V1.0

CONTENTS

CREATING YOUR OWN PARAGON AI AGENT

INITIAL PREPERATION

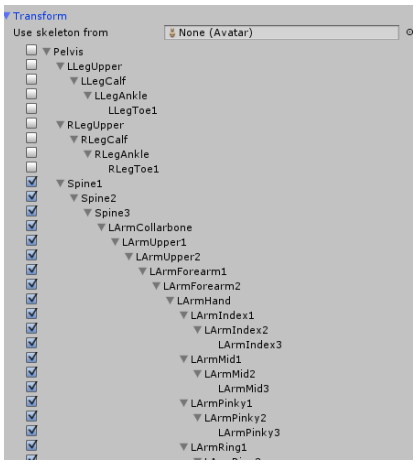
1. Your model must be fully modelled, rigged, and animated.

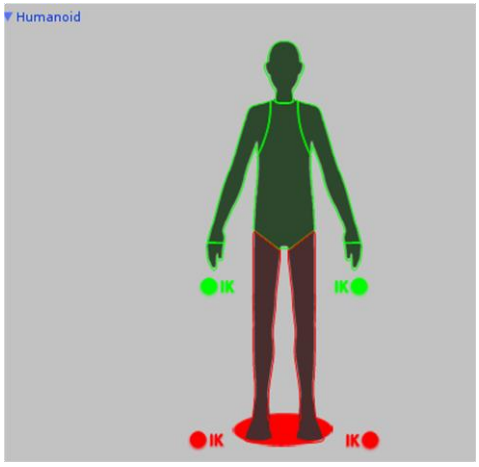
2. You must have created a ragdoll object for your model. The easiest way to do this is usually with the unity ragdoll wizard. The Unity documentation has some pages to help you get through this process.

http://docs.unity3d.com/Manual/wizard-RagdollWizard.html

http://docs.unity3d.com/500/Documentation/Manual/RagdollStability.html

3. You must have created a mask for your model that excludes the legs. This is so that we can play movement animations on the lower body while simultaneously allowing the upper body to play animations such as reloading or firing. It will also allow our agent to aim their gun towards the target. Examples for both generic and humanoid rigs are as follows.





Relevant Documentation: http://docs.unity3d.com/500/Documentation/Manual/class-AvatarMask.html

http://docs.unity3d.com/500/Documentation/Manual/ConfiguringtheAvatar.html

http://docs.unity3d.com/500/Documentation/Manual/class-Avatar.html

3. You must create an "eye" object and make it an object within the ragdoll's hierarchy. The forward axis (represented by the blue arrow in the scene view in local space) must be facing forwards. If you don't get it right though, you can always change it later.

PERPARING THE GUN

If you want your agent to use a gun, the following steps must be taken:

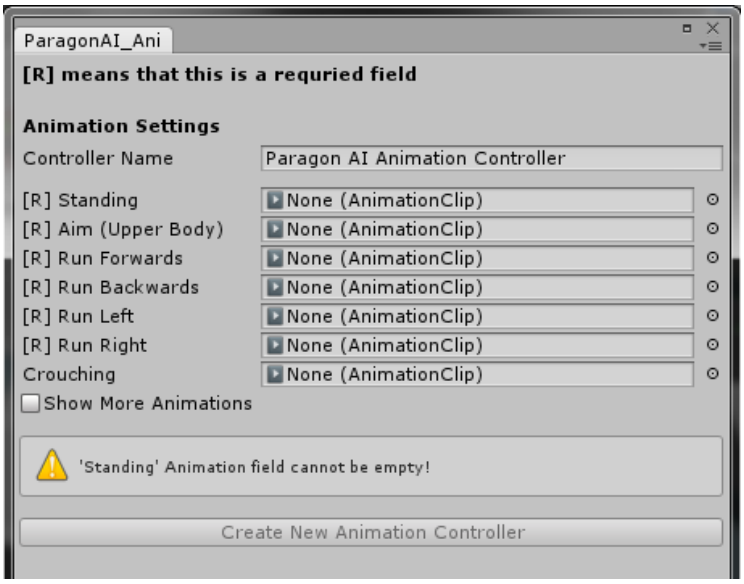
1. You must create a bullet prefab that your agent will fire. You can use one of the default Paragon bullets, or make one on your own.

2. Your agent must have a weapon that is deeper in the hierarchy than the upper torso. Usually, this means that it is a child of the hand.

3. You must create a "bullet spawn" object and make it a child of the gun. The forward axis (represented by the blue arrow in the scene view in local space) must be facing forwards. If you don't get it right though, you can always change it later.

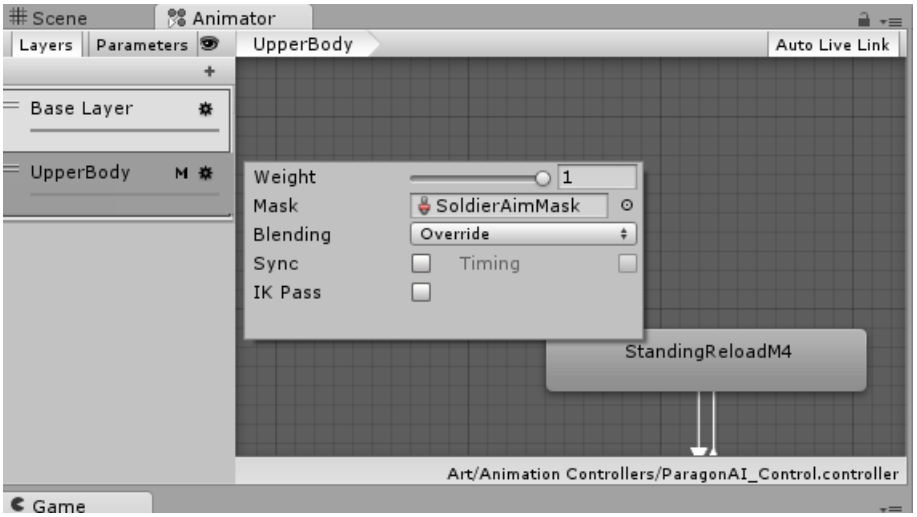
CREATING THE ANIMATION CONTROLLER

1. Navigate to *Assets > Create > Paragon AI Animation Controller.* This should bring up the following window:



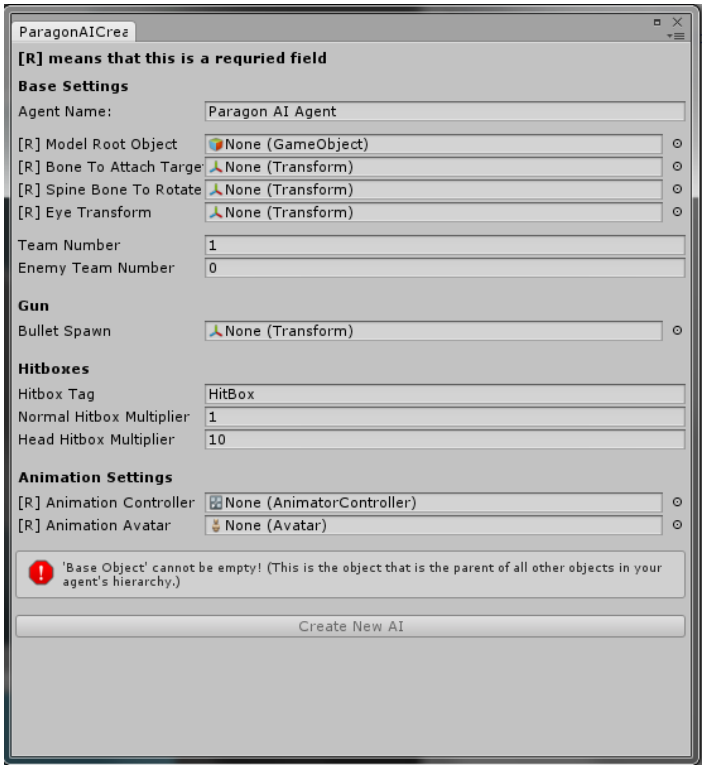
2. Select the appropriate animations and give the controller an appropriate name. Then, click on *Create New Animation Controller.*

3. Finally open your controller and go to the second (lower) layer. Click on the gear. Set its mask to the one we created above and set its weight to 1. Make sure the blending mode is set to "override."



CREATING THE AGENT

1. Navigate to *GameObject > 3DObject > Paragon AI Agent.* This should bring up a window similar to the following:



2. Fill out the appropriate fields. Explanations for each field are as follows:

Agent Name: The name you want to assign to your agent

Model Root Object: The upper-most object in your ragdoll's hierarchy

Bone to Attach Target to: The transform at which other agents will aim at when targeting this agent.

Spine Bone to Rotate: The bone your agent will rotate in order to aim their weapon at their target

Eye Transform: The transform that will be used to determine where your agent performs line of sight checks. The blue arrow.

Team Number: A number that will identify which team this agent is a part of.

Enemy Team Number: One of the numbers that identify which team this agent will be fighting. If you only want them to fire on the player, the default value of 0 should be fine, unless you have changed the team of the player. If you want the agent to fight more than 1 team, you can manually add this later.

Bullet Spawn: The transform at which your bullets will be spawned at

Bullet Object: The prefab that is your enemy's bullet

Bullet Sound: The audio clip that will be played every time your agent fires.

Hitbox tag: The tag that each of your hitboxes will use. Every collider on the object will be transformed into a hitbox.

Normal Hitbox Multiplier: The factor by which incoming damage will be multiplied, by most hitboxes, by default. You can change the value of any individual hitbox later.

Head Hitbox Multiplier: The factor by which incoming damage will be multiplied by the head hitbox

Animation Controller: The animation controller this agent will use.

Animation Avatar: The avatar this agent will use.

When you've filled out the appropriate fields, click *Create New AI*. Make sure to save it as a prefab when you are done.

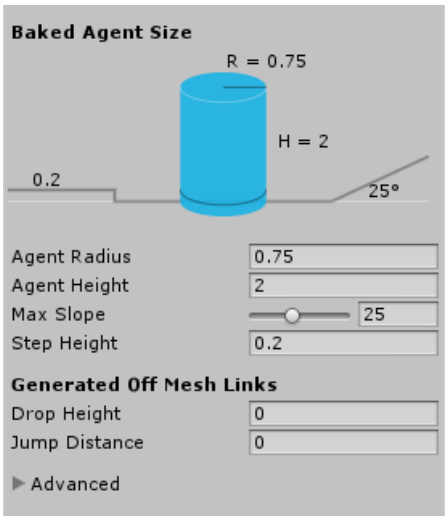
ADDING THE AI CONTROLLER

In order for Paragon AI to work, you'll need an AI Controller in your scene. Simply create an empty Game Object, and add the ControllerScript. Once there, change the layermask variable to include all the layers which your level parts belong to. For best results, the level parts should be on a different layer than your characters or agents.

SETTING UP THE ENVIRONMENT

CREATING THE NAVMESH

In order for Paragon AI to work, you must generate a navmesh. To open the navigation tab, go to *Window>Navigation.* The following window should appear.



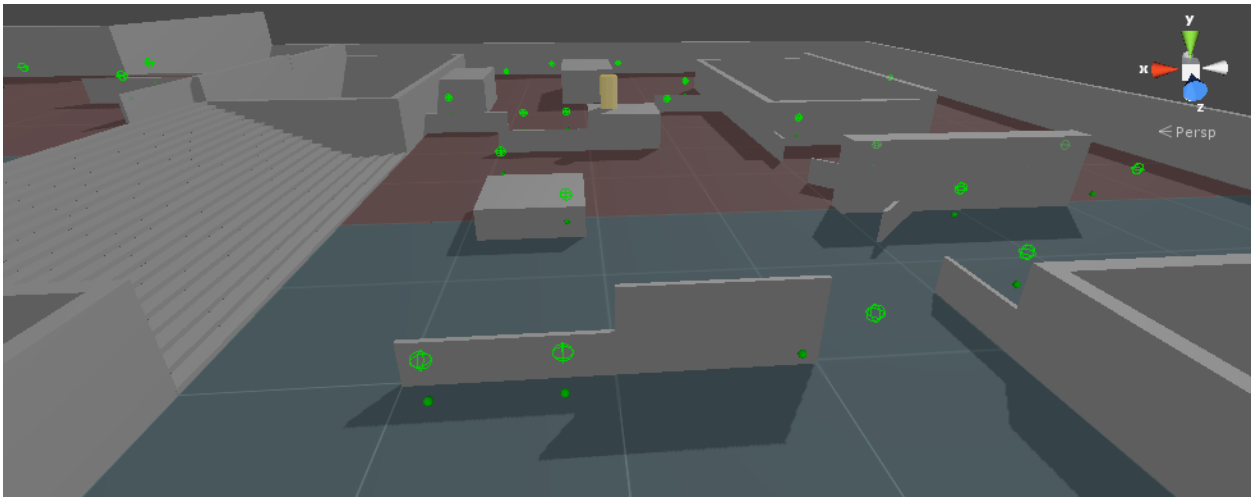
Configure the settings as you see fit. For best results, make sure you go into the *advanced* tab and check *Height Mesh.*

CREATING THE COVER MAP

If you don't want to make a cover map, you can always use the dynamic cover option. However, you won't get the fine-tuning that a custom cover map offers.

While a cover map is not mandatory for use with Paragon AI, it is highly recommended in order to get the best player experience. A cover map consists of manually placed nodes designating positions where your agents can

take cover. The nodes consist of two positions: the spot where your agent will hide, and the spot where they can fire at the enemy. The positions are respectively represented in the editor by a small, solid sphere and a larger wire sphere.



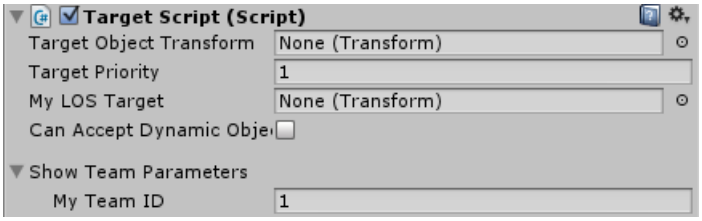
In theory, any gameobject which contains the ParagonAI **CoverNodeScipt** can be a cover node. Paragon AI ships with two prefabs, which should cover 99% of uses. One, the "Short Cover Node" is used for when the agent simply crouches behind a low barrier and stands up without moving laterally to fire. The other, the "Tall Cover Node" is used when you want the AI to move to the side, usually out from behind a tall wall, to fire at the enemy.

It is essential that the firing position (wire sphere) has a clear line of sight to the area you want the agent's target to be in when using the cover node. Conversely, line of sight between the hiding position and the target area should be blocked. The rotation and orientation of the gameobject marking the cover do not matter- only the positions of the firing and hiding positions.

MAKING THE AGENTS ENGAGE/DAMAGE TARGETS

ENGAGING TARGETS

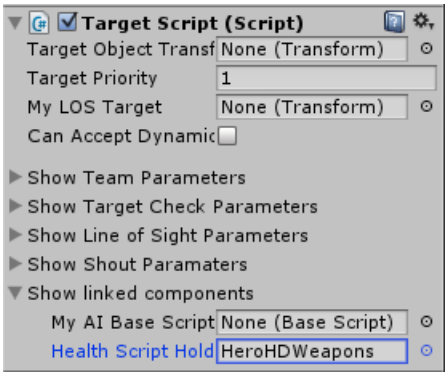
In order to make an agent engage a target in combat, you simply need to add a TargetScript to the appropriate gameobject. Make sure that the *myTeamID* variable corresponds with one of the *enemyTeamNumbers* of your agent. By default, using the default ID of 0 will ensure that the agent will fire at the target.



LETTING TARGETS RECEIVE DAMAGE

In order to allow your target to take damage, you need to make sure the bullets can actually hit their target. Make sure that the bullets's *layerMask* includes that of the objects you want the bullet to hit. Upon hitting a collider, the default Paragon AI bullets will attempt to call a *Damage(float)* method on every script on the impacted object. What you do with this is up to you.

In order to allow an object to receive melee damage, make sure the *Health Script Holder* variable under the Linked Parameters tab is set to the object that contains the script that manages the object's health.



UFPS INTEGRATION

NOTE: The following procedure has been tested with UFPS 1.5.1. Later versions are likely supported, earlier versions are not.

UFPS integration is extremely simple.

1) Search for all the UFPS bullet and explosion prefabs in the project view and set *RequireDamageHandler* to OFF.

2) Make sure that you have added a Paragon AI *TargetScript* an object in your character's hierarchy (The main camera is usually a good choice). For melee to work, you must assign the *TargetScript's healthScriptHolder* variable to the uppermost object in your character's hierarchy (the one with the *PlayerDamageHandler* script). See the above section for more details.

3) Ensure that the layermasks of ParagonAI bullets includes the same physics layer that your player is on. Make sure your Paragon AI explosions have "Should Do Single Hitbox Damage" disabled.

4) Check the layermasks on the ParagonAIController and on the cover nodes. Make sure they include the layers that your level/map objects are on.

Note that upon importing Paragon AI to a UFPS project Unity may randomly swap out animations in the default controller. You will have to manually replace them, if you want to use the default animation controller.

THE DEFAULT BEHAVIOURS

BEHAVIOURS INTRO

Paragon AI's behaviors are the heart of the system. They decide what the agent will do. Some do unique things such as trigger the playing of animations, but all tell the agent where to go. There are three types of behaviors:

Idle Behaviors: The agent performs these behaviors when it is not engaged in combat.

Combat Behaviors: What the agent does when it is actively fighting an enemy.

Override Behaviors: Short-lived behaviors used to allow the agent to react to external factors- for example, running away from a grenade.

IDLE BEHAVIOURS

Wander: One of the simplest idle behaviors. The agent chooses a location within a certain radius of their position and moves to it. Once they near that location, they choose a new one.

Patrol: The agent follows a path marked by a list of transforms held in the **BaseScript.**

Move to Transform: The agent moves to its designated key transform, designated in the **BaseScript**.

Search: The agent actively looks for their target. When the behavior starts, they are given the targets current position. They will then move to this position. When they reach the position designated at the behaviors initialization, their destination will once agent match their targets current position. If the target stays still, they will quickly be found, but a mobile target will be able to avoid detection a while longer.

COMBAT BEHAVIOURS

Berserker: The agent will take the shortest path possible to reach their target. If the agent has a key transform designated in the **BaseScript**, they will move to that transform's position. This behavior is best suited for agents that have little regard for avoiding damage. For example, zombies or heavily armored robots.

Tactical: Tactical agents will make use of cover in order to keep themselves alive and maintain optimal positioning. Most ranged agents will use this behavior. It is suitable for both agents that want to keep at a distance at all times, and those that want to use cover to cover their advance towards a position. The behavior has several parameters in multiple components that can be tweaked to get the best results.

OVERRIDE BEHAVIOURS

Run Away From Grenade: After being passed a warning with a position to move away from and a safe distance, the agent will attempt to run get away from a dangerous object. This behavior ends either after the dangerous object is destroyed, or 3 seconds- whichever comes first.

Investigate Sound: If the agent hears a sound, and they are idle, they will move towards the sounds' position at the **BaseScript**'s designated alert speed.

Use Dynamic Object: The agent will move to a transforms position and face a designated direction. They will then play an animation and call a method on a designated object. The behavior automatically ends when the dynamic object is destroyed, or the behavior is completed.

Dodge: The agent strafes laterally in an attempt to throw off an enemy's aim. This periodically occurs when the agent has line-of-sight to their target.

DYNAMIC OBJECTS

Dynamic objects are an excellent way to make your AI seem smarter. The process of making an agent use a dynamic object is as follows.

1. A script locates an agent, either through tags, pre-made Paragon AI methods, or some other process. It then calls the following method on its target script:

UseDynamicObject(dynamicObjectTransform, dynamicObjectAnimationClipKey, dynamicObjectMethod, requireEngaging)

> ***dynamicObjectTransform***: The transform that marks the position and rotation that the agent will move to and match before using the object.
>
> ***dynamicObjectAnimationClipKey***: The parameter in the agent's animation controller that will be triggered in order to play the animation. If the parameter is not found, no animation will be played.
>
> ***dynamicObjectMethod***: The method that will be called on the object that the **dynamicObjectTransform** belongs to.
>
> ***requireEngaging***: A Boolean value. If true, the agent must be engaging a target to use the dynamic object.

This method returns a Boolean value which is *true* if it succeeds in making the agent try and use the dynamic object. Otherwise, it returns *false*.

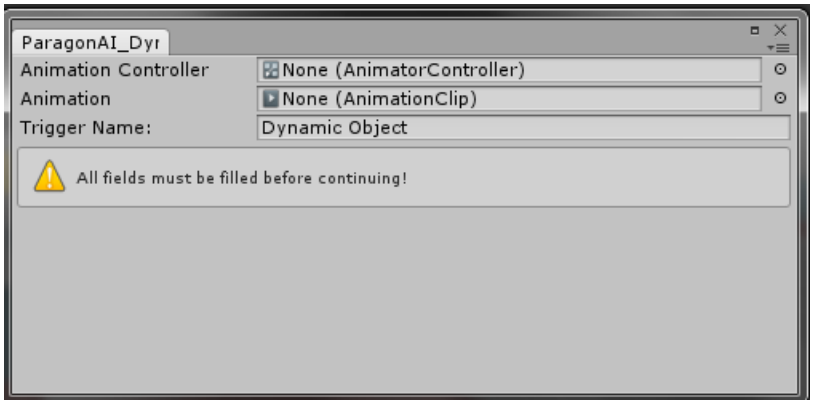
2. The agent moves to and attempts to match the **dynamicObjectTransform**'s position and rotation.

3. The agent attempts to set the animation trigger specified by ***dynamicObjectAnimationClipKey*** *t*o true. The animation is played, if it exists. At the same time the ***dynamicObjectMethod*** is called on the ***dynamicObjectTransform****'s* object. No parameters can be passed through this method.

4. The agent returns to its standard behaviour.

SETTING UP AN ANIMATION CONTROLLER FOR DYNAMIC OBJECT ANIMATIONS

Navigate to *Assets>Create>Add A Paragon AI Dynamic Action.* The following window should appear. Fill out the appropriate fields, and click "Add Action." **You must have a unique Trigger Name for each action!**



CUSTOM BEHAVIOURS

Sometimes, you'll want your agents to do something that is not covered by the default behaviors. To do this, you'll be able to create your own behaviors by extending the **ParagonAI.CustomAIBehaviour** class. By default, it should look something like the following:

```
using UnityEngine;
using System.Collections;

namespace ParagonAI{
public class CustomAIBehaviourTemplate : ParagonAI.CustomAIBehaviour{


    public override void Initiate()
        {
            base.Initiate();
        }

    public override void AICycle()
        {

        }

    public override void EachFrame()
        {

        }

    public override void OnEndBehaviour()
        {

        }
}
}
```

INHERITED METHODS

Initiate() : Automatically called when the behavior begins. If you want all references to be automatically set, you need to call base.Initiate(); in this method.

AICycle() : Automatically called every AI cycle. The frequency can be modified in the **BaseScript**.

EachFrame() : Automatically called every frame.

OnEndBehaviour() : Automatically called when the AI changes its behavior, or dies.

ApplyBehaviour(): Call this method to make the **BaseScript** use this behaviour. The **BaseScript** in question must be attached to the same object as the behavior.

KillBehaviour(): Call this method to destroy this behavior. Automatically calls **OnEndBehaviour().**

INHERITED VARIABLES

targetVector: This automatically provides the BaseScript with a destination to move to.

behaveLevel: Whether this is an Idle or a Combat Behavior.

baseScript: This behavior's **BaseScript**

gunScript: This behavior's **GunScript**

soundScript: This behavior's **SoundScript**

rotateToAimGunScript: This behavior's **RotateToAimGunScript**

animationScript: This behavior's **AnimationScript**

coverFinderScript: This behavior's **CoverFinderScript**

myTransform: The transform of the agent this behavior is attached to.

agent: This agent's NavMeshAgent component

layerMask: The layermask used by this agent for line of sight purposes

HOW TO MAKE AN AGENT USE A CUSTOM BEHAVIOUR

To make an agent use a behavior via the editor:

1. Add your custom behavior component to the same object that holds the baseScript of your agent.
2. In the inspector for your custom behavior, set behaveLevel to either Idle or Combat, depending on what you want it to do.
3. In the inspector for the baseScript:

   If your behavior is an Idle Behavior, set the myIdleBehaviour variable to "*Custom*".

   If your behavior is a Combat Behavior, set the myAIType variable to "*Custom*".

To change an agent's idle or combat behavior to a custom one during runtime:

1. Add the custom behavior component to the same object that holds the baseScript of your agent. (this can be done during runtime with another script or in the editor). You can use the following method during runtime:
2. During runtime, set the behavior's behaveLevel to either Idle or Combat
3: Call the **ApplyBehaviour()** method on your behavior component.

Here's an example of the code required:

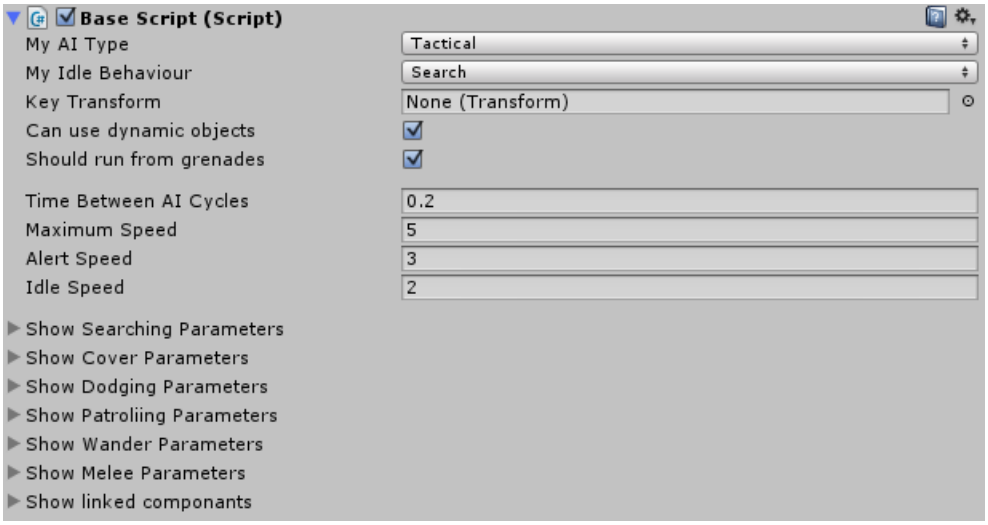
```
ParagonAI.CustomAIBehaviour newBehave = (ParagonAI.CustomAIBehaviour)t.gameObject.AddComponent(typeof(ParagonAI.CustomAIBehaviourTemplate));
newBehave.behaveLevel = ParagonAI.CustomAIBehaviour.BehaviourLevel.Combat;
newBehave.ApplyBehaviour();
```

AN AGENT'S SCRIPTS

Note that some scripts use custom inspectors. As such, they may not update in real-time during gameplay. You will have to set the inspector to run in debug mode in order to see the variable's real-time values.

BASE SCRIPT

This script is the heart of the system; it holds everything else together. It's primarily used to change behavior parameters.



VARIABLES

My AI Type: Determines the AI's behavior in combat.

My Idle Behavior: Determines the AI's behavior when not engaged in combat.

Key Transform: A transform used by a number of behaviors.

Can use dynamic objects: Whether or not this agent can use dynamic objects.

Time between AI Cycles: The time in between AICycle calls on behaviors. A low number will give higher quality behaviors, but a higher number will offer better performance.

Maximum Speed: The maximum speed the agent can move at. This speed will be reached in active combat.

Alert Speed: The speed the agent will move at when they are aware of an enemy, but don't know their exact position.

Idle Speed: The speed the agent will move at when they are completely unaware of an enemy.

Radius to call off search: How close the agent needs to get to the target position, before setting its target position to the agent's current position.

Time between cover safety checks: How often the agent will check to make sure their current cover location is safe.

Maximum time in cover: The maximum time the agent will spend in cover before choosing a new location.

Minimum time in cover: The minimum time the agent will spend in cover before choosing a new location. The exact time will be a randomly chosen number between the minimum time and maximum time. The agent may leave early due to external factors, such as the location not being deemed safe anymore.

Should dodge: Whether the agent should try and dodge

Dodging speed: The speed the agent will move at when dodging.

Dodging time: How long each dodge will take.

Dodging Clear Height: When trying to dodge, the agent will check to see if the direction of their strafe is clear. This variable determines how far off the ground obstructions can be without preventing a dodge from occurring.

Time between dodges: The minimum time between dodges.

Minimum distance from target to dodge: The closest distance the agent can be to their target and still dodge,

Close enough to patrol node distance: How close the agent has to get to a patrol node before proceeding to the next one.

Patrol Nodes: An array that contains the agents patrol nodes. The agent will start at the first node on the list and work their way down. At the end, they will cycle back to the first node. Patrol nodes are represented by transforms. It's recommended you use empty game objects for this purpose.

Show patrol path: Whether or not the patrol path should be visualized in the editor.

Wander Diameter: The diameter of the area in which a random wander destination can be chosen. The center is the agent's current position, or their key transform's position, if the agent has one.

Dist to choose next wander point: How close the agent must get to the current wander destination to choose another one.

Can melee: Whether or not this agent can perform melee attacks.

Melee damage: How much damage is inflicted by each melee attack.

Time between melees: How long an agent must wait between melee attacks.

Melee range: How close an agent must be to a target in order to perform a melee attack.

Time until damage: How long after the animation starts the damage should be dealt. This will allow you to sync up the damage with the actual strike in the animation.

Gun Script: This agent's **GunScript**.

Audio Script: This agent's **AudioScript**.

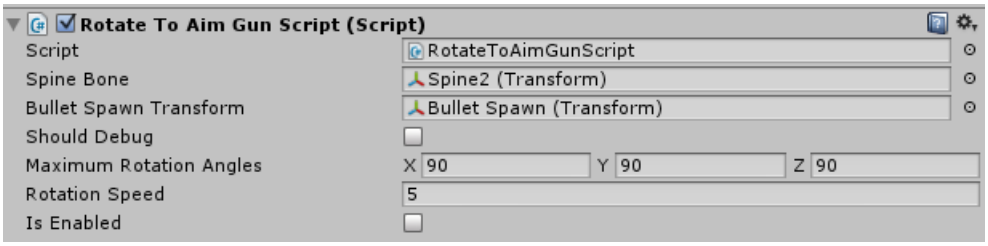
Head Look Script: This agent's **RotateToAimGunScript**.

Animation Script: This agent's **AnimationScript**.

Cover Finder Script: This agent's **CoverFinderScript**.

ROTATE TO AIM GUN SCRIPT

This script rotates a bone in the agent's spine so that they aim their gun at the target.



VARIABLES

Spine Bone: The bone the AI should rotate in order to aim their gun towards the target.

Maximum Rotation Angles: How far the agent can rotate their spine

Bullet Spawn Transform: The transform the AI should orient towards their target.

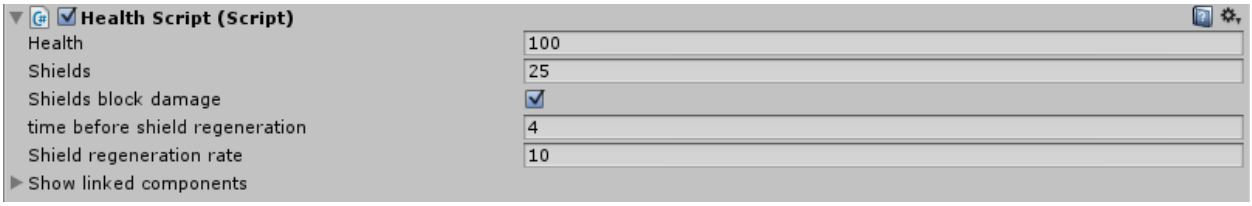
Should Debug: Whether or not the Scene Window should visualize debug information. The blue line points directly from the gun spawn to the target. The red line shows the gun's actual forwards vector. Note that, while the two should get close, they will likely never perfectly line up.

Rotation Speed: How fast the agent can rotate towards their target.

Is Enabled: Whether or not the agent should rotate so that the gun aims towards the target.

HEALTH SCRIPT

This script controls everything health related.



VARIABLES

Health: How much health the agent has. When this reaches zero, the agent will die.

Shields: A second layer of health. When this reaches zero, the agent will behave more defensively until they regenerate. For example, they will not move out of cover to fire at enemies.

Shields block damage: Whether or not the shields must be at zero before damaging health. If true, damage will be applied to the shields until they are down, and then damage will be applied to health. If false, damage will be applied to both the shields and the health at the same time. The only purpose the shields will serve is to determine when the agent should act more defensibly.

Time before shields regeneration: How long the agent must wait after taking a hit before regenerating. The time will reset every time the agent is hit.

Shield regeneration rate: How many units of shields per second the agent will regenerate.

My AI Base Script: This agent's **BaseScript**.

My Target Script: This agent's **TargetScript**.

Rigidbodies: An array containing the rigidbodies that are part of this agent's ragdoll.

Colliders to Enable: An array containing the colliders that should be enabled on this agent's death. Good for gun models and other objects that shouldn't be hit by bullets but should use physics upon death.

Sound Script: This agent's **SoundScript**.

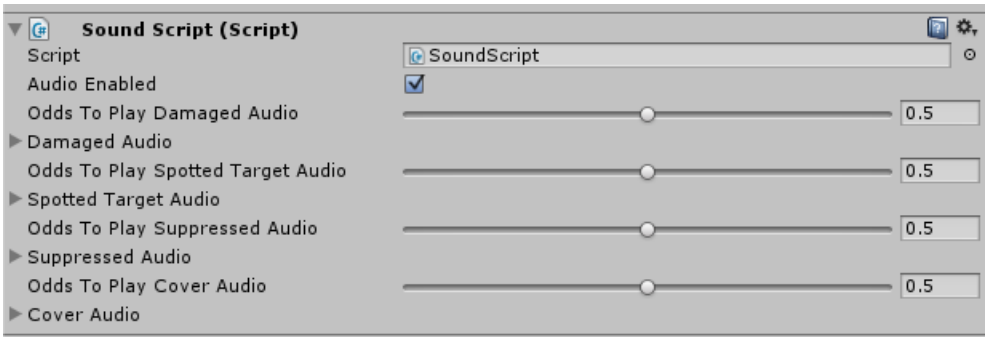
Rotate To Aim Gun Script: This agent's **RotateToAimGunScript**.

Animator: This agent's animator.

GunScript: This agent's **Gunscript**.

SOUND SCRIPT

This script will let you play various sounds during several events during an AI's life cycle.



VARIABLES

Audio Enabled: Whether or not this script can play audio.

Odds to Play Damaged Audio: Odds of the agent playing any damaged audio clip when they receive damage.

Damaged Audio: A list damaged audio clips and the odds they will be played. A higher number means a greater chance the clip will be played.

Odds to Play Spotted Audio: Odds of the agent playing any spotted audio clip when they see a target.

Spotted Audio: A list spotted audio clips and the odds they will be played. A higher number means a greater chance the clip will be played.

Odds to **Play Suppressed Audio**: Odds of the agent playing any suppressed audio clip when their shields reach zero.

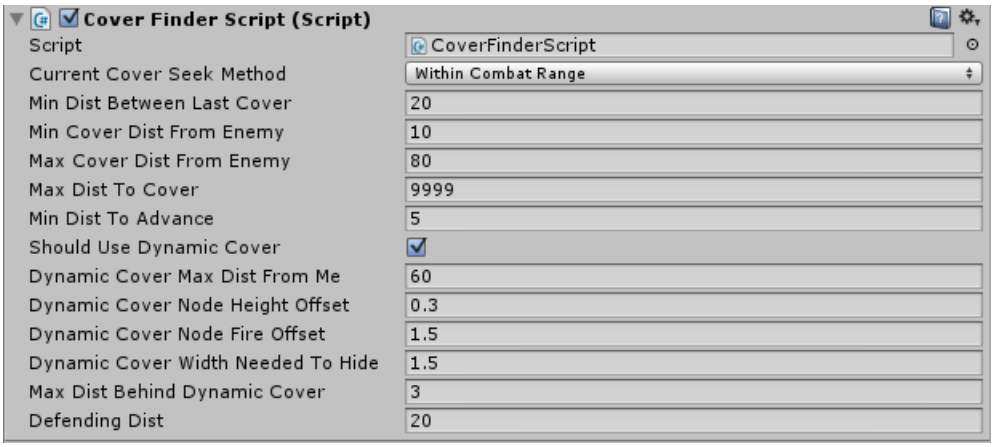
Suppressed Audio: A list suppressed audio clips and the odds they will be played. A higher number means a greater chance the clip will be played.

Odds to Play Cover Audio: Odds of the agent playing any cover audio clip when they are using cover.

Cover Audio: A list cover audio clips and the odds they will be played. A higher number means a greater chance the clip will be played.

COVER FINDER SCRIPT

This script is used to find suitable cover locations.



VARIABLES

Current Cover Seek Method: The method that will be used to find cover. There are three options to choose from.

> *Random Cover:* The agent will pick a completely random piece of valid cover. If this agent's baseScript has a keyTransform, the agent will attempt to find cover that is closer to it than this agent's defendingDist.
>
> *Within Combat Range:* The agent will pick the closest piece of valid cover to them that is within their designated combat range. If this agent's baseScript has a keyTransform, the agent will attempt to find cover that is closer to it than this agent's defendingDist.
>
> *Advance Towards Target:* The agent will generally move in the direction of the target, taking cover along the way. If this agent's baseScript has a keyTransform, the agent will attempt to advance towards this position. This method DOES NOT work with dynamic cover.

Min Dist Between Last Cover: The minimum distance an agent must move after leaving a cover location. Only used with the "*WithinCombatRange*" cover seeking method.

Min Cover Dist From Enemy: The minimum distance cover mist be between the agent and their target. Only used with the "*WithinCombatRange*" cover seeking method.

Max Cover Dist From Enemy: The maximum distance cover mist be between the agent and their target. Only used with the "*Within Combat Range*" and "*Advance Towards Target*" cover seeking methods.

Max Dist To Cover: The maximum distance cover can be from the agent. Only used with the "*WithinCombatRange*" cover seeking method.

Should Use Dynamic Cover: Whether not this agent can dynamically find cover.

Dynamic Cover Max Dist From Me: How far this agent can travel to find dynamic cover. Generally a shorter distance than **MaxDistToCover**.

Dynamic Cover Node Height Offset: How high off the ground line of sight checks for cover safety should be performed. This should be roughly as high as your character's head when crouching.

Dynamic Cover Node Fire Offset: How high off the ground line of sight checks for cover firing should be performed. This should be roughly as high as your character's weapon when standing.

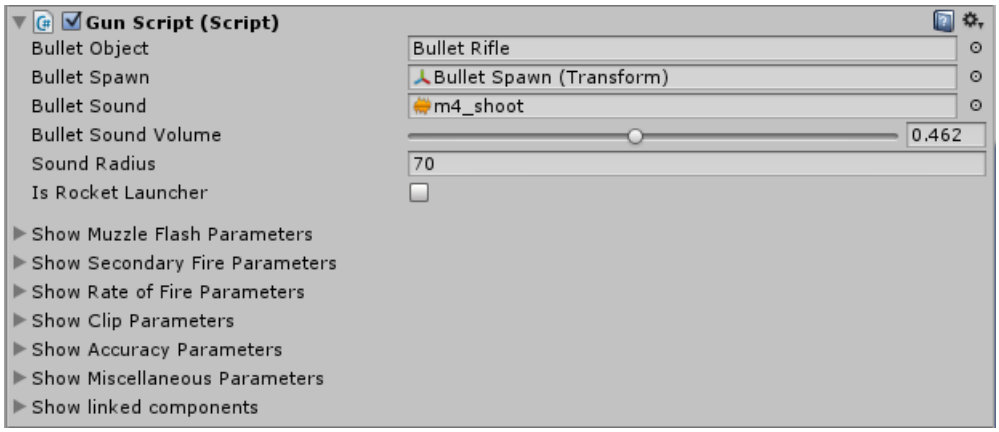
Dynamic Cover Width Needed To Hide: When using tall cover, how far to the side the agent must move to be safe. Should be a little wider than the width of your agent

Max Dist Behind Dynamic Cover: How far behind a line-of-sight blocking object your agent can be at most.

Defending Dist: How close to the agent's Key Transform your agent should take cover when using the "*Within Combat Range*" combat mode or dynamic cover.

GUN SCRIPT

This script manages everything related to firing the AI's weapon.



VARIABLES

Bullet Object: The prefab your agent will create whenever they fire a bullet.

Bullet Spawn: A transform that marks the position and rotation of your bullet.

Bullet Sound: The sound your weapon will make when firing.

Bullet Sound Volume: How loud this bullet sound will be.

Sound Radius: How close other agents must be in order to hear this agent's weapon firing.

Is Rocket Launcher: If your bullet should home in on its target. Only for use with the Paragon AI **BulletScript**.

Muzzle Flash: The prefab that will be used as this agent's muzzle flash.

Flash Spawn: A transform that marks the position and rotation of your muzzle flash.

Flash Duration: How long before the flash is destroyed.

Secondary Fire Object: The prefab used for this weapon's secondary fire.

Secondary Fire Odds: The odds your agent will fire their secondary fire each volley.

Min dist for secondary fire: The minimum distance from the target your agent must be to use their secondary fire.

Max dist for secondary fire: The maximum distance from the target your agent can be to use their secondary fire.

Need Line of sight for secondary fire: Whether or not this agent needs a clear line of sight to the target to use their secondary fire.

Min time between secondary fire: The minimum time your agent must wait in between using their secondary fire.

Min time between volleys: The minimum amount of time this agent must wait in between volleys.

Max extra time between volleys: The maximum amount of extra time this agent must wait in between volleys, in addition to the min time between volleys. The actual number after each volley will be randomly selected.

Min rounds per volley: The minimum number of rounds this agent will fire each volley.

Max rounds per volley: The maximum number of rounds this agent will fire each volley.

Rate of Fire: How many bursts per second this agent will fire. If there is only 1 shot per burst, this is the only variable that determines rate of fire.

Burst rate of Fire: How many rounds will be fired per second. Only used if the number of shots per burst is greater than 1.

Shots per Burst: How many rounds this agent will fire per burst. Most weapons will leave this number at 1.

Pellets per Shot: How many bullet objects will be simultaneously instantiated when this agent fires a shot.

Clip Size: How many rounds this agent can fire before reloading.

Reload Sound: The sound that will play when this agent is reloading.

Reload Sound Volume: The volume of this agent's reloading sound.

Reload Time: How long it takes this agent to reload.

Inaccuracy: How accurate the weapon is. Lower numbers are more accurate, higher numbers are less accurate.

Max Firing Angle: How many degrees off angle the bullet spawn can be and the bullets will still aim themselves towards the target. This is used because the **RotateToAimGunScript** won't aim the gun directly at the target, but rather slightly off-angle.

Max Secondary Firing Angle: How far off angle the bullet spawn can be and this agent will still use their secondary fire.

Time between line of sight checks: How often this weapon will check to make sure they have line of sight.

Dist allowed in front of target for cover: How far behind cover can be and the agent will still fire at them. This is so that they can lay down suppressing fire, even if they don't have a clear line-of-sight.

Cover Transition: Extra time this agent should wait when firing from cover to account for moving out from behind cover.

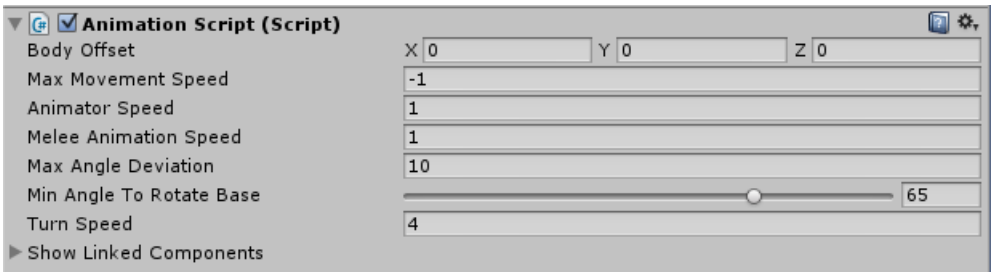
My AI Base Script: This agent's Base Script.

My Animation Script: This agent's animation script.

Audio Source: This agent's audio source.

ANIMATION SCRIPT

This script manages the animations and rotation of the agent.



INCOMING METHODS

SetMyTarget(Transform currentEnemyTransform, Transform losTargetTransform): Sets the target for this script. Note this will be overwritten by the Target Script.

SetIdleBehaviour(CustomAIBehaviour c): Changes the agent's Idle Behavior to c.

SetCombatBehaviour(CustomAIBehaviour c) : Changes the agent's Combat Behavior to c.

OUTGOING METHODS

OnAIDeath(): Called on every script that shares the same object as the Base Script when the AI is killed.

VARIABLES

Body Offset: Used to move your agent's model so that their feet are on the ground. Used if your agent's origin is not at their feet.

Max Movement Speed: The speed at which your agent would be moving at if they were running. If this number is negative, it will automatically match your agent's maximum speed. This is used in case you don't want your agent to be able to run at full speed.

Animator Speed: The speed at which your animations play.

Melee Animation Speed: The speed at which your melee animation will play.

Min Angle Deviation: The maximum angle deviation between your agent and a dynamic object they are trying to use. They will attempt to rotate such that their deviation is less than this angle.

Min Angle To Rotate Base: The minimum angle between your movement vector and the vector pointing from this agent towards their target for your agent's body to directly face their target. This is because blending between side-stepping and moving straight forward/backwards often doesn't work well at 45 degree angles.

Turn Speed: How fast your agent can rotate their body.

My Base Script: This agents **BaseScript**

My AI Base Transform: The transform that contains the agents model.

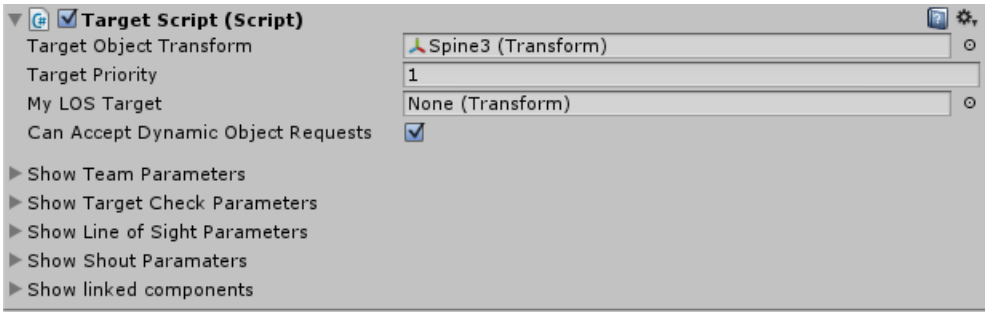
Gun Script: This agents **GunScript**.

Animator: This agents Animator.

Rotate Gun Script: The agents **RotateToAimGunScript**.

TARGET SCRIPT

This script chooses targets for agents. It can also be used to mark a target for other agents to shoot at- even if that target is not an agent itself (such as the player). **Many variables in this script are only used if this target is attached to an agent.**



INCOMING METHODS

WarnOfGrenade(Transform grenadePosition, float distanceToRun): If allowed, the agent will attempt to move away from the given transform.

VARIABLES

Target Object Transform: The transform that other agents will aim at when firing on this target.

Target Priority: A greater number increases the likely hood that other agents will focus fire on this target.

My LOS Target: The transform that determines where other agents will aim their line of sight checks. If null, this will automatically match the Target Object Transform.

Can Accept Dynamic Object Requests: Whether or not this target can take dynamic object requests and pass them on to its **BaseScript**.

My Team ID: A number that identifies which team this agent is on.

Allied team IDs: Teams this agent is allied with. They will alert them to enemies, for example.

Enemy team IDs: Teams this agent will fight against.

Dist To Lose Awareness: The agent will keep track of the last position they saw a target. If a target has moved far enough away from that position (as specified by thus variable), the agent will be able to lose awareness of that target.

Time before Target Expiration: After the agent sees a target, a timer will start at this value. When it reaches zero, if the agent doesn't have a clear line of sight to the target, they will lose awareness of that target.

Time between target checks if engaged: How often the agent will reevaluate its target selection when actively engaged in combat.

Time between target checks if not engaged: How often the agent will reevaluate its target selection when not engaged in combat.

Will Ever Lose Awareness: Whether or not the agent will ever lose a target after spotting them.

Time between LOS Checks: How often the agent will check to see if it can see any targets.

Max Line Of Sight Checks Each Frame: How many targets the agent can check each frame.

Eye Transform: A transform that marks the position and orientation of their field of view.

My field of view: How wide this agent's field of view is.

Debug Field of View: Show the agent's field of view each LoS check.

Shout Dist: How close allied agents must be for this agent to alert them when it spots an enemy target.

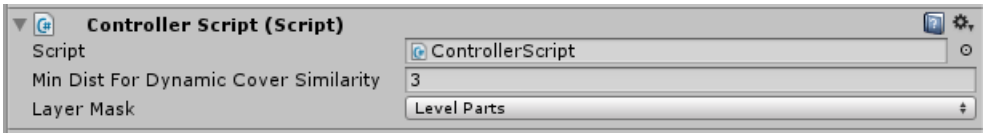
Time Between Reacting To Sounds: How long the agent must wait after reacting to one sound before reacting to another.

My AI Base Script: This target's **BaseScript**.

My Health Script Holder: Some Paragon AI methods deal damage through the **TargetScript**. This value marks the game object which holds the health script attached to this target. It will call the Damage(float damage) method on all scripts on that object. There is no limit or minimum to the number of damage scripts. This value is automatically set to the same object the target script is on if none is assigned in the editor.

CONTROLLER SCRIPT

This script is manages many aspects of the AI



INCOMING METHODS

CreateSound(Vector3 pos, float radius, int[] teams): Creates a sound at the designated position. Any agents within the given radius belonging to one of the given teams will be alerted to this sound.

CreateSound(Vector3 pos, float radius): Creates a sound at the designated position. Any agents within the given radius will be alerted to this sound.

GetCurrentAIsWithID's(int[] teams): Returns an array of transforms containing all the transforms of the targets with any one of the given ID's.

GetCurrentTargetsWithID's(int[] teams): Returns an array of targets containing all the targets of the targets with any one of the given ID's.

GetCurrentTargetsWithinRadius(int[] teams , float radius , Vector3 pos,): Returns an array of targets containing all the transforms of the targets with any one of the given ID's within the given radius.

GeCurrentTargets(): Returns every target currently in the scene.

NOTE: A Target as referenced in these methods is NOT the same as a TargetScript. It is the following class:

CLASSES

Target:

uniqueIdentifier: An integer number unique to this target

teamID: This target's team ID.

transform: The transform of this target.

targetScript: This target's TargetScript.

VARIABLES

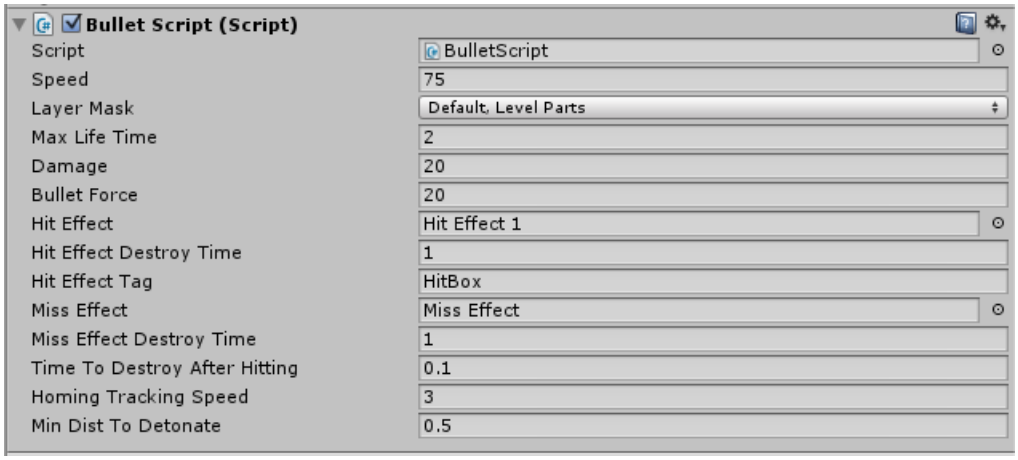
Min Dist For Dynamic Cover Similarity: Minimum distance dynamic cover agents can be from one another.

Layer Mask: A global layer mask that all agents use for line of sight.

DAMAGE SCRIPTS

BULLET SCRIPT

This script is used for AI bullets and missiles.



OUTGOING METHODS

Damage(float damage): Called on the object the bullet hits.

VARIABLES

Speed: How fast the bullet will move.

Layer Mask: The layers of the object this bullet can hit

Max Life Time: The time until the bullet will automatically destroy itself.

Damage: How much damage this bullet will do.

Bullet Force: How much force this bullet will apply to rigidbodies upon hitting them.

Hit Effect: The prefab that will be created when it hits the object marked by the hit effect tag.

Hit Effect Destroy Time: How long until the hit effect is automatically destroyed.

Hit Effect Tag: The tag that marks objects which the bullet will create a hit effect upon impact.

Miss Effect: The prefab that will be created when the bullet hits anything without the hit effect tag.

Miss Effect Destroy Time: How long until the miss effect will automatically be destroyed.

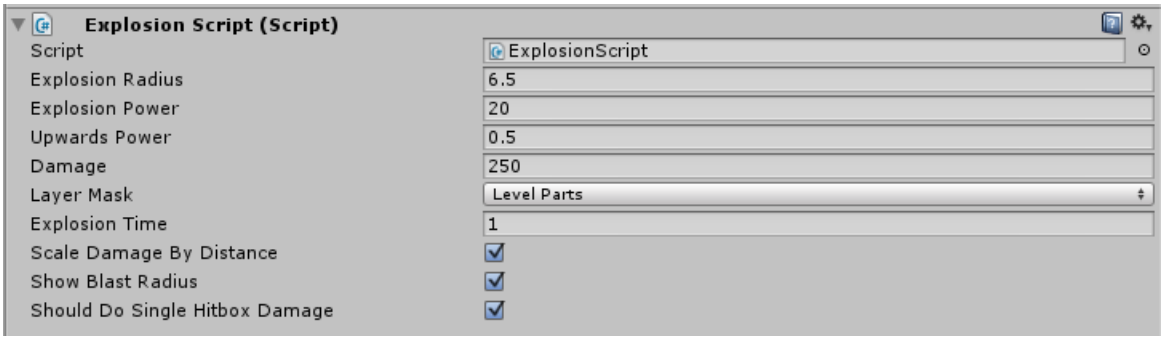
Time to Destroy after Hitting: How long the bullet will exist after hitting an object. Good for making effects such as bullet trails linger after impact.

Homing Tracking Speed: If this bullet's gunscript is designated as firing a missile, this determines how quickly the bullet will rotate to move towards its target.

Min dist to Detonate: If this bullet's gunscript is designated as firing a missile, this variable determines how close to the target the bullet must be to instantiate its hitEffect and destroy itself.

EXPLOSION SCRIPT

This applies damage and force for explosions.



OUTGOING METHODS

SingleHitBoxDamage(float damage): Called on all colliders caught in the explosion.

Damage(float damage): Called on all colliders caught in the explosion.

VARIABLES

Explosion Radius: The maximum distance objects can be from the explosion and still take damage.

Explosion Power: How much force should be applied to objects caught within the blast.

Upwards Power: How much upwards force should be applied to objects caught within the blast.

Damage: How much damage the explosion does.

Layer Mask: Layers of objects that can block the explosion.

Explosion Time: How long the explosion should exist before being automatically destroyed.

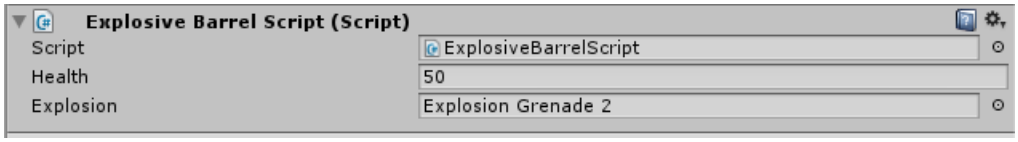
Scale Damage by Distance: Whether or not the amount of damage done to a target should scale by distance.

Show Blast Radius: Whether or not the blast radius should be illustrated in the scene view.

Should Do Single Hitbox Damage: If true, the method SingleHitBoxDamage will be called on targets. If false, Damage will be called. If your health script does not have the method SingleHitBoxDamage, this should be false.

EXPLOSIVE BARREL SCRIPT

This script is used to make objects that explode after taking enough damage.



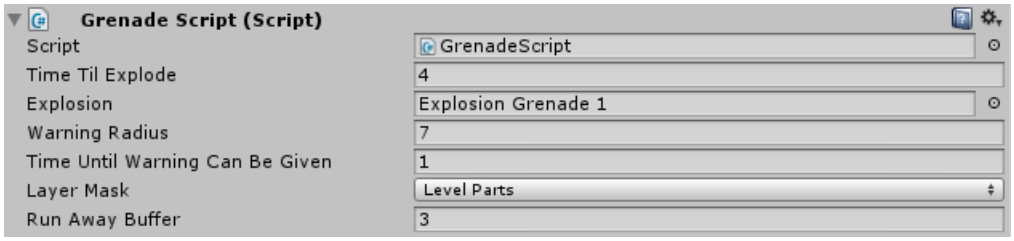
VARIABLES

Health: How much damage the object can take before exploding

Explosion: Prefab that contains this object's explosion.

GRENADE SCRIPT

This script is used for grenades



VARIABLES

Time Til Explode: How long after the grenade is created it takes to explode.

Explosion: The explosion prefab created by this grenade.

Warning Radius: How far away agents can be and be warned of the grenade. Warned agents will attempt to get out of range pf the grenade.

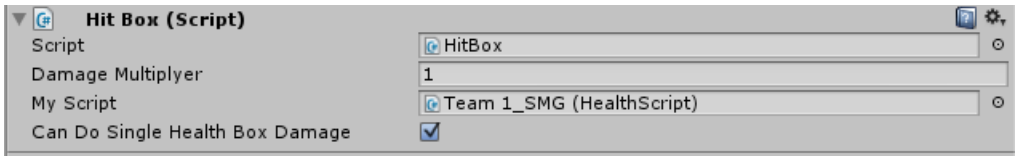
Time Until Warning Can Be Given: How long after the grenade is created until a warning can be given. After the time is up, a warning will be given when it first hits an object.

Layer mask: Layers of objects that can block warning.

Run Away Buffer: Warned agents will be told to run away a distance of the warning radius + the run away buffer.

HITBOX

This script is used to transfer damage and apply it to a health script



INCOMING METHODS

Damage(float damage): Applies damage to the object's **HealthScript**, after multiplying it by the damageMultiplier.

SingleHitBoxDamage(float damage): Applies damage to the object's **HealthScript**, after multiplying it by the damageMultiplier. Only one instance of SingleHitBoxDamage will have an effect per frame. This is used to stop explosions from applying damage multiple times to the same agent across multiple hitboxes.

VARIABLES

Damage Multiplier: How much incoming damage should be multiplied by before being sent to the health script.

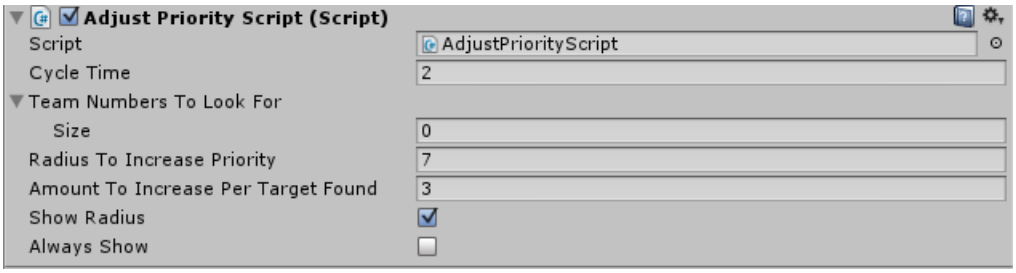
My Script: This agent's health script. This variable can be any MonoBehaviour that has a Damage(float damage) method.

Can Do Single Health Box Damage: Prevents a target with more than one hitbox from taking damage more than once from an explosion. Requires your health script to have Coroutine `SingleHitBoxDamage(float damage)`

HELPER SCRIPTS

ADJUST PRIORITY SCRIPT

This script adjusts the priority of a **TargetScript** when other targets of a given team are within a certain radius.



VARIABLES

Cycle Time: How often this script will update its **TargetScript**'s priority.

Team Numbers to Look For: Which teams should increase the priority. If this is left blank, it will automatically match the team of its **TargetScript**.

Radius to Increase Priority: How close other targets must be in order to increase the priotity.

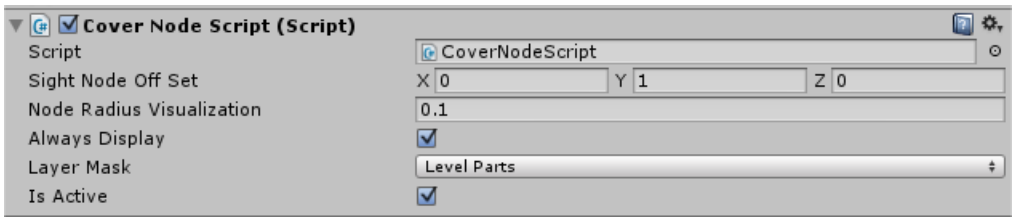
Amount to Increase Per Target Found: How much the **TargetScript**'s priority increases for every target found within the given radius.

Show Radius: Whether the radius should be illustrated in the scene view.

Always Show: Whether or not the radius should always be shown in the scene view, or only when there are targets in range.

COVER NODE SCRIPT

This script marks game objects as cover positions.



METHODS

ValidCoverCheck(Vector3 targetPosition): Checks whether this node is active and is a valid cover location.

VARIABLES

Sight Node Offset: How much the sight node position is offset by the Cover Node. The position is used for line of sight checks to make sure the agent can return fire from this node.

Node radius visualization: The size of the spheres in the scene view that are used to represent the nodes position.

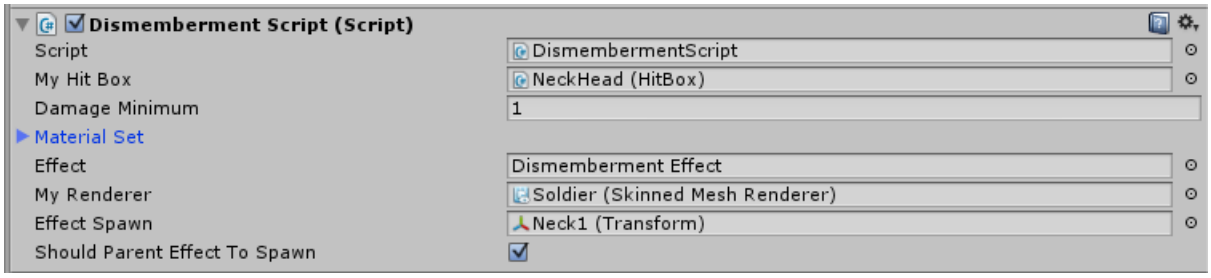
Always Display: If false, the node visualization will only appear when the object is selected.

Layer Mask: Layers of objects that block sight lines.

Is Active: Whether or not this node can be selected.

DISMEMBERMENT SCRIPT

A sample script used to demonstrate the **BaseScript**'s OnAIDeath call. This script can be used to simulate the effect of a part of the AI's body being removed if the AI is killed by enough damage to a specific hitbox.



VARIABLES

My Hit Box: The Hitbox that will be checked. The script will only trigger if the "killing blow" was applied to this hitbox.

Damage Minimum: How much damage must have been done to the hitbox to trigger the dismemberment.

Material Set: What your mesh renderer's material array will look like after this script triggers. This array should be made identical to the renderer's default array, except that the removed areas should be replaced with a 100% transparent material.

Effect: The prefab that will be spawned upon dismemberment. This is usually a particle effect- for example, a shower of sparks if a robot's head is destroyed.

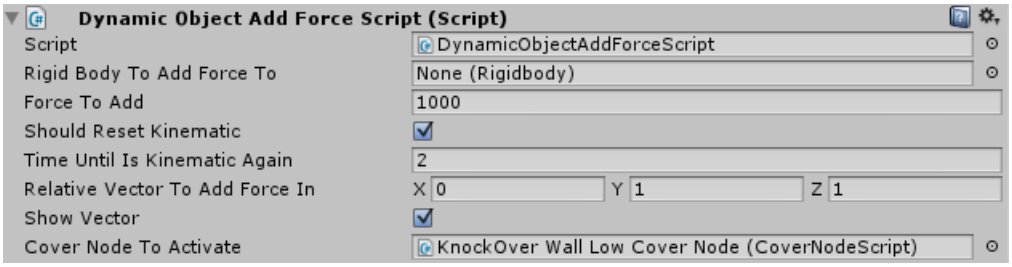
My Renderer: The AI's mesh renderer.

Effect Spawn: The transform at which the effect will be spawned.

Should Parent Effect to Spawn: Whether or not the effect should be made a child of the Effect Spawn.

DYNAMIC OBJECT ADD FORCE SCRIPT

A sample script used in conjunction with the dynamic object system. It applies a force to an object in a given vector.



METHODS

IEnumerator UseDynamicObject(): Applies the force to the object, enables the cover node, and resets the rigidbody back to kinematic after the specified amount of time has passed.

VARIABLES

Rigidbody to Add Force to: The rigidbody which this script affects.

Force To Add: the magnitude of the force that should be applied to the rigidbody.

Should Reset Kinematic: Whether or not the pushed object should become kinematic again after the force is applied.

Time until is Kinematic again: The amount of time after the force is applied that the Rigidbody should wait before becoming kinematic again.

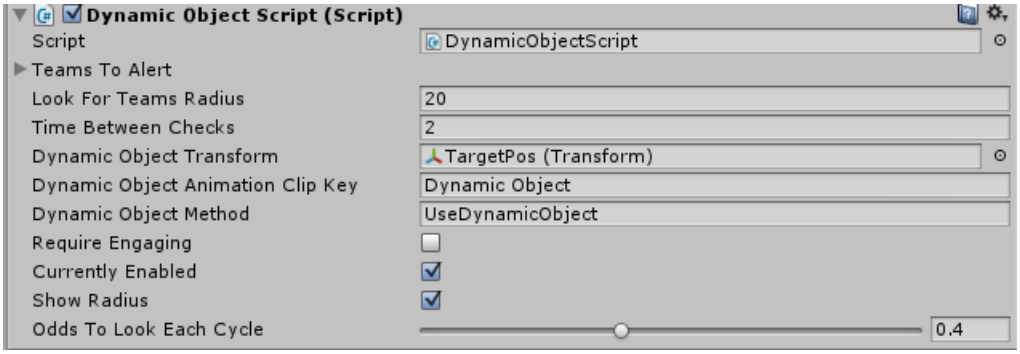
Relative Vector to Add Force in: A vector indicating the direction the force will be applied.

Show Vector: Whether or not the direction of the push vector should be shown in the scene view.

Cover Node to Activate: If you want a cover node to become active after the force is applied, it should be placed here.

DYNAMIC OBJECT SCRIPT

A sample script used in conjunction with the dynamic object system. The will attempt to get agents within a given radius to use a dynamic object



VARIABLES

Teams to Alert: A list containing the team numbers that this script should affect.

Look for teams radius: The maximum distance the agent can be from this object and still be able to use the dynamic object.

Time between checks: How often this script should check for valid agents

Dynamic Object Transform: The transform of the dynamic object.

Dynamic Object Animation Clip Key: The animation trigger that will be called on the agent when it uses the dynamic object. If you have multiple controllers for different agents that will be using this object, they must share the same trigger name. You do not need a trigger if you don't want an animation to be played.

Dynamic Object Method: The method that will be called on the dynamic object.

Require Engaging: Whether or not agents must be engaging to use this dynamic object.

Currently Enabled: Whether or not this object is currently looking for agents to use it.

Show Radius: Whether or not the radius to look for teams should be visualized in the scene view.

Odds to Look Each Cycle: The odds the script will try and select an agent to use the dynamic object each cycle.

TROUBLESHOOTING

AGENTS "FIGHT" OVER THE SAME COVER NODE

Sometime, you may accidently duplicate a cover node and forget to move it. Make sure there is only one node in the area in question.

AGENTS DON'T PLAY ANIMATION 'X'

If you haven't set up your animation controllers to support this animation, it won't be played. For dynamic animations we have a tool to help you. For others, you can either look at a pre-made example to get an idea of what needs to be done, or you can simply create a brand-new controller using the appropriate wizard.

AGENTS PLAYS THE WRONG DYNAMIC ANIMATION

Each controller can only have one trigger with the same name. As such, if you created two with the same name, you'll have to go into the controller and rename one of them. If this fails to resolve the issue, make sure you are passing the TargetScript the correct animation name.

AGENTS DON'T PLAY UPPER BODY ANIMATIONS AND AREN'T AIMING AT THE TARGET

Make sure you have gone into the controller and assigned it a mask which only includes its upper body. For more detail, look to "Creating Your Own Paragon AI Agent" at the beginning of this manual.

AGENTS DON'T MELEE TARGETS

Make sure you have enabled it in the agent's Base Script. Also, in order to receive melee damage, you must have assigned the HealthScriptHolder value of the receivers TargetScript to the object that contains the receiver's health script.

AGENT WAITS A LONG TIME BETWEEN THE GAME STARTING AND MOVING.

Make sure you've saved it as a prefab, delete it, and drag in a new instance of prefab. This is a known bug, but I don't know why it's happening.

AGENT DOESN'T SEE ENEMIES

Make sure your eye is facing the right way, doesn't have anything blocking its line of sight and ensure the field of view in the Target Script is high enough.

BULLETS DON'T SPAWN FACING THE CORRECT DIRECTION

Make sure your bullet spawn is facing the right way.

AGENTS WONT USE DYNAMIC COVER

Make it is enabled in the agent's **CoverFinderScript**. If the problem persists, try increasing the **maxDistBehindDynamicCover** value.

NAMESPACES

Paragon AI scripts are within the ParagonAI namespace.

AGENTS MOVE THROUGH/UNDER THE PLAYER

Add a navmesh obstacle to your player.

ANIMATIONS ON THE DEFAULT ASSETS DON'T PLAY PROPERLY

Sometimes Unity will swap out animations in the Animation Controller for other animations in the project file. I am not sure as to why this happens. The only solution is to open the controller and swap out the animations or recreate the animation controller using the provided tools.