DOCUMENTATION

AI SYSTEM

Produced By:
Jorge Silva

Last Update:
17/02/2022 (Version: 1.00)

Index

Update Log

Version: 1.00 (17/02/2022) – Initial Documentation

Version: 1.01 (23/03/2022) – Tool updated, fixed a bug where the chase behavior would be stuck when losing sight of the player.

Introduction

Welcome to AI System, we thank you for using our tools and hope you enjoy as much as we do.

This documentation will provide every information related to this tool, how do you work with it, and how can you expand from here.

We know that every game has is own way of doing things and probably some behaviors or logic used in this tool might not fulfill your desires, so with this in mind we developed this in a way that is possible to you to change or create new behaviors. We will keep in mind these changes and we’ll try not to update the behavior system but before updating this tool backup all your new behaviors.

We also tried to keep this in a way that you don’t have to change your original Logic so this system will run separately from your Initial Logic.

Getting Started

Installation

If you download this asset from the *Asset Store*, its very likely that the addon will be installed automatically, but if you didn't download from the *Asset Store* follow the next Steps:

- Download the Package.
- Open Unity.
- Go to **Assets > Import Package > Custom Package**.
- Import the package.

First Run

After importing everything, in the top toolbar you'll now see a new button, *"AI System"*, here you will be prompt with a window with all the information related to this tool.

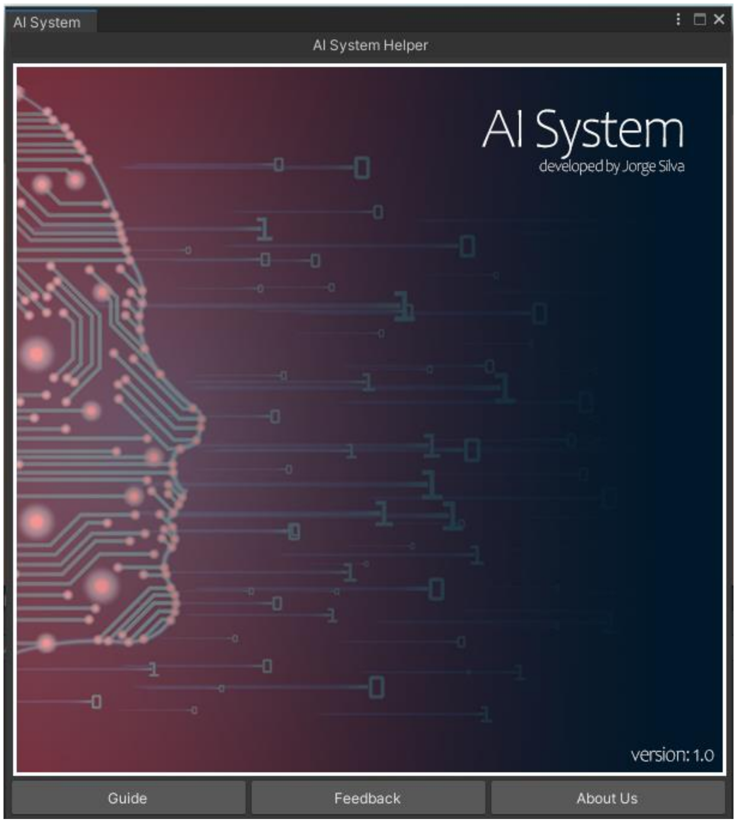
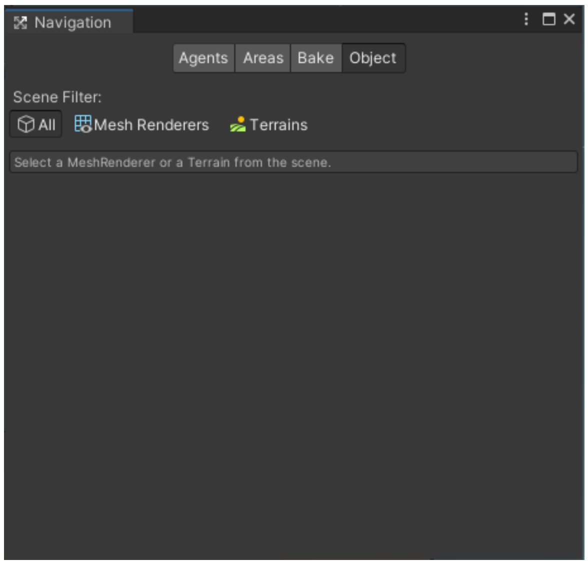


Figure 1 - AI System Helper

Setup the Scene

So, for this to work properly and if you haven't done it already or know about it, you must set up some stuff first. The Logic behind this tool is that you need a Nav Mesh Agent Component to your object and set the environment in a way to be able to bake a Navigation Plane. Follow the instructions if you don't know how to do it, for simplicity the following instructions will be an example, you should set this the way is more convenient for your project.

1. Select all objects related to the environment, then in the inspector in the drop down left of the *Static* text select ***Navigation Static***
2. Go to **Window > AI > Navigation**, and a window should appear



Select a MeshRenderer or a Terrain from the scene.

Figure 2 - Unity AI Navigation Panel

3. In the *Bake* tab press the button that says **Bake** and you're all set you should now see in the Scene View your floor painted with a bluish color.
   a. **NOTE:** We're not going to cover every single setting in the Navigation, for this purpose check the information [here(unity3d.com/Manual/Navigation)](unity3d.com/Manual/Navigation).

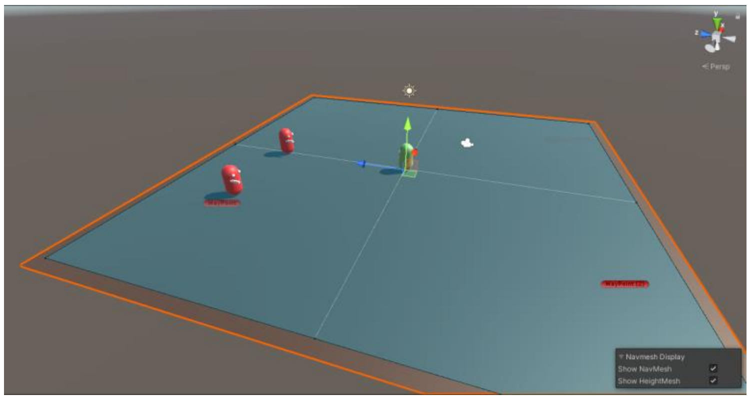


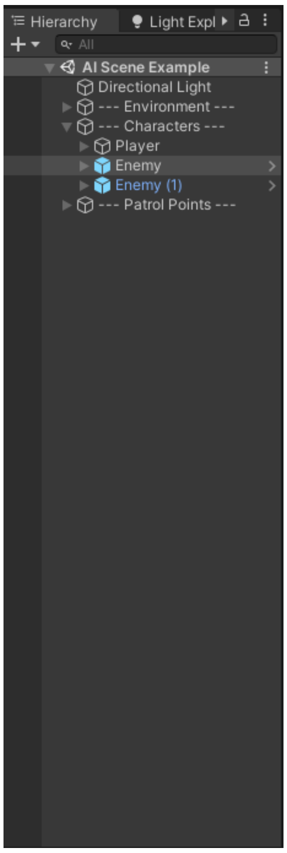
Figure 3 - After Navigation Bake

Add AI System to your Game

After everything is setup, now it’s time to see this tool in action, we recommend that you check the Example Scene in the Scene Folder inside AI System Folder, there you can have an overview how everything is setup.

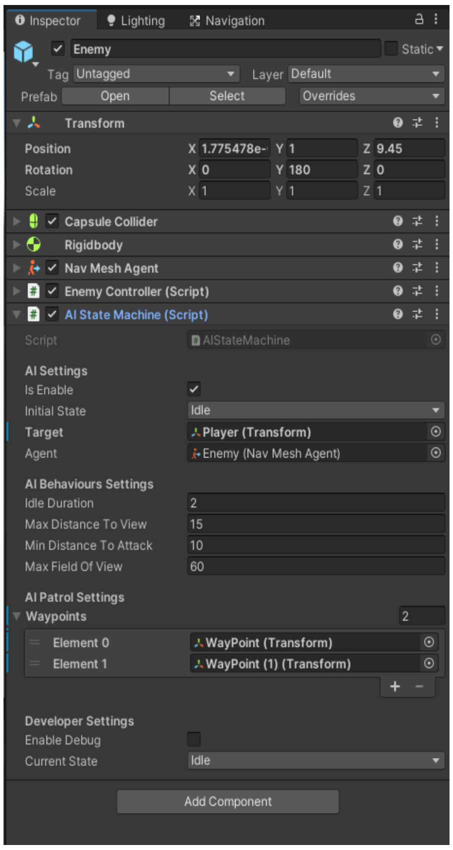
The following instructions will be representing an Enemy.

1. Find your Enemy Prefab on the Hierarchy
2. If your enemy doesn’t have a Nav Mesh Agent Component attached, for this to work you need to attach that component to it.
3. Attach the *AI State Machine* script.



```
AI Scene Example
  Directional Light
  --- Environment ---
  --- Characters ---
    Player
    Enemy
    Enemy (1)
  --- Patrol Points ---
```

Figure 4 - Hierarchy



Enemy — Static
Tag: Untagged — Layer: Default

Transform
- Position: X 1.775478e- Y 1 Z 9.45
- Rotation: X 0 Y 180 Z 0
- Scale: X 1 Y 1 Z 1

Capsule Collider
Rigidbody
Nav Mesh Agent
Enemy Controller (Script)
AI State Machine (Script)

Script: AIStateMachine

AI Settings
- Is Enable: ✓
- Initial State: Idle
- Target: Player (Transform)
- Agent: Enemy (Nav Mesh Agent)

AI Behaviours Settings
- Idle Duration: 2
- Max Distance To View: 15
- Min Distance To Attack: 10
- Max Field Of View: 60

AI Patrol Settings
- Waypoints: 2
  - Element 0: WayPoint (Transform)
  - Element 1: WayPoint (1) (Transform)

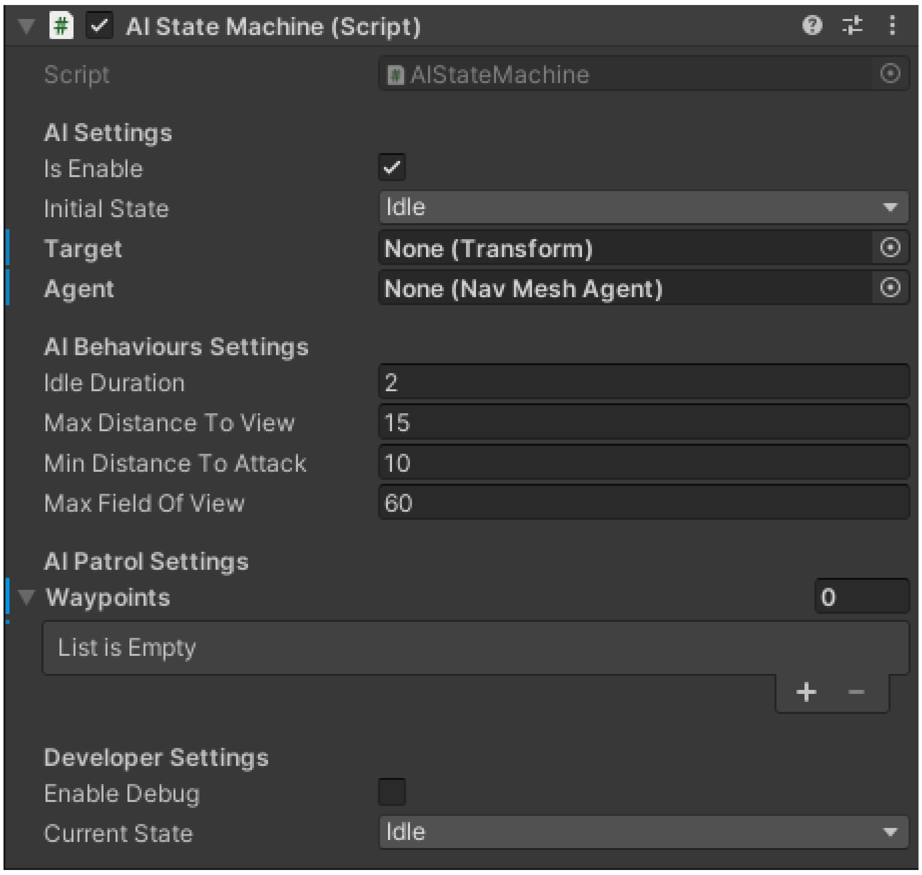
Developer Settings
- Enable Debug: ☐
- Current State: Idle

Figure 5 - Inspector

Figure 6 - AI Navigation

4. Add the *AI State Machine* to every object that needs AI.

Understanding *AI State Machine*


AI State Machine (Script)

Script	AIStateMachine
AI Settings	
Is Enable	✓
Initial State	Idle
Target	None (Transform)
Agent	None (Nav Mesh Agent)
AI Behaviours Settings	
Idle Duration	2
Max Distance To View	15
Min Distance To Attack	10
Max Field Of View	60
AI Patrol Settings	
Waypoints	0
List is Empty	
Developer Settings	
Enable Debug	
Current State	Idle

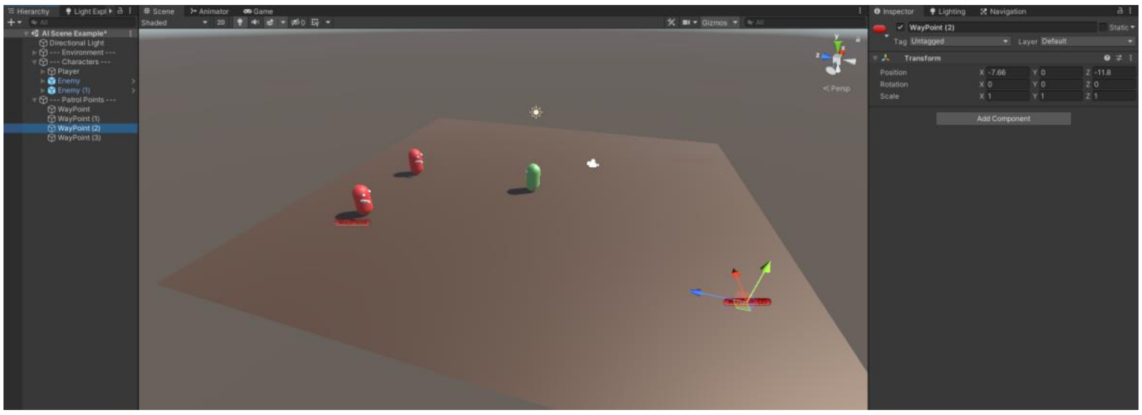
Figure 7 - AI State Machine in Inspector

1. **AI Settings**
   a. **Is Enable:** When active the system will run as expected, this was added because sometimes you need the system and sometimes you don't.
   b. **Initial State:** You can set what is the initial state of the AI System, imagine you have a cut scene and after the cutscene is done you want your object to attack straightway.
   c. **Target:** Set here what you want the AI system to search for.
   d. **Agent:** Although you can attach the object to itself, even if you don't on the start it will get that component.
2. **AI Behaviours Settings**
   a. **Idle Duration:** The amount of time that the system will stay idle.
   b. **Max Distance To View:** The maximum distance between the object and the target to be in range.
   c. **Min Distance to Attack:** The distance required so the object can attack the target.
   d. **Max Field of View:** The angle in which the object can see the Target.
3. **AI Patrol Settings**
   a. **Waypoints:** Empty GameObject that references a point in which the object needs to go.

4. **Developer Settings**
   a. **Enable Debug:** When enable you'll get console logs in the Console to check if everything is working properly
   b. **Current State:** You can see if the system is working properly with this variable.

Creating Waypoints

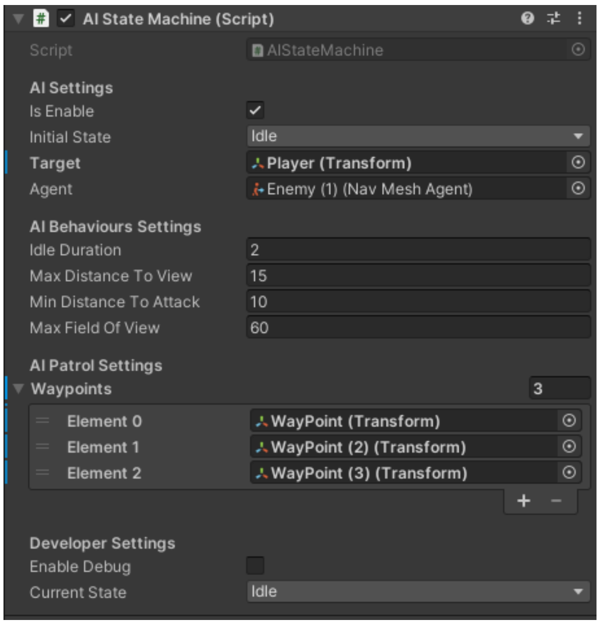
As expected, everyone has its own way of organizing but we recommend that you create an Empty Game Object with a name of *"--- Waypoints ---"* and there add Empty Game Objects and set them in the world where you want the AI System to patrol/wander to.



WayPoint (2)
Tag Untagged Layer Default
Transform
Position X -7.66 Y 0 Z -11.8
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Figure 8 - Waypoints

After creating a waypoint, add it to the Waypoints List in the *AI State Machine*.



AI State Machine (Script)

Script	AIStateMachine
AI Settings	
Is Enable	✓
Initial State	Idle
Target	Player (Transform)
Agent	Enemy (1) (Nav Mesh Agent)
AI Behaviours Settings	
Idle Duration	2
Max Distance To View	15
Min Distance To Attack	10
Max Field Of View	60
AI Patrol Settings	
Waypoints	3
Element 0	WayPoint (Transform)
Element 1	WayPoint (2) (Transform)
Element 2	WayPoint (3) (Transform)
Developer Settings	
Enable Debug	
Current State	Idle

Figure 9 - Waypoints Setup

Multiple AI

Yes, as explained before, you can add as much AI State Machines as you like. Just remember that for this system to work, you need to Set the Navigation, add Nav Mesh Agents to the Object you want the AI System to Work.

Setting Your Character to Work with AI System

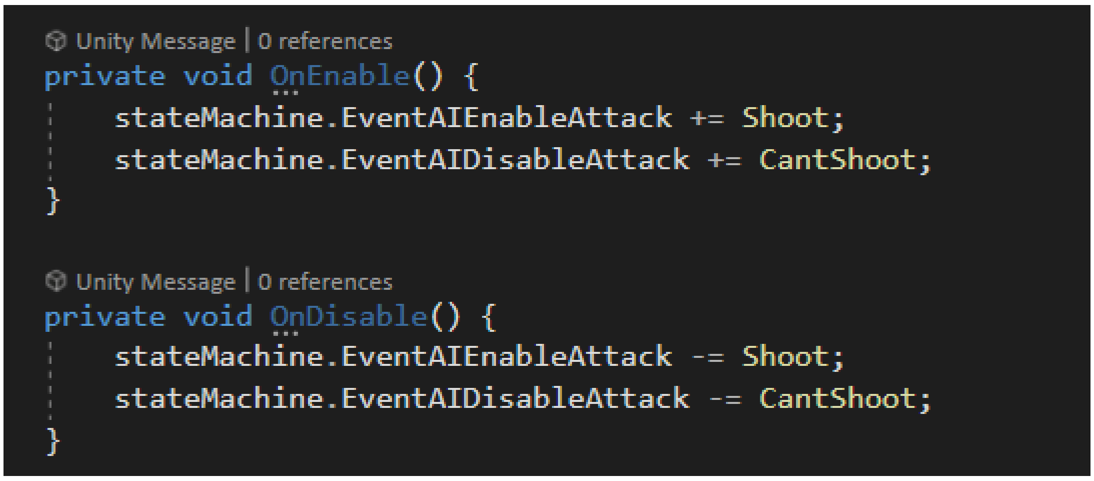
You might be wondering, “Ok, I have the system but what now? What can I do with it?”.

To help you out we’ve implemented Event logic to our AI System, this mean if you have a controller for your enemy, you just need to *Register* the event and every time the AI System works it will fire that event and you can apply your own logic for what his happening.

Imagine you have a complex Enemy/NPC controller, and now you want to implement the AI System, in your controller you need to add the *using jcsilva.AISystem,* then create a Reference to the AI State Machine, and then you need to register the events (you can consult the current events in the ***Event List***).

In a case that you’re getting a NullPointerException, use the Awake Method to get the component.

If you have never worked with events the following image will show how to Register and Unregister events.



```
Unity Message | 0 references
private void OnEnable() {
    stateMachine.EventAIEnableAttack += Shoot;
    stateMachine.EventAIDisableAttack += CantShoot;
}

Unity Message | 0 references
private void OnDisable() {
    stateMachine.EventAIEnableAttack -= Shoot;
    stateMachine.EventAIDisableAttack -= CantShoot;
}
```

Figure 10 - Example of Registering Attack Events

In the example above, we declare that the Event AI Enable Attack, when triggered will call the method Shoot, in this case this method only throws a Debug.log to the console saying I’m shooting. Check the next page, it will have a example of a Enemy Controller.

```
using UnityEngine;
using jcsilva.AISystem;

public class EnemyController : MonoBehaviour {

        [Header("References")]
        [SerializeField] AIStateMachine stateMachine;

        [Header("Enemy Settings")]
        [SerializeField] float fireRate = 1f;
        [SerializeField] float elapsedTime;

        private bool canShoot;

        private void Awake() {
            if (stateMachine == null) {
                stateMachine = GetComponent<AIStateMachine>();
            }
        }

        private void OnEnable() {
            stateMachine.EventAIEnableAttack += Shoot;
            stateMachine.EventAIDisableAttack += CantShoot;
        }

        private void OnDisable() {
            stateMachine.EventAIEnableAttack -= Shoot;
            stateMachine.EventAIDisableAttack -= CantShoot;
        }

        // Update is called once per frame
        void Update() {
            if (canShoot) {
                if(elapsedTime > fireRate) {
                    Shoot();
                    elapsedTime = 0f;
                } else {
                    elapsedTime += Time.deltaTime;
                }
            } else if (!canShoot && elapsedTime > 0f) {
                if(elapsedTime > fireRate) {
                    elapsedTime = 0f;
                } else {
                    elapsedTime += Time.deltaTime;
                }
            }

        }

        private void Shoot() {
            canShoot = true;
        }

        private void CantShoot() {
            canShoot = false;
        }

        private void IsShooting() {
            Debug.Log("I'm Shooting");
        }
    }
```

Figure 11 - Example of Enemy Controller

Add New Behaviours

You can add more behaviours as you like, although the current tool isn't set to be friendly in this area (we're still developing better ways for you to use this 😊), you can create new behaviours using the following instructions (we recommend that you check Behaviour Example):

1. In the Project Tab Navigate to: **AI System > Scripts > AI System > AI Setup**
2. Open AIStates Script (Enum Class)
   a. Insert the new State/s bellow the commented line, this is important, the order in which the AIStates are set is the order which the State Machine will work.
3. Open AIEvents (Enum Class)
   a. Insert the new Event/s bellow the commented line.
4. Create a new C# Script
   a. The location of this script isn't a requirement, you can organize/store whenever you like.
   b. In the beginning of the script add **using jcsilva.AISystem**, so you can have access to the AI System.
   c. Replace the Monobehaviour Inheritance to AIBehaviour
   d. Add the un-implemented methods.
   e. Create a Class constructer with the Monobehaviour and AIStateMachine parameters)
   f. The method **OnBehaviourStart**, will be called whenever the behaviour start is called
   g. The method **OnBehaviourEnd**, will be called whenever the behaviour end is called
   h. The method **OnUpdate**, is called on the update of the AI State Machine.
5. Go back to the **AI System > Scripts > AI System** and open the AI State Machine Script
   a. If you want to keep the Event Logic for easier use, create new Public Actions.
   b. Case you added new Events
      i. In the **nextEvent** dictionary, add the event with the corresponding State.
   c. In the method **InitializeBehaviours** add the new behaviour after the commented line. If you used the same initial logic, you just need to pass as arguments *this* and *this*.
      i. **Note that the order in the AIStates needs to be the same as the list l_behaviours**.

Event List

This is a list of the Current Events in the tool:

1. ***Idle Behaviour***
    a. **EventAIEnableIdle**: Event fired when the Idle behaviour starts
    b. **EventAIDisableIdle**: Event fired when the Idle behaviour Ends
2. ***Patrol Behaviour***
    a. **EventAIEnablePatrol**: Event fired when the Patrol behaviour starts
    b. **EventAIDisablePatrol**: Event fired when the Patrol behaviour Ends
3. ***Chase Behaviour***
    a. **EventAIEnableChase**: Event fired when the Chase behaviour starts
    b. **EventAIDisableChase**: Event fired when the Chase behaviour Ends
4. ***Attack Behaviour***
    a. **EventAIEnableAttack**: Event fired when the Attack behaviour starts
    b. **EventAIDisableAttack**: Event fired when the Attack behaviour Ends

Behaviour Example

```
using UnityEngine;
using UnityEngine.AI;
using jcsilva.AISystem;
public class IdleBehaviour : AIBehaviour {

        // Behaviour Settings
        private bool isActive;
        private float idleDuration;
        private float elapsedIdleTimer;

        // References
        private Transform selfTransform;
        private Transform target;
        private NavMeshAgent agent;

        // Vision Settings
        private float maxDistance;
        private float maxFieldOfView;
        private float minDistanceToAttack;

        public IdleBehaviour(MonoBehaviour self, AIStateMachine aIStateMachine) : base(self, aIStateMachine,
"Idle") {
            this.selfTransform = stateMachine.GetSelfPosition();
            this.target = stateMachine.GetTargetPosition();
            this.idleDuration = stateMachine.GetIdleDuration();
            this.maxDistance = stateMachine.GetMaxDistanceToView();
            this.maxFieldOfView = stateMachine.GetMaxFieldOfView();
        }

        public override void OnBehaviourStart() {
            isActive = true;
            stateMachine.EventAIEnableIdle?.Invoke();
        }

        public override void OnBehaviourEnd() {
            isActive = false;
            elapsedIdleTimer = 0f;
            stateMachine.EventAIDisableIdle?.Invoke();
        }

        public override void OnUpdate() {
            if (isActive) {

                // Check if the has Vision of the enemy
                if (AIUtils.HasVisionOfTarget(selfTransform, target, maxDistance, maxFieldOfView)) {
                    Debug.Log("I see the player");
                    stateMachine.HandleState(AIEvents.SeePlayer);
                    return;
                }
                if (elapsedIdleTimer >= idleDuration) {
                    stateMachine.HandleState(AIEvents.NoLongerIdle);
                } else {
                    elapsedIdleTimer += Time.deltaTime;
                }

            }
        }
    }
```

Figure 12 - Behaviour Idle Example

Road Map

In this section we will talk about our future updates and what we intend to do.

1. **Improve Documentation:** This is an ongoing project, so we need to keep track of every change we make, we’re committed to simplify this documentation as improve the explanations whenever necessary.
2. **Video Tutorials**: We’re working on a video tutorial for this asset, we know that some stuff explained in this document would be better shown by video than by words.
3. **Easier Usage:** As we explained we’re working on a way to improve the way you can modify this asset, we’re already scheming a way to make the creation of new behivours far more easier without the need of writing to much code.
4. **UI Improvements**: As the above point we want to create a UI that’s intuitive and can make most of the heavy lifting.
5. **Performance Improvements:** This is our last road map, as this is a new asset, we aren’t sure about heavy loads from it, we hope we can count on your feedback to improve this.