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 AssetStore Page here

About:

Blaze is a fast, modern and comprehensive enemy AI engine. If you have enemies in your game no matter the genre, Blaze will definitely be of service to you and your game. It's been inspired by AI systems of many modern games making Blaze a powerhouse in it's features and functionality with an easy workflow.

Index:

Video Tutorials:

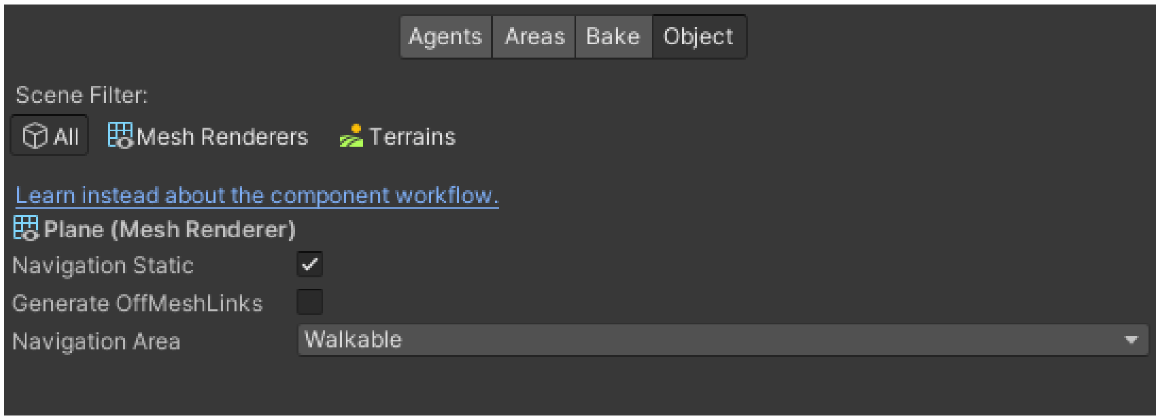
Making Distractions

Vision System

Getting Started:

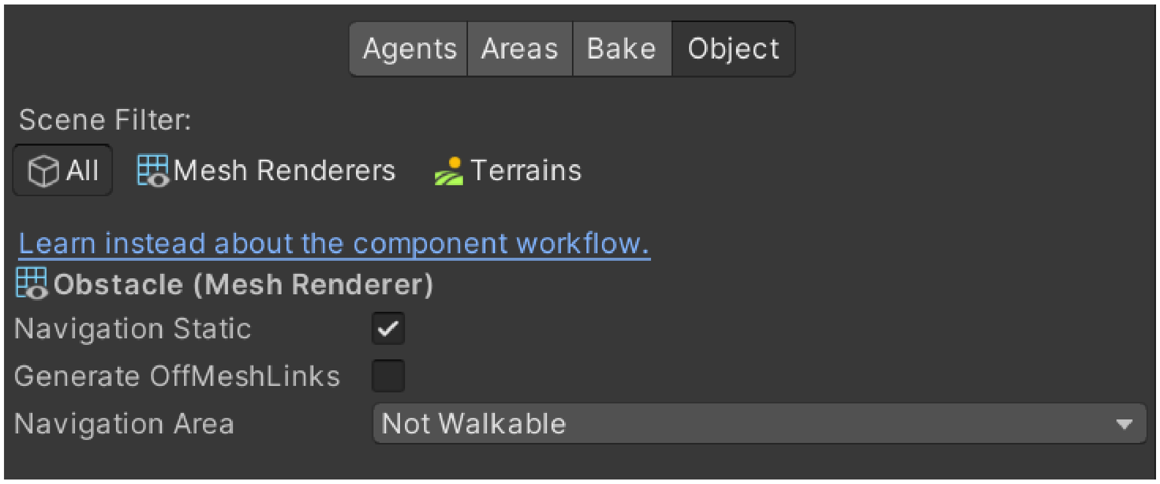
1. Add an empty Plane game object to your scene. This will be the ground the AI walks on.

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*.



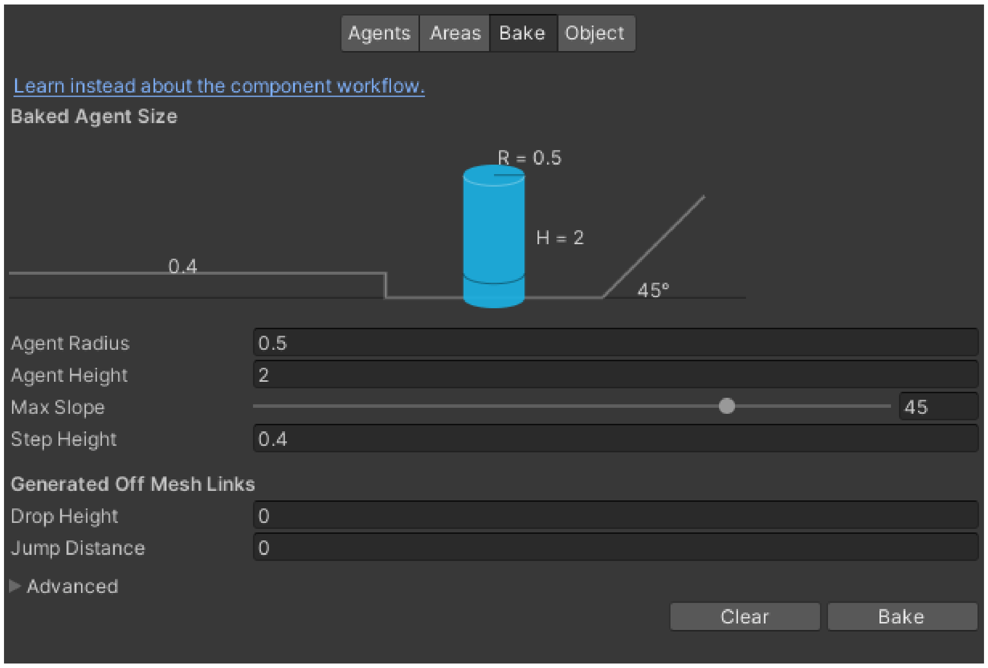
Plane (Mesh Renderer)
Navigation Static ✓
Generate OffMeshLinks
Navigation Area: Walkable

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.*



Obstacle (Mesh Renderer)
Navigation Static ✓
Generate OffMeshLinks
Navigation Area: 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.



Learn instead about the component workflow.

Baked Agent Size

Agent Radius	0.5
Agent Height	2
Max Slope	45
Step Height	0.4

Generated Off Mesh Links

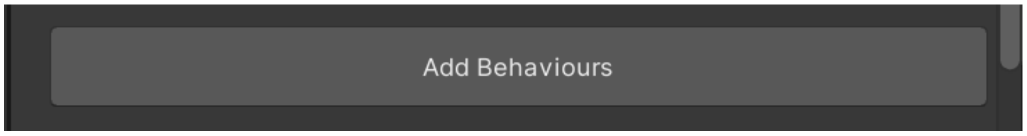
Drop Height	0
Jump Distance	0

▸ Advanced

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.*

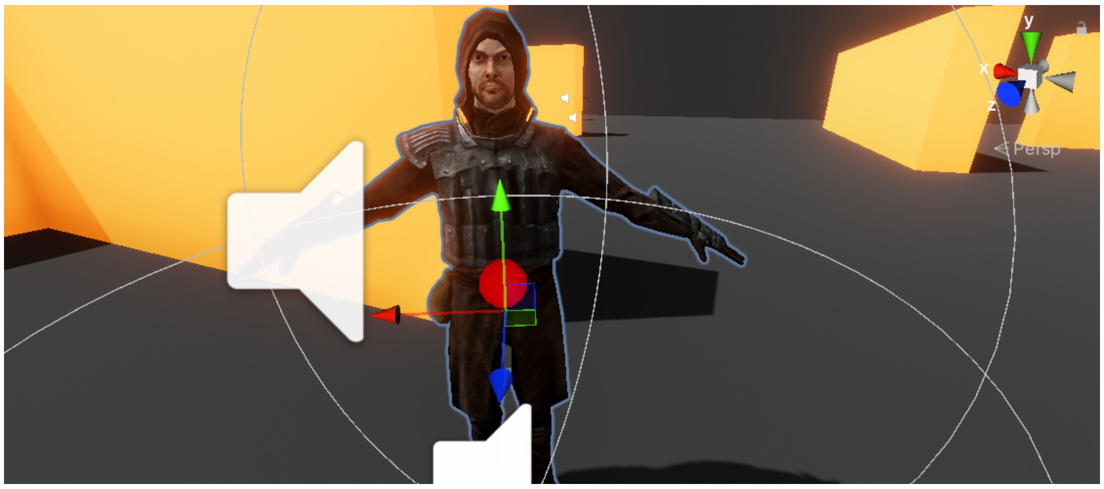
9. On the added Normal State behaviour component, setup the properties by adding the animation names for idle and move - we'll get to the animations later in this doc. In the section **Animations** after Getting Started - and the desired speeds, audios, etc…

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.**

12. 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 from and setting this as the center position of the AI used for calculations. It looks like this:

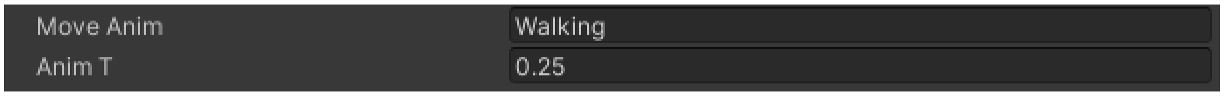


On pressing play, you'll find that your AI is walking around.

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:

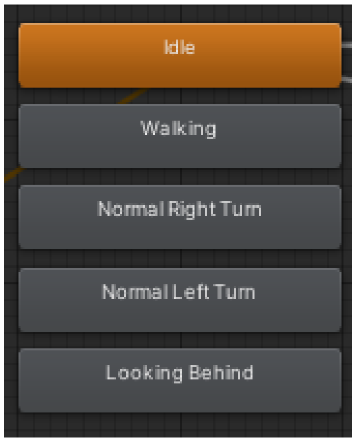


Move Anim	Walking
Anim T	0.25

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.

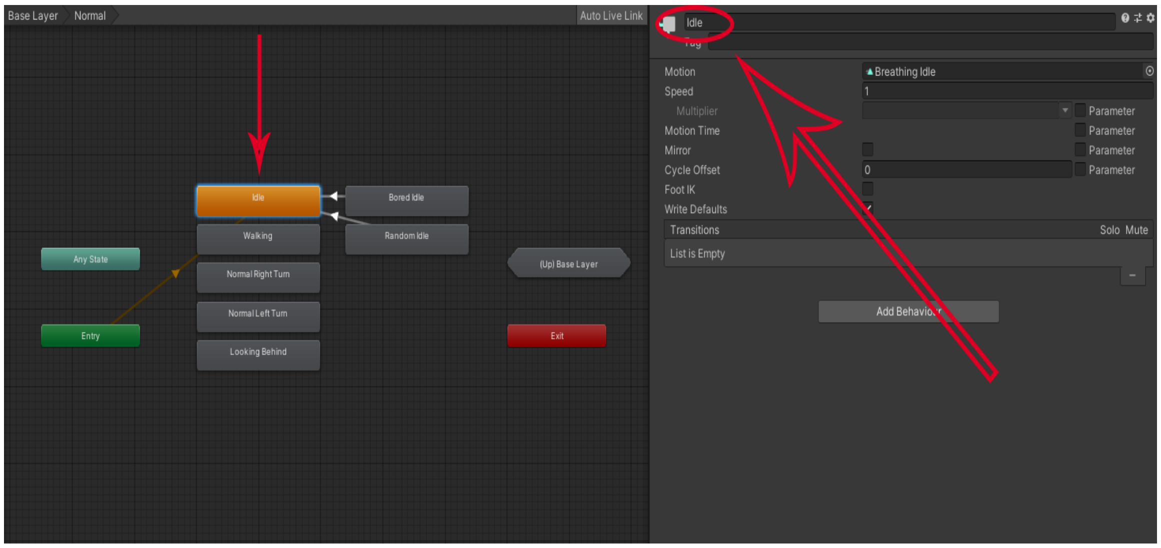
The Anim is the animation name. What is known inside of the Unity Engine as the *Animation State Name.* Which is this:



Make sure to enable loop in all your animation files.

<u>It's the name of your animation inside the Animator. That's it!</u>

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:



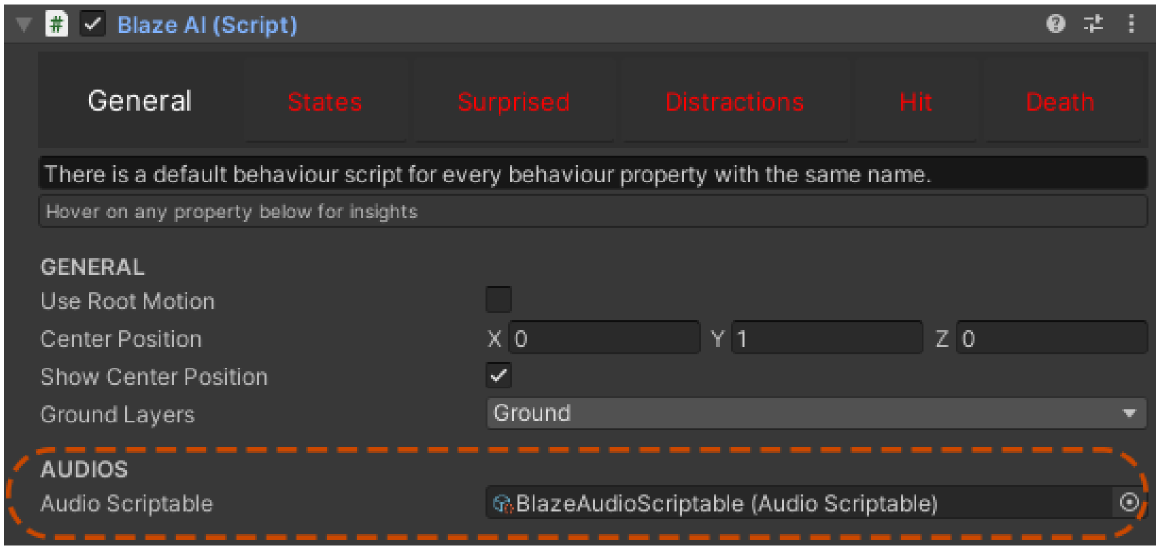
Idle
Tag
Motion: Breathing Idle
Speed: 1
Multiplier: Parameter
Motion Time: Parameter
Mirror: Parameter
Cycle Offset: 0 Parameter
Foot IK
Write Defaults: ✓
Transitions Solo Mute
List is Empty

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 AI (Script)

General | States | Surprised | Distractions | Hit | Death

There is a default behaviour script for every behaviour property with the same name.

Hover on any property below for insights

GENERAL

Use Root Motion

Center Position X 0 Y 1 Z 0

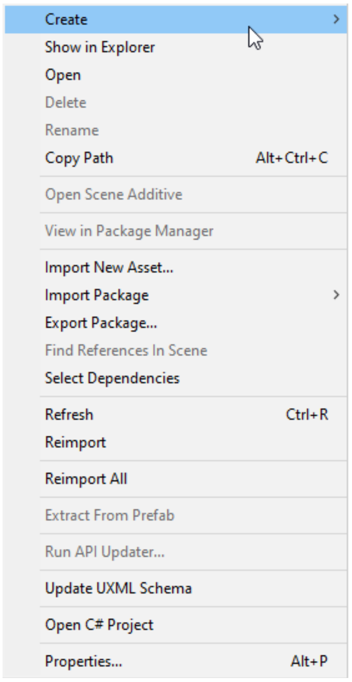
Show Center Position ✓

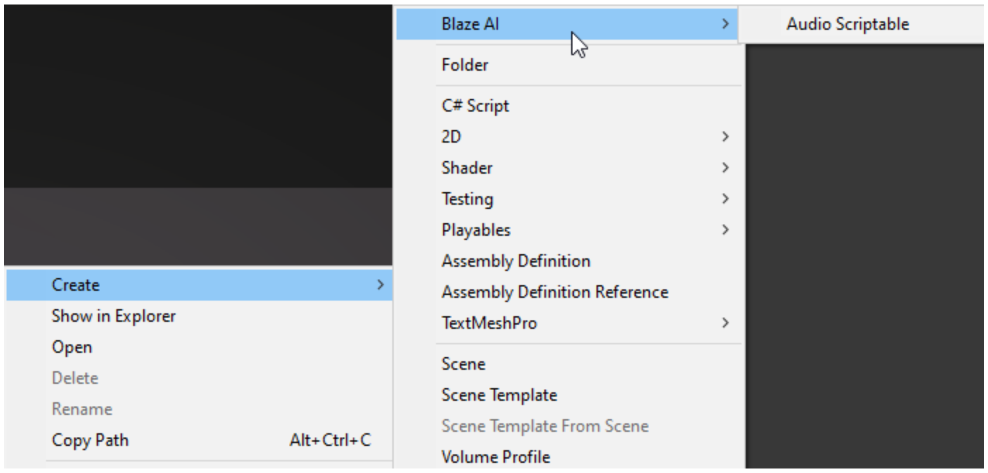
Ground Layers: Ground

AUDIOS

Audio Scriptable: BlazeAudioScriptable (Audio Scriptable)

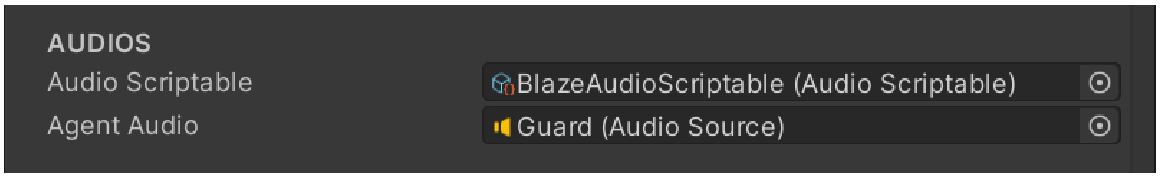
Blaze gives the option to create an Audio Scriptable and set your audio clips to all the necessary states and actions. Create an audio scriptable like this: (right clicking in the project space)





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.



AUDIOS

Audio Scriptable: BlazeAudioScriptable (Audio Scriptable)

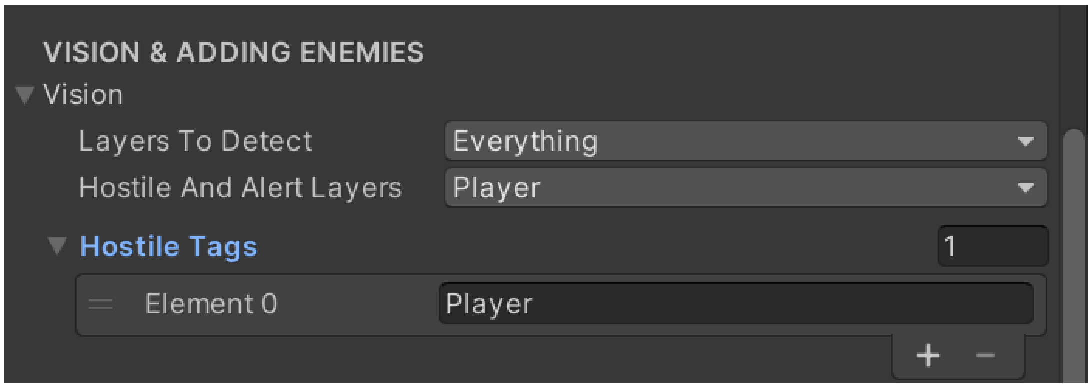
Agent Audio: Guard (Audio Source)

You can also choose to manually set another audio source to your liking that the AI will be using.

Adding Enemies

Simply go to the main Blaze AI inspector. In the General tab, open Vision section.

Let’s start with the easy one. Add your enemy tag name in the *Hostile Tags* property. Like so:



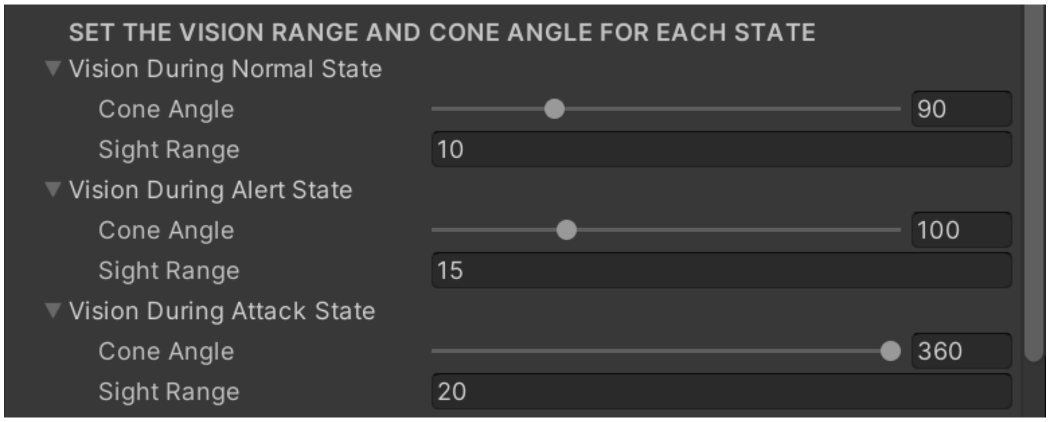
VISION & ADDING ENEMIES
Vision
Layers To Detect: Everything
Hostile And Alert Layers: Player
Hostile Tags: 1
Element 0: Player

Layers To Detect: this should contain all the layers you want the AI vision to check for. 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 and such. Don’t set to Everything.*

Hostile and Alert Layers: this should contain the layer(s) of the enemy and *alert tags (check page 13)*. So add the enemy’s layer here. As seen in the picture.

Finally your enemy needs to have at least one collider to be detected. Multi-colliders are ok. And that’s all you need really to detect an enemy.

Set the vision angle and range of your AI for each state using these properties:



SET THE VISION RANGE AND CONE ANGLE FOR EACH STATE

▼ Vision During Normal State

Cone Angle	90
Sight Range	10

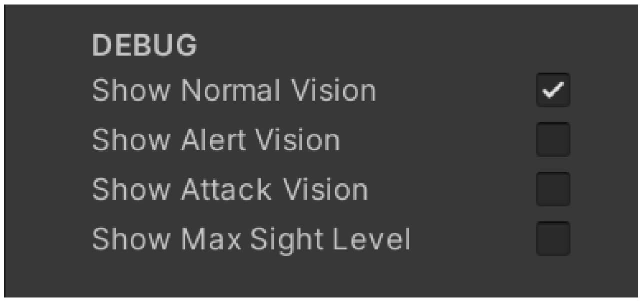
▼ Vision During Alert State

Cone Angle	100
Sight Range	15

▼ Vision During Attack State

Cone Angle	360
Sight Range	20

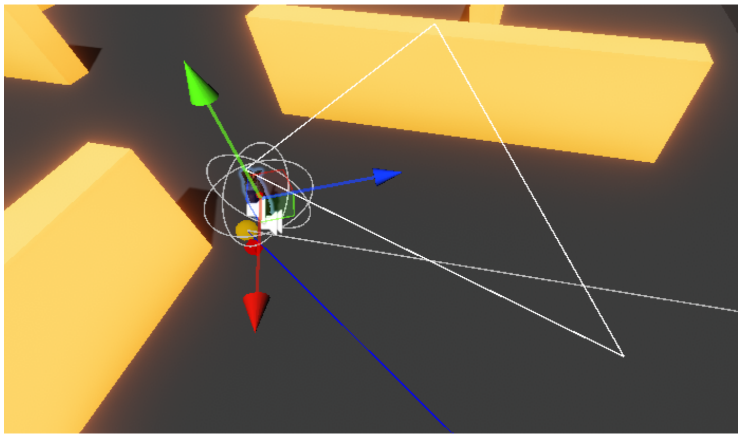
In the DEBUG section below you can choose to show the vision for each state and it'll appear in the scene view.



DEBUG

Show Normal Vision	✓
Show Alert Vision	
Show Attack Vision	
Show Max Sight Level	

Will appear in the scene view as so:

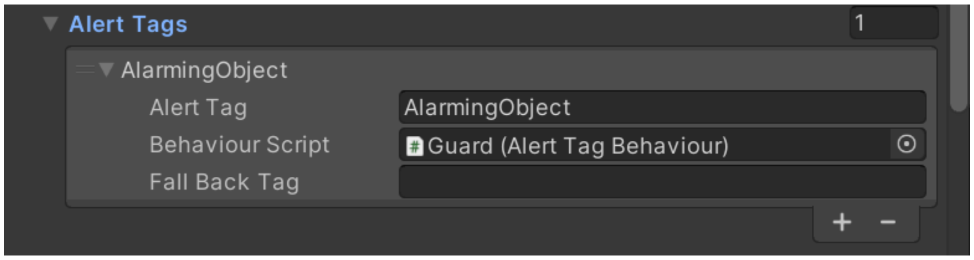


ALERT LAYERS/REACTING TO CERTAIN TAGS

If you've read the previous Adding Enemies 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).**

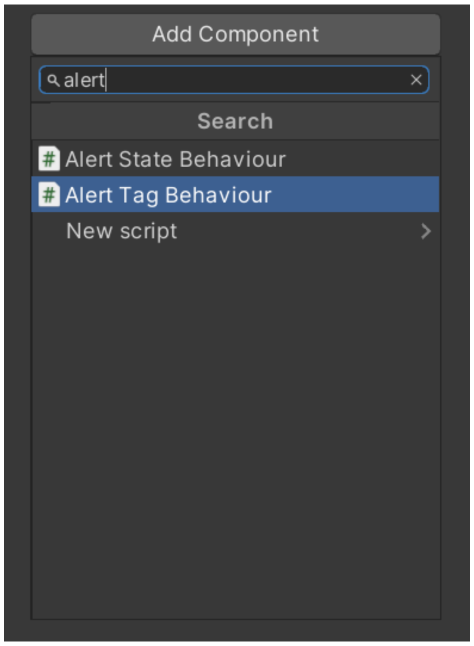
You start by adding the tag name of the game object you want to react to in the Alert Tag property inside the Alert Tags class in Vision.


Alert Tags 1
AlarmingObject
Alert Tag: AlarmingObject
Behaviour Script: Guard (Alert Tag Behaviour)
Fall Back Tag:

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.

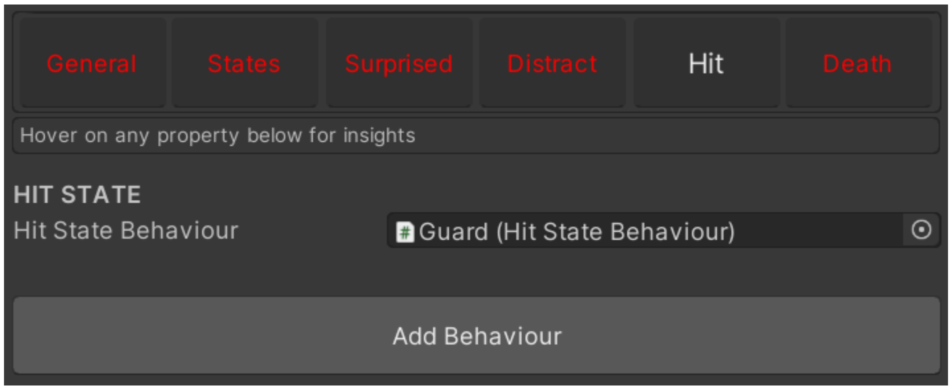
Last thing is the **Fall Back Tag** property. When the AI sees a game object with an alert tag it immediately changes the object tag to the fallback tag in order not to have itself or other AIs reacting to it all over again. Whatever you set in this property will be the new tag name of the game object.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

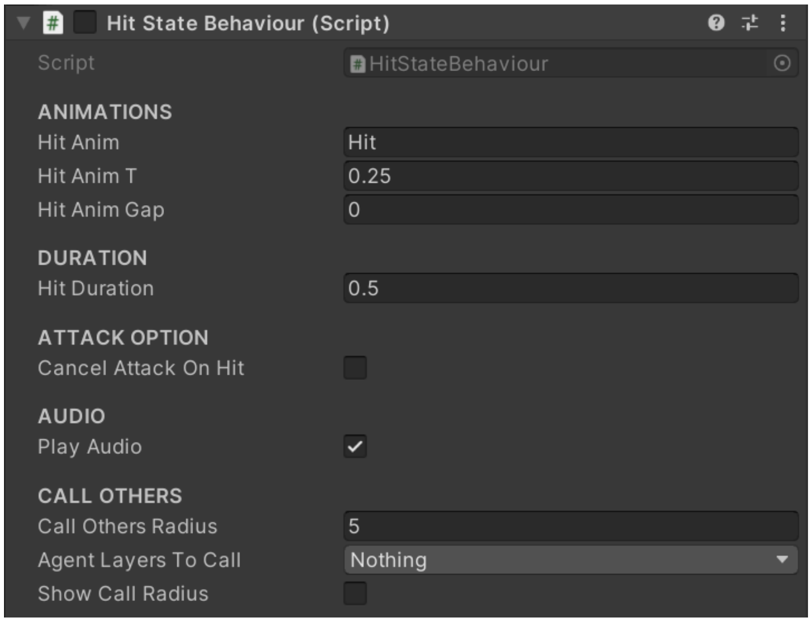


Hover on any property below for insights

HIT STATE

Hit State Behaviour: Guard (Hit State Behaviour)

This is the script after being added to the AI.


Hit State Behaviour (Script)

Script: HitStateBehaviour

ANIMATIONS

Hit Anim	Hit
Hit Anim T	0.25
Hit Anim Gap	0

DURATION

Hit Duration	0.5

ATTACK OPTION

Cancel Attack On Hit	☐

AUDIO

Play Audio	✓

CALL OTHERS

Call Others Radius	5
Agent Layers To Call	Nothing
Show Call Radius	☐

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:

```
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:

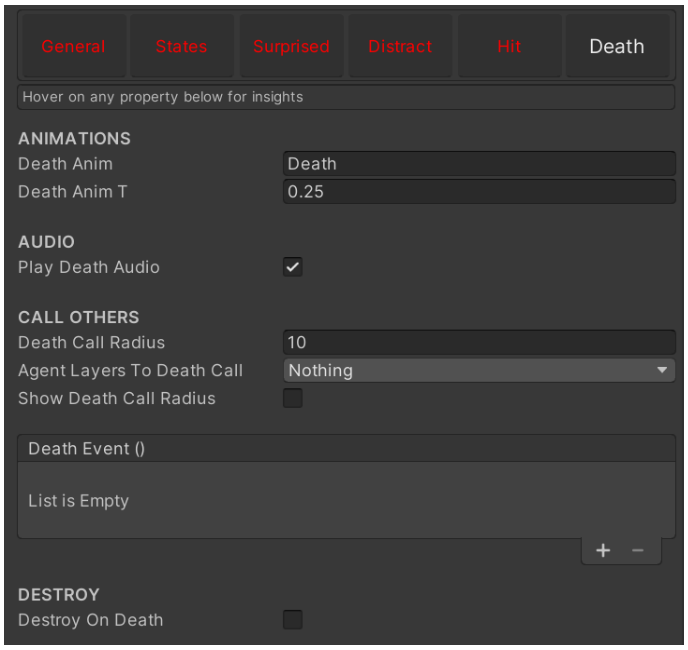
```
public void HitAI(int damagePoints)

{

    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.



Hover on any property below for insights

ANIMATIONS
Death Anim: Death
Death Anim T: 0.25

AUDIO
Play Death Audio: ✓

CALL OTHERS
Death Call Radius: 10
Agent Layers To Death Call: Nothing
Show Death Call Radius: ☐

Death Event ()
List is Empty

DESTROY
Destroy On Death: ☐

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().

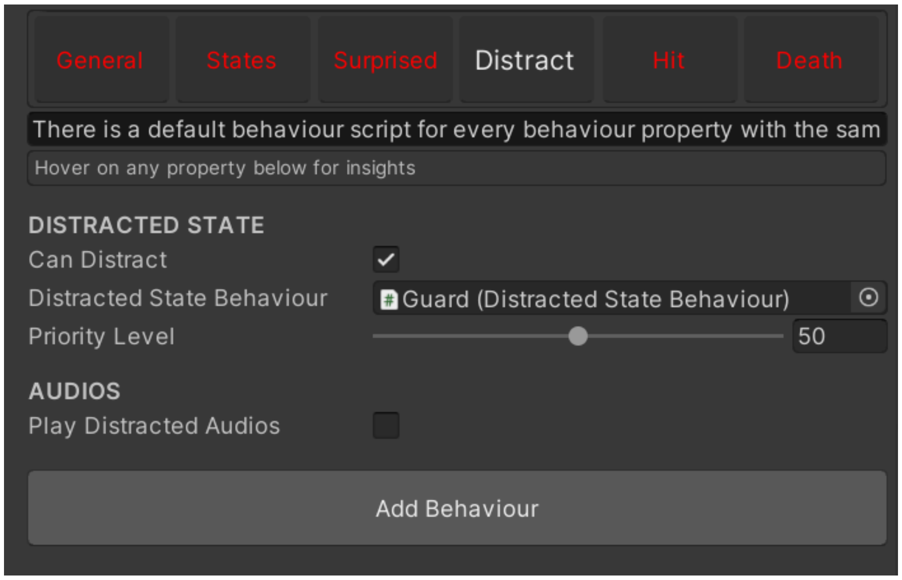
```
void Update()

    {

        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.



There is a default behaviour script for every behaviour property with the sam

Hover on any property below for insights

DISTRACTED STATE

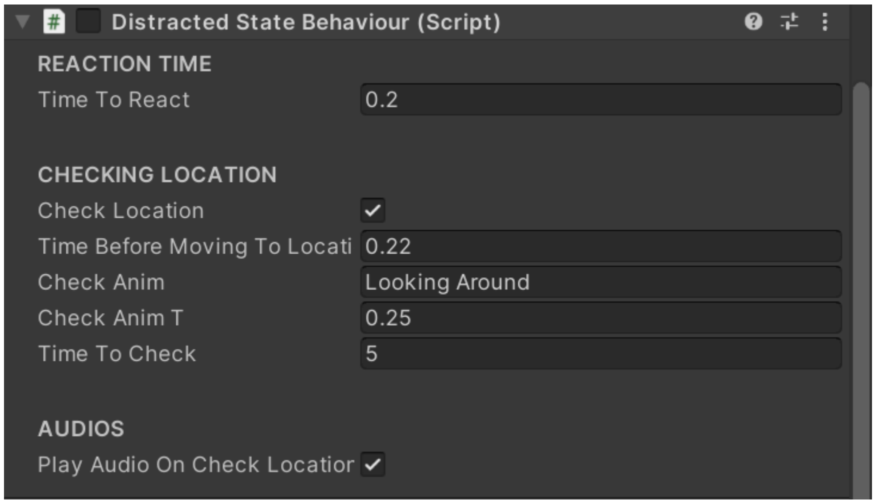
Can Distract	✓
Distracted State Behaviour	Guard (Distracted State Behaviour)
Priority Level	50

AUDIOS

Play Distracted Audios	

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.



Distracted State Behaviour (Script)

REACTION TIME

Time To React	0.2

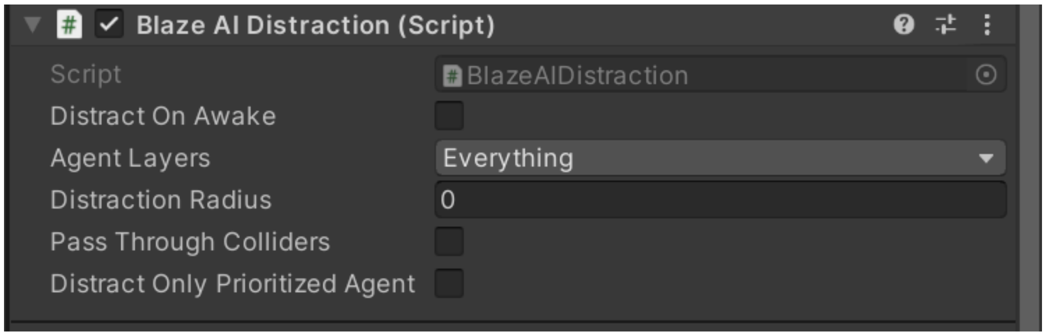
CHECKING LOCATION

Check Location	✓
Time Before Moving To Locati	0.22
Check Anim	Looking Around
Check Anim T	0.25
Time To Check	5

AUDIOS

Play Audio On Check Locatior	✓

Now to distract an AI you must use the script: **Blaze AI Distraction**. Add this script to any game object you want to be the distraction.



Blaze AI Distraction (Script)

Script	BlazeAIDistraction
Distract On Awake	☐
Agent Layers	Everything
Distraction Radius	0
Pass Through Colliders	☐
Distract Only Prioritized Agent	☐

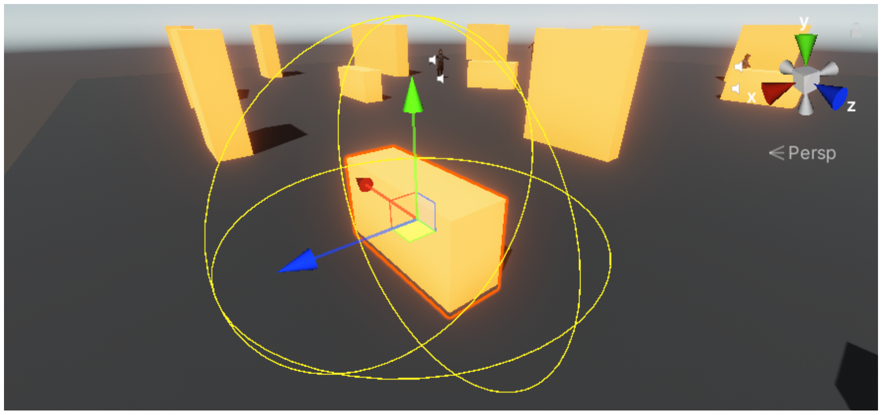
After setting the properties (we'll get to them) you trigger the distraction using the public method of the script: ***TriggerDistraction()***. For the code check the Distraction demo.

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?

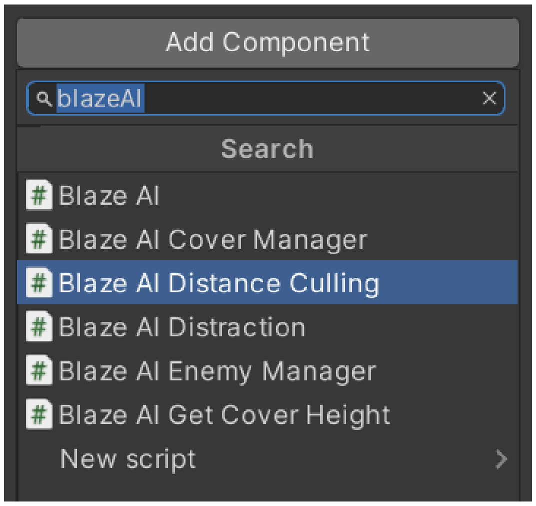
Distract Only Prioritized Agent: If enabled, only the AI with the highest distraction priority in a group will turn and face the distraction. If disabled, the entire group will face and look at the distraction but only the highest priority will check the location.

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

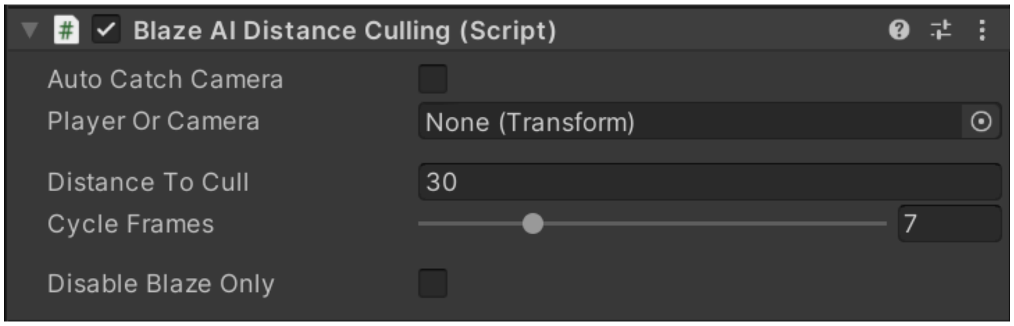
Distance Culling

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.



Blaze AI Distance Culling (Script)

Auto Catch Camera	☐
Player Or Camera	None (Transform)
Distance To Cull	30
Cycle Frames	7
Disable Blaze Only	☐

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.*

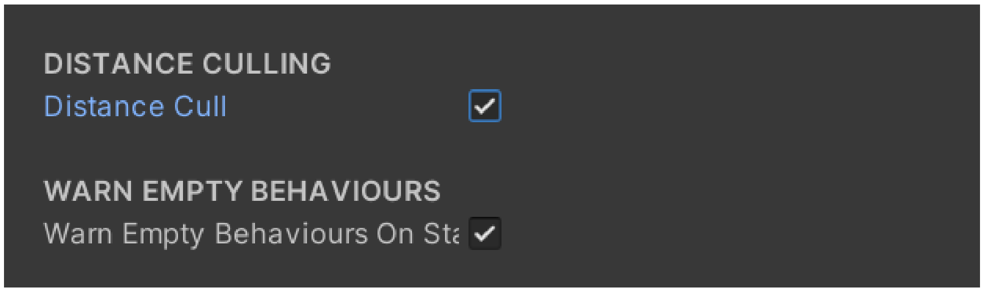
Distance To Cull is the distance you want to disable the AI when it exceeds this value.

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 gameobject. (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.**


DISTANCE CULLING
Distance Cull ✓

WARN EMPTY BEHAVIOURS
Warn Empty Behaviours On Sta ✓

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.

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

<u>*state*</u> – 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.

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.

<u>BlazeAIEnemyManager</u> – 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.***

<u>BlazeAIDistraction</u> – 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.

<u>BlazeAIGetCoverHeight</u> – 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.