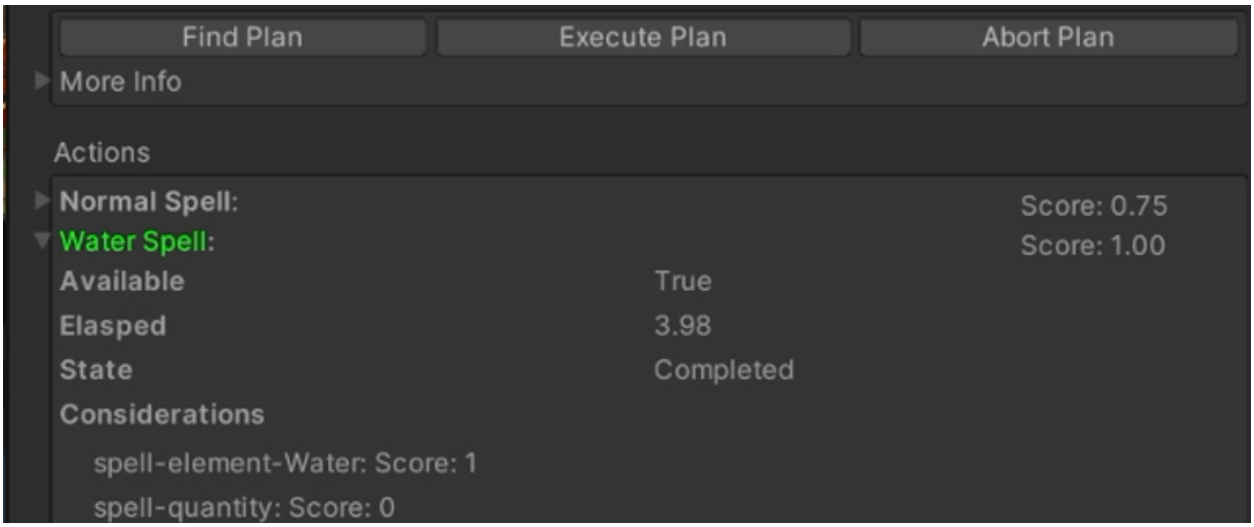


Installation and Getting Started Guide.

Installation

Installing the package is simply unpacking the .unitypackage downloaded via Asset Store through Package Manager.

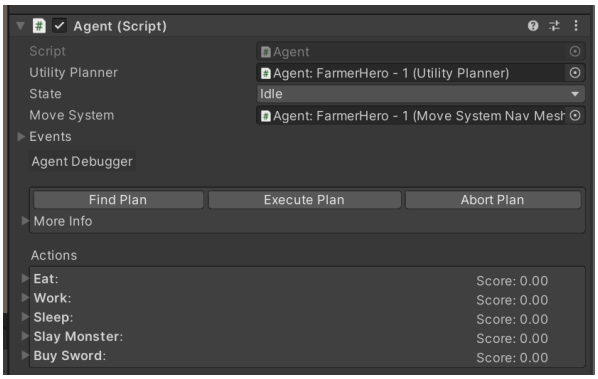
Import the package and you are ready to go!

Setting up an AI

Create an empty game object and attach an Agent Component. An UtilityPlanner component will automatically be added.

Agent Component is responsible for managing its current state, such as idle, moving, planning or executing an action.

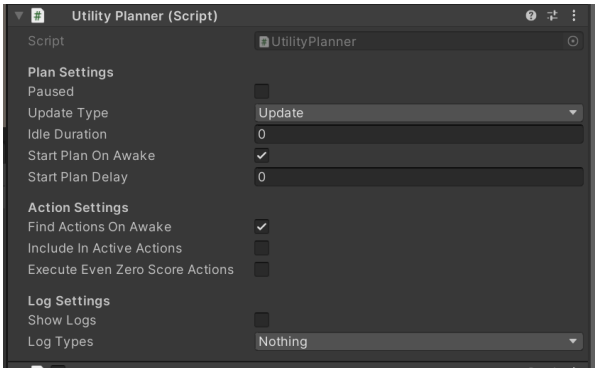
1. **Utility Planner,** reference to utility planner, if you want to change the



way it plans, you can drop in a different one.

2. **State**, shows the current state the agent is in. This is here for debug purposes and not to be changed in the editor.
3. **Events**, has a collection of Unity Events for almost every notable changes in the Agent planning and executing processes.
4. **Agent Debugger** and below is runtime information or debug buttons you can use to help understand your agent.

Utility Planner Component takes your available actions and find the best action given the current circumstances.



1. Paused, setting this to true will pause the planning and executing workflow of the agent.
2. Update Type, determines if it will be called in Update, LateUpdate, FixedUpdate or Script. Script allows you to manually handle it yourself for Turn Base Games.
3. Idle Duration, is the time the agent stays in idle before starting a new plan.
4. Start Plan on Awake, enabled will plan immediately.

5. Start Plan Delay, delays the plan at the start, good if you want to slightly have a different timing to your agents.
6. Find Actions On Awake, automatically searches for actions nested under the Agent.
7. Include In Active Actions, will search even actions that are turned off.
8. Execute Event Zero Score Actions, will allow an action to perform even if it returns 0 score. This is useful if you want your Agent to choose even the worse move. Alternatively in your consideration, return 0.1 instead or go to the action and set the MinScore to be 0.1
9. Show Logs, enabled will allow Debug.Log messages to show. Useful for debugging issues.
10. Log Types, let us filter out what type of logs to show.

Once these 2 components are added, the Agent will be able to think on its own. However, it would need 3 more scripts to actually do something!

1. **Utility Action,** is a monobehaviour that contains a list of consideration and will actually do the action such as “Eat or Sleep”
2. **Consideration,** is a scriptable object that returns a score to a question. i.e How Hungry am I?

3. **Data Context,** is a monobehaviour implementing IAgentDataContext that is used to pass information down to actions & consideration. Think of this as a way of retriesving the agent states / world states. This is automatically searched by the agent.

We will be making an Agent that has to choose between saving a person or not. In these cases, you'd consider a few things, can I save them? do I want to save them? And naturally the context is, what is my relationship to them?

For simplicity sake, I will choose to make the Agent save the person if they are his friend. So let's be the the Agent's friend 😅.

Here's an overview breakdown of the tasks we will be making.

Actions

- Save Person
- Walk Away

Considerations

- should-save

Data Context

- IsFriend : Bool

1. Create the data context implementing IAgentDataContext and attach it to the Agent. When you start, the data context is automatically cached.

```
using UnityEngine;

namespace TinnyStudios.AIUtility.Impl.Examples.Onboarding
{
    public class SaveFriendDataContext : MonoBehaviour, IAgentDataContext
    {
```

```
        public bool IsFriend = true;
    }
}
```

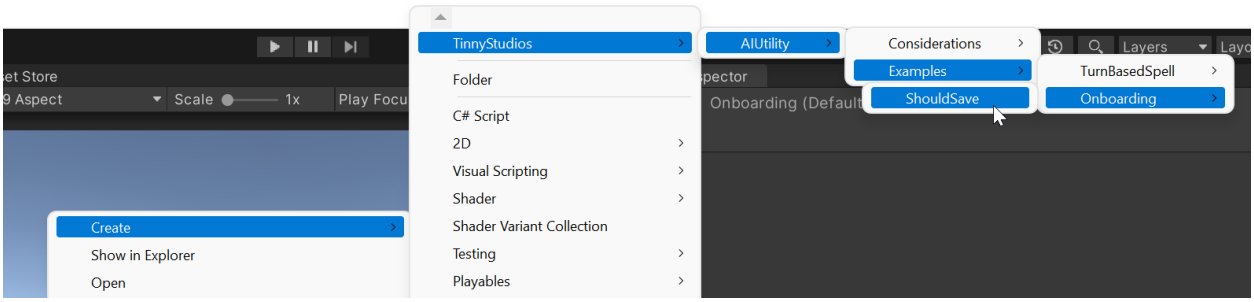
2. Create the Should Save consideration class by inheriting Consideration and implementing GetScore(Agent agent) method. Here we retreive the SaveFriendDataContext from agent.GetContext and return 1 if they are friends. Here we add an invert value so we can choose to have the opposite response.

```
using UnityEngine;

namespace TinnyStudios.AIUtility.Impl.Examples.Onboarding
{
    [CreateAssetMenu(menuName = "TinnyStudios/AIUtility/Examples/Onboarding/ShouldSave")]
    public class ShouldSaveConsideration : Consideration
    {
        public bool Inverted;

        public override float GetScore(Agent agent)
        {
            var context = agent.GetContext<SaveFriendDataContext>();
            if(!Inverted)
                return context.IsFriend ? 1 : 0;
            else
                return context.IsFriend ? 0 : 1;
        }
    }
}
```

Go back to the editor and create the asset for ShouldSave Consideration through the menu. Right click anywhere in the project window and **click Create/TinnyStudios/AIUtility/Examples/Onboarding/ShouldSave**

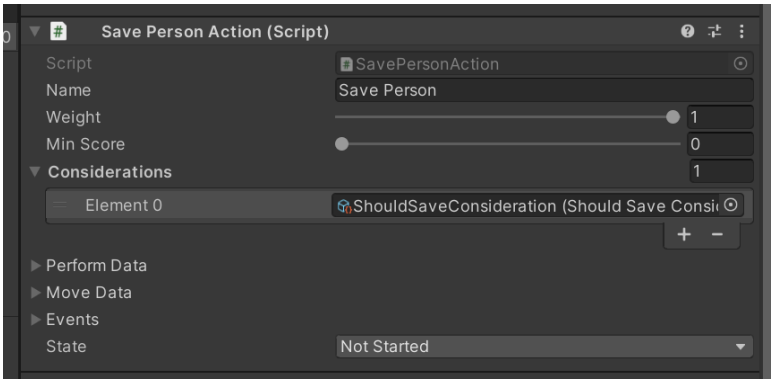


3. Create the action class SavePersonAction inheriting Utility Action. You will need to implement Perform(Agent agent). This method is called when the Agent decided to execute an action as it is picked as the best action. So here let's log "Agent Saved Person" and return EActionStatus.Completed to complete the action immediately. You can return Running to keep going but that will be covered by in the API instead as this is an onboarding tutorial.

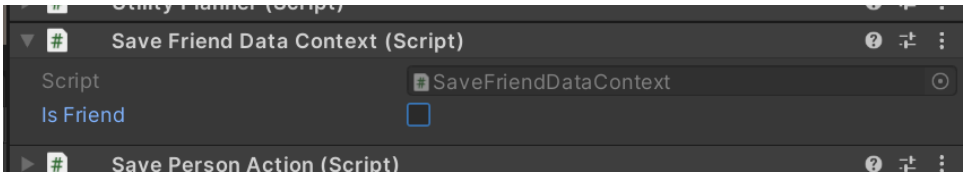
```
using UnityEngine;

namespace TinnyStudios.AIUtility.Impl.Examples.Onboarding
{
    public class SavePersonAction : UtilityAction
    {
        public override EActionStatus Perform(Agent agent)
        {
            Debug.Log("Agent Saved Person");
            return EActionStatus.Completed;
        }
    }
}
```

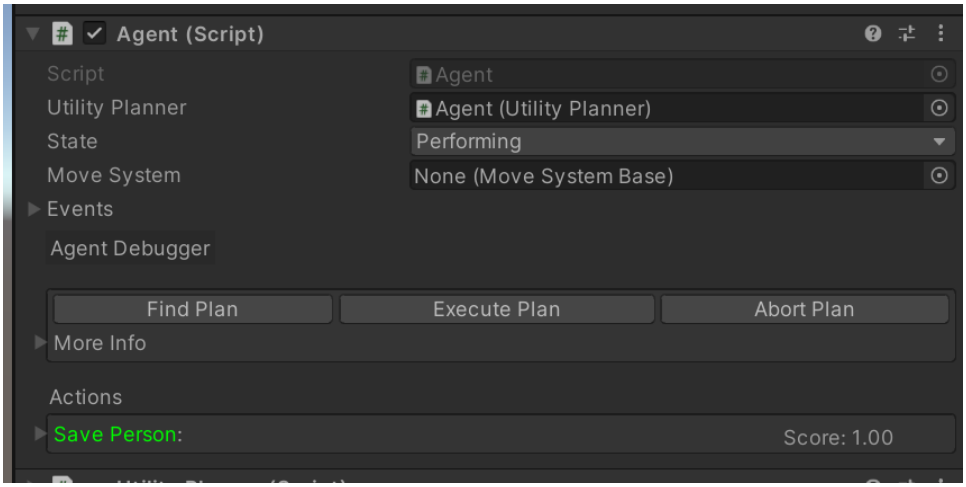
Add the SavePersonAction on your Agent, set the name to Save Person and add the ShouldSaveConsideration in Considerations list.



Press Play and you should see the log "Agent Saved Person" logging every frame. Now if you go to the SaveFriendDataContext and set IsFriend to false, you see it stop.



You will also notice the Action Debugger section shows the action with the current evaluated score and is highlighted green. This means it was last executed.



4. Let’s make the final action WalkAway, it’s almost identical to the SavePersonAction as we are simply logging. Do note, you can pretty much make one action here called ReactAction that broadcast a type of event to subscribers but I’m keeping it very simplistic and concrete to illustrate how Utility AI Framework works.

```
using UnityEngine;

namespace TinnyStudios.AIUtility.Impl.Examples.Onboarding
{
    public class WalkAwayAction : UtilityAction
    {
        public override EActionStatus Perform(Agent agent)
        {
            Debug.Log("Agent Walk away from person");
            return EActionStatus.Completed;
        }
    }
}
```

Add WalkAwayAction to the Agent. Create a new ShouldSave Consideration and mark Inverted as true. Add the consideration to the WalkAwayAction Considerations list.

Press Play again and now when IsFriend is false, we will get the walk away action executed.

Moving

The final core feature to know about is the MoveSystem. The move system is designed for you to override and implement. In this guide, we won't be creating a new move system but using one of the existing example, MoveSystemNavMeshExample.

1. Add MoveSystemNavMeshExample to the Agent.
2. Assign it to the MoveSystem field on the Agent.
3. Add a NavMeshAgent to the Agent and assign it to the MoveSystemNavMeshExample NavAgent field
4. Create a cube, name it ground and mark it static with the following transform properties
    a. Position: 0,0,0
    b. Rotation: 0,0,0
    c. Scale: 20,0.1,20
5. Let's bake the Ground as a Navigation Path. Open the Navigation panel (Windows/AI/Navigation). Select the Bake Tab and click Bake.
6. Select your Agent and add a cube in it with the following transform properties
    a. Position: 0,0.5,0
    b. Rotation: 0,0,0
    c. Scale: 1,1,1

7. Create a new Empty Object called RescueLocation and give it the following transform properties
    a. Position -9,0,-9
8. Select the SavePersonAction, Click MoveData and set the follow properties
    a. Required: True
    b. Destination Transform:
9. Set the MainCamera transform properties to
    a. Position: 0,10,-15
    b. Rotation: 40,0,0

Here's a .unitypackage of the scene in case a step was mis-written or followed.

As an extra exercise, try making the WalkAwayAction move to a different location.

I hope this gives you a starting point on how to use the package. If you would like to get a more in depth understanding, check out the **demo scenes here.**