Game World Time Management Asset

By Andy’s Asset

Table of Contents

Overview

The **Game World Time Management** package contains scripts/C# classes to manage converting real world time to game world time.

Let’s say the game requires an in-game day to last 10 real world minutes, this package will handle that conversion and keep track of in-game time such as in-game days/hours/minutes/seconds.

Package Contents

While this package is quite small, each important piece will be described below.

Scripts

GameWorldTime.cs

This is where all of the logic of managing the game world time resides. This is a Non-MonoBehaviour class that can be easily created from anywhere. Since this class is not a MonoBehaviour, the developer will have to call **AdvanceTime()** as this class does not include an internal timer.

GameWorldTimeBehaviour.cs

This is a MonoBehaviour that contains a private reference to GameWorldTime. This script does not hold any logic, but still extends from the IGameWorldTime interface to ensure all public methods/properties are correctly added.

The main reason for this script is to give the developer the option to add this script to any GameObject and avoid having to manually update the GameWorldTime object. The other reason for this script is to pass Time.deltaTime into **AdvanceTime()**.

The developer does not *need* to use this script, and could instead create an instance of GameWorldTime inside of another script.

IGameWorldTime.cs

Interface for GameWorldTime that can be used for substituting in unit tests and other regular interface things.

RealWorldTimeConstants.cs

This static class contains constants that are related to real world time values such as how many seconds there are in an hour, etc.. You as the developer could update/change these values if your game requires different time lengths. These values are used mainly inside of GameWorldTime.cs to convert in-game time from seconds to minutes using real world time conversions.

Installation Instructions

This package is installed just like any of the other Unity Asset Packages. You can follow the Unity Docs for installing a package from the Unity Asset Store.

Requirements

Since this is a very simple package, and doesn’t use many references from UnityEngine, this package should work on most Unity versions.

But since I am not a future teller, this package was made and tested on Unity 2022.3.20f1 in August of 2024.

Limitations

Going Back in Time

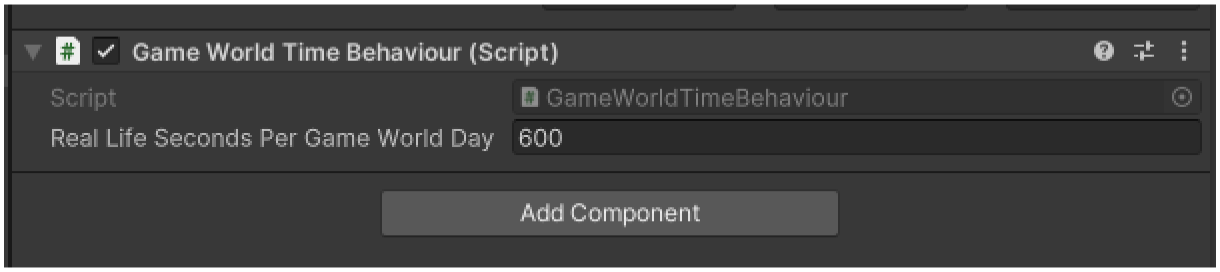
These scripts were only made to advance time forwards, these scripts do not handle going backwards in time and could cause issues if tried. There are some security statements in place to catch passing negative values for setting the current time and advancing the time. If the developer would like to go back in time, it would be wise to set the time to the exact moment you’d like to go back.

Workflows / Tutorials

Since this package is very small, using it in your Unity project is very simple. I will split up using this package using the MonoBehaviour script and using it without the MonoBehaviour script.

MonoBehaviour

Using this package through the MonoBehaviour script is as simple as adding the GameWorldTimeBehaviour.cs script onto any GameObject. Since this script is not a Singleton, it can be added onto multiple GameObjects, and even multiple scripts onto the same GameObject.



Game World Time Behaviour (Script)

Script: GameWorldTimeBehaviour

Real Life Seconds Per Game World Day: 600

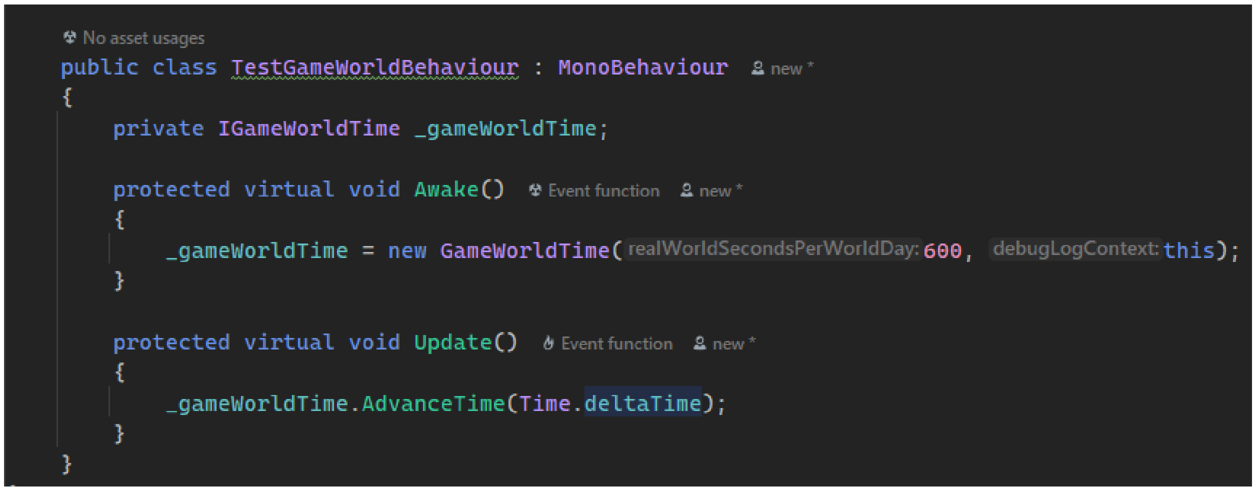
Here is a screenshot of the GameWorldTimeBehaviour script added onto a GameObject. As you can see, there is only one Serialized Field to be set. There is a tooltip assigned to this field to help describe what it means. In this specific case it would take 600 real life seconds (10 minutes) for 1 in-game day.

Lets say, you as the developer would like to make a UI to display the current Game World Time, your UI could take in a reference for GameWorldTimeBehaviour to help decouple your game world time and the UI.

Non-MonoBehaviour

This is for those developers that don’t always love to have a million components on their GameObjects, and like to easily unit test their objects by having C# instances rather than MonoBehaviours.

A good example on how to use this can be found inside of the MonoBehaviour instances provided. The user just has to create a new instance of GameWorldTime.



```
No asset usages
public class TestGameWorldBehaviour : MonoBehaviour  new *
{
    private IGameWorldTime _gameWorldTime;

    protected virtual void Awake()  Event function  new *
    {
        _gameWorldTime = new GameWorldTime( realWorldSecondsPerWorldDay: 600,  debugLogContext: this);
    }

    protected virtual void Update()  Event function  new *
    {
        _gameWorldTime.AdvanceTime(Time.deltaTime);
    }
}
```

In this screenshot, it shows how to use the Non-MonoBehaviour version of GameWorldTime. As you can see I have a local reference to **IGameWorldTime** called **_gameWorldTime**. Inside of the **Awake** method, I am creating a new instance of this object using the **new GameWorldTime(x, y)** initializer. The first parameter describes how many real world seconds it takes for one in-game day to pass. The second parameter is optional and is mainly used for debugging, it is a reference to the Unity Component so debugging through the editor is easier. When a warning shows in the Unity Logger, double clicking the warning should highlight/select the object that was passed as this value. Once again, it is an optional parameter and not necessary.