The character game object will have the following mechanics:

1. Movement behaviour
2. Rotation behaviour
3. The mechanics of synchronizing position and rotation with Transform
4. Trigger event receiver

Movement Behaviour

```
using Atomic.Elements;
using Atomic.Entities;
using Unity.Mathematics;

namespace GameExample.Engine
{
   public sealed class MovementBehaviour : IEntityInit, IEntityFixedUpdate
   {
       private IVariable<float3> _position;
       private IValue<float> _moveSpeed;
       private IValue<float3> _moveDirection;

       //Calls like a MonoBehaviour.Start()
       public void Init(IEntity entity)
       {
           _position = entity.GetPosition();
           _moveSpeed = entity.GetMoveSpeed();
           _moveDirection = entity.GetMoveDirection();
       }

        //Calls like a MonoBehaviour. FixedUpdate()
       public void OnFixedUpdate(IEntity entity, float deltaTime)
       {
           _position.Value += _moveDirection.Value * _moveSpeed.Value * deltaTime;
       }
   }
}
```

Rotation Behaviour

```
using Atomic.Elements;
using Atomic.Entities;
using Unity.Mathematics;

namespace GameExample.Engine
{
   public sealed class RotationBehaviour : IEntityInit, IEntityFixedUpdate
   {
       private IVariable<quaternion> _rotation;
       private IValue<float> _angularSpeed;
       private IValue<float3> _moveDirection;

       public void Init(IEntity entity)
       {
           _rotation = entity.GetRotation();
           _angularSpeed = entity.GetAngularSpeed();
           _moveDirection = entity.GetMoveDirection();
       }

       public void OnFixedUpdate(IEntity entity, float deltaTime)
       {
           float3 upAxis = new float3(0, 1, 0);
           quaternion targetRotation = quaternion.LookRotation(
              _moveDirection.Value, upAxis
           );

           float t = speed * deltaTime;
           _rotation.Value = math.slerp(rotation, targetRotation, t);
       }
   }
}
```

Sync position and rotation with Transform

```
using Atomic.Elements;
using Atomic.Entities;
using Unity.Mathematics;
using UnityEngine;

namespace GameExample.Engine
{
    public sealed class TransformBehaviour : IEntityInit, IEntityUpdate
    {
        private Transform _transform;
        private IReactiveValue<float3> _position;
        private IReactiveValue<quaternion> _rotation;

        public void Init(IEntity entity)
        {
            _transform = entity.GetTransform();
            _position = entity.GetPosition();
            _rotation = entity.GetRotation();

            _transform.SetPositionAndRotation(_position.Value, _rotation.Value);
        }

        public void OnUpdate(IEntity entity, float deltaTime)
        {
            _transform.SetPositionAndRotation(_position.Value, _rotation.Value);
        }
    }
}
```

Trigger Event Receiver

```
using UnityEngine;

namespace GameExample.Engine
{
    [DisallowMultipleComponent]
    public class TriggerEventReceiver : MonoBehaviour
    {
        public event System.Action<Collider> OnEntered;
        public event System.Action<Collider> OnExited;

        private void OnTriggerEnter(Collider collider)
        {
            this.OnEntered?.Invoke(collider);
        }

        private void OnTriggerExit(Collider collider)
        {
            this.OnExited?.Invoke(collider);
        }
    }
}
```

After the game mechanics are written, the developer can incorporate the necessary components into the character by writing a CharacterInstaller class

```
using Atomic.Entities;
using Atomic.Extensions;
using GameExample.Engine;
using Unity.Mathematics;
using UnityEngine;

namespace GameExample.Content
{
   public sealed class CharacterEntityInstaller : SceneEntityInstallerBase
   {
       [SerializeField]
       private float moveSpeed = 3;

       [SerializeField]
       private float angularSpeed = 12;

       [SerializeField]
       private float3 initialDirection;

       [SerializeField]
       private TriggerEventReceiver triggerEventReceiver;

       public override void Install(IEntity entity)
       {
           entity.AddCharacterTag();
           entity.AddGameObject(this.gameObject);
           entity.AddTransform(this.transform);

           entity.AddPosition(new float3Reactive(this.transform.position));
           entity.AddRotation(new quaternionReactive(this.transform.rotation));
           entity.AddBehaviour<TransformBehaviour>();

           entity.AddMoveSpeed(this.moveSpeed);
           entity.AddMoveDirection(this.initialDirection);
           entity.AddBehaviour<MovementBehaviour>();

           entity.AddAngularSpeed(this.angularSpeed);
           entity.AddBehaviour<RotationBehaviour>();

           entity.AddTriggerEventReceiver(this.triggerEventReceiver);
       }
   }
}
```