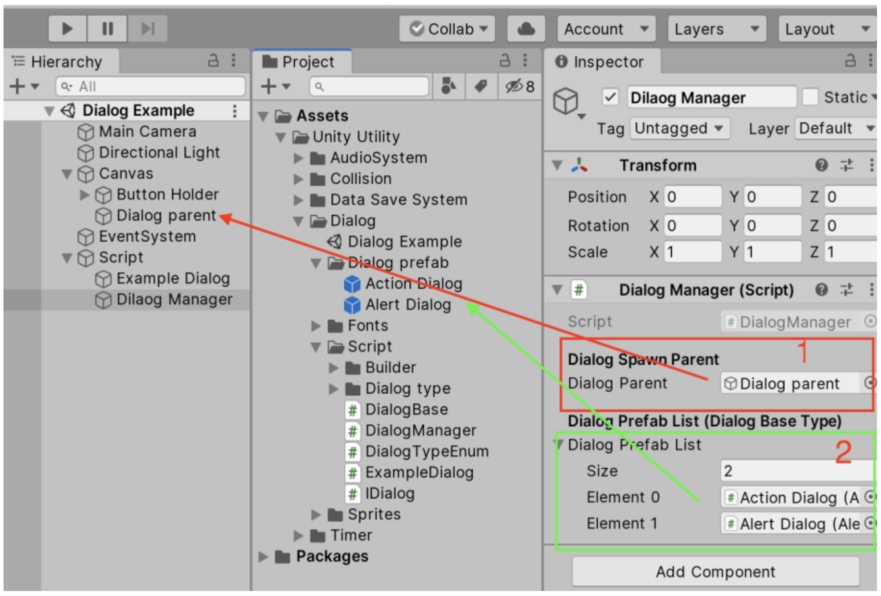
Dialog System

Dialogues are an essential part of a game or application. Creating a dialog and managing its button click event becomes a disaster. If you do not manage your dialogues properly in your project then you may face various difficulties. This system will solve all of your hazards. Every dialog will build using the builder pattern where you can set the button click events during the build that dialog

How to Use: Please follow the following instructions

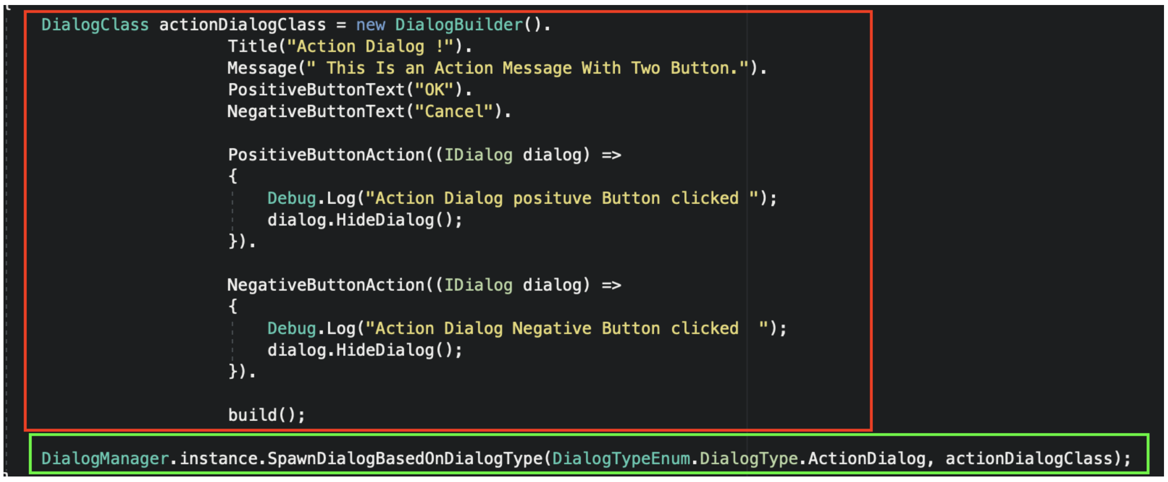
1. Attach DialogManager.cs script in any GameObject.It has two input slots.

   a. **Dialog Parent:** Place the Dialog Parent object (where you want to show the dialog.)Please see the screenShot (number 1 with a red rectangle)



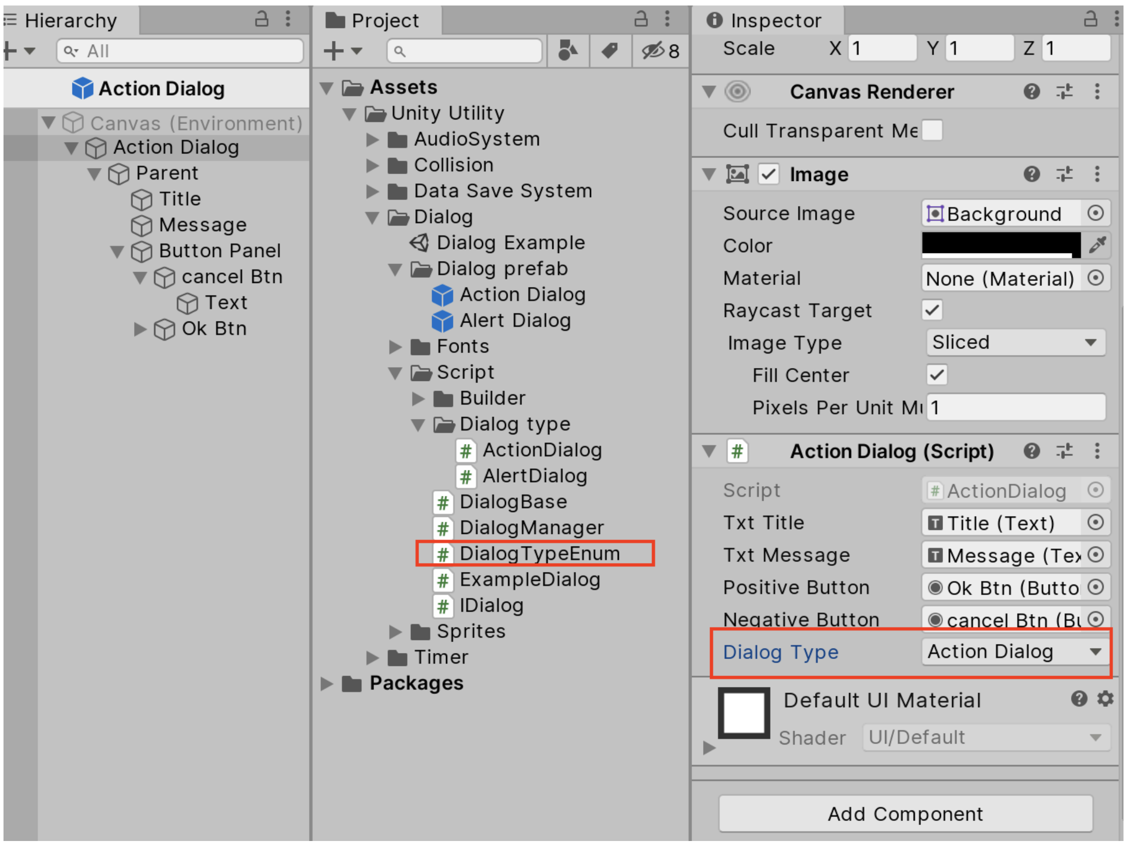
   b. **Dialog Prefab List:** Please add all dialog prefabs here. Please see the screenShot (number 2 with a green rectangle)

2. Please see the below screenshot for building and showing a dialog. The red rectangle area is for building the dialog and the green one is for showing the dialog. Please see the Dialog Example scene and ExampleDialog.cs script for better understanding.



```
DialogClass actionDialogClass = new DialogBuilder().
                              Title("Action Dialog !").
                              Message(" This Is an Action Message With Two Button.").
                              PositiveButtonText("OK").
                              NegativeButtonText("Cancel").

                              PositiveButtonAction((IDialog dialog) =>
                              {
                                  Debug.Log("Action Dialog posituve Button clicked ");
                                  dialog.HideDialog();
                              }).

                              NegativeButtonAction((IDialog dialog) =>
                              {
                                  Debug.Log("Action Dialog Negative Button clicked  ");
                                  dialog.HideDialog();
                              }).

                              build();

DialogManager.instance.SpawnDialogBasedOnDialogType(DialogTypeEnum.DialogType.ActionDialog, actionDialogClass);
```

3. You can create a new dialog or redesign existing dialog easily. Please see the existing dialog prefabs from the Assets/UnityUtility/Dialog/Dialog Prefab folder.



```
Action Dialog
▼ Canvas (Environment)
  ▼ Action Dialog
    ▼ Parent
        Title
        Message
      ▼ Button Panel
        ▼ cancel Btn
            Text
        ▶ Ok Btn
```

```
▼ Assets
  ▼ Unity Utility
    ▶ AudioSystem
    ▶ Collision
    ▶ Data Save System
    ▼ Dialog
        Dialog Example
      ▼ Dialog prefab
          Action Dialog
          Alert Dialog
      ▶ Fonts
      ▼ Script
        ▶ Builder
        ▼ Dialog type
            ActionDialog
            AlertDialog
          DialogBase
          DialogManager
          DialogTypeEnum
          ExampleDialog
          IDialog
      ▶ Sprites
    ▶ Timer
▶ Packages
```

Scale	X 1 Y 1 Z 1
Canvas Renderer	
Cull Transparent Me	☐
Image	✓
Source Image	Background
Color	■
Material	None (Material)
Raycast Target	✓
Image Type	Sliced
Fill Center	✓
Pixels Per Unit Mı	1
Action Dialog (Script)	
Script	ActionDialog
Txt Title	Title (Text)
Txt Message	Message (Tex
Positive Button	Ok Btn (Butto
Negative Button	cancel Btn (Bu
Dialog Type	Action Dialog
Default UI Material	
Shader	UI/Default

Please don't forgot to define the dialog type in dialog prefab

You can add a new dialog type from DialogTypeEnum.cs script. Please see the above screenshot.

Save System

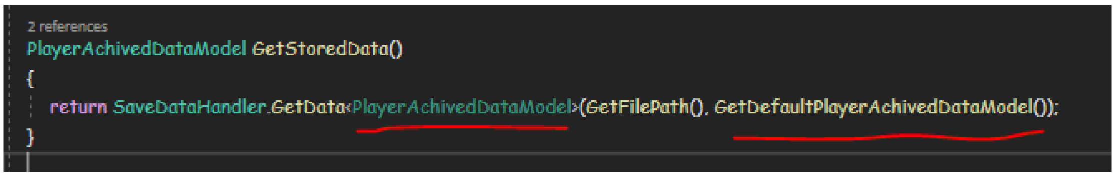
The most important thing of a game or app is to save some (like user information) data permanently. You can use a database or PlayerPrefs for storing data. But using a database, for storing a small amount of data is not recommended and PlayerPrefs has some limitations. So here comes this system. This will help you to save data like (Game data, Score Data, User Data, etc) in JSON format with a few lines of code.

How to Use:

1. **SaveData:** For save Data Please call SaveDataHandler.SaveData(object data, string filePath).

    a. **data:** Data parameter takes an object of a serialized class, which you want to store.
    b. **filePath:** It is the file location. In the current implementation, you call FileHandler.GetPersistantFilePath(string fileName) then this will give you the file location and if it was not existing that location then it will create one.

2. **Get Data:** For Get stored data just call SaveDataHandler.GetData<ClassName>(string filePath, Default Data) function. Please see the following screenshot. This is from SaveDataExampleScene.cs script which you

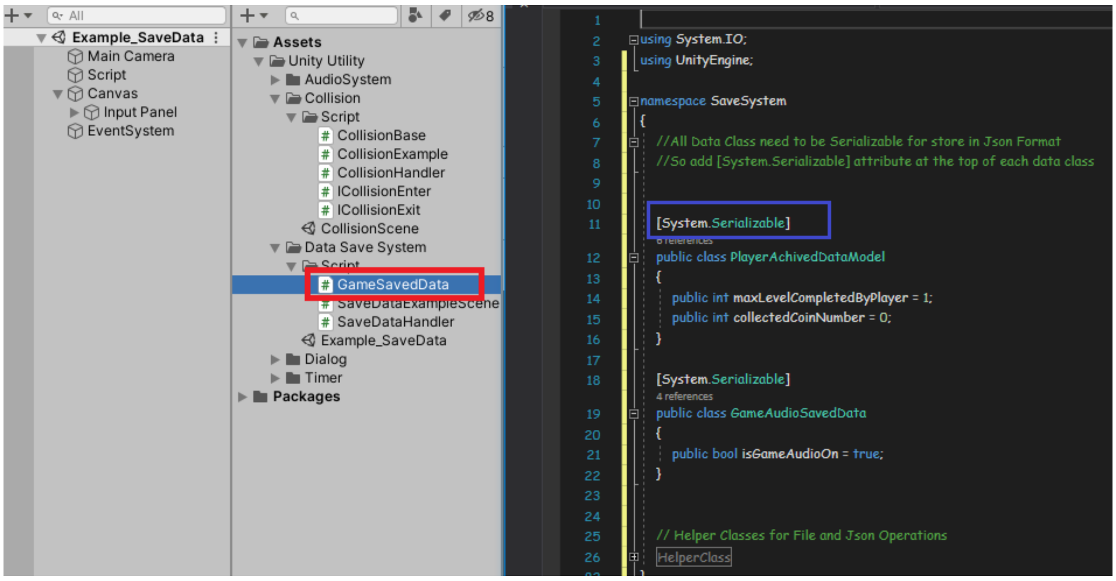


```
2 references
PlayerAchivedDataModel GetStoredData()
{
    return SaveDataHandler.GetData<PlayerAchivedDataModel>(GetFilePath(), GetDefaultPlayerAchivedDataModel());
}
```

can find in Assets/Unity Utility/Data Save System/SaveDataExampleScene.cs

Create a new DataClass:

- Create a new class. You can create anywhere but we recommend you create the data class inside GameSavedData.cs class. **You must have to add [System.Serializable] on top of the class**. Please see the following screenshot.



```

using System.IO;
using UnityEngine;

namespace SaveSystem
{
    //All Data Class need to be Serializable for store in Json Format
    //So add [System.Serializable] attribute at the top of each data class


    [System.Serializable]
    6 references
    public class PlayerAchivedDataModel
    {
        public int maxLevelCompletedByPlayer = 1;
        public int collectedCoinNumber = 0;
    }

    [System.Serializable]
    4 references
    public class GameAudioSavedData
    {
        public bool isGameAudioOn = true;
    }


    // Helper Classes for File and Json Operations
    HelperClass
```

Audio System

Sound is one of the key parts of a game. This system will help you manage all sound functionalities in one place. It will provide the game sound on/off state functionality and automatically play/pause/stop sound based on game sound state.

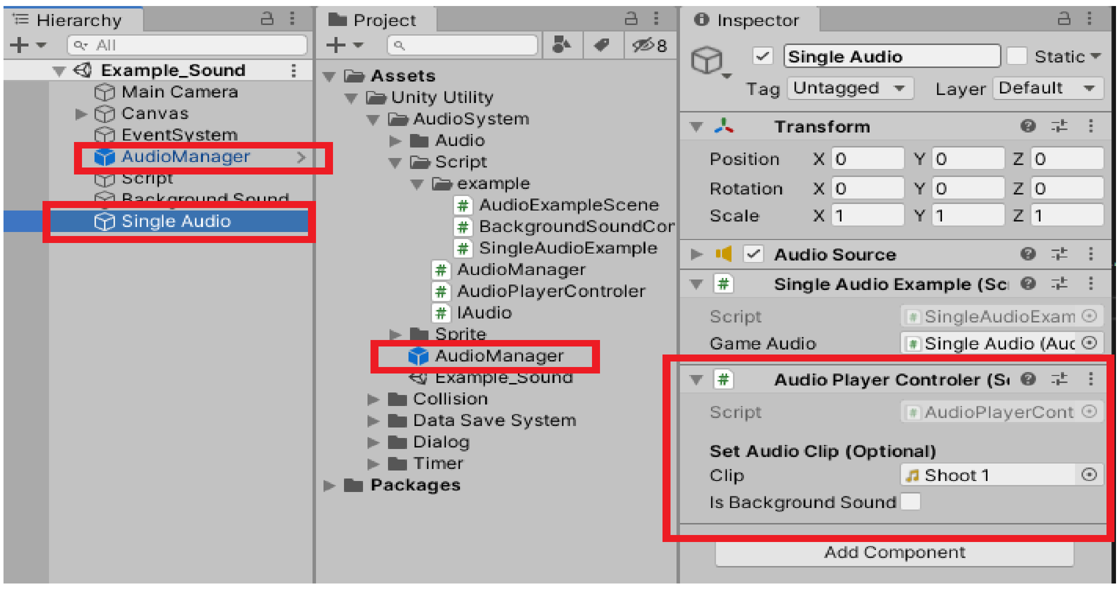
It has two key scripts 1. AudioPlayerController and 2. AudioManager

1. **AudioPlayerController :** This component controls audio play in the project. You need to attach this component to a game object. You can Found it at Assets/Unity Utility/AudioSystem/AudioPlayerController.cs.

2. **AudioManager :** This script is responsible for storing audio data (on/off). You can found it on Unity Utility/AudioSystem/AudioManager.cs.

How to Use:

1. First Drag AudioManager prefab from Unity Utility/AudioSystem/AudioManager and drop it on the project hierarchy.
2. Attach AudioPlayerController.cs script to a GameObjec t(where you want to play sound). See Example screen`s **Background Sound** and **single Audio** GameObject. Please see the following screenshot. Check IsBackgroundSound slot if that is a background sound (that sound will play or pause based on sound state)



Hierarchy
Example_Sound
Main Camera
Canvas
EventSystem
AudioManager
Script
Background Sound
Single Audio

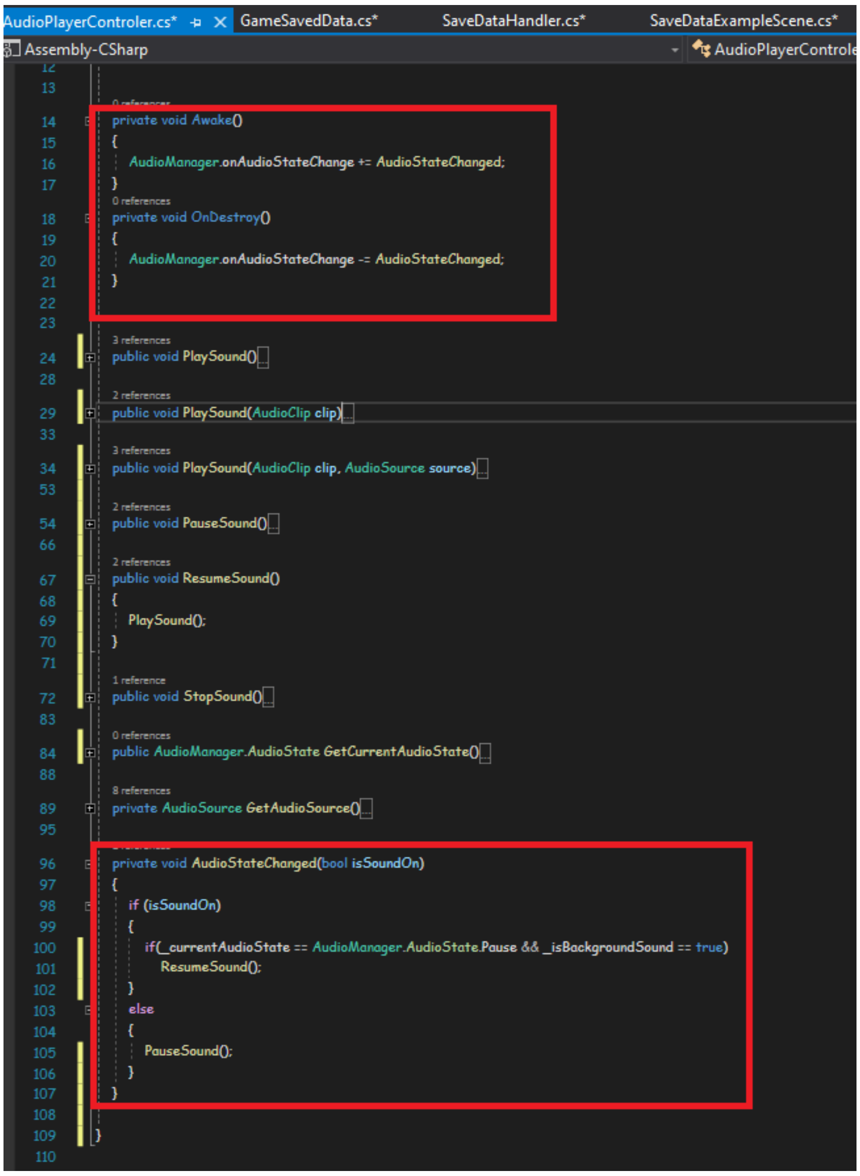
Project
Assets
Unity Utility
AudioSystem
Audio
Script
example
AudioExampleScene
BackgroundSoundCor
SingleAudioExample
AudioManager
AudioPlayerControler
IAudio
Sprite
AudioManager
Example_Sound
Collision
Data Save System
Dialog
Timer
Packages

Inspector
Single Audio Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Audio Source
Single Audio Example (Sc
Script SingleAudioExam
Game Audio Single Audio (Auc
Audio Player Controler (S
Script AudioPlayerCont
Set Audio Clip (Optional)
Clip Shoot 1
Is Background Sound
Add Component

Public API :

1. **PlaySound(), PlaySound(AudioClip clip) ,PlaySound(AudioClip clip,AudioSource source) -** This three function are responsible for play Audio.
2. **PauseSound() -** Pause the Sound
3. **ResumeSound() -** Resume Sound
4. **StopSound() -** Stop Sound
5. **GetCurrentAudioState() -** Get Current Audio State (Playing, Pause, Stop, Idle)

AudioManager script has an event Listener for detecting Audio change state.You need to register a function with that event if you want to track the audio state change event.Please see AudioExampleScene.cs script for this implementation. On that script onSOundStateChange(bool isSoundOn) function called when audio state changed.Please attach the listener to the Awake() function. Please see the following screenshot.



```
0 references
private void Awake()
{
    AudioManager.onAudioStateChange += AudioStateChanged;
}
0 references
private void OnDestroy()
{
    AudioManager.onAudioStateChange -= AudioStateChanged;
}

3 references
public void PlaySound()...

2 references
public void PlaySound(AudioClip clip)...

3 references
public void PlaySound(AudioClip clip, AudioSource source)...

2 references
public void PauseSound()...

2 references
public void ResumeSound()
{
    PlaySound();
}

1 reference
public void StopSound()...

0 references
public AudioManager.AudioState GetCurrentAudioState()...

8 references
private AudioSource GetAudioSource()...

private void AudioStateChanged(bool isSoundOn)
{
    if (isSoundOn)
    {
        if(_currentAudioState == AudioManager.AudioState.Pause && _isBackgroundSound == true)
            ResumeSound();
    }
    else
    {
        PauseSound();
    }
}

}
```

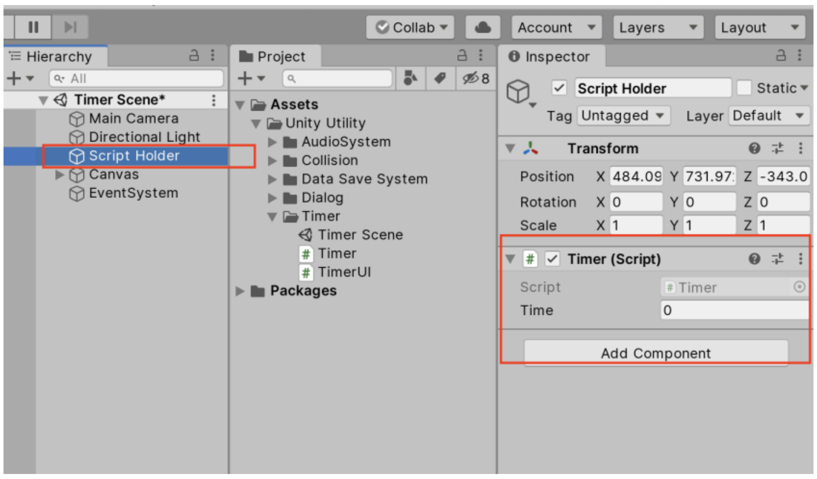
Timer System

Creating elements (bullet, enemy, etc) after a specific time is one of the key requirements of a game. This system will handle the time part automatically. You just need to start the timer with your specific time and it will automatically notify you after completing that time. You can also pause, resume, or stop the timer and also get the elapsed time.

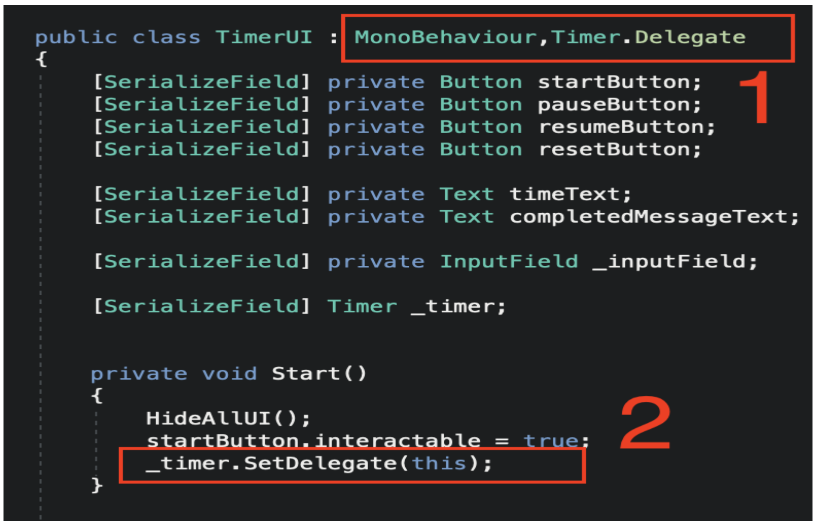
How To Use: Please follow the following steps

- Attach Timer.cs script on a GameObject and put the desired time in the Time variable. Please see the screenshot.

- Now open or create a new script where you want to set timer operations. For example I have a script name TimerUi.cs where I implement timer operations. First, write Timer.Delegate after MonoBegaviour (number 1 rectangle on image). Then in Start or Awake function set the delegate (number 2 rectangle on image).
- You have to call the StartTimer(float time) function for start timer. after your desired time OnTimeComplete() callback function will call automatically. You can also get elapsed time by calling GetElapsedTime() function. Details functions will be discussed in another section.



Script Holder Static
Tag Untagged Layer Default
Transform
Position X 484.09 Y 731.97 Z -343.0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Timer (Script)
Script Timer
Time 0



```
public class TimerUI : MonoBehaviour,Timer.Delegate
{
    [SerializeField] private Button startButton;
    [SerializeField] private Button pauseButton;
    [SerializeField] private Button resumeButton;
    [SerializeField] private Button resetButton;

    [SerializeField] private Text timeText;
    [SerializeField] private Text completedMessageText;

    [SerializeField] private InputField _inputField;

    [SerializeField] Timer _timer;


   private void Start()
   {
       HideAllUI();
       startButton.interactable = true;
       _timer.SetDelegate(this);
   }
```

Please Open **Timer Scene** from Assets/Unity Utility/Timer/Timer Scene and also open TimerUi.cs Script from the same folder for better understanding.

Public Functions:

1. **Start Timer:** To start the timer you have to call StartTimer(float time) function. After completing your passed time this system will call a callback function automatically.

2. **GetElapsedTime :** If you want to know the amount of time that passed after time has started then just call GetElapsedTime() function. This function will return the elapsed time.

3. **Pause Timer :** Call PauseTimer() function for Pause Timer.

4. **Resume Timer :** Call ResumeTimer() function for Resume Timer.

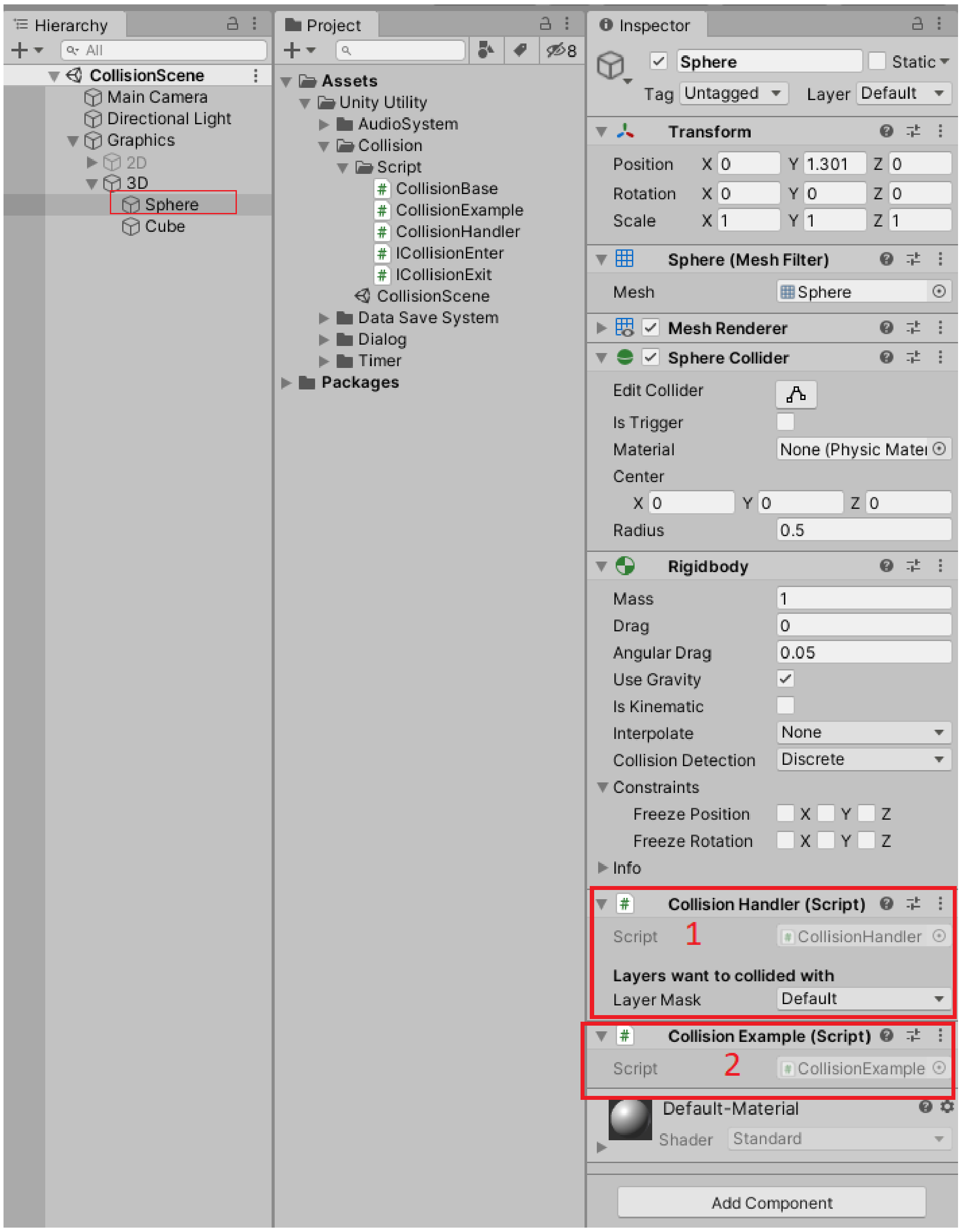
5. **Check Timer set or not :** Call IsTimeSet() function to check is timer is currently set or not.

6. **Check Timer Running or not:** Call GetIsTimerRunning() function for checking timer`s running status.

Collision System

Detecting collisions and handling collision events are not complex but sometimes it becomes a nightmare when you make some small mistakes. Using this system you just need to attach a script and implement an interface for detecting the collision and create action for that event. You don't have to worry about whether the IsTrigger is enabled or not.

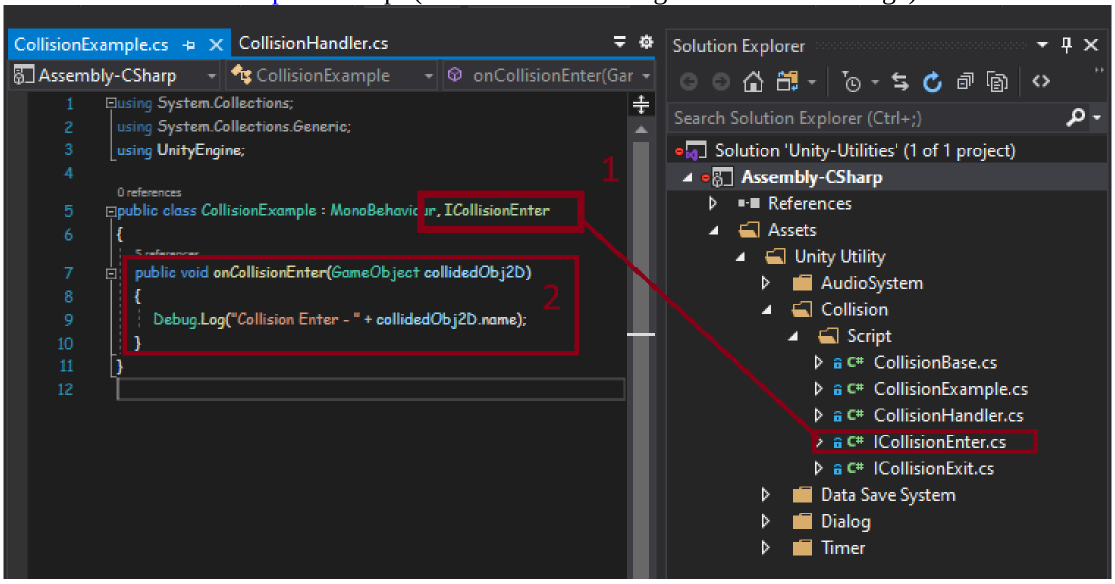
How to Use: Please follow the following instructions:



```
Hierarchy
CollisionScene
  Main Camera
  Directional Light
  Graphics
    2D
    3D
      Sphere
      Cube

Project
Assets
  Unity Utility
    AudioSystem
    Collision
      Script
        CollisionBase
        CollisionExample
        CollisionHandler
        ICollisionEnter
        ICollisionExit
      CollisionScene
    Data Save System
    Dialog
    Timer
Packages

Inspector
Sphere                      Static
Tag Untagged                Layer Default
Transform
  Position   X 0   Y 1.301   Z 0
  Rotation   X 0   Y 0       Z 0
  Scale      X 1   Y 1       Z 1
Sphere (Mesh Filter)
  Mesh                      Sphere
Mesh Renderer
Sphere Collider
  Edit Collider
  Is Trigger                ☐
  Material                  None (Physic Mate
  Center     X 0   Y 0   Z 0
  Radius                    0.5
Rigidbody
  Mass                      1
  Drag                      0
  Angular Drag              0.05
  Use Gravity               ✓
  Is Kinematic              ☐
  Interpolate               None
  Collision Detection       Discrete
  Constraints
    Freeze Position         ☐ X ☐ Y ☐ Z
    Freeze Rotation         ☐ X ☐ Y ☐ Z
  Info
Collision Handler (Script)
  Script        1           CollisionHandler
  Layers want to collided with
  Layer Mask                Default
Collision Example (Script)
  Script        2           CollisionExample
Default-Material
  Shader                    Standard
Add Component
```

- Attach CollisionHandler.cs script to the GameObject (which object you want to detect collision). Please set the LayerMask from that script (Number 2 red rectangle of the above image). It will detect collision only the layers that are set here.

- Attach another script where you write your functionalities after collision on the same Game Object where you attach CollisionHandler.cs script. In our example, we have attached CollisionExampel.cs script (Number 2 red rectangle of the above image).



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

0 references
public class CollisionExample : MonoBehaviour, ICollisionEnter
{
    5 references
    public void onCollisionEnter(GameObject collidedObj2D)
    {
        Debug.Log("Collision Enter - " + collidedObj2D.name);
    }
}
```

- Now open the script where you want to write the logic for collision detection. In this example that is CollisiopnExample.cs script. Now add interface ICollisionEnter after MonoBehaviour (Number 1 rectangle of above image). It will tell you to implement the interface. So please implement the interface (number 2 rectangle from the above image). You have to write your logic inside that function (number 2).
- If you want to detect the collision exit event then please add interface ICollisionExit and also implement that.

It will work on both 2d and 3d projects.

****Please don't forget to attach Rigidbody and collider on Game Objects.****