# Action RPG AI Example

The Action RPG AI example can be found in **SGOAP\Examples\ActionRPExample scene.**

💡 Please keep in mind this is an example on how to integrate SGOAP with game systems and provide a starting point. This is not a full implementation of systems and so some code is not optimized while SGOAP's core, which is the Planner is.

## Overview

This demo shows a pattern and starting point for the following

- An enemy Agent with 4 goals,
  - Find Player
  - Hurt Player
  - Collect Points
  - Collect Health Packs
- A Generic Action executing an ability
- Dynamic Goal Priorities
  - Prioritizing health when low HP
  - Prioritizing killing Player when player's HP is too low
- Dynamic Action Cost
  - Cost based on distance and or other factors
- A pattern for dynamically deciding which object for the agent to pick up.

Below, I’ll go into more details about each of these.

## Goals

**hasActionTarget** is the goal to find a target, this is connected with the **Action Seek.**

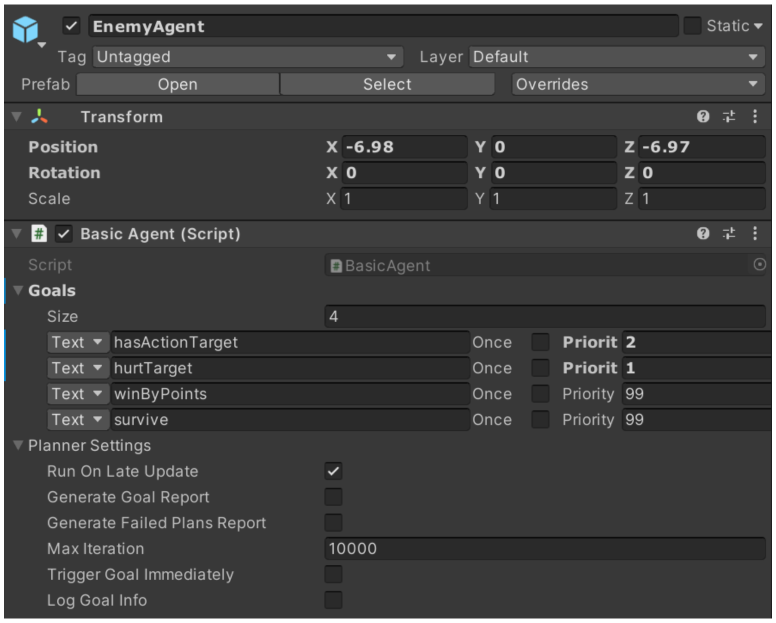
**winByPoints** is the goal for the agent to collect points, this is connected with the **Action PickUp: Points**

**survive** is the goal  for the agent to stay alive, this is connected with the A**ction PickUp: Health**

**hurtTarget** is the goal to hurt a target, this is connected with the **Actions Melee & DamageOverTime**

hasActionTarget priority is set higher so it is checked and will run before you try to see if you can hurt a target.

Trigger Goal Immediately is usually enabled, in our case, we have disabled it so that when 2 goals has equal priorities, the goal with the least cost is picked. For example, if the agent want health and points equally, he should pick the closer one or however decided.



EnemyAgent — Static
Tag Untagged — Layer Default
Prefab: Open | Select | Overrides

Transform

| | X | Y | Z |
|---|---|---|---|
| **Position** | -6.98 | 0 | -6.97 |
| **Rotation** | 0 | 0 | 0 |
| Scale | 1 | 1 | 1 |

Basic Agent (Script)

Script: BasicAgent

**Goals**

Size: 4

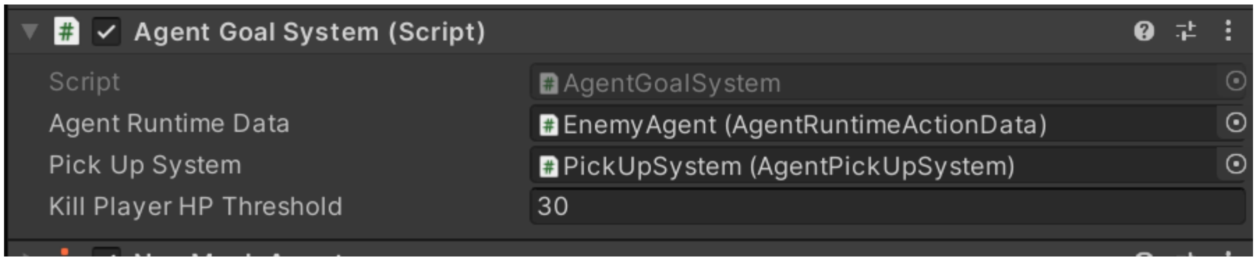
| | | | | |
|---|---|---|---|---|
| Text | hasActionTarget | Once | **Priorit** | **2** |
| Text | hurtTarget | Once | **Priorit** | **1** |
| Text | winByPoints | Once | Priority | 99 |
| Text | survive | Once | Priority | 99 |

Planner Settings

| | |
|---|---|
| Run On Late Update | ✓ |
| Generate Goal Report | |
| Generate Failed Plans Report | |
| Max Iteration | 10000 |
| Trigger Goal Immediately | |
| Log Goal Info | |

## Dynamic Goal Priorities

See Agent Goal System.cs



Agent Goal System (Script)

| | |
|---|---|
| Script | AgentGoalSystem |
| Agent Runtime Data | EnemyAgent (AgentRuntimeActionData) |
| Pick Up System | PickUpSystem (AgentPickUpSystem) |
| Kill Player HP Threshold | 30 |

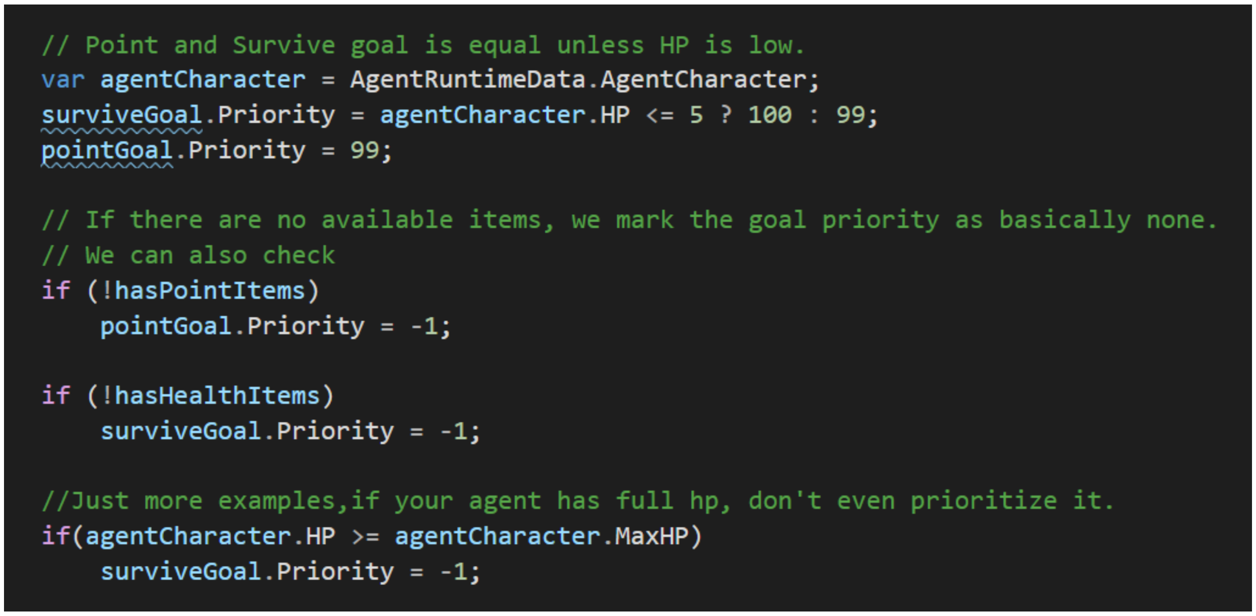
First we check if the pick up system thinks we can collect point or health or not.

```
// Not performant you'll need to cache these.
// Hard coded string for this example but you can use const string or states references.
var pointGoal = AgentRuntimeData.Agent.Goals.FirstOrDefault(x :Goal => x.Key == "winByPoints");
var surviveGoal = AgentRuntimeData.Agent.Goals.FirstOrDefault(x :Goal => x.Key == "survive");
var hurtTargetGoal = AgentRuntimeData.Agent.Goals.FirstOrDefault(x :Goal => x.Key == "hurtTarget");

var hasPointItems = PickUpSystem.IsActionUsable(EItemTrait.Points);
var hasHealthItems = PickUpSystem.IsActionUsable(EItemTrait.Health);
```

We decide that if the agent’s HP is less than or equal to 5, the priority will be 100. (This is our virtual maximum).

If there are no items for point or health, we set the goal priority to -1. (Our virtual minimum).

An interesting part here is if the Agent’s HP is full, we also set the survive goal’s priority to -1. As we don’t want him to collect it. However if your game lets your agent pick up but don’t consume, this  won’t be needed. This demo assumes it’s pick up and consume.

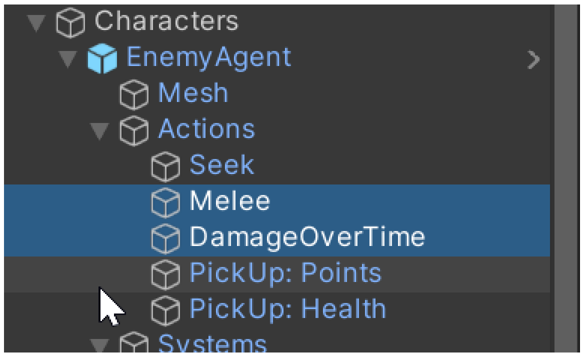


```
// Point and Survive goal is equal unless HP is low.
var agentCharacter = AgentRuntimeData.AgentCharacter;
surviveGoal.Priority = agentCharacter.HP <= 5 ? 100 : 99;
pointGoal.Priority = 99;

// If there are no available items, we mark the goal priority as basically none.
// We can also check
if (!hasPointItems)
    pointGoal.Priority = -1;

if (!hasHealthItems)
    surviveGoal.Priority = -1;

//Just more examples,if your agent has full hp, don't even prioritize it.
if(agentCharacter.HP >= agentCharacter.MaxHP)
    surviveGoal.Priority = -1;
```

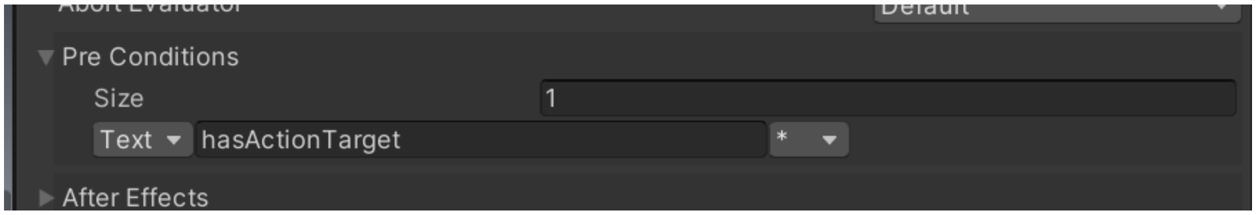
## Generic Action

The example provide two actions, using the same script with different behaviours,

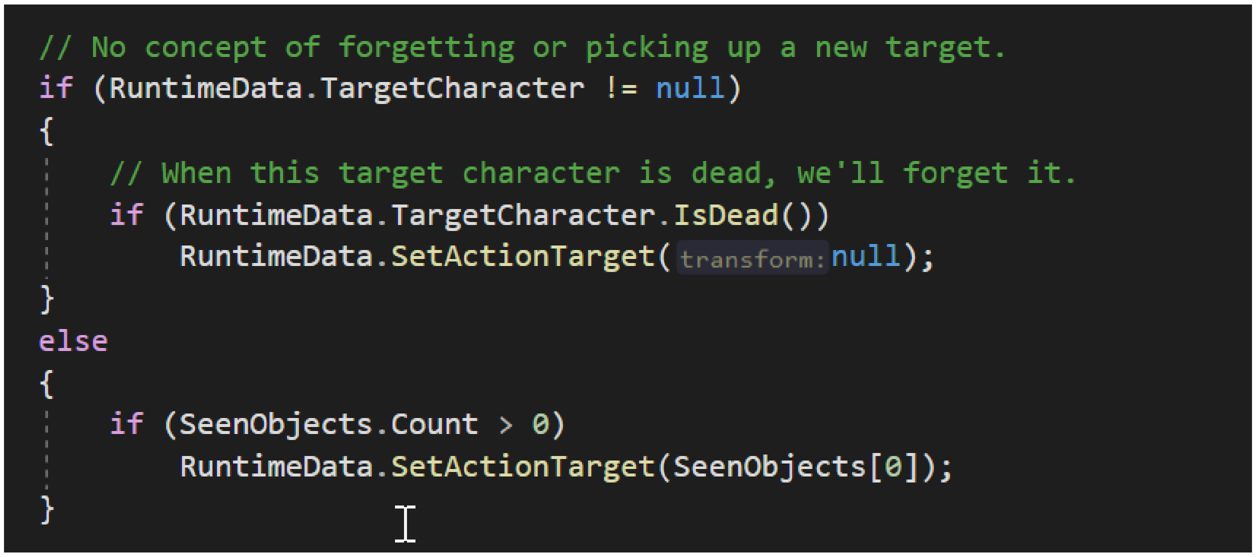
- Melee, walks up to player and immediately deal a damage
- Damage Over Time, walks up to player, deals damage every 0.5 second.



Both actions has the precondition for hasActionTarget which is set by the EyeSensorExample. It targets the AgentRuntimeData and Set the Action Target.

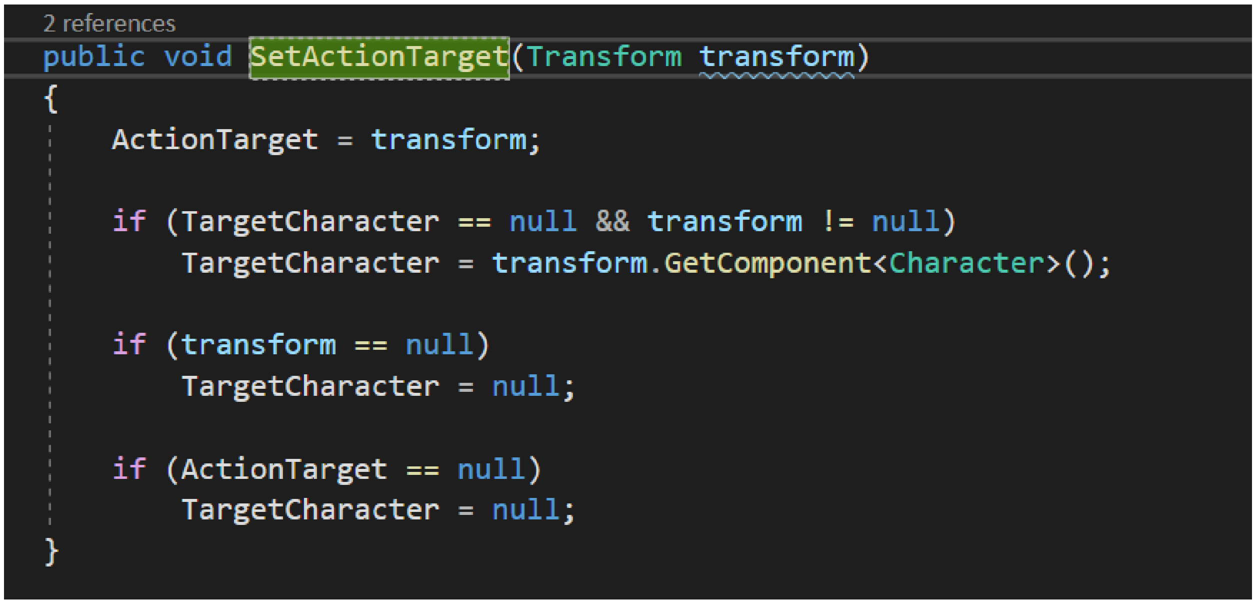


Pre Conditions
Size 1
Text hasActionTarget *
After Effects



```
// No concept of forgetting or picking up a new target.
if (RuntimeData.TargetCharacter != null)
{
    // When this target character is dead, we'll forget it.
    if (RuntimeData.TargetCharacter.IsDead())
        RuntimeData.SetActionTarget( transform: null);
}
else
{
    if (SeenObjects.Count > 0)
        RuntimeData.SetActionTarget(SeenObjects[0]);
}
```

When it is set, we also convert it a custom class Character. This is useful for when  we are dealing with Ability below.



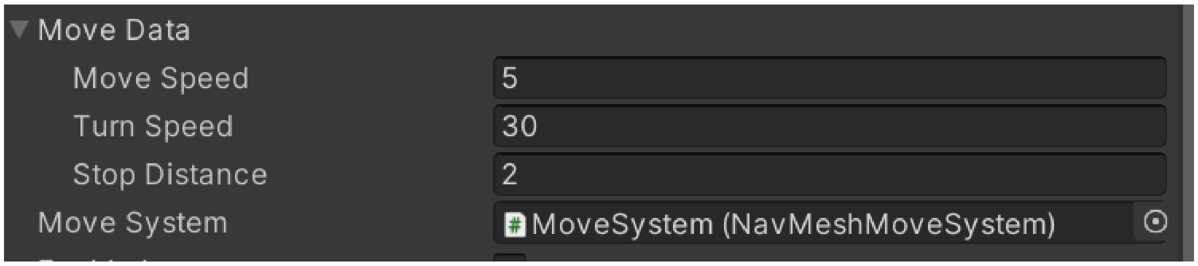
```
2 references
public void SetActionTarget(Transform transform)
{
    ActionTarget = transform;

    if (TargetCharacter == null && transform != null)
        TargetCharacter = transform.GetComponent<Character>();

    if (transform == null)
        TargetCharacter = null;

    if (ActionTarget == null)
        TargetCharacter = null;
}
```

To understand the action, see the features below,

▼ Movement

The action inherits MoveToAction and select its own MoveSystem implementation of NavMeshMoveSystem as this agent rely on Nav Mesh. The Move data determines when this action execute, how should it move.

When Stop Distance is reached, the action execute a Behaviour.



| | |
|---|---|
| ▼ Move Data | |
| Move Speed | 5 |
| Turn Speed | 30 |
| Stop Distance | 2 |
| Move System | MoveSystem (NavMeshMoveSystem) |

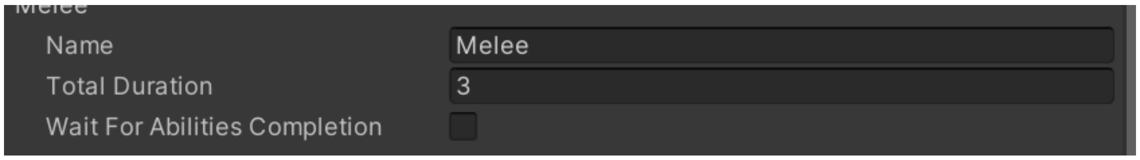
▼ Behaviours

When the Agent moved to its expected location, it'll execute a behaviour which contains, you can add more as you need. i.e in my game, I had events for parry windows and vulnerability that each action leaves the agent.

▼ General

You should name your behaviour something useful. The total duration determines how long this 'behaviour'  runs  for. i.e 3 seconds, regardless of how long the animation or ability is.

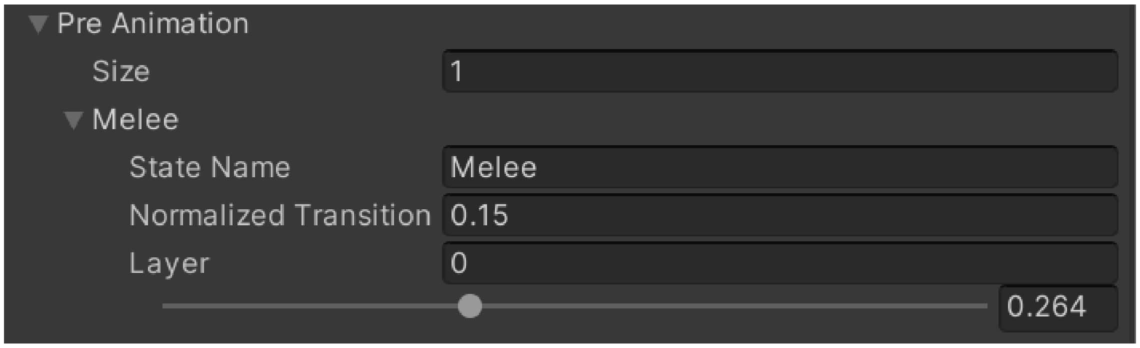
If you set WaitForAbilitiesCompletion, it'll wait until the ability is done.



Name: Melee
Total Duration: 3
Wait For Abilities Completion: ☐

- ▼ PreAnimation

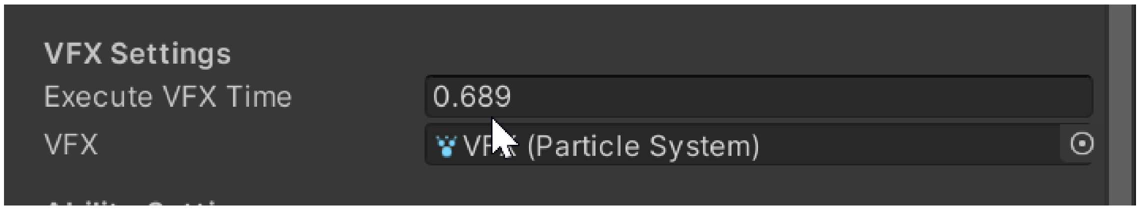
PreAnimation plays an animation state on the animator. The Slider let you visualize the time in seconds where which frame is at.



Pre Animation
Size: 1
Melee
State Name: Melee
Normalized Transition: 0.15
Layer: 0
0.264

- ▼ VFX

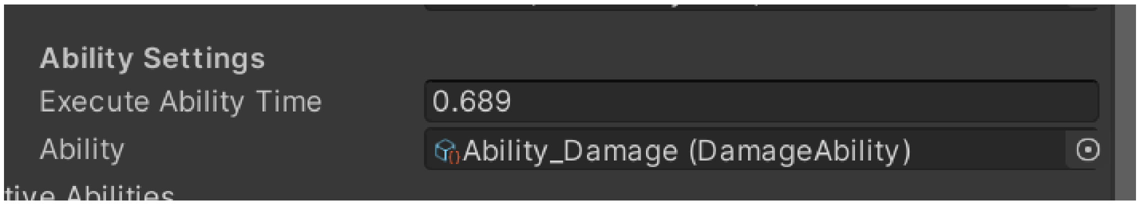
You can choose when the time to play a VFX particle system effect. For example, when swinging a sword, you want the effect to happy a bit later.



VFX Settings
Execute VFX Time: 0.689
VFX: VFX (Particle System)

- ▼ Ability

The ability setting is similar, in this case it is an Ability_Damage calculation, you want this to happen when the you actually check to deal damage.



Ability Settings

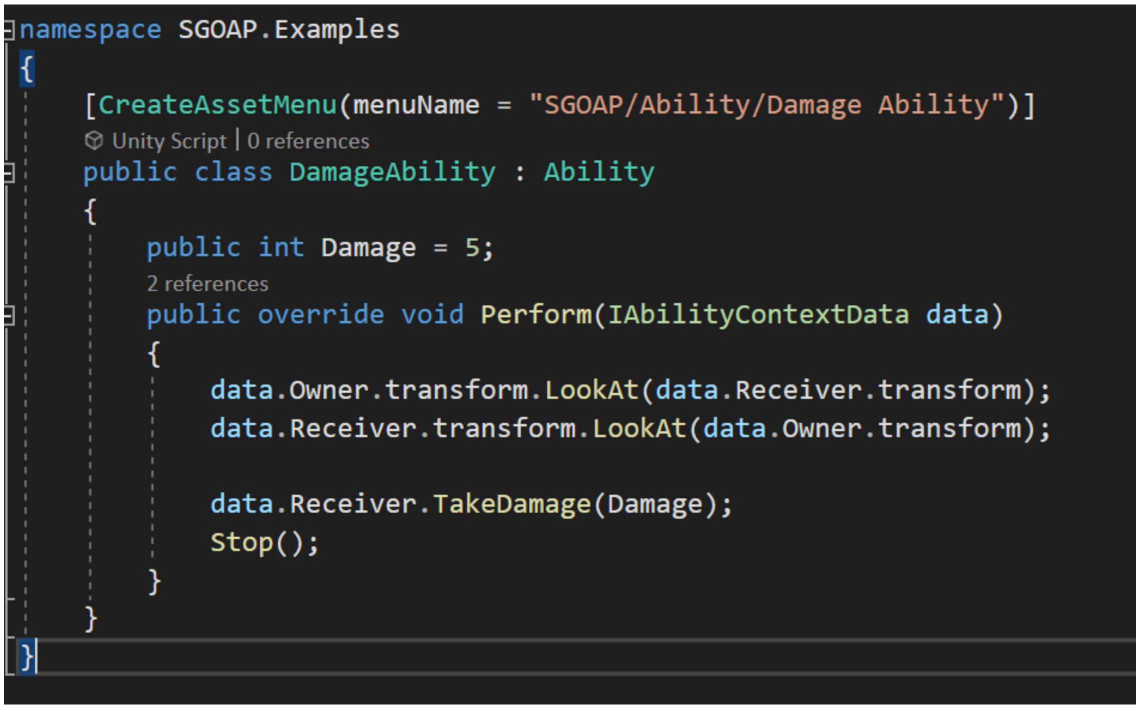
Execute Ability Time: 0.689

Ability: Ability_Damage (DamageAbility)

tive Abilities

The code for it is extremely simplistic as a demonstration,

We pass in an ability data context for the  Owner (Agent) and Receiver (Player) and immediately deal damage. We also immediately Stop the ability.

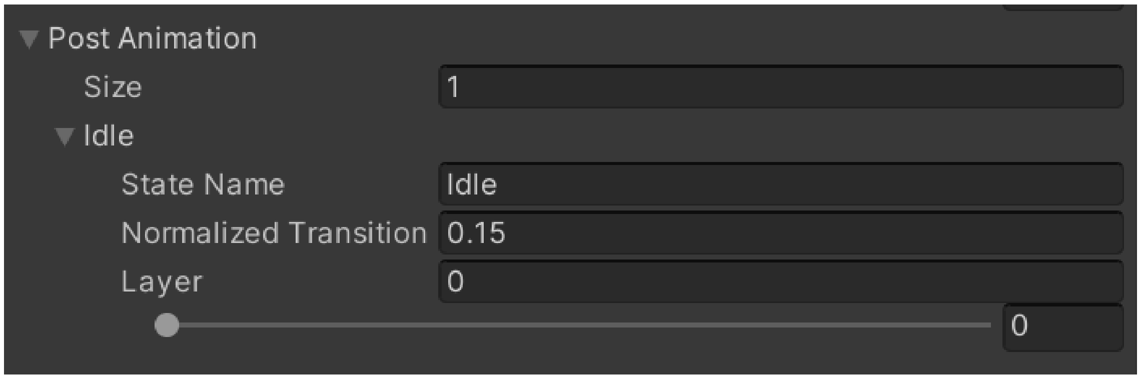


```
namespace SGOAP.Examples
{
    [CreateAssetMenu(menuName = "SGOAP/Ability/Damage Ability")]
    Unity Script | 0 references
    public class DamageAbility : Ability
    {
        public int Damage = 5;
        2 references
        public override void Perform(IAbilityContextData data)
        {
            data.Owner.transform.LookAt(data.Receiver.transform);
            data.Receiver.transform.LookAt(data.Owner.transform);

            data.Receiver.TakeDamage(Damage);
            Stop();
        }
    }
}
```

You can also check out DamageOvertimeAbility.cs for a running spell example.

▼ PostAnimation

Post Animation is pretty much the same as Pre except it runs at the end. In the example, we cross fade back to Idle.

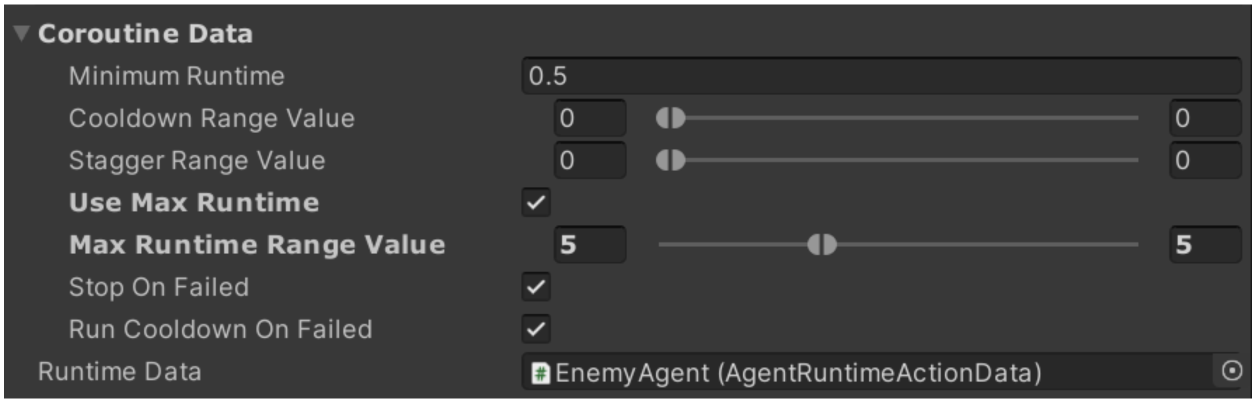


| | |
|---|---|
| ▼ Post Animation | |
| Size | 1 |
| ▼ Idle | |
| State Name | Idle |
| Normalized Transition | 0.15 |
| Layer | 0 |
| | 0 |

## Seek Action

The seek action is mostly the same as generic action, it is worth mentioning a feature  in Coroutine Data. You can set the Max Runtime, so that the action will not run longer than that duration.  This is useful if you want  the Agent to Seek for the player but when the player is dead or gone, will look for other  actions to do.

Alternatively, you can call Agent.ForceReplan() to start a new plan and cancel the current.



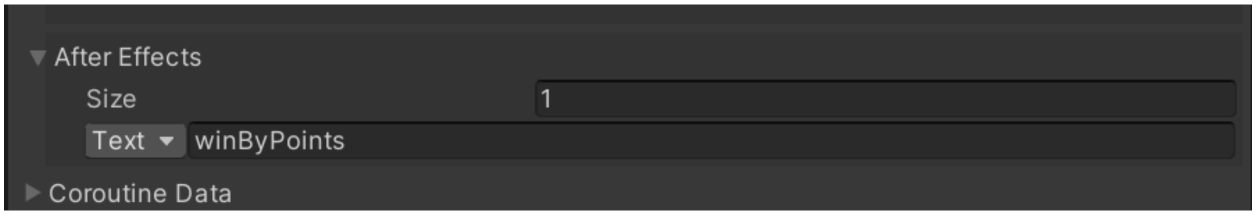
| | | |
|---|---|---|
| ▼ **Coroutine Data** | | |
| Minimum Runtime | 0.5 | |
| Cooldown Range Value | 0 | 0 |
| Stagger Range Value | 0 | 0 |
| **Use Max Runtime** | ✓ | |
| **Max Runtime Range Value** | **5** | **5** |
| Stop On Failed | ✓ | |
| Run Cooldown On Failed | ✓ | |
| Runtime Data | EnemyAgent (AgentRuntimeActionData) | |

## Pick Up & Dynamic Action Cost

Picking up items dynamically is a tricky one and you can resolve it in two ways,

1. having the Agent.State add string literal i.e “PickUpItem: item01” and then create a mapping of item01 → Object.
2. OR approach it coupling and specific to your application. I prefer this method as its simpler to debug and that’s what this example used.

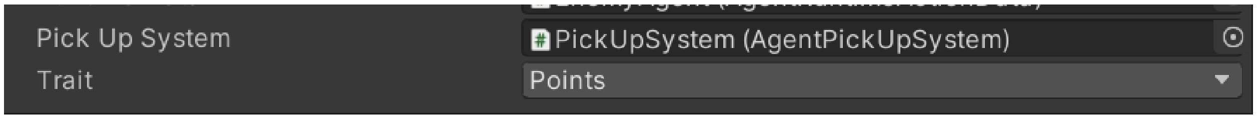
Pick Up Points action has afterEffect of winByPoints. This connects the action to the goal.



After Effects
Size: 1
Text: winByPoints
Coroutine Data

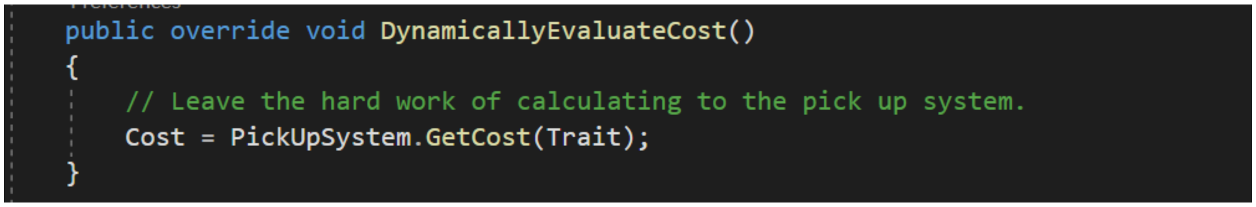
It references a pick up system and let it know it is going to reward you ‘points’. I used the word trait here as it made sense in the context I started with, but reward is a better word for it.

The cost for it is determined dynamically in the Pick Up Action  script. Let’s take a look.



Pick Up System: PickUpSystem (AgentPickUpSystem)
Trait: Points

It overrides DynamicallyEvaluateCost() and set the Cost to whatever the PickUpSystem decides  when you pass it a trait/type of item to pick up.



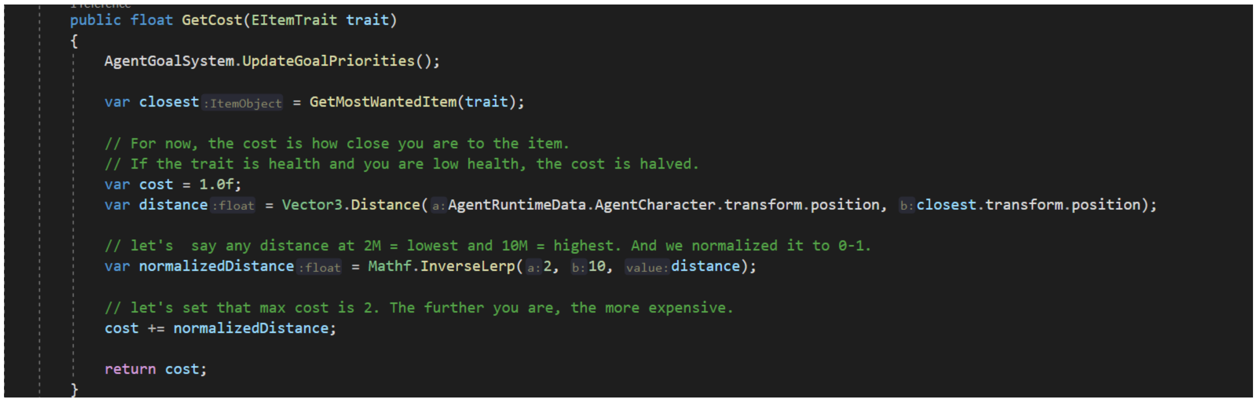
```
public override void DynamicallyEvaluateCost()
{
    // Leave the hard work of calculating to the pick up system.
    Cost = PickUpSystem.GetCost(Trait);
}
```

The PickUpSystem GetCost method  is pretty simple but you can make it as complicated as you need 😀.

We first update the goal priorities. Geet the closest item that matches the trait/reward.

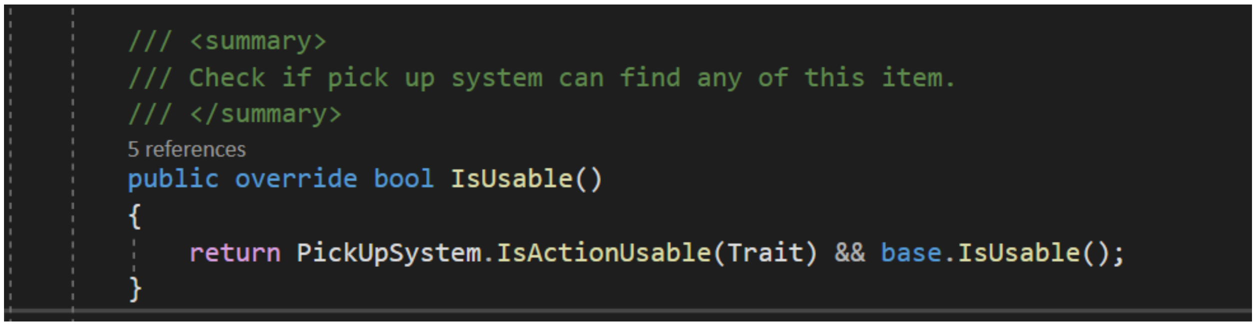
We default the cost to 1 for this action, checks the distance, normalizes it to 0-1 where 1 is too far and 0 is close. i.e the total cost for something 10m away is 1+1 = 2.

What this means is the Agent will choose the goal or action with a lower cost.



```
public float GetCost(EItemTrait trait)
{
    AgentGoalSystem.UpdateGoalPriorities();

    var closest :ItemObject = GetMostWantedItem(trait);

    // For now, the cost is how close you are to the item.
    // If the trait is health and you are low health, the cost is halved.
    var cost = 1.0f;
    var distance :float = Vector3.Distance( a:AgentRuntimeData.AgentCharacter.transform.position,  b:closest.transform.position);

    // let's  say any distance at 2M = lowest and 10M = highest. And we normalized it to 0-1.
    var normalizedDistance :float = Mathf.InverseLerp( a:2,  b:10,  value:distance);

    // let's set that max cost is 2. The further you are, the more expensive.
    cost += normalizedDistance;

    return cost;
}
```

Finally, It overrides IsUsable and check  with  the PickUpSystem if this trait is usable, this avoids the Agent looking for an item that doesn't even exist.



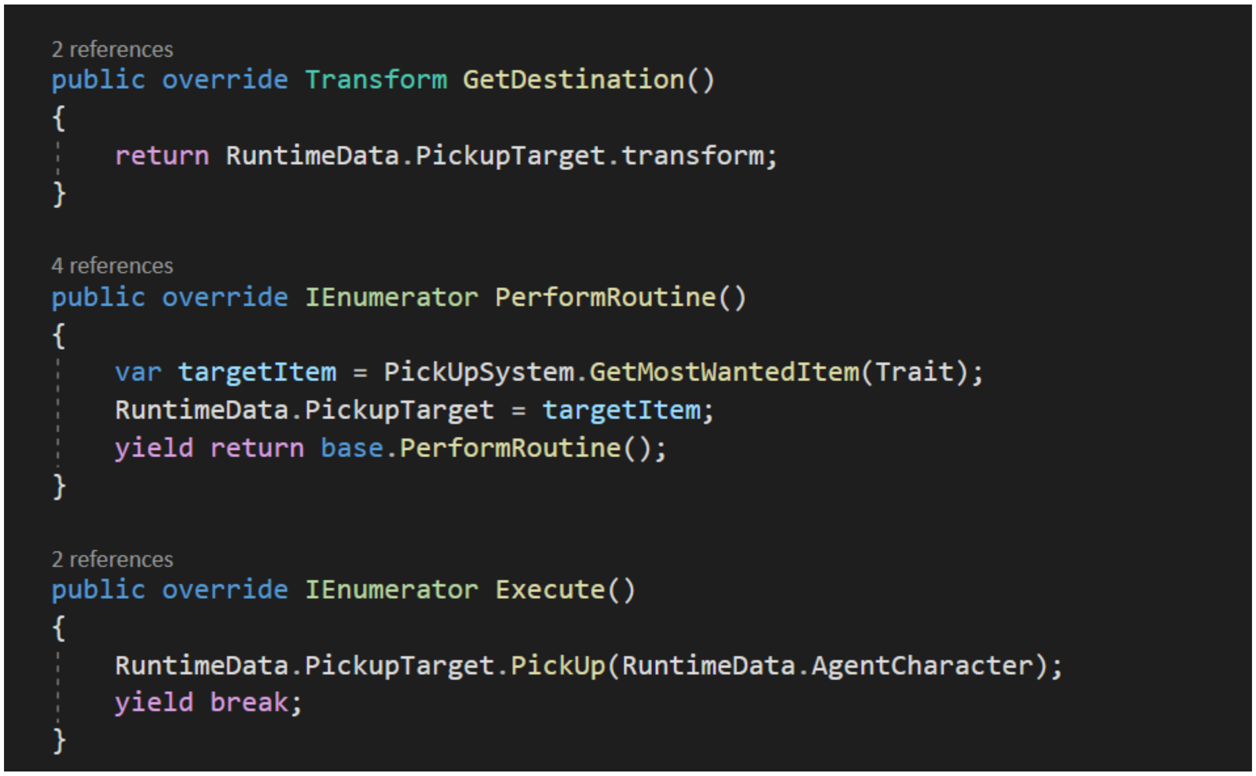
```
/// <summary>
/// Check if pick up system can find any of this item.
/// </summary>
5 references
public override bool IsUsable()
{
    return PickUpSystem.IsActionUsable(Trait) && base.IsUsable();
}
```

This action will run but how does it actually know which Health to go for when there are two?

When it performs, it sets the AgentRuntimeData.PickupTarget to what the PickupSystem thinks is the most wanted item based on the trait.

We also have to override GetDestination and return the transform of our PickupTarget so the MoveSystem knows where to go.

When the move system reaches the item, it'll call execute we simply call PickUp on the item.



```
2 references
public override Transform GetDestination()
{
    return RuntimeData.PickupTarget.transform;
}

4 references
public override IEnumerator PerformRoutine()
{
    var targetItem = PickUpSystem.GetMostWantedItem(Trait);
    RuntimeData.PickupTarget = targetItem;
    yield return base.PerformRoutine();
}

2 references
public override IEnumerator Execute()
{
    RuntimeData.PickupTarget.PickUp(RuntimeData.AgentCharacter);
    yield break;
}
```

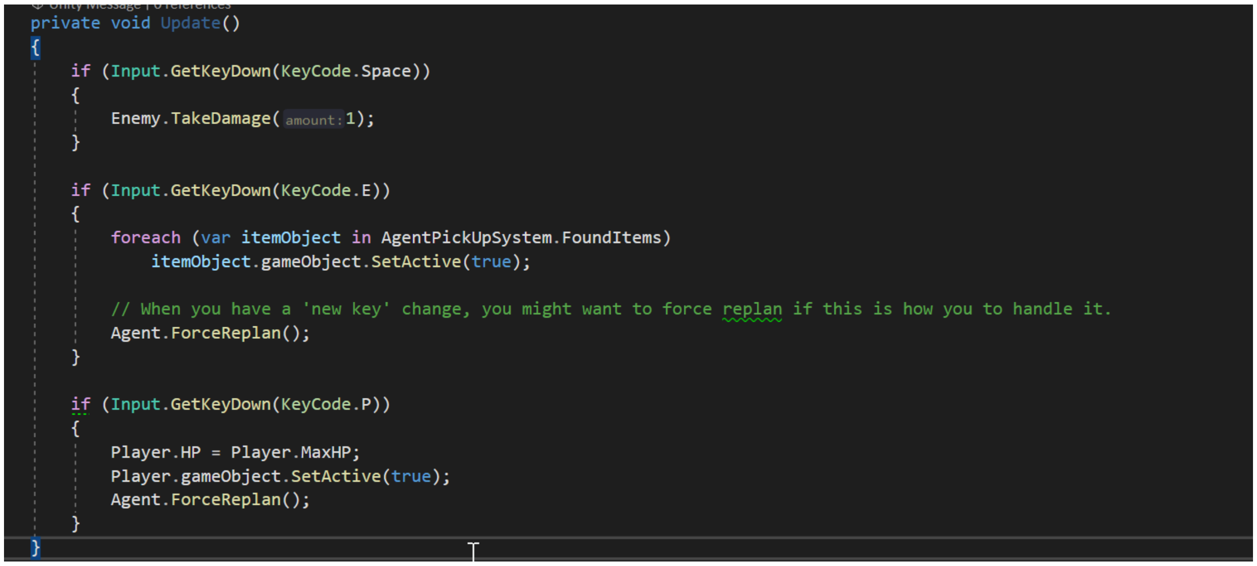
## Demo Script

The Demo Input script gives you a few input to try out.

Space - Deals Damage to Enemy

E - Turns all the items back on AND forces Agent to Replan, for example if the agent is busy attacking the player, the force replan will cancel  that action.

P - Recover Player's HP AND forces Agent to Replan.



```
private void Update()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        Enemy.TakeDamage( amount: 1);
    }

    if (Input.GetKeyDown(KeyCode.E))
    {
        foreach (var itemObject in AgentPickUpSystem.FoundItems)
            itemObject.gameObject.SetActive(true);

        // When you have a 'new key' change, you might want to force replan if this is how you to handle it.
        Agent.ForceReplan();
    }

    if (Input.GetKeyDown(KeyCode.P))
    {
        Player.HP = Player.MaxHP;
        Player.gameObject.SetActive(true);
        Agent.ForceReplan();
    }
}
```

I hope this give you guys a good starting point! AI in a game can be confusing, GOAP is quite so, so as a tip, don’t worry too much about decoupling until you know your Agent well.