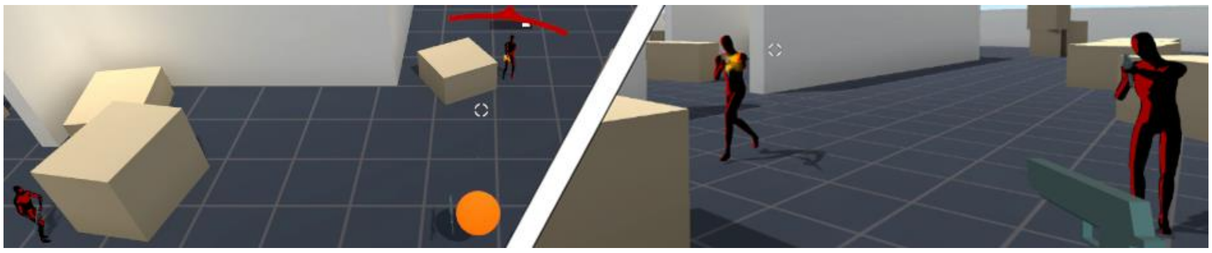
ENEMY AI



Thank you for acquiring the **Enemy AI** asset for the *Unity 3D* engine!

This asset contains ready to use enemy humanoid NPCs, featuring **patrol**, **search** and **engage** actions (with **shooting**, **reloading**, **take cover**, etc.). Provides a fast setup for any custom humanoid avatar!

The enemies can **communicate** with each other, **look** for **suspicious targets**, **protect** itself under **covers**, and **advance** in cover position, seeking for close combat, in a more aggressive **engagement.**

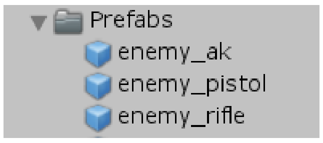
This package features an **AI** containing a fully configurable **Finite State Machine**, with predefined **states**, **actions**, **decisions** and **transitions**. The FSM is defined using **Scriptable Objects**, making easier the task to expand it with new custom states to create new NPC behaviors.

This asset also presents configurable **short** and **long** weapons for the NPC, like pistols, assault rifles, etc. Effects for **shot**, **flash**, **tracer**, and **bullet holes** are included.

Did you enjoy the asset? Please consider leaving a review on the *Unity Asset Store*, it is really important and will be very appreciated!

SETUP NOTES

In order to enable fast prototyping, this package provides **prefabs** with pre-configured enemy NPCs:



```
▼ Prefabs
    enemy_ak
    enemy_pistol
    enemy_rifle
```

You must drag them into the scene, and then set the scene dependent parameters **Aim Target** and **Patrol Waypoints** (optional), under the *StateController* class of the enemy NPC.

To fully configure a custom avatar as an enemy NPC, you will have to setup it from scratch. A detailed tutorial on how to setup the essential files on a custom project is provided by video. The video tutorial can be accessed through the following link:

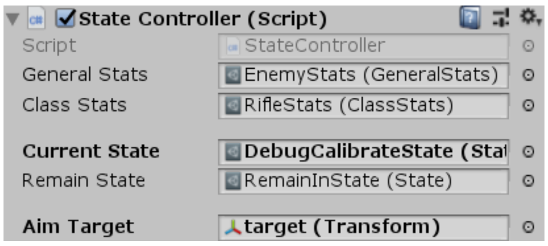
Tutorial: How to setup on a custom project

IMPORTANT NOTES

1) Due to some variations in the armature (skeleton) structure of different avatars, inconsistencies on the orientation of some bones may occur. This can cause anomalies that may need post animation adjustments.

First, when configuring a custom enemy NPC, you must calibrate the aim to focus on the desired target. Once you have set the **Aim Target** for the specific NPC (see *StateController* class), you need to change the default **Current State** in order to perform calibration.

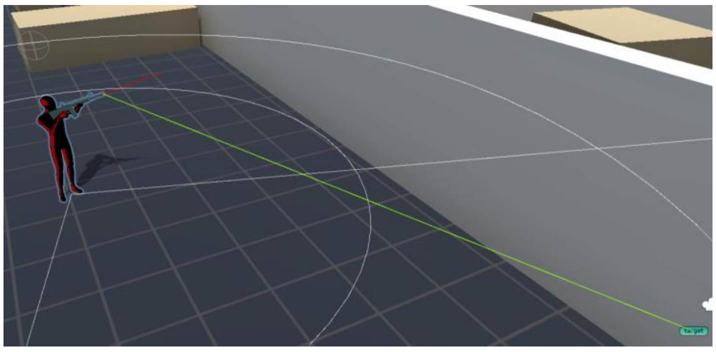
Drag the *DebugCalibrateState* to the StateController **Current State** parameter:



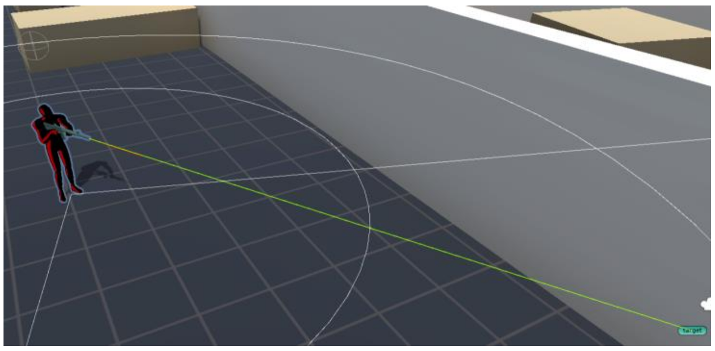
State Controller (Script)

Script	StateController
General Stats	EnemyStats (GeneralStats)
Class Stats	RifleStats (ClassStats)
Current State	**DebugCalibrateState (Stat**
Remain State	RemainInState (State)
Aim Target	**target (Transform)**

Play the scene and go to the enemy NPC. You will see the NPC aiming, and two debug lines for alignment reference (red is forward and green is towards the aim target):



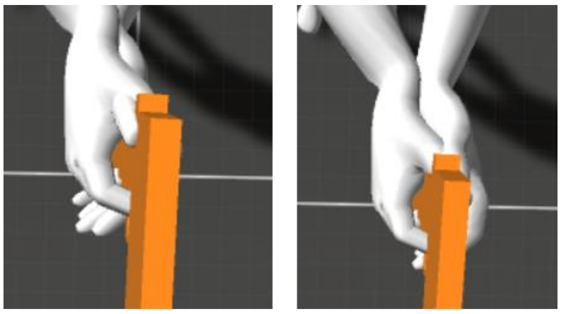
To fix the misaligned directions, open the specific *ClassStats* for the enemy NPC (in this figure is the *RifleStats* asset). Under the **Animation** section, change the **Aim Offset** parameter, until the red debug line is aligned with the green debug line:



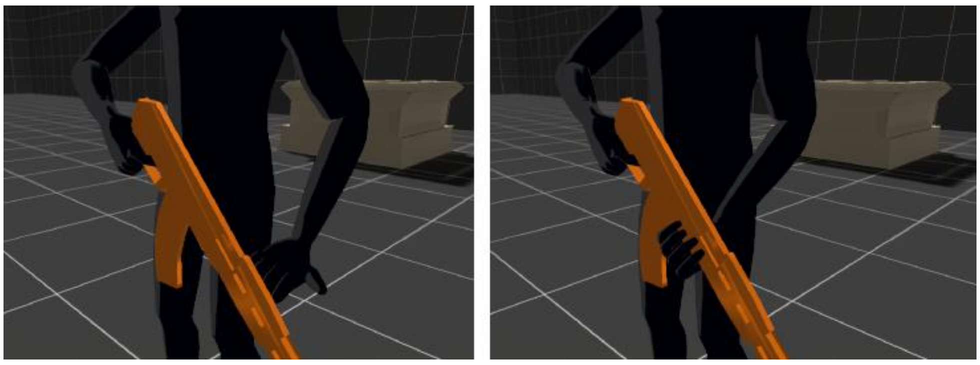
Since the *ClassStats* is a *ScriptableObject*, all changes performed during gameplay will be stored, even after the game is stopped.

After that, you may change the default **Current State** to the desired one for the enemy NPC.

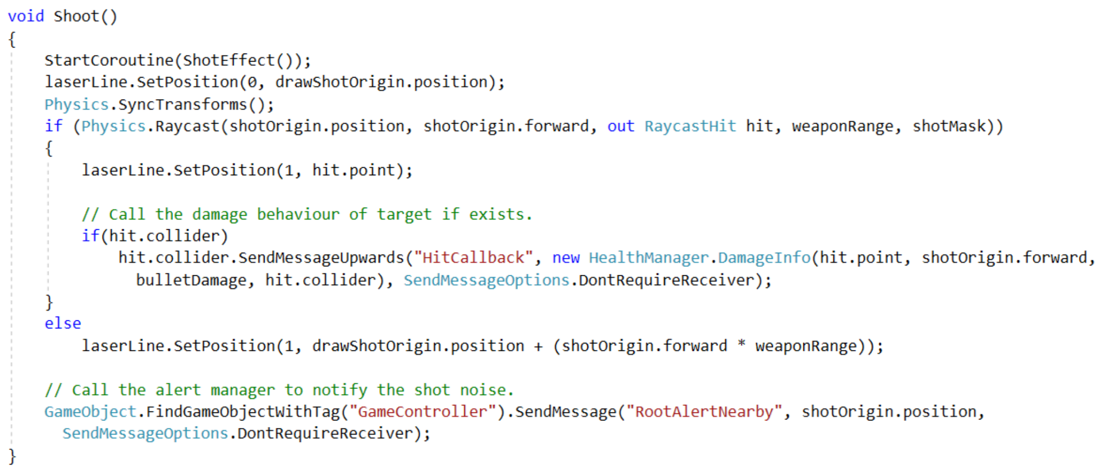
Another setup that may be required refers to next two parameters of the **Animation** section in the *ClassStats* script. The first one, the **Left Arm Aim**, is used to adjust the support hand orientation when aiming with a *short* gun:



The Second one, the **Left Arm Guard**, is used to obtain a correct weapon placement over the support hand, when standing in a guard position with a *long* gun:



2) If you are not using the **Cover + Shooting System** as your player controller, you will need to setup some extra lines of code in your player shooting component. Specifically, in the *Physics.Raycast* section of your shooting script. The figure bellow shows a simple example of a section of code with the extra lines needed.



```
void Shoot()
{
    StartCoroutine(ShotEffect());
    laserLine.SetPosition(0, drawShotOrigin.position);
    Physics.SyncTransforms();
    if (Physics.Raycast(shotOrigin.position, shotOrigin.forward, out RaycastHit hit, weaponRange, shotMask))
    {
        laserLine.SetPosition(1, hit.point);

        // Call the damage behaviour of target if exists.
        if(hit.collider)
            hit.collider.SendMessageUpwards("HitCallback", new HealthManager.DamageInfo(hit.point, shotOrigin.forward,
              bulletDamage, hit.collider), SendMessageOptions.DontRequireReceiver);
    }
    else
        laserLine.SetPosition(1, drawShotOrigin.position + (shotOrigin.forward * weaponRange));

    // Call the alert manager to notify the shot noise.
    GameObject.FindGameObjectWithTag("GameController").SendMessage("RootAlertNearby", shotOrigin.position,
      SendMessageOptions.DontRequireReceiver);
}
```

The first line you'll need to add is inside the *Raycast* conditional of your shooting function. This line checks if the target that the player is shooting has a collider, and then tries to send a message to its hit callback. If the target has a script that inherits from the *HealthManager* class (ex: the enemy NPC), the *TakeDamage* function will be called at this moment:

```
// Call the damage behaviour of target if exists.
if(hit.collider)
       hit.collider.SendMessageUpwards("HitCallback", new
HealthManager.DamageInfo( hit.point, ray.direction, weapons[weapon].bulletDamage,
hit.collider), SendMessageOptions.DontRequireReceiver);
```

The other line you'll have to add is also inside the shoot function, but outside the *Raycast* conditional. This section will call the *AlertManagement* script, added to the game controller, to propagate the noise generated by the player shot:

```
// Trigger alert callback
GameObject.FindGameObjectWithTag("GameController").SendMessage("RootAlertNearby",
ray.origin, SendMessageOptions.DontRequireReceiver);
```

NPC OVERVIEW

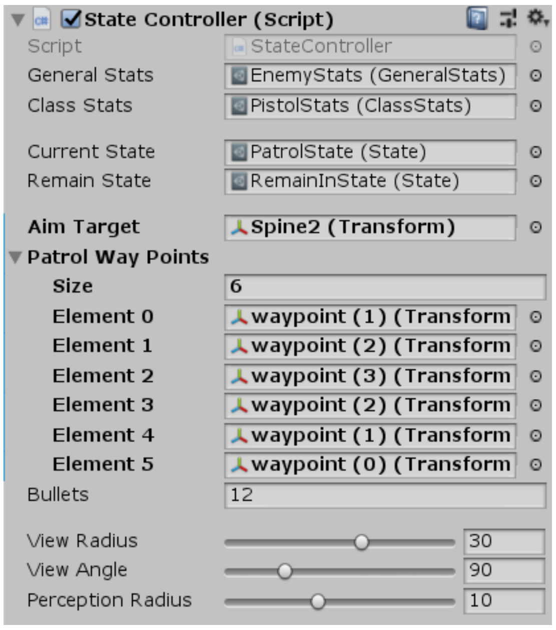
The enemy **NPC artificial intelligence** was defined by a **Finite State Machine**-based **AI** system. The core programming was based on the *Pluggable AI With Scriptable Objects* tutorial by Unity.

This system allows the user to configure the **AI** in the Unity's inspector, using **ScriptableObjects** for **states**, **actions** and **transitions** between those states, which is based on **decisions**. This setup makes easier the task of expanding the current FSM with new states and transitions. For more information, please refer to the Unity tutorial on the following link:

Tutorial: Pluggable AI With Scriptable Objects

STATE CONTROLLER

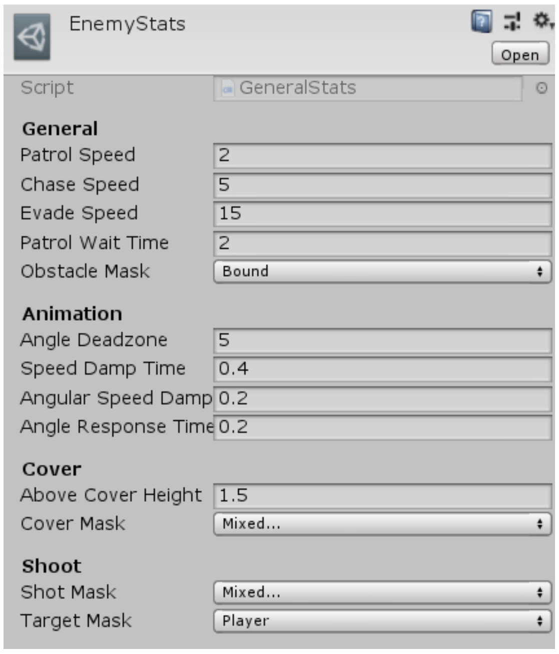
The *StateController* class is the main class for the enemy NPC. It provides references to control the state transitions for the enemy AI **FSM**. It also contains common parameters used by the states of the FSM, like the **aim target** for the enemy to aim, the **waypoints** to navigate, as well as other parameters, that is explained in details bellow.



Script	StateController
General Stats	EnemyStats (GeneralStats)
Class Stats	PistolStats (ClassStats)
Current State	PatrolState (State)
Remain State	RemainInState (State)
Aim Target	Spine2 (Transform)
Patrol Way Points	
Size	6
Element 0	waypoint (1) (Transform
Element 1	waypoint (2) (Transform
Element 2	waypoint (3) (Transform
Element 3	waypoint (2) (Transform
Element 4	waypoint (1) (Transform
Element 5	waypoint (0) (Transform
Bullets	12
View Radius	30
View Angle	90
Perception Radius	10

GENERAL STATS

The *GeneralStats* contains parameters that is common to all NPCs of the same type (ex.: all enemies). These parameters are used by the actions in the states and by the decisions that trigger the transitions of the FSM.



Script	GeneralStats
General	
Patrol Speed	2
Chase Speed	5
Evade Speed	15
Patrol Wait Time	2
Obstacle Mask	Bound
Animation	
Angle Deadzone	5
Speed Damp Time	0.4
Angular Speed Damp	0.2
Angle Response Time	0.2
Cover	
Above Cover Height	1.5
Cover Mask	Mixed...
Shoot	
Shot Mask	Mixed...
Target Mask	Player

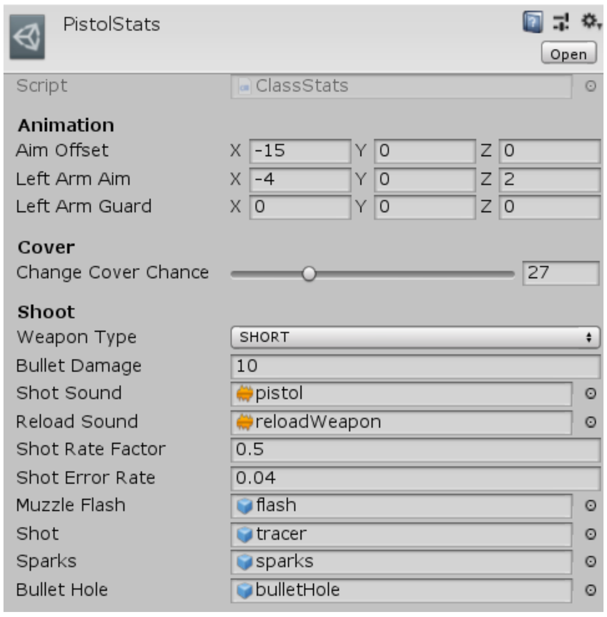
The *GeneralStats* script has the following parameters:

- **Patrol Speed:** the speed of navigation when patrolling.
- **Chase speed:** the speed of navigation when searching/chasing.
- **Evade speed:** the speed of navigation when evading/engaging.
- **Patrol Wait Time:** the amount of time to wait in a waypoint.

- **Obstacle Mask:** The obstacle layer mask, used in physics cast tests.
- **Angle Deadzone:** Clearance angle to avoid aim flickering.
- **Speed Damp Time:** Damping time for the speed parameter.
- **Angular Speed Damp Time:** Damping time for the angularSpeed parameter.
- **Angle Response Time:** Response time for turning an angle into angularSpeed.
- **Above Cover Height:** The low cover height to consider crouch when taking cover.
- **Cover Mask:** The cover layer mask, used in physics cast tests.
- **Shot Mask:** Layer mask containing all the shootable objects in scene.
- **Target Mask:** Layer mask for the enemy NPC target(s).

CLASS STATS

The *ClassStats* contains parameters that is specific to an NPC class (ex.: the weapon type carried by the enemy). These parameters are used by the actions in the states and by the decisions that trigger the transitions of the FSM.


PistolStats

Open

Script: ClassStats

Animation

Aim Offset	X -15	Y 0	Z 0
Left Arm Aim	X -4	Y 0	Z 2
Left Arm Guard	X 0	Y 0	Z 0

Cover

Change Cover Chance: 27

Shoot

Weapon Type	SHORT
Bullet Damage	10
Shot Sound	pistol
Reload Sound	reloadWeapon
Shot Rate Factor	0.5
Shot Error Rate	0.04
Muzzle Flash	flash
Shot	tracer
Sparks	sparks
Bullet Hole	bulletHole

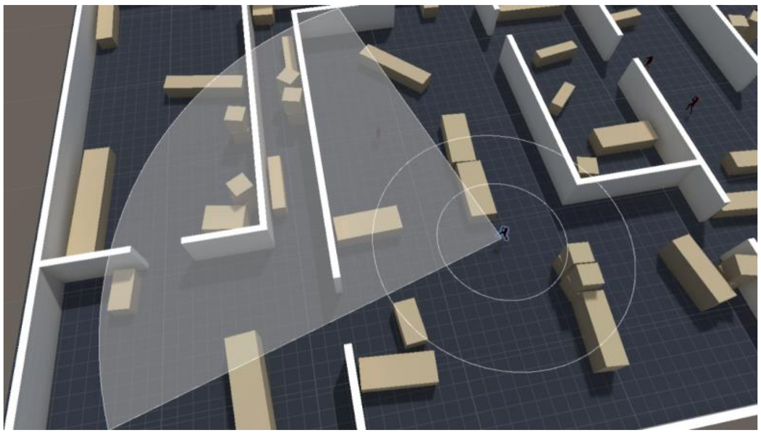
The *ClassStats* script has the following parameters:

- **Aim Offset:** Post animation aim rotation offset for custom NPC avatar.
- **Left Arm Aim:** Post animation left arm rotation when aiming with a *short* gun.
- **Left Arm Guard:** Post animation left arm rotation when guarding with a *long* gun.
- **Change Cover Chance:** Chance to change current cover to some spot that is near the target.
- **Weapon Type:** The NPC weapon type, used to set proper animations. Can be *short* or *long*.
- **Bullet Damage:** The weapon bullet damage.
- **Shot Sound, reload Sound:** The sound for shooting and reloading actions.
- **Shot Rate Factor:** The rate of firing for the weapon.
- **Shot Error Rate:** The NPC accuracy, in a centesimal error margin. Zero is perfect aim.
- **Muzzle Flash, Shot, Sparks, Bullet Hole:** Prefabs for the shot effects.

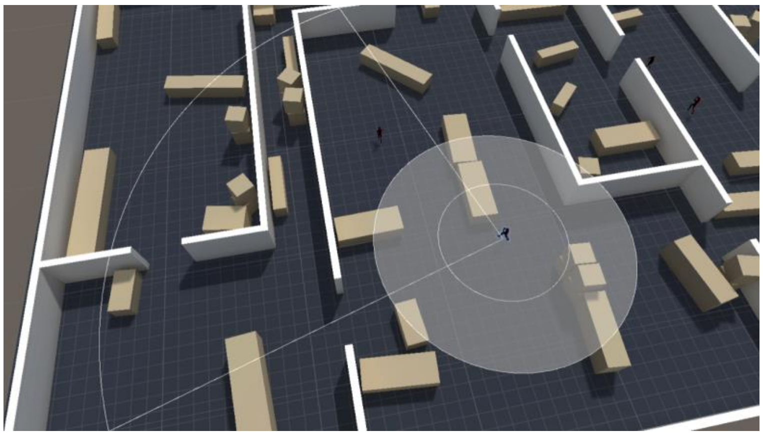
NPC SENSES

The NPC senses are tridimensional areas used to trigger the sense-based decisions on the FSM. They represent a simulation of some senses in intelligent agents. The basic NPC senses, defined on the *StateController* class, are:

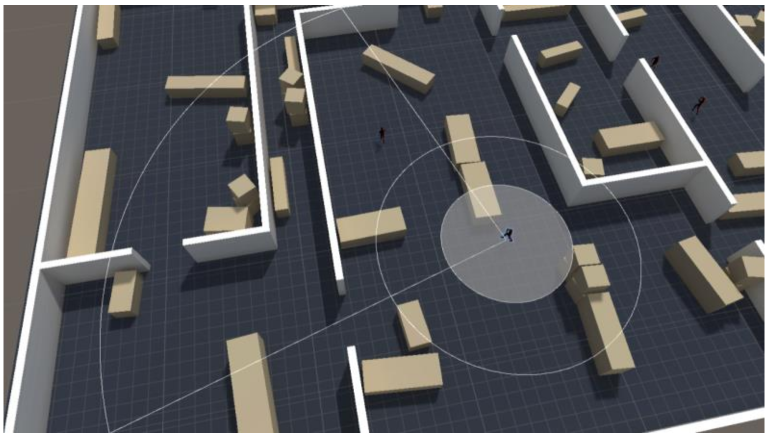
. **View** sense, defined by a **radius** and a **field of view** angle, forming a slice of a sphere. The following figure shows a 2D projection on the ground for the sense:



. **Perception** sense, defined by a **radius**, forming a sphere. Defines the **hear** and **focus** senses. The following figure shows a 2D projection on the ground for the sense:



. **Near** sense, defined by a radius that is half the perception radius, forming a sphere. The following figure shows a 2D projection on the ground for the sense:



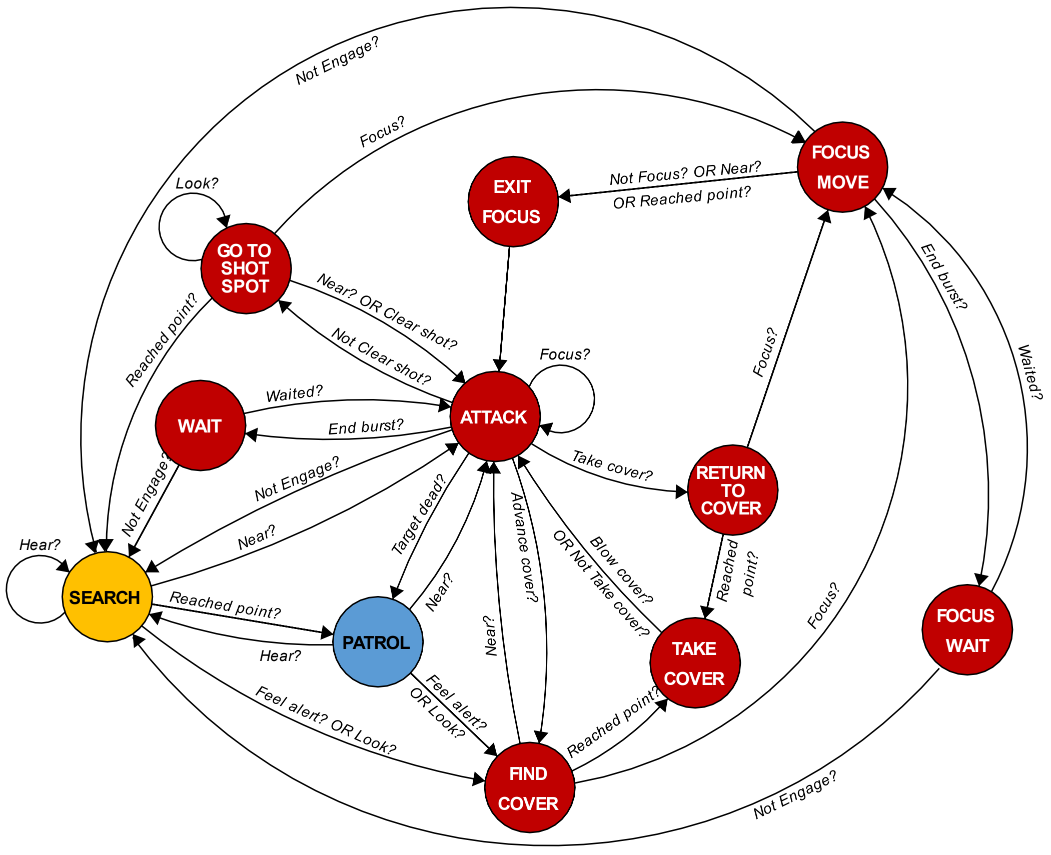
ENEMY HEALTH

The *EnemyHealth* class manages the life of the enemy NPC. It inherits from the template *HealthManager* class. The script has the following external parameters:

- **Health:** The initial/current health of the NPC.
- **Health HUD:** Prefab for the health bar canvas displayed above the enemy NPC.
- **Blood Sample:** The prefab for the game object particle emitted when the NPC is hit.
- **Headshot:** Use or not the headshot multiplier on this NPC. Default multiplier is a 10x factor.

FSM OVERVIEW

The diagram bellow provides an overview of the **Finite State Machine (FSM)** for the enemy NPC:



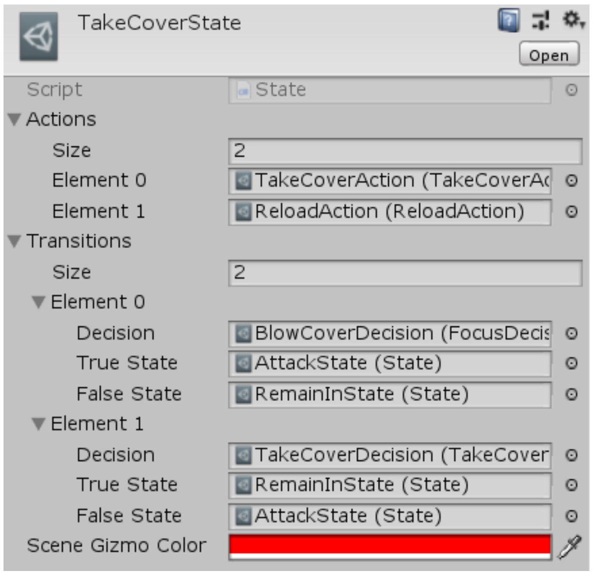
The FSM has two basic elements: The *States*, containing actions for the NPC to perform, and the *Decisions*, which defines whether or not a transition to another state will be performed.

There are 3 alert levels for the states. In order of severity: the clear, the warning, and the engage alert level. Most of the state transitions between the different alert levels are defined by the NPC senses.

How this senses and other conditions affect the current NPC state are detailed in the sections bellow.

A STATE OVERVIEW

A *State* represents a current status for the NPC. It contains *Actions* to be performed by the agent, as well as transitions to other states, defined by *Decisions*. In each iteration of the game loop, the agent will perform the actions inside the state, and will check all decisions related to the transitions in the state.



TakeCoverState

Script	State
▼ Actions	
Size	2
Element 0	TakeCoverAction (TakeCoverAc
Element 1	ReloadAction (ReloadAction)
▼ Transitions	
Size	2
▼ Element 0	
Decision	BlowCoverDecision (FocusDecis
True State	AttackState (State)
False State	RemainInState (State)
▼ Element 1	
Decision	TakeCoverDecision (TakeCover
True State	RemainInState (State)
False State	AttackState (State)
Scene Gizmo Color	🟥

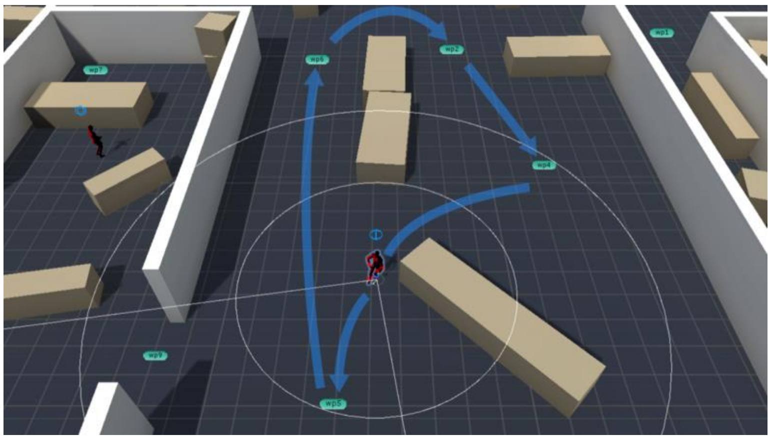
A typical state has the following parameters:

- **Actions:** An array of actions to be performed by the NPC AI when in the current state.
- **Transitions:** A list of transitions from the current state to other states in the FSM. A transition is composed by a decision and two states: one state to go to in case the decision is true, and one to go to otherwise.
- **Scene Gizmo Color:** A debug parameter, related to the state alert level. Sets the debug orb color for the state.

The following sections will provide an overview for each **state** in the FSM. Then, the next sections will provide a description of all **actions** and **decisions** used by the FSM **states** and **transitions**.

PATROL

The **Patrol** state is a clear level state where the NPC **navigates** over defined **waypoints**. The waypoints are defined on the *StateController* class. If no waypoint is set, the NPC will **guard** his **position**.



STATE ACTIONS

There is only 1 action in this state: The **Patrol** action.

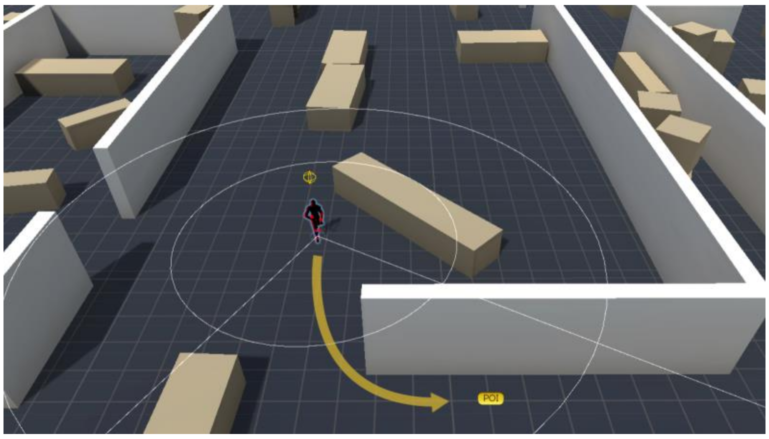
STATE TRANSITIONS

There are 4 transitions from this state (in order of precedence):

- **Near?** decision**:** if *true*, go to the **Attack** state. If *false*, remain in state.
- **Feel Alert?** decision: if *true*, go to the **Find Cover** state, if *false*, remain in state.
- **Look?** decision: if *true*, go to the **Find Cover** state, if *false*, remain in state.
- **Hear?** decision: if *true*, go to the **Search** state, if *false*, remain in state.

SEARCH

The **Search** state is a warning level state where the NPC navigates to a **suspicious** position to **investigate** it. If the suspicion is not confirmed, the alert level is lowered and NPC leaves the state. Otherwise, the alert is raised, and the NPC will enter *engage* level states.



STATE ACTIONS

There is only 1 action in this state: The **Search** action.

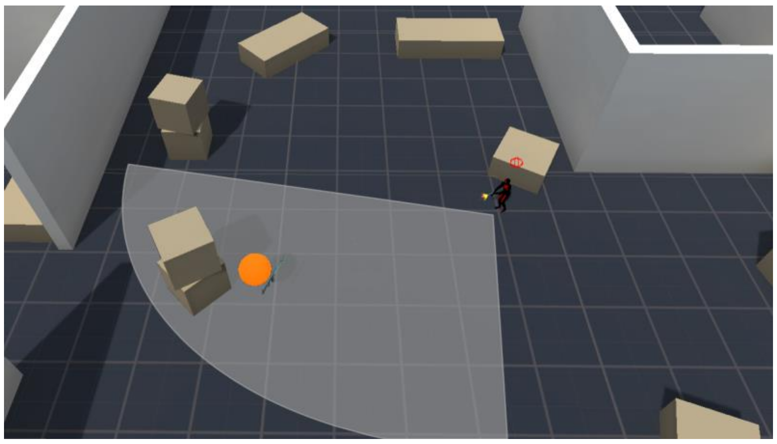
STATE TRANSITIONS

There are 5 transitions from this state (in order of precedence):

- **Near?** decision**:** if *true*, go to the **Attack** state. If *false*, remain in state.
- **Feel Alert?** decision: if *true*, go to the **Find Cover** state, if *false*, remain in state.
- **Look?** decision: if *true*, go to the **Find Cover** state, if *false*, remain in state.
- **Reached Point?** decision: if *true*, go to the **Patrol** state, if *false*, remain in state.
- **Hear?** decision: remain in state, regardless of the outcome. Used to update the NPC current *personal target*.

ATTACK

The **Attack** state is, together with the *Focus Move* state, the main engage level state. Here, the NPC **fires** against the target, **standing still**. For realism, the NPC will alternate between variable firing **bursts** and **waiting** time. Every **attack** round has a random burst of fire, the number of bullets shot. The burst is also stopped when no bullets on mag is left, making the NPC stop to **reload** his weapon. If the NPC loses **sight** of the target, he will keep engaging for a period, until the alert level is lowered and NPC the leaves the state.



STATE ACTIONS

There are 2 actions in this state: The **Attack** and the **Spot Focus** actions.

STATE TRANSITIONS

There are 7 transitions from this state (in order of precedence):

- **Target Dead?** decision**:** if *true*, go to the **Patrol** state. If *false*, remain in state.
- **Advance Cover?** decision: if *true*, go to the **Find Cover** state, if *false*, remain in state.
- **Clear Shot?** decision: if *true*, remain in state, if *false*, go to the **Go to Shot Spot** state.
- **Engage?** decision: if *true*, remain in state, if *false*, go to the **Search** state.

- **End Burst?** decision: if *true*, go to the **Wait** state, if *false*, remain in state.
- **Take Cover?** decision: if *true*, go to the **Return to Cover** state, if *false*, remain in state.
- **Focus?** decision: remain in state, regardless of the outcome. Used to update the NPC current *personal target*.

WAIT

The **Wait** state is an engage level state where the NPC **waits in place**, between firing bursts, without losing track of the target. Every **wait** round has a random wait time. The wait time can be longer, if the NPC is **reloading** his weapon.

STATE ACTIONS

There is only 1 action in this state: The **Reload** action.

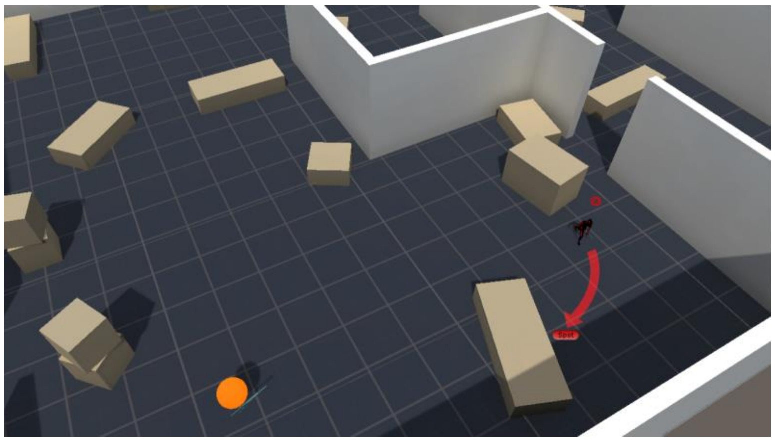
STATE TRANSITIONS

There are 2 transitions from this state (in order of precedence):

- **Waited?** decision**:** if *true*, go to the **Attack** state. If *false*, remain in state.
- **Engage?** decision: if *true*, remain in state, if *false*, go to the **Search** state.

FIND COVER

The **Find Cover** state is an engage level state where the NPC seeks a **cover spot** to **protect** itself from shots. The NPC will choose the nearest cover spot that protects from the **target sight**, ignoring cover spots with the target on the path.



STATE ACTIONS

There is only 1 action in this state: The **Find Cover** action.

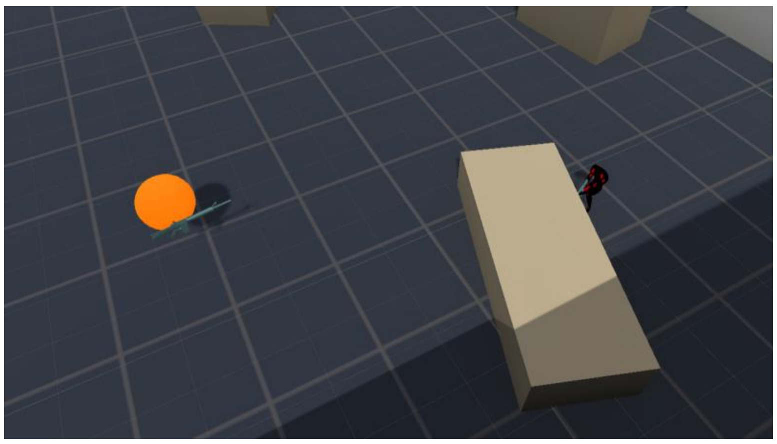
STATE TRANSITIONS

There are 3 transitions from this state (in order of precedence):

- **Near?** decision**:** if *true*, go to the **Attack** state. If *false*, remain in state.
- **Reached Point?** decision**:** if *true*, go to the **Take Cover** state. If *false*, remain in state.
- **Focus?** decision**:** if *true*, go to the **Focus Move** state. If *false*, remain in state.

TAKE COVER

The **Take Cover** state is an engage level state where the NPC protects itself from the target line of sight, using a cover spot. Every **take cover** round has a random wait time. The wait in cover time can be longer, if the NPC is reloading his weapon.



STATE ACTIONS

There are 2 actions in this state: The **Take Cover** and the **Reload** actions.

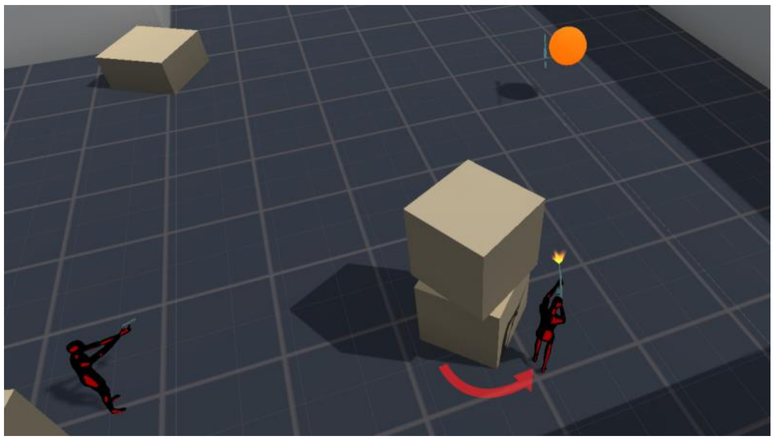
STATE TRANSITIONS

There are 2 transitions from this state (in order of precedence):

- **Blow Cover?** decision**:** if *true*, go to the **Attack** state. If *false*, remain in state.
- **Take Cover?** decision: if *true*, remain in state, if *false*, go to the **Attack** state.

GO TO SHOT SPOT

The **Go to Shot Spot** state is an engage level state where the NPC leaves his current cover spot to find a **clean line of sight** to the target. That shot spot is where the NPC will fire shoots against the target. If the NPC misses the target sight after reaching the last seen position, the alert level is lowered and NPC leaves the state.



STATE ACTIONS

There is only 1 action in this state: The **Go to Shot Spot** action.

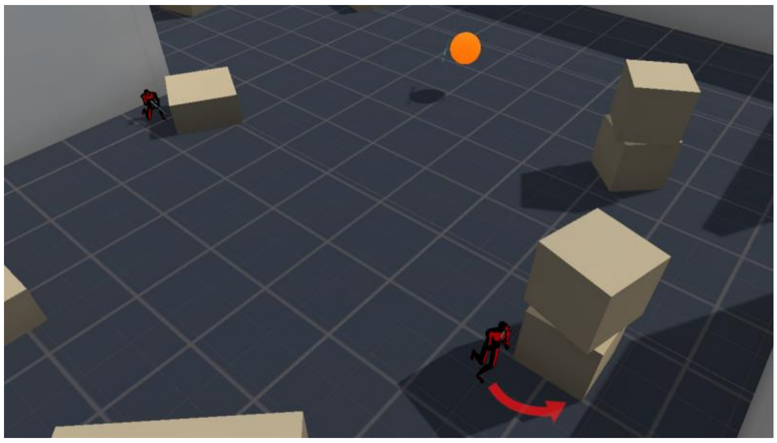
STATE TRANSITIONS

There are 5 transitions from this state (in order of precedence):

- **Near?** decision**:** if *true*, go to the **Attack** state. If *false*, remain in state.
- **Focus?** decision**:** if *true*, go to the **Focus Move** state. If *false*, remain in state.
- **Clear Shot?** decision: if *true*, go to the **Attack** state, if *false*, remain in state.
- **Reached Point?** decision**:** if *true*, go to the **Search** state. If *false*, remain in state.
- **Look?** decision: remain in state, regardless of the outcome. Used to update the NPC current *personal target*.

RETURN TO COVER

The **Return to Cover** state is an engage level state where the NPC leaves his current engagement position to return to his current cover spot, protecting from incoming fire.



STATE ACTIONS

There is only 1 action in this state: The **Return to Cover** action.

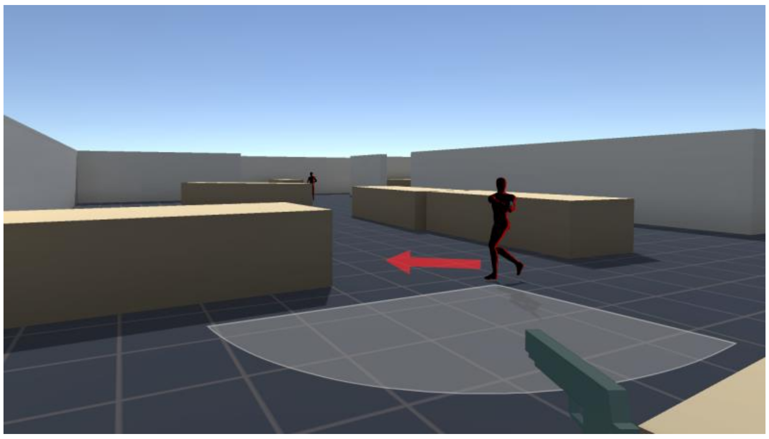
STATE TRANSITIONS

There are 2 transitions from this state (in order of precedence):

- **Reached Point?** decision**:** if *true*, go to the **Take Cover** state. If *false*, remain in state.
- **Focus?** decision**:** if *true*, go to the **Focus Move** state. If *false*, remain in state.

FOCUS MOVE

The **Focus Move** state is, together with the *Attack* state, the main engage level state. This state is achieved when the NPC is **navigating** to a predefined position and the target **approaches** or **shoots** him. Here, the NPC will **fire** against the target, while **strafing** to the predefined position. To avoid vulnerability, the NPC will always face the target while strafing, providing a more challenging engagement.



STATE ACTIONS

There are 2 actions in this state: The **Attack** and the **Focus Move** actions.

STATE TRANSITIONS

There are 5 transitions from this state (in order of precedence):

- **Engage?** decision: if *true*, remain in state, if *false*, go to the **Search** state.
- **Focus?** decision: if *true*, remain in state, if *false*, go to the **Exit Focus** state.
- **Near?** decision: if *true*, go to the **Exit Focus** state. If *false*, remain in state.
- **Reached Point?** decision**:** if *true*, go to the **Exit Focus** state. If *false*, remain in state.
- **End Burst?** decision: if *true*, go to the **Focus Wait** state, if *false*, remain in state.

FOCUS WAIT

The **Focus Wait** state is an engage level state where the NPC **strafes** to a predefined position, **without shooting** the target. Every round has a random **wait** time. The wait time can be longer, if the NPC is reloading his weapon.

STATE ACTIONS

There are 2 actions in this state: The **Focus Move** and the **Reload** actions.

STATE TRANSITIONS

There are 2 transitions from this state (in order of precedence):

- **Waited?** decision**:** if *true*, go to the **Focus Move** state. If *false*, remain in state.
- **Engage?** decision: if *true*, remain in state, if *false*, go to the **Search** state.

EXIT FOCUS

The **Exit Focus** state is an engage level state where the NPC **leaves** the **focus** category states. It is a **transitory** state, activated when the target **leaves** the NPC **proximity**, or a **point of interest** is reached by the NPC.

STATE ACTIONS

There is only 1 action in this state: The **Exit Focus** action.

STATE TRANSITIONS

There is only 1 transition from this state:

- **Focus?** decision**:** go to the **Attack** state, regardless of the outcome. Used only to trigger a state transition.

ACTIONS OVERVIEW

An overview of all actions used by the FSM states (in alphabetical order):

ATTACK ACTION

The NPC will **engage** with the target, **shooting** him with **bursts** of fire. If the target is not on sight, the last seen position is the engagement target, until a predefined engagement time.

EXIT FOCUS ACTION

The NPC will **stop focusing** on the target and will **exit** the **strafe** mode.

FIND COVER ACTION

The NPC will **search** for the **nearest cover** spot, if any.

FOCUS MOVE ACTION

The NPC will **strafe** to a **point of interest**, while **focusing** on player.

GO TO SHOT SPOT ACTION

The NPC will **leave** its **cover spot**, on direction to the target, **seeking** for a **clean sight** to engage.

PATROL ACTION

The NPC will **navigate** through predefined **waypoints**, or **guard** a **position**. Uses the *patrol speed* for navigation.

RELOAD ACTION

The NPC will **reload** its weapon.

RETURN TO COVER ACTION

The NPC will **leave** the current engage **position** and **return** to the **cover** spot.

SEARCH ACTION

The NPC will **navigate** to his current *personal target*, to **investigate**. Uses the *chase speed* for navigation.

SPOT FOCUS ACTION

The NPC will **orientate** itself to the target, always **aiming** at him, while **standing still**.

TAKE COVER ACTION

The NPC will **protect** itself from incoming fire on a **cover** spot.

DECISIONS OVERVIEW

An overview of all decisions used by transitions between FSM states (in alphabetical order):

ADVANCE COVER?

Uses a predefined **chance** to **change** current **cover** spot. Only **advances** if the new cover spot is near the target than the current one.

BLOW COVER?

Uses the NPC **perception** sense. If any target is **within** the sense radius, with a clear view to it, *or* an **alert** is received (incoming shots or alerts by other NPCs), the decision will be triggered. This decision **invalidates** the current NPC cover spot.

CLEAR SHOT?

Check if no nearby covers is an **obstacle** to **aim** properly, *and* if no obstacle is on **sight** to target.

END BURST?

If a certain number of **bullets** is **fired**, the decision will be triggered. The amount is a random number, defined on each round.

ENGAGE?

If the target is **not on sight**, the decision checks if the NPC will keep engaging with his **personal target** (the last seeing position), based on a predefined amount of time.

FEEL ALERT?

Uses the externally set variable **feel alert** (See *AlertManagement* class). Triggered by external events, like incoming **shots** *or* forwarded **alerts** by other NPCs.

FOCUS?

Uses the NPC **perception** sense. If any target is within the sense radius *and* the NPC has a clear view to it, the decision will be triggered.

HEAR?

Uses the NPC **perception** sense. If any target is within the sense radius *and* is in movement, the decision will be triggered.

LOOK?

Uses the NPC **view** sense. If any target is within the sense radius *and* **FOV**, *and* the NPC has a clear view to it, the decision will be triggered.

NEAR?

Uses the NPC **near** sense. If any target is within the sense radius, *and* the NPC has a clear view to it, the decision will be triggered. This decision **invalidates** the current NPC cover spot.

REACHED POINT?

The decision is triggered when the NPC reaches the current navigation target.

TAKE COVER?

Uses a random amount of time to stand on the current cover spot, if any.

TARGET DEAD?

Checks if the target is still alive. Uses the target's *HealthManager* class.

WAITED?

The decision is triggered when the current **wait** time is achieved. The wait time is set randomly on each round.

EXTRAS

This section contains extra information about other assets not directly related to the enemy NPC, like game controller related classes.

ALERT MANAGEMENT

The *AlertManagement* is a game controller class, used to propagate any alert related event. An example of usage is to propagate the noise of the shots between the enemy NPCs, when the player fires. It must be called by the player every time it shoots, using the *RootAlertNearby* function.

If extra waves are used, all enemy NPC that receives the alert will propagate it to all other enemies within its radius, and then subsequently, until the number of propagations waves are reached.

The script has the following external parameters:

- **Alert Radius:** A radius to limit the alert propagation for the wave.
- **Extra Waves:** How many extra waves of propagation will be performed.
- **Alert Mask:** The layer mask for the entities that will be alerted (usually the enemy NPC).

HEALTH MANAGER

The *HealthManager* is a template class to control any game entity life. It is used by both NPC and player, as all as any other object in the game that may suffer damage.

This class contains a basic signature for the *TakeDamage* function, called externally, every time the entity suffers a damage. It is used, for example, by the enemy NPC AI: it tries call the *TakeDamage* function for every target it shoots at.

For each new type of entity, a custom class, inherited from the *HealthManager* class, must be created, containing an implementation for the *TakeDamage* function, as well as any extra features, specific to the entity type.

ACKNOWLEGMENT

The author acknowledges some third-party assistance that facilitated the production of the demo examples within this asset. This includes the great *Unity Tutorials*, the *Carnegie-Mellon University* mocap library, the *SoundBible.com* and *Freesound.org* websites for the free sound samples, and the fellows that contributes to the *Unity Answers* website.

DISCLAIMER

Did you enjoy the asset? Please consider leaving a review on the *Unity Asset Store*, it is really important and will be very appreciated!

Support contact

Author's page