Overview

This toolkit is intended for the creation of FPS style games. It uses a *Swarm Manager* system to gate attacks and can be used to control difficulty. This document and toolkit are still considered to be under some level of development. As the toolkit gets fixed and upgraded so will this document.

Formatting

Below we're going to establish the formatting style to make the document easier to follow.

Script Title

Overview/description of the script.

Variable Name: This will be one of the variables accessible in Unity's Inspector and what will follow is a brief description of the variable.

FunctionName(): This will be a public function that can be called in a number of ways such as through Unity events, animation events, and other generic methods provided by Unity. The list given with scripts will NOT include every public function since some are only made public for programming reasons. These lists will focus on ones intended to be designer facing.

{} - denotes the end of a script overview.

If you're reading a block of text and see a *Name* italicized, that is referencing a script; however, if you see the *Name()* italicized and followed by parenthesis (()) then I am referencing a function. **Variables** will be bolded when mentioned in text.

Important: Red text indicates an important aside, note, or explanation that the programmer deemed deserving of special attention and is often used to help avoid future issues or known edge cases.

If you're reading and a Name appears in a different color then it's being established as a game entity, not just a component but the complete set of components that make up a full entity. This can be a child gameobject with a script or two or a full functioning entity such as an enemy.

Advanced Apologies: there are a number of misspellings with script names and variables. Much of the coding was done during late nights, so spelling accuracy was not the concern of the programmer.

Combat

In order to understand how some of these scripts work and are intended to coordinate it is important to understand how combat functions. There are three distinct entities in your generic combat setup. Player, Swarm Manager, Enemy.

The Player has free control to move, look, and use their weapon to engage enemies, and interact with the environment. When the player fires their weapon, as long as they have ammo in their weapon, a raycast will be drawn outward based on the weapon's distance. The player does have a degree of possible inaccuracy, the ray cast adjusting angle within that area of inaccuracy. If the raycast intersects an Enemy (or anything with a *Health System*) it will do the weapon's predetermined amount of damage.
~~Jump not currently featured in project, on docket to add soon.~~ Jump now implemented.

Enemy agents have a bit different functionality. They operate off Finite State Machines (FSM). They have two major states: Neutral and Attack. In the Neutral state the Enemy may stand still, wander a defined area, follow a path, or explore their environment. Based on their scripting the Enemy has a number of ways they can gain a target for their Attack behavior. Most commonly the target will be the player but the versatility of the script makes it possible for an Enemy agent to become aggressive toward and attack other entities.Once a target is gained the Enemy goes into the Attack state.

The attack state has three distinct states/phases. There is Wait, Zone, and Attack. These are influenced by the Swarm Manager and we'll get into that in a moment. The exact nature of these three states is dependent on developer scripting, so the overview here is the broadest terms. The Wait state is when the Enemy has a target and is aggressive, but hasn't been given any permission by the Swarm Manager to act, so they'll engage in their set Wait Behavior. The Zone state is the bridge between Waiting and Attacking. The Zone behavior is how the AI acts leading up to their attack state. The Attack state is when the Enemy actually initiates their attack animation, and once done they will drop back to their Zone or Wait behavior depending on scripting.

The Swarm Manager is responsible for controlling when Enemy AI has permission to attack. The Swarm Manager has a pool of Attack Tokens and Zone Tokens. Every Enemy attack has a zone cost and an attack cost. When enemies are aggressive the Swarm Manager randomly hands out Zone Tokens to those in their wait state, giving them permission to start gearing up for an attack. Zoning Enemy agents then can be given attack tokens which gives them permission to actually initiate an attack. The Swarm Manager only has a set number of

tokens though, so naturally once it is out of tokens Enemy agents will have to wait till other agents finish their actions and return tokens to the Swarm Manager to be redistributed.

There are options for Enemy AI to ignore the Swarm Manager and act freely.

Scripts

Aggro Checker

Script must be paired with *Creature Control.*

The *Aggro Checker* is a key part of the script package needed to make a fully functioning AI agent. This script determines how the AI notices other entities and determines when the AI will transition from Neutral to Aggressive. There are a number of options that can be mixed and matched to create AI that notices different entities in different ways.

Aggro On Site: This is a boolean. If set to true then the AI will be able to visually notice other entities. For something to get noticed on site it has to be in front of the players **Site Object** within their field of view, and there must be nothing obstructing the line of sight.

Aggro On Touch: This is a boolean. If set to true the AI will be able to become aggressive to objects that physically bump into it.

Aggro On Damaged: This is a boolean. If set to true the AI will be able to recognize what caused them damage, and if one of their allowed targets, become aggressive toward them.

Aggro In Range: This is a boolean. If set to true it will allow the AI to simply add aggro to relevant entities close to it. This method ignores LOS and simply checks if the object is close enough.

Site Aggro Tags: This is a list of strings. You'll want to populate this list with the tags of entities that the AI can add aggro to on site.

Touch Aggro Tags: This is a list of strings. You'll want to populate this list with the tags of entities that the AI can add aggro to on if they bump into each other.

Damage Aggro Tags: This is a list of strings. You'll want to populate this list with the tags of entities that the AI can add aggro to if they do damage to this AI.

Range Aggro Tags: This is a list of strings. You'll want to populate this list with the tags of entities that the AI can add aggro to if they are close enough.

Tags are case and character sensitive. Mismatching capitalization or adding in a space, even at the end, can cause issues with recognizing targets.

Site Distance: This is a float. This is how far in Unity distance units the AI entity can see.

Range Distance: This is a float. This is how close an entity has to be in order to be a valid target for **Aggro in Range.**

Site Object: Important if using Aggro On Site! This is a GameObject and is intended to get around how variable the positioning of the AI's transform can be. You'll want to add a GameObject (can be an empty one) to your AI and place it where you want sight to be measured from. The GameObject should also be outside of the AI's own collision area to avoid issues with the AI blocking its own LOS.

{}

Attack Behavior

Script must be paired with *Creature Control* to work properly.

Attack Behavior is the script that controls the entire Attack state of AI behavior. It has a number of variables that influence behavior and is also what controls between Waiting, Zoning, and Attacking.

Attack Package: This is a list of GameObjects and intended to be used to access Attack Spawners. You can create an Attack Spawner by creating a GameObject on your hierarchy and giving it the *Attack Spawner* script. This is the list that will be randomly chosen from when an AI goes to attack an enemy.

Coordinate with Swarm Manager: This is a boolean. If true the AI will need to check with the Swarm Manager for Zone Tokens and Attack Tokens to gate their aggressive behavior. If false then the AI entity will attack regardless of permission. Set to False sparingly, should only be used for special entities/enemies as ordained by design.

Move Time: This is a float. This determines how frequently the AI entity will find a new location to move to, regardless of state. Keeping it low will make the entity seem more responsive to their target's movements.

Wait Behavior: This is a list of options. This determines how the AI acts towards their target while in the Wait state. Options:

- None
    - Entity will stand still and only act if the target gets close enough.

- Keep Away
    - Entity will keep a certain distance away from their target.
- Get Close
    - Entity will try to get close to their target.
- Find Clear Shot
    - Entity will maintain a distance while trying to make sure they have a clear LOS to their target.

Wait Distance: This is a float. This determines the range for the various **Wait Behavior.**

Randomize Area: This is a float and is purely for creating more active/realistic AI. Putting a number here will allow for some randomization on where the AI moves, giving their movement slight unpredictability.

Walking Anim String: This is a string and is to be paired with the entity's Animator. Your transition to the animation you want to use while the enemy is moving in the attack state should be gated by a boolean value, and the string for that boolean should be copied into here.

Rotate Speed: This is a float. While in the Wait state if your AI isn't moving it will rotate toward their target, and this sets how quickly the entity will rotate.

Sound Set: This is a *RandomSound* reference. You can attach a script to your AI and set it up to make sounds while your AI is in an aggressive state. It should be referenced here so your AI knows when to turn the script on and off.

Audio Source: This is the Audio Source object that your AI will make sounds from. You can drag and drop one from your child entities in the hierarchy if you want more control over its position or just use one attached to the same parent object.

Aggro Sounds: This is a list of audio files. When your entity first transitions from Neutral to Aggressive a sound will be randomly pulled from this list and played.

PlayAttackSound(): This can be called from your attack animation to play the sound you have set to coordinate with the attack (as set in the *Attack Spawner* and *Attack Data.*)

EndAttack(): Important! This HAS to be called toward the end of your attack animation in order for your AI to not break. This lets your AI know that it can start winding the attack down and is important for setting and controlling a number of variables that control AI behavior.

SpawnAttack(): Important! This HAS to be called from your attack animation in order for the actual attack to spawn and any damage be done. This lets your AI know to fire the specific Attack Spawner and launch the coordinating attack.

DestroyAttack(): Important! Use on any attack where the attack object doesn't have its own behavior/built in despawn. This lets your AI know that the damaging portion of the attack is done and it can despawn the damaging part of the attack.

LockRotation(): This can be called in order to stop your AI from rotation toward the player. Mainly to be used in your animation to stop the AI from infinitely tracking the target position while attacking, though is not required.

unLockRotation(): This can be called to unlock the AIs ability to rotate toward their target. This should be called later in any animation that *LockRotation()* is called.

The common order of function calls from an animation are:

1. *LockRotation()*
2. *SpawnAttack()*
3. *PlayAttackSound()*
4. *DestroyAttack()*
5. *unLockRotation()*
6. *EndAttack()*

{}

Creature Control

This is your main AI control script! Dropping it onto your AI will add a bunch of the normal expected scripts.

The *Creature Control* is the master control script for your AI. Removing or disabling it will break and stop all other behavior.

Despawn Timer: This is how long after an AI entity dies before it is removed completely.

Death Anim Trigger String: This is a string and a reference back to your animator set up. If you have a death animation in your animator you should gate it by using a Trigger and match the name of the Trigger here so the animation can be played upon death.

TriggerDie(): This is a function that triggers the AI to go into its dead state and start the despawn process. This will mostly be called from the *HealthSystem* of the gameobject but can be triggered by other external events.

{}

Nuetral Behavior

Script must be paired with *Creature Control* to work properly.

Nuetral Behavior is the script that controls how your AI functions while in a Neutral state. This can be used to stage areas or have AI function passively to create a more believable environment.

Ai Behavior: This is a list of the ways the AI can move/operate while in the Neutral state.
Options:

- None
  - AI will stand still and idle and do nothing.
- Wander
  - The AI will pace around an established area.
- Path
  - The AI will follow a predetermined path.
- Hunt
  - The AI will randomly move around the accessible map.

Wander Distance: This is a float. This is how far away from their original position the AI will potentially move in the Wander mode. It's a radius.

Wander Frequency: This is a float. This determines how often the AI will look for a new position to move to.

Walking Animator Bool: This is a string and is a reference back to the animator. If you have a walking animation this is string should match the name of the bool animator variable controlling access to and from the animation state.

Path Way Points: This is an array/list of Vector 3 positions. The points as listed out are the ones an AI in the pathing state will follow along.

Path Repeat Type: This is a list of how AI reacts when they reach the final pathway endpoint.
Options:

- None
  - AI will stop at the final point and stand still, going idle and doing nothing.
- Loop
  - AI will repeat the path, going right back to the first point on their path list and making it their next destination to begin walking the path again.
- Reverse
  - AI will walk the path forward, backwards, and repeat. Once at the final point the AI will walk the path in reverse waypoint order, and at the beginning again will repeat.

- Wander
    - At the end of the path the AI will go into a wander state that uses the wander variables that can be set on this script.

Break Time: This is a float. This measures how long the AI spends at each waypoint before moving to the next. Time is in seconds.

Hunt Distance: This is a float. It measures how far out each new destination may be. This is a radius measured from the AI's current position, and anywhere inside that radius could be its next movement point.

Hunt Frequency: This is a float. It measures in seconds how long the AI takes in finding a new point. This timer starts as soon as they have found their point, so it is possible for a new point to be sought before the previous one is reached if distance too far out and timer too short.

Sound Set: This is an empty behavior field that you can use to link a *Random Sound* script to. The script will handle turning the script on and off. You can use the script (detailed later) to set up random sounds to be made while the AI is in the neutral state.

Path Marker To Spawn: BE INCREDIBLY CAREFUL! In order for the AI to operate more smoothly in complex 3D environments this object was created to ain path finding. This should always (unless well understood and changed for very good reason (or programming says so)) be the Path Marker gameobject provided with the kit.

Wander Debug To Spawn: This is a GameObject reference. This is where you can plug in the provided Wander Area Debug Tool to help visually debug the wander and hunt behaviors. Nothing breaks if this is left blank, but plugging in random objects may cause some very odd issues.

Debug Visible: This is a bool. If set to true the Path Marker and Wander Area Debug Tool become visible. If set to false they will be rendered invisible to not visually intrude on the game. This will only take effect at the beginning of play, toggling it while the game is rendering has no effect.

SetCentralLocation(Vector3): This can be called to forcefully set a creature with Wander or Hunt behavior. The central location controls the possible area for the AI to move around in.

ResetCentralLocation(): This can be called to reset the central location, the area pinned to control the area the AI wanders and hunts inside. Calling this will set the central location to the AIs current location.

{}

Add Health

Add Health is a script that can be used on a gameobject with a collider to restore health to other gameobjects with a *HealthSystem* script with a degree of control.

Health To Add: This is an int. It is the amount that the target objects health will be adjusted by every tick it is in contact with the relevant collider.

Strings To Trigger: This is a list that can be populated with strings. These are the strings that can be affected by the collider and have their *HealthSystem* adjusted.

Despawn On Pickup: This is a bool. If triggered true then after the first time an object collides with this gameobject and meets the string requirement from **Strings To Trigger** then this gameobject will despawn.

Draw Collider In Game: This is a bool and is for debugging. You can set this to have the collider visible in game to help with visualization or set to false to hide the collider.

{}

Attack Data

Attack Data is the script used to create gameobject's that represent an AIs attack. The gameobject using this script will get referenced by *AttackSpawner* which will be a gameobject attached to the AI.

Damage Behavior: This is a list of options on how damage is done.
Options:

- Raycast
  - Upon spawn the gameobject will perform a raycast, draw a line forward, and try to do damage to the first object it intersects.
- Collider
  - The gameobject will need an active collider or this method will not work. There are several ways for the collider to work. It can do damage on first hit and it can do damage overtime for as long as colliders overlap.
- Parent
  - This will only do damage to the object the gameobject is parented to, if there is no *HealthSystem* on the parent object no damage will be done.

Parent To Attacker: This is a bool. Upon spawn the attack gameobject will become attached to the spawner with a reference to the attacking AI as the grandparent. This is good for melee attacks.

Ignore Attacker: This is a bool. This is an additional way to control what all gets damaged by the attack. If set to true then the parent (grandparent) of the gameobject won't get damaged by the attack, even if the tag matches the *AttackData's* requirement.

Damage On Collision: This is an int. This is for the collider **Damage Behavior.** This is how much damage is done with colliders first intersect.

Damage On Overlap: This is an int. This is for the collider **Damage Behavior.** This is how much damage is done as long as the attack and the target are overlapping.

Overlap Delay: This is a float. This is how often the attack does damage to objects it is overlapping. If set to 0 it will do damage every frame. Otherwise time is in second.

Destroy On Collision: This is a bool. If set to true then the gameobject for the attack will despawn after the first object it collides with that meets the attack requirements.

Destroy Time: This is a float. This is how long it takes for the object to completely despawn in seconds.

Raycast Distance: This is a float. This is for the raycast **Damage Behavior.** This is how far out the raycast will be drawn to check for damage.

Spawn On Collision: This is a slotable gameobject. This is an object that will be spawned after a successful collision with a target that meets damage requirements. This can be used for a visual effect or some other affect. It can even be used to spawn another attack.

When an *AttackData* script spawns another attack the original AI attacker will be passed to the new attack.

Spawn On Parented: This is a slotable gameobject. This is an object that will be spawned when the gameobject becomes parented to another object (attacker or otherwise.) This can be used for a visual effect or some other affect. It can even be used to spawn another attack.

Parent to Collided Object: This is a bool. If true then the attack will become parented to the first object it successfully collides with that meet the damage requirements.

ForceCollisionSpawn(): This can be called to force the **Spawn On Collision** object to be spawned, even when a collision has not occured.

{}

Attack Spawner

The *AttackSpawner* script is designed to be attached to empty game objects that are children of the AI that will be using the attack. The attack spawner is also responsible for how the AI acts once they've been given permission to zone on their target. So it's important to keep in mind what type of attack is being performed and making sure an appropriate zone behavior is chosen.

Zone Behavior: This is a list of behavior options. This determines how the AI acts towards their target while in the Zone state. Options:

- None
  - Entity will stand still and only act if the target gets close enough.
- Keep Away
  - Entity will keep a certain distance away from their target.
- Get Close
  - Entity will try to get close to their target.
- Find Clear Shot
  - Entity will maintain a distance while trying to make sure they have a clear LOS to their target.

Is Melee: This is a bool. It doesn't have any direct measurable impact on behavior and is more for AI on the backend to keep track of melee vs. ranged attacks. This should still be set to true if your attack is considered physical and close-quarters.

Attack Object: If this is blank nothing will happen and the AI may break! This is a slotable gameobject where you attach your attack, the gameobject with the *AttackData* script.

Parent Anim String: This is a string and a reference to the AI's animator. This is the string for the trigger that relates to the animation of the attack. This is important! The attack animation, as discussed in *AttackBehavior* controls the attack and when the actual damaging part of the attack spawns. Make sure your animation has the relevant flags, the information in the *AttackSpawner* is accurate and *AttackData* is complete. This string is case and character sensitive and HAS to be a trigger.

Tags to Damage: This is the list of tags strings the *AttackData* inherits to determine if to do damage to an object or not. If left empty anything with a *HealthSystem* will be damaged, otherwise the gameobject's tag will have to be on the list here in order to be damaged. If it is not on the list and the list has entries the object will be ignored and not damaged.

Tags are case and character sensitive. Mismatching capitalization or adding in a space, even at the end, can cause issues with recognizing targets.

Attack Cooldown: This is a float. This is a measure of time in seconds and is how long after the AI completes this attack they will wait to perform another attack.

Attack Rotate Speed: This is how quickly the AI rotates towards it target while in the attack animation, up until rotation is locked with the *AttackBehavior lockRotation()* function.

Attack Distance: This is a float. For melee attacks this is how close the AI will try to get before initiating the attack.

Range Distance: This is a float. For ranged attacks this is how far the AI can be to initiate the attack, outside that range the AI will not initiate their attack and will wait until their target is close enough.

Attack Sound: this is a slottable audio file. This is the sound the AI will make if the *PlayAttackSound()* function is called by the animation.

{}

Bullet React

This is a generic script that can work on anything with a collider. It operates fairly simply, if a raycast from the player hits the connected object a react object is spawned. You can create a react object any way you feel appropriate. It can be a quickly despawning particle effect to represent a hit or it can be a sprite that looks like a bullet hole.

Hit React Object: This is a list of slotable gameobjects. This is the list that will be pulled from randomly when a player bullet hits the collider. Object to spawn is chosen randomly at moment of impact/spawn.

Debut Hit React: This is a bool. If set to true it will issue a log in the debug command about the gameobject hit and the location of the hit/object spawn.

{}

Footstep Sounds

This is a generic script that is primarily intended to be paired with moving AI, sound functions called from the animation.

Left Foot Sounds: This is a list of slotable audio files. When *PlayLeftSound()* is called one of the audio files from this list are chosen at random to play.

Left Audio Source: This is a slotable AudioSource. You can drag and drop or use the help list to select the audiosource you want to use to play the left foot sounds. Generally advise

using a unique audio source instead of a shared audiosource as it can cause race issues or sound bugs.

Right Foot Sounds: This is a list of slotable audio files. When *PlayRightSound()* is called one of the audio files from this list are chosen at random to play.

Right Audio Source: This is a slotable AudioSource. You can drag and drop or use the help list to select the audiosource you want to use to play the left foot sounds. Generally advise using a unique audio source instead of a shared audiosource as it can cause race issues or sound bugs.

Debug Sounds: This is a bool. If set to true then the debug log will display a notification for anytime one of the functions for this script is called and will detail what gameobject called the function.

PlayLeftSound(): This function is designed to be called from the animator of the attached gameobject anytime you want a footstep from the left foot to play.

PlayRightSound(): This function is designed to be called from the animator of the attached gameobject anytime you want a footstep from the right foot to play.

{}

Gen Change Scale

This is a generic script that when active will change the attached gameobjects localScale.

Scale Change Rante: This is a Vector3. This tells the gameobject how to affect the localScale per axis for every scale update.

Change Speed: This is a float. This is how quickly in seconds the change occurs.

{}

Gen Despawn After Lifetime

This is a generic script that can be used to set a timer, and once the timer is over the object will despawn.

Lifetime: This is an int. Time is measured in frames. That is how many times Update() is run during gametime. Once the set number off frames have passed the object this script is attached to will despawn.

{}

Generic Animation Trigger

This is a generic script that can be used to affect a particular object's animator. Most of the work is done by the public functions that can be called to affect the animator.

SetFloat(string, float): This is a function that can be used to set a float variables value. This function takes in a string which should correspond with the variable name and a float which is the value to change the float to.

SetInt(string, int): This is a function that can be used to set an int variables value. This function takes in a string which should correspond with the variable name and an int which is the value to change the float to.

SetBool(string, bool): This is a function that can be used to set the bool value of an animator variable. The function takes in a string which should correspond with the variable name and a bool which is the value to change the bool to.

SetTrigger: This is a function that calls the SetTrigger function of an animator. It takes in a string which should correspond to the variable name for the trigger being fired.

ResetTrigger: This is a function that calls the ResetTrigger function of an animator. It takes in a string which should correspond to the variable name for the trigger being reset.

{}

Generic Interact

This is a script that is used on objects that the player can interact with. It's fairly simply and uses a UnityEvent upon the interact which can be used to control and influence a number of objects and behaviors in game.

Interact Event: This is a unity event and can be used to trigger a number of actions and behaviors upon the player interacting with the object.

{}

Generic Turn On Off Component

This is a script that has public functions that can be used to turn on and off components of the attached gameobject.

Objects to Affect: This is a list which can be populated with behavior and components attached to the same game object.

TriggeredOn(): This function can be called to make sure all objects in the list of **Objects to Affect** are turned on.

TriggeredOff(): This function can be called to make sure all objects in the list of **Objects to Affect** are turned off.

{}

Gen Event After Lifetime

This is a script that can be used to have various events and behaviors happen after a set amount of time.

Lifetime: This is an int. Time is measured in frames. That is how many times Update() is run during gametime. Once the set number of frames has passed the corresponding Unity Event will occur.

Time End Event: This is a Unity Event. It can have a variety of cascading effects and influences. This will only trigger once at the end of the timer.

{}

Health System

This is an important script! It can be affected by all types of attack, both player and AI. It should be present on all killable AI.

The health system is used by all damagable objects. That includes AI and the player. The *Health System* can also be used on child objects to create more vulnerable/defensible areas on an object. This requires that every object from the source gameobject (the parent of all) to the object the damage collider is attached to has a *HealthSystem* component as this is how the information is passed along. Any one object missing a *HealthSystem* will disrupt the chain.

In example let's say you have the primary gameobject, the root that has all the relevant behavior scripts. This one also has a *HealthSystem* that triggers death, yet this object does not have any colliders since you're doing it part by part so you can modify damage being done. For every part (head, arm, body) there are two degree of objects on the hierarchy. Each object from the root to the object with the collider that can be damaged must have a *Health System*

attached. The **Damage Mod** will be applied at every step, so best practice is to have the main object receiving the damage have the mod you want (2, 0.5, etc) and everything else including the root *HealthSystem* to have a modifier of 1.

Health: This is an int and is how much damage can be taken before the **Is Dead** event is triggered. Note that this doesn't mean the gameobject will die or despawn. You can also leave this blank if this is not the root health system and is just being used to pass damage from one component object back to the root.

Damage Mod: This is a float and how the incoming damage is modified. 1 means the damage is taken as received, 0.5 is half the damage, and 2 is twice the damage. This can be used to create vulnerable or defense areas on an object.

Damage After Death: This is a bool. If set to true then the **Took Damage** event will fire anytime damage is taken, even if the health is at or below 0. If false then the event will no longer fire.

Took Damage: This is a Unity Event that fires anytime damage is taken, whether its from an external force or passed along from a child object.

Is Dead: This is the Unity Event that fires only once after health has reached or fallen below 0. This is the event you want to use on your root *HealthSystem* on an AI to trigger the death of an AI.

Debug Events: This is a bool. If set to true a Debug message will be sent out detailing when an event occurs and from what gameobject.

TakeDamage(float, gameobject): This is a public function that can be called to cause damage. The float is the amount of damage taken and the gameobject is to track what is doing damage to the AI, this is especially important for AI as they can add aggro to objects doing damage to them.

AddHealth(int): This is a public function that can be used to restore some health to the AI. The int is the amount of health to be restored to the AI.

{}

Health System Sounds

This system only works with the *HealthSystem.*

This script is primarily used to have sounds play when the AI is damaged or dies.

Damage Audio Source: This is a slotable audio source. This allows you to set what audiosource to use when sounds are played from this script.

Damage Sounds: This is a list of slotable audio clips. These are the sounds that might play when damage is done to the AI. The sound is chosen randomly from the list.

Damage Sound Possability: This is a float range between 0 - 100 and represents a percentage. This percentage is how likely a **Damage Sound** is to play when damage is done to the *Health System.*

Death Audio Source: This is a slotable audio source. This allows you to set what audiosource to use when sounds are played from this script.

Death Sounds: This is a list of slotable audio clips. These are the sounds that might play when the AI dies. The sound is chosen randomly from the list.

Death Sound Possability: This is a float range between 0 - 100 and represents a percentage. This percentage is how likely a **Death Sound** is to play when the *Health System* hits 0.

Debug Sounds: This is a boolean. A message will be thrown into the debug console whenever one of the sounds plays and will also note what gameobject the sound plays from.

{}

Random Sound

This system can be used to have AI emit ambient and controlled noise. This is primarily intended for an AI to generate random noises but can be used for more controlled sound generation.

An AI can have multiple *RandomSound* scripts, most AI can easily use two, one linked to *NuetralBehavior* and one to *AttackBehavior*.

Audio Source: This is a slotable AudioSource that sets which AudioSource to use for this particular script.

Random Sounds: This is a list of slottable audio clips. These are the sounds that are randomly chosen from to play.

Play Sound Randomly: This is a bool. If set to true then the random timer will take effect. If set to false then a sound might only play when manually called.

Random Sound Frequency: This is a float. This is how often a random sound might play. Time is measured in seconds.

Random Sound Possability: This is a float that is representative of a percentage. This is how likely a sound is to play when called, either through outside function call or at the end of each timer.

Debug Sounds: This is a bool. If set to true information will be displayed in the debug console.

PlaySound(): This is a public function that can be called to run the sound event. This still checks the random possability.

{}

Rigid Body Forward Force

This is a generic script that can be used on gameobjects with a ribidbody. The script will add forward force to the rigidbody once the script is triggered active, but it will only do so once.

Forward Force: This is a float and is the amount of force to apply to the rigidbody.

{}

Generic Spawn

The *GenericSpawn* script can be used to create other gameobjects. This is praimarily intended to be used with empty gameobjects to create enemies but can be used as the designer sees fit.

For spawners, trigger volumes, and other objects that have colliders but aren't intended to be physical objects that block LOS or player fire should be kept on Layer 8 (Volume Spawners). This will prevent non-physical or objects not intended to be physically present from interfeering with gameplay.

Spawn Table: This is a list of gameobjects. These are the objects that are capable to be created by the spawner. The object so spawn is chosen at random. Work is being done to create a weighted table to make some spawns more likely, but currently object is chosen at full random with equal possibilities.

Repeat Spawn: This is a bool. If set to false then this spawner will only spawn one set of objects. The spawner will have to be manually triggered for each set. If set to true then the spawner will follow its delay rules and spawn sets of objects until the max is met.

Objects to Spawn: This is an int. This is how many separate objects will be spawned in one instance. You can either spawn objects one at a time or in groups. So if set to 3 then 3 objects will each be chosen randomly from the **Spawn Table** and created in the world all at the same time.

Max Objects to Spawn: This is an int. This is how many objects the spawner is intended to create, especially if **Repeat Spawn** is set to true. Once this many objects have been spawned the spawner will deactivate and stop spawning objects. If **Fire Only Once** is false then calling *ActivateSpawner()* after the max has been spawned the spawner will reset and continue rules to spawn more objects. This can be used for wave spawners or spawners called under special conditions repeatedly.

Seconds Between Spawns: This is a float, it is a measure of time in seconds. This is for **Repeat Spawn,** if true this is how many seconds between spawn sets. After this length of time the **Objects to Spawn** amount will be created.

Fire Only Once: If set to true then this spawner can only be used till its max rulesets have been met. Once the spawner, either through timer or manual calls, has spawned **Max Objects to Spawn** the spawner will stop working even if called to fire.

Draw Collider In Game: This is a bool for debugging. If set to true then the mesh renderer of the object will be visible in game, otherwise when false the object will be hidden.

ActivateSpawner(): This is the function used to start the spawner working. Spawners do not start the game working, this function has to be called through a Unity Event or other method in order to have the spawner begin producing objects.

{}

Trigger Volume

The *TriggerVolume* script is primarily intended to be used to cue spawners and special events, but it is a fairly flexible script that uses Unity Events to make the possibilities pretty open to designer. This does require the script be attached to an object with proper collision detection (usually toggled to serve as a trigger).

For spawners, trigger volumes, and other objects that have colliders but aren't intended to be physical objects that block LOS or player fire should be kept on Layer 8 (Volume Spawners). This will prevent non-physical or objects not intended to be physically present from interfeering with gameplay.

Player Enter Event: This is a Unity Event that occurs anytime an entity that matches the string requirements in **Strings To Trigger** first comes in contact with the trigger volume.

Player Exit Event: This is a Unity Event that occurs anytime an entity that matches the string requirements in **Strings To Trigger** exits after overlapping with the trigger volume.

Both above events can be repeated if entity walks away and re-enters/re-exits the trigger volume.

Strings To Trigger: This is a list of strings that can be used to control what gameobjects can cause the attached Unity Events to trigger. Best practice, unless for other specific design reasons, to keep to just “Player”.

Draw Collider in Game: This is a bool to aid in level testing and debugging. If set to true the visual portion of the object will be represented in game, otherwise if set to false then the visual element will be made invisible. This does not affect child components.

{}

Ammo Pickup

This is a script intended to be used on objects that will grant the player ammo for their currently equipped weapon. The script can be attached to an object that can collide with the character.

Ammo To Add: This is an int and is the amount of ammo that will be added to the players current inventory ammo on collision.

Despawn On Pickup: This is a bool. If set to true the object will despawn itself once the player has been given ammo.

Draw Collider In Game: This is a bool to aid in level testing and debugging. If set to true the visual portion of the object will be represented in game, otherwise if set to false then the visual element will be made invisible. This does not affect child components.

{}

Camera Look

This is a script that is used in player control. It is attached to the camera object of the player.

Vertical Mouse Sensitivity: This variable controls how quickly the camera moves up and down in response to mouse movement.

{}

FPS Control

This is the primary script for player movement.

Forward Key: This is selectable key code. This is the key that will be used to move the player forward. If left at None it will be automatically mapped to W.

Back Key: This is a selectable key code. This is the key that will be used to move the player backwards. If left at None it will be automatically mapped to S.

Strafe Left Key: This is a selectable key code. This is the key that will be used to strafe the player left. If left at None it will be automatically mapped to A.

Strafe Right Key: This is a selectable key code. This is the key that will be used to strafe the player right. If left at None it will be automatically mapped to D.

Jump Key: This is a selectable key code. This is the key that will be used to cause the player to jump. If left at None it will be automatically mapped to SPACE.

Speed: This is a float and can be used to set how quickly the player moves; however this currently works in tandem with the Character Controller. Values in the character controller can also be used to affect speed and may overwrite this speed if values are too divergent. This is a planned fix.

Jump Strength: This is a float and affects how high and how quickly the player will jump.

Gravity: This is a float and affects how quickly the player will fall back to the ground, it affects **Jump Strength,** higher gravity may even prevent player from jumping at all and slow them down.

Horizontal Mouse Sensitivity: This variable controls how quickly the camera moves left and right in response to mouse movement.

PHUD: This is a slottable gameobject intended to create a HUD. **PHUD** is short for Player-Heads-Up-Display.

{}

FPS Weapon Use

This script controls how the player uses their weapon and is vital to the FPS formula of this project.

Weapon: IMPORTANT! This is a slottable gameobject but is intended to be used only with the master weapon prefab. This is what weapon the player is using and is the weapon that will be spawned in their hands.

Fire Key: This is selectable key code. This is the key that will be used to fire the player's weapon. If left at None it will be automatically mapped to Mouse Left Click.

Reload Key: This is selectable key code. This is the key that will be used to reload the player's weapon. If left at None it will be automatically mapped to R.

Debug Weapon: This is a bool. If set to true it will allow debug lines to be drawn in game to help show where weapon is being aimed and the trajectory of the player shots.

Base Weapon Draw: This is a color. This is the color the line being drawn from the point of weapon will be if **Debug Weapon** is true.

Shot Draw: This is a color. THis is the color the line drawn for each shot fired will be and will be drawn to include the trajectory offset.

Shot Draw Time: This is a float and represents time in seconds for how long the **Shot Draw** lines will be shown in game.

AddAmmo(int): This is a public function that can be used to add ammo to the players inventory. The incoming int is how much ammo is to be added.

UpdateGun(GameObject): This is a public function that can be used to change the weapon the player is using. The incoming gameobject should be a master weapon prefab.

{}

FPS Interactable

This script is used to allow the player to interact with elements in their environment. This is to mostly be used with the *Interactable* script on other objects in the game.

Reload Key: This is selectable key code. This is the key that will be used to interact with objects that have the correct script. If left at None it will be automatically mapped to E.

Interact Range: this is a float and is the maximum distance away the player can be to interact with an object they are looking at.

{}