Conversation System Documentation

Version 1.2 by Leo Li, Sep 13, 2018

For Writers and Designers

The conversation system takes in one large text file containing all conversations that will be used in the game. It automatically organizes the text into a dictionary fashion, where every line or every batch of lines will correspond to a unique key (which can be a word, a phrase, a sentence, or pretty much anything you can type in a word document). Because of this design, the large text file needs to be formatted in a special way, and involves several some uncommon symbols as well.

Here’s an example of how a short conversation would be formatted:

S1 at the dining table | Mom \ So how’s your school today my dear? (S1 positive response) It was great! (S1 negative response) I don’t want to talk about it…

Symbols used:

| – not sure what this is called, but this vertical line symbol separates the key and body of a conversation. S1 at the dining table is the key, and will never be displayed in game. It is a good idea to write something that can summarize the conversation as the key. The rest of the content after | is the body of the conversation, and most of the body will be displayed in game.

\ – the forward slash itself will not display in the game. The text before the slash will appear as the title/speaker of the line, while the text after it will be the actual line. Be careful, do not use it more than once for every line, the system will not recognize anything after a second slash. By line I do not mean a full line of words on a text editor or word document. Here I am referring to a character’s line that will be displayed to the player upon each mouse click.

() – parenthesis signal the end of all character lines and prompt a decision from the player. The text in each pair of parenthesis should be a key, which will lead the player to another conversation starting with that key. The parentheses themselves and the key within will not be displayed in game. The words after the parentheses will be displayed on screen as options for the player to choose from. Currently, there can be a maximum of 4 options. Unused ones will not be displayed and have no impact on the game.

Here are some additional features about parentheses:

1. An empty pair of parentheses will trigger the end of the conversation for that option.

2. If there’s only one option and that is to quit the conversation, you may simply omit the parentheses altogether (or leave one pair of empty parentheses at the end). In this way, the conversation system will automatically exit upon the last line without prompting the player to choose from any options.

3. Again if there’s only one option and that is to jump to a certain key, you may use a single pair of parentheses with the key but no words afterwards. This will again prevent the system from prompting the player for input, thus connecting branches of conversation seamlessly.

Additional note: white spaces in between the special symbols and words are ignored. For example, S1 at the dining table | Mom is no different from S1 at the dining table|Mom. However, do note that S1 positive response is not the same as S 1 positive response.

Of course, not every conversation will consist of one line and a response. Sometimes a character can go on and on before giving the player a chance to take action. For situations like this, we need to incorporate some extra stuff.

S2 at school | Prof. SomeGuy \ Welcome to my History of Media Arts class. ` This semester we will be looking at the innovation of film technology and various film movements. ` Before I go into details, let us first go over the syllabus… ` Danny \ Omg I am falling asleep already. (S2 agree) Yeah me too. (S2 disagree) He seems ok. (S2 whatever) It’s whatever… I am <b>done</b> with college already.

Here’s the breakdown:

` – known as a “grave”, I picked this because it is not frequently used in English. This symbol will separate one character line from another, and you can find this on your keyboard right next to number key 1.

Note: You do not have to include a title/speaker and a “\” for every line. The title will stay for the entire conversation until the player chooses a response and jumps to another conversation. Of course, when another person jumps in and talks, you’ll need to update the title/speaker, like how I did with “Danny:”

<b></b> – These are known as “rich text,” and you would probably be familiar with them if you have coded website before. <b></b> will make the text in them bold, and there are other tags that italicize or even change the color of the text. You can find all the supported tags here:

<https://docs.unity3d.com/Manual/StyledText.html>

Very important, the conversation system currently comes with an option of “typewriter effect,” which is exactly what you think it is. However, it does not work with nested rich text tags. Basically, you can’t make something bold and colorful while having the typewriter effect on

Another very important note: each conversation/batch of lines is separated with a carriage-return – basically a press of ENTER key or a new line. Think of your conversations as paragraphs. Be very careful when you type long character dialogues – when they get to the end of a line, let them wrap themselves to the next line automatically. Do not manually press ENTER and go to the next line. Otherwise you will confuse the system into thinking that this is another conversation.

Some Extra Stuff:

In scenarios where the development of a conversation depends on in-game variables, we can use variable keys. These keys should start with the “#” symbol, and the rest is arbitrary. Doing so will tell the system not to jump to any key, but rather sending out this message after “#” and asking the game to handle it (editing the key). This will need to be implemented on the developer’s side. However, you can use “#” at the beginning of a paragraph (in between conversations) to turn that paragraph into a comment. A paragraph marked as comment will not be picked up by the conversation system and acts purely as a way to comment/communicate. As the writer/designer, you can instruct the developer through comments on how a variable key should perform. Here’s an example:

……a long conversation omitted here…… (key1) Give up. (#key2) Challenge.

#key2 – If player’s attack > 10, go to key3. If not, go to key4

For Developers

User Interface Level:

\*The conversation system prefab has to be a child of a canvas object. It will not be visible by itself.

\*Currently the UI is set for a 4:3 aspect ratio, but it can be modified or changed

Public variables: (visible on the inspector panel)

Title – Text UI element for the title/name of the speaker.

Content – Text UI element showing what the person is saying.

Options – A group of buttons, initially hidden. They are the options the player can click on.

Main\_character\_name – Displayed as the title when the player is picking an option. Can be left empty.

Typewriter\_effect – Basically adding one letter a frame, a little animation created with Unity’s coroutines.

Scripting Level:

The conversation system forces single instance. It can be destroyed and re-instantiated, however only one may exist at a time in the scene.

Static properties:

Gameobject instance – Use this to access the current active instance of the conversation system.

bool active – Whether a conversation is in process. Set to true after entering a conversation and set back to false after exiting one.

Static methods:

Initialize () – Can be only called once and only needs to be called once. Call this at any stage of the game, preferably at the beginning/main menu. The conversation system will look under the path Resources/Conversations for a list of text files that contain conversations. Make sure to have a folder named Resources preferably at the root of your asset folder, as well as a Conversations folder containing all the files. Files will be read in one by one and organized into a collection for later access. Make sure to call this method before invoking any conversation.

Public methods:

EnterConversation (string key) – Use this to enter a conversation, a key is needed.

ExitConversation () – Called automatically when an empty key is reached, kept public just in case.

GoToKey (Text key) – Called from the onClick event on those option buttons, feeding the conversation system information about what the player clicked. Has to be public.

GoToKey (string key) – Called from within the system to switch between keys. Made public to allow more customization.

Event:

key\_reached – When a certain key is reached. Subscribed methods need to have this signature: void MethodName (string key), use this to track the player’s decisions as well as triggering any special event amid a conversation.

variable\_key – When a variable key is reached. Subscribed methods need to have this signature:

void MethodName (out string key), modify the key based on in game situations. Make sure that the key actually exists in conversation files.

One small note:

The forward slash symbol “\” is written as “\\” in code.