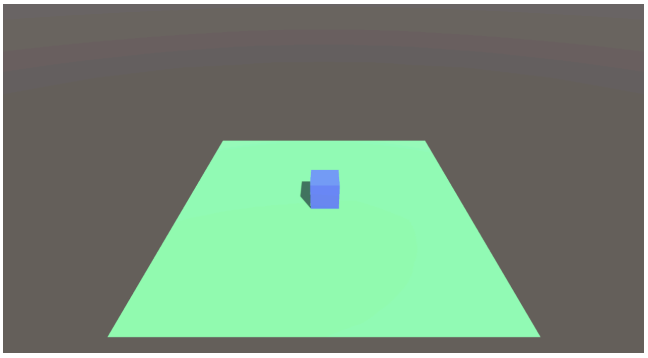
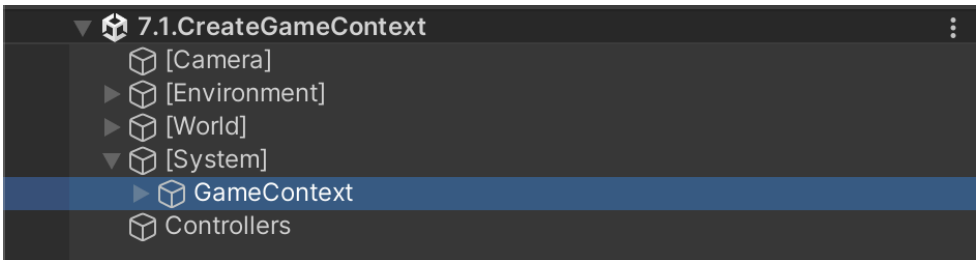
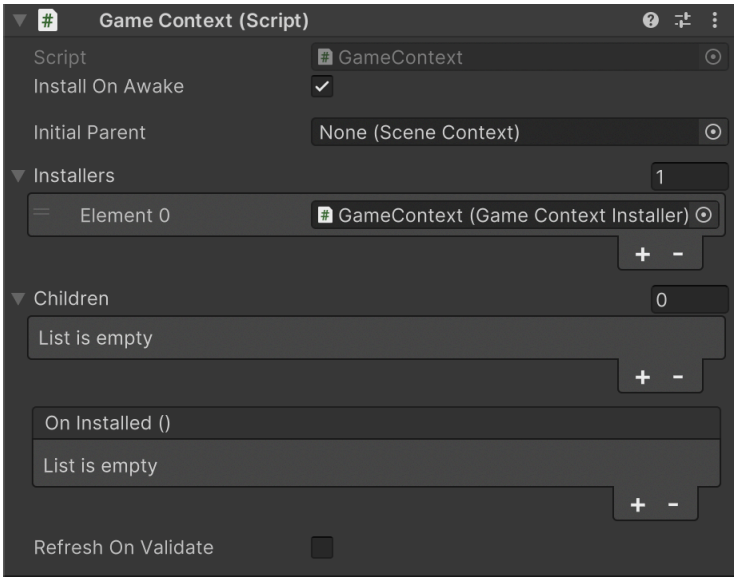
In the second section, “Context", we will explore how you can create game systems using the Atomic Framework.



First of all, let’s look at inspector of the GameContext game object



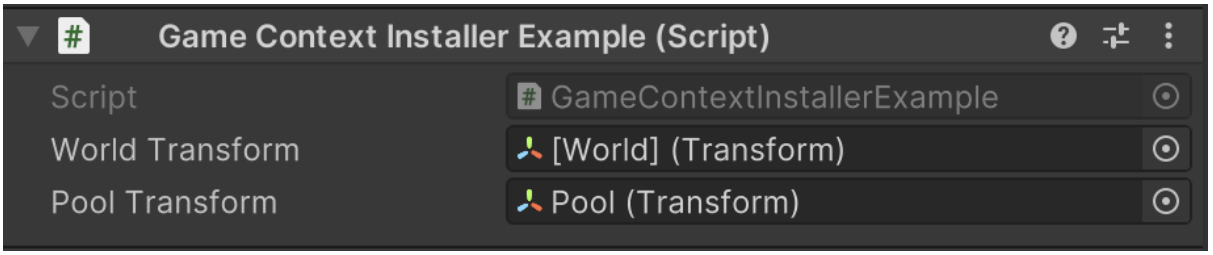
The first component is the GameContext!



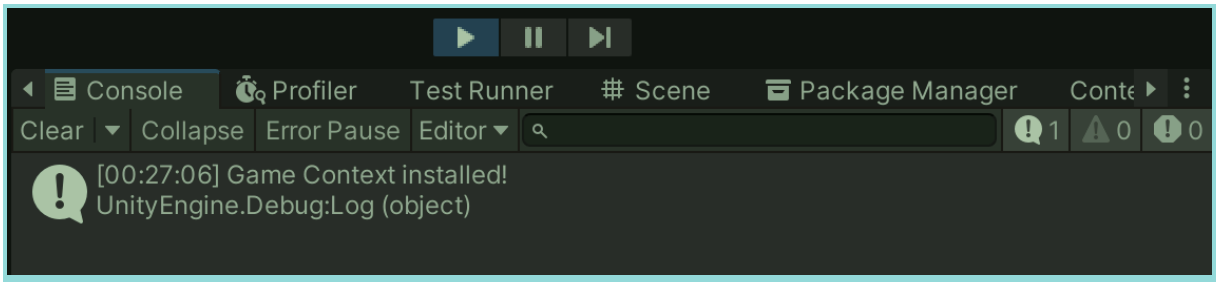
The responsibility of the GameContext component is to store the state and behavior of the game system. For a context instance to be accessed from anywhere in the program, it must be made a singleton.

```
using Atomic.Contexts;

namespace GameExample.Engine
{
    public sealed class GameContext : SingletonSceneContext<GameContext>
    {
    }
}
```

To register data and logic in the context of the game, you need to write a Game Context Installer (similar to Entity)




```
using Atomic.Contexts;
using UnityEngine;

namespace Walkthrough
{
   public sealed class GameContextInstallerExample : SceneContextInstallerBase
   {
       [SerializeField]
       private Transform worldTransform;

       [SerializeField]
       private Transform poolTransform;

       public override void Install(IContext context)
       {
           //Setup values:
           context.AddValue(GameContextAPI.WORLD_TRANSFORM, this.worldTransform);
           context.AddValue(GameContextAPI.POOL_TRANSFORM, this.poolTransform);
           context.AddValue(GameContextAPI.GAME_COUNTDOWN, 30);
           context.AddValue(GameContextAPI.SPAWN_AREA, new Bounds(Vector3.zero, new
Vector3(5, 0, 5)));

           //Setup behaviours:
           context.AddSystem<ContextSystemExample>();

           Debug.Log("Game Context installed!");
       }
   }
}
```



Next, I would like to demonstrate what capabilities a developer has to describe behaviours for game systems. To do this, I wrote an Context System Example that demonstrates the entire interface of interaction with an context:

```
using Atomic.Contexts;
using UnityEngine;

namespace Walkthrough
{
   public sealed class ContextSystemExample :
        IContextInit,
        IContextEnable,
        IContextDisable,
        IContextDispose,
        IContextUpdate,
        IContextFixedUpdate,
        IContextLateUpdate
   {
        public void Init(IContext context)
        {
            Debug.Log($"Init context {context.Name}");

            Transform worldTransform =
context.GetValue<Transform>(GameContextAPI.WORLD_TRANSFORM);
            Debug.Log($"World transform: {worldTransform.name}");

            Transform poolTransform =
context.GetValue<Transform>(GameContextAPI.POOL_TRANSFORM);
            Debug.Log($"Pool transform: {poolTransform.name}");

            int gameCountdown =
context.GetValue<int>(GameContextAPI.GAME_COUNTDOWN);
            Debug.Log($"Game countdown: {gameCountdown}");

            Bounds spawnArea = context.GetValue<Bounds>(GameContextAPI.SPAWN_AREA);
            Debug.Log($"Spawn area: {spawnArea}");
        }

        //Calls like MonoBehaviour.Enable()
        public void Enable(IContext context)
        {
            Debug.Log($"Enable context: {context.Name}");
        }

        //Calls like MonoBehaviour.Disable()
        public void Disable(IContext context)
        {
            Debug.Log($"Disable context: {context.Name}");
        }

        //Calls like MonoBehaviour.OnDestroy()
        public void Dispose(IContext context)
        {
            Debug.Log($"Dispose context: {context.Name}");
        }

        public void Update(IContext context, float deltaTime)
        {
```

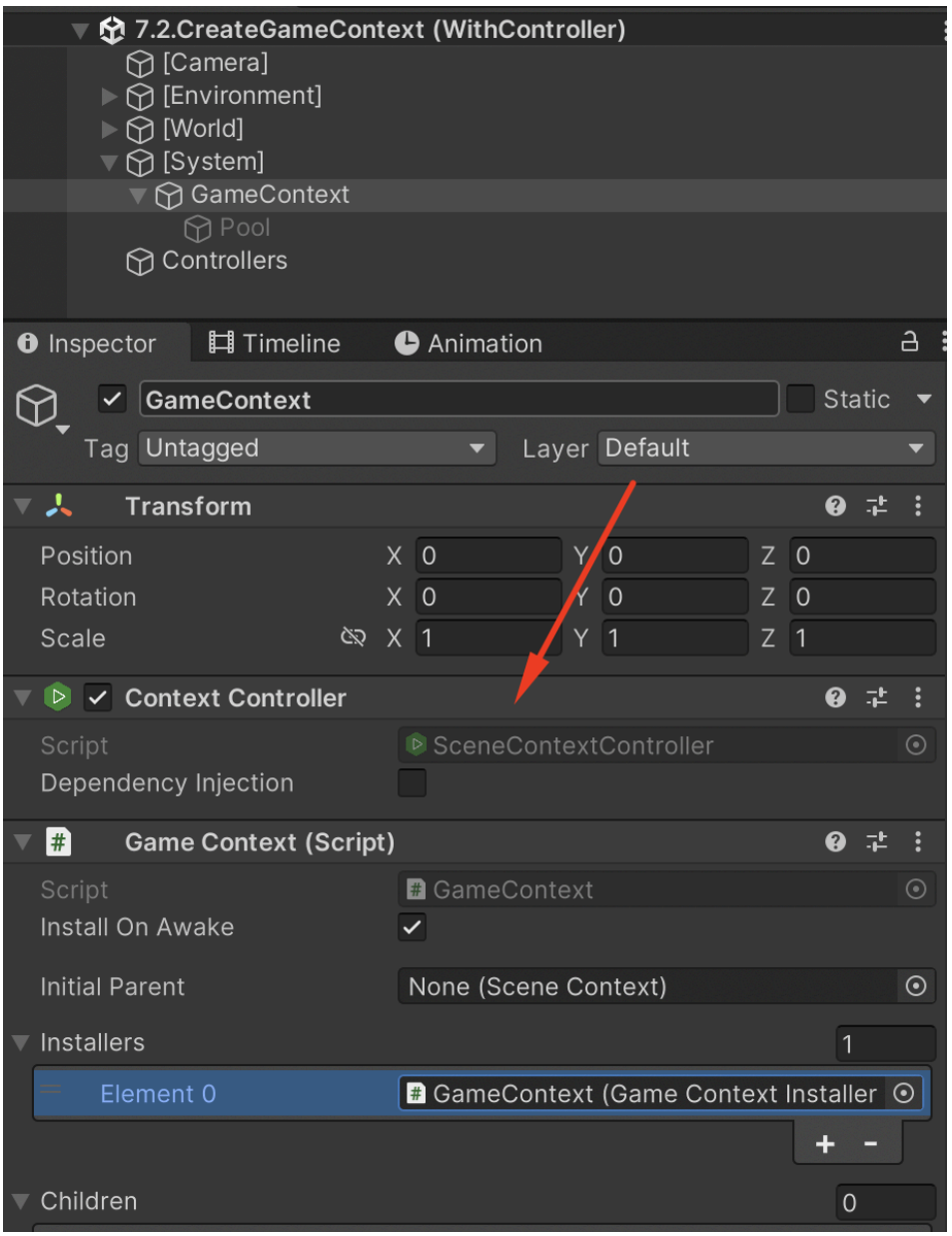
```
            Debug.Log($"Update context: {context.Name}");
        }

        public void FixedUpdate(IContext context, float deltaTime)
        {
            Debug.Log($"Fixed update context: {context.Name}");
        }

        public void LateUpdate(IContext context, float deltaTime)
        {
            Debug.Log($"Late update context: {context.Name}");
        }
    }
}
```

The key feature of the Context System is that it receives Unity events, but it is not a MonoBehaviour itself. Since system and context are closely related to each other, the developer always has the opportunity to access the data of the context and work on them.

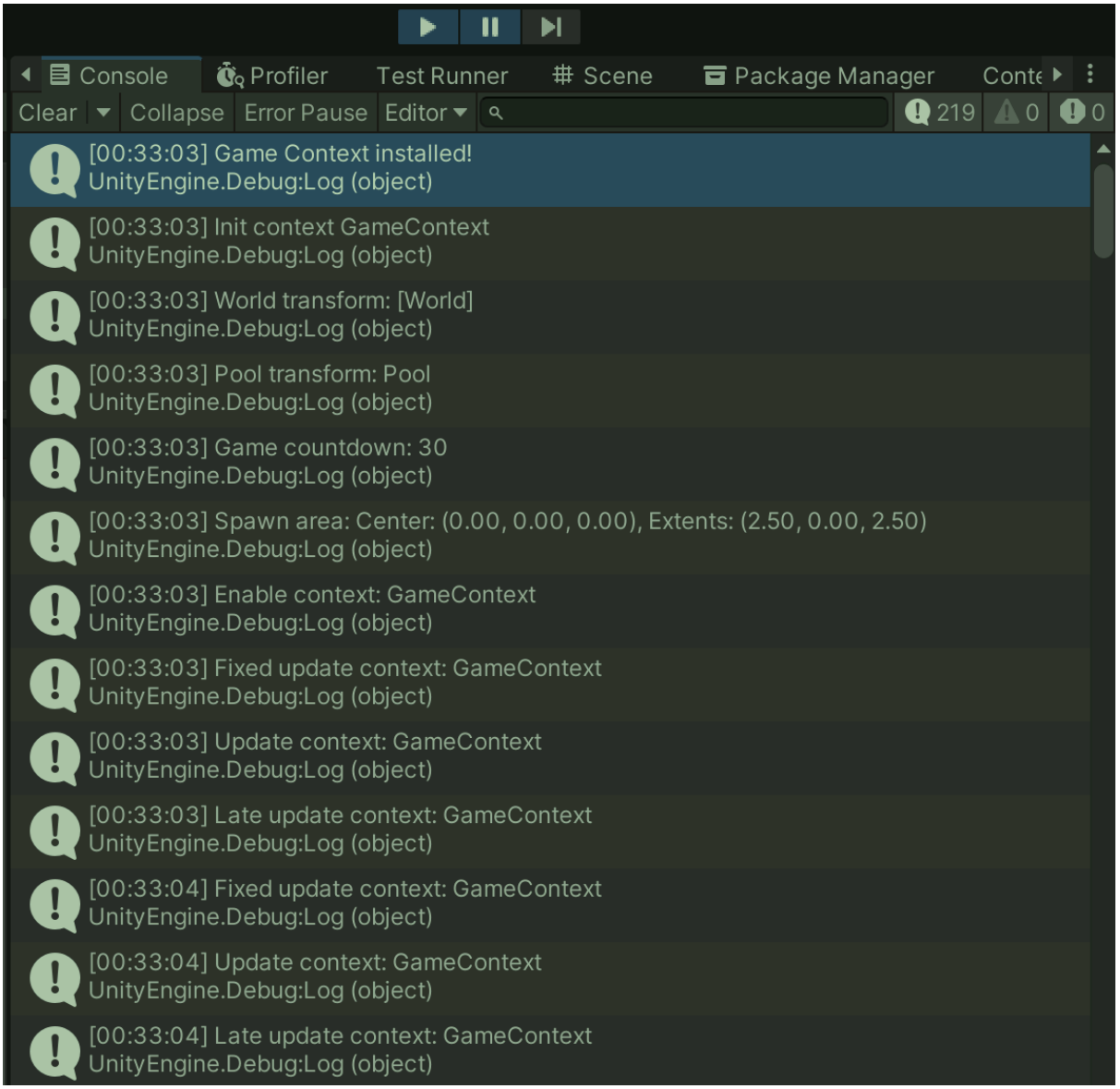
Recommendation: For a more supported architecture, I recommend not creating internal states inside the system, but rather making them shared by placing data in a container with Context values.

If you run the **7.1 Create GameContext** scene then we will see that Unity calls will not be displayed in the console

To fix this, you need to add the ContextController component, which will receive Unity callbacks, delegating them to Context.



This example can be viewed in scene **7.2 Create GameContext**


Console
Profiler
Test Runner
Scene
Package Manager
Clear
Collapse
Error Pause
Editor
219
0
0
[00:33:03] Game Context installed!
UnityEngine.Debug:Log (object)
[00:33:03] Init context GameContext
UnityEngine.Debug:Log (object)
[00:33:03] World transform: [World]
UnityEngine.Debug:Log (object)
[00:33:03] Pool transform: Pool
UnityEngine.Debug:Log (object)
[00:33:03] Game countdown: 30
UnityEngine.Debug:Log (object)
[00:33:03] Spawn area: Center: (0.00, 0.00, 0.00), Extents: (2.50, 0.00, 2.50)
UnityEngine.Debug:Log (object)
[00:33:03] Enable context: GameContext
UnityEngine.Debug:Log (object)
[00:33:03] Fixed update context: GameContext
UnityEngine.Debug:Log (object)
[00:33:03] Update context: GameContext
UnityEngine.Debug:Log (object)
[00:33:03] Late update context: GameContext
UnityEngine.Debug:Log (object)
[00:33:04] Fixed update context: GameContext
UnityEngine.Debug:Log (object)
[00:33:04] Update context: GameContext
UnityEngine.Debug:Log (object)
[00:33:04] Late update context: GameContext
UnityEngine.Debug:Log (object)