Blaze AI Documentation

For any questions and support, visit the [Discord](#) or you can email me directly at: *pathiralgames@gmail.com*

Blaze AI Discord Server [here](#)

Pathiral Tools Discord Server [here](#)

Pathiral Asset Store Page [here](#)

Index:

Video Tutorials:

Making Distractions

Vision System

Getting Started:

1. Add an empty Plane game object to your scene. This will be the ground the AI walks on.

2. Now make sure the plane is selected and open Unity's **Navigation inspector** using *Window > AI > Navigation* and go to the Object tab. Set the **Navigation Static** to *true* and set the **Navigation Area** to *Walkable*.
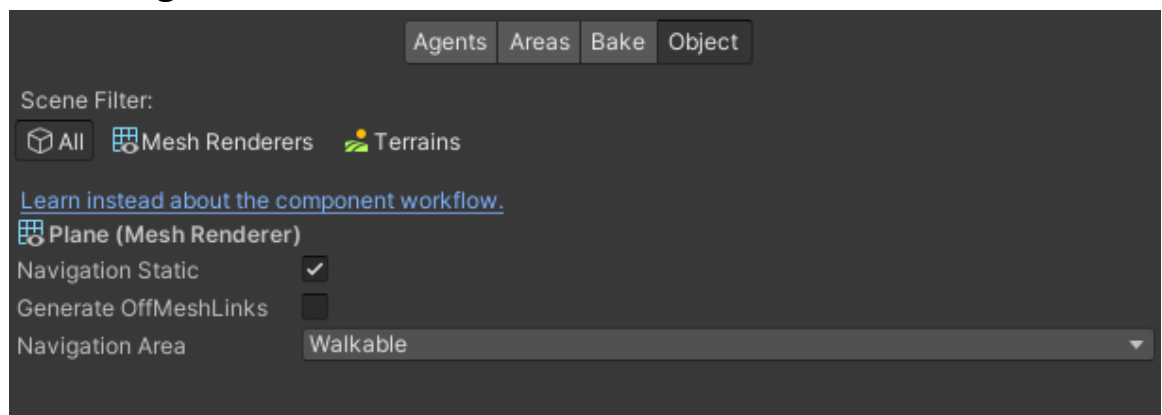
3. **For adding obstacles in your scene**. Click on the object, go to the navigation inspector. Set the **Navigation Static** to true also but the only difference is setting the **Navigation Area** to *Not Walkable.*
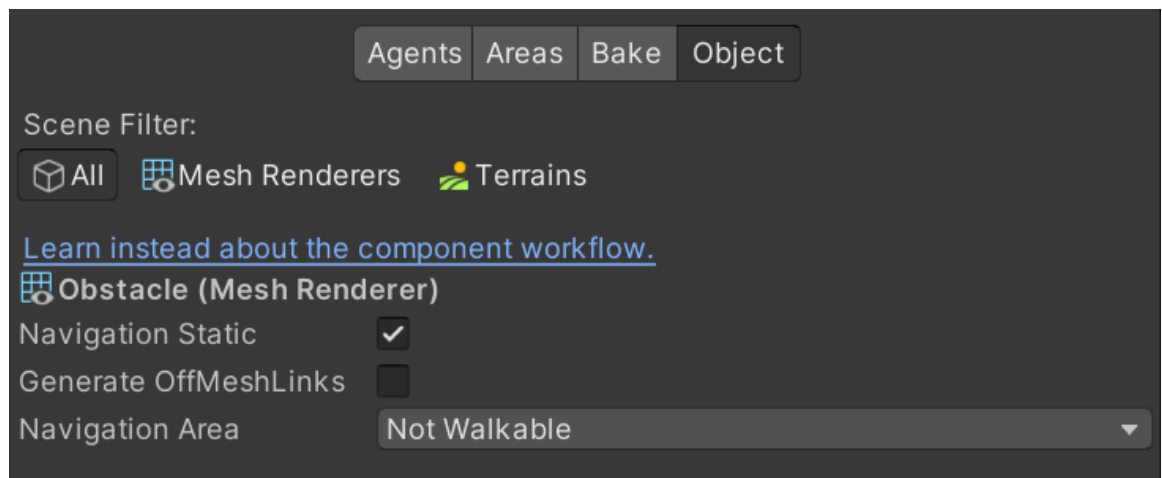
4. Now that you have the plane and obstacles, make sure the plane (ground) is selected and go to the **Bake** tab and click *Bake* below.
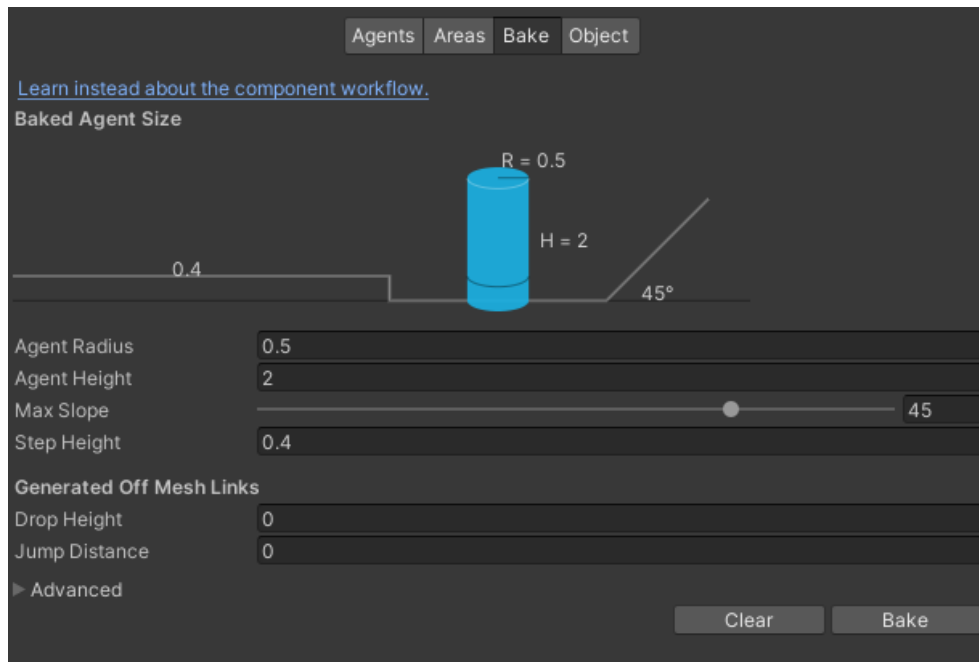
5. Add Blaze AI component to your character game object.

6. You will find that some required components have been added as well. Such as Animator and NavMeshAgent.

7. Clicking on the States tab, you'll find properties requiring a script for each one. You'll find that in other places in the inspector as well. You'll also find **Add Behaviour** button in most tabs. Click on it and it will automatically add the default behaviour(s) and set them to their respective properties.
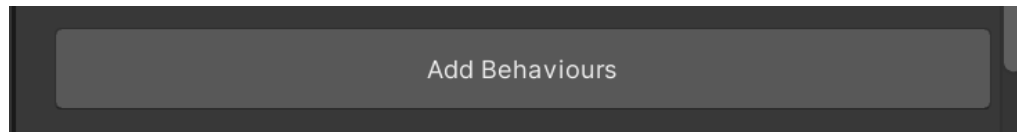
8. **Blaze comes with a behavior script for each and every state property with the same name.** So for example the Normal State Behaviour property has the NormalStateBehaviour script and so on. *You can make your own custom behaviour script and drag/drop on the required state also.*

9. On the added Normal State behaviour component, setup the properties by adding the animation names for idle and move - we'll get to the animations later in this doc. In the section **Animations** after Getting Started - and the desired speeds, audios, etc…

10. Now go back to the **Blaze AI inspector > General tab > Waypoints**. By default, Randomize is enabled. Randomize means waypoints will not be read but rather the AI will generate a new random point on each cycle. In other words, will be patrolling around the navmesh randomly within a specified radius.

11. Inside the Vision class in General tab. You can set your enemies by adding their layers inside **Hostile And Alert Layers** and their tag name inside **Hostile Tags**. The **Layers To Detect** on the other hand is what you generally want the vision to detect. If a layer isn't set, it will be seen right through it's game object. **The hostile needs to have atleast one collider**. Multiple colliders are ok too. **More on that in Adding Enemies section.**

On pressing play, you'll find that your AI is walking around.

You can also build the navmesh using Unity's NavMesh Components. Using the Navmesh Surface component.

Tooltips exist on most properties. Just hover your mouse over any property and it'll popup more info.

Animations

In many parts inside the inspector you'll find that you need to set the **Anim** and **AnimT** of a certain thing. Like these:

Move Anim	Walking
Anim T	0.25
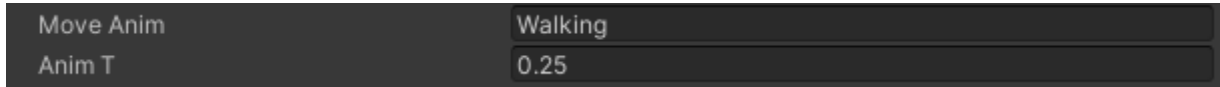
Now what is meant by these?

Let's start with the easy one. The AnimT is the animation transition time. As self-explanatory as it is, is simply the amount of time from a current animation to the animation in question.

The Anim is the animation name. What is known inside of the Unity Engine as the *Animation State Name.* Which is this:
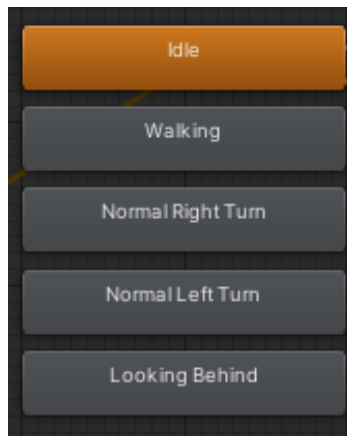
Make sure to enable loop in all your animation files.

It's the name of your animation inside the Animator. That's it!

So my move animation is called **Walking** so I set the property to **Walking**. You simply enter what the animation is called inside the Animator. You can change the animation state name inside the Animator like this:
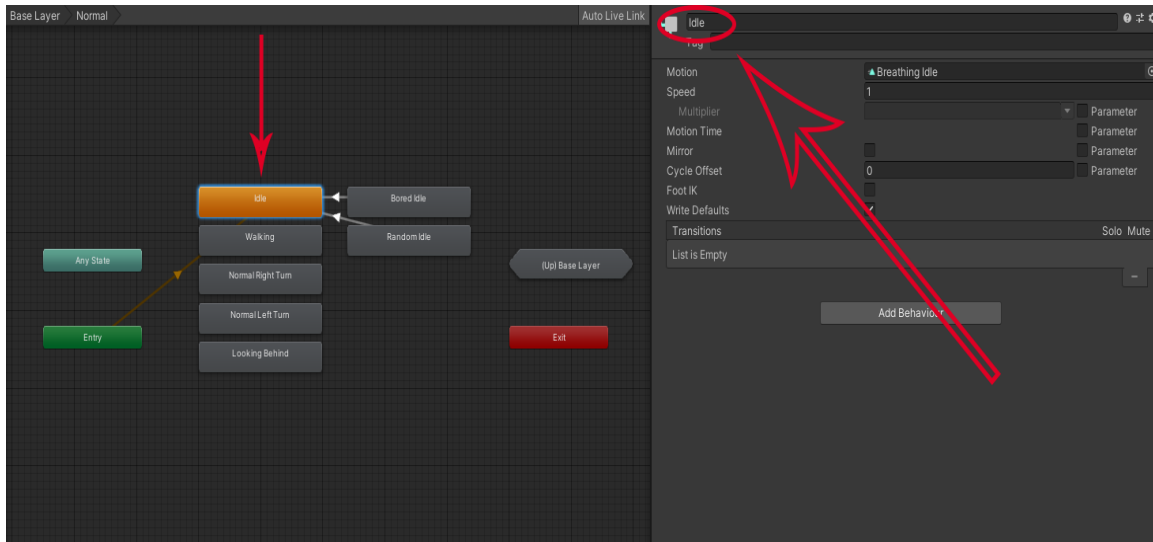
By clicking on the animation and then changing the name from the top right. **But, remember you have changed the animation state name so you'll have to change it's name from Blaze AI to match the Animator.**

You don't need to make transitions between your animations. Simply drag/drop your animations into the Animator and organize them as you please and write their names in Blaze. That's it!

Audios

All audios to be played inside of Blaze and it's behaviours are set inside a scriptable object which is called an **Audio Scriptable**.
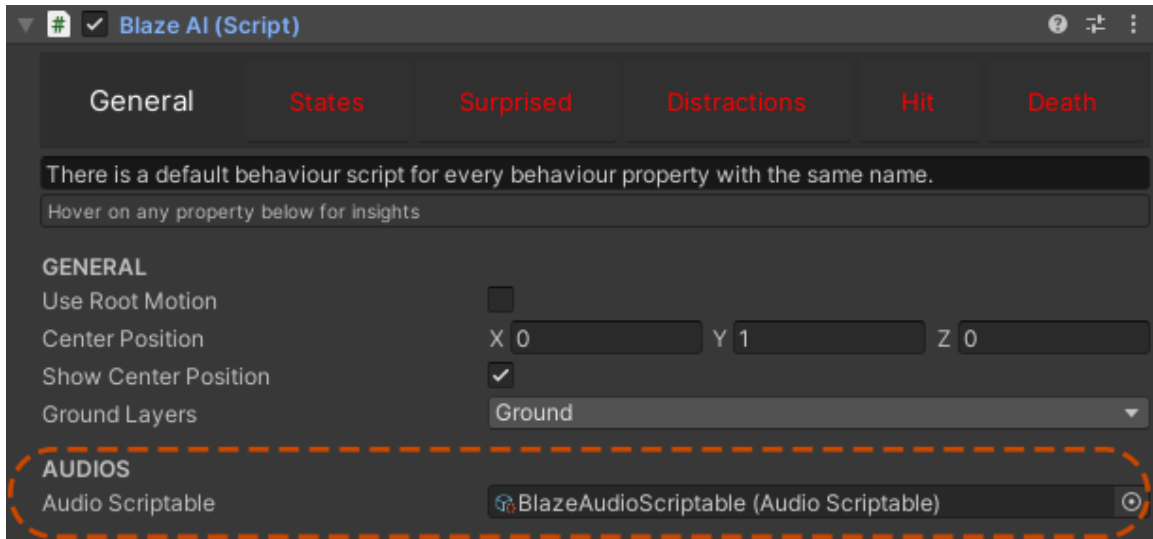
Blaze gives the option to create an Audio Scriptable and set your audio clips to all the necessary states and actions. Create an audio scriptable like this: (right clicking in the project space)
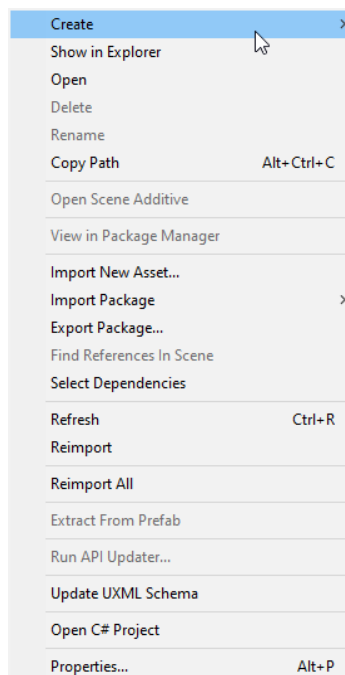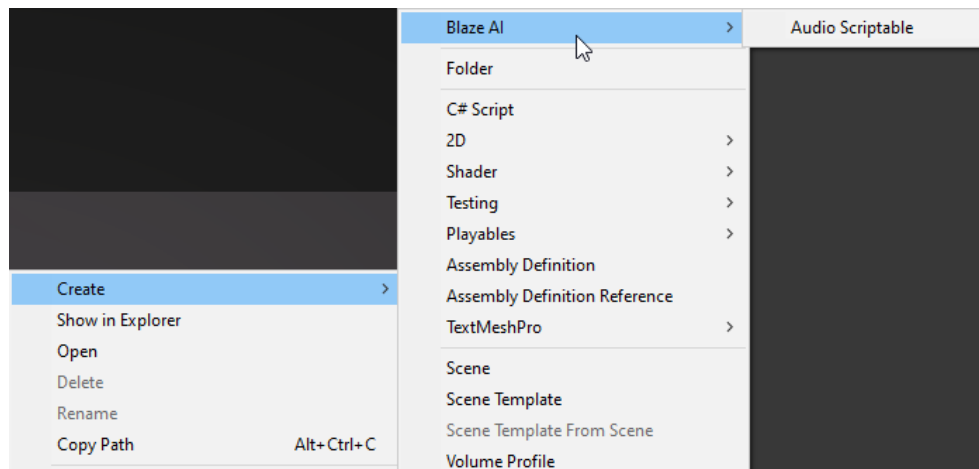
For each state/action, you can add an unlimited number of audios. When an audio call is made the system will choose a random audio in that state, if only one is set then that one will always be played. If empty, no audio will be played.

The central audio of the agent that will play all the audio clips is set automatically in the **Agent Audio** property.
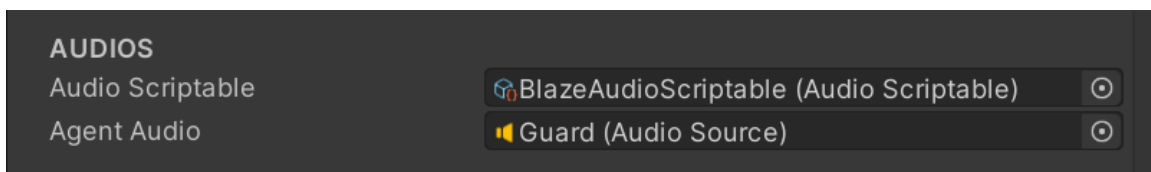
You can also choose to manually set another audio source to your liking that the AI will be using.

Vision & Detecting Enemies

Simply go to the main Blaze AI inspector -> Vision tab and follow the steps.

Set the *Layers To Detect:* this must contain all the layers you want the AI vision to check for like obstacles, walls, etc... **Any game object with a layer that isn't added here will be seen through.** No need to add the enemy's layer here.

P.S: for better performance, make sure you set the layers that are used as obstacles, ground, etc... Don't set to Everything.
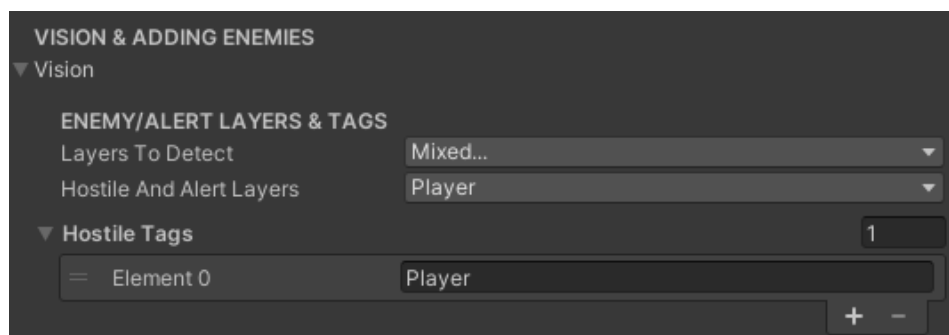
Set the *Hostile and Alert Layers:* this should contain the layer(s) of the enemy. So add the enemy's layer here. As seen in the picture.

In **Hostile Tags**, set the tag name of your player or whatever it is you want the AI to identify as hostile.

Now in the Blaze inspector in General tab there are two properties. **Center Position** and **Show Center Position**. Click on *Show Center Position* to enable it and look in your scene view.

You'll find a red sphere has appeared. Use the *Center Position* property to offset this red sphere to the AI's pelvic or torso area. This is where the vision ray will start. It looks like this:

It is also the minimum point of detection. So any target below this point won't be detected. Here is an example:

Area of detection
if max sight level not set

Area of detection
if max sight level set
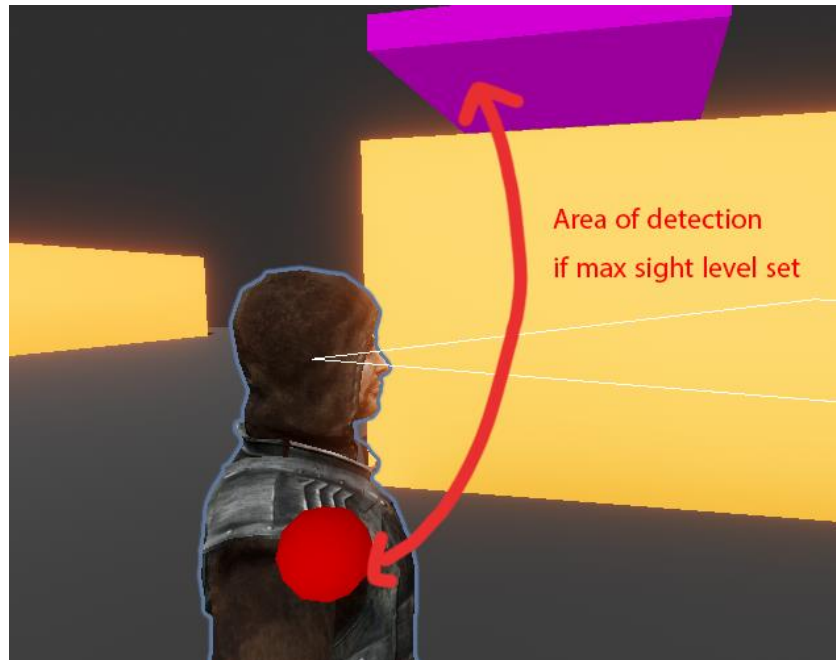
Set the vision cone on the eyes using the *Sight Level* property.

Finally, your enemy needs to have at least a character controller or one collider to be detected. Multi-colliders are ok. And that's all you need to detect an enemy.

Set the vision angle and range of your AI for each state using these properties: **It's always best to have the attack state cone angle at 360 to detect the enemy in all directions.**

SET THE VISION RANGE AND CONE ANGLE FOR EACH STATE
▼ Vision During Normal State
    Cone Angle   90
    Sight Range   10
▼ Vision During Alert State
    Cone Angle   100
    Sight Range   15
▼ Vision During Attack State
    Cone Angle   360
    Sight Range   20
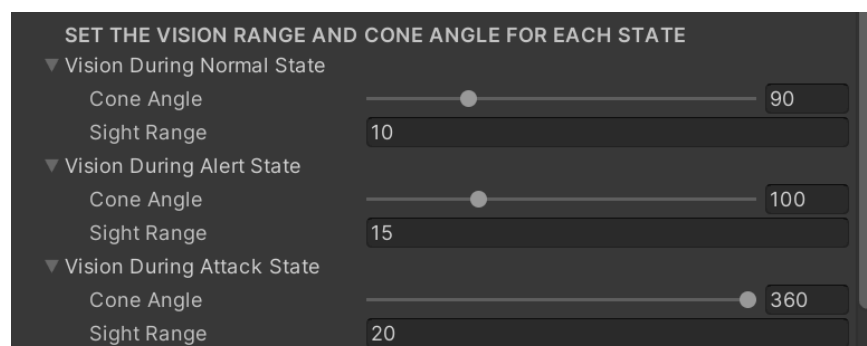
*If you're making a VR game. Then disable **Multi Ray Vision** property in Vision class as it may cause issues.*

In the DEBUG section below, you can choose to show the vision cone for each state and it'll appear in the scene view.
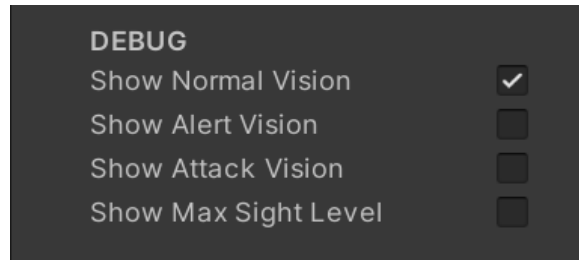
Will appear in the scene view as so:
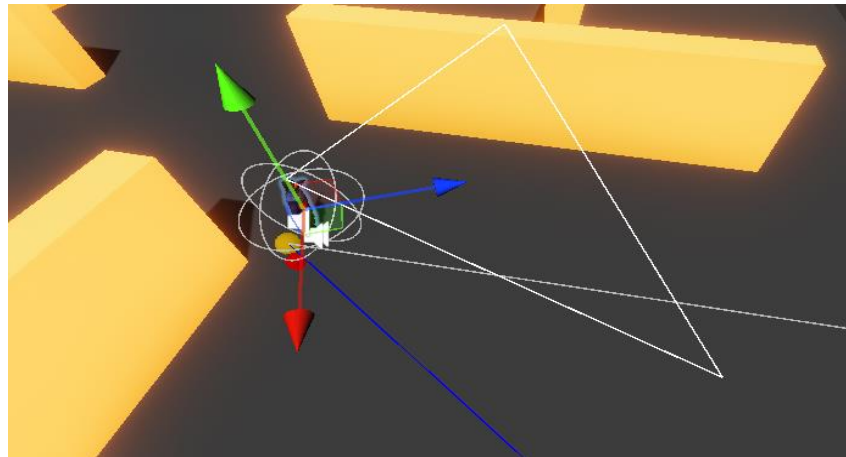
Setting Up Melee/Ranged AI

Follow these steps to set up your melee/ranged AI.

- Go to the main Blaze AI inspector > States tab > click on **Add Behaviours** button.

- You will find that scripts have been added to the game object and set in the state behaviours in States tab.

- *If you're not going to use cover shooter mode then go ahead and remove the added Cover Shooter Behaviour and Going To Cover Behaviour scripts from the game object.*

- Make sure **Cover Shooter Mode** is turned off in the States tab.

- Now expand the **Attack State Behaviour** script and make sure you're in the *Movement tab*.

- Set the **Move Speed** & **Turn Speed** properties to whatever you like. The former is for how fast you want the AI to be in Attack State and this includes chases, moving to target location when called & closing in the distance to launch attack on target. The latter applies to the same mentioned it's just how fast the AI turns to path corners.

- The **Idle Anim** is the animation name to play for the AI in it's combat stance, waiting for it's turn to attack. Set that.

- The **Move Anim** is the movement animation name to play when your AI is chasing the player, getting called by another AI to target location OR moving to the player to launch an attack.

- *Now go to Attack tab.*

- **Ranged** property adapts calculations to make the AI behave like a ranged enemy. So *set to **true** if you want the AI to be ranged* or ***false** if the AI is to be melee*.

- **Distance From Enemy** property is the distance between the target and the AI where the AI will wait for it's turn to attack in it's combat stance *(Idle Anim)*. Also if the distance between the AI & target becomes more than that set in this property the AI will chase it's target to close the distance.

- **Attack Distance** is the distance you want between the AI & it's target during the actual attack. So for example if your AI is melee you want this to be quite small like 0.5 - 1. Because the AI needs to get nearer to target. When triggered to attack, the AI will move to close the distance and then launch the attack when the distance is met.

- **Layers Check On Attacking** is to be used more with ranged. This is for AIs to avoid hitting each other. So set this to the layer of the other AIs if you want to avoid friendly fire.

- **Attacks** is a list (which can be changed dynamically) where you set each attack animation name, the duration of each attack and whether you want an audio to accompany each attack. *One attack will be chosen in random on each attack cycle.*

- **Attack Events** is where you can set events to get triggered when launching the attack. Alongside the attack animation.

- **Attack In Intervals**: let's first understand why this is here. By default the AI adds a script called **BlazeAIEnemyManager** to any hostile it sees. This script manages the attacks of each AI to make them attack one by one. By default, the call time of this script is 5 seconds. So every 5 seconds a random AI is called to attack the target. This can be changed via inspector, it's a public property. You can add this script in editor time to your player and change the value. But, what if you want all AIs to attack with no regard of such limitation? This is where this property comes in. Enabling this will open up another property were you can set two values of min and max. A random number between them will generate on each cycle as a wait time and when finished the AI will attack no matter what. Even if there is another AI about to attack. This is also known sometimes as burst attacks. For a constant time set the two values to the same number. This is good for enemies like zombies where they all attack together no matter what and have a small wait time between each attack. Good for ranged AIs too*.*

Last thing to mention is the damage system is up to you. You need to make your own health for the AIs (check Health section in this docs & health script in Hit & Death demo) and the actual damage system (check Hit docs in this docs). In order for the AI to detect the actual targets check the Detecting Enemies section in this docs.

Setting Up Cover Shooter AI

Follow these steps to set up your cover shooter AI.

- Go to the main Blaze AI inspector > States tab > click on **Add Behaviours** button.

- You will find that scripts have been added to the game object and set in the state behaviours in States tab.

- *If you're not going to use the Attack State Behaviour which is used for melee & ranged then go ahead and remove the added Attack State Behaviour script from the game object.*

- Make sure **Cover Shooter Mode** is turned **on** in the States tab.

- Now expand the **Cover Shooter Behaviour** script and select the **Movement tab**.

- Set the **Move Speed** & **Turn Speed** properties to whatever you like. The former is for how fast you want the AI to be in Attack State and this includes chases, moving to target location and from/to cover. The latter (turn speed) applies to the same mentioned it's just how fast the AI turns to path corners.

- The **Idle Anim** is the animation name to play for the AI when it's waiting for it's shoot cycle.

- The **Move Anim** is the movement animation name to play when your AI is chasing the player, getting called by another AI to target location, moving to/from cover.

- **Now select the Attack tab.**

- **Distance From Enemy** is the max distance that the AI should keep at all costs. If the distance increases than what's set in this property the AI will chase it's target to close this distance then find cover in that proximity. The AI eliminates covers that are farther than this set value.

- **Attack Distance** is the actual distance the AI should move to during it's shoot cycle. Ask yourself, how near do you want the AI from it's target when it's shooting at it. As a cover shooter, it's always best to keep this the same value as *Distance From Enemy*. But you can definitely set it to anything else.

- **Layers Check On Attacking** is for avoiding friendly fire. Before launching the shoot cycle the AI will fire a ray and check if any of the layers set in this property have been hit. If so, the AI will refrain from attacking until there is a clear path then it'll attack. You can set this property to the layers of other AI agents. So before an AI attacks it'll check if any of it's friends are in the way and stops if so. When the path is clear, it'll resume the shooting.

- **Shooting Anim** is the actual shooting animation name you want to play during shooting.

- **Shoot Every** is the interval of time in seconds for the AI to trigger the shoot cycle. Ask yourself: every how many seconds do you want this AI to shoot? This generates a random number between min and max values on each cycle. When the value is generated and the timer is done the AI will move to the player and shoot. For a constant time, set the two values to the same number.

- **Single Shot Duration** set the amount of time in seconds for the single shot. The shoot cycle consists of several shots.

- **Delay Between Each Shot** the amount of time as delay before the next shot is triggered.

- **Total Shoot Time** the overall shoot cycle time. For how long do you want the AI to keep shooting at the enemy. Takes a number between min and max values and generates a random one in between. For a constant time, set the two values to the same number.

- **Attack Event** is where you set events that get triggered on each shot. This is where you should stick your bullets function that plays particle systems, damages enemies, fire a ray cast, etc…

Everything else is self-explanatory and has tooltips (hover by mouse) that give more info.

Last thing to mention is the damage system is up to you. You need to make your own health for the AIs (check Health section in this docs) and the actual damage system (check Hit docs in this docs). In order for the AI to detect the actual targets check the Detecting Enemies section in this docs.

Setting up covers for cover shooter AI

Before following these steps make sure you have baked a navmesh on your plane. If not, follow the getting started section.

- Create your cover game objects or insert your cover models and give them a box collider.

- Then open the Navigation window using **Window** > **AI** > **Navigation**.

- Now that you've opened the navigation menu. Click on all of the cover game objects in the hierarchy and go to the **Object tab** and check on **Navigation Static** and set *Navigation Area* to **Not Walkable** like so
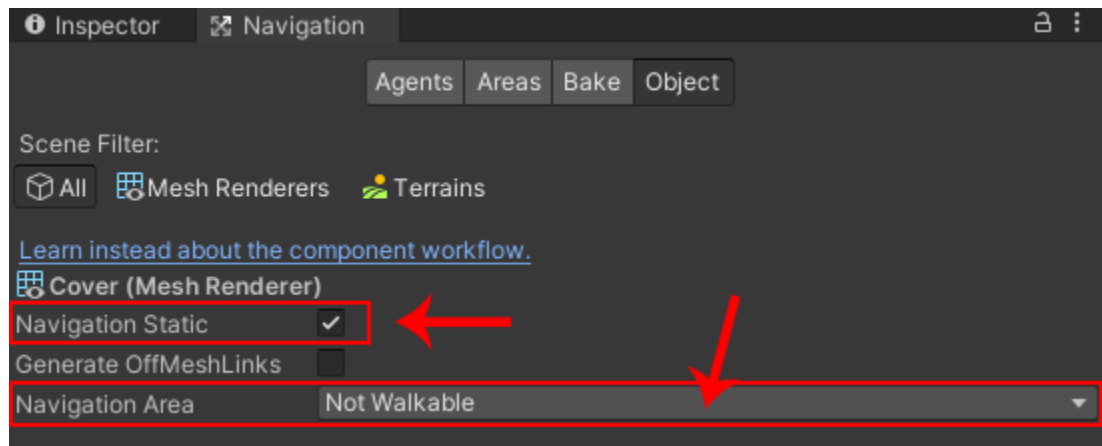
- Now go to the **Bake tab** and click on Bake.
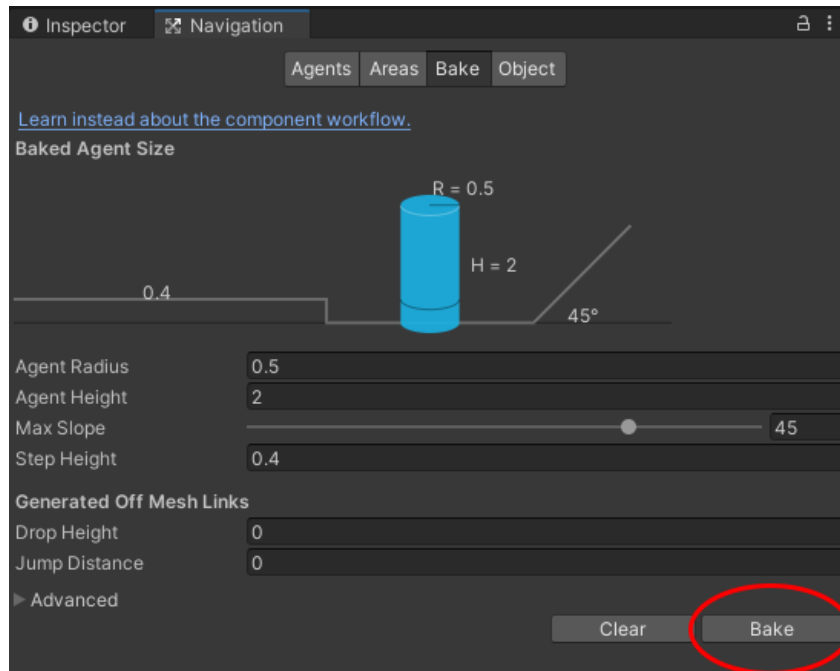
- Now you'll find in the scene view that your covers have carved the navmesh around them as so.
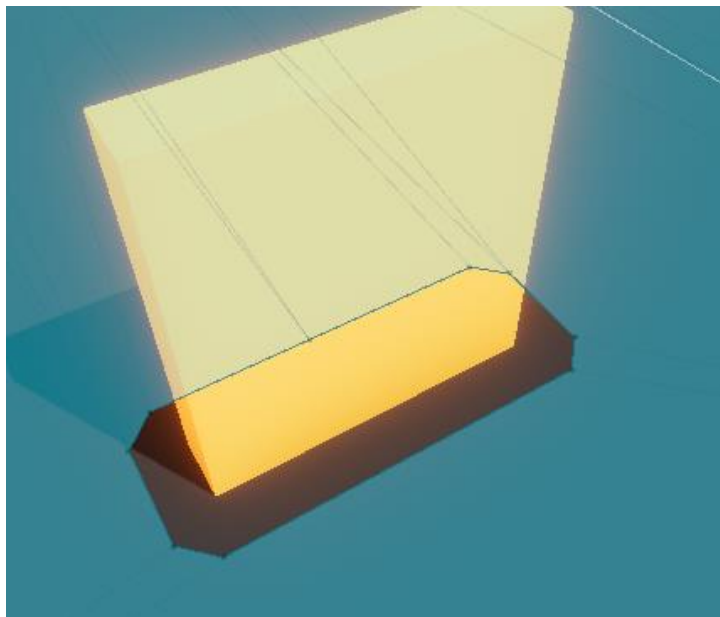
- Last thing for covers setup is giving your covers a layer. It's a lot better to give your covers a unique layer and not set it to Default.

- Now that you have set up the covers it's time to go back to the AI. Expand the **Going To Cover Behaviour** script.

- Set the **Cover Layers** property to the layer of covers.

- **Hide Sensitivity** is how good you want the hiding spot to be. It's very rare that you'll be editing this. So just leave it as is as -0.25.

- **Search Distance** is the distance around the AI to search for covers. Enable **Show Search Distance** below it and in the scene view the radius will be shown as a light blue wire sphere so you can visually assimilate how good the distance is.

- **Min Cover Height** is the minimum height of cover to be eligible for cover. To get the cover height of any game object simply add component **BlazeAIGetCoverHeight** and click on the button. It'll show you what height this game object is. *It needs to have a collider.*

- **High Cover Anim** the animation to play when the AI is hiding behind a cover with a height bigger than that of *Min Cover Height* property.

- **Low Cover Anim** the animation name to play when the cover height equals that of *Min Cover Height*.

- **Rotate To Cover Normal** if checked, when the AI reaches the cover it'll rotate so that it's back is to the cover.

And that's it now your AI can perfectly detect covers and go hide behind them. Everything else is self-explanatory and has tooltips (hover by mouse) that give more info.

Alert Layers/Reacting To Certain Tags

If you've read the previous <u>Adding Enemies</u> part you may be asking, what are Alert Layers?

Alert Layers are the layers of Alert Tags (found below Hostile Tags in Vision class). Alert Tags are game objects with tag names that you want the AI to react to and turns the AI to **Saw Alert Tag** state. The AI will trigger the alert vision and play the alert state's move animation. **Check Demo 3 (full version only).**

You start by adding the tag name of the game object you want to react to in the Alert Tag property inside the Alert Tags class in Vision.
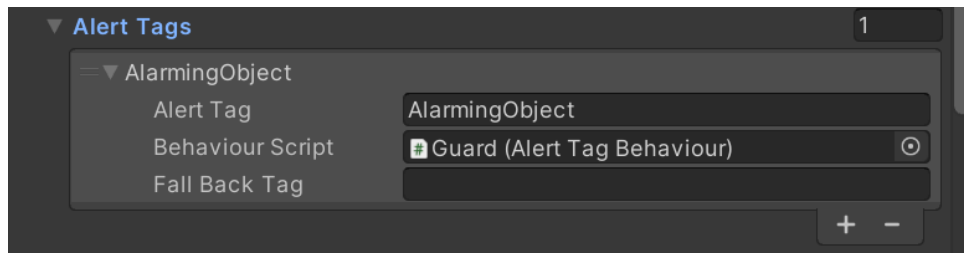
As you can see I want the AI to react and get alerted by any game object with the tag name **AlarmingObject**.

The second property is the behaviour script you want to enable when this tag name is seen. You can set different behaviour to each alert tag. Blaze comes with a default script with numerous properties to customize the behaviour.

Simply add component to your AI: Alert Tag Behaviour
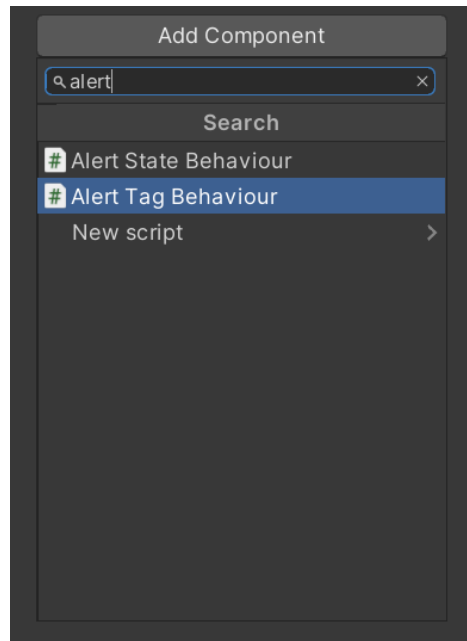
Then after adding the component, drag/drop to the **Behaviour Script** property in Alert Tags.

Last thing is the **Fall Back Tag** property. When the AI sees a game object with an alert tag it immediately changes the object tag to the fallback tag in order not to have itself or other AIs reacting to it all over again. Whatever you set in this property will be the new tag name of the game object. By default, if you leave this property empty the game object will fall back to "Untagged".

That's it! Now when the AI sees an alert tag the behaviour script will enable and when the duration ends. It'll return to Alert State.

Hit

First of all, to enable the AI to turn to the hit state you need to add it's behaviour first. Because like any other state, a state needs a behaviour that instructs it how to act in that specific state. Really simple. Example demo to study is Hit & Death demo.

You start off by going to the Hit tab in Blaze and clicking the Add Behaviour button. This will add the Hit State Behaviour script component to your AI game object and set it to it's property.
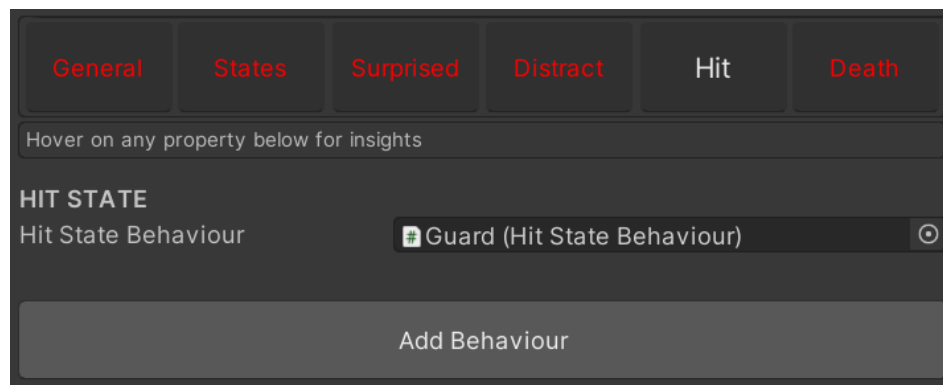
This is the script after being added to the AI.
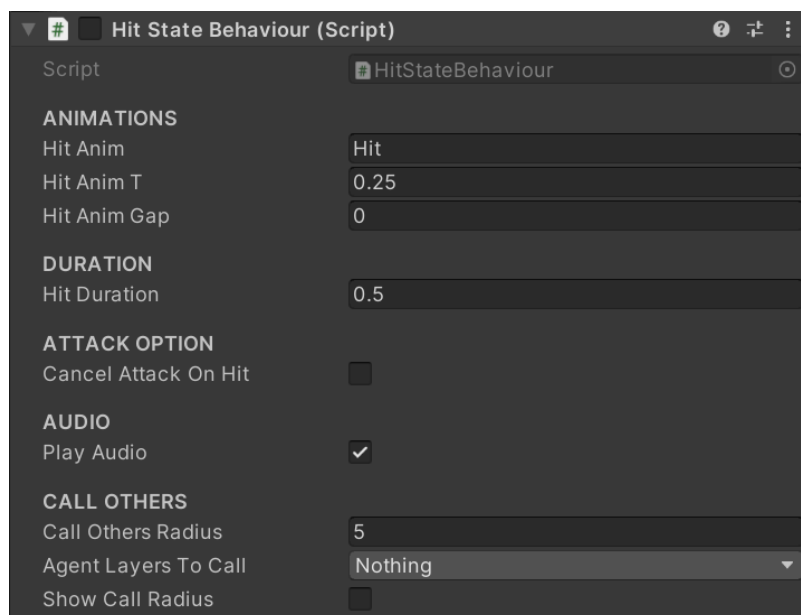
Like other behaviours, the script will be disabled. That's good.

For turning the AI to the hit state, you simply call the Blaze public method *Hit(GameObject hitter = null, bool callOthers = false)*.

This will turn the AI to the hit state and play the hit animation and audio and stay in that state for the time set in the **Hit Duration** property *(in Hit State Behaviour script)* and then exit from it.

The API as seen above has only two parameters:

hitter **parameter:** If *hitter* is passed, upon exiting the hit state the AI will turn to attack state and move to the location of *hitter* and check out the location. If not passed, upon exiting hit state the AI will turn to alert and continue patrolling. It's by default set to *null*. **You can use this to set whether you want the player attacks to be anonymous or not.**

// AI will move after hit state to check location

blaze.Hit(player.gameObject);

// AI will turn to alert state after hit state and continue patrolling

blaze.Hit();

Both are valid but depends on what you're trying to do.

***callOthers* parameter:** If set to true, it'll call the nearby agents. if *hitter* is passed as well, it'll call AIs to the hitter location. If *hitter* isn't passed, it'll call AIs to it's very own location. **Configure the call radius and agent layers from the inspector of Hit State Behaviour.**

// will hit AI and call other AIs to player location

blaze.Hit(player.gameObject, true);

// will hit AI and call other AIs to current AI location

blaze.Hit(null, true);

Health

What about health you're asking? well, health is up to you. That means you can use the one that already comes with the asset. You can find it in the Hit & Death demo. Or you can write your own or use your favorite health asset.

This also means that you need to write another simple script that calls the **Hit()** of Blaze and decrease the points of your health script. Let's check all this out one by one.

First of all a health script is as simple as:

```csharp
using UnityEngine;

public class Health : MonoBehaviour
{
    public int healthPoints = 100;
}
```

This health script or similar should be on your AI. The idea is on attacking you decrement this **healthPoints** variable by the amount of attack damage.

Then you should have some sort of a hit manager function **on your player** in another script that handles both of decrementing the health points and calling the Hit state of Blaze.

Something like:

```
public void HitAI(int damagePoints)

{

    blaze.Hit(gameObject);

    healthScript.healthPoints -= damagePoints;

}
```

So this would be on your player and this is what you call when your player hits the AI.

Death

You don't have to setup any behaviour for this. You simply call the public blaze method *Death(bool callOthers = false, GameObject player = null)*. **Check the Death demo.**

Setting up the properties for death is in the Death tab in the main Blaze AI inspector.
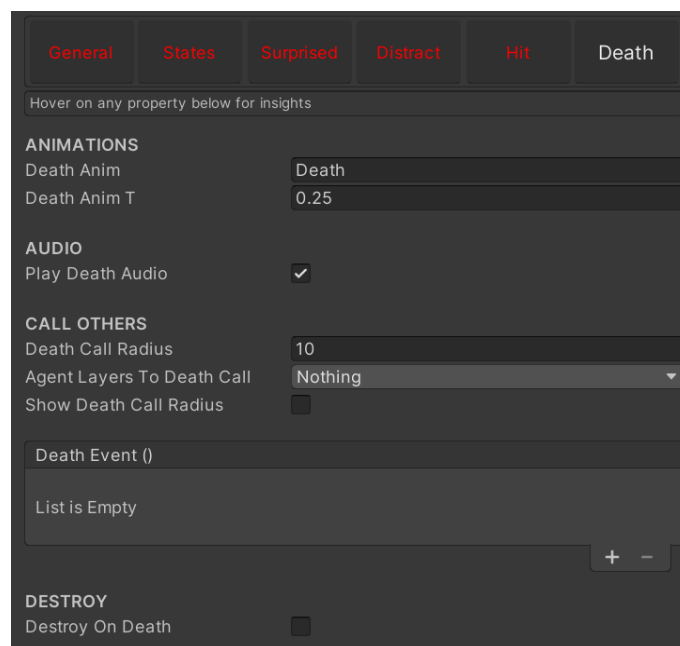
Parameters:

Death() method takes two parameters as seen above and both are **optional** and have default values of **false** and **null** respectively.

callOthers: is whether you want to call other AIs to a location when AI is dead. You pass either **true** or **false**. False is default. Configure the radius and layers from the inspector.

player: being the player game object to send other AIs to check it's location if the first parameter (callOthers) is true. It takes a **game object** and is by default set to **null**. **If this is not passed while callOthers is true the AI will call others to it's own location.**

// will kill AI and call other AIs to current location

blaze.Death(true);

// will kill AI and call other AIs to player location

blaze.Hit(true, player.gameObject);

You can also have a script on your AI that checks for health points. If it's less than or equal to 0 then call Death().

```
void Update()

    {

        if (healthScript.healthPoints <= 0) {

            // this won't call others since nothing is passed

            blaze.Death();

        }

    }
```

Of course the best way to do this is having healthPoints be a property to begin with and check for the health points inside the setter. But this way works too and it's easier if you're a beginner.

Distractions

First of all go to the Distract tab in Blaze AI inspector.

Enable **Can Distract.** Then click on the **Add Behaviour** button.
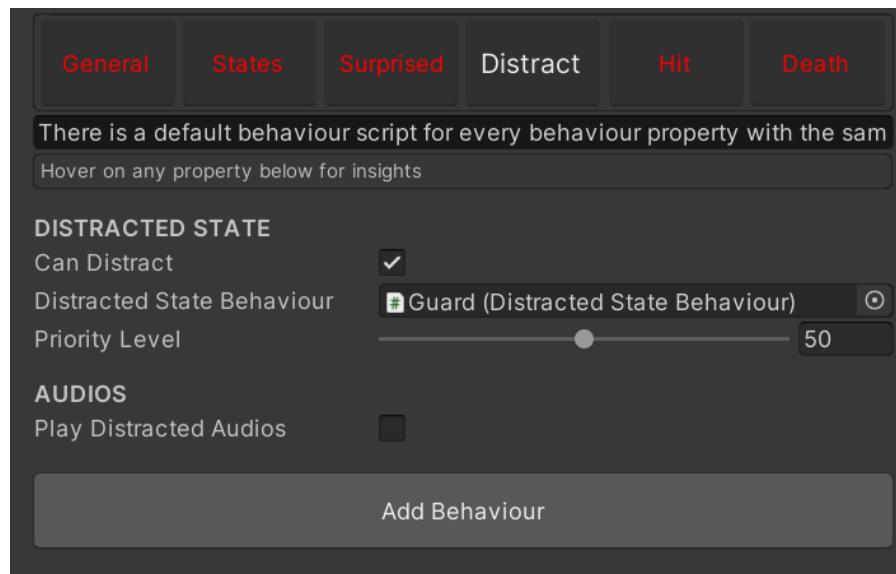
This will add the default distracted state behaviour to your AI and set it to it's state property.

The added script is how you want AI to act when distracted. So fill it as you want.

Now there are two ways to distract an AI.

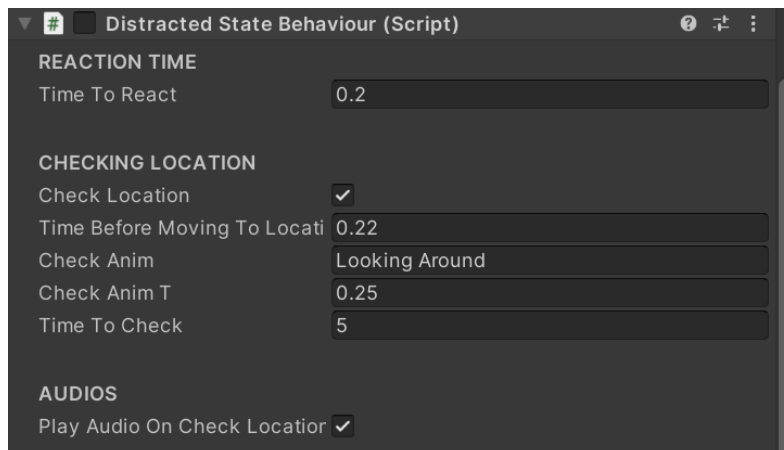You can either call the *Distract(Vector3 location, bool playAudio)* method on a single-specific AI by doing:

blazeScript.Distract(locationOfDistraction, true);

The first parameter is the location of the distraction.

The second parameter is whether the AI should play audio or not.

The second and most preferred way and also **the way to distract groups** is using the script: **Blaze AI Distraction**. Add this script to any game object/location you want to be the distraction.
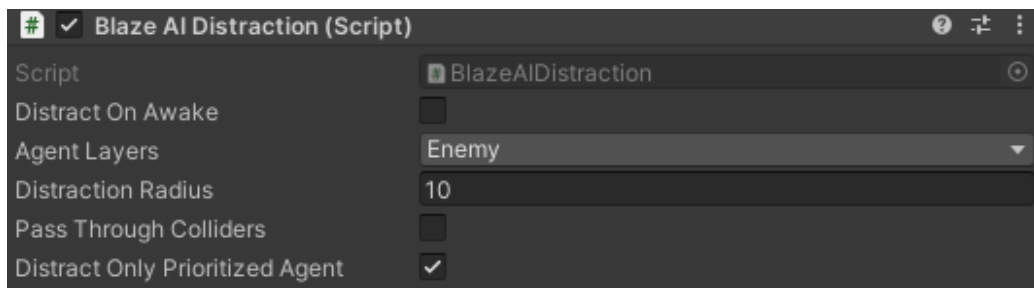
After setting the properties (we'll get to them) you trigger the distraction using the public method of the script: ***TriggerDistraction()***. So you can do:

BlazeAIDistraction script = GetComponent<BlazeAIDistraction>();

script.TriggerDistraction();

This is the preferred way as it does most of the work for you and also enables several AIs to be distracted.

Or you can trigger the distraction on awake using the first property you see in the image.

Now for the properties:

Agent Layers: the layers of the AIs you want to distract.

Distraction Radius: the radius that will check and distract AIs. You can see the distraction radius in the scene view as a yellow wire sphere that changes with the value of this property. So you can visually see how big the distraction radius will be. As so:
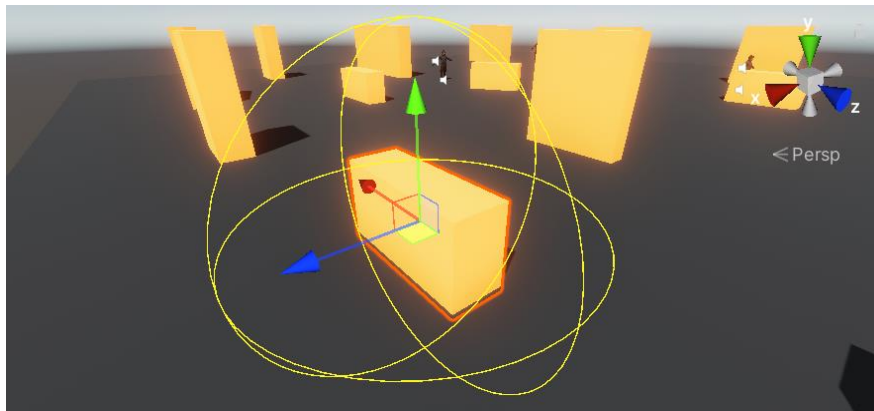
Pass Through Colliders: do you want the distraction to pass through walls and objects and distract the AI or not?

Distract Only Prioritized Agent: If enabled, only the AI with the highest distraction priority in a group will turn and face the distraction. If disabled, the entire group will face and look at the distraction but only the highest priority will check the location.

For seeing how distractions work in action check the Distractions demo.

Distance Culling

This is a great system to drastically improve the performance of your game by disabling the AIs that are far away from the player or camera and re-enabling them when in range. It's also extremely easy to setup. **Take note: this also has APIs so check the APIs section.**

Start off by creating an empty gameobject in your scene and adding to it the **Blaze AI Distance Culling** script.
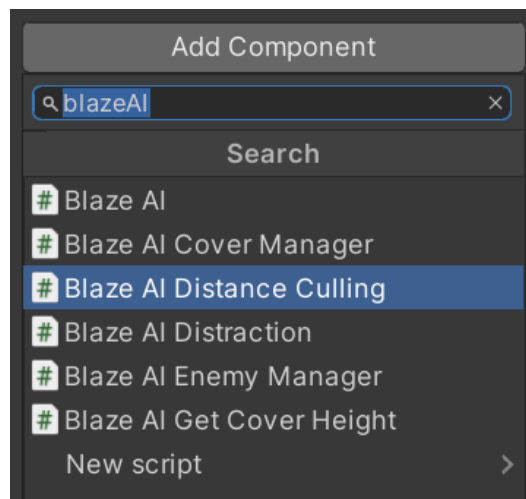
This will be the added script on your empty gameobject.
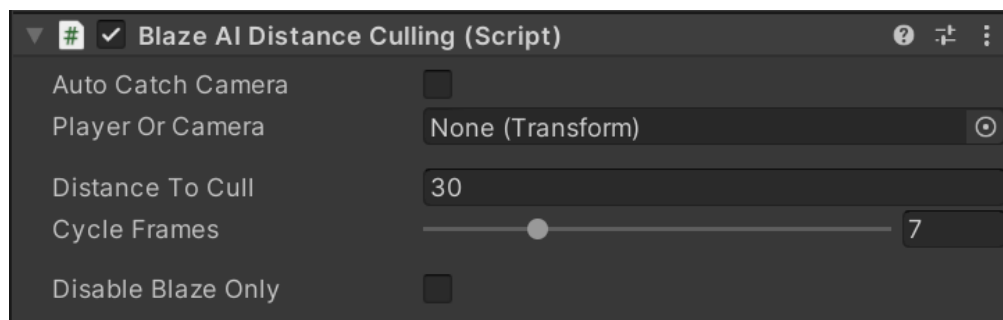
Distance culling uses either the player or the camera to calculate the distance to the AIs and it's up to you to choose which one is best depending on your game.

If the camera is best you can simply enable **Auto Catch Camera.** This will get the main camera on awake. So no need for any more manual work.

However, if your **camera gets spawned during runtime**. Disable Auto Catch Camera and when the camera is ready and spawned have a function in any script that does:

BlazeAIDistanceCulling.instance.playerOrCamera = Camera.main.transform;

If you want to compare the distance to the player character. You can disable Auto Catch Camera and stitch your player transform to the property **Player Or Camera** that appears *as seen in the image of page 21.*

Distance To Cull is the distance you want to disable the AI when it exceeds this value.

Cycle Frames is a way to further improve performance by running the culling cycle every set frames. **By default it runs every 7 frames**. You shouldn't play with this unless as a last resort for bad performance in your game.

Disable Blaze Only will disable blaze component only and play the first idle animation of either normal or alert state. Depending on which state you set to use on Awake. Enabling this will make your culling look more natural with no pop-ins since it doesn't disable the game object. (will make your AI look as if it's waiting)

Now you have set the "central manager" for the distance culling. What's left is instructing the Blaze AIs to use this feature.

Go to your AI, open up the Blaze inspector in the General tab. Scroll down and enable **Distance Cull.**
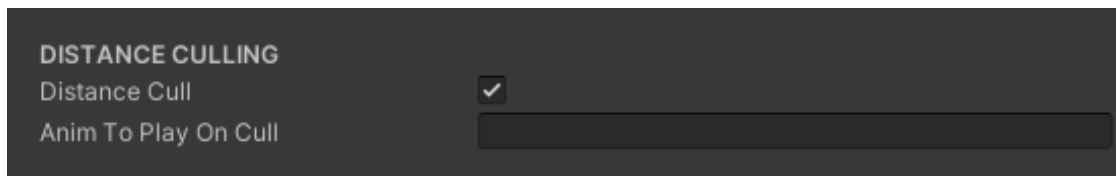
When you enable **Distance Cull**, a new property appears *Anim To Play On Cull*. In this property set the animation name you want to play when the AI is culled. **This will only be considered if *Disable Blaze Only* is enabled in the main culling manager that we saw previously. You can also leave it empty if you don't want to set an animation.**

And that's it! Now as you play the game and move your player or camera further away from any AI, it'll disable that AI and re-enable when in range.

Companion Mode

Using companion mode you can turn any Blaze AI to your friend and make it fight alongside you. You can even use it to make an AI a companion of another AI.

Start by going to the companion tab and click on **"Add Companion Behaviour"**
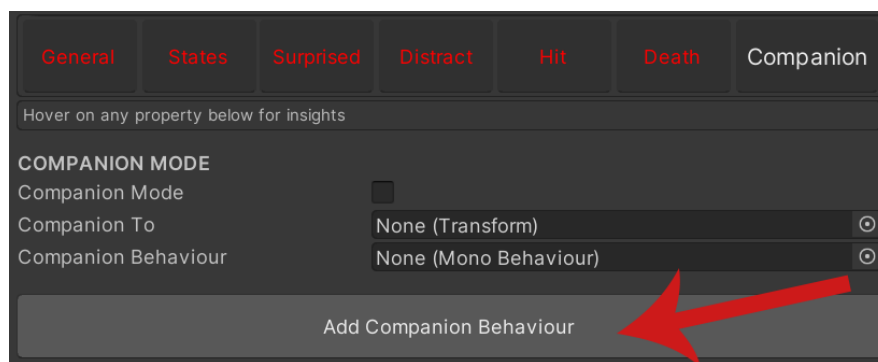
You will notice 2 things after clicking.

First is that the *Companion Mode* property has been set to true. **Second** is that the *Companion Behaviour* property has been set to a newly added script.

Next is drag and drop your **Player** (or any other) to the *Companion To* property. This means setting which game object you want the AI to be a companion to.
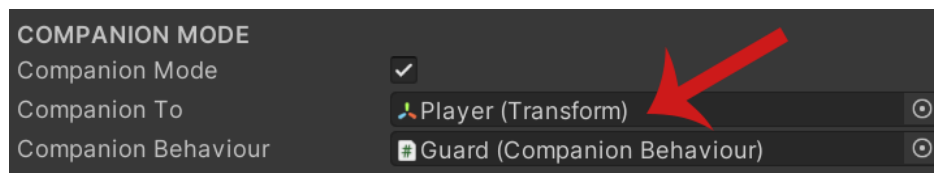
This also works in runtime by the way. So you can change it dynamically.

Now we are done with the main inspector part of things. Now go to the newly added script **Companion Behaviour** to set the properties**.**
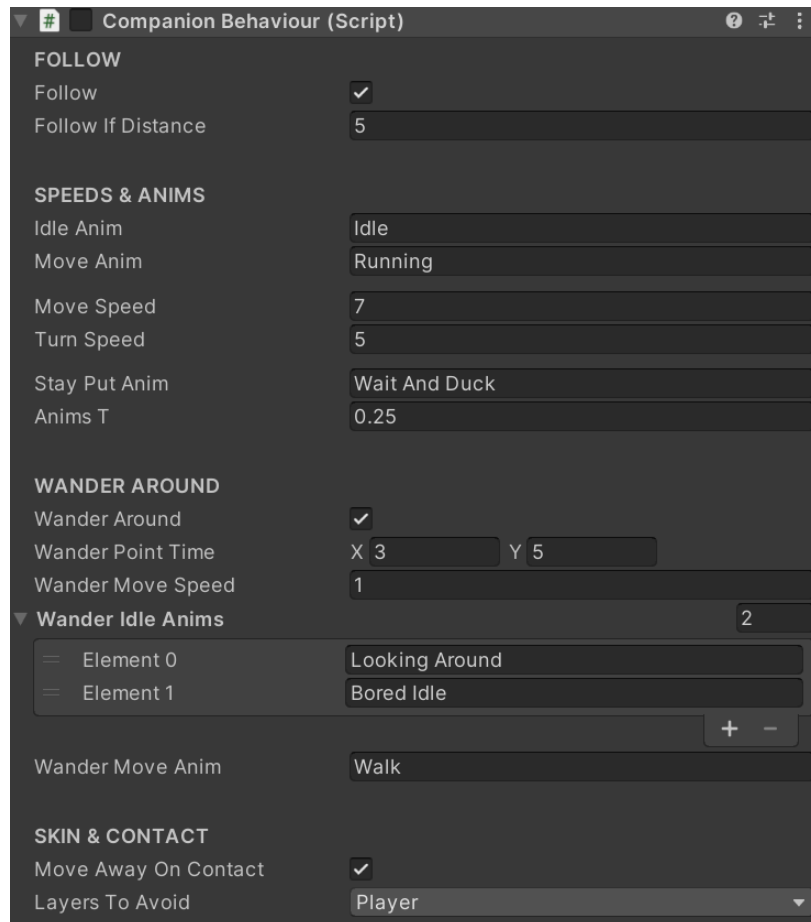
Properties:

Follow : by default it is set to true and this means the AI will follow the companion. If set to false, this means the AI has been commanded to stay put and the AI will oblige and stay put while playing the Stay Put animation indefinitely until follow is back to true again.

Follow If Distance : Set the distance that when exceeded the AI should follow while playing the Move Anim. When the distance is less the AI will stop and play the Idle Anim.

The animation and speed properties are self explanatory. So I will not be covering them.

Wander Around : Using this option you can make the AI wander around the player when the distance has been meet. This can add more life to your game as the AI will be wandering around. The wander radius is the AIs navmesh agent height * 2 + 0.5.

Wander Point Time : The amount of time to spend in each wander point before moving again. A random value will be generated between the two set values for each wander point. For a constant time set the two (X & Y) values to the same value.

Move Away On Contact : Setting this property to true will make the AI move to a random position when it comes to contact with the below set layer. *You can use this to make the AI move when the player comes in contact.*

Layers To Avoid : When the AI comes in contact with any of these set layers it will move to a random position. *You can use this to make the AI move when the player comes in contact.*

Audios
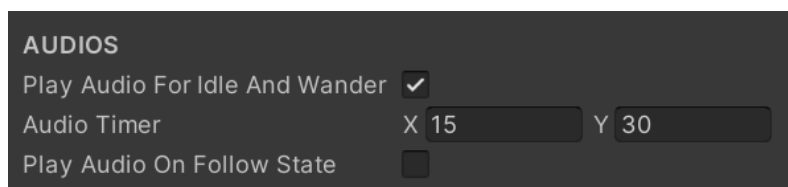
Play Audio For Idle And Wander : Make your companion play a random audio every set time (Audio Timer) when idle or wandering. Set the audios in the audio scriptable. *Check Audio section in this docs to know what the audio scriptable is.*

Audio Timer : A random time will be chosen between the two values on each cycle. For a constant time set the two values to the same number.

Play Audio On Follow State : This will make the companion play an audio when the follow state gets changed. In the audio scriptable you will find the two properties: **Companion On Follow** and **Companion On Stop.** In these you can set audios for each one. So for example if you set the follow state to false. You can make the AI say "Alright, I'll stay here" as if it's being commanded and when follow gets set to true it can go like "I'm on my way".

How the companion system works?

Simply put: when Companion Mode is enabled the Companion Behaviour will run in normal & alert states instead of their default behaviours. As for the attack state, the default attack behaviour runs in order to make the companion attack other enemies alongside it's companion.

Companion APIs?

The companion AI supports the APIs of any other Blaze AI. You can use the APIs to call it to a certain location, set it to attack an enemy, etc… Please check the APIs section in this document and use the API you like.

Public Properties & APIs

All classes/variables in the inspector can be accessed programmatically using the camel-case convention. Making only the first letter small case.

For example:

Use Root Motion -> blaze.useRootMotion

Sight Level inside of Vision class *-> blaze.vision.sightLevel*

Here are handy public properties & APIs (methods) to call:

Properties

state – returns a **State** enum of either:

State.normal

State.alert

State.attack

State.goingToCover

State.distracted

State.surprised

State.sawAlertTag

State.returningToAlert

State.hit

State.death

enemyToAttack – returns the game object of the enemy the AI is targeting.

agentAudio - Returns the main **AudioSource** of the AI that Blaze uses (dynamically generated on start) to play the audios.

distanceToEnemy - Returns the distance (float) between the AI and the targeted enemy. Use this with enemyToAttack to make sure there is a targeted enemy in the first place.

isAttacking - Returns true (bool) when the AI is moving to position for an attack or is already attacking.

sawAlertTagName - Returns the name (string) of the seen alert tag.

sawAlertTagPos - Returns the position (Vector3) of the seen game object with alert tag.

APIs

MoveToLocation (Vector3 location, bool randomize = false) - Force move the agent to any location. The second parameter is whether you want to randomize location point within sphere. This means the AI will go to the location passed but with a little offset. This is especially useful if you're moving several AIs to the same location. To avoid them stopping at the exact same point. **This method can't be used if the AI is in attack state or going to cover state.**

IgnoreMoveToLocation () – Use this to ignore the previous forcing of moving to a location.

StayIdle () – Force the AI to stay idle and then move again after idle time has finished. **This method can't be used if the AI is in attack state or going to cover state.**

IsIdle () – Returns a bool to check whether the AI is idle or not. Idle is when the AI reaches a waypoint and waits for the idle timer to finish before patrolling again.

Distract (Vector3 location, bool playAudio=true) – Distract a single-specific AI to a certain location. The first parameter being the location and the second is a bool for whether audio should be played or not.

Attack () – Force the AI to attack it's target. ***This can only be used when the AI already has a target or else what is it going to attack?*** Use this with the check ***enemyToAttack != null*** to ensure the AI has a target.

StopAttack () – Stop the AI attack.

ChangeState (string state) – Using this method you can change the AI's state between normal and alert only. The method takes a string of either "normal" or "alert" and will change the AI's state to that specific passed state.

SetTarget (GameObject enemy, bool randomizePoint = false, bool attackVisionForFrame = false) – Set an enemy for the AI and go check it's location. The second parameter is whether you want the AIs to stop at the location but with a little offset to avoid crowds of AI in the same exact point. If the third parameter is passed as true, the AI's vision will turn to that of attack state for a single frame and if it catches any hostile in that frame, it'll engage. This is especially useful for cover shooters where the AIs attack vision ignores obstacles and covers. So passing the second parameter helps the AI catch the enemy directly instead of going to it's location first.

Hit (GameObject enemy = null, bool callOthers = false) – Use this method to hit the AI. *For more information check the Hit section in this documentation.*

Death (bool callOthers = false, GameObject player = null) – Kill the AI. _For more information check the Death section in this documentation._

AddDistanceCulling () - This will add the AI to the distance culling manager.

RemoveDistanceCulling (bool enableObject = false) - Remove this AI from the distance culling manager. Takes an optional parameter. If passed as true, will also enable the game object if it's been disabled by the distance culling. Default is false.

CheckDistanceCulling () - Check whether this AI is added to the distance culling manager or not. Returns either true or false.

animManager.Play(animName, animTransitionTime) – using this public method you can play any animation you want even if Blaze is disabled.

Additive Scripts

These are extra scripts provided that increase the functionality of Blaze AI.

BlazeAIEnemyManager – this is added by Blaze when it sees a hostile game object, if it's already added to the hostile object then Blaze will not add it again. This script component is the enemy manager that makes the Blaze AIs attack the hostile one at a time. You can add this before-hand in editor time to be able to control the interval of attacks of AIs. ***Setting callEnemies property to false will prevent any AI from attacking the target.***

BlazeAIDistraction – add this script to any game object you want to act as a distraction. Trigger this distraction programmatically using *TriggerDistraction().* This is how it's triggered in full:

GetComponent<BlazeAIDistraction>().TriggerDistraction();

This will make any Blaze AI be distracted and look at the distraction trigger source direction. You can obviously within the function that calls TriggerDistraction() also play an audio. This will simulate or look like as if the AI has heard a sound and got distracted by it. Sound distractions, this is how it works in games.

BlazeAIGetCoverHeight – Add this script to any cover obstacle with a collider and click on Get Cover Height button and it'll print you it's height in the inspector. Use it with cover shooter setup to be able to set the high and low cover heights.

SetWayPointToPosition - Sets the waypoint of the AI to the current position. Add this script to your AI where Blaze is.

BlazeAICoverManager – Blaze AI that are in cover shooter mode add this script to any obstacle they're about to take cover in and set their transform in the occupiedBy property. When leaving the cover their transform from the same property mentioned earlier.

BlazeAIDistanceCulling - Add this script to any empty gameobject in your scene then enable distance cull in the Blaze inspector of the AIs. The AIs will disable when there distance is more than that set in the distance culling script.

Internal APIs for Behaviours

These are APIs in blaze that you can use when writing your own behaviour scripts. You will see all these methods in fact being used in the standard behaviours.

MoveTo (Vector3 location, float moveSpeed, float turnSpeed, string moveAnimName=null, float animT=0.25f, string direction="front") -> Moves AI to location. **Returns false while moving to location and true when AI reaches location.**

location: destination to move to.

moveSpeed: movement speed while moving to destination.

turnSpeed: speed of rotation while moving to destination.

moveAnimName: name of the movement animation to play.

animT: transition time from current anim to move anim.

direction: sets the direction vector of movement. Takes either "front", "backwards", "left" or "right". By default set to front.

TurnTo (Vector3 location, string leftTurnAnim=null, string rightTurnAnim=null, float animT=0.25f, float turnSpeed=0) -> Turns the AI to a direction while playing animation. **Returns false while turning and true when turning is finished.** The method will automatically determine which turn anim to choose from (left or right)

    location: the location to turn to.

    leftTurnAnim: turning left animation name.

    rightTurnAnim: turning right animation name.

    animT: Transition time from current anim to turning anim.

    turnSpeed: the speed of turning.

RotateTo (Vector3 location, float speed) ->Will rotate the AI to location.

    location: the location to rotate to.

    speed: rotation speed.

NextWayPoint() -> _Sets the_ public **_waypointIndex_** property to the next waypoint index and **returns Vector3 of the destination.**

CheckWayPointRotation() -> Check whether the waypoint reached has a waypoint rotation or not. **Returns true if current waypoint has a rotation and returns false if not.**

WayPointTurning() -> turns the AI to the waypoint rotation and **returns true when done.**

SetState(State stateToTurnTo) -> sets the state of the AI.

      stateToTurnTo: the state you want the AI to turn to. Takes in (BlazeAI.State.normal, etc…) check the State enum in Blaze AI script.

Fleeing & Switching Behaviours

Blaze's design gives you the ability to switch behaviour scripts of any state at run-time. Giving more freedom to the behaviour of your AI.

So, for example, we want the AI to flee when it's health is below 30% and is in attack state. This will require a complete change in the attack-state behaviour. Which Blaze is completely fine with.

I will create a new script that will be the flee behavior for the AI that moves it to the location when enabled. This script will play a running animation and does:

// to move the character

navmeshAgent.Move(moveLocation);

I will add this new script to the AI and create another new script that tracks the health and switches the attack state behaviour to the new behaviour when health is less than 30%.

MonoBehaviour fleeBehaviour;

If (health <= 30) {

   Blaze.attackStateBehaviour = fleeBehaviour

}

With the swapping made to the attack state behaviour, Blaze will automatically disable the old script behaviour and enable the new flee behaviour script. Now, what drives the attack state is the new

behaviour which takes control. Moving the AI to the location far from it's target.

Another quick and easy form of this behaviour switching is enabling/disabling **coverShooterMode** property which will make the AI switch between Attack State Behaviour & Cover Shooter Behaviour in game time with no problems.
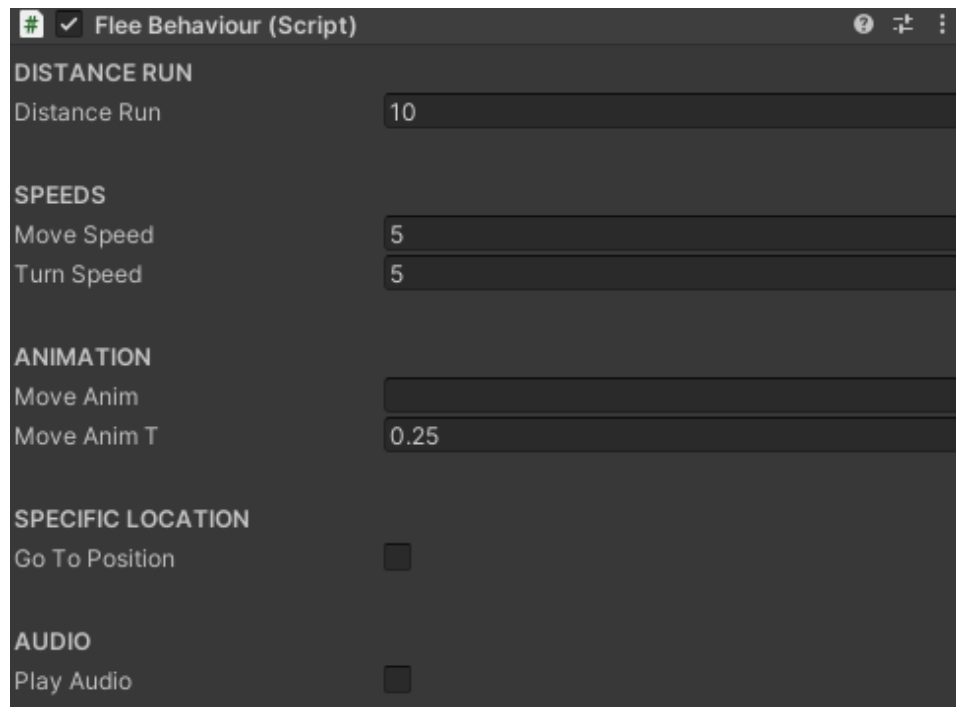
Fleeing

Now, it was important for you to read the first part of this section (Switching Behaviours) before getting to this part. So you know that you can change behaviours at run time and you can make an enemy AI that attacks naturally and then under certain circumstances make them flee.

That said, a much more advanced flee behaviour script already comes with Blaze that you can use.

Let's say, we want to make some sort of animal AI. When it sees the player it flees immediately. This is very easy.

- We start by adding the Flee Behaviour to the AI.

- Set the distance you want the AI to flee away in **Distance Run**.
- Set the **Move** and **Turn** speeds during fleeing.
- Set the running animation in **Move Anim**.

- If you want the AI to go to a specific location when fleeing rather than running randomly, you can enable **Go To Position**. A bunch of other properties will appear and you can set the position inside **Set Position**.
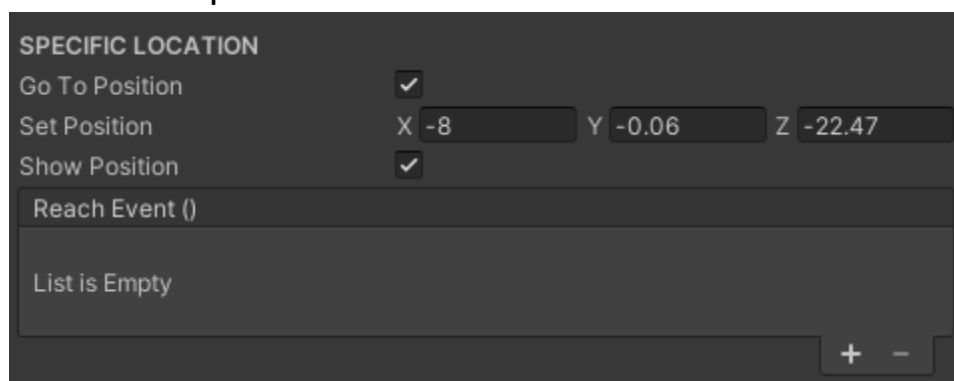
- You can also fire an event when the AI reaches that specific location using **Reach Event**.

- If you enable **Show Position** then you can easily see the **Set Position** location in the scene view as a green sphere. As so:

- Lastly, we need to replace the Attack Behaviour with the Flee Behaviour inside the Blaze inspector in the States tab. Simply drag and drop the flee behaviour to the property.
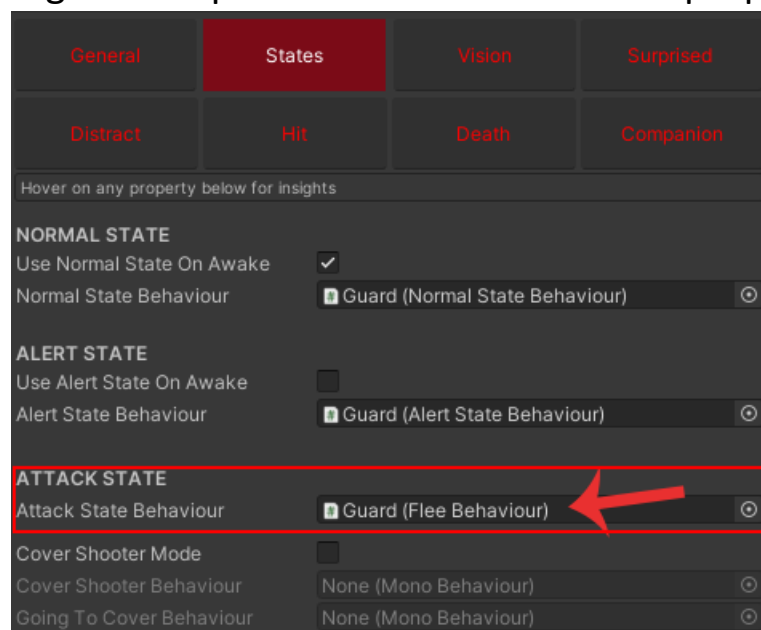
- Make sure you've set the hostile in the Vision class. If you don't know how then please read the **Vision & detecting enemies section.**

- Now when the animal AI sees the player it'll turn to attack state but the behaviour controlling the attack state is the flee behaviour. So the animal AI flees.