GameManager 2 is a Unity package that contains several functionalities to manage common features needed to make games, like Singleton management, multi-scene game states, sequential actions, finite state machines, among other utilities, all balanced between code and editor customization to make them easy to use.

What's new

Version 2.2

- Fixed PropertyDrawers and DecoratorDrawers. Now they are unified and contains a complete example for them.
- Close button for FSM and Director properties.
- UnitySingleton now inherits from MonoBase.
- Fixed GameManager methods to be accessible from UnityEvents.
- Added GMLoadMode to GameManager to manage scene loading when a scene is repeated between the current state and the next one.
- Added ForceChangeGameState to GameManager.
- Added OnStartedLoad event to GameManager.
- Removed LiorTal's ScriptableObjectFactory.

Version 2.1

- Added Scene loading sequencial and async options
- Added FastMono
- Added new MonoBase callbacks: OnDisableOrDestroyNotQuit, OnDisableNotQuit, OnDestroyNotQuit
- Improved FSM and Director editors

Version 2.0

First, GameManager2 fundamentally changes the inner workings compared to the previous version. GameManager used scripting order to take the lead instantiating the needed prefabs to ensure that the game could run from any scene.

GameManager 2 uses a different approach to the same goal. Now, we work with multiple scenes making good use of the new capacities of Unity 3D 5.3+, and we do this by defining game states with multiple scenes, where one scene has the Singleton objects needed for GameManager, and for many Plugins that depends on Singletons, to function.

In addition to the titular GameManager, we have two other modules:

-Director: is an action sequencer with UnityEvents, that has very little coding requirements.
-FSM: is a finite state machine with UnityEvents, ideal to use with non-sequential behaviors.

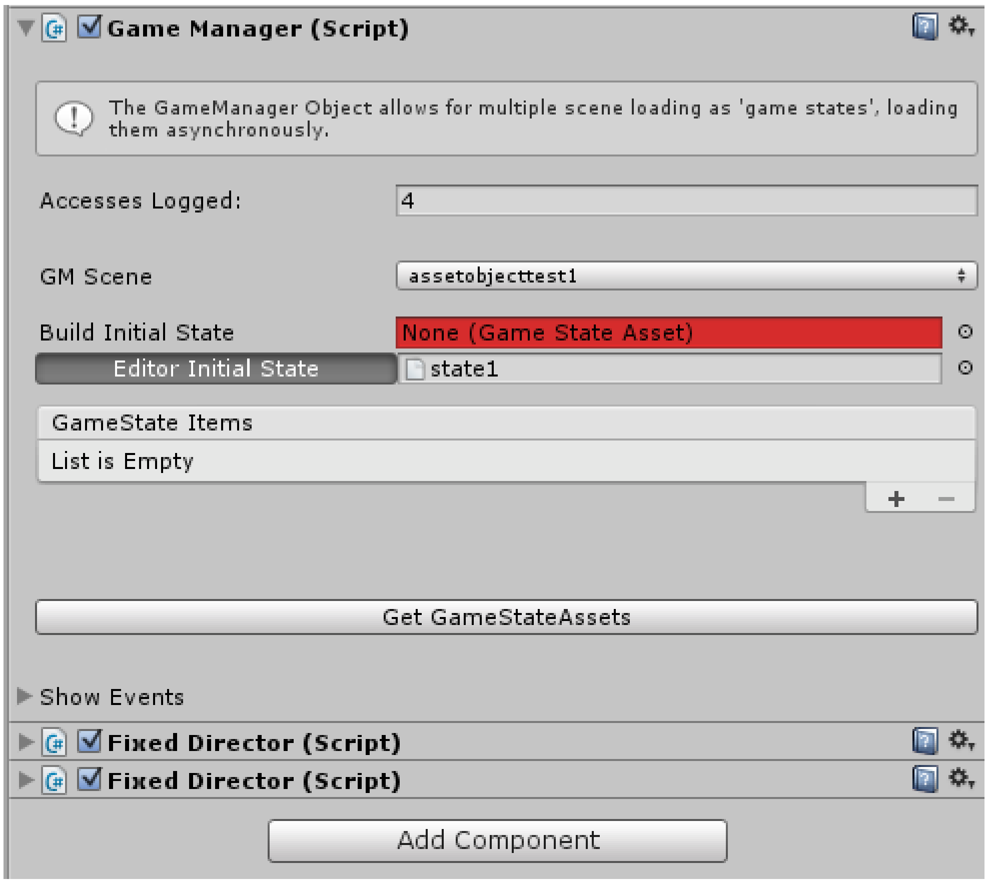
GameManager

With GameManager in the project, we need to designate an Entry point scene, to be used with persistent singleton objects, and anything that we need to be set up in initialization, regardless of what scenes we need to use when we enter play mode.

Then, we define the GameState ScriptableObjects to be used, creating them and assigning them scenes. The scenes have to be in EditorBuildSettings in order to work, as it's always been. We can automate this process by using a button that will be available in the GameState assets when some of the selected scenes aren't in the settings.

With the GameStates created and their scenes assigned, we create a GameObject in our Entry point scene, and we add a GameManager component. With this component added, we can setup the project.

We have to select the Entry point scene as "GM Scene", and then we click on the "Get GameStateAssets" button to get all the GameState assets of the project. We can manage the GameState assets manually if we prefer, using the reorderable list.



Game Manager (Script)

The GameManager Object allows for multiple scene loading as 'game states', loading them asynchronously.

Accesses Logged: 4

GM Scene: assetobjecttest1

Build Initial State: None (Game State Asset)

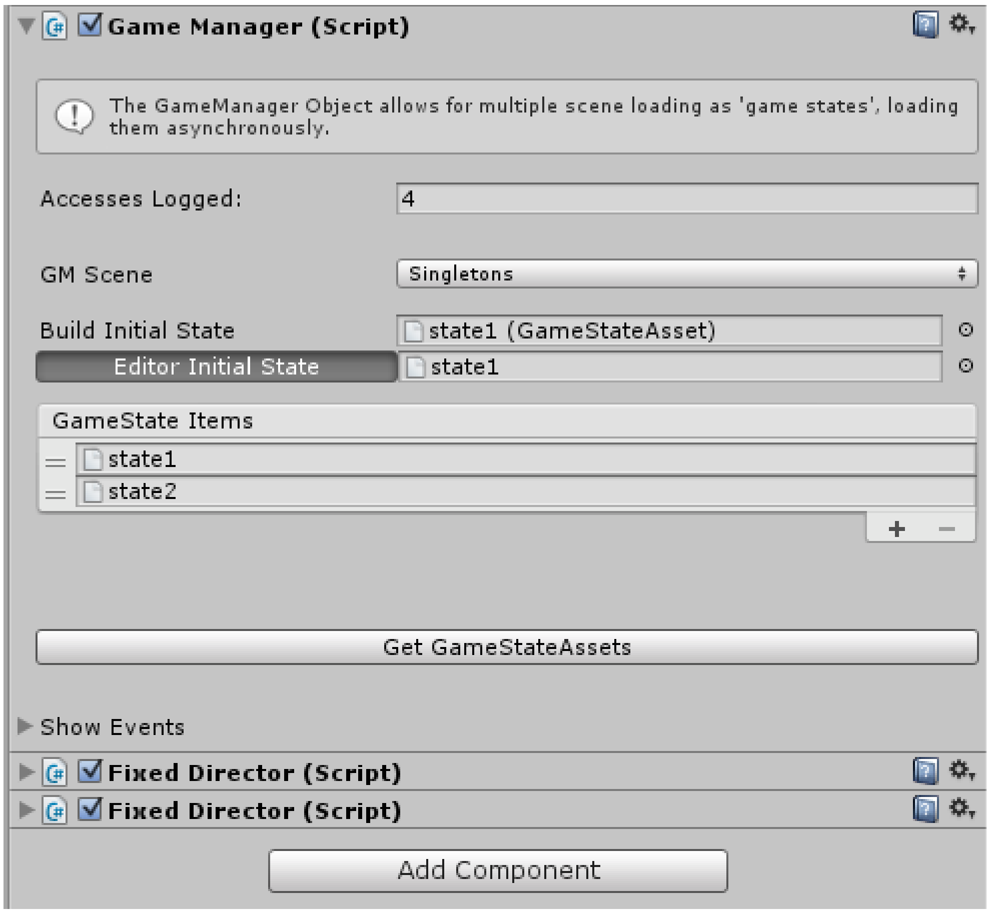
Editor Initial State: state1

GameState Items

List is Empty

Now, we have to select the Build Initial State to be able to build the project, and then we have two options: We can setup the GameManager to ignore open scenes and open a entire GameState when we enter play mode, or we can ignore that functionality and make the game manager start with the opened scenes from the editor.

In order to open scenes when entering play mode, we have to set the "Editor Initial State" toggle button on, and select a GameState for the GameManager to load when starts. To ignore that functionality, we have to set "Editor Initial State" off.



Game Manager (Script)

The GameManager Object allows for multiple scene loading as 'game states', loading them asynchronously.

Accesses Logged: 4

GM Scene: Singletons

Build Initial State: state1 (GameStateAsset)

Editor Initial State: state1

GameState Items
- state1
- state2

With these settings applied, all we have to do is add GameState assets for every state of the game, and populate them with the scenes that they will use.

Using the Events bundled with the system, we can also add advanced functionality, like automatically fade in and out, or loading the AssetBundle data between GameState loads.

Common Modules

Director

The Director module is a sequence of UnityEvents that can be used to wait for asynchronous processes, such as waiting a file to download, doing multi-thread work, using co-routines or simply wait for time to pass.

Used without dropping a single line of code, it can setup sequences of UnityEvent calls to make sure that the method calls are called in the specified order. Using a little scripting with it, it can unlock a lot of power to wait for multiple conditions at the same step, and to wait for asynchronous events.

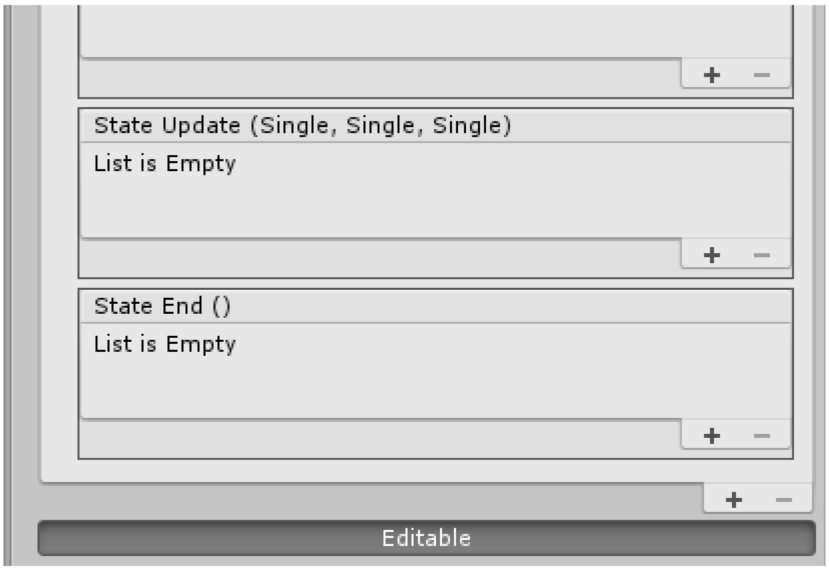
To use a director, simply add a Director component to a GameObject, and with the "Editable" button toggled on, define the required steps for the flow.

For a good example of what is achievable with it, refer to the included Example.

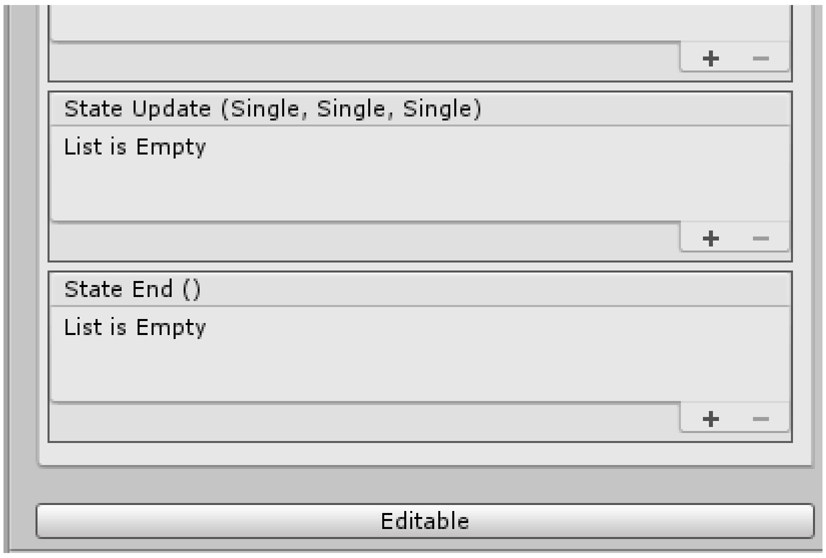
FSM

The FSM module is made with the premise of making a fixed state machine by code, and then making possible to decorate the states with UnityEvents from the editor, without risking the flow of the FSM.

To use it, you have to make an instance of FSM on a GameObject, and then, with the "Editable" option toggled on, create the states in the editor.


State Update (Single, Single, Single)
List is Empty
State End ()
List is Empty

Once the states are created, you can use any script with a reference to that FSM to define the methods that will be accessed from the FSM. You can then assign them in editor or by script with the UnityEvents, or use the Delegates that are much faster but don't have an editor representation. When the states are defined and the methods are assigned, you can toggle the "Editable" button off, and then leave the FSM to be used for less critical tasks from the editor.



For a good example of what is achievable with it, refer to the included Example.

FAQ

- Who is the target of this framework ? Is it for programmers, for designers or both ?

Every module in this framework is designed to be used by programmers and designers on a team, or by individuals with both profiles, as it is in the majority of Unity 3D projects. The GameManager module makes it easy to manage multi scene capabilities, and can be setup without writing a single line of code.

The FSM module, as it says in its documentation chapter, is made with the premise of making a fixed state machine by code, and then making possible to decorate the states with UnityEvents from the editor, without risking the flow of the FSM.

The Director module can be used without dropping a single line of code, but its real power could be unlocked using simple script commands to manage its flow.

- Can the modules be used separately ?

GameManager depends on the Common module, which includes Director and FSM. Both Director and FSM are independent from GameManager and within them.

- What other information references aside this document I have ?

Game Manager Framework includes a dedicated example for every major module of the system. The entire code base is documented and every editor script has tool tips and visual information. Also, all the components will highlight in red every editor reference and configuration necessary to make it work.

If you need help or advice using it, you can contact me and I'll be happy to help.

Contact

Email : fdt.dev@gmail.com
Twitter : https://twitter.com/FDTDev