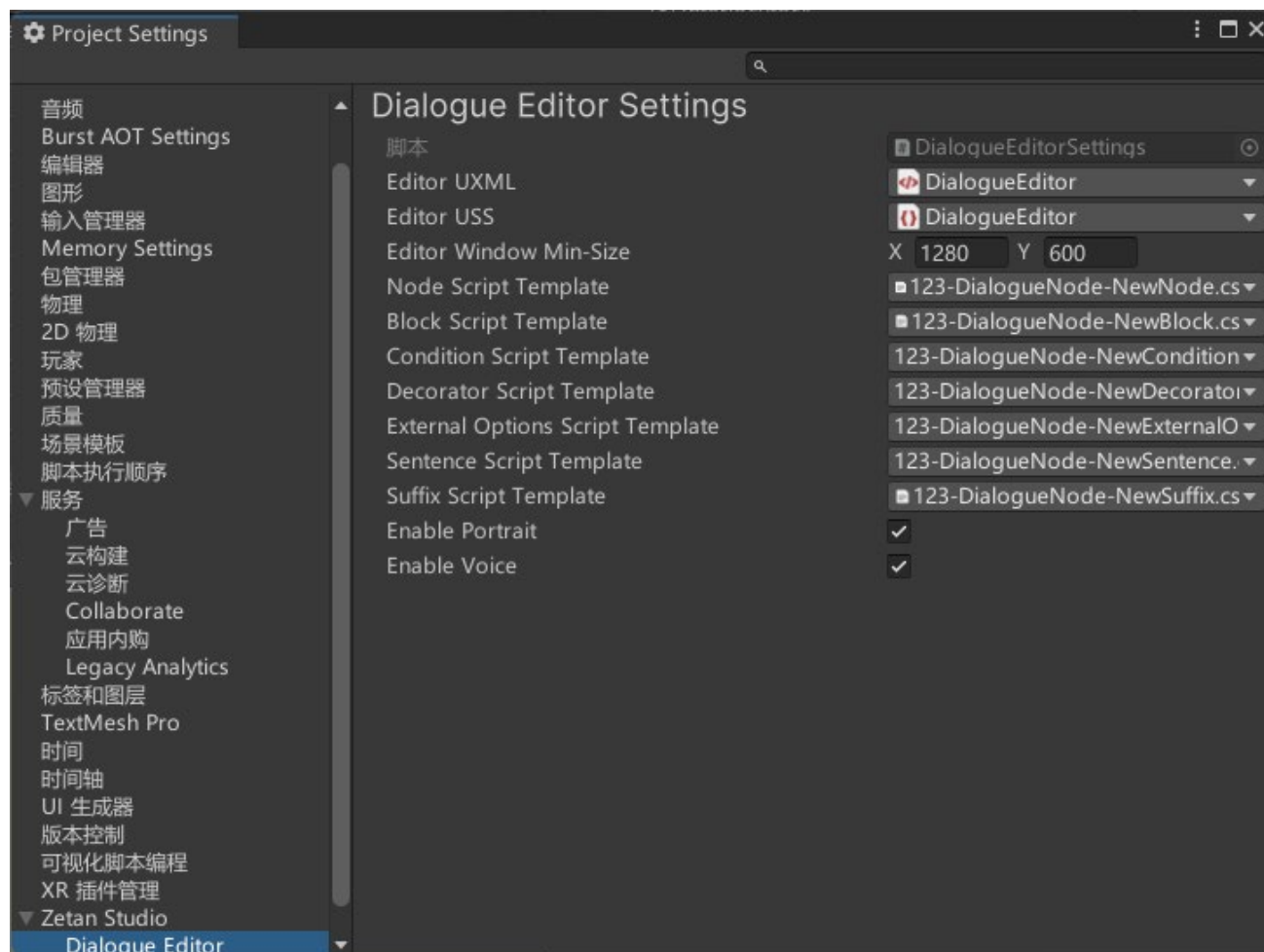


Drag the edge of the option to the blank and release the mouse to pop up the node-quickly-insert menu, which lists the node types that can be connected with the current option, and click to quickly create it, as shown in the figure:



Among them, “Exit” can quickly connect the option to the exit node instead of creating a new exit node.

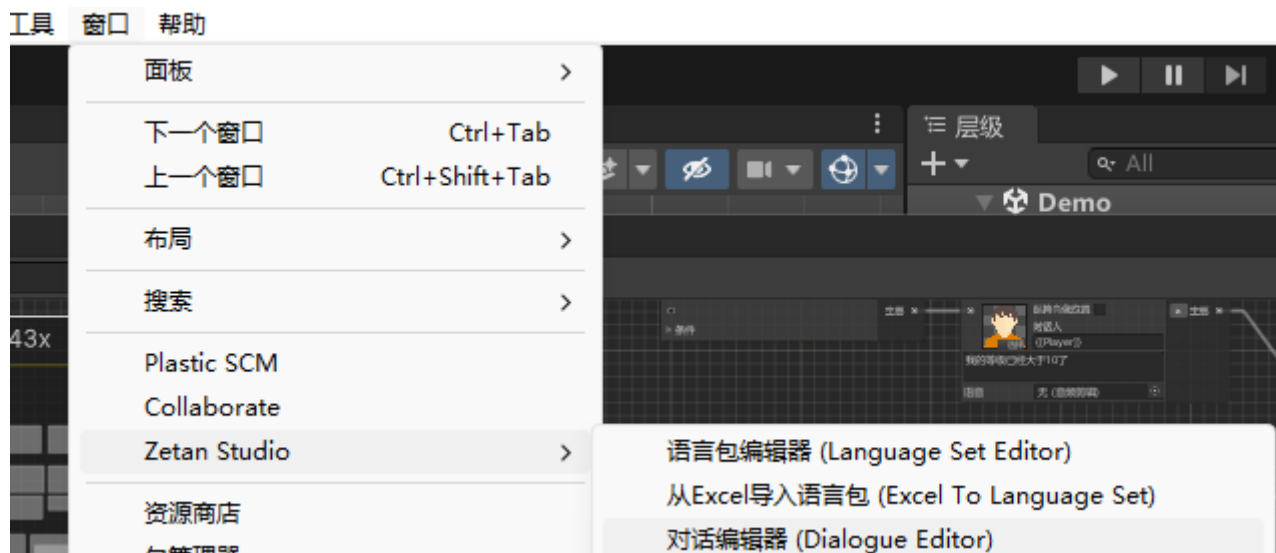
The dialogue editor can be set in “Edit→Project Settings...→Zetan Studio→Dialogue Editor”, as shown in the figure:



Please do not delete the UXML and USS files, which will make the dialogue editor unusable; “Enable Portrait” and “Enable Voice” are checked by default. If you don’t need these two functions, please uncheck them by yourself. After unchecking, the project code will be recompiled. Please wait patiently for completion, do not repeat checking, unchecking or other redundant operations.

How to open the dialogue editor:

Double-click the dialogue asset file, or click the “Open” button in the upper right corner of them, or open it by click “Window→Zetan Studio→对话编辑器 (Dialogue Editor)” from the toolbar of Unity, as shown in the figure:



Section 3: Basic Node Types

3.1 Node Base Type

As the name implies, the node base type is the base type of all node types. As mentioned above, the type is DialogueNode, and its members are as follows:

Members For Runtime	Explanation
Property: ID	The unique identifier of node, it's automatically generated, but you can modify it of course
Field: options	Internal option array of node, which stores its options
Property: Options	Related read only property of field “options”
Field: exitHere	Used to identify whether the dialogue ends at this node
Property: ExitHere	Related read only property of field “exitHere”, the prerequisite for returning Ture is that the node type must inherit the interface IExitableNode
Property: Exitable	Used to identify whether the exit node can be reached from this node
Property: IsValid	Used to check whether the node's content is valid, otherwise the node will not be handled in the dialogue
Indexer: DialogueOption this[int index]	A shortcut to index in the node option array, it is a read only indexer
Operator: bool(DialogueNode)	Used to quickly check if the node is null

Its Editor-Use members are as follows:

Members For Editor	Explanation
Field: <code>_position</code>	Used to record the node position in the editor
Method: <code>string GetName()</code>	Get the custom name of this node (not the type name)
Method: <code>bool CanConnectFrom (DialogueNode, DialogueOption)</code>	Used to determine whether an option of another node can be connected to this node
Method: <code>bool CanConnectTo (DialogueNode, DialogueOption)</code>	Used to determine whether an option of this node can be connected to another node
Method: <code>HashSet<string> GetHiddenFields()</code>	Used to filter the fields to be hidden in the graph node
Method: <code>DialogueNode Copy()</code>	Used to copy nodes in editor

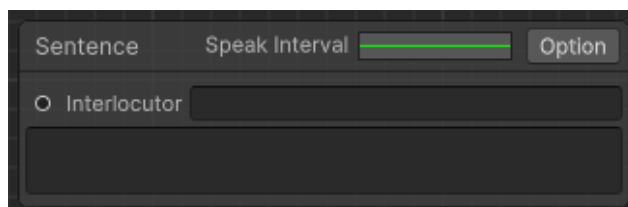
The DialogueNode type also has an embedded static class “Editor” for editor use. Its members are as follows:

Members For Runtime	Explanation
Static Method: <code>string GetGroup(Type)</code>	Get the group name of given node type
Static Method: <code>string GetName(Type)</code>	Get the custom name of given node type
Static Method: <code>string GetDescription(Type)</code>	Get the description of given node type
Static Method: <code>float GetWidth(Type)</code>	Get the graph node’s width of given node type
Static Method: <code>DialogueOption AddOption (DialogueNode, bool, string)</code>	Add an option to the node
Static Method: <code>void RemoveOption (DialogueNode, DialogueOption)</code>	Remove an option from the node
Static Method: <code>void MoveOptionUpward (DialogueNode, int)</code>	Move an option upward
Static Method: <code>void MoveOptionDownward (DialogueNode, int)</code>	Move an option downward
Static Method: <code>void SetAsExit (DialogueNode, bool)</code>	Set the field “exitHere” of given node

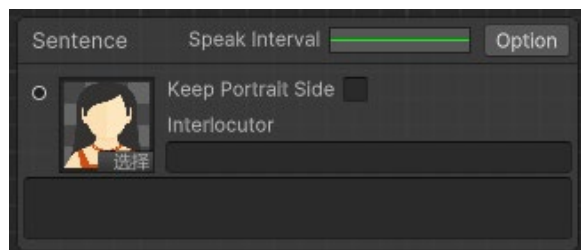
3.2 Sentence Node

The sentence node is the most basic unit in a dialogue. It is used to write the interlocutor’s name and speech content for display on the UI. Its graph node in the dialogue editor is shown in the figure:

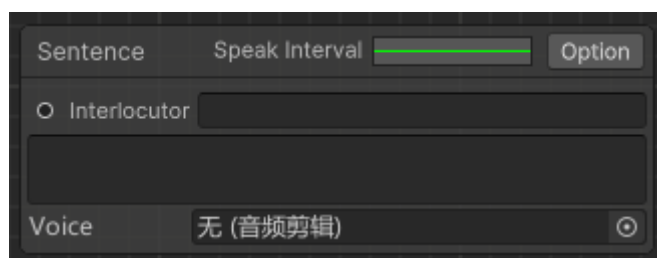
The basic state:



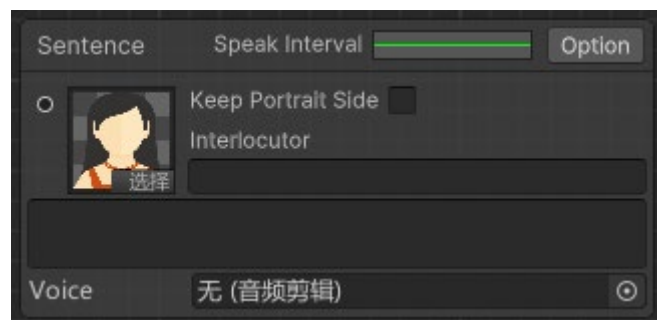
When enable portrait function:



When enable voice function:



When enable both portrait and voice function:



The type of sentence node is `SentenceNode`, inherit from `DialogueNode`, [IExitableNode](#), [IEventNode](#), its members are as follows:

Members For Runtime	Explanation
Property: Interlocutor	String of interlocutor's name
Property: Content	String of speech content
Property: SpeakInterval	The speaking interval is the displaying interval between the text characters when typing animation playing
Property: Portrait	The portrait of the interlocutor, available when the portrait function is enabled
Property: KeepPortraitSide	It is used to identify whether to keep the portrait's side, available when the portrait function is enabled
Property: VoiceOffset	Indicate the start time when playing voice, available when the portrait function is enabled
Property: Voice	The voice to be played when the dialogue window displays this sentence, available when the voice function is enabled
Field: events	Events that this sentence will triggers
Property: Events	Related read only property of field "events"
Property: IsValid	When the interlocutor and content are not empty and all events are valid, this sentence node is valid, too.

For the property "KeepPortraitSide", when setting the portrait in dialogue window, it will first compare the interlocutor's name of the current sentence with previous one, and then automatically select whether to display it on the left or the right, make it form an opposition. If the name is the same, then will not change the side. If the name of interlocutor will be changed for some reason, and you do not want to change its portrait's side, you can set this property to True.

How to use:

Except for branch nodes, the options of other types of nodes can be directly connected to the sentence node, make it be the next sentence.

In the input field of interlocutor and content, if no text is selected, you can right-click to open the keyword objects selection window, and its ID string will be inserted after selection, as shown in the figure:



3.3 Option Decoration Node

The option decoration node is used to decorate the option title text with rich text in the dialogue window, such as bold, italic, random color, etc.

The type of option decoration node is DecoratorNode, inherit from DialogueNode, [SoloMainOptionNode](#), its members are as follows:

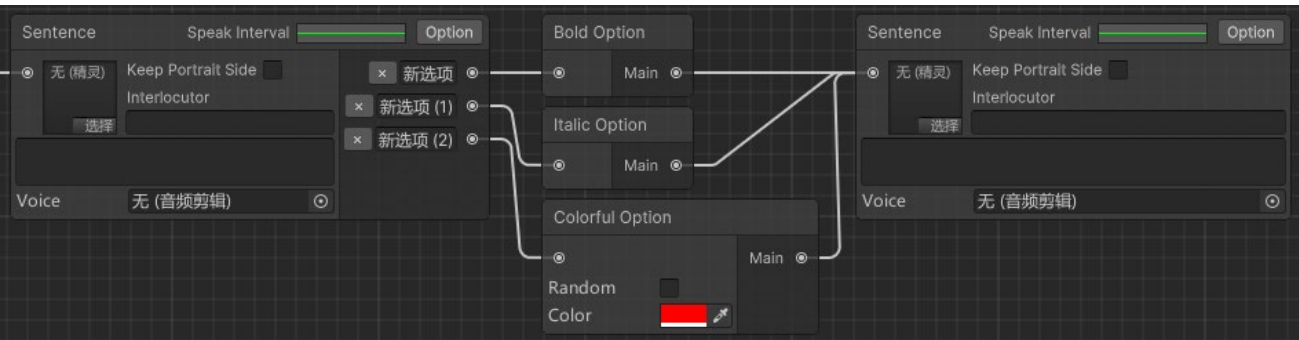
Members For Runtime	Explanation
Constructor: DecoratorNode()	Create a main option automatically
Method: void Decorate(DialogueData, ref string)	The method to decorate option title

Its Editor-Use member is as follows:

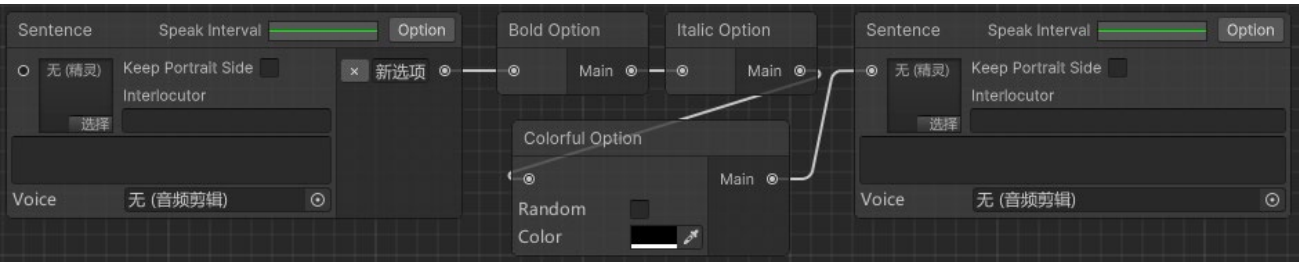
Members For Editor	Explanation
Method: bool CanConnectFrom (DialogueNode, DialogueOption)	Used to specify that only normal options or decoration nodes can be connected to this type of node

How to use:

Connect the normal options to option decoration node, as shown in the figure:



The decoration nodes can also be connected in series to form a mixed decoration, as shown in the figure:



3.4 Conditional Display Node

The conditional display node is used to determine whether the branch from it should be displayed. When it is connected to an option, it will affect whether the option will be displayed in the dialogue window, that is, whether to cull it; when it is connected to a branch node, the actual next sentence of branch node will be affected.

The type of conditional display node is ConditionNode, inherit from DialogueNode, ISoloMainOptionNode, its members are as follows:

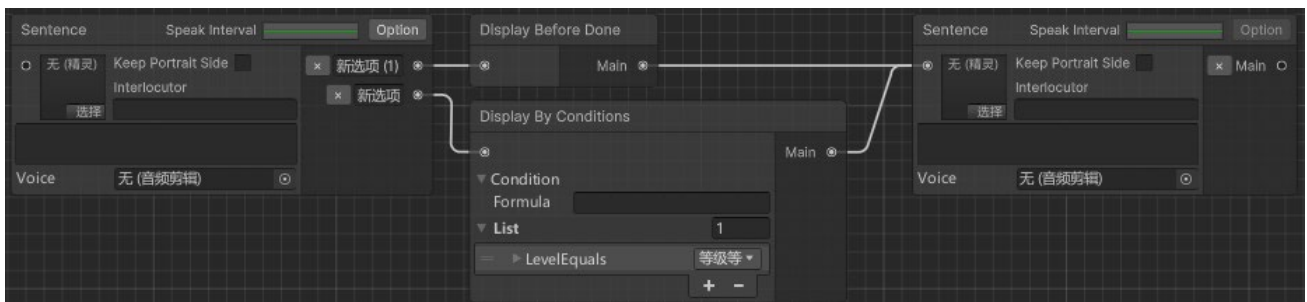
Members For Runtime	Explanation
Constructor: ConditionNode ()	Create a main option automatically
Method: bool Check(DialogueData)	Check whether the conditions are met, and it will judge the conditional display nodes in series
Method: bool CheckCondition(DialogueData)	Internal condition checking method for its subclass override, used by the previous method

Its Editor-Use member is as follows:

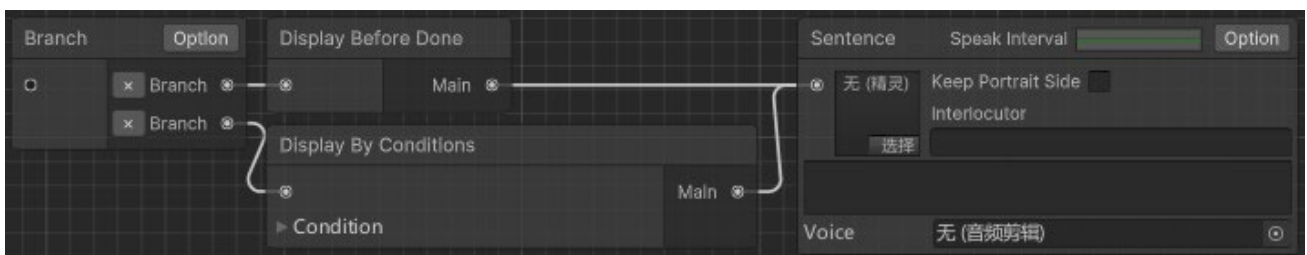
Members For Editor	Explanation
Method: bool CanConnectFrom (DialogueNode, DialogueOption)	Used to specify that only normal options or conditional display nodes or branch nodes can be connected to this type of node

How to use:

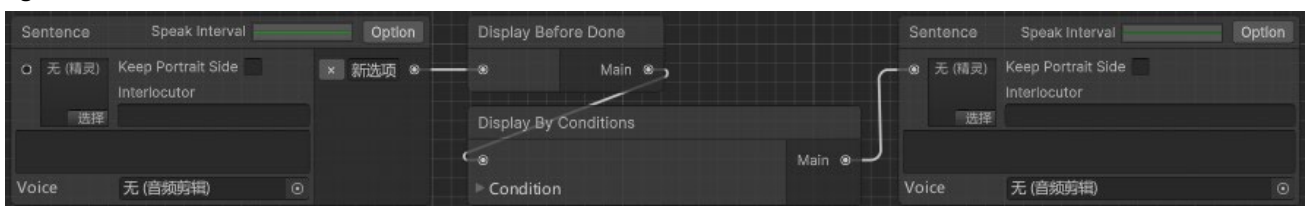
Connect the normal options to conditional display node, as shown in the figure:



You can also connect options of branch node to conditional display node, as shown in the figure:



Of course, the conditional display nodes can also be connected in series to form a mixed condition, as shown in the figure:



You should note that if all options can reach the end node and are all culled, it will force the first of them to display.

3.5 Block Node

The block node is used to determine whether the dialogue window should switch to the next sentence connected by the option when clicking it. For example, the player can enter the branch where the specified option is connected to only when he holds a specific item.

The type of block node is BlockNode, inherit from DialogueNode, ISoloMainOptionNode, its members are as follows:

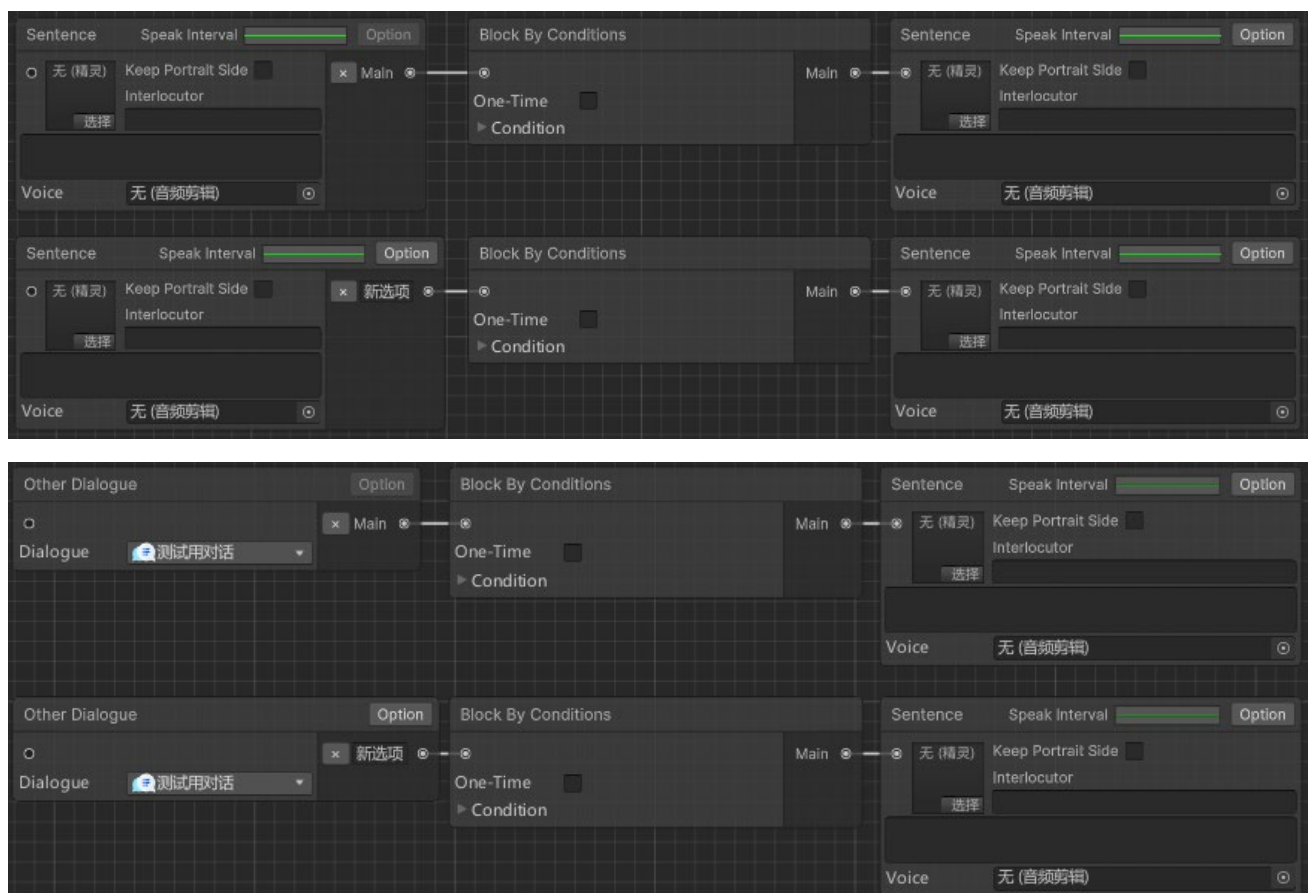
Members For Runtime	Explanation
Property: OneTime	Whether to make this node stop blocking after meeting the enter conditions once, even if the conditions are not met this time
Constructor: BlockNode ()	Create a main option automatically
Method: bool CanEnter(DialogueData, out string)	Check if can enter the branch from this node
Method: bool CheckCondition()	Check enter conditions, returns True if can be entered
Method: string GetBlockReason()	Get a reason message string to tip player why this branch is blocked

Its Editor-Use member is as follows:

Members For Editor	Explanation
Method: bool CanConnectFrom (DialogueNode, DialogueOption)	Used to specify that only sentence nodes or other-dialogue nodes or block nodes can be connected to this type of node

How to use:

Connect the options of sentence node or other-dialogue node to block node, as shown in the figure:



The block node can also be connected in series like the conditional display node, I won't repeat it here.

3.6 Suffix Node

Suffix node is a kind of node that has no option.

The type of suffix node is `SuffixNode`, inherit from `DialogueNode`.

3.7 External Options Node

The external options node can externalize the normal options of its previous node to achieve additional customized option display functions, such as display in random order.

The type of external options node is `ExternalOptionsNode`, inherit from `DialogueNode`, its member is as follows:

Members For Runtime	Explanation
Method: <code>ReadOnlyCollection<DialogueOption> GetOptions(DialogueData, DialogueNode)</code>	Get the actual options, such as options in random order

Its Editor-Use member is as follows:

Members For Editor	Explanation
Method: <code>bool CanConnectFrom(DialogueNode, DialogueOption)</code>	Used to specify that only main options of sentence nodes or other-dialogue nodes can be connected to this type of node

3.8 Bifurcation Node

A bifurcation node is usually used to bifurcate the dialogue flow, it has only two fixed main options for making two branches. It also has no specific implementation, and what its branches are used for depends entirely on user needs.

The type of bifurcation node is `BifurcationNode`, inherit from `DialogueNode`, its members are as follows:

Members For Runtime	Explanation
Property: <code>First</code>	Start node of the first branch
Property: <code>Second</code>	Start node of the second branch
Constructor: <code>BifurcationNode ()</code>	Create two main options automatically
Method: <code>bool CheckIsDone(DialogueData)</code>	As its function is customized, so do its “done” state

3.9 Portrait-Voice Override Node

The portrait-voice override node can replace the portrait or voice of the nearest sentence node which is connect with it and in front of it, so it can be used to implement functions such as replacing portrait and voice by gender of player. Portrait-voice override node can be connected in series, too, and when it’s connected in series, then it will use the first from left to right override node which its override portrait or voice is not null to override.

The type of portrait-voice override node is `PortraitVoiceOverrideNode`, inherit from `DialogueNode`, `IXitableNode`, `ISoloMainOptionNode`, its member is as follows:

Members For Runtime	Explanation
Constructor: <code>PortraitVoiceOverrideNode()</code>	Create two main options automatically

Its Editor-Use members are as follows:

Members For Editor	Explanation
Method: <code>bool CanConnectFrom(DialogueNode, DialogueOption)</code>	Used to specify that only main options of sentence nodes or portrait-voice override nodes can be connected to this type of node

Method: bool CanConnectTo (DialogueNode, DialogueOption)	Used to specify that this type of nodes can only use main options to connected to sentence nodes, portrait-voice override nodes and revert options nodes
---	--

The portrait-voice override node can be subdivided into portrait override node and voice override node, which have one more method to obtain override portrait and override voice respectively.

The type of portrait override node is PortraitOverride, inherit from PortraitVoiceOverrideNode its members are as follows:

Members For Runtime	Explanation
Method: Sprite GetPortrait(DialogueData)	An abstract method to get portrait to override

The type of voice override node is VoiceOverride, inherit from PortraitVoiceOverrideNode its member is as follows:

Members For Runtime	Explanation
Method: AudioClip GetVoice(DialogueData)	An abstract method to get voice to override

How to use:

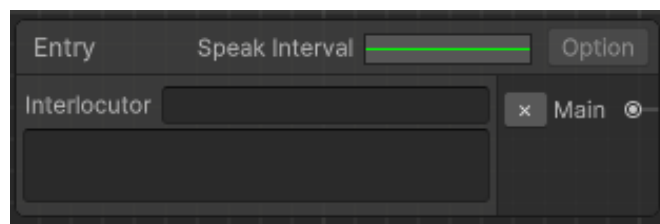
Connect the main option to override node, that will do. As the overridden sentence nodes need to use their main options to connect override nodes, so if the overridden nodes need normal options, you can connect the main options of the last override nodes to revert options nodes to set and revert options that the sentence nodes should have.

Section 4: Built-In Common Nodes

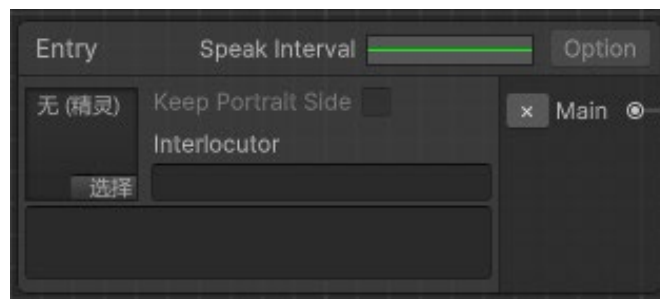
4.1 Root Node

The root node is a kind of sentence node. It is the initial node of a dialogue. There should be and only be one, it has no front node. The dialogue window will traverse and process the dialogue nodes from this node, even the temporary dialogue that initiated by interlocutor and speech content string will generate a temporary root node for them. Therefore, it will set field "exitHere" to True by default. Its graph node in the dialogue editor is shown in the figure:

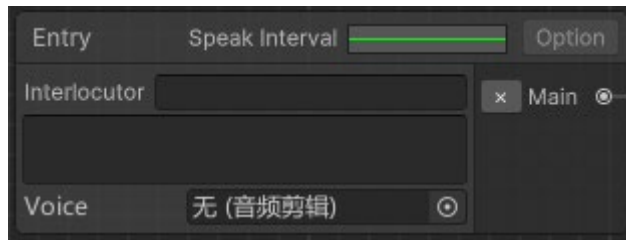
The basic state:



When enable portrait function:



When enable voice function:



When enable both portrait and voice function:



The type of root node is EntryNode, inherit from SentenceNode, its members are as follows:

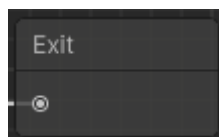
Members For Runtime	Explanation
Property: Interruptable	Used to identify whether the dialogue window can be closed to end the conversation in advance
Constructor: EntryNode()	Set a special ID, create a main option automatically, and set itself as an end node (that is, its "exitHere" is set to True)
Constructor: EntryNode(string, string, bool)	Set the interlocutor and speech content strings and property "Interruptable" based on the previous constructor
Constructor: EntryNode (string, string, string, bool)	Set a custom ID based on the previous constructor

Its Editor-Use member is as follows:

Members For Editor	Explanation
Method: bool CanConnectFrom (DialogueNode, DialogueOption)	Used to specify that no node can be connected to this type of node

4.2 Exit Node

The exit node is a suffix node that is used to set the end node in the editor. It only exists in the editor code, when the game is built, it will not be found, so do not use it in the game logic. Its graph node in the dialogue editor is shown in the figure:



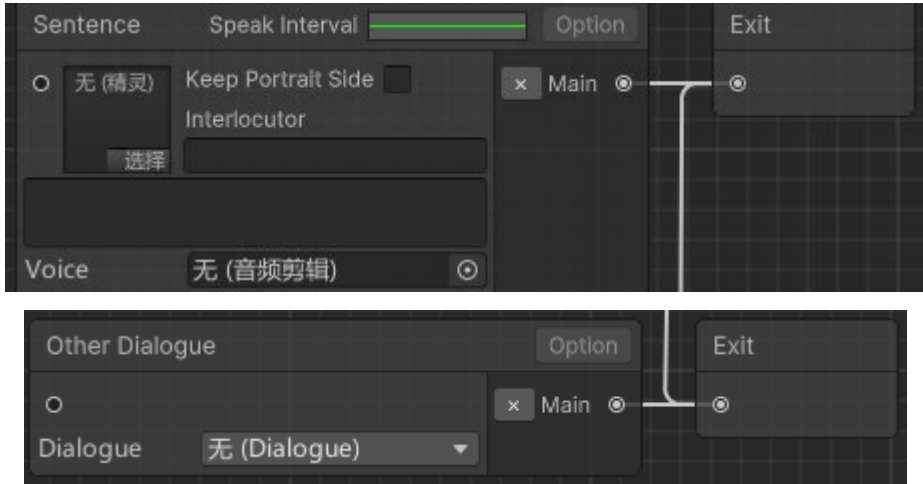
The type of exit node is ExitNode, inherit from SuffixNode, its Editor-Use members are as follows:

Members For Runtime	Explanation
Property: IsValid	Returns True
Constructor: ExitNode()	Automatically set a specified position to avoid overlapping with the root node when creating a new dialogue

Method: bool CanConnectFrom (DialogueNode, DialogueOption)	It is used to specify that only the main options of nodes that inherit the IExitableNode interface can be connected to this type of nodes
---	---

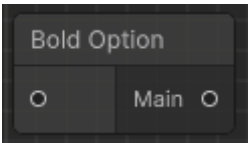
How to use:

Connect the main option of node that inherit the IExitableNode interface to exit node, as shown in the figure:



4.3 Bold Option Node

The bold option node is a kind of option decoration node, will makes the dialogue window display the option title text in bold. Its graph node in the dialogue editor is shown in the figure:



The type of bold option node is BoldDecorator, inherit from DecoratorNode, its members are as follows:

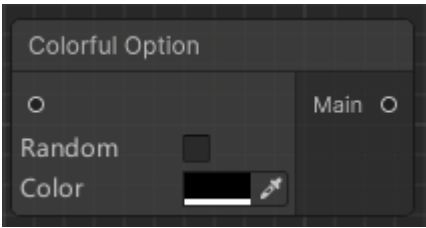
Members For Runtime	Explanation
Property: IsValid	Returns True
Method: void Decorate(DialogueData, ref string)	Output option title string that decorated with bold rich text

See the option decoration node type for the usage method.

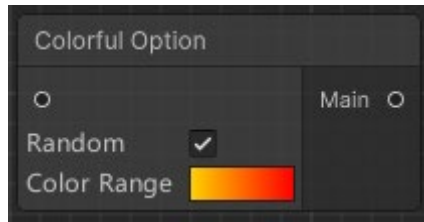
4.4 Colorful Option Node

The colorful option node is a kind of option decoration node, will makes the dialogue window display the option title text in specified color or a random color that is in range. Its graph node in the dialogue editor is shown in the figure:

When uncheck the random color:



When check the random color:



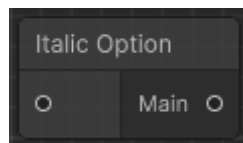
The type of colorful option node is ColorfulDecorator, inherit from DecoratorNode, its members are as follows:

Members For Runtime	Explanation
Property: RandomColor	Whether to use random color
Property: Gradient	Color range of the rich text, available when "RandomColor" is set to True
Property: Color	Color of the rich text, available when "RandomColor" is set to False
Property: IsValid	Returns True if the color is random or not transparent
Method: void Decorate(DialogueData, ref string)	Output option title string that decorated with color rich text

See the option decoration node type for the usage method.

4.5 Italic Option Node

The italic option node is a kind of option decoration node, will makes the dialogue window display the option title text in italic. Its graph node in the dialogue editor is shown in the figure:



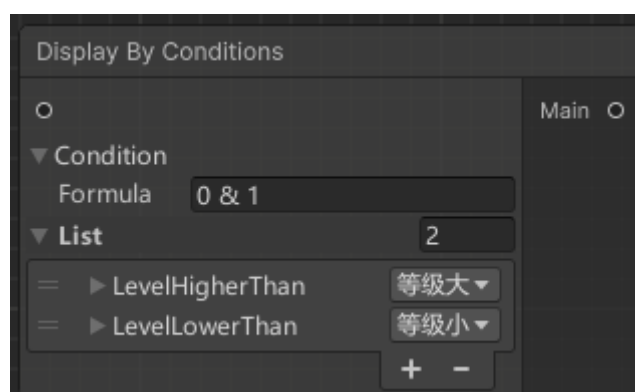
The type of italic option node is ItalicDecorator, inherit from DecoratorNode, its members are as follows:

Members For Runtime	Explanation
Property: IsValid	Returns True
Method: void Decorate(DialogueData, ref string)	Output option title string that decorated with italic rich text

See the option decoration node type for the usage method.

4.6 Display by Conditions Node

The display by conditions node is a kind of conditional display node, it makes a branch be displayed conditionally. Its graph node in the dialogue editor is shown in the figure:



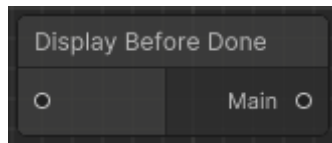
The type of display by conditions node is `CommonCondition`, inherit from `ConditionNode`, its members are as follows:

Members For Runtime	Explanation
Property: Condition	Conditions to be met, its type is ConditionGroup
Property: IsValid	Returns True if "Condition" is valid
Method: bool CheckCondition(DialogueData)	Returns True if "Condition" is met

See conditional display node type and condition group type for usage method.

4.7 Display Before Done Node

The display before done node is a kind of conditional display node, it makes a branch be displayed before it is done ([when is it done?](#)). Its graph node in the dialogue editor is shown in the figure:



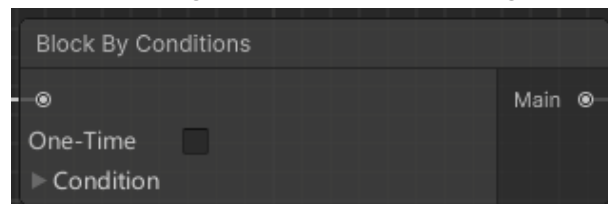
The type of display before done node is `DeleteOnDoneCondition`, inherit from `ConditionNode`, its members are as follows:

Members For Runtime	Explanation
Property: IsValid	Returns True
Method: bool CheckCondition(DialogueData)	Returns True if the node that its main option connected to is done

See conditional display node type for usage method.

4.8 Conditional Block Node

The conditional block node is a kind of block node. You can enter the branch starting from it only when the specified conditions are met. Its graph node in the dialogue editor is shown in the figure:



The type of conditional block node is `ConditionalBlock`, inherit from `BlockNode`, its members are as follows:

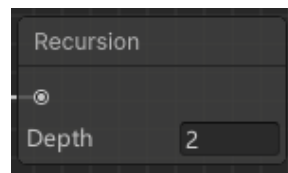
Members For Runtime	Explanation
Property: BlockReason	The message to display when block happens
Property: Condition	Enter conditions to be met, its type is <code>ConditionGroup</code>
Property: IsValid	Returns True if "Condition" is valid
Method: bool CheckCondition()	Returns True if "Condition" is met
Method: string GetBlockReason()	Return block message to tip player

See block node type for usage method.

4.9 Recursion Node

The recursion node is a kind of suffix node, also called "return node", it is used to return to specified sentence node

in front of it, as it will returns to this node again at some times, so it's called recursion node. Its graph node in the dialogue editor is shown in the figure:



The type of recursion node is RecursionSuffix, inherit from SuffixNode, [IManualNode](#), its members are as follows:

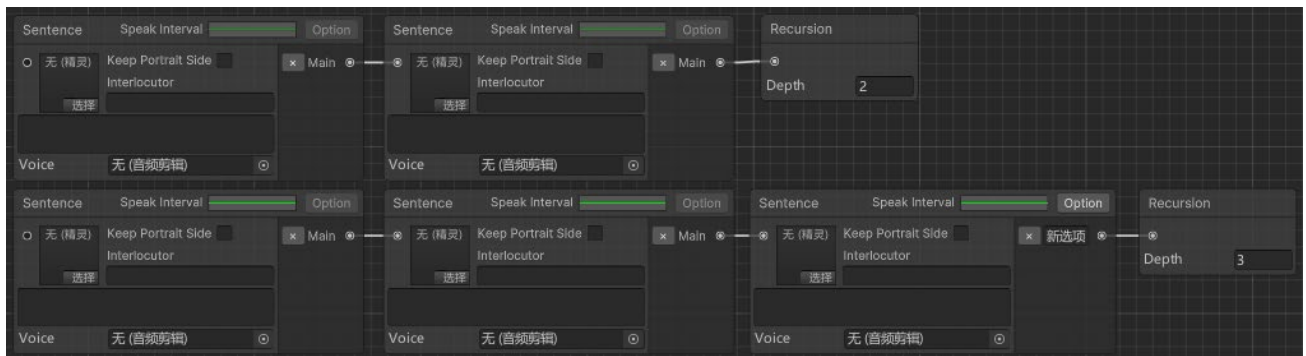
Members For Runtime	Explanation
Property: Depth	Recursion depth, indicates how many sentence nodes should be counted forward
Property: IsValid	Returns true when "Depth" is greater than 1
Method: DialogueNode FindRecursionPoint (EntryNode)	Seek the sentence node to return to, seek until meet the root node
Method: void DoManual(DialogueWindow)	Continue conversation with node the previous method found

Its Editor-Use member is as follows:

Members For Editor	Explanation
Method: bool CanConnectFrom (DialogueNode, DialogueOption)	It is used to specify that only the options of sentence nodes except root node can be connected to this type of nodes

How to use:

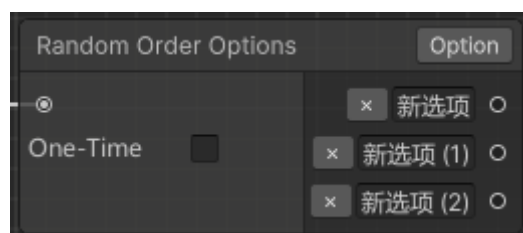
Connect the options of sentence node except root node to recursion node, as shown in the figure:



Note that there should be at least one sentence node between the sentence node to return and the recursion node, that is the sentence node which owns the option connected to the recursion node can't be the node to return, right-click recursion node to locate the node to return.

4.10 Random Order Options Node

The random order options node is a kind of external options node, which is used to display the options that its previous node externalizes here, in random order. Its graph node in the dialogue editor is shown in the figure:



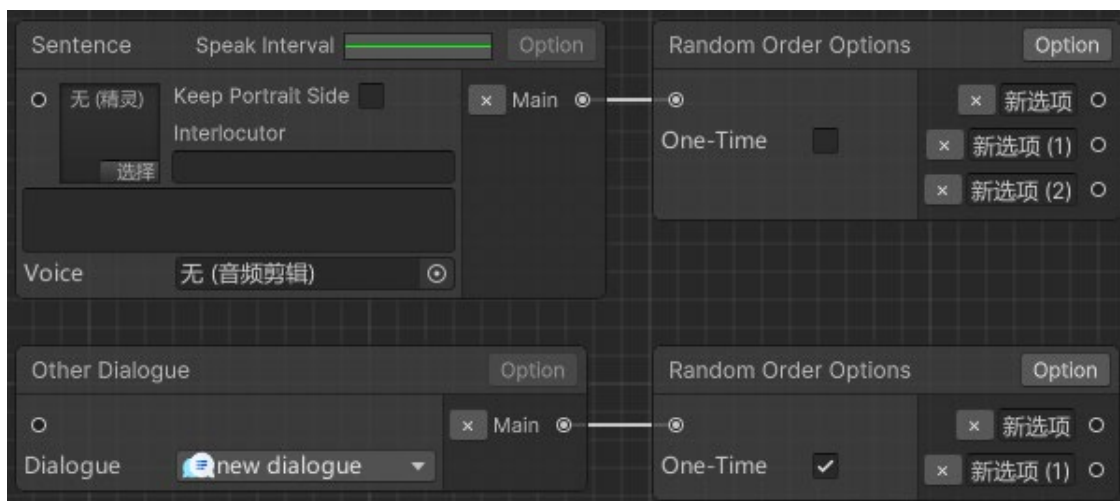
The type of random order options node is RandomOptions, inherit from ExternalOptionsNode, its members are as

follows:

Members For Runtime	Explanation
Property: OneTime	Whether to generate random order only at the first time
Property: IsValid	Returns True
Method: ReadOnlyCollection<DialogueOption> GetOptions(DialogueData, DialogueNode)	Get the options that have random order

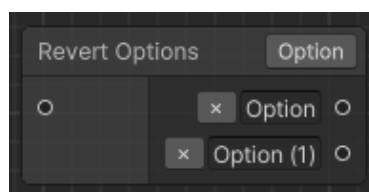
How to use:

Connect the main options of sentence node or other-dialogue node to random order options node, as shown in the figure:



4.11 Revert Options Node

The revert options node is a kind of external options node, it's used to externalize the options should have of the sentence node which needs main option to connect portrait-voice override node and cannot have normal options. Its graph node in the dialogue editor is shown in the figure:

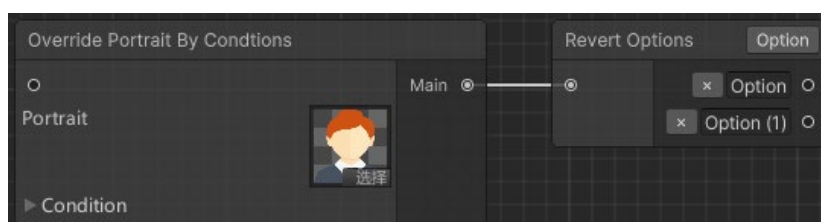


Type of revert options node is RevertOptions, inherit from ExternalOptionsNode, its members are as follows:

Members For Runtime	Explanation
Property: IsValid	Returns True
Method: ReadOnlyCollection<DialogueOption> GetOptions(DialogueData, DialogueNode)	Returns the options of this node, that is property "Options"

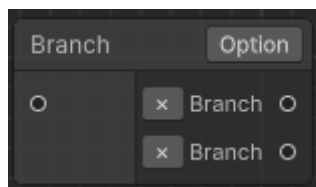
How to use:

Connect the main option of override node to revert options node, as shown in the figure:



4.12 Branch Node

A branch node can make the first branch that meets the display conditions as the next sentence of the previous node. If there is no available branch, then the previous node will be regarded as an end node. Its graph node in the dialogue editor is shown in the figure:



The type of branch node is `BranchNode`, inherit from `DialogueNode`, its members are as follows:

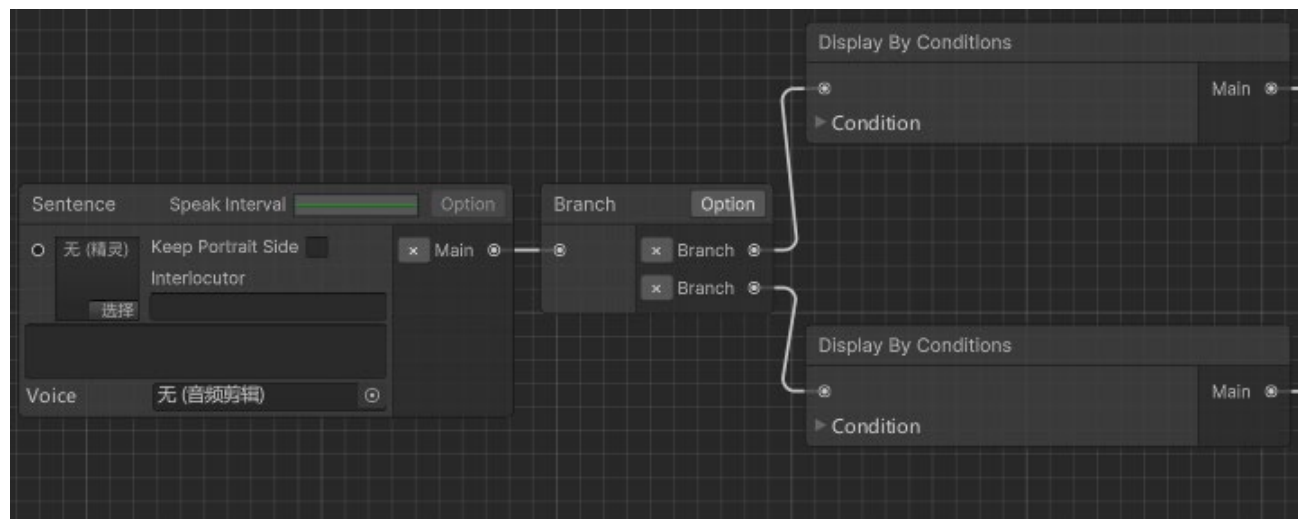
Members For Runtime	Explanation
Property: <code>IsValid</code>	Returns True when there's at least one option and they are all main options
Method: <code>DialogueNode GetBranch (DialogueData)</code>	Get the first branch that satisfies the display conditions

Its Editor-Use member is as follows:

Members For Editor	Explanation
Method: <code>bool CanConnectFrom (DialogueNode, DialogueOption)</code>	It is used to specify that only the main options of sentence nodes and other-dialogue nodes can be connected to this type of nodes

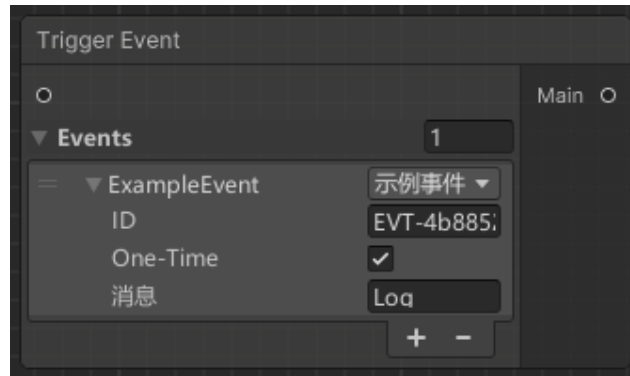
How to use:

Connect the main options of sentence node or other-dialogue node to branch node, and then connect the options of branch node to conditional display nodes, as shown in the figure:



4.13 Event Node

When entering an event node, the given events will be triggered, and then the conversation will continue with its next node. Although sentence nodes also have events, they all trigger events at themselves, but the event nodes can be connected to different options to trigger different events, and they can also be used to trigger events before entering the target node. Its graph node in the dialogue editor is shown in the figure:

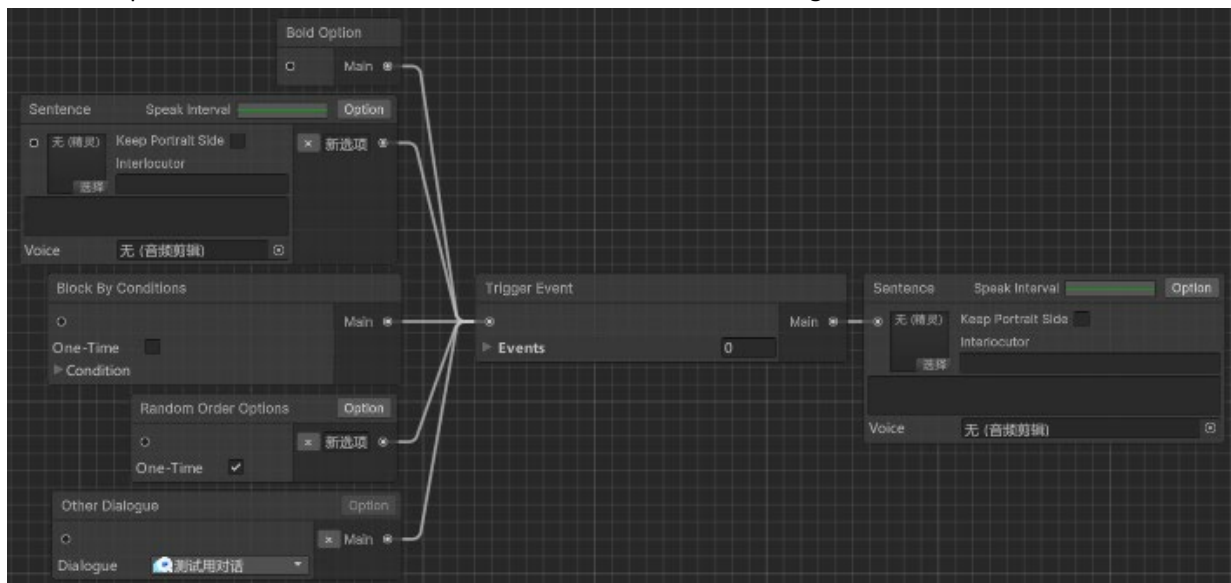


The type of event node is EventNode, inherit from DialogueNode, IEventNode, ISoloMainOptionNode, IManualNode, its members are as follows:

Members For Runtime	Explanation
Property: IsValid	Returns True when all events are valid
Field: events	Events to trigger
Property: Events	Related read only property of field “events”
Constructor: EventNode()	Create a main option automatically
Method: void DoManual(DialogueWindow)	Continue conversation with its next node after trigger events

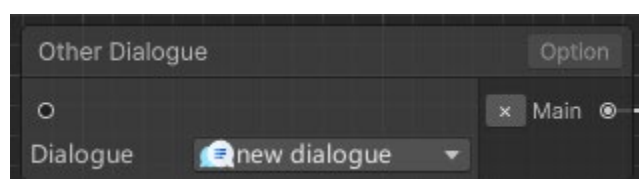
How to use:

Connect the options of non-branch node to event node, as shown in the figure:



4.14 Other-Dialogue Node

The other-dialogue nodes will start a new conversation, but will keep the root node of the first dialogue as the home page (that is, the node to return when click “Home” button of the dialogue window), and return to the original dialogue at the end. It is suitable for scene that the current dialogue is too complicated, and some nodes need to be separated into other sub-dialogs. Its graph node in the dialogue editor is shown in the figure:



The type of other-dialogue node is OtherDialogueNode, inherit from DialogueNode, IExitableNode, IManualNode, its members are as follows:

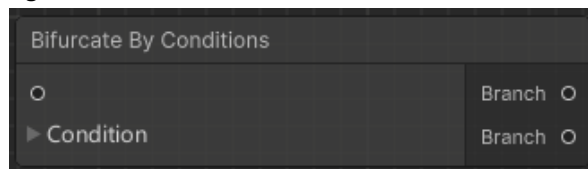
Members For Runtime	Explanation
Property: Dialogue	The sub-dialogue to proceed
Property: IsValid	Returns True if the sub-dialogue is not null and has end node
Method: void DoManual(DialogueWindow)	Start a new conversation, and return to the original dialogue at the end

How to use:

Refer to the sentence node for use, I won't repeat it here. What to note is, the options of other-dialogue nodes are equivalent to the options of the last sentence of the sub-dialogue, for example, its normal options will be displayed on UI as the options of the last sentence of the sub-dialogue.

4.15 Bifurcate by Conditions Node

Obviously, the bifurcate by conditions node is a kind of bifurcation node. When the conditions are met, the first branch will be used to continue the conversation, otherwise the second one will be used. Its graph node in the dialogue editor is shown in the figure:

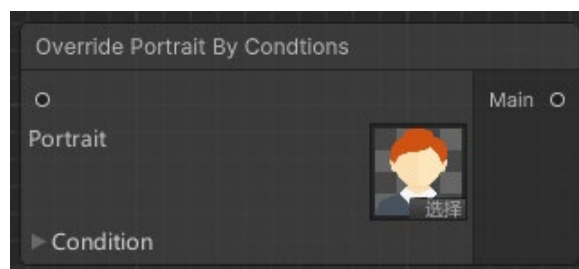


The type of bifurcate by conditions node is ConditionalBifurcation, inherit from BifurcationNode, IManualNode, its members are as follows:

Members For Runtime	Explanation
Property: Condition	Conditions to be met, its type is ConditionGroup
Property: IsValid	Returns True if "Condition" is valid
Method: void DoManual(DialogueWindow)	When the conditions are met, the first branch will be used to continue the conversation, otherwise the second one will be used.

4.16 Override Portrait by Condition Node

The override portrait by condition node is a kind of portrait override node, when getting portrait to override, if the conditions are met, then it will return the reserved portrait, otherwise return null, so that the overridden sentence node can keep the original portrait. Its graph node in the dialogue editor is shown in the figure:

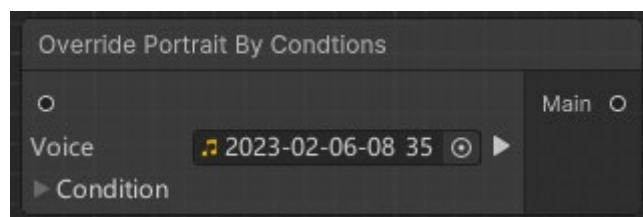


Type of override portrait by condition node is ConditionalPortraitOverride, inherit from PortraitOverride, its members are as follows:

Members For Runtime	Explanation
Property: Portrait	The portrait used to replace
Property: Condition	Conditions to be met, its type is ConditionGroup
Property: IsValid	Returns True if portrait is not null and “Condition” is valid
Method: Sprite GetPortrait(DialogueData)	If the conditions are met, then this method will return the reserved portrait, otherwise return null

4.17 Override Voice by Condition Node

The override voice by condition node is a kind of voice override node, when getting voice to override, if the conditions are met, then it will return the reserved voice, otherwise return null, so that the overridden sentence node can keep the original voice. Its graph node in the dialogue editor is shown in the figure:



Type of override voice by condition node is ConditionalVoiceOverride, inherit from VoiceOverride, its members are as follows:

Members For Runtime	Explanation
Property: Voice	The voice used to replace
Property: Condition	Conditions to be met, its type is ConditionGroup
Property: IsValid	Returns True if portrait is not null and “Condition” is valid
Method: Sprite GetPortrait(DialogueData)	If the conditions are met, then this method will return the reserved voice, otherwise return null

Section 5: Interfaces for Node

5.1 Solo Main Option Node Interface

The solo main option node interface is ISoloMainOptionNode, inherit this interface if you want the node to only have one main option. You need to create a main option in its constructor if a node type inherits this interface.

5.2 Event Node Interface

The event node interface is IEventNode, inherit this interface if you want to invoke event when a node gets done. Its member is as follows:

Members For Runtime	Explanation
Property: Events	Events to trigger

5.3 End Node Interface

The end node interface is IExitableNode, inherit this interface if you want the node can be an end node.

5.4 Process Manually Node Interface

The process manually node interface is `IManualNode`, inherit this interface if you want handle a type of node in other custom way.

Its member is as follows:

Members For Runtime	Explanation
Method: <code>void DoManual(DialogueHandler)</code>	The custom method to handle this node

Section 6: Other Important Types

6.1 Dialogue Option

The dialogue options are important elements of non-suffix nodes, they show as output port in the dialogue editor. The type of dialogue option is `DialogueOption`, its members are as follows:

Members For Runtime	Explanation
Property: Title	Title of the option, that is the text to display on UI options
Property: IsMain	Indicates whether the option is a main option
Property: Next	The node that the option connected
Static Property: Main	A shortcut to create new main option instance
Constructor: <code>DialogueOption()</code>	Parameter-less constructor for serialization
Constructor: <code>DialogueOption(bool, string)</code>	Constructor that passes in option's main state and its title
Operator: <code>bool(DialogueData)</code>	Used to quickly check if the option is null

The `DialogueOption` type also has an embedded static class "Editor" for editor use. Its members are as follows:

Members For Editor	Explanation
Static Method: <code>void SetNext(DialogueOption, DialogueNode)</code>	Set the option's connected node
Static Method: <code>void SetIsMain(DialogueOption, bool)</code>	Set if the option is a main option

6.2 Dialogue Data

The dialogue data type is a class used to store the progress of the dialogue. For example, if you want to implement the function that an option will not appear next time after you clicking on it once, you need the help of this class.

The type of dialogue data is `DialogueData`, its members are as follows:

Members For Runtime	Explanation
Field: ID	The unique identifier of this data, that is the ID of its related node
Field: accessed	The visit state of its related node, indicates whether it has been handled
Property: Accessed	Related read only property of field "accessed"
Field: children	Data of the connected nodes of the options of its related node
Property: Children	Related read only property of field "children"

Field: family	All data of nodes of the dialogue that owns the related node
Property: Family	Related read only property of field “family”
Field: eventStates	The trigger states of the related node’s events
Property: EventStates	Related read only property of field “eventStates”
Property: IsDone	Whether its related node is “done”, see below for detail
Property: AdditionalData	A generic data for sub-class of the base node type
Indexer: DialogueData this[DialogueNode]	A shortcut to access the data of given node
Indexer: DialogueData this[string]	A shortcut to access the data of node that given ID specifies
Indexer: DialogueData this[int]	A shortcut to access the data at the given index of “children”
Constructor: DialogueData(EntryNode)	Input a root node to generate the data of all dialogue’s nodes
Constructor: DialogueData(DialogueNode, Dictionary<string, DialogueData>)	Generate data of the given node, and store it to the given dictionary, this dictionary is their “family” in the fact
Constructor: DialogueData(GenericData)	Input a generic data to generate the data of all dialogue’s nodes
Constructor: DialogueData (string, Dictionary<string, DialogueData>)	Generate an empty data with given ID, and store it to the given dictionary, this dictionary is their “family” in the fact
Method: void Refresh(EntryNode)	This is used to refresh and override original data after the dialogue’s structure change, such as adding and deleting nodes and changing connections, etc.
Method: void Access()	Set “accessed” to True to specify its related node is visited
Method: void AccessEvent(string)	This is used to record the trigger states of one-time events
Method: GenericData GenerateSaveData()	Get a generic data for saving system
Static Method: void Traverse (DialogueData, Action<DialogueData>)	Traverse through the specified data and its “children” from the data itself
Static Method: bool Traverse (DialogueData, Func<DialogueData, bool>)	Traverse through the specified data and its “children”, from the data itself, and this method can be aborted, and the return value represents that did any aborting happen while traversing
Operator: bool(DialogueData)	Used to quickly check if the data is null

When a dialogue data is mark as “IsDone”? This plugin stipulates it as follows:

1. When the node has no subsequent nodes or it’s an end node, its “IsDone” state is its visit state;
2. When the node is block node, if it can be entered and data of its next node is marked as “done”, then its data will be marked as “done”, too;
3. When the node is conditional display node, if the conditions are met and data of its next node is marked as “done”, then its data will be marked as “done”, too;
4. When the node is branch node, if there’s no available branch, then its “IsDone” state is its visit state; If there’s available branch and data of the first node of that branch is marked as “done”, then its data will be marked as “done”;
5. When the node is external options node, if it has been accessed, and its actual options exist an option that its connected node can reach the exit node, and that node has been accessed, then the data of this node will be marked as “done”.
6. Except for the above, if the node is not bifurcation node (as its “done” state is customized) and it has been accessed, and data of all the connected nodes of its options are marked as “done”, then data of this node will be marked as “done”.

6.3 Dialogue Event

The dialogue event type can be used to trigger event in some case in the conversation, generally held by nodes that inherit IEventNode interface. When the conversation proceeds to node which has dialogue events, it will trigger specified events, such as Debug events, trigger setting events, etc., so that it can be used elsewhere.

The type of dialogue event is DialogueEvent, its members are as follows:

Members For Runtime	Explanation
Property: ID	The unique identifier of event, it's automatically generated, but you can modify it of course
Property: OneTime	Indicates whether this event only trigger once, is set to True by default
Property: IsValid	Whether the content of this event is valid
Method: void Invoke(DialogueData)	Try to trigger this event
Method: void CancelInvoke(DialogueData)	Cancel triggering, gets called when the dialogue window is closed or start new dialogue
Method: bool Invoke()	An event triggering method for sub-class, the return value indicates whether the event get triggered successfully
Static Method: string GetName(Type)	Get name of an event type

Its Editor-Use member is as follows:

Members For Editor	Explanation
Method: object Copy()	Copy this event, can be override by sub-class

How to expand:

A sample code for printing event is given below:

```
[SerializeField, Name("Example Event ")]
public sealed class ExampleEvent : DialogueEvent
{
    [field: SerializeField] public string Message { get; private set; }

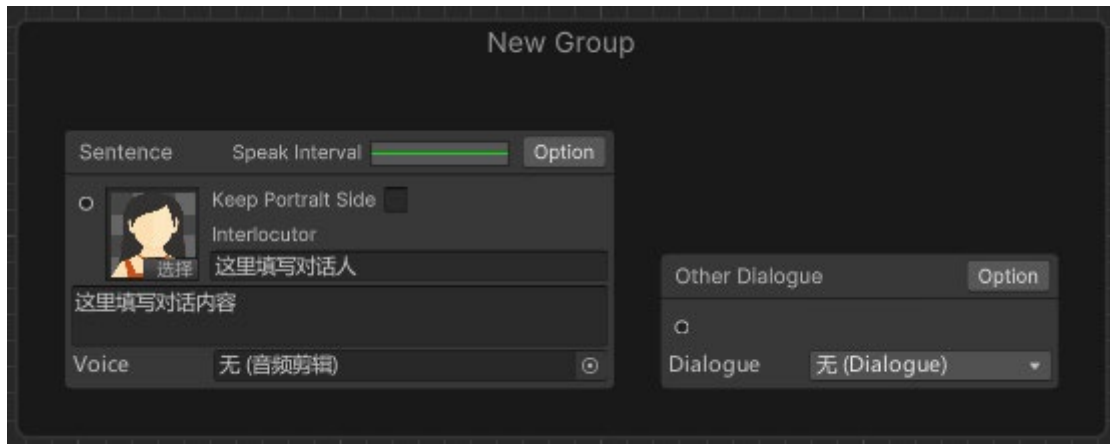
    public override bool IsValid => !string.IsNullOrEmpty(Message);

    protected override bool Invoke()
    {
        Debug.Log(Message);
        return true;
    }
}
```

6.4 Dialogue Group

The dialogue group is used to group the graph nodes in editor, should not use in game logic code. Its graph group in the dialogue editor is shown in the figure:



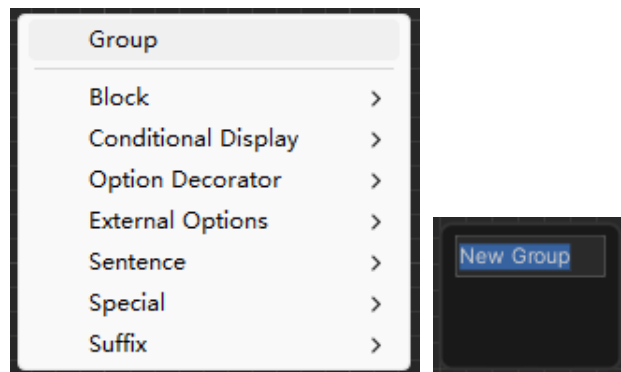


The type of dialogue group is DialogueGroup, its members are as follows:

Members For Runtime	Explanation
Field: _name	Name of the group
Field: _nodes	Nodes in the group
Field: _position	Position of the group
Constructor: DialogueGroup(string, Vector2)	Constructor that passes in group name and position

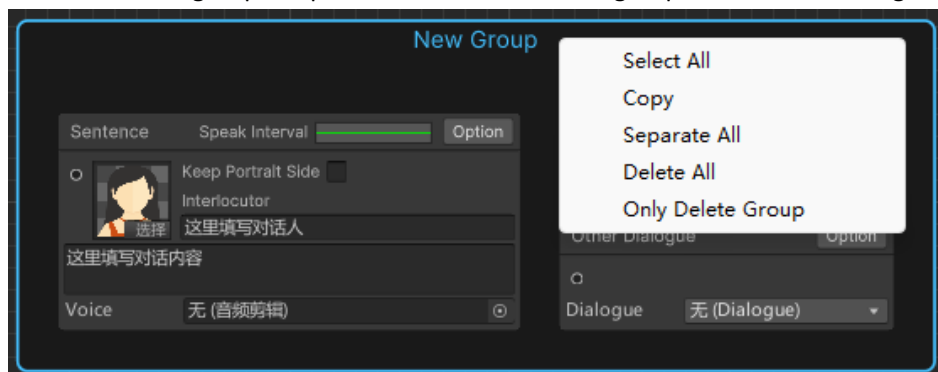
How to use:

Right-click on the blank space in the editor dialogue view to create a group. At this time, you can name the group, as shown in the figure:



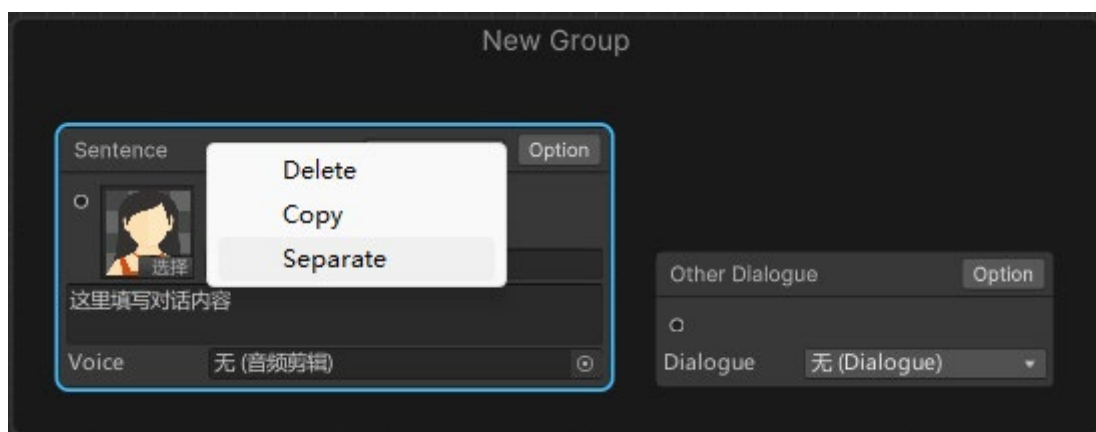
If you forget to name it, double-click the header of the group to rename it.

Right-click on the header of the group to operate the contents of the group, as shown in the figure:



Among them, "Select All" can select all nodes in the group, "Copy" can copy the entire group, "Separate All" will ungroup all nodes in the group, and "Delete all" will delete the group together with all nodes in the group, "Only Delete Group" means to keep the nodes in the group, but delete the group.

Right-click the node in the group to separate the node from the group, as shown in the figure:



6.5 Dialogue Manager

The dialogue manager is used to manage dialogue data, such as add and delete data, save and load, it's a static class. The type of dialogue manager is DialogueManager, its members are as follows:

Members For Runtime	Explanation
Static Field: data	All data of dialogues that have been conducted once
Static Method: DialogueData GetOrCreateData (EntryNode)	Find dialogue data according to the given root node, if there is no one then create one
Static Method: void RemoveData(EntryNode)	Remove the dialogue data of the given root node, which can be used for functions such as answering dialogues, resetting the answering progress after giving up answering, etc.
Static Method: void RemoveData(string)	Remove the dialogue data related to the given root node ID, the usage method is like previous one
Static Method: void SaveData(SaveData)	Write the dialogue data to save to the save data
Static Method: void LoadData(SaveData)	Load dialogue data from the save data and override current one

6.6 Dialogue Handler

The dialogue handler is used to handle dialogue logic, it's the core of dialogue system and used by dialogue window. You need to pass in some callback to make effect of its features, and in this plugin those callbacks are collected into a class called "dialogue handler callbacks", so that we can initialize the dialogue handler conveniently.

The type of dialogue manager is DialogueManager, its members are as follows:

The type of dialogue handler callbacks is DialogueHandlerCallbacks, its members are as follows:

运行时成员	说明
Field: setName	A callback to set interlocutor name text
Field: setContent	A callback to set dialogue content text
Field: preprocessText	A callback to preprocess text, it processes the text slightly, such as only translate it but not handling keywords inside the text
Field: processText	A callback to process text, which can further process the text slightly processed by the previous callback

Field: setPortrait	A callback to set interlocutor portrait
Field: setPortraitDark	A callback to darken portrait
Field: playVoice	A callback to play voice
Field: stopVoice	A callback to stop playing voice
Field: startCoroutine	A callback to start coroutine
Field: stopCoroutine	A callback to stop coroutine
Field: onHandleNode	A callback gets called after handling node
Field: onEndWriting	A callback gets called when typing animation ends
Field: onReachLastSentence	A callback gets called when reach end node
Field: refreshNextAction	A callback to refresh next action
Field: refreshOptions	A callback to refresh option list
Field: sendMessage	A callback to send message to player
Field: getSkipDelay	A callback to get skip delay, it's used to indicate how long to wait before you can skip typing animation. The delay is 0.5s if this callback is null

The type of dialogue handler is DialogueHandler, its members are as follows:

Members For Runtime	Explanation
Field: callbacks	Callbacks collection
Field: window	The dialogue window that this handler belongs to it
Property: Interlocutor	The interlocutor who opens the window, that is, the one who trigger the conversation
Property: Home	The first met root node when open dialogue window
Property: CurrentEntry	Current root node
Property: CurrentEntryData	Data of current root node
Field: currentNode	Current node
Property: CurrentNodeData	Data of current node
Field: continueNodes	Nodes to continue conversation when current one is end
Field: manualNodes	Is used to avoid endless loop when handle node customarily
Property: IsDoingManual	Are we handling node with custom operation now?
Field: isUsingLeftPortrait	Whether the left portrait is being used, it's available when enable portrait function
Field: hasLeftPortrait	Is portrait on the left null? available when enable portrait
Field: hasRightPortrait	Is portrait on the right null? available when enable portrait
Field: currentInterlocutor	The interlocutor of current sentence, available when enable portrait
Field: portraitInterlocutor	The interlocutor string of current portrait side, it's available when enable portrait function
Field: targetContent	The target content text to play typing animation
Field: writeCoroutine	The coroutine to play typing animation
Field: playtime	The playback time of typing animation
Property: IsPlaying	Is the typing animation playing
Field: waitCoroutine	The coroutine to wait until condition is met
Field: resumeOnWaitOver	Whether to refresh interaction buttons when the wait is over

Property: IsWaiting	Is that waiting now
Constructor: DialogueHandler (DialogueHandlerCallbacks, object)	Constructor that passes in callbacks collection and dialogue window
Method: void StartWith(IInterlocutor)	Start a conversation according to the given interlocutor
Method: void StartWith(Dialogue)	Start a conversation according to the given dialogue asset
Method: void StartWith(string, string)	Start a conversation according to the given interlocutor and speech content
Method: void StartWith(EntryNode)	Start a conversation according to the given root node
Method: void ContinueWith(DialogueNode)	Continue with a specified node that belongs to this dialogue
Method: void HandleNode(DialogueNode)	Handle the effect of given node in UI aspect
Method: bool HandleSpecialNodes (ref DialogueNode)	Handle special nodes, such as skipping decorator nodes, condition nodes, etc.
Method: void HandleLastSentence()	Handle the last sentence of current dialogue, such as the node that its "ExitHere" set to True or has no next node
Method: void HandleInteraction()	Handle interaction buttons of current node
Method: void RefreshInteraction()	Refresh interaction buttons of current node, such as setting "nextClicked", refreshing options, etc.
Method: void ResetInteraction()	Reset interaction buttons, such as clearing options
Method: void DisplaySentence(SentenceNode)	Display the speech content of sentence node
Method: IEnumerator Write(SentenceNode)	Play typing animation
Method: void Skip()	Skip typing animation
Method: void EndWriting()	Wind up typing animation
Method: void StopWriting()	Stop typing animation
Method: Coroutine StartCoroutine(IEnumerator)	Start coroutine
Method: void StopCoroutine(Coroutine)	Stop coroutine
Method: string PreprocessText(string)	Preprocess text
Method: string ProcessText(string)	Further process text after preprocessing it
Method: void RefreshNextAction(Action)	Set the next action
Method: void HandleOptions (IList<DialogueOption>)	Handle the passed in options and generate related buttons
Method: void RefreshOptions (List<OptionCallback>)	Refresh option list
Method: void RefreshPortrait(SentenceNode)	Refresh and display portrait of the given sentence, effective when enable portrait
Method: void ResetPortraits()	Reset and hide portraits, effective when enable portrait
Method: void SetPortraitDark(Image, bool)	Set portrait to dark color, effective when enable portrait
Method: void PlayVoice(SentenceNode)	Play voice clip of sentence, effective when enable voice
Method: void StopVoice()	Stop current voice, effective when enable voice
Method: void BackHome()	Return to the home page
Method: void BackHomeImmediate()	Return to the home page without typing animation
Method: void DoOption(DialogueOption)	Perform the function of option's connected node
Method: void DoManual(IManualNode)	Perform the custom action of the given node
Method: void InvokeEvent(DialogueEvent)	Trigger the given event

Method: void WaitUntil(Func<bool>)	Hide interaction buttons until the given condition is met
Method: void PushInteraction(DialogueNode)	Push the node to a stack, when current dialogue ends, it will display options of the top node of that stack as the options of the last sentence. It's usually used by other-dialogue node
Method: DialogueNode PopInteraction()	Pop the node previous method pushed
Method: void Init()	Initialize the dialogue handler

6.7 Interlocutor Interface

If you want some types of interactive object can trigger conversation, you need to inherit the interlocutor interface. The interlocutor interface is `IInterlocutor`, inherit from [IInteractive](#), its member is as follows:

Members For Runtime	Explanation
Property: Dialogue	The dialogue asset which is used to start the conversation

How to use:

Make the type of interactive object that can trigger conversation inherit this interface, as shown in figure:

```
[DisallowMultipleComponent, KeywordsSet("对话人")]
Unity 脚本 (3 个资产引用) | 3 个引用
public class Interlocutor : Character, IInterlocutor, IKeyword
{
    public Gender gender;

    2 个引用
    public bool IsInteractive...

    [field: SerializeField, SpriteSelector]
    2 个引用
    public Sprite Icon { get; private set; }

    [field: SerializeField]
    2 个引用
    public Dialogue Dialogue { get; private set; }

    5 个引用
    bool IInteractive.Interactable { get; set; }

    6 个引用
    string IKeyword.IDPrefix => "NPC";
}
```

Part II: Condition System

Section 1: Concepts

The condition system attached to this plugin is a general-use system that can be used in many cases. It is composed of condition and condition group. The condition group is composed of condition calculation formula and condition list. The calculation formula can make the conditions in the condition list mixed judgments are performed according to the given expression, rather than purely judged one by one.

Section 2: Components

2.1 Condition

The type of condition is Condition, its members are as follows:

Members For Runtime	Explanation
Property: IsValid	Indicates whether contents of this condition are valid
Method: bool IsMet()	Indicates whether this condition is met
Static Method: string GetGroup(Type)	Get group name of specified condition type
Static Method: string GetName(Type)	Get name of specified group
Operator: bool(Condition)	Used to quickly check if the condition is null

How to expand:

A sample code for level condition is given below:

[Serializable]

```
public abstract class LevelCondition : Condition
```

```
{
```

```
    [field: SerializeField, Min(0)]
```

```
    public int Level { get; private set; } = 1;
```

```
    public override bool IsValid => Level >= 0;
```

```
}
```

```
[Serializable, Name("Level Equals To"), Group("Level")]
```

```
public sealed class LevelEqualsTo : LevelCondition
```

```
{
```

```
    public override bool IsMet()
```

```
    {
```

```
        return PlayerManager.player.level == Level;
```

```
    }
```

```
}
```

```
[Serializable, Name("Level Higher Than"), Group("Level")]
```

```
public sealed class LevelHigherThan : LevelCondition
```

```
{
```

```
    public override bool IsMet()
```

```
    {
```

```

        return PlayerManager.player.level > Level;
    }
}
[Serializable, Name("Level Lower Than"), Group("Level")]
public sealed class LevelLowerThan : LevelCondition
{
    public override bool IsMet()
    {
        return PlayerManager.player.level < Level;
    }
}

```

2.2 Condition Group

The type of condition group is ConditionGroup, its members are as follows:

Members For Runtime	Explanation
Field: formula	Conditional calculation formula
Property: Formula	Related read only property of field "formula"
Field: conditions	Conditions list
Property: Conditions	Related read only property of field "conditions"
Property: IsValid	Indicates whether conditions in the list are valid
Method: bool IsMet()	Indicates whether the conditions are met
Operator: bool(ConditionGroup)	Used to quickly check if the condition group is null

For the formula of condition calculation, this plugin stipulates it as follows:

1. The operands are indices of the conditions;
2. The operator can be "("、")"、"|" (or)、"&" (and)、"!" (not);
3. Will not handle invalid input, please fill in normatively;
4. Example: formula "(0 | 1) & !2" indicates that we need to meet condition at index 0 or 1, and not meet condition at index 2;
5. If the formula is empty, we will calculate the conditions with "and" operation mutually.

Part III: Keyword System

Section 1: Concepts

The keyword system attached to this plugin can insert an ID string in a given format into the string, and then replace the ID string with the name of the object corresponding to the ID. If there is an interlocutor with the ID of "NPC0001" and the name of "Xiao Ming", then its ID string is "[NPC]NPC0001". When the string "I am {[NPC]NPC0001}" is inputted, it will be processed into "I am Xiao Ming".

Section 2: Components

2.1 Keyword Interface

Inherit this interface if you want an object of specified type can be a keyword object.

The keyword interface is IKeyword, its members are as follows:

Members For Runtime	Explanation
Property: ID	ID of the keyword object
Property: IDPrefix	The prefix to put in front of ID, is used to categorize keyword. Such as "[NPC]NPC001", the prefix is "NPC" that is bracketed
Property: Name	Name of the keyword object
Property: Color	Rich text color when its name displays on UI text

Its Editor-Use member is as follows:

Members For Editor	Explanation
Property: Group	A group string to group keyword objects inside their class

How to use:

Make the type of keyword object inherit this interface, then implement the members of this interface, as shown in the figure:

```
[DisallowMultipleComponent, KeywordsSet("对话人")]
@ Unity 脚本 (3 个资产引用) | 3 个引用
public class Interlocutor : Character, IInterlocutor, IKeyword
{
    public Gender gender;

    2 个引用
    public bool IsInteractive...

    [field: SerializeField, SpriteSelector]
    2 个引用
    public Sprite Icon { get; private set; }

    [field: SerializeField]
    2 个引用
    public Dialogue Dialogue { get; private set; }

    5 个引用
    bool IInteractive.Interactable { get; set; }
```

2.2 Keyword Type

The keyword type is the keyword manager, it's a static class, too. It's used to replace the ID string in the given string with the name of the object corresponding to the ID. What to note is, in order to ensure the accuracy of keyword conversion, all keyword objects will be collected before each conversion, and the performance will be affected. Therefore, try not to convert keywords in the update method such as "void Update()".

The type of keyword manager is Keyword, its members are as follows:

Members For Runtime	Explanation
Static Method: string Translate(string, bool)	Convert the given ID into name corresponding to it, you are allowed to use rich text to color the name string
Static Method: string Translate(string, bool, Dictionary<string, Dictionary<string, IKeyword>>)	Convert the given ID into name corresponding to it with the given keyword object dictionary, you are allowed to use rich text to color the name string
Static Method: Dictionary<string, Dictionary<string, IKeyword>> CollectKeywords()	Collect all keyword objects before the conversion
Static Method: string HandleKeywords(string, bool)	Convert the IDs in the given string into names corresponding to them, you are allowed to use rich text to color the name string
Static Method: IEnumerable<KeyValuePair<string, string>> ExtractKeyWords(string)	Extract the ID strings in the given string, convert them to the corresponding names, and return them in the form of a list of key value pairs
Static Method: string Generate(IKeyword)	Generate the ID string of given keyword object
Static Method: void SetAsKeywordsField(TextField)	Is the given string a keyword ID string

The Keyword type also has an embedded static class "Editor" for editor use. Its members are as follows:

Members For Editor	Explanation
Static Method: string Translate(string)	Convert the given ID into name corresponding to it
Static Method: string Translate(string, Dictionary<string, Dictionary<string, IKeyword>>)	Convert the given ID into name corresponding to it with the given keyword object dictionary
Static Method: Dictionary<string, Dictionary<string, IKeyword>> CollectKeywords()	Collect all keyword objects before the conversion
Static Method: string HandleKeywords(string)	Convert the IDs in the given string into names corresponding to them
Static Method: IEnumerable<KeyValuePair<string, string>> ExtractKeyWords(string)	Extract the ID strings in the given string, convert them to the corresponding names, and return them in the form of a list of key value pairs
Static Method: void OpenKeywordsSelection(Vector2, Action<string>)	Open the keyword selection window, and pass in the ID string of the selected keyword object to the passed in callback and invoke it
Static Method: void SetAsKeywordsField(TextField)	Make a TextField can be right-clicked to insert keyword ID string

Section 3: Attributes

3.1 Keyword Set Attribute

The keyword set attribute is an attribute used by editor, it's used to root-group the keyword objects in keyword selection window.

The type of keyword set attribute is `KeywordsSetAttribute`, inherit from `Attribute`, its member is as follows:

Members For Runtime	Explanation
Field: name	Name of the set

How to use:

Add it to the type of keyword object, as shown in the figure:

```
[DisallowMultipleComponent, KeywordsSet("对话人")]
Unity 脚本 (3 个资产引用) | 3 个引用
public class Interlocutor : Character, IInterlocutor, IKeyword
{
    public Gender gender;
```

3.2 Get Keywords Method Attribute

Get keywords method attribute is an attribute use by editor, too. Add this attribute to parameter-less static editor methods that is used to get keyword objects. You should note that method must returns `IEnumerable<IKeyword>`.

The type of get keywords method attribute is `GetKeywordsMethodAttribute`, inherit from `Attribute`.

How to use:

Add it to the static editor method, as shown in the figure:

```
[RuntimeGetKeywordsMethod, GetKeywordsMethod]

public static IEnumerable<Interlocutor> GetInterlocutors()
{
    return FindObjectsOfType<Interlocutor>(true);
}
```

3.3 Runtime Get Keywords Method Attribute

Runtime get keywords method attribute is an attribute use by runtime code. Add this attribute to parameter-less static runtime methods that is used to get keyword objects, then you can collect keyword objects when the game is running. You should note that method must returns `IEnumerable<IKeyword>`.

The type of runtime get keywords method attribute is `RuntimeGetKeywordsMethodAttribute`, inherit from `Attribute`.

How to use:

Add it to the static runtime method, as shown in the figure:

```
[RuntimeGetKeywordsMethod, GetKeywordsMethod]

public static IEnumerable<Interlocutor> GetInterlocutors()
{
    return FindObjectsOfType<Interlocutor>(true);
}
```

Part IV: Multilingual Text System

Section 1: Concepts

The multilingual text system attached to this plugin is used to display the text in the game in different languages, it consists of translation mappings, translation sets (that is, a collection of translation mappings), and localization files. When translating, first use an index called “selector” to filter the translation sets to be used in the localization file, and then use the translation key to find the corresponding translation mapping in these translation sets. After finding it, go further and take out the translation content (that is, string written in specific language) of current language, and here, this translation trip is completed. For translation performance, the multilingual text system is not updated in real time. The translation cache is only refreshed after the game is started or the language is switched. This cache is a dictionary keyed by selectors. The dictionary item is a sub-dictionary. This sub-dictionary is the real translation mapping table, which consolidates all translation mappings in all translation sets corresponding to the selector. Therefore, when the game is running, if the content of the translation set changes, it will not be reflected in the game in real time. The multilingual text system can also be used in the editor. For the editor, the translation method has changed: since the editor can obtain and refer to the translation set at will, the translation will directly use the translation content of the first language of the given translation set.

Section 2: Components

2.1 Translation Mapping

The translation mapping is a kind of key value pair, it uses specific key to index the translation content, and translation content is subdivided into translations for each language. Its structure can be expressed as:

Key	Content of Language 1	Content of Language N
“索引 01”	“Index 01”	“インデックス 01”

Among them, the source text of the default language (that is, the developer's native language) can be directly used as the key, which is more convenient, and there is no need to repeatedly search for the source text corresponding to a key. Sometimes the keyed item is missing, so it is impossible to know source text.

For example, if the display text on a button is “OK”, then “OK” is the so-called source text, if you don't use the source text as the key, then assume its key in the translation mapping is “BUTTON_01”, the first language's translation is “OK”, and the second language is “确定”, now assuming that this mapping is missing, the button will directly display the key “BUTTON_01”, but we don't know the real meaning of “BUTTON_01”, so we cannot supplement the translation. This plugin uses the source text as the key. If you don't want this, you can set the “[LanguageIndexOffset](#)” property of [Language](#) type to 0 to cancel it.

If there are cases where the key is the same but the translation is different, you can add the prefix “[DUP+number]” to the repeated key, for example, the repeated “index 001” can be changed to “[DUP02]index 001”. This prefix is automatically stripped when the key is returned directly because no translation was found.

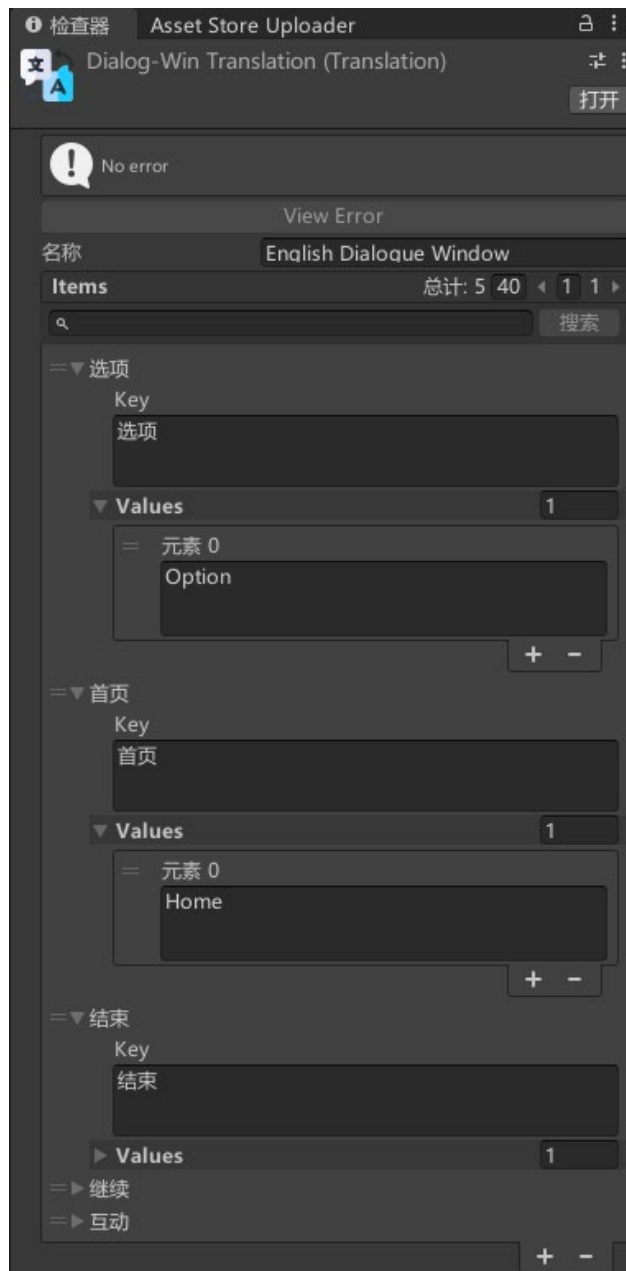
The type of translation mapping is TranslationItem, its members are as follows:

Members For Runtime	Explanation
Property: Key	Key of the translation mapping
Field: values	Translation strings of each language
Property: Values	Related read only property of field “values”

Constructor: TranslationItem()	Parameter-less constructor for serialization
Constructor: TranslationItem (string, params string[])	Constructor that passes in its key and translation strings of multiple languages

2.2 Translation Set

Translation set is a kind of ScriptableObject asset, it's a collection of translation mappings, or you can say it's a table. You can use only one translation set globally, or use multiple set for each in-game system, so that you can divide the work when processing translations. Its inspector view in the editor is shown in the figure:



Type of translation set is TranslationSet, inherit from ScriptableObject, its members are as follows:

Members For Runtime	Explanation
Field: items	List of translation mappings
Property: Items	Related read only property of field "items"

Method: Dictionary<string, string> AsDictionary(int)	Extract the translation strings of the specified language and generate them into a dictionary
Method: string Tr(int, string)	Translate in translation string of specified language
Method: string Tr(string)	Translate in translation string of the first language

Its Editor-Use member is as follows:

Members For Editor	Explanation
Field: _name	Name of the translation set, generally used for noting

The Translation type also has an embedded static class “Editor” for editor use. Its members are as follows:

Members For Editor	Explanation
Static Method: void SetName (Translation, string)	Set the name of the given translation set
Static Method: void SetItems (Translation, TranslationItem[])	Set the translation mapping list of the given translation set

2.3 Localization File

The localization file is a further integration of translation sets, it’s a kind of ScriptableObject asset, it categorizes the translation sets with strings call “selector”. However, there is not a one-to-one relationship between selectors and translation sets, a selector can be used by multiple different translation sets, and a translation set can also be used by multiple different selectors. Its inspector view in the editor is shown in the figure:



The type of localization file is Localization, inherit from ScriptableObject, its members are as follows:

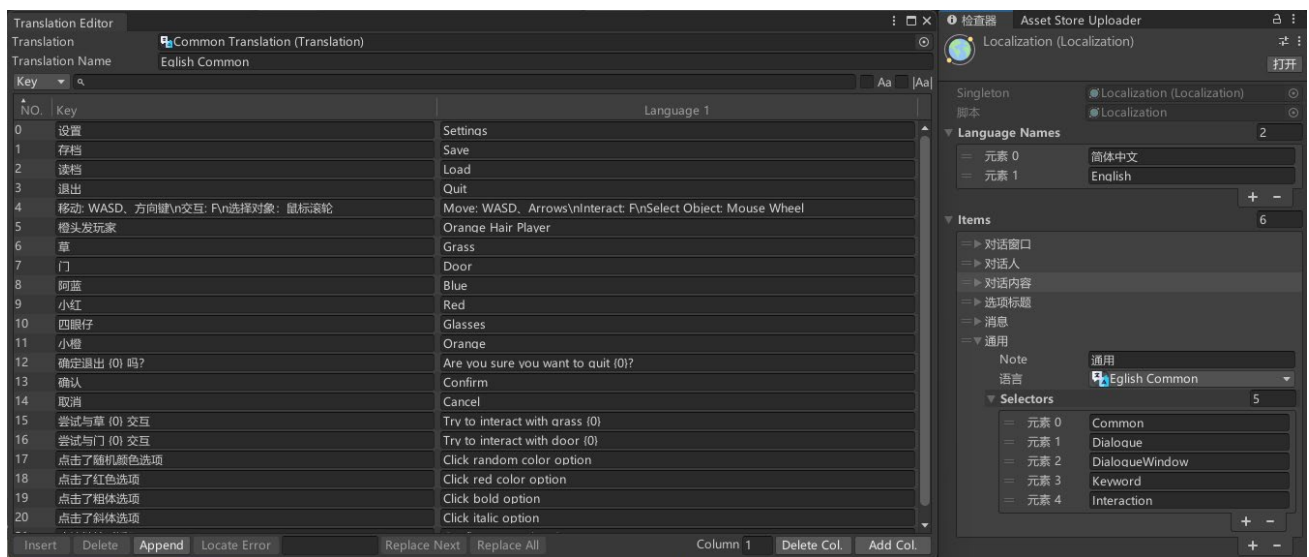
Members For Runtime	Explanation
Field: languageNames	List of language names
Property: LanguageNames	Related read only property of field “languageNames”
Field: items	List of the translation set and selectors pair, type of its item is LocalizationItem
Property: Items	Related read only property of field “items”
Property: Instance	The singleton instance of Localization file
Method: Dictionary<string, List<Dictionary<string, string>>> AsDictionary(int)	Generate a translation cache base on this localization
Method: List<Dictionary<string, string>> FindDictionaries(string, int)	Find a translation dictionary collection according to the given selector

Its Editor-Use member is as follows:

Members For Editor	Explanation
Static Method: void Create()	Try to create a Localization type instance

How to use:

First add a new item to “items”, then set the translation set of that item, and then add the selectors to key the translation set, that is to say, which selectors that are used for the translation will select this translation set when translating, as shown in the figure:



2.4 Localization Group

The localization group is a translation set and selectors pair.

The type of localization group is LocalizationItem, its members are as follows:

Members For Runtime	Explanation
Property: Language	The translation set that this group is using
Field: selectors	List of selectors
Property: Selectors	Related read only property of field “selectors”

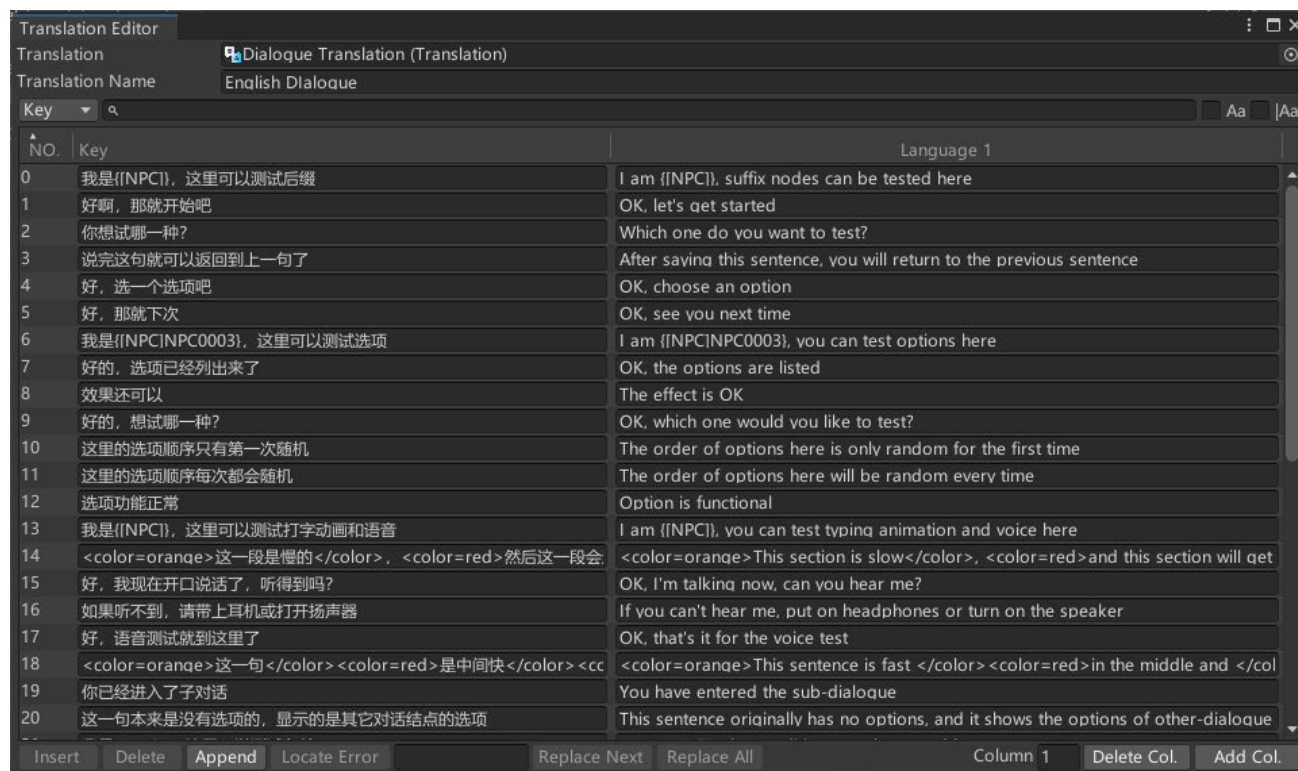
Its Editor-Use member is as follows:

Members For Editor	Explanation
Field: note	The note of this group

Section 3: Edit Language Set

3.1 Translation Set Editor

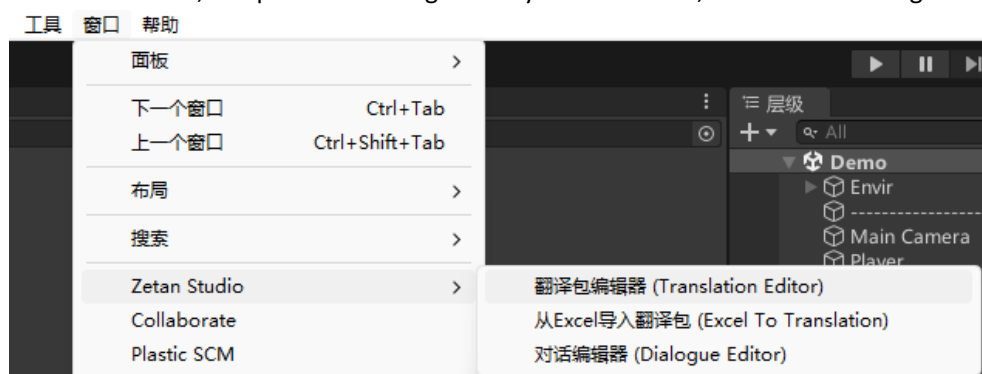
The translation set editor can edit the mappings of translation set like an Excel table, as shown in the figure:



Among them, the “Insert” button will insert a new row under the currently selected row, and the “Append” button will add a new row at the end; the drop-down menu on the left side of the search box can change the column to be searched. When searching, you are not allowed to add or delete rows; the “Delete Col.” button in the lower right corner is used to delete the column indicated in front of it. You should note that the column number starts from 0, excluding the serial number column. For example, the column number 1 at this time represents column “Language 1”; You can drag the list items to a new position to move them; click the header of the column “NO.” to switch between ascending and descending order. When sorting in descending order, you are not allowed to drag.

How to open the translation set editor:

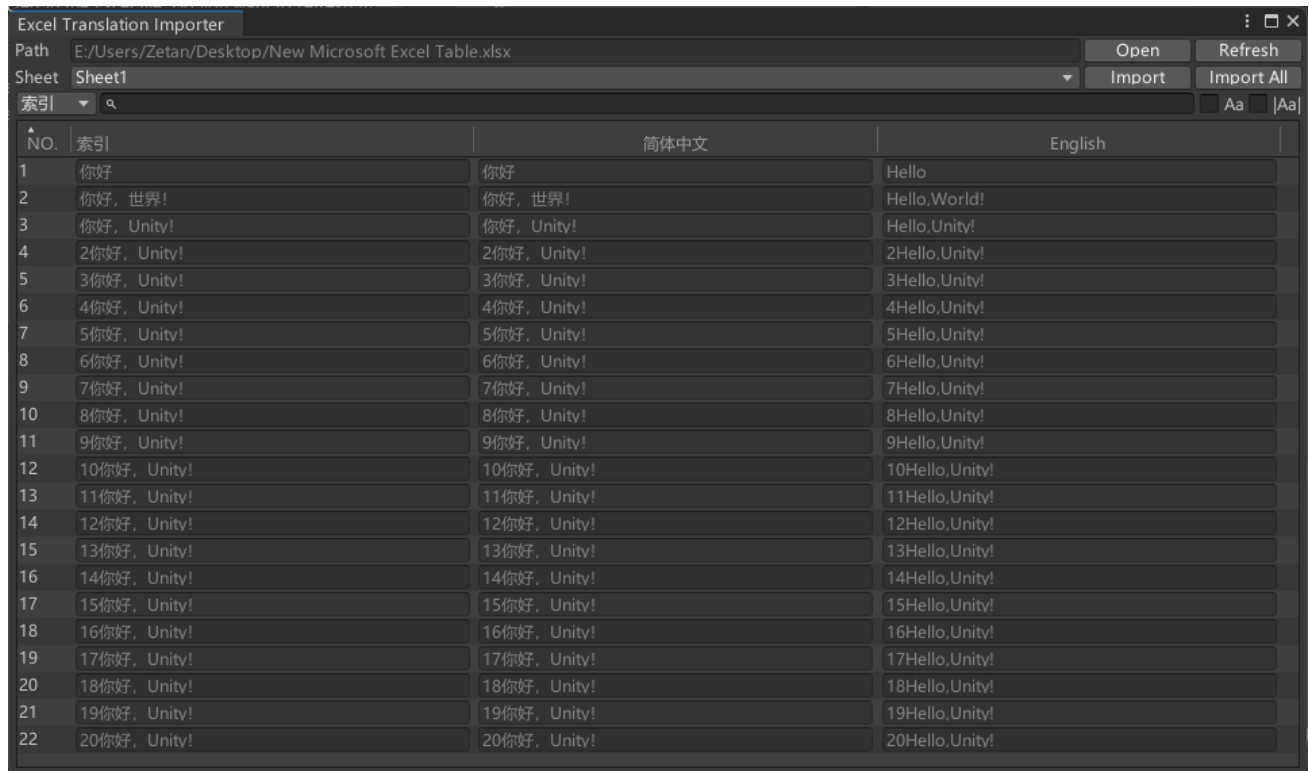
Double-click the translation set asset file, or click the “Open” button in the higher right corner of its inspector to open the translation set editor, or open it according to Unity toolbar menu, as shown in the figure:



You are allowed to open multiple translation set editor at the same time.

3.2 Edit Translation Mappings with Excel

You are allowed to import translation sets from Excel file, and is more convenient to edit translation mappings in Excel. But it's not supported to export translation set to Excel file for now, to use the above translation set editor or use Excel to edit the translation, it depends on you. When importing an Excel file, the content inside will be previewed, as shown in the figure:



NO.	索引	简体中文	English
1	你好	你好	Hello
2	你好, 世界!	你好, 世界!	Hello, World!
3	你好, Unity!	你好, Unity!	Hello, Unity!
4	2你好, Unity!	2你好, Unity!	2Hello, Unity!
5	3你好, Unity!	3你好, Unity!	3Hello, Unity!
6	4你好, Unity!	4你好, Unity!	4Hello, Unity!
7	5你好, Unity!	5你好, Unity!	5Hello, Unity!
8	6你好, Unity!	6你好, Unity!	6Hello, Unity!
9	7你好, Unity!	7你好, Unity!	7Hello, Unity!
10	8你好, Unity!	8你好, Unity!	8Hello, Unity!
11	9你好, Unity!	9你好, Unity!	9Hello, Unity!
12	10你好, Unity!	10你好, Unity!	10Hello, Unity!
13	11你好, Unity!	11你好, Unity!	11Hello, Unity!
14	12你好, Unity!	12你好, Unity!	12Hello, Unity!
15	13你好, Unity!	13你好, Unity!	13Hello, Unity!
16	14你好, Unity!	14你好, Unity!	14Hello, Unity!
17	15你好, Unity!	15你好, Unity!	15Hello, Unity!
18	16你好, Unity!	16你好, Unity!	16Hello, Unity!
19	17你好, Unity!	17你好, Unity!	17Hello, Unity!
20	18你好, Unity!	18你好, Unity!	18Hello, Unity!
21	19你好, Unity!	19你好, Unity!	19Hello, Unity!
22	20你好, Unity!	20你好, Unity!	20Hello, Unity!

Among them, one sheet corresponds to one translation set, the “Import” button will only import current sheet, and the “Import All” button will import all sheets in the table one by one.

How to open translation set Excel importer:

You need to open it by clicking the toolbar menu on the top of Unity, as shown in the figure:



The Excel sheets should have a header, as shown in the figure:

	A	B	C
1	索引	简体中文	English
2	你好	你好	Hello
3	你好, 世界!	你好, 世界!	Hello, World!
4	你好, Unity!	你好, Unity!	Hello, Unity!
5	从小到大	先插入指定位置	然后后面的依次后移一位
6	带娃哇额	打的滴滴	打撒大的滴滴

Section 5: Translator

Translator, as its name suggests, is a class that translates the key in the translation mapping into the translation content of the specified language. It is a static class that is initialized once after game is started or when switching languages, and it caches the translation dictionary of the specified language. Therefore, the translator is not real-time, even if the content of the translation set is modified, it will not be refreshed until the next language switch. The reason for this is that we generally do not add, delete and modify the translation set while the game is running. Moreover, adding, deleting and modifying translation sets can only be done in the editor mode.

The type of translator is Language, its members are as follows:

Members For Runtime	Explanation
Static Field: cache	Translation mappings cache
Static Field: languageIndex	Index of current language
Static Property: LanguageIndex	Related read only property of field "languageIndex"
Static Field: languageIndexOffset	The index offset when the translator indexes the " Values " of a mapping, that is the "LanguageIndex" should subtract
Static Property: LanguageIndexOffset	Related read only property of field "languageIndexOffset"
Static Event Action OnLanguageChanged	Call when "languageIndex" or "languageIndexOffset" is changed
Static Method: void Init()	Initialize the translator, generate new translation cache
Static Method: string Translate (string, List<Dictionary<string, string>>)	Internal translation method, is used to translate the given string according to the given translation dictionaries
Static Method: string Tr(string, string)	Seek and translate given string according to the dictionaries corresponding to the given selector, "Tr" is the abbreviation of "Translate"
Static Method: string Tr (string, string, params object[])	Translate in format, like method "string string.Format(string, string)"
Static Method: string[] TrM (string, string, params string[])	Multiple translation
Static Method: string[] TrM(string, string[])	Multiple translation, but allows to pass in string array
Static Method: string Tr(Translation, int, string)	Translate by the translation contents of the given language of the given translation set
Static Method: string Tr (Translation, int, string, params object[])	Refer to above
Static Method: string[] TrM (Translation, int, string, params string[])	Refer to above
Static Method: string[] TrM (Translation, int, string[])	Refer to above
Static Method: string Tr(Translation, string)	Translate by the translation contents of the first language of the given translation set
Static Method: string Tr (Translation, string, params object[])	Refer to above
Static Method: string[] TrM (Translation, string, params string[])	Refer to above

Static Method: string[] TrM (Translation, string[])	Refer to above
--	----------------

The translator type also has an abbreviated class called “L” directly. For example, when we scripting translation, using “L.Tr(“selector”, “key”)” is equivalent to “Language.Tr(“selector”, “key”)”, save code space.

Section 6: Multilingual Text Component

6.1 Static Multilingual Text

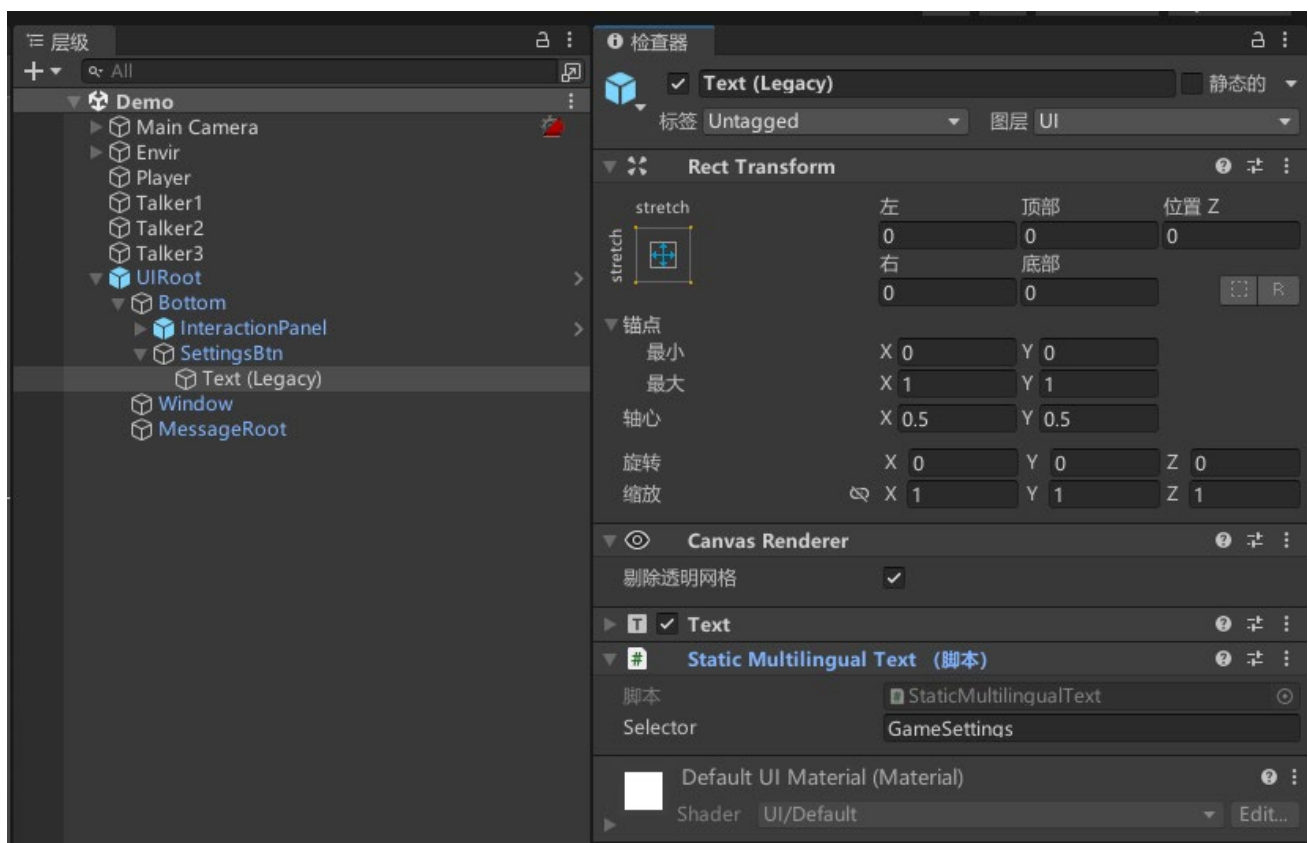
Static text refers to the static text on the UI that will not be modified by other codes, the title "Backpack" of the backpack window is static for example, and the dynamic text is like the speech content text of the dialogue window, which will be frequently modified. The static multilingual text component translates the static original text by specified selector, and it requires the cooperation of the “Text” component.

The type of static multilingual text is StaticMultilingualText, inherit from MonoBehaviour, its members are as follows:

Members For Runtime	Explanation
Field: selector	Selector to select translation sets
Field: component	Reference of “Text” component
Field: original	The original text
Method: void Awake()	Called by Unity, initialize text, listen language switch event
Method: void SetText(string)	Set and translate a new original text
Method: void SetSelector(string)	Set a new selector and translate original text

How to use:

Add static multilingual text component to text object and set its selector, as shown in figure:



6.2 Multilingual Text

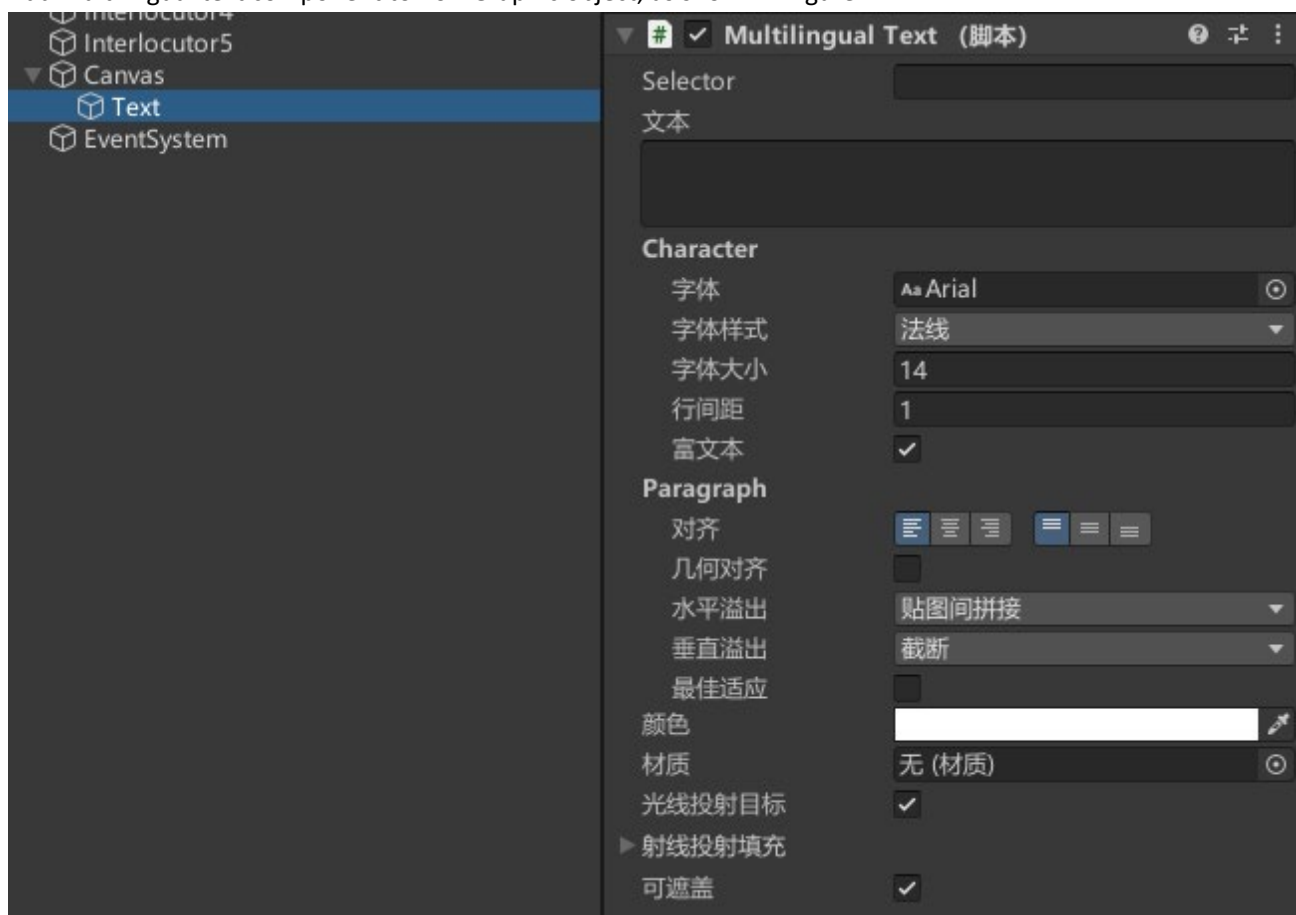
The multilingual text component is similar to the static multilingual text above, but it directly extends the “Text” component and can be used instead of it. You should note that multilingual text is only applicable to situations that require automatic translation when setting the original text, and when the original text is already a translated text, there is no need to use this component, which is multiple translations.

The type of multilingual text is MultilingualText, inherit from Text, its members are as follows:

Members For Runtime	Explanation
Field: m_selector	Selector to select translation sets
Property: selector	Related read only property of field “m_selector”
Property: text	Override the “Text.text”, translate automatically when writing into it
Field: cache	Cache of target text
Field: argsCache	Cache of the arguments when set text in format
Method: void Awake()	Called by Unity, initialize text, listen language switch event
Method: void SetText(string)	Set its text
Method: void SetText(string, params object[])	Set its text in format

How to use:

Add multilingual text component to non-Graphic object, as shown in figure:



Part V: Save System

Section 1: Concepts

Game-saving is to convert the important data of objects that need to be saved in the game into serializable data and save it locally, and then load and restore these data to the specified objects when needed. A save system consists of save data and a save manager.

Section 2: Components

2.1 Save Data

Save data is a sub-class of generic data type, it is nothing more than pre-stored some necessary data.

The type of save data is `SaveData`, inherit from `GenericData`, its member is as follows:

Members For Runtime	Explanation
Constructor: <code>SaveData(string)</code>	Constructor that passes in data version

2.2 Save Manager

Save manager, as its name suggests, is used to save and load the game progress, it's a static class. You can select the slot to save the data if you need.

The type of save manager is `SaveManager`, its members are as follows:

Members For Runtime	Explanation
Const: <code>defaultDataName</code>	Default data file name
Static Field: <code>dataName</code>	Custom data file name
Static Property: <code>DataName</code>	Actual data file name, use default when custom one is empty
Const: <code>defaultEncryptionKey</code>	Default data encryption key
Static Field: <code>encryptionKey</code>	Custom data encryption key
Static Property: <code>EncryptionKey</code>	Actual data encryption key, use default when custom one is empty or its length not match (16 or 32 chars)
Static Property: <code>IsSaving</code>	Is that currently saving data
Static Property: <code>IsLoading</code>	Is that currently loading data
Static Event: <code>Action OnSaveCompletely</code>	Get called when data saving is complete
Static Event: <code>Action OnLoadCompletely</code>	Get called when data loading is complete
Static Method: <code>bool Save(int)</code>	Save data into the file that the specified slot specifies
Static Method: <code>bool Load(int)</code>	Load data from the file that the specified slot specifies

Section 3: Attributes

3.1 Save Method Attribute

Add the save method attribute to specified static method, then it will get called for writing game progress to the save data when saving. That method must have only one `SaveData` type parameter.

The type of save method attribute is SaveMethodAttribute, inherit from Attribute, its members are as follows:

Members For Runtime	Explanation
Field: priority	Invoke priority
Constructor: SaveMethodAttribute(int)	Constructor that passes in priority
Static Method: void SaveAll(SaveData)	Call all methods with this attribute in order of priority, make them write the progress data of their respective systems to the save data, and this method is called by the save manager

3.2 Load Method Attribute

Add the load method attribute to specified static method, then it will get called for loading game progress from the save data when saving. That method must have only one SaveData type parameter.

The type of load method attribute is LoadMethodAttribute, inherit from Attribute, its members are as follows:

Members For Runtime	Explanation
Field: priority	Invoke priority
Constructor: LoadMethodAttribute (int)	Constructor that passes in priority
Static Method: void LoadAll(SaveData)	Call all methods with this attribute in order of priority, make them load the progress data of their respective systems from the save data, and this method is called by the save manager

Part VI: Game UI

Section 1: Concepts

The window system attached to this plugin is type-based. It manages the generation of the UI in the game based on prefabs, and consists of various window classes and window managers. Because it is based on type, the game window has a one-to-one relationship with its window class, and a window class corresponds to only one window GameObject, so that the window manager can correctly open, close or find a given type of game window.

Section 2: Window System

2.1 Window Base Type

The window base type, as the name suggests, is the base type of type of the game window. It uses the alpha and blocksRaycasts properties of the “CanvasGroup” component to open and close the game window. Set alpha to 1, blocksRaycasts to True to open, and set alpha to 0, blocksRaycasts to False to close.

The window base type is Window, inherit from MonoBehaviour, [IFadeAble](#), its members are as follows:

Members For Runtime	Explanation
const WindowStateChanged	Global message type, is used to post and listen messages
Field: animated	Whether to enable the fade in/out animation, it is True by default
Field: duration	The duration of fading animation, default is 0.05s
Field: content	The “CanvasGroup” component to open/close window
Field: closeButton	The window close button of this window, is optional
Field: windowCanvas	The “Canvas” component to arrange the window
Event: OnClosed	The window close event, only trigger once on next closing for each registration
Property: HideOnAwake	Is the window not displayed on awake, default is True
Property: SortingOrder	A write only property to set display order of the window
Property: IsOpen	Is the window opened
Property: IsHidden	Is the window hidden
Property: IFadeAble.MonoBehaviour	Refer the window itself, is used by IFadeAble interface
Property: IFadeAble.FadeTarget	Refer field “content”, is used by IFadeAble interface
Property: IFadeAble.FadeCoroutine	This is used to cache the coroutine of fade in/out animation, is used by IFadeAble interface
Property: LanguageSelector	The “selector” for the language system
Method: bool Open(params object[])	Open the window
Method: bool Close(params object[])	Close the window
Method: bool Hide(bool, params object[])	Hide or unhide the window, only works when it’s opened
Method: bool OnOpen(params object[])	A method for overriding, write your open logic code here
Method: void OnCompletelyOpened()	Get called when the window is faded in completely, and get called immediately if the animation is not enabled

Method: bool OnClose(params object[])	A method for overriding, write your closing logic code here
Method: void OnCompletelyClosed()	Get called when the window is faded out completely, and get called immediately if the animation is not enabled
Method: bool OnHide(bool, params object[])	A method for overriding, write your hiding logic code here
Method: void OnAwake()	A method for overriding, do nothing by default
Method: void OnDestroy_()	A method for overriding, do nothing by default
Method: void RegisterNotification()	A method for overriding, usually used for refresh the window contents when receive messages, it will be called in method "void Awake()", do nothing by default
Method: void UnregisterNotification()	This is used to unregister message listening
Method: void Awake()	Called by Unity, initialize the window
Method: void OnDestroy()	Called by Unity, unregister message listening
Method: string Tr(string)	A shortcut for translation of multilingual text system
Method: string Tr(string, params object[])	A shortcut for format-translation of multilingual text system
Method: string[] TrM(string, params string[])	A shortcut for multiple-translation of multilingual text system
Method: string[] TrM(string[])	A shortcut for multiple-translation of multilingual text system
Static Method: static bool IsName<T>(string) where T : Window	Detect whether the given name is the name of type T
Static Method: static bool IsType<T>(Type) where T : Window	Detect whether the given type is the type T

Its Editor-Use members are as follows:

Members For Editor	Explanation
Method: void OnDrawGizmosSelected()	Gizmos a wire box to preview the bound of "content"
Static Method: void CreateUI()	Create an empty window
Static Method: bool CanCreateUI()	Detect whether the empty window can be created

2.2 Window Prefab Collection

Game windows can be generated on demand using Unity's prefab system, which requires a list to store these window prefabs. That list type is the window prefab collection type, which is a ScriptableObject asset. You can use this type to implement UI themes switching.

The type of window prefab collection is WindowPrefabs, inherit from ScriptableObject, its members are as follows:

Members For Runtime	Explanation
Field: windows	List of window prefabs
Property: Windows	Related read only property of field "windows"
Property: Instance	The singleton instance of window prefab collection
Method: Window FindWindowOfType(Type)	Find the prefab of the given window type

2.3 Window Manager

The window manager is a static class to manage the open, closing and hiding of the window.

The type of window manager is WindowManager, its members are as follows:

Members For Runtime	Explanation
Static Field: startSortingOrder	The start order number for sorting

Static Property: StartSortingOrder	Related read only property of field “startSortingOrder”, it will refresh the sorting orders of opened windows when writing
Static Field: windowsContainer	A parent transform to place windows
Static Property: WindowsContainer	Related read only property of field “windowsContainer”
Static Property: Count	Count of the opened windows
Static Event: Action OnCloseAll	An event gets triggered when closing all windows
Static Event: Action<bool> OnHideAll	An event gets triggered when hiding or unhiding all windows
Static Field: isContainerAutoCreated	Indicates whether the “windowsContainer” is created automatically, if it is, then it will be destroyed when setting a new container
Static Field: isHidingAll	Check if all windows are being hidden, and it is used to avoid recording wrong hiding state
Static Field: caches	Cached windows to reduce window lookup overhead
Static Field: openedWindows	Opened windows, it’s a stack in disguise
Static Field: hiddenStates	The hiding state of windows, it’s used by hiding methods
Static Method: Window OpenWindow (string, params object[])	Open a window according to its name, and returns it when it is opened successfully
Static Method: Window OpenWindow (Type, params object[])	Open a window according to its type, and returns it when it is opened successfully
Static Method: T OpenWindow<T> (params object[]) where T : Window	Open a window of type T, and returns it when it is opened successfully
Static Method: void OpenOrCloseWindow(string)	Open or close a window according to its name
Static Method: void OpenOrCloseWindow(Type)	Open or close a window according to its type
Static Method: void OpenOrCloseWindow<T>() where T : Window	Open or close window of type T
Static Method: Window OpenOrUnhideWindow(string)	Open or unhide a window according to its name
Static Method: Window OpenOrUnhideWindow(Type)	Open or unhide a window according to its type
Static Method: T OpenOrUnhideWindow<T>() where T : Window	Open or unhide a window of type T
Static Method: bool CloseWindow (string, params object[])	Close a window according to its name, the return value indicates if it’s closed successfully
Static Method: bool CloseWindow (Type, params object[])	Close a window according to its type, the return value indicates if it’s closed successfully
Static Method: bool CloseWindow<T> (params object[]) where T : Window	Close a window of type T, the return value indicates if it’s closed successfully
Static Method: void CloseTop()	Close the top visible opened window
Static Method: void CloseAll()	Close all opened windows
Static Method: void CloseAllExceptName (params string[])	Close all windows except those with the specified name
Static Method: void CloseAllExceptType (params Type[])	Close all windows except those with the specified type

Static Method: void CloseAllExcept (params Window[])	Close all windows except some windows
Static Method: void HideAll(bool)	Hide or unhide all windows
Static Method: void HideAllExceptName (bool, params string[])	Hide or unhide all windows except those with the specified name
Static Method: void HideAllExceptType (bool, params Type[])	Hide or unhide all windows except those with the specified type
Static Method: void HideAllExcept (bool, params Window[])	Hide or unhide all windows except some windows
Static Method: void RecordHiddenState(Window)	Record the hiding state of the given window, but when all windows are being hidden, do not record
Static Method: Window FindWindow(string)	Find a window according the given name
Static Method: Window FindWindow (string, bool)	Find a window according the given name, or create it optionally if no window is found
Static Method: Window FindWindow(Type)	Find a window according the given type
Static Method: Window FindWindow(Type, bool)	Similar to above one but by type
Static Method: T FindWindow<T>() where T : Window	Find a window of type T
Static Method: T FindWindow<T>(bool) where T : Window	Find a window of type T, or create it optionally if no window is found
Static Method: void Cache(Window)	Add the given window to the windows cache
Static Method: bool IsWindowOpen(string)	Check whether the window with the specified name is open
Static Method: bool IsWindowOpen(Type)	Check whether the window with the specified type is open
Static Method: bool IsWindowOpen<T>() where T : Window	Check whether the window of type T is open
Static Method: bool IsWindowOpen (string, out Window)	Check whether the window with the specified name is open, and output it
Static Method: bool IsWindowOpen (Type, out Window)	Check whether the window with the specified type is open, and output it
Static Method: bool IsWindowOpen<T>(out T) where T : Window	Check whether the window of type T is open, and output it
Static Method: bool IsWindowHidden(string)	Similar to open checking
Static Method: bool IsWindowHidden(Type)	Similar to open checking
Static Method: bool IsWindowHidden<T>() where T : Window	Similar to open checking
Static Method: bool IsWindowHidden (string, out Window)	Similar to open checking
Static Method: bool IsWindowHidden (Type, out Window)	Similar to open checking
Static Method: bool IsWindowHidden<T>(out T) where T : Window	Similar to open checking
Static Method: void Push(Window)	Push the window to stack, is only used by the open method of Window class

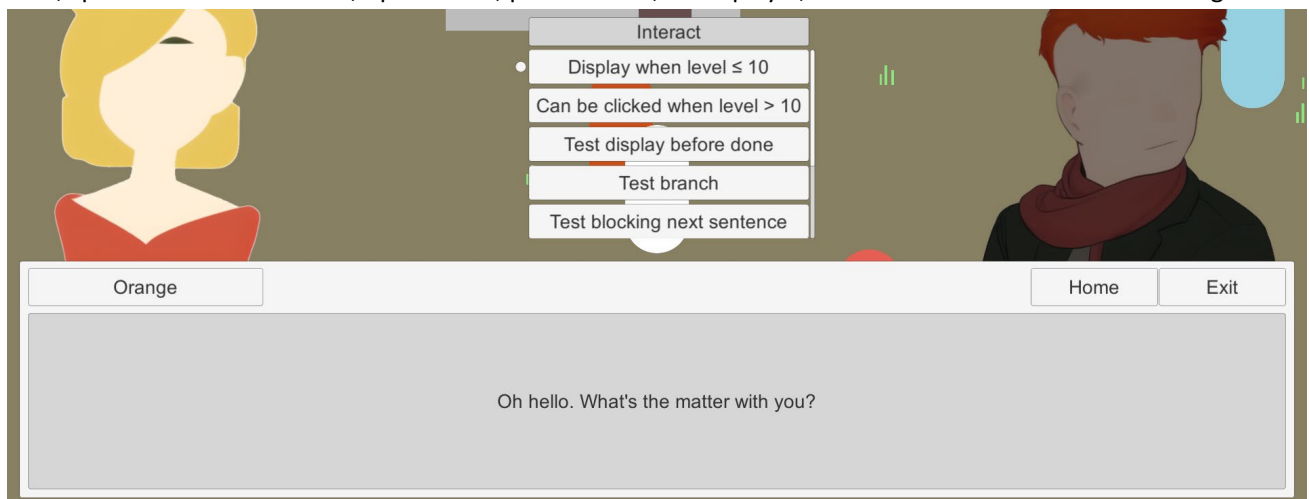
Static Method: Window Peek()	Return the top visible opened window
Static Method: Window Top()	Return the top window
Static Method: void Remove(Window)	Remove a window from the stack, is only used by the closing method of the Window class
Static Method: void Init()	Close all opened window and initialize the itself

Section 3: Generic Window

A generic window is a window designed for hot update, which is somewhat different from the original window system design concept. The original window system uses a type to identify a window, while a generic window uses a given window name to identify a window. Since this plugin does not implement hot update, the API of the general window in the window manager is not listed, and the generic window class will not be described in detail here, but its usage is still similar.

Section 4: Dialogue Window

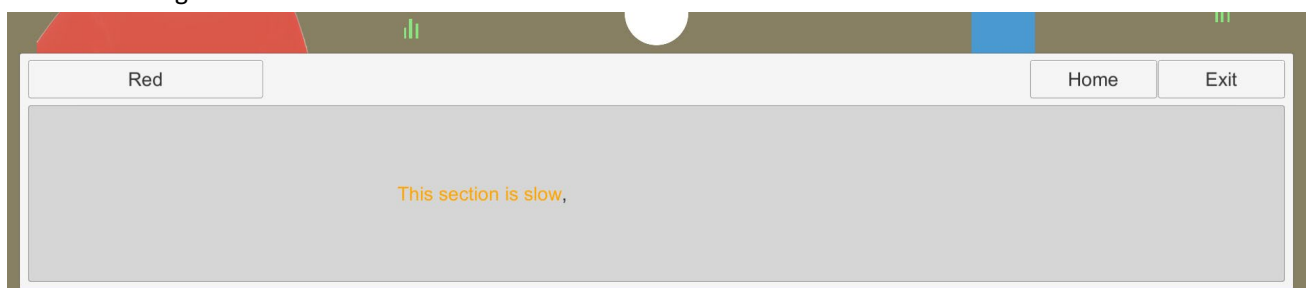
The dialogue window shows the dialogue we designed to the player, it is mainly composed of interlocutor text area, speech content text area, option area, portrait area, voice player, etc. Its default skin is as shown in figure:

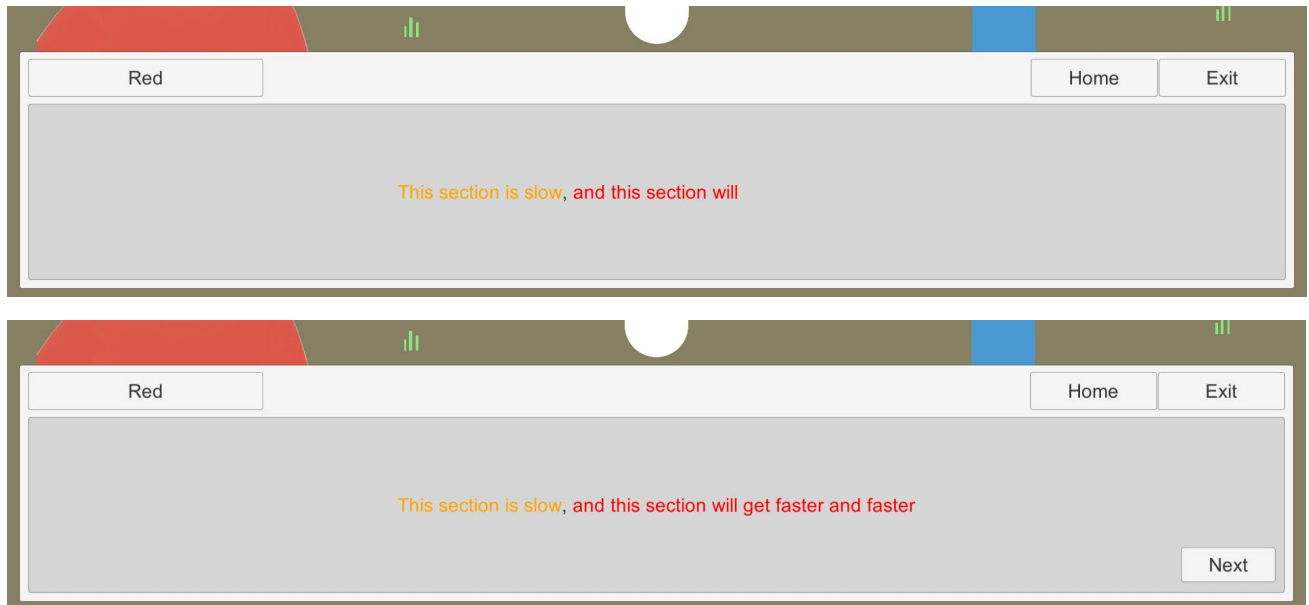


Among them, the “Home” button in the upper right corner can return to the first sentence when the dialogue window was opened, it displays only when the window is opened by interacting with interlocutor; and the “Exit” button can close the dialogue window and end the conversation in advance. They are all optional, and can be removed completely if not needed, it will not report errors at all.

The “Interact” above the options is the title of the option area, which can be modified to other text, for example, the option list can be used as a quest list, and then the title can be changed to "Quests".

The dialogue text will have a typing animation, and the options will not be displayed until the animation ends, as shown in the figure:





The playback speed of the typing animation is controlled by the speaking interval property of the sentence node. You can also click on the content text to skip the typing animation, switch to the next sentence, etc.

The type of the dialogue window is DialogueWindow, inherit from [InteractionWindow<Interlocutor>](#), its members are as follows:

Members For Runtime	Explanation
Field: nameText	Text of interlocutor name, please enable its rich text
Field: contentText	Text of speech content, please enable its rich text
Field: contentButton	Button of the content text, it's optional
Field: homeButton	Home button, it's optional
Field: nextButton	Next sentence button
Field: nextButtonText	Text of the "Next" button
Field: optionArea	GameObject of the option area
Field: optionTitle	GameObject of the option area title
Field: optionTitleText	Text of the title of the option area
Field: optionList	A list container of option
Field: leftPortrait	Portrait on the left, effective when enable portrait function
Field: rightPortrait	Portrait on the right, effective when enable portrait function
Field: adaptiveSize	Does the size of portrait is adaptive according to the portrait sprite?
Field: voicePlayer	Voice clip player, available when enable voice function
Field: skipDelay	The typing animation skip delay, that is, how long should the typing animation play at least before it can be skipped
Field: closeWhenExit	Whether to close dialogue window when reach an end node
Field: handler	An instance of dialogue handler
Field: nextClicked	Action when click the "Next" button, usually switch to the next sentence
Property: Target	The implementation of the abstract property of abstract type "InteractionWindow<Interlocutor>", it refers property "Interlocutor" of the "handler"

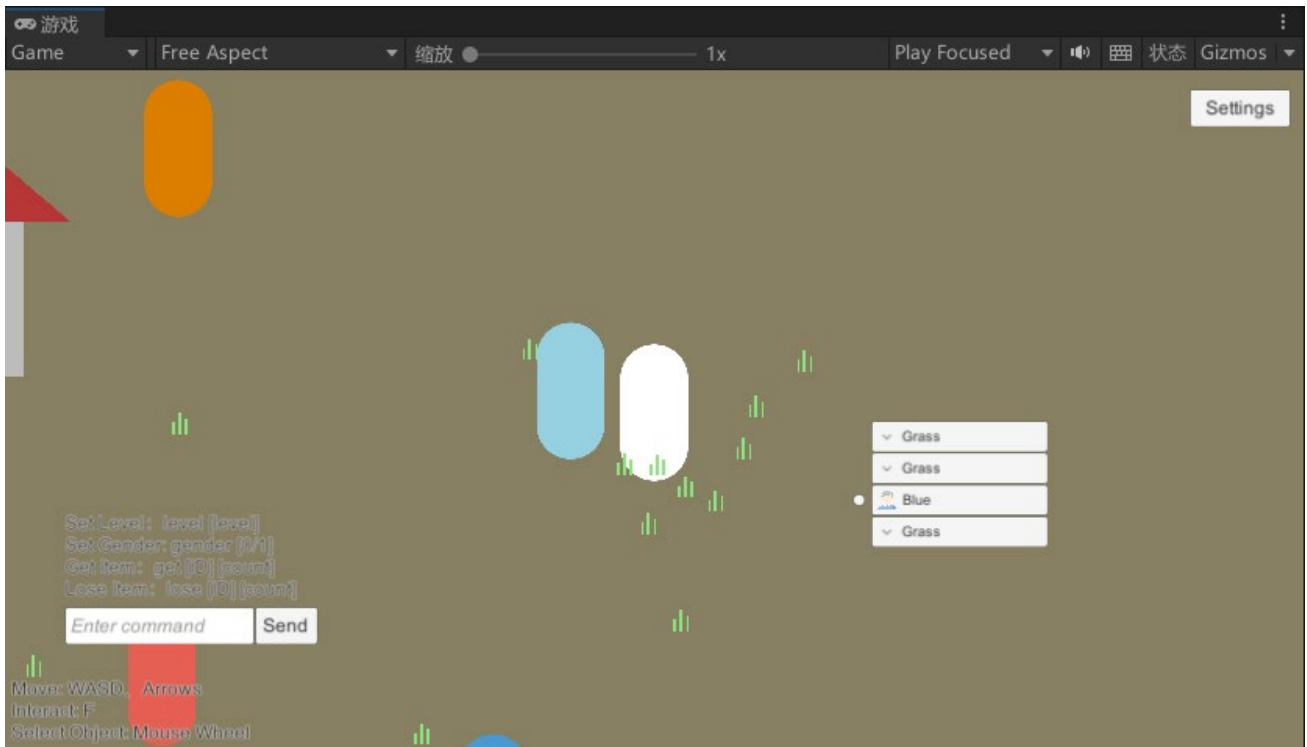
Property: LanguageSelector	The selector used by multilingual text system, it's "Dialogue" here
Method: void OnHandleNode(DialogueNode)	Refresh the hidden state of home button and close button
Method: void OnReachLastSentence()	Actions to perform when reach the end sentence
Method: string HandlingKeywords(string)	Process ID strings inside the given string
Method: void RefreshNextButton(Action, string)	Set the action and text of the next button
Method: void RefreshOptions (List<OptionCallback>, string)	Refresh option list and option area title
Method: void SetPortrait(bool, Sprite)	Refresh the portrait on the given side, effective when enable portrait
Method: void SetPortraitDark(bool, bool)	Darken the portrait on the given side, effective when enable portrait
Method: void Next()	This is used for one-key operation, such as just pressing the submit button to skip the typing animation, switch to the next sentence, or click the selected option, etc.
Method: void OnContentClick()	Actions to perform when click the content text
Method: void NextOption()	Select the next option
Method: void PrevOption()	Select the previous option
Method: void OnAwake()	Create an instance of dialogue handler and set the click action for each available button
Method: bool OnInterrupt()	Try to abort conversation and close window, the premise is that current conversation can be interrupted
Method: bool OnOpen_(params object[])	Handle the passed in arguments before open window
Method: bool OnClose_(params object[])	Clear before close window

Because the dialogue window is a kind of interaction window, if your game has no interactive objects and doesn't need the interaction buttons panel, but you still want to have conversations, just ignore all fields, properties, functions that relate to interaction API, and open the dialogue window by passing in a Dialogue type argument or an EntryNode type argument.

Part VII: Interaction System

Section 1: Concepts

The interactive system attached to this plugin is a list-based system similar to the one of “Genshin Impact”, consisting of interactive objects and interaction panel. Among them, the interaction panel will list the interactive objects near the player in the form of a list for the player to interact with, as shown in the following figure:



Section 2: Components

2.1 Interactive Interface

Interactive interface, as the name suggests, if you want some type of objects to be able to be added to the interactive panel, you have to inherit this interface.

The interactive interface is `IInteractive`, its members are as follows:

Members For Runtime	Explanation
Property: Name	Name of the interactive object
Property: Icon	Icon of the interactive object
Property: IsInteractive	Whether the object can be added to the interaction panel (that is, the interaction list)
Property: Interactable	Whether the object is already added to the interaction panel
Method: bool DoInteract()	Write your interaction logic code here, the return value indicates whether the interaction was successful, when it's successful, the interactive object will be removed from interaction list

Method: void EndInteraction()	This is a method with a concrete implementation, which is used to end the interaction. It gets called when its related InteractionWindow<T> is closed, or you can also call it where you need to end the interaction
Method: void OnInteractable()	This is a method with a specific implementation, but it is empty by default. It can be overridden with an explicit implementation. It gets called when the object is added to the interaction panel
Method: void OnNotInteractable()	This is a method with a specific implementation, but it is empty by default. It can be overridden with an explicit implementation. It gets called when the object is removed from the interaction panel
Method: void OnEndInteraction()	This is a method with a specific implementation, but it is empty by default. It can be overridden with an explicit implementation. It gets called when you call the method "void EndInteraction()"
Static Method: void Push(IInteractive)	Try to push the object to the bottom of interaction list
Static Method: void RemoveFromPanel(IInteractive)	Try to remove the object from the interaction list

How to use:

A sample code for interactive object that is based on 2D trigger is given below:

```
public class Interactive : MonoBehaviour, IInteractive
{
    [field: SerializeField]
    public string Name { get; set; }

    [field: SerializeField]
    public Sprite Icon { get; set; }

    [field: SerializeField]
    public bool IsInteractive { get; set; }

    bool IInteractive.Interactable { get; set; }

    public bool DoInteract()
    {
        Debug.Log("Hello, world!");
        return false;
    }

    void IInteractive.OnInteractable() => Debug.Log("OnInteractable");
    void IInteractive.OnNotInteractable() => Debug.Log("OnNotInteractable");
    void IInteractive.OnEndInteraction() => Debug.Log("OnEndInteraction");
}
```

```

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("Player")) Interactive.PushToPanel(this);
}
private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.CompareTag("Player")) Interactive.PushToPanel(this);
}
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.CompareTag("Player")) Interactive.RemoveFromPanel(this);
}
}

```

2.2 Base Interactive Object

The base type of interactive object is a MonoBehaviour type, and it can be used to be inherited by interactive MonoBehaviour object that has no base type, is a kind of concrete implementation of interactive interface.

The interactive object base type is InteractiveBase, inherit from MonoBehaviour, IInteractive, its members are as follows:

Members For Runtime	Explanation
Field: defaultName	The default name of this interactive object
Property: Name	Refer field "defaultName", can be override as other content
Field: defaultIcon	The default icon of this interactive object
Property: Icon	Refer field "defaultIcon", can be override as other content
Property: IsInteractive	An abstract property for overriding
Property: IInteractive.Interactable	An explicit implementation of interface property
Method: bool DoInteract()	An abstract method for overriding
Method: void OnDestroy()	Called by Unity, will remove itself from the interaction panel
Method: void OnDestroy_()	A destroy method for overriding, it's empty by default
Method: void OnInteractable()	A method for overriding
Method: void OnNotInteractable()	A method for overriding
Method: void OnEndInteraction()	A method for overriding
Method: void IInteractive.OnInteractable()	An explicit implementation of interface method, call the method "void OnInteractable()" of itself
Method: void IInteractive.OnNotInteractable()	An explicit implementation of interface method, call the method "void OnNotInteractable()" of itself
Method: void IInteractive.OnEndInteraction()	An explicit implementation of interface method, call the method "void OnEndInteraction()" of itself

How to use:

A sample code for interactive object base type that is based on 2D trigger is given below:

```

public abstract class Interactive2D : InteractiveBase
{
    public bool activated = true;
}

```

```

protected virtual void OnTriggerEnter2D(Collider2D collision)
{
    if (activated && collision.CompareTag("Player")) IInteractive.Push(this);
}

protected virtual void OnTriggerStay2D(Collider2D collision)
{
    if (activated && collision.CompareTag("Player")) IInteractive.Push(this);
}

protected virtual void OnTriggerExit2D(Collider2D collision)
{
    if (activated && collision.CompareTag("Player")) IInteractive.Remove(this);
}
}

```

And a sample code for interactive object sub type of it is given below:

```

public class Grass : Interactive2D
{
    public override string Name => "Grass";

    public override bool IsInteractive => true;

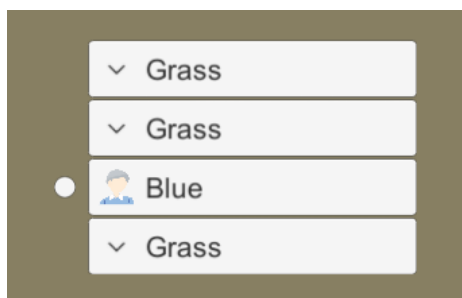
    public override bool DoInteract()
    {
        Debug.Log(name);
        return false;
    }
}

```

Section 3: Interaction UI

3.1 Interaction Panel

The interactive panel is also a list of interactive buttons. It will insert the new interactive object below the list and keep the selected state of currently selected interactive object, instead of automatically selecting the new one when inserting it like “Genshin Impact”, such as the annoying “Cook”. The following figure shows the so-called interaction panel:



The type of the interaction panel is `InteractionPanel`, inherit from [SingletonWindow<InteractionPanel>](#), its members are as follows:

Members For Runtime	Explanation
Field: view	View of button list
Field: buttonPrefab	Prefab of interaction button
Field: buttonParent	Parent transform to place interaction buttons
Field: buttonCacheParent	Parent transform to place cache of interaction buttons
Property: IsOpen	Override the property “Window.IsOpen” and returns True
Property: HideOnAwake	Override the property “Window.HideOnAwake” and returns True
Property: CanInteract	Whether it’s able to press “Interact” button (such as key “F”) to trigger the interaction or not
Property: CanScroll	Whether the selection item can be scrolled, such as “Genshin Impact”, it selects interactive object by mouse wheel
Field: objects	Objects added to the list
Field: buttons	Buttons added to the list
Field: showStates	Stack of panel’s display state, is used for avoiding unhiding at a wrong time
Field: selectedIndex	Child index of selected button
Field: pool	Object pool for interaction buttons
Static Method: bool Push(IInteractive)	Generate an interaction button for the given object
Static Method: void HidePanelBy(IInteractive, bool)	Hide or unhide panel by the given object
Static Method: bool Remove(IInteractive)	Remove button of the given object from the panel
Static Method: void Interact()	Interact with the selected interactive object
Static Method: void Next()	Select the next interactive object
Static Method: void Prev()	Select the previous interactive object upward
Method: void UpdateSelected()	Refresh the selected states of interaction buttons
Method: void UpdateView()	Refresh the view of panel, is used for avoiding blocking the selected button
Method: void OnAwake()	Create object pool
Method: void RegisterNotification()	Listen language switching event
Method: void UnregisterNotification()	Cancel listening language switching event
Method: bool OnOpen(params object[])	The panel is always open, returns false here
Method: bool OnClose(params object[])	The panel is always open, returns false here

3.2 Interaction Window

The interaction window is the base type of window types that handle a type of interactive object. For example, the dialogue window is a kind of interaction window that interacts with the interlocutor.

The type of the interaction window is `InteractionWindow<T>`, inherit from `Window`, the generic parameter `T` must be `IInteractive`, its members are as follows:

Members For Runtime	Explanation
---------------------	-------------

Field: hidePanelWhenInteracting	Whether to hide the window when interacting
Property: Target	The current interactive object
Method: void Interrupt()	Interrupt the interaction
Method: bool OnInterrupt()	A method for overriding, it tries to interrupt the interaction, be called by “void Interrupt()”, the return value indicates it’s able to be interrupted, returns True by default
Method: bool OnOpen(params object[])	A sealed overridden method, handles the display state of panel, it calls the method “bool OnOpen_(params object[])” of its sub-class
Method: bool OnClose(params object[])	A sealed overridden method, handles the display state of panel, it calls the method “bool OnClose_(params object[])” of its sub-class
Method: bool OnOpen_(params object[])	A method for overriding, replace the original method “bool OnOpen(params object[])”
Method: bool OnClose_(params object[])	A method for overriding, replace the original method “bool OnClose(params object[])”

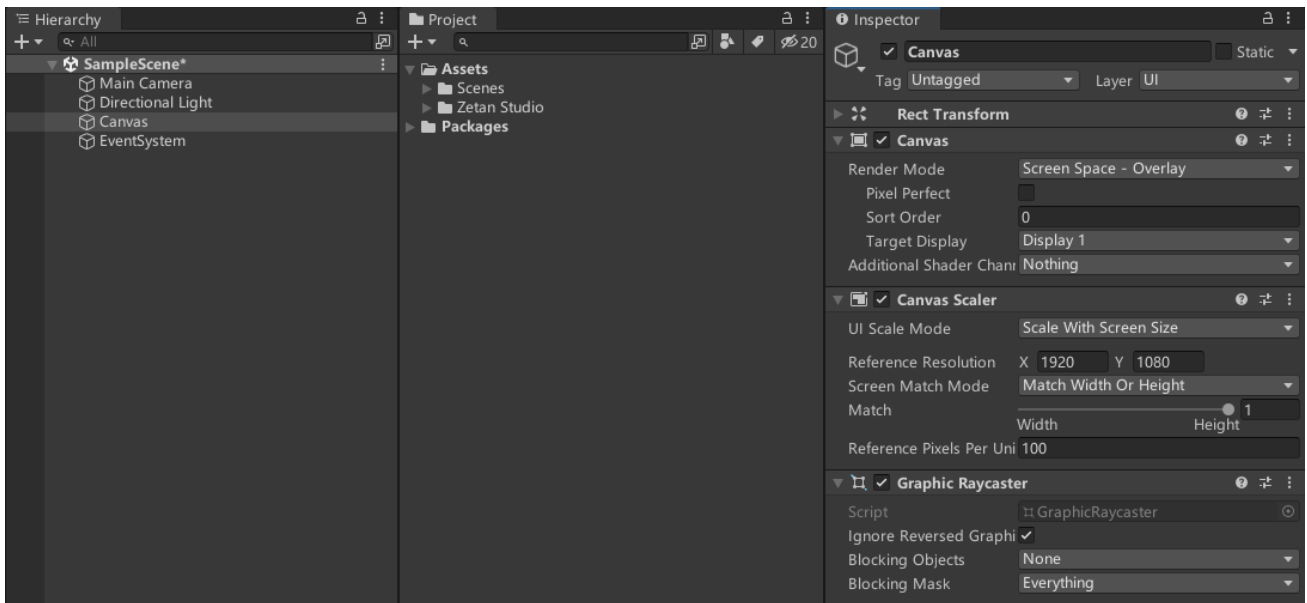
Part VII: Quick Start

Section 1: Using Interaction System for Conversation

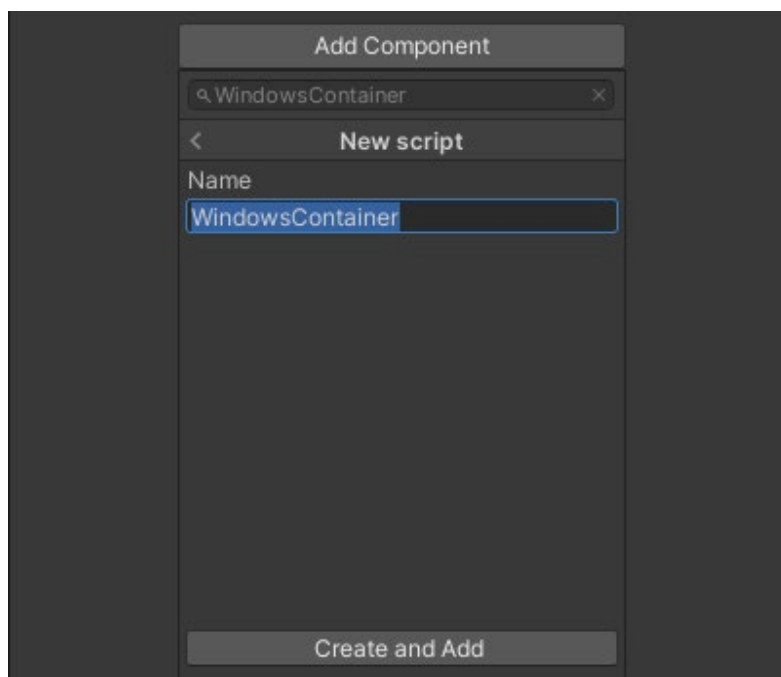
If you're using interaction system for conversation, it will be more complex, and compared to other method in section 2, more code needs to be written personally to use this plugin normally.

1.1 Building UI

1. First of all, we need a UI container to place the dialogue window, so we need to create a new Canvas in the scene first, and adjust the parameters of its Canvas Scaler component, as shown in the figure:



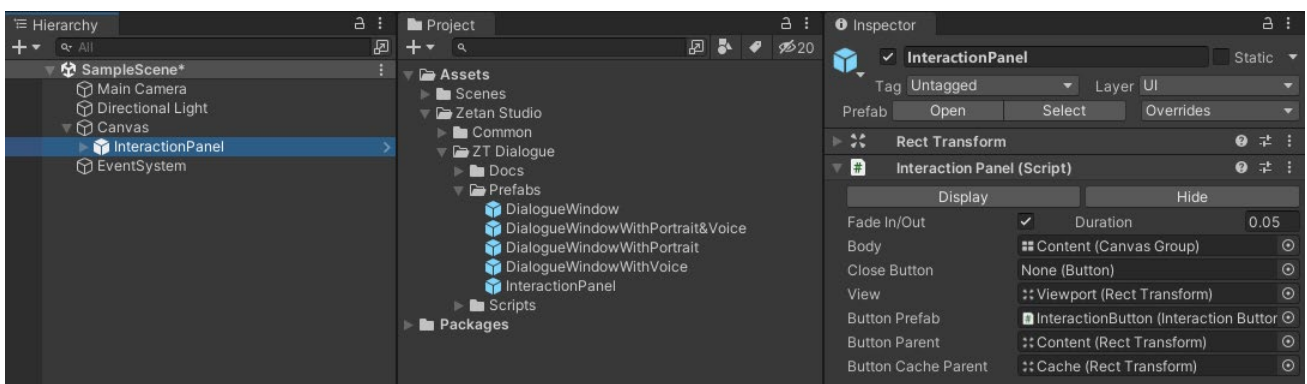
2. Then, create a new script for the Canvas, name whatever you want, as shown in the figure:



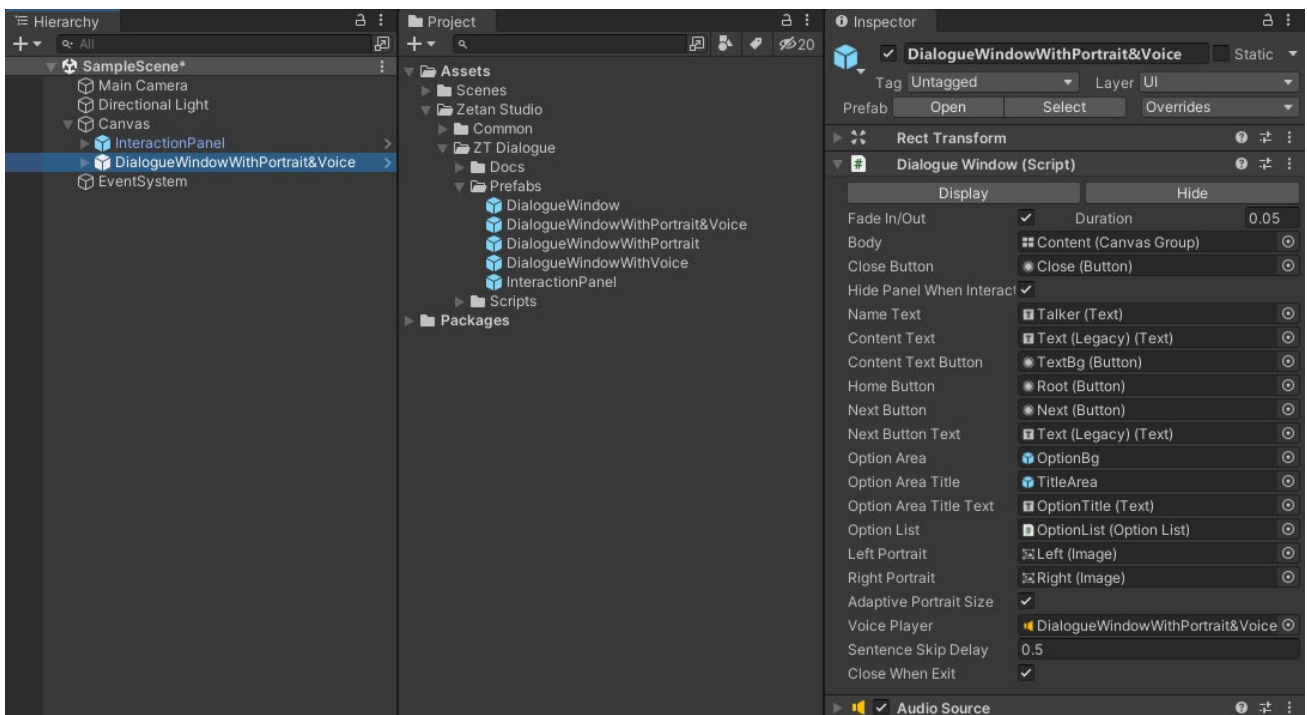
3. Edit the newly created script in the previous step as follows and save it using `UnityEngine;`
using `ZetanStudio.UI;`

```
public class WindowsContainer : MonoBehaviour
{
    private void Awake()
    {
        //This step we're setting the singleton UI container of UI manager.
        WindowManager.WindowsContainer = transform;
    }
}
```

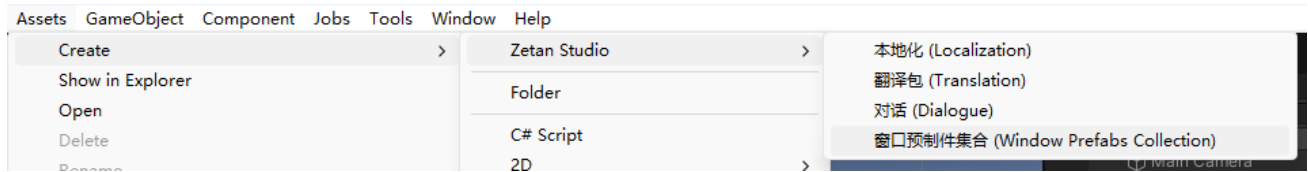
4. Then we drag the prefab of interaction panel onto the canvas as its child object, as shown in the figure:



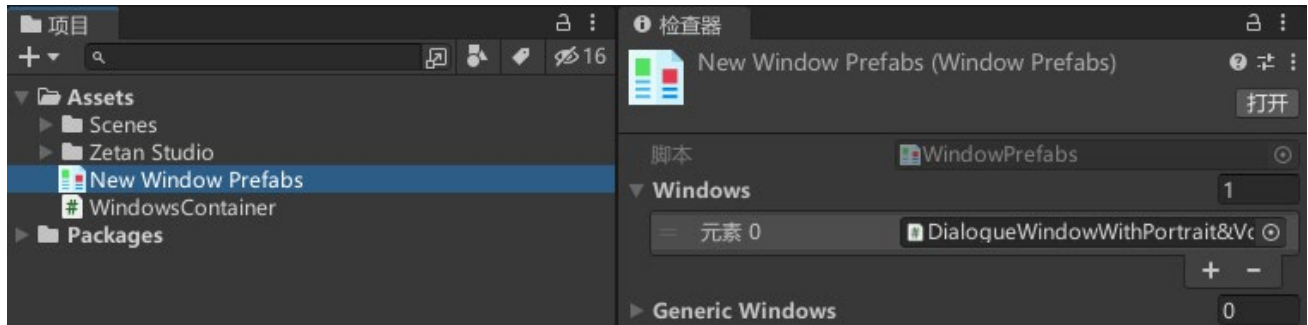
5. Next, we need to place the dialogue window, there are two methods here. The first method is not to generate it through the prefab, but to place it directly in the scene. At this time, we only need to put the `DialogueWindow` prefab under the “Zetan Studio/ZT Dialogue/Prefabs” folder (here there are multiple `DialogueWindow` prefab, please select one of them) directly onto the canvas as its child object, as shown in the figure:



6. The second method is more complicated. You need to create a new window prefab collection asset at first (“WindowPrefabs” type asset), as shown in the figure:



7. Then add the dialogue window prefab to its windows list, as shown in the figure:



8. Then add a window prefab collection parameter to the script created in step 2, and use it as the singleton of the “WindowPrefabs” class in the Awake function, as shown below:

```
using UnityEngine;
```

```
using ZetanStudio.UI;
```

```
public class WindowsContainer : MonoBehaviour
```

```
{
```

```
    [SerializeField]
```

```
    private WindowPrefabs windowPrefabs;
```

```
    private void Awake()
```

```
    {
```

```
        //This step we're setting the singleton instance of WindowPrefabs class.
```

```
        WindowPrefabs.Instance = windowPrefabs;
```

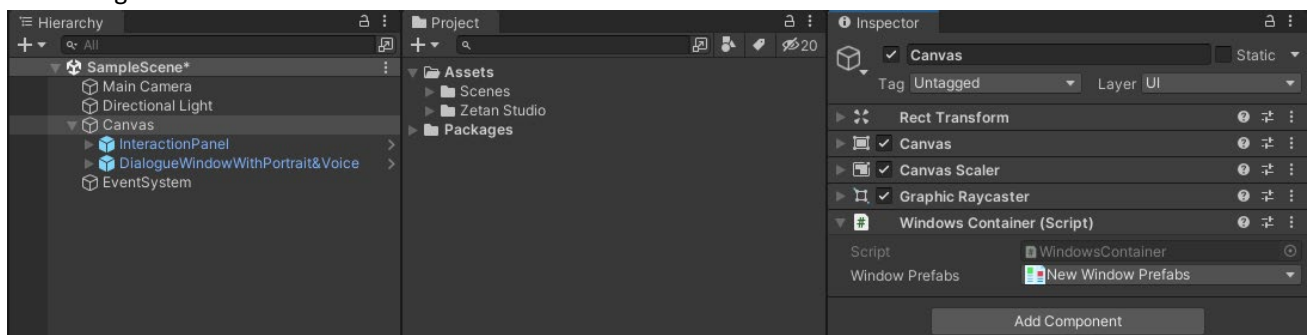
```
        //This step we're setting the singleton UI container of UI manager.
```

```
        WindowManager.WindowsContainer = transform;
```

```
    }
```

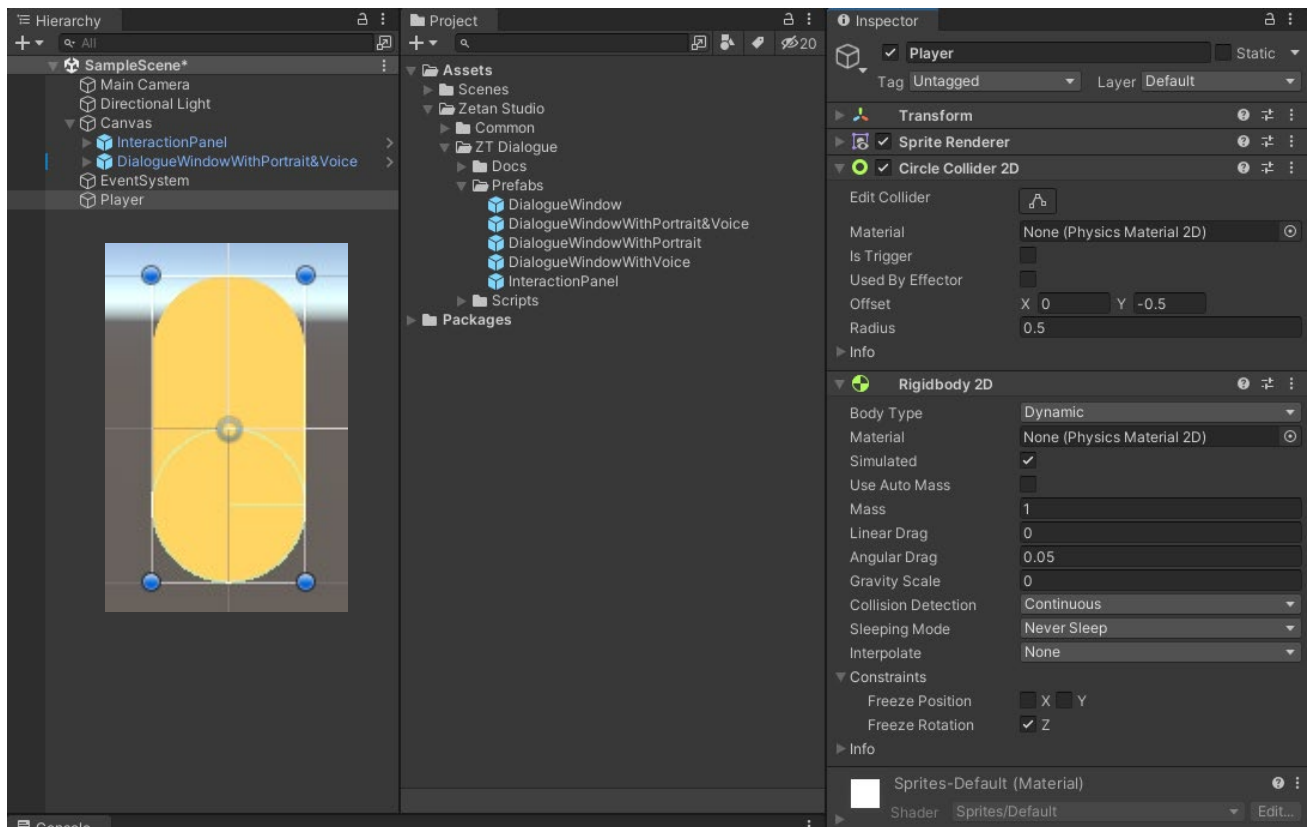
```
}
```

9. Set the “Window Prefabs” of the “Windows Container” component to the asset created in step 7, as shown in the figure:



1.2 Adding Characters

1. Next, we start to make the player character. Create a new 2D object in the scene, here I use a 2D capsule, add a collider and a 2D rigid body to it, adjust the rigid body parameters, and then set its Tag to "Player", as shown in the figure:



2. Then we add a script as follows to the player object to control it:
using UnityEngine;

```
public class Player : MonoBehaviour
{
    [SerializeField]
    private Rigidbody2D rigidbd;
    [SerializeField]
    private float moveSpeed = 5f;

    private Vector3 movement;

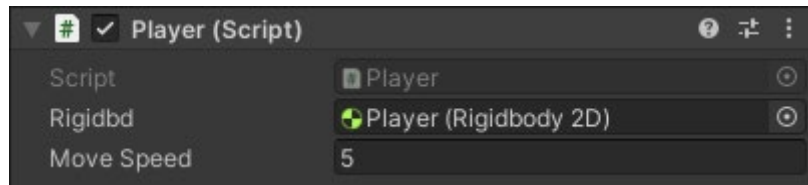
    private void Update()
    {
        movement = Vector3.ClampMagnitude(new Vector3(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"), 1), 1);
    }
}
```

```

private void FixedUpdate()
{
    rigidbd.MovePosition(rigidbd.transform.position + moveSpeed * Time.deltaTime * movement);
}
}

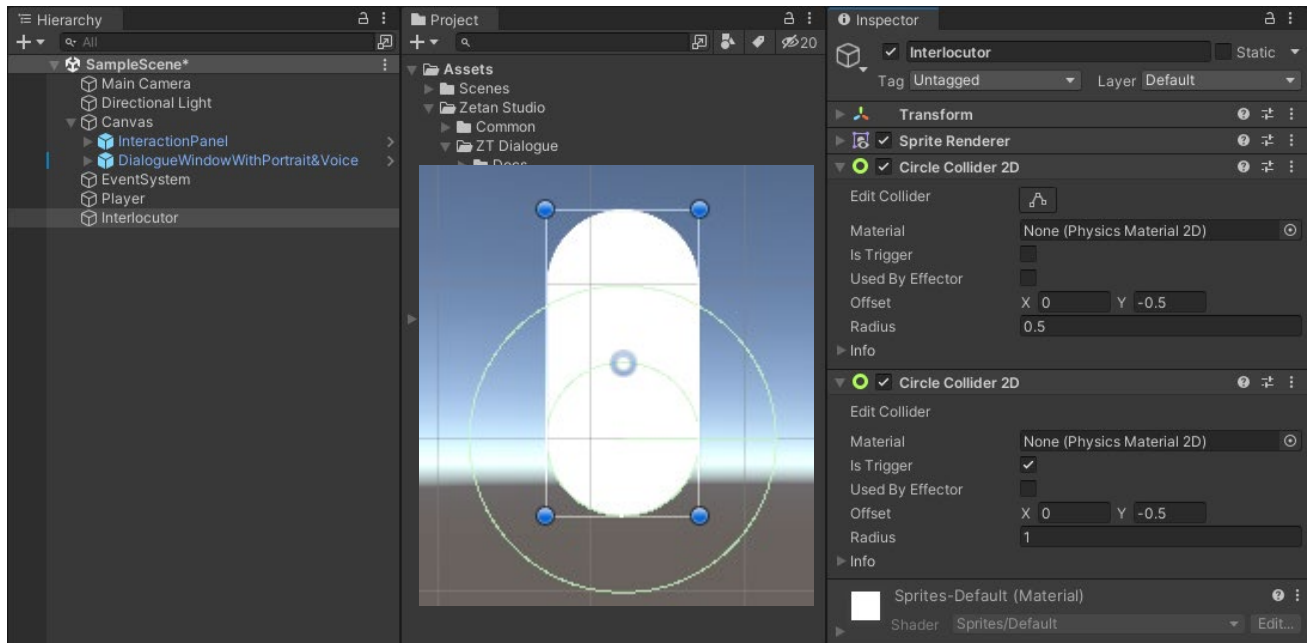
```

3. Drag the player's rigid body component to the "Rigidbody" of the component created in the previous step to reference it, as shown in the figure:



Now if we run the game, we will find that we can move the player character.

4. Then, we start to make the interlocutor. Go ahead and create a new 2D object in the scene, here I still use a 2D capsule and make it a little bit farther away from the player character. Then add a 2D collider and a 2D trigger to the interlocutor, as shown in the figure:



The point to note is that the range of the trigger should be larger than the one of collider.

5. Go ahead and add a new script to the interlocutor as follows:

```

using UnityEngine;
using ZetanStudio.DialogueSystem;
using ZetanStudio.DialogueSystem.UI;
using ZetanStudio.InteractionSystem;
using ZetanStudio.UI;

```

```

public class Interlocutor : MonoBehaviour, IInterlocutor
{
    [field: SerializeField]
    public string Name { get; private set; }
}

```

```

[field: SerializeField]
public Sprite Icon { get; private set; }

[field: SerializeField]
public Dialogue Dialogue { get; private set; }

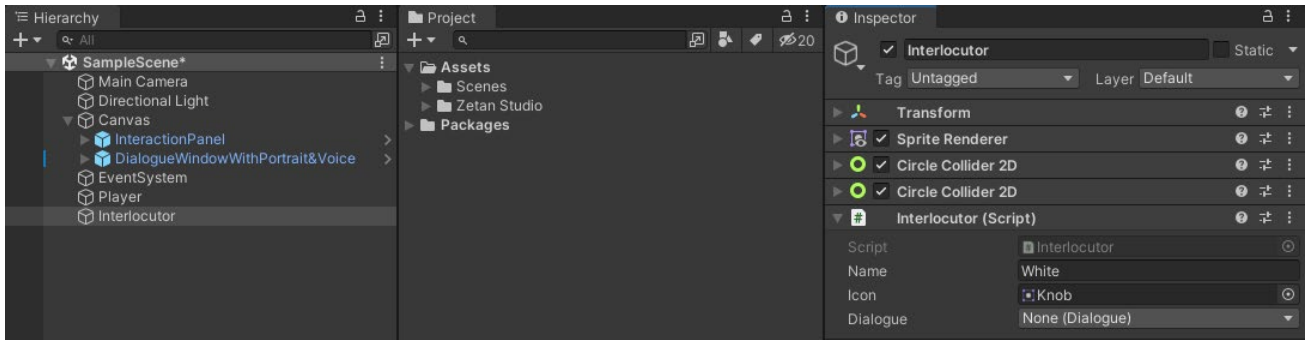
public bool IsInteractive => true;
bool IInteractive.Interactable { get; set; }

bool IInteractive.DoInteract()
{
    return WindowManager.OpenWindow<DialogueWindow>(this);
}
//Call this when player leave the dialogue trigger area.
void IInteractive.OnNotInteractable()
{
    //If we're having conversation with this interlocutor,
    if (WindowManager.IsWindowOpen<DialogueWindow>(out var window) && window.Target == this as
IInterlocutor)
        //Then we interrupt the conversation.
        window.Interrupt();
}

private void OnTriggerEnter2D(Collider2D collision)
{
    //Player enter the trigger area, then add this interlocutor to interaction panel.
    if (collision.CompareTag("Player")) IInteractive.PushToPanel(this);
}
private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.CompareTag("Player")) IInteractive.PushToPanel(this);
}
private void OnTriggerExit2D(Collider2D collision)
{
    //Player leave the trigger area, then remove this interlocutor from interaction panel.
    if (collision.CompareTag("Player")) IInteractive.RemoveFromPanel(this);
}
private void OnDestroy()
{
    IInteractive.RemoveFromPanel(this);
}
}

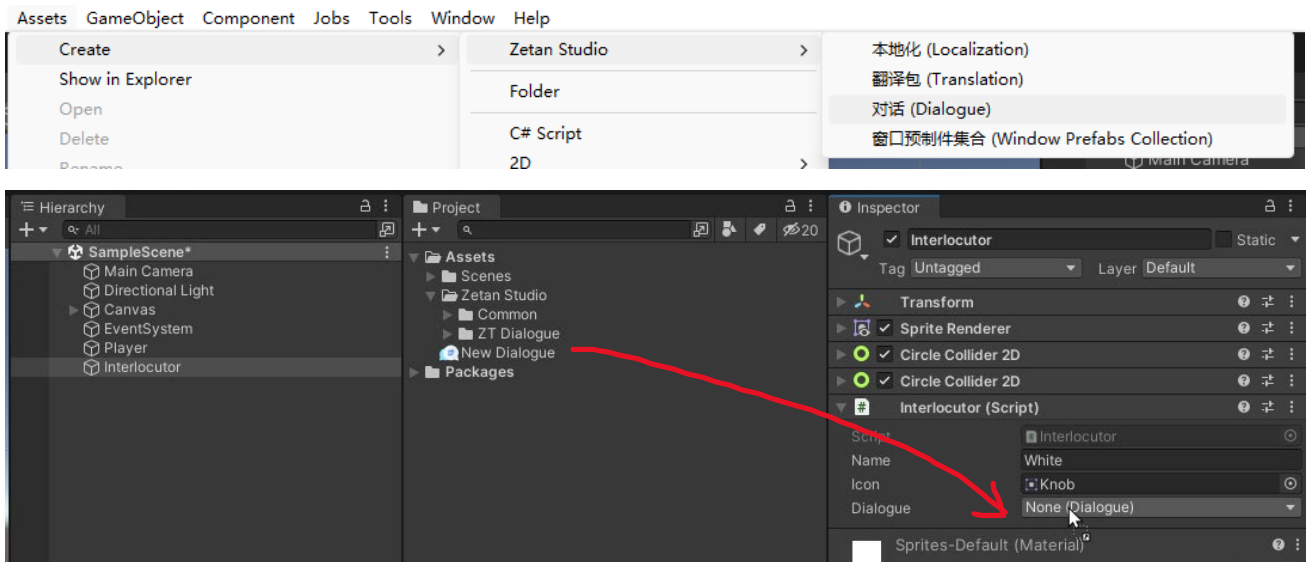
```

6. Setting up the name and icon of interlocutor, as shown in the figure:

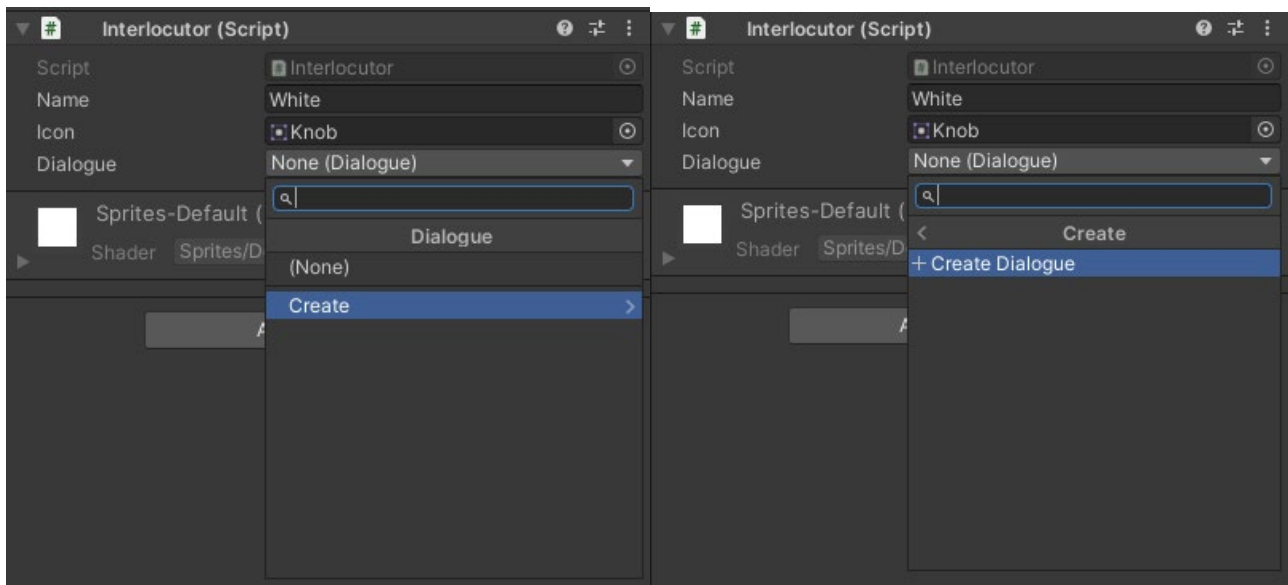


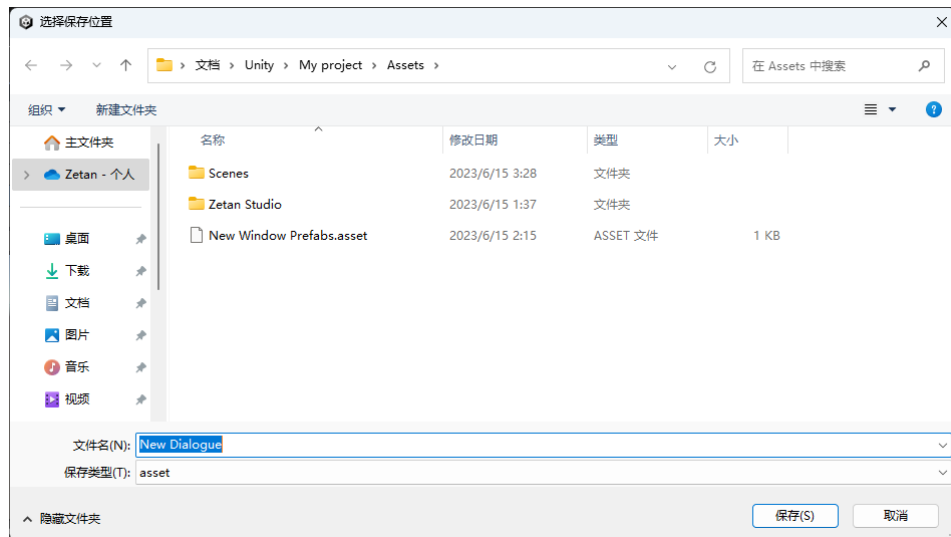
1.3 Creating and Editing Dialogue

1. Create a new dialogue for this interlocutor, which can be referenced by creating it through the toolbar and dragging it to the “Dialogue” of the interlocutor component, as shown in the figure:

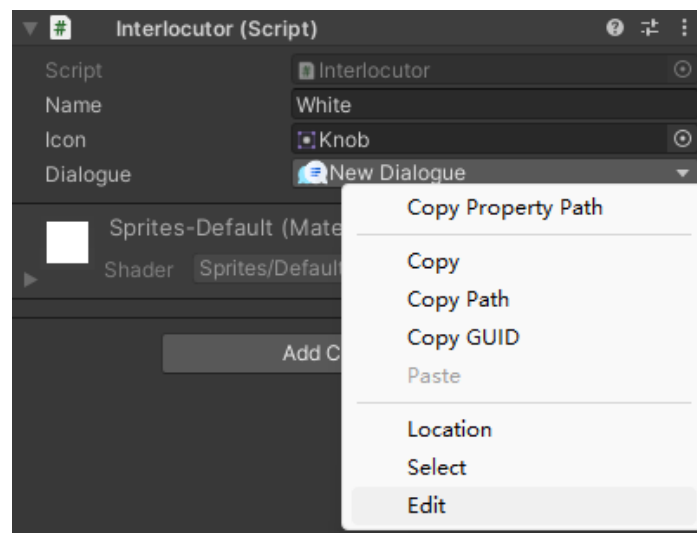


2. Or you can create it in the drop-down window of the “Dialogue” of interlocutor component, as shown in the figure:

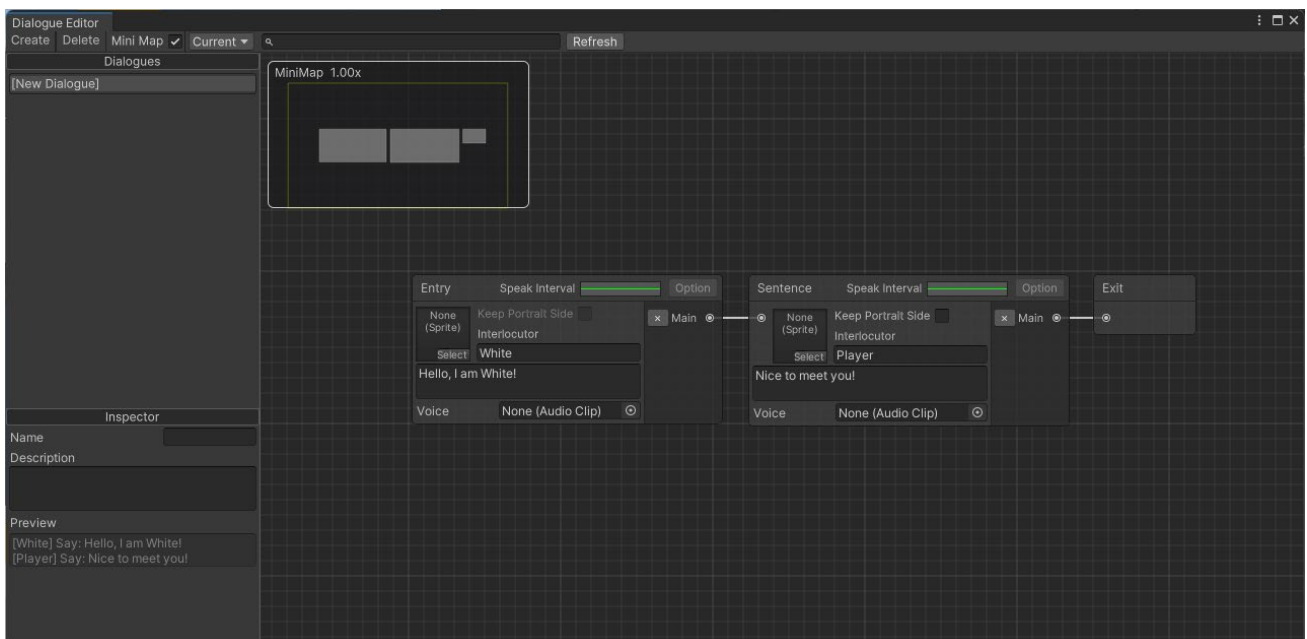




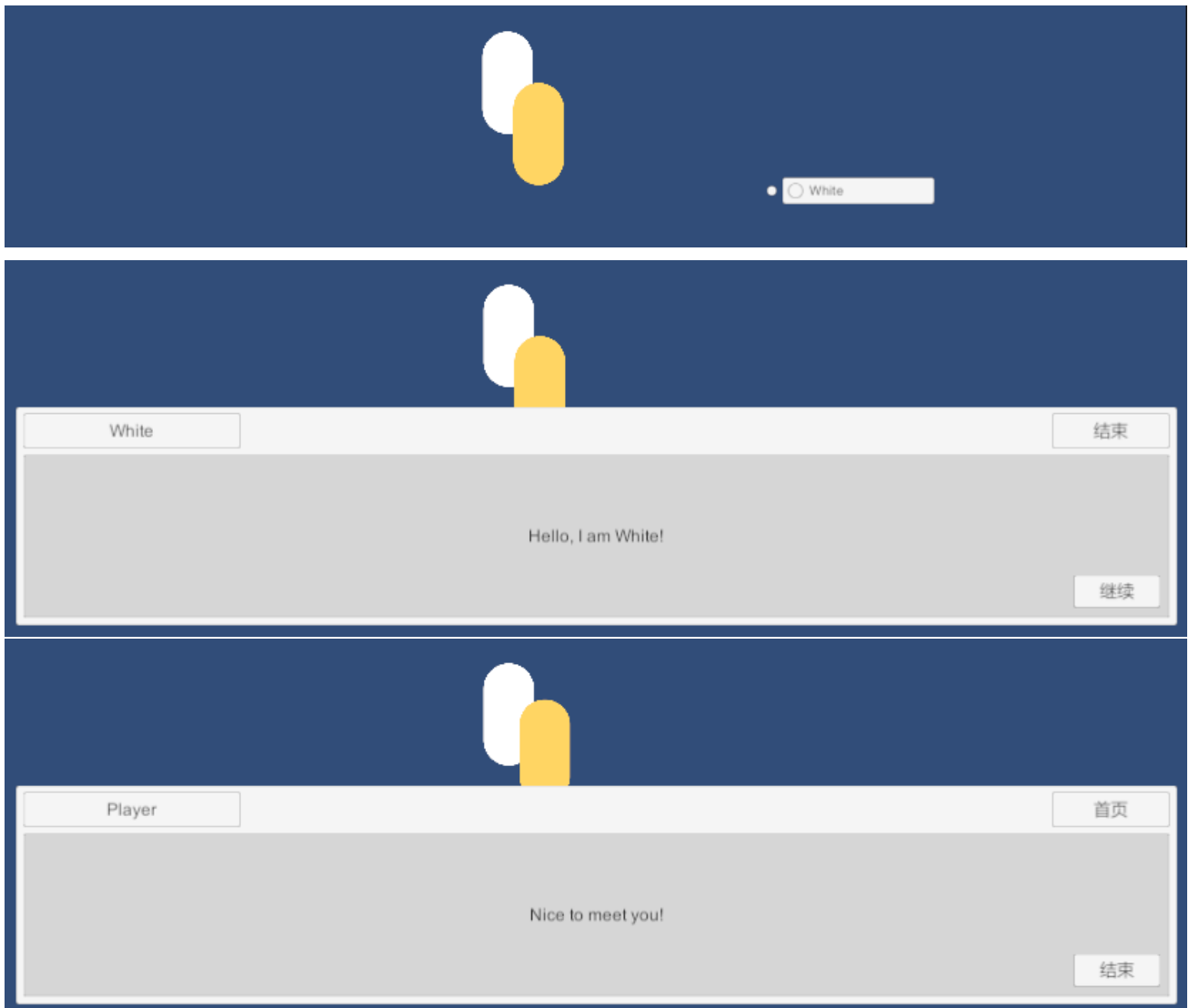
- Now let's edit this dialogue, you can double-click the dialogue asset to open the dialogue editor, or right-click the "Dialogue" drop-down button in the interlocutor component to open it, as shown in the figure



- Edit current dialogue as you want in the opened dialogue editor, as shown in the figure:



5. Last, run the game, move player to the side of interlocutor, and you can trigger the conversation by click button shows in interaction panel, as shown in the figure:

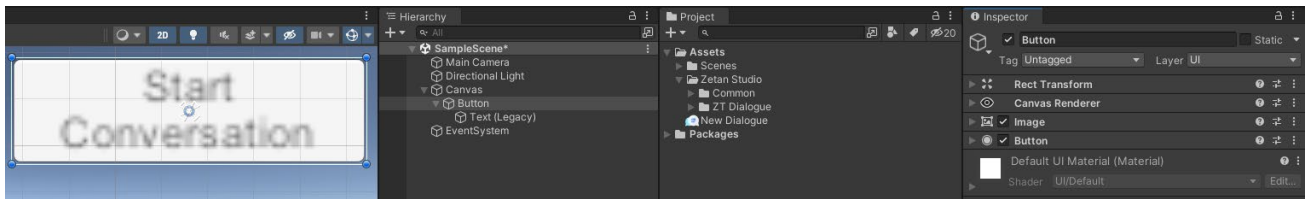


Section 2: Trigger Conversation in Other Way

If there is already an interaction system in the project or the interaction function is not needed, and the interaction system that comes with this plugin is not used, you can refer to this section to learn how to trigger the conversation. It should be noted that the “Home” button of the dialogue window will not be available without opening the dialogue window through the interactive system.

2.1 Building UI

1. The operation of building the UI is consistent with the steps in the first section, except for the operation related to the interactive panel in step 4.
2. We need to pass the dialogue asset into the dialogue window to open it. Here I use a simple example to illustrate how to do it.
3. Assuming that we use a button to trigger the conversation now, we must first add a button to the canvas created in step 1, as shown in the figure:



4. Add a new script to the button created in step 3, as shown below:

using UnityEngine;

using ZetanStudio.DialogueSystem;

using ZetanStudio.DialogueSystem.UI;

using ZetanStudio.UI;

```
public class StartDialogue : MonoBehaviour
```

```
{
```

```
    public Dialogue dialogue;
```

```
    public void TriggerDialogue()
```

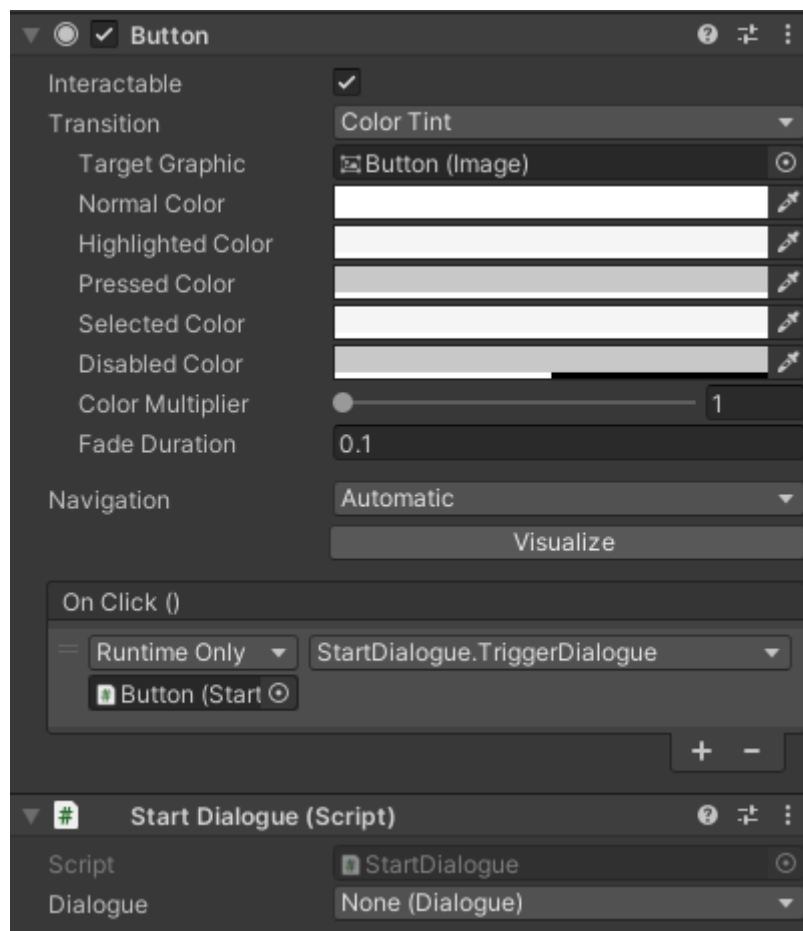
```
{
```

```
        WindowManager.OpenWindow<DialogueWindow>(dialogue);
```

```
}
```

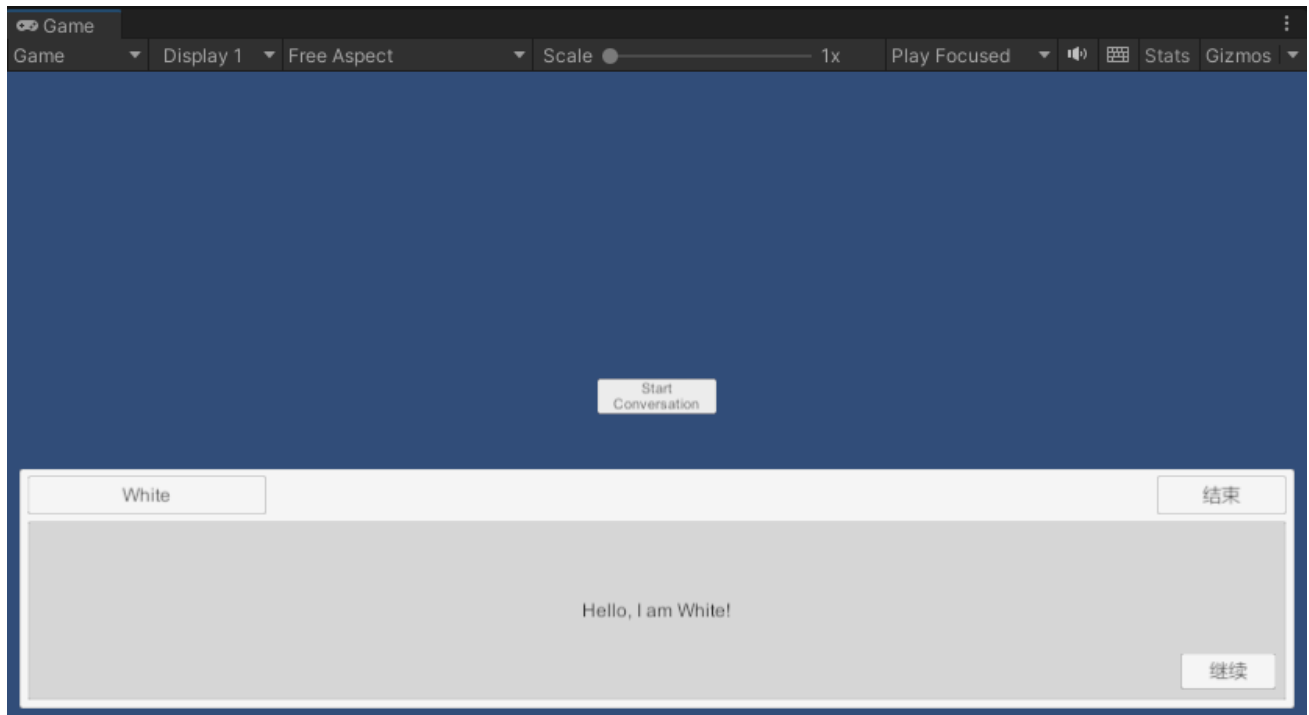
```
}
```

5. Bind the TriggerDialogue method of this script to the click event of the button, as shown in the figure:



2.2 Creating and Editing Dialogue

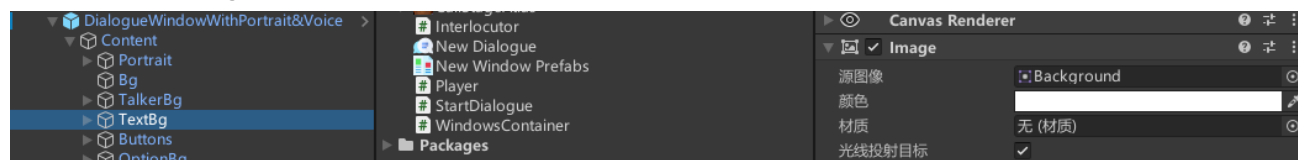
1. The method of creating and editing a dialogue is the same as the first section, except that the object to edit is changed from the interlocutor to the button.
2. After completing the creation and editing of the dialogue, run the game, and you can trigger the dialogue by clicking the “Start Conversation” button, as shown in the figure:



Section 3: Beautify Dialogue Window

In this section, we will take the DialogueWindowWithPortrait&Voice prefab as an example to illustrate how to change the theme of the dialogue window, that is, to beautify the dialogue window. The objects that can be beautified are basically as follows:

1. Modify the Texture of “Content/TextBg” to replace the background of the text area of the dialogue window, as shown in the figure:



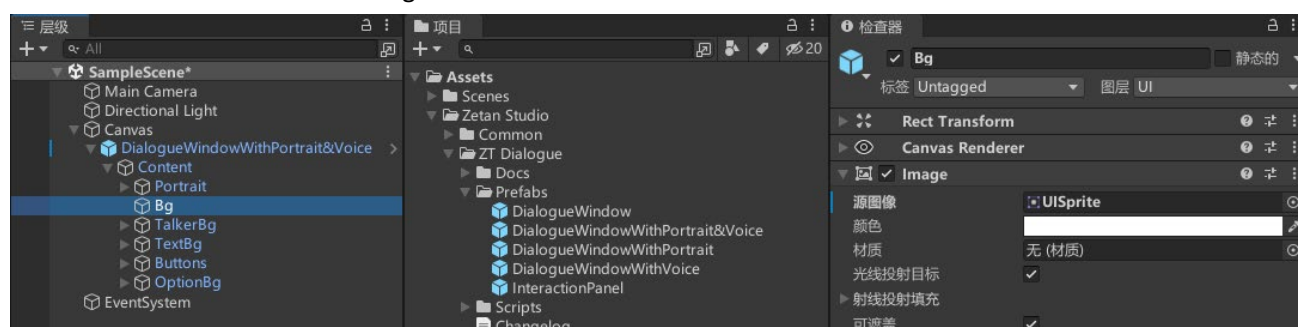
Unmodified:



Modified:



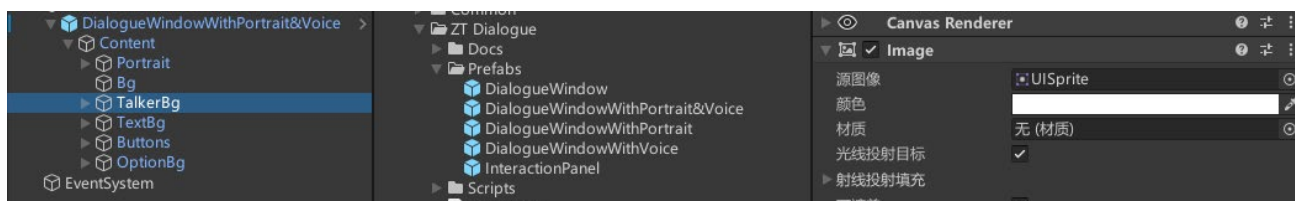
2. Modify the Texture of “Content/Bg” to change the background of the dialogue window, as shown in the figure:



Modified:



3. Modify the Texture of "Content/TalkerBg" to change the background of the interlocutor name area, as shown in the figure:



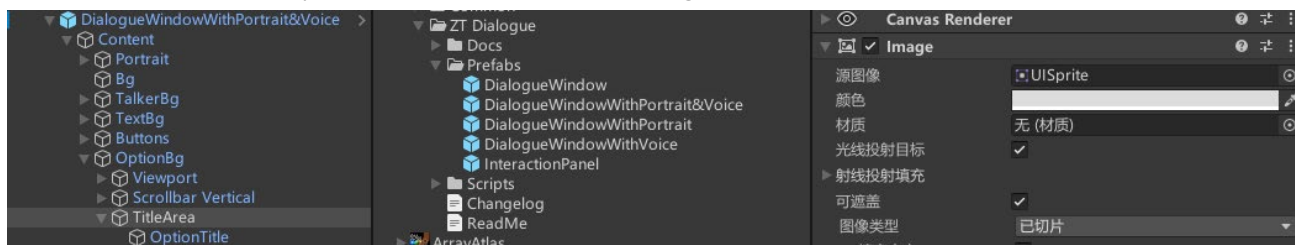
Modified:



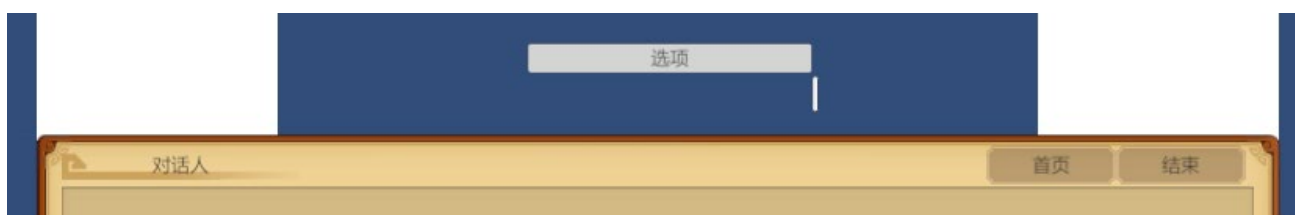
4. Then modify the Texture of the three buttons on the right, the modified effect is as follows:



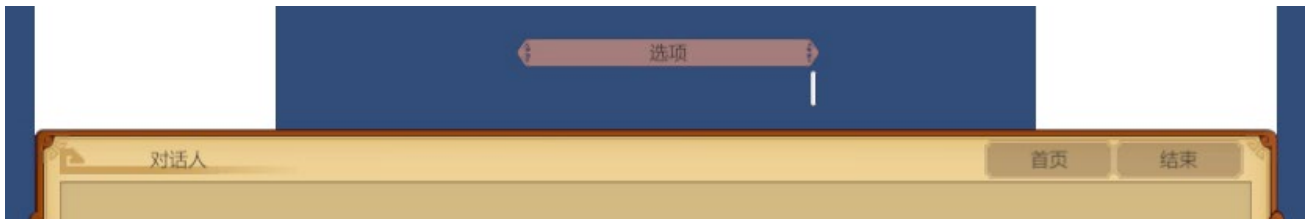
5. Modify the Texture of "Content/OptionBg/TitleArea" to replace the background of the option area title, as shown in the figure:



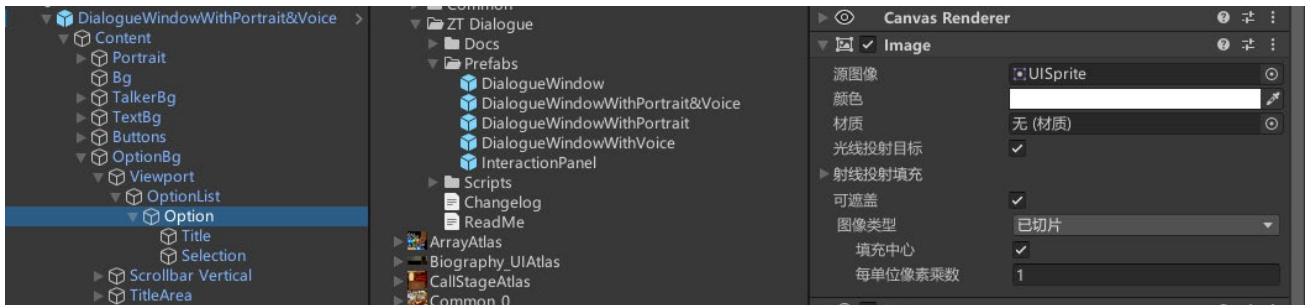
Unmodified:



Modified:



6. Modify the Texture of "Content/OptionBg/Viewport/OptionList/Option" to replace the background of the option button, as shown in the figure:



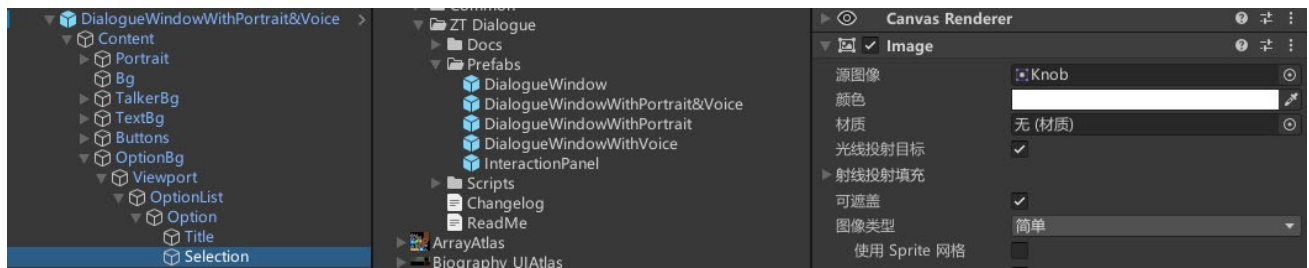
Unmodified:



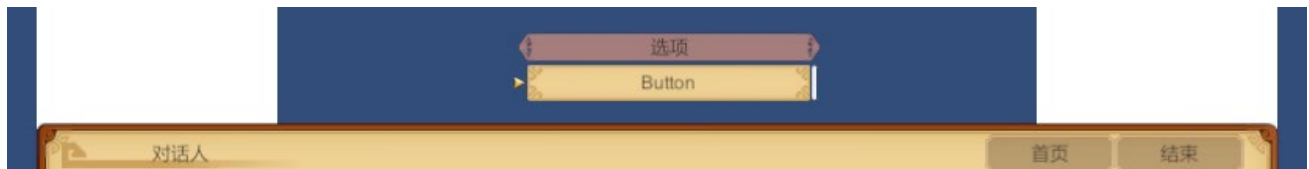
Modified:



7. Modify the Texture of "Content/OptionBg/Viewport/OptionList/Option/Selection" to replace the selected mark on the left side of the option button, as shown in the figure:



Modified:



8. At this point, we have completed the simple beautification of the dialogue window, and the final effect is shown in the figure:

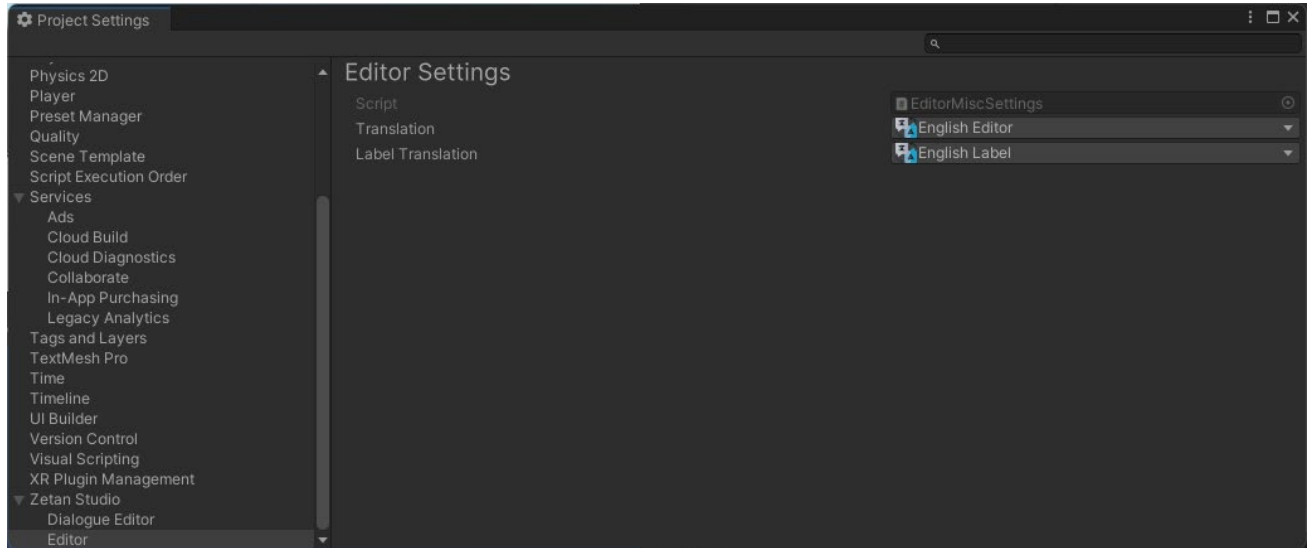


Among them, the portraits on the left and right sides are displayed as white squares, which is a normal phenomenon, and they will be hidden automatically after running the game; the beautification of the scroll bar on the right side of the options area will not be demonstrated here, please handle it yourself.

Appendix

Editor Settings

You can find the editor settings of this plugin in menu “Edit→Project Settings...→Zetan Studio→Editor”, as shown in figure:



Among them, the editor settings have two translation settings, the first is the global translation of the editor, and the second is the translation for the [Label] attributes, you may have to set both of them to get the desired editor translation effect.

Classes

1. Lightweight Generic Data

The lightweight generic data is a type to store simple user-specified data without add additional fields, and those data will be accessed by number indices or string keys.

The type of lightweight generic data is GenericData, its members are as follows:

Members For Runtime	Explanation
Field: intList	Number-indexed integer data
Field: boolList	Number-indexed boolean data
Field: floatList	Number-indexed float data
Field: stringList	Number-indexed string data
Field: dataList	Number-indexed generic data
Field: intDict	Key-indexed integer data
Field: boolDict	Key-indexed boolean data
Field: floatDict	Key-indexed float data
Field: stringDict	Key-indexed string data
Field: dataDict	Key-indexed generic data
Indexer: object this[string key]	A shortcut to access key-indexed data that is supported

Method: int Write(int)	Write a number-indexed integer data and return it again
Method: bool Write(bool)	Write a number-indexed boolean data and return it again
Method: float Write(float)	Write a number-indexed float data and return it again
Method: string Write(string)	Write a number-indexed string data and return it again
Method: GenericData Write(GenericData)	Write a number-indexed generic data and return it again
Method: T WriteAll<T>(T)	Write multiple number-indexed data and return them, the generic parameter T must be IEnumerable
Method: int Write(string, int)	Write a key-indexed integer data and return it again
Method: bool Write(string, bool)	Write a key-indexed boolean data and return it again
Method: float Write(string, float)	Write a key-indexed float data and return it again
Method: string Write(string, string)	Write a key-indexed string data and return it again
Method: GenericData Write(string, GenericData)	Write a key-indexed generic data and return it again
Method: ReadOnlyCollection<int> ReadIntList()	Read all number-indexed integer data
Method: ReadOnlyCollection<bool> ReadBoolList()	Read all number-indexed boolean data
Method: ReadOnlyCollection<float> ReadFloatList()	Read all number-indexed float data
Method: ReadOnlyCollection<string> ReadStringList()	Read all number-indexed string data
Method: ReadOnlyCollection<GenericData> ReadDataList()	Read all number-indexed generic data
Method: ReadOnlyDictionary<string, int> ReadIntDict()	Read all key-indexed integer data
Method: ReadOnlyDictionary<string, bool> ReadBoolDict()	Read all key-indexed boolean data
Method: ReadOnlyDictionary<string, float> ReadFloatDict()	Read all key-indexed float data
Method: ReadOnlyDictionary<string, string> ReadStringDict()	Read all key-indexed string data
Method: ReadOnlyDictionary<string, GenericData> ReadDataDict()	Read all key-indexed generic data
Method: bool TryReadInt(int, out int)	Try to access an integer data according to the number index
Method: bool TryReadBool(int, out bool)	Try to access a boolean data according to the number index
Method: bool TryReadFloat(int, out float)	Try to access a float data according to the number index
Method: bool TryReadString(int, out string)	Try to access a string data according to the number index
Method: bool TryReadData(int, out GenericData)	Try to access a generic data according to the number index
Method: bool ReadInt(int)	Access an integer data according to the number index
Method: bool ReadBool(int)	Access a boolean data according to the number index
Method: bool ReadFloat(int)	Access a float data according to the number index
Method: bool ReadString(int)	Access a string data according to the number index
Method: bool ReadData(int)	Access a generic data according to the number index
Method: bool TryReadInt(string, out int)	Try to access an integer data according to the key index
Method: bool TryReadBool(string, out bool)	Try to access a boolean data according to the key index

Method: bool TryReadFloat(string, out float)	Try to access a float data according to the key index
Method: bool TryReadString(string, out string)	Try to access a string data according to the key index
Method: bool TryReadData (string, out GenericData)	Try to access a generic data according to the key index
Method: bool ReadInt(string)	Access an integer data according to the key index
Method: bool ReadBool(string)	Access a boolean data according to the key index
Method: bool ReadFloat(string)	Access a float data according to the key index
Method: bool ReadString(string)	Access a string data according to the key index
Method: bool ReadData(string)	Access a generic data according to the key index

2. Singleton Asset

The singleton assets refer to singleton ScriptableObject subclass assets, they should be put inside folder named “Resources” or its sub folder. Although asset files can be copied and pasted at will in the editor, try not to do this for singleton assets. The singleton asset type is a generic type, but in order to customize the inspector of the singleton assets, a base type must be added. This base class is not a real singleton asset type and has no function about singleton.

The base type of singleton asset is SingletonScriptableObject, inherit from ScriptableObject, its Editor-Use member is as follows:

Members For Editor	Explanation
Static Method: void FindEditorInstance()	This is an automated method to refresh static instance references of singleton assets when the editor is opened or after recompilation

The type of singleton asset is SingletonScriptableObject<T>, inherit from previous type, the generic parameter T must be ScriptableObject, its members are as follows:

Members For Runtime	Explanation
Static Field: instance	An instance of type T
Static Property: Instance	Related read only property of field “instance”
Constructor: SingletonScriptableObject()	Refer itself to field “instance” to be the singleton

Its Editor-Use members are as follows:

Members For Editor	Explanation
Static Method: T GetOrCreate()	Get or create an instance
Static Method: void CreateSingleton()	Try to create a singleton

3. Singleton Window

For singleton window, there will be a static reference to the specified type of window, try not to access this static reference if there’s no any available instance, because if the reference is empty, it will always use the method “T FindObjectOfType<T>(true)” to lookup, affecting performance.

Type of singleton window is SingletonWindow<T>, inherit from Window, the generic parameter T must be Window, its members are as follows:

Members For Runtime	Explanation
Static Field: instance	An instance of type T
Static Property: Instance	Related read only property of field “instance”

Interfaces

1. Player Name Holder Interface

The player name holder interface is used in the keyword system to replace some special ID strings with the name of player character.

The player name holder interface is `IPlayerNameHolder`, its members are as follows:

Members For Runtime	Explanation
Static Property: Instance	A singleton of player name holder
Property: Name	The name of player character

How to use:

First let the player class inherit this interface, implement the property “Name”, and then refer the instance of the player class to the static property “Instance” of this interface. A sample code for player type is given below:

```
public class Player : MonoBehaviour, IPlayerNameHolder
{
    [field: SerializeField]
    public string Name { get; set; }

    private void Awake()
    {
        IPlayerNameHolder.Instance = this;
    }
}
```

2. Copiable Interface

The copiable interface is an interface used by the dialogue editor, but of course it can be used elsewhere. If you want an object to be copied, inherit this interface.

The copiable interface is `ICopiable`, its member is as follows:

Members For Runtime	Explanation
Method: object Copy();	Returns the copied object of itself

3. Fade Able Interface

The fade able interface is generally used on UI components, such as “Window” type, it uses this interface to perform fade in/out operation on `CanvasGroup`.

The fade able interface is `IFadeAble`, its members are as follows:

Members For Runtime	Explanation
Property: <code>MonoBehaviour</code>	“ <code>MonoBehaviour</code> ” used to perform fade in/out coroutine
Property: <code>FadeTarget</code>	“ <code>CanvasGroup</code> ” used to perform fade in/out
Property: <code>FadeCoroutine</code>	fade in/out coroutine
Static Method: void <code>FadeTo</code> (<code>IFadeAble</code> , float, float, Action)	fade in/out method

4. Scene Loader Interface

The scene loader interface is used to load scene in the save system, because this plugin is not sure how the user should implement the scene loading, such as whether there is a screen fade in/out animation when switching scenes, whether there is a progress bar, etc., so this interface is used instead of a specific scene loader.

The scene loader interface is `ISceneLoader`, its members are as follows:

Members For Runtime	Explanation
Static Property: Instance	A singleton of scene loader
Method: void LoadScene(string, Action)	Loads the scene with the specified name and executes the passed in callback when loading is complete

The usage is similar to the player name holder interface, nothing more than the implementation of property becomes the implementation of method.

5. Message Displayer Interface

The message displayer interface is used to display messages that are send from this plugin, the reason why use it and usage method is similar to above.

The message displayer interface is `IMessageDisplayer`, its members are as follows:

Members For Runtime	Explanation
Static Property: Instance	A singleton of message displayer
Method: void Push(string)	Push the given message to the message displayer

Attributes

1. Initialization Attribute

The types with the initialization attribute will call the static initialization method with the specified name when loading save data, and of course it can also be called in other places that need to initialize. Adding this attribute to a specific type provides better performance than adding the initialization method attribute to a specific static method.

The type of initialization attribute is `InitAttribute`, inherit from `Attribute`, its members are as follows:

Members For Runtime	Explanation
Field: method	Name of the static initialization method, that method should be a parameter-less method or its parameter is optional
Field: priority	Invoke priority
Method: void InitAll()	Call all initialization methods of types with this attribute in order of priority

2. Initialization Method Attribute

The methods with the initialization method attribute will be called when loading save data, and of course it can also be called in other places that need to initialize. Adding this attribute to the initialization method is more convenient than adding the initialization attribute to a specific type, you don't need to pass in the method name, and you don't have to worry about forgetting to update the method name in the initialization attribute after the initialization

method is renamed, but the call performance is even worse.

The type of initialization method attribute is `InitMethodAttribute`, inherit from `Attribute`, its members are as follows:

Members For Runtime	Explanation
Field: <code>priority</code>	Invoke priority
Method: <code>void InitAll()</code>	Call all methods with this attribute in order of priority