FMV Maker Documentation

by Kyle Grenier

Table of Contents

Preface

Before Reading

If a paragraph in this documentation is labeled **OBSOLETE**, it may be ignored. I decided to keep it in the documentation in case it ever may be needed.

Definitions

Scenario: A video clip that leads to another, either through a selection the player makes or through the video clip ending.

Channel: A ScriptableObject that wraps a UnityAction. Used to reduce coupling between scripts, but more generally it is used to receive subscribers and broadcast an event to those subscribers, invoking functions.

Callback: Code that is executed in response to an event starting or finishing.

Concept Overview

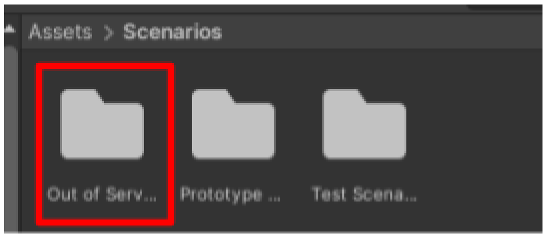
The FMV Maker I created is my attempt at streamlining the FMV creation process. It is based around Scriptable Objects, so creating new scenarios is as easy as creating an object in the Project panel and dropping in the required fields. In addition, the custom editor enables us to easily add in buttons with a popup time automatically filled in.

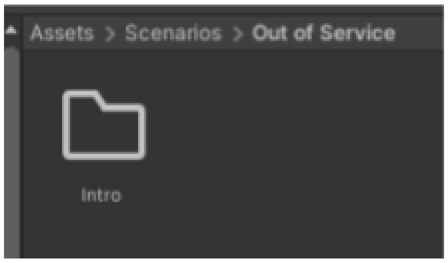
Project Creation

Creating the Project

To create a new scenario, navigate to the Assets/Projects folder. Here, create a name for the project. In our case, it would be "Out of Service". This folder will serve as the root of our project and will hold all of the videos, Scriptable Objects, scenario folders, etc. From here on, the **root** of your project will refer to this folder, *NOT* the root of the Unity project.

Every project needs an introduction, a scenario to start the FMV. Inside of the newly created project folder, right click and create a new folder title "Intro". This folder will house all of the content that the introduction scenario contains.



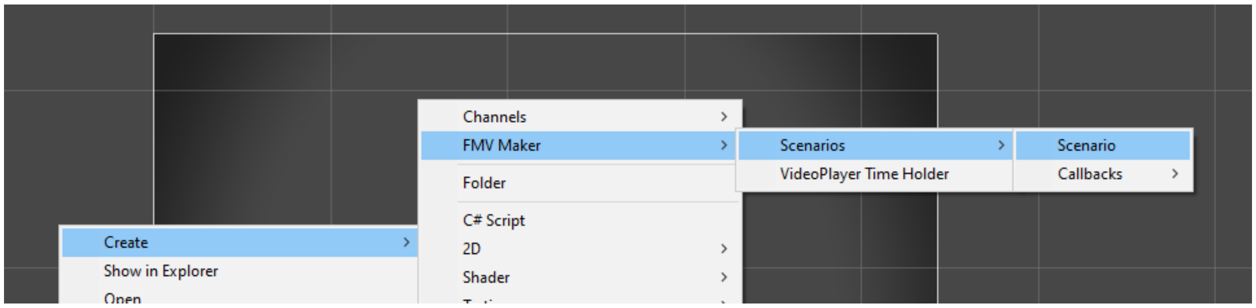


Example project setup

Scenarios

Adding Scenarios

With the project created, it is time to add scenarios. To start, we'll need a scenario that serves as the introduction, the starting point in our FMV. Navigate inside of your project's Intro folder, right click, hover over Create -> FMV Maker -> Scenarios, and left click on "Scenario". This will create a new Scenario Scriptable Object. This Scriptable Object holds all of this scenario's data. Name the newly created Scriptable Object to "Intro".



Creating a new Scenario

Customizing Scenarios

Our created scenarios need to have data assigned to them. To customize a scenario, left click on the scenario asset (Scriptable Object) and navigate to the Inspector. All of the data that needs to be assigned is shown. Let's start by adding a video clip. This is the video clip that will be shown to players and is the particular video clip associated with this scenario.

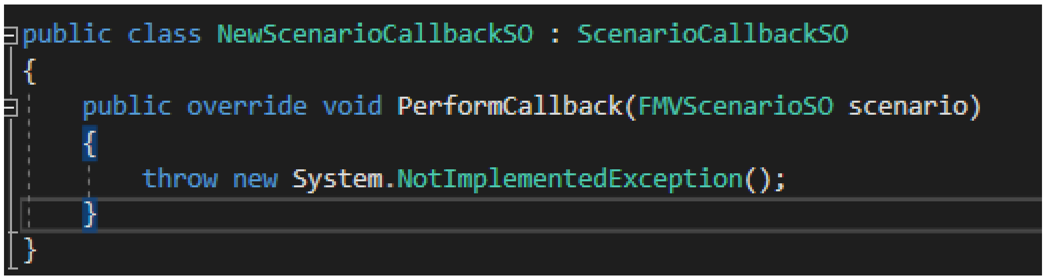
Inside of the current scenario's folder located at Assets/Projects/<*ProjectName*>/<*ScenarioName*>, import the associated video clip. Once imported, drag the video clip into the Scenario asset's Video Clip field. The video clip is now associated with the scenario.

The "Choices To Be Made" boolean should be checked if the Scenario will display choices to the player, typically near the end of the Scenario's video clip. Once selected, a list of Popups will appear. Here is where you add any popups you would like to appear during the FMV. Adding popups will be covered in a later section.

If the boolean is unchecked, a field requesting a scenario to lead into will appear. If there is a scenario to be played after the current scenario, drag and drop it into this field. If not, you may leave it blank. **Note** that if it is left blank the video will abruptly pause at the end of the current scenario's clip. Therefore, it's recommended that you include some sort of "finale" Scenario to serve as the end of the FMV.

Scenario Callbacks

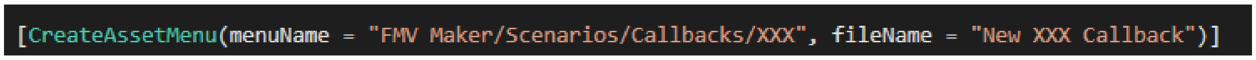
A Scenario Callback is a special ScriptableObject that contains custom code that can be executed at the start and/or the end of a Scenario. To create your own Scenario Callback, navigate to Assets/Scripts/ScriptableObjects/Scenario Callbacks. In this directory, create a new C# script titled <*your_Scenario_callback's_name*>SO. The SO is short for ScriptableObject. **Note** that this is simply a naming convention and is not required to make the Scenario Callback function. Make the class derive from ScenarioCallbackSO instead of MonoBehaviour. Right click the red squiggly line, left click on "Quick Actions and Refactorings…", then left click on "Implement Abstract Class." This shortcut will import all of the required methods that must be implemented. Now remove the Start() and Update() methods. You should be only left with the PerformCallback() method.



```
public class NewScenarioCallbackSO : ScenarioCallbackSO
{
    public override void PerformCallback(FMVScenarioSO scenario)
    {
        throw new System.NotImplementedException();
    }
}
```

A new Scenario Callback ready for custom implementation.

On the line above the "public class" statement, write the following attribute:



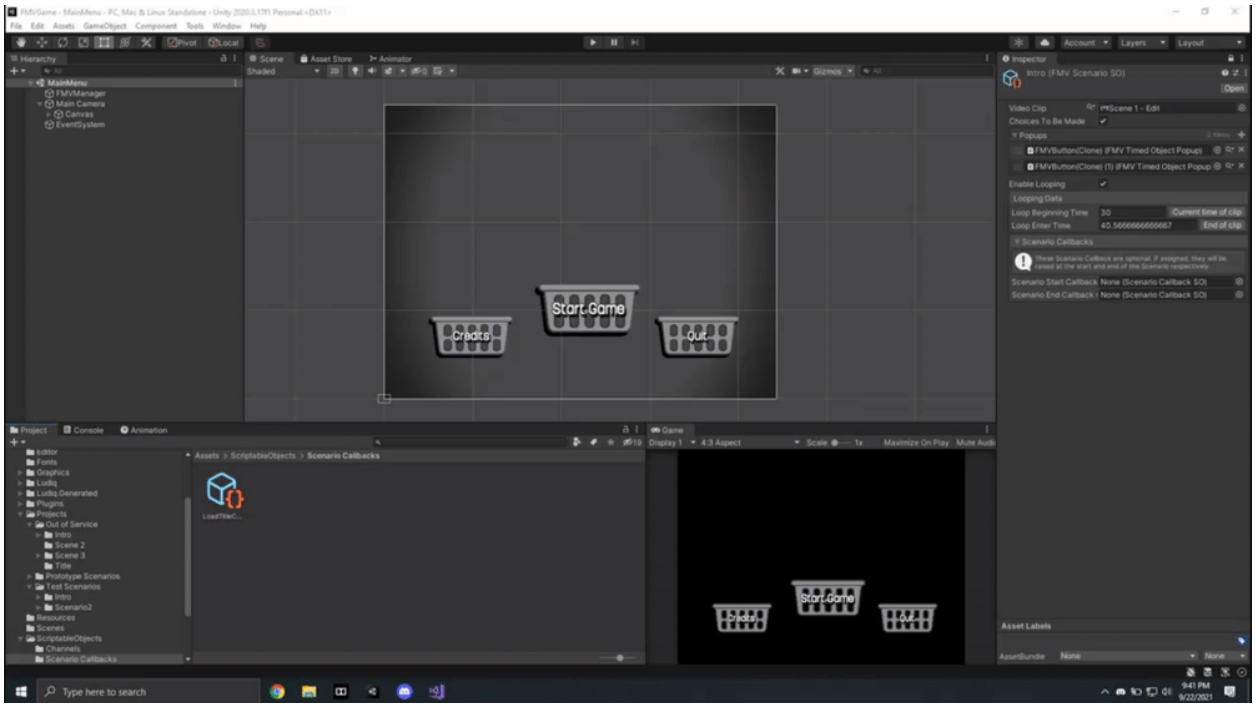
```
[CreateAssetMenu(menuName = "FMV Maker/Scenarios/Callbacks/XXX", fileName = "New XXX Callback")]
```

Replace "XXX" with the name of the Scenario Callback. For example, a Scenario Callback with a script name of "LoadMainMenuSO.cs" would have a menuName of "FMV Maker/Scenarios/Callbacks/Load Main Menu" and a fileName of "New Load Main Menu Callback".

Now that your Scenario Callback is set up, you can write your own custom functionality in the script. **Note** that the Scenario Callback's execution will begin in the PerformCallback() method. This method's parameter is the FMVScenarioSO (Scenario) which invoked the Scenario Callback.

Once you finish writing your Scenario Callback, navigate to Assets/ScriptableObjects/Scenario Callbacks, then right click, hover over Create -> FMV Maker -> Scenarios -> Callbacks, and left click on the name of your Scenario Callback. If you do not see your Scenario Callback, make sure you wrote the CreateAssetMenu attribute correctly. For more information about this attribute, visit this document.

To assign a Scenario Callback to a Scenario, navigate to your project's root folder then into a Scenario's folder. Left click on the Scenario ScriptableObject, and expand the “Scenario Callbacks” foldout group. From there, you can drag in any Scenario Callback asset you created into the fields.

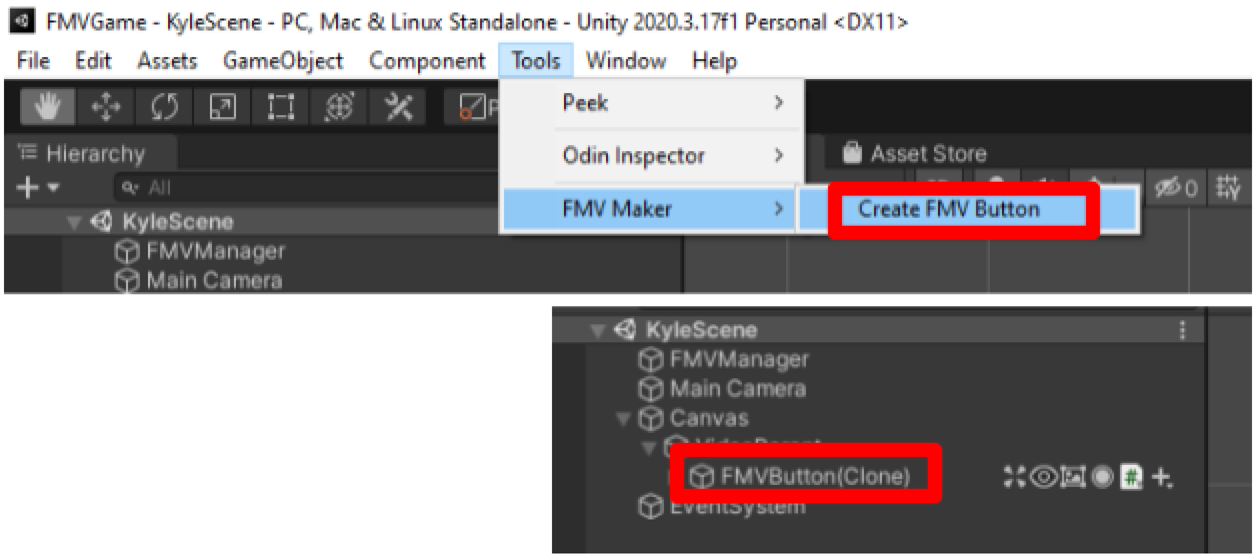

Intro (FMV Scenario SO)
Video Clip: Scene 1 - Edit
Choices To Be Made: ✓
Popups
FMVButton(Clone) (FMV Timed Object Popup)
FMVButton(Clone) (1) (FMV Timed Object Popup
Enable Looping: ✓
Looping Data
Loop Beginning Time: 30
Loop Enter Time: 40.5666666666667
Scenario Callbacks
These Scenario Callback are optional. If assigned, they will be raised at the start and end of the Scenario respectively.
Scenario Start Callback: None (Scenario Callback SO)
Scenario End Callback: None (Scenario Callback SO)

Dragging in a Scenario Callback titled LoadTitleCallback into a Scenario's start and end callback fields.

Popups

Creating Scenario Popups

A popup is a GameObject that *pops up* at some point during a scenario. Although it can be any GameObject that the Unity UI can render, more often than not it will be a button that prompts the user to make a choice. For this reason, a customer editor tool has been created to automatically create a button with all of the components required to use the FMV Maker. To access this tool, navigate to the Unity toolbar at the top of the screen, select Tools -> FMV Maker, and left click “Create FMV Button”. Upon clicking, a GameObject will be instantiated in the Hierarchy named “FMV Button(Clone)”.



Using the Create FMV Button tool to instantiate a new FMV Button into the Hierarchy

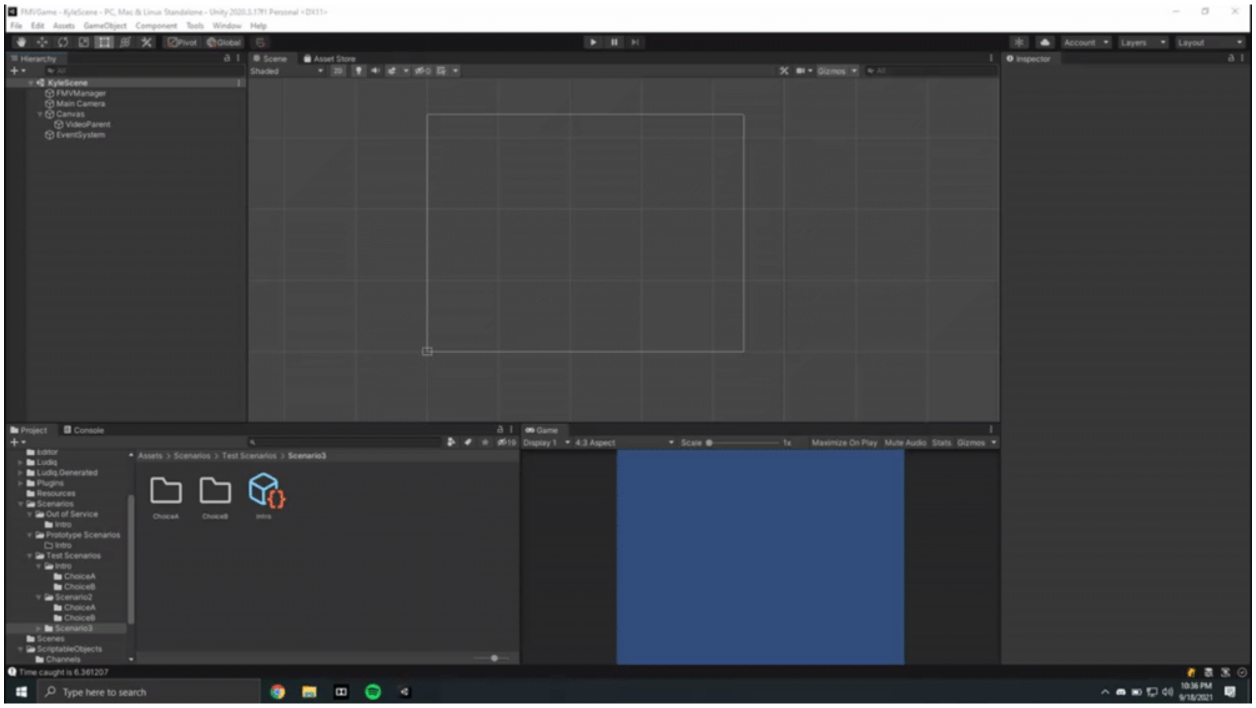
Upon left clicking the instantiated button, navigate to the Inspector and scroll to the bottom of the page. You’ll see two unique components: “FMV Timed Object Popup” and “FMV Scenario Progressor”. The FMV Timed Object Popup script is responsible for instantiating the button at the requested time in the Scenario. The FMV Scenario Progressor is responsible for progressing the current Scenario to the Scenario provided upon clicking the button.

The FMV Scenario Progressor’s “Next Scenario” field is self explanatory. You’ll follow the steps above to creating a new Scenario and drag in the newly created Scenario into this field.

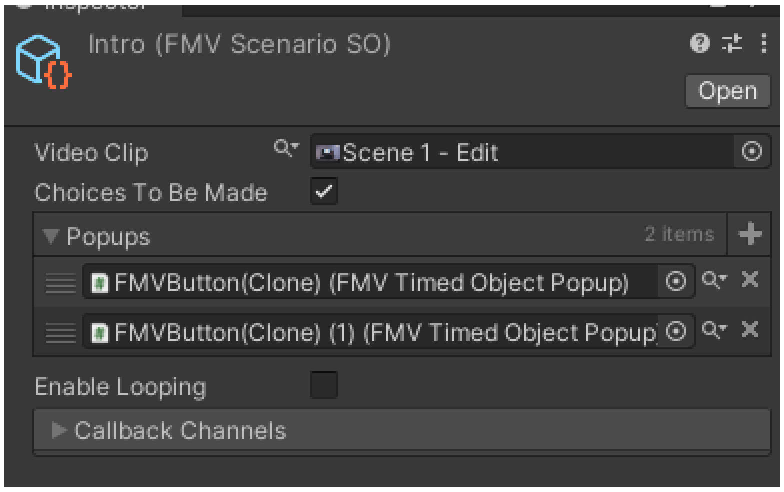
The FMV Timed Object Popup’s “Popup Time” is also self explanatory, but there are a few automated ways to make use of it. Chances are if you’ve switched to Play

Mode the Popup Time is not zero but a seemingly random float value. This float value is the elapsed time of the last scenario's video clip when the game was stopped. If an FMV Button were to be created while the game were running or paused, the Popup Time would be the time elapsed in the currently playing video clip. That being said, if used correctly, the Popup Time should not need to be adjusted.

Once a popup is created and modified to your liking, *including* position, scale, coloring, text (can be modified on the child Text object), etc, you are required to make a prefab out of it. If multiple popups will be instantiated in the Scenario, please put each prefab into separate folders in your project's Scenario directory.


Creating an FMV Button and creating a prefab from it inside of the correct folder.

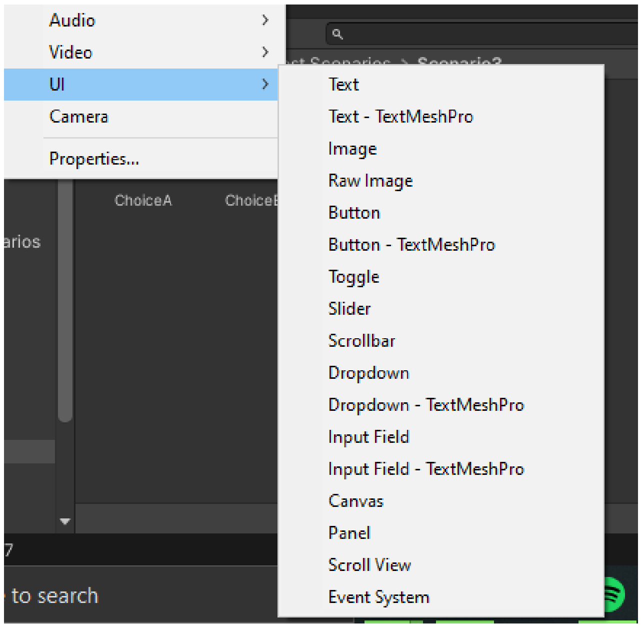
These prefabs are what will be instantiated by the Scenario and are what must be dragged into the Scenario's Popups list. Therefore, please click on the current Scenario and drag each prefab into the Popups list. Now, when this Scenario is played, your buttons will be instantiated.


Intro (FMV Scenario SO)
Open
Video Clip: Scene 1 - Edit
Choices To Be Made: ✓
Popups: 2 items
FMVButton(Clone) (FMV Timed Object Popup)
FMVButton(Clone) (1) (FMV Timed Object Popup
Enable Looping
Callback Channels

An "Intro" Scenario featuring two button popups.

Other Popups

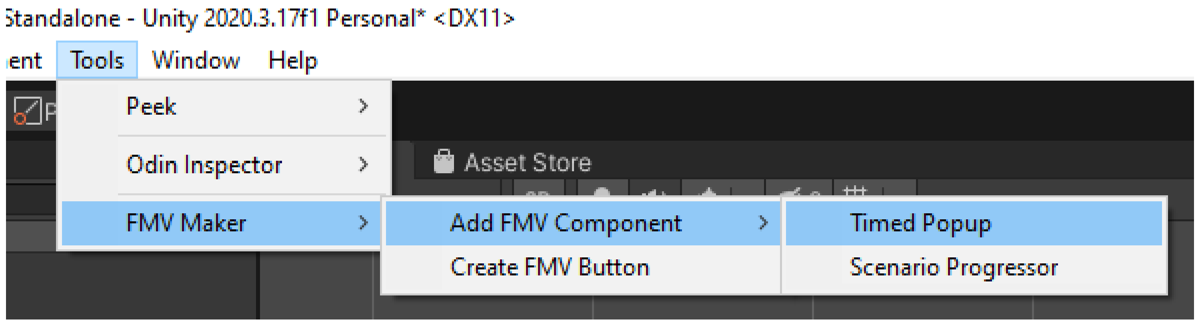
If for any reason you require popups other than the premade buttons, you must create a GameObject capable of being rendered by the Unity UI system (a proper UI element). To create your own popup, start by creating a base UI GameObject as a child of the "VideoParent" GameObject - the VideoParent is the child of the Canvas. By base GameObject I am referring to any UI GameObject: an Image, Panel, Text, etc.



Examples of base UI GameObjects.

From there, modify the object to your liking. Before saving it as prefab, you'll need to ensure that *at least* the FMVTimedObjectPopup component is added to the

GameObject. You can either add the component yourself, or with the GameObject selected in the Hierarchy, navigate to Tools -> FMV Maker -> Add FMV Component, and left click on “Timed Popup”. Similarly, if your popup is able to be interacted with and progress to another Scenario, left click on “Scenario Progressor” in the same menu. **Note** that creating custom Scenario Progressors can get complicated for objects other than buttons, which are inherently able to be interacted with and have the simple OnClick() UnityAction which can be subscribed to. If you need help creating custom Scenario Progressors, don’t hesitate to contact me :).

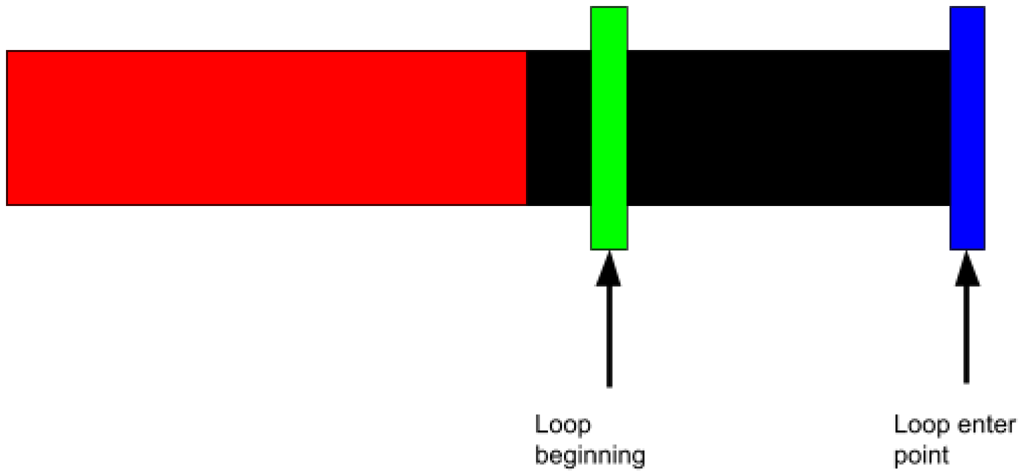


Adding the FMVTimedObjectPopup component to the currently selected GameObject in the Hierarchy.

Looping

Looping

Looping enables a Scenario to contain a portion of its video clip to loop. There are two variables to consider when it comes to looping: the loop beginning and the loop enter point. The loop enter point is the time (in seconds) the video will jump back to the beginning of the loop. That being said, the loop beginning point is the time (in seconds) that the loop begins at. For further clarification, take a look at the graphic below of a video scrubber below. **Note** that this scrubber isn't a feature of this tool; it is only included in this documentation for clarification.

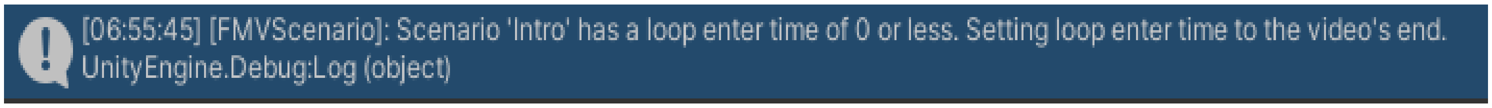


When the scrubber (point in time the video is playing at) reaches the loop enter point, the video will automatically jump back to the loop beginning and play until the loop enter point is reached again. From there, the process repeats.

To enable looping on a Scenario, navigate to your project's root folder, then to any Scenario you wish to enable looping for. With the Scenario asset selected, navigate to the Inspector. A field is shown called "Enable Looping". Left click on the field to enable looping; a checkmark should appear to indicate it is enabled. A new set of fields will appear under a group titled "Looping Data". Here, you can adjust the two variables mentioned above. If you know the loop beginning and loop enter times, you may enter them here. If you do not, you can left click the "Current time of clip" button to the right of the loop beginning time field to set the loop beginning time to the current time of the playing or last-played video clip. Additionally, you can enter Play Mode and while the Scenario is playing press this button to adjust the loop beginning time on-the-fly. In a similar manner, you can left click the button "End of clip" to the right of the loop enter time to set it to the end of the Scenario's video clip.

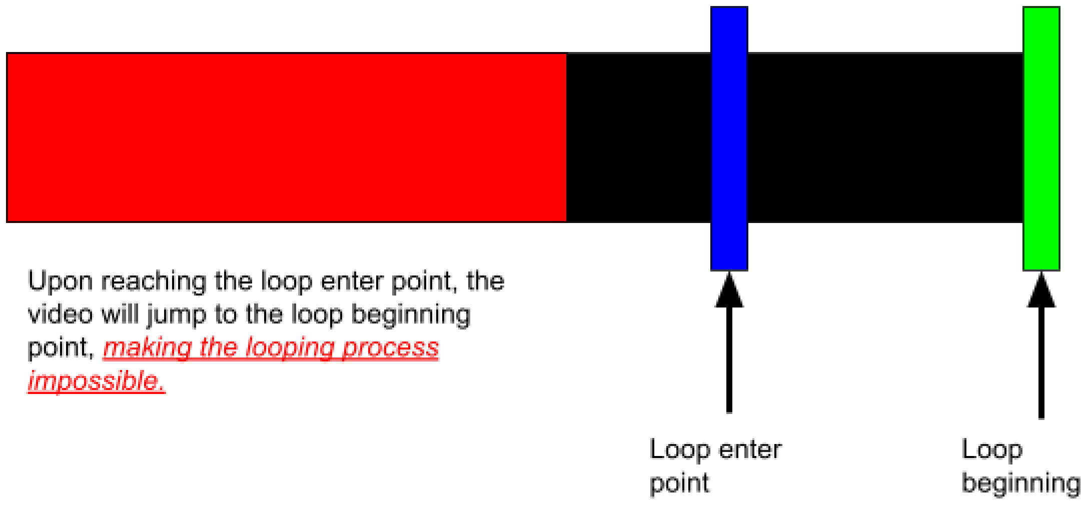
<u>There are a few important things to consider when adjusting these variables:</u>

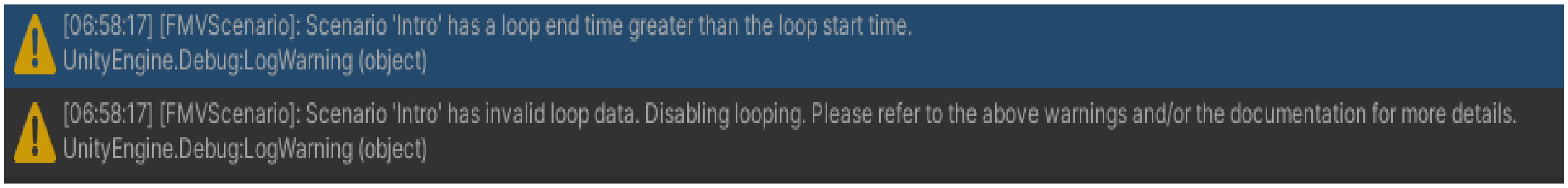
- If the loop enter time is left at 0 and “Enable Looping” is checked, the system will automatically set the time to the end of the Scenario’s video clip next time the Scenario is played. A message will be displayed in the Console to confirm this change. **Note** that this change is permanent and will consist between playthroughs unless modified afterwards.



```
[06:55:45] [FMVScenario]: Scenario 'Intro' has a loop enter time of 0 or less. Setting loop enter time to the video's end.
UnityEngine.Debug:Log (object)
```

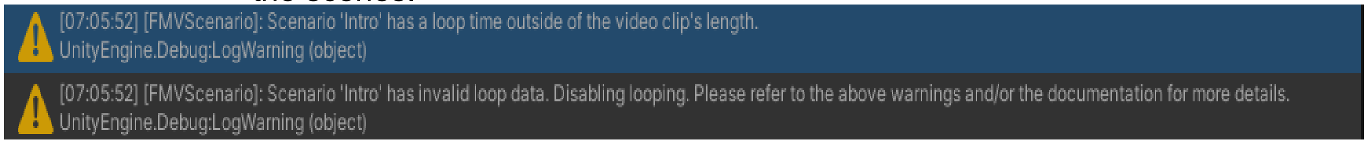
- The loop enter time cannot be less than the loop beginning time. This would make the looping process impossible. If you do set the loop enter time to be less than the loop beginning time, a warning will be displayed in the console and looping will be disabled for that playthrough of the Scenario. If left untouched, the warning will persist and looping will continue to be disabled.


Upon reaching the loop enter point, the video will jump to the loop beginning point, *making the looping process impossible.*
Loop enter point
Loop beginning



```
[06:58:17] [FMVScenario]: Scenario 'Intro' has a loop end time greater than the loop start time.
UnityEngine.Debug:LogWarning (object)
[06:58:17] [FMVScenario]: Scenario 'Intro' has invalid loop data. Disabling looping. Please refer to the above warnings and/or the documentation for more details.
UnityEngine.Debug:LogWarning (object)
```

- Neither the loop beginning time nor the loop enter time can be outside of the length of the Scenario’s video clip. This means that neither can be longer than the video clip plays for, and cannot be less than zero. If this occurs, a warning will be displayed in the console. The system prevents the loop beginning time and loop enter time from being set to a number less than zero, but this is not guaranteed to occur if the Scenario is modified via code. For that reason and many more, it is highly unadvised to modify this system’s scripts without knowledge of how it works behind the scenes.



```
[07:05:52] [FMVScenario]: Scenario 'Intro' has a loop time outside of the video clip's length.
UnityEngine.Debug:LogWarning (object)
[07:05:52] [FMVScenario]: Scenario 'Intro' has invalid loop data. Disabling looping. Please refer to the above warnings and/or the documentation for more details.
UnityEngine.Debug:LogWarning (object)
```

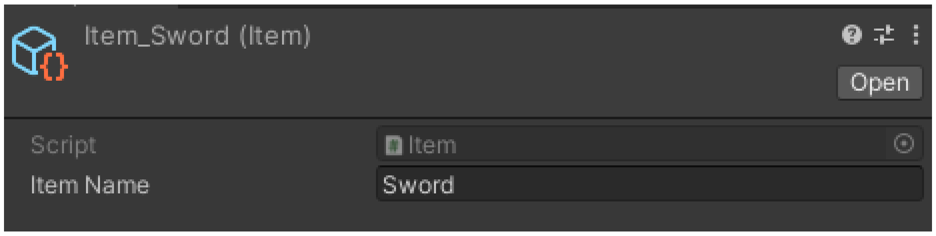
Inventory System

Inventory System

The inventory holds items that the player collects throughout their play session. Players can use the items they collect to open up new choices to make in a given scenario. For example, a player can pick up either a knife or a sword. Later, they are prompted to attack an enemy. They can use their fists or the item they picked up earlier in the game. The story will play out differently depending on which item the player uses, but they can't use an item if they don't have it in their inventory first.

Creating Items

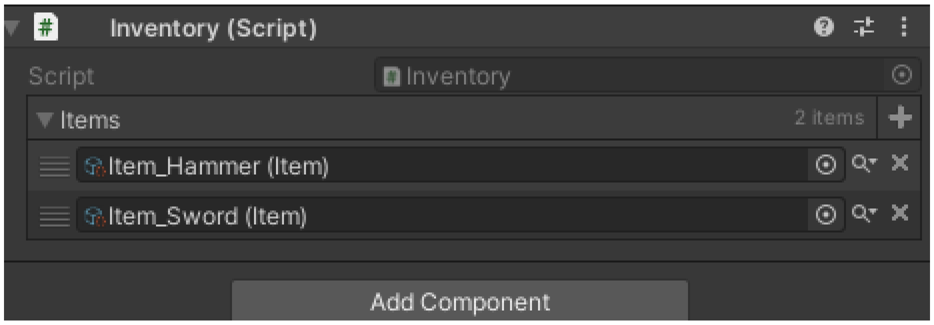
To create inventory items, navigate to the Assets/ScriptableObjects/Inventory Items directory. This folder contains all of the items the player can find and use throughout the game. Right click on an empty space inside of the folder, hover over Create -> FMV Maker, and left click on "Inventory Item". A new ScriptableObject asset will be created. Please name it "Item_*<item_name>*", where *<item_name>* is the name of the item. Once created, left click on the asset and navigate to the Inspector. There you can input a name for the item in the "Item Name" field.


Item_Sword (Item)

Script	Item
Item Name	Sword

An item with the name "Sword"

The Inventory is a component attached to the FMVManager object in the scene. **Note** that it is not present on the title screen FMVManager. Behind the scenes, the inventory is simply a list of items that populates as the player collects them. To add a new item to the inventory, left click on the FMVManager object in the hierarchy and click the + next to the list titled "Items" on the Inventory component. Double click on the item you wish to add to the inventory. To remove an item, click the X to the right of the item in the list.



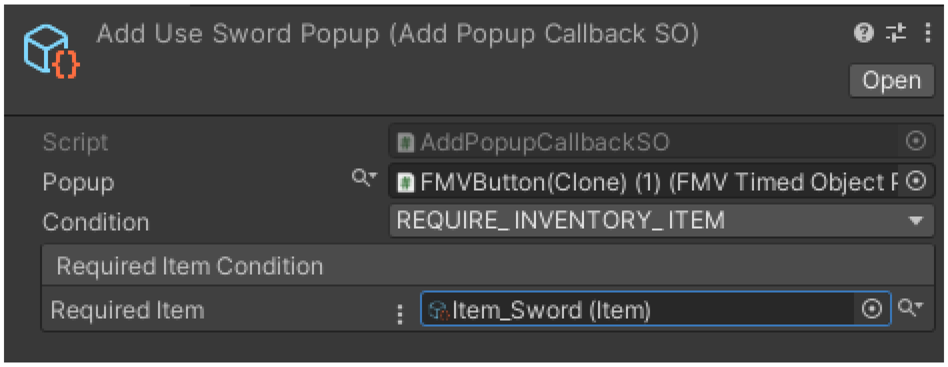
An inventory with two items: a Hammer and a Sword.

Adding Items via Scenario Callbacks

Items can also be added to the inventory via Scenario callbacks. To create a new callback which adds an item to the inventory, navigate to the directory Assets/ScriptableObjects/Scenario Callbacks, then right click and hover over FMV Maker -> Scenarios -> Callbacks , then left click on "Add Inventory Item". Name the asset to something along the lines of "Pickup<*item_name*>Callback". Left click on the asset and navigate to the Inspector. Drag in the item you would like to add to the inventory. From here, you can apply the Scenario callback to any scenario. For more information about using Scenario callbacks, see above.

The "Add Popup" Scenario Callback

Another important Scenario callback related to the inventory system is the Add Popup Callback. This callback adds a Scenario popup given a condition that must be met to the Scenario it's attached to. To create a new Add Popup Callback, navigate to Assets/ScriptableObjects/Scenario Callbacks, right click and hover over FMV Maker -> Scenarios -> Callbacks, and left click on "Add Popup". Rename the asset to something along the line of "Add<*popup_name*>Callback". Left click on the asset and navigate to the Inspector. Drag the popup to be added into the first field, "Popup". Then select a condition that must be met for this popup to be added to the Scenario. By default, the condition is NONE which means that the popup will always be added to the Scenario. The other condition is REQUIRE_INVENTORY_ITEM which requires the player to possess a certain item in their inventory in order for the popup to be added. This allows us to display buttons such as "Use Item" if and only if the player has that item in their inventory. Once the condition is set to REQUIRE_INVENTORY_ITEM, drag the required item into the field "Required Item". Once set, the callback is ready for use and can be incorporated into a Scenario of your choice. **Note** that this callback is best used as a Scenario Start Callback.


Add Use Sword Popup (Add Popup Callback SO)

Open

Script AddPopupCallbackSO

Popup FMVButton(Clone) (1) (FMV Timed Object F

Condition REQUIRE_INVENTORY_ITEM

Required Item Condition

Required Item Item_Sword (Item)

An Add Popup Callback which adds a "Use Sword" popup to the Scenario. This popup will only be added if the player has the Sword in their inventory.

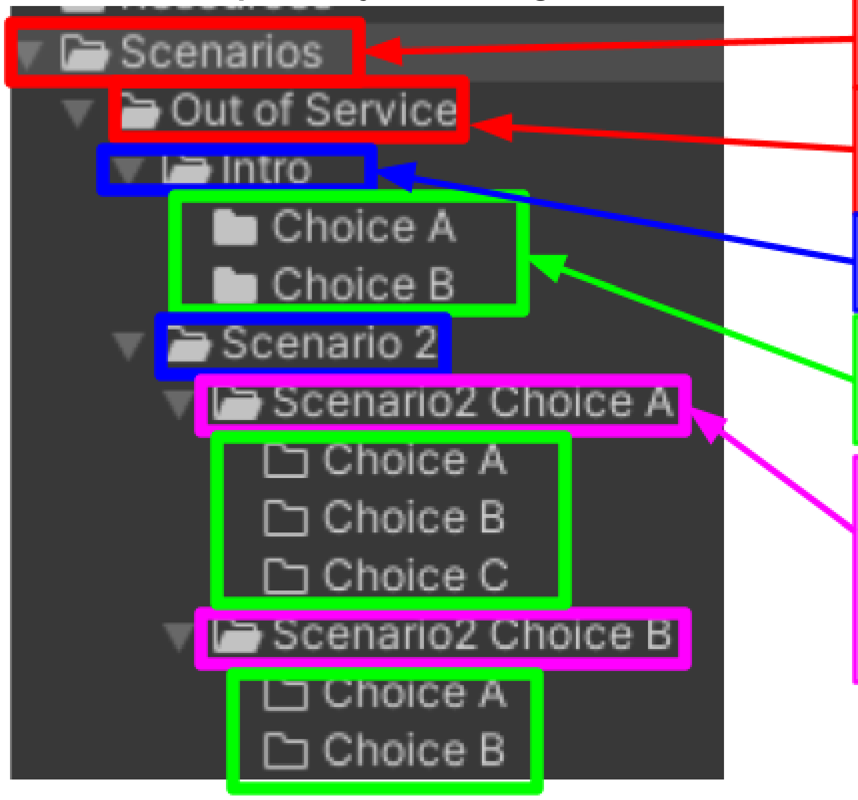
Organization

The Importance of Organization

Behind the scenes, we're using a sort of tree structure, where each node of the tree represents a Scenario, and one Scenario leads into another, with the final node of the tree representing an ending to the FMV. This system I created definitely has its flaws, most notably being unable to view the project as a tree structure, and linking Scenario ScriptableObjects into other Scenario ScriptableObjects can become complex. Therefore, organization is extremely important for a project such as this. Although this has been mentioned throughout the documentation, I've created an example of how I believe a project such as this should be organized that can be seen below. Ideally, we'll create a flowchart that has tags or keywords that we can use instead of "Choice A, Choice B" and/or "Scenario 2, Scenario 3," etc. We'll discuss this in more detail later on.

*****SEE BELOW*****

Proposed system of organization



Folder to store projects. We'll only be using the "Out of Service" folder.

Folder to house our project. Serves as the "root" of our project. In our case, this is the only project folder we'll be utilizing.

A folder for each Scenario is housed inside of the project folder. Here is a folder to house an "Intro" scenario.

Each choice the player can make in the Scenario should have its own folder. Here, the Intro Scenario has two choices the player can make.

If a Scenario has multiple versions, each version will have its own folder inside of the parent Scenario directory. In this example, Scenario 2 has two versions. In the previous Scenario, if the player selects Choice A, Scenario2 ChoiceA will be played. If the player selects Choice B in the previous Scenario, Scenario2 Choice B will be played.

***Following this setup, ideally "Intro" will lead into "Scenario 2", "Scenario 2" leads into "Scenario 3", etc. This may change depending on how we utilize the flowchart, or we use both the flowchart in conjunction with this naming system.