DEW: Dialogue Editor Window Manual

About

DEW is a custom editor window for the Unity editor created by Lazuli Software. It assists with creating, editing and storing dialogue trees for use in games. DEW provides sample code to help users parse the dialogue trees in their software.

This manual will go over the controls of the editor window, and how to use the files created by the window.

Contents

Set Up

File placement

The package that DEW comes in is very easy to set up.
Firstly, we recommend not moving the assets from their original folder. The most important thing is to keep the script DEW in the same folder as DEW Data, the software will not work properly if they are seperated.

To open the window in Unity Editor go to the Window menu and open DEW.

Sample Scene

This asset comes with a sample scene. It can be found in the Sample Assets folder as DEW Sample Scene. The scene will play through a dialogue tree in the Unity editor console. Make sure the editor console does not have collapse set on. Once the scene has started press any key to progress through the dialogue. If a choice is presented, press the corresponding number key (the numpad will not work) to choose your answer.

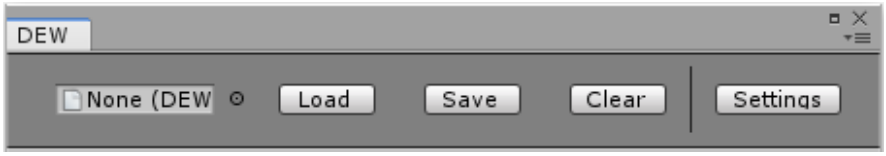
The console output is from the script DEWSampleCode reading the scriptable object Sample DEWDialogue. In this script you can find how the dialogue tree is parsed. For more information on how to read a DEWDialogue object read Parsing the Dialogue Tree (page 8).
DEWSampleCode can be used to read other DEWDialogue objects but may not work properly if there is a choice with more than 10 options.

What is a Dialogue Tree?

A dialogue tree is a data tree for storing dialogue. They are used to store dialogue which branches, this means the dialogue can go in mulitple directions based on conditions.
The most common form of branching dialogue is one where the player is asked a question and presented with multiple answers. The answer that the player gives will change what the next dialogue will say. In this case the condition for the branch is the player’s answer.

Controls

Window Controls



Object field - This lets you select a scriptable object of the class SODialogue. If one is selected it will load from and save to this file.

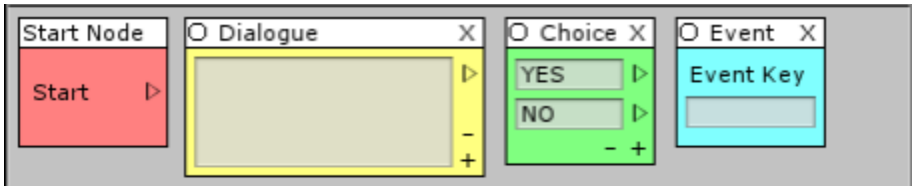
Button [Load] - This button will load data from the SODialogue file in the object field, if there isn't one selected it wont do anything.

Button [Save] - This button will save the current data to a SODialogue file. If there is a file in the object field it will overwrite it. If there in no file selected it will open a file picker and you can choose where to save the file.

Button [Clear] - This button will clear all the workspace and only leave a start node.

Button [Settings] - This button will open the settings menu. The data in the window will not change when the settings menu opens.

Node Controls



Header **-** Click and drag the node header to move the node. The header will darken while it is clicked.

Button [X] - This button will close the node. Start nodes do not have this button and cannot be closed.

Button [>] - This button will activate the arrow. The arrow will be drawn to your mouse click on the top left of a node to connect the arrow to another node.

Button [+] **-** This button will add another section to a node. Be aware that there is a limit to the amount of sections a node can have.

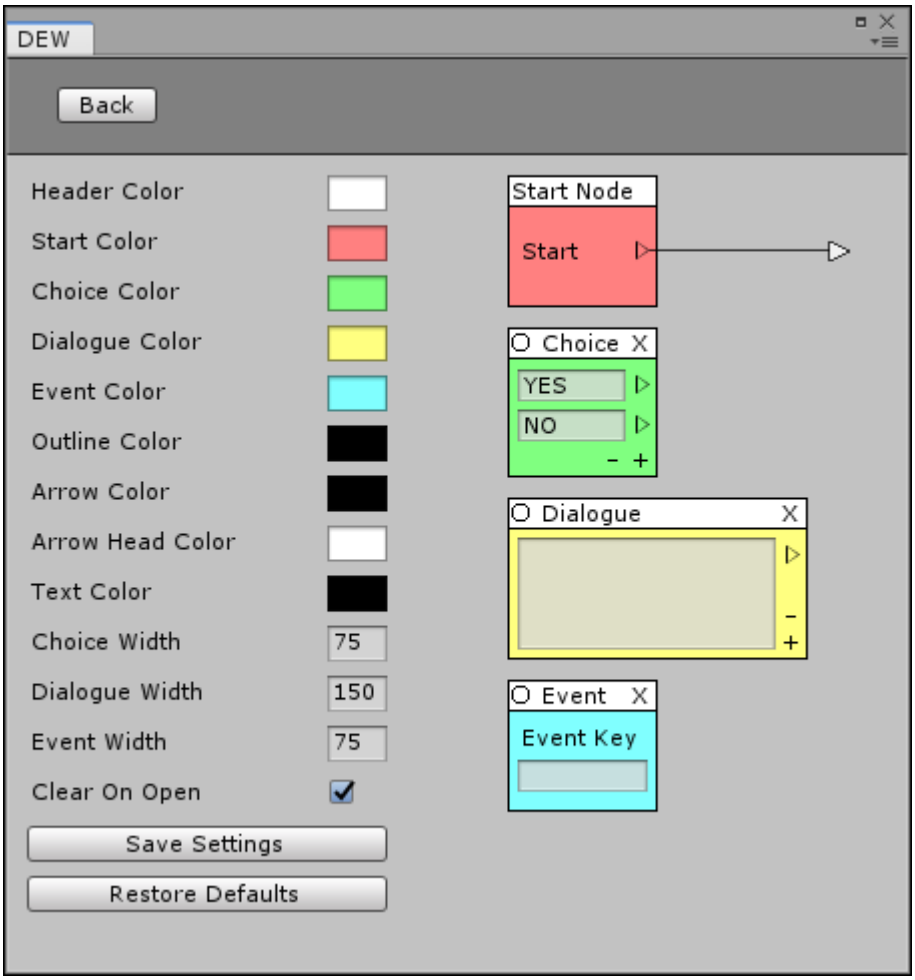
Button [-] - This button will remove a section from a node. Be aware that there is a minium number of sections that a node must have.

Creating Nodes

Right click in the workspace will open the node menu. Click on either *Dialogue, Choice* or *Event* to create the corresponding node. Left clicking anywhere outside the menu will close it.

Settings Menu

The settings menu allows you to change various attributes of the editor window. It displays sample nodes which will update with any changes in real time.



Button [Back] - This will return you to the main screen. Any changes will not be saved.

*Color field*s - These will change the various colours of nodes and arrows. There is a colour field for node headers, the start node, choice nodes, dialogue nodes, event nodes, node outlines, arrow lines, arrow header, and text colour for nodes.

Int fields - These will change the widths of nodes to allow room for more text. There is a width setting for choice nodes, dialogue nodes, and event nodes.

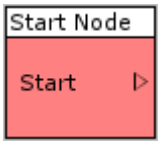
Toggle (Clear On Open) - This will clear the window of all nodes when it is opened. When the toggle is false the window will retain the tree from when the window was closed.

Button [Save Settings] - This button will save the current settings to memory.

Button [Restore Defaults] - This button will reset all settings to their default values. These changes will not be saved.

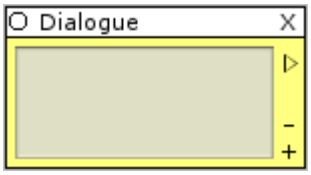
Types of Nodes

Start Node



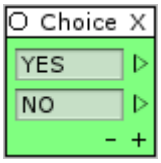
The start node is the first node in the tree. It's only purpose is to connect to a dialogue node to tell the code where the tree starts. The start node can not be connected to and can only connect to dialogue nodes.

Dialogue Node



The dialogue node is holds the majority of text in the dialogue tree. It starts with one text box but more can added. The dialogue node can connect to other dialogue nodes, choice nodes and event nodes.

Choice Node



The choice node is what creates braches in the tree. It is shows what the player's choices will be in the dialogue. It starts with two choices, "YES" and "NO", but they can be edited and more choices can be added. It can connect to as many nodes as there are choices. Not every choice has to connect to a node. It can connect to dialogue and event nodes.

Event Node

The event node is used to call UnityEvents in the dialogue tree. It only holds a text field labelled "Event Key". When the dialogue tree is parsed you can use the event key to trigger a UnityEvent at the specified part of the tree. To do this you would need to send a collection of events with the tree too. The event node cannot connect to any nodes.

Connecting Nodes

Nodes are connected with arrows. Pressing the [>] button on a node will connect an arrow to your mouse, click on the O symbol in the top left and the arrow will connect to that node. Start nodes cannot be connected to.

Saving & Loading

Dialogue trees are saved to DEWData scriptable object files. DEW can also load the saved dialogue trees from DEWData files.
When saving data having a DEWData file in the object field will change what happens. If there is a file in the object field the file's data will be overwritten with the new saved data.
If there is no file in the object field the file picker will appear. If a file is overwritten in the file picker references inside Unity Editor to it will be lost.
The data saved to files will only be nodes that are connected in the tree connected to the start node. Event nodes will not be saved unless they have a key in them.

Data will not be loaded unless there is a file in the object field.
When data is loaded it may not be in the same positions as it was saved. To save memory space only the position of dialogue nodes and choice nodes are saved. The start node will be placed in the top left of the window and event nodes will be placed to the right of the node that connects to them.

Using the Data

Dialogue Struct

When data is saved it is stored as an array of the struct Dialogue. Each Dialogue struct holds:

The contents of one dialogue node stored in an array of strings named *m_words*. Access with *GetWords()*.

The struct's identifier stored as an int named *m_ID.* This int is not unsigned but will be a positive number. Access with *GetID().*

All possible next Dialogue structs stored as an array of ints named *m_nextID*. Access with *GetNextID().*

All choices in a branch of the tree stored as an array of strings named *m_choices*. Access with *GetChoices().*

All possible event keys stored as an array of strings named *m_eventKeys*. Access with *GetEventKeys().*

A Vector4 named *m_position*. This is stored so files can be loaded nicely.

All data is stored privately and will need to be accessed through their respective public functions. This prevents the data from being accidently changed. The Dialogue struct also holds various functions to help interpret the stored data.

MultipleNext() will return a bool. If the current Dialogue branches it will return true.

IsLast() will return a bool. If the current Dialogue struct does not connect to any other Dialogue structs this will return true.

HasEvent() will return a bool. If the current Dialogue struct has any event keys this will return true.

DEWDialogue

This is the scriptable object class dialogue trees are saved to. This class holds a few functions which are useful for parsing the data.

GetFirstDialogue() - This will return the first Dialogue struct in it's tree.

GetDialogue(int _id) - This will return the Dialogue struct which has the id sent in. If no struct with the id exists it will return an empty Dialogue struct and send a debug message to the console.

GetAllDialogue() - This will return an array of all the Dialogue structs.

Parsing the Dialogue Tree

The easiest way to navigate through the tree is with a coroutine as you need to wait on input. I will go over how the provided sample code goes through the tree.

1. Get the first dialogue struct with DEWDialogue GetFirstDialogue() and assign to a variable. I'll call it current Dialogue.
2. Display each text box of current Dialogue. I wait for an input after each one.
3. Check if current Dialogue is at the end of the tree with Dialogue.IsLast().
    a. Check if current Dialogue has an event with Dialogue.HasEvent().
        i. Call event with string from Dialogue.GetEventKeys()[0].
    b. Finish navigating tree.
4. Check if current Dialogue has a choice with Dialogue.MultipleNext().
    a. Draw all choices from Dialogue.GetChoices().
    b. Wait until player has chosen and assign it to an integer, I'll call it choiceIndex.
    c. Get the id of the next Dialogue with Dialogue.GetNextID()[choiceIndex] and check if it is -1.
        i. Check if current Dialogue has an event and the event key is not an empty string.
            1. Call event with event key from Dialogue .GetEventKeys()[choiceIndex].
        ii. Finish navigating tree.
    d. Set the current Dialogue to the next one with DEWDialogue.GetDialogue(Dialogue.GetNextID()[choiceIndex]).
    e. Return to step 2.
5. Set the current Dialogue to the next one with DEWDialogue.GetDialogue(Dialogue.GetNextID()[0]).
6. Return to step 2.

To return to step 2 I use a while loop. In the loop I use the keyword break to exit the navigation seen at step 3b and step 4cii.

Contact

If you require assistance with setting up DEW or have any queries about the software do not hesitate to contact us at Lazuli Software.

Email: LazuliSoft@gmail.com
Website: LazuliSoftware.wordpress.com