Studio Leaves's Unity 3D Assets Tutorials and Guide

Online Documentation

Email: gaetano@studioleaves.com
Website: www.studioleaves.com
Web Demo

- Cone Of Visibility and Patrolling
- Simple Pathfinder
-

Supported Games:

Tutorials:

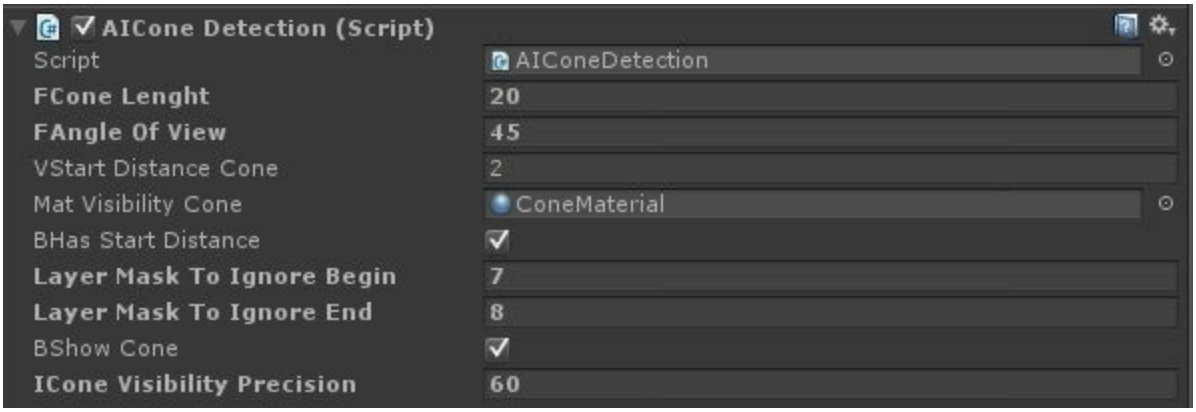
1. Studio Leaves's FOV Cone of Visibilty for Stealth Game (v1.0)

1.1. Cone Of Visibility

To use this plugin you have to Attach ***AIConeDetection*** Script on a GameObject into the scene. All GameObjects that are into Cone are in an ArrayList called **GameObjectIntoCone** so if you want to take this ArrayList you have simply take the ArrayList in this way:

```
MyObject.GetComponent<AIConeDetection>().GameObjectIntoCone
```

Then you can tune up some properties.



1. **FConeLenght**: The Length of the Cone
2. **FAngleOfView**: The Angle of View of the Cone
3. **VStartDistanceCone**: The distance between the starting point of the cone and the main Game Object.
4. **MatVisibilityCone:** The Material of the the Cone
5. **BHasStartDistance**: Boolean: True if the Cone have a start distance, false if not. (*I will delete this with next release*).
6. **LayerMaskToIgnoreBegin**, **LayerMaskToIgnoreEnd**: If you want to ignore some object into the scene you **must** add a layer to you GameObject. All layers are numbered, so if you want to ignore GameObjects with Layer 4 to 6, you have to set up the interval 4 to 6.
7. **BRenderCone**: Show/Hide the cone render: the visibility check will work anyway
8. **IConeVisibilityPrecision**: Adjust the cone visibility precision: more precision will use more resources. Use **10** as default.
9. **(NEW!) bIs2D**: If you check this option the cone of visibility will work on 2D space. If not, the Cone of Visibility will work in 3D Space. If you want a cone that work and rotate on X and Z axes, you should not check this option, if you have a 2D game that use X and Y axes to play, so use this option. If you are using the Static Patrolling script remember to check the option!

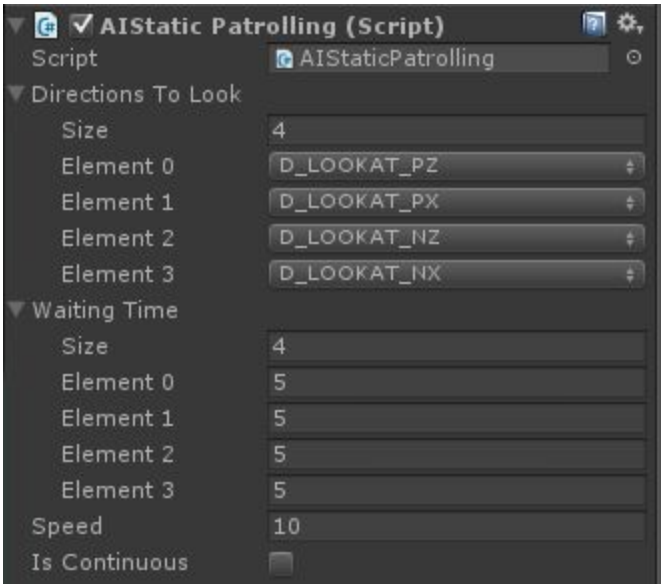
Future Release and Features:

1. Inclination of the Cone
2. Optimization of the Cone, when the cone is out of the camera FOV it doesn't render anything.

1. 2. Static Patrolling

There are two main Script in this package. The first is a *Static Patrolling* the second is *Dynamic Patrolling*. The first script allow your Actors to rotate it self with a list of rotation. The second script allow your Actors to walk through viewpoints placed into the Scene. *Dynamic Patrolling* and *Static Patrolling* combined with ***FOV Cone of Visibility*** allow you to create Enemy that can patrol the scene and catch the Protagonist or you can create Camera as Metal Gear Solid.

The only thing that you have to do is to add `AIStaticPatrolling` to your Actor into scene. The Actor will interpolate the sequence of directions.



1. **DirectionsToLook**: Is an Array with all direction to look, for now the script support only 4 direction:
    a. **D_LOOKAT_PZ**: Look at **+Z** direction.
    b. **D_LOOKAT_PX**: Look at **+X** direction.
    c. **D_LOOKAT_NZ**: Look at **-Z** direction.
    d. **D_LOOKAT_NX**: Look at **-X** direction.
    e. **(NEW!) D_LOOKAT_DOWN**: Look down direction only on 2D!
    f. **(NEW!) D_LOOKAT_UP**: Look up direction only on 2D!
    g. **(NEW!) D_LOOKAT_LEFT**: Look left direction only on 2D!
    h. **(NEW!) D_LOOKAT_RIGHT**: Look right direction only on 2D!.

2. **WaitingTime**: This array define the time that Actor wait before to start the new interpolation. **The DirectionToLook size and the WaitingTime size must be the same!** You have to define a timing for every position.
3. **Speed:** The speed to reach one point to another.

You can also take another properties from the script:

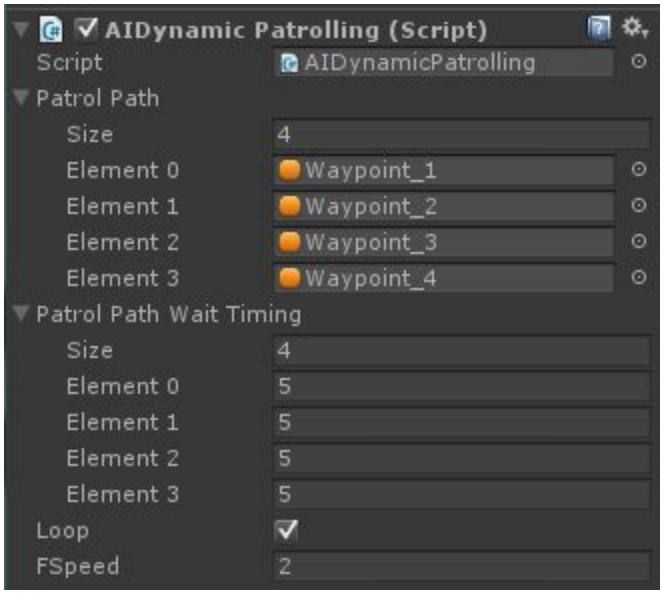
1. `bool ActivePatrolling` : this allow you to active of unactive the Patrolling.
2. `Vector3 DirectionToLook` : Return the direction that the Actor actually is looking.
3. `PATROLLINGDIRECTION DirectionIDToLook` : Return the direction that the Actor is looking.
4. `STATICPATROLLINGSTATE PatrollingState` : Return the current state of Actor, `STATICPATROLLINGSTATE.PS_ROTATING`, `STATICPATROLLINGSTATE.PS_WAITING` .

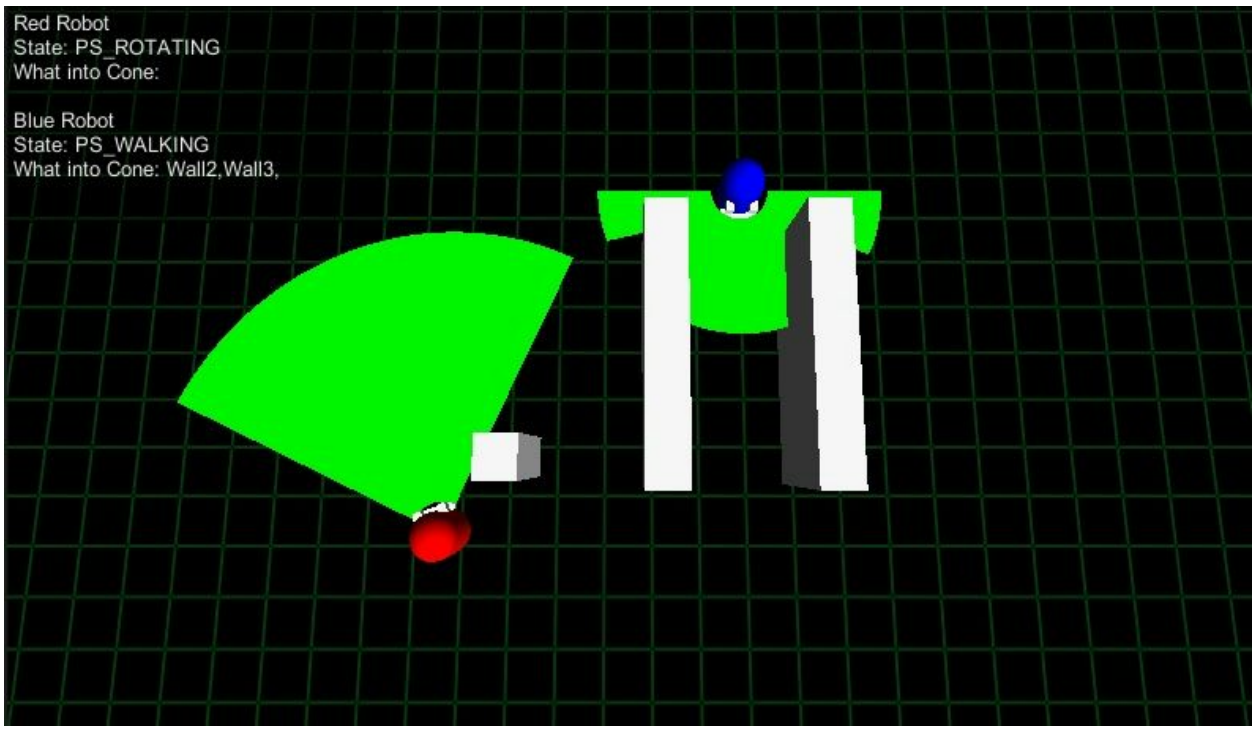
You can access to script by:

```
MyObject.GetComponent<AIStaticPatrolling >()
```

1. 3. Dynamic Patrolling:

The only thing that you have to do is to add `AIDynamicPatrolling` to your Actor.



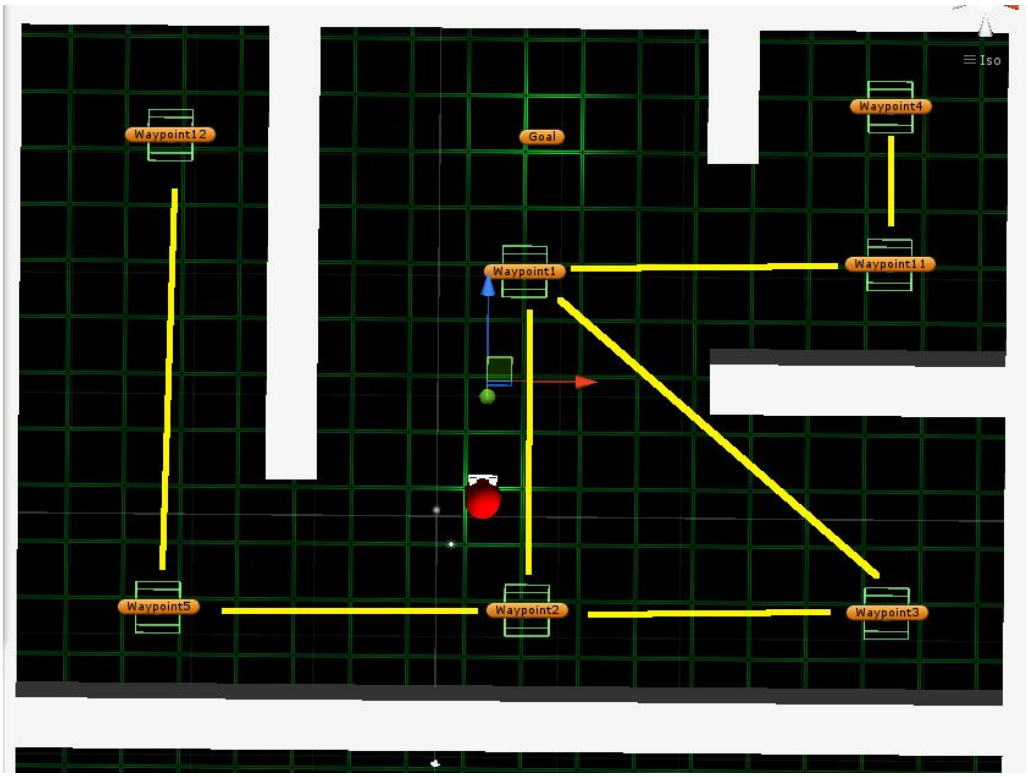


Future Release and Features:

2. Studio Leaves's PathFinder

The Pathfinder will help your bots to walk through the best path. We reccomend to use a flat groudn for your scene. We will support multilevel scene soon. To use this pathfinder you must follow the these simple step:

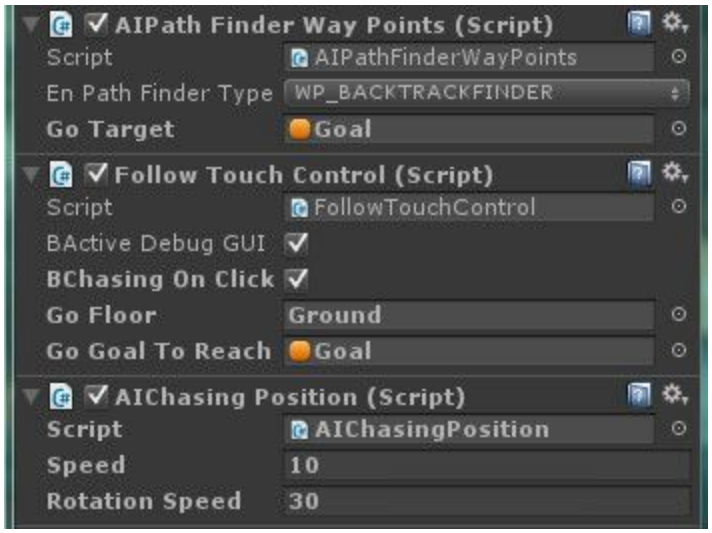
Insert Waypoints in you Scene: The first thing you have to do is to put the Waypoint into the scene, this allow your bot to find the best path to reach the **Goal**. All Waypoints **must be connect to at least another Waypoint**. Use the **Prefabs** called "**Waypoint**" it will be **tagged as "waypoint"**. You can create your own waypoint, but remember to tag the GameObject as "waypoint" and to add a Collider.



Now that you have put your waypoints in your scene, we reccomend to lift the waypoints, so they will not intersacate with the Plane.

The last thing we have to do is to place the “Goal”. The “Goal” is the final point where your bot will reach. Now, find the **Prefabs “Goal”** and put it into the scene. Your scene is ready to work!

Setup your bot: To try a bot as Prefabs, you can try to use the “**EnemyPathfinder**” prefabs into package. Now i will explain all the Script that will help you to create a perfect walking bot!



AIPathFinderWayPoints: This is the main script to take the Path from the attached **GameObject** to **Goal** through the waypoints. You must use the method:

```
MyGameObject.GetComponent<AIPathFinderWayPoints>().GetPathToFollow( Vector3
iStartPosition );
```

This method return an ArrayList of Waypoints from the position of the attached GameObject to Goal. So you have to translate your GameObject through this waypoints.

- `EnPathFinderType` : For now ignore this variable, this is used to allow tyou to choose other Pathfinder Algorithm, but for now, BackTracking Algorithm is the only available.
- `GoTarget` : Is the Goal that your bot will Reach.

I will explain now two Utility Script that help you to use the Pathfinder.

AIChasingPosition: This an example script that allow the GameObject to follow a Path of Waypoints. The only thing that you have to attach the script to the bot and call this method:

```
MyGameObject.GetComponent<AIChasingPosition>().StartChase( ArrayList
iPathToFollow );
```

The bot will start to chase the Path looking forward the next position to reach.

- `Speed`: Is the Speed of walking.
- `RotationSpeed`: Is the Rotation speed when the bot look at new position to reach.

Another Utility Script is **AIFollowTouchControl**, this script will help you to find the Path on the Ground with a simple click. **You have to put in your scene a GameObject with a collider to simulate the Ground**, so attach this script to you Bot. Now your bot is ready to follow a path when you click on the ground.

- `BActiveDebugGUI`: Enabled if you want to see the Path founded on the the screen.
- `BChasingOnClick`: If Enabled, the bot will follow the path after you have clicked on the Floor. If Disabled, the bot will follow the path everytime you move the mouse on the floor.
- `BActiveDebugGUI`: Enabled if you want to see the Path founded on the the screen.
- `GoFloor`: Put your GameObject Floor here.
- `GoGoalToReach`: Put your GameObject Goal here.