

Welcome to Wise Feline

Wise Feline is a utility AI system. Wise Feline allows you to make an AI which feels unscripted and immersive and allows you to do this in a fraction of the time it takes to do AI in a scripted and sequential manner. Wise Feline uses Utility theory and by extension the Utility AI approach championed by Dave Mark in the games industry which is used in games as big as the sims and guild wars 2.

Philosophy

An AI system usually needs to perform two functions, listing the actions which can be taken and choosing an action from the list to execute. Utility AI does this by giving all possible actions a score between 0 and 1 and then chooses the action with the highest score.

Each action has a set of considerations which estimate the action's utility at the moment. Each consideration has a score between 0 and 1 based on the current parameters supplied to it. Each action's score is calculated by calculating the score of all of its considerations and then multiplying them by each other.

This is how humans take decisions. For example if you have actions below:

- Buying food
- Eating food
- having a rest

Then the eating food action probably would have two considerations for calculating if the agent is hungry or not and also calculating if the agent has food or not. The consideration for being hungry is a continuous value between 0 and 1 but the having food available consideration is a binary which is either 0 or 1. The score of the action is calculated by multiplying these two so when the agent doesn't have food, the score for eating food will always be 0 since you don't have any food to eat.

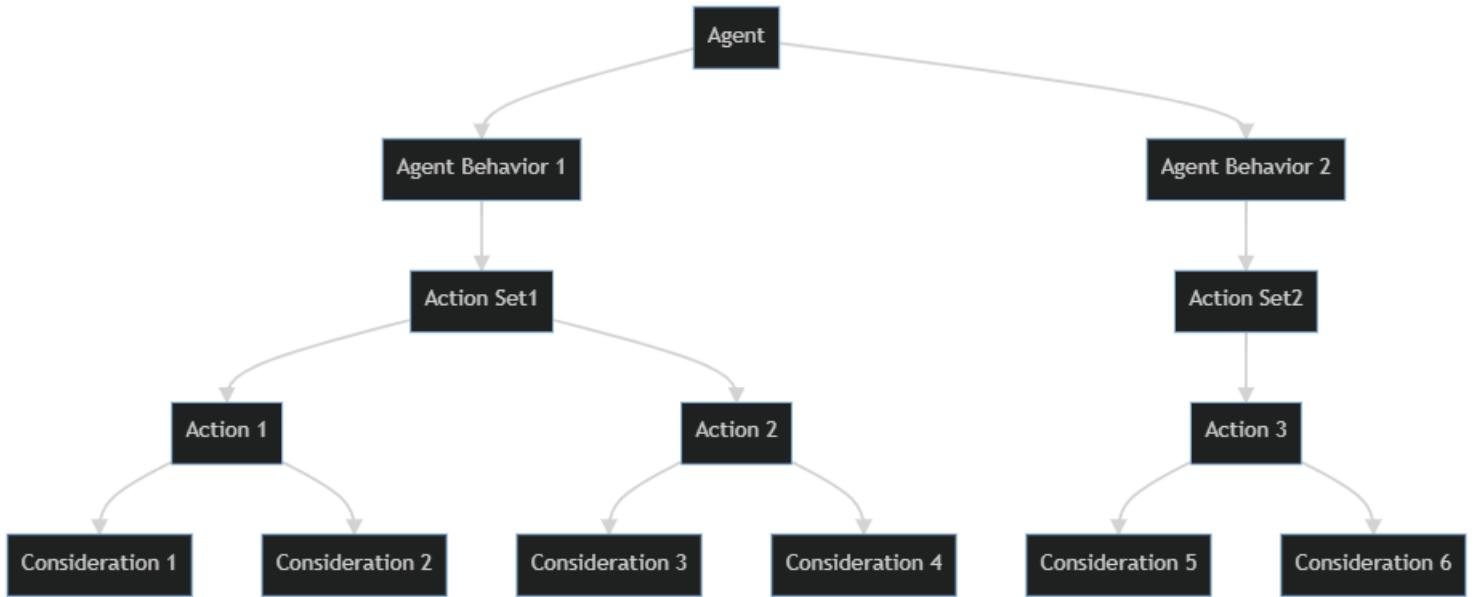
Buying food on the other hand has considerations for not being tired and not having food and being hungry and these should return their values in a way that if you are very hungry, the score becomes high even if you are tired. Resting action probably should only give a high score if you are tired but not that hungry.

Each consideration is a function of a curve which its x axis is the value that we read from the agent/environment and its y axis is the output score of the consideration. So for hunger we read the hunger value of the agent and return it and then its value is given to a curve as x axis and the resulting y axis is the actual score of the consideration. In this way we can make a consideration more sensitive to certain values so for example having a hunger of 0.5 can return 0.5 in the case of a linear curve and 0.9 or 0.1 in the case of custom curves.

These curves for considerations are the main way that designers make the agent react the way they want. For example if you want the agent to go find some food and eat no matter what if the hunger goes above 0.9, you add a consideration to buying food and having food which gives the score of 1 back if hunger is above 0.9 and you might even give a reverse consideration to actions which are not important when you are hungry so as hunger goes up it returns a lower score. The same hunger consideration but with different curves can be used for the purpose in different actions.

How the system works

This is the architecture diagram of the system



Any game object which wants to be controlled by utility AI should have a [brain](#) component attached. This component executes the main utility AI logic by executing all action considerations once in a while and choose a high scoring action to do. Then it calls callbacks of the chosen action which can drive the agent in the world. The action code has access to components of the game object because the brain component of the game object is passed to it in its [`OnInitialized\(\)`](#) callback.

[Actions](#) which an agent can do are organized inside [action sets](#) and have their own considerations listed under them with custom curves per action. A group of action sets themselves are stored in an asset

called [Agent Behavior](#) which is what you attach to a brain to use.

So the root of the tree is the brain component which has an Agent Behavior asset attached and then each agent behavior contains action sets as its children/branches and each action set contains a set of actions which contains their own considerations.

You define [actions](#) and [considerations](#) by deriving classes from [ActionBase](#) and [ConsiderationBase](#).

To create each agent's AI you:

- Create an [Agent Behavior](#) by right clicking in the project view and choosing Create>NoOpArmy>Wise Feline>Agent Behavior.
- Name it whatever makes sense.
- Select the agent behavior asset you just created.
- Open the [Wise Feline window](#) by going to Window>NoOpArmy>Wise Feline window.
- Now add actions and considerations and action sets to the behavior.

Different pages of this manual describe each of these items in detail. Read them and come back to this for the bigger picture again to understand it well. There is also a [demo](#) which helps you by showing how to implement a semi-realistic sample.

Considerations

[Considerations video tutorial](#)

Each action's score is calculated by calculating scores of all of its considerations and then multiplying them with each other. Considerations are defined in code by deriving from `ConsiderationBase` class. Like components each consideration should be in a file which shares the same name with the consideration class.

After defining a consideration in code, you can add it to actions of a Agent Behavior asset in the wise feline window.

To add considerations to actions

- Implement a consideration in a class. For example create a `HungerConsideration.cs` file and implement `HungerConsideration` inside it like this. The example code is posted below.
- Select the desired agent behavior asset and open the Wise Feline window at `Window>NoOpArmy>Wise Feline`.
- Select the action which you want to add the consideration to by clicking on it. You might need to select the appropriate action set first.
- Now in the considerations list for the action click Add Consideration and from the context menu choose `HungerConsideration`
- Set the curve and the parameters of the consideration as you desire.

The consideration can be defined like this:

```
public class HungerConsideration : ConsiderationBase
{
    private CatBehavior behavior;

    protected override void OnInitialized()
    {
        base.OnInitialized();
        behavior = Brain.GetComponent<CatBehavior>();
    }

    protected override float GetValue(Component target)
    {
        return behavior.Hunger;
    }
}
```

`OnInitialized()` is called when the consideration is initialized after the `Brain` component is created or the agent behavior asset is added to a brain at runtime.

`GetValue()` is called whenever the action wants to calculate its score.

What you return in `GetValue` should be a number between 0 and 1 (both inclusive) which will be fed to the curve of the consideration as the value for x axis and the y axis value at that point is the score of the consideration used in score calculations.

For more information, see the API reference for the [ConsiderationBase](#) class.

Actions

[Actions Video tutorial](#)

Actions are the most important part of a utility AI system alongside considerations. An action is a concrete and most of the time atomic unit of work which an agent can do. Shoot, Find health pack, Find cover, Take cover in the found cover, eat food and Choose the best camera to shot the scene with are all examples of actions. However an action doesn't have to be low level and can be pretty high level, you can have a hierarchy of them in your game and for example have a set of actions for groups of NPCs. They can be things like: attack the target, spread members in the room, defend sensitive points, find weak spots to bombing every place and then some NPC actions which take command from the group so when the group chooses to attack the target, a member NPC might execute its shoot action or move toward target action or even cover the attackers action.

To create an action and add it to an action set in an agent behavior asset:

- Create a class derived from [ActionBase](#).
- Name the action class and its file the same.
- Select an agent behavior asset and open the *Window>NoOpArmy>WiseFeline* window.
- Select an action set and then click add action and choose the action you created.

Here is an example action for going toward a food which should be saved at [GoTowardFood.cs](#)

```
public class GoTowardFood : ActionBase
{
    private CatBehavior behavior;

    protected override void OnInitialized()
    {
        base.OnInitialized();
        behavior = Brain.GetComponent<CatBehavior>();
    }

    protected override void OnStart()
    {
        base.OnStart();
        behavior.GetComponent<MeshRenderer>().material.color = Color.cyan; //for
        visualization purposes only
    }

    protected override void UpdateTargets()
    {
        ClearTargets();
        AddTargets(FoodManager.instance.foods); //FoodManager has the list of all foods in
    }
}
```

```

the world
}

protected override void OnUpdate()
{
    base.OnUpdate();
    behavior.IncreaseHunger(0.01f * Time.deltaTime);
    behavior.IncreasePlayfulness(0.05f * Time.deltaTime);
    var foodPos = ChosenTarget.transform.position;//chosenTarget is the best food target found by scoring
    behavior.MoveTowardPosition(foodPos);
}
}

```

There are several things to note:

First there are a set of callbacks in an action which you can override. `OnInitialize()` is called as soon as the action is added to the brain or in the first frame if the behavior is in the brain when the game object is created. `OnUpdate`, `OnLateUpdate`, `OnFixedUpdate`, `OnStart` and .. are called in the same `MonoBehaviour` callbacks from the brain.

Secondly there are two types of actions, targeted and none-targeted. Targeted ones are those which work on a target like shoot and eat pack. This action is targeted. Non-targeted ones are those which don't have a target like idle or useHealthPackFromInventory. To make an action targeted, simply check `NeedsTarget` checkbox in at least one of its NPCs. To learn more about this check the [targetting system](#) page.

To say more about targetting, If the action is targeted then the `chosenTarget` field has the best target in it which you should shoot at, eat or move toward or whatever which is right for your action. An action becomes a targeted one if at least one of its [considerations](#) are targeted and have the `NeedTarget` checkbox checked in the Wise Feline window. A targeted action means all considerations are calculated for all targets which you gave to the action in the `UpdateTargets()` callback and the one with the highest score is in the `chosenTarget` field. `UpdateTargets()` is called once per update target operation which its frequency is chosen in the `brain` component. The scores are calculated once per think operation which is again their frequency is chosen in the brain component. Here you just define how the action will execute and that's it.

As you can see, to access the component attached to your game object, you can get them from the `brain` component of your game object which is passed to you in `OnInitialize()`. When filling the list of targets, you can fill it with whatever component type and you can find them using any method you want. We have a set of utility methods in the [Search](#) class but you don't have to use them and are not encouraged to do so either. They are simple wrappers on top of unity's raycasting functionality which work fine for simpler games and samples but for games with huge worlds and lots of agents, you should

probably use custom data structures. We will probably release helper modules for these later on. You don't have to clear and rebuild the list of targets every time either but in that case you should be careful to not add something multiple times and also to remove objects which are not good target candidates as well. Also if your action does not need any targets, You don't need to add any targets in `UpdateTargets()` but still have to implement an empty one.

For more information, see the API reference for the [ActionBase](#) class.

Targetting system

Actions can have targets but how the targetting system works? If an action has one or more considerations which their **NeedTarget** field is set to true in the behavior designer UI then the action is an action with targets.

When the action gets calculated to have a score, each consideration which has its **NeedTarget** checked, gets calculated per target and the target will be sent to it. the rest of the considerations will be calculated only once for the action. Now the action has a list of targets and their scores and it will use the target with the highest score as its target and calculates its score by multiplying the score of that target with the score of considerations which did not have targets.

So if the Shoot action and GetHealthPack action both have 3 targets. Shoot action's final score will be like 0.96 and it has a target selected which is the target which caused the action to have the highest score and the GetHealthPack action will have say a score of 0.72 and it is the highest score it could get with the best target score. There might be a target for Shoot which it is far enough that the shoot action for that target would score 0.34 and lose to GetHealthPack.

How to choose which consideration should have targets?

Considerations which need to calculate something regarding the target you are considering should have **NeedTarget** checked in the designer UI. The distance to target consideration or what's the target's health consideration are two good examples. However the considerations which calculate something about yourself like how much ammo do I have don't need **HasTarget** set and will be calculated once for the agent itself.

How can I fill the list of targets in Update Targets()?

You can fill this in the action by using any mechanism which is good for your game. You can use raycasts, overlap functions, influence maps, AI Tag system or any custom system to find the targets. When filling this in, be careful to not give every possible object as a target and instead count the objects which can be potentially good targets in and keep the rest out. Calculating each targetted consideration per target takes time and you don't want to add objects which you already know will not be good targets. A melee action would not want anything far away to be added for example. If the action was rush toward enemy to do melee attack, you might want to include far away targets but if you want to just do the attack and you know it doesn't work more than 3 meters away, then probably doesn't make sense to include anything more than 3-4 meters away.

There are a few methods like `AddTarget()` `ClearTargets()` and `RemoveTarget()` which help you to fill the list of targets. The samples in the plugin have example actions which do this.

Built-in actions and considerations

Wise Feline contains a set of built-in [actions](#) and [considerations](#) which help you when prototyping things quickly or even building the real AI of your game. These can be found in the ActionAndConsiderationLibrary folder in the UtilityAI folder of the package.

Design philosophy

Built-in actions and considerations work on the assumption that you put all the data the AI needs in the blackboards of the GameObjects participating in the AI. The objects might be something in the environment which you can interact with or an agent or a player controlled object.

Sometimes the data might be stored as a set of tags if it is static info about the object or can benefit from spatial queries like give me the objects with this property in my 100 meters radius.

For these assumptions we have actions and considerations which can read/write blackboard data, tags and smart objects and we allow you to search influence maps without writing code. Also you can write into influence maps without writing code using the [InfluencerAgent](#). You can create influence map views which are multiple maps combined by math operations without writing a single line of code.

We even have some animation integration which works with the built-in actions and considerations and answers some of your needs.

This all means that you can easily prototype and even complete your AI by writing a bit of code to integrate animations or doing very specific or performance sensitive behaviors. To prove the points above, we built the more complex life simulation demo with less than 10 custom actions and considerations combined and even those were not that much code and mostly added some visual effects/animations to the built-in ones.

Movement components

The movement actions need to move agents because without movement we could not make that many actions :) That however needs movement to be abstracted a bit so your agents don't have to follow our code too much. So we made multiple movement component which you can choose from.

The different ones implement an interface called [IAIMovement](#) so the actions don't differentiate between an agent which uses a [NavMeshAgent](#) component or a [CharacterController](#) or any other way to move.

The interface has two methods to move to a position and one for stopping to move. They are supposed to be called only once and then the movement component will move the agent there. The Methods aren't supposed to be called every frame. The movement methods do the same thing but the different overloads take the destination position by different ways. You can give a [GameObject](#) to this and ask to move to its position or give it a [Vector3](#) and ask to move to it. Feel free to use them in your own actions too.

These are the components so far.

CharacterControllerBasedAIMovement

You can attach this component to an agent to move it using built-in actions if the agent has a CharacterController component.

NavmeshBasedAIMoement

You can attach this component to an agent to move it using built-in actions if the agent has a NavMeshAgent component.

Animation integration

To integrate animations with your AI actions, your actions need to set animator's parameters so it knows what action is executing. We have a parameter in built-in actions which allow you to set a trigger when the action starts. We also allow you to set a velocity parameter in movement components to make them more useful in real-world scenarios and also to teach you how to potentially integrate more complex animation scenarios.

List of built-in Actions

We have created a group of actions which are general enough which can simulate many of the actions you need for your games. You will need to inherit from them to apply animations and other custom things but they are a good starting point to create your AI. We will make them more powerful as time goes.

Most of our actions inherit from [BlackboardActionBase](#) which allows you to modify the agent's blackboard over time.

BlackboardActionBase

You will not use this action on its own most of the time but our other actions inherit from it.

This action has a [BlackboardChanger](#) which allows you to apply a change to blaockboard values for the agent over time. The changer can apply a change once or multiple times with delay or every frame. It can even multiply the every frame changes of type float by delta time. The action has a boolean to reset the changer in [OnStart](#) which effectively is whenever the action is selected. You can also apply a delay at startup before starting the blackboard changes.

BlackboardActionWithSmartObjectTarget

Finds potential targets from smart objects and optionally claims and uses them. It also frees them in OnFinish if the claiming and using is switched on. Also the action will fail if it tries to claim/use the smart

object unsuccessfully.

This action doesn't execute any animations or movement and just tries to find targets using the smart object search parameters specified and then optionally claim and use the chosen target in `OnStart()` (when the action starts executing) and then free the chosen target in `OnFinish()`.

The search parameters are the smart object tags which the object should/should not have, the radius and the slot states of acceptable slots.

Read on [smart objects](#) for more information on how the search functionality works but basically you can say I want my object to have one of these tags and don't want it to have these other tags and you can also say if you want it to have any slot owned by you or any free slots or you don't care about the slots.

- You need at least one consideration with `NeedTarget` set to true to evaluate the targets of this and choose the best one.

BlackboardActionWithTagTarget

This action updates its targets based on AI tags system and executes blackboard changes which you choose. The action doesn't do anything with the targets but it is useful because it will not execute if one of the targets don't get a high enough score for this action to win in the brain as the selected action.

For example: Using this action and correct properties you can increase the wormth key of the blackboard if the character is in proximity of any object with the fire tag. You set the radius to say 5 and add fire to include tags and add a `TargetDistance` consideration which chooses the target if it is close enough and maybe add a `BlackboardFloatConsideration` to only choose it when you have low wormth. There are actions to move toward fire as well so don't worry about it.

- You need at least one consideration with `NeedTarget` set to true to evaluate the targets of this and choose the best one.

CreateGameObjectAction

This action creates gameobjects, can be used for shooting, providing loot and other similar things

It asks for the prefab of the object to create and the number of them needed and if it should create one after each delay or they all should be instantiated at once.

Also a checkbox `shouldAdaptBrainRotation` which if set then the object inherits the rotation of the agent creating it which is good for bullets and alike.

DestroySelfAction

Destroys the GameObject executing the action

This action takes a delay as a parameter and if the action is switched off before the delay finishes, the destruction will not happen.

DestroyTargetWithTagAction

Destroys the target object, can be used for killing, eating and similar actions.

The targets are found using the [tag system](#).

The action takes the radius of the search and uses the agent's position as the center. The target should have at least one of the included tags and none of the excluded ones.

You need at least one consideration with NeedTarget set so it can evaluate targets to see if they are a good fit to destroy or not

Movement actions

These actions don't work if a movement component is not attached to the agent. See above for the list of them. The movement speed is whatever is set on the NavMeshAgent or any other movement component you attached to the agent.

MoveAndPatrol

Patrols the agent on the way points provided as children of a GameObject which is set on a blackboard key. Optionally can finish the action after one full cycle finished.

MoveToGameObjectAction

Moves the agent to the position of a GameObject set on a blackboard key. The action succeeds after it reaches the destination and fails if the movement method fails to move (returns false).

MoveToVector3Action

Moves the agent to the position of a Vector3 set on a blackboard key. The action succeeds after it reaches the destination and fails if the movement method fails to move (returns false).

MoveToTargetWithTag

Makes a list of potential targets using the AI Tag system and moves toward the chosen target if the action is selected to execute.

Needs a consideration with NeedTarget set so targets can be evaluated and chosen.

MoveToSmartObject

Finds a set of smart objects as potential targets and if the action gets selected to execute then it moves toward the chosen target.

The search criteria can be specified for slots of the smart object to make sure we own a slot in the object/a free slot can be found. You can also choose none.

Needs a consideration with `NeedTarget` set so targets can be evaluated and chosen.

NoopAction

This action is our favorite and doesn't do anything. It optionally can stop movement too.

It can be used in cases that you only desire the agent to stop moving and probably change some blackboard attributes. Let's say you want to cancel a movement action which is moving the agent somewhere and you don't want anything to happen after cancellation. This is the action for those times. This was the programming job of the life sim demo.

List of built-in Considerations

Some considerations have specific requirements and some of them might only work with `NeedTarget` on or off. Also some considerations might return binary 0 or 1 values and some return a value between 0 and 1 and are continuous. Min and max range values matter for some of them as well.

These are all mentioned per consideration.

In a binary consideration, the values of your consideration curve between 0 and 1 don't matter because it always returns either 0 or 1.

And as said in the considerations page, if `NeedTarget` is false then the agent itself is sent to `GetValue` as target and if it is true, all targets of the action are sent to it one by one to get a score.

AITagConsideration

This consideration checks if the target or the executing agent has or doesn't have a specific tag. It takes a tag and also an operation enum which you can set to the desired tag and operation.

The `ShouldHaveTag` operation causes the consideration to return 1 if the object has the tag and 0 otherwise. The `ShouldNotHaveTag` operation returns 1 if the object does not have the tag and 0 otherwise.

- This consideration is binary.
- This consideration works both for targets and the executing agent.
- The target of the consideration should have the `AITags` component for this to work.

BlackboardBoolConsideration

Checks a boolean key in a blackboard. This returns 1 if the key exists and is true and if it doesn't exist or is false returns 0.

- This consideration is binary.
- This consideration works both for targets and the executing agent.
- The target of the consideration should have the **BlackBoard** component for this to work.

BlackboardFloatConsideration

Checks a float key in the blackboard and if it exists, returns its value, otherwise returns 0.

You can invert the value of this consideration and the value is always normalized between min range and max range.

- This consideration is continuous.
- This consideration works both for targets and the executing agent.
- The target of the consideration should have the **BlackBoard** component for this to work.

BlackboardGameObjectDistanceConsideration

Returns the distance between the executing agent and a GameObject which is specified in a key in the blackboard.

You can invert the value of this consideration and the value is always normalized between min range and max range.

For example, if you want distances less than 20 meters to be considered then you should set max range to 20. Any value larger than 20 will be normalized to 20 and returns 1 and 10 will return 0.5 and 0 will return 1. You can invert the results if you want lower distances to return values closer to 1 and higher ones move toward 0.

- This consideration is continuous.
- This consideration works both for targets and the executing agent.
- The target of the consideration should have the **BlackBoard** component for this to work.

BlackboardIntRangeconsideration

Checks an integer key in the blackboard and returns its value. If the key doesn't exist then it returns 0.

You should set the range to values which indicate the possible values of the int. if the int can be 1, 2 or 3 then the min range should be 1 and max range should be 3 and then 1 will return 0, 2 will return 0.5 and 3 will return 1. You can invert the result with the checkbox if needed as well.

- This consideration is continuous.

- This consideration works both for targets and the executing agent.
- The target of the consideration should have the **BlackBoard** component for this to work.

BlackboardIntValueConsideration

This consideration checks an int key in the blackboard and returns 1 if the value equals the expected value; otherwise, it returns 0.

- This consideration is binary.
- This consideration works both for targets and the executing agent.
- The target of the consideration should have the **BlackBoard** component for this to work.

BlackboardConsiderationKeyExists

This consideration returns 1 if the specified key name exists and otherwise returns 0

- This consideration is binary.
- This consideration works both for targets and the executing agent.
- The target of the consideration should have the **BlackBoard** component for this to work.

ConstBoolConsideration

Returns a constant value. This consideration is mostly used for testing.

If `isTrue` is set then the `trueValue` is returned, otherwise the `falseValue` is returned.

- This consideration is binary.
- This consideration doesn't care about `NeedTarget` and targets sent to it.

InfluenceMapConsideration

This consideration searches the influence map specified for a value and returns 1 if it finds something. Otherwise it returns 0.

Because the search is heavy we optionally store the result in a blackboard Vector3 key to be used by an action. If you leave the `queryResultKeyName` empty then we don't store the result anywhere.

The search uses the agent/target's position as the center and takes the radius in cell count as radius. The search condition and value has to be specified as well. For example if `searchCondition` is set to greater and `searchValue` to 0.3 then if a cell with values higher than 0.3 is found in the area noted by the radius and agent/target's position on the map, the consideration returns 1 and the map coordinates of the cell are stored in the blackboard key.

- This consideration is binary.
- This consideration works both with and without targets. The thing that targets effect is the center point of the search.

- The consideration only returns valid scores for SmartObjects if you set useNearestSlotPosition.

SmartObjectDistanceConsideration

Returns the distance to a target which is a smart object. You can choose to get the distance to a specific slot with a specific condition like the distance to the closes slot which is free.

If you check the box for useNearestSlotPosition then the consideration works only if the target is a smart object. The slotStatus allows you to ask for any slot's position or a slot which is free/owned by you.

For example if you choose FreeOrOwned then the distance to the closes slot which is either free or owned by the executing agent is returned.

As other distance considerations, this can be calculated on the XZ plane and can be inverted. Also the distance score is effected by minRange and maxRange so if you want to give a score of 1 to distances higher than 50 meters then maxRange should be 50. In this case 60 meters will return 1, 25 meters will return 0.5 and 10 meters returns 0.2. You should check inverse if you want 10 meters return 0.8 and 0 meters return 1 and 50 meters and above return 0.

- This consideration is continuous.
- This consideration is only meaningful with NeedTarget set to true and without targets always returns 0.
- The consideration only returns valid scores for SmartObjects if you set useNearestSlotPosition.

TargetDistanceConsideration

Returns the distance to a target. This normalizes to a value between 0 and 1 based on minRange and maxRange values so their value is extremely important for this to be meaningful.

If you want your score to check for distances less than 100 meters then you need to set maxValue to 100 so the consideration returns 1 for 100 meters away and greater. It will return 0.75 for 75 meters and so on. You can invert the consideration by checking inverse so it returns 1 for 0 meters and 0 for 100 meters and 0.25 for 75 meters.

- This consideration is continuous.
- This consideration is only meaningful with NeedTarget set to true and without targets always returns 0.

ValidGameObjectPathConsideration

This consideration returns 1 if a valid path from the agent's NavmeshAgent to the position of the GameObject stored in the key is present, otherwise returns 0. The NeedTarget causes the GameObject to be read from the target but always a path from the owning agent is calculated. So you can check if your agent has a valid path to a position which a target specifies but you cannot check if that target has a valid path to somewhere.

- This consideration is binary.
- This consideration works both for targets and the executing agent but only the reading of the vector3 happens on targets and the path is calculated for the agent itself anyways.
- The agent with the action owning this should have a NavMeshAgent component attached. The target of the consideration should have the **BlackBoard** component for this to work.

ValidPathConsideration

This consideration returns 1 if a valid path from the agent's NavmeshAgent to the position stored in the Vector3 key is present, otherwise returns 0. The NeedTarget causes the vector3 to be read from the target but always a path from the owning agent is calculated. So you can check if your agent has a valid path to a position which a target specifies but you cannot check if that target has a valid path to somewhere.

- This consideration is binary.
- This consideration works both for targets and the executing agent but only the reading of the vector3 happens on targets and the path is calculated for the agent itself anyways.
- The agent with the action owning this should have a NavMeshAgent component attached. The target of the consideration should have the **BlackBoard** component for this to work.

Action Sets

An action set contains a set of actions and is mainly used like a directory for grouping similar actions together in the editor window and doesn't introduce any functional difference to your actions and their scoring. Other than visual organization uses, you can add an action set to a **Brain** component at runtime which is more specific than adding all action sets of an agent behavior asset.

To add an action set

- Select an agent behavior asset in the project view and open the *Window>NoOpArmy>Wise Feline* window or just double click on the behavior.
- Click Add Action Set button to add a new action set.

Then the action set can be deleted by selecting it and pressing the delete button. It also can be renamed by selecting it and then changing the name property in the inspector of the wise feline window. Of course while selected you can add actions to it as well which is the main point of having action sets. Each agent behavior should have at least 1 action set.

Brain component

Brain is the main component in the Wise Feline system which allows you to direct a game object using your Agent Behavior assets. You should attach this component to any game object which you want to control using utility AI. The main parameter that the component takes is the Agent Behavior asset to use but there are additional parameters and methods which are discussed in the reference linked below.

Keep in mind that this component allows you to add and remove behaviors at runtime so you can add region specific behaviors or situation specific behaviors when an agent needs to consider them during gameplay. This makes your job much easier when designing action scores with considerations. If this feature did not exist you had to have lots of flag considerations with 0 and 1 responses for the times that action was possible or not but thanks to this feature you can simply add jungle specific behaviors to agents only when they enter the jungle zone and remove it when they exit.

The component allows you to choose how often decision making (i.e. action scoring) runs and how often the agent updates its list of targets to consider. You see this by how often your actions and considerations are called to calculate their scores. However the `Update()`, `FixedUpdate()` and `LateUpdate()` callbacks of your current action with its current target (if applicable) will be called every `Update()`, `FixedUpdate()` and `LateUpdate()` no matter how low or high you set the frequency for thinking or updating targets.

For more information, see the API reference for the [Brain](#) class.

Agent Behavior asset

An Agent Behavior asset defines a list of actions which are scored and executed when the behavior is attached to a **Brain** component. To modify the agent behavior asset you use the Wise Feline window.

To create a new agent behavior:

- right click in the project view and go to the Create menu, then choose *NooPArmy>Wise Feline>Agent Behavior*.
- Name the behavior whatever you want. Usually this is the name of the agent type or the name of set of behaviors. For example this can be enemy behaviors, forest behaviors or more specifically enemy at forest behaviors.
- Then select the created asset and go to the *Window>NoOpArmy>Wise Feline* window.

Now here you can add action sets which behave as folders for organizing your actions. In each action set you can add actions and for each action you can define a set of considerations which effects how the action is scored at each think operation. Closing the window automatically serializes all changes and saves them to the disk.

More details

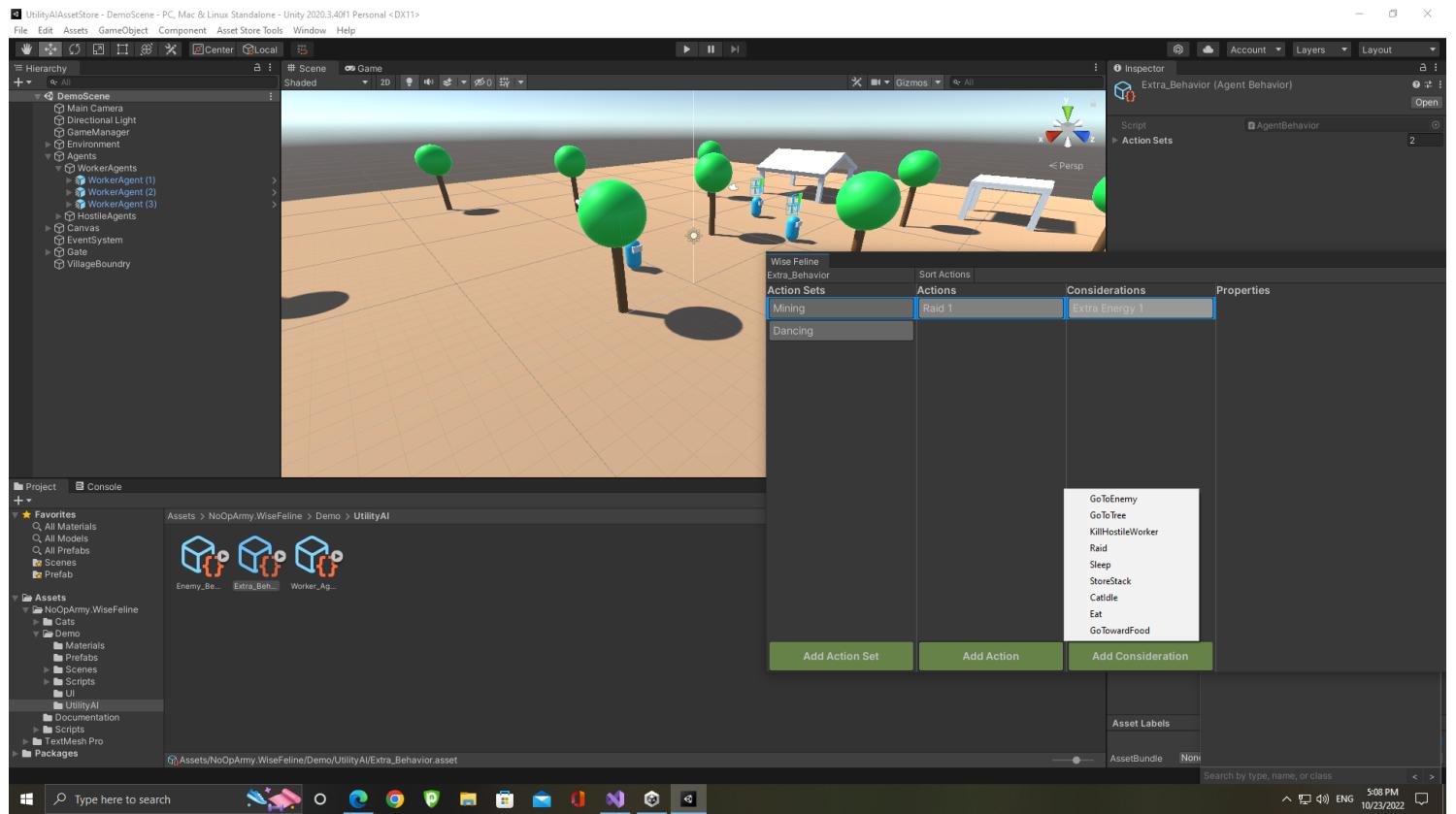
You can attach agent behavior assets to **Brain** components at edit time or add/remove them dynamically to a **Brain** at runtime. Basically this is the data which is used at runtime to see what actions are available to an agent and how each action's score should be calculated.

The **Brain** component has properties about what agent behavior assets to use for the game object and how often it should execute the AI and the agent behavior asset contains what actions are available to execute and how their scores should be calculated.

For more information, see the API reference for the [AgentBehavior](#) class.

Wise Feline Editor Window

The editor window can be opened at Window>NoOpArmy>Wise Feline



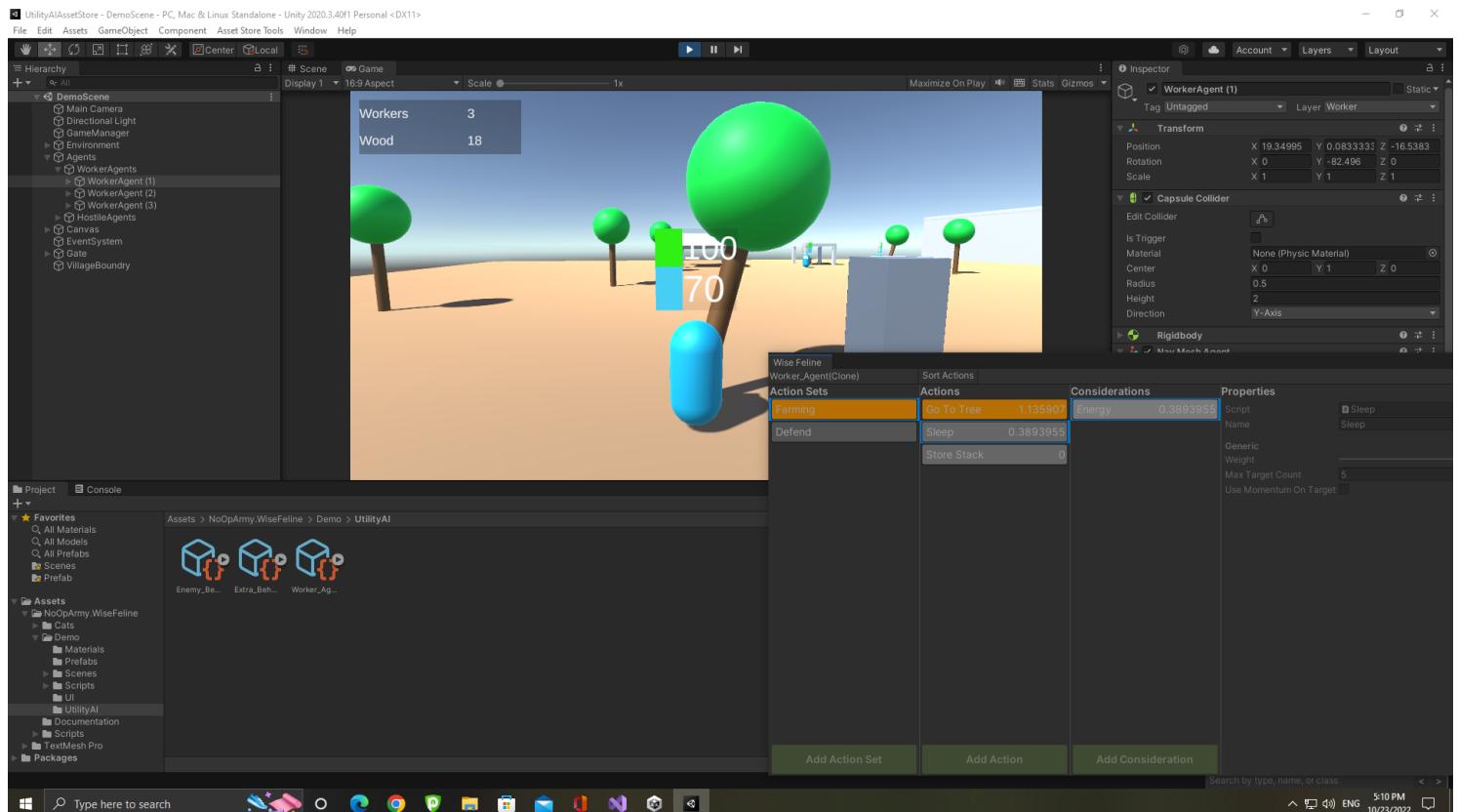
When the window is open you can select a [Behavior Agent](#) asset and then edit the actions, considerations and action sets available in the asset.

At runtime the window can be used to [debug](#) the scores of actions to see if curves and waits are set correctly to get the results you want. You just need to select game objects which have the [brain](#) attached in the hierarchy/scene and then open this window to see how its scores are being calculated.

How to Debug

You can debug your AI scoring by selecting a game object which has a [Brain](#) component attached at runtime while the Wise Feline window is open. The window then shows the list of all action sets and actions available to the agent and shows you what score each of the actions got. You can also sort the list of actions by score.

The window looks like this



As you can see the left most column are the action sets and then you got actions and then considerations. The window updates whenever scores are updated.

Profiling and optimizations

Profiling

Profiling the AI might be a bit tricky because both thinking and looking for targets happens once in a while but if they are heavy, they can create spikes. We have markers called `WiseFelineThink` and `WiseFelineGetTargets` which you can look for in the CPU profiler to see how much time they take but it is more likely that your actions are more important to optimize than these unless you have really expensive considerations and in an order which causes them to always execute.

Optimization

To optimize your AI, please do the least amount of both Think and Get Targets as you can and set the order of your considerations in a way that the most expensive considerations which contain raycasts and path findings get executed last. In this way if the other considerations cause the score to be 0, the expensive ones will be skipped.

Also keep in mind that sometimes you can create game specific world databases using octrees and similar structures which allows you to do your queries faster. We are going to eventually help in this as well after the main AI features are done.

Design Resources

There is a good amount of resources regarding utility AI from Dave Mark's book to his videos and other things which all can be found in this wiki page of a utility AI sandbox project [here ↗](#).

We are not associated with that project or Mr. Mark but his book and these videos were very useful to us so we thought it might help you guys too and hopefully we give back a bit to him by sending a few potential book purchases over.

The demos

The description for both demos should be read in order. Even the light demo's description is not repeated in the complete one's so don't skip a section unless you are very familiar with everything and can easily jump around the tutorials.

Light Demo

This [video](#) describes what this sample is and how it works.

There is a Demos folder in the asset which contains either 1 or 2 demos depending on the version of the package you have. One of them is the light demo which is described here and the other one is the complete demo described in the next section.

How to run the demo

Simply running the scene in the demos/LifeSimLight/Scenes folder runs the demo. You can use the fly cam to observe the agents. Just use WASD to move and hold right click and move the mouse to look around.

Important folders

- All actions and considerations use the [built-in actions and considerations](#) which rely on [blackboards](#) and our [movement components](#) to work. Read the related pages linked here to learn more about them.
- Scripts contains all the custom code for the sample which is a simple script just setting some blackboard keys and the fly cam.
- Prefabs Contains all prefabs, behavior definitions and blackboard definitions.
- The Art folder outside this demo and in the Demos folder contains all the artwork

The AI design

The agents use the same AI and it is pretty simple.

In the Prefabs/Agent/AI demo you can find the blackboard definition and the agent behavior files related to the agent in the demo.

The blackboard definition contains 4 floats:

- Hunger which shows how hungry the agent is.
- Energy which shows how tired an agent is.
- Bladder which shows how much does the agent need to go to the WC.
- Entertainment which shows how bored the agent is.

It also contains 4 GameObject keys which tells the agent which WC, TV, Refrigerator and Bed belong to this agent. `LightDemoAgent.cs` sets the object keys based on what the designer dragged to its slots in the inspector.

```
myBlackBoard.SetGameObject("Bed", bedGameObject);
myBlackBoard.SetGameObject("WC", wcGameObject);
myBlackBoard.SetGameObject("Refrigerator", refrigeratorGameObject);
myBlackBoard.SetGameObject("TV", tvGameObject);
```

Before doing this it also needs to get the blackboard component and set the values of the float attributes to something random so all different agents don't start doing the same thing like this

```
myBlackBoard = GetComponent<BlackBoard>();
myBlackBoard.SetFloat("Hunger", Random.Range(0f, 1f));
myBlackBoard.SetFloat("Energy", Random.Range(0f, 1f));
myBlackBoard.SetFloat("Bladder", Random.Range(0f, 1f));
myBlackBoard.SetFloat("Entertainment", Random.Range(0f, 1f));
```

This randomization needs to be done in a smarter way but this is simple enough and good enough for the sample. You probably want to set these based on some parameters like starting hour of the simulation or a scenario/quest/any other thing.

The same `LightDemoAgent.cs` calculates some personalized values per agent to change the 4 float attributes using them over time so an agents gets hungry faster and another needs more sleep.

```
void Start()
{
    personalizedHunger = Random.Range(0.01f, 0.02f);
    personalizedEnergy = Random.Range(0.01f, 0.02f);
    personalizedBladder = Random.Range(0.01f, 0.02f);
    personalizedEntertainment = Random.Range(0.01f, 0.02f);
}
```

Then every frame it does something similar to this for all attributes

```
float hunger = myBlackBoard.GetFloat("Hunger");
if (hunger > 0)
    myBlackBoard.SetFloat("Hunger", hunger - personalizedHunger * Time.deltaTime);
```

As you can see it reads the current hunger from the blackboard and then decreases it by the personalized amount and sets it on the blackboard again. For all values 1 means they are full and there is

no need to do an action to increase them and 0 means they are not full. You might have designed it differently so a full bladder means bad and an empty one is good but we wanted them to be uniform so a full bladder = 1 and means you don't need to go to the bathroom.

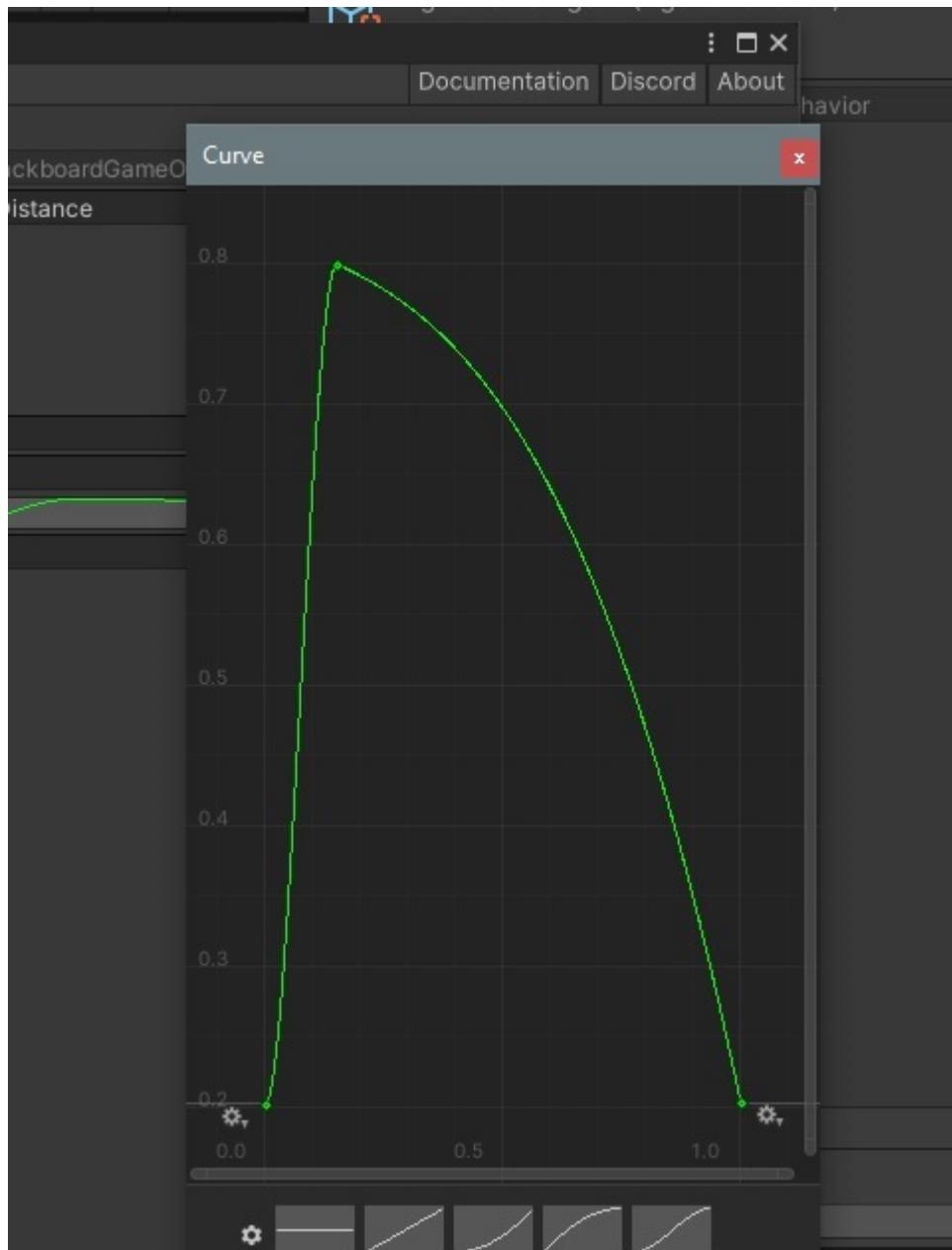
Now let's look at the AI behavior made for the utility AI of this agent in Prefabs/Agent/AI/LightDemoAgent.asset.

It contains 8 actions which 4 of them are for moving toward a place like the TV for entertainment or the bed for sleeping which are named MoveToX and 4 others which actually do the thing like watching TV or sleeping.

All movement actions use `MoveToGameObjectAction` which moves to a gameobject specified by a Gameobject key on the blackboard. All other actions use `BlackboardActionBase` which allows them to change blackboard keys. This is the action that almost all other built-in actions inherit from it.

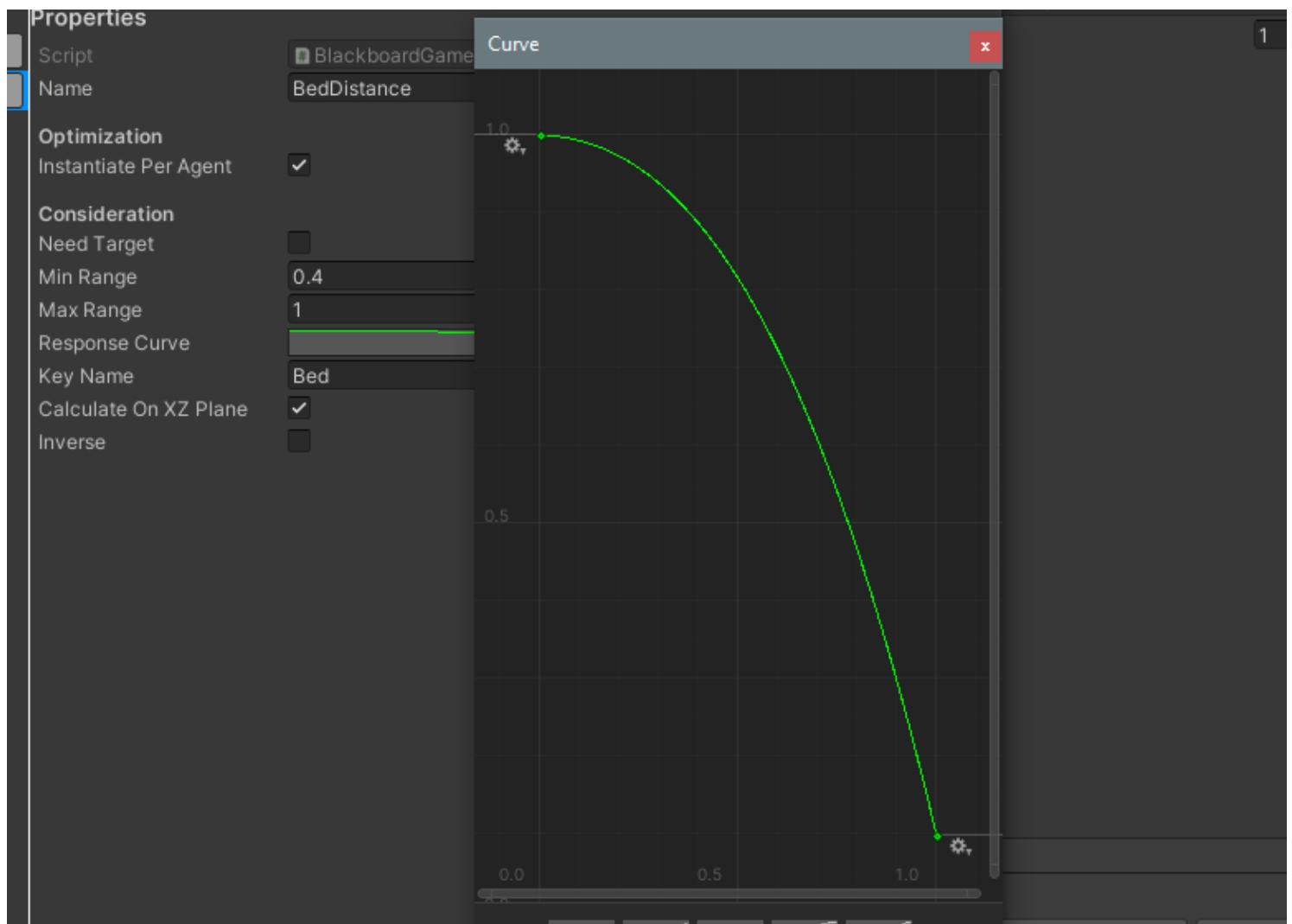
All actions use two considerations, one is for checking the float for their attribute on the blackboard which is the same on both and the other is a distance check consideration which has different curves so when the agent is far from the object, the MoveTo scores higher and when it gets close, the actual action scores higher.

For example MoveToBed uses a BlackboardFloatConsideration which it named EnergyFloat and a BlackboardGameObjectDistance Consideration which it uses to check the distance with the bed. The curve is like this



This curve means for values close to the max range which is set to 10 it will return a low score and for values close to 0 it returns a very low score as well but for stuff in the middle it returns high values.

The SleepAction shares the EnergyFloat consideration which checks the float attribute in the blackboard but its BedDistance has a very different curve.



The range is different and is between 0.4 and 1 meters and the closer to 0.4 it is , it returns a higher value so as soon as we are close to the 0.4 then this scores very high so this action is chosen instead of MoveToBed.

All other actions and their move ones are designed similar to sleep and move to bed and you can take a look at them yourself.

Complete life simulation / Immersive world demo

This is a [Video Tutorial](#) of a good part of the complete sample but you still need to read this for the description of every action and consideration.

This sample uses influence maps, smart objects, utility Ai, AI tags, blackboards and all other features to create an immersive world.

How to run the demo

Simply running the scene in the demos/LifeSimDemo/Scenes folder runs the demo.

- You can use the fly cam to observe the agents. Just use WASD to move and hold right click and move the mouse to look around.
- You can also focus on characters by clicking on them. There is a camera button to follow that object in the UI and also the UI shows attributes of the agent and its current action.
- Use the Tab key to move between agents and focus on them one by one. To focus on the next type of agent like go from humans to cats or from cats to mice, use Shift+Tab.

Important folders

- Materials contains all the materials assigned to the different characters and objects.
- UI contains the atlas for the UI icons you see on the screen.
- SmartObjects contains all smart object definitions which are used in the project. These define how many slots an object has and where are the slot positions and things like that.
- Prefabs contains all prefabs used in the game and their ai definitions like agent behaviors, blackboard definitions, influence maps templates and ... Smart object definitions are not defined here and are in their own folder. For example Prefabs/Cat contains 4 prefabs for the base cat and the 3 different colors of cat and an AI folder containing the cat's influence map template, blackboard definition for all cat attributes and the agent behavior file for the Utility AI's Brain component.
- Scripts contains a few general scripts like the follow cam and the main game/simulation manager and folders containing other scripts.
- Scripts/UI contains the code for custom UI controls and the main UI Manager which updates the UI.
- Scripts/SmartObjectBehaviors contains the custom SmartObjectBehavior classes we implemented for the car and mouse hole smart objects.
- Scripts/Agents contains the main agent code for all agent types outside their AI code which changes their blackboard attributes over time. It also contains the code required for cat companionship behaviors like cats greeting each other or humans petting cats.
- Scripts/Actions contains all of our custom actions for the demo.
- Scripts/Considerations contains all considerations for the demo.
- The Art folder outside this demo and in the Demos folder contains all the artwork

The AI design

The AI of this is much more complex because many more things are going on so we describe it in parts.

Trees

We want to spawn oaks for humans to gather and mice to eat. Trees and oaks are the things you need to look into.

- `OakTree.cs` is attached to trees and spawns oaks from the trees. The script finds where to spawn the next oak by looking at oaks influence map and then if it found a free place, it spawns an oak. The

```
check is like this if (oakMap.SearchForValueWithRandomStartingPoint(0.5f, SearchCondition.Less, oakMap.WorldToMapPosition(transform.position), 3, out resultPosition))
```

- The Oak prefab has an `InfluencerAgent` attached which writes the oak's template on the oaks map only once. They also have an `AITags` component so it has a tag which allows others to query it.

These oaks are eaten by mice and humans and are gathered by humans as their job so new ones are needed so the tree generates new ones every few seconds which you can check in `OakTree.cs`

Mice

Mice have their houses/mouse holes and live in there, come out and eat oaks and will try to run away if cats try to hunt them. Since cats eat mice, our mice are the only agents which breed more of themselves. Actually the holes generate new mice when needed.

We are going to look at the hole and the mice behavior now. The Prefabs/Mouse folder contains both the `MouseAgent` and the `MiceHole` prefabs. It also contains the AI folder which contains the influence map template, blackboard definition and agent behavior assets for the mouse and the smart object definition for the hole.

The hole is a smart object which has 3 slots for 3 mice and it has a behavior attached called `MouseBreedingBehavior`. The `SmartObject` component makes this prefab a smart object and it needs a smart object definition to know how many slots it has and it also needs to add the behaviors to the behaviors list of the `SmartObject` component. The breeding behavior is pretty simple so let's take a quick look at it.

```
private void Start()
{
    StartCoroutine(CheckForSpawningFirstMouse());
}

public IEnumerator CheckForSpawningFirstMouse()
{
    while (true)
    {
        yield return new WaitForSeconds(Random.Range(0, 15));

        bool hasAtLeastOneMouse = false;
        foreach (var item in smartObject.GetSlots())
        {
            if (item.owner != null)
            {
                hasAtLeastOneMouse = true;
                break;
            }
        }
    }
}
```

```

        }

        if (!hasAtLeastOneMouse)
        {
            InstantiateNewMouse();
        }
    }
}

```

This code simply breeds new mice whenever the house is empty but we also have code in the behavior so whenever the mouse uses the house, if it is not hungry, it breeds a new one and becomes fully hungry.

```

public override void OnUsedByAgent(GameObject agent, SmartObject smartObject, int slot)
{
    base.OnUsedByAgent(agent, smartObject, slot);
    if (agent.GetComponent<BlackBoard>().GetFloat("Hunger") > 0.4f)//The mouse can only
breed if it is not hungry :
    StartCoroutine(BreedNewMouse(agent));
}

IEnumerator BreedNewMouse(GameObject agent)
{
    yield return new WaitForSeconds(3);

    if (smartObject.GetFreeWithoutOwnerSlotIndex() != -1)//if there is a free slot
    {
        if (agent != null) // May be killed by cats or humans
            agent.GetComponent<BlackBoard>().SetFloat("Hunger", 0);

        InstantiateNewMouse();
    }
}

private void InstantiateNewMouse()
{
    var smartObject = GetComponent<SmartObject>();
    var slotIndex = smartObject.GetFreeWithoutOwnerSlotIndex();
    var slotPosition = smartObject.GetSlotPosition(slotIndex);
    var newMouse = Instantiate(mousePrefab, slotPosition, Quaternion.identity);

    smartObject.SetOwner(newMouse, slotIndex);
}

```

```
newMouse.GetComponent<MouseAgent>().SetHome(smartObject);  
}
```

The `OnUsedByAgent()` method is called by the smart object on its behaviors when one of its slots are being used by an agent. Smart objects can be claimed/reserved and then used and slots can have owners. These houses can have 3 slots for 3 mice to live in them and when those die, the slots become free. Here when a mice start using a slot it means she came back home to sleep. We check if the mouse's hunger level is not that bad then it can breed a new mouse. Of course we check to see if a free slot exists or not. if no free slots exist `smartObject.GetFreeWithoutOwnerSlotIndex()` returns -1 so we make sure that is not the case. To learn more about smart object you can read their docs [here](#).

The mouse itself has only few attributes in its blackboard definition.

- Attacked is a bool which is set to true when a cat attacks the mouse so it freezes in its place.
- Hunger is a float which shows the mouse's hunger level. breeding behavior of the hole sets this to 0 when the mouse is breeding and eating oak increases it.
- OakLocation is a Vector3 set by the influence map system described below. This is a location which has a more density of oaks for eating.
- LowDangerPosition is another Vector3 set by another influence map described below. This is a place with less cats and humans which we can move to.

The MouseAgent prefab uses a [BlackBoard component](#) with this definition to read/write these values. It also has an [InfluencerAgent](#) which puts its influence on the Mouse map when it moves on. This map is used by humans and cats and other systems to know where the mice are. It also uses an [AITags](#) to assign itself a tag so others can query the AI Tags system for the Mouse tag and find mouse agents. It also has a [Brain](#) component which executes the utility AI logic for the mouse and we describe below. The custom script [MouseAgent](#) is attached to the prefab which does the following activities.

- It tells the main manager script when it is created and when it is destroyed so it can be added/removed in the list of trackable objects for the camera.
- Has methods to get/set its house which is used by the breeding behavior above and other code.
- Set its hunger to 0.1 at start. Mice don't get hungry over time in this script.
- Handles the selection logic for the UI which is described later on and is the same for all agent types.

The brain component references an agent behavior asset with a couple of actions and their considerations. The file is Prefabs/Mouse/AI/MouseAgentBehavior.asset. This behavior contains the following actions and their considerations.

Sleep

This action is for the mouse sleeping and Uses NoopAction which does nothing so the mouse just stays where it is. The code for the action truely does nothing other than an optional stopping of the

movement if you set its flag to true. This is the `OnStart()` which gets executed whenever the action is selected to execute.

```
protected override void OnStart()
{
    base.OnStart();
    if(StopMoving)
    {
        var movement = Brain.GetComponent<IAIMovement>();
        if (movement != null)
            movement.StopMoving();
    }
}
```

If you want to know more about the movement components and how they move the agents, you can read more at the [built-in actions and considerations page](#).

The only consideration used for this action is the `TimeOfDayConsideration` which is set to return 1 between 3AM and 9AM. This is one of the 3 custom considerations in the simulation project which does not use a built-in consideration. The code of the `GetValue()` method is like this

```
protected override float GetValue(Component target)
{
    int currentHour = DayAndNightCycle.Instance.GetHour();
    if (currentHour >= MinAcceptableHourRange && currentHour < MaxAcceptableHourRange)
    {
        return 1;
    }
    return 0;
}
```

As you can see it simply reads the current hour from the `DayAndNightCycle` script and if it is between the specified hours, it returns 1 and otherwise it returns 0.

Search For Oak

This action searches for a high density of oak in an area to move to it. The action itself is actually just a `MoveToVector3Action` which takes a blackboard key to move to. However the key is shared between the action and its `InfluenceMapConsideration` which searches the okas map around the mouse position and puts the resulting high density position in the same key. As a result of this we only do the map search once and if the search finds a good enough point and the mouse is hungry enough, this action scores high and the movement toward the point happens.

The action itself can be read about in the [Built-in](#) page so we will not repeat how it moves to the vector3 specified but the considerations worth taking a look at.

The HungerConsideration is a [BlackboardFloatConsideration](#) which checks a float value in the blackboard to find its score. It uses the hunger value and the closer to 1 it is, it returns a higher value but the inverse checkbox is checked so actually it returns higher scores when the score is close to 0. This is a built-in consideration as well. This consideration returns 0.7 at max so the action can never score higher than 0.7. We discuss the reason when we look at the other actions but basically there are other actions which are more priority than this like eating an oak if it is close enough which should execute if their score is high so this search action should never score higher than them.

The other consideration is of the type [InfluenceMapConsideration](#) and searches the Oak map in 20 meters radius for values greater than 0.8 so if we have a position close to an oak in our 20 meters, this will find it and puts it in the blackboard key specified which is set to the same OakLocation key that the action uses for movement. This consideration only returns 1 if the influence map search is successful and otherwise returns 0 so the action will not execute if we cannot find a point with a value higher than 0.8 in our 20 meter radius.

The order of considerations matter since if the first consideration returns 0 in non-editor builds of the project, the rest of the considerations are not executed so you should put more expensive considerations like influence map searches after the simpler ones like blackboard checks. Expensive things are the things which take memory/CPU like path finding, influence map searches, ray casts and collision checks and other spatial queries.

Move To Target Oak

This action moves to a specific oak we targetted and uses the [MoveToTargetWithTag](#) which is a built-in action. The Search For Oak action searches the area for a high density of oak and moves toward it but this one chooses a specific oak to move to and eat. [MoveToTargetWithTag](#) asks for a tag to find potential targets and a radius to look for them and uses the [AITags](#) system to do the search. Here we look in our 8 meters radius for object with the Oak tag.

If we wanted to we could exclude objects which have another tag like oaks which are dirty or ruined or ... or we could look for a more general tag like mouse food and set the tag both on oaks and walnuts and maybe other things which mice eat which i don't like to mention here.

The considerations of this action are different from the search one so when in the correct situations, this cores higher than the search. The radius is smaller here so between 20 to 8 meters, the search action will obviously win if it can find something but if we have something in our 8 meters then this action will have targets so its considerations are executed to find a good target. The HungerConsideration is similar to the search for oak action's hunger consideration and only checks the blackboard to see how hungry the mouse is. However its max value is 0.9 unlike the one for the search action which at most returns 0.7. The

curve shapes are almost the same but the max point chosen for them is different in the curve so the curve for this one goes to a higher point. The other consideration which has NeedTarget set to true is the TargetDistance consideration and for targets closer than 4 meters, it returns a value closer to 1, the closer it is to you so a target with 2 meters of distance will score higher than one in 3 meters.

Actions with Considerations with NeedTarget set to true execute their considerations with NeedTarget set once per target and then choose the target with the highest score. This target is put in `chosenTarget` field of the `ActionBase` class which is accessible to all actions which you implement. Then if the action with its best target wins the competition with other actions and gets selected, you know which target is your best one and you should use. The considerations without NeedTarget set execute only once for the action and its brain and are not supposed to look at targets to return their score. You can learn more about this at the [targetting page](#).

The influence map consideration for the search action and the target distance consideration for this action can return scores up to 1 but since the hunger consideration for this action can return higher values the one in the search action, this action wins if we have a target close by. This is very important because when we have a target in close proximity, the influence map search can find a good position too. The next action has a similar to these two as well but the hunger for that returns values up to 1.

Eat Oak

EatOak action uses `DestroyTargetWithTag` action which is a built-in one and destroys its chosen target found by its tag. The built-in actions have a changer argument which changes the blackboard for the `GameObject` which the brain is attached to. We use the changer here to add a value of 0.2 once to the Hunger key of the blackboard so as a result this eating action increases the Hunger attribute by 0.2 (1 means full and no hunger actually) and also destroys the oak so it essentially does anything needed for eating. If you wanted to show a VFX, you could put it in the `OnDestroy()` method of a script attached to the oak or could inherit from this action and override `OnFinish()` to show the effect.

The considerations of this action are set in a way that when we are close enough to an oak, it scores higher than Move To Oak and Search For Oak. There are 2 considerations. The TargetDistance one looks for targets which are at most 1 meter away and the closer they are, it returns a higher score. And the BlackboardFloatConsideration looks at the Hunger key but for most values it returns one unless we are really close to 1 and don't need to it. This means that if we are close to an oak by accident and we are half full, we still will eat it since we can have more food and it is close so why not. This is probably in line with what a real mouse would do as well. The fact that both of the considerations for this action return values up to 1 means that when we are close to an oak, Move To Oak scores 0.9 and Search For Oak scores 0.7 but Eat Oak scores 1.

Move To Home

The mouse moves to its home which is set at birth time. This is a custom action which we take a look at its code.

It first caches its movement component which is a built in component. This one uses NavmeshBasedMovement component since it uses a NavMeshAgent to move around and not a CharacterController.

```
protected override void OnInitialized()
{
    agentMovement = Brain.GetComponent<IAIMovement>();
}
```

This is good practice to cache components which you want to use a lot to save yourself calling GetComponent every time this wants to execute.

```
protected override void UpdateTargets()
{
}

protected override void OnStart()
{
    base.OnStart();
    var home = Brain.GetComponent<MouseAgent>().GetHome();

    if (!agentMovement.MoveToPosition(home.transform.position, ReachToDistanceSuccessfully))
        ActionFailed();
}
```

We don't have targets in this action so `UpdateTargets()` is empty and actually is not called either. It would have been called if at least one of the considerations had `NeedTarget` set to true.

In `OnStart()` which is called each time the action is selected to execute once, we simply get our home from the `MouseAgent` component and ask the movement component to move to it. If it returns false it means we failed so we fail the action. In this simulation this will almost never happen since the path finding to our home should fail for this to not succeed. The movement function asks for a callback to call when it succeeds and the callback contains the following code.

```
void ReachToDistanceSuccessfully()
{
    var home = Brain.GetComponent<MouseAgent>().GetHome();
    var mySlotIndex = home.GetFirstOwnedSlotIndex(Brain.gameObject);
    // This should not happen unless we have force removals/takeovers which we don't yet.
    // However the mouse commits suicide if this happens which is extreme but whatever,
    // unrealistic things which don't happen in the simulation don't need messing up with.
    if (mySlotIndex == -1)
    {
```

```

        Destroy(Brain.gameObject);
        ActionFailed();
        return;
    }

    var slotPos = home.GetSlotPosition(mySlotIndex);
    if (home.Claim(mySlotIndex, Brain.gameObject))
    {
        Brain.transform.position = slotPos;
        home.Use(mySlotIndex, Brain.gameObject);
    }

    ActionSucceeded();
}

```

As you can see this method simply checks if our slot is still ours and if yes, first tries to claim the slot of the smart object which should succeed and then sets our position to the exact position of the slot since the NavMeshAgent moves very close to it but not exactly on it. Then it calls the `Used()` method of the smart object which calls `OnUsed` in all behaviors and causes the breeding behavior described above to make a new mouse if this mouse's hunger value is high enough and free slots exist in this house.

As you can see in the joke comment in the code, if we add something to the simulation so the mice can forcefully take each other's houses then our mice commits suicide and destroys itself since it has no home to return too but this doesn't happen in our simulation since ownerships are respected here and there is some good law and order coded in the sim :)

By the way whenever you call succeeded or failed on an action, it no longer executes and the brain starts thinking again to evaluate all actions and their considerations to see which one it should select as the next best action to execute.

Now let's see when the mouse wants to go home. There are only 2 considerations. There is a hunger consideration which is a `BlackboardFloatConsideration` and checks hunger and its curve returns values close to 1 if we are near full so a hungry mouse never goes home. The other consideration is a custom one which checks if the home has free slots to go back to or not. The `GetValue()` for the consideration looks like this

```

protected override float GetValue(Component target)
{
    MouseAgent mouse = target.GetComponent<MouseAgent>();
    if (mouse != null && mouse.HomeHasFreeSlot())
        return 1;
    else

```

```

        return 0;
    }
}

```

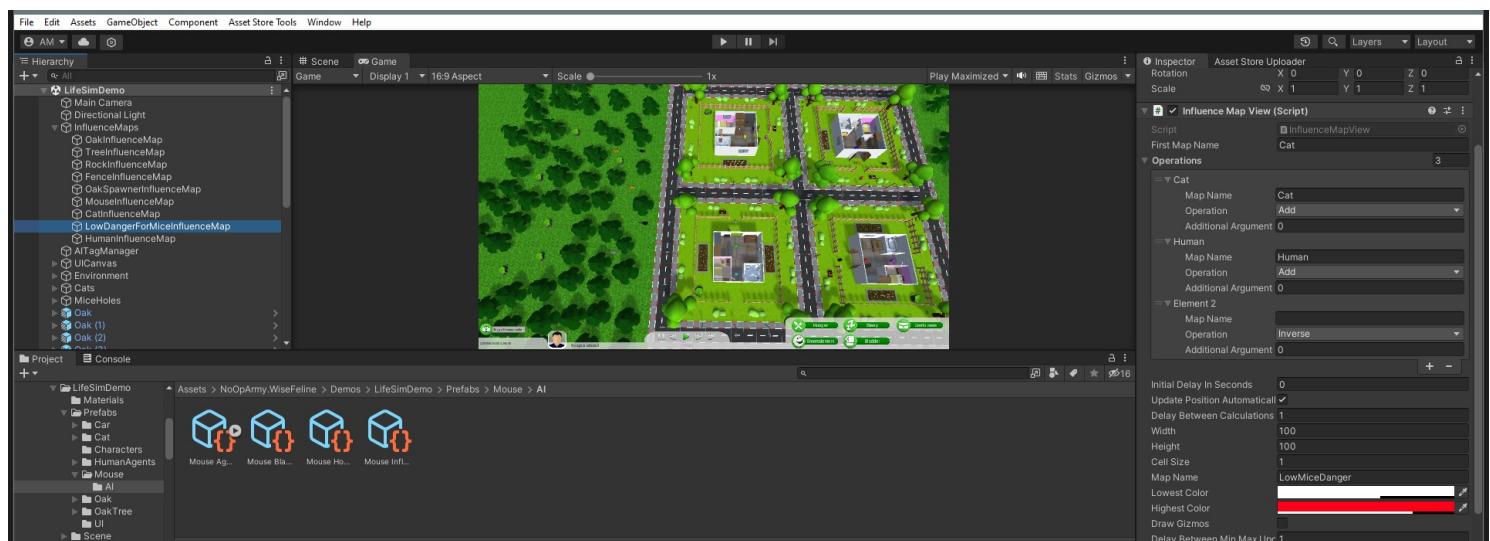
As you can see it is a very simple code which simply checks if the mouse has a home and if the home has empty slots to make more children or not. If there is no free slots then the mouse cannot make more mice so does not have the motivation to go back. It does not sound good but keeps the simulation interesting by keeping the mice outside when there are too many of them so cats can eat them and then they can make more of themselves. We are not trying to code any moral or social or ... messages into this. We just tried to make it look interesting so keep your interpretations to yourselves.

As a result of these considerations which both can return up to 1, if a mouse's home has free slots and it is mostly full, it will go home to make more mouse. Maybe we should have called in incubator or something else. They sleep anywhere and eat anywhere and only make babies at home? whatever.

Before making this action happen only when we need to make new mice, we did not have the move to low danger locations action and mice would go home a lot. As a result cats searched for high density areas for mice and would roam around mice holes and ate them as soon as they came out. It was not that interesting so we made this action more rare and added more intelligence to mice.

Move To Low Danger Location

This action uses a built in [MoveToVector3Action](#) and shares its blackboard key with its influence map consideration. The interesting thing is that the influence map that the search consideration looks at is the map of humans and cats combined and a low danger position is defined as a position which the least amount of cats and humans are around it. This action has two considerations which one of them is the same [BlackboardFloatConsideration](#) which looks at the hunger value and after your hunger is higher than half or so its value goes up fast and for 0.8 or so and higher returns 0.8 which is its highest value. The other consideration is an [InfluenceMapConsideration](#) which checks the LowMiceDanger map in its 40 meter radius for values higher than 0.6. If you look at the object for the map



It is an [influence map view](#) which combines other maps. It starts with the cats map and adds cats and humans to it and then inverses it so higher values are for the places which are actually empty of cats and humans. It uses cats twice because they are more dangerous for mice.

As a result this consideration returns 1 when it can find somewhere which is relatively empty of humans and cats. The found position is put in the blackboard under the same LowDangerPosition key which the action uses to move to its position.

Under Attack

This action is for mice to freeze when they are under attack from cats so the hunt is more beautiful. The action has only one consideration which checks the bool key called Attacks in the blackboard and if true returns 1. As a result when under attack this usually gets the highest score since it does not care about anything else including hunger which most other mice actions check.

The action is a NoOp like the one described above and causes the mouse to freeze and get eaten by the cat.

Cats

Cats are maybe the most complex agents or at least are on par with human agents so you are in for a treat. They seek each other to greet and rub their heads. They hunt mice to eat. They sleep and get petted by humans and ... They have less actions than humans but more custom code in their actions. All this said everything is relatively simple and much simpler than other Ai algorithms so don't worry and read on.

The cats have 4 prefabs. 1 for the base cat prefab and 3 prefabs for gray, black and orange cats. The differences are very minimal and other than the meshes, they just have their colors as AI Tags so they can great their friendly factions and not other factions. They don't fight so don't worry about that.

The cat prefabs have an [AITags](#) component which assigns them a Cat tag and a tag based on their color which is Gra, Orange or Black. They have an [InfluencerAgent](#) which adds their influence using their template to the Cat influence map. They have a [Brain](#) which executes the utility Ai and also the [NavMeshBasedAIMovement](#) and [NavMeshAgent](#) components for movement. There is a [Blackboard](#) component which has a definition defining all blackboard keys for the cat. The keys are

- Hunger which shows how hungry the cat is. The closer to 0 it is, the cat is more hungry.
- Emotion which shows how much entertainment and love from others the cat needs. The closer to 0 it is, the need is higher for greetings and petting behaviors.
- MouseLocation is a vector3 used by the search action and its influence map consideration to find the locations which have a high density of mouse in them.
- TargetCat, Waiting and Companionship are used in the companionship system used later on which is the system governing how cats greet each other and how humans pet them.

Let's look at all the actions and their considerations one by one.

Morning Sleep

This uses the [NoopAction](#) and just stops the movement of the cat which uses the same [NavMeshAgent](#) and [NavmeshBasedMovement](#) that the mice use too. The only consideration is the [TimeOfDay](#) consideration which for this one returns 1 when the time range is between 6AM and 10AM.

This manual is created to be read in order so read the mouse one first to know more about the actions and considerations used.

Night Sleep

The only difference between this and the morning sleep is the time range that the cat goes to sleep. The time range for the [TimeOfDay](#) consideration is between 4PM and 8PM.

Search For Mice

This action uses [MoveToVector3Action](#) to move to a vector3 key on the blackboard attached to the cat but the key is shared with one of its considerations which searches to influence map for mice to find a place which has a lot of mice in it. The considerations are

- HungerConsideration which is a [BlackboardFloatConsideration](#) and checks the Hunger key in the cat's blackboard and for the values closer to 0 returns a higher value but its max value is about 0.7 so considerations for actually eating a mouse get higher score than this when the mouse is close enough.
- The other one is an [InfluenceMapConsideration](#) which looks in the mouse map in the 30 meter radius of the cat to find values greater than 0.8 which means there are a lot of mice in that area. If successful then it puts the position in the MouseLocation key which the action uses to move to. This is similar to the search for oak action in the mouse prefab above which has more description.

All mice have an [InfluencerAgent](#) component and put their template on the map where they are so the map allows us to find them pretty easily. If we wanted to we could try to find mice which are alone or places which mice exist but no cats are around but no need to make it more complex for the sample really.

Hunt Mouse

This action has custom code and is defined in [HuntingMouseAction.cs](#). This said the custom action inherits from [MoveToTargetWithTag](#) and essentially tries to move to objects tagged with Mouse but does some additional things on top.

The important parts of the code are

```

protected override void OnStart()
{
    base.OnStart();
    trailRenderer.startColor = Color.red;
    trailRenderer.endColor = Color.red;
}

protected override void OnFinish()
{
    base.OnFinish();

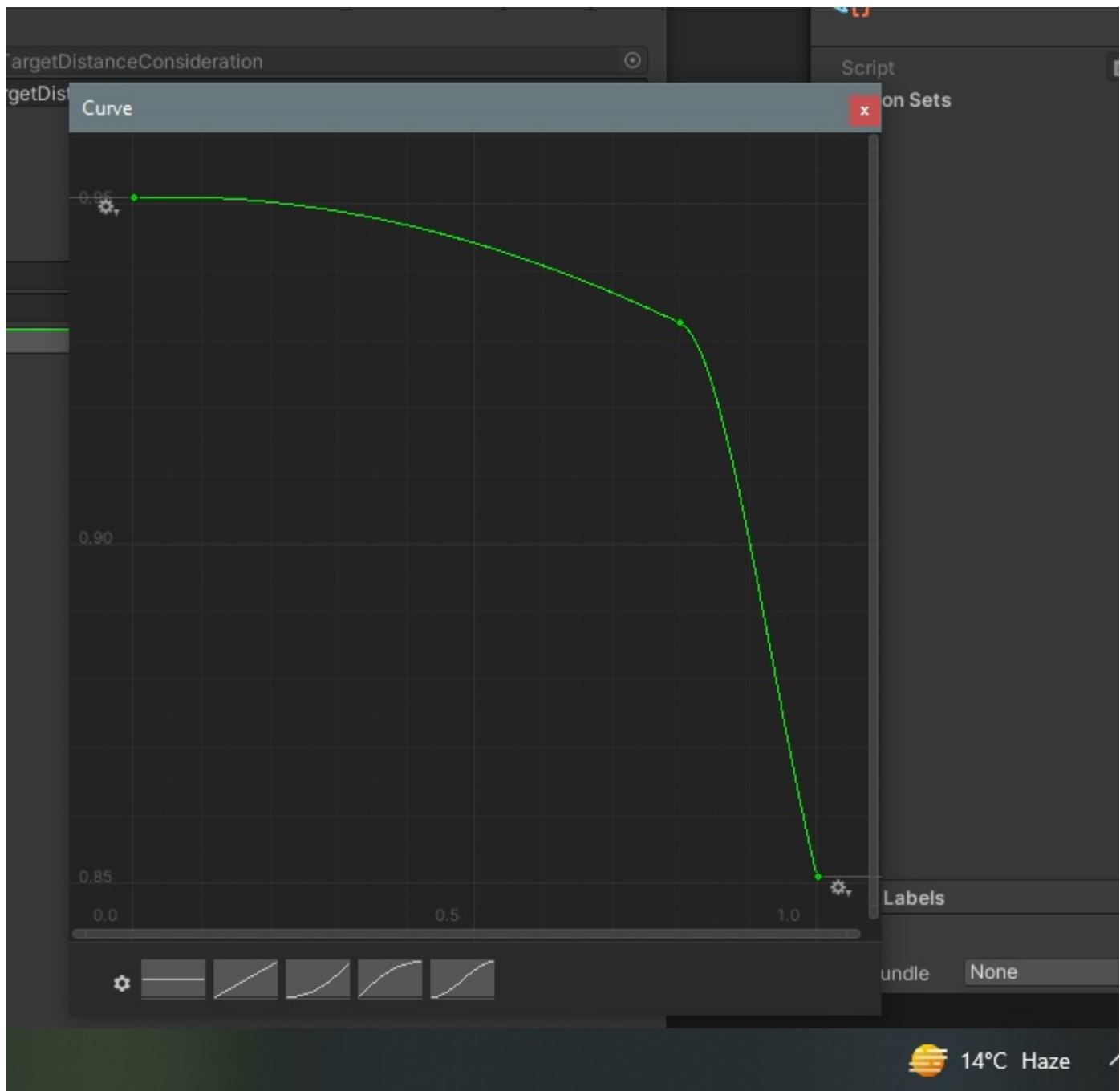
    trailRenderer.startColor = Color.white;
    trailRenderer.endColor = Color.white;
}

```

As you can see the only modifications the action does are making the trail renderer of cats red when they start hunting and turning it back to white when the action finishes. In this way you can easily add your code on top of the built-in actions or your other custom made ones.

The action is set to look for objects tagged Mouse in its 12 meter radius and will move toward the best of them if chosen. When arrived it calls `ActionSucceeded()` which causes `OnFinish()` to get called. `OnFinish()` is called in the case of failures or even another action getting a higher score and forcing this to change as well so there is no situation that the action changes and `OnFinish()` not getting called.

There are two considerations. HungerConsideration which looks at the Hunger float on the blackboard and returns values up to about 0.9. The curve returns 1 for higher values which means less hungry but the inverse is checked so actually the closer to 0 you are, this returns a higher value. This does not return 1 so the actual mouse eating behavior has a chance of scoring higher. The other consideration is a TargetDistance which considers targets up to 12 meters and the closer a target is, it returns a higher value and actually returns something close to 0.95 for a big range of values because it makes sense for it and it wants to move toward the target so 2 meters and 5 are not that different for this action. The curve looks like this:



We don't show all curves here but I encourage you to examine them and see what shapes we used for each action. Dave Mark's [book](#) and talks in the [resources](#) section talk about these a lot as well.

Eat Mouse

This action has some custom code and inherits from `DestroyTargetWithTag` which destroys its target and finds targets using their tags. The custom code on top of that other than changing the trail renderer's color checks the mouse to see if it is already being attacked by another cat or not. If not it sets the `Attacked` boolean to true in the mouse's blackboard and then destroys it. Here is the code of the action

```
protected override void OnStart()
{
```

```

base.OnStart();
trailRenderer.startColor = Color.red;
trailRenderer.endColor = Color.red;
MouseAgent mouseAgent = ChosenTarget.GetComponent<MouseAgent>();
if (!mouseAgent.WaitToBeEaten())
    ActionFailed();
}

```

```

protected override void OnFinish()
{
    base.OnFinish();

    trailRenderer.startColor = Color.white;
    trailRenderer.endColor = Color.white;
}

```

As you can see the important part of the code is what `WaitToBeEaten()` returns and its code looks like this:

```

public bool WaitToBeEaten()
{
    if (myBlackboard.GetBool("Attacked"))
        return false;

    myBlackboard.SetBool("Attacked", true);
    GetComponent<Brain>().Think();

    return true;
}

```

This code checks to see if the mouse is already being attacked, then it returns false and we will not try to take food from another cat. Otherwise we mark it as attacked and nobody else will try to take it from us.

The main work happens in the parent class which is `DestroyTargetWithTagAction` and its code looks like this:

```

protected override void UpdateTargets()
{
    ClearTargets();
    AddTargets(AITagsManager.Instance.GetTagsAroundPoint(Brain.transform.position, radius,
    includedTags, excludedTags));
}

```

```

protected override void OnStart()
{
    base.OnStart();
    startTime = Time.time;
}

protected override void OnUpdate()
{
    base.OnUpdate();
    if (Time.time - startTime >= delayForDestruction)
    {
        if (ChosenTarget != null)
        {
            GameObject.Destroy(ChosenTarget.gameObject);
            ActionSucceeded();
        }
        else //Something else already destroyed that object
        {
            ActionFailed();
        }
    }
}

```

The code is pretty simple. In `UpdateTargets()` we clear all targets and add the new ones using the [AI Tags system](#). Then after a target and the action are chosen, in `OnStart()` we just set start time and based on the delay, when the time to kill finally arrives, we destroy the target if it still exists and call it a success and otherwise fail the action.

In the example of `EatMouseAction` in the child class's `OnStart()` we might fail the action if the mouse is being attacked by another cat and the `OnUpdate()` will never gets called and the other cat will destroy the mouse.

Also the action's changer is set to increase the Hunger by 0.5 (fill its bar by half) which is the result of eating a mouse.

Seek Cats

There is a system for cats to greet each other and for humans to pet them and the system requires work from both parties involved so the code is a bit complex. The system is used in this action and the rest of the cat actions and the petting action in humans. So we describe it here.

There is an interface which both humans and cats implement and it starts a companionship, ends it, waits for one to start and checks if a cat/human busy in another companionship act. It does not differ between a human petting a cat or two cats rubbing their heads with each other.

```

/// <summary>
/// agents like other cats and humans which want to do friendly actions with cats need to
implement this interface
/// The reason is that the petting behavior or cat greetings can only happen in sequence and
a cat can be in one of these at a time
/// </summary>
public interface ICatCompanionship
{
    /// <summary>
    /// Tells a cat taht this cat is being petted by a human or being greeted by a cat and
should wait for this action to finish
    /// </summary>
    /// <param name="otherCat"></param>
    public abstract void WaitForOtherCompanionship(GameObject otherCat);

    /// <summary>
    /// Is the agent busy in another Companionship
    /// </summary>
    /// <returns></returns>
    public abstract bool IsBusy();

    /// <summary>
    /// A Companionship gets started involving this agent
    /// </summary>
    public abstract void StartCompanionship();

    /// <summary>
    /// Ends a Companionship action and this agent is free to do another now
    /// </summary>
    public abstract void EndCompanionship();
}

```

The Start and End methods are called by the cat/human which initiates the companionship on the other cat and `IsBusy()` is naturally called to check if the other party is busy with another cat/human or not. `WaitForOtherCompanionship()` is called by the other cat/human to tell the cat to stop moving and wait for the companionship to happen.

The seek action itself has custom code but it inherits from `MoveToTargetWithTag`. It's code looks like this:

```

protected override void OnInitialized()
{
    base.OnInitialized();
    CatAgent catAgent = Brain.GetComponent<CatAgent>();
    if (catAgent != null)

```

```

        includedTags = includedTags.Concat(catAgent.SeekingCatTag).ToArray();
    }

    protected override void OnStart()
    {
        base.OnStart();

        var blackboard = Brain.GetComponent<BlackBoard>();

        targetCat = ChosenTarget.GetComponent<CatAgent>();

        if (targetCat.IsBusy())
            ActionFailed();

        targetCat.WaitForOtherCompanionship(Brain.gameObject);
        blackboard.SetGameObject("TargetCat", ChosenTarget.gameObject);
        blackboard.SetBool("Companionship", true);
    }

    protected override void OnUpdate()
    {
        base.OnUpdate();

        if ((targetCat.transform.position - Brain.transform.position).magnitude < 1f)
            ActionSucceeded();
    }
}

```

First I want to show you how simply we implemented factions. Each `CatAgent` component has a `SeekingCatTag` array which allows it to say cats of which category it seeks. Gray cats only seek the Gray tag and the orange and black ones seek both Orange and Black in their cat prefabs so here we add the tags set there to the list of tags which this action will try to find and move to in `OnInitialized()`.

Then in `OnStart()` we check if the other cat is busy `if (targetCat.IsBusy())` and if yes we fail the action. Otherwise we tell the target to wait for us using the interface methods mentioned above and set the other cat as our `TargetCat` in our blackboard and set the bool which causes the companionship action to score high. Companionship is the action used in cats for greeting and in humans to pet a cat. The last thing we do is that in our update, whenever we are close to the other tag, finish the action by declaring success.

The `CatAgent` component's waiting function and `IsBusy()` are defined like this

```

public void WaitForOtherCompanionship(GameObject otherCat)
{

```

```

        myBlackboard.SetBool("Waiting", true);
        myBlackboard.SetGameObject("TargetCat", otherCat);
        GetComponent<IAIMovement>().StopMoving();
    }

    public bool IsBusy()
    {
        return myBlackboard.GetBool("Waiting") || myBlackboard.GetBool("Companionship");
    }

```

We return that we are busy if we started a companionship in our seek action or if another cat called us to wait and our Waiting is set. `WaitForOtherCompanionship()` simply stops the cat's movement component and sets Waiting and TargetCat.

The agent has other important functions which are described in the next action.

The considerations for the seek action are many but they are very simple. There is a TargetDistance which considers targets up to 50 meters and at max returns 0.8 for targets which are at its 0.1 mark which is 5 meters when you multiply it by the 50 meter max distance. Then it has a BlackboardBoolConsideration which checks the Waiting key and only returns 1 when it is false. The EmotionConsideration is a BlackboardFloatConsideration which checks the Emotion key and returns high values up to 1 for emotions more than 0.1 and has inverse checked so essentially for anything lower than 0.9 it returns a high value. Cats are very much into playing so this makes sense. The IsCatFreeConsideration has NeedTarget set to true so works on the targets of the seek action and its code looks like this

```

protected override float GetValue(Component target)
{
    CatAgent cat = target.GetComponent<CatAgent>();
    if (cat != null && cat.IsBusy() || target.gameObject == Brain.gameObject ||
    target.GetComponent<BlackBoard>().GetFloat("Emotion") > 0.6f)
        return 0;
    else
        return 1;
}

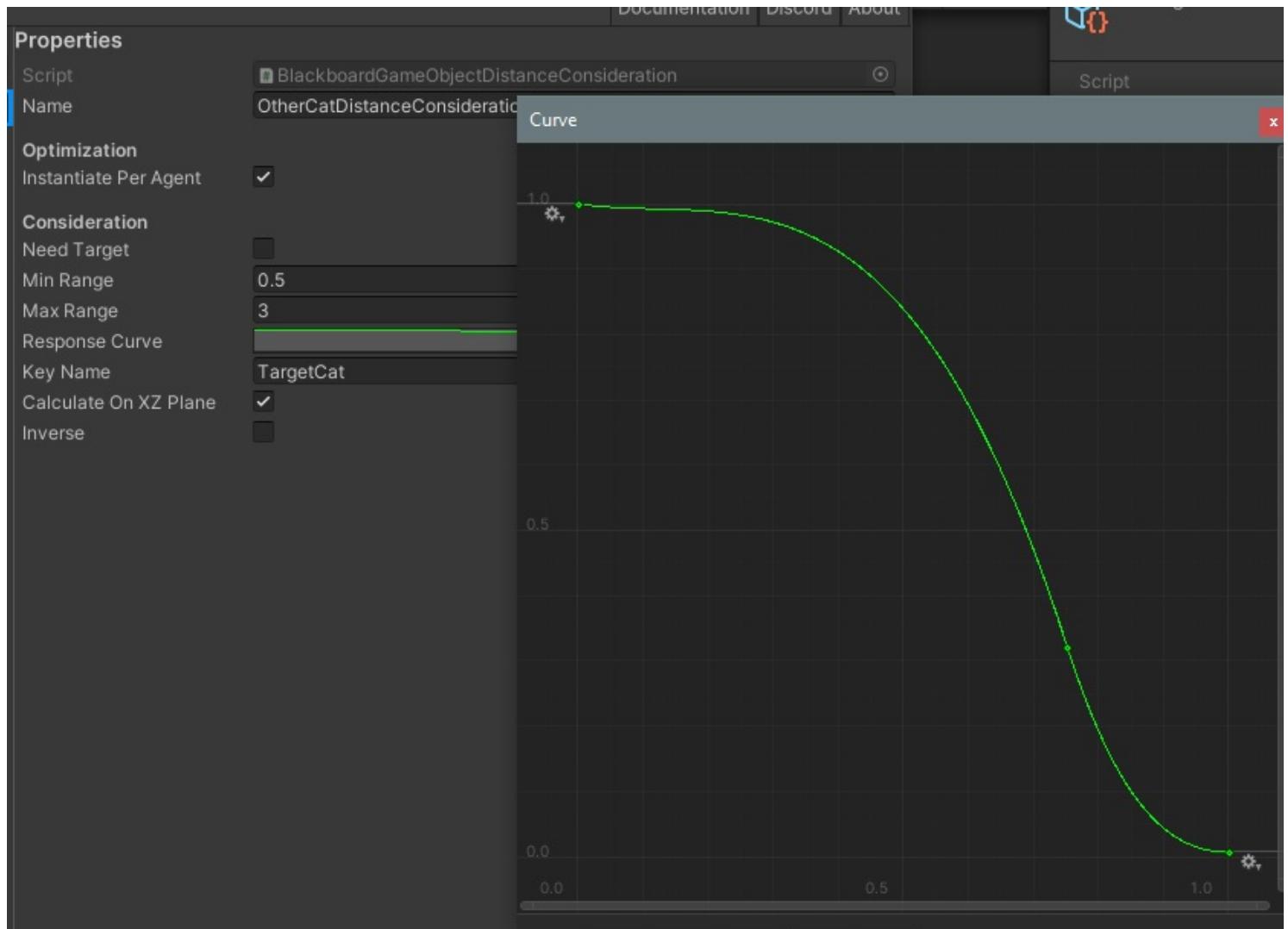
```

If we are busy or the target is ourselves or the target doesn't have a low enough emotion, then this fails. We don't want to greet targets which have an already high emotion. Since the consideration is binary only the 0 and 1 positions of the curve matter which return 0 for 0 and 1 for 1.

Companionship

This action is the actual greeting action of the cats and actually is used in humans when they pet cats too. It happens when a cat's Companionship bool key in the blackboard is set to true and its distance to another cat is low enough. The considerations are just a bool for the Companionship key which returns 1

if the bool is set. And a BlackboardGameObjectConsideration which checks the distance to a GameObject set in the TargetCat key and for distances between 0.5 and 3 meters it returns a value between 1 and 0. The curve returns a high value up to half of the max distance and looks like this



Keep in mind that this action is executed on both greeting cats and the human and the cat he/she is petting.

Let's look at the action code itself.

```
protected override void OnInitialized()
{
    base.OnInitialized();
    myCompanionship = Brain.GetComponent<ICatCompanionship>();
    myBlackboard = Brain.GetComponent<BlackBoard>();
}

protected override void OnStart()
{
    base.OnStart();
```

```

    targetCompanionship =
myBlackboard.GetGameObject("TargetCat").GetComponent<ICatCompanionship>();
    targetCompanionship.StartCompanionship();

}

protected override void OnUpdate()
{
    base.OnUpdate();

    if (myCompanionship is CatAgent)
    {
        if (myBlackboard.GetFloat("Emotion") >= 0.9f)
            ActionSucceeded();
    }
    else if (myCompanionship is HumanAgent)
    {
        if (myBlackboard.GetFloat("Entertainment") >= 0.9f)
            ActionSucceeded();
    }
}

protected override void OnFinish()
{
    base.OnFinish();

    targetCompanionship.EndCompanionship();
}

```

In `OnInitialized()` we get our `ICompanionship` component and it might be a `CatAgent` or `HumanAgent`. As said before this action is used on both for greeting/petting a cat. In `OnStart()` we get our target's `ICompanionship` based on what is set in the blackboard in the seek action. Then we call its `StartCompanionship()` and in `OnFinish()` we call its `EndCompanionship()`.

`OnUpdate()` is a bit more interesting since it checks the Emotion or Entertainment key of the blackboard depending on the fact that we are actually attached to a human or a cat and if it is greater than 0.9, the action is succeeded. The `is` keyword in C# can check object types and since we know only these two classes implemented the interface we got using `GetComponent<T>()`, we can know if we are a human or a cat by checking this. We could have a boolean for the designer to set or check the tags of the object but this is also a way to check this.

The changer of the action is set to repeat an action with delay and to add 0.1 once per second to the Emotion of the cat executing this action so eventually 0.9 is reached. Obviously the one attached to a human will increase its own Entertainment and not Emotion.

Now let's look at `StartCompanionship()` and `EndCompanionship()` in `CatAgent`.

```
public void StartCompanionship()
{
    if (myBlackboard.GetBool("Companionship") == false)
    {
        myBlackboard.SetBool("Companionship", true);
        myBlackboard.SetBool("Waiting", false);
        GetComponent<Brain>().Think();
    }
}

public void EndCompanionship()
{
    myBlackboard.SetBool("Companionship", false);
}
```

The Waiting key should be true when we are waiting for a companionship to start but when it gets started, it should become false and Companionship key should be set to true. When the companionship finishes then the Companionship key should be set to false which is what these two methods do. These key changes cause the other cat to go from waiting action to Companionship to increase its Emotion.

Wait For Companionship Cat

This action happens when its single consideration which checks the Waiting key in the blackboard returns 1 and does nothing. It uses `BlackboardBoolConsideration` with a curve which returns 1 when the Waiting boolean key is set to true. And the `NoopAction.cs` does nothing and it even does not stop movement since the previous action which was seeking another cat or the `WaitForCompanionship()` method called by the other human/cat already stopped its movement. The Emotion of this cat doesn't change in in waiting but after this the cat will go to Companionship which will increase its Emotion.

Other methods of CatAgent component

```
private void Awake()
{
    indicator = GetComponent<SelectedIndicator>();
    myBlackboard = GetComponent<BlackBoard>();

    personalizedHunger = Random.Range(minPersonalizedValue, maxPersonalizedValue);
    personalizedEmotion = Random.Range(minPersonalizedValue, maxPersonalizedValue);
}

void Start()
{
    myBlackboard.SetFloat("Hunger", Random.Range(0, 0.7f));
```

```

myBlackboard.SetFloat("Emotion", Random.Range(0, 0.7f));
myBlackboard.SetBool("Companionship", false);

LifeSimDemoManager.Instance.AddFollowableAgent(indicator, AgentType.Cat);
}

private void OnDestroy()
{
    if (LifeSimDemoManager.Instance != null)
        LifeSimDemoManager.Instance.RemoveFollowableAgent(indicator, AgentType.Cat);
}

void Update()
{
    float hunger = myBlackboard.GetFloat("Hunger");
    if (hunger > 0)
        myBlackboard.SetFloat("Hunger", hunger - personalizedHunger * Time.deltaTime);

    float emotion = myBlackboard.GetFloat("Emotion");
    if (emotion > 0)
        myBlackboard.SetFloat("Emotion", emotion - personalizedEmotion * Time.deltaTime);

    // Check for mouse click or touch input
    if (Input.GetMouseButtonUp(0))
    {
        // Cast a ray from the camera to the mouse position
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        // Check if the ray hits the object with a collider
        if (Physics.Raycast(ray, out hit))
        {
            // Check if the hit object is the one you want
            if (hit.collider.gameObject == gameObject)
            {
                LifeSimDemoManager.Instance.SetAsSelectedAgent(indicator, AgentType.Cat);
            }
        }
    }
}

```

The selection and indicator logic is explained later on in the general simulation code section but the rest of the code is pretty easy and similar to the light simulation's human code. We set initial values for Emotion, Hunger and other blackboard keys and then find some random personalized values for the cat. In Update we reduce Hunger and Emotion by these random and personalized values so all cats don't

tend to do the same thing all the time. There are other ways to achieve this like more random selection behaviors in the brain but we chose to use highest score for ease of understanding in the demos.

Human agents

The human agent's behavior has similarities to the light simulation's human agents above so read up on that one first. The HumanAgent prefab has its own general code in `HumanAgent.cs`. It has the `Brain` for Utility AI, a `Blackboard` component, a `NavMeshAgent` and a `NavMeshBasedAIMovement`, and of course an `InfluencerAgent` to add its influence to the Human map.

The blackboard for humans is in the AI folder in `Prefs/HumanAgent/AI` alongside the agent behavior and the blackboard definition.

- Hunger is the attribute which fills when the agent eats food and depleats over time.
- Energy is the attribute which fills when the agent sleeps and depleats over time.
- Bladder is the attribute which fills when the agent goes to the WC and depleats over time.
- Entertainment is the attribute which fills when the agent watches TV or pets cats and depleats over time.
- Usefulness is the attribute which fills when the agent works and depleats over time.
- Job is an int key which shows what is the job of the agent.
- OakLocation is a Vector3 which is set by an influence map consideration and used by the actions that the human uses to gather or eat oaks.
- TargetCat is a GameObject key which is used by the system which pets cats to know what cat to pet.
- Companionship is a bool key which causes the human agent to give a high score to the petting action.

The influence map template is nothing special to look at so let's go through the AI behaviors.

Many of the actions set an animation parameter name for the On Start Animation Trigger and we actually have an animator with those parameters but unity did not allow us to submit the plugin with mixamo animations so we have capsules for the characters in the demo. You can use this feature of the built-in actions to drive your animations.

Move To WC

Most of the move actions use smart objects but the WC one does not move to a smart object because we wanted it to be different. Smart Objects have ownerships so agents will not use each other's beds but they'll use each other's WCs thanks to this difference.

This action uses the built-in `MoveToTargetWithTag` action and tries to move to an object in 30 meters with the tag WC. As said above it also sets a trigger in the animator with the name Walking in its `OnStart()`. Describing how animations work is beyond the scope of this document and you can read unity's documentation on that. This might not be enough for your animation needs. Even the fact that the

NavMeshBasedAIMovement allows you to set the velocity of the movement in an animation parameter might not be enough either but in these cases you can inherit from the action and do whatever you need there and call the `base` methods to do the movement.

The considerations of the action are a TargetDistance consideration which considers up to 100 meters and gets executed per target that the tagging system finds and also a consideration which checks the Bladder float key on the blackboard. The blackboard one returns up to 0.7 and the target distance one returns up to about 0.85 so the actual Wse WC can score higher than this.

Use WC

This action uses the `BlackboardActionWithTarget` class which finds all objects with the tag WC in 2 meters radius and if it is chosen it simply increases the Bladder float key on the blackboard by 0.5 every second.

It has two considerations

- A BlackboardFloatConsideration which checks the Bladder and it has inverse checked so actually instead of returning 1 for values closer to 1, it will inverse the curve and returns values close to 1 for the times that Bladder is closer to 0.
- The other one is a TargetDistance consideration which for distances between 0.5 and 1.5 meters returns a score which is close to one for the lower half of the distance and naturally anything less than 0.5 will have the 0.5 meters score which is 1.

This action is chosen over the Move To WC one because its curve will return scores up to 1 and this will score higher when close to a WC. The Move To WC action cannot return a value much higher than 0.7 for score because one of its considerations returns 0.7 as its max value.

Move To Bed

This action uses `MoveToSmartObject` and it checks 100 meters and the object should have a Bed tag and of course be Owned by the agent. The considerations are naturally a check for the energy of the agent and another for the target distance check. The distance check uses `SmartObjectDistanceConsideration` and sets the required slot status as Owned.

When the slot status contains owned then the smart object system checks the actual ownership of the agent toward that smart object.

Sleep

Each agent has his/her own place to sleep. Either a bed or a furniture is his/her sleeping place. The action uses `BlackboardActionWithSmartObjectTarget` class and the slot status is set to Owned. The radius is 1 meters and the tag is Bed for the search for the target in the smart object system. The action increases by 0.05 every second. The considerations are `SmartObjectDistanceConsideration` and the usual `BlackboardFloatConsideration`. The float checks the Energy key on the blackboard and has inverse

checked. The distance checker has the slot status is set to Owned so the distance check cares if the object is owned by the agent or not. The action itself had this too so each bed will only score high and only gets included in the list of targets for their owners. And I'm being very very lose when I call all of them beds. Some agents sleep on the couch!

Move To Entertainment Unit

This action finds smart objects with a specific tag which in this case is Entertainment and then moves to the chosen target. The class that it uses is [MoveToSmartObject](#). This action will fail if no free slots can be found on the smart object and also will fail if there is no path/possibility of movement to the object by the agent's [movement component](#).

The action has two considerations very similar to the ones the Move To WC above had but the ownership requirements of the smart object would mean that no two agents will use the same Entertainment slot on an object. The considerations are

- A EntertainmentFloat consideration which is a [BlackboardFloatConsideration](#) checking the Entertainment float key on the blackboard and returning up to 0.5. It has inverse checked so an empty Entertainment will return 0.5.
- And a TargetDistanceConsideration which for up to 100 meters return a value between 0 and 0.8 and naturally for closer distances will return a higher value so the closest target is chosen.

The reason that the non-targetted Entertainment float check returns 0.5 at maximum is that going to an entertainment object is one of the least important things when other options are going to WC, eating, sleeping and ...

Entertainment

Similar to Use WC, simply increased the Entertainment attribute of the agent.

[BlackboardActionWithSmartObjectTarget](#) is the action class used and it finds smart object targets and the changer of the action increases Entertainment increases by 0.1 per second. The target search happens with the smart object system in 3 meters for objects which have the Entertainment tag. This action does not choose a target which does not have a free slot just like the Move To Entertainment so a TV used by somebody else will not be used by this agent.

There are two considerations which one of them is a targetted consideration checking target distance called SmartObjectDistanceConsiderations. This consideration can optionally check the distance with the closes slot of a specific status instead of the position of the GameObject. We use the slot position here because the slot is far away from the GameObject transform. This one doesn't use ownership but beds and furniture which agents sleep on use ownership. The other one is a BlackboardFloatConsideration which checks the Entertainment key of the blackboard. The curve returns up to 1 and has the inverse checked. This means this action will be chosen over the movement one when it scores high since that one will never scores 1.

Go To Refrigerator

This action is the same as Move To WC and moves to a target which has the Food tag. The differences are the tags and the blackboard key checked and used which is Hunger and the curve shapes.

Eat

This is very similar to Use WC as well and just increases Hunger attribute. The considerations are for checking Hunger level and target distance as well. Check Use WC for more details.

Drive

This is the first job action that we are encountering. All humans have jobs either as oak gatherers, programmers or drivers. Each job requires at least two actions like Go To Car/Go to laptop and then drive/programming.

The Job key in the blackboard tells what job the agent has and changing it would actually change the job of the agent. 0 is programming, 1 is driving and 2 is oak gathering.

This action is the driving apart after the human went to his/her vehicle. The class used for the action is [DriveAction](#) which is a custom action which inherits from [BlackboardActionWithSmartObjectTarget](#). The action that this inherits from finds targets by using the smart object system and does nothing else other than being able to change the blackboard. The custom action just claims and uses the smart object and waits for it to finish driving and the going home time arrives and then it finishes the action successfully. The [CarBehavior](#) which is described below is a smart object behavior with a coroutine which simply moves the car around.

```
protected override void OnStart()
{
    base.OnStart();
    carSmartObject = ChosenTarget.GetComponent<SmartObject>();
    carBehavior = ChosenTarget.GetComponent<CarBehavior>();

    if (carSmartObject.Claim(0, Brain.gameObject))
    {
        carSmartObject.Use(0, Brain.gameObject);
    }
}

protected override void OnUpdate()
{
    base.OnUpdate();
    //waits for the driving to finish and final hour of the job arrives and then finishes
    //the action successfully
    while (carBehavior.IsDriving)
        return;
}
```

```

        while (DayAndNightCycle.Instance.GetHour() < endDrivingTime)
            return;

        carSmartObject.FreeAllSlots();

        ActionSucceeded();

    }

```

As you can see the `OnUpdate()` which is the important part simply waits for the driving and time to finish and then frees the object and declares success.

The considerations are:

- BlackboardInValueConsideration which checks a blackboard key to be a specific int value. It is a binary consideration which returns 1 only when the int key is equal to ExpectedValue. Here Job is checked to be 1 (means driven)
- The SmartObjectDistanceConsideration checks up to 3 meters and it returns 1 for almost all of them. Keep in mind that this will only consider targets added by the action so a far away car will not be considered at all. The search radius is 3 meters in the action.
- There is a Time Of Day consideration which only return 1 between 10:00-18:00.

Car behavior

The car is a smart object which is used by human agents which have the driver job. The CarSmartObject prefab uses the NavMeshAgent component for movement. It has a SmartObjectSlotOwnerSetter component which sets the owners of slots based on what you put in its SlotOwners array. It has a CarBehavior smart object behavior script and it also has a SmartObject component which references a smart object definition with one slots and a reference to the driving behavior (described below) as its behavior.

`CarBehavior.cs`'s main methods which are overriden for the behavior are like this

```

public override void OnClaimedByAgent(GameObject agent, SmartObject smartObject, int slot)
{
    base.OnClaimedByAgent(agent, smartObject, slot);
    agent.GetComponent<HumanAgent>().StartDrivingState(smartObject);
}

public override void OnUsedByAgent(GameObject agent, SmartObject smartObject, int slot)
{
    base.OnUsedByAgent(agent, smartObject, slot);
    StartCoroutine(Driving());
}

```

```

public override void OnFreedByAgent(GameObject agent, SmartObject smartObject, int slot)
{
    base.OnFreedByAgent(agent, smartObject, slot);
    smartObject.FreeAllSlots();

    agent.GetComponent<HumanAgent>().EndDrivingState(smartObject);
}

```

As you can see it just tells the human agent to go to driving state when it is claimed and when freed tells it to go back to normal state. The complex driving logic is in the `Driving()` coroutine called in the `OnUsedByAgent()`

```

private IEnumerator Driving()
{
    isDriving = true;
    navAgent.SetDestination(firstPosition.transform.position);

    while (!IsReached())
    {
        yield return new WaitForSeconds(1);
    }

    navAgent.SetDestination(secondPosition.transform.position);

    while (!IsReached())
    {
        yield return new WaitForSeconds(1);
    }

    navAgent.SetDestination(parkPosition.transform.position);

    while (!IsReached())
    {
        yield return new WaitForSeconds(1);
    }

    isDriving = false;
}

private bool IsReached()
{
    if (!navAgent.pathPending)
    {
        if (navAgent.remainingDistance <= navAgent.stoppingDistance)

```

```

    {
        if (!navAgent.hasPath || navAgent.velocity.sqrMagnitude == 0f)
        {
            return true;
        }
    }
    return false;
}

```

As you can see it is not in fact complex. It starts moving toward a first position and then when reached moves to the second and then to the parking position and then stops. `IsReached()` simply checks to see if it arrived to the destination or not.

The reason that it drives like a drunk driver who never was a good driver to begin with, is that NavMeshAgent's movement logic is not made for driving and similar behavior and is more aligned with how a creature would move. Also we did not set street paths so it behaves like this and it is an insult to drunk drivers actually!

Anyways this is how this code drives the car when the agent is acting as driver.

Go To Car

This action uses `MoveToSmartObject` class to move to the car for driving it. Each car is owned by an agent and the ownership is set by the `SmartObjectSlotOwnerSetter` component on the cars. The action searches for smart objects with the tag Car in 100 meters.

The considerations are:

- DriveJobValue which uses `BlackboardIntValueConsideration` which checks to see if an int key is the expected value or not and is a binary consideration. This one returns 1 for Job being equal to 1. The curve returns something a bit higher than 0.7 so the actual driving action above has a bigger chance when the agent is close to the car.
- There is a `TimeOfDay` consideration which returns 1 for the time between 10AM to 6PM.
- And of course a distance consideration for targets which always returns 1 because we don't need to differ between the targets other than checking ownership which the action does when adding targets. We have to have a consideration which scores all targets anyways so we have to include something. Theoretically a distance one can become useful later on in the demo so we added this. Otherwise a constant 1 would be enough. We know each agents only has one car and we have no buying/selling.

Eat Oak

Some of the humans go out to do their oak gathering job and while close to an oak and hungry, they can eat one. It will look intelligent and logical to do so. The action class used is `DestroyTargetWithTag`

because the action needs to find oak targets and if one of them is selected, it destroys the target and its changer increases Hunger by 0.15. The action finds targets by looking for objects with the Oak tag in its 3 meters radius using the [tag system](#)

The considerations are a float which checks Hunger and if it is relatively low, it returns values up to 1. And one which checks the target's distance which checks up to 3 meters and its curve returns something close to 1 for almost all values. This means that if hungry and close to an oak, the agent will eat it no matter what. We could check if there are any mice around or not and other things like only do this if very hungry or only if the time to go home is very far away from now but as is, our humans will eat oaks until they are more than half full.

Go To Oak Hunt

This action moves to a location which the influence map says there are lots of oaks in. The considerations naturally are a check for the int value of the job which returns 0.5 if the Job key is 2. There is an [InfluenceMapConsideration](#) which serches the Oak map for values greater than 0.7 in its 100 meters radius. This will put the result in the OakLocation key to move to. There is also a time of day consideration which is only returning 1 in the time range of 6AM to 4PM.

This action returns at most 0.5 so the more important action of actually gathering or eating oak can score higher in close ranges.

Move To Oak For Gathering

This uses [MoveToTargetWithTag](#) and tries to find objects tagged Oak in the 10 meter radius and move to the selected one if the action is selected. This action happens after the previous action brings the agent close enough to oaks. The considerations are designed in a way so if oaks are close by then this will score higher. The radius for the previous action was 100 meters.

The considerations are a time of day consideration so this happens only from 6:00 to 16:00 and a check to see if Job key is equal to 2. And an important one which checks target distances. It checks the target up to 10 meters and for close ones other than very close ones returns up to 0.9. The job key one returns 0.7 so it has less score than the eating/gathering ones.

Gather Oak

This is a simple [DestroyTargetWithTag](#) which has a small radius and looks for objects tagged Oak in 2 meters. It has a delay for destroying the oak, which looks good and differentiates it from eating as well. The changer on the action increases Usefulness key of the blackboard by 0.1. The considerations naturally are a Job key check to be 2 and a target distance and these return values up to 0.8 combined so this is chosen over the above actions other than eating oak if the agent is hungry.

Programming

This action uses `BlackboardActionWithSmartObjectTarget` to find its laptop and use it and the action's changer increases usefulness by 0.03 per second. The considerations are a check for the Job key to be 0, a target distance check and a time of day check like the other job considerations. The distance one checks ownership of the target too. It checks between 0.5 to 2 meters and returns up to 1. Since programmers usually start late, their time of day for work is between 12AM to 12PM. The Job key checker that checks to see if it is programmers favorite number is 0 or not and if 0 then it returns 0.5 on the curve.

Go To Laptop

This action is for moving toward the laptop for programming. This action uses `MoveToSmartObject` class and searches for targets in 100 meters and searches for objects which have the Laptop tag. The considerations are the same as the programming job with different curves and values.

The Job key checker returns 0.5 when the value is correct. The distance one doesn't matter so the distance returns 1 for all of them. The reason taht it doesn't matter is tht there is only one laptop owned by us but there is only one.

Seek Cat

The custom action is `SeakingCatAction` and it inherits from `MoveToTargetWithTag`. This simply finds the best cat in the area and then sets Companionship and TargetCat keys in its blackboard so it can pet the cat. Read more in the description of this action in the cats section. Keep in mind that humans pet all cats and don't add their preferences for specific cats. It is amazing that with only few lines of code we have this relatively complex faction system and the cats and humans can easily change factions and do everything else like changing the faction of others. We can even have events like holidays well even gray cats great others and then go back to their usual after the event.

The considerations are a check for the human's own Entertainment key which returns up to 1 and has inverse checked, A target distance check which returns at least 0.5 and a `IsCatFreeConsideration` which checks if the cat is interested in doing this with us and is free or not. This consideration is described in detail in the cat section but the main code for it is this.

```
CatAgent cat = target.GetComponent<CatAgent>();
if (cat != null && cat.IsBusy() || target.gameObject == Brain.gameObject ||
target.GetComponent<BlackBoard>().GetFloat("Emotion") > 0.6f)
    return 0;
else
    return 1;
```

Pet Cat

This action is the same `CatCompanionshipAction` which cats use and only gets a high score when a target cat exists and is set in the blackboard. The only difference is that the changer increases the

Entertainment value for the human and not Emotion which is something cats have and apparently humans don't :)

The considerations are only two which checks the two relevant blackboard keys and both of them if they have the correct value, will return 1.

The interface implemented in the HumanAgent like this

```
public void StartCompanionship()
{
    if (myBlackboard.GetBool("Companionship") == false)
    {
        myBlackboard.SetBool("Companionship", true);
        GetComponent<Brain>().Think();
    }
}

public void EndCompanionship()
{
    myBlackboard.SetBool("Companionship", false);
}

public void WaitForOtherCompanionship(GameObject otherCat)
{
}

public bool IsBusy()
{
    return true;
}
```

As you can see since the human will not be at the receiving end of this, `IsBusy()` and `WaitForOtherCompanionship()`'s implementation don't matter.

Kill Mouse

Kill Mouse is another action like Eat Oak to make humans smarter. It uses `DestroyTargetWithTagAction` and if the human is close to a mouse, then the human destroys the mouse. The action checks for objects with the tag Mouse in 1.5 meters radius. The only consideration is a distance check always returning one for all targets so if mice are super close to a human, they get killed.

The HumanAgent code

The code of the `HumanAgent` class is actually first initializing blackboard values.

```

private void Awake()
{
    indicator = GetComponent<SelectedIndicator>();
    myBlackboard = GetComponent<BlackBoard>();

    myBlackboard.SetFloat("Hunger", Random.Range(0.3f, 0.7f));
    myBlackboard.SetFloat("Energy", Random.Range(0.3f, 0.7f));
    myBlackboard.SetFloat("Bladder", Random.Range(0.3f, 0.7f));
    myBlackboard.SetFloat("Entertainment", Random.Range(0.3f, 0.7f));
    myBlackboard.SetFloat("Usefulness", Random.Range(0.3f, 0.7f));
    myBlackboard.SetInt("Job", job);
    myBlackboard.SetBool("Companionship", false);
}

```

The job value is read from the job exposed to the inspector so you can change the jobs of the agents in the inspector.

```

void Start()
{
    personalizedHunger = Random.Range(minPersonalizedValue, maxPersonalizedValue);
    personalizedEnergy = Random.Range(minPersonalizedValue / 2f, maxPersonalizedValue / 2f);
    personalizedBladder = Random.Range(minPersonalizedValue, maxPersonalizedValue);
    personalizedEntertainment = Random.Range(minPersonalizedValue, maxPersonalizedValue);
    personalizedUsefulness = Random.Range(minPersonalizedValue / 2, maxPersonalizedValue
/ 2);

    LifeSimDemoManager.Instance.AddFollowableAgent(indicator, AgentType.Human);
}

```

We set the personalized random values for the changes to main attributes so the agents act differently from each other and different simulation runs are different. This is what we do with all other agents including the light simulation as well.

The update function simply decreases all attributes based on the chosen personalized values.

```

void Update()
{
    float hunger = myBlackboard.GetFloat("Hunger");
    if (hunger > 0)
        myBlackboard.SetFloat("Hunger", hunger - personalizedHunger * Time.deltaTime);

    float energy = myBlackboard.GetFloat("Energy");
}

```

```

if (energy > 0)
    myBlackboard.SetFloat("Energy", energy - personalizedEnergy * Time.deltaTime);

float bladder = myBlackboard.GetFloat("Bladder");
if (bladder > 0)
    myBlackboard.SetFloat("Bladder", bladder - personalizedBladder * Time.deltaTime);

float entertainment = myBlackboard.GetFloat("Entertainment");
if (entertainment > 0)
    myBlackboard.SetFloat("Entertainment", entertainment - personalizedEntertainment
* Time.deltaTime);

float usefulness = myBlackboard.GetFloat("Usefulness");
if (usefulness > 0)
    myBlackboard.SetFloat("Usefulness", usefulness - personalizedUsefulness
* Time.deltaTime);

// Check for mouse click or touch input
if (Input.GetMouseButtonDown(0))
{
    // Cast a ray from the camera to the mouse position
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;

    // Check if the ray hits the object with a collider
    if (Physics.Raycast(ray, out hit))
    {
        // Check if the hit object is the one you want
        if (hit.collider.gameObject == gameObject)
        {
            LifeSimDemoManager.Instance.SetAsSelectedAgent(indicator, AgentType.Human);
        }
    }
}
}

```

It has some code regarding the selection logic like other agents which is discussed below as well.

The car driving behavior calls these two methods.

```

internal void StartDrivingState(SmartObject smartObject)
{
    humanMesh.SetActive(false);
    GetComponent<NavMeshAgent>().enabled = false;
}

```

```

        transform.SetParent(smartObject.transform);
        transform.position = smartObject.transform.position;
        transform.LookAt(smartObject.transform.forward);
    }

internal void EndDrivingState(SmartObject smartObject)
{
    humanMesh.SetActive(true);
    GetComponent<NavMeshAgent>().enabled = true;
    transform.SetParent(null);
    transform.position = smartObject.GetSlotPosition(0);
}

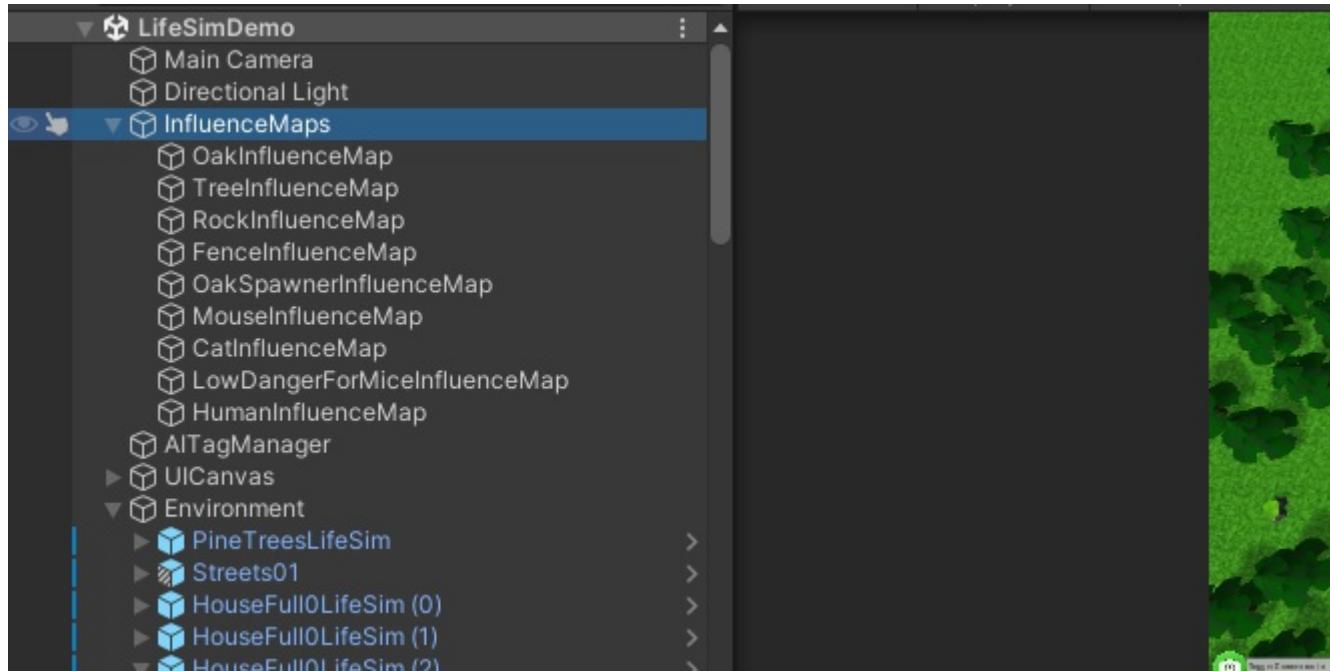
```

The methods simply hide the agent when it starts driving and makes it a child of the car and does the reverse when driving finishes.

The rest is the `ICatCompanionship` interface which is described in the actions. And this concludes our tour of the AI in the codebase.

Maps

We use several influence maps and influence map views as it got mentioned in the review above. All of the maps and views are children of the `InfluenceMaps` object.



General simulation code

The simulation needs some additional code for the follow camera and updating the UI and advancing time of day. Here we are going to take a very brief look at those scripts to make your job a bit easier.

- `CameraFollow.cs` is a very simple camera follow code which tolerates having a null target as well.
- `DayAndNightCycle.cs` moves time forward in a 24 hour day and changes its rotation too so you can attach it to your directional light which changes the sky color as well.
- `LifeSimDemoManager.cs` This class manages the change of simulation speed, has the code for changing the agent camera follows using Tab and Shift+Tab and other general things. There is no AI code in this class.
- `SelectedIndicator.cs` Is the code which selects the agent you click on and turns on the indicator on top of the agent's head. The agents themselves check this too and call methods in the `LifeSimDemoManager` to change the UI and do some other things. Needless to say the way that we code this can be improved a lot but this has nothing to do with the AI.
- `UIMenu.cs` Manages most UI code to show the correct icon and update the attribute bars and ... The event managers are attached in the editor and not in code.
- `SliderController.cs` Is the custom bar for the attributes of agents.
- `GameSpeedButton.cs` This handles the looks of the button which shows what speed the game is running on.

And after these many pages this is finally over. We are going to improve this tutorial and improve it and make more interesting stuff for you. Thanks for reading and please rate and review the assets if you liked them. Also send us your feedback on [discord](#) or using [email](#).

Blackboards

[Blackboards video tutorial](#)

A Blackboard is a data structure which allows you to share data between different components of your AI without having direct dependencies between them. A blackboard is a **Dictionary** like data structure which allows you to add data by defining it as a set of key value pairs.

Let's say you want to know what is the safest location to go to, you can define a safe location key in the blackboard and store the **Vector3** of the location in it. Our implementation of blackboards is more strongly typed than a **Dictionary<string, object>** and you should define all keys with the data type of their values before being able to set/get them.

Getting Started

To define a blackboard you first need to create a blackboard definition. To do that right click in your project and click on Create>NoOpArmy>Wise Feline>BlackBoard Definition. Then you need to assign that definition to a **BlackBoard** component. Then you can set or get the value of the keys you defined in the definition.

You have multiple methods in the **BlackBoard** component like **GetFloat(key)**, **SetFloat()**, **GetVector3(key)** and **SetVector3(key, value)**. You have these pairs for all data types and you can set and get the values in your code as you need.

How can I use this?

Your AI can use this in considerations and actions and other parts of your game. Let's say you have an action like, GoToLocation, you can take a blackboard key which you know contains a **Vector3**.

A small tutorial

- Create a definition and call it b.
- Add a float key called speed.
- Create a new script called **TestBlackBoard** and attach it with a **BlackBoard** component to a **GameObject**.
- Set the definition in the blackboard to the one you made above.
- Open the **TestBlackBoard** script you made and add some code like this.

```
public class TestBlackBoard : MonoBehaviour
{
    private BlackBoard board;

    void Start()
    {
        board = GetComponent<BlackBoard>();
    }
}
```

```
{  
    board = GetComponent<BlackBoard>();  
    board.SetFloat("speed", 3);  
    var speed = board.GetFloat("speed");  
    print(speed);  
}  
}
```

The code should print 3.

This module is only available in the [ultimate](#) edition of the Wise Feline package

AI Tags module

AI tags is a module which allows you to attach tags to objects at edit time and also at runtime. It also allows you to query all objects with tags in a sphere to decide based on the result or do anything with them. You can use this sub-system with Utility AI and it has a sample consideration or as a fully separate module outside of your utility AI.

How to use AI Tags

Add the `AITags` component to GameObjects which should have tags. You can add tags in the inspector or at runtime using the available methods.

Then you can use methods of the `AITagsManager` singleton to get objects around a point and check their tags. Also if you know an object has the `AITags` component, you can check its tags in code by getting the component directly.

`AiTageManager` singleton is created automatically if you don't add it to a component in your scene. It has methods for getting objects around a point which include and/or exclude some tags. At its most complex form you can give it a list of tags which at least one of them should exist in an object to qualify and also another array for tags which if any of them on the object then the object disqualifies from being returned.

Use-Cases

A few small examples might help. Imagine you want to implement generic behaviors like, I want my agent to eat all plants and avoid all animals. You can avoid hard coding this by tagging all plants with both plant and their specific plant name as tags and also tag all animals with animal and their specific animal name and then use search functionality of the AI Tags system to find all plants or all animals.

Also imagine the game has a dragon which can set things on fire and you don't want to specifically code all components for this, you can simply add firable to things which can be set on fire, even better you can add the tag at runtime to something after sometime, let's say there is an ice monster which can not be set on fire unless there is no minions around. you can add the firable tag to it at runtime after all of its minions died.

This module is only available in the [ultimate](#) edition of the Wise Feline package or the separate [Influence Maps package](#)

Influence maps module

[Great video about influence maps by Dave Mark](#)

Agents need to know what is going on around them in the world to be able to make good decisions. Having a toolbox which allows you to know spatial information about the world like:

- where are the main paths which enemies use?
- Where is the best position to throw my grenade?
- Where is the busiest area of the map around here?
- Where can I find some friendly forces to join?
- Where are the most trees located so I can spawn some rabbits?

is very useful.

Influence maps allow you to survey the environment of your game and find spatial info in a way which raycasts and octrees don't allow. An Influence map gives you info on what parts of the map something exists too much and on what part it doesn't exist. It can give you max and min points and busy and sparse parts. It can give you info not only on things which are present in the map but also about the events which happen before or will happen later.

The way influence maps work is that the map is a 2d array which agents propagate their influence on. For example you can have a map which all enemies propagate their position and attack range on and another where trees propagate their existence on.

Now agents can find out where they should flee to using the map points with smaller values in the enemies map and they also can find where they should fire at, by finding points with higher values.

You can search for higher and lower values in a circle in a specific part of the map or in the whole map. You can also combine multiple maps and then search on the resulting map so for example you can find a point which there are both lots of trees and enemies to set the trees there on fire to burn your enemies alive. Combining this system with AI tags, Smart Objects, The octree and Utility AI allows you to do very smart and immersive things in your games which most game designers only dream of.

Components and programming

In general this system can be used both as a set of components and also as a set of pure .NET classes without any components. The **InfluencerAgent** component allows you to propagate yourself on a map and the InfluenceMapComponent component allows you to create a map by just attaching it to a

GameObject. The [InfluenceMapView](#) component allows you to create a map by combining other maps and the [InfluenceMapCollection](#) component is a singleton which allows you to find maps by their name.

Each agent can propagate its value to the map but the value should be a circle with gradient values most of the time. It doesn't make sense to calculate all values every time. For this reason, you create a set of influence map templates which agents use as their stamp to propagate their value. In this way an enemy can propagate 1 in its position and reduce its influence value up to some distance like 15 meters to 0. The fall off of the value is chosen by the curve that you define in the template. The template values are calculated once and are just copied to the positions which need them afterwards.

To create your first influence map

- Right click in the project view and go to Create>NoOpArmy>WiseFeline>Influence Map Template and name your template linear. then choose a linear curve and a value of 50 for the radius.
- Add two gameObjects to the scene called agent and map.
- Add an [InfluenceMapComponent](#) component to the map and an [InfluencerAgent](#) to the agent. Name them map enemies (this is the name that you can use to get the map from InfluenceMapCollection).
- Select the agent GameObject and then select the map and the linear template for the agent. You can choose a map by dragging it or entering its name.
- Select the map GameObject and then set the size and resolution of the map to some values like 1000 for width and height and 1 for cell size.

Now if you press play and move the agent around, you can see the propagation of the agent on the map in scene view gizmos. If the gizmos are not shown, check the box to draw them in the [InfluenceMapComponent](#) component.

InfluenceMapComponent

This component allows you to create an influence map with the size and resolution specified and can draw the values of the map as gizmos in the scene view for you. It also has methods to search the map for values and also convert positions from map coordinates to world coordinates and vice versa.

The [InfluenceMapComponent](#) component has all methods you need in most cases but also gives you access to the map in the [Map](#) property and the map has all of the search functionality you need.

Influence maps are 2D arrays and they have a different resolution from your world so you need to convert positions of your GameObjects from world coordinates to map coordinates to use them with influence maps. Also when you get a coordinate from an influence map search or some other operation, you need to convert it from map coordinates to world coordinates so it can be used by your GameObjects.

InfluenceMapCollection

The `InfluenceMapCollection` singleton allows you to get maps by their name and maps created using the `InfluenceMapComponent` and `InfluenceMapView` components automatically register themselves to it. This component is created automatically if you don't have one in the scene.

How the search works

The search functionality takes a value and an enum which is a comparison operator. For example if you give it 0.6 and greater, then it will try to find a value bigger than 0.6 in the radius and stops as soon as it finds one. It starts searching from a random position in the specified search area so different agents don't end up choosing the same location.

Videos

These are the imap videos

<https://www.youtube.com/watch?v=hnnQ7eEkeCQ> <https://www.youtube.com/watch?v=9Lk9ibddw60&t=118s> <https://www.youtube.com/watch?v=O2bDqOW1i5c>
<https://www.youtube.com/watch?v=kuAwzUzzIg&t=19s>

This module is only available in the [ultimate](#) edition of the Wise Feline package

Smart Objects module

[Our video tutorial for this](#)

Agents need to know what objects are available in the environment and what they can do with them. We usually hard-code these facts in the agent code and look for specific objects and the agent code itself executes a behavior/interaction with the object.

Smart objects allow you to look for an object which can do a specific task/behavior or fulfil a specific need. Instead of the agent executing a behavior on the object, the object executes a behavior on the agent and dictates what it can do and how it is done.

For example you can look for things which can heal you or give you immunity and then both a healthpack and an armor can advertise themselves to you as objects which can get the job done. Then your AI can choose which one to use based on different parameters like how good they are or how dangerous it is to move to them.

Instead of the agent executing an animation to equip the armor and then change its own properties, The armor has a behavior which takes the agent as a parameter and gets executed when the agent calls `Use()` on the armor object. The method can then play an equipping animation on the agent and change its components to achieve the desired effect.

In this way Smart objects allow you a degree of flexibility which can be very useful for games with dynamic environments. For example if a character need wormth during winter and at the moment, clothing, fire and vehicles provide it with different qualities, your designers can easily add a magic hat to the game which advertises itself for giving the best type of wormth and it can have a behavior which attaches itself to the agent and executes an animation on the agent which puts then hat on the head of the agent. Then it plays the happiness sound effect of the agent. This all can happen without agent code being changed so it is very useful for mods, updates and in general very dynamic logic and is heavily used in games like the sims.

Smart objects can also have multiple slots which can be used by multiple agents at the same time. An obvious example is a car which can be used by multiple agents but also a bed or a house can be used like that.

Technical details

- A GameObject which is declared as smart object has a SmartObject component attached.
- A SmartObject component has a smart object definition assigned.

- A smart object definition asset is created from the right click menu of the project view and defines all the behaviors a smart object supports and all of its slots and their properties.
- The `SmartObjectManager` singleton manages all smart objects and allows you to query for smart objects which have specific behaviors and/or tags.

As said before a smart object is made of a set of behaviors and a set of slots which can be used by an agents. Each smart object slot can be reserved and then used by an agent and when it is used, Behaviors of the smart object get executed for that agent. Both the smart object and the agent can free a slot.

Smart Object slots

A slot is a reservable and usable part of a smart object. A freezer usually has 1 slot because it can only be used by one agent at a time while a car has probably 4.

Smart Object definitions

A smart object is defined by a `SmartObjectDefinition` scriptable object asset and is attached to a `GameObject` using a `SmartObject` component. The definition asset defines how many slots the object has and their properties and the component defines the behaviors which the smart object can do. A smart object should have at least 1 slot. Optionally each slot can have additional custom behaviors or only support the behaviors the main object supports. Also each slot has a position offset from the object it is defined for and the agent's ideal position from the smart object position can be set as well. You can even choose the agent's ideal rotation toward the slot.

Smart Object Behaviors

A smart object behavior can be defined by inheriting from the `SmartObjectBehaviorBase` class. You can override virtual calls for the time that an agent successfully reserves a slot of the object and when the agent uses a slot or frees it successfully. The agent's `GameObject` with the `SmartObject` component and the slot index are passed to the call and the behavior can execute whatever functionality it wants on the agent or the smart object itself.

Behaviors are attached to smart object's `GameObject` as a component and assigned in the `SmartObject` component.

The agent has no requirements and any `GameObject` can reserve and use smart objects. If an agent is destroyed which reserved or was using a slot of a smart object, the slot is automatically considered free.

This is the behavior base class's code.

```
/// <summary>
/// Behaviors of smart objects inherit from this class.
/// A smart object behavior operates on the agent which reserves, uses or frees a
slot in a smart objects. These events are defined in this base class as virtual methods
```

```

to override.

/// </summary>
public class SmartObjectBehaviourBase : MonoBehaviour
{
    /// <summary>
    /// Called when a slot of the object is successfully used by an agent
    /// </summary>
    /// <param name="agent"></param>
    /// <param name="smartObject"></param>
    /// <param name="slot"></param>
    public virtual void OnUsedByAgent(GameObject agent, SmartObject smartObject,
int slot)
    {

    }

    /// <summary>
    /// Called when a slot of the smart object is successfully claimed by an object
    /// </summary>
    /// <param name="agent"></param>
    /// <param name="smartObject"></param>
    /// <param name="slot"></param>
    public virtual void OnClaimedByAgent(GameObject agent, SmartObject smartObject,
int slot)
    {

    }

    /// <summary>
    /// Called when an agent frees the smart object
    /// </summary>
    /// <param name="agent"></param>
    /// <param name="smartObject"></param>
    /// <param name="slot"></param>
    public virtual void OnFreedByAgent(GameObject agent, SmartObject smartObject,
int slot)
    {

    }
}

```

As you can see there are three methods to override which give you all the information you need to execute your logic and since this is a component, you can get any other info you need from the inspector or other objects.

SmartObject component

You attach this component to each GameObject which you want to declare as a smart object. The definition is assigned to this component and the behaviors are assigned here as well. Behaviors can be both slot specific and general. The general ones work on all slots.

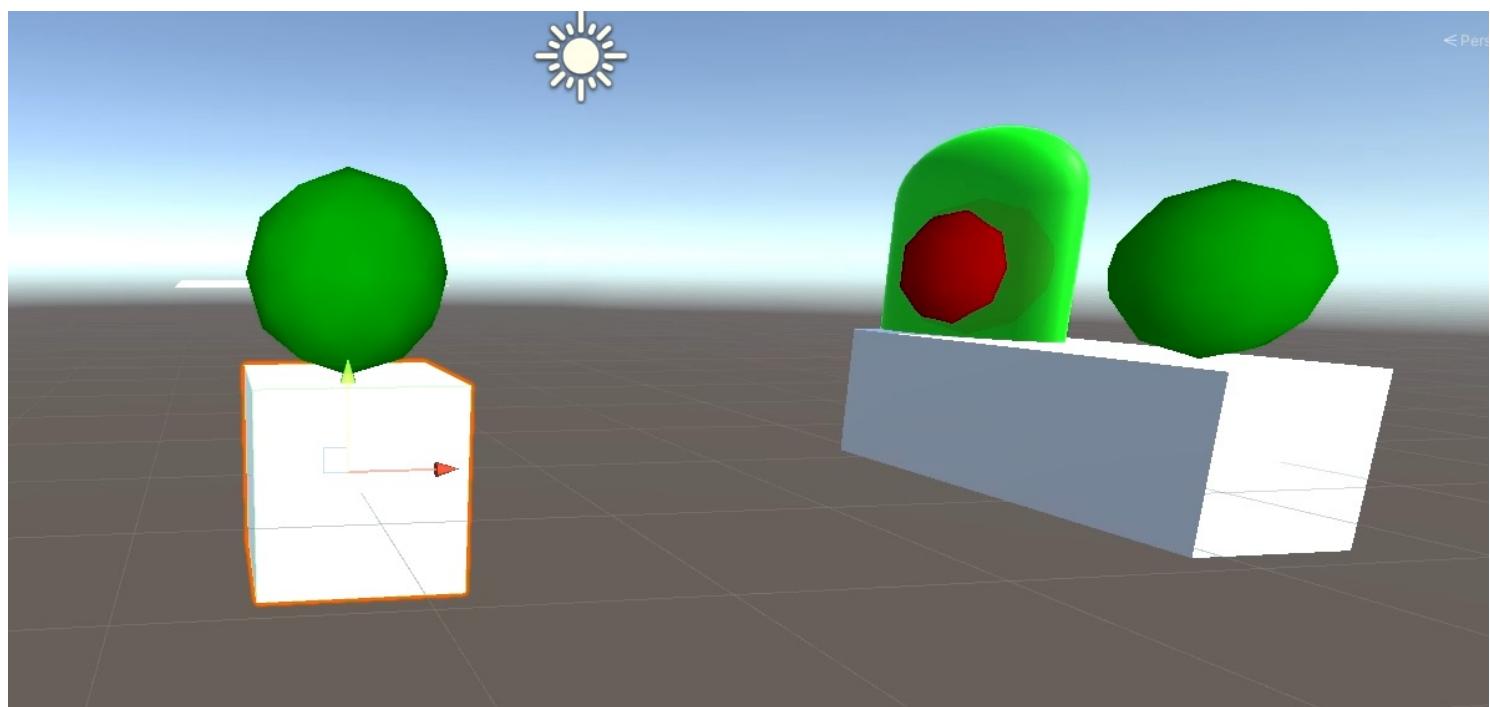
There are some other settings regarding tags which the smart object has, whether the object should update its position in the octree for queries periodically or not and ...

SmartObjectsManager

This singleton is created automatically if it is not attached to an object in the scene and allows you to find smart objects around a point and with a radius and filter the acceptable smart objects by the number of free slots they have and the behaviors they support. You can also filter them by including and excluding smart objects with specific tags. More details on how to query for smart objects are provided below.

Gizmos

Smart Objects have gizmos which show you where each slot is and if it is free (green), claimed (blue) or in use (red).



The object on the right has two slots which one of them is in-use by the capsule. the other slot and the slot of the other object are free so they are green. They are drawn on their location based on their offset too.

A simple demo

The demo folder of the smart objects folder contains a simple demo which has two agents, a car and a refrigerator. One of the agents looks for something to eat first and grabs the frig and the frig's eat behavior causes it to turn green and then after that finishes it searches for something drivable and goes to the car. The other agent at the start searches for something drivable and goes to the car. The car waits for both agents to sit and then moves to a destination.

Let's go through the different parts together.

Smart Object behaviors

We have only two behaviors defined for this demo. Eating and driving. Here is the eating behavior

```
/// <summary>
/// This behavior give the agent some food to eat
/// This is a simple demo so we just change the color of the agent to green
/// </summary>
public class EatFoodBehaviour :
NoOpArmy.WiseFeline.SmartObjects.SmartObjectBehaviourBase
{

    /// <summary>
    /// Called when the agent reserves this so nobody else touches it
    /// </summary>
    /// <param name="agent"></param>
    /// <param name="smartObject"></param>
    /// <param name="slot"></param>
    public override void OnClaimedByAgent(GameObject agent, SmartObject smartObject,
int slot)
    {
        base.OnClaimedByAgent(agent, smartObject, slot);
        //Just change our own color to indicate reserving happend
        GetComponent<Renderer>().material.color = Color.blue;
        agent.transform.LookAt(transform.position);
    }

    /// <summary>
    /// Called when the agent using us frees us
    /// </summary>
    /// <param name="agent"></param>
    /// <param name="smartObject"></param>
    /// <param name="slot"></param>
    public override void OnFreedByAgent(GameObject agent, SmartObject smartObject,
int slot)
```

```

    {
        base.OnFreedByAgent(agent, smartObject, slot);
        //Just change color and size to defaults
        GetComponent<Renderer>().material.color = Color.white;
        transform.localScale = Vector3.one;
    }

    /// <summary>
    /// Called when the agent uses us
    /// </summary>
    /// <param name="agent"></param>
    /// <param name="smartObject"></param>
    /// <param name="slot"></param>
    public override void OnUsedByAgent(GameObject agent, SmartObject smartObject,
int slot)
{
    base.OnUsedByAgent(agent, smartObject, slot);
    //Just do change some colors on ourself and the agent so we can see the object
    //is used and the agents which ate before driving are observable
    agent.GetComponent<Renderer>().material.color = Color.green;
    agent.transform.Translate(0, 0, 1);
    transform.localScale = Vector3.one * 0.5f;
    //free the object
    smartObject.Free(slot, agent);
}
}
}

```

As you can see this behavior just changes colors of itself and the agent to help you observe it is doing something. When claimed, it turns blue and when freed it turns white. Also when used it turns the agent to green to show it is not hungry anymore.

the driving behavior is more involved but essentially is no different.

```

public class DrivingBehaviour : SmartObjectBehaviourBase
{
    /// <summary>
    /// The direction this will drive toward
    /// </summary>
    [Tooltip("The direction this will drive toward")]
    public Vector3 movementDirection;

    /// <summary>
    /// Driving speed
    /// </summary>
    public float speed = 5;
}

```

```

/// <summary>
/// The destination on the XZ plane
/// </summary>
public Vector3 destinationXZ;

/// <summary>
/// The slots which are claimed by a user. We use this to check if all slots are
filled or not.
/// We could use a counter or get all slots and see if all are claimed or not but
this can be useful later on
/// </summary>
private List<int> claimedSlots = new List<int>();

public override void OnClaimedByAgent(GameObject agent, SmartObject smartObject,
int slot)
{
    base.OnClaimedByAgent(agent, smartObject, slot);
    claimedSlots.Add(slot);
}

public override void OnFreedByAgent(GameObject agent, SmartObject smartObject,
int slot)
{
    base.OnFreedByAgent(agent, smartObject, slot);
    agent.transform.SetParent(null);
    claimedSlots.Remove(slot);
}

public override void OnUsedByAgent(GameObject agent, SmartObject smartObject,
int slot)
{
    agent.transform.SetParent(smartObject.transform);
    base.OnUsedByAgent(agent, smartObject, slot);
    //if all slots are filled, drive into the night
    if (AreAllslotsFilled(smartObject))
    {
        StartCoroutine(Drive(smartObject));
    }
}

IEnumerator Drive(SmartObject obj)
{
    //super simple driving logic
    var targetY = Quaternion.LookRotation(movementDirection).eulerAngles.y;
    while (Mathf.Abs(obj.transform.eulerAngles.y - targetY) > 0.05f)

```

```

    {
        obj.transform.Rotate(0, 90 * Time.deltaTime, 0);
        obj.transform.position += movementDirection * speed * Time.deltaTime;
        yield return null;
    }

    //Take control of the camera
    Camera.main.transform.SetParent(obj.transform);

    //Move toward a destination
    while (Vector3.Distance(new Vector3(obj.transform.position.x, destinationXZ.y,
obj.transform.position.z), destinationXZ) > 1)
    {
        obj.transform.position += movementDirection * speed * Time.deltaTime;
        yield return null;
    }

    //Arrived at destination. Free the camera and all agents
    Camera.main.transform.SetParent(null);
    obj.FreeAllSlots();
}

public bool AreAllslotsFilled(SmartObject smartObject)
{
    return smartObject.GetSlotCount() == claimedSlots.Count;
}
}

```

This behavior waits for all of its slots to get occupied and then starts moving at the direction specified in the inspector toward a destination and when arrived at the destination, frees all of its slots. One interesting thing it does is that it even takes control of the camera when it starts moving and this shows you the flexibility of this system where a behavior can do almost anything. In the video tutorial linked above, we quickly made a space ship using this behavior.

Agent code

Now let's take a look at the agent code.

```

/// <summary>
/// This script allows your demo agent to eat food and then drive somewhere.
/// you can choose for each agent to do either both or one of them or do neither which
is not useful.
/// </summary>
public class SmartObjectDemoPlayer : MonoBehaviour
{

```

```

/// <summary>
/// Should the agent go eat something before driving somewhere
/// </summary>
[Tooltip("Should the agent go eat something before driving somewhere")]
public bool shouldEat;

/// <summary>
/// Should the agent find something drivable and drive somewhere
/// </summary>
[Tooltip("Should the agent find something drivable and drive somewhere")]
public bool shouldDrive = true;

IEnumerator Start()
{
    if (shouldEat)
        yield return StartCoroutine(Eat());
    if (shouldDrive)
        yield return StartCoroutine(Drive());
}

IEnumerator Eat()
{
    List<(SmartObject smartObject, SmartObject.FilterResult data)> objects =
SmartObjectsManager.Instance.GetSmartObjectsAroundPoint<EatFoodBehaviour>
(transform.position, 30, true);
    if (objects.Count > 0)
    {
        int slot = objects[0].smartObject.GetFreeSlotIndex();
        if (slot != -1)
        {
            float w = 0;
            var initialPosition = transform.position;
            while (w < 1)
            {
                transform.position = Vector3.Lerp(initialPosition,
objects[0].smartObject.transform.position, w);
                w += Time.deltaTime / 2;
                yield return null;
            }

            var b1 = objects[0].smartObject.Claim(slot, this.gameObject);
            if (b1)
            {
                yield return new WaitForSeconds(1);
                var b2 = objects[0].smartObject.Use(slot, this.gameObject);
            }
        }
    }
}

```

```

        }
    }

IEnumerator Drive()
{
    List<(SmartObject smartObject, SmartObject.FilterResult data)> objects =
SmartObjectsManager.Instance.GetSmartObjectsAroundPoint<DrivingBehaviour>
(transform.position, 30, true);
    if (objects.Count > 0)
    {
        int slot = objects[0].smartObject.GetFreeSlotIndex();
        if (slot != -1)
        {
            float w = 0;
            var initialPosition = transform.position;

            while (w < 1)
            {
                transform.position = Vector3.Lerp(initialPosition,
objects[0].smartObject.GetAgentSlotPosition(slot).position, w);
                w += Time.deltaTime / 2;
                yield return null;
            }

            var b1 = objects[0].smartObject.Claim(slot, this.gameObject);
            if (b1)
            {
                yield return new WaitForSeconds(1);
                var b2 = objects[0].smartObject.Use(slot, this.gameObject);
            }
        }
    }
}

```

The main code which you need to pay attention to are the two coroutines for eating and driving. They both use the `SmartObjectsManager.Instance.GetSmartObjectsAroundPoint<T>()` method to find smart objects which support the behavior they want and then find the free slots of the smart object found by calling `GetFreeSlotIndex()` method of the `SmartObject` component. Then they try to claim and use the object with some movement code added on top.

The rest of the code in this is ordinary unity code and a quite simple one for that matter.

How these connect?

- You create smart object definitions by right clicking in the project window and then going to *Create>NoOpArmy>Wise Feline>SmartObject Definition*.
- ◦ You fill in the number of slots and their offsets in the definition.
- Then attach a **SmartObject** component to the GameObject you want to make a smart object and assign this definition to it.
- For adding behaviors you first write them and then attach them as components to the same GameObject.
- At last you assign the behaviors in the SmartObject component as general or slot specific behaviors.

The agent/player doesn't need any code other than its own behavior attached to it.

Remembrance

Remembrance is a memory and emotions module for your AI agents so they can remember what happened in the world and react to it. Your NPCs can hold grudges against players and other NPCs, return a favor or even try to seduce you or bribe you. They can remember what happened by whom to whom and where and when. You can also attach any custom data to your memories.

How it works

NPCs or any other objects in your game record their memories in a container and then the container can be searched for values with different criteria. They can also set float emotion values per object or in general.

Memory

There are two components that you can use or you can use the underlying data containers. Each of the listed components have a `AddMemory()` method which records a new memory to be remembered. They all also have multiple overloads of a querying method which gives you the closest memory to a specific time with additional arguments which limits the acceptable memories. For example you can return only memories with a specific target or of a specific type or both. There are methods caching the last recorded memory with a specific action id or target for specific common use-cases as well. They usually are of the form `GetLastXXX()`

Memory component

This is the component you want most of the times.

This component is more advanced and can remove memories from the middle of the list and uses a `List<T>` as its underlying data container. When the maximum capacity is reached, the first item in the list is removed after each addition. This component has the forgetting functionality where you can specify in each memory, after how many seconds it should be forgotten and the component forgets that memory after the specified seconds.

SimpleMemory component

This component is simpler and uses a `Queue<T>` as its underlying data container and when reaching capacity, dequeues the first item in the queue. This component cannot forget memories and cannot remove any memory other than the first one in the queue but if it is enough for your use-cases then have better performance in scenarios which item removal from the list becomes a bottleneck in your game.

Since queues don't have indexing, all searches use the `foreach` construct in C# and are slower than for loops. A foreach structure calls `MoveNext()` and `Current()` methods of the queue to get to the next object

and then read it.

MemoryData

The `MemoryData` class is the type that the methods in the component accept and return.

Properties of a memory

A memory is stored using the `MemoryData` class and it contains a set of properties which can be used for querying

- **MemoryType** This is an integer value which can be used to find out what type the memory is based on your custom types in your game. for example if your `additionalData` can be of multiple types, this can be used to see what type it should be casted to.
- **StartTime** The time that this memory started. The constructor sets this automatically.
- **EndTime** The time that this memory ended. Calling `End(float time)` on the `MemoryData` object sets this automatically.
- **ForgetAfterSeconds** If greater than 0 then the memory will be forgotten after this many seconds.
- **Position** The position that this memory happened in. If you need additional data like room name/level name/bounds, you can put them in additional data or other optional fields.
- **ActionType** This field is of type `System.Type` in .NET and allows you to set the type of the class which caused this memory as an action so you know for example if the shooting was using the `Spell` class or the `FireBall` class. This could be derived from other fields in theory but having it is handy specially in the Utility AI integration.
- **Initiator** The object which initiated the memory. You can put any unity object including GameObjects and components in this as a reference. It can also be null.
- **Target** The target of this memory, for example the killed object in a murder memory. This can be null if the memory has no targets.
- **ActionId** This can be used for a unique id of an action instance. For example if you want to store exactly which magic spell of an NPC hit you, you can store its id here and put the NPC itself in the initiator field and the spell type in the ActionType field.
- **AdditionalData** is any custom data that you want to attach to this memory. usually you should find out what this can be casted to based on `ActionType`, `actionId` or `memoryType`

Emotions

There is a component called `EmotionComponent` and an Emotion Collection asset which define what emotions exist and allow you to set and get their values. Emotions can be general or per object. for example your NPC might be generally sad but happy toward a tree they really like or they can have an emotion only for objects, like affection but it might not be meaningful in your game mechanics to have a general affection emotion for the agents.

All emotions are floats between 0 and 1 and define how they are combined with a similar emotion. For example if two objects share an emotion like hate toward each other, then you can ask them to align their hates so if I hate my friend by 0.9 and he hates me by 0.6, first of all we are probably not friends and secondly, there are methods to let our hates affect each other so both are set at the higher value or are averaged based on the settings in the emotions collection.

Debuggers

Both modules have their own specific debugger windows to help you see what values exist in an agent's memory or what emotions it holds.

Remembrance

The Wise Feline Den free package has a set of features helping you make better game AI and other systems faster. These are heavily used in our [other products like Utility AI and Influence maps](#).

We have

- Combat Scripts
- Movement Scripts
- Blackboards
- Data Recorder
- and more

All of these are available in the menu at the left and can be learned and used to your advantage. Check them to see what great stuff we are giving away for free. Level up your game with these awesome tools.

Cookbooks

The cookbooks will be showing you how to implement a specific mechanic, an algorithm or a game genre in Utility AI and other parts of our tool set in both Unity and Unreal.

Some features might be available in one of the engines or in one version of our plugin but not the other but even in those cases we describe potential alternatives or how you can resolve the problem. Don't get too fixated on having a specific feature in a specific way and instead focus on the behavior you want to achieve. This said we strive for feature parity and we almost always have it to a very large extent but the engines themselves are very different with different sets of features and philosophies. To see the rest of the series click [here](#).

Wise Feline Utility AI for different game genres

Different algorithms are more suitable in different places and here we want to take an overview of usefulness of utility AI for different game mechanics and genres. Of course all types of algorithms and toolsets can be used in many different situations but it is important to use what is more useful, more efficient or faster/easier to use for the job. We made this library and use it for our internal projects and are selling it too so you know our biases too and everybody else have theirs too but we try to be clear and objective as much as we can.

Features which Utility AI can add to almost all types of games

Live worlds

Utility AI allows you to make worlds which their NPCs live for themselves and have their own desires, goals and act in their own and their societies interest.

Deep Worlds

Having complex AI and live worlds would mean that the worlds become very deep and when you play in them, you feel immersed in a universe or situation or point in time unlike anything else you've ever seen.

Story telling systems

The best stories have conflict, events and situations. The best tool to create these situations is utility AI because it is not rigid and it can react to what happened in the environment. Specially if you combine it with tags, smart objects and influence maps then it can use the environment effectively and know a great deal about the world which helps it make decisions which seem real and in context.

Personality Changes

A character's personality usually changes in real life and in stories. We love to see these and these changes surprise us. they bring us pain and pleasure. We also get excited about them. In general when something unexpected or unknown happens, feelings of hope and fear increase.

Utility AI can easily make this possible without you trying too much to offer specific changes in the AI design. If your AI has personality attributes and the considerations check those attributes to decide what to do, then you can alter the behavior by changing the attributes.

Let's say you decide if a healer NPC goes near a danger zone to help a wounded soldier or not by checking its health and braveness and if he has health pack or not. With Utility AI you can easily change a healer's braveness the more it saves people or the more he doesn't get killed or even better when somebody important to the NPC dies or a dramatic event happens. Imagine a dragon attacking a city and you want it to cause all NPCs to get frightened, you can reduce the braveness of characters by half

and the very weak souls can remain in a shelter enough to die of starvation. It is very hard to code this without utility AI.

Player generated behaviors even for NPCs

Since utility AI by nature is very parameterized and all actions are considered all the time, if you allow players to define events, items or NPCs behavior by modifying their attributes then they can change behaviors enough that a new sort of user generated content is made.

Consider the healer example in the personality changes part, if a player can craft their own items and the item can increase people's braveness then you can have a pill which people take to get brave and this can value a lot. You can buy 100 of them for yoursoldiers. That is if you think it is a good strategy of course.

To keep the game balanced you can increase the cost of larger changes which the items affect or require lowering of character XP for the crafter or ask them to sacrifice something big/do a hard quest for the ability to make powerful items. Let's say there is a crafting ability and the first increase in braveness requires one reduction in crafting but the second increase requires 2 and then third one requires 4 and it goes up exponentially. In this way you can enable players to make content for the game in a way that is very meaningful and valuable but without utility AI designing and using these sorts of things is very hard.

Easy behavior modification

Adding new behaviors or modifying the current ones is much easier with Utility AI since you can add the action to the list of actions and order doesn't matter. Other than order, the fact that you add it to a single place and you cannot forget a transition makes it easier too.

The AI still needs to have correct considerations and curves for a correct score but this exists for every action no matter when it is added and also allows you to compartmentalize the score calculation per action which helps. Every consideration is explicit so if the new action should not happen when the character has a gun, no order or lack of existence of a transition indicates that implicitly but the action itself has a consideration for that. Every programmer or designer with experience knows explicit things are easier to know about and change since they are in front of you and you cannot miss them.

Natural behavior

Because Utility AI considers all actions all the time, the behavior tends to become more natural and considering all the context which is possible but harder to get using other ways of making AI.

Immersive Worlds

When characters of a world act as if their own life matters and the events in the world really change things around, you feel that things are real and get immersed in it. Specially when events have lasting

impact, you take them more seriously. Utility AI helps you to incorporate these things in a much easier way than AIs with rigid structures.

Usually events either change attitudes and personal characteristics of people or change their environment and utility AI can easily deal with both thanks to curve shape changes, reading of attributes from variables dynamically and also things like tags and influence maps which help it to make decisions based on the environment. If you want to cause players to be afraid of a place which is attacked on a regular basis, it is enough to put this data somewhere in a knowledge base and then add actions which do something in relation to these sorts of data and modify other actions based on them.

For example, bring water home action can avoid those areas and guards can guard toward that direction more heavily but also running away action can be added for the time people are in the dangerous areas at night. Doing this in a scripted AI system with rigid structures like behavior trees or state machines is really hard and tuning something like a goal oriented system for correct behavior is next to impossible. This becomes much harder if you want these to be dynamic and not predefined.

Utility AI helps you to compartmentalize each action's considerations and scoring mechanism and easily make them effected by these events and other changes. It is still work to do but much less than the other options.

Examples in Different game genres

Third person shooters and FPS games

Shooters did not need very complex AIs back then where the actions were limited and the cover points were pretty well-known and static but these days that the maps are dynamic and the number of weapons is huge. It would be very helpful if your AI could decide its action based on some heuristics to see what action is the best in this moment in time.

Utility AI allows you to do that so your NPCs can play and decide like a human does and appear very smart and interesting to play against, by your actual players. The NPC can surprize the players and force them to apply new tactics just like when they play against real humans.

Adventure games

The main point of most adventure games is to make you feel you are going through a journey and the more natural and immersive it becomes, it's better. Utility AI allows you to make NPCs which are not very simple and stupid and have their own desires, have personalities and modify their behavior based on what you will do and what happen.

RPG games

Roll playing games intend to simulate a situation which you can immerse yourself in. their ultimate goal is to help you feel that you are in an environment or situation. However the AI for NPCs fall short of this

by having stupid dialog or a very strict and limited behavior. Utility AI is the cure for this to give the NPCs their own desires and lives.

Utility AI allows you to give your NPCs lots of behaviors and desires so they act in a believable manner. You can give them considerations which checks what other NPCs and players are doing and the personality of the NPC itself when it wants to choose what to do.

In RPGs events and quests matter a lot. Utility AI can help your NPCs a lot to react based on what events and quests have happened or are going on, and act based on those. Another good thing is that if a designer adds a new quest which causes say a rain of fire, it can easily add an action to the list of actions for NPCs to go home or to a shelter when this happens without getting worried that if it is executing in the correct place in a behavior tree, or if all needed transitions are added in the FSM of the characters.

Card and board games

This genre also benefits a lot from being able to code the AI in the way that a human thinks. Imagine that you are making a poker or blackjack. How the AI should decide what card to play next or what to do in general? Should it draw another card or not? Utility AI allows you to encode how a player plays into your AI.

You can code considerations which highlight what situations a card is a good one to play. Then you allow them all to compete with each other and the best action for the situation is chosen. You don't have to exactly say when this action should happen which is very hard when you don't know what the opponent will exactly do. In a simple platformer you have a set of limited actions which you will work against but in a card game or a board game either you have to have a huge amount of memory and analyze the complete space of the game which works with games with less randomness like chess or go but not as well for games with randomness in their rules but a very good AI which is fun to play with can be made using actions and considerations relatively easily if you research how human players think when choosing their next move and how do they assess usefulness of each action.

Simulation games

This is the genre that had one of its most famous games made using Utility AI. The sims uses Utility AI and you can find lots of materials about it online from GDC talks to university papers. Utility AI is literally simulating how people make decisions so they are best suited to simulate how intelligent entities behave in a world. All other algorithms should try to mimic how the brain works but utility AI is simulating it directly. It is possible to mimic the behavior with other algorithms partially but utility AI is the best way to do this.

MMOs and MMORPGS

MMORPGs are essentially role playing games so whatever applies to RPGs applies to these as well. Only worlds are deeper and expectations are higher because NPC behaviors are compared to human behaviors and they usually are immersion breakers and not immersion makers. Utility AI can help you

break this and make NPCs which are deep, have many actions and act according to their interests and what happened in the world and not based on a simple and small tree. Also you can share the behavior between NPCs with minor personality changes since the behaviors are not rigid anyways and just different NPCs decide a bit more differently compared to others.

A brave soldier and his coward comrade can use the same behavior but the brave one have a higher accuracy of attacking and tends to choose more risky behaviors.

Sandbox games

Sandbox games are known for lots of choices and possibilities. A good AI for a sandbox games should be able to deal with unexpected situations and act smartly or interestingly in them. The best AI to react to dynamic unknown situations is Utility AI.

Survival games

Again a survival game is a specific sort of sandbox game which somehow is an RPG too which is about surviving so whatever which applies to them, will apply to these as well.

Metaverses or metaverse games

The idea of the metaverse is that different experiences made by different players can be mixed up and interact with each other. The only AI algorithm which can work in an unexpected environment given that the way it calculates considerations and execute actions is compatible with the environment is utility AI. The rules are not hard coded and are softly specified so you check if any enemy object is around and shoot at them and having low high health or lots of enemies and no cover gives the shooting action a high score. The only key is to identify enemies in a way that all experiences can specify it. Using tags is a good way of doing this. If that is the case that all enemies of your NPC/agent have an enemy tag attached to them then your AI can react to them even if it doesn't know how exactly they would behave. Faction names or even a memory of what actions an agent took can be used for this as well. This will not be perfect and not as good as when you exactly know what will be the behavior of those enemies but still Utility AI is the only system which allows you to do this.

In theory any AI which checks conditions which are calculated based on dynamic parameters will be more adaptable to experiences which not all of them are authored by you. but in action a state machine or behavior tree or even a planner will only work if the conditions are known to produce the best result for that situation. Their transitions have order and are more explicit. This can be helpful in a very scripted situation but not when the situation is unknown or has too many parameters. They don't work based on heuristics and that is the major difference between them and utility AI.

If you have a state machine which transitions from a state based on heuristics and not only when it is sure that this specific transition to another state is the best one then it is effectively limited utility AI which each action can only be switched to a selected number of others and not all of them. Also if a behavior tree's nodes are selected in a similar manner then it is the same thing and the only difference is

that the actions and their score comparisons have implicit orderings and each action will only activate after a threshold or if you basically have a root with all actions as its children evaluated in parallel and then one selected based on their score, then it is utility AI.

In the same spirit, Utility AI can be turned into a behavior tree or state machine or can become a bit more rigid in its structure if you use appropriate considerations. Imagine a consideration which gives a score higher than 0 for an action only if the currently executing action is shooting, that can be used to have some actions effectively only available in the graph of your decisions as children of the shooting action. Usually such a graph doesn't exist in utility AI or at least it is not easy to draw but if you want to turn your utility AI into that, you can.

But that is not the point. The point is not to show each AI algorithm can more or less implement the others if you stretch it with some degree of success. It is not a useful practice and it is better to use the tool which is suitable for that specific sort of way that you want to offer your AI behaviors. And in this sense and compared to other known algorithms, utility AI works best in unknown environments which the decisions don't have definitive answers. Wait the real world is like that! :) And this is another difference of utility AI compared to the others. It is derived from a theory which tries to decode human behavior in the real world which what we want to simulate and not an abstraction created to be able to model a decision tree with accuracy.

This is one of the cookbooks showing you how to implement a specific mechanic, an algorithm or a game genre in Utility AI and other parts of our tool set

Cool downs

Once in a while you need to implement a cooldown for an action so it doesn't execute for x number of seconds after it executes once. To implement such a mechanic you need to remember when an action got executed and how much you should wait before you can execute it again. To do this the action should have a consideration which returns 0 before the cooldown time finishes and returns 1 after that so the rest of the score is not affected by it.

It can be easily hardcoded if you want to do it for a specific action. Let's call the action Attack1 and let's say we want to execute it at most once per 5 seconds so with 5 seconds of cooldown. You can store a last executed time in the action itself and set it to current time when action gets activated and then have a consideration which subtracts the current time from the stored time in the action and if less than 5 returns 0 and otherwise returns 1. The curve for the consideration should return 0 for 0 and 1 for one as well so a linear curve would be perfect.

The method to override in Unity is `OnStart()` and in Unreal it is `OnActivated()`. These get called when an action is selected to execute once.

This said you can implement it more flexibly by using a blackboard and in this way you can have an item which if the character eats or an action which he does and reduces the cooldown. For example he can eat a potion to reduce his tiredness and Attack1 requires tiredness of less than 10. Let's say tiredness is between 0 and 100 and it decreases when not attack 15 per second. Attack1 increases tiredness by 90 as well. There is also a pray action which has a 50% chance of reducing tiredness to 0 or increasing it by 10.

There is a consideration for the Attack1 which checks the blackboard float key of tiredness and if less than 10 returns 1 and otherwise returns 0. A code in a components of the agent decreases tiredness by 15 per second if the attack actions are not being executed. Eat fruit action decreases tiredness by say 50 and its consideration give it a high score when you are tired and a fruit is close by. The pray action gets a random number and if less than 0.5 then it increases tiredness by 5 and otherwise sets it to 0. Its consideration will see if there is no fruits close by and we are tired and the situation requires a not tired character.

Now to make it more interesting, some characters which have a more religious/spiritual attribute can use this more effectively and we can have a cool down for praying itself so an agent can use it once per few battles to not make the battle scenes look like a church in chaos. Also if this was based on player choices then it could improve based on how many temples a player built in a city building game or how much did the player sent his army to rick havoc on cities in a strategic game. To be clear, sending armies to

ruin cities should decrease the effectiveness of prayers and other spiritual spells and actions in most games unless you are making a game about an evil God.

Remembrance

The Wise Feline Den free package has a set of features helping you make better game AI and other systems faster. These are heavily used in our [other products like Utility AI and Influence maps](#).

We have

- Combat Scripts
- Movement Scripts
- Blackboards
- Data Recorder
- and more

All of these are available in the menu at the left and can be learned and used to your advantage. Check them to see what great stuff we are giving away for free. Level up your game with these awesome tools.

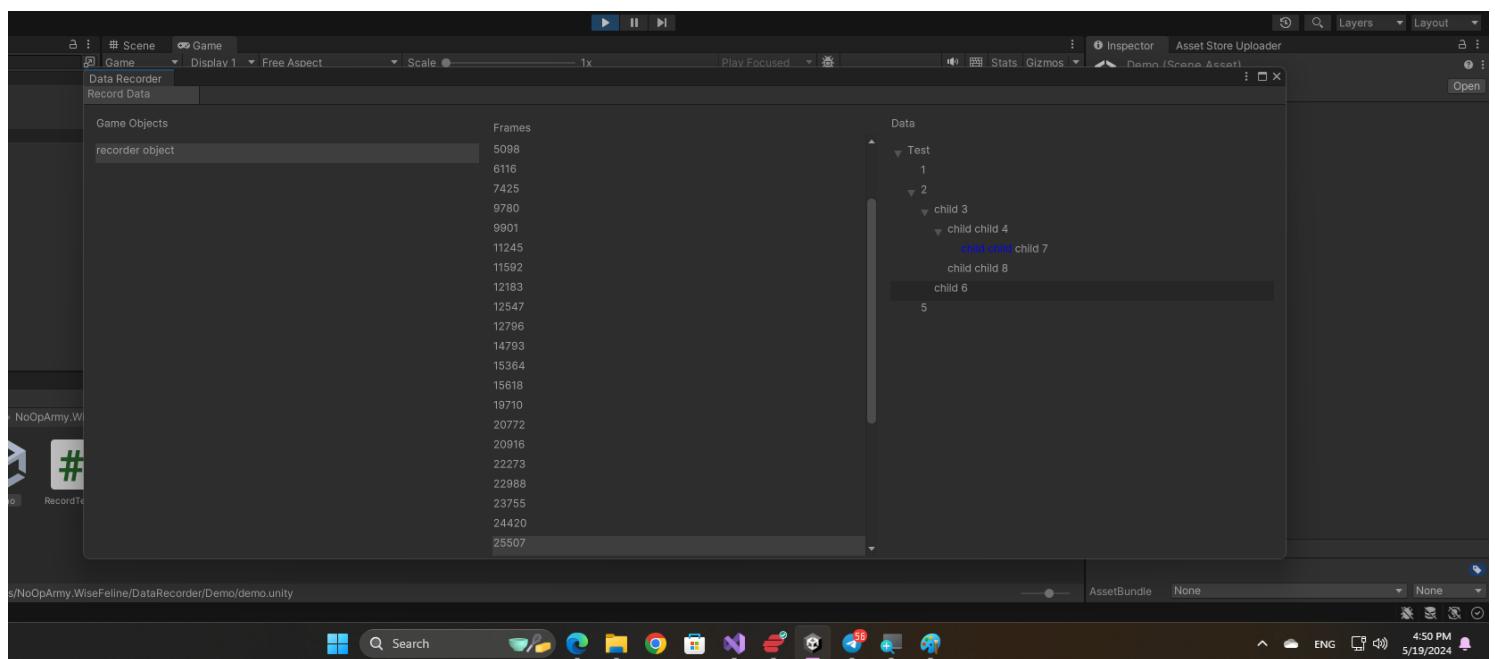
Data Recorder

There Data Recorder allows you to record data from GameObjects over time and then look at them during or after the play session.

The window can be opened by going to the *Window>NoOpArmy>Data Recorder* menu. You have to attach **DataRecorder** component to GameObjects you want to record and your other scripts can use it to record data every frame.

The data format

The data is a set of text records which can use ritch text and can have a level. They are shown in a tree like structure.



As you can see here, the left most panel is the list of objects with data recorders on them, the middle panel is the frames and the right side one shows the data. The data is a tree of sub-records created based on levels you give to them.

How to use

After attaching a **DataRecorder** component, you can get it and call its methods to add records and their sub-records.

You can use **GetCurrentFrameRecord()** method of the DataRecorder to get/create the frame data for the current frame. Any frame you call this, you'll have an entry in the data recorder. The index this function returns can be used to add sub records which are shown in the data section.

The **RecordTester** script in the demo folder of the **DataRecorder** looks like this.

```
private DataRecorder recorder;

void Start()
{
    recorder = GetComponent<DataRecorder>();
}

private void Update()
{
    if (UnityEngine.Random.value < 0.999f)
        return;
    int index = recorder.GetCurrentFrameRecord();

    var sub0 = new SubRecord();
    sub0.text = $"Test";
    sub0.indentationLevel = 0;
    recorder.AddSubRecord(sub0, index);

    var sub = new SubRecord();
    sub.text = $"1";
    sub.indentationLevel = 1;
    recorder.AddSubRecord(sub, index);
    var sub2 = new SubRecord();
    sub2.text = $"2";
    sub2.indentationLevel = 1;
    recorder.AddSubRecord(sub2, index);

    var sub3 = new SubRecord();
    sub3.text = $"child 3";
    sub3.indentationLevel = 2;
    recorder.AddSubRecord(sub3, index);

    var sub4 = new SubRecord();
    sub4.text = $"child child 4";
    sub4.indentationLevel = 3;
    recorder.AddSubRecord(sub4, index);

    var sub7 = new SubRecord();
    sub7.text = $"<color=blue>child child</color> child 7";
    sub7.indentationLevel = 4;
    recorder.AddSubRecord(sub7, index);

    var sub8 = new SubRecord();
    sub8.text = $"child child 8";
    sub8.indentationLevel = 3;
    recorder.AddSubRecord(sub8, index);
```

```

var sub6 = new SubRecord();
sub6.text = $"child 6";
sub6.indentationLevel = 2;
recorder.AddSubRecord(sub6, index);

var sub5 = new SubRecord();
sub5.text = $"5";
sub5.indentationLevel = 1;
recorder.AddSubRecord(sub5, index);
}

```

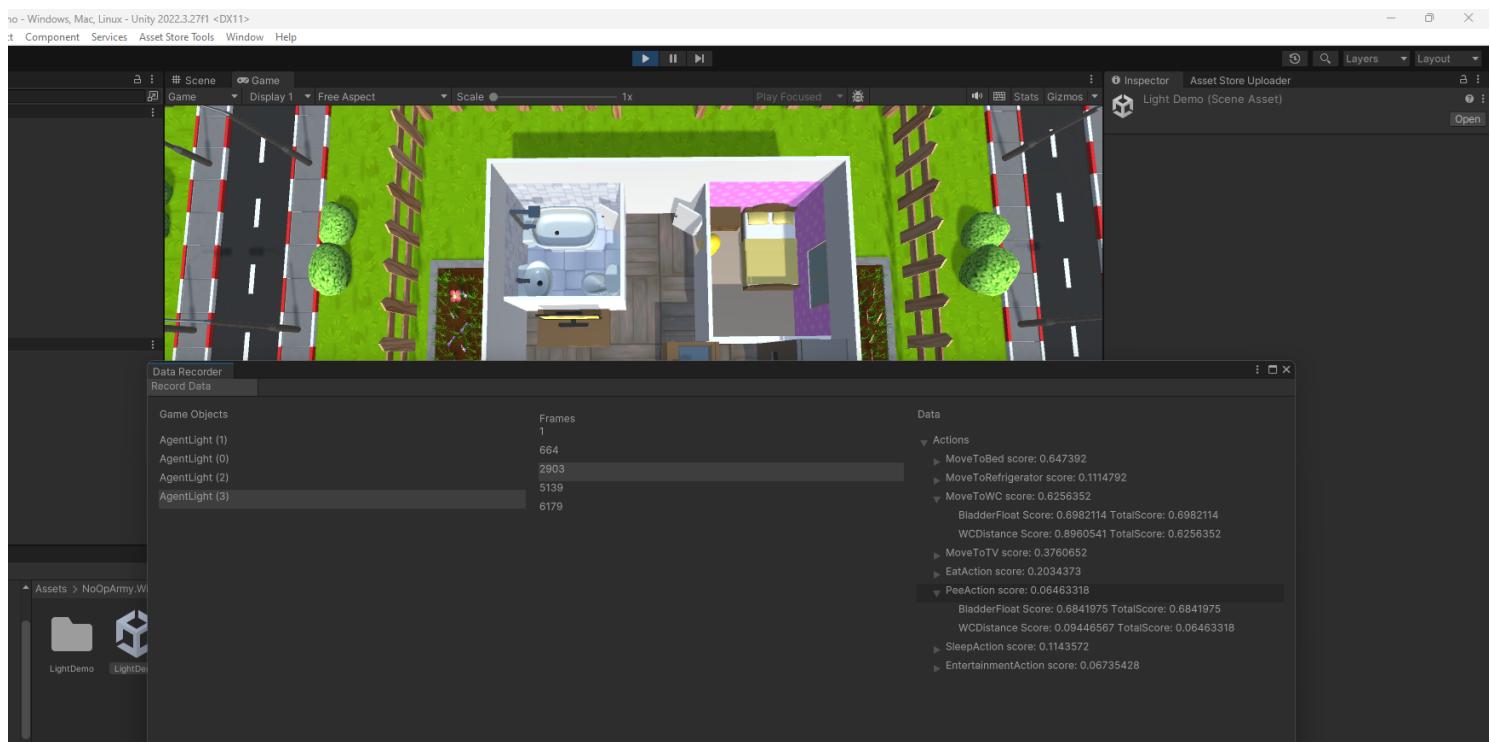
As you can see `AddSubRecord()` is used to add sub-records per frame and each sub-record has a level which tells which level of the tree it should be in. If you log an attack and then 3 related info about it, then their level should be attack's level + 1. Think of it as indentation level.

Also as you can see, you can use unity's `<color=blue>text</color>` and other tags to make your text color something special or make it bold/italic/ any other thing which unity supports.

You can integrate this into any system of your game which can benefit from recording hierarchical data over time and being able to look at it afterwards.

Utility AI integration

The `Brain` component integrates with Data Recorder so if you attach a `DataRecorder` to a `GameObject` with a `Brain` then the actions and considerations and their scores will be recorded whenever the brain thinks.



If you don't have our other packages, we encourage you to take a look at them to add very [immersive AI](#)  to your game.

Blackboards

[Blackboards video tutorial](#)

A Blackboard is a data structure which allows you to share data between different components of your AI without having direct dependencies between them. A blackboard is a **Dictionary** like data structure which allows you to add data by defining it as a set of key value pairs.

Let's say you want to know what is the safest location to go to, you can define a safe location key in the blackboard and store the **Vector3** of the location in it. Our implementation of blackboards is more strongly typed than a **Dictionary<string, object>** and you should define all keys with the data type of their values before being able to set/get them.

Getting Started

To define a blackboard you first need to create a blackboard definition. To do that right click in your project and click on Create>NoOpArmy>Wise Feline>BlackBoard Definition. Then you need to assign that definition to a **BlackBoard** component. Then you can set or get the value of the keys you defined in the definition.

You have multiple methods in the **BlackBoard** component like **GetFloat(key)**, **SetFloat()**, **GetVector3(key)** and **SetVector3(key, value)**. You have these pairs for all data types and you can set and get the values in your code as you need.

How can I use this?

Your AI can use this in considerations and actions and other parts of your game. Let's say you have an action like, GoToLocation, you can take a blackboard key which you know contains a **Vector3**.

A small tutorial

- Create a definition and call it b.
- Add a float key called speed.
- Create a new script called **TestBlackBoard** and attach it with a **BlackBoard** component to a **GameObject**.
- Set the definition in the blackboard to the one you made above.
- Open the **TestBlackBoard** script you made and add some code like this.

```
public class TestBlackBoard : MonoBehaviour
{
    private BlackBoard board;

    void Start()
```

```
{  
    board = GetComponent<BlackBoard>();  
    board.SetFloat("speed", 3);  
    var speed = board.GetFloat("speed");  
    print(speed);  
}  
}
```

The code should print 3.

Movement scripts

You should be able to move your AI characters easily and we implemented an interface with two implementations for it. There is

- NavMesh based one.
- CharacterController based one.

The interface is `IAIMovement`

```
/// <summary>
/// This interface is implemented by all AI movement types which allow an agent to choose a
target and move toward it.
/// </summary>
public interface IAIMovement
{
    /// <summary>
    /// Should contain the movement destination
    /// </summary>
    Vector3 Destination { get; }

    /// <summary>
    /// Moves the character to a position specified by a Vector3 and calls a callback
when arrived
    /// </summary>
    /// <param name="position"></param>
    /// <param name="callback"></param>
    /// <returns>True if the movement can happen, false otherwise. Path Finding failures and
other issues can prevent the movement to happen.</returns>
    bool MoveToPosition(Vector3 position, Action callback = null);

    /// <summary>
    /// Moves the character to the position of a GameObject and calls a callback
when arrived
    /// </summary>
    /// <param name="position"></param>
    /// <param name="callback"></param>
    /// <returns>True if the movement can happen, false otherwise<. Path Finding failures
and other issues can prevent the movement to happen.</returns>
    bool MoveToPosition(GameObject position, Action callback = null);

    /// <summary>
    /// Stops the movement no matter if we arrived at the set destination or not
    /// </summary>
```

```
    void StopMoving();  
}
```

CharacterControllerBasedAIMovement

This one should be used if your NPC has a [CharacterController](#) attached.

NavmeshBasedAIMoement

You should this component to move your character if it has [NavMeshAgent](#) component attached.

Combat Scripts

We've implemented a few scripts so you can implement range and melee attacks with different types of health and damage structures with armor and other stuff. The scripts are pretty simple and easy to use and customize. We are going to list them below.

All of these scripts are in DenUtilityComponents folder in the Den folder

Attack

This component has methods for doing range and melee attacks and it can either apply the damage to all hit objects or just the first one. It can optionally change the material color of the GameObject which it is attached to when the attack starts and can delay the actual attack so the player has a chance to respond to the telegraphed attack.

This is a sample signature of one of the methods `Coroutine MeleeAttackWithDelay(Vector3 offset, float radius, float damageAmount, float castDelay, Action Done = null, bool applyToAllHits = false)`. The offset is offset from the object's own position. There are range attacks too.

The way checks work

The checks use no layers and use `CheckSphere()` like this `int count = Physics.OverlapSphereNonAlloc(position, radius, results);` so it does not allocate.

In the future we can support layers or other shapes if needed.

IDamagable

Any script which wants to take damage from this system should implement this interface. It only has two methods for increasing health/healing and applying damage.

```
/// <summary>
/// Any component which wants to receive damage can implement this interface
/// </summary>
public interface IDamagable
{
    /// <summary>
    /// Applies damage to the GameObject
    /// </summary>
    /// <param name="amount"></param>
    /// <param name="damager"></param>
    void ApplyDamage(float amount, GameObject damager);

    /// <summary>
```

```
/// Heals the GameObject
/// </summary>
/// <param name="amount"></param>
/// <param name="healer"></param>
void IncreaseHealth(float amount, GameObject healer);
}
```

The [Health](#) component described below does implement this interface.

Health

This component implements a health system for your game objects which need it. For effects it only can change material colors for debugging purposes but all of its methods are overridable and whenever the health changes it fires its [OnHealthChanged](#) events.

It can automatically destroy the GameObject when the health reaches 0 and it can add a delay before doing this.

Other Scripts

These are a set of useful scripts which are not in a specific category

Faction scripts

The [Faction](#) component allows you to assign a faction to an agent. The faction is an int id but there is also a scriptable object to define each id's faction.

You create the scriptable object using NoOpArmy>Wise Feline>Faction Collection menu and each faction has an id, a name and a description.

The Id in the faction collection should be the same as the id in the [Faction](#) component. Later on we will add more to this system by adding a faction manager system and ...

NPCZoneTrigger

This script can be used by NPCs to know when a player or another NPC enters their zone. You attach NPCZoneTrigger to a trigger and set the NPC which it should notify and then if any GameObject other than that NPC enters/leaves the collider, it notifies the NPC.

The NPC should implement an interface called [INPCZoneMessageReceiver](#) for this to work. This interface has two methods for the time that an object enters and leaves the zone.

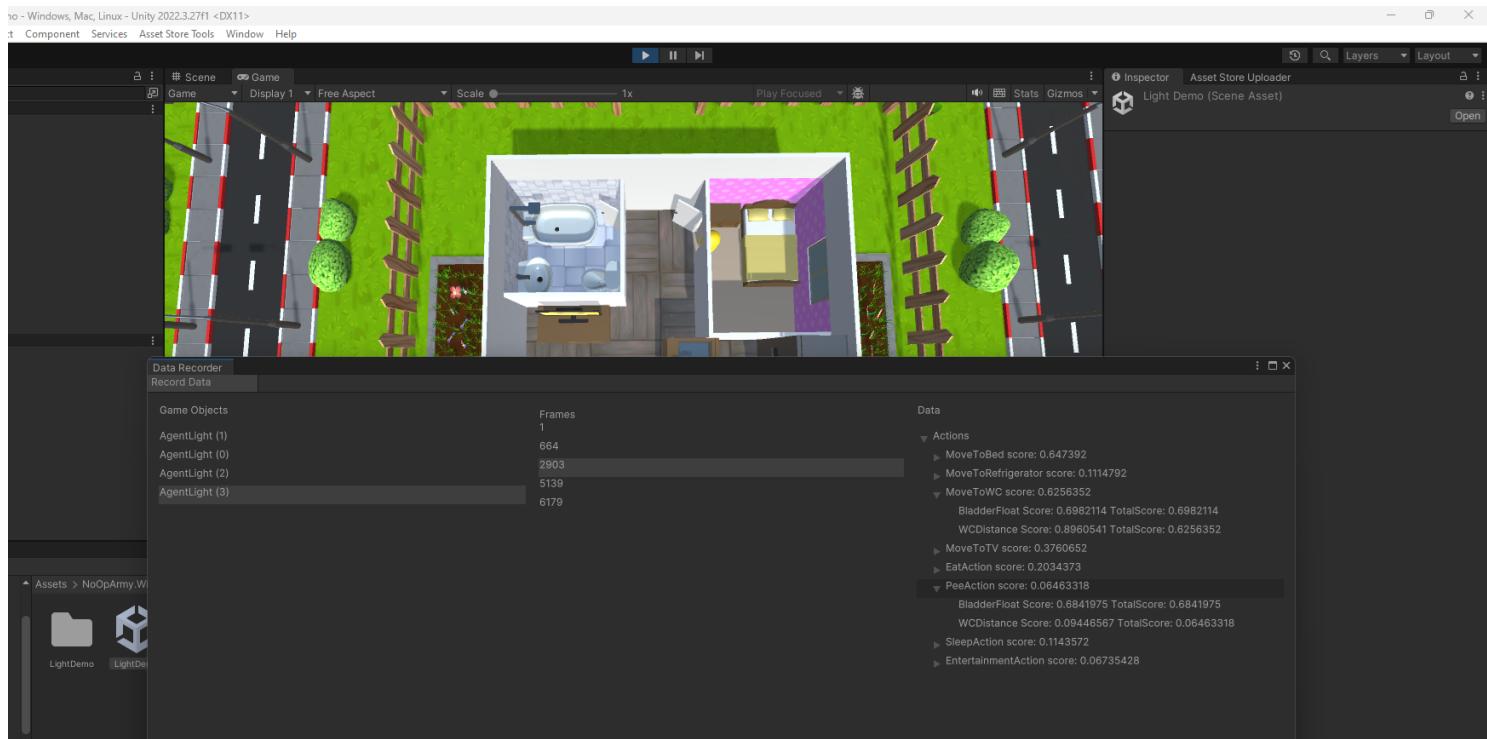
This can be used to create dialog for NPCs or set their blackboard values to know somebody else is in the area/left the area.

DestroyObjectWithDelay

Destroys the object with some delay. Mostly used for magic spells or projectiles to destroy them if a collision or something similar haven't destroyed them.

Utility AI integration

The **Brain** component integrates with Data Recorder so if you attach a **DataRecorder** to a **GameObject** with a **Brain** then the actions and considerations and their scores will be recorded whenever the brain thinks.



If you don't have our other packages, we encourage you to take a look at them to add very [immersive AI](#) to your game.

This module is only available in the [ultimate](#) edition of the Wise Feline package or the separate [Influence Maps package](#)

Influence maps module

[Great video about influence maps by Dave Mark](#)

Agents need to know what is going on around them in the world to be able to make good decisions. Having a toolbox which allows you to know spatial information about the world like:

- where are the main paths which enemies use?
- Where is the best position to throw my grenade?
- Where is the busiest area of the map around here?
- Where can I find some friendly forces to join?
- Where are the most trees located so I can spawn some rabbits?

is very useful.

Influence maps allow you to survey the environment of your game and find spatial info in a way which raycasts and octrees don't allow. An Influence map gives you info on what parts of the map something exists too much and on what part it doesn't exist. It can give you max and min points and busy and sparse parts. It can give you info not only on things which are present in the map but also about the events which happen before or will happen later.

The way influence maps work is that the map is a 2d array which agents propagate their influence on. For example you can have a map which all enemies propagate their position and attack range on and another where trees propagate their existence on.

Now agents can find out where they should flee to using the map points with smaller values in the enemies map and they also can find where they should fire at, by finding points with higher values.

You can search for higher and lower values in a circle in a specific part of the map or in the whole map. You can also combine multiple maps and then search on the resulting map so for example you can find a point which there are both lots of trees and enemies to set the trees there on fire to burn your enemies alive. Combining this system with AI tags, Smart Objects, The octree and Utility AI allows you to do very smart and immersive things in your games which most game designers only dream of.

Components and programming

In general this system can be used both as a set of components and also as a set of pure .NET classes without any components. The **InfluencerAgent** component allows you to propagate yourself on a map and the InfluenceMapComponent component allows you to create a map by just attaching it to a

GameObject. The `InfluenceMapView` component allows you to create a map by combining other maps and the `InfluenceMapCollection` component is a singleton which allows you to find maps by their name.

Each agent can propagate its value to the map but the value should be a circle with gradient values most of the time. It doesn't make sense to calculate all values every time. For this reason, you create a set of influence map templates which agents use as their stamp to propagate their value. In this way an enemy can propagate 1 in its position and reduce its influence value up to some distance like 15 meters to 0. The fall off of the value is chosen by the curve that you define in the template. The template values are calculated once and are just copied to the positions which need them afterwards.

To create your first influence map

- Right click in the project view and go to Create>NoOpArmy>WiseFeline>Influence Map Template and name your template linear. then choose a linear curve and a value of 50 for the radius.
- Add two gameObjects to the scene called agent and map.
- Add an `InfluenceMapComponent` component to the map and an `InfluencerAgent` to the agent. Name them map enemies (this is the name that you can use to get the map from `InfluenceMapCollection`).
- Select the agent GameObject and then select the map and the linear template for the agent. You can choose a map by dragging it or entering its name.
- Select the map GameObject and then set the size and resolution of the map to some values like 1000 for width and height and 1 for cell size.

Now if you press play and move the agent around, you can see the propagation of the agent on the map in scene view gizmos. If the gizmos are not shown, check the box to draw them in the `InfluenceMapComponent` component.

InfluenceMapComponent

This component allows you to create an influence map with the size and resolution specified and can draw the values of the map as gizmos in the scene view for you. It also has methods to search the map for values and also convert positions from map coordinates to world coordinates and vice versa.

The `InfluenceMapComponent` component has all methods you need in most cases but also gives you access to the map in the `Map` property and the map has all of the search functionality you need.

Influence maps are 2D arrays and they have a different resolution from your world so you need to convert positions of your GameObjects from world coordinates to map coordinates to use them with influence maps. Also when you get a coordinate from an influence map search or some other operation, you need to convert it from map coordinates to world coordinates so it can be used by your GameObjects.

InfluenceMapCollection

The `InfluenceMapCollection` singleton allows you to get maps by their name and maps created using the `InfluenceMapComponent` and `InfluenceMapView` components automatically register themselves to it. This component is created automatically if you don't have one in the scene.

How the search works

The search functionality takes a value and an enum which is a comparison operator. For example if you give it 0.6 and greater, then it will try to find a value bigger than 0.6 in the radius and stops as soon as it finds one. It starts searching from a random position in the specified search area so different agents don't end up choosing the same location.

Videos

These are the imap videos

<https://www.youtube.com/watch?v=hnnQ7eEkeCQ> <https://www.youtube.com/watch?v=9Lk9ibddw60&t=118s> <https://www.youtube.com/watch?v=O2bDqOW1i5c>
<https://www.youtube.com/watch?v=kuAwzUzzIg&t=19s>

InfluenceMapView

An InfluenceMapView component allows you to create a map by combining other maps. For example you can make a threat view by adding this component to a GameObject and then give it the enemies map and bases map and tell it to add them. The InfluenceMapView can have a name and be found using the [InfluenceMapCollection](#) class as well. It also asks you how often it should be recalculated from the maps it has and there is no limit in number of maps and operations, however the more maps you add to it, the heavier it will be to calculate.

Adding, subtracting and other operations on maps

You can add two maps to each other to search on both of them combined. You also can subtract them from each other and do other operations as well.

These methods are available

- *Add* which adds another map to the map calling the add and either returns a new map or fills the data in the map you supplied.
- *Subtract* which subtracts another map from the map calling the subtract and either returns a new map or fills the data in the map you supplied.
- *Multiply* which multiplies a number with the map calling the multiply and either returns a new map or fills the data in the map you supplied.
- *Invert* which inverts the map calling the invert and either returns a new map or fills the data in the map you supplied.
- *Normalize* which normalizes the values of the map between 0 and 1 based on its minimum and maximum.
- *GetCellsArray* Returns the raw array that the map uses. Be careful to not modify this and just use it to calculate other operations which we did not write.
- *SetCellsArray* sets the array of cells of a map.

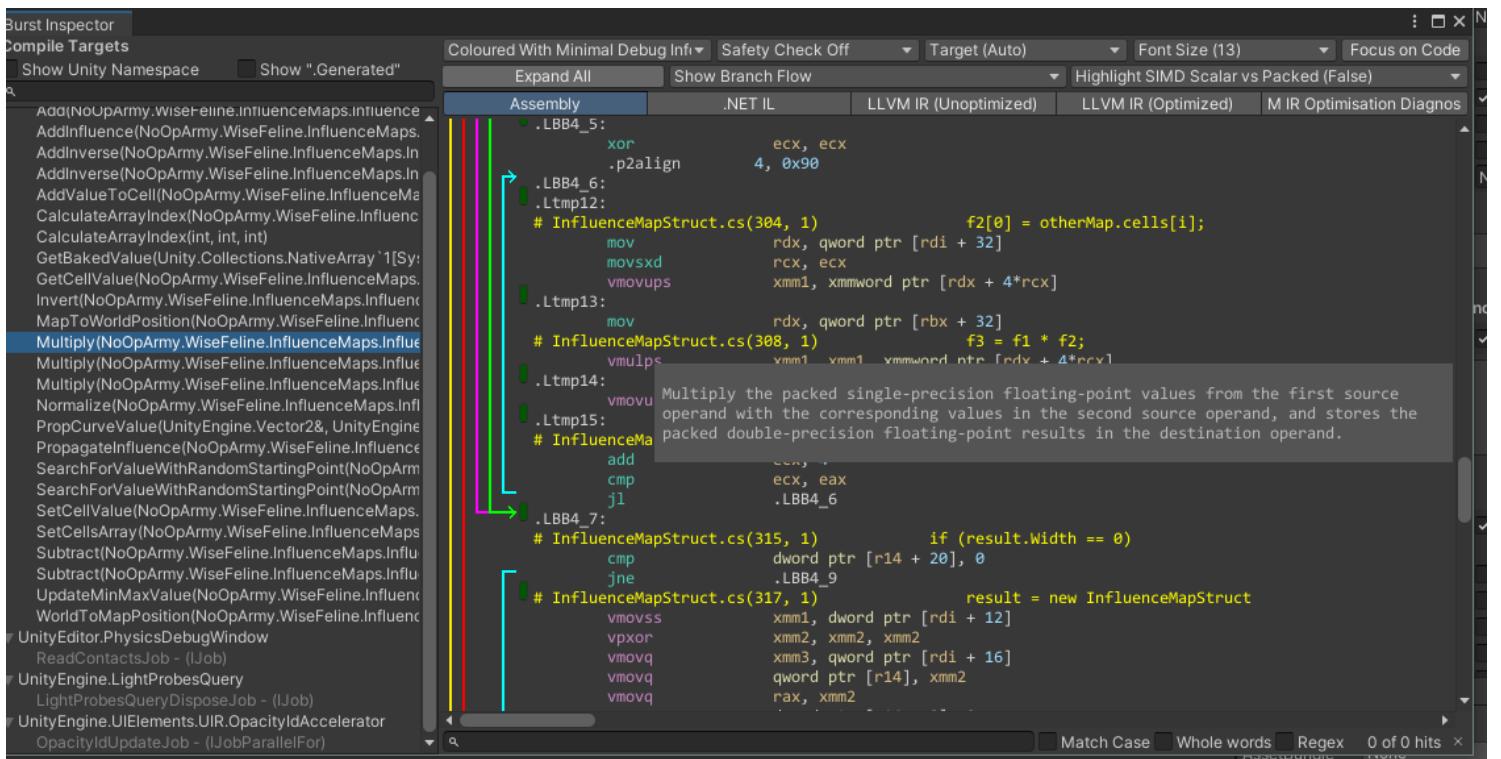
Burst support

Influence maps support burst and the operations become a lot faster. To enable burst support you need to be on Unity 2022 or later and do the following:

- Install burst in your project from the package manager.
- Define WF_BURST in the other settings part of the player settings of your project.

All of the components will work as before but instead of the `InfluenceMap` class they'll use `InfluenceMapStruct` struct under the hood. The struct has a reference to a native array and lots of burst compilable methods. You can technically use this in jobs and ECS/entities but we don't directly support any of them.

All of the Add/Multiply/... map operations use float4 and generate SIMD packed instructions which means doing multiple operations using a single instruction.



The screenshot shows the Unity Burst Inspector window. On the left is a tree view of Unity scripts, with several InfluenceMap-related methods selected. The main pane displays the generated assembly code. A tooltip is shown over the `vmulps` instruction at line 308, which performs a packed multiplication of single-precision floating-point values. The tooltip provides the C# code for the method and the assembly instruction, along with a detailed explanation of the SIMD operation.

```
Assembly
.LBB4_5:
    xor     ecx, ecx
    .p2align 4, 0x90
.LBB4_6:
    .Ltmp12:
        # InfluenceMapStruct.cs(304, 1)           f2[0] = otherMap.cells[i];
        mov     rdx, qword ptr [rdi + 32]
        movsxd rcx, ecx
        vmovups xmm1, xmmword ptr [rdx + 4*rcx]
    .Ltmp13:
        mov     rdx, qword ptr [rbx + 32]
        # InfluenceMapStruct.cs(308, 1)           f3 = f1 * f2;
        vmulps xmm1, xmmword ptr [rdx + 4*rcx]
    .Ltmp14:
        vmoveu Multiply the packed single-precision floating-point values from the first source
        operand with the corresponding values in the second source operand, and stores the
        result in the destination operand.
    .Ltmp15:
        # InfluenceMapStruct.cs(315, 1)           if (result.Width == 0)
        add     rax, rax
        cmp     ecx, eax
        jle    .LBB4_6
    .LBB4_7:
        # InfluenceMapStruct.cs(315, 1)           if (result.Width == 0)
        cmp     dword ptr [r14 + 20], 0
        jne    .LBB4_9
    .# InfluenceMapStruct.cs(317, 1)           result = new InfluenceMapStruct
        vmovss  xmm1, dword ptr [rdi + 12]
        vpxor   xmm2, xmm2, xmm2
        vmovq   xmm3, qword ptr [rdi + 16]
        vmovq   qword ptr [r14], xmm2
        vmovq   rax, xmm2
```

Sample utility AI integration

There is a consideration in the actions and considerations library folder which searches an influence map and writes the result in a blackboard and returns a score based on the fact that the search was successful or not. In this way the action doesn't have to recalculate the search and can use the value in the blackboard to do its operation.

Let's say you want to see if there is a good place to throw a grenade at. If you add the InfluenceMapConsideration to it and ask it to search the enemies map for a value higher than 0.8. If the action scores high enough, you can read the position this consideration found from the blackboard key you entered instead of re-executing the search function.

Demos

There are a few small demos located at the demos folder of the influence maps folder. Take a look at them and also the immersive [demo for ultimate which uses influence maps](#)

Remembrance

Remembrance is a memory and emotions module for your AI agents so they can remember what happened in the world and react to it. Your NPCs can hold grudges against players and other NPCs, return a favor or even try to seduce you or bribe you. They can remember what happened by whom to whom and where and when. You can also attach any custom data to your memories.

How it works

NPCs or any other objects in your game record their memories in a container and then the container can be searched for values with different criteria. They can also set float emotion values per object or in general.

Memory

There are two components that you can use or you can use the underlying data containers. Each of the listed components have a `AddMemory()` method which records a new memory to be remembered. They all also have multiple overloads of a querying method which gives you the closest memory to a specific time with additional arguments which limits the acceptable memories. For example you can return only memories with a specific target or of a specific type or both. There are methods caching the last recorded memory with a specific action id or target for specific common use-cases as well. They usually are of the form `GetLastXXX()`

Memory component

This is the component you want most of the times.

This component is more advanced and can remove memories from the middle of the list and uses a `List<T>` as its underlying data container. When the maximum capacity is reached, the first item in the list is removed after each addition. This component has the forgetting functionality where you can specify in each memory, after how many seconds it should be forgotten and the component forgets that memory after the specified seconds.

SimpleMemory component

This component is simpler and uses a `Queue<T>` as its underlying data container and when reaching capacity, dequeues the first item in the queue. This component cannot forget memories and cannot remove any memory other than the first one in the queue but if it is enough for your use-cases then have better performance in scenarios which item removal from the list becomes a bottleneck in your game.

Since queues don't have indexing, all searches use the `foreach` construct in C# and are slower than for loops. A foreach structure calls `MoveNext()` and `Current()` methods of the queue to get to the next object

and then read it.

MemoryData

The `MemoryData` class is the type that the methods in the component accept and return.

Properties of a memory

A memory is stored using the `MemoryData` class and it contains a set of properties which can be used for querying

- **MemoryType** This is an integer value which can be used to find out what type the memory is based on your custom types in your game. for example if your `additionalData` can be of multiple types, this can be used to see what type it should be casted to.
- **StartTime** The time that this memory started. The constructor sets this automatically.
- **EndTime** The time that this memory ended. Calling `End(float time)` on the `MemoryData` object sets this automatically.
- **ForgetAfterSeconds** If greater than 0 then the memory will be forgotten after this many seconds.
- **Position** The position that this memory happened in. If you need additional data like room name/level name/bounds, you can put them in additional data or other optional fields.
- **ActionType** This field is of type `System.Type` in .NET and allows you to set the type of the class which caused this memory as an action so you know for example if the shooting was using the `Spell` class or the `FireBall` class. This could be derived from other fields in theory but having it is handy specially in the Utility AI integration.
- **Initiator** The object which initiated the memory. You can put any unity object including GameObjects and components in this as a reference. It can also be null.
- **Target** The target of this memory, for example the killed object in a murder memory. This can be null if the memory has no targets.
- **ActionId** This can be used for a unique id of an action instance. For example if you want to store exactly which magic spell of an NPC hit you, you can store its id here and put the NPC itself in the initiator field and the spell type in the ActionType field.
- **AdditionalData** is any custom data that you want to attach to this memory. usually you should find out what this can be casted to based on `ActionType`, `actionId` or `memoryType`

Emotions

There is a component called `EmotionComponent` and an Emotion Collection asset which define what emotions exist and allow you to set and get their values. Emotions can be general or per object. for example your NPC might be generally sad but happy toward a tree they really like or they can have an emotion only for objects, like affection but it might not be meaningful in your game mechanics to have a general affection emotion for the agents.

All emotions are floats between 0 and 1 and define how they are combined with a similar emotion. For example if two objects share an emotion like hate toward each other, then you can ask them to align their hates so if I hate my friend by 0.9 and he hates me by 0.6, first of all we are probably not friends and secondly, there are methods to let our hates affect each other so both are set at the higher value or are averaged based on the settings in the emotions collection.

Debuggers

Both modules have their own specific debugger windows to help you see what values exist in an agent's memory or what emotions it holds.

Queries

Both of the components and all the containers have multiple overloads of the `GetClosestMemoryToTime()` method. This method in its simplest form only takes a time and gives you the closest memory to that time.

The order of all the methods here is O(N) because to find the closest time in a list which might not be sorted by time, you have to go through all items. sorted versions will be added later on which will have a O(N/2) order but because the number of memories is not high in each container, this is not a huge concern.

Other overloads of the method limit the acceptable memories so for example the overload which takes an `ActionType` and an initiator and a target, only search in the memories which have the specified `actionType`, initiator and target and return the one which is closest to the specified time from that list.

The time is based on whatever unit you have your game for time and is provided by you yourself when creating `MemoryData` objects or calling `End()` on them. If you send 0, then the earliest memory with the specified conditions will be returned. If you specify now's time like `Time.time` then the closest memory to the current moment will be returned. however you can use any other time like between 11:00 and 13:00 in the school driver which wants to know which children did not come home with him and he did not see them.

Complex unanticipated queries

If none of the overloads provided satisfy your condition. There is an overload which takes a predicate as a lambda and can check anything you want. A predicate is a function taking an argument of type `MemoryData` and returning a bool which indicates if this object/memory is acceptable or not.

For example let's say you want to know the number of enemies killed below water to go down there and find them to loot them as a fish robot which observes the battlefield. You would write a query like

```
var underWaterKills = GetComponent<Memory>().GetClosestMemoryToTime(item => item.MemoryType == 2 && item.Position.y < seaLevel), Time.time);
```

A `MemoryType` of 2 means killing actions in our custom game code. and the code above gives you the code to get all kill actions under water, recorded in your head.

Emotions

The emotions module allows your agents to have multiple emotions. Emotions can be general or toward a specific object which can be anything from a chair to a zone or another agent.

Each emotion is a floating point value between 0 and 1 and cannot be outside this range. All emotions are named and can be combined with each other. So if you feel love toward an object by a value of 0.8 and they feel love toward you by 0.5, these can be combined by making both emotions to change to a third value. the third value can be the smaller of the two or the larger one or the average. Using more complex formulas makes it hard to understand and usually it is not needed and if you need those, nothing forces you to use the combining feature of this module. We will implement more complex combinations if they are needed in your games.

EmotionComponent

This component should be attached to any agent which wants to have emotions.

It takes two emotion collections for general emotions and per object emotions. Collections are just the name of emotions and how they are combined with a similar emotion in another agent.

This component has methods to increase/decrease emotions or set them.

You can use it like this.

```
GetComponent<EmotionComponent>().AddEmotion(angerIndex, -0.1f);
```

As you can see, we are using something called `angerIndex` to decrease anger by -0.1. The emotion index is so you don't supply string names of emotions for every get and set. This is similar to how animation parameters allow you to use hashes instead of parameter names. To get the index of each emotion you should use `GetGeneralEmotionIndex()` or `GetPerObjectEmotionIndex()`. this is because the collection of per object emotions is different from general emotions. You can assign the same collection to both though.

This is how the sample takes the index of each emotion

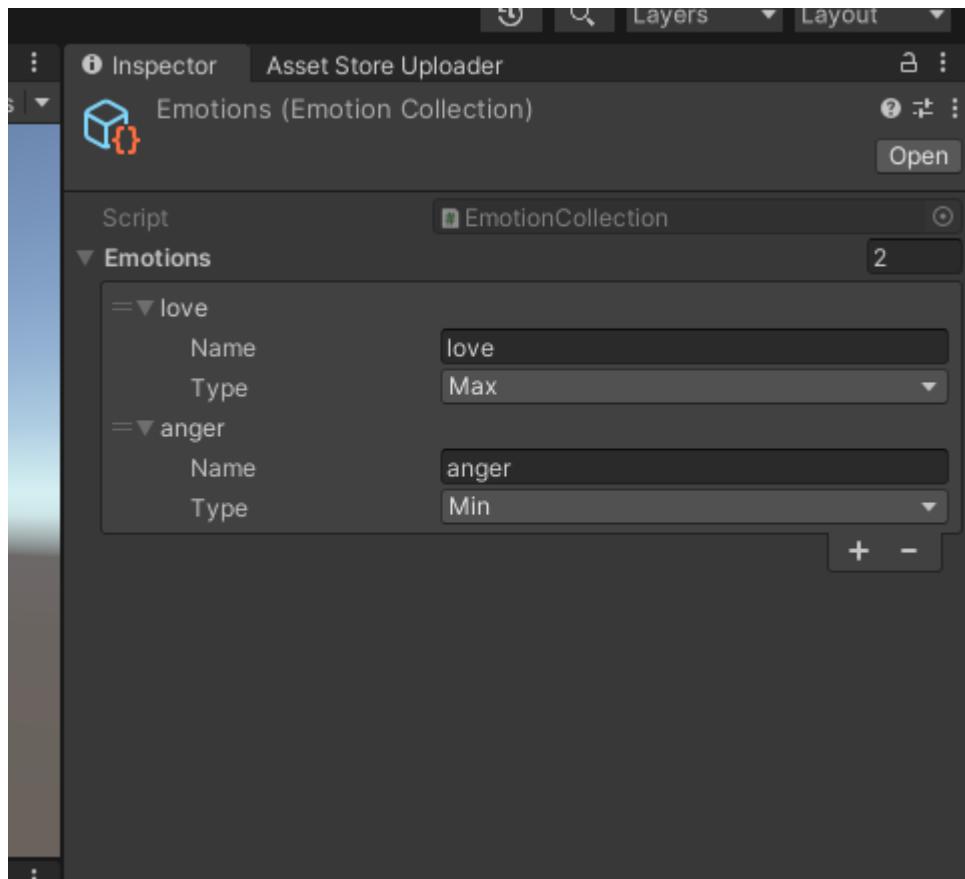
```
emotion = GetComponent<EmotionComponent>();  
loveIndex = emotion.GetGeneralEmotionIndex("love");  
angerIndex = emotion.GetGeneralEmotionIndex("anger");  
emotion.SetEmotion(loveIndex, 1);  
emotion.SetEmotion(angerIndex, 0.4f);
```

As you can see it stores the index in a variable and then uses it to set the emotion to a value. You should store these and then re-use them when adding/setting emotions. They are O(N) and the lists are small but still, the whole point is to re-use them to avoid the search.

Emotion Collections

You create these scriptable objects using right click in the project window. You should go to Create>Wise Feline>Emotion collection.

Each collection lists all relevant emotions and how they would be combined with another collection when two objects try to combine their emotions so their feelings toward another affect each other.



Emotion combination

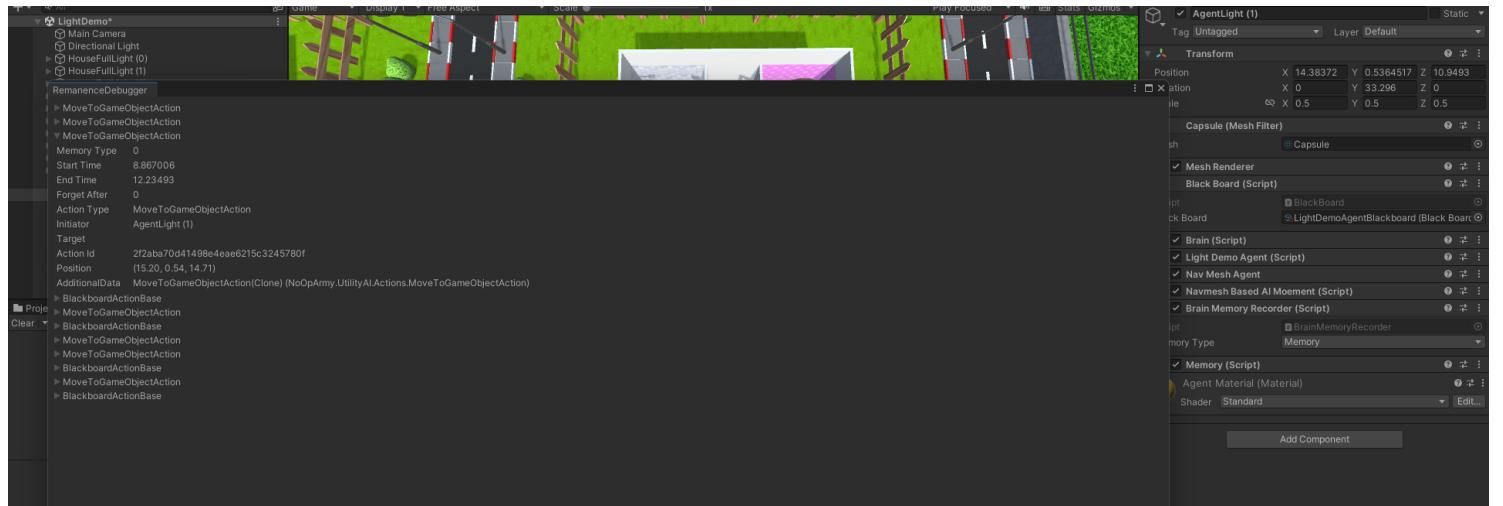
The combination is not automatic and you need to call the `Effect(GameObject other)` method of the `EmotionComponent` for it to calculate. The calculation is bidirectional and effects both objects feeling toward each other.

Remembrance Debuggers

There are two debugger windows helping you figure out what exists in the memory of an agent or what feelings it has.

Remembrance memory debugger

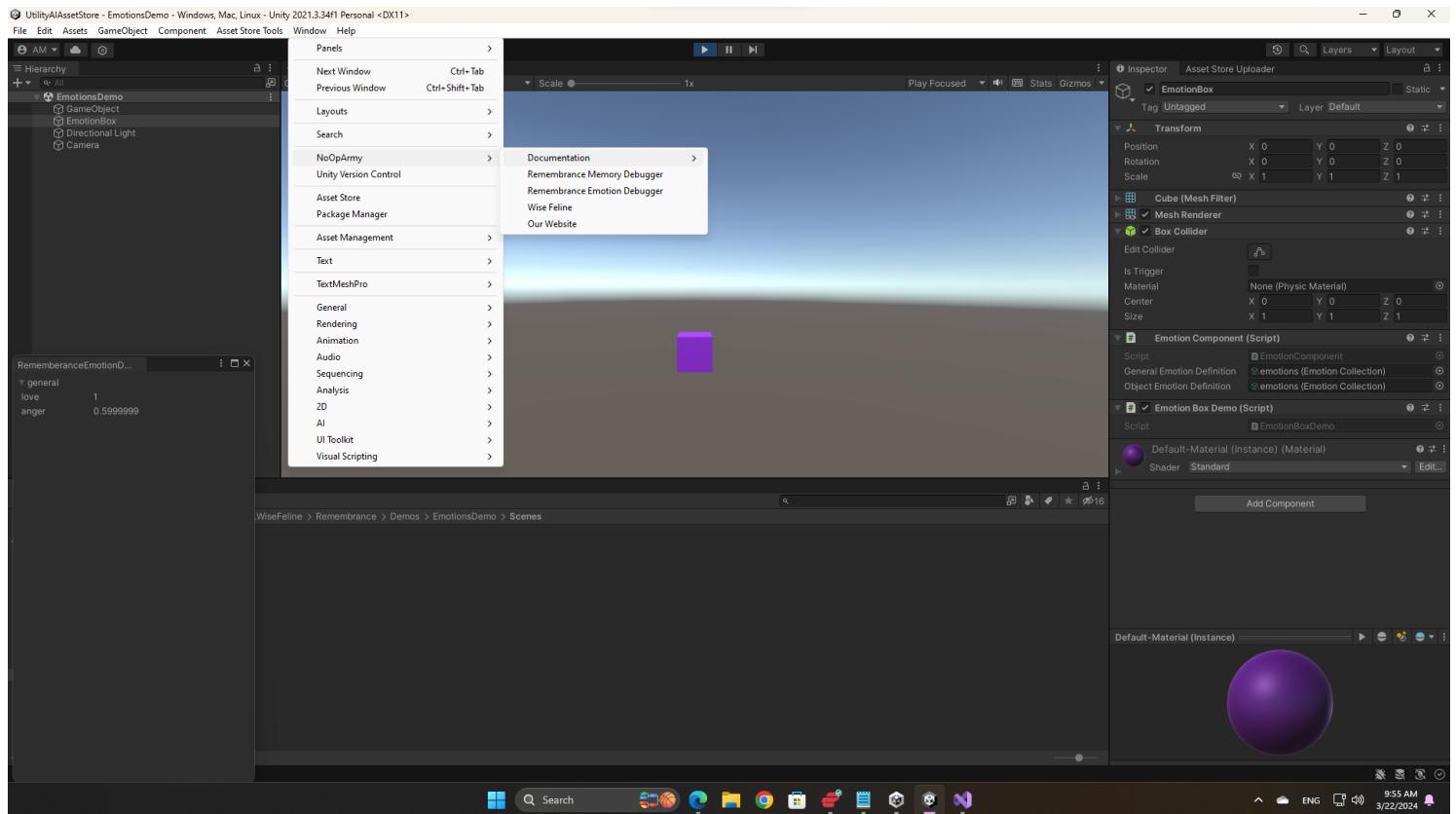
You can open this window from the Window>NoOpArmy>Remembrance Memory Debugger menu.



When a GameObject with a **Memory** or **SimpleMemory** is selected, this window shows all of its memories at runtime. You can inspect what the agent remembers, what will be forgotten and when and what data is stored in each memory.

Remembrance Emotion Debugger

There is a window at Window>NoOpArmy>Remembrance Emotion Debugger menu.



This window shows all general emotions and per object emotions of an agent.

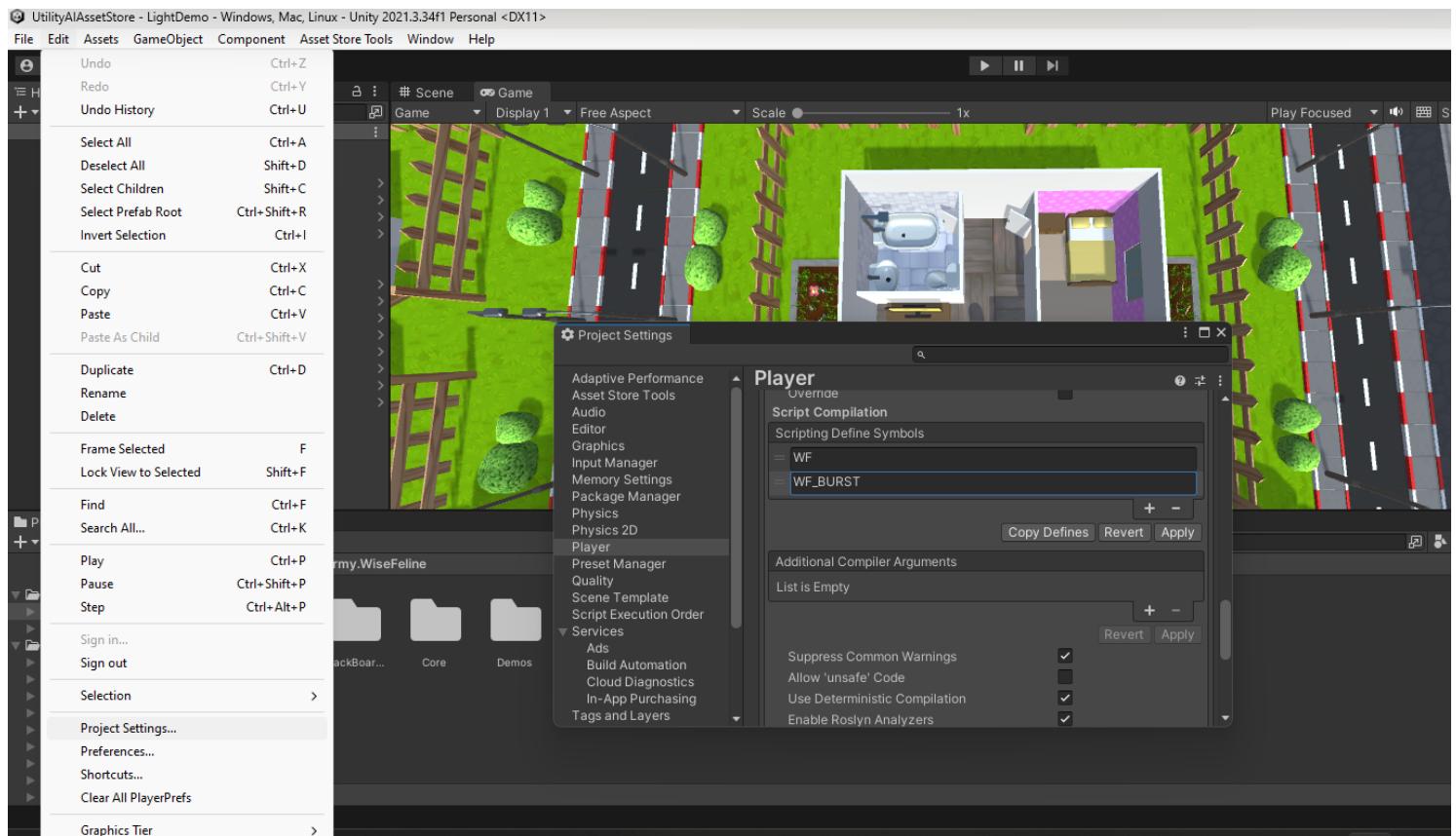
Low level containers

The components above use two container classes to do all of their insertion, removal and search operations and these can be used instead of the components directly by your classes. You might desire this if you want to use a different `MemoryData` class with different fields but you still want to use this library for searching and other functionalities.

The `Memory` component uses `MemoryContainer<T>` which uses lists and the `SimpleMemory` component uses `CircularMemoryContainer<T>` which uses queues. The components are mostly wrappers around these with a few values to be set in the inspector. They call the methods of the containers and return the result back. the `Memory` component also calls a method in its `Update()` for the forgetting feature.

Utility AI integration

For this to become available, you should add the WF define to the list of defines in your player settings. Go to the edit>Project Settings menu and then choose the player tab, in other settings you can add define symbols in the Scripting Define Symbols section.



If you attach the **BrainMemoryRecorder** component to an agent with a Utility AI brain (both Lite and Ultimate), then it records all action changes as memories. It records the current action as a new memory and ends the previous action. The type of the action is recorder in the **type** field of the memory and the target and initiator are set to the agent and its target respectively.

There is also a sample consideration available which gives a score back based on if an action happened in the last x seconds or not.

Namespace NoOpArmy

Classes

[Singleton<T>](#)

Class Singleton<T>

Namespace: [NoOpArmy](#)

Assembly: APIRef.dll

```
public class Singleton<T> : MonoBehaviour where T : MonoBehaviour
```

Type Parameters

T

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← Singleton<T>

Derived

[AITagsManager](#), [InfluenceMapCollection](#), [SmartObjectsManager](#)

Inherited Members

MonoBehaviour.IsInvoking(), MonoBehaviour.CancelInvoke(), [MonoBehaviour.Invoke\(string, float\)](#),
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#), [MonoBehaviour.CancelInvoke\(string\)](#),
[MonoBehaviour.IsInvoking\(string\)](#), [MonoBehaviour.StartCoroutine\(string\)](#),
[MonoBehaviour.StartCoroutine\(string, object\)](#), [MonoBehaviour.StartCoroutine\(IEnumerator\)](#),
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#), [MonoBehaviour.StopCoroutine\(IEnumerator\)](#),
MonoBehaviour.StopCoroutine(Coroutine), [MonoBehaviour.StopCoroutine\(string\)](#),
MonoBehaviour.StopAllCoroutines(), [MonoBehaviour.print\(object\)](#), MonoBehaviour.useGUILayout,
MonoBehaviour.runInEditMode, Behaviour.enabled, Behaviour.isActiveAndEnabled,
[Component.GetComponent\(Type\)](#), Component.GetComponent<T>(),
[Component.TryGetComponent\(Type, out Component\)](#), Component.TryGetComponent<T>(out T),
[Component.GetComponent\(string\)](#), [Component.GetComponentInChildren\(Type, bool\)](#),
[Component.GetComponentInChildren\(Type\)](#), [Component.GetComponentInChildren<T>\(bool\)](#),
Component.GetComponentInChildren<T>(), [Component.GetComponentsInChildren\(Type, bool\)](#),
[Component.GetComponentsInChildren\(Type\)](#), [Component.GetComponentsInChildren<T>\(bool\)](#),
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#),
Component.GetComponentsInChildren<T>(), [Component.GetComponentsInChildren<T>\(List<T>\)](#),
[Component.GetComponentInParent\(Type, bool\)](#), [Component.GetComponentInParent\(Type\)](#),
[Component.GetComponentInParent<T>\(bool\)](#), Component.GetComponentInParent<T>(),
[Component.GetComponentInParent\(Type, bool\)](#), [Component.GetComponentInParent\(Type\)](#),
[Component.GetComponentInParent<T>\(bool\)](#),
[Component.GetComponentInParent<T>\(bool, List<T>\)](#), Component.GetComponentInParent<T>()

[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode()
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectOfType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Properties

Instance

```
public static T Instance { get; }
```

Property Value

T

Methods

Awake()

```
protected void Awake()
```

OnDestroy()

```
protected void OnDestroy()
```

Namespace NoOpArmy.SmartObjects.Sample Classes

[DrivingBehaviour](#)

[EatFoodBehaviour](#)

This behavior give the agent some food to eat This is a simple demo so we just change the color of the agent to green

[SmartObjectDemoPlayer](#)

This script allows your demo agent to eat food and then drive somewhere. you can choose for each agent to do either both or one of them or do neither which is not useful.

Class DrivingBehaviour

Namespace: [NoOpArmy.SmartObjects.Sample](#)

Assembly: APIRef.dll

```
public class DrivingBehaviour : SmartObjectBehaviourBase
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← [SmartObjectBehaviourBase](#) ← DrivingBehaviour

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#) , [MonoBehaviour.CancelInvoke\(\)](#) , [MonoBehaviour.Invoke\(string, float\)](#) , [MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) , [MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) , [MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) , [MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(Coroutine\)](#) , [MonoBehaviour.StopCoroutine\(string\)](#) , [MonoBehaviour.StopAllCoroutines\(\)](#) , [MonoBehaviour.print\(object\)](#) , [MonoBehaviour.useGUILayout](#) , [MonoBehaviour.runInEditMode](#) , [Behaviour.enabled](#) , [Behaviour.isActiveAndEnabled](#) , [Component.GetComponent\(Type\)](#) , [Component.GetComponent<T>\(\)](#) , [Component.TryGetComponent\(Type, out Component\)](#) , [Component.TryGetComponent<T>\(out T\)](#) , [Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) , [Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) , [Component.GetComponentInChildren<T>\(\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) , [Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) , [Component.GetComponentInChildren<T>\(bool, List<T>\)](#) , [Component.GetComponentInChildren<T>\(\)](#) , [Component.GetComponentInChildren<T>\(List<T>\)](#) , [Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) , [Component.GetComponentInParent<T>\(bool\)](#) , [Component.GetComponentInParent<T>\(\)](#) , [Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) , [Component.GetComponentInParent<T>\(bool\)](#) , [Component.GetComponentInParent<T>\(bool, List<T>\)](#) , [Component.GetComponentInParent<T>\(\)](#) , [Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) , [Component.GetComponents<T>\(List<T>\)](#) , [Component.GetComponents<T>\(\)](#) , [Component.CompareTag\(string\)](#) , [Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) , [Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) , [Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,

[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

destinationXZ

The destination on the XZ plane

```
public Vector3 destinationXZ
```

Field Value

Vector3

movementDirection

The direction this will drive toward

```
[Tooltip("The direction this will drive toward")]
public Vector3 movementDirection
```

Field Value

Vector3

speed

Driving speed

```
public float speed
```

Field Value

[float](#)

Methods

AreAllslotsFilled(SmartObject)

```
public bool AreAllslotsFilled(SmartObject smartObject)
```

Parameters

`smartObject` [SmartObject](#)

Returns

`bool` ↗

OnClaimedByAgent(GameObject, SmartObject, int)

Called when a slot of the smart object is successfully claimed by an object

```
public override void OnClaimedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

`agent` GameObject

`smartObject` [SmartObject](#)

`slot` [int](#) ↗

OnFreedByAgent(GameObject, SmartObject, int)

Called when an agent frees the smart object

```
public override void OnFreedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

`agent` GameObject

`smartObject` [SmartObject](#)

`slot` [int](#) ↗

OnUsedByAgent(GameObject, SmartObject, int)

Called when a slot of the object is successfully used by an agent

```
public override void OnUsedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

agent `GameObject`

smartObject [SmartObject](#)

slot [int](#)

Class EatFoodBehaviour

Namespace: [NoOpArmy.SmartObjects.Sample](#)

Assembly: APIRef.dll

This behavior give the agent some food to eat This is a simple demo so we just change the color of the agent to green

```
public class EatFoodBehaviour : SmartObjectBehaviourBase
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← [SmartObjectBehaviourBase](#) ← EatFoodBehaviour

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,

[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Methods

OnClaimedByAgent(GameObject, SmartObject, int)

Called when the agent reserves this so nobody else touches it

```
public override void OnClaimedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

agent GameObject

smartObject [SmartObject](#)

slot [int](#)

OnFreedByAgent(GameObject, SmartObject, int)

Called when the agent using us frees us

```
public override void OnFreedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

agent GameObject

smartObject [SmartObject](#)

slot [int](#)

OnUsedByAgent(GameObject, SmartObject, int)

Called when the agent uses us

```
public override void OnUsedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

agent GameObject

smartObject [SmartObject](#)

slot [int](#)

Class SmartObjectDemoPlayer

Namespace: [NoOpArmy.SmartObjects.Sample](#)

Assembly: APIRef.dll

This script allows your demo agent to eat food and then drive somewhere. you can choose for each agent to do either both or one of them or do neither which is not useful.

```
public class SmartObjectDemoPlayer : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← SmartObjectDemoPlayer

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,

[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

shouldDrive

Should the agent find something drivable and drive somewhere

```
[Tooltip("Should the agent find something drivable and drive somewhere")]
public bool shouldDrive
```

Field Value

[bool](#) 

This script allows your demo agent to eat food and then drive somewhere. you can choose for each agent to do either both or one of them or do neither which is not useful.

shouldEat

Should the agent go eat something before driving somewhere

```
[Tooltip("Should the agent go eat something before driving somewhere")]
public bool shouldEat
```

Field Value

[bool](#) 

This script allows your demo agent to eat food and then drive somewhere. you can choose for each agent to do either both or one of them or do neither which is not useful.

Namespace NoOpArmy.UtilityAI.Actions

Classes

[BlackboardActionBase](#)

Actions which want to do some blackboard changes along side their custom code can derive from this

[BlackboardActionWithSmartObjectTarget](#)

Finds potential targets from smart objects and optionally claim and use them. It frees them inOnFinish if the claiming and using is on. Also the action will fail if it tries to claim/use the smart object unsuccessfully.

[BlackboardActionWithTagTarget](#)

This action updates its targets based on AI tags system and executes blackboard changes which you choose. The action doesn't do anything with the targets but will not execute if one of the targets don't get a high score. You need at least one consideration with NeedTarget set to true for this to evaluate the targets.

[CreateGameObjectAction](#)

This action creates gameobjects, can be used for shooting, providing loot and other similar things

[DestroySelfAction](#)

Destroys the GameObject executing the action

[DestroyTargetWithTagAction](#)

Destroys the target object, can be used for killing, eating and similar actions

[MoveAndPatrol](#)

Patrols the agent on the way points provided as children of a GameObject which is set on a blackboard key

[MoveToGameObjectAction](#)

Moves the agent to the position of a GameObject set on a blackboard key.

[MoveToSmartObject](#)

Finds a set of smart objects as potential targets and if the action gets selected to execute then it moves toward the chosen target.

[MoveToTargetWithTag](#)

Makes a list of potential targets using the AI Tag system and moves toward the chosen target if the action is selected to execute. Needs a consideration with NeedTarget set so targets can be evaluated and chosen.

[MoveToVector3Action](#)

Moves the agent to the position of a Vector3 set on a blackboard key.

NoopAction

This action is our favorite and doesn't do anything. It optionally can stop movement too.

Class BlackboardActionBase

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Actions which want to do some blackboard changes along side their custom code can derive from this

```
public class BlackboardActionBase : ActionBase
```

Inheritance

[object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← BlackboardActionBase

Derived

[BlackboardActionWithSmartObjectTarget](#), [BlackboardActionWithTagTarget](#), [CreateGameObjectAction](#),
[DestroySelfAction](#), [DestroyTargetWithTagAction](#), [MoveToGameObjectAction](#), [MoveToSmartObject](#),
[MoveToTargetWithTag](#), [MoveToVector3Action](#), [NoopAction](#), [MoveAndWaitAndChangeFloat](#)

Inherited Members

[ActionBase.priority](#), [ActionBase.isInterruptable](#), [ActionBase.maxTargetCount](#),
[ActionBase.useMomentumOnTarget](#), [ActionBase.Considerations](#), [ActionBase.Brain](#),
[ActionBase.IsInitialized](#), [ActionBase.Score](#), [ActionBase.Weight](#), [ActionBase.ChosenTarget](#),
[ActionBase.AddTarget\(Component\)](#), [ActionBase.AddTargets<T>\(T\[\]\)](#),
[ActionBase.AddTargets<T>\(List<T>\)](#), [ActionBase.ClearTargets\(\)](#), [ActionBase.RemoveTarget\(Component\)](#),
[ActionBase.OnInitialized\(\)](#), [ActionBase.OnLateUpdate\(\)](#), [ActionBase.OnFixedUpdate\(\)](#),
[ActionBase.ActionSucceed\(\)](#), [ActionBase.ActionSucceeded\(\)](#), [ActionBase.ActionFailed\(\)](#), [AIObject.guid](#),
[AIObject.Name](#), [ScriptableObject.SetDirty\(\)](#), [ScriptableObject.CreateInstance\(string\)](#) ↗,
[ScriptableObject.CreateInstance\(Type\)](#) ↗, [ScriptableObject.CreateInstance<T>\(\)](#), [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#), [Object.Equals\(object\)](#) ↗, [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↗ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↗ ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ↗ ,

[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

ResetChangerOnStart

Resets the changer whenever this action is selected. Mostly useful for changers which execute a specific number of times

```
[Tooltip("Resets the changer whenever this action is selected. Mostly useful for changers  
which execute a specific number of times")]  
public bool ResetChangerOnStart
```

Field Value

[bool](#)

Actions which want to do some blackboard changes along side their custom code can derive from this

blackBoard

The blackboard to use for changes

```
protected BlackBoard blackBoard
```

Field Value

BlackBoard

Actions which want to do some blackboard changes along side their custom code can derive from this
changer

The change you want to happen to the blackboard during this action

```
[Tooltip("The change you want to happen to the blackboard during this action")]
public BlackBoardChangeer changer
```

Field Value

BlackBoardChangeer

Actions which want to do some blackboard changes along side their custom code can derive from this

changerStartingDelayInSeconds

The start delay before changes start taking effect.

```
[Tooltip("The start delay before changes in seconds")]
public float changerStartingDelayInSeconds
```

Field Value

float ↗

Actions which want to do some blackboard changes along side their custom code can derive from this

forceUpdateTargetsOnFinish

If true, upon completion of this action, it will prompt the brain to immediately update the target for all actions.

```
[Tooltip("If true, upon completion of this action, it will prompt the brain to immediately
update the target for all actions.")]
public bool forceUpdateTargetsOnFinish
```

Field Value

bool ↗

Actions which want to do some blackboard changes along side their custom code can derive from this

Methods

OnFinish()

Called when the action is finished, either by failing or succeeding in achieving a desired behaviour

```
protected override void OnFinish()
```

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

OnUpdate()

Should be used like MonoBehaviour's Update for the action

```
protected override void OnUpdate()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class BlackboardActionWithSmartObjectTarget

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Finds potential targets from smart objects and optionally claim and use them. It frees them inOnFinish if the claiming and using is on. Also the action will fail if it tries to claim/use the smart object unsuccessfully.

```
public class BlackboardActionWithSmartObjectTarget : BlackboardActionBase
```

Inheritance

[object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← BlackboardActionWithSmartObjectTarget

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnUpdate\(\)](#) ,
[ActionBase.priority](#) , [ActionBase.isInterruptable](#) , [ActionBase.maxTargetCount](#) ,
[ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) , [ActionBase.Brain](#) ,
[ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) , [ActionBase.ChosenTarget](#) ,
[ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnInitialized\(\)](#) , [ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) ,
[ActionBase.ActionSucceed\(\)](#) , [ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) ,
[AIObject.Name](#) , [ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ↗ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↗ , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) ↗ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↗ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↗ ,

Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsOfType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsOfType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

ClaimAndUseSmartObject

Should the action claim and use the smart object in OnStart and then Free the object in OnFinish?

```
[Tooltip("Should the action claim and use the smart object in OnStart and then Free the  
object in OnFinish?")]  
public bool ClaimAndUseSmartObject
```

Field Value

[bool](#)

Finds potential targets from smart objects and optionally claim and use them. It frees them inOnFinish if the claiming and using is on. Also the action will fail if it tries to claim/use the smart object unsuccessfully.

SlotStatus

Finds smart objects with the chosen slot status.

```
[Tooltip("Finds smart objects with the chosen slot status.")]
```

```
public SmartObjectsManager.SearchQueryOptions SlotStatus
```

Field Value

[SmartObjectsManager.SearchQueryOptions](#)

Finds potential targets from smart objects and optionally claim and use them. It frees them inOnFinish if the claiming and using is on. Also the action will fail if it tries to claim/use the smart object unsuccessfully.

excludedTags

The tags which if one of them is on the smart object, it is excluded from the search

```
[Tooltip("The tags which if one of them is on the smart object, it is excluded from  
the search")]  
public string[] excludedTags
```

Field Value

[string\[\]](#)

Finds potential targets from smart objects and optionally claim and use them. It frees them inOnFinish if the claiming and using is on. Also the action will fail if it tries to claim/use the smart object unsuccessfully.

includedTags

The tags which at least one of them should be on the smart object so it is included in the search results

```
[Tooltip("The tags which at least one of them should be on the smart object so it is  
included in the search results")]  
public string[] includedTags
```

Field Value

[string\[\]](#)

Finds potential targets from smart objects and optionally claim and use them. It frees them inOnFinish if the claiming and using is on. Also the action will fail if it tries to claim/use the smart object unsuccessfully.

searchRadius

Radius of the search for smart objects

```
public float searchRadius
```

Field Value

[float](#)

Finds potential targets from smart objects and optionally claim and use them. It frees them inOnFinish if the claiming and using is on. Also the action will fail if it tries to claim/use the smart object unsuccessfully.

Methods

OnFinish()

Called when the action is finished, either by failing or succeeding in achieving a desired behaviour

```
protected override void OnFinish()
```

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class BlackboardActionWithTagTarget

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

This action updates its targets based on AI tags system and executes blackboard changes which you choose. The action doesn't do anything with the targets but will not execute if one of the targets don't get a high score. You need at least one consideration with NeedTarget set to true for this to evaluate the targets.

```
public class BlackboardActionWithTagTarget : BlackboardActionBase
```

Inheritance

[Object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← BlackboardActionWithTagTarget

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnStart\(\)](#) ,
[BlackboardActionBase.OnUpdate\(\)](#) , [BlackboardActionBase.OnFinish\(\)](#) , [ActionBase._priority](#) ,
[ActionBase.isInterruptable](#) , [ActionBase.maxTargetCount](#) , [ActionBase.useMomentumOnTarget](#) ,
[ActionBase.Considerations](#) , [ActionBase.Brain](#) , [ActionBase.IsInitialized](#) , [ActionBase.Score](#) ,
[ActionBase.Weight](#) , [ActionBase.ChosenTarget](#) , [ActionBase.AddTarget\(Component\)](#) ,
[ActionBase.AddTargets<T>\(T\[\]\)](#) , [ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) ,
[ActionBase.RemoveTarget\(Component\)](#) , [ActionBase.OnInitialized\(\)](#) , [ActionBase.OnLateUpdate\(\)](#) ,
[ActionBase.OnFixedUpdate\(\)](#) , [ActionBase.ActionSucceed\(\)](#) , [ActionBase.ActionSucceeded\(\)](#) ,
[ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) , [AIObject.Name](#) , [ScriptableObject.SetDirty\(\)](#) ,
[ScriptableObject.CreateInstance\(string\)](#) ↗ , [ScriptableObject.CreateInstance\(Type\)](#) ↗ ,
[ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceID\(\)](#) , [Object.GetHashCode\(\)](#) ,
[Object.Equals\(object\)](#) ↗ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,

[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsOfType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsOfType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectsInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectOfType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

excludedTags

Tags which if any of them are on the AITags component of the object, the object cannot be considered a potential target

```
[Tooltip("Tags which if any of them are on the AITags component of the object, the object  
cannot be considered a potential target")]  
public string[] excludedTags
```

Field Value

[string](#)[]

This action updates its targets based on AI tags system and executes blackboard changes which you choose. The action doesn't do anything with the targets but will not execute if one of the targets don't get a high score. You need at least one consideration with NeedTarget set to true for this to evaluate the targets.

includedTags

Tags which at least one of them should be in the AITags component of the object to be considered a valid target.

```
[Tooltip("Tags which at least one of them should be in the AITags component of the object to  
be considered a valid target.")]  
public string[] includedTags
```

Field Value

[string](#) []

This action updates its targets based on AI tags system and executes blackboard changes which you choose. The action doesn't do anything with the targets but will not execute if one of the targets don't get a high score. You need at least one consideration with NeedTarget set to true for this to evaluate the targets.

radius

The radius of the search for tags

```
[Header("Search properties")]  
[Tooltip("The radius of the search for tags")]  
public float radius
```

Field Value

[float](#) []

This action updates its targets based on AI tags system and executes blackboard changes which you choose. The action doesn't do anything with the targets but will not execute if one of the targets don't get a high score. You need at least one consideration with NeedTarget set to true for this to evaluate the targets.

Methods

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class CreateGameObjectAction

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

This action creates gameobjects, can be used for shooting, providing loot and other similar things

```
public class CreateGameObjectAction : BlackboardActionBase
```

Inheritance

[object](#) ↴ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← CreateGameObjectAction

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnFinish\(\)](#) ,
[ActionBase._priority](#) , [ActionBase.isInterruptable](#) , [ActionBase.maxTargetCount](#) ,
[ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) , [ActionBase.Brain](#) ,
[ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) , [ActionBase.ChosenTarget](#) ,
[ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnInitialized\(\)](#) , [ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) ,
[ActionBase.ActionSucceed\(\)](#) , [ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) ,
[AIObject.Name](#) , [ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ↴ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↴ , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) ↴ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↴ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↴ , [Object.Destroy\(Object, float\)](#) ↴ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↴ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↴ , [Object.FindObjectsOfType\(Type, bool\)](#) ↴ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↴ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↴ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↴ ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ↴ ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) ↴ , [Object.FindObjectsOfType<T>\(\)](#) ,

Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

createAtTheSameTime

Should we create all objects at the same time or once after each delay

```
[Tooltip("Should we create all objects at the same time or once after each delay")]
public bool createAtTheSameTime
```

Field Value

[bool](#)

This action creates gameobjects, can be used for shooting, providing loot and other similar things

creationDelay

Delay to create an object

```
[Tooltip("Delay to create an object")]
public float creationDelay
```

Field Value

[float](#)

This action creates gameobjects, can be used for shooting, providing loot and other similar things

creationPosition

The position offset to create the object

```
[Tooltip("The position offset to create the object")]
public Vector3 creationPosition
```

Field Value

Vector3

This action creates gameobjects, can be used for shooting, providing loot and other similar things

isPositionRelative

Is the position specified from origin or the position of the executing game object.

```
[Tooltip("Is the position specified from origin or the position of the executing
game object.")]
public bool isPositionRelative
```

Field Value

bool 

This action creates gameobjects, can be used for shooting, providing loot and other similar things

objectCountToCreate

How many objects to create

```
[Tooltip("How many objects to create")]
public int objectCountToCreate
```

Field Value

[int](#)

This action creates gameobjects, can be used for shooting, providing loot and other similar things

objectToCreate

The GameObject to create

```
[Tooltip("The GameObject to create")]
public GameObject objectToCreate
```

Field Value

GameObject

This action creates gameobjects, can be used for shooting, providing loot and other similar things

shouldAdaptBrainRotation

Should the object adapt the brain's object rotation

```
[Tooltip("Should the object adapt the brain's object rotation")]
public bool shouldAdaptBrainRotation
```

Field Value

[bool](#)

This action creates gameobjects, can be used for shooting, providing loot and other similar things

Methods

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

OnUpdate()

Should be used like MonoBehaviour's Update for the action

```
protected override void OnUpdate()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class DestroySelfAction

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Destroys the GameObject executing the action

```
public class DestroySelfAction : BlackboardActionBase
```

Inheritance

[object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← DestroySelfAction

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnFinish\(\)](#) ,
[ActionBase._priority](#) , [ActionBase.isInterruptable](#) , [ActionBase.maxTargetCount](#) ,
[ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) , [ActionBase.Brain](#) ,
[ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) , [ActionBase.ChosenTarget](#) ,
[ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnInitialized\(\)](#) , [ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) ,
[ActionBase.ActionSucceed\(\)](#) , [ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) ,
[AIObject.Name](#) , [ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ↗ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↗ , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) ↗ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↗ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↗ ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ↗ ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) ↗ , [Object.FindObjectsOfType<T>\(\)](#) ,

Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

delayForDestruction

How much time should happen before destruction happens

```
[Tooltip("How much time should happen before destruction happens")]
public float delayForDestruction
```

Field Value

[float](#)

Destroys the GameObject executing the action

Methods

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

OnUpdate()

Should be used like MonoBehaviour's Update for the action

```
protected override void OnUpdate()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class DestroyTargetWithTagAction

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Destroys the target object, can be used for killing, eating and similar actions

```
public class DestroyTargetWithTagAction : BlackboardActionBase
```

Inheritance

[object](#) ↴ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← DestroyTargetWithTagAction

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnFinish\(\)](#) ,
[ActionBase._priority](#) , [ActionBase.isInterruptable](#) , [ActionBase.maxTargetCount](#) ,
[ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) , [ActionBase.Brain](#) ,
[ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) , [ActionBase.ChosenTarget](#) ,
[ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnInitialized\(\)](#) , [ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) ,
[ActionBase.ActionSucceed\(\)](#) , [ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) ,
[AIObject.Name](#) , [ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ↴ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↴ , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) ↴ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↴ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↴ , [Object.Destroy\(Object, float\)](#) ↴ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↴ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↴ , [Object.FindObjectsOfType\(Type, bool\)](#) ↴ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↴ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↴ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↴ ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ↴ ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) ↴ , [Object.FindObjectsOfType<T>\(\)](#) ,

Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

delayForDestruction

How much time should pass before destruction happens

```
[Tooltip("How much time should pass before destruction happens")]
public float delayForDestruction
```

Field Value

[float](#)

Destroys the target object, can be used for killing, eating and similar actions

excludedTags

```
public string[] excludedTags
```

Field Value

[string](#)[]

Destroys the target object, can be used for killing, eating and similar actions

includedTags

```
public string[] includedTags
```

Field Value

[string](#)[]

Destroys the target object, can be used for killing, eating and similar actions

radius

```
public float radius
```

Field Value

[float](#)

Destroys the target object, can be used for killing, eating and similar actions

Methods

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

OnUpdate()

Should be used like MonoBehaviour's Update for the action

```
protected override void OnUpdate()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class MoveAndPatrol

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Patrols the agent on the way points provided as children of a GameObject which is set on a blackboard key

```
public class MoveAndPatrol : ActionBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← MoveAndPatrol

Inherited Members

[ActionBase.priority](#) , [ActionBase.isInterruptable](#) , [ActionBase.maxTargetCount](#) ,
[ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) , [ActionBase.Brain](#) ,
[ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) , [ActionBase.ChosenTarget](#) ,
[ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnUpdate\(\)](#) , [ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) ,
[ActionBase.OnFinish\(\)](#) , [ActionBase.ActionSucceed\(\)](#) , [ActionBase.ActionSucceeded\(\)](#) ,
[ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) , [AIObject.Name](#) , [ScriptableObject.SetDirty\(\)](#) ,
[ScriptableObject.CreateInstance\(string\)](#) , [ScriptableObject.CreateInstance\(Type\)](#) ,
[ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) , [Object.GetHashCode\(\)](#) ,
[Object.Equals\(object\)](#) , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , [Object.FindObjectsOfType<T>\(\)](#) ,
[Object.FindObjectsByType<T>\(FindObjectsSortMode\)](#) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
[Object.FindObjectsByType<T>\(FindObjectsInactive, FindObjectsSortMode\)](#) ,

Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

FinishAfterOneCycle

Should the action finishes after going through the points once?

```
[Tooltip("Should the action finishes after going through the points once")]
public bool FinishAfterOneCycle
```

Field Value

[bool](#)

Patrols the agent on the way points provided as children of a GameObject which is set on a blackboard key

gameObjectKeyName

Name of the key in the blackboard of the GameObject which contains all patrol points as children

```
[Tooltip("Name of the key in the blackboard of the GameObject which contains all patrol
points as children")]
public string gameObjectKeyName
```

Field Value

[string](#)

Patrols the agent on the way points provided as children of a GameObject which is set on a blackboard key

Methods

OnInitialized()

Called when the action is initialized

```
protected override void OnInitialized()
```

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class MoveToGameObjectAction

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Moves the agent to the position of a GameObject set on a blackboard key.

```
public class MoveToGameObjectAction : BlackboardActionBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← MoveToGameObjectAction

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnUpdate\(\)](#) ,
[BlackboardActionBase.OnFinish\(\)](#) , [ActionBase._priority](#) , [ActionBase.isInterruptable](#) ,
[ActionBase.maxTargetCount](#) , [ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) ,
[ActionBase.Brain](#) , [ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) ,
[ActionBase.ChosenTarget](#) , [ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) , [ActionBase.ActionSucceed\(\)](#) ,
[ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceId() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>()

Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

gameObjectKeyName

Name of the GameObject key in the blackboard

```
[Tooltip("Name of the GameObject key in the blackboard")]
public string gameObjectKeyName
```

Field Value

[string](#)

Moves the agent to the position of a GameObject set on a blackboard key.

Methods

OnInitialized()

Called when the action is initialized

```
protected override void OnInitialized()
```

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class MoveToSmartObject

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Finds a set of smart objects as potential targets and if the action gets selected to execute then it moves toward the chosen target.

```
public class MoveToSmartObject : BlackboardActionBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← MoveToSmartObject

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnUpdate\(\)](#) ,
[BlackboardActionBase.OnFinish\(\)](#) , [ActionBase._priority](#) , [ActionBase.isInterruptable](#) ,
[ActionBase.maxTargetCount](#) , [ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) ,
[ActionBase.Brain](#) , [ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) ,
[ActionBase.ChosenTarget](#) , [ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) , [ActionBase.ActionSucceed\(\)](#) ,
[ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
[ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ,

[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

SlotStatus

Criteria that the slots should meet to be considered a potential target

```
public SmartObjectsManager.SearchQueryOptions SlotStatus
```

Field Value

[SmartObjectsManager.SearchQueryOptions](#)

Finds a set of smart objects as potential targets and if the action gets selected to execute then it moves toward the chosen target.

excludedTags

The tag which none of them should be on the smart object so the object is considered a target.

```
public string[] excludedTags
```

Field Value

[string\[\]](#)

Finds a set of smart objects as potential targets and if the action gets selected to execute then it moves toward the chosen target.

includedTags

The tags which at least one of them should be on the smart object to be valid

```
public string[] includedTags
```

Field Value

[string](#)[]

Finds a set of smart objects as potential targets and if the action gets selected to execute then it moves toward the chosen target.

searchRadius

Radius of the search for SmartObjects

```
[Tooltip("Radius of the search for SmartObjects")]
public float searchRadius
```

Field Value

[float](#)[]

Finds a set of smart objects as potential targets and if the action gets selected to execute then it moves toward the chosen target.

Methods

OnInitialized()

Called when the action is initialized

```
protected override void OnInitialized()
```

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class MoveToTargetWithTag

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Makes a list of potential targets using the AI Tag system and moves toward the chosen target if the action is selected to execute. Needs a consideration with NeedTarget set so targets can be evaluated and chosen.

```
public class MoveToTargetWithTag : BlackboardActionBase
```

Inheritance

[object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← MoveToTargetWithTag

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnUpdate\(\)](#) ,
[BlackboardActionBase.OnFinish\(\)](#) , [ActionBase.priority](#) , [ActionBase.isInterruptable](#) ,
[ActionBase.maxTargetCount](#) , [ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) ,
[ActionBase.Brain](#) , [ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) ,
[ActionBase.ChosenTarget](#) , [ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) , [ActionBase.ActionSucceed\(\)](#) ,
[ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
[ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ↗ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↗ , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) ↗ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↗ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↗ ,

Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

excludedTags

Tags which none of them should be present on the AI Tags component of the object to be a potential target.

```
public string[] excludedTags
```

Field Value

[string](#)[]

Makes a list of potential targets using the AI Tag system and moves toward the chosen target if the action is selected to execute. Needs a consideration with NeedTarget set so targets can be evaluated and chosen.

includedTags

Tags which at least one of them should be present on the AI Tags component of the object to be a potential target.

```
public string[] includedTags
```

Field Value

[string](#)[]

Makes a list of potential targets using the AI Tag system and moves toward the chosen target if the action is selected to execute. Needs a consideration with NeedTarget set so targets can be evaluated and chosen.

radius

The radius of the search

```
[Tooltip("The radius of the search")]
public float radius
```

Field Value

[float](#)

Makes a list of potential targets using the AI Tag system and moves toward the chosen target if the action is selected to execute. Needs a consideration with NeedTarget set so targets can be evaluated and chosen.

Methods

OnInitialized()

Called when the action is initialized

```
protected override void OnInitialized()
```

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class MoveToVector3Action

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

Moves the agent to the position of a Vector3 set on a blackboard key.

```
public class MoveToVector3Action : BlackboardActionBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← MoveToVector3Action

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnUpdate\(\)](#) ,
[BlackboardActionBase.OnFinish\(\)](#) , [ActionBase._priority](#) , [ActionBase.isInterruptable](#) ,
[ActionBase.maxTargetCount](#) , [ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) ,
[ActionBase.Brain](#) , [ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) ,
[ActionBase.ChosenTarget](#) , [ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) , [ActionBase.ActionSucceed\(\)](#) ,
[ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceId() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>()

Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

positionKeyName

Name of the Vector3 key in the blackboard

```
[Tooltip("Name of the Vector3 key in the blackboard")]
public string positionKeyName
```

Field Value

[string](#)

Moves the agent to the position of a Vector3 set on a blackboard key.

Methods

OnInitialized()

Called when the action is initialized

```
protected override void OnInitialized()
```

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Class NoopAction

Namespace: [NoOpArmy.UtilityAI.Actions](#)

Assembly: APIRef.dll

This action is our favorite and doesn't do anything. It optionally can stop movement too.

```
public class NoopAction : BlackboardActionBase
```

Inheritance

[object](#) ↴ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← NoopAction

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnUpdate\(\)](#) ,
[BlackboardActionBase.OnFinish\(\)](#) , [ActionBase._priority](#) , [ActionBase.isInterruptable](#) ,
[ActionBase.maxTargetCount](#) , [ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) ,
[ActionBase.Brain](#) , [ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) ,
[ActionBase.ChosenTarget](#) , [ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnInitialized\(\)](#) , [ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) ,
[ActionBase.ActionSucceed\(\)](#) , [ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) ,
[AIObject.Name](#) , [ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ↴ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↴ , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceID\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) ↴ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↴ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↴ , [Object.Destroy\(Object, float\)](#) ↴ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↴ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↴ , [Object.FindObjectsOfType\(Type, bool\)](#) ↴ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↴ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↴ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↴ ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ↴ ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) ↴ , [Object.FindObjectsOfType<T>\(\)](#) ,
[Object.FindObjectsByType<T>\(FindObjectsSortMode\)](#) , [Object.FindObjectsOfType<T>\(bool\)](#) ↴ ,

Object.FindObjectsOfType<T>(FindObjectsInactive, FindObjectsSortMode),
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Methods

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected override void OnStart()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Namespace NoOpArmy.UtilityAI.Considerations

Classes

[BlackboardBoolConsideration](#)

Checks a boolean key in a blackboard. This returns 1 if the key exists and is true and if it doesn't exist or is false returns 0.

[BlackboardConsiderationKeyExists](#)

This consideration returns 1 if the specified key name exists and otherwise returns 0

[BlackboardFloatConsideration](#)

Checks a float key in the blackboard and if it exists, returns its value, otherwise returns 0

[BlackboardIntRangeConsideration](#)

Checks an integer key in the blackboard and returns its value. If the key doesn't exist then it returns 0

[BlackboardIntValueConsideration](#)

This consideration checks an int key in the blackboard and returns 1 if the value equals the expected value; otherwise, it returns 0.

[InfluenceMapConsideration](#)

This consideration searches the influence map specified for a value and returns 1 if it finds something. Otherwise it returns 0

[SmartObjectDistanceConsideration](#)

Returns the distance to a target which is a smart object. You can choose to get the distance to a specific slot with a specific condition like the distance to the closest slot which is free.

[TargetDistanceConsideration](#)

Returns the distance to a target. This normalizes to a value between 0 and 1 based on minRange and maxRange values

Enums

[BlackboardConsiderationKeyExists.KeyType](#)

The blackboard key type

Class BlackboardBoolConsideration

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

Checks a boolean key in a blackboard. This reutnrs 1 if the key exists and is true and if it doesn't exist or is false returns 0.

```
public class BlackboardBoolConsideration : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ← BlackboardBoolConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,

[Object.FindAnyObjectByType\(Type\)](#) , [ObjectFindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

keyName

Name of the boolean key to check

```
[Tooltip("Name of the boolean key to check")]
public string keyName
```

Field Value

[string](#)

Checks a boolean key in a blackboard. This reutnrs 1 if the key exists and is true and if it doesn't exist or is false returns 0.

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action

stores in its target list in the `UpdateTargets` call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#) ↗

A value between `minRange` and `maxRange` which indicates the current value/score of the consideration

Class BlackboardConsiderationKeyExists

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

This consideration returns 1 if the specified key name exists and otherwise returns 0

```
public class BlackboardConsiderationKeyExists : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ← BlackboardConsiderationKeyExists

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,

[Object.FindAnyObjectByType\(Type\)](#) , [ObjectFindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

keyName

Name of the key in the blackboard

```
[Tooltip("Name of the key in the blackboard")]
public string keyName
```

Field Value

[string](#)

This consideration returns 1 if the specified key name exists and otherwise returns 0

keyType

Type of the key to check for existence

```
[Tooltip("Type of the key to check for existence")]
public BlackboardConsiderationKeyExists.KeyType keyType
```

Field Value

[BlackboardConsiderationKeyExists.KeyType](#)

This consideration returns 1 if the specified key name exists and otherwise returns 0

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

Enum BlackboardConsiderationKeyExists.KeyType

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

The blackboard key type

```
public enum BlackboardConsiderationKeyExists.KeyType
```

Fields

Bool = 2

Float = 0

GameObject = 3

Int = 1

Object = 5

UnityObject = 4

Vector3 = 6

Class BlackboardFloatConsideration

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

Checks afloat key in the blackboard and if it exists, returns its value, otherwise returns 0

```
public class BlackboardFloatConsideration : ConsiderationBase
```

Inheritance

[object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ← BlackboardFloatConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ↗ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↗ , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) ↗ , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↗ ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ↗ ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ↗ ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) ↗ , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ↗ ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ↗ ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectsInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ↗ ,
[Object.FindObjectType\(Type\)](#) ↗ , [Object.FindFirstObjectOfType\(Type\)](#) ↗ ,
[Object.FindAnyObjectOfType\(Type\)](#) ↗ , [Object.FindObjectType\(Type, bool\)](#) ↗ ,

[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

keyName

Name of the key in the blackboard

```
[Tooltip("Name of the key in the blackboard")]
public string keyName
```

Field Value

[string](#)

Checks afloat key in the blackboard and if it exists, returns its value, otherwise returns 0

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

Class BlackboardIntRangeConsideration

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

Checks an integer key in the blackboard and returns its value. If the key doesn't exist then it returns 0

```
public class BlackboardIntRangeConsideration : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ← BlackboardIntRangeConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,

[Object.FindAnyObjectByType\(Type\)](#) , [ObjectFindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

keyName

Name of the key in the blackboard

```
[Tooltip("Name of the key in the blackboard")]
public string keyName
```

Field Value

[string](#)

Checks an integer key in the blackboard and returns its value. If the key doesn't exist then it returns 0

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

Class BlackboardIntValueConsideration

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

This consideration checks an int key in the blackboard and returns 1 if the value equals the expected value; otherwise, it returns 0.

```
public class BlackboardIntValueConsideration : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ← BlackboardIntValueConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,

[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectOfType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

expectedValue

The consideration returns 1 if the value equals the expected value; otherwise, it returns 0.

```
[Tooltip("The consideration returns 1 if the value equals the expected value; otherwise, it  
returns 0.")]  
public int expectedValue
```

Field Value

[int](#)

This consideration checks an int key in the blackboard and returns 1 if the value equals the expected value; otherwise, it returns 0.

keyName

Name of the key in the blackboard

```
[Tooltip("Name of the key in the blackboard")]  
public string keyName
```

Field Value

[string](#)

This consideration checks an int key in the blackboard and returns 1 if the value equals the expected value; otherwise, it returns 0.

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

Class InfluenceMapConsideration

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

This consideration searches the influence map specified for a value and returns 1 if it finds something. Otherwise it returns 0

```
public class InfluenceMapConsideration : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIOBJECT](#) ← [ConsiderationBase](#) ← InfluenceMapConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.UpdateRange\(float, float\)](#) ,
[AIOBJECT.guid](#) , [AIOBJECT.Name](#) , [ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , [Object.FindObjectsOfType<T>\(\)](#) ,
[Object.FindObjectsByType<T>\(FindObjectsSortMode\)](#) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
[Object.FindObjectsByType<T>\(FindObjectsInactive, FindObjectsSortMode\)](#) ,
[ObjectFindObjectOfType<T>\(\)](#) , [Object.FindObjectType<T>\(bool\)](#) ,
[Object.FindFirstObjectByType<T>\(\)](#) , [Object.FindAnyObjectByType<T>\(\)](#) ,
[Object.FindFirstObjectByType<T>\(FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType<T>\(FindObjectsInactive\)](#) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectType\(Type, bool\)](#) ,

```
Object.FindFirstObjectByType\(Type, FindObjectsInactive\) ,
Object.FindAnyObjectByType\(Type, FindObjectsInactive\) , Object.ToString() , Object.name ,
Object.hideFlags , object.Equals\(object, object\) , object.GetType\(\) , object.MemberwiseClone\(\) ,
object.ReferenceEquals\(object, object\)
```

Fields

mapName

```
[Tooltip("The name of the map to consider.")]
public string mapName
```

Field Value

[string](#)

This consideration searches the influence map specified for a value and returns 1 if it finds something. Otherwise it returns 0

queryResultKeyName

Stores the result of the influence map query to be used by the actions in a Vector3 blackboard key. The results will be in x and z components in influence map cells coordinates. Leave this empty if you don't need it.

```
[Tooltip("Stores the result of the influence map query to be used by the actions in a
Vector3 blackboard key. The results will be in x and z components in influence map cells
coordinates. Leave this empty if you don't need it.")]
public string queryResultKeyName
```

Field Value

[string](#)

This consideration searches the influence map specified for a value and returns 1 if it finds something. Otherwise it returns 0

radius

The radius to search in map cells

```
[Tooltip("The radius to search in map cells")]
public int radius
```

Field Value

[int](#)

This consideration searches the influence map specified for a value and returns 1 if it finds something. Otherwise it returns 0

searchCondition

The search condition

```
public SearchCondition searchCondition
```

Field Value

[SearchCondition](#)

This consideration searches the influence map specified for a value and returns 1 if it finds something. Otherwise it returns 0

searchValue

The value to search for. For example if this is 0.6 and condition is greater, then the search functionality will look for a value bigger than 0.6 in the radius around the target

```
public float searchValue
```

Field Value

[float](#)

This consideration searches the influence map specified for a value and returns 1 if it finds something. Otherwise it returns 0

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

OnInitialized()

Fired when the consideration is initialized

```
protected override void OnInitialized()
```

Class SmartObjectDistanceConsideration

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

Returns the distance to a target which is a smart object. You can choose to get the distance to a specific slot with a specific condition like the distance to the closes slot which is free.

```
public class SmartObjectDistanceConsideration : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIOBJECT](#) ← [ConsiderationBase](#) ← SmartObjectDistanceConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIOBJECT.guid](#) , [AIOBJECT.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,

[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectOfType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

SlotStatus

The search query to use for finding the smart object if useNearestSlotPosition is true

```
[Tooltip("The search query to use for finding the smart object if useNearestSlotPosition  
is true")]  
public SmartObjectsManager.SearchQueryOptions SlotStatus
```

Field Value

[SmartObjectsManager.SearchQueryOptions](#)

Returns the distance to a target which is a smart object. You can choose to get the distance to a specific slot with a specific condition like the distance to the closes slot which is free.

inverse

Invert the final score (0.8 --> 0.2

```
[Tooltip("Invert the final score ( 0.8 --> 0.2")]  
public bool inverse
```

Field Value

[bool](#)

Returns the distance to a target which is a smart object. You can choose to get the distance to a specific slot with a specific condition like the distance to the closes slot which is free.

useNearestSlotPosition

If true, score is computed using the best slot position, not the smart object position.

```
[Tooltip("If true, score is computed using the best slot position, not the smart  
object position.")]  
public bool useNearestSlotPosition
```

Field Value

[bool](#)

Returns the distance to a target which is a smart object. You can choose to get the distance to a specific slot with a specific condition like the distance to the closes slot which is free.

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

Class TargetDistanceConsideration

Namespace: [NoOpArmy.UtilityAI.Considerations](#)

Assembly: APIRef.dll

Returns the distance to a target. This normalizes to a value between 0 and 1 based on minRange and maxRange values

```
public class TargetDistanceConsideration : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ← TargetDistanceConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,

[Object.FindAnyObjectByType\(Type\)](#) , [ObjectFindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

inverse

Invert the final score (0.8 --> 0.2

```
[Tooltip("Invert the final score ( 0.8 --> 0.2")]
public bool inverse
```

Field Value

[bool](#)

Returns the distance to a target. This normalizes to a value between 0 and 1 based on minRange and maxRange values

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action

stores in its target list in the `UpdateTargets` call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#) ↗

A value between `minRange` and `maxRange` which indicates the current value/score of the consideration

Namespace NoOpArmy.UtilityAI.Sample

Classes

[BlackboardGameObjectDistanceConsideration](#)

Returns the distance between the executing agent and a GameObject which is specified in a key in the blackboard.

[ConstBoolConsideration](#)

Returns a constant value. This consideration is mostly used for testing

[PlayWithBlackboard](#)

[ValidPathConsideration](#)

This consideration returns 1 if a valid path from the agent's NavmeshAgent to the position stored in the Vector3 key is present, otherwise returns 0. The NeedTarget causes the Vector3 to be read from the target but always a path from the owning agent is calculated

Class

BlackboardGameObjectDistanceConsideration

Namespace: [NoOpArmy.UtilityAI.Sample](#)

Assembly: APIRef.dll

Returns the distance between the executing agent and a GameObject which is specified in a key in the blackboard.

```
public class BlackboardGameObjectDistanceConsideration : ConsiderationBase
```

Inheritance

[Object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ←

BlackboardGameObjectDistanceConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ↗ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↗ , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) ↗ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↗ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↗ ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ↗ ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) ↗ , [Object.FindObjectsOfType<T>\(\)](#) ,
[Object.FindObjectsByType<T>\(FindObjectsSortMode\)](#) , [Object.FindObjectsOfType<T>\(bool\)](#) ↗ ,
[Object.FindObjectsByType<T>\(FindObjectInactive, FindObjectsSortMode\)](#) ,
[Object.FindObjectOfType<T>\(\)](#) , [Object.FindObjectOfType<T>\(bool\)](#) ↗ ,
[Object.FindFirstObjectOfType<T>\(\)](#) , [Object.FindAnyObjectOfType<T>\(\)](#) ,

```
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,  
Object.FindAnyObjectByType<T>(FindObjectsInactive) , Object.FindObjectsOfTypeAll\(Type\) ,  
Object.FindObjectOfType\(Type\) , Object.FindFirstObjectByType\(Type\) ,  
Object.FindAnyObjectByType\(Type\) , Object.FindObjectOfType\(Type, bool\) ,  
Object.FindFirstObjectByType\(Type, FindObjectsInactive\) ,  
Object.FindAnyObjectByType\(Type, FindObjectsInactive\) , Object.ToString() , Object.name ,  
Object.hideFlags , object.Equals\(object, object\) , object.GetType\(\) , object.MemberwiseClone\(\) ,  
object.ReferenceEquals\(object, object\)
```

Fields

keyName

Name of the key in the blackboard

```
[Tooltip("Name of the key in the blackboard")]  
public string keyName
```

Field Value

[string](#)

Returns the distance between the executing agent and a GameObject which is specified in a key in the blackboard.

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#) ↗

A value between minRange and maxRange which indicates the current value/score of the consideration

Class ConstBoolConsideration

Namespace: [NoOpArmy.UtilityAI.Sample](#)

Assembly: APIRef.dll

Returns a constant value. This consideration is mostly used for testing

```
public class ConstBoolConsideration : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIOBJECT](#) ← [ConsiderationBase](#) ← ConstBoolConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIOBJECT.guid](#) , [AIOBJECT.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectsInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectType\(Type, bool\)](#) ,

[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

falseValue

The value to return if isTrue is not set.

```
public float floatValue
```

Field Value

[float](#)

Returns a constant value. This consideration is mostly used for testing

isTrue

Is this consideration in the true condition or not

```
public bool isTrue
```

Field Value

[bool](#)

Returns a constant value. This consideration is mostly used for testing

trueValue

The value to return if isTrue is set.

```
public float trueValue
```

Field Value

[float](#) ↗

Returns a constant value. This consideration is mostly used for testing

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#) ↗

A value between minRange and maxRange which indicates the current value/score of the consideration

Class PlayWithBlackboard

Namespace: [NoOpArmy.UtilityAI.Sample](#)

Assembly: APIRef.dll

```
public class PlayWithBlackboard : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← PlayWithBlackboard

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,

[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

blackboard

The blackboard to read/write to

```
public BlackBoard blackboard
```

Field Value

[BlackBoard](#)

changer

```
public BlackBoardChangeer changer
```

Field Value

[BlackBoardChangeer](#)

colorChange

```
public BlackBoardChange colorChange
```

Field Value

[BlackBoardChange](#)

Class ValidPathConsideration

Namespace: [NoOpArmy.UtilityAI.Sample](#)

Assembly: APIRef.dll

This consideration returns 1 if a valid path from the agent's NavmeshAgent to the position stored in the Vector3 key is present, otherwise returns 0 The NeedTarget causes the Vector3 to be read from the target but always a path from the owning agent is calculated

```
public class ValidPathConsideration : ConsiderationBase
```

Inheritance

[Object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ← ValidPathConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.UpdateRange\(float, float\)](#) ,
[AIObject.guid](#) , [AIObject.Name](#) , ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ↗ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↗ , ScriptableObject.CreateInstance<T>() , Object.GetGetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) ↗ , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↗ ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ↗ ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ↗ ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) ↗ , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ↗ ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ↗ ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ↗ ,
[Object.FindObjectType\(Type\)](#) ↗ , [Object.FindFirstObjectByType\(Type\)](#) ↗ ,

[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

navMeshPath

`public NavMeshPath navMeshPath`

Field Value

NavMeshPath

This consideration returns 1 if a valid path from the agent's NavmeshAgent to the position stored in the Vector3 key is present, otherwise returns 0 The NeedTarget causes the Vector3 to be read from the target but always a path from the owning agent is calculated

vector3KeyName

`public string vector3KeyName`

Field Value

[string](#)

This consideration returns 1 if a valid path from the agent's NavmeshAgent to the position stored in the Vector3 key is present, otherwise returns 0 The NeedTarget causes the Vector3 to be read from the target but always a path from the owning agent is calculated

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

OnInitialized()

Fired when the consideration is initialized

```
protected override void OnInitialized()
```

Namespace NoOpArmy.WiseFeline

Classes

[AIObject](#)

This is the base class of Wise Feline AI assets. You never need to use this directly.

[ActionBase](#)

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

[ActionSet](#)

Action sets are like directories which group a set of actions together but you don't need to deal with them in code unless you are making debugging/editor tools or highly advanced systems An Agent Behavior asset is made of a set of ActionSet objects which each of them contain a set of actions.

[AgentBehavior](#)

This class is used to serialize a group of action sets as a scriptable object. You usually create one of these by going to the Assets>Create>NoOpArmy>WiseFeline>AgentBehavior menu and then select it and open the Window>NoOpArmy>WiseFeline to add action sets, actions and considerations to it and modify their parameters.

[Brain](#)

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

[CharacterControllerBasedAIMovement](#)

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

[ConsiderationBase](#)

Considerations should inherit from this class. Each action has a list of these and multiplies the value of calling their GetValue methods to calculate its score. Each consideration is either a consideration for the target of the action or the action itself. The target based ones will be executed once per target when the action score is being calculated for that target.

[ExtensionMethods](#)

These are different methods extending types with methods we needed to make the code more readable.

[NavmeshBasedAIMoement](#)

Movement component allows you to move an agent on the NavMesh. This component implements the IAIMovement interface so generic actions can use it.

[ReadOnlyAttribute](#)

This attribute has a property drawer which causes the property to be drawn without the ability to be edited.

[ReflectionUtilities](#)

Contains utility methods for working with data types defined as actions and considerations and ... You should not need to use this unless you are making debug/editor tools for Wise Feline

[Search](#)

This is a utility class which you can use to find GameObjects in the scene. However this is not the most performant way to do so and you can use any mechanism which suits your game. However these methods have the minimum like not allocating memory for colliders

Structs

[ActionData](#)

This struct represents the runtime info for a an action and its final score

[ActionScoringData](#)

Enums

[Brain.ActionSelectionAlgorithm](#)

Different action selection modes

Class AIObject

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

This is the base class of Wise Feline AI assets. You never need to use this directly.

```
public class AIObject : ScriptableObject
```

Inheritance

[object](#) ← Object ← ScriptableObject ← AIObject

Derived

[ActionBase](#), [ActionSet](#), [ConsiderationBase](#)

Inherited Members

ScriptableObject.SetDirty(), [ScriptableObject.CreateInstance\(string\)](#),
[ScriptableObject.CreateInstance\(Type\)](#), ScriptableObject.CreateInstance<T>(), Object.GetInstanceId(),
Object.GetHashCode(), [Object.Equals\(object\)](#), Object.Instantiate(Object, Vector3, Quaternion),
Object.Instantiate(Object, Vector3, Quaternion, Transform), Object.Instantiate(Object),
Object.Instantiate(Object, Transform), [Object.Instantiate\(Object, Transform, bool\)](#),
Object.Instantiate<T>(T), Object.Instantiate<T>(T, Vector3, Quaternion),
Object.Instantiate<T>(T, Vector3, Quaternion, Transform), Object.Instantiate<T>(T, Transform),
[Object.Instantiate<T>\(T, Transform, bool\)](#), [Object.Destroy\(Object, float\)](#), Object.Destroy(Object),
[Object.DestroyImmediate\(Object, bool\)](#), Object.DestroyImmediate(Object),
[Object.FindObjectsOfType\(Type\)](#), [Object.FindObjectsOfType\(Type, bool\)](#),
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#),
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#),
Object.DontDestroyOnLoad(Object), [Object.DestroyObject\(Object, float\)](#),
Object.DestroyObject(Object), [Object.FindSceneObjectsOfType\(Type\)](#),
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#), Object.FindObjectsOfType<T>(),
Object.FindObjectsByType<T>(FindObjectsSortMode), [Object.FindObjectsOfType<T>\(bool\)](#),
Object.FindObjectsByType<T>(FindObjectInactive, FindObjectsSortMode),
Object.FindObjectOfType<T>(), [Object.FindObjectOfType<T>\(bool\)](#),
Object.FindFirstObjectOfType<T>(), Object.FindAnyObjectOfType<T>(),
Object.FindFirstObjectOfType<T>(FindObjectInactive),
Object.FindAnyObjectOfType<T>(FindObjectInactive), [Object.FindObjectsOfTypeAll\(Type\)](#),
[Object.FindObjectOfType\(Type\)](#), [Object.FindFirstObjectOfType\(Type\)](#),
[Object.FindAnyObjectOfType\(Type\)](#), [Object.FindObjectOfType\(Type, bool\)](#),

[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

Name

The name of the asset

```
public string Name
```

Field Value

[string](#)

This is the base class of Wise Feline AI assets. You never need to use this directly.

guid

The unique id of the asset.

```
[HideInInspector]  
public string guid
```

Field Value

[string](#)

This is the base class of Wise Feline AI assets. You never need to use this directly.

Class ActionBase

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

```
public abstract class ActionBase : AIObject
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← ActionBase

Derived

[BlackboardActionBase](#), [MoveAndPatrol](#)

Inherited Members

[AIObject.guid](#) , [AIObject.Name](#) , ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetGetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectsInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,

[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

ChosenTarget

The target with the best score. The type of this component depends on the type that you yourself store in the list in the UpdateTargets callback. If the action doesn't have any targets and target based considerations then this field doesn't matter and it value should be considered undefined.

protected Component ChosenTarget

Field Value

Component

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

_isInterruptable

Is this action interruptable mid-execution by another action which has a higher score

public **bool** _isInterruptable

Field Value

[bool](#)

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

_maxTargetCount

Maximum number of targets which this action should consider

```
[SerializeField]  
protected int _maxTargetCount
```

Field Value

[int](#)

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

_priority

This is the priority of the action. Actions will be chosen for execution if their score is non-zero even if actions with lower priorities have higher scores

```
public int _priority
```

Field Value

[int](#)

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

_useMomentumOnTarget

Should the action add a 25% score bonus to the current target to not change the target multiple times too quickly when scores are too close.

```
[SerializeField]  
protected bool _useMomentumOnTarget
```

Field Value

[bool](#)

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

Properties

Brain

The brain component which the action is executing for.

```
protected Brain Brain { get; }
```

Property Value

[Brain](#)

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

Considerations

List of the considerations for the action. The score of the action is the result of multiplication of the scores of all of these considerations

```
public List<ConsiderationBase> Considerations { get; }
```

Property Value

[List](#) <[ConsiderationBase](#)>

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

IsInitialized

Is the action initialized?

```
public bool IsInitialized { get; protected set; }
```

Property Value

[bool](#) ↗

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

Score

The last calculated score of the action in the last think operation.

```
public float Score { get; }
```

Property Value

[float](#) ↗

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

Weight

The wait of the action which is multiplied by the score to allow you to prioritize some actions

```
public float Weight { get; }
```

Property Value

[float](#) ↗

AI actions should derive from this class to define a custom action which can be added to the set of actions for an agent

Methods

ActionFailed()

```
protected void ActionFailed()
```

ActionSucceed()

```
protected void ActionSucceed()
```

ActionSucceeded()

```
protected void ActionSucceeded()
```

AddTarget(Component)

Adds a component to the targets list of the action

```
protected void AddTarget(Component t)
```

Parameters

t Component

AddTargets<T>(List<T>)

Adds an array of targets to the list of targets for the action

```
protected void AddTargets<T>(List<T> targets) where T : Component
```

Parameters

targets [List](#)<T>

Type Parameters

T

AddTargets<T>(T[])

Adds an array of targets to the list of targets for the action

```
protected void AddTargets<T>(T[] targets) where T : Component
```

Parameters

targets T[]

Type Parameters

T

ClearTargets()

Clears the list of targets for the action

```
protected void ClearTargets()
```

OnFinish()

Called when the action is finished, either by failing or succeeding in achieving a desired behaviour

```
protected virtual void OnFinish()
```

OnFixedUpdate()

Should be used like MonoBehaviour's FixedUpdate for the action

```
protected virtual void OnFixedUpdate()
```

OnInitialized()

Called when the action is initialized

```
protected virtual void OnInitialized()
```

OnLateUpdate()

Should be used like MonoBehaviour's LateUpdate for the action

```
protected virtual void OnLateUpdate()
```

OnStart()

Should be used like MonoBehaviour's Start for the action

```
protected virtual void OnStart()
```

OnUpdate()

Should be used like MonoBehaviour's Update for the action

```
protected virtual void OnUpdate()
```

RemoveTarget(Component)

Removes a component from the list of targets for the action

```
protected void RemoveTarget(Component t)
```

Parameters

t Component

UpdateTargets()

Fires when you need to update the list of targets

```
protected abstract void UpdateTargets()
```

Struct ActionData

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

This struct represents the runtime info for a an action and its final score

```
public struct ActionData
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

ActionData(ActionSet, ActionBase, float, int, int)

```
public ActionData(ActionSet set, ActionBase action, float score, int priority, int index)
```

Parameters

set [ActionSet](#)

This struct represents the runtime info for a an action and its final score

action [ActionBase](#)

This struct represents the runtime info for a an action and its final score

score [float](#)

This struct represents the runtime info for a an action and its final score

priority [int](#)

This struct represents the runtime info for a an action and its final score

index [int](#)

This struct represents the runtime info for a an action and its final score

Fields

Action

Which action this instance refers to.

```
public ActionBase Action
```

Field Value

[ActionBase](#)

This struct represents the runtime info for a an action and its final score

ActionSet

The action set this instance refers to.

```
public ActionSet ActionSet
```

Field Value

[ActionSet](#)

This struct represents the runtime info for a an action and its final score

Index

index of the action in the main list

```
public int Index
```

Field Value

[int](#)

This struct represents the runtime info for a an action and its final score

Priority

Priority of the action

```
public int Priority
```

Field Value

[int](#)

This struct represents the runtime info for a an action and its final score

Score

The score of the action in the last think operation.

```
public float Score
```

Field Value

[float](#)

This struct represents the runtime info for a an action and its final score

Struct ActionScoringData

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

```
public struct ActionScoringData
```

Inherited Members

[ValueType.Equals\(object\)](#) , [ValueType.GetHashCode\(\)](#) , [ValueType.ToString\(\)](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Constructors

ActionScoringData(float, int, int)

```
public ActionScoringData(float score, int priority, int index)
```

Parameters

score [float](#)

priority [int](#)

index [int](#)

Fields

Index

index of the action in the main list

```
public int Index
```

Field Value

[int](#)

Priority

Priority of the action

```
public int Priority
```

Field Value

[int](#)

Score

The score of the action in the last think operation.

```
public float Score
```

Field Value

[float](#)

Class ActionSet

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

Action sets are like directories which group a set of actions together but you don't need to deal with them in code unless you are making debugging/editor tools or highly advanced systems An Agent Behavior asset is made of a set of ActionSet objects which each of them contain a set of actions.

```
public class ActionSet : AIObject
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← ActionSet

Inherited Members

[AIObject.guid](#) , [AIObject.Name](#) , ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) , [ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceId() , Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) , Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) , Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) , Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) , Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) , [Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) , [Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) , [Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) , [Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) , [Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) , Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) , Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) , [Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() , Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) , Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) , Object.FindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) , Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() , Object.FindFirstObjectOfType<T>(FindObjectsInactive) , Object.FindAnyObjectOfType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) , [Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) , [Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) , [Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,

[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

Behavior

The agent behavior that this instance belongs too.

```
[HideInInspector]  
public AgentBehavior Behavior
```

Field Value

[AgentBehavior](#)

Action sets are like directories which group a set of actions together but you don't need to deal with them in code unless you are making debugging/editor tools or highly advanced systems An Agent Behavior asset is made of a set of ActionSet objects which each of them contain a set of actions.

Properties

Actions

The list of actions which are contained in this action set.

```
public List<ActionBase> Actions { get; }
```

Property Value

[List](#) <[ActionBase](#)>

Action sets are like directories which group a set of actions together but you don't need to deal with them in code unless you are making debugging/editor tools or highly advanced systems An Agent Behavior asset is made of a set of ActionSet objects which each of them contain a set of actions.

Class AgentBehavior

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

This class is used to serialize a group of action sets as a scriptable object. You usually create one of these by going to the Assets>Create>NoOpArmy>WiseFeline>AgentBehavior menu and then select it and open the Window>NoOpArmy>WiseFeline to add action sets, actions and considerations to it and modify their parameters.

```
[CreateAssetMenu(menuName = "NoOpArmy/Wise Feline/AgentBehavior")]
public class AgentBehavior : ScriptableObject
```

Inheritance

[object](#) ← Object ← ScriptableObject ← AgentBehavior

Inherited Members

ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,

[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

OnThinkDone

Fires when an action is done

```
public Action<ActionData> OnThinkDone
```

Field Value

[Action](#) <ActionData>

This class is used to serialize a group of action sets as a scriptable object. You usually create one of these by going to the Assets>Create>NoOpArmy>WiseFeline>AgentBehavior menu and then select it and open the Window>NoOpArmy>WiseFeline to add action sets, actions and considerations to it and modify their parameters.

SelectedAction

The action currently selected

```
[HideInInspector]  
public ActionBase SelectedAction
```

Field Value

[ActionBase](#)

This class is used to serialize a group of action sets as a scriptable object. You usually create one of these by going to the Assets>Create>NoOpArmy>WiseFeline>AgentBehavior menu and then select it and open the Window>NoOpArmy>WiseFeline to add action sets, actions and considerations to it and modify their parameters.

SelectedActionSet

The action set selected at the moment in the UI

```
[HideInInspector]  
public ActionSet SelectedActionSet
```

Field Value

[ActionSet](#)

This class is used to serialize a group of action sets as a scriptable object. You usually create one of these by going to the Assets>Create>NoOpArmy>WiseFeline>AgentBehavior menu and then select it and open the Window>NoOpArmy>WiseFeline to add action sets, actions and considerations to it and modify their parameters.

Properties

ActionSets

The list of action sets in the asset

```
public List<ActionSet> ActionSets { get; }
```

Property Value

[List](#)<[ActionSet](#)>

This class is used to serialize a group of action sets as a scriptable object. You usually create one of these by going to the Assets>Create>NoOpArmy>WiseFeline>AgentBehavior menu and then select it and open the Window>NoOpArmy>WiseFeline to add action sets, actions and considerations to it and modify their parameters.

Methods

Clone()

Clones the behavior for runtime execution by a Brain component.

```
public AgentBehavior Clone()
```

Returns

[AgentBehavior](#)

Class Brain

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

```
public class Brain : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← Brain

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,

[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

_thinkDuration

The wait period between calculations of action scores

```
public float _thinkDuration
```

Field Value

[float](#) 

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

_updateTargetsDuration

The wait period between calls to update target lists. Since updating this list is usually heavy, you should do this less often than score calculations and deal with null targets in your actions.

```
public float _updateTargetsDuration
```

Field Value

[float](#) 

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

actionSelectionAlgorithm

The algorithm the brain uses for selecting the best action

```
[Tooltip("The algorithm the brain uses for selecting the best action")]
public Brain.ActionSelectionAlgorithm actionSelectionAlgorithm
```

Field Value

[Brain.ActionSelectionAlgorithm](#)

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

context

Assign any component which your AI always needs to this to avoid calling GetComponent in all actions/considerations

```
public Component context
```

Field Value

Component

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

itemCountForRandomActionSelection

The top N items are considered for random action selection and this value is what N is.

```
[Tooltip("The top N items are considered for random action selection and this value is what  
N is.")]  
public int itemCountForRandomActionSelection
```

Field Value

int ↗

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

scoreDeltaForMidExecutionActionChange

If the score difference between the chosen action and the currently executing one is higher than this, then the actions changes to the chosen one

```
[Tooltip("If the score difference between the chosen action and the currently executing one  
is higher than this, then the actions changes to the chosen one")]  
public float scoreDeltaForMidExecutionActionChange
```

Field Value

[float](#) ↗

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

Properties

Behavior

```
public AgentBehavior Behavior { get; }
```

Property Value

[AgentBehavior](#)

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

currentAction

Gets the currently executing action class

```
public ActionBase currentAction { get; }
```

Property Value

[ActionBase](#)

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

Methods

AddActionSet(ActionSet)

Adds an action set to the set of behaviors this brain has available to score and execute

```
public void AddActionSet(ActionSet set)
```

Parameters

set [ActionSet](#)

The action set to add

Remarks

Duplicate actions which already exist in the brain will be added again too so you should design your action sets accordingly. You usually do not need multiple instances of an action in the same brain

AddBehavior(AgentBehavior)

Adds actions inside a behavior agent asset to the actions of this brain

```
public void AddBehavior(AgentBehavior agentBehavior)
```

Parameters

agentBehavior [AgentBehavior](#)

The agent behavior asset to add

Remarks

Duplicate actions which already exist in the brain will be added again too. You should design your behaviors so they don't have duplicate actions. You usually do not need multiple instances of an action in the same brain

PauseExecutingActions(bool)

Pauses/Resumes executing the current action.

```
public void PauseExecutingActions(bool pause)
```

Parameters

`pause` [bool](#)

pauses execution if true and resumes it if false

PauseThinking(bool)

Pauses/Resumes thinking which changes action scores

```
public void PauseThinking(bool pause)
```

Parameters

`pause` [bool](#)

Pauses the thinking if true and resumes it if false.

RemoveActionSet(ActionSet)

Removes a specific action set from the list of behaviors of this brain

```
public void RemoveActionSet(ActionSet set)
```

Parameters

`set` [ActionSet](#)

RemoveBehavior(AgentBehavior)

Removes actions of a behavior from this brain

```
public void RemoveBehavior(AgentBehavior agentBehavior)
```

Parameters

`agentBehavior` [AgentBehavior](#)

Think()

Calculates all action scores and selects the next action to execute. This might or might not change the currently executing action

```
public void Think()
```

UpdateActionsTargets()

Updates the target list of all actions so the updated targets can be used while thinking

```
public void UpdateActionsTargets()
```

Events

OnActionFailed

This event gets fired when an action fails

```
public event Action<ActionBase> OnActionFailed
```

Event Type

[Action](#) <[ActionBase](#)>

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

OnActionSucceeded

This event gets fired when an action succeeds

```
public event Action<ActionBase> OnActionSucceeded
```

Event Type

[Action](#) <ActionBase>

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

OnBehaviorListModified

Fires when the list of behaviors available to this brain is modified by calling AddBehavior() or RemoveBehavior()

```
public event Action OnBehaviorListModified
```

Event Type

[Action](#)

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

OnCurrentActionChanged

Fires when the executing action changes. The first parameter is the previous action and the second one is the action we are changing too If the first parameter is null then either the previous action is fully finished successfully or with a failure or this is the first action to execute

```
public event Action<ActionBase, ActionBase> OnCurrentActionChanged
```

Event Type

[Action](#) <ActionBase, ActionBase>

You should attach this component to any GameObject which wishes to be an AI agent controlled by a set of actions

Enum Brain.ActionSelectionAlgorithm

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

Different action selection modes

```
public enum Brain.ActionSelectionAlgorithm : byte
```

Fields

HighestScore = 0

Choose the action with the highest score

RandomConsideringPriority = 1

Randomly choose from the top N without considering priority

RandomWithoutConsideringPriority = 2

Randomly choose from the top N without considering priority so only actions with priorities as high as the top one will be considered

WeightedRandomConsideringPriority = 3

Weighted Randomly choose from the top N without considering priority

WeightedRandomWithoutConsideringPriority = 4

Weighted Randomly choose from the top N without considering priority so only actions with priorities as high as the top one will be considered

Class CharacterControllerBasedAIMovement

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

```
[RequireComponent(typeof(CharacterController))]
public class CharacterControllerBasedAIMovement : MonoBehaviour, IAIMovement
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← CharacterControllerBasedAIMovement

Implements

[IAIMovement](#)

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) , [MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) , [MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) , [MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) , [MonoBehaviour.StartCoroutine Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) , MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) , MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout , MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled , [Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() , [Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) , [Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) , [Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) , Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) , [Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) , [Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) , Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) , [Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) , [Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() , [Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) , [Component.GetComponentsInParent<T>\(bool\)](#) ,

[Component.GetComponentInParent<T>\(bool, List<T>\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

CalculateOnXZPlane

```
public bool CalculateOnXZPlane
```

Field Value

[bool ↗](#)

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

arrivalDistance

The distance to destination which means we arrived

```
[Tooltip("The distance to destination which means we arrived")]
public float arrivalDistance
```

Field Value

[float ↗](#)

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

speed

Movement speed of the character used by the CharacterController.SimpleMove() function

```
[Tooltip("Movement speed of the character used by the CharacterController.SimpleMove()
function")]
public float speed
```

Field Value

[float](#)

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

Properties

Destination

Should contain the movement destination

```
public Vector3 Destination { get; }
```

Property Value

Vector3

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

Methods

MoveToPosition(GameObject, Action)

```
public bool MoveToPosition(GameObject target, Action callback)
```

Parameters

target GameObject

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

callback [Action](#)

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

Returns

[bool](#)

This component allows you to move the character to a destination. This component implements the IAIMovement interface so generic actions can use it. The component uses the CharacterController component to move and does not use path finding and a NavmeshAgent

MoveToPosition(Vector3, Action)

Moves the character to a position specified by a Vector3 and calls a callback when arrived

```
public bool MoveToPosition(Vector3 position, Action callback)
```

Parameters

position Vector3

callback [Action](#)

Returns

[bool](#)

Ture if the movement can happen, false otherwise. Path Finding failures and other issues can prevent the movement to happen.

StopMoving()

Stops the movement no matter if we arrived at the set destination or not

```
public void StopMoving()
```

Class ConsiderationBase

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

Considerations should inherit from this class. Each action has a list of these and multiplies the value of calling their GetValue methods to calculate its score. Each consideration is either a consideration for the target of the action or the action itself. The target based ones will be executed once per target when the action score is being calculated for that target.

```
public abstract class ConsiderationBase : AIObject
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← ConsiderationBase

Derived

[BlackboardBoolConsideration](#), [BlackboardConsiderationKeyExists](#), [BlackboardFloatConsideration](#),
[BlackboardIntRangeConsideration](#), [BlackboardIntValueConsideration](#), [InfluenceMapConsideration](#),
[SmartObjectDistanceConsideration](#), [TargetDistanceConsideration](#),
[BlackboardGameObjectDistanceConsideration](#), [ConstBoolConsideration](#), [ValidPathConsideration](#),
[AITagConsideration](#)

Inherited Members

[AIObject.guid](#) , [AIObject.Name](#) , [ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , [Object.FindObjectsOfType<T>\(\)](#) ,
[Object.FindObjectsByType<T>\(FindObjectsSortMode\)](#) , [Object.FindObjectsOfType<T>\(bool\)](#) ,

```
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode),  
Object.FindObjectOfType<T>() , Object.FindObjectOfType<T>\(bool\) ,  
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,  
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,  
Object.FindAnyObjectByType<T>(FindObjectsInactive) , Object.FindObjectsOfTypeAll\(Type\) ,  
Object.FindObjectType\(Type\) , Object.FindFirstObjectType\(Type\) ,  
Object.FindAnyObjectType\(Type\) , Object.FindObjectType\(Type, bool\) ,  
Object.FindFirstObjectType\(Type, FindObjectsInactive\) ,  
Object.FindAnyObjectType\(Type, FindObjectsInactive\) , Object.ToString() , Object.name ,  
Object.hideFlags , object.Equals\(object, object\) , object.GetType\(\) , object.MemberwiseClone\(\) ,  
object.ReferenceEquals\(object, object\)
```

Fields

instantiatePerAgent

If true, a new instance of the consideration will be created per action per instance of the brain component which needs this. Otherwise a single instance will be used for all of them. If you don't use fields in the class and the GetValue method is stateless, then turn this off to allocate less memory.

```
[Header("Optimization")]  
[SerializeField]  
[Tooltip("If you don't need a new instance of this class per brain turn this off, this will  
share a unique class with all agents.")]  
protected bool instantiatePerAgent
```

Field Value

[bool](#)

Considerations should inherit from this class. Each action has a list of these and multiplies the value of calling their GetValue methods to calculate its score. Each consideration is either a consideration for the target of the action or the action itself. The target based ones will be executed once per target when the action score is being calculated for that target.

maxRange

Maximum value that the consideration can return from GetValue

```
[SerializeField]
protected float maxRange
```

Field Value

[float](#) ↗

Considerations should inherit from this class. Each action has a list of these and multiplies the value of calling their GetValue methods to calculate its score. Each consideration is either a consideration for the target of the action or the action itself. The target based ones will be executed once per target when the action score is being calculated for that target.

minRange

Minimum value that the consideration can return from GetValue

```
[SerializeField]
protected float minRange
```

Field Value

[float](#) ↗

Considerations should inherit from this class. Each action has a list of these and multiplies the value of calling their GetValue methods to calculate its score. Each consideration is either a consideration for the target of the action or the action itself. The target based ones will be executed once per target when the action score is being calculated for that target.

Properties

Brain

The Brain component that this consideration is being attached to one of its actions

```
protected Brain Brain { get; }
```

Property Value

Brain

Considerations should inherit from this class. Each action has a list of these and multiplies the value of calling their GetValue methods to calculate its score. Each consideration is either a consideration for the target of the action or the action itself. The target based ones will be executed once per target when the action score is being calculated for that target.

Score

The score of this consideration returned from the last GetValue call.

```
public float Score { get; }
```

Property Value

[float](#)

Considerations should inherit from this class. Each action has a list of these and multiplies the value of calling their GetValue methods to calculate its score. Each consideration is either a consideration for the target of the action or the action itself. The target based ones will be executed once per target when the action score is being calculated for that target.

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected abstract float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action

stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

OnInitialized()

Fired when the consideration is initialized

```
protected virtual void OnInitialized()
```

UpdateRange(float, float)

Updates the range of acceptable values from GetValue

```
public void UpdateRange(float min, float max)
```

Parameters

min [float](#)

max [float](#)

Class ExtensionMethods

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

These are different methods extending types with methods we needed to make the code more readable.

```
public static class ExtensionMethods
```

Inheritance

[object](#) ← ExtensionMethods

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

MaxIndex(float[])

Gets the index of the item with the highest value in the array.

```
public static int MaxIndex(this float[] array)
```

Parameters

array [float](#)[]

Returns

[int](#)

MaxIndices(float[])

Returns the indices of all items with the maximum value in the array.

```
public static int[] MaxIndices(this float[] array)
```

Parameters

array [float](#)[]

Returns

[int](#)[]

Class NavmeshBasedAIMoement

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

Movement component allows you to move an agent on the NavMesh. This component implements the IAIMovement interface so generic actions can use it.

```
[RequireComponent(typeof(NavMeshAgent))]  
public class NavmeshBasedAIMoement : MonoBehaviour, IAIMovement
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← NavmeshBasedAIMoement

Implements

[IAIMovement](#)

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,

[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectOfType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Properties

Destination

Should contain the movement destination

```
public Vector3 Destination { get; set; }
```

Property Value

Vector3

Movement component allows you to move an agent on the NavMesh. This component implements the IAIMovement interface so generic actions can use it.

Methods

MoveToPosition(GameObject, Action)

```
public bool MoveToPosition(GameObject go, Action callback)
```

Parameters

go GameObject

Movement component allows you to move an agent on the NavMesh. This component implements the IAIMovement interface so generic actions can use it.

callback [Action](#)

Movement component allows you to move an agent on the NavMesh. This component implements the IAIMovement interface so generic actions can use it.

Returns

[bool](#)

Movement component allows you to move an agent on the NavMesh. This component implements the IAIMovement interface so generic actions can use it.

MoveToPosition(Vector3, Action)

Moves the agent to a position

```
public bool MoveToPosition(Vector3 position, Action callback)
```

Parameters

position Vector3

callback [Action](#)

Returns

[bool](#)

Movement component allows you to move an agent on the NavMesh. This component implements the IAIMovement interface so generic actions can use it.

StopMoving()

Stops the movement no matter if we arrived at the set destination or not

```
public void StopMoving()
```

Class ReadOnlyAttribute

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

This attribute has a property drawer which causes the property to be drawn without the ability to be edited.

```
public class ReadOnlyAttribute : PropertyAttribute
```

Inheritance

[object](#) ← [Attribute](#) ← [PropertyAttribute](#) ← [ReadOnlyAttribute](#)

Inherited Members

[PropertyAttribute.order](#) , [Attribute.Equals\(object\)](#) , [Attribute.GetCustomAttribute\(Assembly, Type\)](#) ,
[Attribute.GetCustomAttribute\(Assembly, Type, bool\)](#) ,
[Attribute.GetCustomAttribute\(MemberInfo, Type\)](#) ,
[Attribute.GetCustomAttribute\(MemberInfo, Type, bool\)](#) ,
[Attribute.GetCustomAttribute\(Module, Type\)](#) , [Attribute.GetCustomAttribute\(Module, Type, bool\)](#) ,
[Attribute.GetCustomAttribute\(ParameterInfo, Type\)](#) ,
[Attribute.GetCustomAttribute\(ParameterInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly\)](#) ,
[Attribute.GetCustomAttributes\(Assembly, bool\)](#) , [Attribute.GetCustomAttributes\(Assembly, Type\)](#) ,
[Attribute.GetCustomAttributes\(Assembly, Type, bool\)](#) , [Attribute.GetCustomAttributes\(MemberInfo\)](#) ,
[Attribute.GetCustomAttributes\(MemberInfo, bool\)](#) ,
[Attribute.GetCustomAttributes\(MemberInfo, Type\)](#) ,
[Attribute.GetCustomAttributes\(MemberInfo, Type, bool\)](#) , [Attribute.GetCustomAttributes\(Module\)](#) ,
[Attribute.GetCustomAttributes\(Module, bool\)](#) , [Attribute.GetCustomAttributes\(Module, Type\)](#) ,
[Attribute.GetCustomAttributes\(Module, Type, bool\)](#) , [Attribute.GetCustomAttributes\(ParameterInfo\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo, bool\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo, Type\)](#) ,
[Attribute.GetCustomAttributes\(ParameterInfo, Type, bool\)](#) , [Attribute.GetHashCode\(\)](#) ,
[Attribute.IsDefaultAttribute\(\)](#) , [Attribute.IsDefined\(Assembly, Type\)](#) ,
[Attribute.IsDefined\(Assembly, Type, bool\)](#) , [Attribute.IsDefined\(MemberInfo, Type\)](#) ,
[Attribute.IsDefined\(MemberInfo, Type, bool\)](#) , [Attribute.IsDefined\(Module, Type\)](#) ,
[Attribute.IsDefined\(Module, Type, bool\)](#) , [Attribute.IsDefined\(ParameterInfo, Type\)](#) ,
[Attribute.IsDefined\(ParameterInfo, Type, bool\)](#) , [Attribute.Match\(object\)](#) , [Attribute.TypeId](#) ,
[object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Class ReflectionUtilities

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

Contains utility methods for working with data types defined as actions and considerations and ... You should not need to use this unless you are making debug/editor tools for Wise Feline

```
public static class ReflectionUtilities
```

Inheritance

[object](#) ← ReflectionUtilities

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Methods

GetAllDerivedTypes(Type)

Get all types derived from a specific type defined in the project.

```
public static Type[] GetAllDerivedTypes(Type BaseType)
```

Parameters

baseType [Type](#)

Returns

[Type](#)[]

Class Search

Namespace: [NoOpArmy.WiseFeline](#)

Assembly: APIRef.dll

This is a utility class which you can use to find GameObjects in the scene. However this is not the most performant way to do so and you can use any mechanism which suits your game. However these methods have the minimum like not allocating memory for colliders

```
public static class Search
```

Inheritance

[object](#) ← Search

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Remarks

We intend to release additional modules which makes it easier to work with different geo-spatial structures and gameplay types and search for objects

Methods

FindClosestCollider(Vector3, float, LayerMask, ref Collider[])

Find the closes collider in a sphere without allocating a new array for the raycast results.

```
public static Collider FindClosestCollider(Vector3 center, float radius, LayerMask  
layerMask, ref Collider[] colliders)
```

Parameters

center Vector3

The center of the sphere

radius [float](#)

The radius of the sphere

layerMask LayerMask

The layermask which we should do the cast against

colliders Collider[]

The array of colliders to be used for the cast operation

Returns

Collider

The closes collider or null if no colliders can be found

GetOverlappingColliders(Vector3, float, LayerMask, ref Collider[])

Gets the array of overlapping colliders with a sphere

```
public static void GetOverlappingColliders(Vector3 center, float radius, LayerMask  
layerMask, ref Collider[] colliders)
```

Parameters

center Vector3

The center of the sphere

radius float

The radius of the sphere

layerMask LayerMask

The layer mask to do the cast against

colliders Collider[]

The array of colliders to be filled by the cast operation

GetSortedTransforms(Vector3, float, LayerMask, ref Collider[], ref List<Transform>, int)

Gets the list of colliders overlapping by a sphere sorted with their distance from the center. This is an expensive method only good for samples and quick prototyping because it uses a relatively expensive sort operation

```
public static void GetSortedTransforms(Vector3 center, float radius, LayerMask layerMask,  
ref Collider[] colliders, ref List<Transform> transforms, int size = 0)
```

Parameters

center Vector3

The center of the sphere

radius [float](#)

The radius of the sphere

layerMask LayerMask

The layer mask to cast against

colliders Collider[]

The array of colliders to use for the casting

transforms [List](#)<Transform>

The transforms list to fill

size [int](#)

The maximum size of the list

Namespace NoOpArmy.WiseFeline.AITags

Classes

[AITags](#)

Attach this component to any object which needs to have tags attached to it

[AITagsManager](#)

Add an instance of this to an object in each scene which needs to use AITags components

Class AITags

Namespace: [NoOpArmy.WiseFeline.AITags](#)

Assembly: APIRef.dll

Attach this component to any object which needs to have tags attached to it

```
public class AITags : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← AITags

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
[Component.GetComponentInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) ,
[Component.GetComponentInParent<T>\(bool, List<T>\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,

[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

delayBetweenPositionUpdates

How much time should we wait before updating this object's position in the octree

```
public float delayBetweenPositionUpdates
```

Field Value

[float](#) ↗

Attach this component to any object which needs to have tags attached to it

previousUpdate

When did the previous update happen

```
[HideInInspector]  
public float previousUpdate
```

Field Value

[float](#) ↗

Attach this component to any object which needs to have tags attached to it

tagsList

The initial list of tags you want to be added to the object (this is used by the inspector)

```
public List<string> tagsList
```

Field Value

[List](#) <[string](#) ↗>

Attach this component to any object which needs to have tags attached to it

updatePositionInOctreeAutomatically

Checking this will cause the object to be updated in the octree. If the object moves, you should check this and set the delay based on its speed and the accuracy you require

```
public bool updatePositionInOctreeAutomatically
```

Field Value

[bool](#)

Attach this component to any object which needs to have tags attached to it

Methods

AddItem(string)

Adds a tag at runtime

```
public void AddItem(string tag)
```

Parameters

[tag](#) [string](#)

HasAnyOfTheTags(string[])

```
public bool HasAnyOfTheTags(string[] tags)
```

Parameters

[tags](#) [string](#)[]

Attach this component to any object which needs to have tags attached to it

Returns

[bool](#)

Attach this component to any object which needs to have tags attached to it

HasTag(string)

Does this object has the tag

```
public bool HasTag(string tag)
```

Parameters

tag [string](#)

Returns

[bool](#)

RemoveItem(string)

Removes a tag at runtime

```
public void RemoveItem(string tag)
```

Parameters

tag [string](#)

SupportsFilter(string[], string[])

```
public bool SupportsFilter(string[] includeTags, string[] excludeTags)
```

Parameters

includeTags [string](#)[]

Attach this component to any object which needs to have tags attached to it

excludeTags [string](#)[]

Attach this component to any object which needs to have tags attached to it

Returns

[bool](#) ↗

Attach this component to any object which needs to have tags attached to it

Class AITagsManager

Namespace: [NoOpArmy.WiseFeline.AITags](#)

Assembly: APIRef.dll

Add an instance of this to an object in each scene which needs to use AITags components

```
[DefaultExecutionOrder(-10000)]  
public class AITagsManager : Singleton<AITagsManager>
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← [Singleton<AITagsManager>](#) ← AITagsManager

Inherited Members

[Singleton<AITagsManager>.Instance](#) , [Singleton<AITagsManager>.Awake\(\)](#) ,
[Singleton<AITagsManager>.OnDestroy\(\)](#) , MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() ,
[MonoBehaviour.Invoke\(string, float\)](#) , [MonoBehaviour.InvokeRepeating\(string, float, float\)](#) ,
[MonoBehaviour.CancelInvoke\(string\)](#) , [MonoBehaviour.IsInvoking\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string\)](#) , [MonoBehaviour.StartCoroutine\(string, object\)](#) ,
[MonoBehaviour.StartCoroutine\(IEnumerator\)](#) , [MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) ,
[MonoBehaviour.StopCoroutine\(IEnumerator\)](#) , MonoBehaviour.StopCoroutine(Coroutine) ,
[MonoBehaviour.StopCoroutine\(string\)](#) , MonoBehaviour.StopAllCoroutines() ,
[MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout , MonoBehaviour.runInEditMode ,
Behaviour.enabled , Behaviour.isActiveAndEnabled , [Component.GetComponent\(Type\)](#) ,
Component.GetComponent<T>() , [Component.TryGetComponent\(Type, out Component\)](#) ,
Component.TryGetComponent<T>(out T) , [Component.GetComponent\(string\)](#) ,
[Component.GetComponentInChildren\(Type, bool\)](#) , [Component.GetComponentInChildren\(Type\)](#) ,
[Component.GetComponentInChildren<T>\(bool\)](#) , Component.GetComponentInChildren<T>() ,
[Component.GetComponentsInChildren\(Type, bool\)](#) , [Component.GetComponentsInChildren\(Type\)](#) ,
[Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,

[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectOfType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

worldCenterObject

`public Transform worldCenterObject`

Field Value

Transform

Add an instance of this to an object in each scene which needs to use AITags components

worldSize

`public float worldSize`

Field Value

[float](#)

Add an instance of this to an object in each scene which needs to use AITags components

Properties

tree

`public PointOctree<AITags> tree { get; }`

Property Value

[PointOctree<AITags>](#)

Add an instance of this to an object in each scene which needs to use AITags components

Methods

GetTagsAroundPoint(Vector3, float)

Returns all of the AITag objects in the sphere

```
public AITags[] GetTagsAroundPoint(Vector3 point, float radius)
```

Parameters

point Vector3

radius [float](#)

Returns

[AITags](#)[]

GetTagsAroundPoint(Vector3, float, string, string)

Get objects which match the query

```
public List<AITags> GetTagsAroundPoint(Vector3 point, float radius, string includedTags,
                                         string excludedTags)
```

Parameters

point Vector3

radius [float](#)

includedTags [string](#)

A tag which the object should include, leave it empty if you don't want to check any tag for inclusion

excludedTags [string](#)

A tag which the object should not have, leave it empty if you don't want to exclude anything

Returns

[List](#)<[AITags](#)>

GetTagsAroundPoint(Vector3, float, string[], string[])

Get objects which match the query

```
public List<AITags> GetTagsAroundPoint(Vector3 point, float radius, string[] includedTags,  
string[] excludedTags)
```

Parameters

point Vector3

radius [float](#)

includedTags [string](#)[]

A tag which the object should include, leave it empty if you don't want to check any tag for inclusion

excludedTags [string](#)[]

A tag which the object should not have, leave it empty if you don't want to exclude anything

Returns

[List](#)<[AITags](#)>

Remove(AITags)

```
public void Remove(AITags ai)
```

Parameters

ai [AITags](#)

Add an instance of this to an object in each scene which needs to use AITags components

Namespace NoOpArmy.WiseFeline.Actions

Classes

[MoveAndWaitAndChangeFloat](#)

Moves the agent toward a vector3 destination and finishes the action after a specific amount of time is passed. WILL BE RENAMED IN THE NEXT RELEASE

Class MoveAndWaitAndChangeFloat

Namespace: [NoOpArmy.WiseFeline.Actions](#)

Assembly: APIRef.dll

Moves the agent toward a vector3 destination and finishes the action after a specific amount of time is passed. WILL BE RENAMED IN THE NEXT RELEASE

```
public class MoveAndWaitAndChangeFloat : BlackboardActionBase
```

Inheritance

[object](#) ↗ ← Object ← ScriptableObject ← [AIObject](#) ← [ActionBase](#) ← [BlackboardActionBase](#) ← MoveAndWaitAndChangeFloat

Inherited Members

[BlackboardActionBase.changerStartingDelayInSeconds](#) ,
[BlackboardActionBase.forceUpdateTargetsOnFinish](#) , [BlackboardActionBase.ResetChangerOnStart](#) ,
[BlackboardActionBase.changer](#) , [BlackboardActionBase.blackBoard](#) , [BlackboardActionBase.OnStart\(\)](#) ,
[BlackboardActionBase.OnFinish\(\)](#) , [ActionBase._priority](#) , [ActionBase.isInterruptable](#) ,
[ActionBase.maxTargetCount](#) , [ActionBase.useMomentumOnTarget](#) , [ActionBase.Considerations](#) ,
[ActionBase.Brain](#) , [ActionBase.IsInitialized](#) , [ActionBase.Score](#) , [ActionBase.Weight](#) ,
[ActionBase.ChosenTarget](#) , [ActionBase.AddTarget\(Component\)](#) , [ActionBase.AddTargets<T>\(T\[\]\)](#) ,
[ActionBase.AddTargets<T>\(List<T>\)](#) , [ActionBase.ClearTargets\(\)](#) , [ActionBase.RemoveTarget\(Component\)](#) ,
[ActionBase.OnLateUpdate\(\)](#) , [ActionBase.OnFixedUpdate\(\)](#) , [ActionBase.ActionSucceed\(\)](#) ,
[ActionBase.ActionSucceeded\(\)](#) , [ActionBase.ActionFailed\(\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
[ScriptableObject.SetDirty\(\)](#) , [ScriptableObject.CreateInstance\(string\)](#) ↗ ,
[ScriptableObject.CreateInstance\(Type\)](#) ↗ , [ScriptableObject.CreateInstance<T>\(\)](#) , [Object.GetInstanceId\(\)](#) ,
[Object.GetHashCode\(\)](#) , [Object.Equals\(object\)](#) ↗ , [Object.Instantiate\(Object, Vector3, Quaternion\)](#) ,
[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate\(Object\)](#) ,
[Object.Instantiate\(Object, Transform\)](#) , [Object.Instantiate\(Object, Transform, bool\)](#) ↗ ,
[Object.Instantiate<T>\(T\)](#) , [Object.Instantiate<T>\(T, Vector3, Quaternion\)](#) ,
[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#) , [Object.Instantiate<T>\(T, Transform\)](#) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) ↗ , [Object.Destroy\(Object, float\)](#) ↗ , [Object.Destroy\(Object\)](#) ,
[Object.DestroyImmediate\(Object, bool\)](#) ↗ , [Object.DestroyImmediate\(Object\)](#) ,
[Object.FindObjectsOfType\(Type\)](#) ↗ , [Object.FindObjectsOfType\(Type, bool\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ↗ ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ↗ ,
[Object.DontDestroyOnLoad\(Object\)](#) , [Object.DestroyObject\(Object, float\)](#) ↗ ,
[Object.DestroyObject\(Object\)](#) , [Object.FindSceneObjectsOfType\(Type\)](#) ↗ ,

[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

positionKeyName

Name of the Vector3 key in the blackboard

```
[Tooltip("Name of the key in the blackboard")]
public string positionKeyName
```

Field Value

[string](#)

Moves the agent toward a vector3 destination and finishes the action after a specific amount of time is passed. WILL BE RENAMED IN THE NEXT RELEASE

waitingTime

The amount of time which the action should wait in seconds

```
[Tooltip("The amount of time which the action should wait in seconds")]
public float waitingTime
```

Field Value

[float](#)

Moves the agent toward a vector3 destination and finishes the action after a specific amount of time is passed. WILL BE RENAMED IN THE NEXT RELEASE

Methods

OnInitialized()

Called when the action is initialized

```
protected override void OnInitialized()
```

OnUpdate()

Should be used like MonoBehaviour's Update for the action

```
protected override void OnUpdate()
```

UpdateTargets()

Fires when you need to update the list of targets

```
protected override void UpdateTargets()
```

Namespace NoOpArmy.WiseFeline.BlackBoards

Classes

[BlackBoard](#)

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

[BlackBoardChange](#)

Describes properties of a change to a blackboard when it comes to key name and type

[BlackBoardChangeer](#)

Specifies the properties of a change to a blackboard when it comes to number of times to do the change and delay between changes and ...

[BlackBoardDefinition](#)

This class defines the keys which can exist in a blackboard and their data types. Create instances of this using the unity editor from the create menu and assign them to the blackboard component to use. The blackboard component then will only accept these keys with these data types.

[KeyDefinition](#)

This class represents blackboard key definitions.

Enums

[BlackBoardChangeer.ChangeType](#)

The type of change to happen

[KeyDefinition.KeyType](#)

The possible key types for the blackboard

Class BlackBoard

Namespace: [NoOpArmy.WiseFeline.BlackBoards](#)

Assembly: APIRef.dll

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

```
public class BlackBoard : MonoBehaviour
```

Inheritance

[Object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← BlackBoard

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>()

[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

blackBoard

The blackboard definition this component uses

```
[Tooltip("The blackboard definition this component uses")]
public BlackBoardDefinition blackBoard
```

Field Value

[BlackBoardDefinition](#)

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

bools

Dictionary of bool keys for the blackboard

```
public Dictionary<string, bool> bools
```

Field Value

[Dictionary](#)<[string](#), [bool](#)>

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

floats

Dictionary of float keys for the blackboard

```
public Dictionary<string, float> floats
```

Field Value

[Dictionary](#) <[string](#), [float](#)>

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

gameObjects

Dictionary of GameObject keys for the blackboard

```
public Dictionary<string, GameObject> gameObjects
```

Field Value

[Dictionary](#) <[string](#), GameObject>

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

ints

Dictionary of int keys for the blackboard

```
public Dictionary<string, int> ints
```

Field Value

[Dictionary](#) <[string](#), [int](#)>

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

objects

Dictionary of object keys for the blackboard

```
public Dictionary<string, object> objects
```

Field Value

[Dictionary](#)<[string](#), [object](#)>

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

unityObjects

Dictionary of UnityEngine.Object keys for the blackboard

```
public Dictionary<string, Object> unityObjects
```

Field Value

[Dictionary](#)<[string](#), Object>

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

vector3s

Dictionary of Vector3 keys for the blackboard

```
public Dictionary<string, Vector3> vector3s
```

Field Value

[Dictionary](#) <[string](#), Vector3>

This component represents a blackboard which allows you to read and write its key value pairs and use them in your other component as you see fit. Let's say you have an enemy agent which wants to use an influence map to search for health packs, the name of the health pack can be set in a key in the blackboard and then the consideration for searching can read the map name from the blackboard

Methods

GetBool(string)

Retrieves a boolean value from the blackboard.

```
public bool GetBool(string key)
```

Parameters

key [string](#)

The key name for the boolean value.

Returns

[bool](#)

The boolean value associated with the key.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as a boolean.

GetFloat(string)

Retrieves a float value from the blackboard.

```
public float GetFloat(string key)
```

Parameters

key [string](#)

The key name for the float value.

Returns

[float](#)

The float value associated with the key.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as a float.

GetGameObject(string)

Retrieves a GameObject value from the blackboard.

```
public GameObject GetGameObject(string key)
```

Parameters

key [string](#)

The key name for the GameObject value.

Returns

[GameObject](#)

The GameObject value associated with the key.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as a GameObject.

GetInt(string)

Retrieves an integer value from the blackboard.

```
public int GetInt(string key)
```

Parameters

key [string](#)

The key name for the integer value.

Returns

[int](#)

The integer value associated with the key.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as an integer.

GetObject(string)

Retrieves an object value from the blackboard.

```
public object GetObject(string key)
```

Parameters

key [string](#)

The key name for the object value.

Returns

[object](#)

The object value associated with the key.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as an object.

GetUnityObject(string)

Retrieves a Unity Object value from the blackboard.

```
public Object GetUnityObject(string key)
```

Parameters

key [string](#)

The key name for the Unity Object value.

Returns

Object

The Unity Object value associated with the key.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as a Unity Object.

GetVector3(string)

Returns the Vector3 for the key specified

```
public Vector3 GetVector3(string key)
```

Parameters

key [string](#)

The key to return its value

Returns

Vector3

The Vector3 associated with the key

Exceptions

[ArgumentException](#)

Throws if the key is not defined in the definition as a Vector3

HasBool(string)

Does the key name exist in the bool key value pairs?

```
public bool HasBool(string key)
```

Parameters

key [string](#)

Name of the key to check

Returns

[bool](#)

True if the key exists and false otherwise

HasFloat(string)

Does the key name exist in the float key value pairs?

```
public bool HasFloat(string key)
```

Parameters

key [string](#)

Name of the key to check

Returns

[bool](#)

True if the key exists and false otherwise

HasGameObject(string)

Does the key name exist in the GameObject key value pairs?

```
public bool HasGameObject(string key)
```

Parameters

key [string](#)

Name of the key to check

Returns

[bool](#)

True if the key exists and false otherwise

HasInt(string)

Does the key name exist in the int key value pairs?

```
public bool HasInt(string key)
```

Parameters

key [string](#)

Name of the key to check

Returns

[bool](#)

True if the key exists and false otherwise

HasObject(string)

Does the key name exist in the object key value pairs?

```
public bool HasObject(string key)
```

Parameters

[key](#) [string](#)

Name of the key to check

Returns

[bool](#)

True if the key exists and false otherwise

HasUnityObject(string)

Does the key name exist in the UnityObject key value pairs?

```
public bool HasUnityObject(string key)
```

Parameters

[key](#) [string](#)

Name of the key to check

Returns

[bool](#)

True if the key exists and false otherwise

HasVector3(string)

Does the key name exist in the vector3 key value pairs?

```
public bool HasVector3(string key)
```

Parameters

key [string](#)

Name of the key to check

Returns

[bool](#)

True if the key exists and false otherwise

SetBool(string, bool)

Sets a boolean value in the blackboard.

```
public void SetBool(string key, bool value)
```

Parameters

key [string](#)

The key name for the boolean value.

value [bool](#)

The boolean value to set.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as a boolean.

SetFloat(string, float)

Sets a float value in the blackboard.

```
public void SetFloat(string key, float value)
```

Parameters

key [string](#)

The key name for the float value.

value [float](#)

The float value to set.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as a float.

SetGameObject(string, GameObject)

Sets a GameObject value in the blackboard.

```
public void SetGameObject(string key, GameObject value)
```

Parameters

key [string](#)

The key name for the GameObject value.

value [GameObject](#)

The GameObject value to set.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as a GameObject.

SetInt(string, int)

Sets an integer value in the blackboard.

```
public void SetInt(string key, int value)
```

Parameters

key [string](#)

The key name for the integer value.

value [int](#)

The integer value to set.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as an integer.

SetObject(string, object)

Sets an object value in the blackboard.

```
public void SetObject(string key, object value)
```

Parameters

key [string](#)

The key name for the object value.

value [Object](#)

The object value to set.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as an object.

SetUnityObject(string, Object)

Sets a Unity Object value in the blackboard.

```
public void SetUnityObject(string key, Object value)
```

Parameters

key [string](#)

The key name for the Unity Object value.

value [Object](#)

The Unity Object value to set.

Exceptions

[ArgumentException](#)

Thrown if the key is not defined in the blackboard definition as a Unity Object.

SetVector3(string, Vector3)

Sets a Vector3 in the blackboard

```
public void SetVector3(string key, Vector3 value)
```

Parameters

key [string](#)

The key name for the Vector3

value [Vector3](#)

The Vector3 to set

Exceptions

[ArgumentException](#)

Throws if the key is not defined in the blackboard definition as a Vector3

Class BlackBoardChange

Namespace: [NoOpArmy.WiseFeline.BlackBoards](#)

Assembly: APIRef.dll

Describes properties of a change to a blackboard when it comes to key name and type

```
[Serializable]
public class BlackBoardChange
```

Inheritance

[object](#) ← BlackBoardChange

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

addInsteadOfSet

If true, adds the value specified to the current value of the key when applicable Otherwise just set the key to the specified value

```
[Tooltip("If true, adds the value specified to the current value of the key when applicable.
otherwise just set the key to the specified value")]
public bool addInsteadOfSet
```

Field Value

[bool](#)

Describes properties of a change to a blackboard when it comes to key name and type

boolValue

Bool value to add/set if the type is Bool

```
public bool boolValue
```

Field Value

[bool](#)

Describes properties of a change to a blackboard when it comes to key name and type

floatValue

Float value to add/set if the type is Float

```
public float floatValue
```

Field Value

[float](#)

Describes properties of a change to a blackboard when it comes to key name and type

gameObjectValue

GameObject value to add/set if the type is GameObject

```
public GameObject gameObjectValue
```

Field Value

GameObject

Describes properties of a change to a blackboard when it comes to key name and type

intValue

Int value to add/set if the type is Int

```
public int intValue
```

Field Value

[int](#)

Describes properties of a change to a blackboard when it comes to key name and type

key

The key name and type to change

```
[Tooltip("The key name and type to change")]
public KeyDefinition key
```

Field Value

[KeyDefinition](#)

Describes properties of a change to a blackboard when it comes to key name and type

maxFloatValue

max float value to add/set if the type is Float

```
public float maxFloatValue
```

Field Value

[float](#)

Describes properties of a change to a blackboard when it comes to key name and type

maxIntValue

Min Int value to set if the type is Int

```
public int maxValue
```

Field Value

[int](#) ↗

Describes properties of a change to a blackboard when it comes to key name and type

minFloatValue

Min float value to set if the type is Float

```
public float minValue
```

Field Value

[float](#) ↗

Describes properties of a change to a blackboard when it comes to key name and type

minIntValue

Max Int value to set if the type is Int

```
public int minValue
```

Field Value

[int](#) ↗

Describes properties of a change to a blackboard when it comes to key name and type

multiplyByDeltaTimeForAdds

Multiply the value to add by delta time before adding

```
[Tooltip("Multiply the value to add by delta time before adding")]
public bool multiplyByDeltaTimeForAdds
```

Field Value

bool ↗

Describes properties of a change to a blackboard when it comes to key name and type

objectValue

Object value to add/set if the type is Object

```
public object objectValue
```

Field Value

object ↗

Describes properties of a change to a blackboard when it comes to key name and type

unityObjectValue

UnityEngine.Object value to add/set if the type is UnityObject

```
public Object unityObjectValue
```

Field Value

Object

Describes properties of a change to a blackboard when it comes to key name and type

vectorValue

Vector3 value to add/set if the type is Vector3

```
public Vector3 vectorValue
```

Field Value

Vector3

Describes properties of a change to a blackboard when it comes to key name and type

Methods

ApplyToBlackboard(BlackBoard)

Does the change operation based on the value of the other properties of the class

```
public void ApplyToBlackboard(BlackBoard blackBoard)
```

Parameters

blackBoard BlackBoard

The blackboard to do the changes too

Class BlackBoardChangeer

Namespace: [NoOpArmy.WiseFeline.BlackBoards](#)

Assembly: APIRef.dll

Specifies the properties of a change to a blackboard when it comes to number of times to do the change and delay between changes and ...

```
[Serializable]  
public class BlackBoardChangeer
```

Inheritance

[object](#) ← BlackBoardChangeer

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

change

The key name and type which you want the change to happen to

```
[Tooltip("The key name and type which you want the change to happen to")]  
public BlackBoardChange change
```

Field Value

[BlackBoardChange](#)

Specifies the properties of a change to a blackboard when it comes to number of times to do the change and delay between changes and ...

changeType

The type of change which you want to happen to the blackboard

```
[Tooltip("The type of change which you want to happen to the blackboard")]
public BlackBoardChangeer.ChangeType changeType
```

Field Value

[BlackBoardChangeer.ChangeType](#)

Specifies the properties of a change to a blackboard when it comes to number of times to do the change and delay between changes and ...

delayInSeconds

The delay between changes

```
[Tooltip("The delay between changes in seconds")]
public float delayInSeconds
```

Field Value

[float](#)

Specifies the properties of a change to a blackboard when it comes to number of times to do the change and delay between changes and ...

maxExecutionCount

Maximum number of times this gets executed in the repeat with delay mode. 0 means unlimited

```
[Tooltip("Maximum number of times this gets executed in the repeat with delay mode. 0
means unlimited")]
public int maxExecutionCount
```

Field Value

[int](#)

Specifies the properties of a change to a blackboard when it comes to number of times to do the change and delay between changes and ...

Methods

Reset()

```
public void Reset()
```

Update(BlackBoard)

```
public void Update(BlackBoard blackBoard)
```

Parameters

blackBoard BlackBoard

Specifies the properties of a change to a blackboard when it comes to number of times to do the change and delay between changes and ...

Enum BlackBoardChangeer.ChangeType

Namespace: [NoOpArmy.WiseFeline.BlackBoards](#)

Assembly: APIRef.dll

The type of change to happen

```
public enum BlackBoardChangeer.ChangeType : byte
```

Fields

EveryFrame = 3

Do a change every frame and multiply the value change by delta time in the case of additive changes

None = 0

No change

Once = 1

Only once change the blackBoard

RepeatWithDelay = 2

Repeat the change for a specified number of times and with a specific delay

Class BlackBoardDefinition

Namespace: [NoOpArmy.WiseFeline.BlackBoards](#)

Assembly: APIRef.dll

This class defines the keys which can exist in a blackboard and their data types. Create instances of this using the unity editor from the create menu and assign them to the blackboard component to use. The blackboard component then will only accept these keys with these data types.

```
[CreateAssetMenu(menuName = "NoOpArmy/Wise Feline/BlackBoard Definition")]
public class BlackBoardDefinition : ScriptableObject
```

Inheritance

[Object](#) ← Object ← ScriptableObject ← BlackBoardDefinition

Inherited Members

ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectsInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,

[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

definitions

List of key definitions

```
public List<KeyDefinition> definitions
```

Field Value

[List](#) <[KeyDefinition](#)>

This class defines the keys which can exist in a blackboard and their data types. Create instances of this using the unity editor from the create menu and assign them to the blackboard component to use. The blackboard component then will only accept these keys with these data types.

Class KeyDefinition

Namespace: [NoOpArmy.WiseFeline.BlackBoards](#)

Assembly: APIRef.dll

This class represents blackboard key definitions.

```
[Serializable]  
public class KeyDefinition
```

Inheritance

[object](#) ← KeyDefinition

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

keyName

Unique name of the key in the blackboard

```
public string keyName
```

Field Value

[string](#)

This class represents blackboard key definitions.

keyType

Type of the key

```
public KeyDefinition.KeyType keyType
```

Field Value

[KeyDefinition.KeyType](#)

This class represents blackboard key definitions.

Enum KeyDefinition.KeyType

Namespace: [NoOpArmy.WiseFeline.BlackBoards](#)

Assembly: APIRef.dll

The possible key types for the blackboard

```
public enum KeyDefinition.KeyType : byte
```

Fields

Bool = 4

A boolean value

Float = 2

A floating point number

GameObject = 6

A game object

Int = 3

A 32 bit integer

Object = 1

A System.Object for any type you need and we did not support

UnityObject = 5

A UnityEngine.Object for any unity object which we did not support and you need

Vector3 = 0

A UnityEngine.Vector3

Namespace NoOpArmy.WiseFeline. Considerations

Classes

[AI`Tag`Consideration](#)

This consideration checks if the target or the executing agent has or doesn't have a specific tag.

Enums

[AI`Tag`Consideration.TagOperation](#)

Class AITagConsideration

Namespace: [NoOpArmy.WiseFeline.Considerations](#)

Assembly: APIRef.dll

This consideration checks if the target or the executing agent has or doesn't have a specific tag.

```
public class AITagConsideration : ConsiderationBase
```

Inheritance

[object](#) ← Object ← ScriptableObject ← [AIObject](#) ← [ConsiderationBase](#) ← AITagConsideration

Inherited Members

[ConsiderationBase.instantiatePerAgent](#) , [ConsiderationBase.minRange](#) , [ConsiderationBase.maxRange](#) ,
[ConsiderationBase.Score](#) , [ConsiderationBase.Brain](#) , [ConsiderationBase.OnInitialized\(\)](#) ,
[ConsiderationBase.UpdateRange\(float, float\)](#) , [AIObject.guid](#) , [AIObject.Name](#) ,
ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectsInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectType\(Type, bool\)](#) ,

[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

operation

The operation to do on the tag to find the score

```
public AITagConsideration.TagOperation operation
```

Field Value

[AITagConsideration.TagOperation](#)

This consideration checks if the target or the executing agent has or doesn't have a specific tag.

tag

The tag to consider

```
public string tag
```

Field Value

[string](#)

This consideration checks if the target or the executing agent has or doesn't have a specific tag.

Methods

GetValue(Component)

This method should return the current value of the consideration which is used for multiplication with other considerations to calculate the score of the action

```
protected override float GetValue(Component target)
```

Parameters

target Component

If the consideration has a target then this value is the target component which the consideration should be calculated for. The actual type of this depends on the type of the component that the action stores in its target list in the UpdateTargets call. If the consideration doesn't have a target, then the Brain component of the executing agent itself is passed to the method.

Returns

[float](#)

A value between minRange and maxRange which indicates the current value/score of the consideration

Enum AITagConsideration.TagOperation

Namespace: [NoOpArmy.WiseFeline.Considerations](#)

Assembly: APIRef.dll

```
public enum AITagConsideration.TagOperation
```

Fields

ShouldHaveTag = 0

The object should have the tag

ShouldNotHaveTag = 1

The object should not have the tag

Namespace NoOpArmy.WiseFeline.Influence Maps

Classes

[InfluenceMap](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

[InfluenceMapCollection](#)

[InfluenceMapComponent](#)

A helper component which makes it easier to use influence maps Attach this component to a GameObject to create an influence map at awake and use it as a reference for InfluencerAgent component. The map field is the actual influence map that you can use to search for values or directly change cell values

[InfluenceMapComponentBase](#)

[InfluenceMapDecayer](#)

Decays an influence map using the value you gave it and clamps the values to not go above/beyond specified values

[InfluenceMapTemplate](#)

This asset type is used as a template for the stamp shape of agents which influence the map Usually different agents based on their behavior need both different sizes and fall off curve shapes

[InfluenceMapView](#)

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

[InfluenceMapView.ViewOperation](#)

[InfluencerAgent](#)

This component can be attached to an agent so it influences the map

[RectangularInfluencerAgent](#)

This component can be attached to an agent so it influences the map

Enums

[InfluenceMapDecayer.OperationMode](#)

The operations done on the map

InfluenceMapView.ViewOperation.Operation

InfluencerAgent.Behavior

The way that this map will add its influence

SearchCondition

The stop condition for search

Class InfluenceMap

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

```
public class InfluenceMap
```

Inheritance

[object](#) ← InfluenceMap

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

InfluenceMap(int, int, float)

Creates an influence map

```
public InfluenceMap(int mapWidth, int mapHeight, float cellSize)
```

Parameters

`mapWidth` [int](#)

Width of the map in cells

`mapHeight` [int](#)

Height of the map in cells

`cellSize` [float](#)

Size of each cell in meters

InfluenceMap(float[,], float)

Creates an influence map from a pre-defined array of cells

```
public InfluenceMap(float[,] cells, float cellSize)
```

Parameters

cells [float](#)[] []

cellSize [float](#)

Fields

anchorLocation

anchor location of the map in the world. y value is ignored in the main logic it is only useful for drawing gizmos unless you are using the 2d function

```
public Vector3 anchorLocation
```

Field Value

Vector3

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

Properties

CellSize

The cell size in meters

```
public float CellSize { get; }
```

Property Value

[float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

Height

Height of the map in cells

```
public int Height { get; }
```

Property Value

[int](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

MaxValue

Maximum value which a cell in the map has at the moment

```
public float MaxValue { get; }
```

Property Value

[float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

MinValue

Minimum value which a cell in the map has at the moment

```
public float MinValue { get; }
```

Property Value

[float ↗](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

Width

Width of the map in cells

```
public int Width { get; }
```

Property Value

[int ↗](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

Methods

Add(InfluenceMap, InfluenceMap)

Adds two maps with the same size to each other and returns a new map containing the result

```
public InfluenceMap Add(InfluenceMap otherMap, InfluenceMap result = null)
```

Parameters

[otherMap InfluenceMap](#)

[result InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

Add(InfluenceMap, float, InfluenceMap)

Adds two maps with the same size to each other and returns a new map containing the result

```
public InfluenceMap Add(InfluenceMap otherMap, float strength, InfluenceMap result = null)
```

Parameters

otherMap [InfluenceMap](#)

strength [float](#)

The other map will be multiplied by this value to increase/decrease its effect

result [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

Add(float, InfluenceMap)

```
public InfluenceMap Add(float value, InfluenceMap result = null)
```

Parameters

value [float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

result [InfluenceMap](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

Returns

[InfluenceMap](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

AddAndClamp(float, float, float, InfluenceMap)

```
public InfluenceMap AddAndClamp(float value, float min, float max, InfluenceMap result  
= null)
```

Parameters

value [float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

min [float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

max [float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

result [InfluenceMap](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

Returns

[InfluenceMap](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

AddInfluence(int, int, InfluenceMapTemplate, float)

Adds an influence stamp to the map

```
public void AddInfluence(int centerX, int centerY, InfluenceMapTemplate template, float magnitude = 1)
```

Parameters

centerX [int](#)

centerY [int](#)

template [InfluenceMapTemplate](#)

The template which should be propagated

magnitude [float](#)

Remarks

Templates are scriptable objects and can be created in the project window in the editor or at runtime depending on your needs

AddInfluenceWithLineOfSight(int, int, InfluenceMapTemplate, InfluenceMap, float)

Adds an influence stamp to the map

```
public void AddInfluenceWithLineOfSight(int centerX, int centerY, InfluenceMapTemplate template, InfluenceMap obstaclesMap, float magnitude = 1)
```

Parameters

centerX [int](#)

centerY [int](#)

template [InfluenceMapTemplate](#)

The template which should be propagated

`obstaclesMap` [InfluenceMap](#)

The map which contains obstacles and we check line of sight in

`magnitude` [float](#)

Remarks

Templates are scriptable objects and can be created in the project window in the editor or at runtime depending on your needs

AddInverse([InfluenceMap](#), [InfluenceMap](#))

Adds the inverse of a map to the current map and returns a new map containing the result

```
public InfluenceMap AddInverse(InfluenceMap otherMap, InfluenceMap result = null)
```

Parameters

`otherMap` [InfluenceMap](#)

`result` [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

AddInverse([InfluenceMap](#), [float](#), [InfluenceMap](#))

Adds the inverse of a map to the current map and returns a new map containing the result

```
public InfluenceMap AddInverse(InfluenceMap otherMap, float strength, InfluenceMap result = null)
```

Parameters

`otherMap` [InfluenceMap](#)

strength [float](#)

The other map will be multiplied by this value to increase/decrease its effect

result [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

AddRectangularInfluence(int, int, int, int, float)

Adds a rectangular template of a constant value to the map

```
public void AddRectangularInfluence(int startX, int startY, int width, int height, float  
value = 1)
```

Parameters

startX [int](#)

startY [int](#)

width [int](#)

height [int](#)

value [float](#)

Remarks

This can have negative width and height to move left/down from the starting point

AddValueToCell(int, int, float)

Adds a value to a cell

```
public void AddValueToCell(int x, int y, float value)
```

Parameters

x [int](#)

y [int](#)

value [float](#)

CheckMapForLineOfSight(int, int, int, int)

Checks to see if there is any line of sight between two points. Returns true if there is any and false otherwise.

```
public bool CheckMapForLineOfSight(int x1, int y1, int x2, int y2)
```

Parameters

x1 [int](#)

y1 [int](#)

x2 [int](#)

y2 [int](#)

Returns

[bool](#)

GetCellValue(int, int)

Get a cell's value

```
public float GetCellValue(int x, int y)
```

Parameters

x [int](#)

y [int](#)

Returns

[float](#)

GetCellsArray()

Returns the cells of the map, keep in mind that this is unsafe if you modify the array yourself You need this if you want to do operations on the map's array

```
public float[,] GetCellsArray()
```

Returns

[float](#)[]

A reference to the array that the map uses

Remarks

This method does not return a copy so be careful to not modify the array and just use it to calculate new maps

Invert(InfluenceMap)

Inverts the map values so 0.1 will become 0.9 and 0.6 will become 0.4

```
public InfluenceMap Invert(InfluenceMap result = null)
```

Parameters

[result](#) [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

MapToWorldPosition(int, int)

Converts a position from the map coordinates to unity's world space coordinates The y value of the returned vector is always 0 and x and z are set to the converted values

```
public Vector3 MapToWorldPosition(int x, int y)
```

Parameters

x [int](#)

y [int](#)

Returns

Vector3

MapToWorldPosition(int, int, Terrain)

Converts a position from the map coordinates to unity's world space coordinates The y value of the returned vector is always 0 and x and z are set to the converted values

```
public Vector3 MapToWorldPosition(int x, int y, Terrain terrain)
```

Parameters

x [int](#)

x position on the influence map

y [int](#)

y position on the influence map

terrain [Terrain](#)

The terrain to get the height from

Returns

Vector3

MapToWorldPosition2d(int, int)

Converts a position from the map coordinates to unity's world space coordinates in 2d plane. The z value of the returned vector is always 0 and x and y are set to the converted values

```
public Vector3 MapToWorldPosition2d(int x, int y)
```

Parameters

x [int](#)

y [int](#)

Returns

Vector3

Multiply(InfluenceMap, InfluenceMap)

Multiplies a map to this map and returns a new map containing the result The maps should be the same size

```
public InfluenceMap Multiply(InfluenceMap otherMap, InfluenceMap result = null)
```

Parameters

otherMap [InfluenceMap](#)

result [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

Exceptions

[ArgumentException](#)

Multiply(InfluenceMap, float, InfluenceMap)

Multiplies a map to this map and returns a new map containing the result The maps should be the same size

```
public InfluenceMap Multiply(InfluenceMap otherMap, float strength, InfluenceMap result = null)
```

Parameters

otherMap [InfluenceMap](#)

strength [float](#)

The other map will be multiplied by this value to increase/decrease its effect

result [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

Exceptions

[ArgumentException](#)

Multiply(float, InfluenceMap)

Multiplies a map with such a value

```
public InfluenceMap Multiply(float value, InfluenceMap result = null)
```

Parameters

value [float](#)

The value that all map cells will be multiplied with

result [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

MultiplyAndClamp(float, float, float, InfluenceMap)

```
public InfluenceMap MultiplyAndClamp(float value, float min, float max, InfluenceMap result  
= null)
```

Parameters

value [float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

min [float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

max [float](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

result [InfluenceMap](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

Returns

[InfluenceMap](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

Normalize(InfluenceMap)

Normalizes the values in a map to values between 0 and 1

```
public InfluenceMap Normalize(InfluenceMap result = null)
```

Parameters

result [InfluenceMap](#)

Returns

[InfluenceMap](#)

PropagateInfluence(int, int, int, AnimationCurve, float)

Propagates an influence stamp based on a curve, a radius and a magnitude

```
public void PropagateInfluence(int centerX, int centerY, int radius, AnimationCurve  
influenceCurve, float magnitude = 1)
```

Parameters

centerX [int](#)

centerY [int](#)

radius [int](#)

influenceCurve AnimationCurve

magnitude [float](#)

Remarks

This is slower than AddInfluence and is used by the template to bake its values but you can use it if the number of different templates you need is high enough that the amount of memory is used is not acceptable

PropagateInfluenceWithLineOfSight(int, int, int, AnimationCurve, InfluenceMap, float)

Propagates an influence stamp based on a curve, a radius and a magnitude

```
public void PropagateInfluenceWithLineOfSight(int centerX, int centerY, int radius,  
AnimationCurve influenceCurve, InfluenceMap obstaclesMap, float magnitude = 1)
```

Parameters

centerX [int](#)

centerY [int](#)

radius [int](#)

influenceCurve [AnimationCurve](#)

obstaclesMap [InfluenceMap](#)

This is the main influence map class of the system which contains the actual map logic and can be created in C# using its constructor and does not depend on any component/MonoBehaviour

magnitude [float](#)

Remarks

This is slower than AddInfluence and is used by the template to bake its values but you can use it if the number of different templates you need is high enough that the amount of memory is used is not acceptable

SearchForValueWithRandomStartingPoint(float, SearchCondition, Vector2Int, int, out Vector2Int)

Searches for a value in an area specified by a center and a radius and starts at a random point in the area to not return the same value as the result of the flood fill algorithm

```
public bool SearchForValueWithRandomStartingPoint(float searchValue, SearchCondition condition, Vector2Int centerCell, int radius, out Vector2Int result)
```

Parameters

searchValue [float](#)

condition [SearchCondition](#)

centerCell Vector2Int

radius [int](#)

result Vector2Int

Returns

[bool](#)

SearchForValueWithRandomStartingPoint(float, SearchCondition, out Vector2Int)

Searches for a value in an area specified by a center and a radius and starts at a random point in the area to not return the same value as the result of the flood fill algorithm

```
public bool SearchForValueWithRandomStartingPoint(float searchValue, SearchCondition condition, out Vector2Int result)
```

Parameters

searchValue [float](#)

condition [SearchCondition](#)

result Vector2Int

Returns

[bool](#)

SetCellValue(int, int, float)

Sets a cell's value to the specified value

```
public void SetCellValue(int x, int y, float value)
```

Parameters

x [int](#)

y [int](#)

value [float](#)

SetCellsArray(float[,])

Sets the cells of the map to the array you created

```
public void SetCellsArray(float[,] cells)
```

Parameters

cells [float](#)[]

Subtract(InfluenceMap, InfluenceMap)

Subtract a map from this map and returns a new map containing the result The maps should be the same size

```
public InfluenceMap Subtract(InfluenceMap otherMap, InfluenceMap result = null)
```

Parameters

otherMap [InfluenceMap](#)

result [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

Exceptions

[ArgumentException](#)

Subtract(InfluenceMap, float, InfluenceMap)

Subtract a map from this map and returns a new map containing the result The maps should be the same size

```
public InfluenceMap Subtract(InfluenceMap otherMap, float strength, InfluenceMap result  
= null)
```

Parameters

otherMap [InfluenceMap](#)

strength [float](#)

The other map will be multiplied by this value to increase/decrease its effect

result [InfluenceMap](#)

If you have a map already allocated for the result, you can supply it here

Returns

[InfluenceMap](#)

Exceptions

[ArgumentException](#)

UpdateMinMaxValue()

```
public void UpdateMinMaxValue()
```

WorldToMapPosition(Vector3)

Converts a position from unity world space to map coordinates. The y component of the vector has no effect and x and z are converted to 2d map coordinates

```
public Vector2Int WorldToMapPosition(Vector3 position)
```

Parameters

position Vector3

Returns

Vector2Int

Remarks

Keep in mind that the position can be outside the map depending on where the position is and the map size

WorldToMapPosition2d(Vector3)

Converts a position from unity world space to map coordinates. It uses the 2D x,y plane instead of x,z
The z component of the vector has no effect and x and y are converted to 2d map coordinates

```
public Vector2Int WorldToMapPosition2d(Vector3 position)
```

Parameters

position Vector3

Returns

Vector2Int

Remarks

Keep in mind that the position can be outside the map depending on where the position is and the map size

Class InfluenceMapCollection

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

```
public class InfluenceMapCollection : Singleton<InfluenceMapCollection>
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ←
[Singleton](#)<[InfluenceMapCollection](#)> ← InfluenceMapCollection

Inherited Members

[Singleton<InfluenceMapCollection>.Instance](#) , [Singleton<InfluenceMapCollection>.Awake\(\)](#) ,
[Singleton<InfluenceMapCollection>.OnDestroy\(\)](#) , MonoBehaviour.IsInvoking() ,
MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,

[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

mapItems

```
public Dictionary<string, InfluenceMapComponentBase> mapItems
```

Field Value

[Dictionary](#)<[string](#), [InfluenceMapComponentBase](#)>

Methods

GetMap(string)

Gets a map by its name

```
public InfluenceMapComponentBase GetMap(string name)
```

Parameters

name [string](#)

Returns

[InfluenceMapComponentBase](#)

Register(string, InfluenceMapComponentBase)

Registers a map in the collection. This is called automatically by the influence map component

```
public void Register(string mapName, InfluenceMapComponentBase influenceMap)
```

Parameters

mapName [string](#)

influenceMap [InfluenceMapComponentBase](#)

Exceptions

[ArgumentException](#)

Unregister(string)

Registers a map in the collection. This is called automatically by the influence map component

```
public void Unregister(string mapName)
```

Parameters

mapName [string](#)

Class InfluenceMapComponent

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

A helper component which makes it easier to use influence maps Attach this component to a GameObject to create an influence map at awake and use it as a reference for InfluencerAgent component. The map field is the actual influence map that you can use to search for values or directly change cell values

```
public class InfluenceMapComponent : InfluenceMapComponentBase
```

Inheritance

[Object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← [InfluenceMapComponentBase](#) ← InfluenceMapComponent

Inherited Members

[InfluenceMapComponentBase.is2dMap](#) , [InfluenceMapComponentBase.Map](#) ,
[InfluenceMapComponentBase.IsMapValid\(\)](#) ,
[InfluenceMapComponentBase.AddAndClampValue\(float, float, float\)](#) ,
[InfluenceMapComponentBase.MultiplyAndClampValue\(float, float, float\)](#) ,
[InfluenceMapComponentBase.AddInfluence\(int, int, InfluenceMapTemplate, float\)](#) ,
[InfluenceMapComponentBase.AddInfluenceWithLineOfSight\(int, int, InfluenceMapTemplate, InfluenceMapComponentBase, float\)](#) ,
[InfluenceMapComponentBase.AddRectangularInfluence\(int, int, int, int, float\)](#) ,
[InfluenceMapComponentBase.PropagateInfluenceWithLineOfSight\(int, int, int, AnimationCurve, InfluenceMapComponentBase, int\)](#) ,
[InfluenceMapComponentBase.PropagateInfluence\(int, int, int, AnimationCurve, int\)](#) ,
[InfluenceMapComponentBase.WorldToMapPosition\(Vector3\)](#) ,
[InfluenceMapComponentBase.SearchForValueWithRandomStartingPoint\(float, SearchCondition, Vector2Int, int, out Vector2Int\)](#) ,
[InfluenceMapComponentBase.GetCellValue\(int, int\)](#) ,
[InfluenceMapComponentBase.MapToWorldPosition\(int, int\)](#) , MonoBehaviour.IsInvoking()
MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,

MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout , MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled , [Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() , [Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) , [Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) , [Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) , Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) , [Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) , [Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) , Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) , [Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) , [Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() , [Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) , [Component.GetComponentsInParent<T>\(bool\)](#) , [Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() , [Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) , [Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() , [Component.CompareTag\(string\)](#) , [Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) , Component.SendMessageUpwards(string, object) , [Component.SendMessageUpwards\(string\)](#) , Component.SendMessageUpwards(string, SendMessageOptions) , [Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) , [Component.SendMessage\(string, object, SendMessageOptions\)](#) , [Component.SendMessage\(string, SendMessageOptions\)](#) , Component.BroadcastMessage(string, object, SendMessageOptions) , [Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) , [Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform , Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) , Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) , Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) , Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) , Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) , [Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) , [Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) , [Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) , [Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) , [Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) , Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) , Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,

[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

CreateOnAwake

Create the map on awake

```
public bool CreateOnAwake
```

Field Value

[bool](#)

A helper component which makes it easier to use influence maps Attach this component to a GameObject to create an influence map at awake and use it as a reference for InfluencerAgent component. The map field is the actual influence map that you can use to search for values or directly change cell values

cellSize

Cell size of the map

```
[Tooltip("Cell size of the map")]
public float cellSize
```

Field Value

[float](#)

A helper component which makes it easier to use influence maps Attach this component to a GameObject to create an influence map at awake and use it as a reference for InfluencerAgent component. The map field is the actual influence map that you can use to search for values or directly change cell values

height

Height of the map in cells

```
[Tooltip("Height of the map in cells")]
public int height
```

Field Value

[int](#)

A helper component which makes it easier to use influence maps Attach this component to a GameObject to create an influence map at awake and use it as a reference for InfluencerAgent component. The map field is the actual influence map that you can use to search for values or directly change cell values

mapName

//MapName

```
public string mapName
```

Field Value

[string](#)

A helper component which makes it easier to use influence maps Attach this component to a GameObject to create an influence map at awake and use it as a reference for InfluencerAgent component. The map field is the actual influence map that you can use to search for values or directly change cell values

width

Width of the map in cell count

```
[Tooltip("Width of the map in cell count")]
public int width
```

Field Value

int ↗

A helper component which makes it easier to use influence maps Attach this component to a GameObject to create an influence map at awake and use it as a reference for InfluencerAgent component. The map field is the actual influence map that you can use to search for values or directly change cell values

Methods

CreateOnStartup()

```
[Obsolete("This method is deprecated and will be removed. Use CreateMap() instead")]
public void CreateOnStartup()
```

Class InfluenceMapComponentBase

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

```
public class InfluenceMapComponentBase : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← InfluenceMapComponentBase

Derived

[InfluenceMapComponent](#), [InfluenceMapView](#)

Inherited Members

MonoBehaviour.IsInvoking(), MonoBehaviour.CancelInvoke(), [MonoBehaviour.Invoke\(string, float\)](#),
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#), [MonoBehaviour.CancelInvoke\(string\)](#),
[MonoBehaviour.IsInvoking\(string\)](#), [MonoBehaviour.StartCoroutine\(string\)](#),
[MonoBehaviour.StartCoroutine\(string, object\)](#), [MonoBehaviour.StartCoroutine\(IEnumerator\)](#),
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#), [MonoBehaviour.StopCoroutine\(IEnumerator\)](#),
MonoBehaviour.StopCoroutine(Coroutine), [MonoBehaviour.StopCoroutine\(string\)](#),
MonoBehaviour.StopAllCoroutines(), [MonoBehaviour.print\(object\)](#), MonoBehaviour.useGUILayout,
MonoBehaviour.runInEditMode, Behaviour.enabled, Behaviour.isActiveAndEnabled,
[Component.GetComponent\(Type\)](#), Component.GetComponent<T>(),
[Component.TryGetComponent\(Type, out Component\)](#), Component.TryGetComponent<T>(out T),
[Component.GetComponent\(string\)](#), [Component.GetComponentInChildren\(Type, bool\)](#),
[Component.GetComponentInChildren\(Type\)](#), [Component.GetComponentInChildren<T>\(bool\)](#),
Component.GetComponentInChildren<T>(), [Component.GetComponentsInChildren\(Type, bool\)](#),
[Component.GetComponentsInChildren\(Type\)](#), [Component.GetComponentsInChildren<T>\(bool\)](#),
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#),
Component.GetComponentsInChildren<T>(), [Component.GetComponentsInChildren<T>\(List<T>\)](#),
[Component.GetComponentInParent\(Type, bool\)](#), [Component.GetComponentInParent\(Type\)](#),
[Component.GetComponentInParent<T>\(bool\)](#), Component.GetComponentInParent<T>(),
[Component.GetComponentsInParent\(Type, bool\)](#), [Component.GetComponentsInParent\(Type\)](#),
[Component.GetComponentsInParent<T>\(bool\)](#),
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#), Component.GetComponentsInParent<T>(),
[Component.GetComponents\(Type\)](#), [Component.GetComponents\(Type, List<Component>\)](#),
[Component.GetComponents<T>\(List<T>\)](#), Component.GetComponents<T>(),
[Component.CompareTag\(string\)](#),
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#),

[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

is2dMap

Is this map used for a 2d game in the x,y plane

```
[Tooltip("Is this map used for a 2d game in the x,y plane")]
public bool is2dMap
```

Field Value

[bool](#)

Properties

Map

The actual influence map object which can be used for searching or setting values

```
public InfluenceMap Map { get; protected set; }
```

Property Value

[InfluenceMap](#)

Methods

AddAndClampValue(float, float, float)

```
public void AddAndClampValue(float value, float min, float max)
```

Parameters

value [float](#)

min [float](#)

max [float](#)

AddInfluence(int, int, InfluenceMapTemplate, float)

Adds the inverse of a map to the current map and returns a new map containing the result

```
public void AddInfluence(int x, int y, InfluenceMapTemplate myMapTemplate, float magnitude = 1)
```

Parameters

x [int](#)

y [int](#)

myMapTemplate [InfluenceMapTemplate](#)

magnitude [float](#)

AddInfluenceWithLineOfSight(int, int, InfluenceMapTemplate, InfluenceMapComponentBase, float)

Adds the inverse of a map to the current map and returns a new map containing the result

```
public void AddInfluenceWithLineOfSight(int x, int y, InfluenceMapTemplate myMapTemplate, InfluenceMapComponentBase obstaclesMap, float magnitude = 1)
```

Parameters

x [int](#)

y [int](#)

myMapTemplate [InfluenceMapTemplate](#)

obstaclesMap [InfluenceMapComponentBase](#)

The map to check line of sight in

magnitude [float](#)

AddRectangularInfluence(int, int, int, int, float)

Adds a rectangular template of a constant value to the map

```
public void AddRectangularInfluence(int x, int y, int width, int height, float value = 1)
```

Parameters

x [int](#)

y [int](#)

width [int](#)

height [int](#)

value [float](#)

GetCellValue(int, int)

Get a cell's value

```
public float GetCellValue(int x, int y)
```

Parameters

x [int](#)

y [int](#)

Returns

[float](#)

IsMapValid()

```
public bool IsMapValid()
```

Returns

[bool](#)

MapToWorldPosition(int, int)

Converts a position from the map coordinates to unity's world space coordinates

```
public Vector3 MapToWorldPosition(int x, int y)
```

Parameters

x [int](#)

y [int](#)

Returns

Vector3

MultiplyAndClampValue(float, float, float)

```
public void MultiplyAndClampValue(float value, float min, float max)
```

Parameters

value [float](#)

min [float](#)

max [float](#)

PropagateInfluence(int, int, int, AnimationCurve, int)

Propagates an influence stamp based on a curve, a radius and a magnitude

```
public void PropagateInfluence(int x, int y, int radius, AnimationCurve curve, int magnitude = 1)
```

Parameters

x [int](#)

y [int](#)

radius [int](#)

curve AnimationCurve

magnitude [int](#)

PropagateInfluenceWithLineOfSight(int, int, int, AnimationCurve, InfluenceMapComponentBase, int)

Propagates an influence stamp based on a curve, a radius and a magnitude

```
public void PropagateInfluenceWithLineOfSight(int x, int y, int radius, AnimationCurve  
curve, InfluenceMapComponentBase ObstaclesMap, int magnitude = 1)
```

Parameters

x [int](#)

y [int](#)

radius [int](#)

curve AnimationCurve

ObstaclesMap [InfluenceMapComponentBase](#)

magnitude [int](#)

SearchForValueWithRandomStartingPoint(float, SearchCondition, Vector2Int, int, out Vector2Int)

Searches for a value in an area specified by a center and a radius and starts at a random point in the area to not return the same value as the result of the flood fill algorithm

```
public bool SearchForValueWithRandomStartingPoint(float searchValue, SearchCondition  
searchCondition, Vector2Int position, int radius, out Vector2Int result)
```

Parameters

searchValue [float](#)

searchCondition [SearchCondition](#)

position Vector2Int

radius [int](#)

result Vector2Int

Returns

[bool](#)

WorldToMapPosition(Vector3)

Converts a position from unity world space to map coordinates. The y component of the vector has no effect and x and z are converted to 2d map coordinates

```
public Vector2Int WorldToMapPosition(Vector3 position)
```

Parameters

position Vector3

Returns

Vector2Int

Remarks

Keep in mind that the position can be outside the map depending on where the position is and the map size

Class InfluenceMapDecayer

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

Decays an influence map using the value you gave it and clamps the values to not go above/beyond specified values

```
public class InfluenceMapDecayer : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← InfluenceMapDecayer

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,

[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

InitialDelayInSeconds

Initial Delay In Seconds

```
[Tooltip("Initial Delay In Seconds")]
public float InitialDelayInSeconds
```

Field Value

[float](#) ↗

Decays an influence map using the value you gave it and clamps the values to not go above/beyond specified values

decayValue

The value used for decaying

```
public float decayValue
```

Field Value

[float](#) ↗

Decays an influence map using the value you gave it and clamps the values to not go above/beyond specified values

delayBetweenCalculations

Delay between calculations of the map

```
[Tooltip("Delay between calculations of the map")]
public float delayBetweenCalculations
```

Field Value

[float](#) ↗

Decays an influence map using the value you gave it and clamps the values to not go above/beyond specified values

maxValue

The maximum value each map cell should be in

```
[Tooltip("The maximum value each map cell should be in")]
public float maxValue
```

Field Value

[float](#)

Decays an influence map using the value you gave it and clamps the values to not go above/beyond specified values

minValue

The minimum value each map cell should be in

```
[Tooltip("The minimum value each map cell should be in")]
public float minValue
```

Field Value

[float](#)

Decays an influence map using the value you gave it and clamps the values to not go above/beyond specified values

operation

The operation to do

```
public InfluenceMapDecayer.OperationMode operation
```

Field Value

[InfluenceMapDecayer.OperationMode](#)

Decays an influence map using the value you gave it and clamps the values to not go above/beyond specified values

Enum InfluenceMapDecayer.OperationMode

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

The operations done on the map

```
public enum InfluenceMapDecayer.OperationMode
```

Fields

Add = 0

Add the decayValue to the map

Multiply = 1

Multiply the decay value by the map

Class InfluenceMapTemplate

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

This asset type is used as a template for the stamp shape of agents which influence the amp Usually different agents based on their behavior need both different sizes and fall off curve shapes

```
[CreateAssetMenu(menuName = "NoOpArmy/Wise Feline/InfluenceMapTemplate")]
public class InfluenceMapTemplate : ScriptableObject
```

Inheritance

[Object](#) ← Object ← ScriptableObject ← InfluenceMapTemplate

Inherited Members

ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectsInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectType\(Type, bool\)](#) ,
[Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,

[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

Curve

The fall off curve of the influence

```
[Tooltip("The fall off curve of the influence")]
public AnimationCurve Curve
```

Field Value

AnimationCurve

This asset type is used as a template for the stamp shape of agents which influence the amp Usually different agents based on their behavior need both different sizes and fall off curve shapes

Radius

Radius of the agent in cell count

```
[Tooltip("Radius of the agent in count")]
public int Radius
```

Field Value

[int](#)

This asset type is used as a template for the stamp shape of agents which influence the amp Usually different agents based on their behavior need both different sizes and fall off curve shapes

baked

The baked version of the animation curve

```
public NativeArray<float> baked
```

Field Value

NativeArray<[float](#)>

This asset type is used as a template for the stamp shape of agents which influence the amp Usually different agents based on their behavior need both different sizes and fall off curve shapes

Properties

Map

```
public InfluenceMap Map { get; }
```

Property Value

[InfluenceMap](#)

This asset type is used as a template for the stamp shape of agents which influence the amp Usually different agents based on their behavior need both different sizes and fall off curve shapes

MapSize

```
public int MapSize { get; }
```

Property Value

[int](#)

This asset type is used as a template for the stamp shape of agents which influence the amp Usually different agents based on their behavior need both different sizes and fall off curve shapes

Methods

Init()

```
public void Init()
```

OnEnable()

```
public void OnEnable()
```

Class InfluenceMapView

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

```
public class InfluenceMapView : InfluenceMapComponentBase
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← [InfluenceMapComponentBase](#) ← InfluenceMapView

Inherited Members

[InfluenceMapComponentBase.is2dMap](#) , [InfluenceMapComponentBase.Map](#) ,
[InfluenceMapComponentBase.IsMatchValid\(\)](#) ,
[InfluenceMapComponentBase.AddAndClampValue\(float, float, float\)](#) ,
[InfluenceMapComponentBase.MultiplyAndClampValue\(float, float, float\)](#) ,
[InfluenceMapComponentBase.AddInfluence\(int, int, InfluenceMapTemplate, float\)](#) ,
[InfluenceMapComponentBase.AddInfluenceWithLineOfSight\(int, int, InfluenceMapTemplate, InfluenceMapComponentBase, float\)](#) ,
[InfluenceMapComponentBase.AddRectangularInfluence\(int, int, int, int, float\)](#) ,
[InfluenceMapComponentBase.PropagateInfluenceWithLineOfSight\(int, int, int, AnimationCurve, InfluenceMapComponentBase, int\)](#) ,
[InfluenceMapComponentBase.PropagateInfluence\(int, int, int, AnimationCurve, int\)](#) ,
[InfluenceMapComponentBase.WorldToMapPosition\(Vector3\)](#) ,
[InfluenceMapComponentBase.SearchForValueWithRandomStartingPoint\(float, SearchCondition, Vector2Int, int, out Vector2Int\)](#) ,
[InfluenceMapComponentBase.GetCellValue\(int, int\)](#) ,
[InfluenceMapComponentBase.MapToWorldPosition\(int, int\)](#) , MonoBehaviour.IsInvoking() ,
MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,

[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,

```
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode),  
Object.FindObjectOfType<T>() , Object.FindObjectOfType<T>\(bool\) ,  
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,  
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,  
Object.FindAnyObjectByType<T>(FindObjectsInactive) , Object.FindObjectsOfTypeAll\(Type\) ,  
Object.FindObjectType\(Type\) , Object.FindFirstObjectType\(Type\) ,  
Object.FindAnyObjectType\(Type\) , Object.FindObjectType\(Type, bool\) ,  
Object.FindFirstObjectType\(Type, FindObjectsInactive\) ,  
Object.FindAnyObjectType\(Type, FindObjectsInactive\) , Object.ToString() , Object.name ,  
Object.hideFlags , object.Equals\(object, object\) , object.GetType\(\) , object.MemberwiseClone\(\) ,  
object.ReferenceEquals\(object, object\)
```

Fields

InitialDelayInSeconds

Initial Delay In Seconds

```
[Tooltip("Initial Delay In Seconds")]  
public float InitialDelayInSeconds
```

Field Value

[float](#)

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

cellSize

Cell size of the map

```
[Tooltip("Cell size of the map")]  
public float cellSize
```

Field Value

[float](#)

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

delayBetweenCalculations

Delay between calculations of the map

```
public float delayBetweenCalculations
```

Field Value

[float](#)

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

firstMapName

The first map to start operations from

```
public string firstMapName
```

Field Value

[string](#)

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

height

Height of the map in cells

```
[Tooltip("Height of the map in cells")]
public int height
```

Field Value

[int](#)

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

mapName

//MapName

```
public string mapName
```

Field Value

[string](#)

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

operations

The operations which will be done on the map consecutively

```
public List<InfluenceMapView.ViewOperation> operations
```

Field Value

[List](#) <[InfluenceMapView](#).[ViewOperation](#)>

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

updatePositionAutomatically

If the object moves, you should check this and set the delay between calculations based on speed and the accuracy you require

```
[Tooltip("If the object moves, you should check this and set the delay between calculations  
based on speed and the accuracy you require")]
```

```
public bool updatePositionAutomatically
```

Field Value

[bool](#) ↗

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

width

Width of the map in cell count

```
[Tooltip("Width of the map in cell count")]
public int width
```

Field Value

[int](#) ↗

This class allows you to calculate a map from multiple others. You could previously do this without a view using code but this makes it easier to make maps from other maps without writing a line of code

Class InfluenceMapView.ViewOperation

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

```
[Serializable]
public class InfluenceMapView.ViewOperation
```

Inheritance

[object](#) ← InfluenceMapView.ViewOperation

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

additionalArgument

The additional argument needed by the operation like the strength of the second map or the value to multiply with in case of MultiplyByValue

```
[Tooltip("The additional argument needed by the operation like the strength of the second
map or the value to multiply with in case of MultiplyByValue")]
public float additionalArgument
```

Field Value

[float](#)

mapName

```
public string mapName
```

Field Value

[string](#) ↗

operation

```
public InfluenceMapView.ViewOperation.Operation operation
```

Field Value

[InfluenceMapView.ViewOperation.Operation](#)

Enum InfluenceMapView.ViewOperation.Operation

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

```
public enum InfluenceMapView.ViewOperation.Operation
```

Fields

Add = 0

AddInverse = 1

Inverse = 6

Multiply = 3

MultiplyByValue = 4

Normalize = 5

Subtract = 2

Class InfluencerAgent

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

This component can be attached to an agent so it influences the map

```
public class InfluencerAgent : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← InfluencerAgent

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,

[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

behavior

The behavior of the map. You can use a single template, a list of them with the ability to change the index of the current one or directly use a radius and a curve

```
[Tooltip("The behavior of the map. You can use a single template, a list of them with the  
ability to change the index of the current one or directly use a radius and a curve")]  
public InfluencerAgent.Behavior behavior
```

Field Value

[InfluencerAgent.Behavior](#)

This component can be attached to an agent so it influences the map

directCurve

The curve to use for the direct value behavior

```
public AnimationCurve directCurve
```

Field Value

AnimationCurve

This component can be attached to an agent so it influences the map

directRadius

The radius to use for the direct value behavior

```
public int directRadius
```

Field Value

[int](#)

This component can be attached to an agent so it influences the map

mapComponent

The map that this agent influences

```
[Header("The map to influence")]
[Tooltip("The map that this agent influences")]
public InfluenceMapComponent mapComponent
```

Field Value

[InfluenceMapComponent](#)

This component can be attached to an agent so it influences the map

mapIndex

The index in the list to use when TemplateList behavior is selected. It should be between 0 and list's count - 1

```
public int mapIndex
```

Field Value

[int](#)

This component can be attached to an agent so it influences the map

mapName

The name of the map which should be read from the InfluenceMapCollection This will only be used if the mapComponent is null

```
[Tooltip("The name of the map which should be read from the InfluenceMapCollection. This
will only be used if the mapComponent is null.")]
public string mapName
```

Field Value

string

This component can be attached to an agent so it influences the map

mapTemplatesList

The list of templates when the TemplateList behavior is selected

```
public List<InfluenceMapTemplate> mapTemplatesList
```

Field Value

List<InfluenceMapTemplate>

This component can be attached to an agent so it influences the map

myMapTemplate

The template that the agent uses for its influence stamp shape

```
[Tooltip("The template that the agent uses for its influence stamp shape")]
public InfluenceMapTemplate myMapTemplate
```

Field Value

InfluenceMapTemplate

This component can be attached to an agent so it influences the map

obstaclesMapComponent

The map that this agent considers obstacles for line of sight

```
[Header("The obstacles map for line of sight")]
[Tooltip("The map that this agent considers obstacles for line of sight")]
public InfluenceMapComponent obstaclesMapComponent
```

Field Value

[InfluenceMapComponent](#)

This component can be attached to an agent so it influences the map

obstaclesMapName

The name of the obstacles map which should be read from the InfluenceMapCollection This will only be used if the obstacleMapComponent is null

```
[Tooltip("The name of the obstacles map which should be read from the  
InfluenceMapCollection. This will only be used if the mapComponent is null.")]  
public string obstaclesMapName
```

Field Value

[string](#)

This component can be attached to an agent so it influences the map

updateInterval

The update interval of the agent. 0.5 means the agent updates its position on the map two times a second.

```
[Tooltip("The update interval of the agent. 0.5 means the agent updates its position on the  
map two times a second.")]  
public float updateInterval
```

Field Value

[float](#)

This component can be attached to an agent so it influences the map

updatePositionAutomatically

Checking this will cause the object influence to be updated in the influence map automatically. If the object moves, you should check this and set the interval based on its speed and the accuracy you require

```
[Tooltip("If the object moves, you should check this and set the interval based on its speed  
and the accuracy you require")]  
public bool updatePositionAutomatically
```

Field Value

[bool](#) 

This component can be attached to an agent so it influences the map

useLineOfSightWhenCalculatingInfluence

Set this to true if your agents need to consider obstacles and their line of sight when adding their influence. Usually only range units in shooter games might need this.

```
[Tooltip("Set this to true if your agents need to consider obstacles and their line of sight  
when adding their influence. Usually only range units in shooter games might need this.")]  
public bool useLineOfSightWhenCalculatingInfluence
```

Field Value

[bool](#) 

This component can be attached to an agent so it influences the map

Properties

AgentMap

```
public InfluenceMapComponentBase AgentMap { get; set; }
```

Property Value

[InfluenceMapComponentBase](#)

This component can be attached to an agent so it influences the map

ObstaclesMap

```
public InfluenceMapComponentBase ObstaclesMap { get; set; }
```

Property Value

[InfluenceMapComponentBase](#)

This component can be attached to an agent so it influences the map

Methods

SetMap(InfluenceMapComponent)

Use this to set the map later on if it cannot be set automatically at Start()

```
public void SetMap(InfluenceMapComponent map)
```

Parameters

map [InfluenceMapComponent](#)

Exceptions

[InvalidOperationException](#)

SetObstaclesMap(InfluenceMapComponent)

```
public void SetObstaclesMap(InfluenceMapComponent map)
```

Parameters

map [InfluenceMapComponent](#)

This component can be attached to an agent so it influences the map

Enum InfluencerAgent.Behavior

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

The way that this map will add its influence

```
public enum InfluencerAgent.Behavior
```

Fields

DirectValue = 2

Uses a curve and radius directly This is much slower than the two other approaches since it needs to evaluate the curves but templates evaluate their curves only once at initialization Use this only if using templates is cost prohibitive in terms of memory.

SingleTemplate = 0

Uses a single template defined in myMapTemplate

TemplateList = 1

Uses a list of tempates defined in templatesList and mapIndex

Class RectangularInfluencerAgent

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

This component can be attached to an agent so it influences the map

```
public class RectangularInfluencerAgent : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← RectangularInfluencerAgent

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
[Component.GetComponentInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) ,
[Component.GetComponentInParent<T>\(bool, List<T>\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,

[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

TemplateHeight

Height of the rectangle

```
[Tooltip("Height of the rectangle")]
public int TemplateHeight
```

Field Value

[int](#)

This component can be attached to an agent so it influences the map

TemplateWidth

Width of the rectangle

```
[Tooltip("Width of the rectangle")]
public int TemplateWidth
```

Field Value

[int](#)

This component can be attached to an agent so it influences the map

Value

The value to add for this object.

```
public float Value
```

Field Value

[float](#)

This component can be attached to an agent so it influences the map

agentPositionOffset

The offset applied to agent position to calculate starting x and y of the rectangle which then we move to right and up to fill.

```
[Tooltip("The offset applied to agent position to calculate starting x and y of the  
rectangle which then we move to right and up to fill.")]  
public Vector2Int agentPositionOffset
```

Field Value

Vector2Int

This component can be attached to an agent so it influences the map

gizmoColor

```
public Color gizmoColor
```

Field Value

Color

This component can be attached to an agent so it influences the map

mapComponent

The map that this agent influences

```
[Tooltip("The map that this agent influences")]  
public InfluenceMapComponent mapComponent
```

Field Value

[InfluenceMapComponent](#)

This component can be attached to an agent so it influences the map

mapName

The name of the map which should be read from the InfluenceMapCollection This will only be used if the mapComponent is null

```
[Tooltip("This will only be used if the mapComponent is null.")]
public string mapName
```

Field Value

[string](#)

This component can be attached to an agent so it influences the map

shouldDrawGizmos

```
public bool shouldDrawGizmos
```

Field Value

[bool](#)

This component can be attached to an agent so it influences the map

updateInterval

The update interval of the agent. 0.5 means the agent updates its position on the map two times a second.

```
[Tooltip("The update interval of the agent. 0.5 means the agent updates its position on the
map two times a second.")]
public float updateInterval
```

Field Value

[float](#)

This component can be attached to an agent so it influences the map

updatePositionAutomatically

Checking this will cause the object influence to be updated in the influence map automatically. If the object moves, you should check this and set the interval based on its speed and the accuracy you require

```
[Tooltip("If the object moves, you should check this and set the interval based on its speed  
and the accuracy you require")]  
public bool updatePositionAutomatically
```

Field Value

bool ↗

This component can be attached to an agent so it influences the map

Properties

AgentMap

```
public InfluenceMapComponentBase AgentMap { get; set; }
```

Property Value

InfluenceMapComponentBase

This component can be attached to an agent so it influences the map

Methods

SetMap(InfluenceMapComponent)

Use this to set the map later on if it cannot be set automatically at Start()

```
public void SetMap(InfluenceMapComponent map)
```

Parameters

map InfluenceMapComponent

Exceptions

[InvalidOperationException](#) ↗

Enum SearchCondition

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps](#)

Assembly: APIRef.dll

The stop condition for search

```
public enum SearchCondition : byte
```

Fields

Equal = 0

Greater = 1

GreaterOrEqual = 4

Less = 2

LessOrEqual = 5

NotEqual = 3

Namespace NoOpArmy.WiseFeline.InfluenceMaps.Sample

Classes

[FleeFromEnemies](#)

This component uses the influence map to flee from enemies and always tries to put the player in a position with less enemies

[MoveToRandomPosition](#)

This component just moves an object to a random position so it can be used as a test enemy

Class FleeFromEnemies

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps.Sample](#)

Assembly: APIRef.dll

This component uses the influence map to flee from enemies and always tries to put the player in a position with less enemies

```
public class FleeFromEnemies : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← FleeFromEnemies

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,

[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

EnemiesMap

```
public InfluenceMapComponent EnemiesMap
```

Field Value

[InfluenceMapComponent](#)

This component uses the influence map to flee from enemies and always tries to put the player in a position with less enemies

flee

```
public bool flee
```

Field Value

[bool](#)

This component uses the influence map to flee from enemies and always tries to put the player in a position with less enemies

speed

```
public float speed
```

Field Value

[float](#)

This component uses the influence map to flee from enemies and always tries to put the player in a position with less enemies

Methods

Start()

```
public IEnumerator Start()
```

Returns

[IEnumerator](#)

This component uses the influence map to flee from enemies and always tries to put the player in a position with less enemies

Class MoveToRandomPosition

Namespace: [NoOpArmy.WiseFeline.InfluenceMaps.Sample](#)

Assembly: APIRef.dll

This component just moves an object to a random position so it can be used as a test enemy

```
public class MoveToRandomPosition : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← MoveToRandomPosition

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
[Component.GetComponentInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) ,
[Component.GetComponentInParent<T>\(bool, List<T>\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,

[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

area

```
public Rect area
```

Field Value

Rect

This component just moves an object to a random position so it can be used as a test enemy

Namespace NoOpArmy.WiseFeline. Remembrance

Classes

[BrainMemoryRecorder](#)

[CircularMemoryContainer<T>](#)

[Memory](#)

This component allows an agent or any other object to remember things which happen and query about them

[MemoryContainer<T>](#)

[MemoryData](#)

This data structure defines a memory entry

[SimpleMemory](#)

Class BrainMemoryRecorder

Namespace: [NoOpArmy.WiseFeline.Remembrance](#)

Assembly: APIRef.dll

```
public class BrainMemoryRecorder : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← BrainMemoryRecorder

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,

[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Class CircularMemoryContainer<T>

Namespace: [NoOpArmy.WiseFeline.Remembrance](#)

Assembly: APIRef.dll

```
[Serializable]
public class CircularMemoryContainer<T> where T : MemoryData
```

Type Parameters

T

Inheritance

[object](#) ← CircularMemoryContainer<T>

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

CircularMemoryContainer(int)

```
public CircularMemoryContainer(int maxCount)
```

Parameters

maxCount [int](#)

Fields

memories

```
public Queue<T> memories
```

Field Value

[Queue](#) <T>

Methods

AddMemory(T)

```
public T AddMemory(T memory)
```

Parameters

memory T

Returns

T

GetAllMemories()

```
public Queue<T> GetAllMemories()
```

Returns

[Queue](#) <T>

GetClosestMemoryToTime(Predicate<T>, float, bool)

```
public T GetClosestMemoryToTime(Predicate<T> pred, float time, bool useEndTime = true)
```

Parameters

pred [Predicate](#) <T>

time [float](#)

`useEndTime` [bool ↗](#)

Returns

T

GetClosestMemoryToTime(float, bool)

`public T GetClosestMemoryToTime(float time, bool useEndTime = true)`

Parameters

`time` [float ↗](#)

`useEndTime` [bool ↗](#)

Returns

T

GetClosestMemoryToTime(string, float, bool)

`public T GetClosestMemoryToTime(string actionId, float time, bool useEndTime = true)`

Parameters

`actionId` [string ↗](#)

`time` [float ↗](#)

`useEndTime` [bool ↗](#)

Returns

T

GetClosestMemoryToTime(string, Object, float, bool)

```
public T GetClosestMemoryToTime(string actionId, Object target, float time, bool useEndTime = true)
```

Parameters

actionId [string](#)

target [Object](#)

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(Type, float, bool)

```
public T GetClosestMemoryToTime(Type type, float time, bool useEndTime = true)
```

Parameters

type [Type](#)

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(Type, Object, float, bool)

```
public T GetClosestMemoryToTime(Type type, Object target, float time, bool useEndTime = true)
```

Parameters

type [Type](#)

target Object

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(Type, Object, Object, float, bool)

```
public T GetClosestMemoryToTime(Type type, Object initiator, Object target, float time, bool  
useEndTime = true)
```

Parameters

type [Type](#)

initiator Object

target Object

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(Object, float, bool)

```
public T GetClosestMemoryToTime(Object initiator, float time, bool useEndTime = true)
```

Parameters

initiator Object

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(Object, Object, float, bool)

```
public T GetClosestMemoryToTime(Object initiator, Object target, float time, bool useEndTime  
= true)
```

Parameters

initiator Object

target Object

time [float](#)

useEndTime [bool](#)

Returns

T

Class Memory

Namespace: [NoOpArmy.WiseFeline.Remembrance](#)

Assembly: APIRef.dll

This component allows an agent or any other object to remember things which happen and query about them

```
public class Memory : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← Memory

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,

[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

everLastingMemories

```
public MemoryContainer<MemoryData> everLastingMemories
```

Field Value

[MemoryContainer<MemoryData>](#)

This component allows an agent or any other object to remember things which happen and query about them

everLastingMemoriesCount

```
public int everLastingMemoriesCount
```

Field Value

[int↗](#)

This component allows an agent or any other object to remember things which happen and query about them

memories

```
public MemoryContainer<MemoryData> memories
```

Field Value

[MemoryContainer<MemoryData>](#)

This component allows an agent or any other object to remember things which happen and query about them

normalMemoriesCount

```
public int normalMemoriesCount
```

Field Value

[int ↗](#)

This component allows an agent or any other object to remember things which happen and query about them

Methods

AddMemory(MemoryData, bool)

Adds a new memory

```
public void AddMemory(MemoryData memory, bool isEverLasting = false)
```

Parameters

memory [MemoryData](#)

isEverLasting [bool ↗](#)

Awake()

```
protected void Awake()
```

CreateAndGetListOfAllMemories()

Gets all memories, both the ones which last for ever and those which can be forgotten/removed due to capacity being reached. This allocates a new list per call.

```
public List<MemoryData> CreateAndGetListOfAllMemories()
```

Returns

[List ↗ <MemoryData>](#)

FindMostRecentMemoryEndTime(string)

```
public float FindMostRecentMemoryEndTime(string guid)
```

Parameters

guid [string](#)

This component allows an agent or any other object to remember things which happen and query about them

Returns

[float](#)

This component allows an agent or any other object to remember things which happen and query about them

FindMostRecentMemoryEndTime(string, Object)

```
public float FindMostRecentMemoryEndTime(string guid, Object target)
```

Parameters

guid [string](#)

This component allows an agent or any other object to remember things which happen and query about them

target [Object](#)

This component allows an agent or any other object to remember things which happen and query about them

Returns

[float](#)

This component allows an agent or any other object to remember things which happen and query about them

FindMostRecentMemoryEndTime(Type)

```
public float FindMostRecentMemoryEndTime(Type type)
```

Parameters

type [Type](#)

This component allows an agent or any other object to remember things which happen and query about them

Returns

[float](#)

This component allows an agent or any other object to remember things which happen and query about them

GetAllEverLastingMemories()

Gets all memories which will never be forgotten and removed

```
public List<MemoryData> GetAllEverLastingMemories()
```

Returns

[List](#) <[MemoryData](#)>

GetAllNormalMemories()

Gets all memories which can be forgotten

```
public List<MemoryData> GetAllNormalMemories()
```

Returns

[List](#) <[MemoryData](#)>

GetClosestMemoryToTime(Predicate<MemoryData>, float)

Gets the closest memory which matches a condition

```
public MemoryData GetClosestMemoryToTime(Predicate<MemoryData> pred, float time)
```

Parameters

pred [Predicate<MemoryData>](#)

time [float](#)

Returns

[MemoryData](#)

Remarks

Pass Time.time to getting the closest to now and pass 0 to get the earliest

GetClosestMemoryToTime(float)

Gets the closest memory to this time.

```
public MemoryData GetClosestMemoryToTime(float time)
```

Parameters

time [float](#)

Returns

[MemoryData](#)

Remarks

Pass Time.time to getting the closest to now and pass 0 to get the earliest

GetClosestMemoryToTime(string, float)

Get the closest memory with the specified criteria

```
public MemoryData GetClosestMemoryToTime(string actionId, float time)
```

Parameters

actionId [string](#)

time [float](#)

Returns

[MemoryData](#)

Remarks

Pass Time.time to getting the closest to now and pass 0 to get the earliest

GetClosestMemoryToTime(string, Object, float)

Gets the closest memory with the specified criteria

```
public MemoryData GetClosestMemoryToTime(string actionId, Object target, float time)
```

Parameters

actionId [string](#)

target [Object](#)

time [float](#)

Returns

[MemoryData](#)

Remarks

Pass Time.time to getting the closest to now and pass 0 to get the earliest

GetClosestMemoryToTime(Type, float)

Gets the memory closest to the time passed which is of the specified type

```
public MemoryData GetClosestMemoryToTime(Type type, float time)
```

Parameters

type [Type](#)

time [float](#)

Returns

[MemoryData](#)

Remarks

Pass Time.time to getting the closest to now and pass 0 to get the earliest

GetClosestMemoryToTime(Type, Object, float)

Gets the memory of a specific type and with a specific target which is closest to the passed time.

```
public MemoryData GetClosestMemoryToTime(Type type, Object target, float time)
```

Parameters

type [Type](#)

target Object

time [float](#)

Returns

[MemoryData](#)

Remarks

Pass Time.time to getting the closest to now and pass 0 to get the earliest

GetClosestMemoryToTime(Type, Object, Object, float)

Gets the memory closest to time with the specified time and target and initiator

```
public MemoryData GetClosestMemoryToTime(Type type, Object initiator, Object target,  
float time)
```

Parameters

type [Type](#)

initiator Object

target Object

time [float](#)

Returns

[MemoryData](#)

Remarks

Pass Time.time to getting the closest to now and pass 0 to get the earliest

GetClosestMemoryToTime(Object, float)

Get the closest memory with the specified initiator. This is good for checking if somebody done anything in a while/what is the last thing he/she has done.

```
public MemoryData GetClosestMemoryToTime(Object initiator, float time)
```

Parameters

initiator Object

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(Object, Object, float)

Gets the closest memory with the specified criteria

```
public MemoryData GetClosestMemoryToTime(Object initiator, Object target, float time)
```

Parameters

initiator Object

target Object

time [float](#)

Returns

[MemoryData](#)

Remarks

Pass Time.time to getting the closest to now and pass 0 to get the earliest

GetCurrentActionRuntime()

```
public float GetCurrentActionRuntime()
```

Returns

[float](#)

This component allows an agent or any other object to remember things which happen and query about them

GetMemoryAtIndex(int, bool)

Gets the memory at index

```
public MemoryData GetMemoryAtIndex(int index, bool isEverLasting)
```

Parameters

index [int](#)

isEverLasting [bool](#)

Returns

[MemoryData](#)

RemoveMemory(MemoryData)

Removes a memory

```
public void RemoveMemory(MemoryData memory)
```

Parameters

memory [MemoryData](#)

Update()

```
protected void Update()
```

Class MemoryContainer<T>

Namespace: [NoOpArmy.WiseFeline.Remembrance](#)

Assembly: APIRef.dll

```
[Serializable]
public class MemoryContainer<T> where T : MemoryData
```

Type Parameters

T

Inheritance

[object](#) ← MemoryContainer<T>

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

MemoryContainer(int)

```
public MemoryContainer(int maxCount)
```

Parameters

maxCount [int](#)

Fields

memories

```
public List<T> memories
```

Field Value

[List](#)<T>

Methods

AddMemory(T)

```
public T AddMemory(T memory)
```

Parameters

memory T

Returns

T

ForGetOldThings(float)

```
public void ForGetOldThings(float currentTime)
```

Parameters

currentTime float

GetAllMemories()

```
public List<T> GetAllMemories()
```

Returns

[List](#)<T>

GetClosestMemoryToTime(Predicate<T>, float, bool)

```
public T GetClosestMemoryToTime(Predicate<T> pred, float time, bool useEndTime = true)
```

Parameters

pred [Predicate](#)<T>

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(float, bool)

```
public T GetClosestMemoryToTime(float time, bool useEndTime = true)
```

Parameters

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(string, float, bool)

```
public T GetClosestMemoryToTime(string actionId, float time, bool useEndTime = true)
```

Parameters

actionId [string](#)

`time` [float](#)

`useEndTime` [bool](#)

Returns

T

GetClosestMemoryToTime(string, Object, float, bool)

```
public T GetClosestMemoryToTime(string actionId, Object target, float time, bool useEndTime = true)
```

Parameters

`actionId` [string](#)

`target` Object

`time` [float](#)

`useEndTime` [bool](#)

Returns

T

GetClosestMemoryToTime(Type, float, bool)

```
public T GetClosestMemoryToTime(Type type, float time, bool useEndTime = true)
```

Parameters

`type` [Type](#)

`time` [float](#)

`useEndTime` [bool](#)

Returns

T

GetClosestMemoryToTime(Type, Object, float, bool)

```
public T GetClosestMemoryToTime(Type type, Object target, float time, bool useEndTime  
= true)
```

Parameters

type [Type](#)

target Object

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(Type, Object, Object, float, bool)

```
public T GetClosestMemoryToTime(Type type, Object initiator, Object target, float time, bool  
useEndTime = true)
```

Parameters

type [Type](#)

initiator Object

target Object

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(Object, float, bool)

```
public T GetClosestMemoryToTime(Object initiator, float time, bool useEndTime = true)
```

Parameters

initiator Object

time [float](#)

useEndTime [bool](#)

Returns

T

GetClosestMemoryToTime(Object, Object, float, bool)

```
public T GetClosestMemoryToTime(Object initiator, Object target, float time, bool useEndTime  
= true)
```

Parameters

initiator Object

target Object

time [float](#)

useEndTime [bool](#)

Returns

T

GetMemoryAtIndex(int)

```
public T GetMemoryAtIndex(int index)
```

Parameters

index [int](#)

Returns

T

RemoveMemory(T)

```
public void RemoveMemory(T memory)
```

Parameters

memory T

Class MemoryData

Namespace: [NoOpArmy.WiseFeline.Remembrance](#)

Assembly: APIRef.dll

This data structure defines a memory entry

```
[Serializable]
public class MemoryData
```

Inheritance

[object](#) ← MemoryData

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Constructors

MemoryData(int, float, float, Vector3, Type, Object, Object, string, object)

```
public MemoryData(int memoryType, float startTime, float forgetAfterSeconds,
Vector3 position, Type actionType, Object initiator, Object target, string actionId,
object additionalData)
```

Parameters

memoryType [int](#)

This data structure defines a memory entry

startTime [float](#)

This data structure defines a memory entry

forgetAfterSeconds [float](#)

This data structure defines a memory entry

position Vector3

This data structure defines a memory entry

actionType [Type](#)

This data structure defines a memory entry

initiator Object

This data structure defines a memory entry

target Object

This data structure defines a memory entry

actionId [string](#)

This data structure defines a memory entry

additionalData [object](#)

This data structure defines a memory entry

Properties

ActionId

The unique id of the action happening to identify instances of a System.Type

```
public string ActionId { get; set; }
```

Property Value

[string](#)

This data structure defines a memory entry

ActionType

The Type of the class initiating this action. The initiator is the GameObject/component which caused this and this is the type of utility AI action or state machine state or ... which caused this

```
public Type ActionType { get; set; }
```

Property Value

[Type](#) ↗

This data structure defines a memory entry

AdditionalData

Any additional data you need for this memory to be casted based on memoryType

```
public object AdditionalData { get; set; }
```

Property Value

[object](#) ↗

This data structure defines a memory entry

EndTime

The time this finished

```
public float EndTime { get; set; }
```

Property Value

[float](#) ↗

This data structure defines a memory entry

ForgetAfterSeconds

The number of seconds which this will be forgotten after. This will never be forgotten if the value is less than or equal to 0

```
public float ForgetAfterSeconds { get; set; }
```

Property Value

[float](#)

This data structure defines a memory entry

Initiator

The object which caused this memory to happen. Can be null.

```
public Object Initiator { get; set; }
```

Property Value

Object

This data structure defines a memory entry

IsEnded

Is this memory ended or still happening

```
public bool IsEnded { get; }
```

Property Value

[bool](#)

This data structure defines a memory entry

MemoryType

Type of the memory, usually an enum in your project which help you cast any additional data or in general find out what memory type is this

```
public int MemoryType { get; set; }
```

Property Value

[int ↗](#)

This data structure defines a memory entry

OnEnded

```
public Action OnEnded { get; set; }
```

Property Value

[Action ↗](#)

This data structure defines a memory entry

OnStarted

```
public Action OnStarted { get; set; }
```

Property Value

[Action ↗](#)

This data structure defines a memory entry

Position

The position this memory happen

```
public Vector3 Position { get; set; }
```

Property Value

Vector3

This data structure defines a memory entry

StartTime

The time this started

```
public float StartTime { get; set; }
```

Property Value

[float](#)

This data structure defines a memory entry

Target

The target of the memory. Can be null.

```
public Object Target { get; set; }
```

Property Value

Object

This data structure defines a memory entry

Methods

End(float)

Ends the action. Like if memory is combat with giant then this should be called when the battle ends.

```
public void End(float time)
```

Parameters

`time` [float](#)

GetTime(bool)

```
public float GetTime(bool useEndTime = true)
```

Parameters

`useEndTime` [bool](#)

This data structure defines a memory entry

Returns

[float](#)

This data structure defines a memory entry

Class SimpleMemory

Namespace: [NoOpArmy.WiseFeline.Remembrance](#)

Assembly: APIRef.dll

```
public class SimpleMemory : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← SimpleMemory

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,

[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
ObjectFindObjectOfType<T>() , [Object.FindObjectType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

everLastingMemories

```
public CircularMemoryContainer<MemoryData> everLastingMemories
```

Field Value

[CircularMemoryContainer<MemoryData>](#)

everLastingMemoriesCount

`public int everLastingMemoriesCount`

Field Value

[int↗](#)

memories

`public CircularMemoryContainer<MemoryData> memories`

Field Value

[CircularMemoryContainer<MemoryData>](#)

normalMemoriesCount

`public int normalMemoriesCount`

Field Value

[int↗](#)

Methods

AddMemory(MemoryData, bool)

`public void AddMemory(MemoryData memory, bool isEverLasting = false)`

Parameters

`memory` [MemoryData](#)

`isEverLasting` [bool](#)

CreateAndGetListOfAllMemories()

```
public Queue<MemoryData> CreateAndGetListOfAllMemories()
```

Returns

[Queue](#) <[MemoryData](#)>

FindMostRecentMemoryEndTime(string)

```
public float FindMostRecentMemoryEndTime(string guid)
```

Parameters

`guid` [string](#)

Returns

[float](#)

FindMostRecentMemoryEndTime(string, Object)

```
public float FindMostRecentMemoryEndTime(string guid, Object target)
```

Parameters

`guid` [string](#)

`target` Object

Returns

[float](#)

FindMostRecentMemoryEndTime(Type)

```
public float FindMostRecentMemoryEndTime(Type type)
```

Parameters

[type](#) [Type](#)

Returns

[float](#)

GetAllEverLastingMemories()

```
public Queue<MemoryData> GetAllEverLastingMemories()
```

Returns

[Queue](#) <[MemoryData](#)>

GetAllNormalMemories()

```
public Queue<MemoryData> GetAllNormalMemories()
```

Returns

[Queue](#) <[MemoryData](#)>

GetClosestMemoryToTime(Predicate<MemoryData>, float)

```
public MemoryData GetClosestMemoryToTime(Predicate<MemoryData> pred, float time)
```

Parameters

pred [Predicate](#)<MemoryData>

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(float)

```
public MemoryData GetClosestMemoryToTime(float time)
```

Parameters

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(string, float)

```
public MemoryData GetClosestMemoryToTime(string actionId, float time)
```

Parameters

actionId [string](#)

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(string, Object, float)

```
public MemoryData GetClosestMemoryToTime(string actionId, Object target, float time)
```

Parameters

actionId [string](#)

target [Object](#)

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(Type, float)

```
public MemoryData GetClosestMemoryToTime(Type type, float time)
```

Parameters

type [Type](#)

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(Type, Object, float)

```
public MemoryData GetClosestMemoryToTime(Type type, Object target, float time)
```

Parameters

type [Type](#)

target Object

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(Type, Object, Object, float)

```
public MemoryData GetClosestMemoryToTime(Type type, Object initiator, Object target,  
float time)
```

Parameters

type [Type](#)

initiator Object

target Object

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(Object, float)

```
public MemoryData GetClosestMemoryToTime(Object initiator, float time)
```

Parameters

initiator Object

time [float](#)

Returns

[MemoryData](#)

GetClosestMemoryToTime(Object, Object, float)

```
public MemoryData GetClosestMemoryToTime(Object initiator, Object target, float time)
```

Parameters

initiator Object

target Object

time [float](#)

Returns

[MemoryData](#)

GetCurrentActionRuntime()

```
public float GetCurrentActionRuntime()
```

Returns

[float](#)

Namespace NoOpArmy.WiseFeline.Smart Objects

Classes

[SlotDefinition](#)

[SlotSpecificSmartObjectBehavior](#)

This class represents a behavior assignment to a smart object for a specific slot

[SmartObject](#)

Attach this component to any object which you want to declare as smart object

[SmartObject.FilterResult](#)

Represents the data which shows if a behavior is supported by an object

[SmartObject.SmartObjectSlotData](#)

Represents runtime data about the status of a slot

[SmartObjectBehaviourBase](#)

Behaviors of smart objects inherit from this class. A smart object behavior operates on the agent which reserves, uses or frees a slot in a smart objects. These events are defined in this base class as virtual methods to override.

[SmartObjectDefinition](#)

Defines the characteristics of a smart object

[SmartObjectSlotOwnerSetter](#)

Sets the owners of the smart object slots to the elements of the array

[SmartObjectsManager](#)

Manages smart objects and allows you to query them based on their behaviors, tags and position An instance of this is created automatically if one cannot be found in the scene

Enums

[SlotState](#)

Different states a slot can be in

[SmartObjectsManager.SearchQueryOptions](#)

Use this enum to customize a search query to return objects which you own or are free or all objects

Class SlotDefinition

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

```
[Serializable]
public class SlotDefinition
```

Inheritance

[object](#) ← SlotDefinition

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

agentOffsetFromPosition

The agent's ideal position to use the slot from the slot's position

```
[Tooltip("The agent's ideal position to use the slot from the slot's position")]
public Vector3 agentOffsetFromPosition
```

Field Value

Vector3

agentRotation

The agent's ideal rotation to use the slot

```
[Tooltip("The agent's ideal rotation to use the slot")]
public Vector3 agentRotation
```

Field Value

Vector3

positionOffset

Position offset of the slot relative to the object itself

```
[Tooltip("Position offset of the slot relative to the object itself")]
public Vector3 positionOffset
```

Field Value

Vector3

Class SlotSpecificSmartObjectBehavior

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

This class represents a behavior assignment to a smart object for a specific slot

```
[Serializable]
public class SlotSpecificSmartObjectBehavior
```

Inheritance

[object](#) ← SlotSpecificSmartObjectBehavior

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

behaviors

The behavior to use

```
public List<SmartObjectBehaviourBase> behaviors
```

Field Value

[List](#)<[SmartObjectBehaviourBase](#)>

This class represents a behavior assignment to a smart object for a specific slot

slotIndex

The slot index which uses this behavior

```
public int slotIndex
```

Field Value

[int↗](#)

This class represents a behavior assignment to a smart object for a specific slot

Enum SlotState

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Different states a slot can be in

```
public enum SlotState
```

Fields

Claimed = 1

Reserved by an agent

Free = 0

Free to be used by any agent

InUse = 2

In use by an agent

Class SmartObject

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Attach this component to any object which you want to declare as smart object

```
public class SmartObject : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← SmartObject

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
[Component.GetComponentInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) ,
[Component.GetComponentInParent<T>\(bool, List<T>\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,

[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Fields

behaviors

The behaviors which all slots offer

```
[Tooltip("The behaviors which all slots offer")]
public List<SmartObjectBehaviourBase> behaviors
```

Field Value

[List ↗ <SmartObjectBehaviourBase>](#)

Attach this component to any object which you want to declare as smart object

definition

The smart object definition this component will use

```
[Tooltip("The smart object definition this component will use")]
public SmartObjectDefinition definition
```

Field Value

[SmartObjectDefinition](#)

Attach this component to any object which you want to declare as smart object

delayBetweenPositionUpdates

If we should update the position of the octree, how many seconds of delay should be between the updates

```
[Tooltip("If we should update the position of the octree, how many seconds of delay should
be between the updates")]
public float delayBetweenPositionUpdates
```

Field Value

[float ↗](#)

Attach this component to any object which you want to declare as smart object

previousUpdate

The last time this object's position has updated on the octree

```
[HideInInspector]  
public float previousUpdate
```

Field Value

[float](#)

Attach this component to any object which you want to declare as smart object

priorityMultiplier

Multiplies the definition's priority by this number

```
public float priorityMultiplier
```

Field Value

[float](#)

Attach this component to any object which you want to declare as smart object

slotBehaviors

Slot specific behaviors

```
[Tooltip("Slot specific behaviors")]  
public List<SlotSpecificSmartObjectBehavior> slotBehaviors
```

Field Value

[List](#)<[SlotSpecificSmartObjectBehavior](#)>

Attach this component to any object which you want to declare as smart object

tags

Smart object tags

```
[Tooltip("Smart object tags")]
public List<string> tags
```

Field Value

[List](#) <[string](#)>

Attach this component to any object which you want to declare as smart object

updatePositionInOctreeAutomatically

Should the position of this object be updated on the octree. If the object doesn't move this should be false

```
[Tooltip("Should the position of this object be updated on the octree. If the object doesn't
move this should be false")]
public bool updatePositionInOctreeAutomatically
```

Field Value

[bool](#)

Attach this component to any object which you want to declare as smart object

Methods

AddSlot(SlotDefinition)

Adds a slot to the smart object

```
public void AddSlot(SlotDefinition def)
```

Parameters

```
def SlotDefinition
```

AddSlot(Vector3, Vector3, Vector3)

Adds a slot to the smart object

```
public void AddSlot(Vector3 positionOffset, Vector3 agentOffsetPosition,  
Vector3 agentRotation)
```

Parameters

positionOffset Vector3

Offset of the slot from the object's position

agentOffsetPosition Vector3

Offset of the ideal agent position from the slot

agentRotation Vector3

Idle agent rotation for the slot

Claim(int, GameObject, bool)

Tries to claim the slot for the agent

```
public bool Claim(int slot, GameObject user, bool forceOwnership = false)
```

Parameters

slot [int](#)

user GameObject

forceOwnership [bool](#)

Returns

[bool](#)

Free(int, GameObject, bool)

Frees the object from its agent

```
public bool Free(int slot, GameObject user, bool forceOwnership = false)
```

Parameters

slot [int](#)

user GameObject

forceOwnership [bool](#)

Returns

[bool](#)

Free(GameObject)

Release the first slot owned by the provided user

```
public bool Free(GameObject user)
```

Parameters

user GameObject

Returns

[bool](#)

Return true if freeing up a slot.

FreeAllSlots()

Frees all slots by their users

```
public void FreeAllSlots()
```

GetAgentSlotPosition(int)

Gets the position and rotation of the agent from the slot

```
public (Vector3 position, Quaternion rotation) GetAgentSlotPosition(int index)
```

Parameters

index [int](#)

Returns

([Vector3](#) [position](#), [Quaternion](#) [rotation](#))

GetAgentSlotPosition(GameObject)

Gets the position and rotation of the agent from the slot

```
public (Vector3 position, Quaternion rotation) GetAgentSlotPosition(GameObject owner)
```

Parameters

owner [GameObject](#)

Returns

([Vector3](#) [position](#), [Quaternion](#) [rotation](#))

GetClosestSlotPosition(GameObject, SearchQueryOptions)

Find closest slot position to target based on the search query option

```
public Vector3 GetClosestSlotPosition(GameObject target,
```

```
SmartObjectsManager.SearchQueryOptions queryOption)
```

Parameters

target GameObject

queryOption [SmartObjectsManager.SearchQueryOptions](#)

Returns

Vector3

The function returns Vector3.positiveInfinity when no slot meets the search query conditions.

GetFirstOwnedSlotIndex(GameObject)

Returns the first slot we own

```
public int GetFirstOwnedSlotIndex(GameObject potentialOwner)
```

Parameters

potentialOwner GameObject

Returns

[int](#)

GetFreeSlotIndex()

Finds and returns a free slot

```
public int GetFreeSlotIndex()
```

Returns

[int](#)

Index of the first free slot it finds, -1 if no free slot can be found

GetFreeWithoutOwnerSlotIndex()

Finds and returns a free slot

```
public int GetFreeWithoutOwnerSlotIndex()
```

Returns

[int ↗](#)

Index of the first free slot it finds, -1 if no free slot can be found

GetOwner(int)

Gets the owner of the object

```
public GameObject GetOwner(int slotIndex)
```

Parameters

slotIndex [int ↗](#)

Attach this component to any object which you want to declare as smart object

Returns

GameObject

GetPriority()

Returns the priority of this object which says how good it can do the job. The value is arbitrary and can be used by your code to determine which of the objects are better to use. This alongside tags and other features can help you choose a better object. for example a ready bomb to demolate somewhere instead of the material to make one and a bomb making kit.

```
public float GetPriority()
```

Returns

[float](#)

GetSlotCount()

Returns the number of slots the smart object has

```
public int GetSlotCount()
```

Returns

[int](#)

GetSlotData(int)

Gets a copy of the slot's runtime data

```
public SmartObject.SmartObjectSlotData GetSlotData(int index)
```

Parameters

[index](#) [int](#)

Returns

[SmartObject.SmartObjectSlotData](#)

GetSlotDefinition(int)

Gets the slot definition

```
public SlotDefinition GetSlotDefinition(int index)
```

Parameters

index [int](#)

Attach this component to any object which you want to declare as smart object

Returns

[SlotDefinition](#)

GetSlotDefinition(GameObject)

Gets the slot definition

```
public SlotDefinition GetSlotDefinition(GameObject owner)
```

Parameters

owner GameObject

Returns

[SlotDefinition](#)

GetSlotPosition(int)

Gets the position of the slot

```
public Vector3 GetSlotPosition(int index)
```

Parameters

index [int](#)

Returns

Vector3

GetSlots()

returns the slot data of the smart object which you can use to get slot states, claiming/using GameObject and definition

```
public List<SmartObject.SmartObjectSlotData> GetSlots()
```

Returns

[List ↗ <SmartObject.SmartObjectSlotData>](#)

RemoveSlotAtIndex(int)

Removes a slot from the smart object

```
public void RemoveSlotAtIndex(int index)
```

Parameters

index [int ↗](#)

SetOwner(GameObject, int)

Sets the owner of the object

```
public void SetOwner(GameObject go, int slotIndex)
```

Parameters

go GameObject

slotIndex [int ↗](#)

Attach this component to any object which you want to declare as smart object

SetOwnerOfAllSlots(GameObject)

Sets the owner of all slots to the gameobject

```
public void SetOwnerOfAllSlots(GameObject go)
```

Parameters

go GameObject

SupportsBehavior(Type, SearchQueryOptions, GameObject, string[], string[])

Does this object support our requirements of behaviors and tags

```
public SmartObject.FilterResult SupportsBehavior(Type behavior,
SmartObjectsManager.SearchQueryOptions options = SearchQueryOptions.None, GameObject
potentialOwner = null, string[] tagsToInclude = null, string[] tagsToExclude = null)
```

Parameters

behavior [Type](#)

The behavior type the object should support

options [SmartObjectsManager.SearchQueryOptions](#)

Attach this component to any object which you want to declare as smart object

potentialOwner GameObject

Attach this component to any object which you want to declare as smart object

tagsToInclude [string](#)[]

Should we check if the object has any of these tags, pass null

tagsToExclude [string](#)[]

Tags to exclude so an object with any of these tags doesn't count. Pass null if you don't want to check any tags

Returns

[SmartObject.FilterResult](#)

Returns if this object supports the query and if yes is it slot specific and with what slots

SupportsTagFilter(string[], string[])

```
public bool SupportsTagFilter(string[] includeTags, string[] excludeTags)
```

Parameters

includeTags [string](#)[]

Attach this component to any object which you want to declare as smart object

excludeTags [string](#)[]

Attach this component to any object which you want to declare as smart object

Returns

[bool](#)

Attach this component to any object which you want to declare as smart object

Use(int, GameObject, bool)

Tries to reserve the object for the agent

```
public bool Use(int slot, GameObject user, bool forceOwnership = false)
```

Parameters

slot [int](#)[]

user GameObject

forceOwnership [bool](#)

Returns

bool ↴

Class SmartObject.FilterResult

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Represents the data which shows if a behavior is supported by an object

```
public class SmartObject.FilterResult
```

Inheritance

[object](#) ← SmartObject.FilterResult

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

isSlotSpecific

Is the support slot specific or all slots support the behavior

```
public bool isSlotSpecific
```

Field Value

[bool](#)

Represents the data which shows if a behavior is supported by an object

isSupported

Is the type supported

```
public bool isSupported
```

Field Value

bool ↗

Represents the data which shows if a behavior is supported by an object

slots

List of the slots in case of the slot specific support

```
public List<int> slots
```

Field Value

List ↗ <int ↗ >

Represents the data which shows if a behavior is supported by an object

Class SmartObject.SmartObjectSlotData

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Represents runtime data about the status of a slot

```
public class SmartObject.SmartObjectSlotData
```

Inheritance

[object](#) ← SmartObject.SmartObjectSlotData

Inherited Members

[object.Equals\(object\)](#) , [object.Equals\(object, object\)](#) , [object.GetHashCode\(\)](#) , [object.GetType\(\)](#) ,
[object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#) , [object.ToString\(\)](#)

Fields

definition

```
public SlotDefinition definition
```

Field Value

[SlotDefinition](#)

Represents runtime data about the status of a slot

owner

Owner of the slot

```
[SerializeField]  
public GameObject owner
```

Field Value

GameObject

Represents runtime data about the status of a slot

state

`public SlotState state`

Field Value

[SlotState](#)

Represents runtime data about the status of a slot

user

`public GameObject user`

Field Value

GameObject

Represents runtime data about the status of a slot

Class SmartObjectBehaviourBase

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Behaviors of smart objects inherit from this class. A smart object behavior operates on the agent which reserves, uses or frees a slot in a smart objects. These events are defined in this base class as virtual methods to override.

```
public class SmartObjectBehaviourBase : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← SmartObjectBehaviourBase

Derived

[DrivingBehaviour](#), [EatFoodBehaviour](#)

Inherited Members

MonoBehaviour.IsInvoking(), MonoBehaviour.CancelInvoke(), [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine), [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines(), [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentInParent<T>()

[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode()
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectInactive) ,
Object.FindAnyObjectByType<T>(FindObjectInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Methods

OnClaimedByAgent(GameObject, SmartObject, int)

Called when a slot of the smart object is successfully claimed by an object

```
public virtual void OnClaimedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

agent GameObject

smartObject [SmartObject](#)

slot [int](#)

OnFreedByAgent(GameObject, SmartObject, int)

Called when an agent frees the smart object

```
public virtual void OnFreedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

agent GameObject

smartObject [SmartObject](#)

slot [int](#)

OnUsedByAgent(GameObject, SmartObject, int)

Called when a slot of the object is successfully used by an agent

```
public virtual void OnUsedByAgent(GameObject agent, SmartObject smartObject, int slot)
```

Parameters

agent GameObject

smartObject [SmartObject](#)

slot [int](#)

Class SmartObjectDefinition

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Defines the characteristics of a smart object

```
[CreateAssetMenu(menuName = "NoOpArmy/Wise Feline/SmartObject Definition")]
public class SmartObjectDefinition : ScriptableObject
```

Inheritance

[object](#) ← Object ← ScriptableObject ← SmartObjectDefinition

Inherited Members

ScriptableObject.SetDirty() , [ScriptableObject.CreateInstance\(string\)](#) ,
[ScriptableObject.CreateInstance\(Type\)](#) , ScriptableObject.CreateInstance<T>() , Object.GetInstanceID() ,
Object.GetHashCode() , [Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,

`Object.hideFlags` , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) , [object.ReferenceEquals\(object, object\)](#)

Fields

priority

How good this smart object is at doing its job

```
[Tooltip("How good this smart object is at doing its job. Agents use this to decide if they  
should use this object or another")]  
public float priority
```

Field Value

[float](#)

Defines the characteristics of a smart object

slots

The slots of the smart object

```
[Tooltip("The slots of the smart object")]  
public SlotDefinition[] slots
```

Field Value

[SlotDefinition\[\]](#)

Defines the characteristics of a smart object

Class SmartObjectSlotOwnerSetter

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Sets the owners of the smart object slots to the elements of the array

```
public class SmartObjectSlotOwnerSetter : MonoBehaviour
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← SmartObjectSlotOwnerSetter

Inherited Members

MonoBehaviour.IsInvoking() , MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
[Component.GetComponentInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) ,
[Component.GetComponentInParent<T>\(bool, List<T>\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,
[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,

[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectsInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectByType<T>() , Object.FindAnyObjectByType<T>() ,
Object.FindFirstObjectByType<T>(FindObjectsInactive) ,
Object.FindAnyObjectByType<T>(FindObjectsInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectByType\(Type\)](#) ,
[Object.FindAnyObjectByType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectByType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectByType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Class SmartObjectsManager

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Manages smart objects and allows you to query them based on their behaviors, tags and position An instance of this is created automatically if one cannot be found in the scene

```
public class SmartObjectsManager : Singleton<SmartObjectsManager>
```

Inheritance

[object](#) ← Object ← Component ← Behaviour ← MonoBehaviour ← [Singleton<SmartObjectsManager>](#) ← SmartObjectsManager

Inherited Members

[Singleton<SmartObjectsManager>.Instance](#) , [Singleton<SmartObjectsManager>.Awake\(\)](#) ,
[Singleton<SmartObjectsManager>.OnDestroy\(\)](#) , MonoBehaviour.IsInvoking() ,
MonoBehaviour.CancelInvoke() , [MonoBehaviour.Invoke\(string, float\)](#) ,
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#) , [MonoBehaviour.CancelInvoke\(string\)](#) ,
[MonoBehaviour.IsInvoking\(string\)](#) , [MonoBehaviour.StartCoroutine\(string\)](#) ,
[MonoBehaviour.StartCoroutine\(string, object\)](#) , [MonoBehaviour.StartCoroutine\(IEnumerator\)](#) ,
[MonoBehaviour.StartCoroutine_Auto\(IEnumerator\)](#) , [MonoBehaviour.StopCoroutine\(IEnumerator\)](#) ,
MonoBehaviour.StopCoroutine(Coroutine) , [MonoBehaviour.StopCoroutine\(string\)](#) ,
MonoBehaviour.StopAllCoroutines() , [MonoBehaviour.print\(object\)](#) , MonoBehaviour.useGUILayout ,
MonoBehaviour.runInEditMode , Behaviour.enabled , Behaviour.isActiveAndEnabled ,
[Component.GetComponent\(Type\)](#) , Component.GetComponent<T>() ,
[Component.TryGetComponent\(Type, out Component\)](#) , Component.TryGetComponent<T>(out T) ,
[Component.GetComponent\(string\)](#) , [Component.GetComponentInChildren\(Type, bool\)](#) ,
[Component.GetComponentInChildren\(Type\)](#) , [Component.GetComponentInChildren<T>\(bool\)](#) ,
Component.GetComponentInChildren<T>() , [Component.GetComponentsInChildren\(Type, bool\)](#) ,
[Component.GetComponentsInChildren\(Type\)](#) , [Component.GetComponentsInChildren<T>\(bool\)](#) ,
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#) ,
Component.GetComponentsInChildren<T>() , [Component.GetComponentsInChildren<T>\(List<T>\)](#) ,
[Component.GetComponentInParent\(Type, bool\)](#) , [Component.GetComponentInParent\(Type\)](#) ,
[Component.GetComponentInParent<T>\(bool\)](#) , Component.GetComponentInParent<T>() ,
[Component.GetComponentsInParent\(Type, bool\)](#) , [Component.GetComponentsInParent\(Type\)](#) ,
[Component.GetComponentsInParent<T>\(bool\)](#) ,
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#) , Component.GetComponentsInParent<T>() ,
[Component.GetComponents\(Type\)](#) , [Component.GetComponents\(Type, List<Component>\)](#) ,

[Component.GetComponents<T>\(List<T>\)](#) , Component.GetComponents<T>() ,
[Component.CompareTag\(string\)](#) ,
[Component.SendMessageUpwards\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessageUpwards\(string, object\)](#) , [Component.SendMessageUpwards\(string\)](#) ,
[Component.SendMessageUpwards\(string, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, object\)](#) , [Component.SendMessage\(string\)](#) ,
[Component.SendMessage\(string, object, SendMessageOptions\)](#) ,
[Component.SendMessage\(string, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object, SendMessageOptions\)](#) ,
[Component.BroadcastMessage\(string, object\)](#) , [Component.BroadcastMessage\(string\)](#) ,
[Component.BroadcastMessage\(string, SendMessageOptions\)](#) , Component.transform ,
Component.gameObject , Component.tag , Object.GetInstanceID() , Object.GetHashCode() ,
[Object.Equals\(object\)](#) , Object.Instantiate(Object, Vector3, Quaternion) ,
Object.Instantiate(Object, Vector3, Quaternion, Transform) , Object.Instantiate(Object) ,
Object.Instantiate(Object, Transform) , [Object.Instantiate\(Object, Transform, bool\)](#) ,
Object.Instantiate<T>(T) , Object.Instantiate<T>(T, Vector3, Quaternion) ,
Object.Instantiate<T>(T, Vector3, Quaternion, Transform) , Object.Instantiate<T>(T, Transform) ,
[Object.Instantiate<T>\(T, Transform, bool\)](#) , [Object.Destroy\(Object, float\)](#) , Object.Destroy(Object) ,
[Object.DestroyImmediate\(Object, bool\)](#) , Object.DestroyImmediate(Object) ,
[Object.FindObjectsOfType\(Type\)](#) , [Object.FindObjectsOfType\(Type, bool\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsSortMode\)](#) ,
[Object.FindObjectsByType\(Type, FindObjectsInactive, FindObjectsSortMode\)](#) ,
Object.DontDestroyOnLoad(Object) , [Object.DestroyObject\(Object, float\)](#) ,
Object.DestroyObject(Object) , [Object.FindSceneObjectsOfType\(Type\)](#) ,
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#) , Object.FindObjectsOfType<T>() ,
Object.FindObjectsByType<T>(FindObjectsSortMode) , [Object.FindObjectsOfType<T>\(bool\)](#) ,
Object.FindObjectsByType<T>(FindObjectInactive, FindObjectsSortMode) ,
Object.FindObjectOfType<T>() , [Object.FindObjectOfType<T>\(bool\)](#) ,
Object.FindFirstObjectOfType<T>() , Object.FindAnyObjectOfType<T>() ,
Object.FindFirstObjectOfType<T>(FindObjectInactive) ,
Object.FindAnyObjectOfType<T>(FindObjectInactive) , [Object.FindObjectsOfTypeAll\(Type\)](#) ,
[Object.FindObjectOfType\(Type\)](#) , [Object.FindFirstObjectOfType\(Type\)](#) ,
[Object.FindAnyObjectOfType\(Type\)](#) , [Object.FindObjectOfType\(Type, bool\)](#) ,
[Object.FindFirstObjectOfType\(Type, FindObjectsInactive\)](#) ,
[Object.FindAnyObjectOfType\(Type, FindObjectsInactive\)](#) , Object.ToString() , Object.name ,
Object.hideFlags , [object.Equals\(object, object\)](#) , [object.GetType\(\)](#) , [object.MemberwiseClone\(\)](#) ,
[object.ReferenceEquals\(object, object\)](#)

Methods

GetSmartObjectsAroundPoint(Vector3, float)

Returns all of the objects in the sphere

```
public SmartObject[] GetSmartObjectsAroundPoint(Vector3 point, float radius)
```

Parameters

point Vector3

radius [float](#)

Returns

[SmartObject](#)[]

GetSmartObjectsAroundPoint(Vector3, float, Type, SearchQueryOptions, GameObject, string[], string[])

Get objects which match the query

```
public List<(SmartObject smartObject, SmartObject.FilterResult data)>
GetSmartObjectsAroundPoint(Vector3 point, float radius, Type behavior = null,
SmartObjectsManager.SearchQueryOptions queryOptions = SearchQueryOptions.None, GameObject
potentialOwner = null, string[] tagsToInclude = null, string[] tagsToExclude = null)
```

Parameters

point Vector3

radius [float](#)

behavior [Type](#)

queryOptions [SmartObjectsManager.SearchQueryOptions](#)

potentialOwner GameObject

tagsToInclude [string](#)[]

tagsToExclude [string](#)[]

Returns

[List](#)<([SmartObject](#) smartObject, [SmartObject.FilterResult](#) data)>

GetSmartObjectsAroundPoint<T>(Vector3, float, SearchQueryOptions, GameObject, string[], string[])

Get objects which match the query

```
public List<(SmartObject smartObject, SmartObject.FilterResult data)>
GetSmartObjectsAroundPoint<T>(Vector3 point, float radius,
SmartObjectsManager.SearchQueryOptions queryOptions = SearchQueryOptions.None, GameObject
potentialOwner = null, string[] tagsToInclude = null, string[] tagsToExclude = null) where T
: SmartObjectBehaviourBase
```

Parameters

point Vector3

radius [float](#)

queryOptions [SmartObjectsManager.SearchQueryOptions](#)

potentialOwner GameObject

tagsToInclude [string](#)[]

tagsToExclude [string](#)[]

Returns

[List](#)<([SmartObject](#) smartObject, [SmartObject.FilterResult](#) data)>

Type Parameters

T

Enum SmartObjectsManager.SearchQueryOptions

Namespace: [NoOpArmy.WiseFeline.SmartObjects](#)

Assembly: APIRef.dll

Use this enum to customize a search query to return objects which you own or are free or all objects

```
public enum SmartObjectsManager.SearchQueryOptions : byte
```

Fields

Free = 2

Returns all objects which meet other criteria of the search and have free slots

FreeOrOwned = 3

Returns all objects which meet other criteria of the search and either have free slots or slots you own

None = 0

Returns all objects meeting the other criteria of the search

Owned = 1

Returns all objects meeting other criteria of the search which at least have a slot which you own