Choice Engine 2

Documentation

Contents

Introduction

Choice Engine 2 is a complete overhaul of the original asset. The reimplementation offers a wider array of features, better optimization, cleaner UI and much more.

The asset can be used with minimal knowledge of unity, and practically no coding experience. However, intermediate knowledge of unity and C# is recommended to maximize the potential of unity and the asset.

Contact Info

If you experience any issues or difficulties, please feel free to contact me directly.

Email: **cinationproject@gmail.com**

Unity Connect: Link

Tips

1) You can inspect most of the variables in the inspector by hovering your mouse over them.

2) Before using the asset make sure you understand the basics of unity. You can achieve this by either getting accustomed to the official documentation on unity's website or following any other online tutorials.

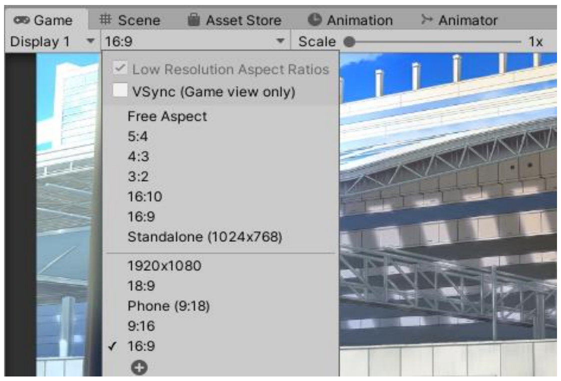
3) Make sure to familiarize yourself with the demo and all of the available functions before creating a new project.

4) All **IDs,** throughout the asset, **must** be unique. Non-unique ids can and **will** cause issues.

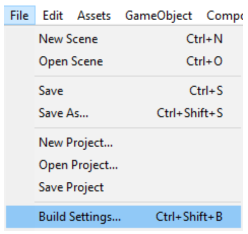
5) Unpack all prefabs before using them. If you see a blue object on the scene, it means that it is a prefab. Prefabs are directly linked with their sources in the assets folder, and that may create problems. To unpack a prefab (unlink it from its source), right-click on the prefab and select **Unpack Prefab Completely"**. Once the object turns grey, you are good to go.

6) In-game U.I. can be freely modified as long as no objects are deleted. If you wish to delete/add new objects, please make sure that you have a good understanding of their usage (a.k.a by which scripts they are used, etc.).

7) The aspect ratio/default resolution of the game can be changed in the **Game** tab.



8) All the new scenes **must** be added to the build setting in the order they appear, before running the game. The Build settings can be accessed via **File -> Build Settings**. A scene can be added to the build settings by opening **it**, opening the building settings and clicking on **Add Open Scenes.**

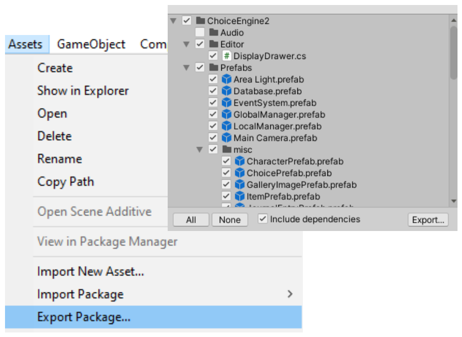


9) All saves are stored in Application.persistentDataPath. You can find the exact location on your device by running the following code in any script:

```
Debug.Log(Application.persistentDataPath);
```

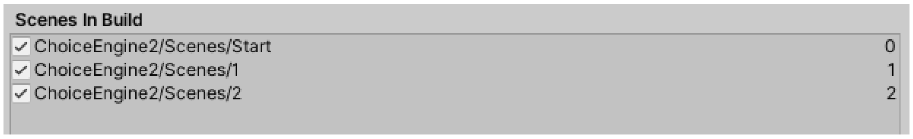
This line will simply output a local directory in the console window.

10) Always backup! It is highly recommended that you back up your project after each development session. You can back up your project as a single file by clicking **Assets -> Export Package.** A window will appear next asking you to confirm what to back up. If your project doesn't show up in its entirety, exit the window, click on the root folder of your project in the inspector, and open the window again. Make sure to click "include dependencies" before exporting.



Testing The Demo

1) Create a simple unity 2D project.
2) Import Choice Engine and make sure to include dependencies if prompted.
3) Open the **Start** scene, located within the scene folder.
4) Add the opened scene to the **Build Settings** (please follow the instructions in the **Tips** section if you are unsure how).
5) Do the same for the rest of the scene in the folder in order of their names.



Scenes In Build	
✓ ChoiceEngine2/Scenes/Start	0
✓ ChoiceEngine2/Scenes/1	1
✓ ChoiceEngine2/Scenes/2	2

6) Open the **Start** scene again and Press the play button at the top of the screen.

7) You're done, you can now freely test the demo and any available features.

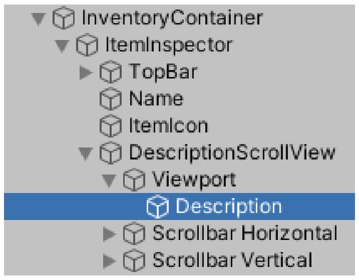
Please note that you should always open the **Start** scene first since all important objects are initiated there.

UI Guide

All the UI elements are by default located under the **Canvas** object. To keep things organized, elements are separated into groups, with each group being independent of the rest. For example, the entire inventory system is stored under the **Inventory Container** object, keeping the inventory UI separated from other modules. All of the scripts that handle UI elements are directly attached to the canvas object. The scripts with the "window" suffix are responsible for displaying various UI elements (usually dynamically generated), while scripts with the "manager" suffix host the functions utilized by these elements.

We will not be exploring each object under the **canvas** window, as most of the objects are self-explanatory, but we shall highlight some important/non-intuitive aspects.

1) In most cases, lists and descriptions are all located in scroll views. **Scroll-Views** allow users to scroll through content, as the name implies, and are perfect for list generation and long walls of text. The image on the right depicts an unfolded scroll view, under the Inventory container object. You can read more about the structure and function of scroll view on the official unity page, but the main takeaway should be that the content of the scroll view itself is always located under the viewport of the scroll view. For example, the description object on the previous image, which contains a text component, can be found at **DescriptionScrollView -> Viewport -> Description.** The same logic applies to all other scroll views.

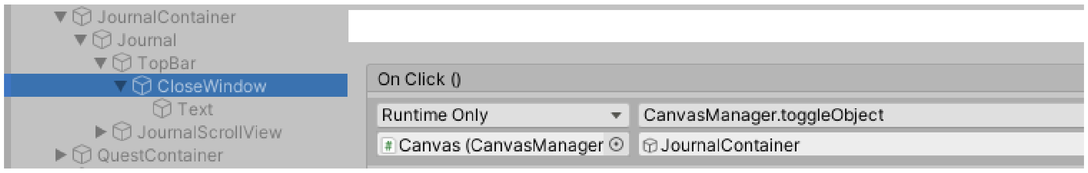


```
▼ InventoryContainer
  ▼ ItemInspector
    ▶ TopBar
      Name
      ItemIcon
    ▼ DescriptionScrollView
      ▼ Viewport
          Description
      ▶ Scrollbar Horizontal
      ▶ Scrollbar Vertical
```

2) The **ClickCatcher** object is used to detect clicks/taps anywhere on the screen. It is primarily used to detect events when either the **Skip** or **Auto** functions are active. It performs the double action of not allowing the player to click on any UI elements while the functions are active and stopping their execution once the player clicks anywhere on the screen.

3) The **Overlay** object is used as a fading screen when loading levels. It simply goes from transparent to black and back, while also preventing the player from clicking anything in the meantime.

4) Buttons are generally configured in two ways. Ones that are responsible for toggling windows (aka disabling and enabling objects) are usually directly assigned functions via the inspector.



```
▼ JournalContainer
  ▼ Journal
    ▼ TopBar
      ▼ CloseWindow
          Text
    ▶ JournalScrollView
▶ QuestContainer
```

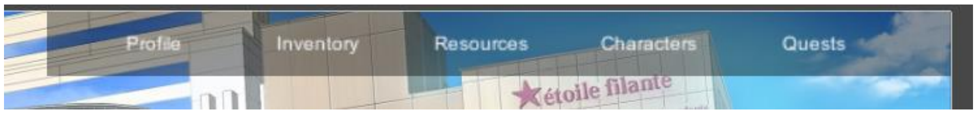
On Click ()

Runtime Only | CanvasManager.toggleObject

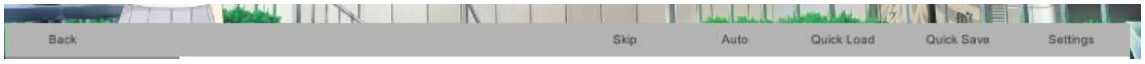
Canvas (CanvasManager | JournalContainer

However, must button, especially ones generated automatically, are assigned their functions via scripts.

5) By default, you can access the **Journal**, **Characters, Quests, Inventory, Resources,** and **Player Profile** tabs via the menu in the top right section of the screen.



You can access common features such as **Quick Save/Load, Back, Skip/Auto** and **Settings** via the menu above the main text area.



Lastly, you can open the quick access menu by pressing **escape** on your keyboard (The bindings can be changed via the **KeyManager.cs** script).

You are encouraged to fully customize any of the latter UI elements to fit your project.

Important Scripts

In this section, we will take a closer look at the scripts that comprise the asset. This is a generic overview, so we will not be inspecting the technical side of the scripts, nor each variable and function individually. Remember that you can always inspect each variable on the inspector by hovering over it with your mouse.

1) **[LocalManager] LevelManager.cs**: Level Manager is the only non-unique script of the asset. While other objects/scripts can only have one instance which exists throughout the play-session, the level manager is designed to be present on each separate game scene. The script itself is responsible for hosting level-specific data, such as the **nodes** list and the **starting** node variable for each scene.

   It is worth mentioning that the nodes list is hidden by default since unity is not really good at coping with large lists. You are instead encouraged to use the **Visualizer** window which can be accessed via **Window -> Visualizer.** The latter allows easier access to the nodes list, while also reducing the load on the memory and providing additional features (see more in the visualizer section).

2) **[GlobalManager] NodeManager.cs**: The Node Manager is a script responsible for handling all the node related functionality like displaying nodes and choices, verifying node access requirements, removing/adding inventory, modifying character affection and status, and more

3) **[GlobalManager] UserFunctions.cs**: As the name suggests, this script is the container for custom made functions. For instance, when entering a node, you have the option to specify the functions to

be triggered via the **FunctionsToTrigger** list, under the **OnNodeEnter** section of the node. The aforementioned list is used to invoke functions from UserFunctions.cs directly.

4) **[GlobalManager] SoundManager.cs**: The sound manager is responsible for handling audio sources, playing and fading audio, alongside with any other events associated with audio playback.

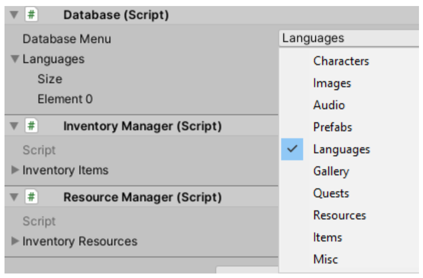
5) **[GlobalManager] TagManager.cs**: The tag manager allows you to create custom tags. The tags can be of any form and will be replaced during runtime. At the moment there are three types of available tags: **Character name**, **Player name**, and custom. Depending on the type of ttag, you will need to specify the appropriate information. For example, for a tag of type **CharacterName**, you would need to specify the ID of the character whose name you wish to replace the tag with. Choosing **PlayerName** as the tag type would replace the tag with the current name of the player character.

   By default, only the node text is filtered for tags. However, you can freely use the **replaceTags()** function to replace the tags in any string in any script. For example, if you want to replace the tags in string **X** inside some script, you could use "**X = TagManager.core.replaceTags(X);".** Please pay attention to the fact that whatever you want to use a function from a different script, you need to reference that script in some way. In this case, the tag manager has a reference to itself called core (just like most scripts in Choice Engine), and thus we reference that script as "TagManager.core" before referring to any functions within.

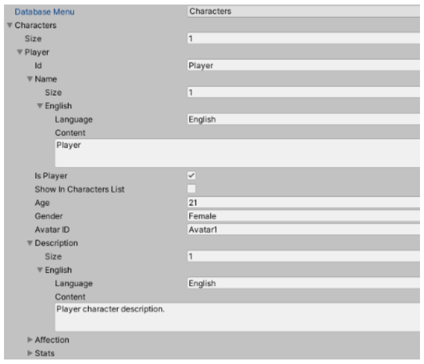
6) **[GlobalManager] KeyManager.cs**: This script handles key bindings. If you wish to bind a key to a specific function within **UserFunctions.cs**, you can simply specify the name of the function that you wish to trigger, alongside with the tag of the key. Please refer to the official documentation if you are uncertain how the keys are labeled within unity.

7) **[Database] Database.cs**: The database accommodates most of the configurable elements in Choice Engine, apart from the nodes and choices. The database has a dual purpose. First of all, it serves as a hub for easier access to all data. Secondly, it hosts non-serializable objects such as sprites and audioclips. This is important because lists of classes containing said objects cannot be exported/saved. Thus, it is way more efficient to reference these objects by their database indexes rather than using them directly. For example, if you want to change the current background when entering a node, you would specify the ID of the background image in the database, rather than using the image itself. This way, you only need one instance of a sprite or audioclip which can then be referenced for a variety of purposes.

   Sections of the database can be accessed via the navigation menu at the top of the script.

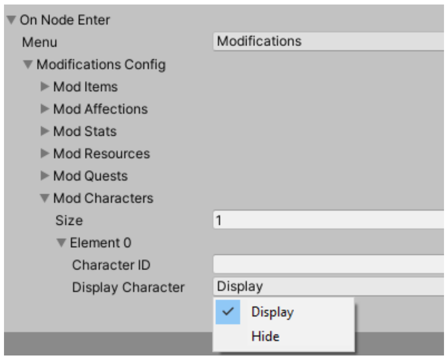


1) **Characters**: A list of all characters.

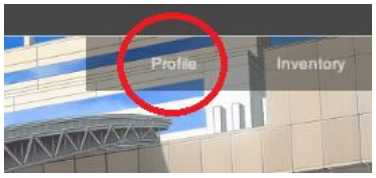


```
Database Menu                Characters
▼ Characters
    Size                     1
  ▼ Player
      Id                     Player
    ▼ Name
        Size                 1
      ▼ English
          Language           English
          Content
          Player
      Is Player              ✓
      Show In Characters List
      Age                    21
      Gender                 Female
      Avatar ID              Avatar1
    ▼ Description
        Size                 1
      ▼ English
          Language           English
          Content
          Player character description.
    ▶ Affection
    ▶ Stats
```

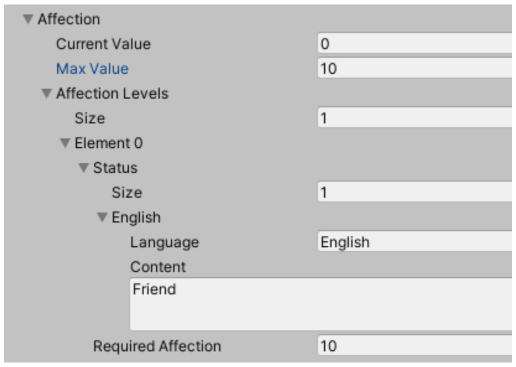
Characters in this list will appear in the **Characters Window.** If the character is marked as a **player** or if **ShowInCharactersList** is set to false, the character will not be displayed. Keep in mind that you can always access the player's profile separately, and you can also "unlock" any character in the game by utilizing **OnNodeEnter** events.



```
▼ On Node Enter
  Menu                       Modifications
  ▼ Modifications Config
    ▶ Mod Items
    ▶ Mod Affections
    ▶ Mod Stats
    ▶ Mod Resources
    ▶ Mod Quests
    ▼ Mod Characters
        Size                 1
      ▼ Element 0
          Character ID
          Display Character  Display
                             ✓ Display
                               Hide
```

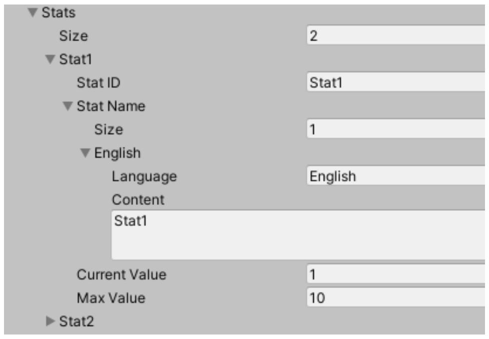


Character affections can be configured via the **Affections** list. Each character can have a maximum affection value and separate affection levels. Whenever an affection threshold is reached, the **Status** of the character will change. You can freely modify character affections by utilizing the **Mod Affections** list on any node on **OnNodeEnter** events.



```
▼ Affection
    Current Value            0
    Max Value                10
  ▼ Affection Levels
      Size                   1
    ▼ Element 0
      ▼ Status
          Size               1
        ▼ English
            Language         English
            Content
            Friend
        Required Affection   10
```

Character stats can be likewise configured via the **Stats** list. Just like with affections, stats can be modified in-game during **OnNodeEnter** events, via the **Mod Stats** list.



```
▼ Stats
    Size                     2
  ▼ Stat1
      Stat ID                Stat1
    ▼ Stat Name
        Size                 1
      ▼ English
          Language           English
          Content
          Stat1
      Current Value          1
      Max Value              10
  ▶ Stat2
```

2) **Images**: A list of all images used in the game. This list is used to store and retrieve images by using their IDs, without the need to reference the image directly.

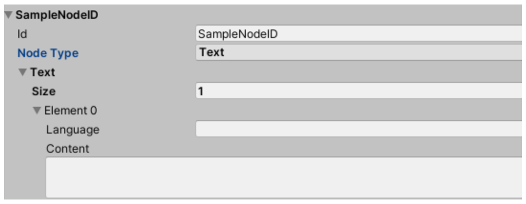
3) **Audio**: A list of all Audio Clips. This list is used to store and retrieve audio clips by using their IDs, without the need to reference the clips directly.

4) **Prefabs**: A list of all Prefabs. This list is used to store and retrieve prefabs using their IDs, without the need to reference them directly.

5) **Languages**: A list of all Languages.

   A notable section of the database is the **Language** section. Choice Engine implements a multi-language system that allows the developer to have their projects translated in any number of languages.

   Each language must have a unique name, which can then be used when configuring any text fields that support the language feature.

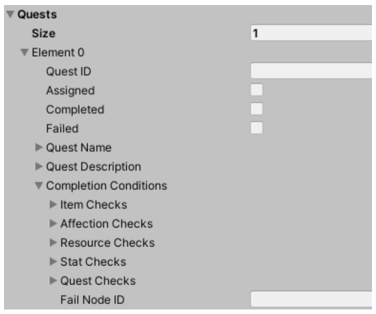


▼ SampleNodeID	
Id	SampleNodeID
Node Type	Text
▼ Text	
Size	1
▼ Element 0	
Language	
Content	

   Text fields that support the feature will be represented by a **list** instead of a single field. The list can be used to create separate elements for each language by specifying the name of the language in the **language** field, and the translated text in the **content field**.

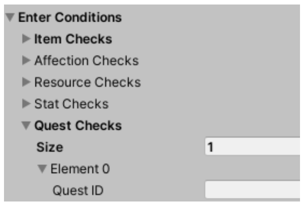
6) **Quests**: A list of all quests.

   Once the quest has been created, it can be assigned to the player in the **OnNodeEnter** section of any node. Quests can be completed and failed either based on the node entered, or on custom conditions defined in the **Completion conditions** section of the quest.



▼ Quests	
Size	1
▼ Element 0	
Quest ID	
Assigned	☐
Completed	☐
Failed	☐
▶ Quest Name	
▶ Quest Description	
▼ Completion Conditions	
▶ Item Checks	
▶ Affection Checks	
▶ Resource Checks	
▶ Stat Checks	
▶ Quest Checks	
Fail Node ID	

   In addition, quest completion can be used to restrict access to a node via the **Enter Conditions** of the node.



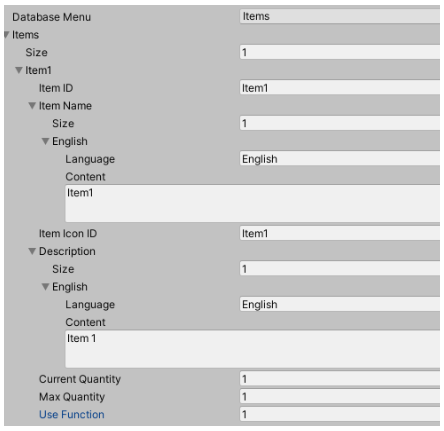
▼ Enter Conditions	
▶ Item Checks	
▶ Affection Checks	
▶ Resource Checks	
▶ Stat Checks	
▼ Quest Checks	
Size	1
▼ Element 0	
Quest ID	

   You may also notice that the entry conditions of a node and the completion conditions of a quest are identical. This has been done on purpose, to keep quests independent of nodes. Either or both can be used to determine quest completion.

7) **Gallery**: A list of all unlockable CG.

8) **Resources / Items**: Items are resources that are of a similar nature but are used for different purposes. Items are generally designed to be objects, while resources are envisioned to be used as currencies.

   Items are resources in the database are **not** reflected in the player's inventory. If you wish to add items/resources directly to the player's inventory you can either do that by utilizing the **Mod Items / Mod Resources** lists on **OnNodeEnter** events, or you can directly add them to the **Inventory Items / Inventory Resources** lists, attached to the database object.



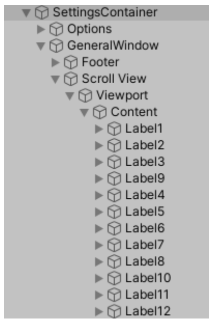
```
Database Menu              Items
▼Items
  Size                     1
  ▼Item1
    Item ID                Item1
    ▼Item Name
      Size                 1
      ▼English
        Language           English
        Content
        Item1
    Item Icon ID           Item1
    ▼Description
      Size                 1
      ▼English
        Language           English
        Content
        Item 1
    Current Quantity       1
    Max Quantity           1
    Use Function           1
```

   Besides their basic characteristics, both items are resources can be used to invoke functions from **UserFunctions.cs.** This can be achieved by specifying the name of the functions to be invoked in the **Use Function** field. The player will be able to "use" the item by opening the item in the inventory and clicking on the **use** button.

8) **[Database] InventoryManager.cs** and **ResourceManager.cs:** These scripts handle the acquisition, modification, and removal of items are resources respectively.

9) **[Canvas] BackgroundManager.cs:** The background manager handles all background transitions and allows you to select the type and speed of background transitions, as well as the starting background.

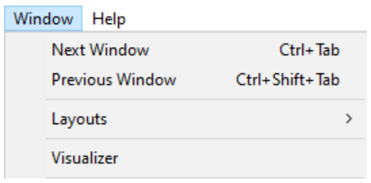
10) **[Canvas] Settings.cs:** This script handles all the in-game settings. You can freely pre-configure all of the available settings, but keep in mind that each option can be also configured in-game. The visualisual aspect of each parameter can be found under **Canvas -> Settings Container -> GeneralWindow -> Scroll View -> Viewpoer -> Content**.



```
▼ SettingsContainer
  ▶ Options
  ▼ GeneralWindow
    ▶ Footer
    ▼ Scroll View
      ▼ Viewport
        ▼ Content
          ▶ Label1
          ▶ Label2
          ▶ Label3
          ▶ Label9
          ▶ Label4
          ▶ Label5
          ▶ Label6
          ▶ Label7
          ▶ Label8
          ▶ Label10
          ▶ Label11
          ▶ Label12
```

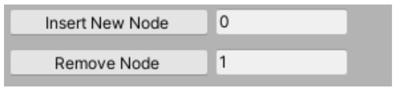
11) **Other scripts:** The scripts not mentioned in the section are purely functional/non-configurable scripts. If you wish to inspect those, or any of the aforementioned scripts more closely, feel free to inspect each script directly. All scripts are well-commented and clearly structured for ease of modifications.

Visualizer



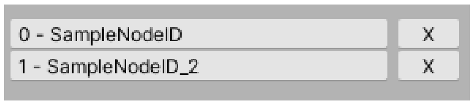
The visualizer component allows you to easily create, delete and manage nodes. You can access the component by clicking on **Window -> Visualizer.**

To be able to use the visualizer properly, please make sure that the **LocalManager** prefab exists on your scene and is unpacked (right-click on the object on the scene, then click on **Unpack prefab completely**. If the option is not there, then you are good to go).

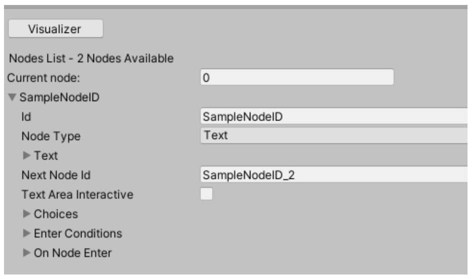


Once you access the visualizer, you will be able to manage the **nodes** list. You can add new nodes at chosen positions in the list and delete any nodes by using the **Insert** and **Remove** buttons.

Please do note that the field next to the buttons allow you to specify the **index** positions of the nodes, not their **ids.** You can easily obtain the index position of a node in a list by looking at the left-hand side of the respective node button in the nodes list.



You can also delete any nodes by directly clicking the **x** button to the right of the node.



Nodes List - 2 Nodes Available
Current node: 0
▼ SampleNodeID
Id: SampleNodeID
Node Type: Text
▶ Text
Next Node Id: SampleNodeID_2
Text Area Interactive
▶ Choices
▶ Enter Conditions
▶ On Node Enter

To the right of the nodes list, is the node-inspector. The inspector has to modes, the default mode, which allows you to edit the node itself, and the visualizer mode, which allows you to see the connections between nodes. You can switch between the two modes by clicking the button on the top of the container.

You can also navigate to a specific node, using its index by changing the value of the **Current node** field. Lastly, you can likewise access any node via the visualizer by simply clicking on it.

Nodes

Nodes are the core of Choice Engine. Each node is comprised of three core aspects: **node information**, **enter conditions**, and "**on enter**" events. We shall look at each one of these aspects in more detail.

Node Information

1) **Node ID:** The ID of the node can be any **unique** string. Node **IDs** are generally used to refer to nodes by both users and scripts. It is important that the ID of each node is unique.

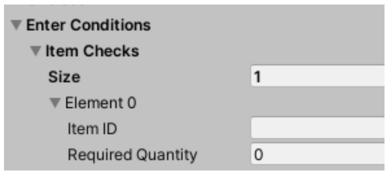
2) **Node Type:** The type of a node determines the content that will be displayed. There are three node types: **Text, Choice,** and **Both.** Text nodes are those that solely displays a message (dialogue) when accessed. Choice nodes display a list of choices to the player. Lastly, nodes categorized as **Both** will display both text and choices to the player.

   Note: When selecting the **Choice** or **Both** types, the text field will not be interactable. In other words, the player will not be able to click on the text field to access the next node, and they will have to select one of the choices/options provided.

Enter conditions

These are conditions that need to be fulfilled in order for the node to become accessible. If the player hasn't fulfilled the required conditions, they will be redirected to another node (Defined via the **FailNodeID** variable).

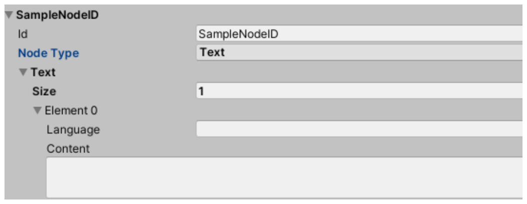
For example, you can utilize the **Item Checks** list to look for specific items. The player will only be able to access the node if the have the specified items in the required quantities.



▼ Enter Conditions	
▼ Item Checks	
Size	1
▼ Element 0	
Item ID	
Required Quantity	0

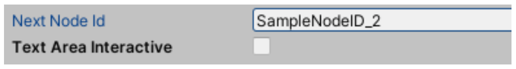
On Enter Events

Once the **entry conditions** have been satisfied, the player will gain access to the node. If the node is a **Text** node, the player will be presented with a message in the text area. This message will correspond with the content of the text field, based on the language chosen by the player (refer to the database section for more information on languages).



▼ SampleNodeID	
Id	SampleNodeID
Node Type	Text
▼ Text	
Size	1
▼ Element 0	
Language	
Content	

Once the player clicks or taps on the main text area, located in the **TextAreaContainer**, under the canvas, the node defined in the **NextNodeID** field will be loaded.

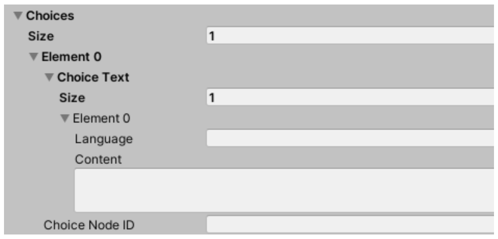


Next Node Id	SampleNodeID_2
Text Area Interactive	☐

It is also possible to prohibit the player from advancing to the next node by making the text area non-interactive.

If the node is set as a **Choice** type node, the player will be presented with a list of options defined in the **Choices** list.

Each option (choice) must have a unique ID (within the same list), and content equivalent to each language. All options will be displayed in the **ChoiceAreContainer**, under the canvas.



Regardless of the node type, additional events can be configured via the **OnNodeEnter** section. We shall take a closer look at each available option:

1) Audio To Play: This list enables you to play any audio from the database by simply using its ID. The engine handles the creation of audio sources and all transitions automatically, and there are no limitations on the number of audio clips playing at the same time. You can also specify the delay before playing any audio clip, whether the audio will be looped or played once, and the category of the audio. The category of the audio is only used for volume adjustment in the settings.

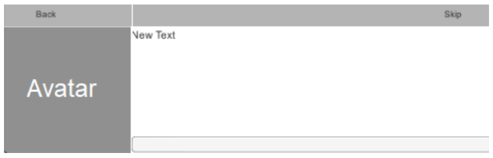
   Note: If you wish to enable/disable or adjust the audio **fade,** please head to **GlobalManager -> SoundManager.cs.**

2) Audio To Stop: This list can be used to stop any currently playing audio clip by using its ID. You can additionally specify a delay before the audio is stopped.

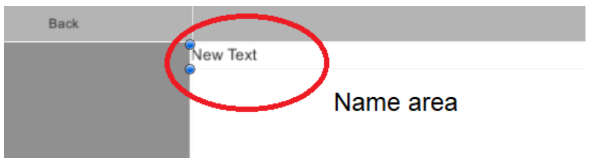
3) Background: If you wish to change the current background, you can select the **New** option from the dropdown, and specify the ID of the image you want to use as the new background, from the database, in the **Background ID** field.

   Note: If you wish to adjust the background **fade** settings, please head to **Canvas -> BackgroundContainer -> BackgroundManager.cs.**

4) Display Avatar: This option allows you to display or hide a character avatar. Character avatars are located in the **TextAreaContainer,** under the canvas, and are displayed next to the text area.



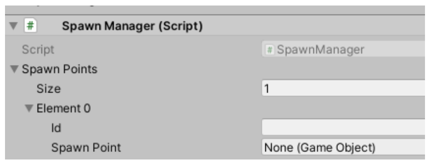
5) Display Name: This option can be used to display/hide a character's name above the text area.



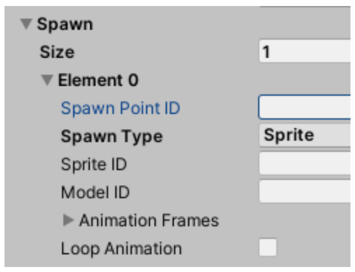
6) Hide Dialogue Window: If you wish to hide the entire dialogue window, you can do so using this option. In addition, you can specify how long the window will be hidden.

7) Request Name: This option allows you to change the default player name, or the name of any character (which can be configured in the database by default), in-game. If you choose to change a character's name, you will also need to specify the ID of that character in the database. Both options will result in a pop-up, prompting the player for input.

8) Spawn: In Choice Engine, you can spawn any prefabs/sprites on the screen. Before using this option, head to **Canvas -> SpawnManager.cs** and configure any number of spawn points on the screen, via the **spawn points** list. A spawn point can be any game object on the scene, since only the location of said object will be taken into account when spawning prefabs/sprites, and each spawn point must have a unique ID. By default, spawn points should be located under the **Spawn Container** object, under canvas.



```
▼ # Spawn Manager (Script)
Script              SpawnManager
▼ Spawn Points
  Size              1
  ▼ Element 0
    Id
    Spawn Point     None (Game Object)
```



```
▼ Spawn
  Size              1
  ▼ Element 0
    Spawn Point ID
    Spawn Type      Sprite
    Sprite ID
    Model ID
    ► Animation Frames
    Loop Animation  ☐
```

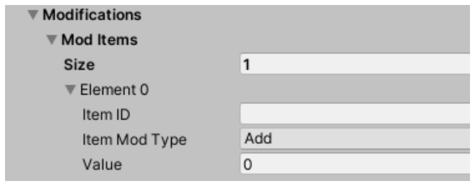
Once the spawn points have been set up, you can proceed to spawn any prefabs or sprites. To spawn a sprite or model, first select the appropriate **spawn type.** Then specify the ID of the sprite/model (from the database), and the ID of one of the spawn points previously created. If you choose to spawn an animated sprite, you will also need to specify the animation type and the separate animation frame settings.

9) Despawn: A list of spawn points (spawn point IDs) to clear.

10) Modifications: This subsection allows you to modify most database entries. For instance, if you wish to give the player an item, simply open the **Mod Items** list, select **Add** as the **Item Mod Type,** choose an item ID from the database, and set the quantity to be added.

In a similar manner you can:

a) Add and remove resources.
b) Assign, complete and fail quests.
c) Change character affections.
d) Change character stats.
e) Remove inventory items.



```
▼ Modifications
  ▼ Mod Items
    Size            1
    ▼ Element 0
      Item ID
      Item Mod Type Add
      Value         0
```

11) Load Level: You may choose to load another scene once the node has been entered. You will need to specify the index of the scene to be loaded (in the **Build Settings**), and you can optionally specify a new starting node that will be loaded once the scene is loaded. The latter option can be useful if

you wish to split your project into several scenes, and load different starting nodes based on previous events.

12) Display Message: This option allows you to display any notifications to the player.

13) Functions To Trigger: Custom functions can be triggered using this list. To trigger a function, it must exist within the UserFunctions.cs script. Once the function has been created, you will be able to specify its name and trigger the delay via this list.

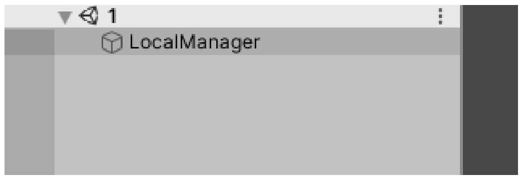
About Choice Engine

In the section, we will take a more general look at how Choice Engine is structured.

Choice Engine is designed to make the creation of visual novels and text-based games as easy as possible within unity. At the same time, the emphasis is also given on scalability and modifiability.

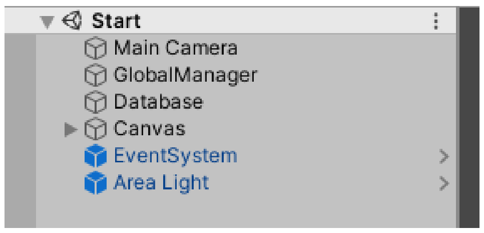
When making a game with this asset, you will generally have at least two scenes. The **starting** scene, and the primary **game** scene. The starting scene is one where all of the **global** objects are being initiated. Objects that are initiated on the starting scene will object be created once and keep existing until the game has ended. This is why you **should always** start your game from the starting scene.

Game scenes are designed to generally be empty. As you will see later on, you only need one object on any given game scene, the **level manager** object. The latter is essentially where all of your nodes reside and from where they can be configured. In other words, all levels in the asset apart from the starting level are designed to be empty.



1
LocalManager

The starting scene is comprised of three primary objects. The **database**, the **global manager**, and the **canvas**. Each one of these objects contains the scripts associated with the objects. For example, the database object contains **inventory, resource,** and **database** scripts. Each one of the latter is used to store data which is then used by the node manager and other systems. In the same manner, the canvas contains all the scripts associated with **UI.**

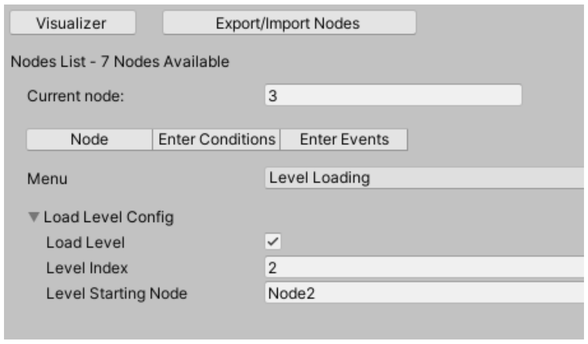


Start
Main Camera
GlobalManager
Database
Canvas
EventSystem
Area Light

The place you are going to be spending the most time is the **Visualizer** window. The latter can be accessed on any scene that contains a **level manager** object and allows you to easily edit the **nodes** list. It is highly recommended that you utilize this window over editing the nodes list directly, as the window has been specifically optimized to be clean, and render GUI smoothly, unlike the default list inspector. For instance, if you were to create a list of around a thousand nodes, the default list renderer will most likely cause severe lag, due to the way it is implemented. In fact, this has been a severe limitation of the predecessor of this asset, since users with weaker machines were forced to create multiple scenes when dealing with a large number of nodes.

The visualizer window can be accessed by clicking on **Window -> Visualizer.**

You can easily transition between levels without any restrictions. For example, if you wish to have more than one scene, you can create any number of scenes, and transition between them using the **level load** feature on any node. Furthermore, you can specify the starting node of the level to be loaded, making transitioning between nodes even more flexible.



Nodes List - 7 Nodes Available

Current node: 3

Menu: Level Loading

Load Level Config
- Load Level: ✓
- Level Index: 2
- Level Starting Node: Node2

Creating a game from scratch

Here is a quick guide to creating a new Choice Engine 2 project from scratch.

1) Create a unity 2D project.

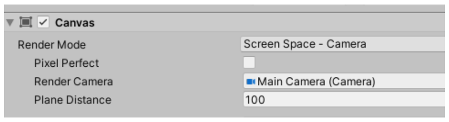
2) Import Choice Engine from the asset store.

3) Create a new scene. This will be your starting scene. All important objects will be initiated in this scene.

4) Open the Build Settings and add the new scene to the catalog (Check out **tips** if you don’t know how).

5) Navigate to the prefabs folder and drag the **Canvas, Main Camera, Global Manager, Event System, Area Light** and **Database** objects onto the scene. Once the objects are on the scene, right-click on each object and select “Unpack prefab completely”, this will disconnect the objects on the scenes from their prefab equivalents and will allow you to freely manipulate the objects.

6) Make sure to select the “Screen Space Camera” mode on the canvas, and drag the main camera, created by default, into the appropriate field.



Canvas

Render Mode	Screen Space - Camera
Pixel Perfect	
Render Camera	Main Camera (Camera)
Plane Distance	100

7) Click on the **Canvas** object and open the **Menu Functions** script. Inside the script, define the index of the scene to be loaded when the user creates a new game. The index of a scene is the number to the right of the scene name in the build settings.

8) Open the **Canvas** component and enable the **Main Menu object.**

9) From here one you can configure each of the imported objects in accordance with the documentation. More specifically, you should configure each script in the **Important Scripts** section, and customize the UI in accordance with your project.

10) Once the starting scene is configured, you can proceed to configure individual game scenes. To create a new game scene, simply create a new empty scene, delete everything on it, and add the **Local Manager** prefab to the scene.

    Once the prefab was been added, right-click on the prefab and select **Unpack Prefab Completely**. You can now proceed to configure the nodes on your scene. Nodes can be configured directly via the **Nodes** list on the **Local Manager** object, or by using the **Visualizer** window.

11) You are done. If you wish to skip this process entirely, you can choose to use the demo scenes as a template.

Conclusion/Misc

Thank you for choosing Choice Engine 2. If you have any suggestions or advice on improving the asset, please let me know. If you like the asset as is, consider leaving a review on the asset store, it only takes a minute and helps me out immensely.