DIALOGUE LANG

Created By Conor McDonagh Rollo

Table of Contents

Introduction

Thank you for choosing to use Dialogue-Lang(DL)! This guide will be your go-to for understanding the core concepts and systems that make DL work; It will also teach you how to modify the systems.

How does DL work?

DL in simple terms, takes a specifically formatted text file, and transforms it into an interactive dialogue that you can have with an NPC upon interaction. It breaks the text file down into a few functioning parts: Dialogue Sections, Dialogue Choices, and Uniform variables.

Dialogue Sections

Dialogue Sections are the highest level of the systems moving parts. They contain two strings; A header, and a redirect. Headers are the name of the section and are used to reference that section from other areas of the DL script. The redirect is used in the case that there is a condition added to access the current dialogue section. In the case that you don't meet the criteria for accessing that dialogue section, you can add an else condition to script that will redirect you to some other dialogue section.

There are also two lists contained within the dialogue section; The lines and the choices. The lines part is just a string list that holds the section's dialogue lines that are given a typewriter effect in-game with a little delay between the writing when a new line is detected. The choices list holds all of the possible response options that the player is given.

Dialogue Choices

Dialogue Choices are the interactable part of the dialogue system and they are pretty simple in how they work. They contain some text and the header of the section that you go to once the button is clicked. Of course they aren't limited to just that, you can also add conditions to these choices for whether they are displayed or not, allowing you to tailor the choices that the player should be able to see.

Uniform Variables

These are variables that the code can access, but are initialized inside the DL script. The vanilla options for types are very barebones as you are meant to tailor it to your own needs. You have the option to get and set the variables as a plain C# object, or you can get and set it as an integer. It is possible to modify the code to use strings, bools, or even your own custom data structures.

Parsing

Now it is probably correct to assume that you would be wondering how all of this comes together, and that is a very good question. The answer is that DL script is treated as a sort of barebones interpreted programming language. The text is read by the parser and goes through a process called two-pass-parsing. In the first pass, it notes all of the conditioned

sections, collects the defined uniforms, and then saves them for use in the second pass. Now in the second pass, all of the dialogue sections get detailed, all of the conditionals get evaluated to either true or false, and all of the buttons and redirects are generated.

Sounds complicated

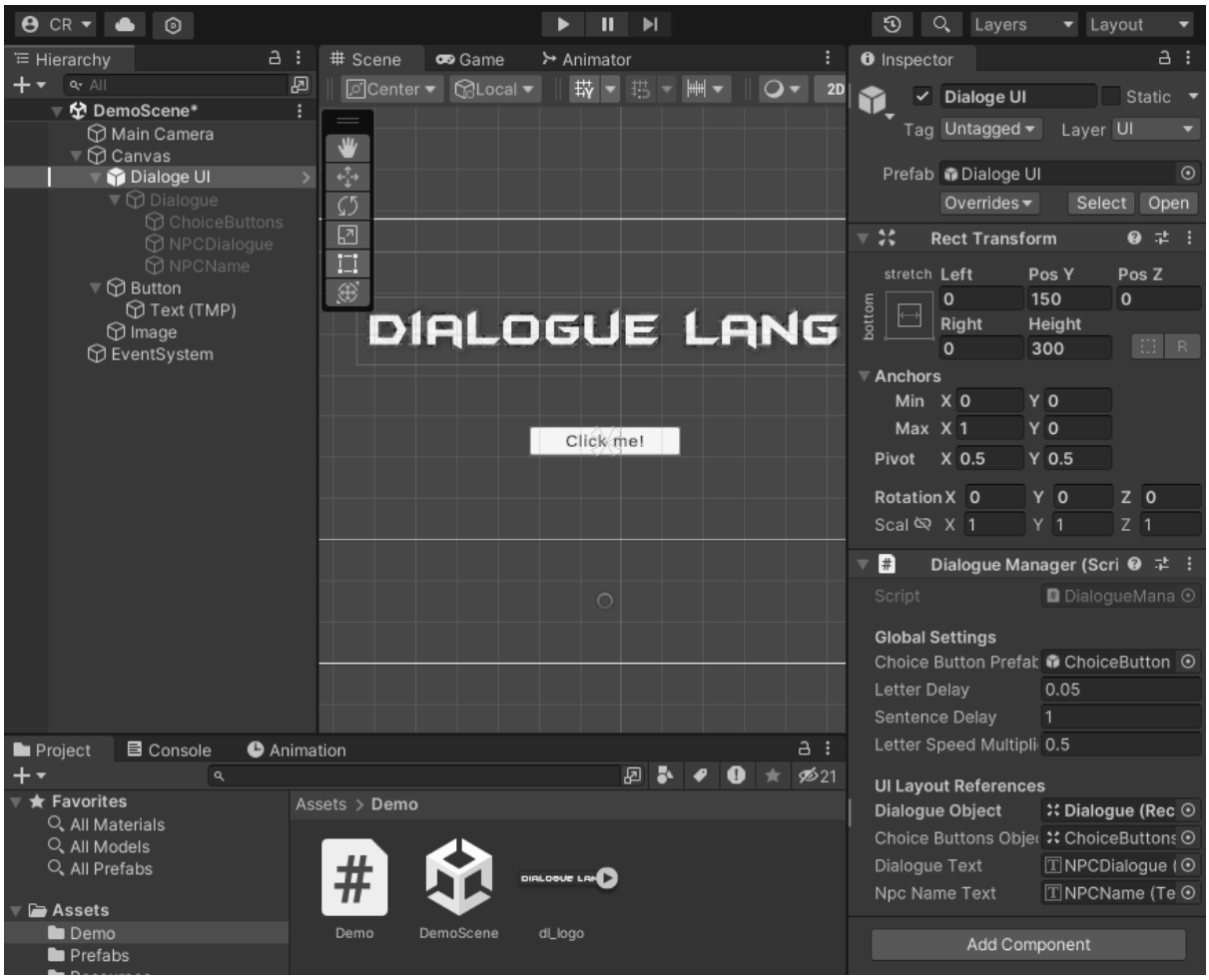
It's actually the complete opposite! The parsing side of the system doesn't really need to be touched, all that's really required is to set it up similarly to the demo scene using the provided prefabs. The syntax was designed to read very easily even by non-programmers. Now let's get into the nitty-gritty!

Usage

Once you import the package into unity, setup is fairly easy, just follow along.

Getting Started

Start by adding a canvas to your scene if you don't already have one. Then add the Dialogue UI prefab as a child of the canvas. If the Dialogue Manager script that is attached to the canvas doesn't have the UI objects references, drag them into the relevant slots. This is important so that all dialogue components that are added to the scene can modify the correct UI.



Make an interactable NPC

In this example we simply use a button, but for your project you can use whatever way you want to trigger an interaction. Just call the attached dialogue script's interact function. What we do here, is add a button, attach the dialogue script, detail the name of the NPC and the name of the file that contains the dialogue (which needs to be inside the root of a folder named "Resources" in the root of your project). We set the onclick event of the button to trigger the Dialogue.Interact() function. I have also attached a demo script to update a uniform variable, but I will show that in the next part. You may notice that I put the demo script's update dialogue function in the dialogue's On Invoke event, that will be explained as well.

The DL Script

Below is the code for the dialogue which is inside demo.txt:

```
count = 0

if [count == 0] [initial] else [section2]
Welcome to the dialogue lang demo! Select the "Update variable" optio

Update Variable [INVOKE]
Exit [EXIT]

if [count == 1] [section2] else [end]
And that's how conditions streamline this whole process!

Thanks! [INVOKE]

[end]
That's it..
That's the demo.

Exit [EXIT]
```

Let's step through it line by line.

1. This is a uniform variable which gets updated with the demo script. It has 3 parts: the name (count), the assignment operator (=), and the initial value (0).
2. You can make as many of these variables as you want.
3. This is the declaration of the first dialogue section. The first dialogue section is always called "initial" (without capitals). This is also a conditional dialogue section which is what the "if" is for. A condition starts with the if, and it then followed by the condition enclosed by square brackets. This checks if the count uniform variable is equal to zero, then the initial dialogue section is defined. In the case that it isn't, we use the else, followed by another dialogue section that we would rather redirect to. (also in square brackets)
4. This is what dialogue will display when entering the initial dialogue section, nothing fancy here.
5. It can however be split into multiple lines to break up the sentences.
6. This part is where the choices are detailed. The first choice button is first given the text that will be displayed on it (Update Variable). And is followed by what dialogue section you wish to go to when clicking it. However, there is no dialogue section named "INVOKE". This is actually a reserved keyword that calls the On Invoke event of the current dialogue.
7. Similarly, "EXIT" is also reserved as a way to quit the dialogue.

8. Buttons can also be subject to conditions but cannot be subject to else conditions. Anyways, that concludes this current dialogue section. The rest of the code follows the exact same syntax.

Lines 9-13 are another dialogue section, which is also conditional, similar to the first. Lastly lines 14-18 are just a regular dialogue section without any conditions.

Updating the uniforms

Let's take a peek at how exactly I am updating the uniforms in the demo script.

```
0 references
public void UpdateDialogue()
{
    dialogue.SetDialogueVariable("count", dialogue.GetDialogueVariableAsInt("count") + 1);
}
```

Inside this we have our dialogue script reference named "dialogue", we are accessing the SetDialogueVariable function. The first paramater is the name of the dialogue variable as it is in the text file. The second is what you want to set it as, so I want 1 + whatever the dialogue variable is set as right now. For that we can use GetDialogueVariableAsInt, with the parameter being the name of the variable again.

Conclusion

That is just a simple way to use it but there is plenty of room to scale it up and create massively complex dialogue interactions making it immersive for the player and easy for you.