Node-Based Quest System

Created by Mark Philipp

Mphilipp622@gmail.com

Last Updated 8/18/2018

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.
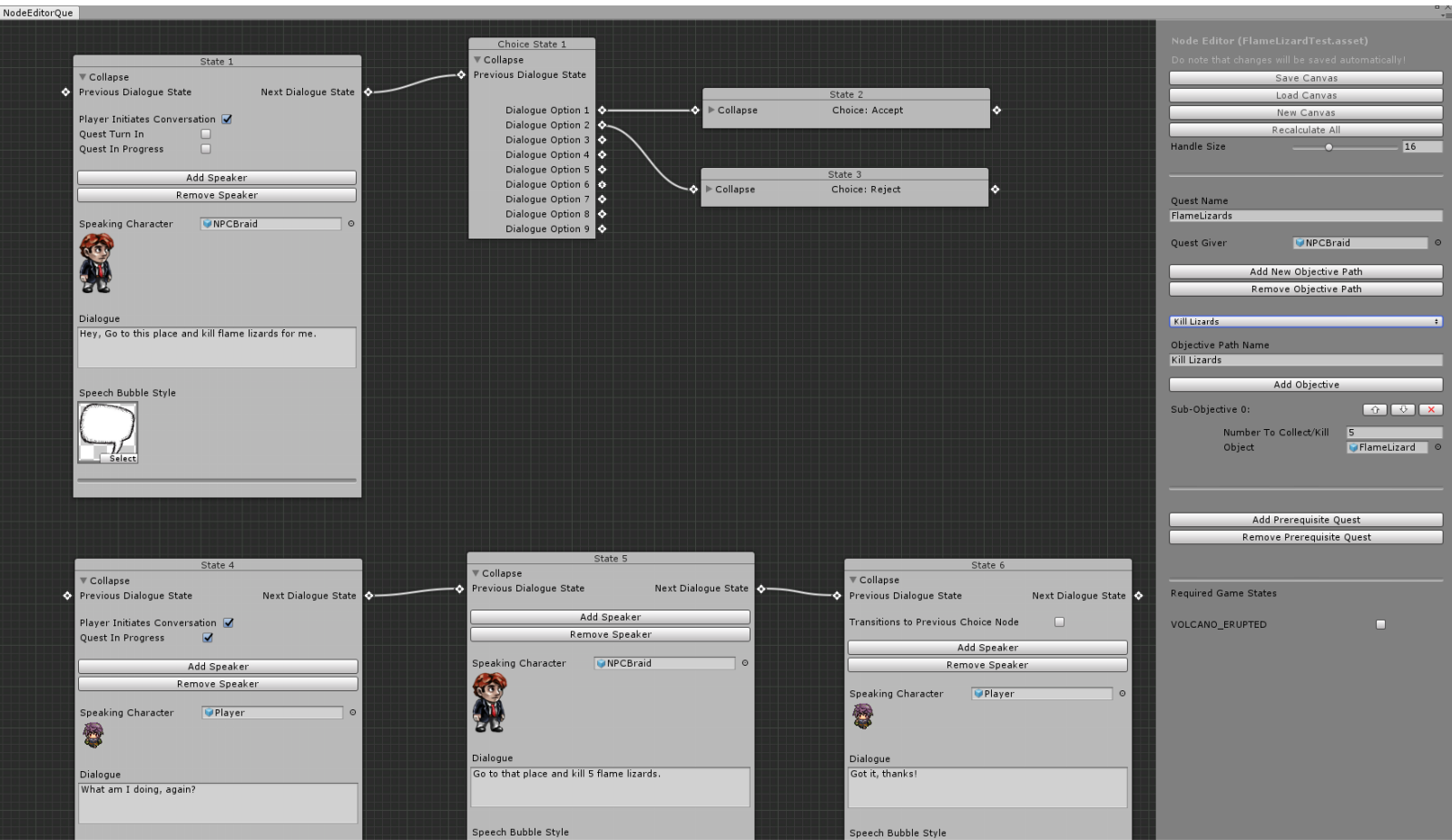
Acknowledgements

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.

I cannot find the website I got the purple-hair character from. I would like to give proper credit. If anybody reads this and knows who the artist was, please email me and let me know and I will update this document accordingly.

Setting up your Project

This section will outline project-specific settings that you'll need for this asset to work properly.

Setting up the 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**.
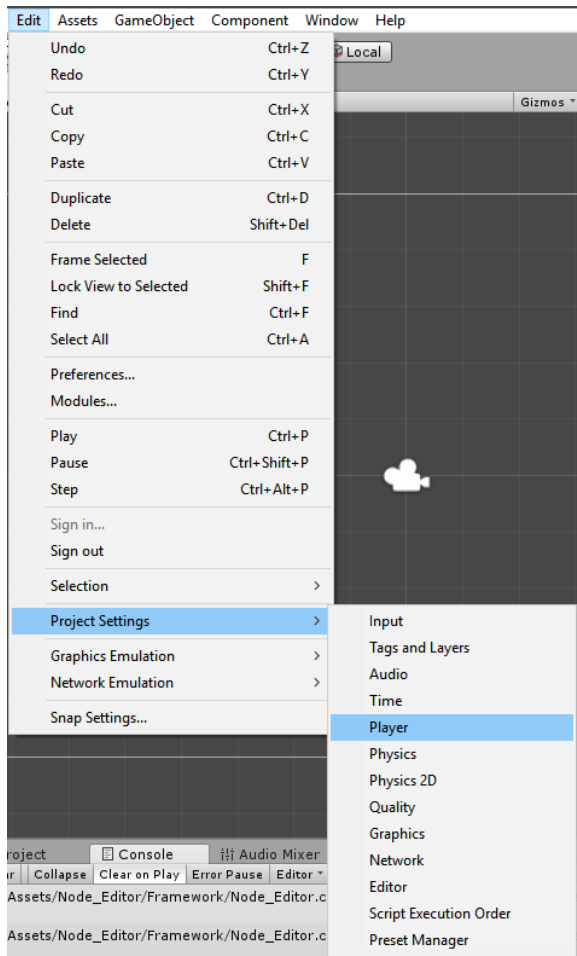
In the inspector, on the right, scroll down to the **Configuration** section. In the **Scripting Runtime Version** section, choose **.NET 4.X Equivalent.**
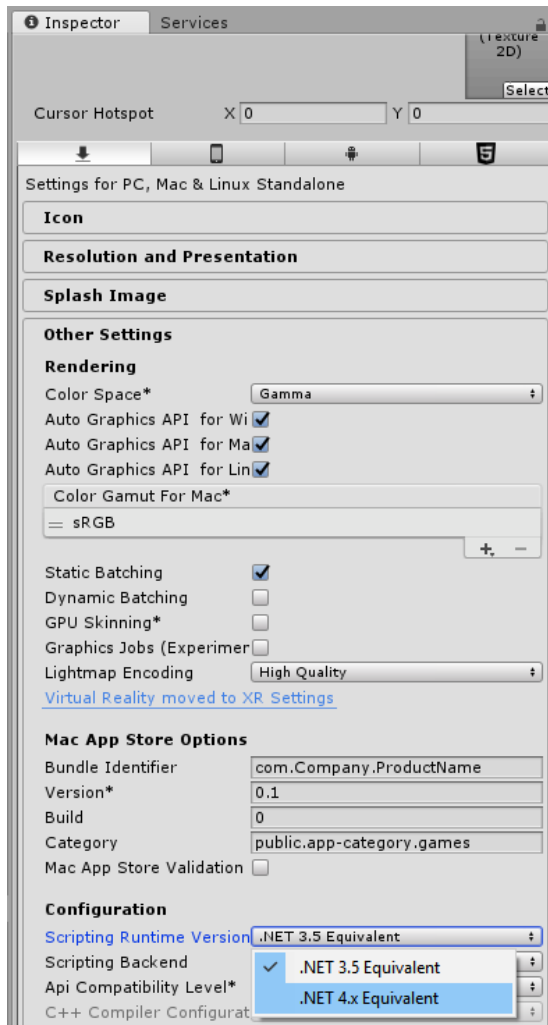
Choose to restart the editor and you should be good to go.

Setting up the Input Manager

Several custom inputs are required for the code to work properly. The following must be added under **Edit -> Project Settings -> Input**:

- **InteractDialogue**
- **InteractQuest**
- **InteractTurnInQuest**
- **NextLine**
- **DialogueSelect1**
- **DialogueSelect2**
- **DialogueSelect3**
- **DialogueSelect4**
- **DialogueSelect5**
- **DialogueSelect6**
- **DialogueSelect7**
- **DialogueSelect8**

- **DialogueSelect9**

These keys can be assigned to whatever you want, but the names must exist in the input manager for the code to run without error. In my project, DialogueSelect 1 – 9 are all assigned to the 1 – 9 keys, respectively.

Setting up Tags

The code heavily relies on specific tags existing. I understand this can be a pain to incorporate into existing projects and I hope to work on a better implementation in the future.

The following tags must exist:

- **Player**
- **Character**
- **Enemy**
- **Location**

The list of tags would likely increase as more diverse quest objectives are added, such as Items. The player must be assigned to the "Player" tag. Any character that speaks or gives a quest must be tagged as "Character". Any enemy that will be part of a quest must be tagged with "Enemy". Best practice would be to tag any enemy as "Enemy" regardless of their usage in a quest. Any Game Object that serves as a location for a quest must be tagged with "Location".

Video Documentation

You can watch a video tutorial [here](). I recommend this tutorial as a good introduction for using the asset.

How to Use the Quest Editor

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 "Quest" → "Quest Editor".
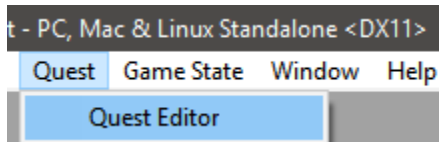
This will open the editor window.

Save/Load Features
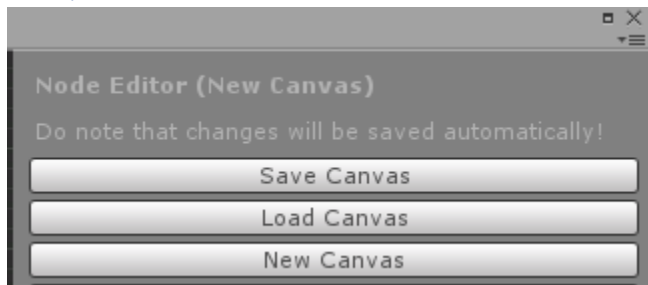
Load Canvas

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
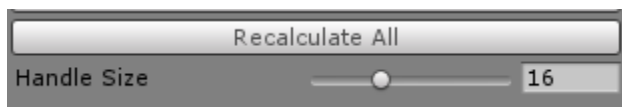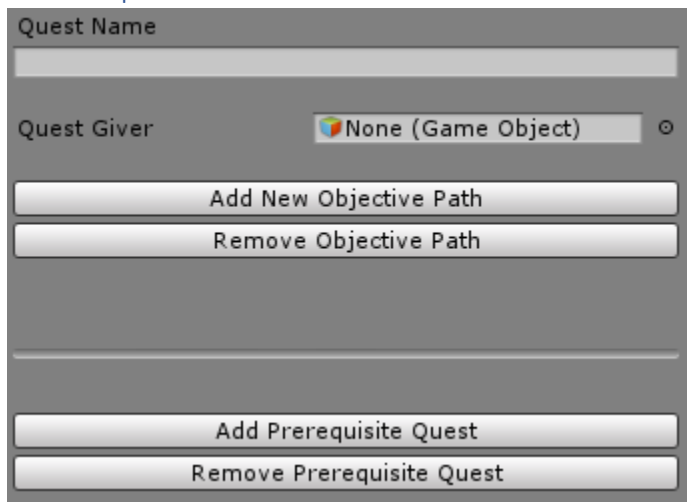
Recalculate All

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.

Quest Options
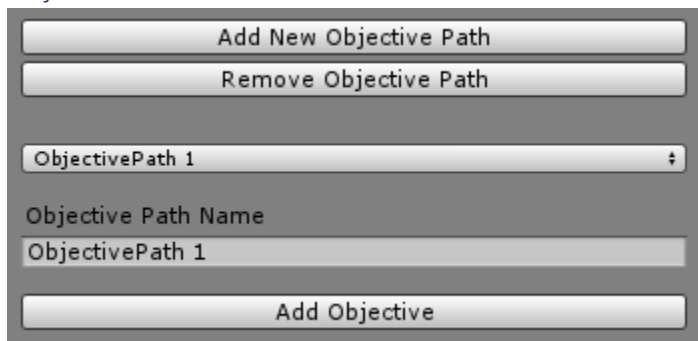
Quest Name

This is the name of the quest. This variable can be accessed in C# using Quest.GetQuestName() where "Quest" is the specific quest you wish to access, meaning this function is not static.

Quest Giver

This is the character game object that will give this quest to the player. **This game object must be a prefab that exists in your project**. You can drag and drop the prefab into the field or click the circle icon next to the field to select a prefab from your project.

Objective Paths

In this system, an "Objective Path" refers to a path that contains one or more quest objectives. A single quest may have one or more objective paths. The motivation for this system was so a quest could have concurrent paths that contain their own objectives. In this way, you can create questlines that aren't necessarily linear. This was largely inspired by how the Witcher 3 handled quests.

Add New Objective Path

This option will add a new objective path to this quest. Upon clicking this button, a dropdown menu and a new button will appear. The **Objective Path** dropdown allows you to choose which objective path you are currently modifying. If you have multiple objective paths, they will show up in the list of objective paths.

This is the name of the objective path. This name has no use outside of the editor. It is only there so the user can keep track of what path they are modifying. This is for your convenience only. None of the backend code relies on this name.

Add Objective

Clicking this button will add a new objective to the currently selected objective path. These objectives are listed as "sub-objectives" since they are part of an objective path.

In C#, the objective paths are accessed in the Quest.objectives private member variable. This variable is a 2D List (a matrix), where the outer list represents an objective path, and the inner lists represent the objectives within that path. For example, say I have two objective paths named ObjectivePath 1 and ObjectivePath 2. If I access Quest.objectives[0][2], then I will be getting the third objective in ObjectivePath 1.

Sub Objectives
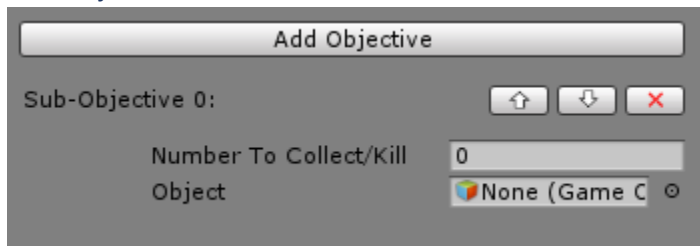
Up/Down/Delete

Use these buttons to change the order of your objectives or to remove them from the objective path. **The order these objective determines the order in which the objectives must be completed.** This means that each objective path has linear objective requirements. **E.G:** Sub-Objective 0 says to kill 5 enemies and Sub-Objective 1 says to travel to a location. The player must first kill the 5 enemies before the quest will recognize they have travelled to the location in sub-objective 1.

Number to Collect/Kill

This field allows you to specify how many of an item should be collected or how many of an enemy should be killed. **This field only matters for collection/grinding quests**. If you want the player to go to a location, then you can leave this field at "0" and specify a location prefab in the "Object" field.

Object

This field allows you to specify a prefab for the quest objective. Typically, you will want to put an enemy, item, or a transform that represents a location. As mentioned above, if you put a location in this field, you can leave the "Number to collect/kill" field at 0. If you put an item or an enemy in this field, you will want to ensure you have some number specified in "Number to collect/kill" field.

Prerequisite Quests
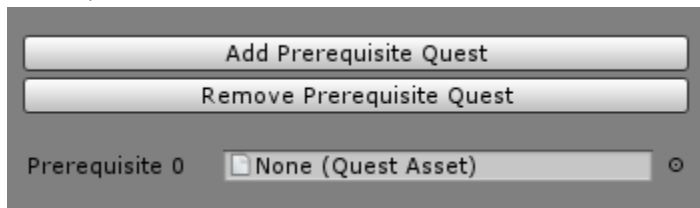
Prereq quests allow you to specify which quests must be completed before the player has access to this quest.

Add Prerequisite Quest

This button will add a new prerequisite quest. You can add just about any number of prerequisite quests.

Remove Prerequisite Quest

This button will remove the prerequisite quest **at the bottom of the list**. I hope to update this feature in the future to include the ability to delete a specific prerequisite quest instead of only popping off the back of the list.

Prerequisite Field

This field allows you to drag and drop a quest asset or use the circle on the right to select a quest from the project. Quest Assets are the files that are created by the quest editor system. When you create a quest and save the canvas, one of these assets will be created. These assets are what you drag and drop into this field.

Required Game States:

Game States are discussed elsewhere in this document. The point of this feature is to allow the user to restrict access to a quest unless a specific combination of game states is true. For instance, maybe you only want the player to get this quest after a volcano eruption has devastated a nearby village. You can put a checkmark next to "VOLCANO_ERUPTED" to tell this quest to only be available when that game state has been set.

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. The dialogue system has been documented in the Node-Based Dialogue System documentation, which is included in the unity package, but some of the details will be repeated here.

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.
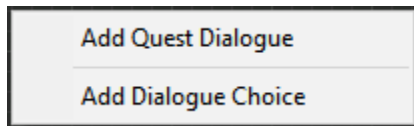
Quest Dialogue Node
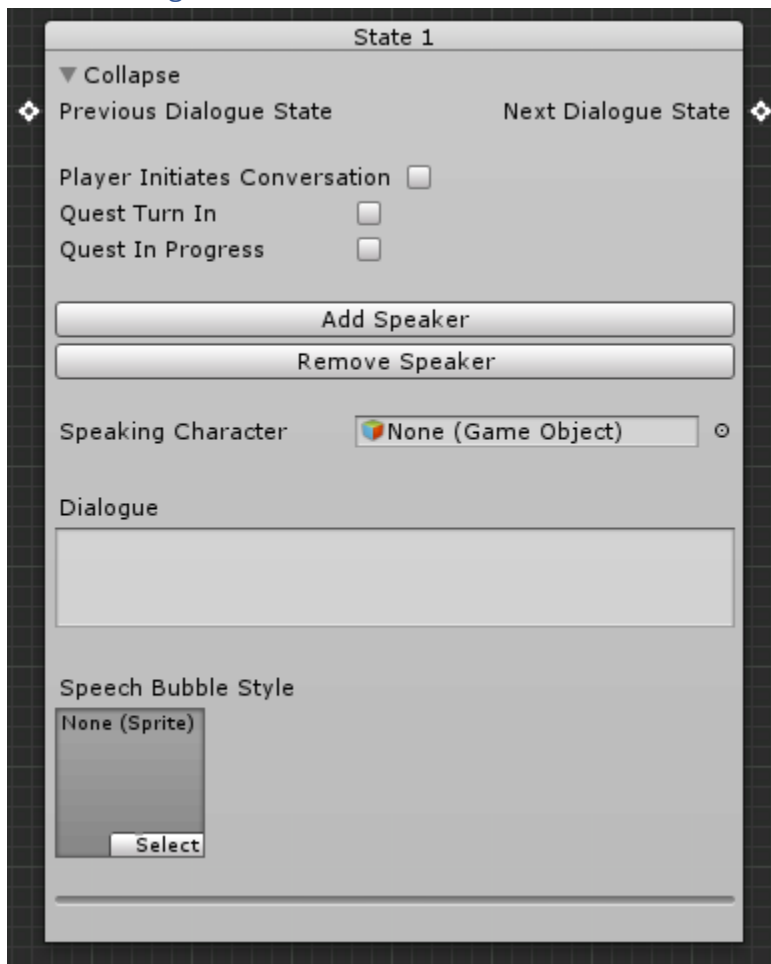
Collapse

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.
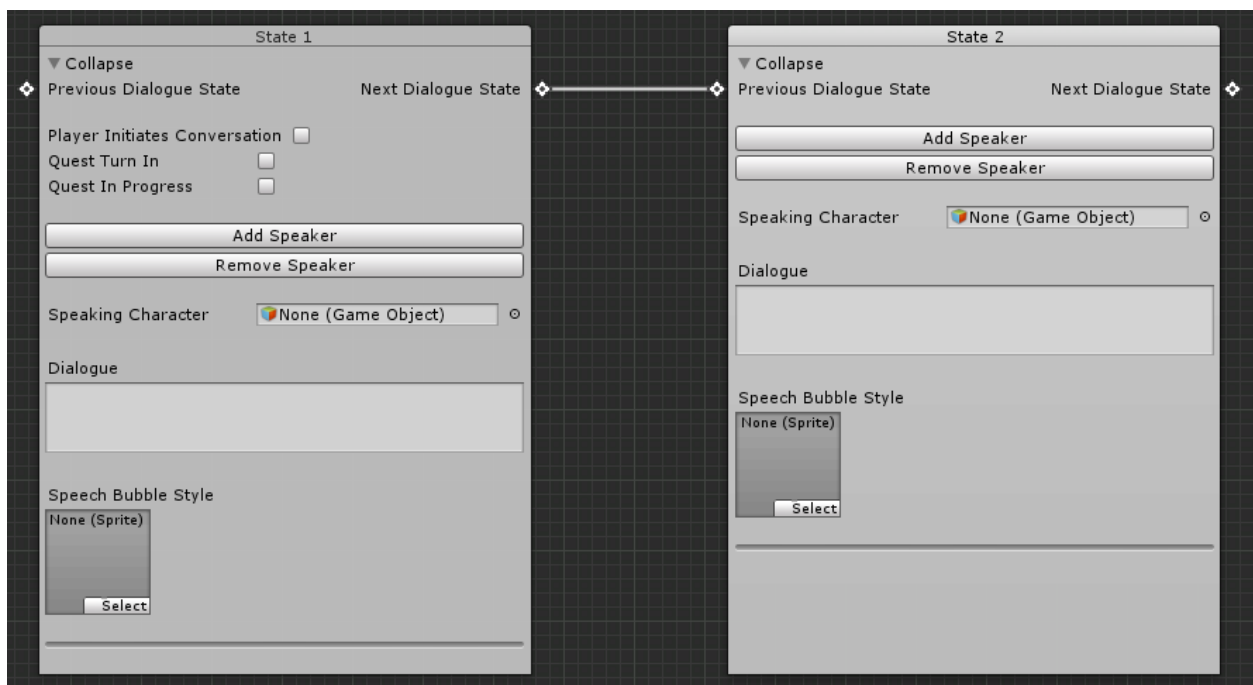
Player Initiates Conversation

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.

Quest Turn In

This option will only show up on start states. Checking this box specifies that this specific dialogue will occur when turning the quest in.

Quest In Progress

This option will only show up on start states. Checking this box specifies that this specific dialogue will occur when speaking to the quest **giver after you have accepted the quest but before you have finished the quest.**

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.

Best Practice for Quest Dialogue

Generally, most questlines in games have dialogue related to the following:

- Being told what the quest is and allowing you the option to select or decline the quest
- Dialogue between quest giver and player when player speaks to quest giver after accepting the quest and before turning the quest in
- Dialogue exchange between quest giver and player when player turns in the quest

selecting the "Quest Turn In" and "Quest In Progress" options accordingly. See this screenshot for an example:

State 1

▼ Collapse
Previous Dialogue State   Next Dialogue State

Player Initiates Conversation ☑
Quest Turn In ☐
Quest In Progress ☐

Add Speaker
Remove Speaker

Speaking Character   NPCBraid ⊙

Dialogue
Hey, Go to this place and kill flame lizards for me.

Speech Bubble Style

Select

Choice State 1

▼ Collapse
Previous Dialogue State

Dialogue Option 1
Dialogue Option 2
Dialogue Option 3
Dialogue Option 4
Dialogue Option 5
Dialogue Option 6
Dialogue Option 7
Dialogue Option 8
Dialogue Option 9

State 2

▶ Collapse   Choice: Accept

State 3

▶ Collapse   Choice: Reject

State 4

▼ Collapse
Previous Dialogue State   Next Dialogue State

Player Initiates Conversation ☑
Quest In Progress ☑

Add Speaker
Remove Speaker

Speaking Character   Player ⊙

Dialogue
What am I doing, again?

Speech Bubble Style

Select

State 5

▼ Collapse
Previous Dialogue State   Next Dialogue State

Add Speaker
Remove Speaker

Speaking Character   NPCBraid ⊙

Dialogue
Go to that place and kill 5 flame lizards.

Speech Bubble Style

Select

State 6

▼ Collapse
Previous Dialogue State   Next Dialogue State

Transitions to Previous Choice Node ☐

Add Speaker
Remove Speaker

Speaking Character   Player ⊙

Dialogue
Got it, thanks!

Speech Bubble Style

Select

State 7

▼ Collapse
Previous Dialogue State   Next Dialogue State

Player Initiates Conversation ☑
Quest Turn In ☑

Add Speaker
Remove Speaker

Speaking Character   Player ⊙

Dialogue
Job's done.

Speech Bubble Style

Select

State 8

▼ Collapse
Previous Dialogue State   Next Dialogue State

Add Speaker
Remove Speaker

Speaking Character   NPCBraid ⊙

Dialogue
Thank you! Those lizards were a pain in the ass.

Speech Bubble Style

Select

State 9

▼ Collapse
Previous Dialogue State   Next Dialogue State

Transitions to Previous Choice Node ☐

Add Speaker
Remove Speaker

Speaking Character   Player ⊙

Dialogue
Anytime, my dude.

Speech Bubble Style

Select

Quest Nodes That Come from a Choice Node

Quest nodes that have an input coming from a choice node will have some context-specific additional options that a normal Quest 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 or quest choice.
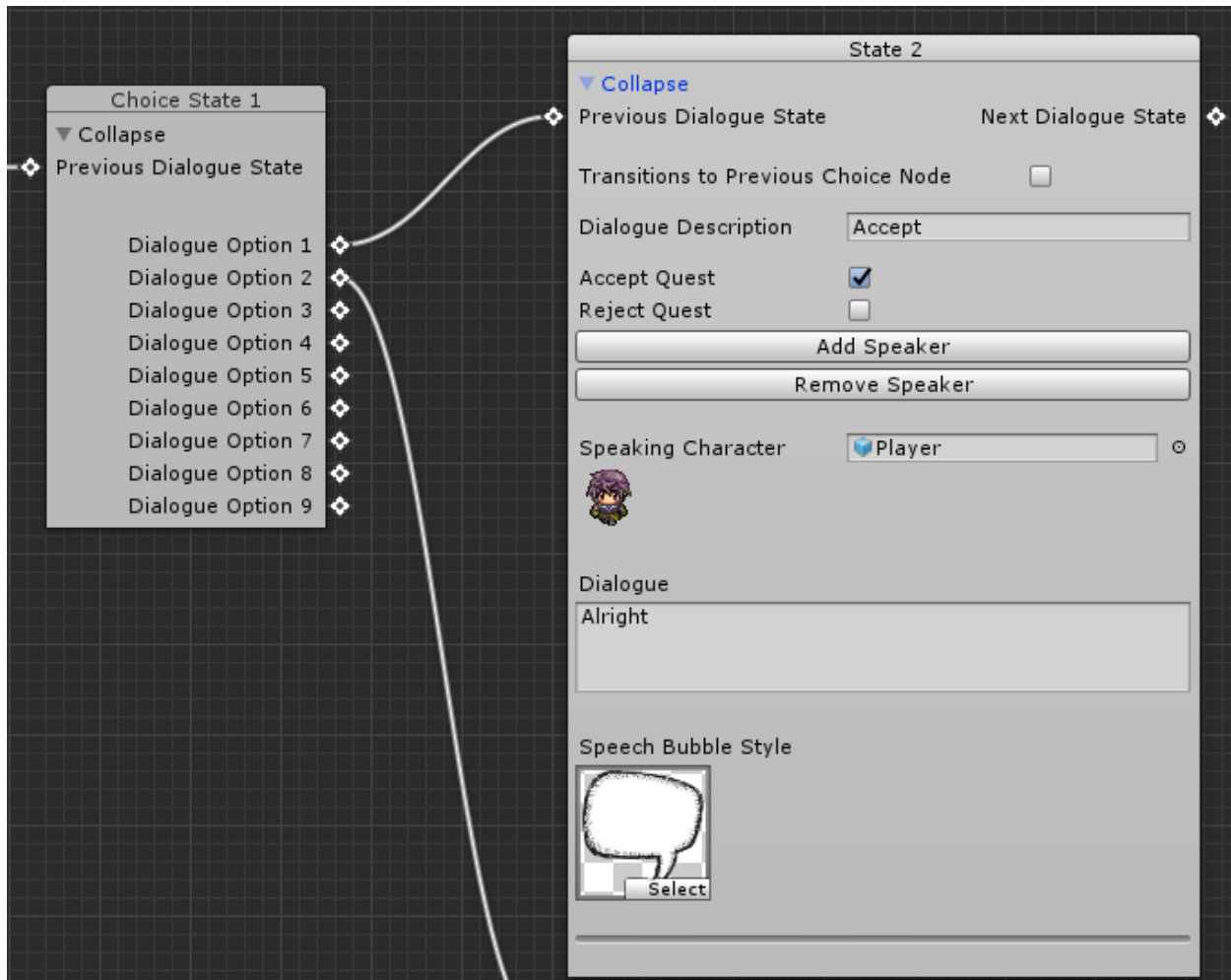
Transitions to Previous Choice Node

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.

Checking this box will specify this state as an accept state for the quest. This means that if the user makes this selection in the conversation, the quest will be accepted and added to the player's pool of active quests.

Checking this box will specify this state as a reject state for the quest. This means that if the user makes this selection in the conversation, the quest will be rejected and NOT added to the player's active quest pool. Currently, there's no implementation that locks the quest from the player permanently upon rejection. This means that currently, the player could reject a quest and accept it at another time. Locking the quest permanently upon rejection would have to be implemented by you but I may add it in the future.

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.
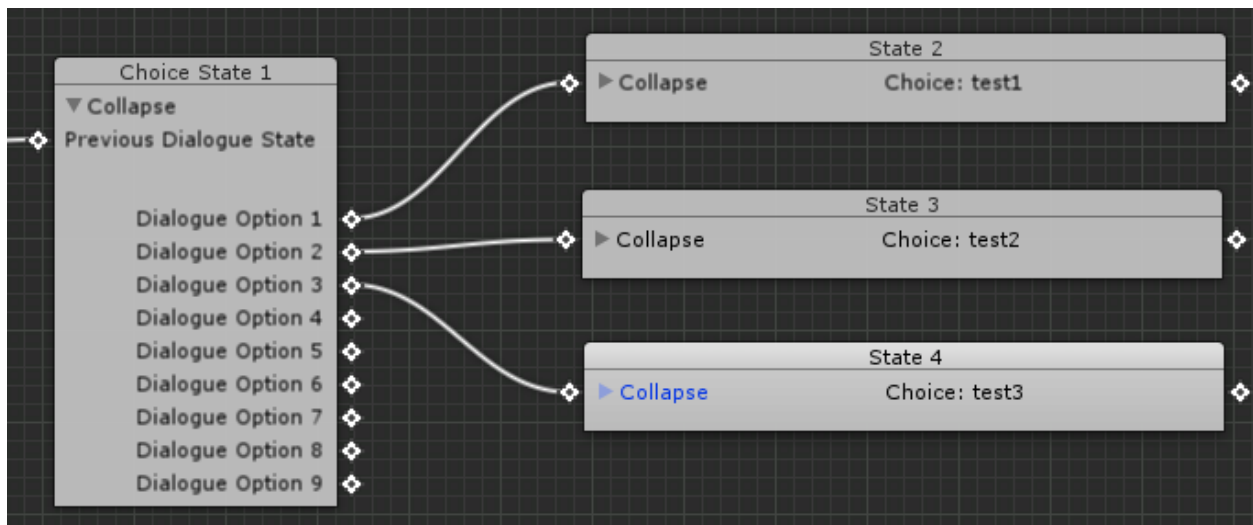
Collapse

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.

Game States

Game states are exactly like they sound. They are used to track the state of the game. They're not as complex as you might think. Basically, a game state is key-value pair that takes a string and specifies it as being true or false. An example of a common game state would be if the player has defeated a certain boss. We could create a game state called BIG_BAD_BOSS_DEAD and set it to true when the boss's Die() function is called when the player kills them. We could create a quest that requires BIG_BAD_BOSS_DEAD to be true before the player can access the quest. Game-state sensitive quests are a common feature in almost any video game that contains quests.

How to Add and Remove Game States

To access the Game State Editor, choose **Game State → Game State Editor** from the menu bar in Unity.
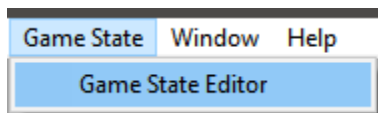
Upon opening the editor, you will see a window pop up that looks like this:
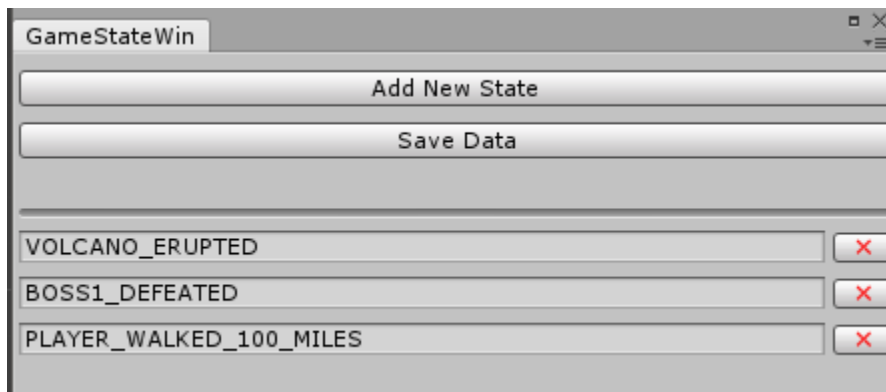
Add New State

This button will add a new state to the list. Once a new state is added, simply type the name of the new state in the empty field.

Delete a State

Clicking the icon next to a   state will remove it from the list of states.

Save Data

This will save the current game state data into your Resources/GameState/States folder. The data will also auto-save when creating or removing a state or when closing the window.

How Game State Data is Saved

Game States are saved as a custom asset in the **Resources/GameState/States** folder. A list of all the game states are stored in another custom asset called **GameStateData.asset,** which can be found in **Resources/GameState.**

These files must be located inside the Resources folder. This is because there is a lot of code using Resources.Load() to load the data at runtime.

Coding With Game States

Since you will likely need to script events that update game states, I will outline some of the main code you will need to utilize.

In C#, the game state is stored in **GameState.cs** in a single variable:

```
private Dictionary<string, bool> gameStates;
```

This dictionary is a key-value pair that has a string key and returns a Boolean value. For instance, VOLCANO_ERUPTED would be stored into the dictionary and would have a bool associated with it. When we want to set the bool to true, we call:

```
gameStates["VOLCANO_ERUPTED"] = true;
```

Additionally, we can add states to the dictionary simply by calling:

```
gameStates.Add("VOLCANO_ERUPTED", false);
```

This statement will add the VOLCANO_ERUPTED state to the dictionary with a default value of false. Note that adding and removing states is already done for you using the **GameState.cs** script, which is attached to the **GameState prefab**. Additionally, **adding and removing game states does not require you to touch the code** since the Game State Editor tool exists.

Setting a State to True

To set a state to true from any script, you can use this syntax:

```
Messenger.Broadcast<string>("Unlock Game State", "VOLCANO_ERUPTED");
```

In the above code, "VOLCANO_ERUPTED" is the name of the state you are unlocking. This message is sent to **GameState.cs** to be processed. **GameState.cs** then broadcasts a message out to any script that is listening for "Game State Updated".

Setting a State to False

To set a state to false from any script, use this syntax:

```
Messenger.Broadcast<string>("Lock Game State", "VOLCANO_ERUPTED");
```

In the above code, "VOLCANO_ERUPTED" is the name of the state you are unlocking. This message is sent to **GameState.cs** to be processed. **GameState.cs** then broadcasts a message out to any script that is listening for "Game State Updated".

Listening for a State Change

To tell a script to listen for a state change, use the following inside your MonoBehaviour script:

```
void OnEnable()
{
        Messenger.AddListener<string>("Game State Update", FunctionName);
}
```

In the above code, **FunctionName** should be replaced with the name of the function you wish this script to run when it receives a message that a game state has updated. **The function you make must take a string parameter.**

You must also tell the Messenger system that the listener has been removed to ensure proper functioning. Usually, you want to put this on any script that listens for a game state change:

```
private void OnDisable()
{
        Messenger.RemoveListener<string>("Game State Update", FunctionName);
}
```

Here is a general template of how you would add and use a game state listener:

```
public class TestListener : MonoBehaviour
{

    private void OnEnable()
    {
        Messenger.AddListener<string>("Game State Update", MyFunction);
    }

    private void OnDisable()
    {
        Messenger.RemoveListener<string>("Game State Update", MyFunction);
    }

    private void MyFunction(string stateName)
    {
        if (stateName == "VOLCANO_ERUPTED")
            Debug.Log("Oh no, a volcano erupted!");
    }
}
```

GameState Singleton

Game states only work if the **GameState** prefab is in your scene. GameState.cs is a singleton, which means it can be called on from any script in your project. For instance, in **GameStateChecker.cs**, line 41 has this code:

```
if (GameState.gameState.AreAllStatesUnlocked(requiredStates))
        ExecuteProcedure();
```

GameState.gameState will grab the singleton instance of GameState and allow you access to its public member variables and functions.
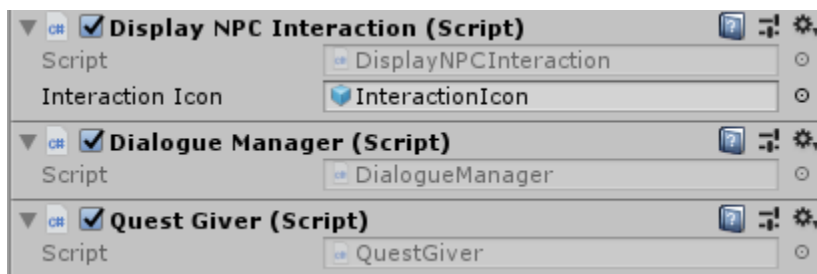
Prefabs

Required Prefabs for a Scene

Your scene must contain the **ConversationHandler, DialogueLoader, QuestLoader, GameState, and DialogueCanvas** prefabs. These prefabs can be found in **Assets/Resources/Prefabs.** Inside the prefabs folder, you'll find these assets in the **Dialogue, Loaders, and Globals** folders.

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.

Setting Up Quest Giver Prefabs

Characters that will be speaking and giving quests in the game must have certain components attached to them. The main components required are **DialogueManager.cs, QuestGiver.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, these prefabs must have a child GameObject called "DialoguePosition". This Game Object is nothing but a Transform and is used for getting Vector coordinates for instantiating the dialogue box for the speaker.

These prefabs 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 **SampleScene.unity** scene to see how these prefabs are implemented.