Node-Based Dialogue Creator

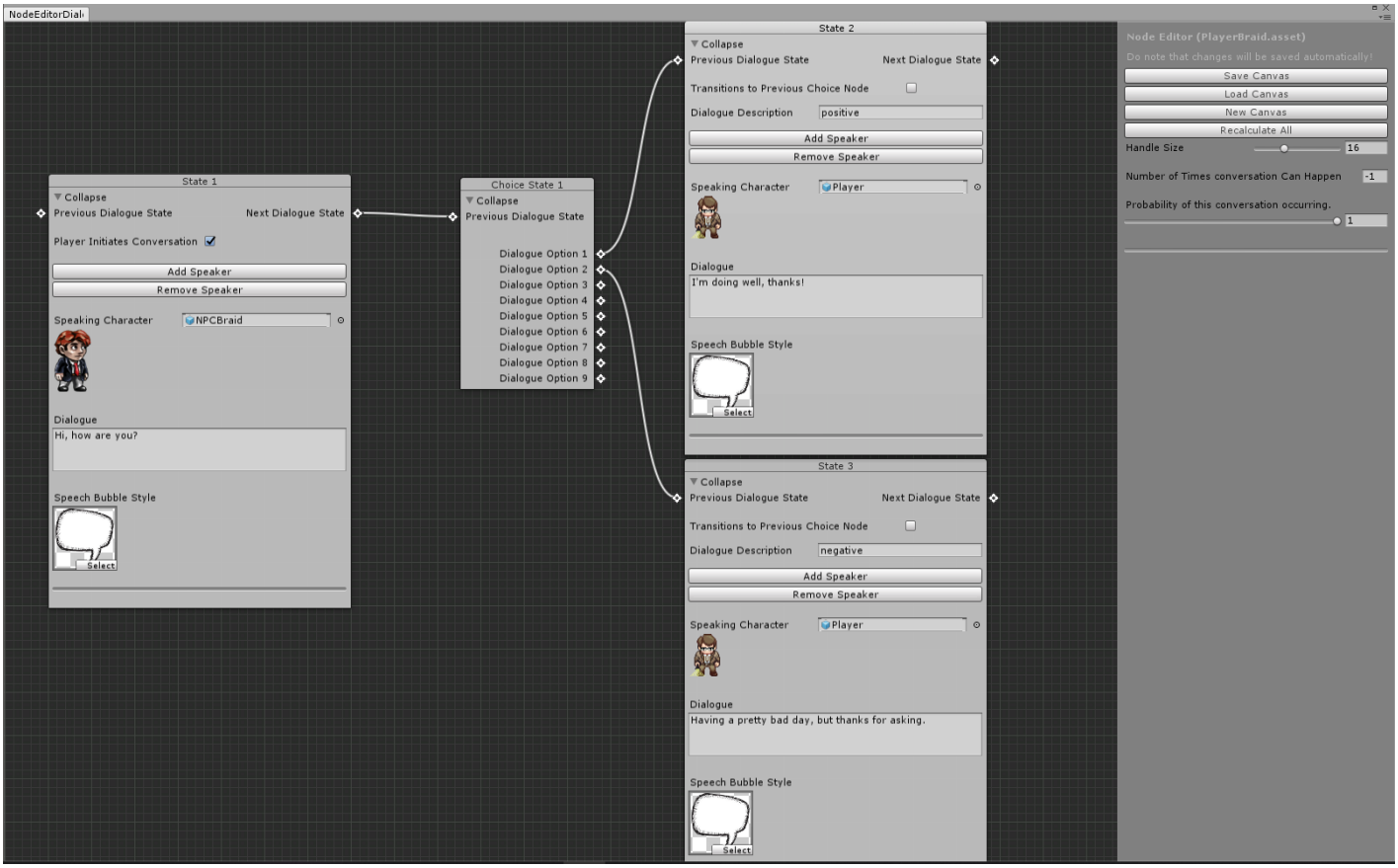
Created by Mark Philipp

Last Updated 7/5/2018

Contents

Introduction

This asset is an extension for the Unity Editor. It provides a custom editor window that allows for easy creation of dialogue interactions by using finite state machines (FSM). With this asset, you'll be able to create dynamic conversations with branching dialogue choices between multiple characters, including players and NPCs. Best of all, you won't have to touch any code unless you wish to extend the asset for your own purposes. This document will outline all the details you need to know to use the tool.



Acknowledgements

Node Editor Framework

This foundation that this editor was built on came from Unity Forum user Seneral's Node_Editor_Framework. Seneral's framework was largely built from an opensource framework that was created in a Unity forum thread, which was begun by Unity user unimechanic. I want it to be known that this asset could not have been created without the help of the many Unity forum users who helped in the development of the base framework.

Tuple Implementation

Tuples were used extensively for handling the finite state machine transition functions. To my surprise, C# does not natively contain a Tuple implementation, so I made use of Michael Bartnett's Tuple implementation.

Messaging System

For the implementation of the Observer design pattern, I used Magnus Wolffelt and Julie Iaccarino's CSharpMessenger Extended.

Graphics

All the sprites and animations that are used in the demo scene were taken from the internet. The player sprite sheet was found on File Army from user Constantinople2u.

The 2D Tracer sprite was created by Abysswolf (Daniel Oliver), who makes some awesome 2D art. I recommend viewing more of his work.

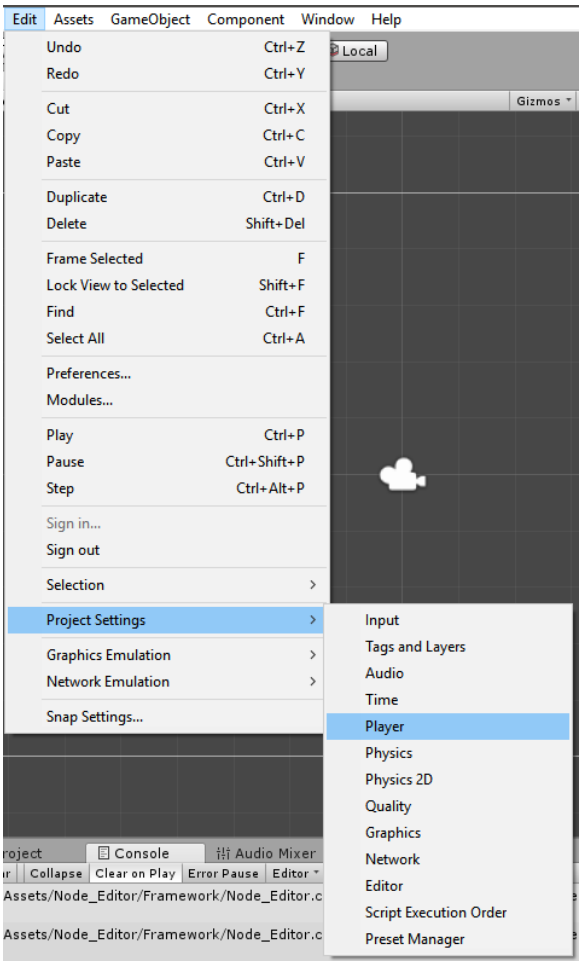
I cannot find an author for the Link sprite. I found the sprite sheet on kisspng.

Megaman and Braid are both taken from Pinterest. Of course, Megaman is a Capcom IP and Braid is the creation of Jonathon Blow and David Hellman.

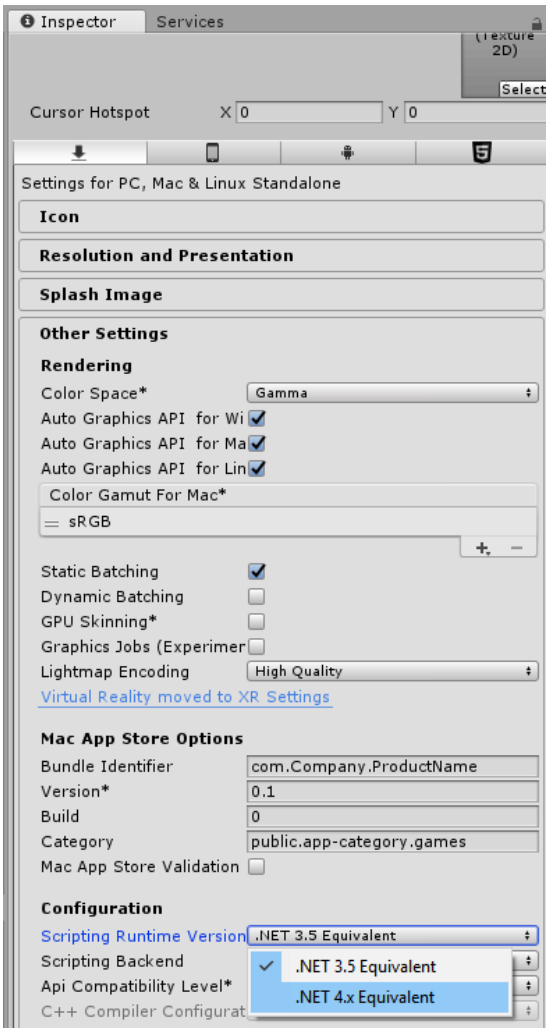
Setting up your Compiler

This asset requires .NET 4.0 features. These features can be enabled in Unity by following this procedure:

In your Unity project, go to **Edit →Project Settings →Player**.



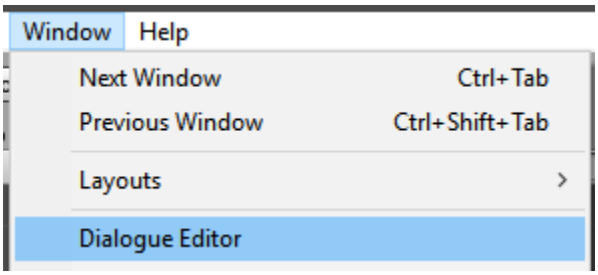
In the inspector, on the right, scroll down to the **Configuration** section. In the **Scripting Runtime Version** section, choose **.NET 4.X Equivalent.**



Choose to restart the editor and you should be good to go.

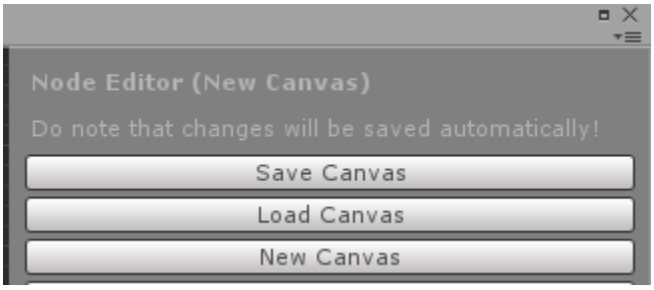
How to Use the Editor

Using the editor is quite simple. Make sure you've imported all the necessary files into your Unity project. Once the files are imported, simply navigate to the menu bar in Unity and choose "Window" → "Dialogue Editor".



This will open the editor window.

Save/Load Features

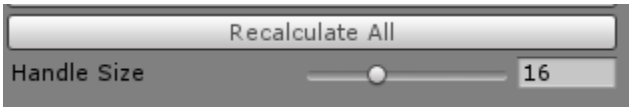


Save Canvas: Opens an explorer window, allowing you to choose a location to save the Dialogue asset to. The default file path is **Assets/Resources/Dialogue**. I highly recommend saving your files in this directory to ensure proper functioning. This file path can be changed, however, inside DialogueLoader.cs and NodeEditorDialogue.cs.

Load Canvas: Opens an explorer window, allowing you to choose a Dialogue asset file to load for editing. The default file path is **Assets/Resources/Dialogue**. However, you can load a Dialogue asset that is located outside of this folder too. Upon loading, the node editor will populate the window with all the data from the Dialogue asset file.

New Canvas: Clears the editor window and creates a new canvas for you to work with.

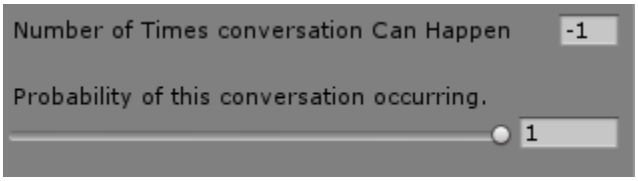
Miscellaneous Features



Recalculate All: Forces a Calculate() function call on all the nodes in the scene. You will likely not use this very much. It is an artifact that remains from the original Node Editor framework that was pulled from Seneral's implementation.

Handle Size: Changing this value will change the size of the Input and Output circles that show up on the left and right sides of a node. The higher the value, the larger the circles and vice versa.

Conversation Options



Number of Times Conversation Can Happen: This value represents the number of times this specific dialogue can occur in the game. The default value of -1 is used to represent **infinity.** If this value is set to -1, then this dialogue can occur an unlimited number of times in the game. If this value is set to 1, for instance, then when you access this dialogue in the game one time, you will never be able to access it again.

Probability of this conversation occurring: This slider represents a value between 0.0 and 1.0 where 0 = 0% and 1.0 = 100%. This feature allows you to create conversations that are rare, common, or somewhere in between. For example, let's say you have 3 different conversations that the player can have with a specific NPC. Which conversation occurs when talking to the NPC depends on the probability of each individual conversation. A value of 1.0 ensures that this conversation will always happen.

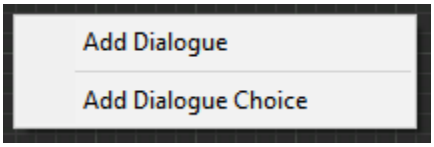
A note about the probability. The implementation is done using independent probabilities. This means that the other conversations with the same participants involved do not effect the probability of another conversation with the same participants. This means if a conversation has a 30% chance of occurring (0.3), then it has a 30% chance regardless of any of conversation's probabilities.

How to Use the Nodes

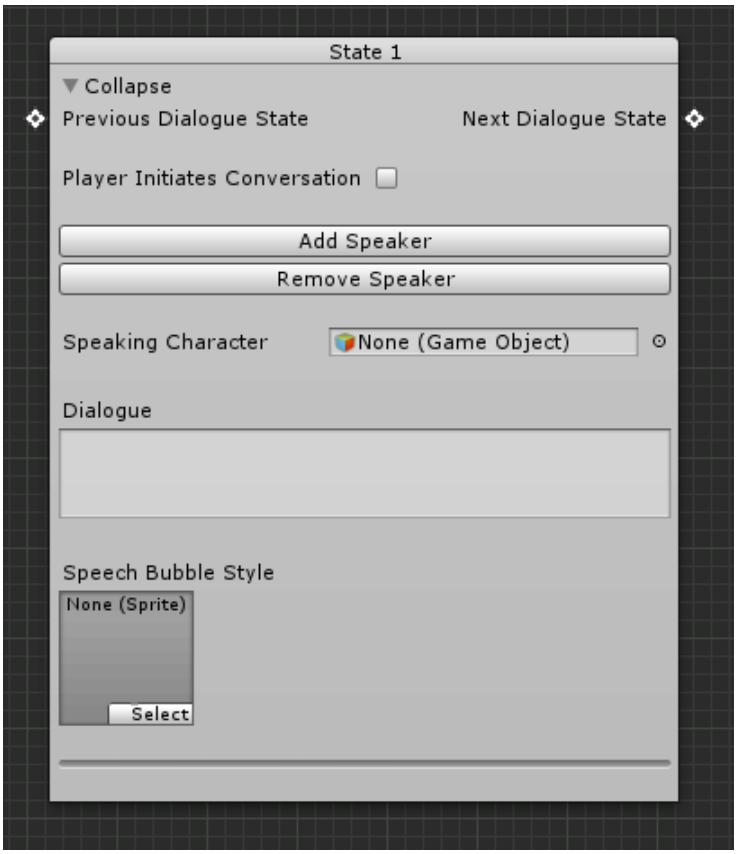
Before diving into this section, note that when I refer to the "node editor", I am referring to the left-hand side of the editor window. This is the side with the blackish-grey grid pattern in the background. The node editor is where you place nodes and connect them together.

Placing New Nodes

To place a new node, right-click an empty spot in the node editor and choose the type of node you wish to create.



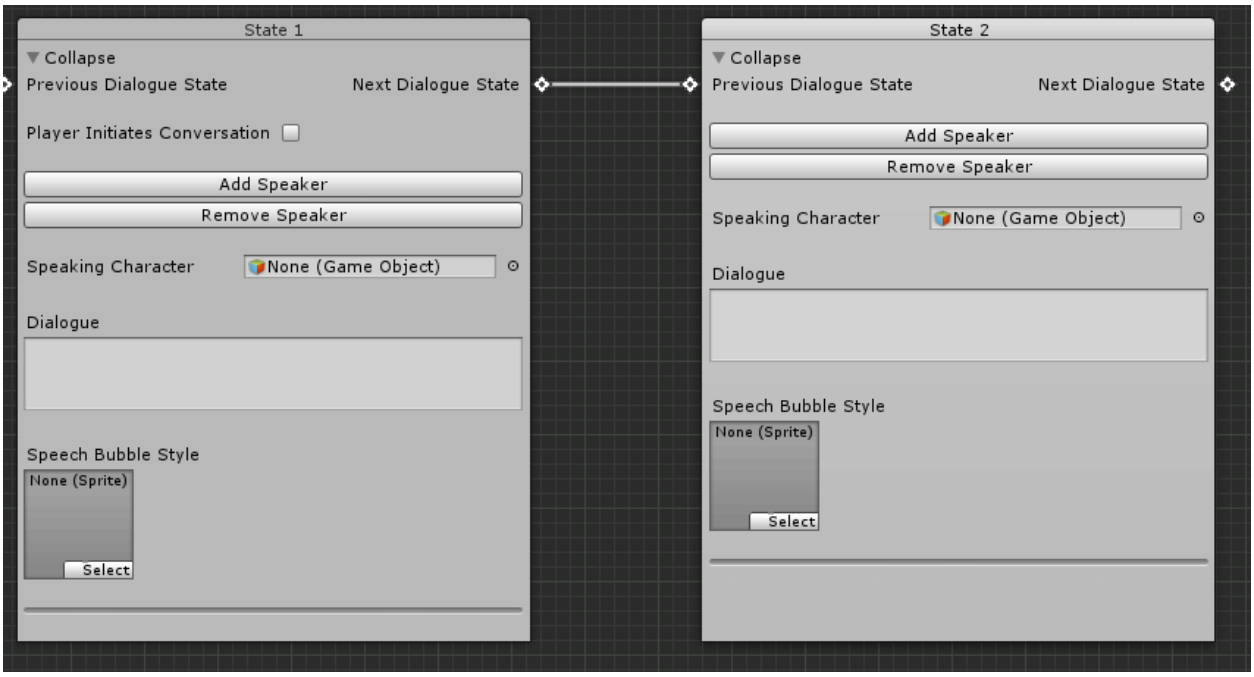
Dialogue Node



Collapse: Clicking the triangle next to the "Collapse" text will collapse or de-collapse the node. Collapsing a node is a good way to clean up your workspace.

Previous Dialogue State: This is a NodeInput. A different node connects its "Next Dialogue State" output into here. Connecting a node into this input tells the FSM that there is a state that flows into this state. If a node has a node connected to the input, then the node CANNOT be a start state.

Next Dialogue State: This is a NodeOutput. The user can click and drag on the diamond next to the text to create a Bezier curve and connect it to another node's **Previous Dialogue State**. This tells the FSM that this node transitions to a different node.



Player Initiates Conversation: This option only exists on start states. Selecting this option will tell the FSM that this conversation is started by a player character and NOT by an NPC. By leaving this unchecked, you can create NPC-to-NPC conversations or you can create a conversation with a player where the NPC begins the conversation when they are in range of the player.

Add Speaker: This button adds another speaker to this state. Having multiple speakers on a single state means that multiple characters will deliver a piece of dialogue simultaneously when this state is accessed in the conversation. Each speaker can have their own speech bubble and dialogue. **If you have multiple speakers in a conversation, then all the speakers in a conversation must be close enough to each other for the conversation to activate**. If any of the speakers are not within circular trigger collider proximity of the other speakers, then the conversation will not happen.

Remove Speaker: Pressing this button will remove the bottom-most speaker from the list of speakers. Using the screenshot above as an example, if you pressed “Remove Speaker”, then Megaman and his data would be removed from this state as a Speaker, leaving only Braid and his data.

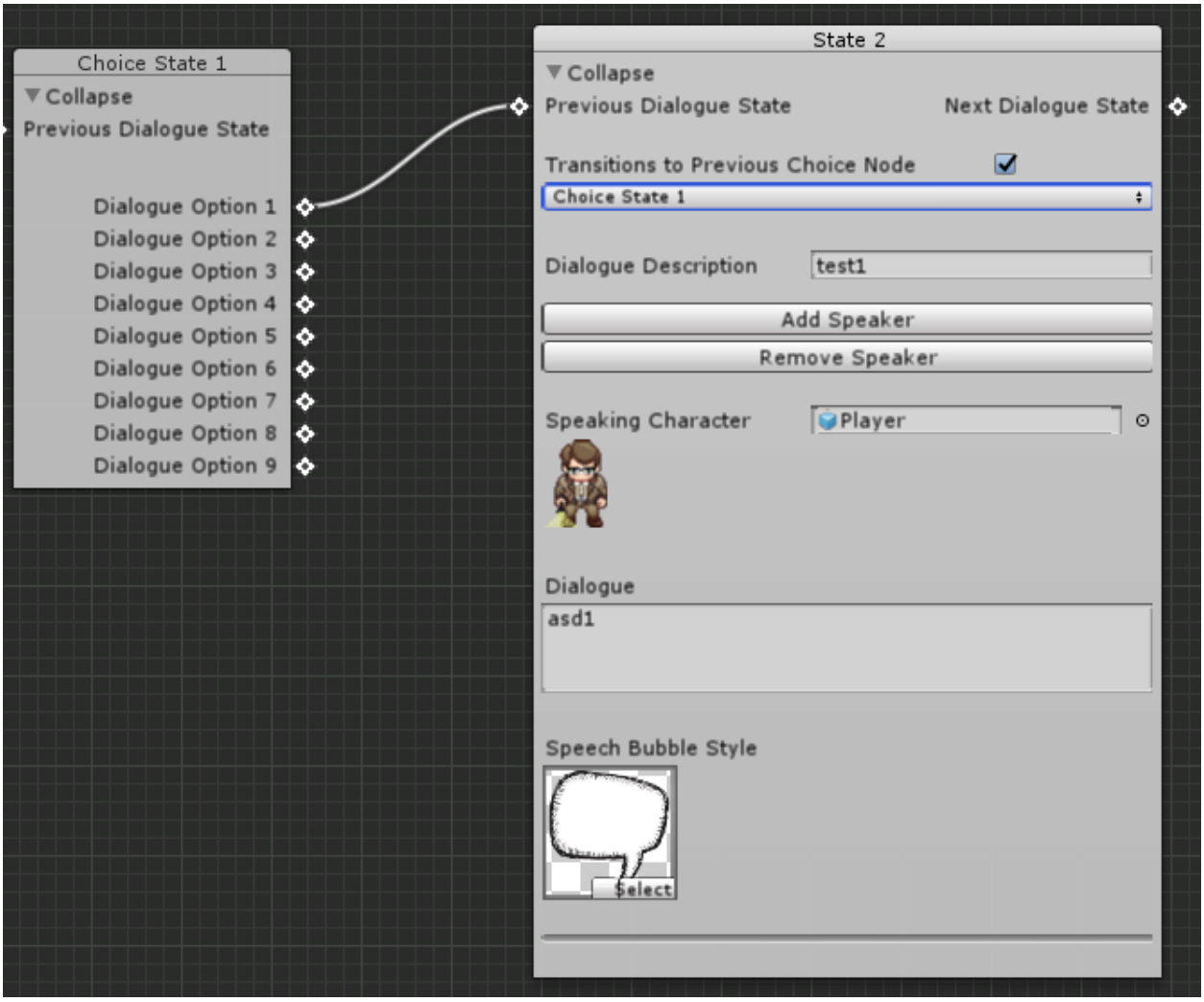
Speaking Character: This field allows you to select a prefab GameObject in your project, which specifies that object as the speaker of this specific dialogue. The GameObject MUST be a prefab or else you will not see it in the list of valid GameObjects to select from.

Dialogue: This is the dialogue that the character will say when this state is accessed. This is where your storytelling occurs.

Speech Bubble Style: This object field allows you to select a style for the speech bubble, which displays the associated dialogue. Be sure to select the whole speech bubble sprite and not the body or tail sprites. Refer to the Speech Bubbles section for further details on how to properly use this feature.

Dialogue Nodes That Come from a Choice Node

Dialogue nodes that have an input coming from a choice node will have some context-specific additional options that a normal dialogue node does not have. Note that the speaking character is automatically set to the player on these nodes. This is because we expect the player to be the only character that can make a dialogue choice.

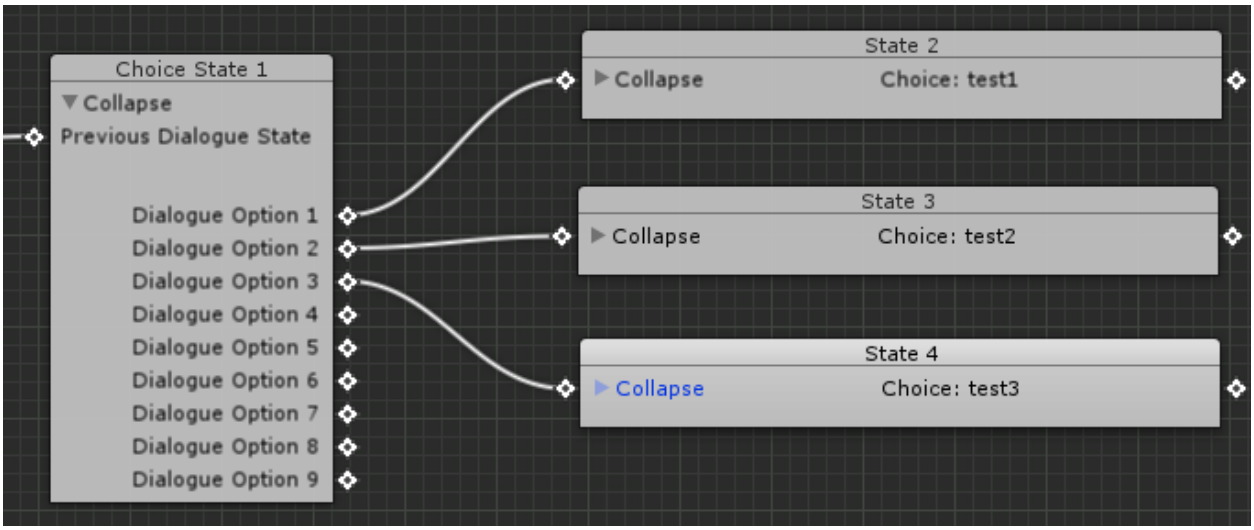


Transitions to Previous Choice Node: Selecting this option tells this state to transition back to a choice state. The choice state can be selected by the dropdown menu below the toggle. The popup menu only shows up when the toggle is selected. The name of each choice state can be seen at the top of the choice node. Note that this toggle only shows up on nodes that do not have any outputs AND is part of a branch that contains a dialogue choice node.

Dialogue Description: This is a description of this choice. The description is what shows up during user input. For instance, if the player is given a choice to make and choice 1 is described as “Be polite”, then when the player is asked for an input, they will see “1. Be polite” as a choice.

Dialogue Choice Nodes

Dialogue choice nodes allow you to create branching dialogue in your conversations. As of this writing, the dialogue choice node only allows a maximum of 9 branches per choice node. Each option corresponds with the button input required to make that choice during a conversation. For instance, dialogue option 1 will be associated with the alphanumeric 1 key on the keyboard. Option 2 is associated with the 2 key, etc.



Collapse: Just like with Dialogue Nodes, clicking the collapse triangle will make the node compact. You will no longer see the text on the node and it will clear up some space in the editor. Clicking the triangle on a node that is collapsed will open the node up again.

Previous Dialogue State: This is the input for the choice state. Whatever node connects to this input is the node that comes immediately before the choice state in the conversation.

Dialogue option 1 – 9: Each dialogue option is a branch that can be connected to the inputs of other nodes.

Speech Bubbles

Speech bubbles are the way that dialogue is delivered in-game. Their behavior is handled by the **DialogueBox.cs** script. They are instantiated in the scene using the **DialogueBox.prefab**, which can be found in **Assets/Resources/Prefabs/Dialogue.** Note that these prefabs are considered UI elements, therefore they require your scene to contain the **DialogueCanvas.prefab** object, which is found in **Assets/Resources/Prefabs/Dialogue**.

Dialogue boxes are instantiated onto a character's child GameObject called "DialoguePosition". DialoguePosition is simply a transform that you position around the character. Wherever you position is where the dialogue box's center point will be upon instantiation. Note that any speaking character MUST have this transform as a child object or else **DialogueBox.cs** will not know where to instantiate to.

Speech Bubble Graphics

The speech bubble graphics are probably the most awkward part of this asset to setup. The graphics are just png files. However, the png files must be split up into a full image, a body, and a tail segment. To do this, you'll want to use an image editor (I use Paint.net, which is a simple editor), and crop the body from the tail and save each part as separate png files. However, you must keep the original file intact. Ultimately, you'll have 3 different png files for your speech bubble: the whole image, the body, and the tail.

When selecting a speech bubble style in the node editor, be sure to choose the whole sprite and not the body or tail sprites. The code on the backend will know what to do with the whole sprite file and will be able to load the body and fails separately at in-game.

Each item must follow a specific naming convention as well. For instance, say we have a png file called ExclamationDialogue.png, which represents the whole image. The body must be named ExclamationDialogueBody.png and the tail must be ExclamationDialogueTail.png.

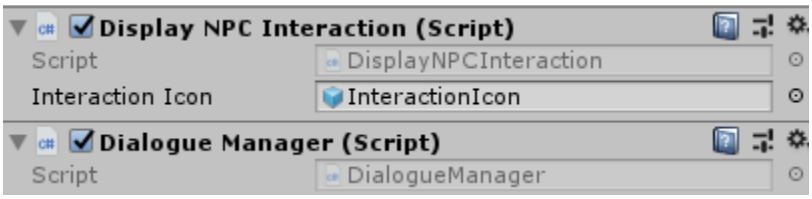
Also note that all items should match the same width dimensions as the base image. This is not required, but it is highly recommended to ensure the sprite scaling looks proper in-game. The height dimensions can be modified. In fact, the height dimensions must be modified since you must crop the body and tail into separate images.

The reason the speech bubble is cropped into separate segments is, so the body can be scaled upwards with the text while allowing the tail to remain static.

I understand that this is a weird way of implementing the sprites for the speech bubbles. This is a high on the list of priorities for refactoring so that it is easier to use.

Setting Up Speaker Prefabs

Characters that will be speaking in the game must have certain components attached to them. The main components required are **DialogueManager.cs** and **DisplayNPCInteraction.cs**. However, **DisplayNPCInteraction.cs** is optional and is only required for the demo scene to work properly. You can implement your own interaction behavior if you prefer. If you do so, I'd recommend referring to the **DisplayNPCInteraction.cs** file to see how the dialogue is accessed.



Additionally, speakers must have a child GameObject called "DialoguePosition". This GameObject is nothing but a Transform and is used for getting Vector coordinates for instantiating the dialogue box for the speaker.

NPC Speakers must be tagged as "Character" and the Player must be tagged as "Player". If these tags are not in place, **DialogueLoader.cs** will not work properly.

I highly recommend referring to the **DialogueTest.unity** scene to see how these prefabs are implemented.

Required Prefabs for the Scene

Your scene must contain the **ConversationHandler, DialogueLoader, and DialogueCanvas** prefabs. These prefabs can be found in **Assets/Resources/Dialogue.**

Additionally, it's recommended that speaking characters are included in the scene at run-time and that their GameObject is active. The DialogueLoader prefab runs when the scene starts and parses all the dialogue once and then deletes itself. This means that if some of your speakers are not in the scene and active when the scene starts, then any dialogue that includes them as a speaker will not be loaded.