Blaze AI Documentation

Thank you for purchasing **Blaze AI**. For any questions and support you can email me directly at: *pathiralgames@gmail.com*

Blaze AI community server here

Browse **Pathiral**'s packages 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:

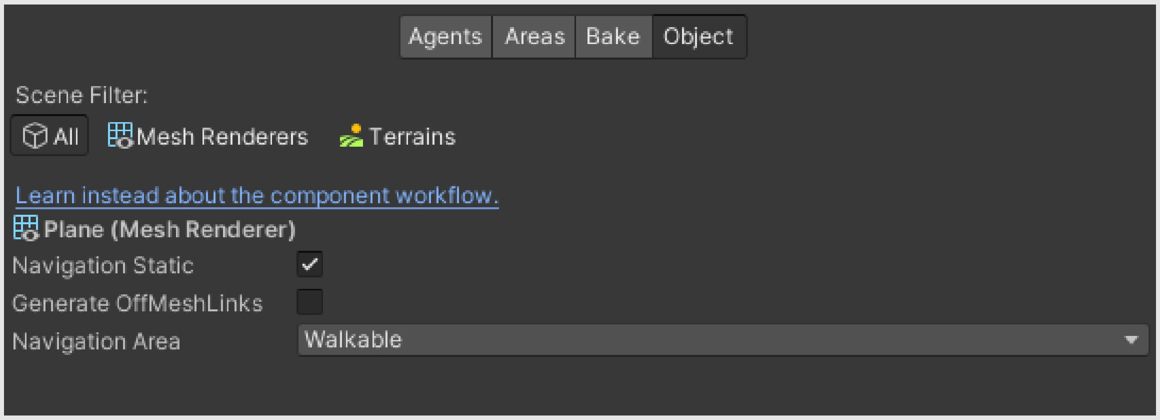
Making Distractions

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)

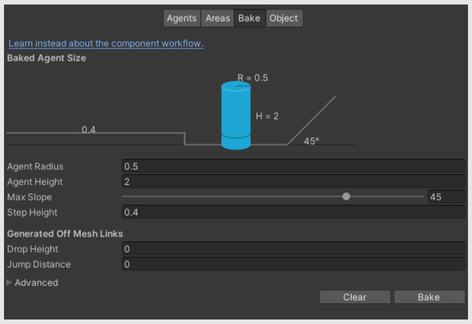
Navigation Static ✓

Generate OffMeshLinks

Navigation Area: Walkable

3. For adding obstacles, you do the same as above. Set the **Navigation Static** to true also but the only difference is setting the **Navigation Area** to *Not Walkable.*

4. Now make sure the plane is selected and go to the **Bake** tab and click *Bake*.



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

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 (behaviour) for each one. You'll find that in other places in the Blaze inspector as well.

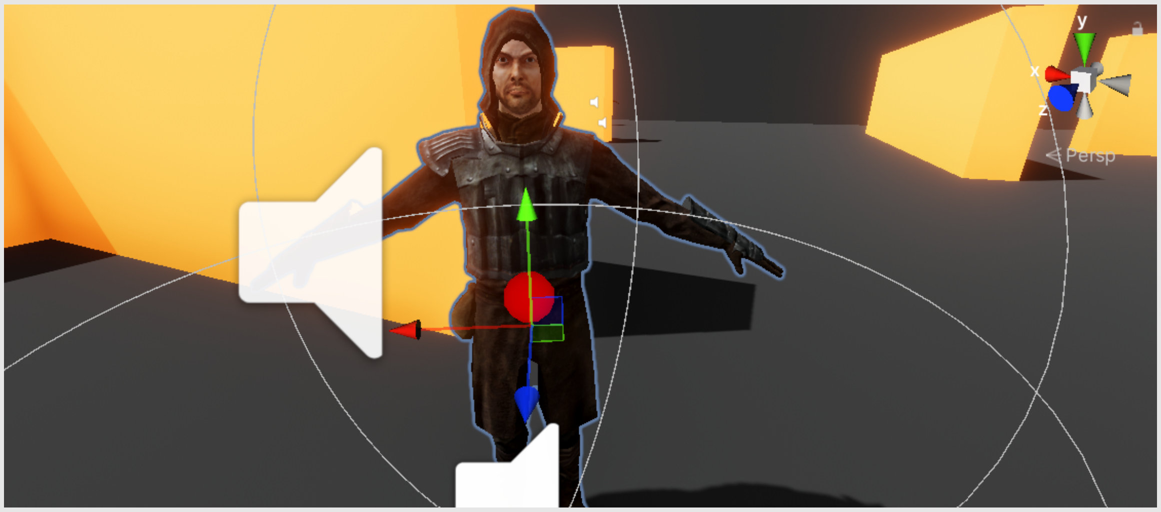
8. Blaze comes with a behavior script for each and every property with the same name. So for example the Normal State Behaviour property has the NormalStateBehaviour script and so on. You only need to search for them by Add Component. Then drag and drop the script to the property with the same name in Blaze inspector. So do add the normal state behaviour script and stitch it to the property. On doing so, you'll find the behaviour script got disabled. (Check is gone)

9. On this 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.

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:

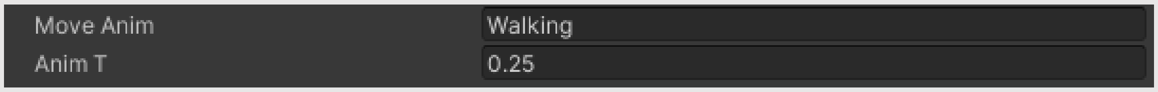


On pressing play, you'll find that your AI is walking around.

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:

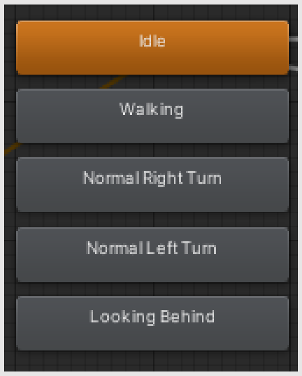


Move Anim: Walking
Anim T: 0.25

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.

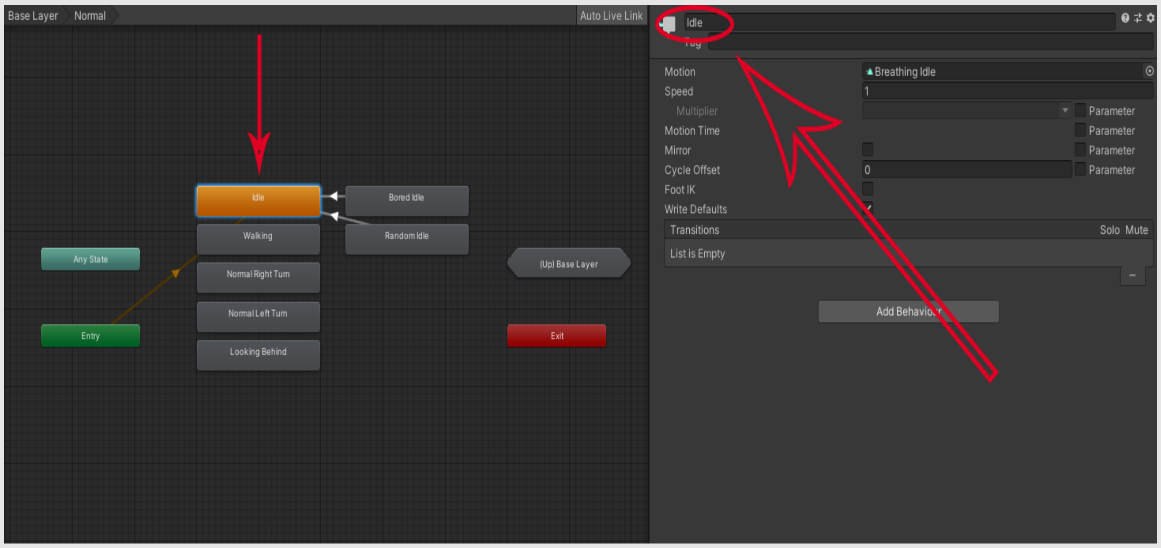
The Anim is the animation name. What is known inside of the Unity Engine as the *Animation State Name.* Which is this:



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:



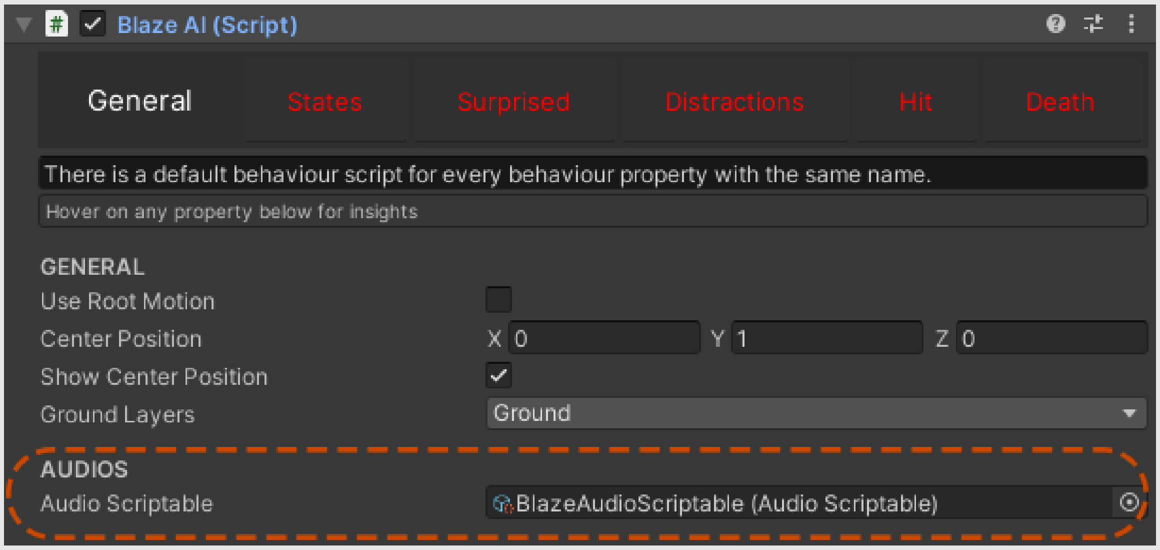
Idle

Motion: Breathing Idle
Speed: 1
Multiplier: Parameter
Motion Time: Parameter
Mirror: Parameter
Cycle Offset: 0 Parameter
Foot IK
Write Defaults: ✓

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.

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**.



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

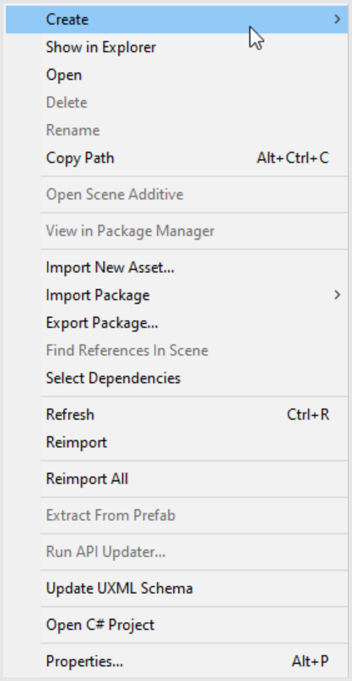
Show Center Position ✓

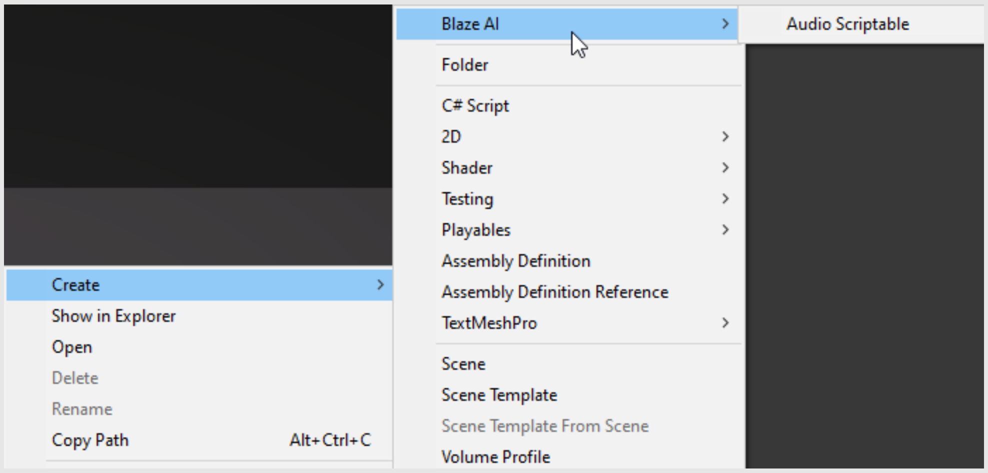
Ground Layers: Ground

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)

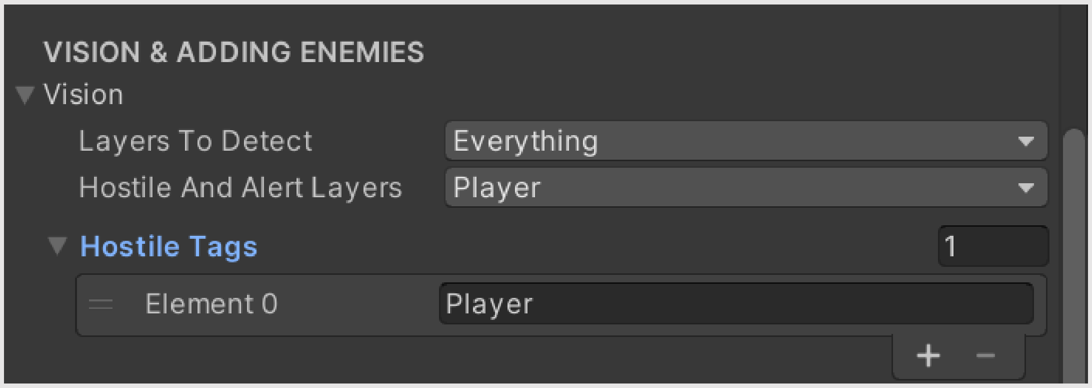




For each state/action, you can add an unlimited number of audios. When an audio call is made the system will choose a random audio in that state, if only one is set then that one will always be played. If empty, no audio will be played.

Adding Enemies

Simply go to the main Blaze AI inspector. In the General tab, open Vision section and add the 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 choose the layers that are used as obstacles, ground and such. Don't use Everything.*

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

You may be asking, what are Alert Layers?

Alert Layers are the layers of Alert Tags (found below Hostile Tags property). Alert Tags are game objects with specific tag names that you want the AI to react to and turns the AI to alert state. Check Demo 3 (full version only).

Public Properties and 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) - Use this to force move the agent to any location. ***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.

SetEnemy (GameObject enemy) – Set an enemy for the AI. The AI's vision will turn to that of attack state for a single frame and if it catches any hostiles in that frame, it'll go into action. If not, the AI will still go to attack state and go check the passed enemy location.

Hit (GameObject enemy=null) – Use this method to hit the AI. It'll go into hit state and exit when the hit state duration finishes. Now this method can take an optional parameter of type GameObject. The passed game object should be of whatever hit the AI. If the parameter has been passed, the AI after exiting the hit state will go into attack state and move to that game object's location. If nothing passed, after exiting hit state, it'll go to alert state and continue patrolling since the AI doesn't know what hit it.

Death () – this will trigger the death state. The agent will play the death animation and audio.

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 and it'll print you it's height in the console log. Use it with cover shooter to be able to set the high and low cover heights.

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

<u>BlazeAICoverManager</u> – 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.

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.