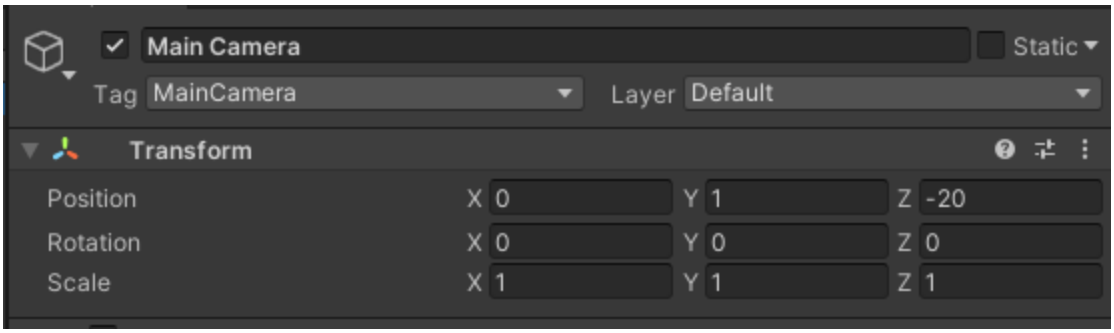
Part 2: Move & Shoot

Under the **AgentSoldier/Actions** GameObject

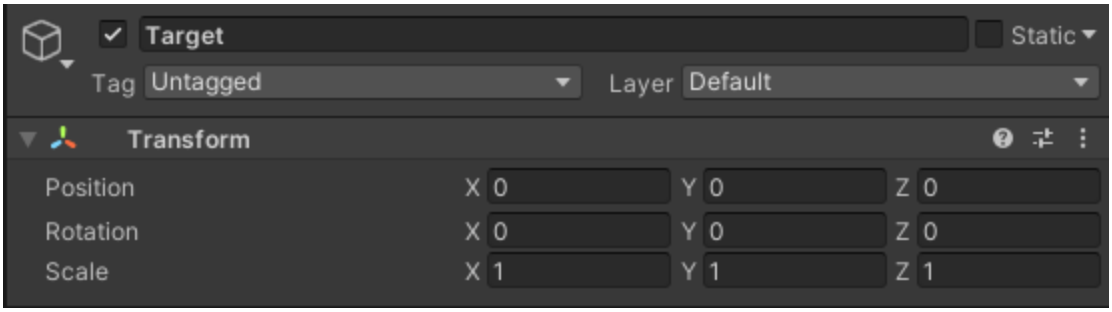
1. Add an empty GameObject and call it GoTo
2. Create a **GoToAction C# class** and **attach it on the GoTo Game Object.**

```
using SGoap;
using UnityEngine;

public class GoToAction : BasicAction
{
    public Transform Target;

    public float Range = 4;
    public float MoveSpeed = 2;

    public override EActionStatus Perform()
    {
        AgentData.Target = Target;

        var distanceToTarget = Vector3.Distance(Target.position, transform.position);
        if (distanceToTarget <= Range)
            return EActionStatus.Success;

        var directionToTarget = (Target.position - transform.position).normalized;
        AgentData.Position += directionToTarget * Time.deltaTime * MoveSpeed;

        // Returning Running will keep this action going until we return Success.
        return EActionStatus.Running;
    }
}
```

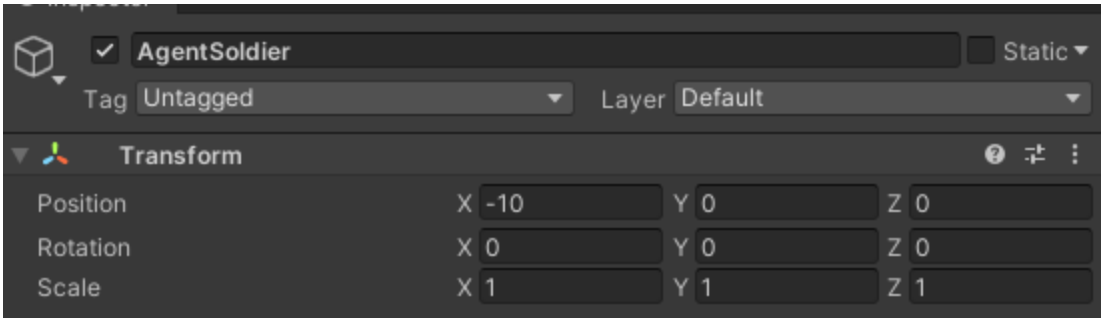
Set your **MainCamera** position to (0,0,-20)



Create a **Sphere** named **Target** and set its position at (0,0,0)



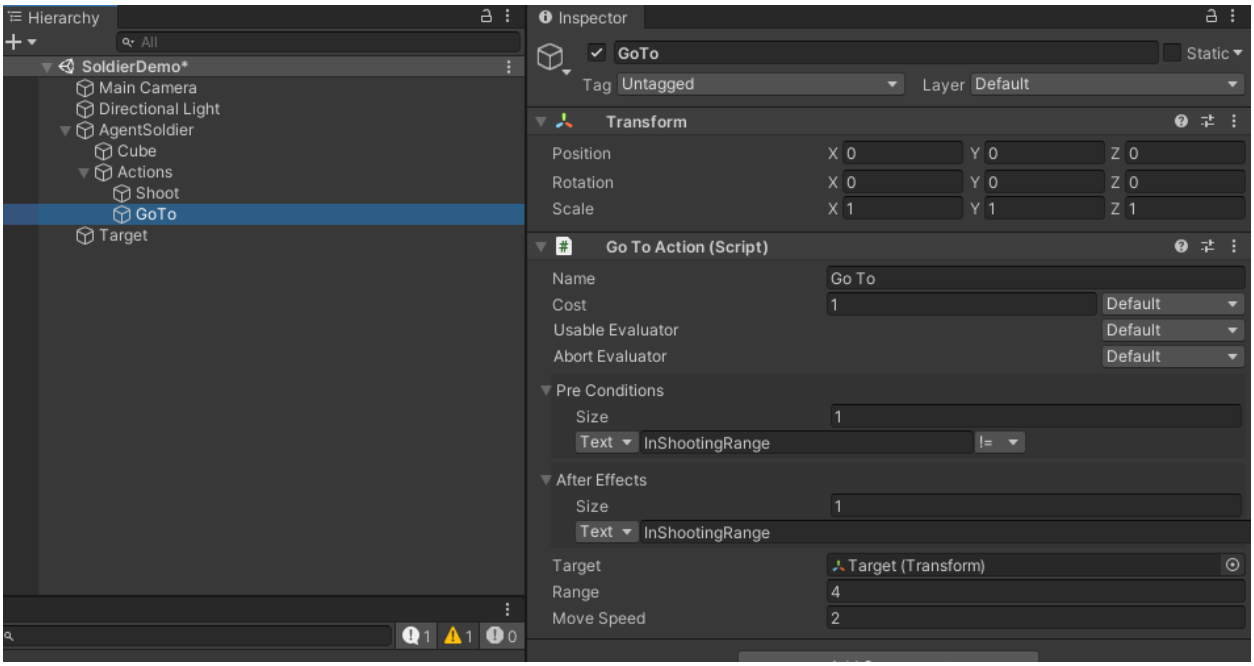
Set your **SoldierAgent** position to (-10,0,0)



Under the **AgentSoldier/Actions/Goto** GameObject

1. Attach the GoToAction script

2. (Optional) Set the name to Go To
3. Assign the **Target** to **Target (The sphere)**
4. Add a precondition **InShootingRange ! =** (This action can only start if the agent is not in shooting range) make sure you click the ! = Operator!
5. Add an effect **InShootingRange**

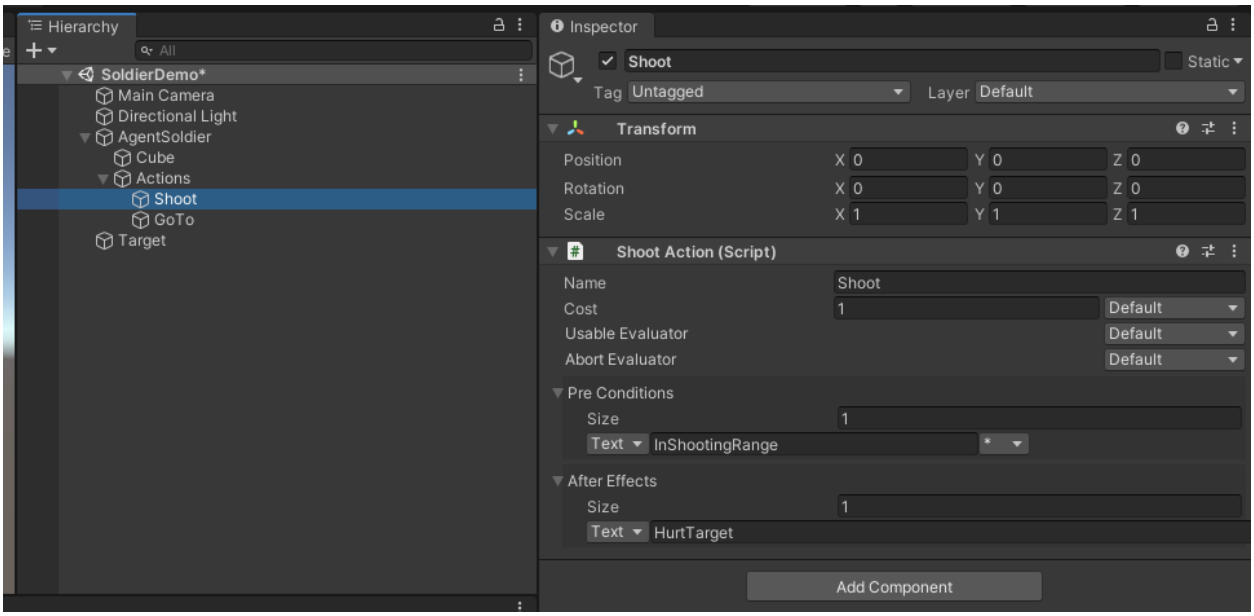


We're almost there!

> ❓ The screenshot used in the image above has a Target Property. Yours won't as the screenshot is outdated.

Under the **AgentSoldier/Actions/Shoot** GameObject

Add a precondition of **InShootingRange ***



Open the RealTime tab in the Goap Agent Debugger Window and press Play.

The Agent will move to the target and shoot.

> 💡 In the Agent Debugger RealTime panel, you may notice that the GoTo action precondition is still valid even though we made sure it will not run when the Agent is **InShootingRange.** This is because the Effects are not persistent and is only used by the planner.

To fix this issue, we have to add the **InShootingRange** when in range and remove it when it is out of range. Usually this will be driven by a Sensor or a System. For the tutorial's sake, we'll just update the action.

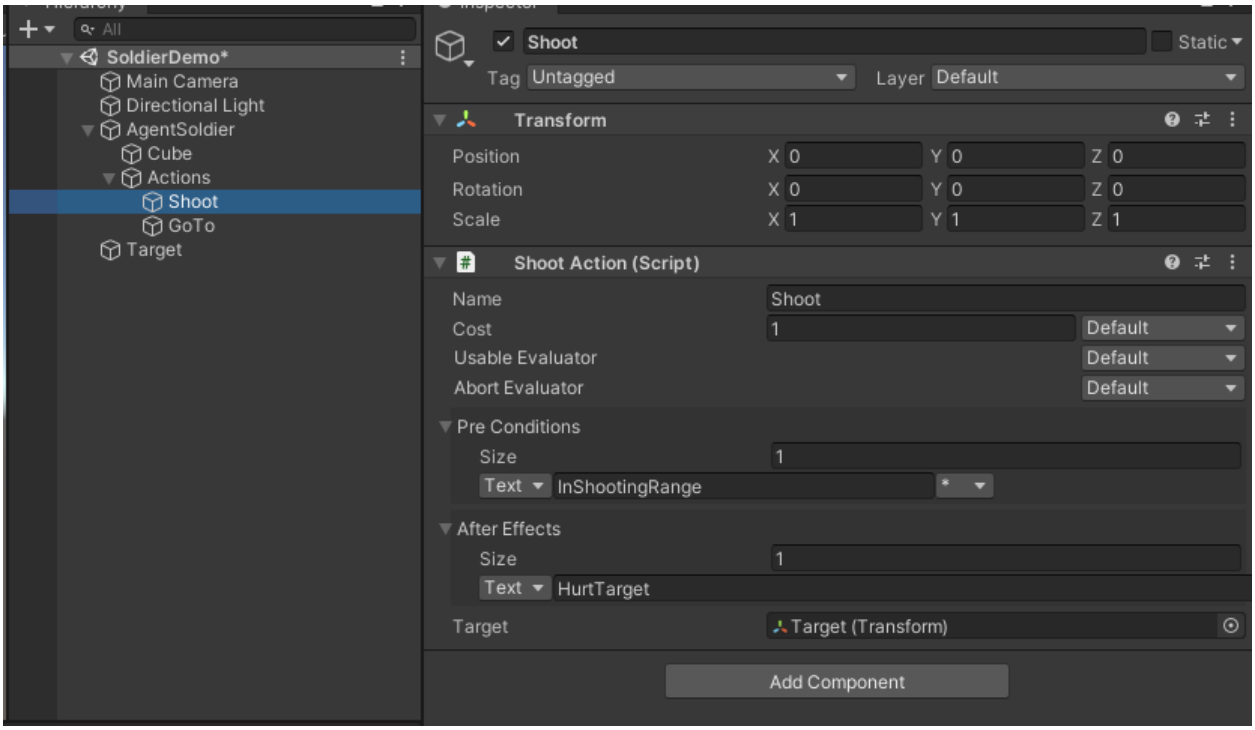
```
using SGoap;
using UnityEngine;

public class GoToAction : BasicAction
```

```
{
    ...

    public float DistanceToTarget => Vector3.Distance(Target.position, transform.position);

    private void Update()
    {
        // Add or Remove an Agent's state.
        if (DistanceToTarget <= Range)
            States.SetState("InShootingRange", 1);
        else
            States.RemoveState("InShootingRange");
    }

    public override EActionStatus Perform()...
}
```

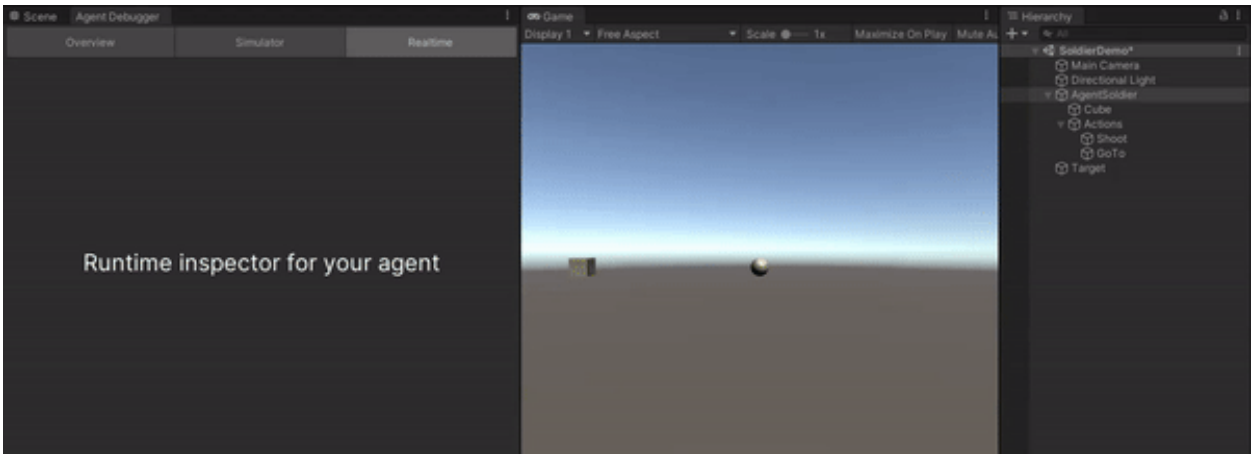
Finally, open the ShootAction.cs script and implement a terribly bad way of firing a bullet!

```
using SGoap;
using UnityEngine;

public class ShootAction : BasicAction
{
    public override float CooldownTime => 1;

    public override EActionStatus Perform()
    {
        var bullet = GameObject.CreatePrimitive(PrimitiveType.Sphere);
        bullet.transform.localScale = Vector3.one * 0.2f;
        bullet.transform.position = AgentData.Position;
        bullet.transform.GoTo(AgentData.Target.position, 1);

        Destroy(bullet.gameObject, 1.2f);

        return EActionStatus.Success;
    }
}
```

Assign your Target in the Inspector.



Press Play and you should see your agent shooting!



That's it, now you know how an agent can plan a set of actions and modify his state.

Practical Advice

Chaining a GoTo action to multiple actions can be a big hassle and changing the speed can be very tricky. I highly recommend creating a GoToActionBase and

actions that moves to a destination will inherit GoToAction. This way, you can eliminate the need to chain to a GoToAction and also avoid the need for a StateMachine. To see an example of how this is done, check out

You may have noticed how we referenced the Target twice, this is a code smell and can be improved by using Sensors.