Visual Novel Engine for Unity

By Michael Long (Foolish Mortals), michael@foolish-mortals.net

http://u3d.as/nLD

Summary

A light weight code base suitable for visual novels and simple cut scenes. Useful for any game that has dialogue between characters. Created because all of the VN frameworks I tried for Unity weren't suitable for my work on my upcoming game Triple-M.

Built using Unity 5.1+ using Unity's UI. Does not rely on any external tools or code.

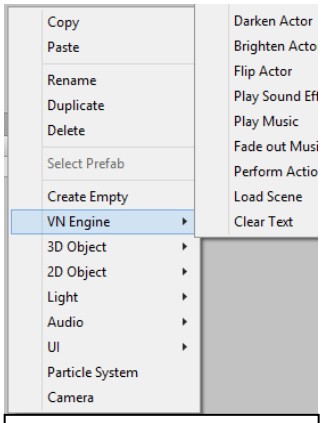
Tutorial

The best way to learn how to use this VN Engine is through the example Tutorial scene included in the VN Engine/Example Scenes folder. It covers almost every feature, and you can follow along in the inspector as it plays through the Conversations. This documentation is more for those wishing a more in-depth and technical understanding.

You can also watch some videos about it.

Setting Up

1. Import all the assets into your project.
2. Right click anywhere in the Hierarchy inspector (the inspector where all of your GameObjects in your scene are) and mouse over the 'VN Engine' sub menu.
3. Select 'Create DialogueCanvas'. Every scene that uses the VN Engine **requires** a DialogueCanvas.
4. Select 'New Conversation' from the 'VN Engine' sub menu. This will create a new GameObjects that will serve as the basis of our conversation.
5. Select the Conversation GameObjects, right click and navigate the 'VN Engine' sub menu again and click on elements to add to this conversation.
6. Expand the 'DialogueCanvas' object and select 'SceneManager'. Drag the newly created Conversation into the 'Starting_Conversation' field.
7. Play the scene and your conversation should run.

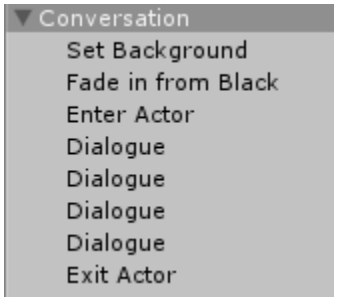


Right clicking in the Inspector to access the VN Engine context menu

DialogueCanvas

All information and interactions with this VN Engine is done through the DialogueCanvas. Feel free to user your own different canvas for other parts of your game. Be careful when changing the DialogueCanvas. If something breaks after a change, try analyzing the original DialogueCanvas, or in a worst case scenario start fresh from the DialogueCanvas prefab. You will most likely want to change the look of the UI.

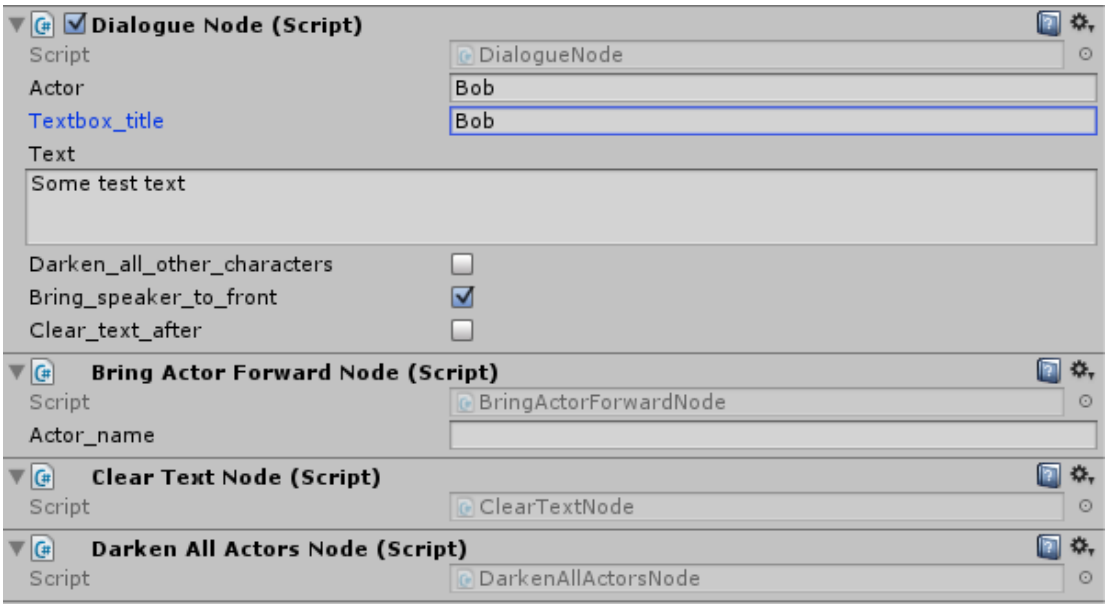
Conversations



All VN Engine instructions are carried out through Conversations. Any GameObject with a ConversationManager script attached to it is a Conversation. All children of this base Conversation object may contain directions. Any Node type script may be placed as children of Conversations.

Nodes are scripts that inherit the Node class. View Node.cs for more information. Nodes are the individual instructions carried out in a conversation. Consider them like stage directions. Have an actor appear, have them say their lines, change background, etc.

ConversationManagers uses Unity's GetComponentsInChildren to find all scripts that inherent Node in its children. This means you can either place Nodes as their own child gameobjects, or add multiple Node scripts to the same object. In this case, instructions are executed going top to bottom.

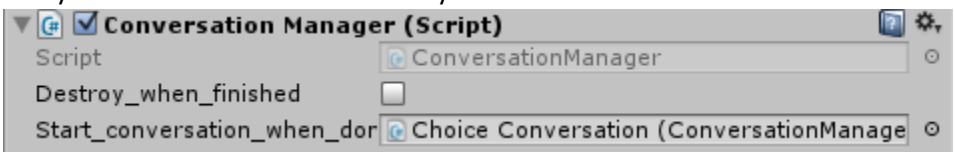


The order of execution of the above example would be: Dialogue, Bring Actor Forward, Clear Text, Darken All Actors.

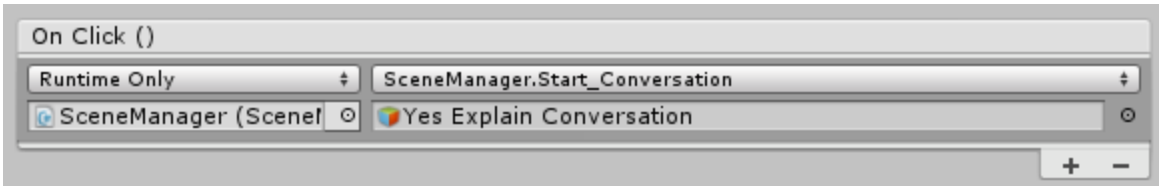
Starting a Conversation

There are various ways to Start a conversation.

- At the start of a scene: A conversation can automatically be started when the scene loads by dragging a Conversation into SceneManager's 'Starting Conversation' field.
- Every Conversation can automatically start another Conversation when it finishes.



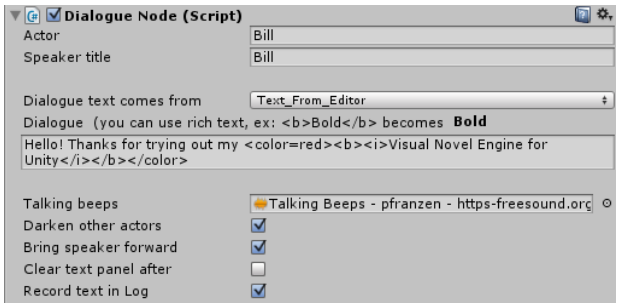
- When you right click and select the VN Engine menu, you can create a Change Conversation Node
- When a UI Button is clicked: Select the Button, go to the 'On Click' fields and click the Plus sign. Drag the SceneManager object into the newly created field. Select SceneManager against from the leftmost dropdown, then select the method 'SceneManager.Start_Conversation'. Then drag in the GameObject containing your conversation.



- When a GameObject is clicked: Drag the 'ClickToStartConversation.cs' script onto your GameObject. Then drag in the conversation object you want started into the 'conversation_to_start' field of the script.
  Note: Object must have a Collider or Collider2D in order for clicks to be registered.
- You can call ConversionManager.Start_Conversation() to start that Conversation

Dialogue Nodes

Dialogue is the bread and butter of any visual novel. Dialogue Nodes print out the given dialogue one character a time. They can be linked with Actors on the scene, so the speaking Actor brightens, is moved to the front, or other Actors darken.



Dialogue Nodes support Unity's Rich Text formatting. https://docs.unity3d.com/Manual/StyledText.html.
***Note**: Tags must be closed IN ORDER.*

Ex: <color=red>Visual Novel Engine for Unity</color> becomes ***Visual Novel Engine for Unity***

Dialogue Nodes always have an AudioSource on the same object. This AudioSource is automatically played, and stopped with the running of the Dialogue Node. This is suitable for playing sound effects associated with the dialogue (a knocking on the door), or voiced lines.

In addition, Dialogue Nodes also have a field called 'Talking Beeps'. These are ***extremely short*** audio clips played every time a text character is printed out. Think of it like the Actor is talking. This is featured in some games like 'Undertale'.

Actors

Dialogue is normally spoken by characters who are on-screen. These character images are called Actors. Actors are stored in: Resources/VN Engine/Actors. This package includes several actors. The key component of an Actor is the Actor script attached to it. Actors have an Actor script and an Image. The name of an Actor in the Actor script is very important. Ensure that the Actor name in the Actor script is the same as the GameObject's name.

To create your own Actor, copy one of the existing Actors, change the image and scale appropriately, and change the Actor's name in the Actor script and its GameObject name.

NOTE: Image size is important, and should be considered carefully when deciding how many characters you want on-screen at the same time. It is recommended to use the 'Set Native Size' button on the Image of the actor, and then setting the Rect Transform's scale so it fits properly on the screen.

Actor Positions

Actors are positioned on either the LEFT, RIGHT or CENTER of the screen. You may place as many characters on either side of the screen as you like, but any more than 3 on a side may look crowded and start overlapping, and having too many characters in the CENTER can squish the LEFT and RIGHT a bit. It is recommended to have 6 or fewer characters on screen at the same time.

Actors will automatically adjust their positions when Actors enter or exit the scene. They will try to provide as much spacing as possible between them, while attempting to stay on their half of the screen. If you wish to adjust how much space each Actor gets when in the CENTER position, change ActorManager.cs's `space_per_actor_in_CENTER` space variable.

1 Character on the LEFT, 1 in CENTER, 1 on RIGHT

2 Characters on the LEFT, 2 on the RIGHT

***Note**: Actors no longer use the GameObjects Left and Right inside of ActorPositions (which have been removed). They simply use OFFSCREEN_LEFT and OFFSCREEN_RIGHT to determine when it's out of the user's sight to remove them.*

Actors can also be placed in custom positions. To do so, create an empty game object on your canvas where you want the Actor to appear, and drag that empty game object into the 'custom_position' field of the EnterActorNode (is only visible if you have the 'Destination' field set to 'CUSTOM').

Escape Menu

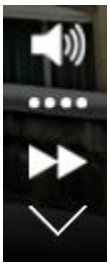
Hit the 'Escape' button on your keyboard to see the pause menu. Alternate keys to access this menu can be specified in the PauseManager ('Toggle_pause_key') and can be changed in the Unity project's input settings.

In the menu you adjust levels of volume for sound effects, music and master volume, the speed at which characters are displayed when characters are speaking their lines and the font used in the dialogue text box.

You may access the Volume in code by looking at AudioManager.cs. You may access the text display speed (text scroll speed) by looking at VNSceneManager.cs. You will most likely want to change the look of this settings menu.

All aspects in this Escape Menu are saved using Unity's Player Preferences.

Sidebar



Mute: Mutes all audio by setting the main camera's AudioListener volume to 0.

Auto: Automatically proceed through dialogue when the text is done displaying AND/OR the voice clip associated with the dialogue is done.

Fast: Hold down the 'Fast' button to quickly fast forward through dialogue.

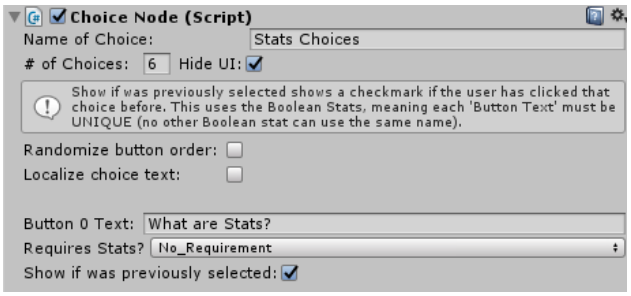
Hide: Hides all UI in the DialogueCanvas until the user clicks any button.

Menu shown at the bottom right hand side of the text window.

Log: Shows a log of all text shown in the dialogue text box. Can also be accessed by moving the mouse scroll wheel over while over the dialogue text area. Can be accessed via variable 'Conversation_log' in VNSceneManager.

Items: Brings up the Items menu. See ***Items***

Choices



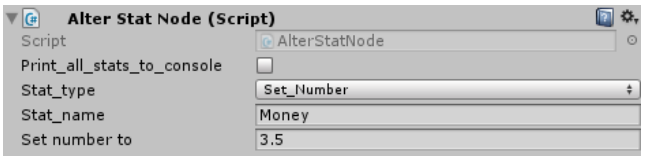
Choices are used to present the Player with a series of buttons, each representing a choice. Each Choice node can have up to 20 choice buttons. Choice buttons can be hidden or disabled depending on Stats or destroyed GameObjects.

'Show if was previously selected' displays a checkmark (or other image) next to any choice button you've clicked before. These are recorded using Stats (see below). When a Choice button is clicked, a boolean stat called: [Name of choice]: [Button text] is set to true. For the choice in the image above, the Boolean "Stats Choices: What are Stats?" would be set to true if the user clicked that choice button.

The text in Choices can be localized if you check the 'Localize choice text' toggle. *See Localization*

Stats

Stats are used to record things in-game. Ex: Player's money, their Strength stat, progress/decisions made, etc. You can manipulate stats using AlterStatsNodes (or right click/VN Engine/Alter Stats). Stats are used in Choices and IfNodes.



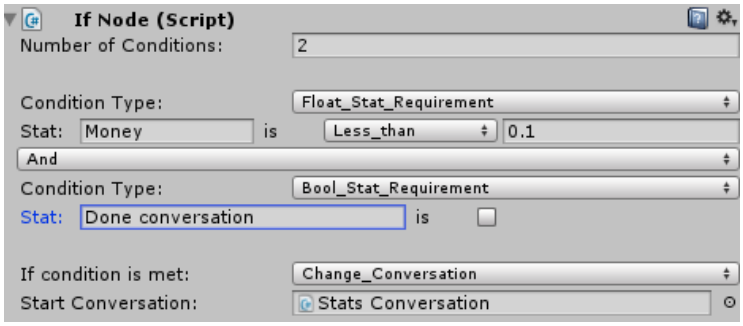
There are three types of Stats: floating point numbers, booleans and strings. These Stats are automatically saved and loaded, and exist between scenes. They can be accessed by the static class StatsManager.cs

You can print out stat values in dialogue nodes with the following syntax:

[f:floating_point_variable_name] prints out a floating point number stat. Floating point numbers are numbers that can have decimal values.
[b:boolean_variable_name] prints out a Boolean stat. Boolean stats are either True or False.
[s:string_variable_name] prints out a string stat. Strings are any combinations of characters (like words).

If Nodes

If nodes are for conditional actions. If the conditions are met, some action is performed. These conditions can be upon Stats, or destroyed GameObjects. Actions can either be changing conversations, jumping to the middle of a conversation or custom actions (programmed like using a standard Button).



If multiple conditions are listed, they are separated by AND's and OR's. They are evaluated in order from top to bottom. This means the last condition is the most important one. More documentation in IfNode.cs.

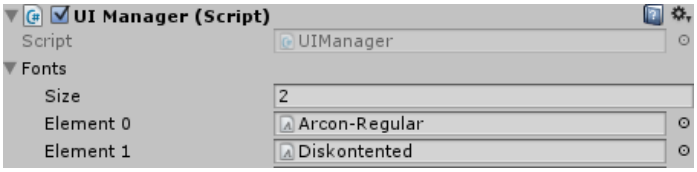
Items

The Item Menu is a menu where the player is able to click on small pictures of Items and view their description. To disable this feature, either delete any objects related to Items or remove the Item category button at the top of the in-game menu. ***Items are accessed by ItemNodes.***

All Items are required to be Prefabs in a ***Resources/Items*** folder, are UI Buttons, and have the Item script attached them.

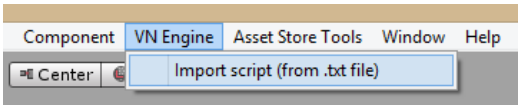
Fonts

The pause menu provided with the VNEngine allows the changing of fonts for the dialogue text. To populate this menu, place all fonts you want to use in any ***Resources/Fonts*** folder. Then navigate to DialogueCanvas/SceneManager object, and drag your fonts into the Fonts array of the UIManager.



The fonts provided are free for commercial use, and their licenses are included in the Fonts folder.
http://www.1001fonts.com/arcon-font.html
http://www.1001fonts.com/diskontented-font.html

Importing Dialogue from a .txt file



You now have the option to import simple dialogue and conversations from a .txt file. This is useful if you don't want to write the script inside of Unity itself. In the VN Engine/Text folder there is a file called 'ImportingDialogueExample.txt'. This file shows all relevant information and features for importing from a file. If you wish to extend the importing functionality (and add more features), you can edit `VNEngineEditor.cs` and change the `ImportTxtScriptFile()` method.

Localization

Localization is the process of modifying your application for different languages. Making a high quality application in different languages will take a lot of work. The VN Engine has some features that allows you to write entries in different languages in a CSV file, which will then be fetched in the current language settings.

Set_Language() in UIManager.cs allows you to set your language in-game. The default language is English. You can set these values directly yourself by accessing LocalizationManager.cs, which is a static class. CSV's are reading using CSVReader.cs, which outputs a Dictionary containing 1 Dictionary for each language. The Language is automatically saved and loaded using Unity's Player Preferences and can be seen in UIManager.cs.

The user can change their language using the pause/escape menu. A restart is required after a language change.

These Dictionaries are accessed by Keys. Each row in the CSV must have a UNIQUE key. When you supply the Key for the row you want, the text for that row in the current language will be fetched.

The UI is automatically translated using the Localized_UI_CSV field in UIManager. To make a Text element be translated, add a row to the UI Localization CSV (with a key), then add a 'Localize Text Element' script to it and fill in the Key field.

Dialogue Nodes can use localized text from the Localized_Dialogue_CSV in the VN Scene Manager. You will have to enter the Key for the row of dialogue you wish to use.

Example CSV's for location can be found in the VN Engine/Localization folder.

CSV's must be saved in UTF-8 format, or characters with accents such as é will not be properly imported.

Saving & Loading

The major addition to V2.0 is the saving and loading of VN elements of the scene. This saving feature DOES NOT SAVE EVERYTHING. Saving can be done from the Escape/Pause menu. Below is a list of what elements are saved on the scene:

- The current node of the current conversation
- Which conversations have been deleted/completed
- Play time
- Time saved
- Background music *(must be stored in Resources/Music)*
- Background image *(must be stored in Resources/Backgrounds)*
- Static Images
- Actors on the scene (and their positions)
- Text log (history of all text displayed in the dialogue box)
- Items (viewed in the ItemPanel) *(must be stored in Resources/Items)*

If using saving, make every SAVEABLE FEATURE NODE have its own uniquely named gameobject (instead of a bunch of Node scripts on the same gameobject)

How does it work?

Nodes that have features that require saving (setting a background, setting the music, changing the actor's image) call the following code:

```
SaveManager.SetSaveFeature(this, object_you_are_modifying.gameObject);
```

This marks the current Node as a Node that needs to be executed upon loading. If you wish to add new Nodes that have features that require saving, use the above line.

If your game requires additional saving logic, you must add it to `SaveFile.cs` in the commented out sections saying '`MODIFY THIS SECTION TO LOAD THINGS SPECIFIC TO YOUR GAME`'. `SaveFile.cs` and `SaveManager.cs` are the key classes for saving and loading. For loading to work, your scene must be added to Unity's Build Settings.

Saves are stored by serializing list of `SaveFile.cs` to the user's hard drive. SaveManager.cs handles this job. By default, the file is saved as `Application.persistentDataPath/saved_games.gd`.

Saving

Below is some c# code for saving. It's that simple.

```
SaveFile s = new SaveFile();
s.Save();
```

Loading

To get the SaveFiles from your saved_games.gd file, you will need to call `SaveManager.LoadFromFile()`. Then it is advised to let the user choose from the SaveFiles to load. After the user has decided on SaveFile to load, call that SaveFile's `Load()` method. The saved scene will automatically be loaded after this method is called.

An example of a load saves menu is in the Example Scenes/LoadingSaves scene. This scene uses a class called `UILoadSaveController.cs` to create 1 UI button per save file.

You should add a GameObject with the ResetStats script attached to it in your Main Menu, to ensure your Stats get reset between Loads.

Debug Logging

Many debug messages only appear if you've set `verbose_debug_logs` to true in your VNSceneManager.cs. Set it to false to receive fewer messages in your debug console.

Extending the Engine

To add more types of Nodes, copy and paste NodeTemplate.cs, change the name of the class and fill out the 3 methods. View Node.cs for more information. Look at the other Node scripts for examples. This is meant to be extremely simple.

To change what keys are used for proceeding through dialogue, Select the SceneManager in the DialogueCanvas and change the next_button's and superspeed_button field.

Credits

Coding: Michael Long

Character Sprites: konett, shared under CC-By 3.0 https://creativecommons.org/licenses/by/3.0/ No changes have been made to the sprites.

http://lemmasoft.renai.us/forums/viewtopic.php?f=52&t=24893

If you wish to use the character sprites, you must attribute the above author and provide a link to the CC-By 3.0 license.

Talking Beeps Sound: from pfrazen at https://freesound.org/people/pfranzen/sounds/267328/ shared under the https://creativecommons.org/licenses/by/3.0/ CC-By 3.0 Attribution license. No changes have been made.

Contact Me!

Michael Long, michael@foolish-mortals.net, http://foolish-mortals.net

Patch Notes

V2.5.3. (9/26/2017)

- Added ability to localize Choices

V2.5.2 (9/3/2017)

- Fixed bug where ChangeActorImageNodes wouldn't work after an actor has exited and entered again
- Created better interface for 'EnterActorNodes'

V2.5.1 (6/11/2017)

- Fixed bug where looping conversations broke after the 2nd loop
- Fixed bug with DialogueNodes not having an AudioSource
- Added ability for EnterActorNodes to place Actors at custom locations
- Further improved save functionality by relying less on the Resources folder

V2.5 (5/6/2017)

- Updated UI
- Added icons, sounds to buttons
- Added RPG tutorial scene
- Overhauled the save system (No longer need to place most things in Resources folder)
- Added ability to delete individual saves in LoadingSaves scene
- Added CENTER position for actors
- Added ChangeAudioSnapshot node
- Added asynchronous loading of levels
- Added saving of Autoplay using player preferences
- Added SetAutoplayNode
- Added GenerateRandomNumberNode for Stat checks. Updated tutorial.
- Changed how Actor positions are calculated (no longer stupid!)
- Improved SceneManager options + look in Editor
- Small bug fixes
- Improved performance on Instantiating Actors in large scenes
- Made Actors independent of Canvas size
- Added 'Verbose debug logs' option to VNSceneManager
- Fixed bug with Loading where Items would be duplicated
- Added ActorChangeSideNode
- Added CanSaveNode

V2.4 (1/8/2017)

- Made fading in/out for EnterActor/ExitActor times actually reflect the time the user enters
- Fixed ClearStats() from StatsManager
- Added option to check if a gameobject is active to IfNode
- Fixed bug when actors are lightening

V2.3

- Added a mini log that appears when the mouse scroll wheel is moved and hovering over the text panel
- Vastly improved Log and Items interface
- Improved WaitNode to have a setting to wait until a Boolean is set to true
- Fixed minor bug with fading actors and lightening them immediately afterwards
- Fixed bug about loading saves with the game time being set to 0, and actors being null
- Added String stats (and added them to IfNodes and ChoiceNodes
- Added condition_met/not met field to stat comparisons in IfNodes and ChoiceNodes
- Changed stat comparison logic if the stat isn't found: bools default to false, floats default to 0, strings default to ""
- Added option to show checkmarks if a choice has been selected before
- Added option to choices to hide the dialogue UI
- Added printing of stats to dialogue nodes

V2.2

- Added 'Documentation' option to open README.pdf in VN Engine top menu
- Added Dialogue Location from a CSV
- Added UI Localization from a CSV
- Fixed UnityEditor errors when building
- Added importing a dialogue via .txt file
- Added proper line wrapping for DialogueNodes when printing out characters (words will no longer jump to the next line if they need more space)
- Added rich text formatting to DialogueNodes
- Added 'Talking beeps' option to DialogueNodes (think characters talking from Undertale)
- Updated Choice UI
- Added sliding in UI animation

V2.1

- Hotfix for fixing missing prefabs from the right click VN Engine context menu
- Added Hide/Show UI nodes
- Added option to hide UI in the VN Scene Manager at scene start

V2.0

- Added Saving and Loading of entire scenes!
- Added saving of Audio/Text settings using Player Preferences
- Added Font size select in menu
- Added Font selection to pause menu and automatic saving/loading of Font preference
- Added Items
- Added Stats
- Vastly improved Choice nodes
- Added If nodes
- Added PerformAction nodes
- Fixed issue of audio being muted
- Added audio muting options to pause menu
- Improved error logging