**“Encoded” – Visual novel Scripting Language Documentation**

**February 2018**

Encoded Ver. 2.0

Unity Ver. 2017.2.0f3

**Joey Tong**

**Abstract**

Encoded is a command based scripting language purpose built for creating interactive visual novel experiences for PC/Unity. It is in essence an interpreter for Unity that streamlines the process of doing things like setting up scenes with sounds, backgrounds, animated 2D sprites, transitions, etc. Commands are interpreted from top to bottom, left to right. The parser detects unique terms separated by spaces. (‘ ‘) Commands cannot span multiple lines, all syntax and logic must be on the same line. Encoded makes use of a **Label** system, **Labels** mark points in the script to which you can go to at any time using the **Jump** command OR through use of the **Jump** command embedded within the **Choice** command. (See more in **Branching/Choices**)

**Interpreter**

The entirety of a script is parsed line-by-line before execution. This is done every time the game loads. Syntax errors will be thrown during this parsing. The parser will also print the parsed command line-by-line during this time. During execution, the interpreter executes parsed commands line-by-line in parallel with the progression of the game. For example, the command “jump myLabel” will be parsed by breaking it into its component parts, the command “jump” and the identifier “myLabel” and placed into a command list to execute. Whitelines are NOT placed into this list. Thus, the commands list indices will NOT correspond to the line your commands are written on in the script file.

**File type & language support**

Script files are saved in the .txt format with UTF-8 encoding, enabling use of characters for most languages. They are built into Unity text assets upon execution of the game. Languages that are known to be supported at this time are English, Chinese, and Japanese.

**Commenting**

You can comment a line of code in the script by using an octothorpe at the start of the line.  
(e.g. “#comment”)

**Commands**

**Choices** – creates a branching point where users are asked to make a choice. Initialization of choices involves specifying the identifier for the choice (used for saving progress, checking choices for game progression), the number of choices available, the display text for every choice, and a jump command to the label a choice should lead to. (See ***jump*** & ***label***)

**define** – Defines and initializes dialogue entities. Requires a display name and an identifier. (Dialogue entities are not bound to on-screen characters.)

**hide** – Removes character entities from the screen. Requires a character identifier. The “All” identifier will remove all characters from the screen. Hide supports fade transitions.

**if**– Used to check past selected choices with a choice identifier and a value.

**jump**– “jumps” to the specified label. Game execution will automatically move to the first line after the specified label. Can jump both forward and backward in the script.

**label**– for creating markers/checkpoints in the script that can be jumped to at any time. Requires a name for the label being created followed by a semicolon

**lighting**– Changes the ambient lighting in the scene. Currently supports “Neutral” lighting, “Evening” lighting and “night” lighting.

**play –** Plays music or sound effects. Requires sound type identifier of “music” or “sound” followed by the file/asset name in double quotes. File extension not necessary.

**Return –** Ends execution of the game loop.

**Scene –** Displays the background for a scene

**Show** – Shows the specified character entity with the specified animations playing. Also used to play animations on an already displayed character. Thus, it does not support multiple instances of the same character entity. This can be worked around by creating copies of the character prefab with different names.

**stop –** Currently Unimplemented