UAI Creator – Guide

Welcome

This is the official guide for the Unity tool UAI Creator.

Make sure to use the newest version of the guide, which can be found at:
https://github.com/DevOelgaard/UnityUtilityAiSystem/tree/master

This tool is currently a prototype and by no means ready for production. You are free to use it as you see fit but be aware that it isn't a priority to make the updates backwards compatible at this point.

Feedback is much welcomed; you can reach me at: DevOelgaard@gmail.com.

Thank you for trying the UAI Creator.

1 Quick Start

This chapter helps you get started with a simple 'hello world' example.

This guide was made for Unity version 2021.2.10f1 on Windows, if you are running a different version or system, the guide might not be 100% accurate.

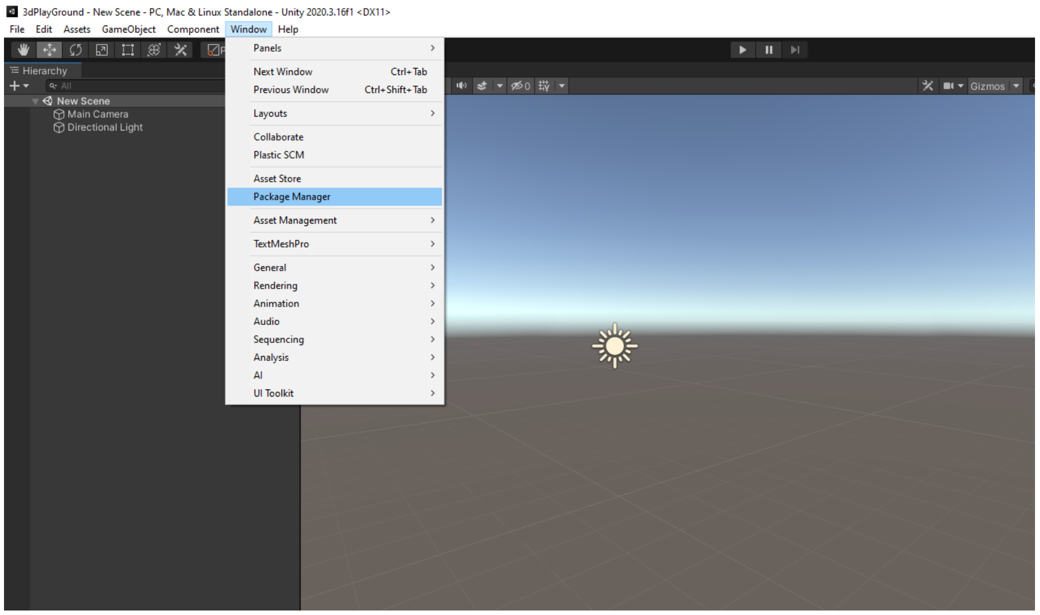
Unity version:

Make sure your Unity version is >2021.2.10f1.

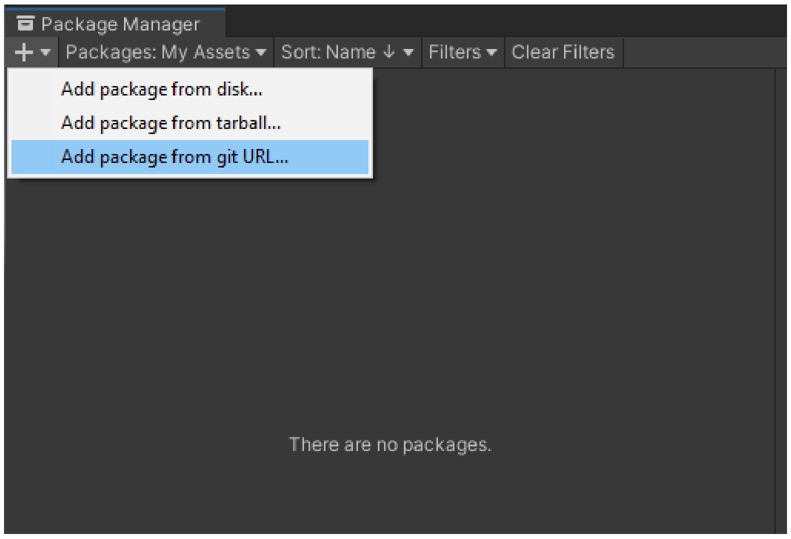
Follow this Link to get the newest Unity version: https://unity3d.com/get-unity/update.

1.1 Installation

1. Open Package Manager: Window -> Package Manager

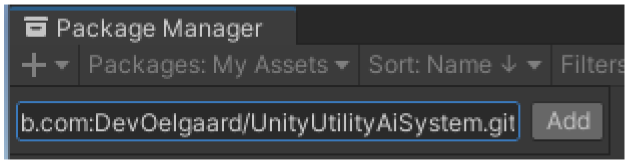


2. In top-left corner of package manager: + -> Add package from git URL...



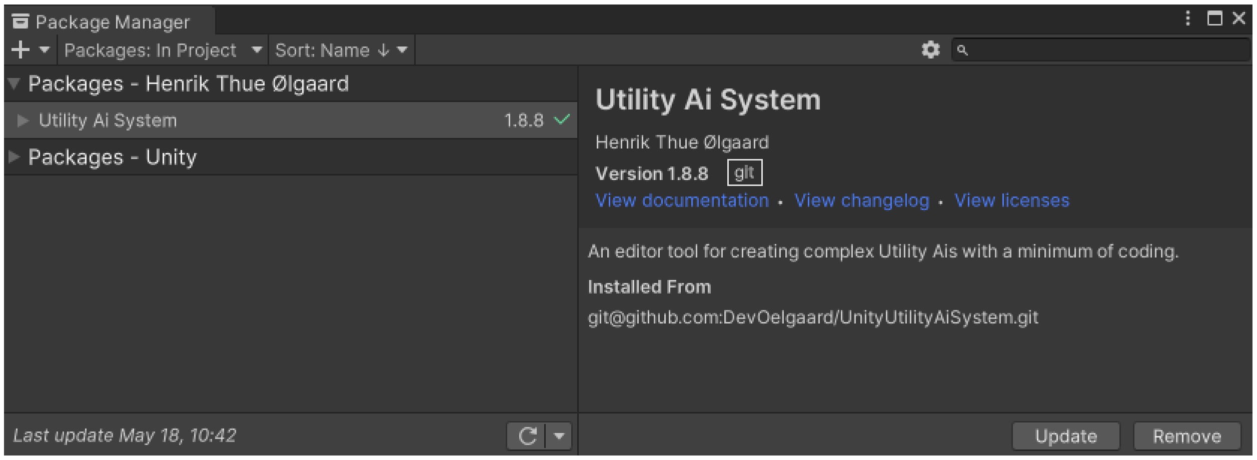
There are no packages.

3. Add one of the following links and press ‘Add’
    a. HTTPS: https://github.com/DevOelgaard/UnityUtilityAiSystem.git
    b. SSH: git@github.com:DevOelgaard/UnityUtilityAiSystem.git (Requires SSH Key)
    c. For help See: https://docs.unity3d.com/Manual/upm-git.html#Git-HTTPS



b.com:DevOelgaard/UnityUtilityAiSystem.git

4. The package is now installed and ready for use.



Utility Ai System

Henrik Thue Ølgaard

Version 1.8.8 git

An editor tool for creating complex Utility Ais with a minimum of coding.

Installed From

git@github.com:DevOelgaard/UnityUtilityAiSystem.git

1.2 Hello World

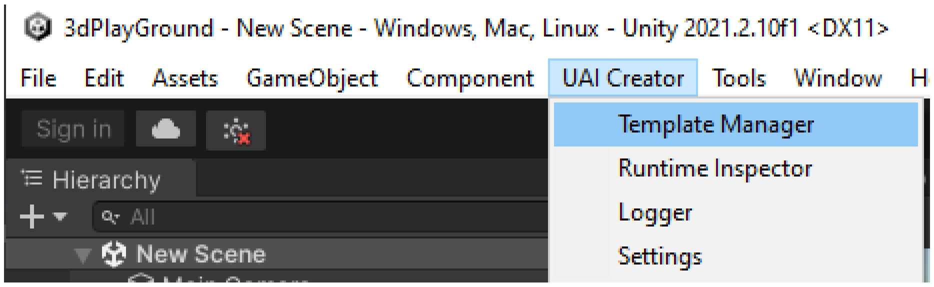
1.2.1 Code

1. Create a new C# script anywhere in your assets folder and name it “PrintTxt.cs” and open it in your favorite IDE.
2. Place the following code inside the PrintTxt.cs file

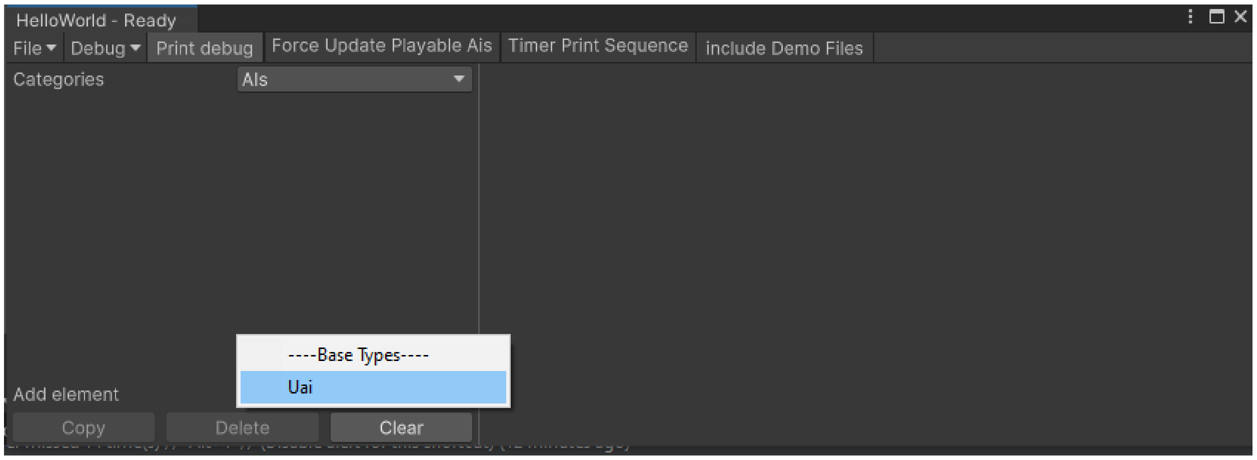
```
using System.Collections.Generic;
using UnityEngine;

public class PrintTxt : AgentAction
{

    // Here you define the parameters to be shown in the various UIS
    protected override List<Parameter> GetParameters()
    {
        return new List<Parameter>
        {
            // The format is: new Parameter(ParameterName, 
DefaultValue)
            // The Parameter supports all primitive types and some 
others like Unity.Color
            new Parameter("Text to print", "")
        };
    }

    // Called the first time the action is selected
    public override void OnStart(IAiContext context)
    {
        PrintText(context);
    }

    // Called at each successive selection of the action
    public override void OnGoing(IAiContext context)
    {
        PrintText(context);
    }

    private void PrintText(IAiContext context)
    {
        var text = GetParameter("Text to print").Value as string;

        var agent = context.Agent as AgentMono;

        Debug.Log(agent.name + ": " + text);
    }
}
```

1.2.2 UAI Template

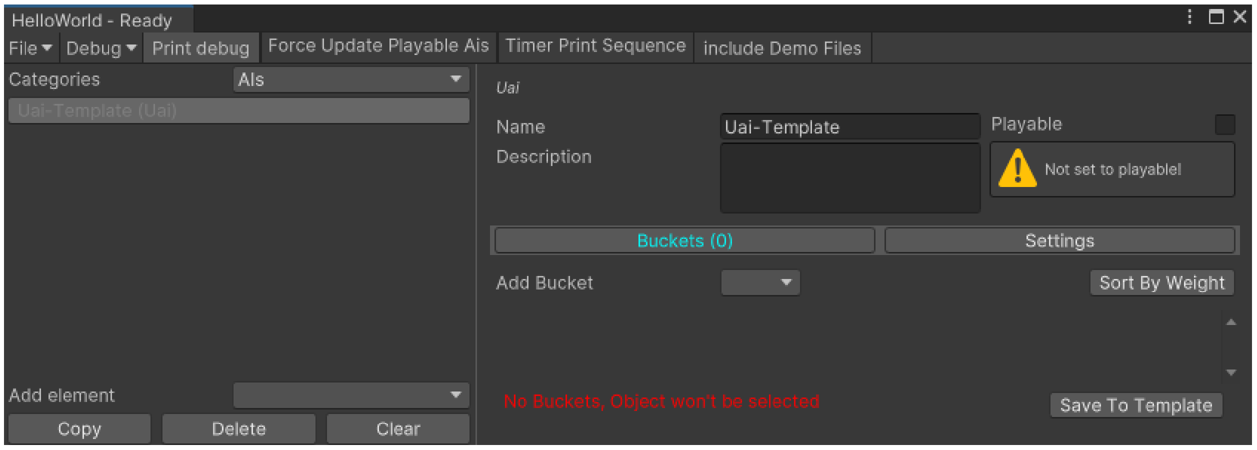
1. In the top toolbar Open UAI Creator -> Template Manager



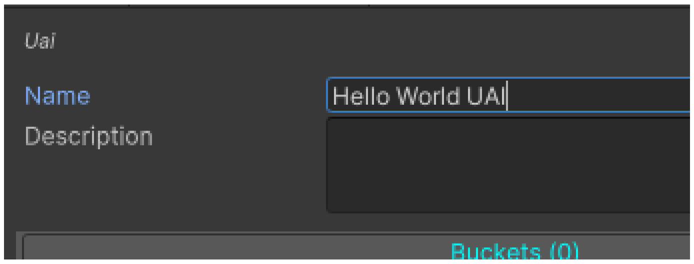
2. If prompted press “Create New Project” and select an empty folder named “HelloWorld” anywhere on your system.
    a. WARNING!!! If you chose a non-empty folder you might lose the content of the folder
3. In the ‘AIs’ category Add element -> Uai



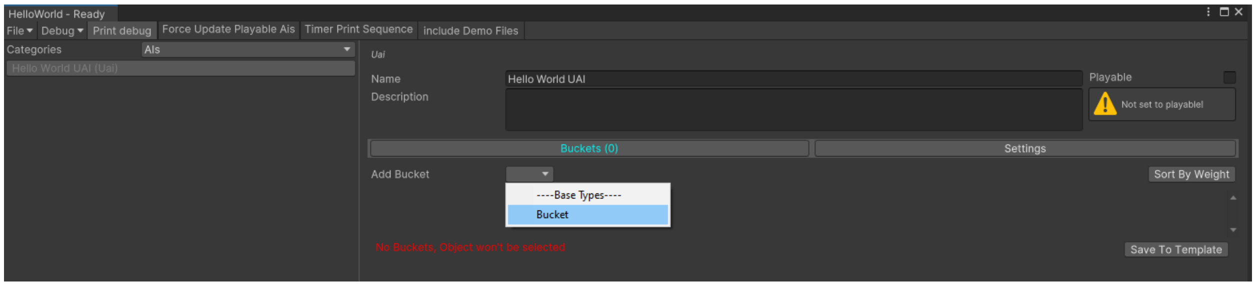
4. Double click the new Uai-Template (Uai)



5. Name the UAI: ‘Hello World UAI’



6. Add bucket



Hello World UAI (Uai)

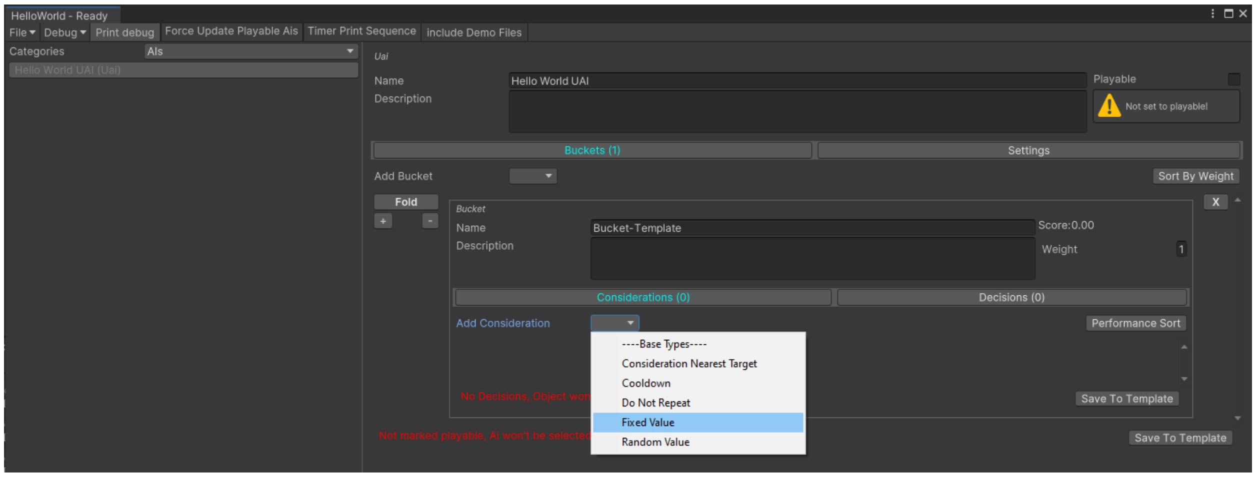
Name: Hello World UAI

Description

Not set to playable!

No Buckets, Object won't be selected

7. Expand the bucket and add Consideration ‘Fixed Value’



Hello World UAI (Uai)

Name: Hello World UAI

Description

Not set to playable!

Name: Bucket-Template

Description

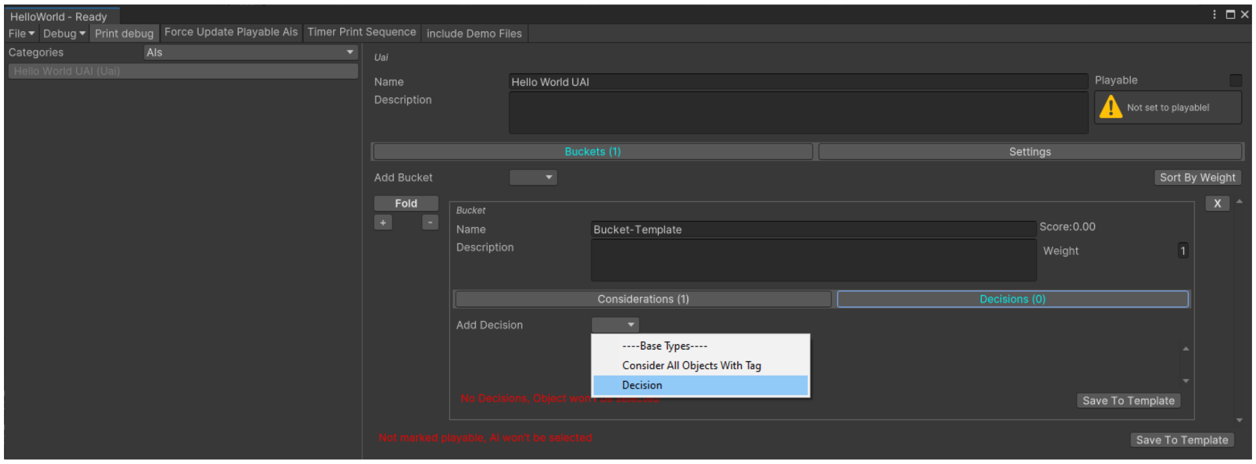
Score:0.00

Weight: 1

No Decisions, Object won

Not marked playable, AI won't be selected

8. In Decisions tab: Add decision



Hello World UAI (Uai)

Name: Hello World UAI

Description

Not set to playable!

Name: Bucket-Template

Description

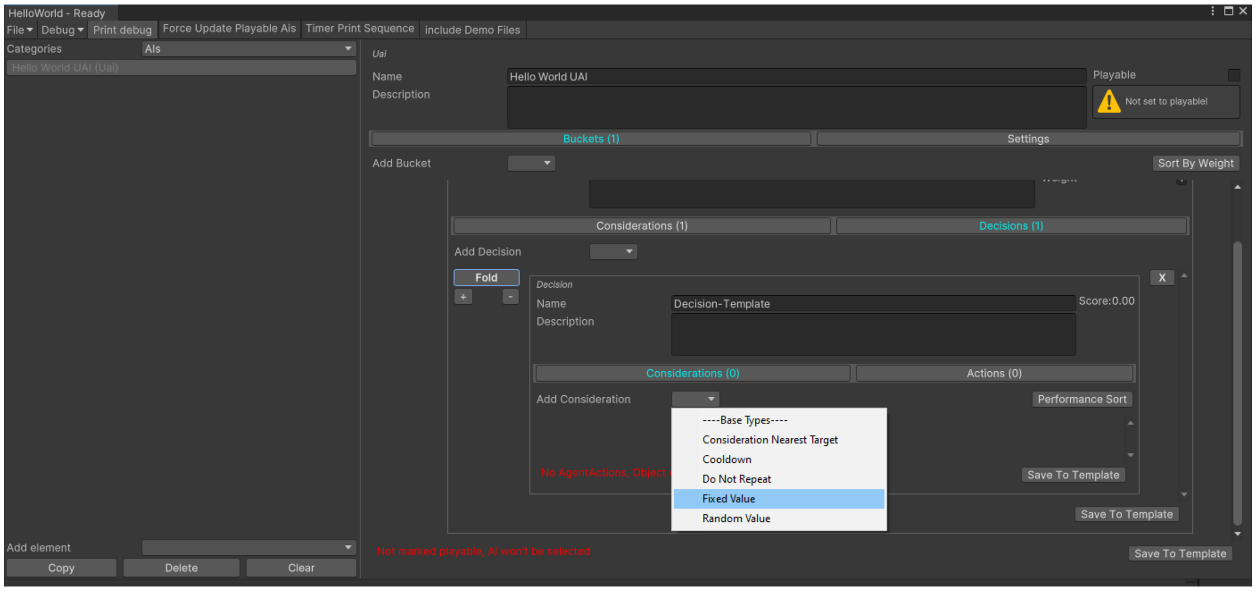
Score:0.00

Weight: 1

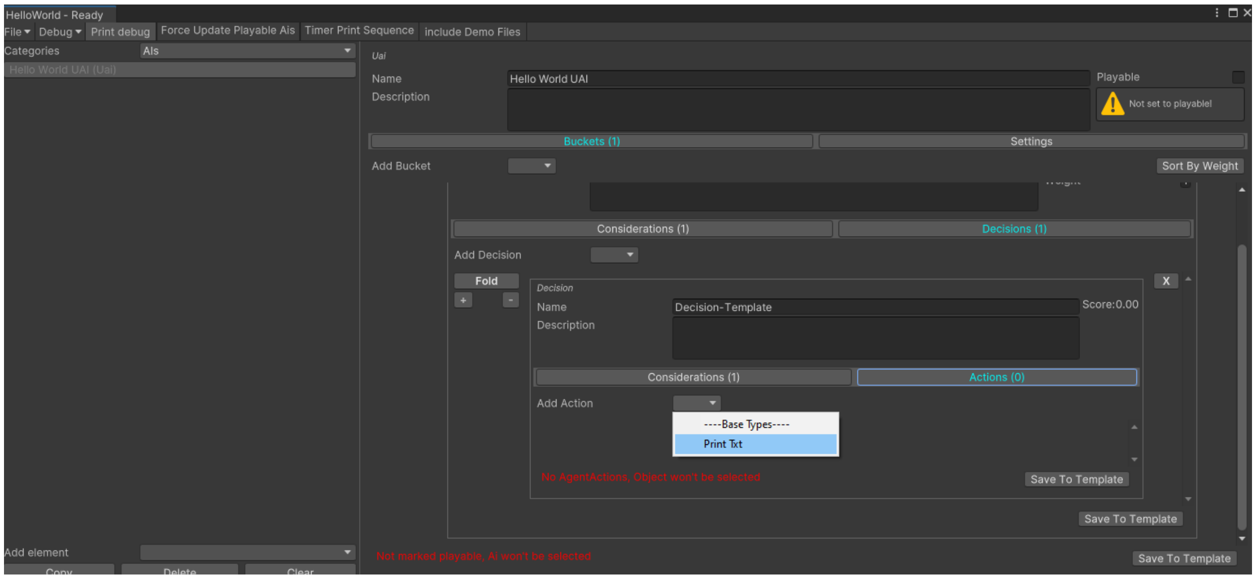
No Decisions, Object won't be selected

Not marked playable, AI won't be selected

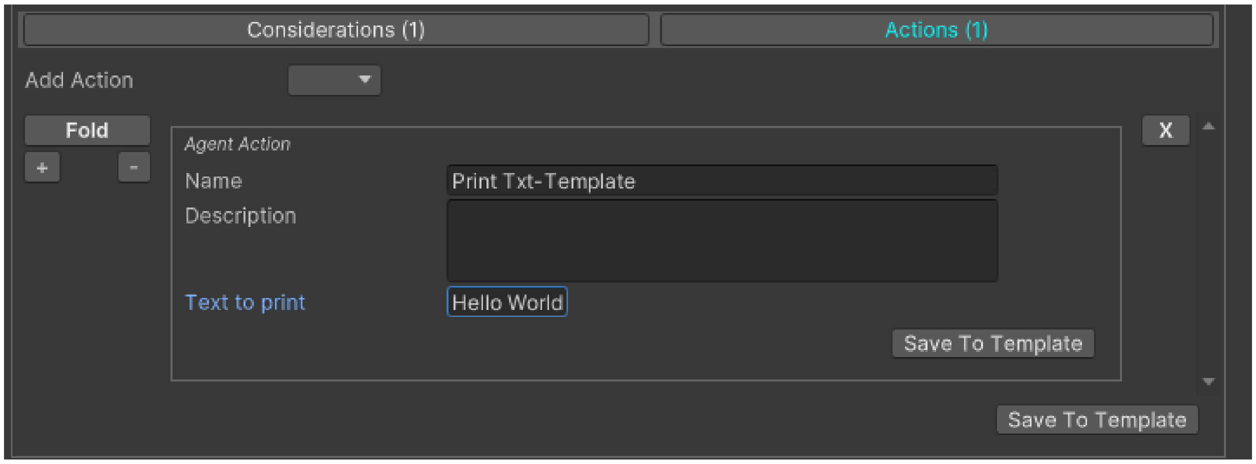
9. Expand the decision and add Consideration ‘Fixed Value’



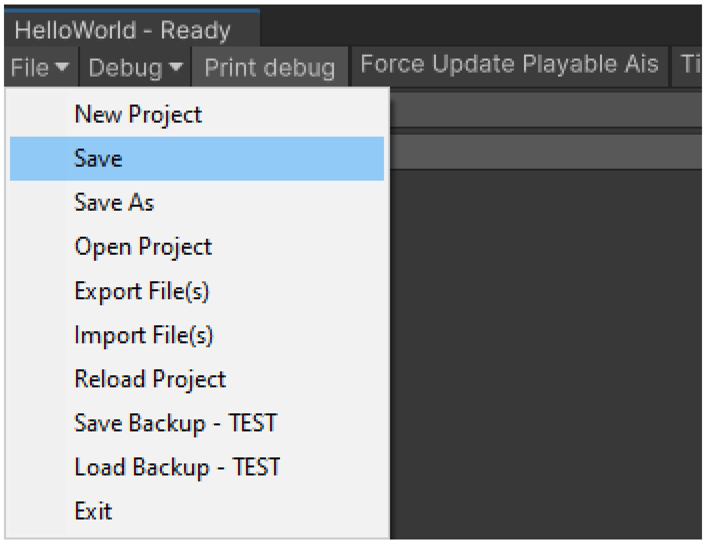
10. In actions tab add the Action: ‘Print Txt’. *This is the action you implemented in code before*



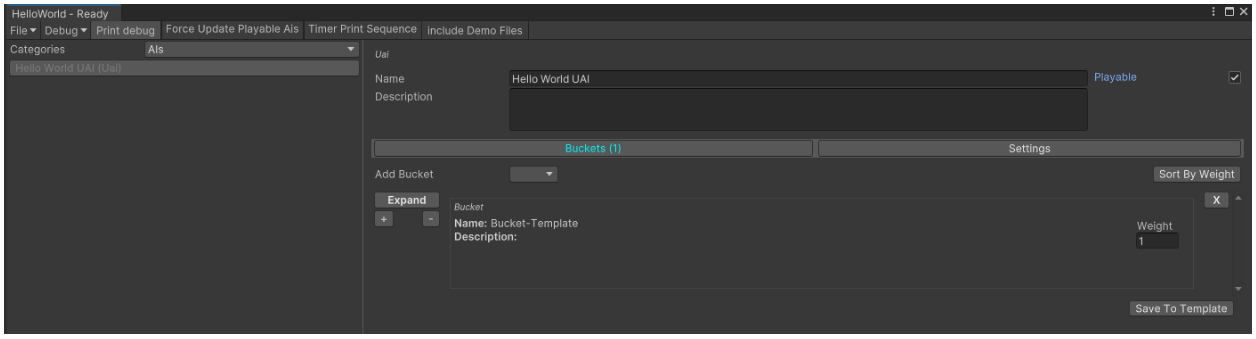
11. Expand the Print Txt action and input ‘Hello World’ in the ‘Text to print’ field. *You defined this by overriding the GetParameters() method in the code.*



12. In the toolbar: File -> Save and wait for the top left name to say: “HelloWorld – Ready”.
    a. If you lose your work, you can now reload you project by: File -> Reload Project



13. In the top-right corner Toggle ‘Playable’ On – Highlight this



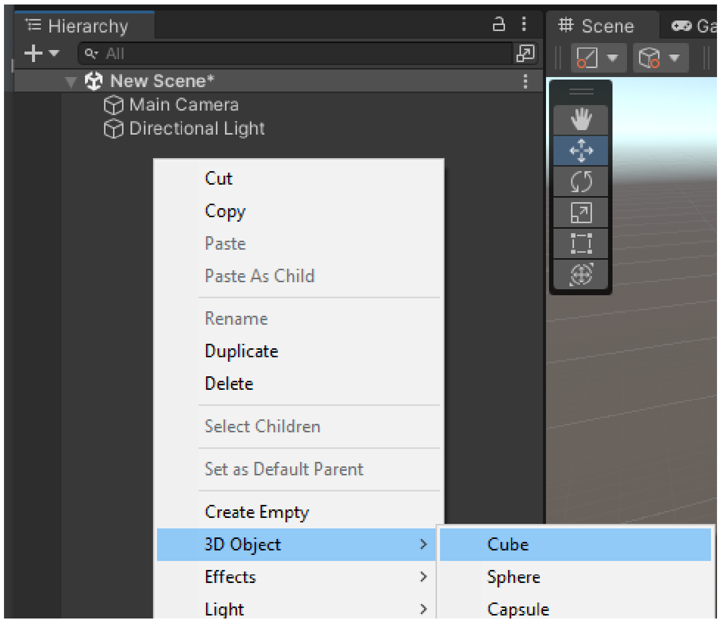
Name: Hello World UAI
Description:
Playable ✓

Name: Bucket-Template
Description:
Weight: 1

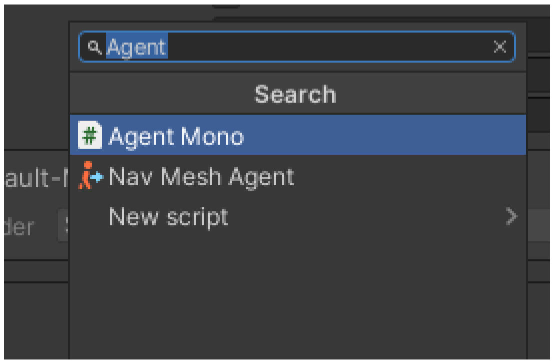
14. Close the Template Manager

1.2.3 Add an agent

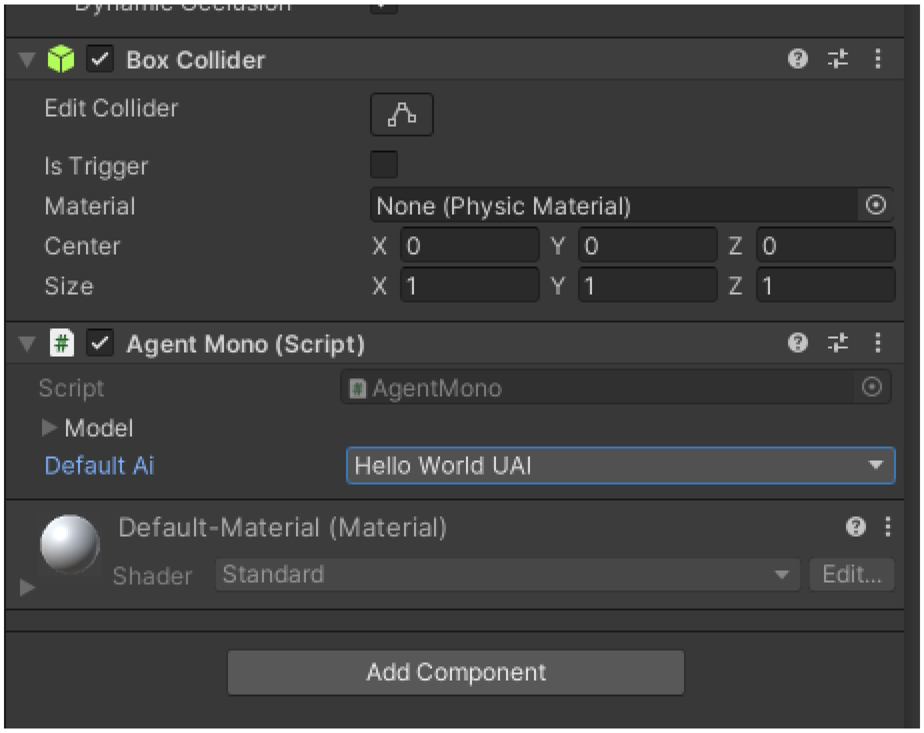
1. In the Hierarchy: Right Click -> 3D Object -> Cube and name it ‘Agent’



2. Select the ‘Agent’ and in its inspector: Add Component -> Agent Mono



3. In the Agent Mono component, set the default Ai to ‘Hello World UAI’



Box Collider

Edit Collider

Is Trigger

Material: None (Physic Material)

Center: X 0 Y 0 Z 0

Size: X 1 Y 1 Z 1

Agent Mono (Script)

Script: AgentMono

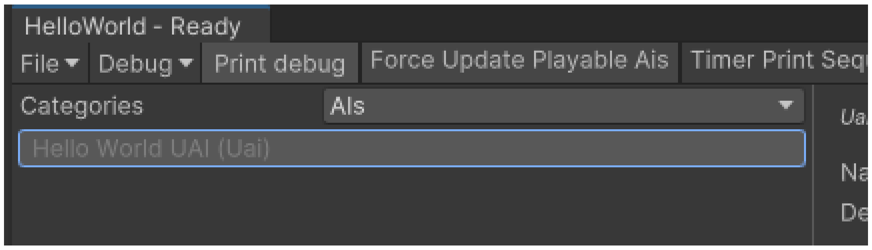
Model

Default Ai: Hello World UAI

Default-Material (Material)

Shader: Standard

a. If the AI name isn't shown return to the Template Manager and press 'Force update Playable Ais' – *This error can happen right after installing the package*



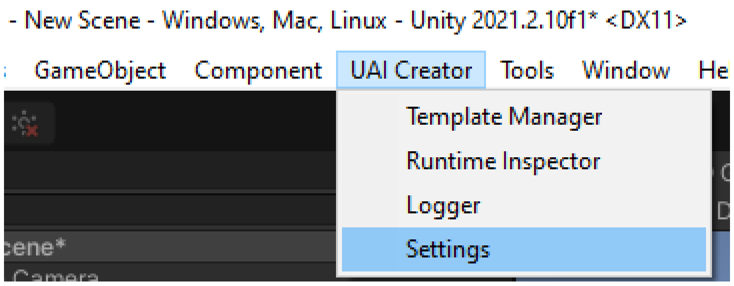
HelloWorld - Ready

Categories: AIs

Hello World UAI (Uai)

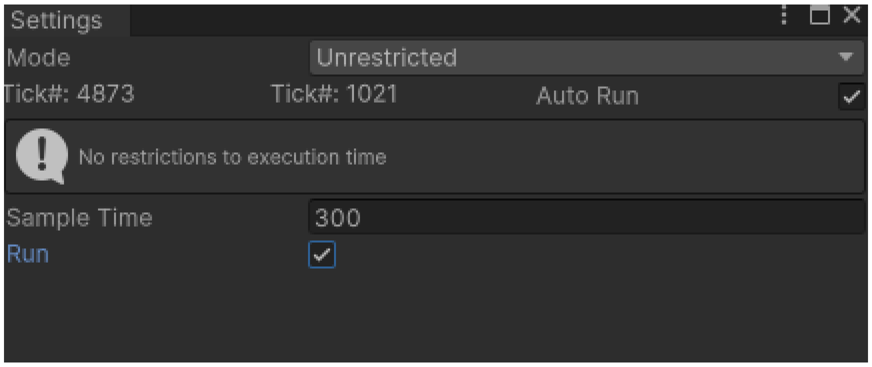
1.2.4 Finally

1. Run the game
2. In the toolbar: UAI Creator -> Settings. To open the UAI Settings



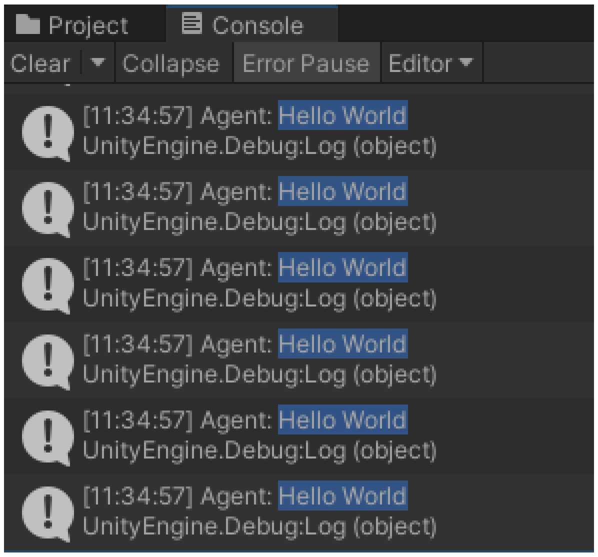
- New Scene - Windows, Mac, Linux - Unity 2021.2.10f1* <DX11>

3. Set the mode to 'Unrestricted' and toggle 'Run' on



```
Settings
Mode                Unrestricted
Tick#: 4873         Tick#: 1021         Auto Run ✓
No restrictions to execution time
Sample Time         300
Run                 ✓
```

4. The agent prints Hello World To the console every frame



```
[11:34:57] Agent: Hello World
UnityEngine.Debug:Log (object)
[11:34:57] Agent: Hello World
UnityEngine.Debug:Log (object)
[11:34:57] Agent: Hello World
UnityEngine.Debug:Log (object)
[11:34:57] Agent: Hello World
UnityEngine.Debug:Log (object)
[11:34:57] Agent: Hello World
UnityEngine.Debug:Log (object)
[11:34:57] Agent: Hello World
UnityEngine.Debug:Log (object)
```

1.2.5 Challenge

Make the agent:

1. Say: ‘Hi’ the first time the action is selected
2. Count the number of consecutive selections and print it to console
3. Change color on every selection *Hint inn the Template Manager press the ‘Include Demo Files’, to get access to more considerations and actions*

1.2.6 Challenge – Solution

1: Change the OnStart method in the PrintTxt method

2: Use the OnGoing method and a private property in the PrintTxt method

3: Add the change color action to the Hello World UAI

2 Overview

This chapter is meant to give you an overview of the UAI. You can read it all or skip to the sections representing the topics you wish to learn more about. After reading this chapter you will have a general understanding of, what was happening during the Hello World example.

2.1 The basics

A UAI is created by putting together multiple parts. At the top level you have the **UAI**, which consists of a collection of **Buckets,** each **bucket** holds a **collection** of **Considerations** and a collection of **Decisions**. Each **decision** holds a collection of **Considerations** and **Agent Actions**.

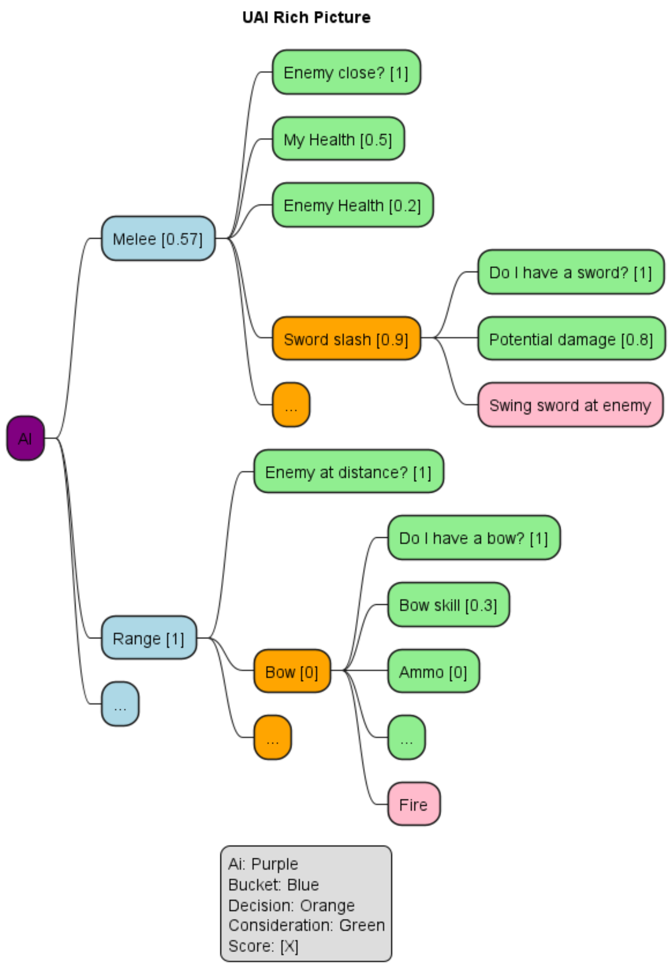


Figure 1 Overview of a UAI

The diagram on Figure 1 shows an example of a UAI, with its intended use. Every object has one responsibility, which is:

UAI: Used by an agent as a container for all UAI objects, including decision making objects (more on that later).

Bucket: Collection for a group of similar Decisions, which shares one or more considerations.

Decision: Holds Agent Actions and Considerations relevant to those actions.

Considerations: Reads data from the game world and converts it to a number (score) between 0-1, where a higher number represents the associated actions usefulness for the agent.

Agent Action: Executes code in the attached system / game.

2.2 UAI

The UAI consists of a collection of Buckets and 4 Decision Making Objects. By attaching a UAI to an agent, the agent will be able to select actions from within the UAIs buckets.

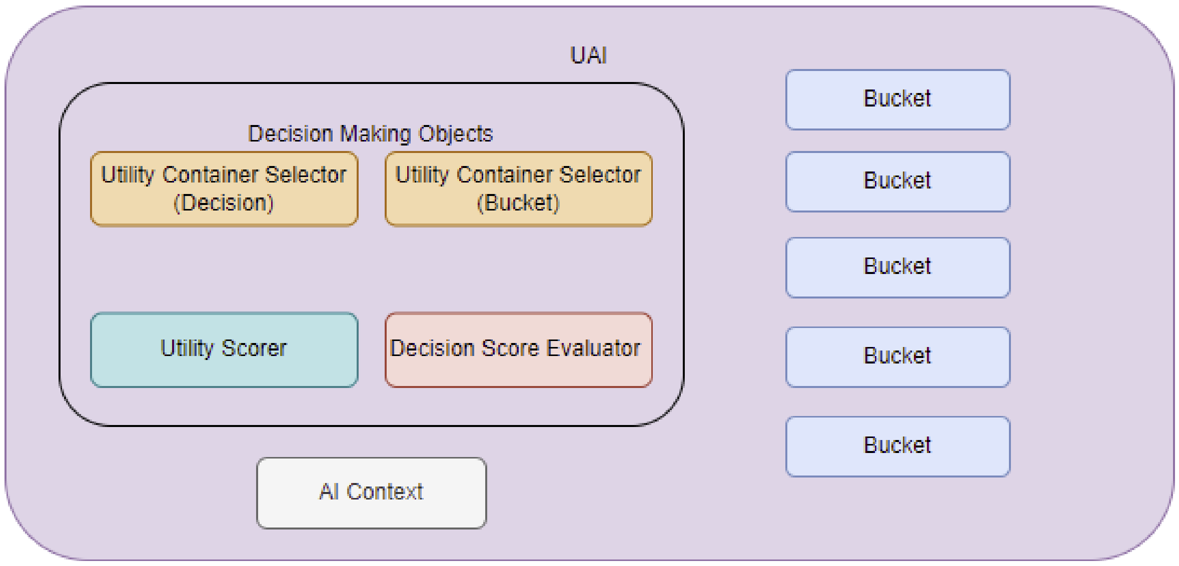


Figure 2 The UAI's components

2.3 Bucket

A bucket has a utility, a weight, a collection of considerations and a collection of decisions. The utility is calculated by the UAIs **Utility Scorer** and multiplied by the buckets weight. The decisions utilities are also multiplied by the buckets weight.

A bucket should be used to group all decisions which shares the same considerations, for example all ranged attacks, could be placed in the same bucket and a common consideration would be: "Do I have a ranged weapon".

If any of the bucket's considerations returns a score of 0, it indicates that all decisions in the bucket are invalid. I.e., if the agents don't have a ranged weapon, he shouldn't consider any ranged decisions.

The weight defines the priority of the bucket compared to other buckets, where a higher weight equals a higher priority, by increasing the likelihood of the bucket's decisions being selected, (due to a higher total utility). Help for defining weight values can be found in 4.1 - Weight.

The utility represents how much the agent would benefit from choosing this decision. The rule for calculating the utility is defined by the **Utility Scorer**.

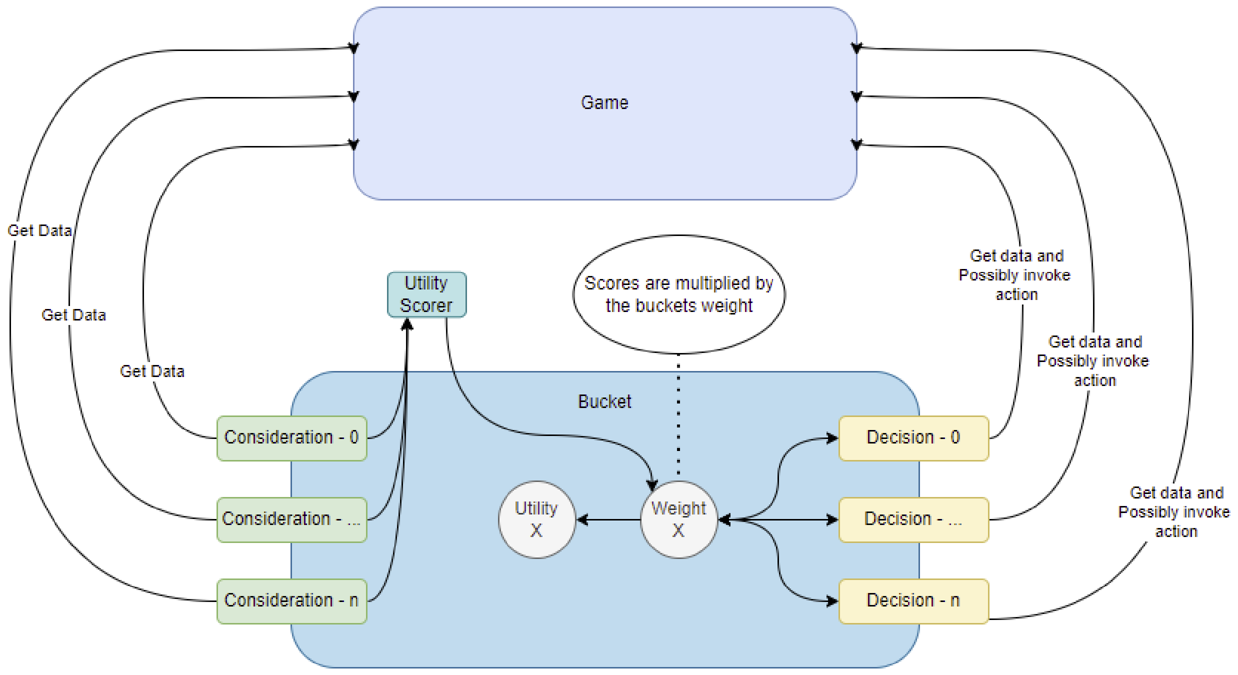


Figure 3 The Bucket's components and interactions

2.4 Decision

The decision has a utility, a collection of considerations and a collection of agent actions. The score is calculated using the parent UAI's **Utility Scorer** and is multiplied by the parent **Bucket's** weight.

A decision represents a specific thing an agent can do, like "Fire bow at target". The considerations should be relevant to the given decision. I.e., for the decision "Fire bow at target", the considerations could be: "Chance to hit target", "Potential damage to target" and "Desire to damage target".

The utility represents how much the agent would benefit from choosing this decision. The rule for calculating the utility is defined by the **Utility Scorer**.

The actions define the code, that must be executed in the game. Int the "Fire bow at target" example, the actions could be: "Animate fire bow", "Deal damage to target" and "Reduce ammunition". Another approach could be to use a single action called: "Fire bow at target", and let all logic concerning animation, damage, and ammunition be handled by the game, but this depends on the given use case.

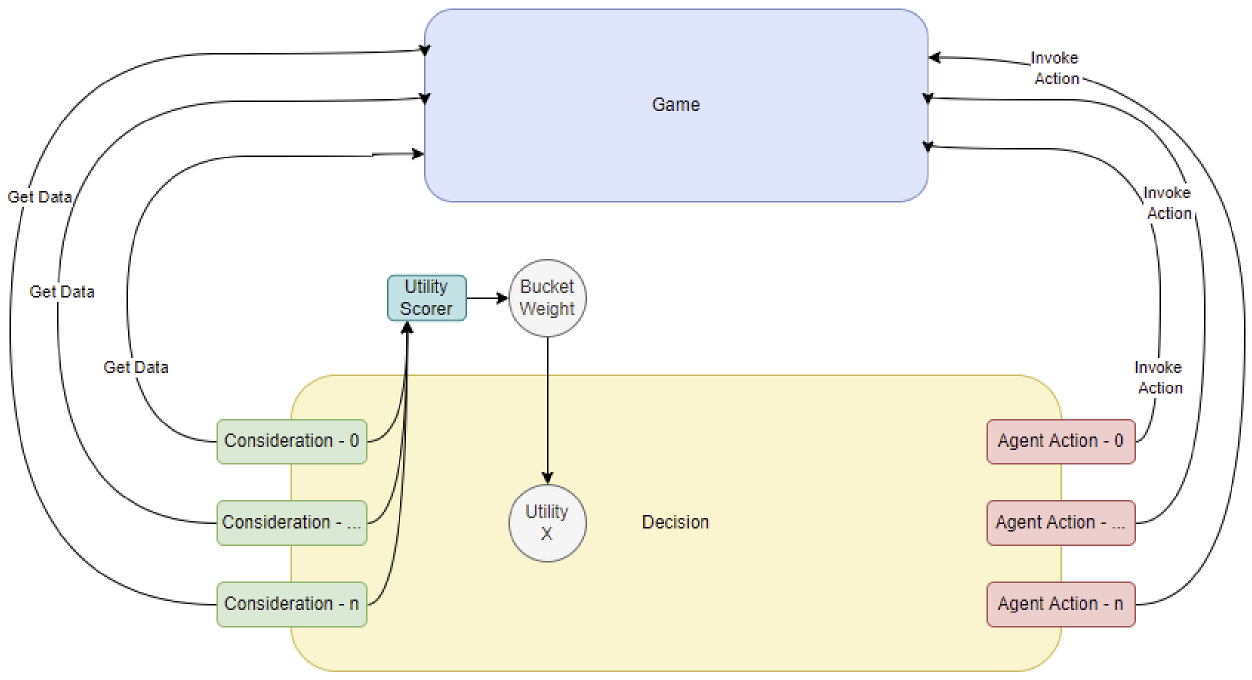


Figure 4 The Decision's components and interactions

2.5 Consideration

The consideration can be considered the “Input” of the UAI, as this is the place, where data is extracted from the game and converted to a normalized score between 0-1.

Some basic considerations are shipped with the UAI Creator, but in almost all cases, some user defined considerations need to be implemented.

The consideration uses a **Response Curve** to convert the game data to a normalized score.

A score of 0 means that the consideration is impossible, and that the associated decision/bucket should be invalidated. An example of could be “Do I have a ranged weapon?”, and it could be attached to a bucket containing ranged attacks. If the agent does not have a ranged weapon, there is no reason to consider ranged attacks.

The performance tag is set by the user and is used by the UAI Creator to determine the order of execution of considerations. For performance reasons it is desirable to calculate performance-light considerations before performance-heave considerations.

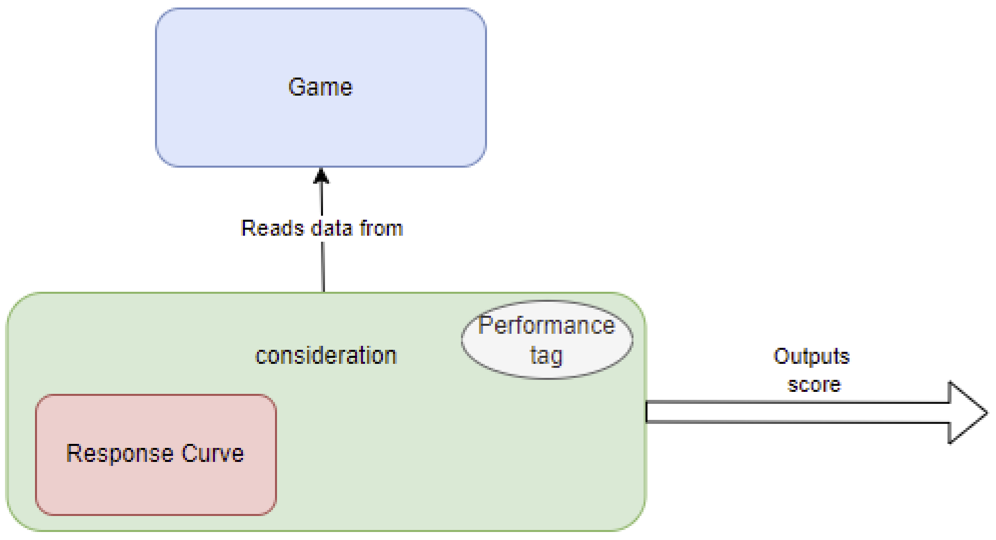


Figure 5 The Consideration's components and interactions

There are multiple variants of considerations. The description above applies to all types of considerations, except as stated below.

2.5.1 Modifier

Used to change the weight of the parent decision/bucket.

The returned score is not used for calculating the parents' score.

Example use case: The agent the decision "Heal Self", which is placed in a combat bucket with a weight of 5. If the agents' health drops below 5%, the "Heal Self" decision should be considered as life saving instead of basic combat. So, the Consideration-Modifier returns a score of 9 effectively raising the weight of the "Heal Self" decision.

2.5.2 Setter

Indicates, that the consideration saves data in the AI Context.

Setters are evaluated before any other considerations since other considerations might depend on them.

A use case is: For an agent to move he must consider: "Clear path to target?", "Time to travel path", and "Threats on the path". Individually these considerations would need to calculate the path to the target. Which probably is a performance heavy operation. To avoid this the "Clear path to target?" considerations, finds the path, and stores it in the AI Context, for the other considerations to use.

2.5.3 Boolean

Used for simple questions which could invalidate the decision.

The returned score is not used for calculating the parents' score.

The evaluation of consideration-booleans, are prioritized over basic considerations with the same performance tag, because these considerations have a high chance of invalidating the decision, thereby saving CPU time, by not evaluating redundant considerations.

Consideration-booleans aren't part of the parents score because their success value is always 1, which means, that a decision with many Boolean considerations would score higher than intended.

An example is: "Do I have a Ranged Weapon?", which has the potential to negate all ranged attack, but if the agent has a ranged weapon, doesn't say anything about the agents' utility of using the ranged weapon.

2.5.4 Code

When implementing a consideration, the CalculateBaseScore method should be overridden. This method should return a float, representing the considered game data. The response curve will then convert the returned value to a normalized score.

Other methods to implement could be: BaseScoreBelowMinValue and BaseScoreAboveMaxValue. These defines, what the consideration should return if the game data was outside the defined range.

GetParameter – Must be implemented. Defines the parameters for the consideration.

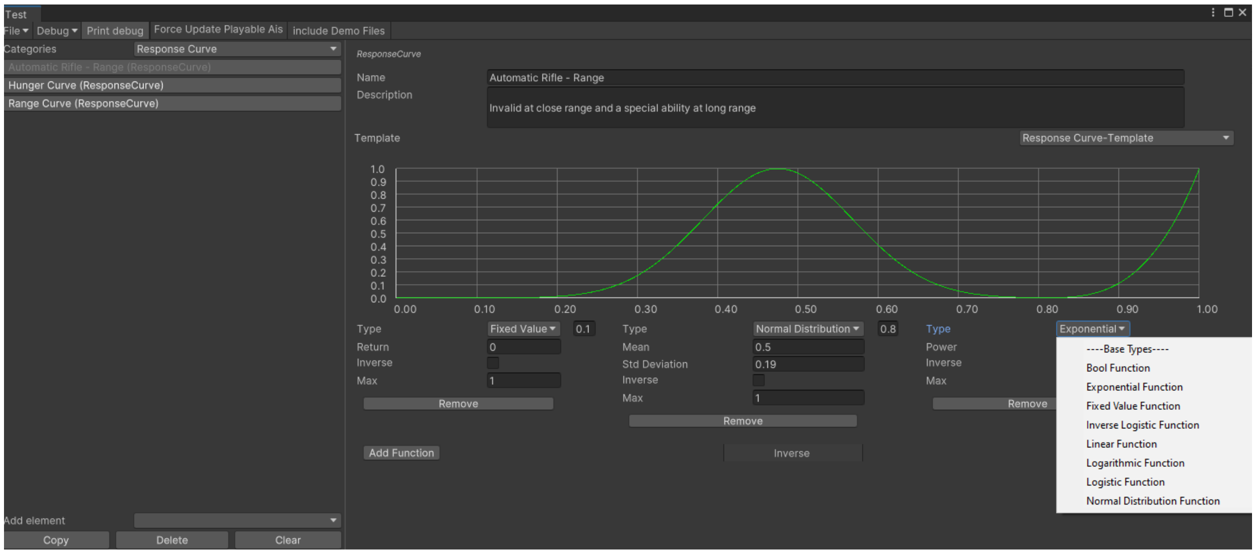
Sample code can be found in the folder: "*Packages/UtilityAiSystem/Runtime/Demo/Considerations*"

2.6 Response Curves

The Response Curve is used to define how game data should be converted.

In the example on Figure 6 Screenshot of the Response Curve UI, an agent has an automatic rifle. At close range the rifle shouldn't be used and returns a value of 0. The rifle is most effective at medium range, which is represented by the normal distribution between X=0.1 and X=0.8. Finally, the rifle has some special ability, which is very effective at long ranges.

You can create your own response functions by creating a class, that inherits from **ResponseFunction**.



Categories: Response Curve

Name: Automatic Rifle - Range

Description: Invalid at close range and a special ability at long range

Type: Fixed Value 0.1
Return: 0
Inverse:
Max: 1

Type: Normal Distribution 0.8
Mean: 0.5
Std Deviation: 0.19
Inverse:
Max: 1

Type: Exponential
Power
Inverse
Max

Figure 6 Screenshot of the Response Curve UI

2.7 Agent Action

The agent action can be considered the “Output” of the UAI.

The UAI Creator comes with some Unity specific Agent Actions, but in most cases, they should be implemented by the user.

2.7.1 Code

The following functions should be used, when implementing an agent action.

OnStart – Executed the first time an action is selected.

OnGoing – Executed on each consecutive selection.

OnEnd – Executed when the action was selected during last tick, but not selected this tick.

GetParameter – Must be implemented. Defines the parameters for the agent action.

Sample code can be found in the folder: “*Packages/UtilityAiSystem/Runtime/Demo/AgentActions*”

2.8 AI Context

Used to share data between UAI Objects.

All UAI Objects has read/write access to the context.

2.8.1 Code

The relevant fields and methods are:

object GetContext(object key, AiObjectModel requester), the key specifies, which data to access. The requester is the requesting UAI Object. If the requester is null, data will be searched from global available data. If the requester is a UAI Object, data will be searched in the context of the requester, all its ancestors and globally.

Void SetContext(object key, object value, AiObjectModel owner), sets a value for a given key, and specifies the owner of the data. If no owner is set the data is stored globally. Define the owner when you want the data to be shared between descendants.

LastSelectedDecision, LastSelectedBucket, CurrentEvaluatedDecision and CurrentEvaluatedBucket are used to access the bucket and decision who “won”, the last tick, and the bucket and decision, which is currently being evaluated.

2.9 Decision Making Objects (DMO)

Decision Making Objects, references the objects, which in union handles the decision-making logic.

2.9.1 Decision Score Evaluator (DSE)

The DSE is the main DMO and serves as an interface to the underlying DMOs. The DSE is responsible for the continuous evaluation, until the best set of Agent Actions has been found.

In most cases you should forget about this one and leave it as it is.

2.9.2 Utility Container Selector (UCS)

Any UAI has two UCS's one for selecting buckets (Bucket Selector) and one for selecting decisions (Decision Selector). They can be changed from the UI (Template Manager).

There are many ways to select the next utility container (bucket or decision). The UAI Creator comes with the following UCS options:

Highest Score: Selects the highest scoring utility container.

Random X Highest: Select a random utility container among the X highest scorers. Can be set to select a random utility, where the chance of being selected is relative to the score (Percentage chance). Can set a max allowed deviation from the highest scorer, so only decisions which appears logical to the user is considered.

You can add your own UCS by creating a class that inherits from UtilityContainerSelector and implementing the abstract methods.

Sample code can be found in the folder:
"*Packages/UtilityAiSystem/Runtime/Services/Scorers/UtilityContainerSelector*"

2.9.3 Utility Scorer

Is responsible for calculating the utility from a set of considerations. Currently the UAI Creator comes with the following utility scorers:

Any utility scorer returns 0 if a considerations score is <= 0.

Average scorer: Returns the average score of all considerations for a given utility container.

Highest score: Returns the highest score of all considerations for a given utility container.

Lowest score: Returns the lowest score of all considerations for a given utility container.

You can add your own Utility Scorer by creating a class, that implements the IUtilityScorer, but remember:

1. Return 0 if any consideration returns <=0.
2. Only use the score from considerations which are scorers
    a. Use Consideration.IsScorer to check this.

Sample code can be found in the folder:
"*Packages/UtilityAiSystem/Runtime/Services/Scorers/UtilityScorer*"

2.10 AgentMono

The AgentMono component allows monobehaviours to use the UAI. Attach it to a monobehaviour through the Unity Inspector and set the settings as you wish.

The settings are:

Default Ai: Sets the default AI to be selected on game start (Identified by name). If an error occurs, and the UAI can't be found at game start, the first playable UAI will be selected.

Auto Tick: If enabled the agent automatically makes decisions. This should be used for most games, that isn't turn based or requires an agent to only act as a response to an event.

Ms Between Ticks: Set the number of milliseconds that must have passed before the next decision.

Frames Between Ticks: Set the number of frames that must have passed before the next decision.

The … between ticks are equal to a logical AND, meaning, that both must be true before a new Tick can be selected.

Use the … between ticks, settings to improve performance, by limiting the resources spent on decision making for an agent.

This is very use case specific. In an FPS or racing game the user might expect the agents, to react instantly to the users' inputs, and these should be set very low. In other games it might seem weird, if all agents act instantaneously to changes in the game world.

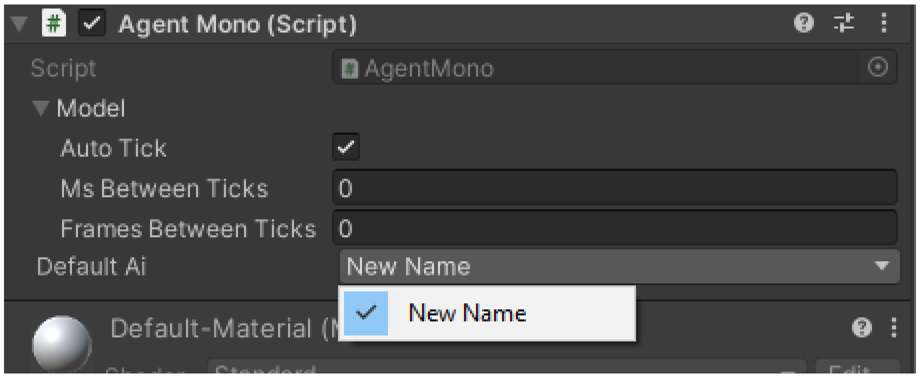

Agent Mono (Script)
Script: AgentMono
Model
Auto Tick: ✓
Ms Between Ticks: 0
Frames Between Ticks: 0
Default Ai: New Name

Figure 7 Screenshot of the Agent Mono in the Inspector

2.11 Parameters

Parameters can be defined and used in the UI for almost all UAI Objects. Do this by overriding the GetParameters() method, and return a list with the parameters you wish to use.

There are currently two types of parameters, these are:

Parameter:

- Supports the following types of values:
    - Int, float, string, long, bool, UnityEngine.Color

ParameterEnum:

- Use this if you need an enum as value.

To define a parameter simply: new Parameter(*parameterName, someValueType)*. I.e., new Parameter("Health Param", 3.4f).

The UAI Creator registers the value type and applies the correct UI component, where needed.

To use a parameter from code use: GetParameter(*parameterName*).Value

The value is returned as an object so remember to convert/cast it to the correct type.

ATTENTION: During serialization all floats are stored as doubles and after the first load stored as doubles in the parameters value field. Because of this you must use Convert.ToSingle(*parameterValue*), when dealing with floats.

ATTENTION 2: Because the parameters are saved as objects, there is a lot of boxing and unboxing going on, whenever the parameter values are being read or written. Which means, that a way to high amount of memory allocation takes place.

PROMISE: This solution was made, to uphold a deadline, but fixing these problems is one of the highest priority items in the pipeline, and it will be fixed no later than version 1.10.1.

3 Advanced

4 Examples

4.1 Weight