Character Line View Tutorial/Documentation

By ArcherZenmi

Table of Contents

1) Available Features

Abstract

Hi there! Thanks for taking interest in my little corner of the internet.

Character Line View is a module designed to integrate rpg-style text animations (i.e complex “type-writer effects”) with YarnScript’s dialogue management tools. You know how in real life, people sometimes pause during their sentences, talk fast or slowly, or, you know, have a voice? Character Line View gives a stream-lined way of creating these effects through YarnScript’s attribute and command system.

The other main functionality of Character Line View is that it puts your “Characters” first and foremost. Each character who speaks can have custom textboxes, custom voices, and custom icons expressing various emotions. The best part is that once you set these up in the inspector, you can easily manipulate them through tags and attributes right from your dialogue script!

A large part of these features are inspired by the Renpy Visual Novel engine, so shout-outs to them! (https://www.renpy.org/)

Features

- Animations
    - Complex “type-writer effects” for your text
    - Intersperse cutscenes/animations with your dialogue
- Character Features
    - Custom icons depending on the character’s emotion
    - Custom Textboxes for each character
    - Custom voices for each character (the beeps and boops from Undertale, Animal Crossing, or most rpg’s)
- Assets
    - Includes some character textboxes from one of my games, if you’re lazy about making textboxes :3
    - A few character icons to get you started (though you’ll need your own to make full use of the custom emotion tags)

2) Basic Set-up

To use this module, I'll assume you have a working knowledge of YarnSpinner. It's the Unity standard for videogame dialogue, so you won't regret learning it if you haven't already (https://docs.yarnspinner.dev/).

Once you drag'n drop the "Character Line View" folder into your project's "Assets" folder, this Basic Set-up chapter includes the bare necessities of what you need for Character Line View to start displaying dialogue for you. All subsequent chapters are focused on more specific functionality to make cool effects, but if all you want is a functioning dialogue display then this'll do just that.

2.1) Required Packages

1. Cinematic Studio
    a. Cinematic Studio is part of the standard Unity registry. If you don't know how to install packages, this'll be a good place to start (https://learn.unity.com/tutorial/the-package-manager-1?uv=2019.3).
2. Input System
    a. This is another package from the standard Unity registry. After installing make sure to enable it in the inspector, or your game probably won't run.
3. DoTween
    a. DoTween is used internally to animate the displayed dialogue. Note that you do NOT have to know how DoTween works in order to use Character Line View.
    b. Installation: https://assetstore.unity.com/packages/tools/animation/dotween-hotween-v2-27676
        i. I HIGHLY recommend that you install from Unity's asset store, rather than the DoTween website. That way it's easier to stay up-to-date with versions.
    c. Documentation: http://dotween.demigiant.com/getstarted.php
4. YarnSpinner
    a. This is the Unity standard for video game dialogue (did I say that already?). YarnSpinner helps you neatly organize dialogue for any

game, and allows for easy extension should you feel compelled to make your own Character Line View v2.

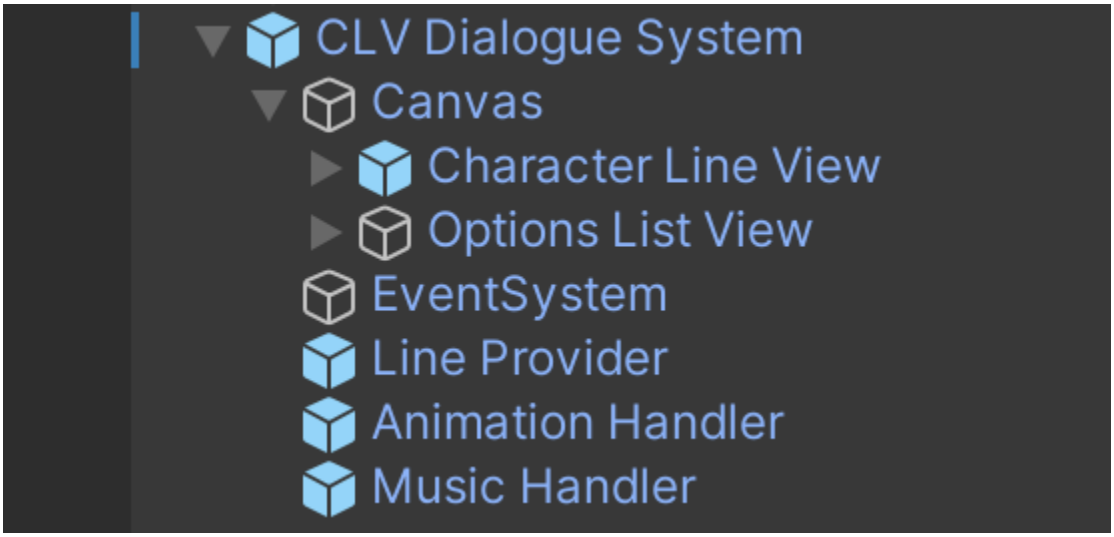
b. Installation: https://docs.yarnspinner.dev/using-yarnspinner-with-unity/installation-and-setup
c. Documentation: https://docs.yarnspinner.dev/

2.2) Prefabs

All the required pre-fabs are in the "Prefabs" folder. Throughout this tutorial, you'll primarily be using two of these prefabs. "CLV Dialogue System" and "Character Icon".

1. CLV Dialogue System

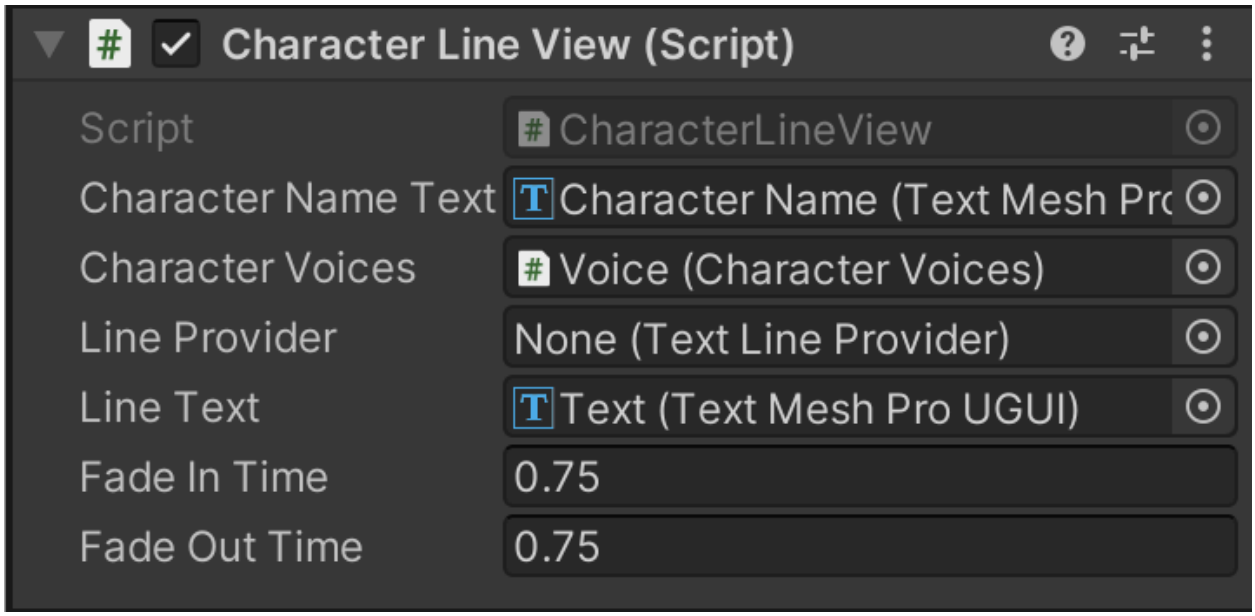
This is a convenient prefab to replace YarnSpinner's Dialogue System prefab. It already has all the connections needed in the inspector for Character Line View to function properly. Just drag it into your scene hierarchy and it's good to go! (after following the steps in Basic Set-up anyway)



If perchance CLV Dialogue System doesn't work (maybe YarnSpinner released an update) or you want to customize it, it's useful to understand how you'd build this yourself.

A. Drag and drop YarnSpinner's normal Dialogue System prefab into your scene hierarchy.
B. Add the following prefabs into the positions shown above: "Character Line View", "Line Provider", and "Animation Handler."

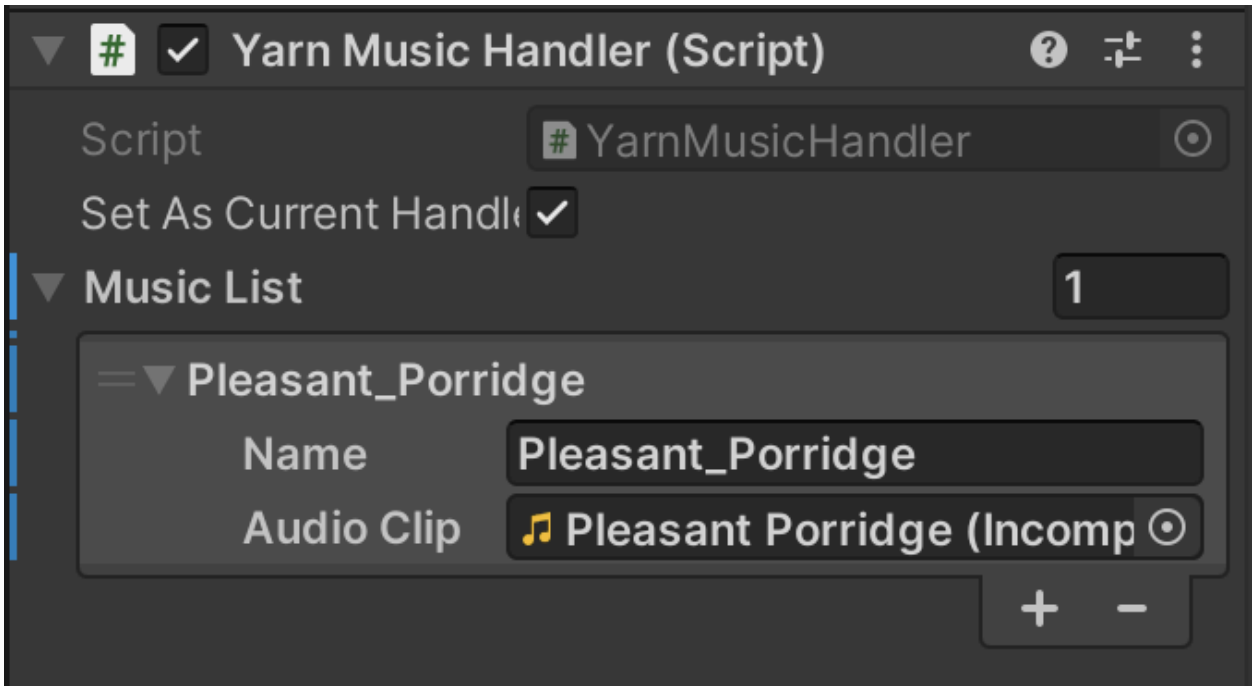
C. Click on Character Line View. Drag and drop the Line Provider from the scene hierarchy into the “Line Provider” field.



D. Click on Animation Handler. Drag and drop the Character Line View from the scene hierarchy into the “Dialogue Box” field. Unless you’re juggling multiple Animation Handlers in your scene, you’ll also want to tick the “Set As Current Handler” box.

E. Click on Music Handler. Much like Animation Handler, you’ll want to tick the “Set As Current Handler” box unless you specifically want another Gameobject playing music. Then, you can add music by adding an audio

clip, along with the name used to call it in YarnScript.



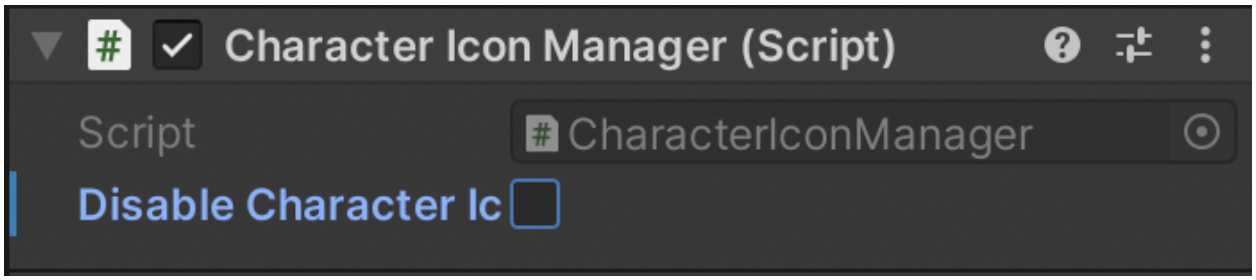
F. Finally, don't forget to add Character Line View as one of the Dialogue Views in the Dialogue System inspector!

2. Character Icon

If you want character icons for your game, drag n' drop some "Character Icon" prefabs as a child of "Character Icon Manager". Afterwards, you can give that character their name to be used in YarnScript. Give them as many emotions as

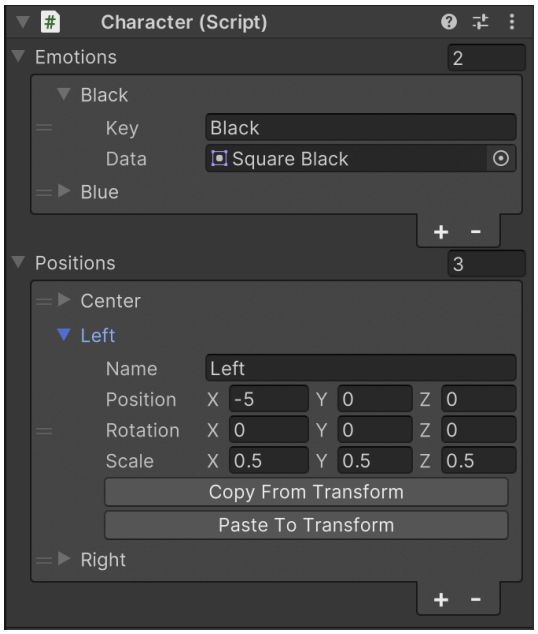
you want, with the name of the emotion and an image/icon to display for it.

Afterwards, make sure to un-check the “Disable Character Icons” button. Otherwise, CLV will assume you don’t want character icons.



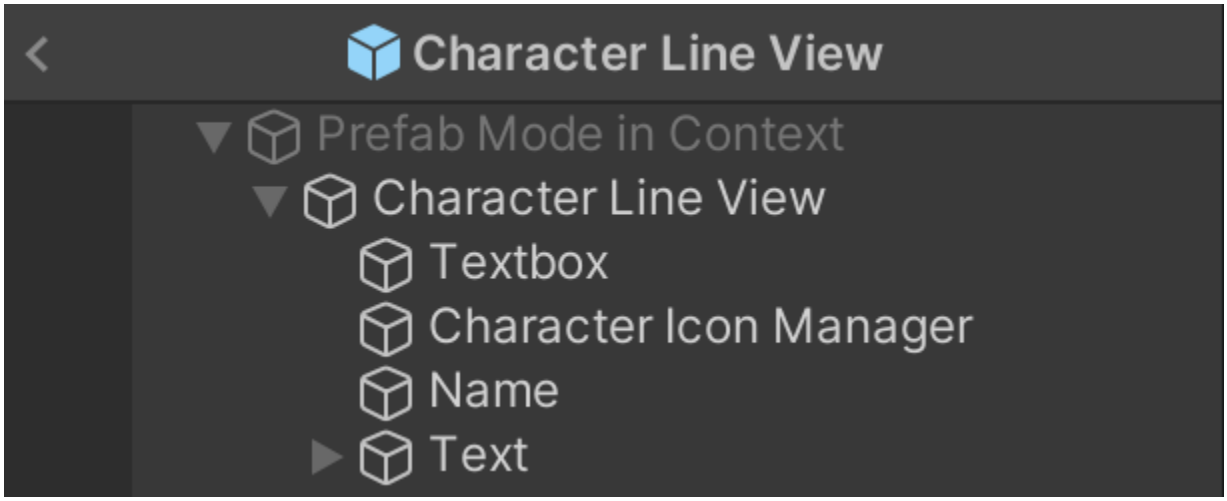
3. Character

If, instead, you want full VN-style drawings in your game to represent characters, then use the “Character” prefab. You can specify emotions/sprites that a character can display through the “emotion” command, and positions to move to through the “move” command (more on these in the Commands section). Once set up, these can be called from YarnScript so you can control the narrative more effectively.



2.3) Customize Characters

Customizing the UI for your cast of characters will mainly consist of editing Character Line View's children via the inspector.



Here are the children that can be customized:

1. Textbox
    a. This is used to assign specific text boxes for each of your characters. Browse the inspector, and look for the "Character Textbox (Script)" component, and add as many textboxes as you please. Each textbox should have the associated character's name that's used in YarnScript, and an image to use as the textbox.
2. Character Icon Manager
    a. No alteration for this node is required. Simply add Character Icon's as children, as specified in section 2.2.
3. Voice
    a. This is an audiosource that can play some beeps and boops on loop whenever a character is talking (think Undertale or Animal Crossing). Enter the inspector, and look for the "Character Voices (Script)" component. Each character should have their name used in YarnScript, and an associated audio file to play for them.
    b. Note: The recommended length for the audio file is 0.5 seconds.

3) Writing Your Script in Yarn

Now that you've set-up everything, it's time to see all the cool stuff that you can do with Character Line View! The default CLV Dialogue System prefab comes with an example script you'll find in the "Yarn Scripts" folder. However, for the most part, you'll want to write your own scripts with the info you'll find below.

We're going to cover animating the dialogue text with attributes, using tags to change the displayed emotion/icon, and using commands to intersperse animations/cutscenes with your dialogue.

3.1) CLV Attributes

Attributes are implemented using YarnSpinner's mark up system. However, CLV's attributes are intended to be used as self-closing attributes. They may work if used as normal attributes, but may also have some undefined behavior.
Note that any effects that attributes have only persist for that line. That's to say, if I set the cps to 5 in line 1, then by line 2 it'll already be set back to the default value of 60.

Wait Attribute

- Signature
    - [w=int or float/]
        - w: How many seconds to pause the dialogue.
- Description
    - The wait attribute is used to add pauses in the middle of a line.
- Example
    - "Sylveon: I'll be seeing you...[w=1.0/] IN HELL!"
    - Wait Attribute.mp4

Cps Attribute

- Signature
    - [cps=int/]
        - cps: Characters to display per second.
    - [cps default=bool/]

        - default: If set to true (or false), the cps will be reset to the default speed.
- Description
    - The cps attribute is used to control the speed at which characters are displayed.
- Example
    - Jolteon: [cps=4/]OH NOOOOO[cps default=true/] jk lol.

Voice Attribute

- Signature
    - [voice default=bool/]
        - default: If set to true (or false), the loop speed of the voice will reset to the default speed.
    - [voice speed=int/]
        - speed: How many times per second the voice should be played.
    - [voice mute=bool/]
        - mute: Set to true or false to have the voice on or off respectively.
- Description
    - The voice attribute is used to control how the voice sounds (i.e its speed or muting) during a line.
- Example
    - Sylveon: [cps=3/][voice speed=3/]<color=red>YOU'RE NEXT #emotion:dark_normal
    - Voice Attribute.mp4
    - Sylveon: Now you can hear me![voice mute=true/]
Now you can't hear me![voice mute=false/]
Now you can! #emotion:happy
    - Voice Attribute 2.mp4
- Usage Tips
    - In most cases, you want the voice to be set to default. As long as your cps is set above 10, allowing CLV to simply loop the voice will sound the best. The recommended length for the voice clip is 0.5 seconds.

    - For very low cps's, consider changing the speed of the voice to match the cps. Slow dialogue can look really dramatic, but is often off-putting if coupled with normal looping audio.

Done Attribute

- Signature
    - [done/]
        - No properties.
    - [done nw=bool/]
        - nw: If set to true (or false), the dialogue will automatically go to the next without player interaction.
- Description
    - The done attribute signals that the dialogue stops its animation immediately, and considers that as the "end of the line."
- Example
    - Sylveon: The FitnessGram™ Pacer Test is a multistage aerobic capacity test that progressively gets more difficult as it continues. The 20 meter pacer test will begin in 30 seconds.[done nw=true/]
    - Sylveon: Line up at the start. The running speed starts slowly, but gets faster each minute after you hear this signal. beep A single lap should be completed each time you hear this sound. Ding[done nw=true/]
    - Sylveon: Remember to run in a straight line, and run as long as possible. The second time you fail to complete a lap before the sound, your test is over. The test will begin on the word start.
    - Done Attribute.mp4

Fast Attribute

- Signature
    - [fast/]
- Description
    - The fast attribute cancels all text animations up until that point. It's recommended to use alongside the done attribute.
- Example

- Sylveon: Cause you've got the body and charming looks![done/]
Not the intelligence. #emotion:joy
- Sylveon: Cause you've got the body and charming looks![fast/]
Not the intelligence. #emotion:nervous
- Fast Attribute.mp4

3.2) Unity Richtext Attributes

CLV supports the usage of richtext attributes! In general, this means bold text, italic text, changing font size, and changing color. More information on Unity's Richtext system can be found here. (https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/StyledText.html)

3.3) Tags

In CLV, tags are primarily used to control what emotion/icon to use for a character. Some sub-uses (singular) is controlling the voice.

Emotion Tag

- Signature
  - #emotion:current_emotion
    - current_emotion: The name/emotion associated with the icon to display.
- Description
  - The emotion tag is used to change what icon/emotion to display for the character.
- Tips
  - You don't need to use the emotion tag for every line. CLV remembers the last emotion used by a character, and will keep using it (even if other character speak as well) until the emotion is changed.
  - However, at the start of a node it's good practice to explicitly give your characters emotions. You don't know what node your jumping from, and even if you do it makes it easier to test specific scenes from the editor without needing to replay your whole game.

No Voice Tag

- Signature
    - #no_voice
- Description
    - Adding this tag will mute the character's voice at the start of the line. This can be over-ridden at any point by unmuting via the voice attribute.

3.4) Commands

CLV supports interspersing your dialogue with cutscenes, and this is done via commands.

Animate Command

- Signature
    - <<animate animationName>>
        - animationName (string): The name of the timeline asset to play.
    - <<animate animationName async>>
        - animationName (string): The name of the timeline asset to play.
        - async (bool): If true, dialogue will continue displaying during the cutscene. If false, the dialogue box will hide itself, let the animation play, and re-display again (during which the player can't skip the cutscene). By default, this is set to false.
- Description
    - Plays a timeline asset stored in the current AnimationHandler game object (see section 2.2). In most cases, this is just the AnimationHandler under CLV Dialogue System.
- Tips
    - Each call to animate acts as a "syncronization point." That's to say, if any timeline is currently playing when the animate command is called, the previous timeline will skip to the end and the new timeline will begin playing.
    - Since AnimationHandler is designed to play timeline assets, naturally AnimationHandler has a PlayableDirector component. Be sure to use it when creating your timeline assets, and don't forget to drag n' drop

your timeline onto the “Animation Timeline” property in AnimationHandler’s script component.

Music New Command

- Signature
  - <<music_new musicName fadeIn fadeOut)
    - musicName (string): The name of the music to play (as defined in the Music Handler Gameobject).
    - fadeIn (float): How many seconds to fade in the volume for the new music. If unspecified, value is 0.0.
    - fadeOut (float): How many seconds to fade out the volume for the old music. If unspecified, value is 0.0.
- Description
  - Play a new music and swap out the old (if any exist).

Music Resume Command

- Signature
  - <<music_resume fadeIn>>
    - fadeIn (float): How many seconds to fade in the volume for the new music. If unspecified, value is 0.0.
- Description
  - Resume the current music if it's paused.

Music Pause Command

- Signature
  - <<music_pause fadeOut>>
    - fadeOut (float): How many seconds to fade out the volume for the old music. If unspecified, value is 0.0.
- Description
  - Pauses the music.

Emotion Command

- Signature
  - <<emotion characterName emotionName>>

        - characterName (string): The name of the GameObject (with a Character component) to change the emotion of.
        - emotionName (string): The name of the emotion to display.
- Description
    - Sets a Character's sprite according to the given emotion.

Move Command

- Signature
    - <<move characterName positionName time>>
        - characterName (string): The name of the GameObject (with a Character component) to move.
        - positionName (string): The name of the position to move to.
        - time (float): How long it should take for the character to finish moving.
- Description
    - Moves/translates a Character to a pre-specified position.

4) User Input

Standard Input

In the package folder, go to “Character Line View -> Input -> CLV Input Manager”. The user of Character Line View is able to advance dialogue via the “Continue” action under the “UI” action map. If you want to check what buttons are supported by default, here’s what you need.

Custom Input

If you want to change what input advances dialogue, it’s recommended that you make your own Input Action Reference rather than edit the pre-existing one. (https://www.youtube.com/watch?v=Pzd8NhcRzVo (The video’s a little outdated, but still one of the most polished and easy to understand)

After making your Input Action Reference, go to Character Line View in your scene tree. Then, drag your input action into “Continue Action Reference” under the Dialogue Advance Script component.

At the end of the day, do whatever you want with the input. If you don’t like Unity’s Input System, you can always change the aforementioned Dialogue Advance Script component to suit your needs. (https://docs.yarnspinner.dev/using-yarnspinner-with-unity/components/dialogue-view/dialogue-advance-input)

5) Extending this Module

This chapter is intended for those that want to extend the functionality of Character Line View. In it, I'll give a few pointers as to how Character Line View is designed, and what places it's extendible in. If you're satisfied with what it already has, or are ok with just making a new Dialogue View through YarnSpinner's system, feel free to skip this chapter.

(also, I'll be a lot more loose in this chapter, as telling you how to extend this module also means I need to be more open about not only CLV's successes but also its limitations. If you have any suggestions for how I can improve as a developer, I'd love to hear your feedback!)

5.1) Prefabs

Not much to mention here. Section 2.2 already covers this extensively, and the unmentioned CLV Option View is just there to make the Option List View look more pretty. If there's anything you'd like me to cover here, let me know.

5.2) Attributes

Here, I'll describe how to create new CLV attributes (section 3.1) for more complex dialogue animations. In order to do so, you'll want to go through the package folder and find "Scripts -> In-text Attributes -> IIntextAttribute". IIntextAttribute is an interface that, when implemented, can be used as an attribute in YarnScript. Here's the method you'll want to implement:

RunAttributeCommand

- Signature
    - public bool RunAttributeCommand(
      CharacterLineView clv, MarkupAttribute attribute, ref Sequence tweenList, ref int currentCps);
        - clv: The CharacterLineView that's parsing this command.
        - attribute: The attribute that's invoking this command, along with any properties.

        - tweenList: The DoTween Sequence that's used to animate the dialogue.
        - currentCps: The current cps at which dialogue is displayed.
- Description
    - RunAttributeCommand is the method that'll run whenever this attribute is called in a line of dialogue.
    - Before this method is called, all "RunAttributeCommand"'s for previous attributes have been called, and the sequence will be up to date to this attribute's current position (that's to say, the sequence can animate the dialogue up till this point).
    - After this method finishes, CLV will automatically animate the dialogue from this attribute's position to the next attribute's position using the "currentCps" argument. Note that, since currentCps is a reference parameter, it can be altered.

Admittedly, the attributes are a lot more tightly coupled with CharacterLineView than I'd like. By passing in clv as a parameter, the attributes can access more pivotal information by calling clv's fields directly. However, if you have any suggestions of what "standard" set of parameters I could pass in for looser coupling, it'd help out a lot.

After creating your attribute, you'll want to go through the package folder and find "Scripts -> CharacterLineViewGlobals". In it, look for the "intextAttributesDict" and add your attribute there. By doing this, CLV will consider your attribute valid and it can then be used when writing your dialogue.

5.3) UI/Scene Tree

Here, I'll describe how you can add children to the Character Line View game object in a way that updates based on the current dialogue line. CLV uses this system to update its character textbox and icon. To do this, you'll want to go through the package folder and find "Scripts -> ICharacterDisplayable". Any game object that implements ICharacterDisplayable can update whenever the player moves to a new dialogue. Here's the method you'll want to implement:

SetDisplayable

- Signature
    - public void SetDisplayable(LocalizedLine dialogueLine);
- Description
    - SetDisplayable is called whenever Character Line View receives a new line to display. “dialogueLine” contains the raw displayed line itself, and can be used to access any associated tags.
- Tips
    - Since dialogeLine gives the raw data, it also may include un-parsed rich text. As such, it’s mostly useful for accessing any metadata and altering your displayable based on that.

6) Closing Remarks/Future Plans

As a game developer, I've always wanted to put stories and characters at the forefront of my games. As such, Character Line View is an important module to me, and I add more features to it every time I make a new Unity game. However, my first priority is still to make games. So, most public releases of CLV will likely coincide with game releases, and I won't be giving frequent updates otherwise. In short, this means two things.

1. If there's any bugs or confusing documentation, please let me know and I'll be happy to add a small update to fix it.
2. If you have any suggestions for cool new features, I'd love to hear it! I can add it into CLV, and publicly release it on the next major update. This way, we can both benefit from better features!

Here's some of my current future plans:

6.1) Planned Features

- Vpunch Tag
    - Another idea from renpy (https://www.renpy.org/doc/html/transitions.html). Having a vpunch/hpunch tag should make it easier to add emphasis to certain lines.
- NoSkip Tag
    - Having this tag will stop the player from being able to skip a line. Useful when trying to time certain dialogues with animations.
- Letter update callback/Better voice
    - Currently voices at slow cps's are implemented via coroutines. However, this implementation has been a little prone to error.
    - However, if I update the code to make use of DoTween's OnUpdate callback, I can check each time whether the letters change or not. If they do, I can call my own OnLetterUpdate callback.
    - Mumbo jumbo aside, this'll not only allow the voices to be more accurate, it'll also allow for it to be customizable! For instance, characters can do their voice whenever letters are displayed, but not say anything if it's just spaces or periods.

6.2) Planned Modules

- AnimatedSceneChanger
    - This module actually already exists, but is currently an MVP (minimal viable product) that only allows fade in/out transitions.
    - In the future, I'm hoping to leverage shaders to create an ImageDissolve effect for scene transitions (https://www.renpy.org/doc/html/transitions.html). This'd allow for fancy transitions to be made at very low production cost, since all you need is to make some black n' white images from Photo Shop.
- RPG Navigation System
    - As a story-game lover, it's no surprise that I like rpg's. Thus, I also plan to make an rpg module where you can play in a top-down 2D environment.
    - I intend to focus more on the navigation side of things, and not as much on any battle systems. (maybe some day)

6.3) Copyrighted Materials

Since copyright is important, I'll dedicate a quick section to mentioning the important points.

When using CLV, it goes without saying that Pokémon characters are copyrighted. Thus, even though the textboxes I've provided aren't explicitly Pokémon related, you may not want to use those default character names. The "voices" I give also come from a Pokemon Mystery Dungeon game, but that's very minor info that you can decide what to do with.

More importantly, the Chalkboard font I provide comes pre-packaged with the mac laptop I used to make CLV. Idk what that implies to copyright, for all I know it may be completely fine, but I mention just in case if you'd rather be safe and use your own fonts for any commercial games..

6.4) Closing

Once again, thank you for taking interest in Character Line View. I've always wanted to develop my own games, and have been working the past few years of my life to achieve that goal. However, this is the first time ever that I'll be releasing my own tools with extensive documentation like this. I sincerely hope you find this useful and, as always, I appreciate any constructive criticism. Thanks for reading, and let's keep making games!

- ArcherZenmi