Overview

Key Features

- **Precise timing**: The Timer Tool allows you to measure time intervals accurately, ensuring precise execution of time-dependent events in your Unity projects.
- **Timer Events**: You can associate specific UnityEvents with the Timer to trigger actions at defined time points during the Timer's execution.
- **Looping**: The Timer can be set to automatically restart after completion, enabling repetitive or continuous timing functionality.
- **Time Modes**: Choose between Real-Time and Game-Time modes to control whether the Timer is affected by changes to the game's timescale.

Getting Started

Installation

Method 1: UnityPackage

1. Download the latest TimerTool UnityPackage from Downloads, or from the Releases page of the GitHub repository.
2. Open your Unity project.
3. Go to "Assets" in the top menu and select "Import Package" > "Custom Package..."
4. Locate the downloaded TimerTool UnityPackage and select it.
5. The UnityPackage will be imported into your project, and you can start using TimerTool immediately.

Method 2: Source Code Project

1. Clone or download the TimerTool GitHub repository to your local machine from here.
2. Open Unity and create a new project or use an existing one.
3. Drag and drop the TimerTool source code files and folders into the "Assets" folder of your Unity project.
4. Unity will import the TimerTool scripts, and you can now start using TimerTool in your project.

Basic Setup

Before using the Timer Tool, make sure you have set up your Unity project and have a basic understanding of Unity scripting. The Timer Tool operates independently, so you can use it in any GameObject or script within your project.

To begin using the Timer Tool, follow these steps:

1. Add the necessary "using" statement at the top of your script:

```
using TimerTool;
```

2. Create a new instance of the Timer class and pass in the required parameters:

```
// Example: Creating a new Timer that lasts for 10 seconds, loops, and uses real-time.
Timer myTimer = new Timer(10f, true, true, null, null, new List<TimerEvent>());
```

3. Update the Timer each frame to keep it active and execute timer events:

```
// Example: Call this in the Update method of your MonoBehaviour script.
void Update()
{
    myTimer.Update();
}
```

Timer Functionality

Creating a Timer

To create a new Timer, use the Timer constructor and provide the required parameters:

```
public Timer(float duration, bool isLooped, bool usesRealTime, UnityEvent onFinish, UnityEvent<
```

- *duration*: The total duration (in seconds) of the Timer.
- *isLooped*: A flag indicating whether the Timer should restart after completing its duration.
- *usesRealTime*: A flag indicating whether the Timer uses real-time or game-time.
- *onFinish*: A UnityEvent invoked when the Timer completes its duration.
- *onUpdate*: A UnityEvent invoked each frame with updated TimerData during the Timer's execution.
- *timerEvents*: A list of TimerEvents representing events to be triggered during the Timer's execution.

Pausing and Resuming the Timer

You can pause and resume the Timer as needed:

```
// Pause the Timer
myTimer.Pause();

// Resume the Timer
myTimer.Resume();
```

Stopping the Timer

To stop the Timer's execution:

```
myTimer.Stop();
```

Stopping the Timer will prevent any further updates and event triggering.

Looping the Timer

By setting the *isLooped* flag to true when creating the Timer, it will automatically restart after completing its duration:

```
// Create a Timer that loops and lasts for 5 seconds
Timer loopingTimer = new Timer(5f, true, true, null, null, new List<TimerEvent>());
```

Time Modes (Real-Time vs. Game-Time)

The Timer can operate in two modes: Real-Time and Game-Time.

- Real-Time: The Timer is unaffected by changes to the game's timescale (e.g., pausing, slow-mo).
- Game-Time: The Timer adjusts its progress based on changes to the game's timescale.

You can specify the desired mode when creating the Timer:

```
// Create a Timer using real-time
Timer realTimeTimer = new Timer(10f, false, true, null, null, new List<TimerEvent>());

// Create a Timer using game-time
Timer gameTimeTimer = new Timer(10f, false, false, null, null, new List<TimerEvent>());
```

Timer Events

Adding Timer Events

Timer Events are actions triggered at specific times during the Timer's execution. To add Timer Events to your Timer, use the *AddEvent* or *AddEvents* methods:

```
// Create a Timer
Timer myTimer = new Timer(20f, false, true, null, null, new List<TimerEvent>());

// Create a TimerEvent that triggers at 5 seconds and calls the specified UnityEvent
TimerEvent myTimerEvent = new TimerEvent { KeyTime = 5f, KeyEvent = new UnityEvent<TimerData>()

// Add the TimerEvent to the Timer
myTimer.AddEvent(myTimerEvent);
```

Resetting Timer Events

Timer Events can be reset to allow them to be triggered again in future Timer executions:

```
// Reset all Timer Events associated with the Timer
foreach (TimerEvent timerEvent in myTimer.Events)
{
    timerEvent.Reset();
}
```