

Alek AI Framework

Contact with developer:

Email: alekgameshelp2@gmail.com

discord: <https://discord.gg/erzHBcx3ES>

document summary:

1. Alek AI Introduction
2. Asset Setup
3. Scriptable Objects
4. Creating an AI
5. Patrol Behaviour
6. Target Detection
7. Chasing & Attacking
8. Health & Damage
9. Additional Actions
10. Alerts
11. Sound Effects
12. Optimization/Quality
13. Specific Module Documentation

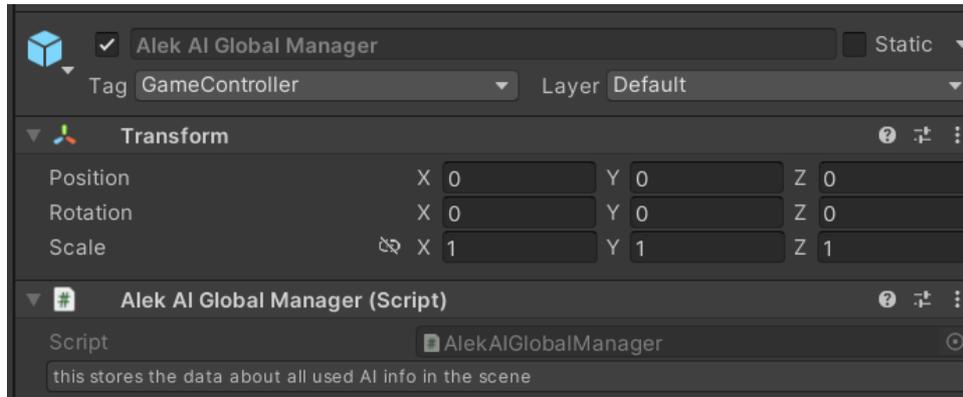
1. Alek AI Introduction:

hi, read the description of the asset for exact info about it. Keep in mind that this documentation tells you about all the features of the AI, even those in the additional modules, which you might not have.

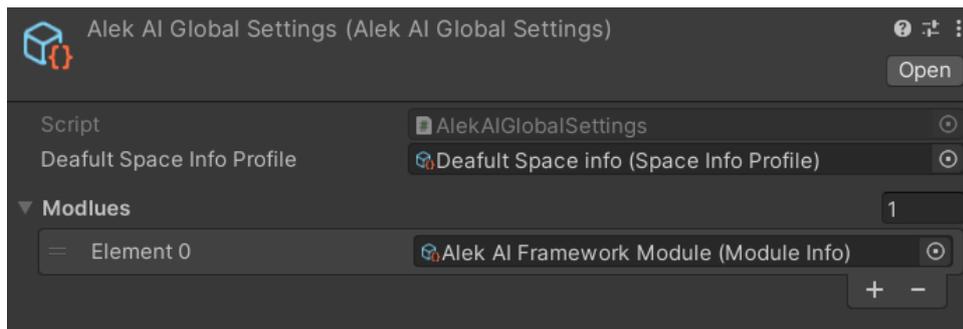
2. Asset Setup:

Alek AI Framework has a modular structure, and still manages to have little to no setup required.

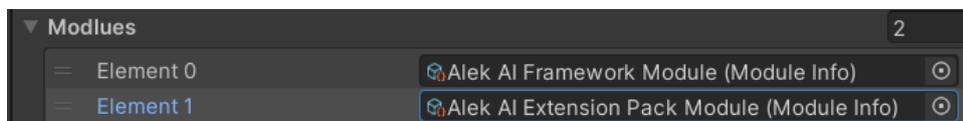
If you bought just the main framework, all you need to do is add an Alek AI Global manager Object to your scene. you can find a prefab with it in a folder located at Framework/Essential Prefabs/



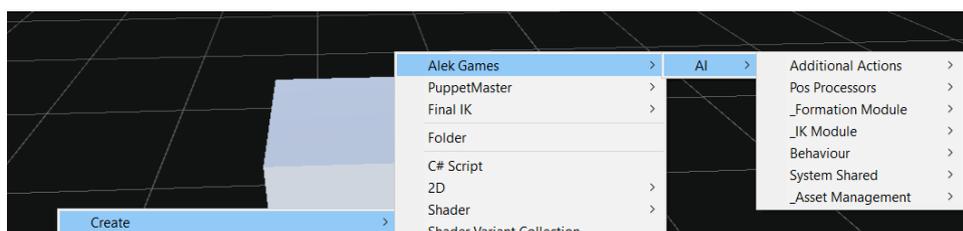
If you bought a module that extends a feature of the main framework, it is advised to specify that module in the Alek AI Global Settings (doing so will link the new features to the behaviour maker). You can do so by selecting the Alek AI Global Settings (by default located at: AI/_Framework/Essential Prefabs/) and filling the modules array with the new module info.



For example, the extension pack brings many new features into the AI, so it has a module info. To integrate it you need to reference it in the modules array as said before. It should look like this:



3. Scriptable Objects:

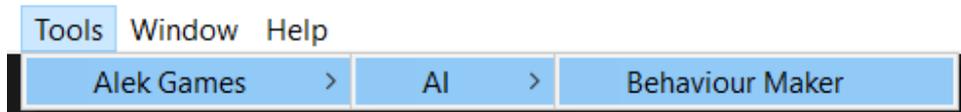


1. Additional Actions – make the AI move in a certain way:
 - AdditionalActionHolder – the simplest additional action system available.
 - AnimationParameterActionProvider – just reverses the current animation. May be useful for parrying
 - DirectionActionProvider – dynamically adjusts the additional action based on a direction. Useful for hit reactions or dodging
 - HideActionProvider – used for cover shooter/runaway behaviour. Makes the AI go to a spot that is considered hidden from the target
2. Position Processors – processes the given vector:
 - AIBaseHeightPosProcessor – snaps the y position to the AI base y position
 - NavMeshSamplePositionProcessor – returns the closest point on the nav mesh to provided point
 - RaycastPositionProcessor – snaps the position to the ground
 - InterceptionPositionProcessor - Tries to anticipate where the target is going, and intercept that point
3. Formations – for the formation module
 - GroupMemberID – gathers info about the AI.
 - JustAttackQueueFormationProfile – queues the AI's attacks so that there is a limited amount of AI trying to attack the target at once
 - LineFormationProfile – forms a line from the AI before the targets
 - SurroundFormationProfile – makes the AI surround the targets
4. IK – inverse kinematics systems:
 - DynamicBlockProfile – info about how the AI should block the targets attacks
5. Behaviour:
 - AttackProfile – if you use the default weapon system, this will keep the info about how the AI should act during the attack
 - BehaviourProfile – hold info about the main things AI will do. (movement, animations, additional actions)
 - DetectorProfile - if you use the default detection system, this will determine how and when the AI will detect the target
 - EnemyBehaviourActionsList – a list of actions the AI can be doing
 - SpaceInfoProfile – provides the basic world info like distances considered little/ big, what is ground etc.
6. Shared – a lot of systems may use these, not even the AI:
 - WayPointsInfo – stores info about a path
 - TagProfile – a way to check tags of an object.
7. Asset Management- stuff for the whole asset:
 - AlekAIGlobalSettings – mostly what the editors should know
 - ModuleInfo – info about all the scripts extending certain interfaces in a module

4. Creating an AI:

With Alek AI Framework it is incredibly easy to create your own AI using the Behaviour Maker.

To start open the Behaviour Maker Editor Window (see where to do so below)

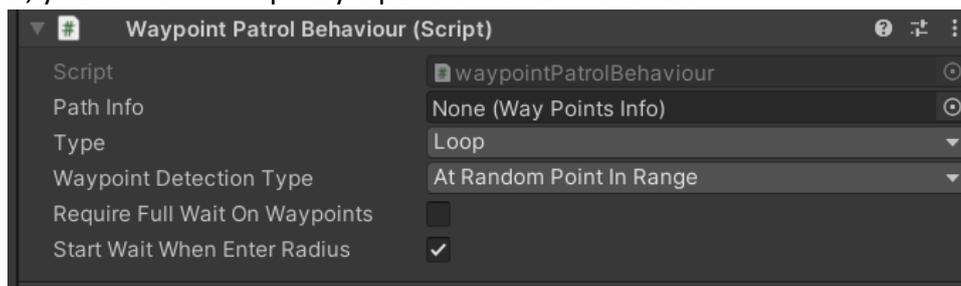


Once you click on it, the window will open. Follow the instructions that are in it (the help boxes), and you will have a fully functional AI.

5. Patrol Behaviour:

Alek AI Framework uses an interface for handling patrolling. All scripts that properly extend the `IPatrolBehaviour` interface can serve as patrol behaviours. You specify it in the Behaviour Manager. Here I will explain how to use all the default patrol behaviours.

1. `StandingPatrolBehaviour` – it needs no setup, just add it and the AI will stand
2. `WaypointPatrolBehaviour` – this behaviour makes the AI follow a path. Once you add it, you will need to specify a path and how to follow it.



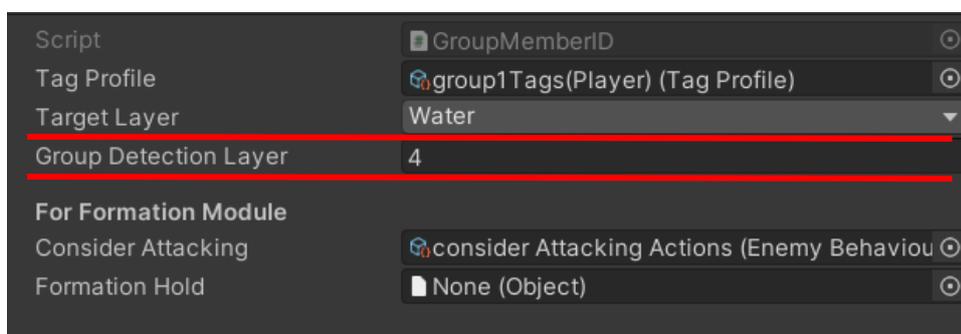
3. `FollowPatrolBehaviour` – follows a target. Just assign a detector (for a simple transform follow use `TransformProviderTargetDetector`)
4. `RandomPointsPatrolBehaviour` – roams around a set perimeter

6. Target Detection:

To detect a target, you will need a system extending the IDetector interface. You specify the detector in the behaviour manager. Here are the default ones:

1. BlindTargetDetector – never sees anything
2. DynamicTargetDetector – dynamically detects targets based on the colliders on them, this is the most advanced system included in the AI.
3. PresetTargetDetector – checks if sees a preset transform
4. TransformProviderTargetDetector – always sees a certain transform

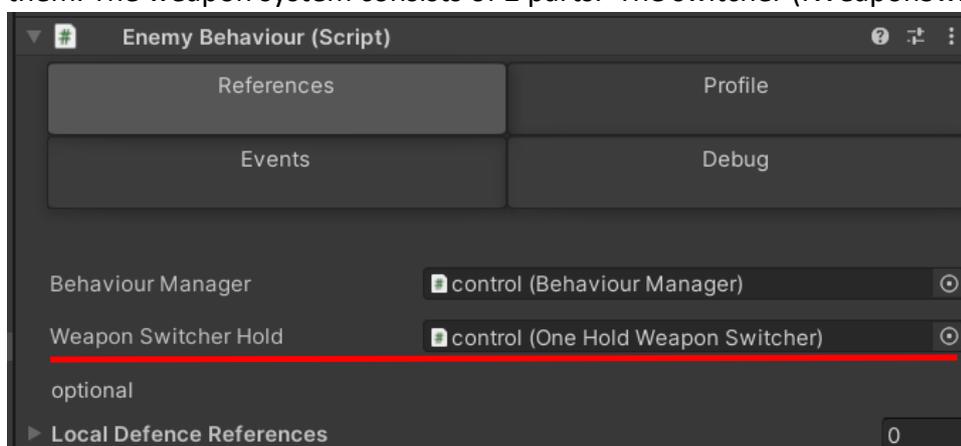
There is quite a lot that can go wrong with setting up target detection. Most likely it is due to detector collider layer and tag, and detectors target layer. It might be the group member fault, as it is responsible for applying these when you enter play mode. First of all, make sure that the layers and tags specified in the member info match the ones in the scene.



The target layer will be what the AI will try to find, and the group detection layer will be what other AI will have to look for in order to find this one. The tag profile will specify what tags the AI should look for/avoid while detecting targets

7. Chasing & Attacking:

With Alek AI Framework you can use one of a few attack systems and even switch between them. The weapon system consists of 2 parts. The switcher (IWeaponSwitcher interface):



And a Weapon (IWeapon interface)

The weapon switcher gives the behaviour the weapon it should use, and the weapon gives the AI the desired distance to the target, and info about the attack (when to do so, movement during it etc.)

There are a few included attack systems in the AI by default:

1. Switchers:
 - ByDistanceWeaponSwitcher – switchers weapons by distance
 - OneHoldWeaponSwitcher – holds just one weapon, doesn't support weapon switching
2. Weapons:
 - SimpleMeleeWeapon – allows for singular attacks and movement
 - SimpleRangedWeapon – makes the AI do continuous shoots, reloads and movement. Useful for making something like a gun.
 - AdvancedMeleeWeapon – allows for sequence attacks (a few single attacks in a row) and except for that is the same as SimpleMeleeWeapon

There are a few ways of making the AI chasing feel better. if you are making an:

1. agro (aggressive) AI, there are 2 values that can make it nicer:



(this is the

behaviour profile main tab). The stop chase time offset prevents random chase stopping, and investigation time is the time that the AI will try to "find" its target. The investigation only appears when the target was not found by the detector, not destroyed.

2. Runaway behaviour, the stop chase time offset is important if you have a detector of a limited angle. If so, it is the time that the AI will run away, and after that the investigation will work like in an agro behaviour, and if you set it to a small value (0.5-1.5) it will create a "turn around and see if the target is still chasing" type of a behaviour. If your detector is not angle limited during chasing though, stop chase time offset will not be as significant.

8. Health & Damage:

Alek AI has an advanced health and damage system, that has 3 parts: casting, receiving (IDamagable) and handling (IHealth). The casters try to find a receiver and then damage it. Then the receiver passes the damage to the handler that applies it to itself. There are many parts of this system so let's go through them in the cast, receive, handle order:

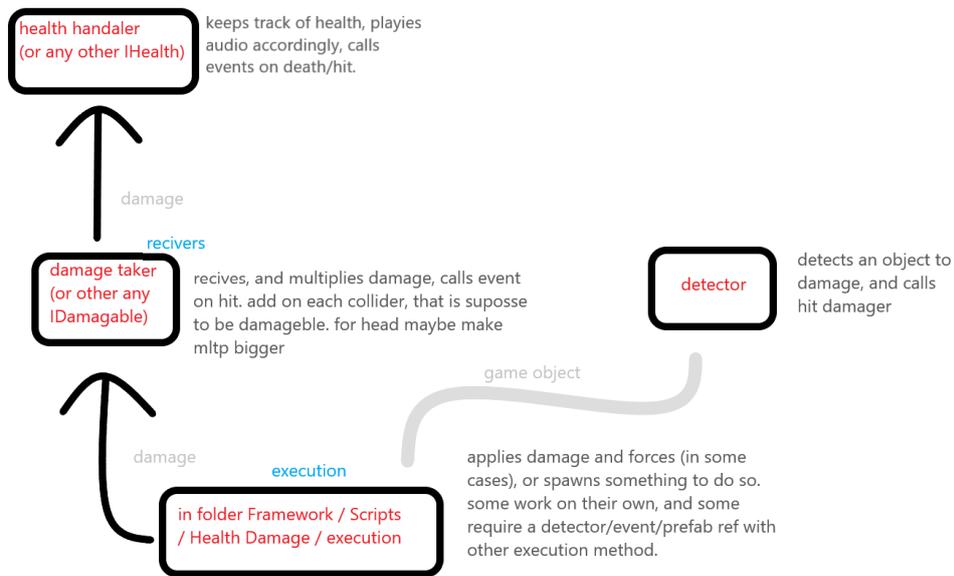
1. Casting – stuff that deals the damage (weapons). this splits into 2 ways of casting:
 - IDamager + detector:
 - a. IDamager:

- SimpleDamager – just damages the object
- AdvancedDamager – much more advanced version of SimpleDamager. This can apply forces, limit damage on repeated collisions (so if your sword colliders with multiple parts of the body you don't multiply the damage by 15, but deal an appropriate amount of damage), apply damage based on the velocity of the hit and more
- b. Detectors
 - CollisionDamageDetector – detects by collider events (Collisions/Triggers)
 - RayDamageDetector – detects by checking a ray from last position to current one. Useful for bullets, or other fast-moving items. Keep in mind when using this, the point it is checking can't be surrounded by a collider, so your bullet can't have any collider.
 - ShapeDamageDetector – detects colliders in a certain shape
- Other:
 - a. CollisionDamager – fall (self) damage
 - b. ExplosionDamager – creates an explosion. Can deal more damage based on the amount of the collider exposed to the explosion, and apply custom explosion force
 - c. StabDamager – deals damage based on a stab. It takes in consideration the depth of the stab and the time the stabber is in the wound. Keep in mind this needs a custom stab system. (like the one from Hurricane VR)
- 2. Receivers – what receives and possibly edits the damage value:
 - DamageTaker – takes damage, and multiplies it by a value, you can use it to make some places more sensitive (so headshots deal more damage, feet shots deal very little)
- 3. Handler – most likely HealthHandler, as the AI requires it.

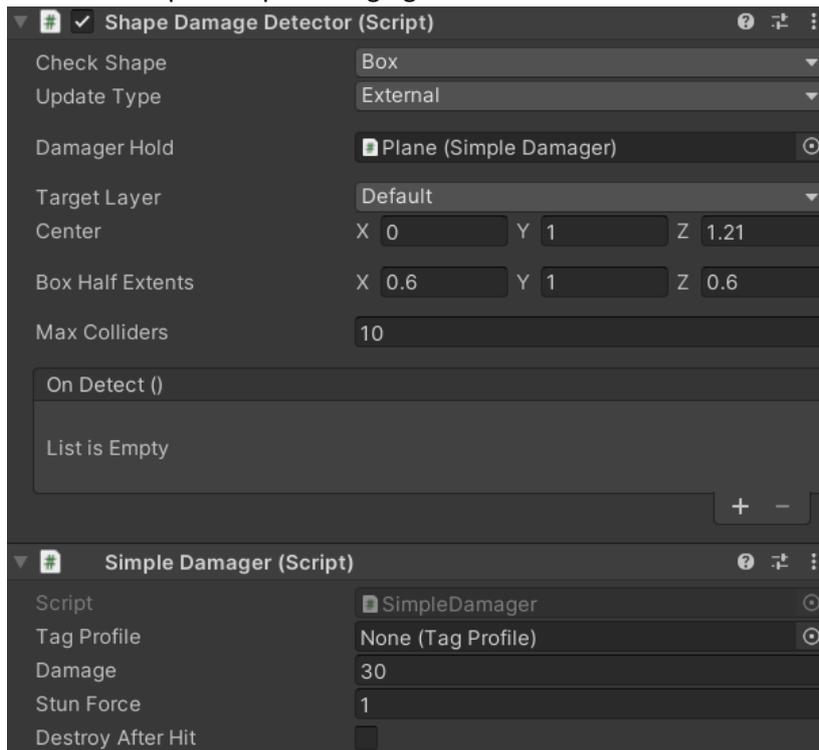
The most common version of damage dealing will be based on SimpleDamager/AdvancedDamager with the ShapeDamageDetector/CollisionDamageDetector. You need both in order for the system to work. I repeat both the components. One says what to damage (detector), other one applies the damage (damager). As long as the object that you try to damage is on the correct layer (possibly specify what to look for in the detector) and has the correct tag (specify in the damager), and a damage receiver (Damage taker), it will

work. As I know a lot of people find it confusing, I made a graph to show it

handlers



The final setup of shape damaging should look like this:

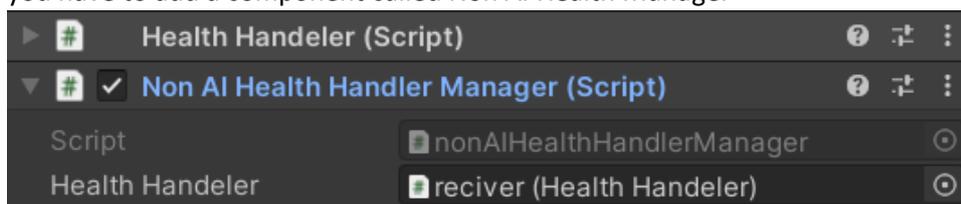


The final setup of collision damage should look like this:

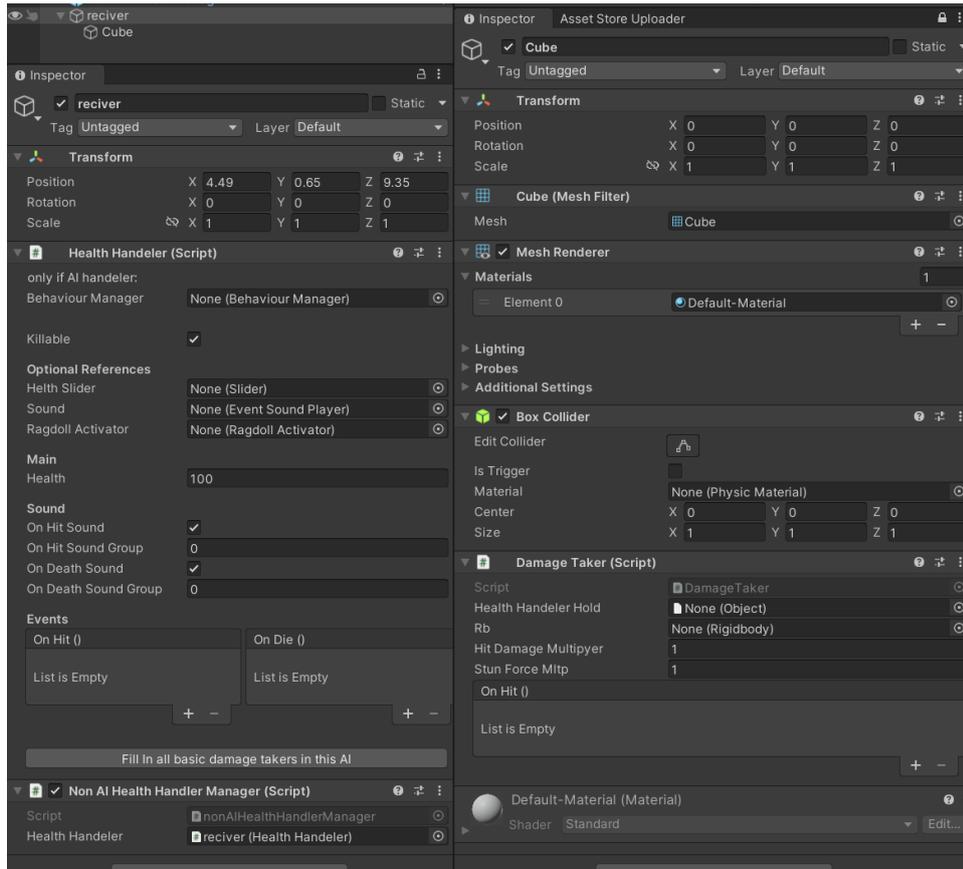


Keep in mind that if you are using collisions, the detector should be attached to the object with the rigidbody on it (this is just how unity works)

You should remember too, that if you want to use the health handler on something that is not an AI, you have to add a component called Non AI Health Manager

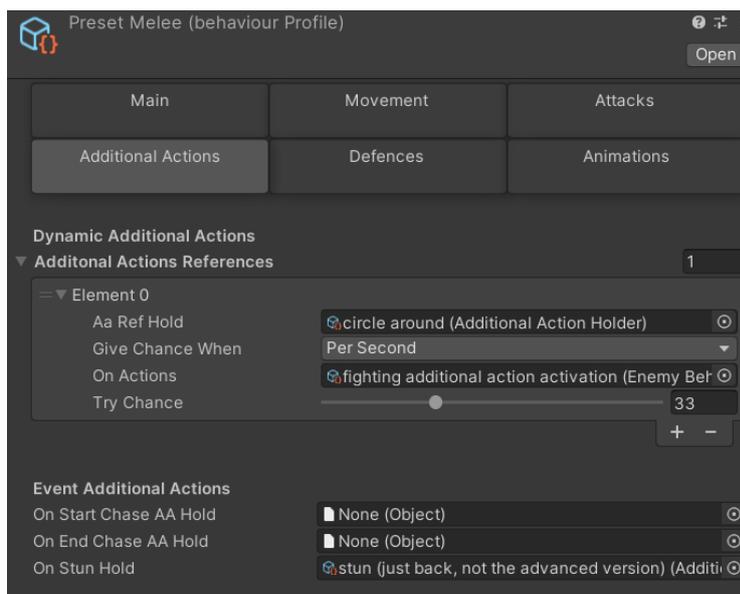


This is an example of a cube with 100 health that can be damaged:

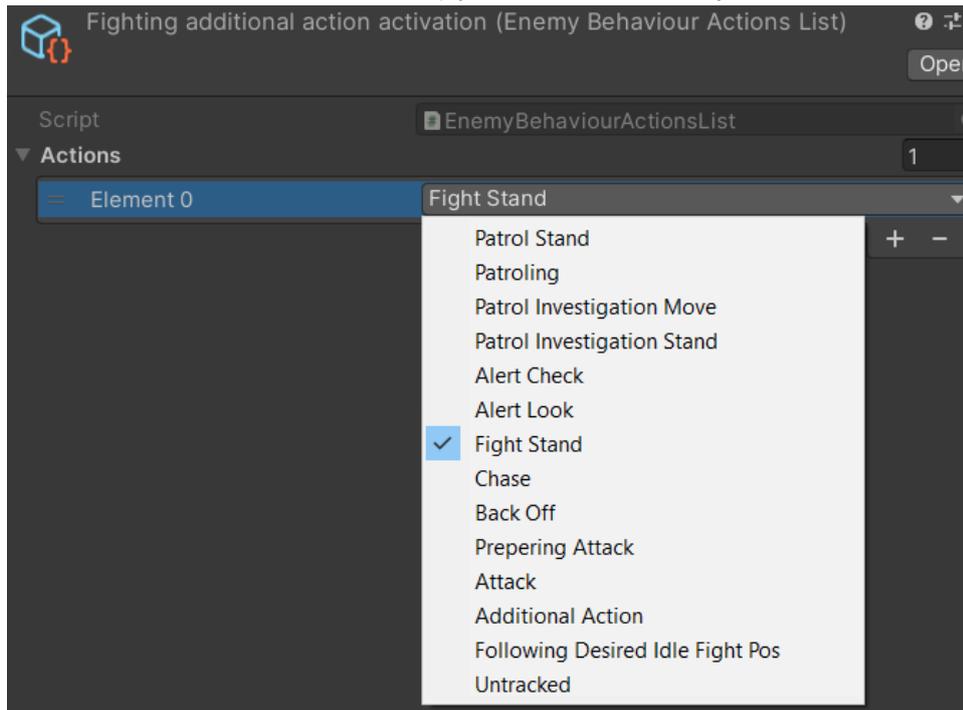


9. Additional Actions

These actions over right what the AI is doing in the current moment, and make it do something else. You can assign an additional action for the AI to do it certain situations.



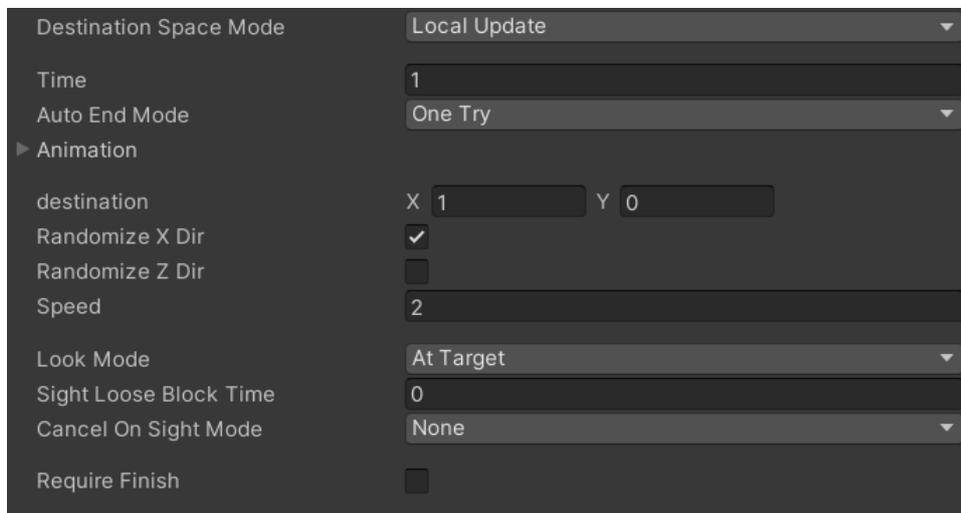
It can do one on a certain action (specified in an EnemyBehaviourActionsList)



or

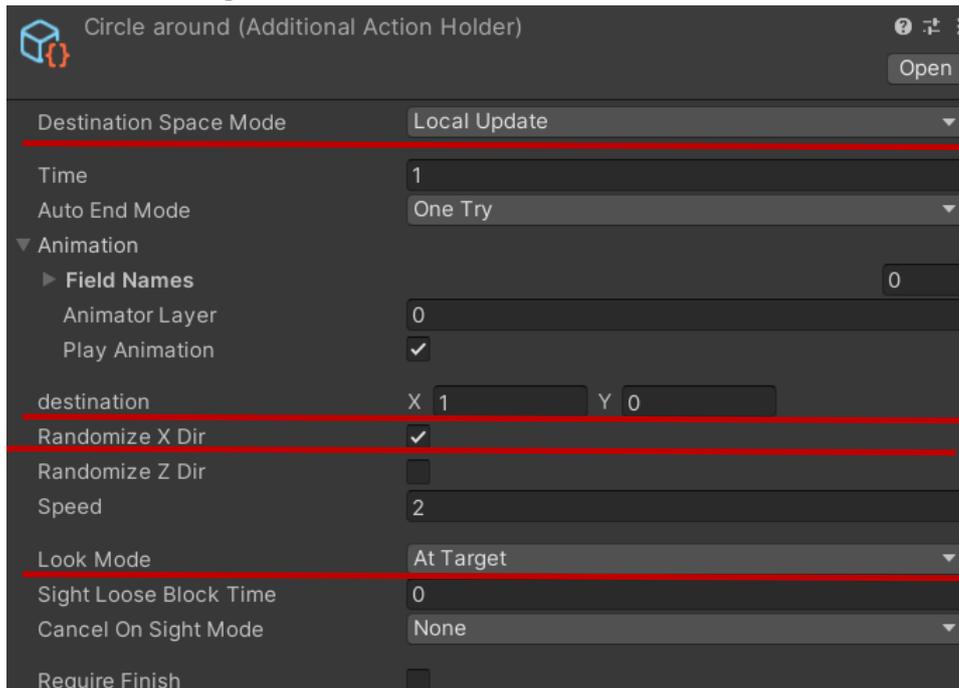
the AI can do an additional action on an event (start/end chasing, get hit (stun), parry, reload and more)

You can specify what additional action to do with an IAdditionalActionProvider. It gives out an additionalActionsSettings

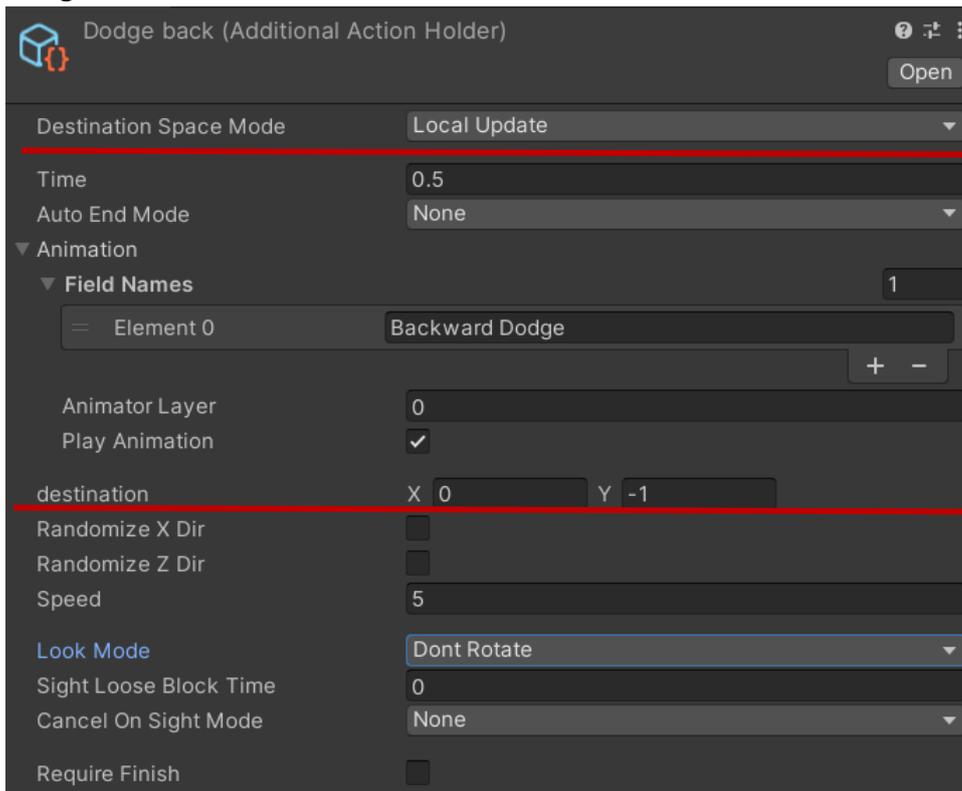


With these values the AI know what it should do. Here are a few things you can achieve with additional actions, and how to do them.

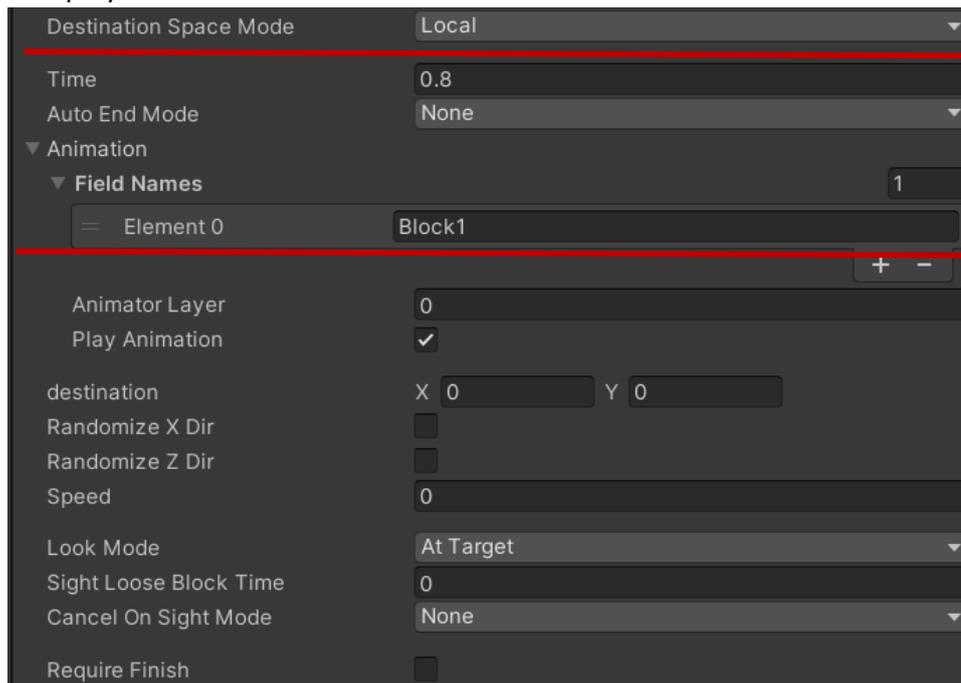
1. Circle around target



2. Dodge back



3. Just play an animation



And much more.

You can use the dynamic systems for additional actions mentioned in the scriptable objects part too, they just create the additionalActionsSettings on their own

10. Alerts

The alert system supports 4 alert types and has 2 parts. Casting and receiving.

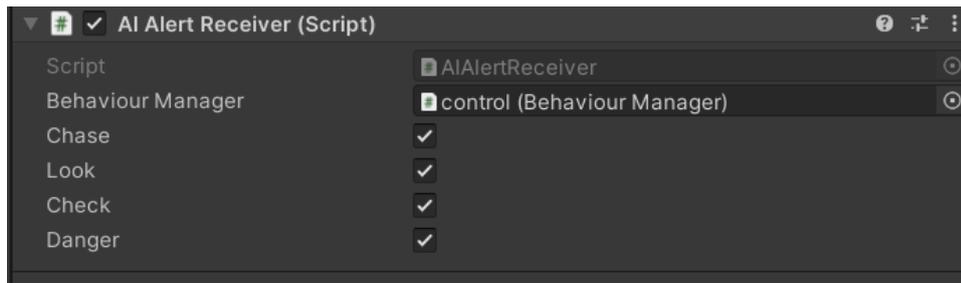
Casting can be done by a few systems

1. AIAAlertMaker – allows to cast 3 alert types (only while patrolling):
 - Check – AI goes to a certain position
 - Look – AI looks at something
 - Chase – AI starts to chase something

If you want to make an AI cast the chase alert to its current target assign the alert maker to the IDetector target changeable array.

2. DangerAlertMaker – casts a danger alert. This will make the AI use its defence system (Dodge/Block)

Receiving is done by the AIAAlertReceiver



Just pick what alerts you wish to receive, and fill in the behaviour manager, and that is it

11. Sound Effects

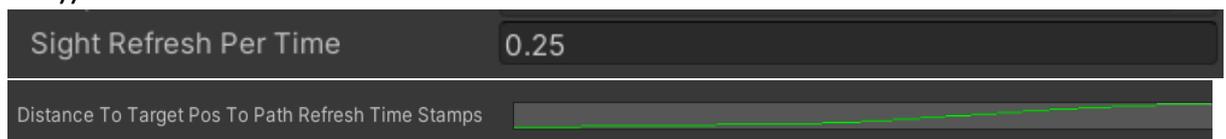
Alek AI has a nice way of playing sound on events, or collisions.

1. Dynamic (on collisions) - requires 2 components:
 - a. dynamicSoundDetection – detects rigidbody collisions
 - b. DynamicSoundPlayer – plays correct sounds requested by the dynamicSoundDetection
2. Events – use EventSoundPlayer and assign proper groups. When you want to play a sound call one of a few public functions available (names explain what they do)

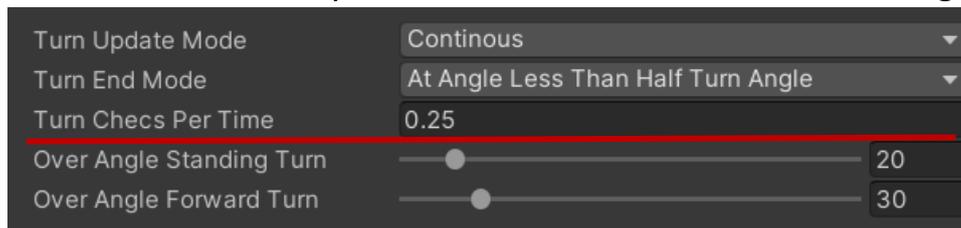
12. Optimization/Quality

There are quite a few settings that can improve the quality with a performance cost, and vice versa.

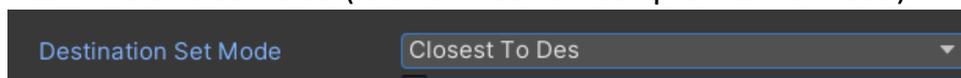
1. Refresh Rates – you can control how often does the AI refresh its detection/navigation system (times/sec). Both settings can be found in corresponding profiles (detection profile/ behaviour profile (movement tab))



- Turning – AI's turning feature is not really that heavy, so you may not see a difference, but you can disable it or how often it is being updated



- Additional Actions – most additional actions use coroutines, so if you are having performance troubles, you might want to consider disabling them. Keep in mind that additional actions make the AI look 200% better, so it might be worth keeping them.
- Target Detection – except for the refresh time, target detection can be optimized differently. If you don't have to, it is better to use preset target detector than the dynamic one.
- Destination set mode (found in behaviour profile main tab)



13. Extending the system on your own

There are many ways to extend the framework with your own code.

Firstly, I will talk about how to link events into the system. Many scripts have simple UnityEvents, which you can use as usual, but some use the IStartEnd interface. Many scripts can be assigned to automatically get the calls, and act on them (any system that can start and end (fe. Aim IK (handAim), DynamicBlocker etc.)). To add events to an IStartEnd event, you can use the StartEndEvents Component. Just assign it into a IStartEnd slot, and fill in the events, and it will work.

The more advanced way to extend the system on your own is replacing certain parts of it using the interfaces. Here is what you can replace by creating your own scripts that use certain interfaces (and how to do so):

- Detection System (Sight) (IDetector) - important thing to note in the detect function

```
public bool detect(bool chasing, out Transform target);
```

, if you return the target as null, the system will not assume you don't see the target (it will just keep the old one), the return value (bool) determines that. If you want to use an optimized version and just change the way of detection, you can use the abstract timedDetector class. It will optimize the call amount and add some cool features (like min sight maintain time) automatically.

- Patrolling Behaviour (IPatrolBehaviour) - each frame you will need to provide a result (set of information of what the AI should do)

```
5 references  
public void update(out Vector3 destination, out bool forceSetDestination, out EnemyBehaviour.actions resultAction, out EnemyBehaviour.movementAction resultMovementAction);
```

the adjustToPosition function is called when the AI should update its values to the current position (as if has just been reset). If your patrol behaviour is still based on waypoints, you can use the abstract wayPointFollower class. it allows you to follow waypoints and stop when you want to.

3. Additional Actions (IAdditionalActionsProvider) - you can generate additionalActionsSettings on your own dynamically, even based on some parameters.
4. Weapons (Attacking) (IWeapon/ IWeaponSwitcher) - the IWeapon interface may be a bit confusing, but the main thing is wantAttack(). If it returns true, and the AI accepts the placementPreferences (is in the correct distance to the target) (GetPlacementPreferences()) the AI will attack. Once it attacks, the Attack function

```
2 references  
public void Attack(bool seeTarget, out attackVisuals? newVisuals);
```

will be called if you decide to "out" the newVisuals, the AI will reset what animation/movement is currently going on in it and start the newly provided one. Additionally, if the AI is not attacking, and the wantToReload(); function returns true, the AI will play the additional action specified in

```
4 references  
public additionalActionsSettings getReloadAdditionalAction(EnemyBehaviour behaviour);
```

similarly with the public bool canParry() function, but that happens during the attack. If the getParryForce() returns something less than the attacks parry force, the AI will play the additional action specified in

```
public additionalActionsSettings getParryAdditionalAction(EnemyBehaviour behaviour);
```

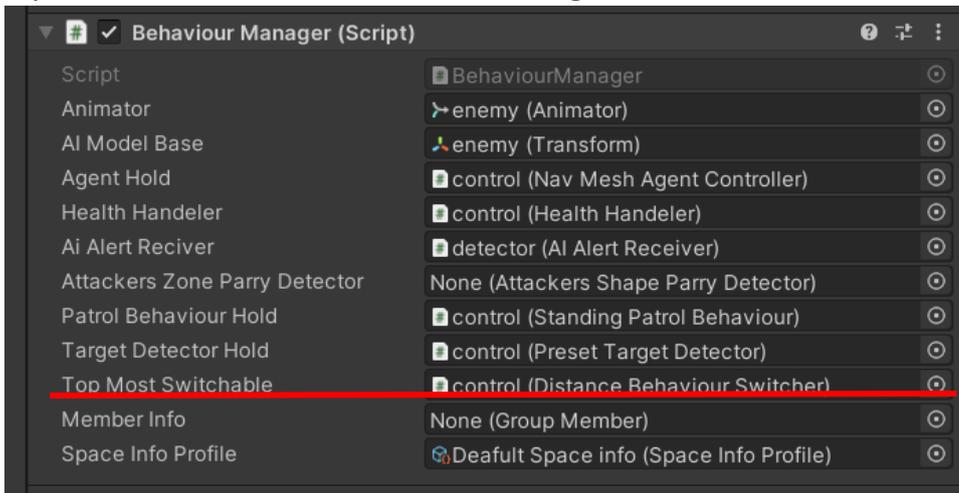
. Regarding the IWeaponSwitcher, just return the wanted weapon.

5. Pathfinding (IAgent) - assign all of your pathfinder values to the interface, and the AI will use your pathfinding system (remember that the AI has its own rotation system)
6. Defences (IDefence) - if you want something custom to happen on danger alert, this is the way. The defend function will provide all the info you will need.
7. target position altering (IPosProcessor) - return the changed position.
8. Blocking certain actions (IProcess) - return the amount of the process that is done, some scripts require all the processes to be finished before doing something.
9. IStartEnd – explained before.
10. Targeted scripts changing target (ITargetChangable) - for example, the default detectors have an array that changes the "target" transform according to the detectors target. All scripts extending this interface, can know exactly what that target is with no need for special references.

14. Specific Module Documentation

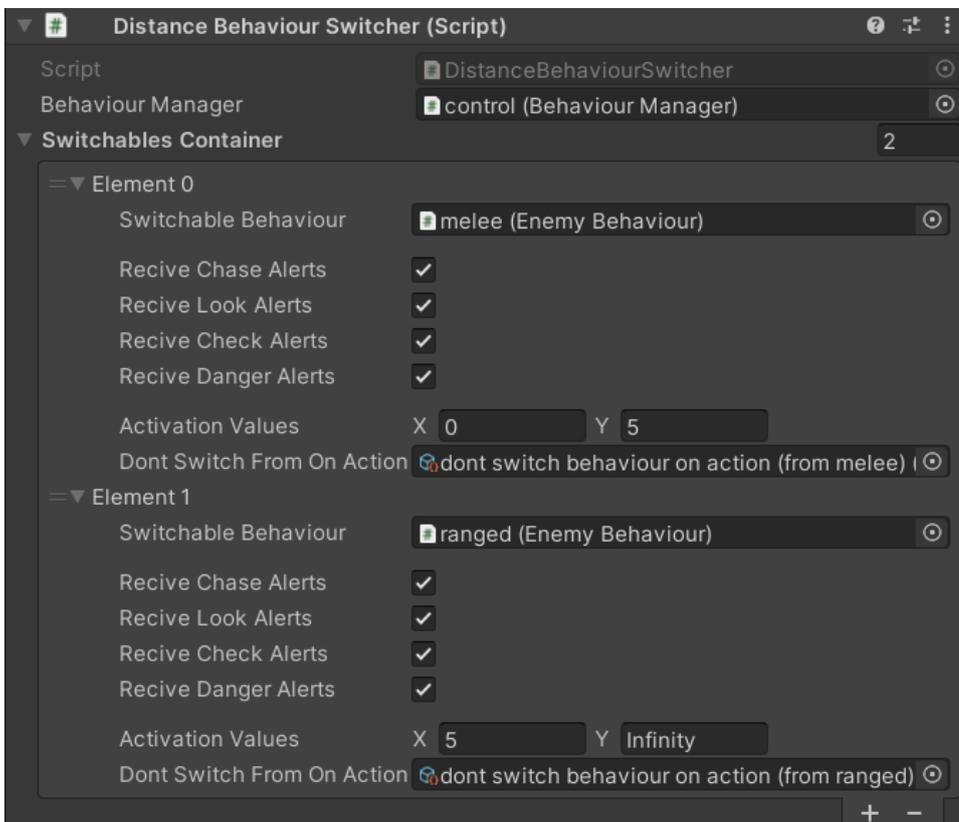
There are many modules available, and even though some were already documented in the previous points, here are specifics of each:

1. Behaviour Switching – to use it, you must have a class extending the ISwitchable interface, like the one included in this module. All you have to do is assign the topmost Switchable in the Behaviour Manager to the Switcher like so:



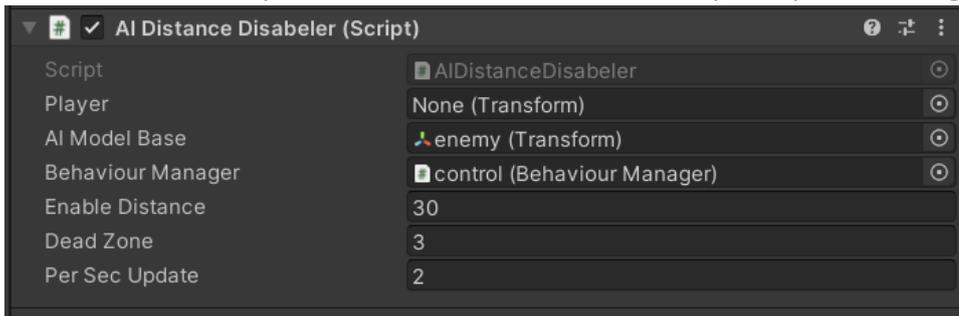
and then

just assign all the behaviours (or another switcher for multilevel switching) to the switcher.

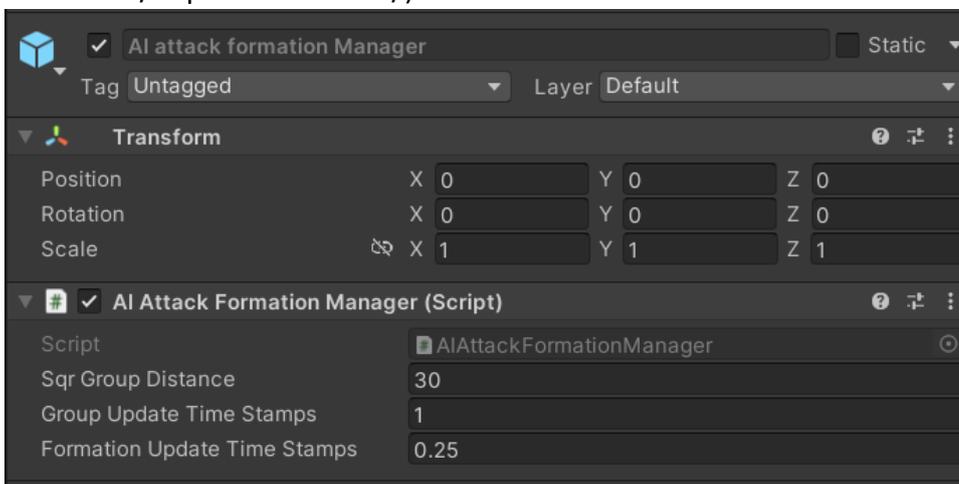


this

module also allows you to disable the AI if it is far away, to optimize the game.

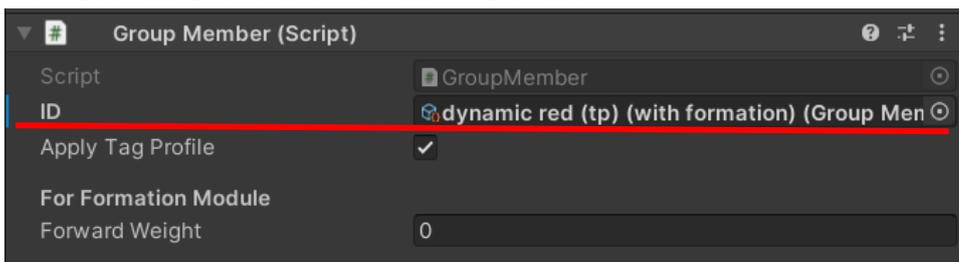


2. Extension Pack – the extension pack contains many additional scripts, all of them where already explained in the previous points of the documentation
3. Formation – the formation module depends on another scene manager apart from the Alek AI Global Manager. The AIAttackFormationManager (you can find it at Formation/Important Prefabs/)

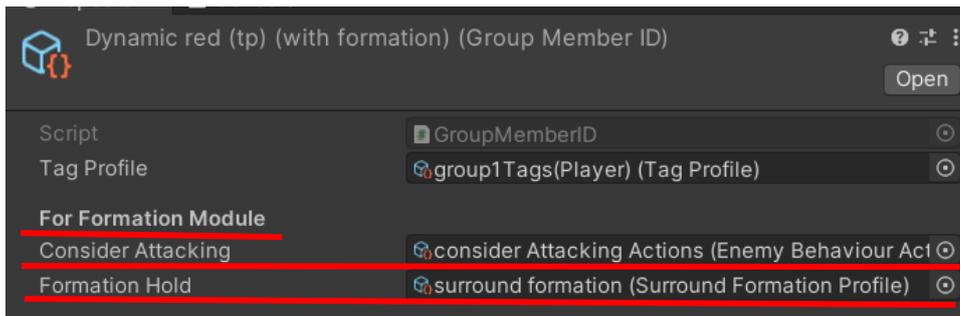


is

responsible for grouping of the AI. It creates groups based on the distances of the AI from each other and the member Info of each AI



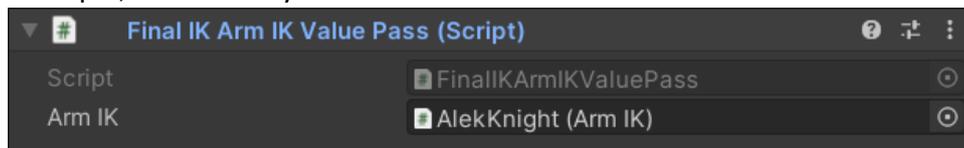
remember to assign the Group Member ID for each AI member Info with all of the "For Formation" values assigned



once you

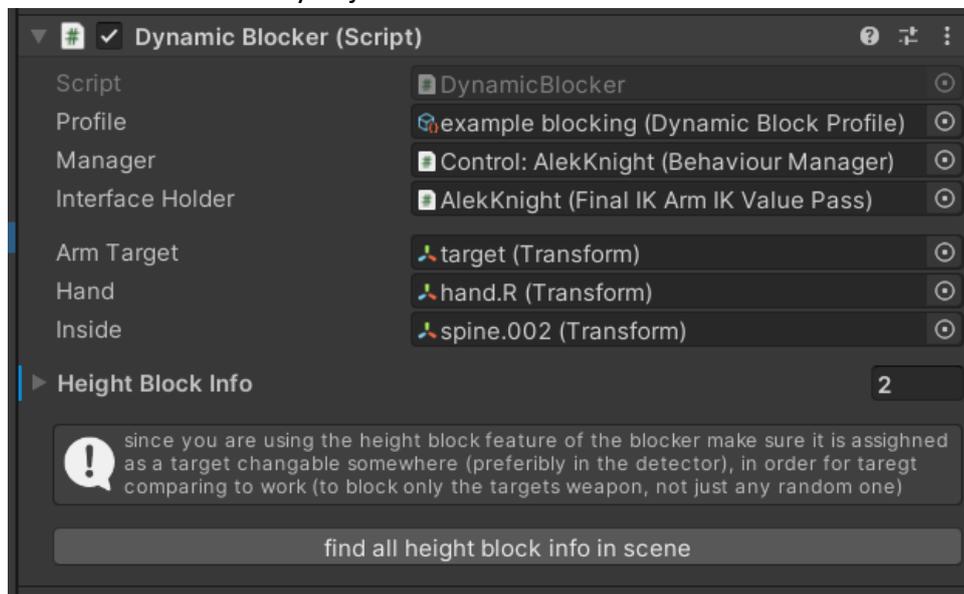
have that ready you should see the formation effects.

4. IK – this module supports 2 IK system by default (Animation Rigging and Final IK) is very simple to use except for the dynamic blocking, so let's split this description into 2 points.
 - a. Dynamic blocking – this system allows the AI to place a weapon in the way of a strike. to set it up you need an arm IK system (go to any YouTube tutorial on how to set that up). Once you have that you will need to add the Dynamic Blocker component as well as an IK value pass (InverseKinematics interface (for example, for Final IK you use:



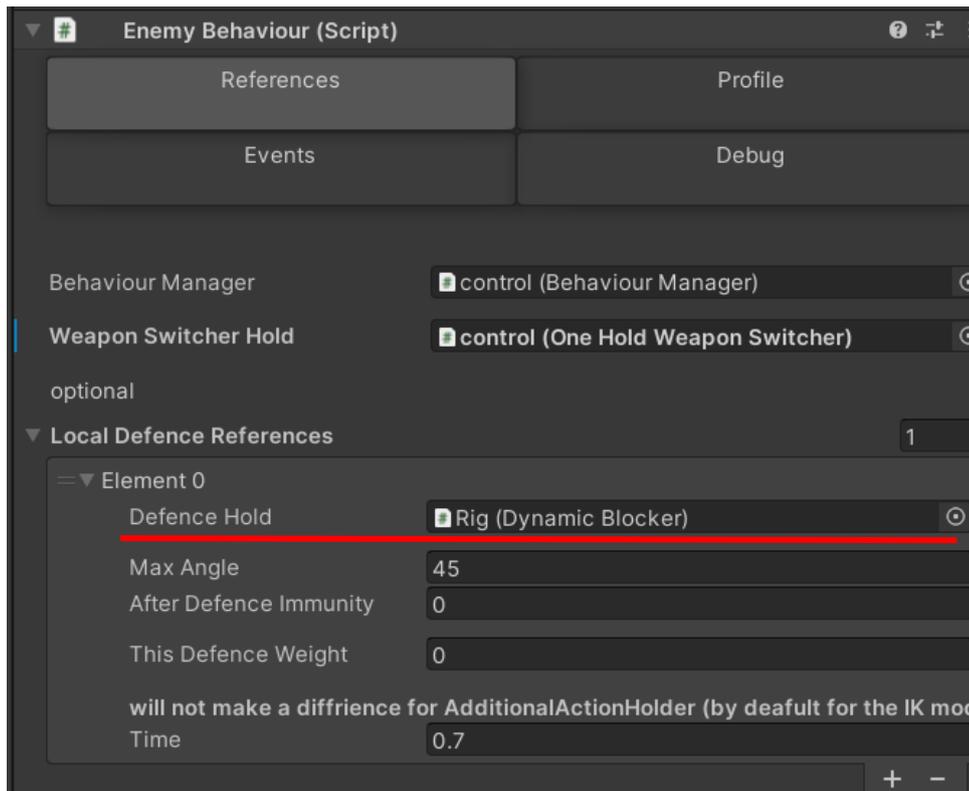
)) you

can add them onto any object.

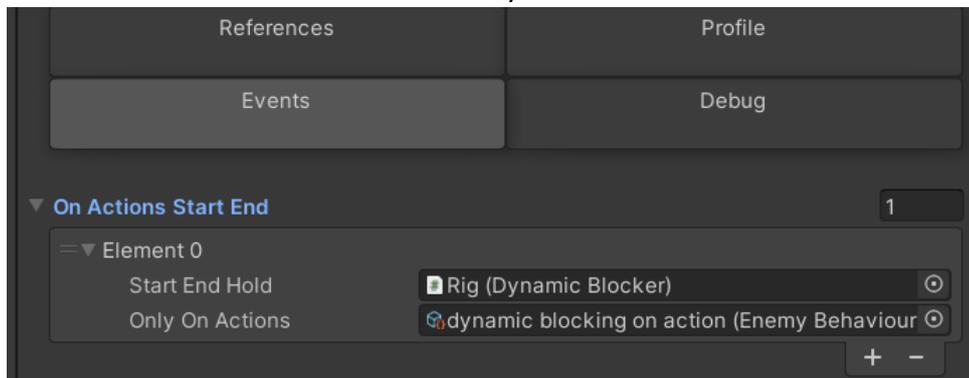


the

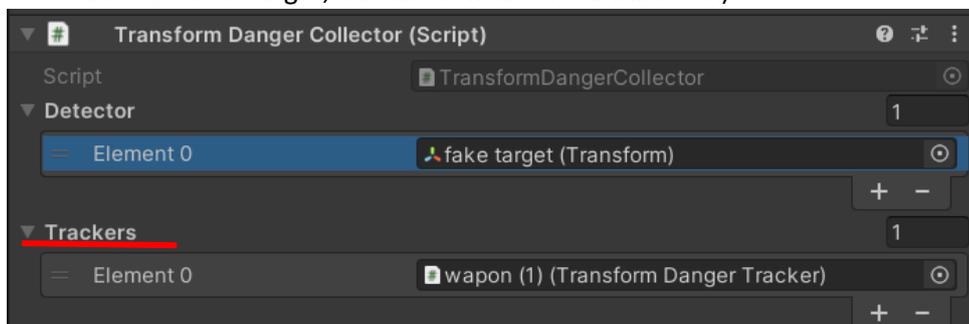
main thing you must set up is the dynamic blocking profile. to make the setup easier please be patient and preferably make your AI stand in place (set speed to 0), to be able to test quite a few times in play mode without interruption. For incoming blocking the next thing to do is assign it to the enemy behaviour. if you what to use the incoming blocking do so as a defence system, so reference it in the local defence references



if you want to use the height block system, you have to do 3 things. Firstly, reference it in the on Action Start end in the enemy behaviour

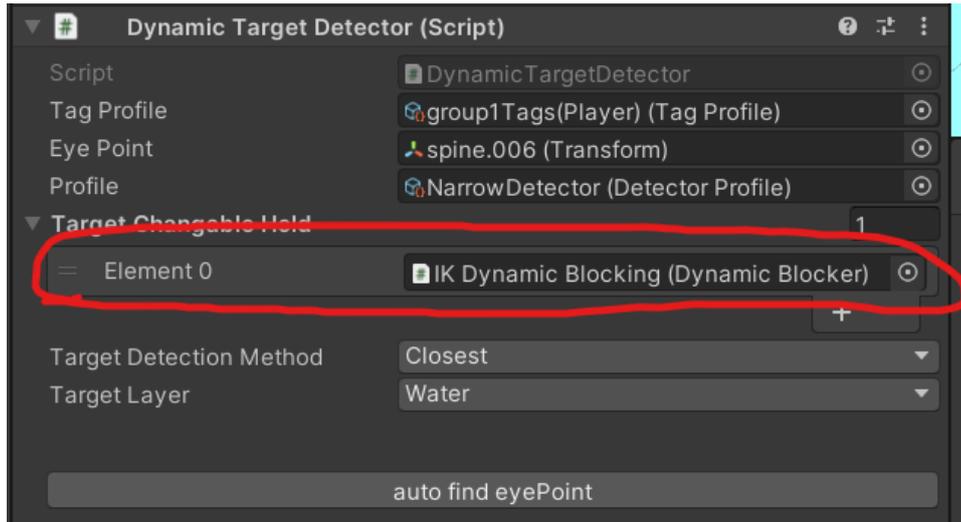


and then you will have to add a transform danger collector. Place it on the target and reference all the transforms that may be detected as target by the AI. (detector is what the AI can detect as target, trackers is what the AI will block)



then just add a transform danger tracker (to the targets weapon, assign the centre as the weapon centre), and reference the tracker in the collector. Then assign the dynamic

blocker as a target changeable in the target detector



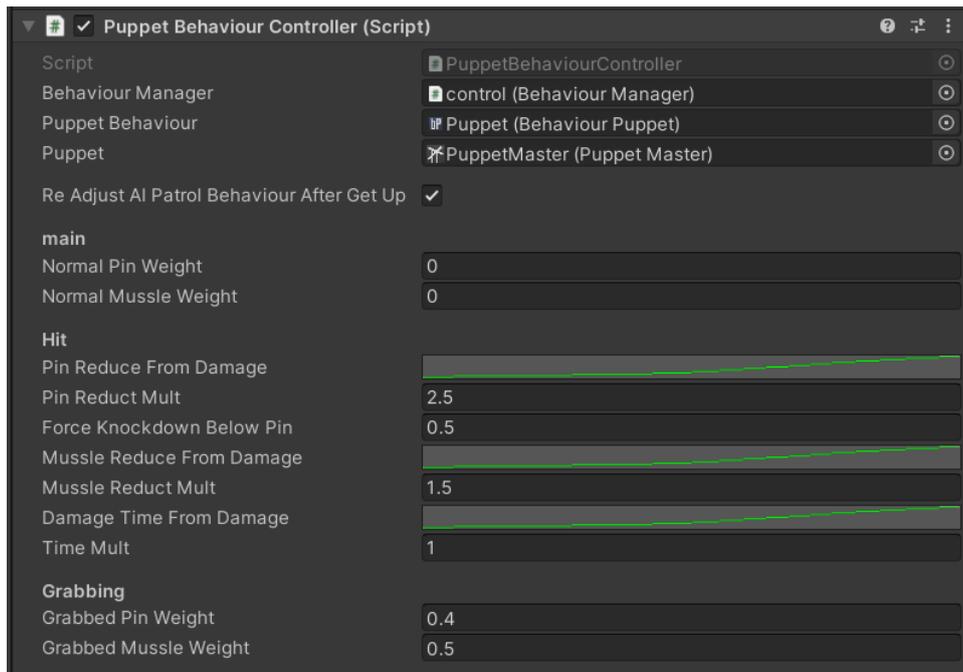
- b. Other – other AI systems just require an IK value pass (InverseKinematics interface) and a way to start them. You should probably use the Events tab of the enemy behaviour. There you can assign something to start and end on a certain behaviour action.



5. Integrations – you can integrate Alek AI Farmwork with almost any system, and this is what this module does. I integrate my asset with other assets requested by the community.
- a. A* ppp – set up the a*ppp on the place of the Nav Mesh Agent and replace the default agent controller with the A* edition of it.

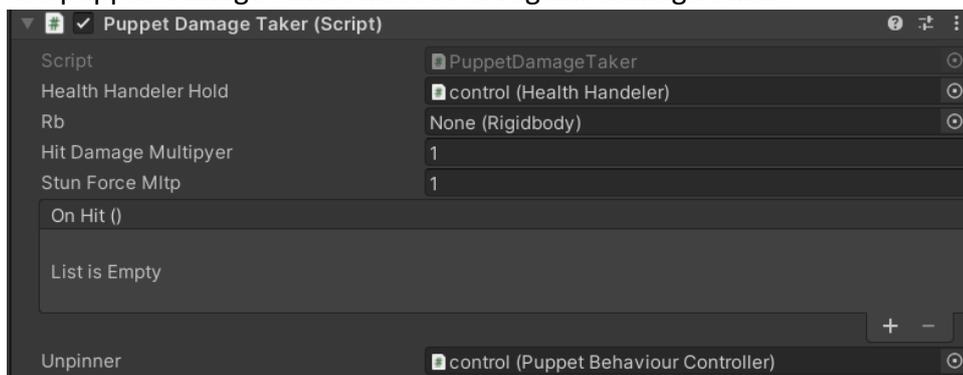


- b. Puppet Master - Honestly this is the most fun thing about the hole unity engine. To set it up, just create the puppet as intended, and add the AI on top of it. The thing you might notice is that there is already a root object, created by the puppet master. To avoid having 2 of those, select the option of creating the AI control that doesn't create a root (in the behaviour maker). The rest of AI creation is the same. Once you crate the AI, you will have to add a Puppet Behaviour controller.



and

use puppet damage takers instead of regular damage takers



And that is it for the documentation. If you wish for more, message me and I will probably add what you want

