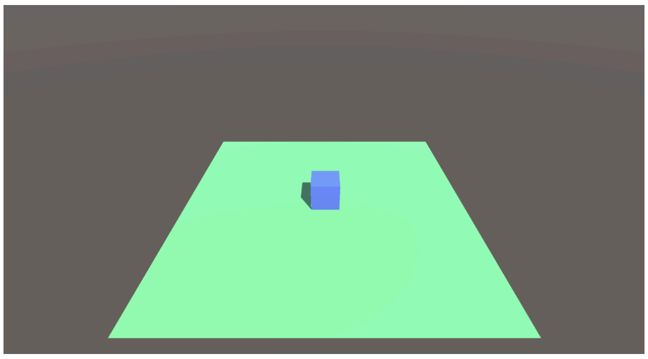
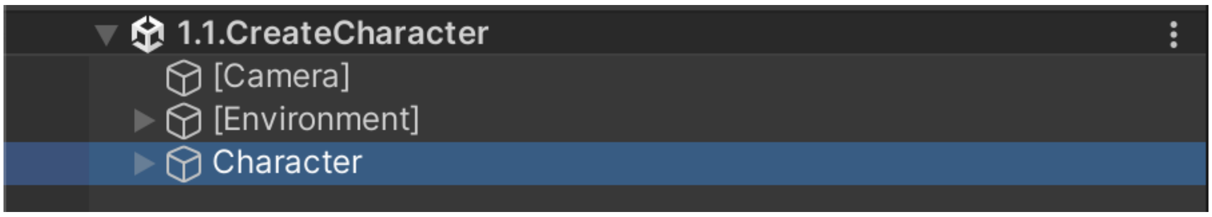
In the first section, “Entities", we will explore how you can create game objects using the Atomic Framework.

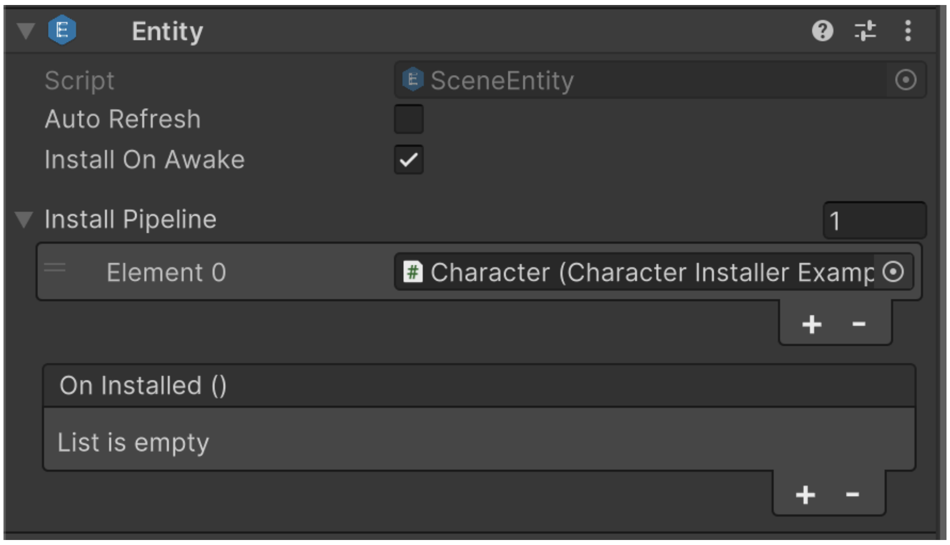


First of all, let’s look at inspector of the Character game object



1.1.CreateCharacter
[Camera]
[Environment]
Character

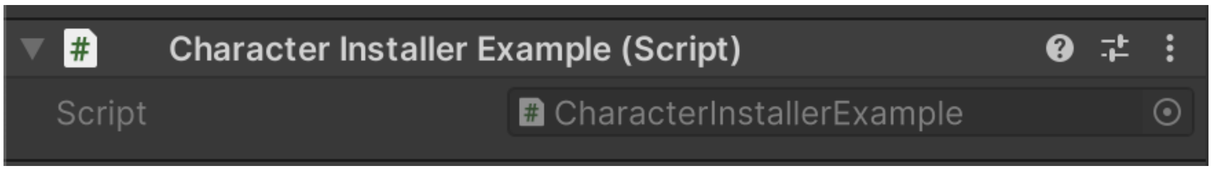
The first component is the Entity!



Entity
Script: SceneEntity
Auto Refresh: ☐
Install On Awake: ✓
Install Pipeline: 1
Element 0: Character (Character Installer Examp
On Installed ()
List is empty

The responsibility of the Entity component is to store the state and behavior of the Character game object. In order for this data and logic to get there, you need to write the Entity Installer component, which will configure the Entity component.

I wrote an example of the Entity Installer below



Character Installer Example (Script)
Script: CharacterInstallerExample

```
using Atomic.Entities;
using UnityEngine;

namespace Walkthrough
{
   public sealed class CharacterInstallerExample : SceneEntityInstallerBase
   {
       public override void Install(IEntity entity)
       {
           //Setup values:
           entity.AddValue(EntityAPI.GAME_OBJECT_KEY, this.gameObject);
           entity.AddValue(EntityAPI.TRANSFORM_KEY, this.transform);
           entity.AddValue(EntityAPI.HEALTH_KEY, 3);
           entity.AddValue(EntityAPI.SPEED_KEY, 4);

           //Setup tags:
           entity.AddTag(EntityAPI.CHARACTER_TAG);
           entity.AddTag(EntityAPI.MOVEABLE_TAG);

           //Setup behaviours:
           entity.AddBehaviour<EntityBehaviourExample>();

           Debug.Log("Character installed!");
       }
   }
}
```

Next, I would like to demonstrate what capabilities a developer has to describe behaviors for game objects. To do this, I wrote an Entity Behavior Example that demonstrates the entire interface of interaction with an entity:

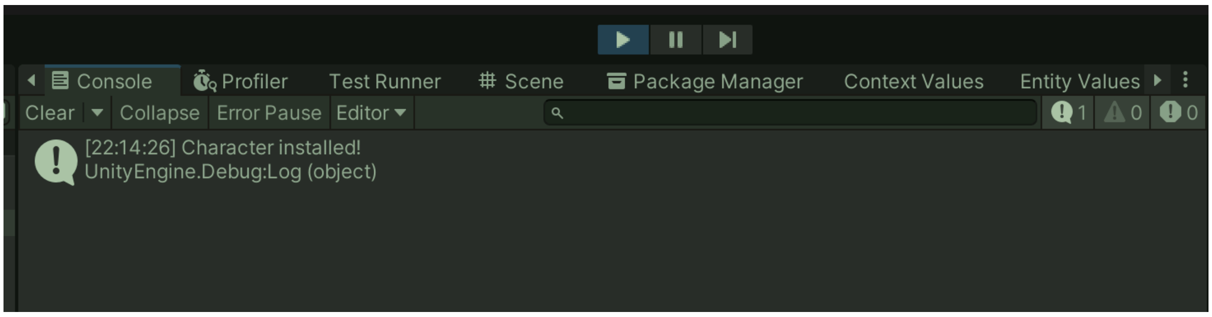
```
using Atomic.Entities;
using UnityEngine;

namespace Walkthrough
{
   public sealed class EntityBehaviourExample :
       IEntityInit,
       IEntityEnable,
       IEntityDisable,
       IEntityDispose,
       IEntityUpdate,
       IEntityFixedUpdate,
       IEntityLateUpdate
   {

       //Calls like MonoBehaviour.Start()
       public void Init(IEntity entity)
       {
           Debug.Log($"Init {entity.Name}");

           GameObject gameObject =
entity.GetValue<GameObject>(EntityAPI.GAME_OBJECT_KEY);
           Debug.Log($"GameObject active: {gameObject.activeSelf}");

           Transform transform =
entity.GetValue<Transform>(EntityAPI.TRANSFORM_KEY);
           Debug.Log($"Position: {transform.position}");
```

```
            int health = entity.GetValue<int>(EntityAPI.HEALTH_KEY);
            Debug.Log($"Health: {health}");

            float speed = entity.GetValue<int>(EntityAPI.SPEED_KEY);
            Debug.Log($"Speed: {speed}");

            Debug.Log($"Is Character {entity.HasTag(EntityAPI.CHARACTER_TAG)}");
            Debug.Log($"Is Moveable {entity.HasTag(EntityAPI.MOVEABLE_TAG)}");
            Debug.Log($"Is Coin {entity.HasTag(EntityAPI.COIN_TAG)}");
        }

        //Calls like MonoBehaviour.Enable()
        public void Enable(IEntity entity)
        {
            Debug.Log($"Enable {entity.Name}");
        }

        //Calls like MonoBehaviour.Disable()
        public void Disable(IEntity entity)
        {
            Debug.Log($"Disable {entity.Name}");
        }

        //Calls like MonoBehaviour.OnDestroy()
        public void Dispose(IEntity entity)
        {
            Debug.Log($"Dispose {entity.Name}");
        }

        //Calls like MonoBehaviour.Update()
        public void OnUpdate(IEntity entity, float deltaTime)
        {
            Debug.Log($"Update {entity.Name}");
        }

        //Calls like MonoBehaviour.FixedUpdate()
        public void OnFixedUpdate(IEntity entity, float deltaTime)
        {
            Debug.Log($"Fixed Update {entity.Name}");
        }

        //Calls like MonoBehaviour.LateUpdate()
        public void OnLateUpdate(IEntity entity, float deltaTime)
        {
            Debug.Log($"Late Update {entity.Name}");
        }
    }
}
```

The key feature of the Entity Behaviour is that receives Unity events, but it is not a MonoBehaviour itself. Since behavior and essence are closely related to each other, the developer always has the opportunity to access the data and tags of the game object and work on them.

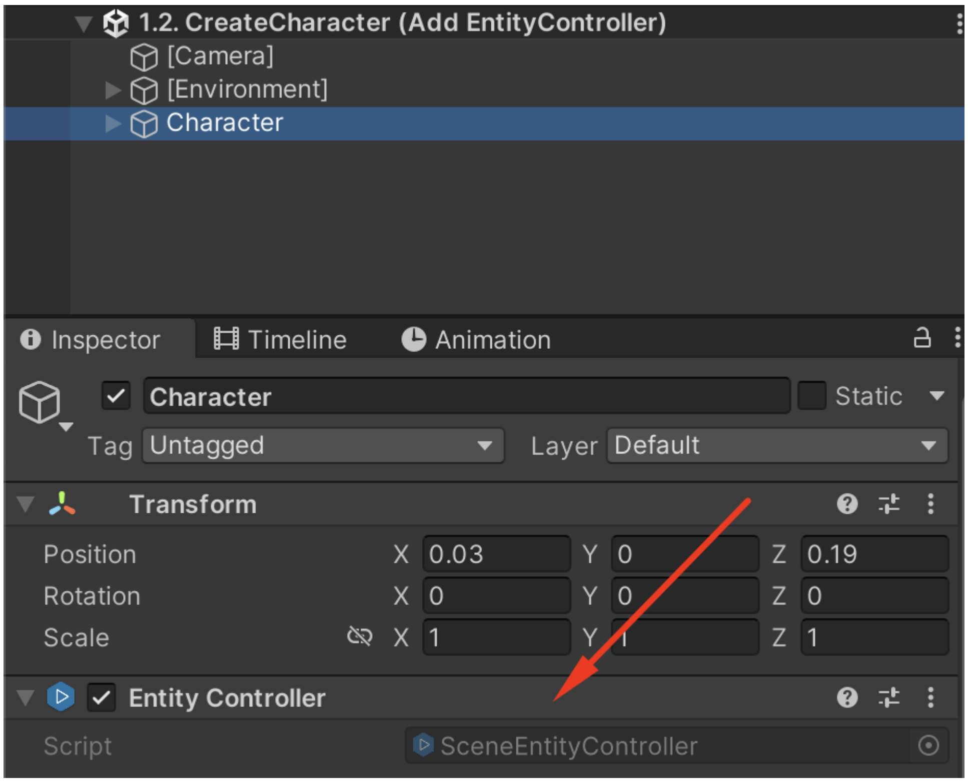
Recommendation: For a more supported architecture, I recommend not creating internal states inside the behavior, but rather making them shared by placing data in a container with Entity values.

If you run the **1.1 Create Character** scene then we will see that Unity calls will not be displayed in the console



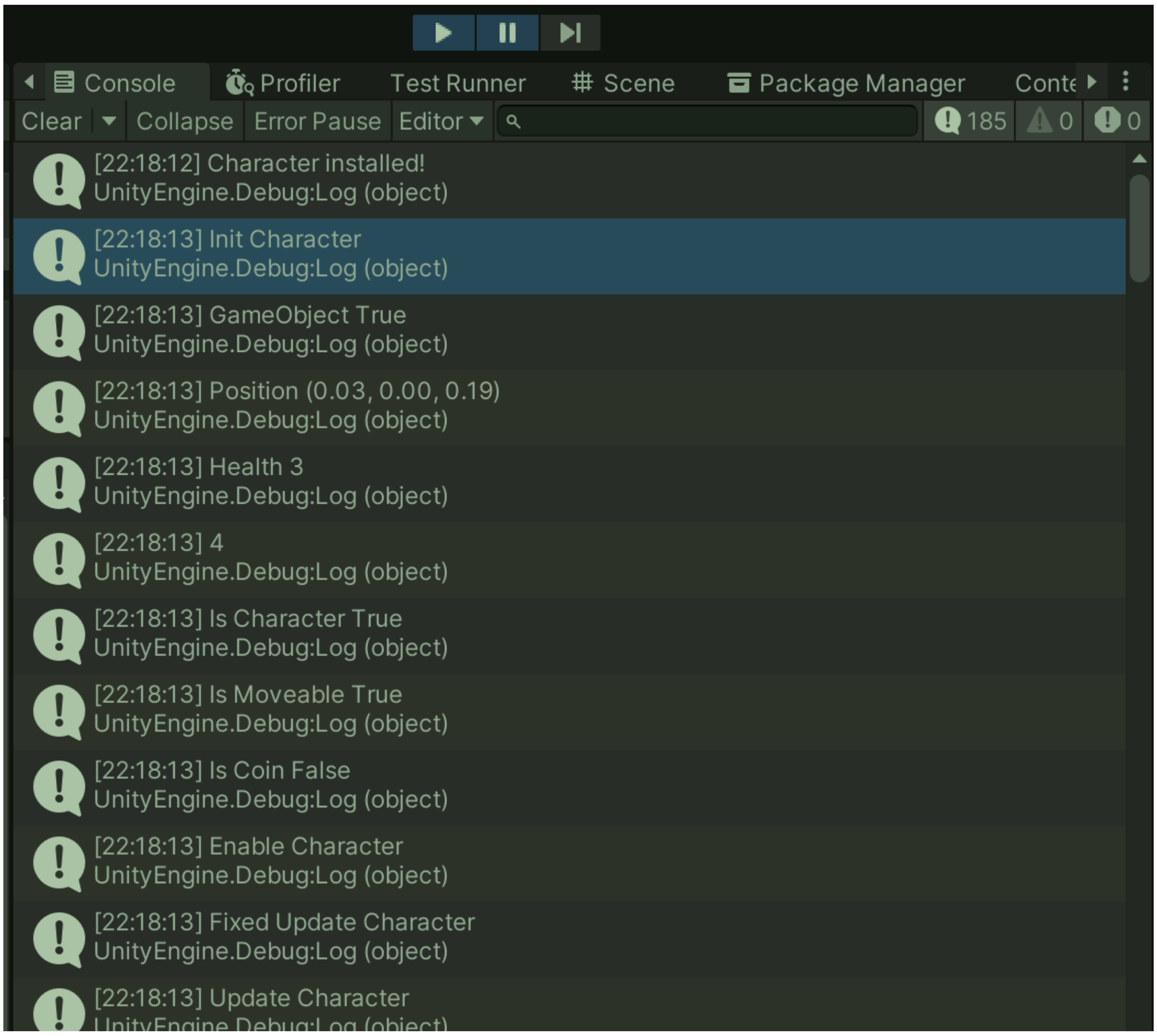
```
[22:14:26] Character installed!
UnityEngine.Debug:Log (object)
```

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



Character
Tag Untagged
Layer Default
Position X 0.03 Y 0 Z 0.19
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Script SceneEntityController

This example can be viewed in scene **1.2 Create Character**



```
[22:18:12] Character installed!
UnityEngine.Debug:Log (object)
[22:18:13] Init Character
UnityEngine.Debug:Log (object)
[22:18:13] GameObject True
UnityEngine.Debug:Log (object)
[22:18:13] Position (0.03, 0.00, 0.19)
UnityEngine.Debug:Log (object)
[22:18:13] Health 3
UnityEngine.Debug:Log (object)
[22:18:13] 4
UnityEngine.Debug:Log (object)
[22:18:13] Is Character True
UnityEngine.Debug:Log (object)
[22:18:13] Is Moveable True
UnityEngine.Debug:Log (object)
[22:18:13] Is Coin False
UnityEngine.Debug:Log (object)
[22:18:13] Enable Character
UnityEngine.Debug:Log (object)
[22:18:13] Fixed Update Character
UnityEngine.Debug:Log (object)
[22:18:13] Update Character
UnityEngine.Debug:Log (object)
```