

Unity Playable Ads Kit | Playable Ads Package

Topic Overview:

- [What is it?](#)
 - [How to set it up?](#)
 - [Playable Ads Kit Editor Window](#)
 - [How do the elements work?](#)
 - [Playable Ads Kit Editor Window](#)
 - [Default Folder Hierarchy Element](#)
 - [CTA Controller Element](#)
 - [Playable Base Canvas Element](#)
 - [Banner Controller Element](#)
 - [EndCard Controller Element](#)
 - [Tutorial Controller Element](#)
 - [Audio Manager Element](#)
 - [Timer Controller Element](#)
 - [Object Pool Manager Element](#)
 - [Unity Packages Element](#)
 - [Additional Helpful Sources](#)
 - [Playable Ads Kit Utilities](#)
 - [Playable Assets](#)
-

What is it?

We consistently generate Playable Ads. This package streamlines the playable ad development process, providing essential assets and elements, including basic playable components. It automates the playable ad development process.

This package allows you to start your playable ad processes quickly and facilitates a faster onboarding process, thanks to its inclusion of essential elements.

Its purpose is to simplify your workflow, so we kindly encourage you to invest the time to peruse and comprehend its functionality.

How to set it up?

Please follow these steps to set up the Unity Translation Package for your Unity project.

1. **Install** the **Luna Plugin** via the Package Manager

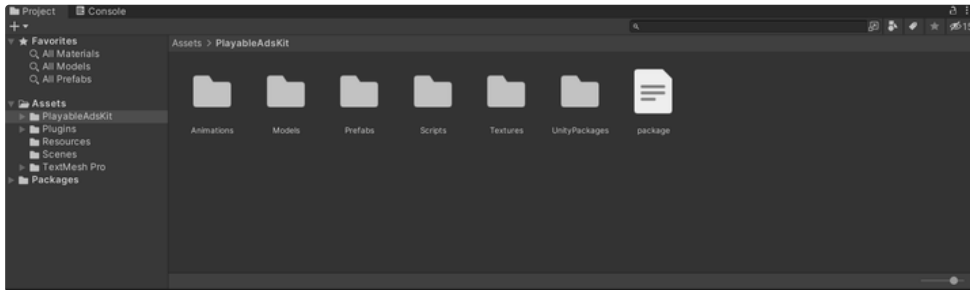
✖ The **PlayableAdskit** package is designed to work even if Luna is not installed in your Unity project. This is because we want this setup to run smoothly in Unity projects that are not specifically targeting Luna. However, if you wish to make use of store connections and playable ad features seamlessly, the **Luna Plugin must be downloaded** and added to your project.

2. **Download** the latest Unity package “**Playable Ads Kit**” from the following link:

[PlayableAdskit](#)

3. Once the download is complete, **open** your **Unity Project**.
4. To **import** the **package**, you have two options:

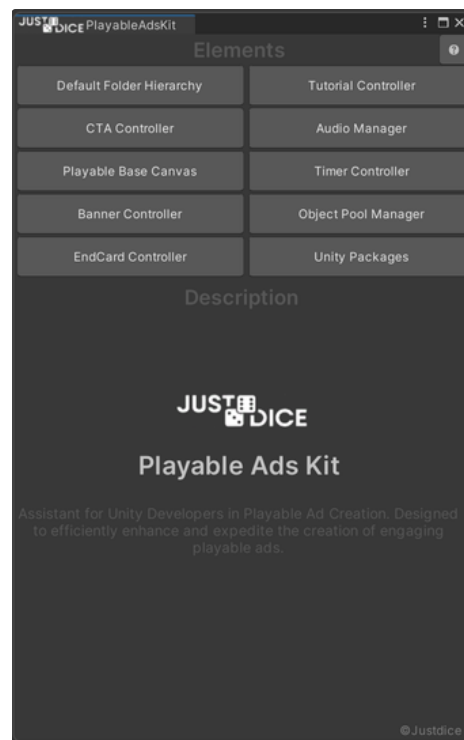
- *Option 1:* Double-click the downloaded Unity package file, and Unity will automatically open and prompt you to import the package.
- *Option 2:* In the Unity Editor, go to the "Project" window, right-click, and choose "Import Package" > "Custom Package...". Locate the downloaded package file and select it to begin the import process.



Expected post-import appearance

- Download and Import **DoTween (HOTween v2)** package as part of the Luna installation. You can find it under **PlayableAdsKit > UnityPackages**.
- You can **access** the **Playable Ads Kit** tool by going to **Tools>JustDice>Playable Ads Kit** or by using the shortcut **Ctrl + F1**.

Playable Ads Kit Editor Window



Playable Ads Kit Editor Window

Through the **"Playable Ads Kit" editor window**, you can view the elements that will be used during the onboarding stage of playable development and provide quick access to functions under the **Elements section**.

Once a button for an element is selected, in the **Description section**, you'll find a title, a brief description, and the characteristics of the chosen element. These characteristics can be customized to achieve more tailored results as per your needs.

i You can access the **help** and information page about the tool by clicking on the **'?'** button located in the top right corner of the editor window.

How do elements work?

Default Folder Hierarchy Element


```
8  public class DefaultFolderHierarchy : KitButtonBase
9  {
10     private readonly string _mainFolderPath = "Game";
11
12     private readonly string _animationsFolderPath = "Animations";
13     private readonly string _audiosFolderPath = "Audios";
14     private readonly string _fontsFolderPath = "Fonts";
15     private readonly string _materialsFolderPath = "Materials";
16     private readonly string _modelsFolderPath = "Models";
17     private readonly string _particlesFolderPath = "Particles";
18     private readonly string _prefabsFolderPath = "Prefabs";
19     private readonly string _scenesFolderPath = "Scenes";
20     private readonly string _scriptsFolderPath = "Scripts";
21     private readonly string _texturesFolderPath = "Textures";
22
23     private readonly string _prefabsUIFolderPath = "UI";
24     private readonly string _scriptsBehavioursFolderPath = "Behaviours";
25     private readonly string _scriptsControllersFolderPath = "Controllers";
26     private readonly string _scriptsHelpersFolderPath = "Helpers";
27     private readonly string _scriptsManagersFolderPath = "Managers";
28 }
```

Default Folder Hierarchy Element

It allows you to create the folder structure to be used in the playable development process with a single button click. During the folder creation process, a scene to be used for the playable named “*Main*” is also generated, and the scene is opened automatically.

This is designed solely for creating the most commonly used folder structure with a single click. There are no issues with using any other file organization method.

 Ensure that the newly created **scene** is **active** in the **Luna** plugin under **Settings/Basic/Scenes in Build** section.

 If you want to personalize the folder names in the default folder structure that will be imported from the Default Folder Hierarchy element, you can make changes to the folder names inside the **DefaultFolderHierarchy** class, which derives from the **KitButtonBase** base class. You can access the DefaultFolderHierarchy class from *PlayableAdsKit/Scripts/Editor/DefaultFolderHierarchy.cs*

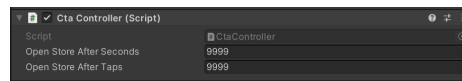
```

8 public class DefaultFolderHierarchy : MonoBehaviour
9 {
10     private readonly string _mainFolderPath = "Game";
11
12     private readonly string _animationsFolderPath = "Animations";
13     private readonly string _audiosFolderPath = "Audios";
14     private readonly string _fontsFolderPath = "Fonts";
15     private readonly string _materialsFolderPath = "Materials";
16     private readonly string _modelsFolderPath = "Models";
17     private readonly string _particlesFolderPath = "Particles";
18     private readonly string _prefabsFolderPath = "Prefabs";
19     private readonly string _scenesFolderPath = "Scenes";
20     private readonly string _scriptsFolderPath = "Scripts";
21     private readonly string _texturesFolderPath = "Textures";
22
23     private readonly string _prefabsUIFolderPath = "UI";
24     private readonly string _scriptsBehavioursFolderPath = "Behaviours";
25     private readonly string _scriptsControllersFolderPath = "Controllers";
26     private readonly string _scriptsHelpersFolderPath = "Helpers";
27     private readonly string _scriptsManagersFolderPath = "Managers";
28 }

```

DefaultFolderHierarchy default folder names

CTA Controller Element

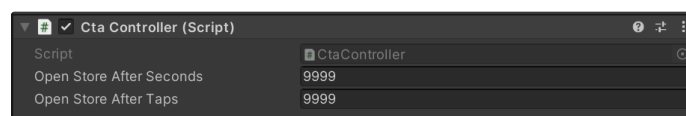


Cta Controller Element

This element allows us to import the **CtaController** singleton class. It enables store redirections and triggering Luna events provided by the Luna Plugin.

✖ **CtaController** class is **crucial for playable ads** because it helps us direct players to the store through this class. The other essential scripts are also redirects players to the store through this singleton class` methods.

CtaController Class



Cta Controller Class

The CTA Controller class contains methods that will redirect the player to the store. It showcases specific conditions on its two fields, which can trigger store calls both within Unity and from the Luna Playground.

⚠ We typically leave these fields as 9999 because, without an end card, directly sending the player to the store through time-based counting or by tapping **can be rejected by ad networks**. Therefore, we prefer to redirect players to the store through an end card or banner instead of directly calling the store. However, these fields are included within the CtaController class for cases where it might be necessary.

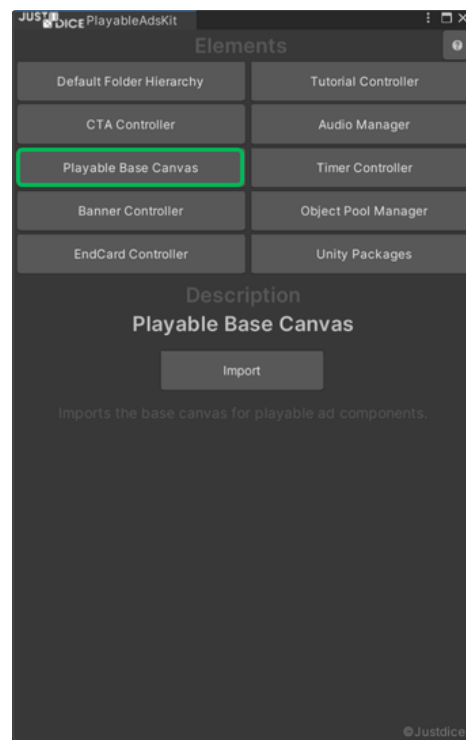
You can call the methods that will redirect the player to the store as follows:

```
1 // Store call without event
2 CtaController.Instance.OpenStore();
3
4 // Store call with GameEnded event
5 CtaController.Instance.OpenStoreWithGameEnded()
```

i `Luna.Unity.LifeCycle.GameEnded();`

is a predefined event provided by Luna. Some networks require this event to be called at the end of the game. Therefore, it is generally recommended to call this event either externally in any script of the game or through the CTAController.

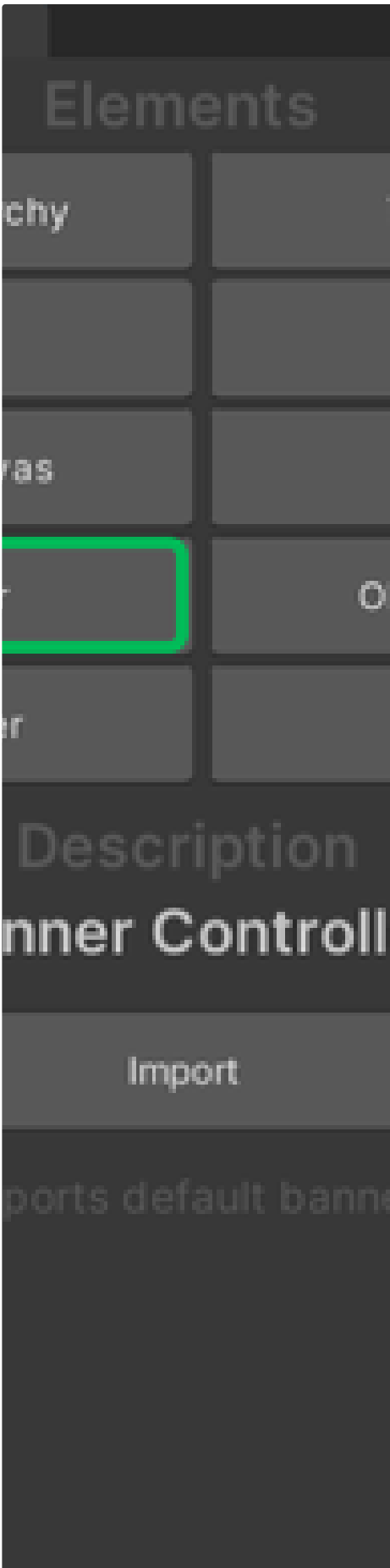
Playable Base Canvas Element

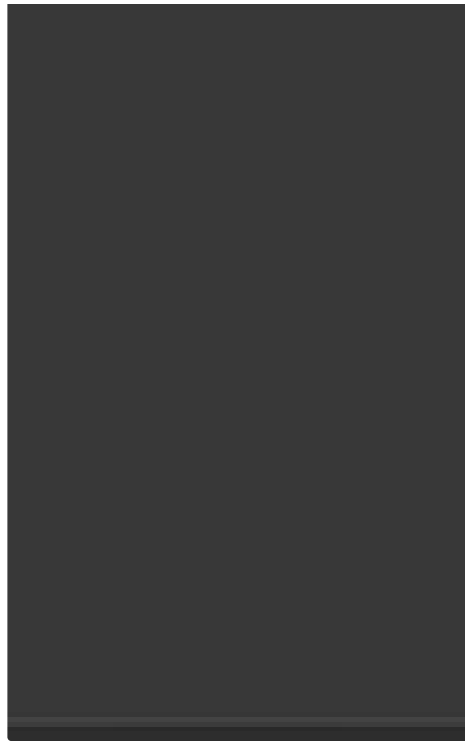


Playable Base Canvas Element

It creates the canvas and event system that will be used within the playable, incorporating UI elements. By default, it's set to a reference resolution of `1125x2436` and a match ratio of `height:1`. This is because it ensures a narrow mobile screen, preventing UI elements from overflowing the screen's boundaries on a narrow display.

Banner Controller Element



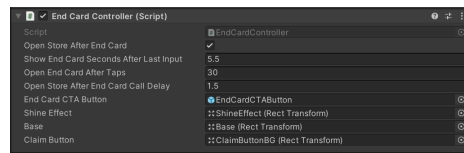


Banner Controller Element

It serves to import the banner that will directly lead the user to the store within the game. The banner, without the need for an end card, redirects the player to the store when tapped. The imported banner contains simple animations and a basic placeholder structure, primarily for providing developers/designers with ideas.

- [-] If you want to customize the banner, after making design changes, you can add the requirements through the **BannerController** class.

EndCard Controller Element

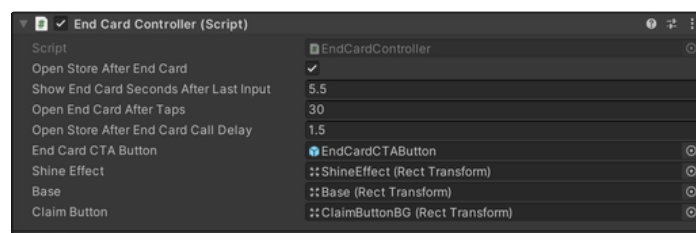


EndCard Controller Element

The EndCard is the element that signals the completion of the playable ad and displays information about the product along with its logo to the player. **The goal is to direct the player to the store through this EndCard.**

⚠ Another crucial element for Playable Ads is the EndCardController because it handles the display of the end card at the end of the game and the subsequent redirection of the player to the store.

EndCardController Class



EndCardController Class

There are several fields on the EndCardController related to the timing of EndCard calls. The default values of these fields are as shown in the image above,

- **"Open Store After End Card"** is used to determine whether the store will automatically open after the end card display.
- **"Show End Card Seconds After Last Input"** indicates how many seconds after the player's last interaction with the playable ad the EndCard will be called.
- **"Open End Card After Taps"** specifies the number of taps by the user before the EndCard is called.
- **"Open Store After End Card Call Delay"** typically, players are automatically directed to the store after the end card. This delay indicates how many seconds after calling the end card method the auto-redirect to the store will occur. It's essential to consider the duration of end card animations when setting this delay.

These fields **can be adjusted** both **within Unity** and **through the Luna Playground**.

✖ The maximum duration for **IronSource** network between the player's last interaction and calling the store connection is **7 seconds**. This means that the sum of "**Show End Card Seconds After Last Input**" + "**Open Store After End Card Call Delay**" must not exceed this duration.

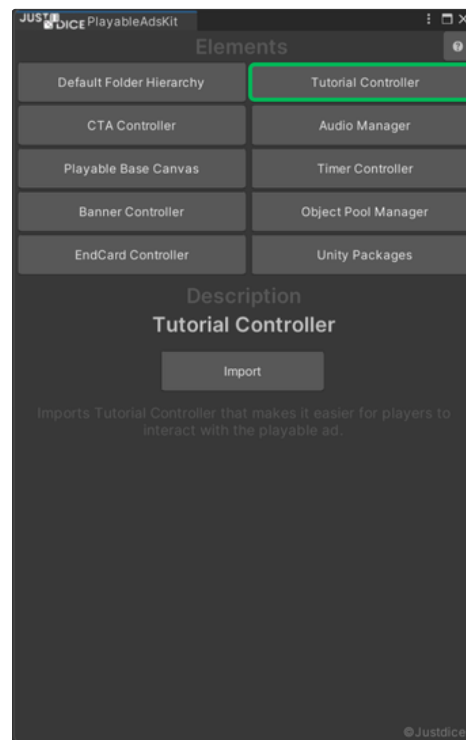
EndCard can be called at any moment during the game or after an event from the singleton EndCardController class through the following code:

```
1 // Calls the endcard
2 EndCardController.Instance.OpenEndCard();
```

In some cases, if you want to **close the EndCard** and continue the game from where it left off, you can call the method to close the EndCard as follows:

```
1 // Closes the endcard
2 EndCardController.Instance.CloseEndCard();
```

Tutorial Controller Element



Tutorial Controller Element

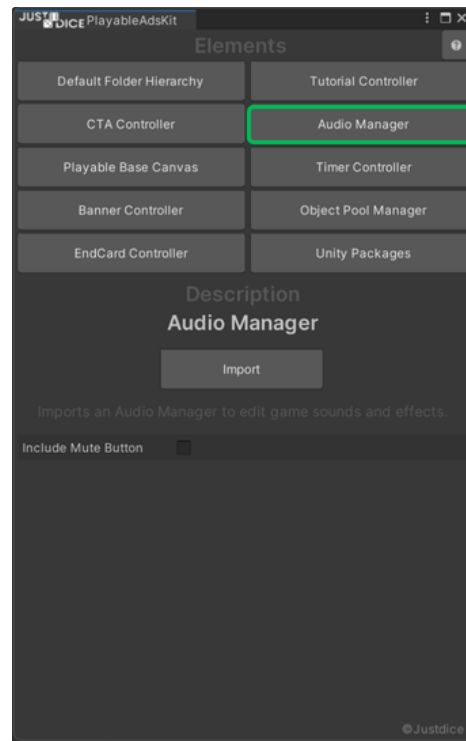
It imports the element that contains the necessary tutorial tools for introducing the game to the player. Among these introductory tools, there are **tutorial text** and **tutorial hand**. These are typically used to explain tasks that the player needs to perform or the logic of the game. With placeholder objects, along with simple animations, these tutorial elements are attached to objects in the scene through the tutorial controller and then displayed.

Default locations on the scene are assigned for the tutorial text parent and tutorial hand, which can be changed or removed as needed.

For the Tutorial Hand, Playable Ads Kit includes 3 animations (Swerve, Horizontal, and Click) by default.

📖 The methods that control the tutorial elements can be accessed through **TutorialController.cs**.

Audio Manager Element

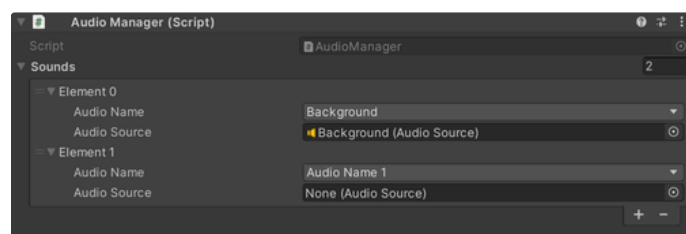


Audio Manager Element

It allows the **Audio Manager** singleton class to be imported, enabling **control over** the **in-game music** and **sound effects**.


Additionally, by enabling the '**Include Mute Button**' toggle, you can add a button in the game that allows players to mute the game's sounds.

AudioManager Class



AudioManager Class

The Audio Manager singleton class is used to assign sounds that will be used in the game. Later, these sounds **can be called** in the game **using the Audio Name** as needed. The *Sounds* list consists of elements that contain one *Audio Name* and one *Audio Source* each.

 The names for the sounds to be used in the game can be assigned from the **AudioNames** enum in **AudioManager.cs**

```

5 namespace PlayableAdsKit.Scripts.Base
6 {
7     [Serializable]
8     public enum AudioName
9     {
10         Background,
11         AudioName1,
12         AudioName2,
13         AudioName3
14     }
15
16     [Serializable]
17     public struct Sound
18     {
19         public AudioName AudioName;
20         public AudioSource AudioSource;
21     }
22
23     public class AudioManager : SingletonBehaviour<AudioManager>
24     {
25

```

AudioName enum

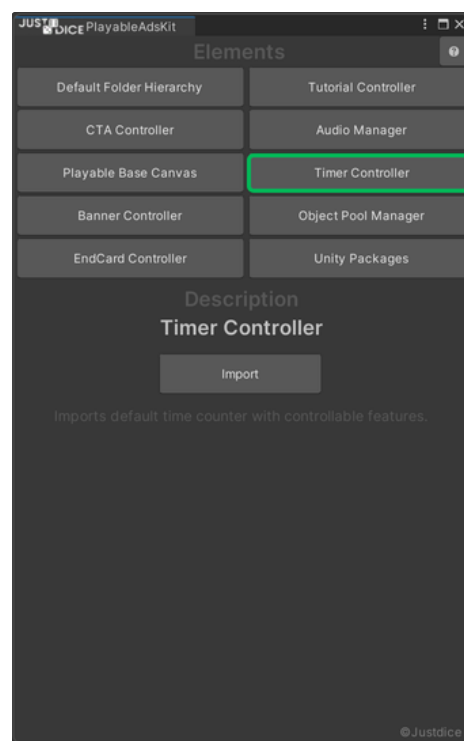
In the game, sounds can be played, paused, and stopped as needed as follows;

```

1 // Plays AUDIO_NAME sound
2 AudioManager.Instance.PlaySound(AUDIO_NAME);
3
4 // Pauses AUDIO_NAME sound.
5 AudioManager.Instance.PauseSound(AUDIO_NAME);
6
7 // Stops AUDIO_NAME sound.
8 AudioManager.Instance.StopSound(AUDIO_NAME);

```

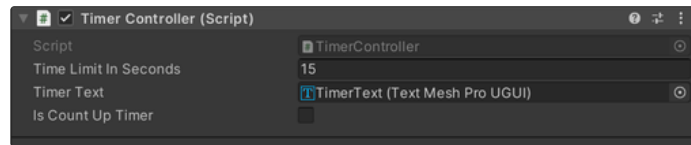
Timer Controller Element



Timer Controller Element

It imports the TimerController, which provides **time control within the game**. It is suitable for usage when time-based conditions are part of the game structure. Along with the TutorialController class, it also creates placeholder timer canvas elements.

TimerController Class



TimerController Class

Setting time limits and configuring the time flow logic is done through the TimerController

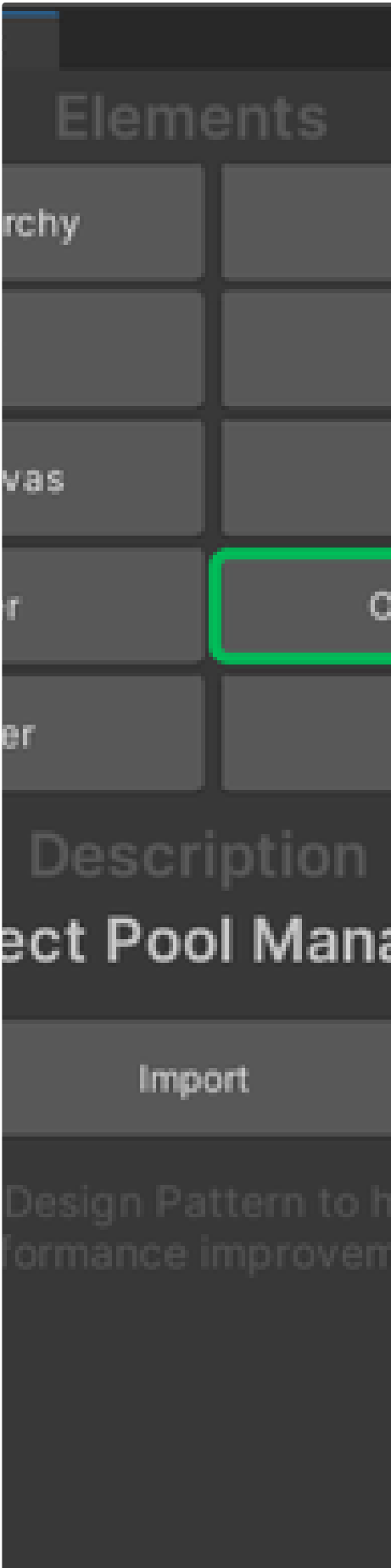
- **"Time Limit in Seconds"** helps determine the game duration in seconds.
- **"Is Count Up Timer"** is used to set whether the counter will start from 0 and count up to the specified *"Time Limit in Seconds"*, or count down from the specified value to 0.

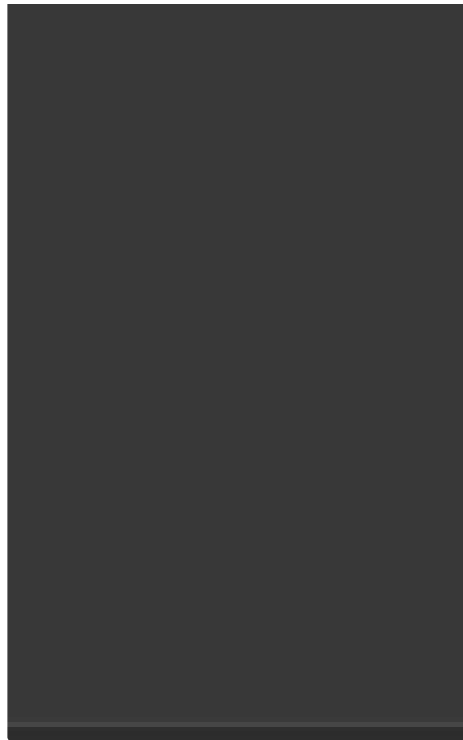
After the specified time, it triggers an event that can be used for specific in-game situations.

Here are some **useful methods** that can be used **for the TimerController** singleton class:

```
1 // Starts timer
2 TimerController.Instance.StartTimer();
3
4 // Stops timer
5 TimerController.Instance.StopTimer();
6
7 // Toggles timer
8 TimerController.Instance.ToggleTimer();
9
10 // Sets timer to specific value _VALUE_TO_SET_
11 TimerController.Instance.SetTimer(_VALUE_TO_SET_);
12
13 // Resets timer
14 TimerController.Instance.ResetTimer();
```

Object Pool Manager Element

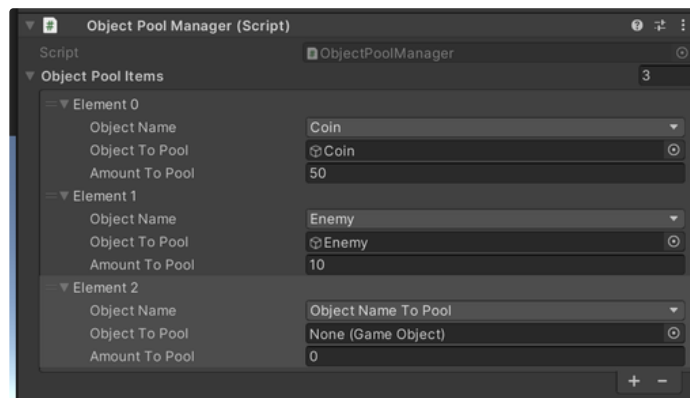




Object Pool Manager Element

It is used for importing the Object Pool Manager, one of the optimization design patterns that can be used for creating and destroying objects. It can be helpful in playable ad development when specific objects need to be used a certain number of times.

ObjectPoolManager Class



ObjectPoolManager Class

At the beginning of the game, the necessary objects are called from the prepared pool for each “Object Pool Item” using the ‘ObjectName.’ Afterward, these objects are used for different purposes. Once their usage is complete, the objects are simply deactivated and added back to the pool, making them available for reuse when needed.

The “Object Pool Item” to be used in the object pool is determined through the ObjectPoolManager, specifying the ‘ObjectName’, the object’s prefab, and the pool size.

[-] Appropriate **object names** for objects can be configured through the **ObjectName** enum in **ObjectPoolManager.cs**.

```

6 namespace PlayableAdsKit.Scripts.Base
7 {
8     public enum ObjectName
9     {
10         Coin,
11         Enemy,
12         ObjectNameToPool
13     }
14
15     [Serializable]
16     public class ObjectPoolItem
17     {
18         [SerializeField] private ObjectName _objectName;
19         [SerializeField] private GameObject _objectToPool;
20         [SerializeField] private int _amountToPool;
21
22         public ObjectName ObjectName => _objectName;
23         public GameObject ObjectToPool => _objectToPool;
24         public int AmountToPool => _amountToPool;
25         public List<GameObject> PooledObjects { get; set; } = new List<GameObject>();
26     }
27
28     public class ObjectPoolManager : MonoBehaviour
29     {
30         [SerializeField] private ObjectPoolItem[] _objectPoolItems;
31
32         public static ObjectPoolManager Instance { get; private set; }
33
34         private void Awake()
35         {
36             if (Instance != null && Instance != this)
37                 Destroy(gameObject);
38             else
39                 Instance = this;
40
41             CreatePool();
42         }
43     }

```

ObjectName enum

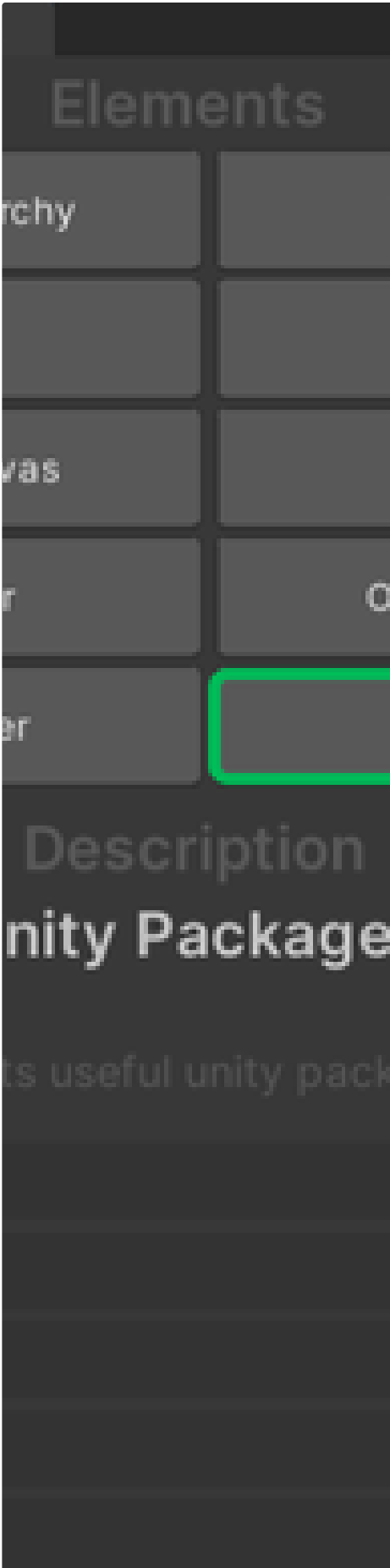
The usage of objects from the Object Pool within the game is achieved with the following code snippets:

```

1 // Call pooled object
2 GameObject pooledObject = ObjectPoolManager.
3     Instance.GetPooledObject(_POOLED_OBJECT_NAME_);
4
5 ...
6
7 // Deactivate pooled object
8 pooledObject.SetActive(false);

```


Unity Packages Element





Unity Packages Element

It's an element that serves as a centralized **hub for** commonly used **packages** in the playable development process. Through this element, packages that can be used during playable development can be imported.

 If the **AnimationBaker** package is **imported**, the Animation Baker folder **must** be **excluded** from the *Code* section in **Luna Plugin**.

Additional Helpful Sources

Playable Ads Kit Utilities

The **PlyAdsKitUtils** static class, which contains some **methods to assist** in the playable development process, can be accessed from any script, allowing you to use the necessary utility methods.

You can access this class through *PlayableAdsKit/Scripts/Utilities/PlyAdsKitUtils.cs* path.

```
1 // Calling utility methods
2 PlyAdsKitUtils.METHOD_NAME_TO_CALL();
```

Playable Assets

Commonly used or highly likely to be used **assets** in the playable process have been **included in the package**.

These assets can be accessed through the following paths;

- *PlayableAdsKit/Models*
- *PlayableAdsKit/Textures*.