**Impulse Framework**

This barebones framework is designed to help you expedite the creation of your next idea in Unity. It includes a number of highly customizable prebuilt systems such as a scene manager, mobile-optimized main menu, cameras, and more.

**General Setup**

If you choose not to use prebuilt managers that come with the framework, feel free to skip this section. However, if you do not already have a solution in mind for switching scenes, setting up the main menu, or handling music, it is highly recommended that you configure the general setup.

In the Build Settings, set the Splash scene to 0 and Menu to 1. Unity preloads everything in each scene, with the exception of the first scene (scene 0). For optimal performance, you should keep your splash scene as lightweight as possible and try not to add too many more objects.

**Set a Custom Splash Image**

Most games have a splash image or studio logo shown before the game begins. This framework can be set up to display a custom splash image before loading the main menu.

A video demonstration of setting the splash image, setting up scene loading, and customizing the menu can be found here: <https://www.youtube.com/watch?v=btNqHCoRwB8&list=PLLXw4Fw6qNw5WVLPn1hhJNEcwXjxt3b9j&index=1>

1. Locate the Resources/Prefabs/Scene/SplashFadeIn object in the project files (not in the hierarchy!). Select the ImageToFade child object.
2. Set the Source Image of the Image component to whatever splash image you want to display.

**Set Scene Loading Parameters (fade in/out, interpolation, duration)**

The framework provides a scene manager to handle transitioning between scenes, such as fading in/out and transition delay.

Refer to the video in the section above for a video walkthrough of the scene manager.

1. Locate the Resources/Prefabs/Scene/SceneManager object.
2. In the SceneManager component, you can specify the Duration of fade in/out as well as the Interpolation of the fade. If you do not want to fade in/out scenes, set the duration to 0.

**Customize the Main Menu**

The framework provides a customizable main menu that is contained within a single scene in order to remain mobile-optimized.

Refer to the video in the section above for a video walkthrough of the main menu.

1. Open \_Scenes/Menu.unity
2. Open the MenuSystem object. You’ll notice a main menu and options menu are already set up for you, but are inactive.
3. Create a new child object under MenuSystem and attach the MenuScreen script to it.
4. Add your new menu elements to this new child object.
5. Set your new child object as inactive once you are finished with it.

To switch menus using UGUI OnClick(), call the MenuManager.ChangeMenuAndFade() or MenuManager.ChangeMenu() function.

**Utility**

**Playing Music and Managing Playlists**

The music manager and music playlist system allow for easy playback and organization of background music within scenes.

For a video demonstration of the music manager and music playlists: <https://www.youtube.com/watch?v=jQGTqGalGVw&index=2&list=PLLXw4Fw6qNw5WVLPn1hhJNEcwXjxt3b9j>

1. Drag the MusicManager prefab from Assets/Prefabs/Music/MusicManager into your splash scene, or whichever scene is the first one in your build settings. The MusicManager is persistent from scene to scene, so you do not need to instantiate it in each scene.
2. In each scene where you want music to be played, create a new empty game object and attach the MusicPlaylist.cs script. This script can be found in Assets/Scripts/Music/MusicPlaylist.cs. I recommend naming the game object ‘MusicPlaylist’. Then, just populate the Music List array in the game object with song files. Leave ‘Activate On Awake’ to true if you want the playlist to begin playing as soon as the scene is loaded.

**Subscribing To and Posting Notifications (Event Messaging System)**

The event messaging system can be found in “Assets/Scripts/Utility/Notifications”. However, you do not need to put any scripts from that folder onto your gameobjects to use the notification system.

POSTING A NOTIFICATION

Begin by creating a const string to represent the notification you will post. In this example, we are creating an OnClick() function for when the user clicks on a cell in a menu. The script is called ProgramCellSelect.cs and this is our const string:

// The notification sent when this tile is selected.

public const string ProgramCellSelectNotification = "ProgramCell.SelectNotification";

The function we want to call for the OnClick() event is simply:

public void OnProgramCellSelectClick()

{

this.PostNotification(ProgramCellSelectNotification);

}

The PostNotification() function is all you need to do to call to post a notification.

SUBSCRIBING TO A NOTIFICATION

To make something happen when a notification is posted, you need to subscribe to that notification. Subscribers can be other game objects or states within a state machine waiting for an event to happen.

In our example, we want to get which menu cell was clicked. The menu is only shown during a ShowMenu state, so we begin by adding subscriptions to the notifications that enable while the state is active and disable when the state is not:

void OnEnable()

{

this.AddObserver(OnInventoryCellSelectNotification, ProgramCellSelect.ProgramCellSelectNotification);

}

void OnDisable()

{

this.RemoveObserver(OnInventoryCellSelectNotification, ProgramCellSelect.ProgramCellSelectNotification);

}

The first argument is the function we call when we receive the notification (see below) and the second is the notification we are listening for.

Now that we are subscribed to the notification, we can write the code to act upon that notification:

void OnInventoryCellSelectNotification(object sender, object args)

{

// Get selected cell.

ProgramCellSelect selectedProgramCell = sender as ProgramCellSelect;

}

Note that your notification function needs to have ‘sender’ and ‘args’ arguments. ‘sender’ is the object posting the notification. We use this to determine which menu cell was clicked by the user. ‘args’ is additional data that might be included with the notification.

**Cameras**

**Top-Down Camera**

This camera is best suited for 2D games.

For a video demonstration of the top-down camera: <https://www.youtube.com/watch?v=DLTyrbMxytA&list=PLLXw4Fw6qNw5WVLPn1hhJNEcwXjxt3b9j&index=3>

1. Locate the script in Assets/Scripts/Camera/TopDownFollow\_Camera.cs.
2. Attach this script to a camera object in your scene.
3. Drag a Transform into the Follow Target parameter. This is the object the camera will try to follow.
4. Set the Target Offset and Move Speed parameters to your liking. Target Offset is x,y,z distance from the follow target (the camera position offset relative to the follow target object). Move speed is how fast the camera moves when the object moves.

**Third Person Camera**

This camera is based on the camera used in many popular MMORPG games and automatically zooms in when the follow target is obstructed by an object.

For a video demonstration of the third person camera: <https://www.youtube.com/watch?v=DDdnLPPZXLg&index=4&list=PLLXw4Fw6qNw5WVLPn1hhJNEcwXjxt3b9j>

1. Locate the script in Assets/Scripts/Camera/Third\_Person\_Camera.cs
2. Attach this script to a camera in your scene.
3. Create an empty game object and rename it to ‘LookAt’. This is the object the camera will focus on and follow.
4. Make the LookAt object a child object of the gameobject you want to follow.
5. Assign the LookAt object in the Target Look Transform parameter of the Third Person Camera component on the camera.
6. To add mouse controls such as zoom-in with the mouse scrollwheel, attach the Third\_Person\_Mouse\_Input.cs script to the camera. This script is located in the Assets/Scripts/Camera folder.

**AI**

AI scripts in the framework are based around a Faction component that specifies what faction a gameobject belongs to. For a gameobject to be used with the AI scripts, it must have the Faction.cs script attached along with a faction specified (factions can be neutral in addition to friendly or hostile).

Faction.cs is located in the Assets/Scripts/AI folder.

**Viewcones**

These are procedurally generated cones that can be used to give a gameobject the ability to ‘see’ other gameobjects.

*TODO: Change viewcones from using tags to factions instead.*

**Sphere Detector**

The sphere detector projects an invisible sphere around an object. The idea is other objects within this sphere are ‘detected’ by the object, similar to radar. This system does not actually involve AI behavior, but can be useful in setting one up.

For a video demonstration of the sphere detector: <https://www.youtube.com/watch?v=1ZLkDv9OUNc&list=PLLXw4Fw6qNw5WVLPn1hhJNEcwXjxt3b9j&index=7>

1. Locate the script in Assets/Scripts/AI/Detector.cs
2. Attach the Detector script to a gameobject.
3. In the Detector component, assign allied and enemy factions. Objects belonging to a faction that is not assigned will show up under the ‘Detected Neutral’ array during runtime.

The Detector component includes useful methods for fetching data during runtime:

*TODO: Insert methods here.*