RAYCAST_PRO V1.0.5

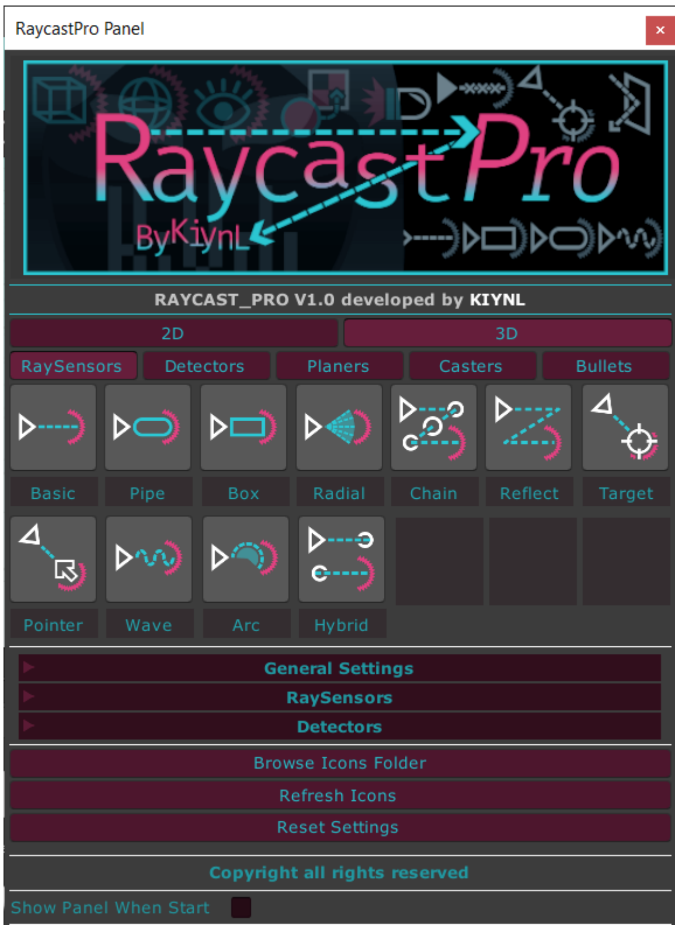
KIYN_L

GENERAL:

Dear customer, with thanking to you for your purchase this asset, we hope that the Raycast Pro will improving your professional workflow and accuracy of projects. In the introduction below, you will get to know the function of different parts of the tool. Also, if you are satisfied, suggest, criticize and report any problem, please refer to Kian.tianxuan@gmail.com.

This tool consists of different cores, each of which works in a complementary or independent way. This system was chosen because it increases the extensibility of the tool and allows special features to reach all subsets.

To see the main menu of the plugin after installation, you can go in ***Tools*** > ***RaycastPro_Panel***. In the following menu, you can easily access most of the components and fast setup them to the Scene.

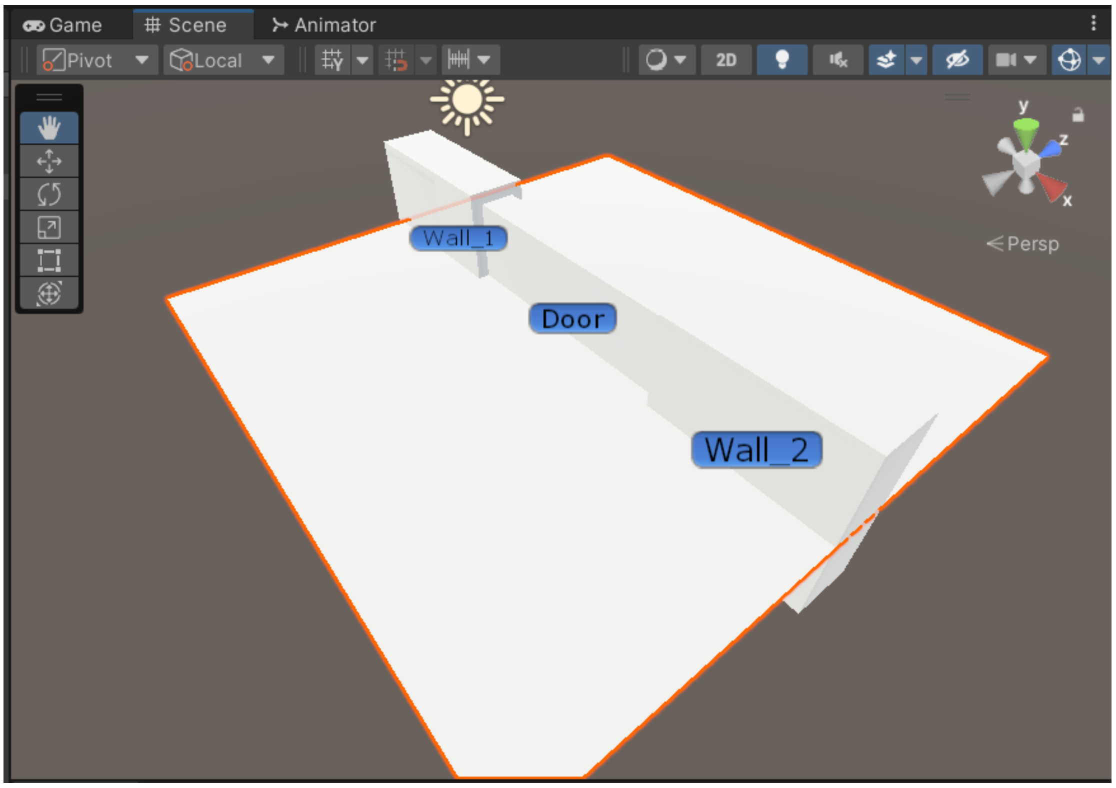


CONTENTS

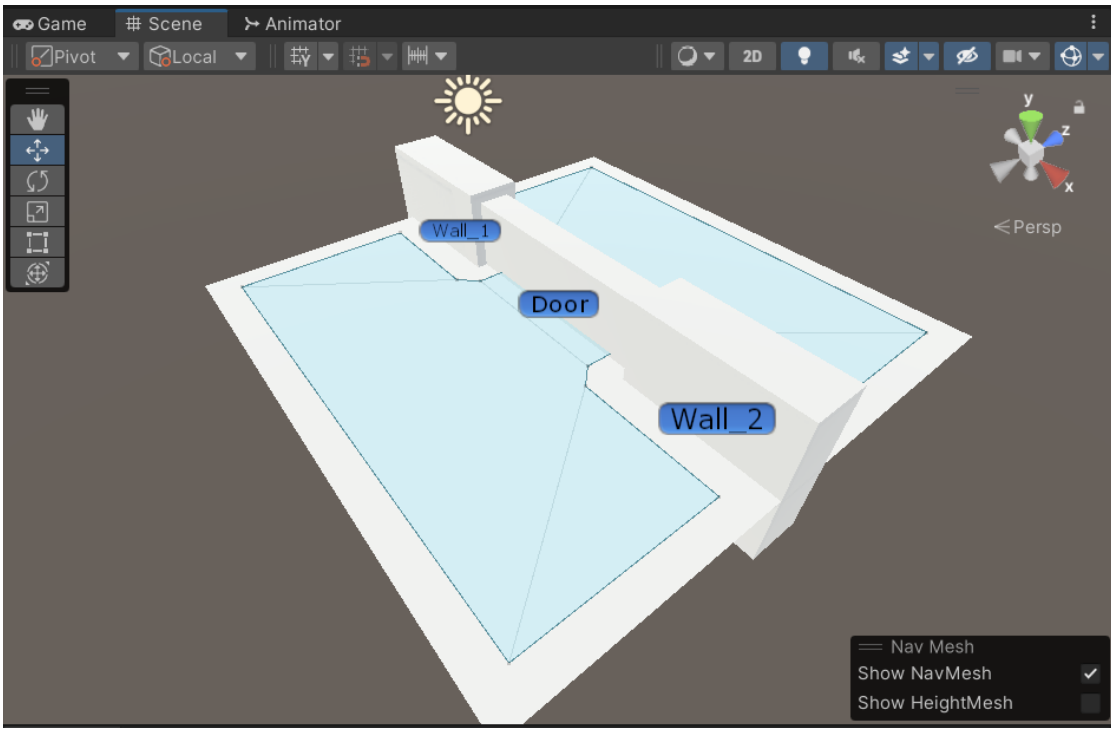
HOW TO SETUP

In the following tutorial, we will learn how to create a simple control with the help of a Pointer Ray and Pipe Ray. It is necessary to have some navigation knowledge before starting the tutorial.

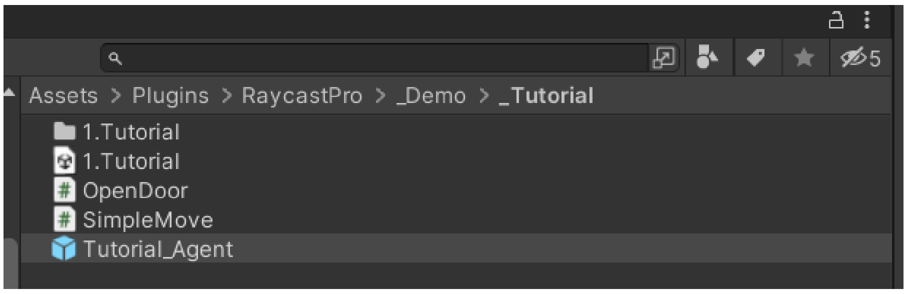
1. First step, make a scene like below, and make the floor and walls 1 and 2 static.



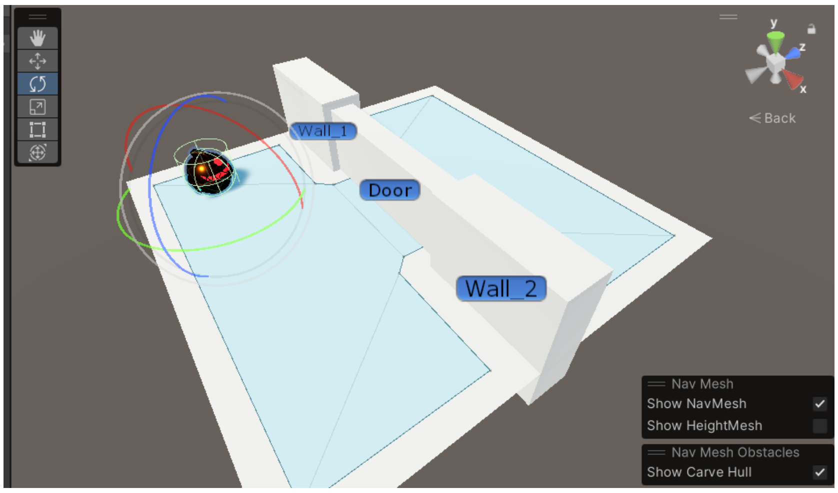
2. After finishing the work, open the menu Windows => AI => Navigation and from Bake Panel, press Bake Button.



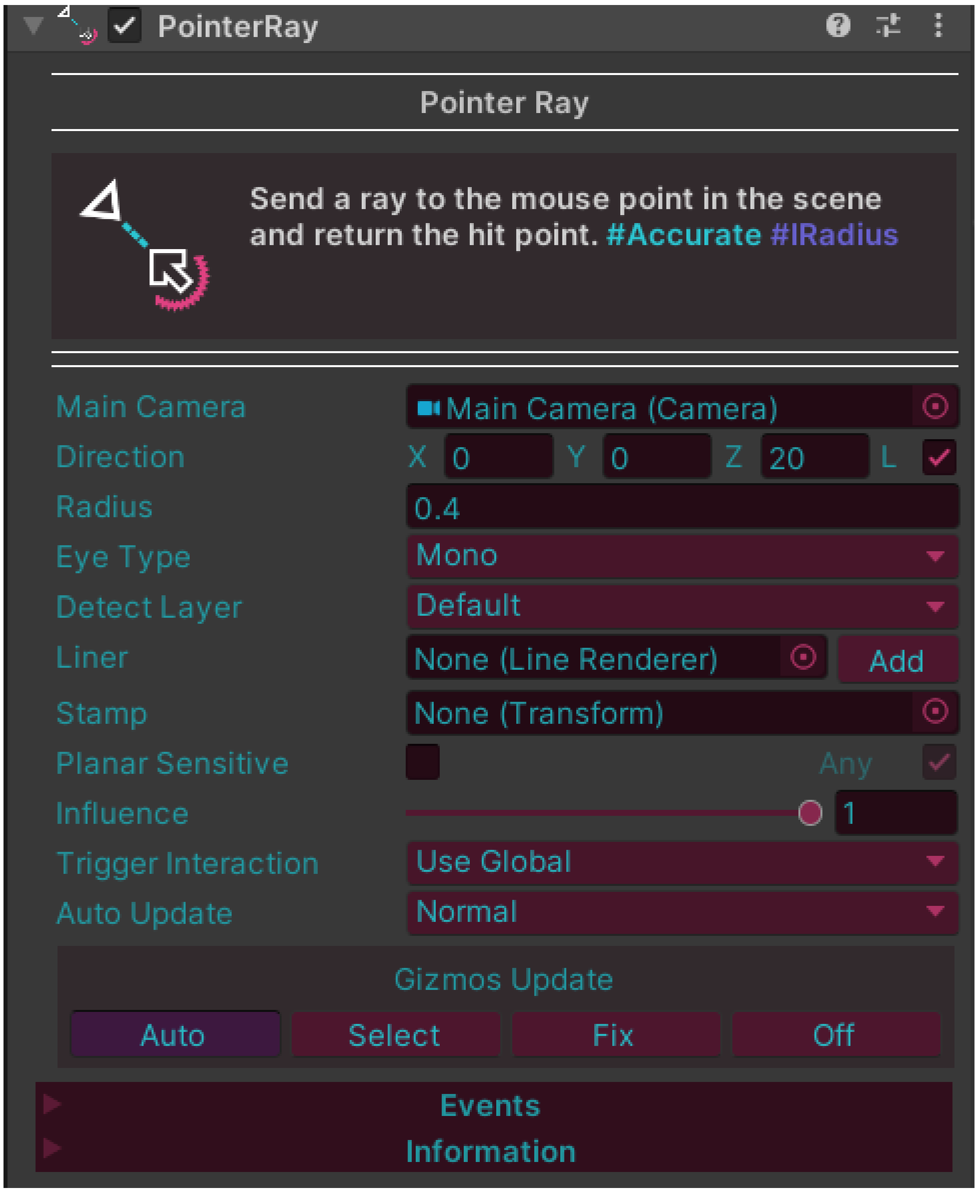
3. After doing this, you can take the following prefab from the Tutorial folder and place it on the plane. This prefab includes navmesh agent.


Assets > Plugins > RaycastPro > _Demo > _Tutorial

- 1.Tutorial
- 1.Tutorial
- OpenDoor
- SimpleMove
- Tutorial_Agent



4. To write a simple click mouse control, just select the PointerRay component and add it to the camera. Then increase the length of Ray in the Z direction as below. Now, by pressing the Play key in Editor, you can see the Pointer Ray gizmo on the screen.

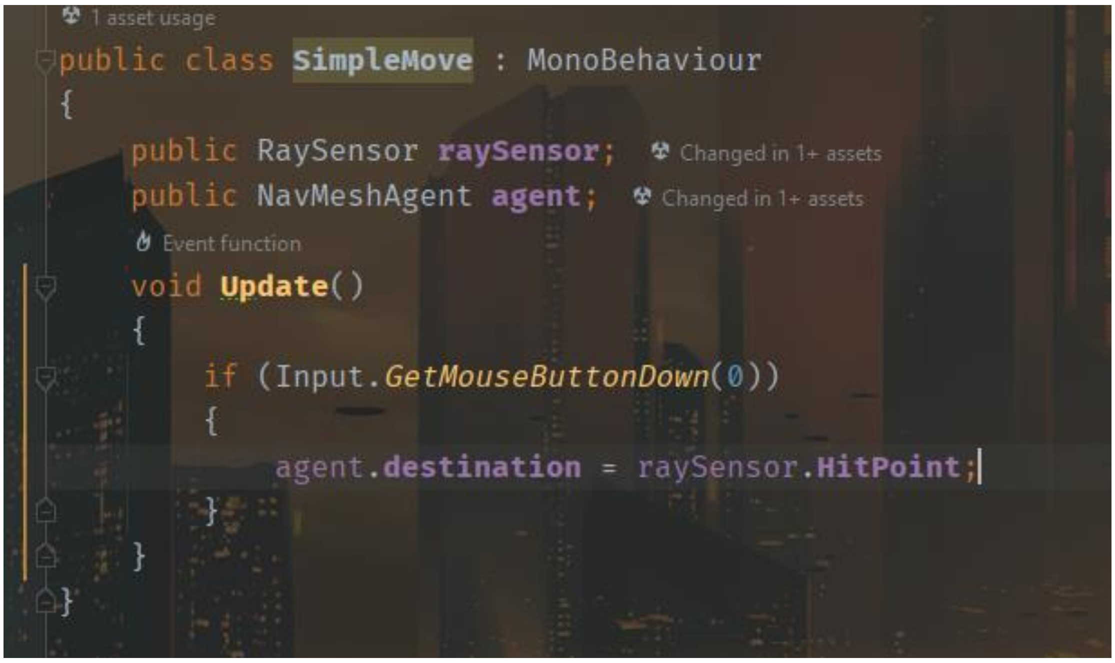


Send a ray to the mouse point in the scene and return the hit point. #Accurate #IRadius

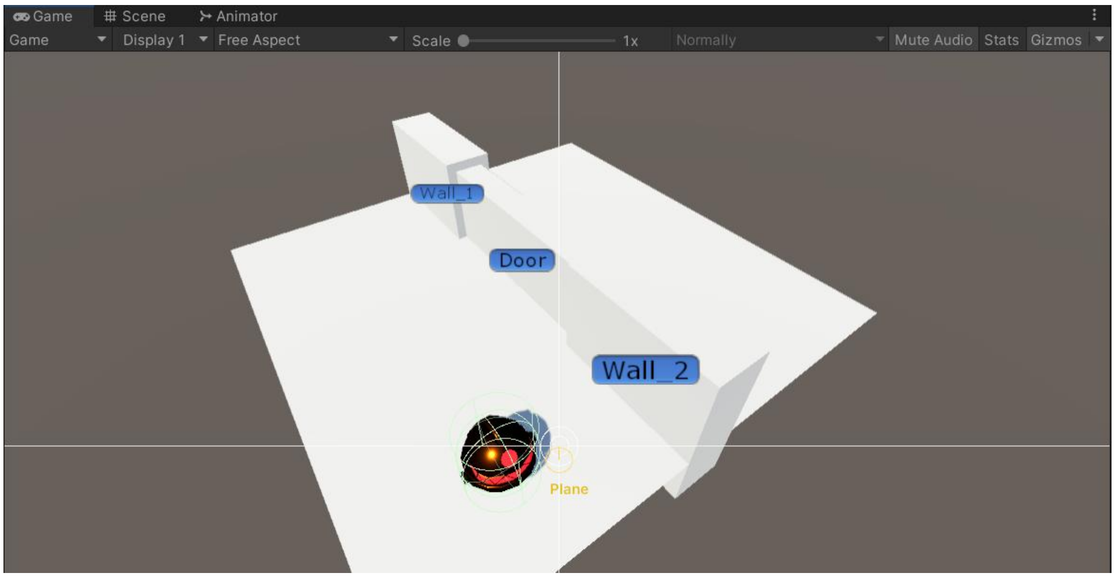
Main Camera	Main Camera (Camera)
Direction	X 0 Y 0 Z 20 L ✓
Radius	0.4
Eye Type	Mono
Detect Layer	Default
Liner	None (Line Renderer) Add
Stamp	None (Transform)
Planar Sensitive	Any ✓
Influence	1
Trigger Interaction	Use Global
Auto Update	Normal

5. Now it is enough to move the agent with a few lines. Create a script like below and connect it to the agent body. Enter the agent and pointer ray in

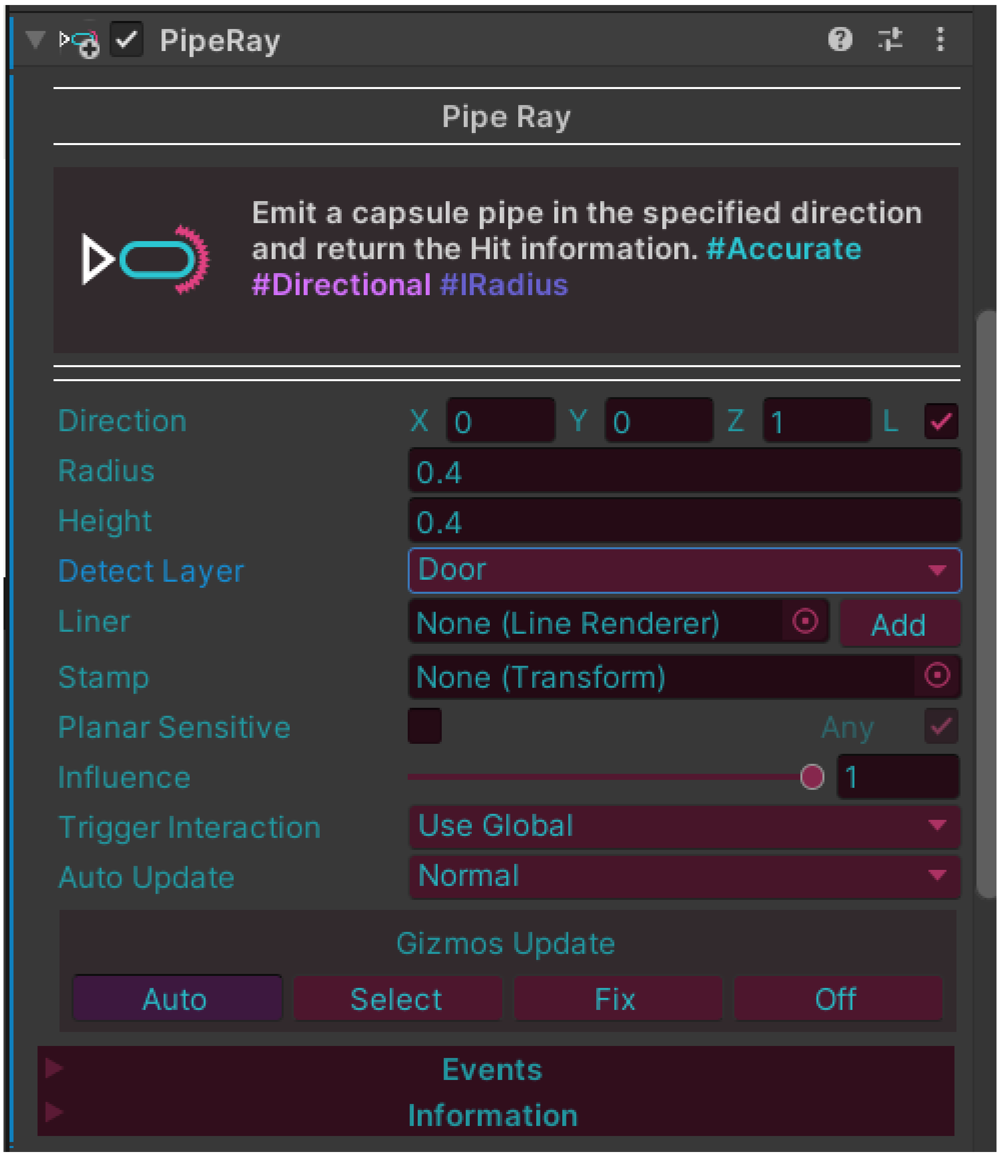
inspector and you will see agent moving by clicking on the plane.



```
1 asset usage
public class SimpleMove : MonoBehaviour
{
    public RaySensor raySensor;   Changed in 1+ assets
    public NavMeshAgent agent;   Changed in 1+ assets
    Event function
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            agent.destination = raySensor.HitPoint;
        }
    }
}
```



6. Now, in the next step, to open the door, you need to add a pipe ray to the Agent as shown below. Set the dimensions as below and create a layer called Door and set it to Ray.



PipeRay

Pipe Ray

Emit a capsule pipe in the specified direction and return the Hit information. #Accurate #Directional #IRadius

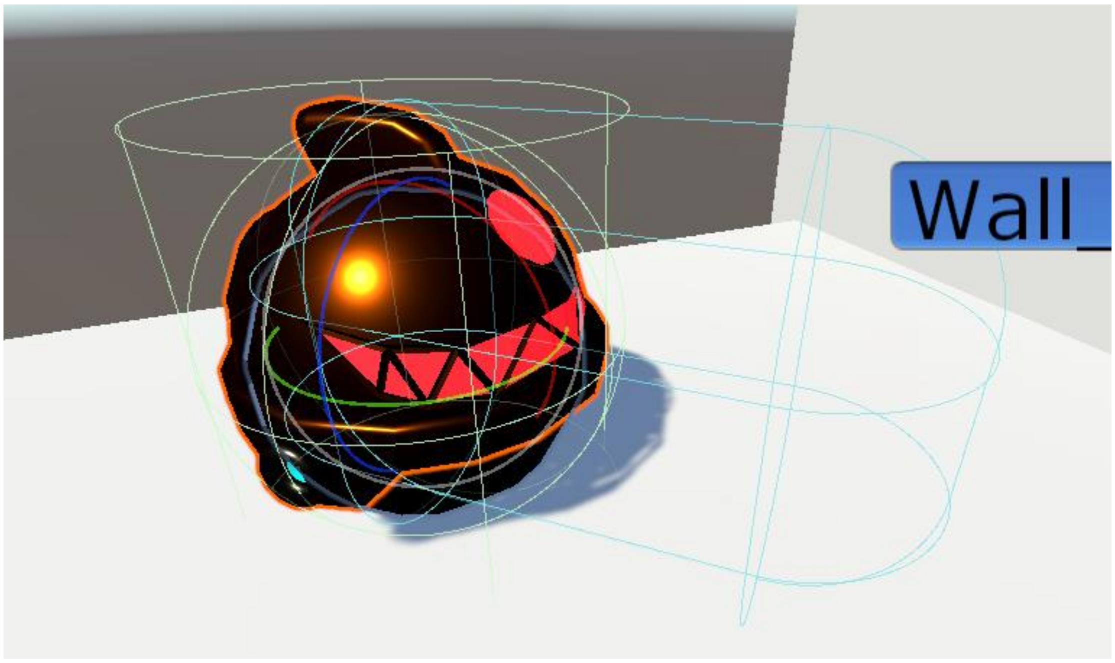
Direction	X 0 Y 0 Z 1 L ✓
Radius	0.4
Height	0.4
Detect Layer	Door
Liner	None (Line Renderer) Add
Stamp	None (Transform)
Planar Sensitive	☐ Any ✓
Influence	1
Trigger Interaction	Use Global
Auto Update	Normal

Gizmos Update

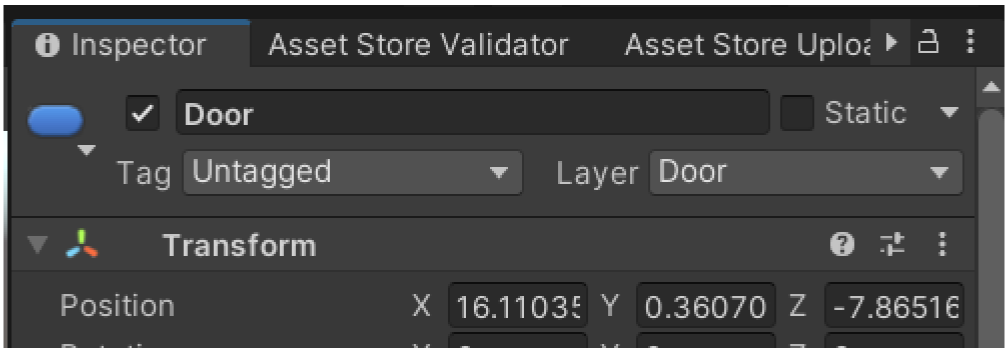
Auto Select Fix Off

Events

Information



7. Now go to the Door object as shown below, set its layer to Door and also add the Nav Mesh Obstacle component to it.

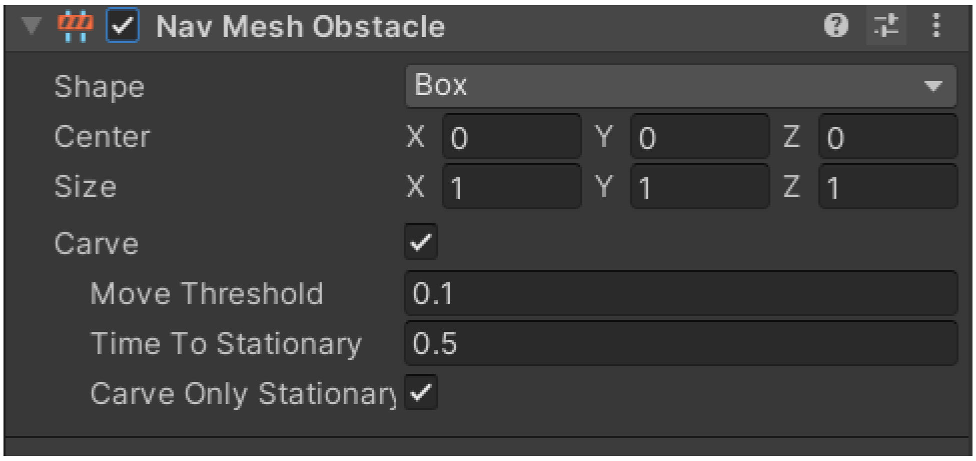


Door

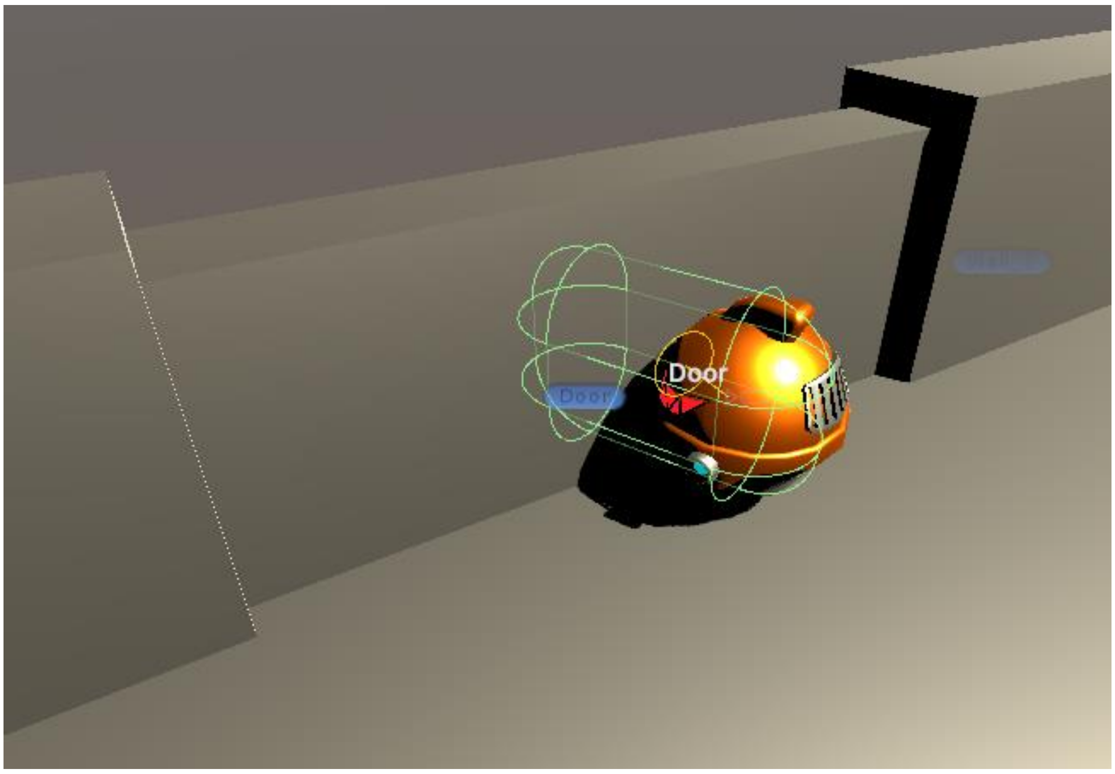
Tag Untagged

Layer Door

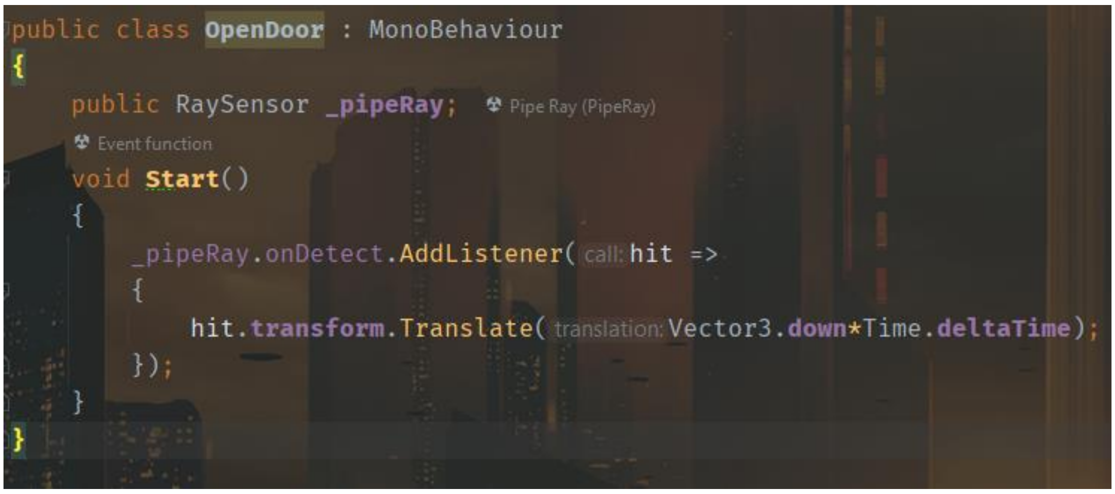
Position X 16.11035 Y 0.36070 Z -7.86516



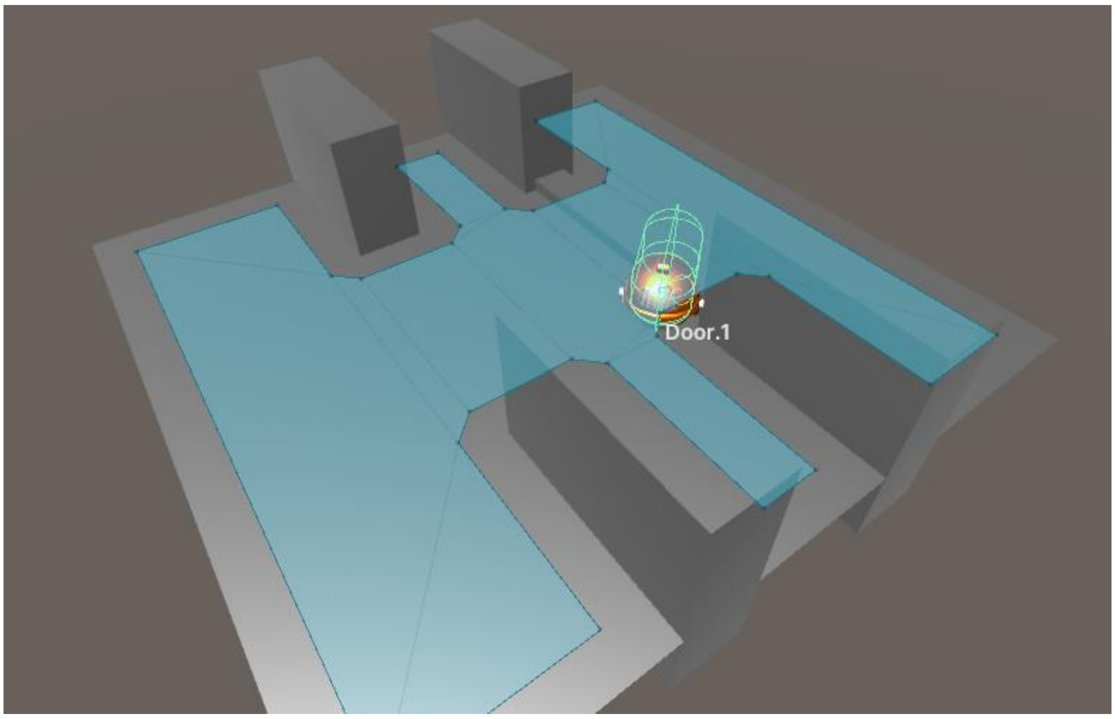
Shape	Box					
Center	X	0	Y	0	Z	0
Size	X	1	Y	1	Z	1
Carve	✓					
Move Threshold	0.1					
Time To Stationary	0.5					
Carve Only Stationary	✓					



8. **Give the following code to the Agent and just enter the pipe ray in the inspector. After that, every object that is in the Door layer goes to the ground.**



```
public class OpenDoor : MonoBehaviour
{
    public RaySensor _pipeRay;   Pipe Ray (PipeRay)
    Event function
    void Start()
    {
        _pipeRay.onDetect.AddListener( call: hit =>
        {
            hit.transform.Translate( translation: Vector3.down*Time.deltaTime);
        });
    }
}
```

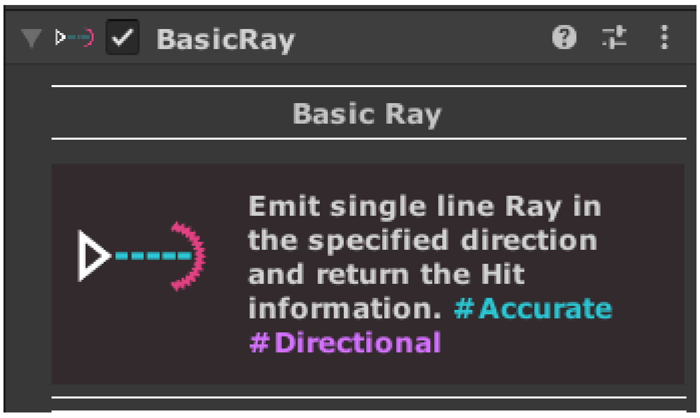


DESCRIPTION

After completing the tutorial above, you may be familiar with Ray's functionality. Now, the following explanations are for getting to know the more advanced and extensive features of this plugin.

TAGS:

Currently, there are about 70 components in this plug-in, whose brief description is provided in the component's header. In addition, the following hashtags will help you to learn more about its function and the ambiguities of its use.



Emit single line Ray in the specified direction and return the Hit information. #Accurate #Directional

- #Accurate: This tag means full accuracy of ray in matching between gizmo and physics calculation. (rays without this tag can working up to 99% of match.)
- #Directional: This tag means ray support in directional mode. Some rays without this tag don’t need it logically.
- #Pathray: These types of rays can cast in a chain of vectors and support some special **RCPro** features.
- #CDetector: It is completely Collider Detector. The focus of this detector is on finding colliders.
- #RDetector: It is completely Raycast Detector. The focus of this detector is on finding Raycast hits.
- #LOS_Solver: This tag shows that the desired detector covers Los directly and you don't need to implement it manually.
- #Recursive: This tag is for rays that have used recursive functions to determine the path.
- #Virtual: This tag shows that the component is virtual and cannot be adjusted in the editor.
- #Dependent: This tag shows that the component needs other tools from the same set to be used.

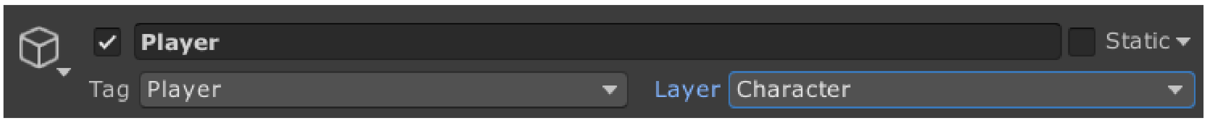
- **#IRadius:** This tag indicates that ray uses IRadius interface, which usually supports radius.
- **#IPulse:** This tag, which belongs to detectors, covers a variable called **pulse**, which creates a gap between the processing time and will greatly help optimization.
- **#INonAllocator:** This tag indicates that the desired Detector supports NonAllocator detection. This reduces the possibility of generating garbage, but because the detectors also perform other filtering, it does not necessarily mean that they become zero.
- **#Scalable:** For components whose detection dimensions change by changing the transform's scale.
- **#Rotatable:** For components whose direction is also changed by changing the transform's rotation.
- **#Experimental:** Experimental tools have more limited functionality and may cause crashes. Be careful while using them.

GENERAL PARAMETERS:

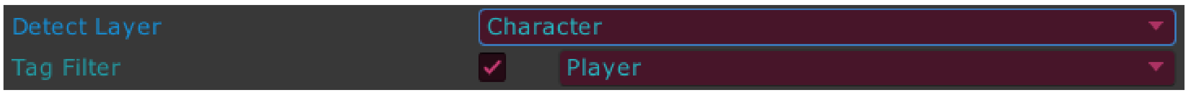
The following variables exist in most components and are effective for use in all tools.

DETECT LAYER:

The detection layer is the most important parameter of all the components with which you can identify and separate the detectable colliders. Specifying layers is accessible in the header of each GameObject.


✓ Player Static
Tag Player Layer Character

In **Detectors**, **also Tag filter** has been added for more control.


Detect Layer Character
Tag Filter ✓ Player

INFLUENCE:

Influence parameter is used for ease of coding. From where it is considered, your ray works together with other scripts. The influence value can be used as a coefficient. For example: ***HP -= influence * damage***

TRIGGER ITERACTION:

Specifies how ray collides with ***IsTrigger*** colliders. ***Use Global*** means to use project settings and ***Collide*** means to accept ***IsTrigger*** for hitting.

AUTO UPDATE:

This parameter determines how the ray will be updated, keep in mind that the ***Manuel*** option by disabling the permanent update may limit Gizmo processing in the Scene.

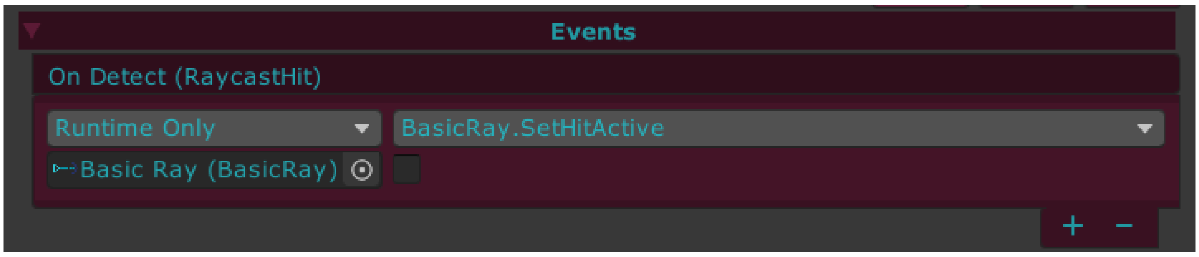
GIZMOS:

This option specifies how the gizmo should be updated. The **Select** option works only when object is selected, and **Fix** will permanently active. It is recommended to set it to **Off** when you are sure It works to increase scene performance.

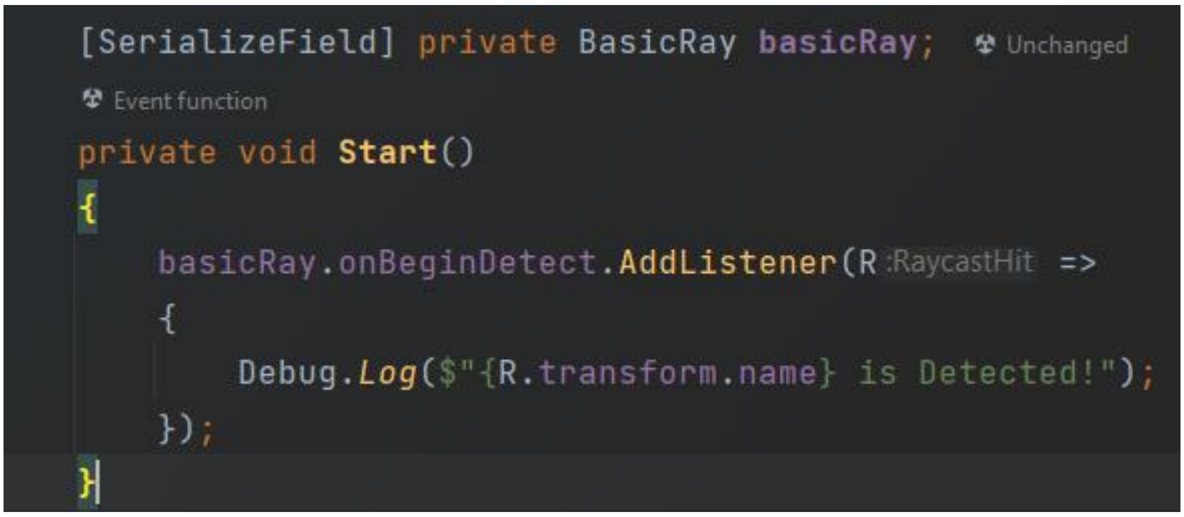
Auto: Automatic mode shows the gizmo when detecting in-game and it works in Scene as well as during Select.

EVENTS:

Events help you to have detection and non-detection events and to be able to control scripts through it. Keep in mind that some functions are placed in the body of the component itself for ease of coding.


Events
On Detect (RaycastHit)
Runtime Only ▾ | BasicRay.SetHitActive ▾
Basic Ray (BasicRay)

This code shows you how to submit orders to events from Script.



```
[SerializeField] private BasicRay basicRay;  Unchanged
 Event function
private void Start()
{
    basicRay.onBeginDetect.AddListener(R :RaycastHit  =>
    {
        Debug.Log($"{R.transform.name} is Detected!");
    });
}
```

RAYSENSORS:

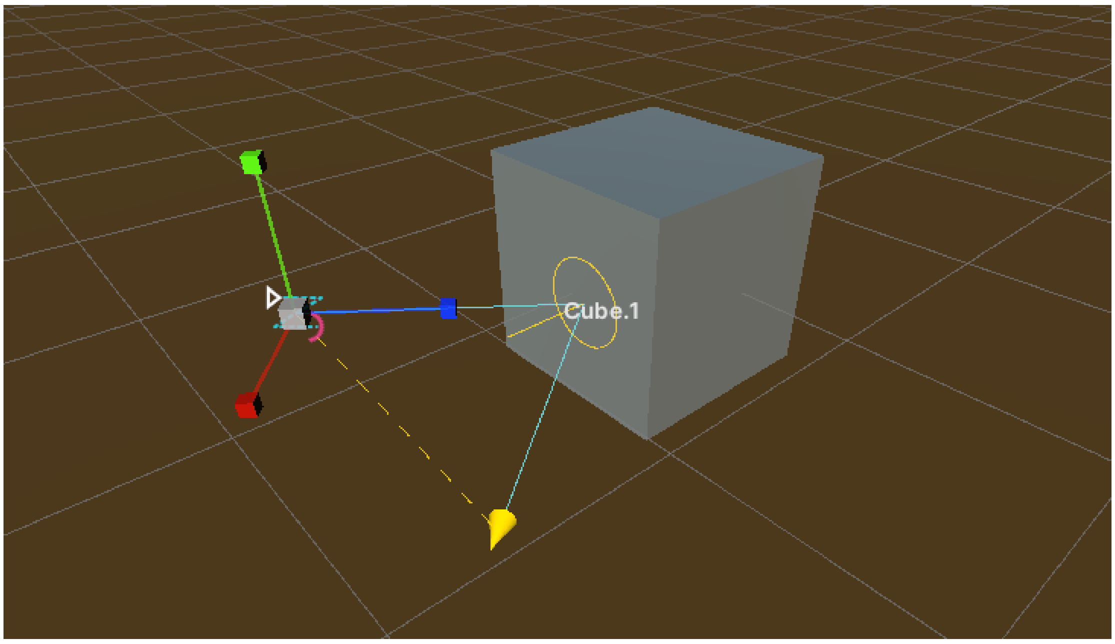
In general, RaySensors return an output of type Raycast Hit. They can cover the ray path by a ***LineRenderer*** and leave a ***stamp*** at the hit point.

LINER:

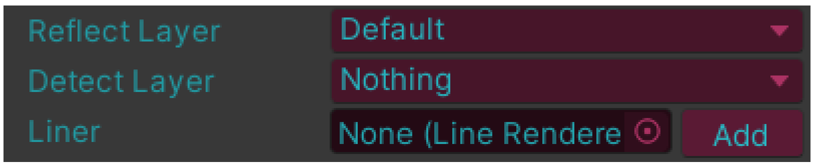
1. Although adding a Liner is very simple and doesn't really require a long explanation, in this example I will teach a Reflect Projection. Just add a ReflectRay from the RaycastPro panel.

2. Bend the angle a bit, increase length by adjust Z, so that it touches a Cube, then change the **Reflect layer** to

Default and **Detect Layer** to Nothing so that the Ray only reflects itself and you will see something like below.

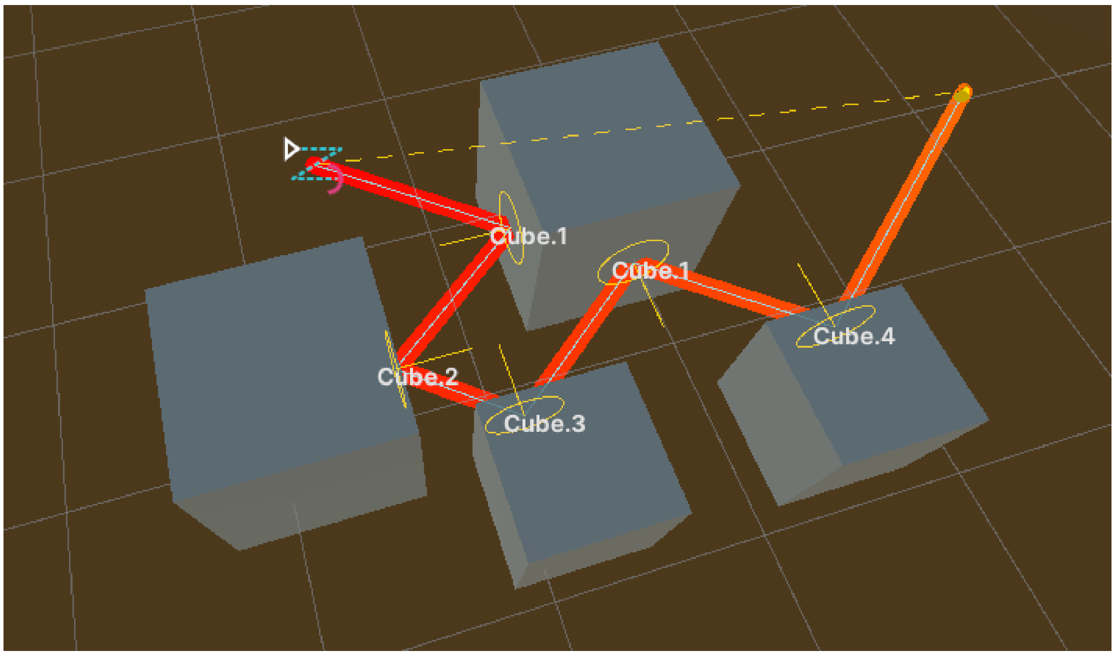


3. Click Add next to the Liner to automatically add it to the Ray. Now is the only need. To adjust the color and dimensions of the projection.

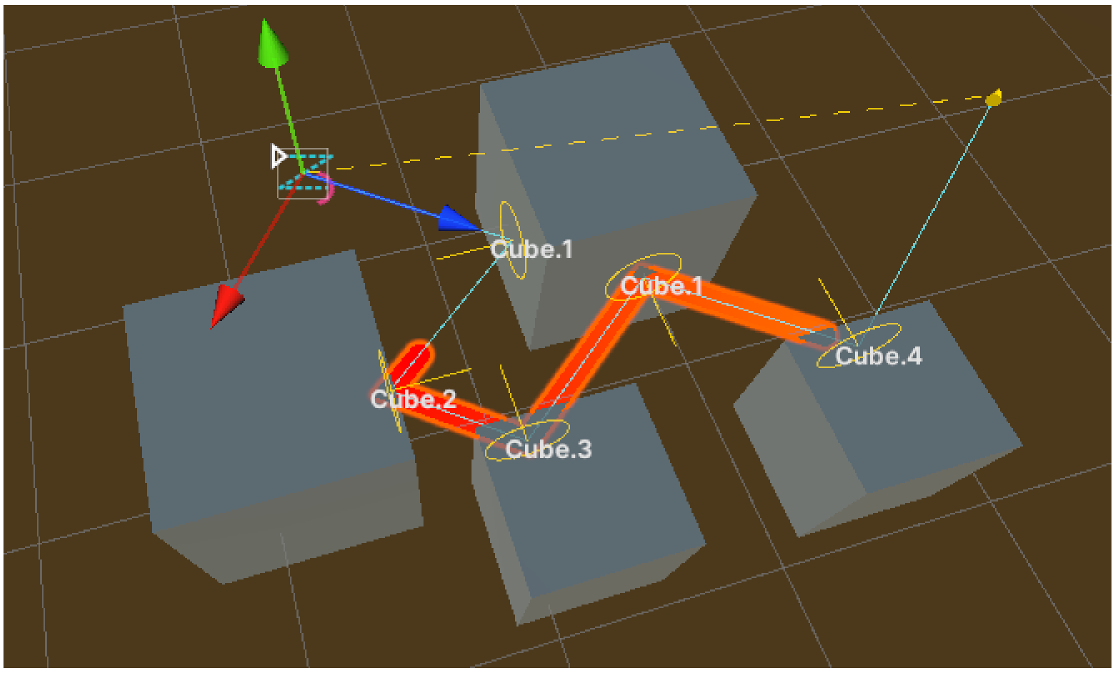


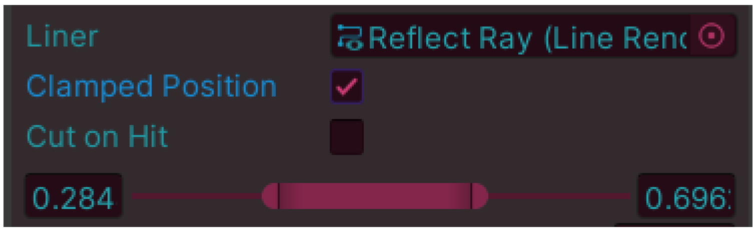
Reflect Layer	Default	
Detect Layer	Nothing	
Liner	None (Line Rendere	Add

4. By placing the Gizmo in **Fix** mode, you can see and adjust the projection in real time in the Editor. There is no problem by changing parameters and adding Cube.



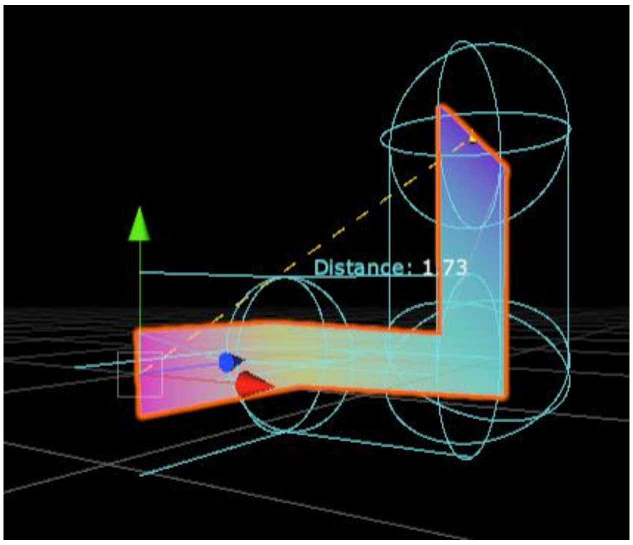
5. And By changing the Clamped Position values, the LineRenderer will be cut in its area.

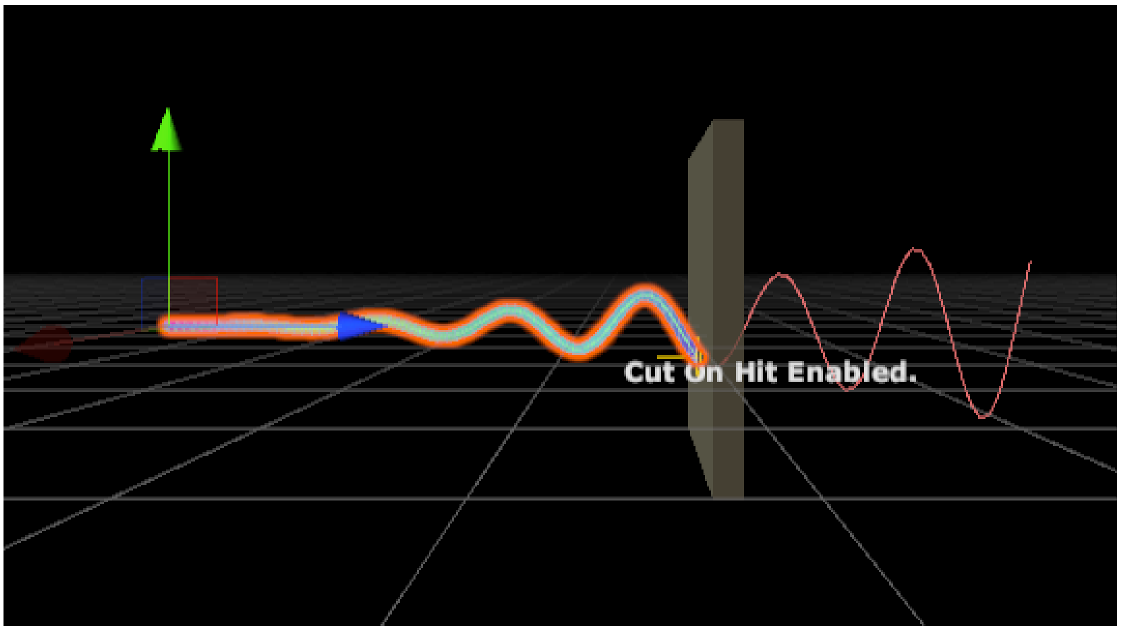




Liner — Reflect Ray (Line Renc
Clamped Position ✓
Cut on Hit
0.284 — 0.696

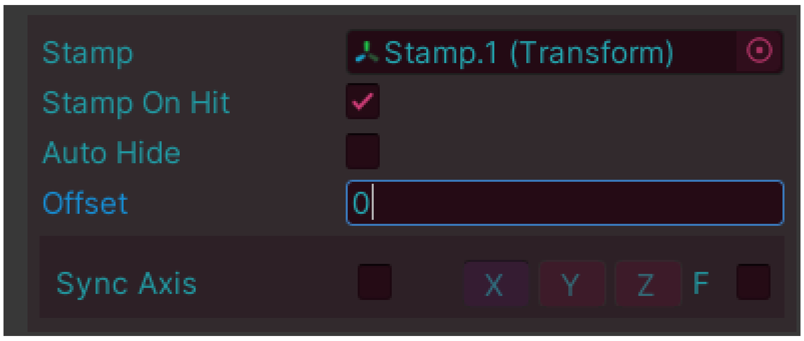
Cut on Hit: It will be cut off when hitting the collider.



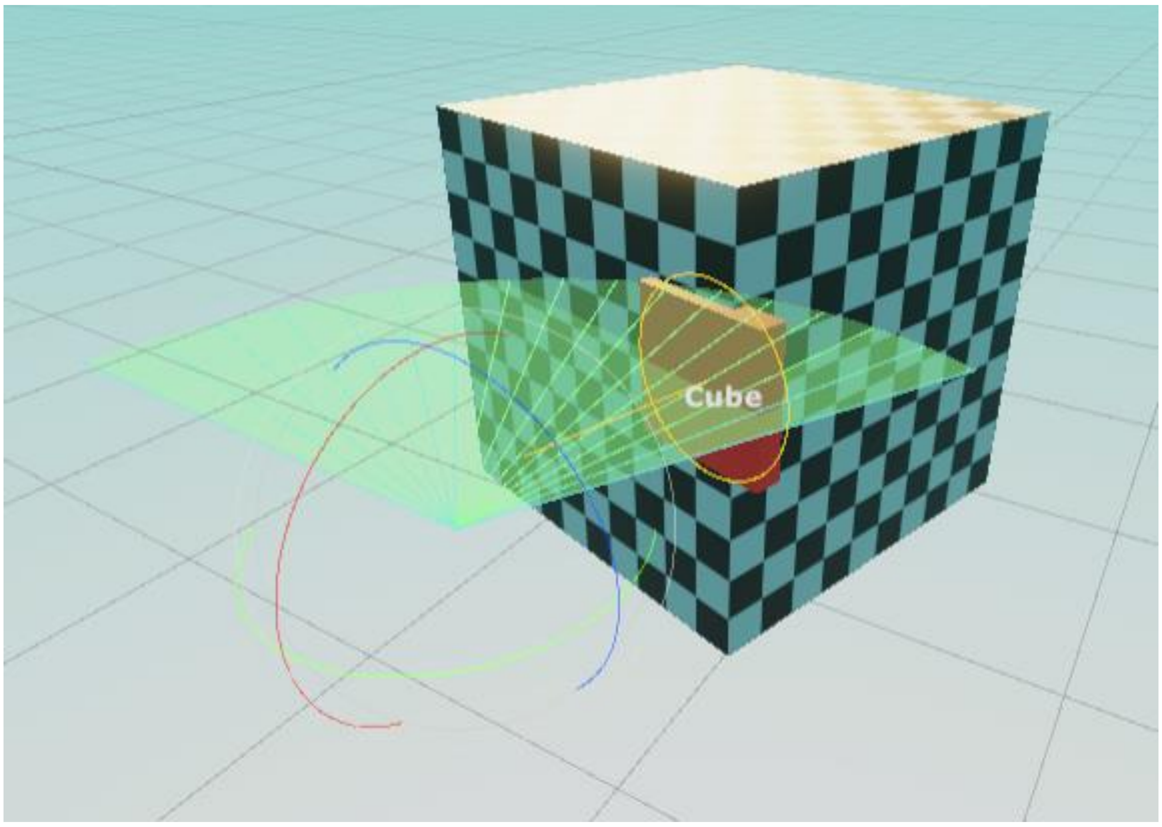


STAMP:

In the stamp settings, ***Stamp on Hit*** moves its location to the Hit Point when it hits, and it is placed on the Tip otherwise. At the bottom of the ***Auto Hide*** option, it will display the stamp only when encountered. Other options are related to specifying offset and direction.



Stamp: Stamp.1 (Transform)
Stamp On Hit: ✓
Auto Hide:
Offset: 0
Sync Axis: X Y Z F

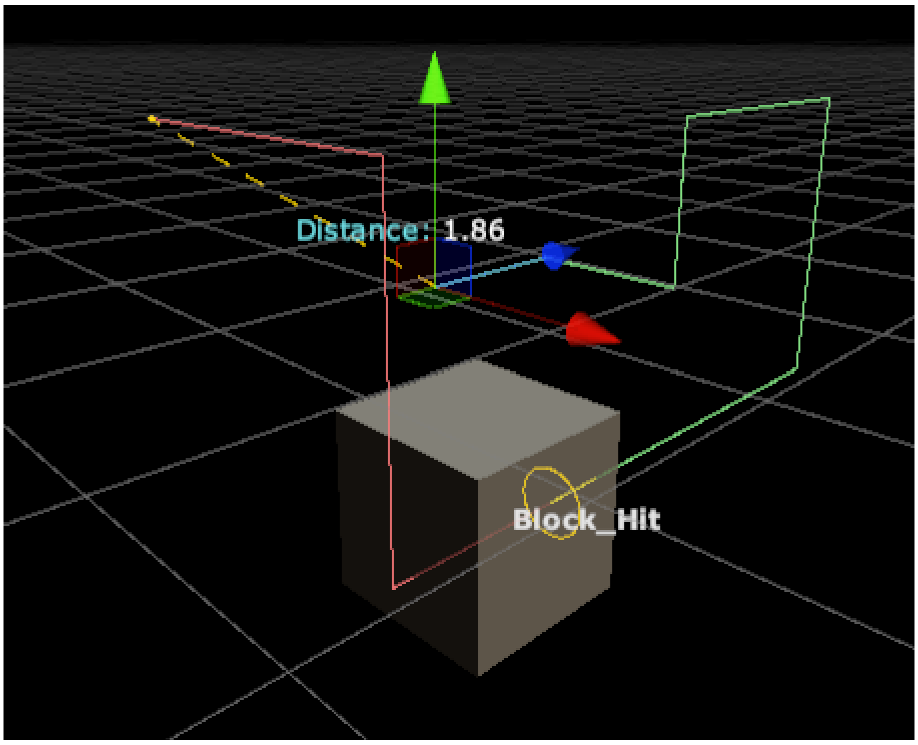


Important Note: If your stamp layer is the same as the ray detection layer, you may see a crazy behavior in scene, This is because Ray tries to target the stamp itself. Make sure that the stamp layer is not the same as the **Detect Layer**.

PATH RAY:

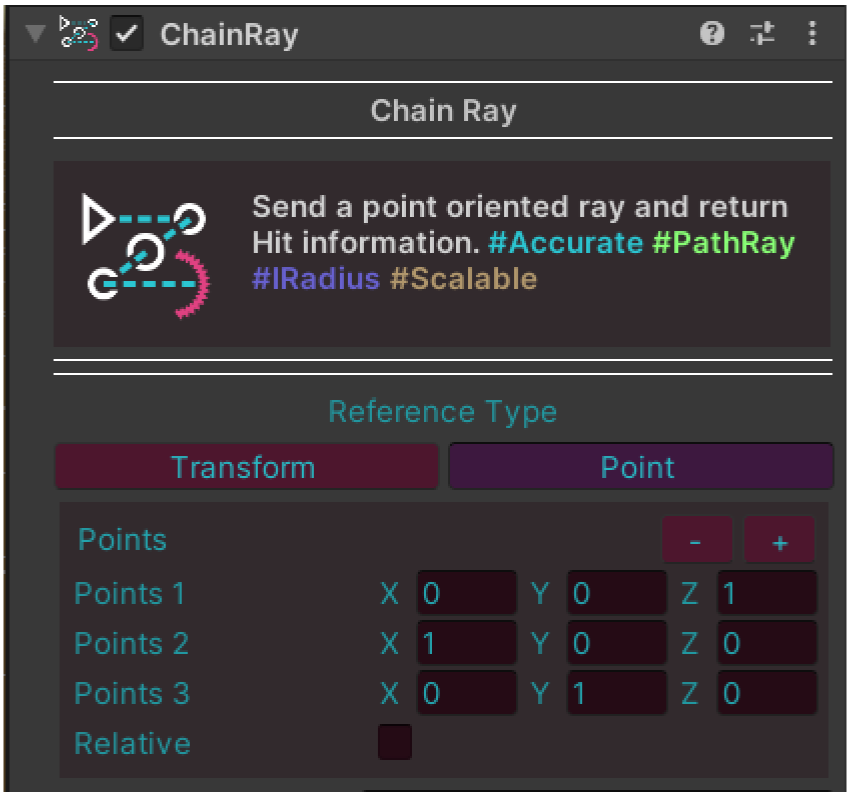
Basically, this category of Rays includes a list of points called **Path Points** and **Detect Index.** These points are calculated in different Path Rays and their system is like this, they send direct rays from one point to the next point in a chain, and the first detection point interrupts the process.

In the example below, you can see a **ChainRay:**



PATH RAYS LIST:

1. ChainRay: This Ray normally receives the points and releases the Ray in its path. In Transform mode you will be able to easily animate points.



Send a point oriented ray and return Hit information. #Accurate #PathRay #IRadius #Scalable

Reference Type

Transform | Point

Points - +

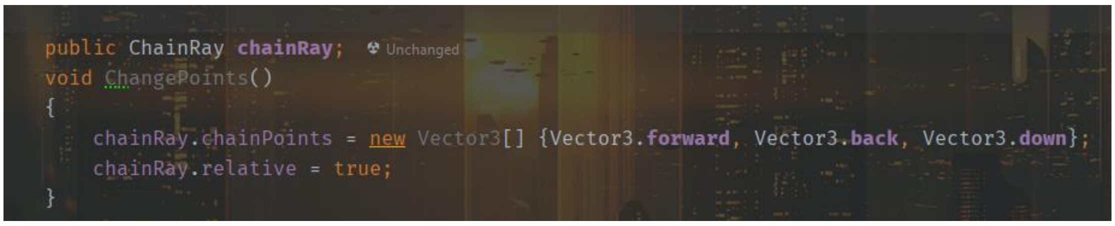
Points 1 X 0 Y 0 Z 1

Points 2 X 1 Y 0 Z 0

Points 3 X 0 Y 1 Z 0

Relative ☐

Also, access to change points in the code is also open and easy.



```
public ChainRay chainRay;  Unchanged
void ChangePoints()
{
    chainRay.chainPoints = new Vector3[] {Vector3.forward, Vector3.back, Vector3.down};
    chainRay.relative = true;
}
```

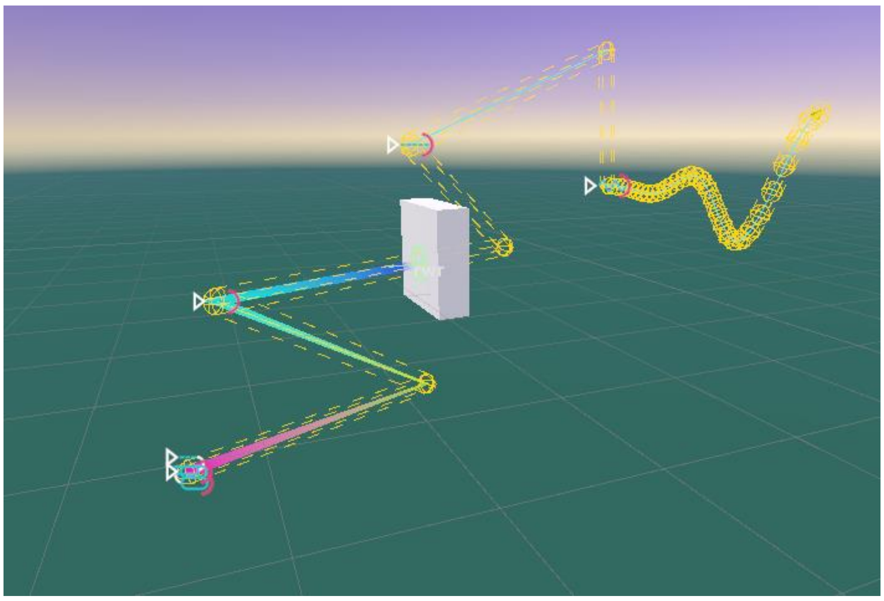
2. ReflectRay: This Ray, like other Rays, can process collisions in the Reflection Editor itself, such as the impact of a billiard ball, and the collision points will change dynamically. The important part is to place the Reflect layer.

3. WaveRay: This Ray is an algorithmic, mathematical processor that can help you in randomization or accurate processing of sinusoidal functions, although it is natural that it consumes more performance than other Rays.

4. ArcRay: This Ray is a Velocity-based processor that includes local and global states both in the initial and general angle, so that you can use it in Trajectory calculation.

5. HybricRay: And finally, if you need to combine Rays to create a nested expression, Hybrid Ray solves this problem by stacking other Rays together and processing them in one shot.

PathCast: This option processes all Rays one after one as PathCast. This option will show a yellow dotted gizmo around the ray and the ray will be cast seamlessly; your hybrid hit point will be displayed with an offset to its source ray.



Hint: You can remove the PathCast option in PathRays when you only need their Liner to avoid unnecessary processing.

PROPERTIES

Direction: Automatically selects the exact direction in either World or Local mode.

LocalDirection: Returns pure Local direction without the effect of Normalize.

TipDirection: Returns the subtraction of Tip from Base as the direction.

HitDirection: Returns the direction of the last ray break on the surface as the direction.

Tip: Returns the position of the Ray's tip.

TipTarget: Automatically returns the location of the last hit point of the Ray and replaces the Tip point if there is no detection.

HitDistance: Returns the exact distance of the ray to the hit point.

ContinuesDistance: Considers and returns the remaining distance of Ray from the point of impact.

DirectionLength: Returns the size of Direction itself in pure form.

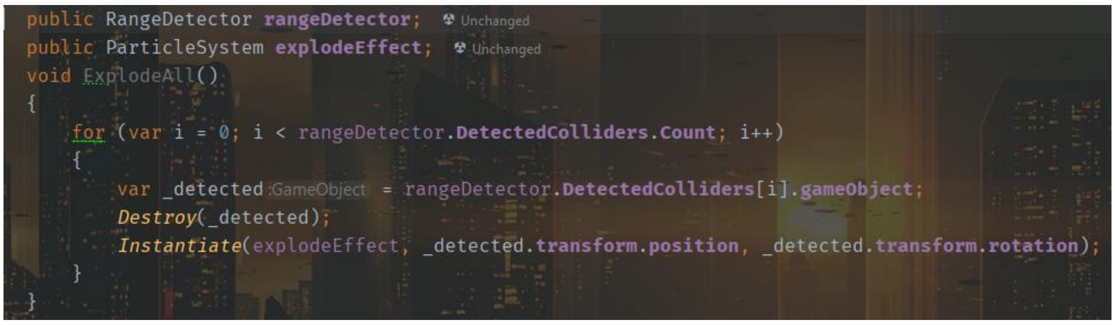
RayLength: This option calculates and returns the total length of the Ray.

DETECTORS:

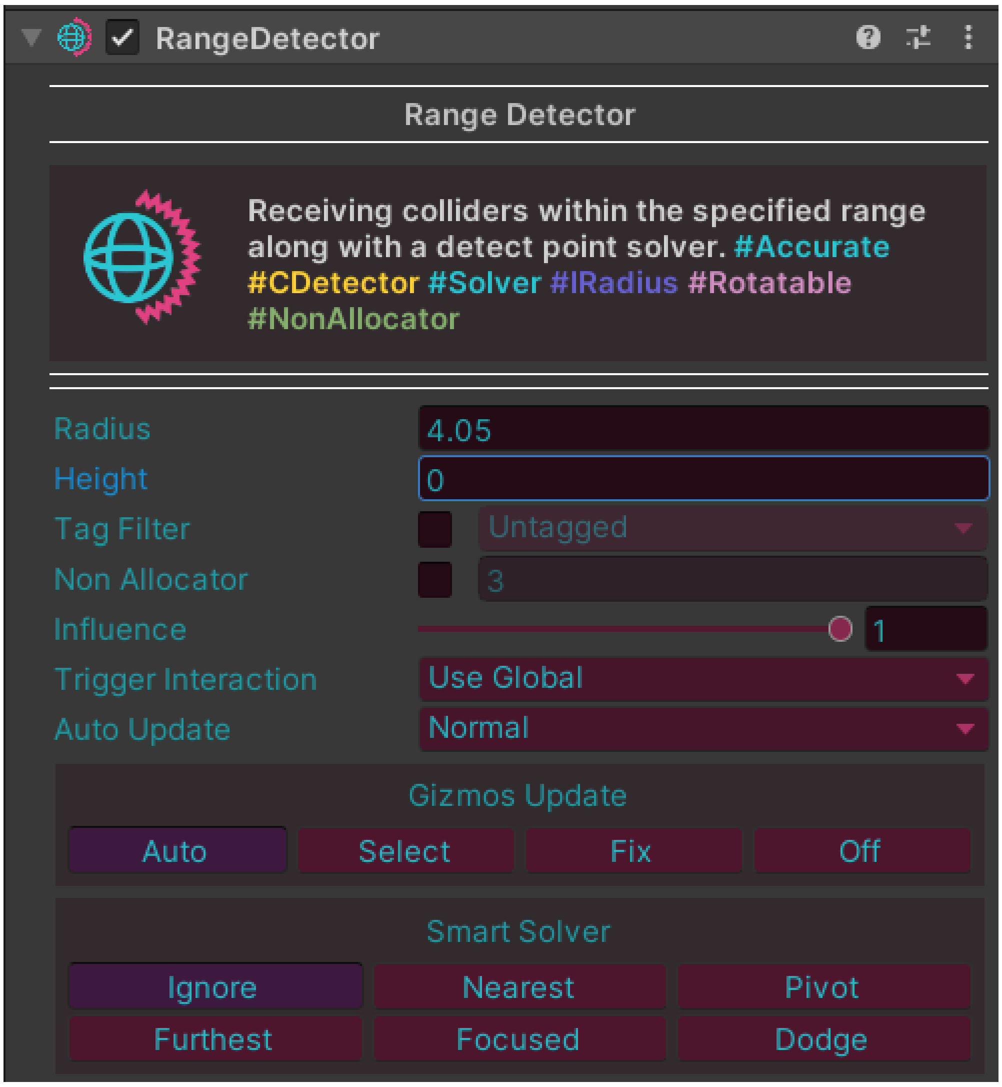
Detectors have various applications, whose working method is based on sending multiple Raycasts and receiving colliders. In the RaycastPro plugin, the detector tools use their high potential to accelerate and accurately perform detection.

COLLIDER DETECTORS:

The most common and probably the first type you will come across are **Collider Detectors**, they simply detect Colliders in their area and pass through their filters. In the meantime, after setting the parameters that you explain, you only need to access the anthologies by getting the Detected Colliders member.



```
public RangeDetector rangeDetector;  Unchanged
public ParticleSystem explodeEffect;  Unchanged
void ExplodeAll()
{
    for (var i = 0; i < rangeDetector.DetectedColliders.Count; i++)
    {
        var _detected :GameObject  = rangeDetector.DetectedColliders[i].gameObject;
        Destroy(_detected);
        Instantiate(explodeEffect, _detected.transform.position, _detected.transform.rotation);
    }
}
```


RangeDetector

Range Detector

Receiving colliders within the specified range along with a detect point solver. #Accurate #CDetector #Solver #IRadius #Rotatable #NonAllocator

Radius	4.05
Height	0
Tag Filter	Untagged
Non Allocator	3
Influence	1
Trigger Interaction	Use Global
Auto Update	Normal

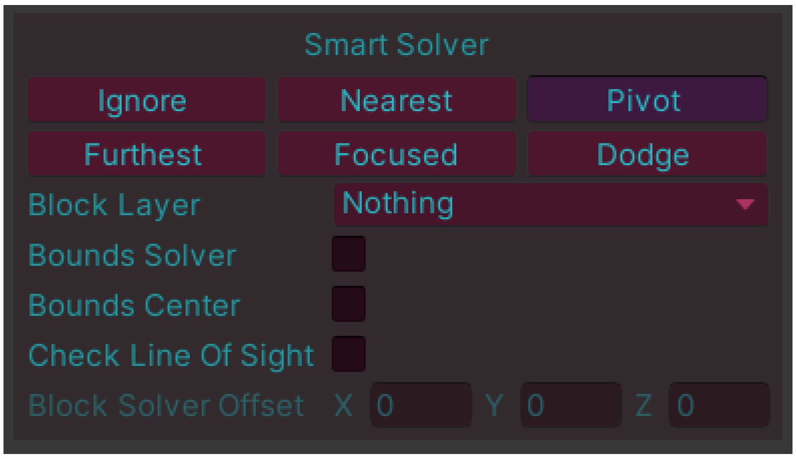
Radius: The Radius parameter in the Range Detector along with the Height are the dimension setting indicators, and you can visually and live determine the volume of the area whose colliders need to be detected.

TagFilter: As the name suggests, in addition to the Detect Layer, the Tag filter has also been added to the Collider Detectors.

NonAllocator: This tick helps to avoid updating the dimensions of List, With the active option, You will need to reserve enough memory in the detection array and the selection of colliders will be limited to the incoming array, this will prevent the creation of Garbage, Although the Raycast Pro plugin has extensive settings and filtering, and the overall goal is to make the tools more efficient than just optimization.

SMART SOLVER:

One of the important features of Detectors in Raycast Pro is their Smart Solver option. You can switch between different taps and see how they work. While this is very simple way you can implement an intelligent Line Of Sight system. The desired options specify the detect point to which **LineCast** will be aligned to it.


Smart Solver

Ignore | Nearest | Pivot

Furthest | Focused | Dodge

Block Layer: Nothing

Bounds Solver: ☐

Bounds Center: ☐

Check Line Of Sight: ☐

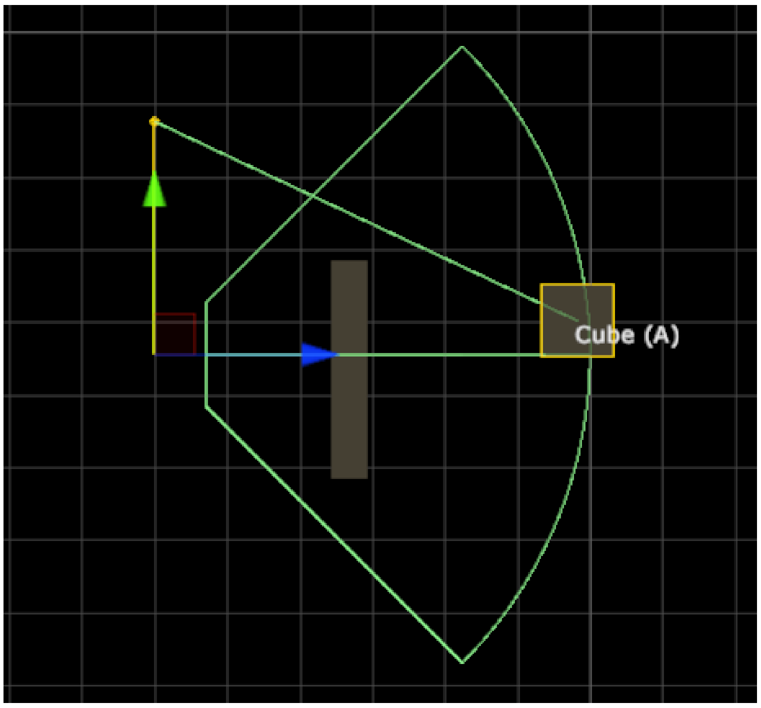
Block Solver Offset: X 0 Y 0 Z 0

Block Layer: A layer intended for blocking objects. If it is the same as the detect layer, it will detect the front object and block the backs.

Bounds Solver: This option is considered for optimization and limits the detection point in the bounds of a cube.

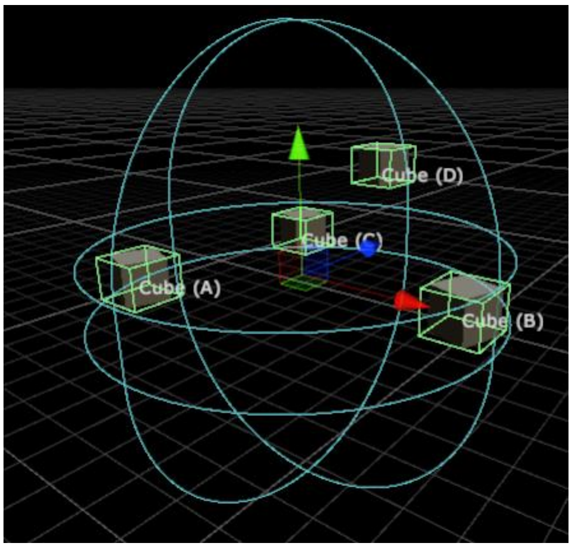
Check Line Of Sight: By activating this option, the blocking line will be checked.

Block Solver Offset: This option for Offset is selected from the starting point of view to avoiding obstacles and you can see how it works in the image below.



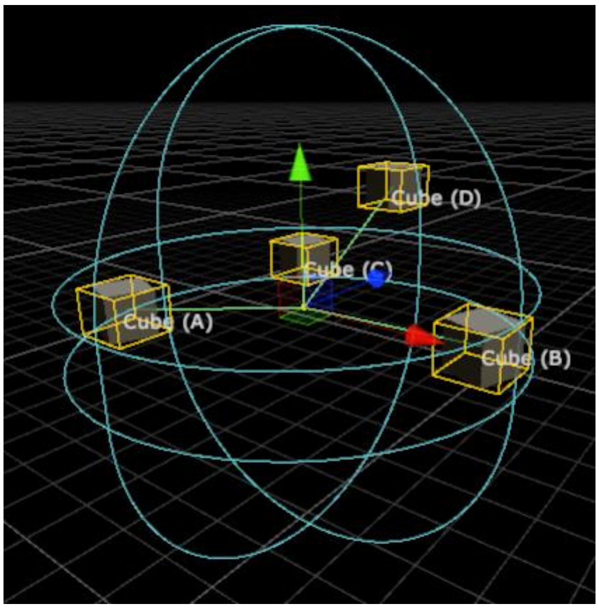
IGNORE:

It only avoids determining the detection point and line of sight and has the best performance.



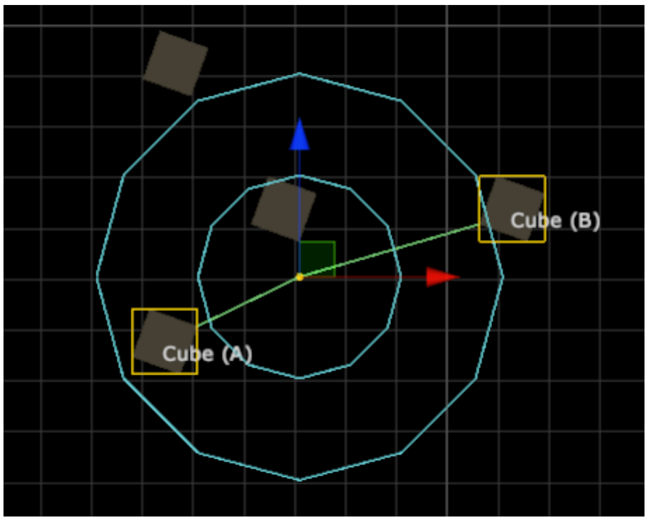
PIVOT:

It places the line-of-sight point in the pivot of every collider and has the highest performance after Ignore.



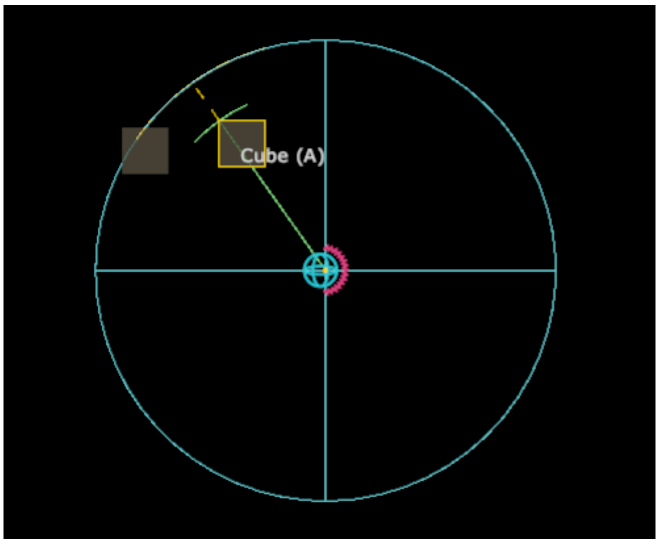
NEAREST:

The detection point is the closest to the center of the detector.



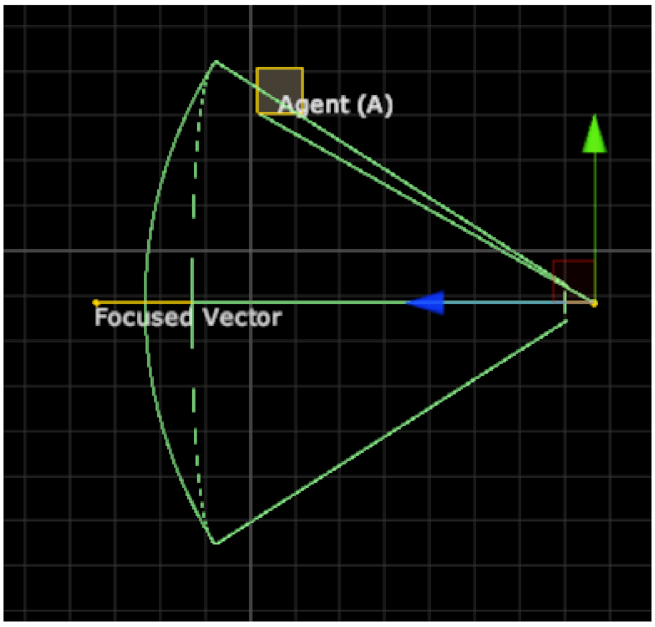
FURTHEST:

Places the detection point at the farthest place from the center of the detector. This option is used when you need your Collider to be 100% inside the area.



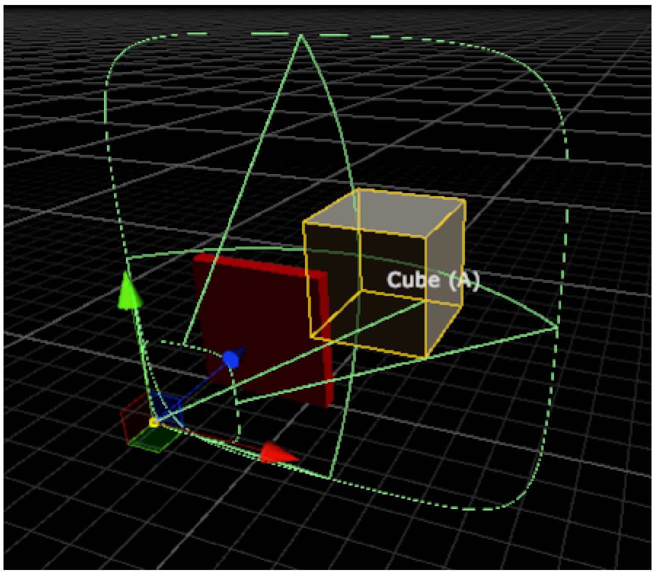
FOCUSED:

As seen in the image below, the detector tries to find the closest point to the focus vector. This option is used for Sight Detector, because your Agent is able to easily detect another Collider from the edge.



DODGE:

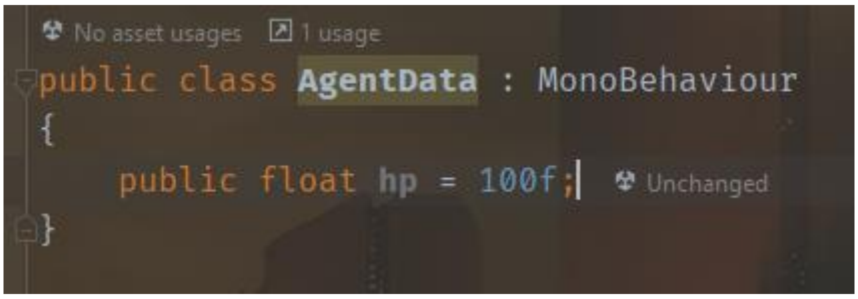
It always tries to find the best way to avoid blocking, however, it has the slowest performance compared to other detectors. It is recommended to activate the bounds solver when using it.



HOW TO GET COMPONENT TYPES (OPTIMIZE WAY)

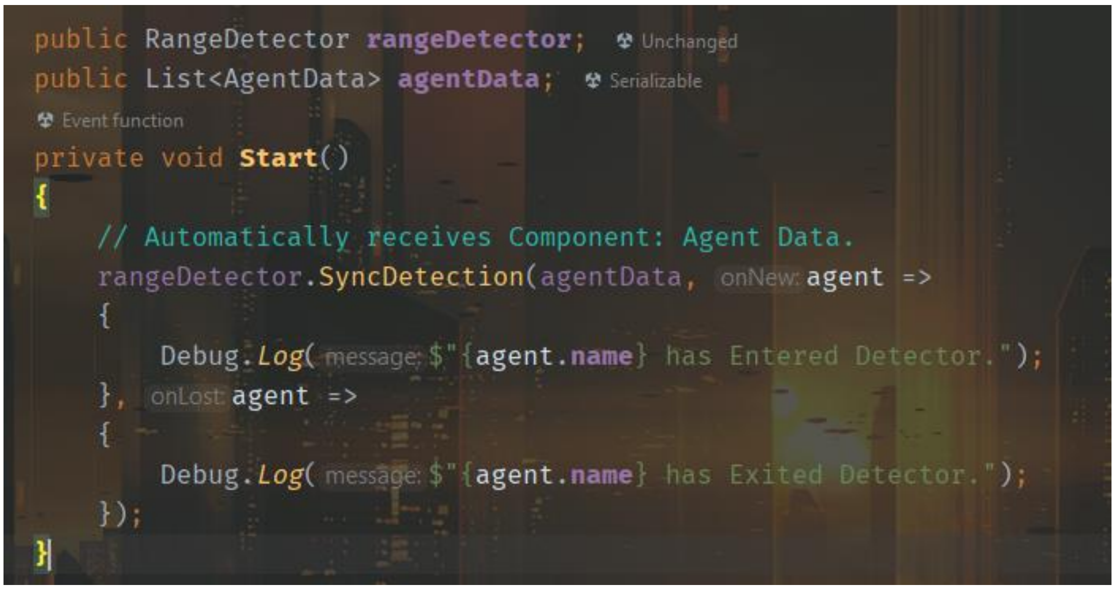
But if you intend to get a specific component in the update function, **GetComponent** is heavy method. The **SyncDetection** function transfers Colliders to the list during the entry and exit event and is optimized to a large extent.

1. Create a script called Agent Data and give it to the enemy characters. Then by Defining a list of AgentData and a little of code, Detector will automatically add it to the list.



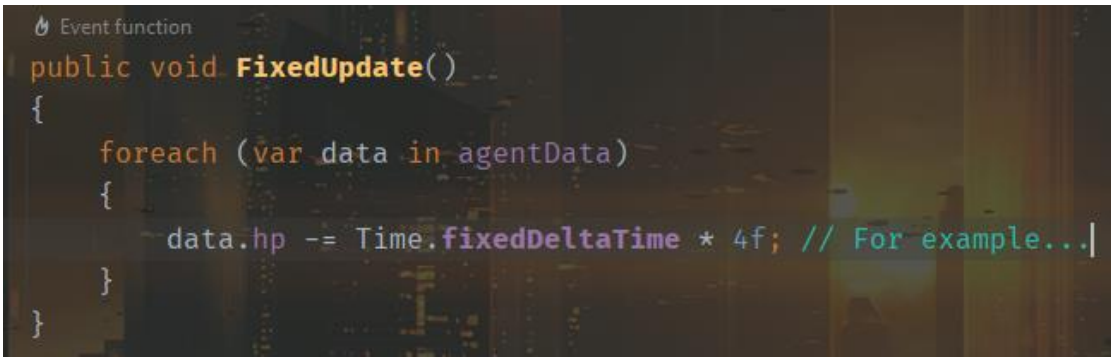
```
public class AgentData : MonoBehaviour
{
    public float hp = 100f;
}
```

Add this Script to your **Detector Carrier**:



```
public RangeDetector rangeDetector;  Unchanged
public List<AgentData> agentData;  Serializable
Event function
private void Start()
{
    // Automatically receives Component: Agent Data.
    rangeDetector.SyncDetection(agentData, onNew: agent =>
    {
        Debug.Log( message: $"{agent.name} has Entered Detector.");
    }, onLost: agent =>
    {
        Debug.Log( message: $"{agent.name} has Exited Detector.");
    });
}
```

2. After access, it is easy to change any member of this list with codes like below.

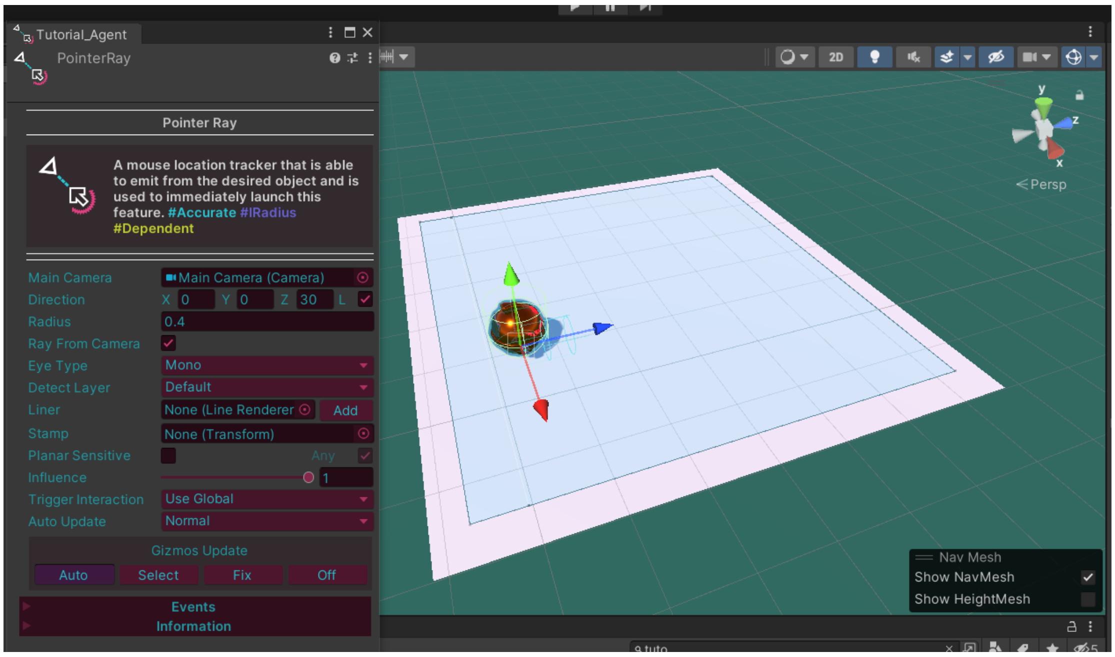


```
Event function
public void FixedUpdate()
{
    foreach (var data in agentData)
    {
        data.hp -= Time.fixedDeltaTime * 4f; // For example...
    }
}
```

HOW TO SETUP TARGET DETECTOR

In this tutorial, we want to describe how to work with **Target Detector**, although it seems to have a simple function, but by combining it with **Sight Detector**, you can have a member detector that gives you the percentage of sight.

1. So, first of all, make a controllable scene with the help of Pointer Ray and Tutorial Agent and make sure it moves. Place the agent in the Default layer along with the Player tag and enter the next step.


Tutorial_Agent

PointerRay

Pointer Ray

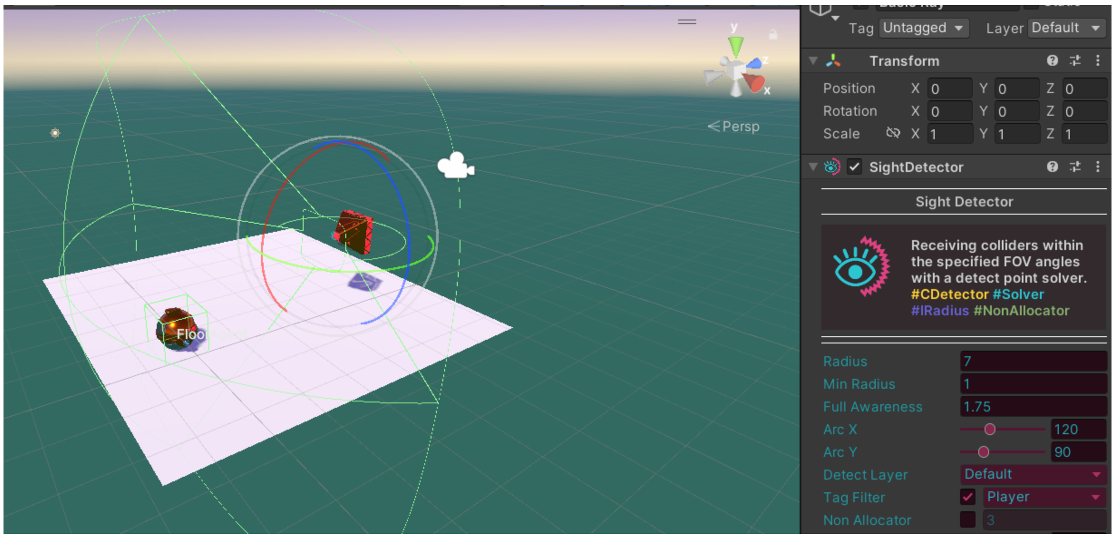
A mouse location tracker that is able to emit from the desired object and is used to immediately launch this feature. #Accurate #IRadius #Dependent

Main Camera: Main Camera (Camera)
Direction: X 0 Y 0 Z 30 L ✓
Radius: 0.4
Ray From Camera: ✓
Eye Type: Mono
Detect Layer: Default
Liner: None (Line Renderer) Add
Stamp: None (Transform)
Planar Sensitive: Any ✓
Influence: 1
Trigger Interaction: Use Global
Auto Update: Normal

Gizmos Update: [✓ Auto] Select Fix Off

Events
Information

2. After doing this, you need to mount a Sight Detector on your Agent's eye. Now you need a Sight Detector which can be your own or the enemy. For a quick explanation, I use a Pyrocaster. Find it in the demo Prefabs and after unpacking, place a Sight Detector on its tip. Set dimensions and size to standard and make sure you are using Tag Filter of Player.


Tag Untagged Layer Default

Transform
Position: X 0 Y 0 Z 0
Rotation: X 0 Y 0 Z 0
Scale: X 1 Y 1 Z 1

✓ SightDetector

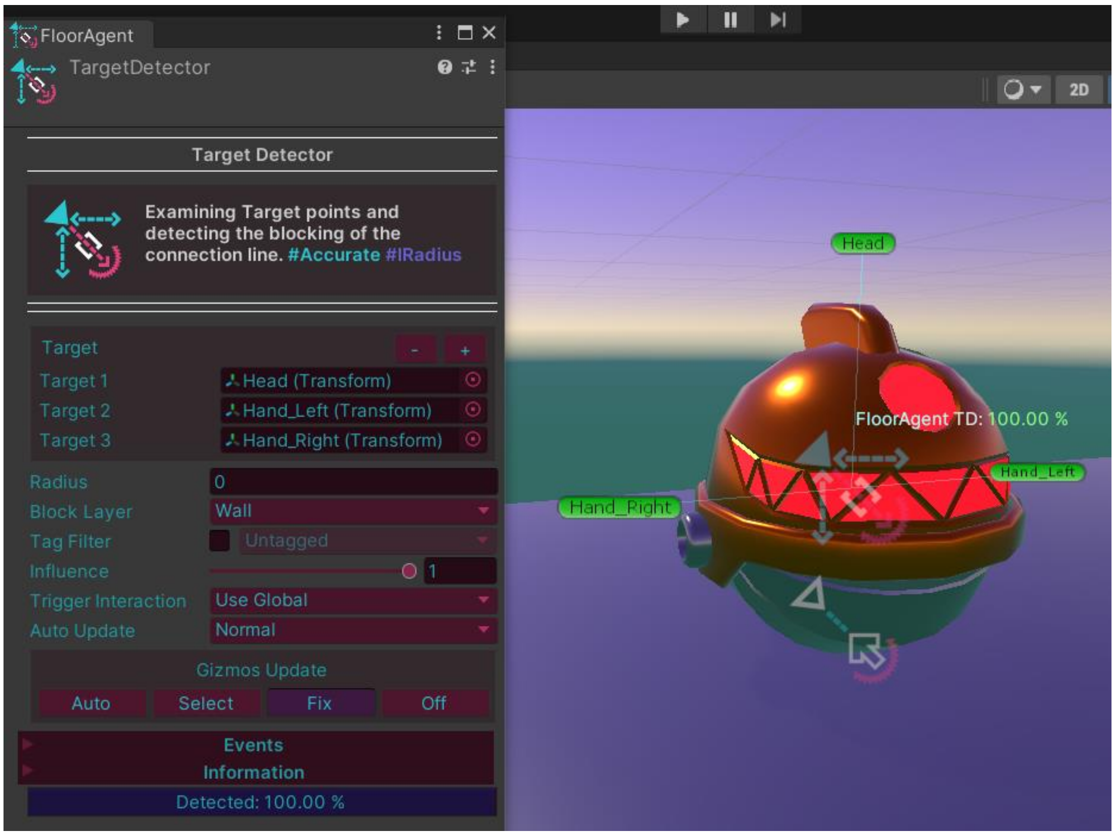
Sight Detector

Receiving colliders within the specified FOV angles with a detect point solver. #CDetector #Solver #IRadius #NonAllocator

Radius: 7
Min Radius: 1
Full Awareness: 1.75
Arc X: 120
Arc Y: 90
Detect Layer: Default
Tag Filter: ✓ Player
Non Allocator: 3

3. If the agent is working properly, it will be displayed in real-time and you don't need continuous tests. Now you need to install a Target Detector on your Agent and

specify the parts. Since this agent has no hands and feet, you can choose hypothetical points around him. If the targets are correct, it will display 100% visibility, now you need to set the Layer block to Wall. Also make sure the component is disabled.


FloorAgent

TargetDetector

Target Detector

Examining Target points and detecting the blocking of the connection line. #Accurate #lRadius

Target	
Target 1	Head (Transform)
Target 2	Hand_Left (Transform)
Target 3	Hand_Right (Transform)
Radius	0
Block Layer	Wall
Tag Filter	Untagged
Influence	1
Trigger Interaction	Use Global
Auto Update	Normal

Gizmos Update: Auto Select Fix Off

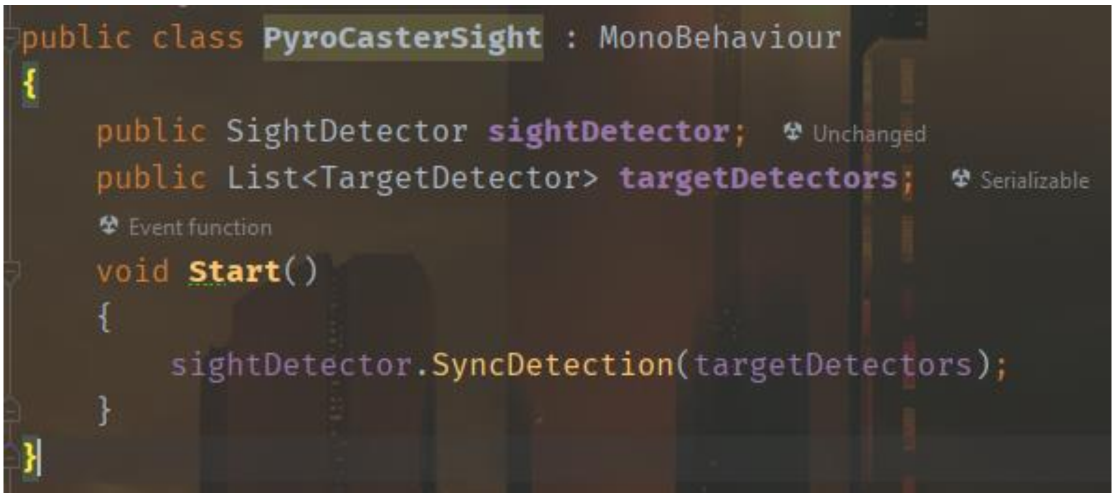
Events

Information

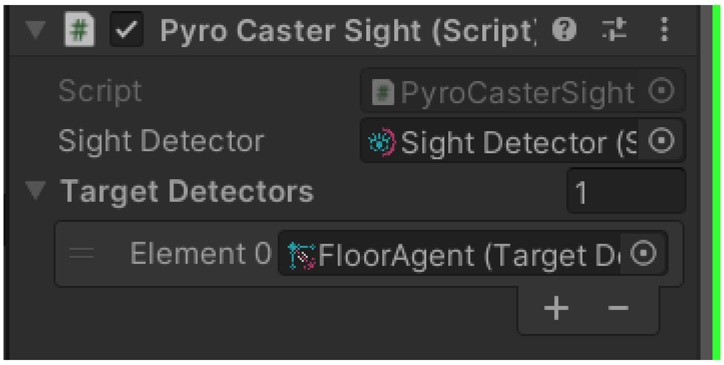
Detected: 100.00 %

4. The way it works is that our Sight Detector always finds the Target Detector component and then casts it from its point instead of Casting from within the Agent itself and returns visibility information. This method is very flexible and you can improve your target detector. To run this system, it is enough to create a list of Target Detectors in the special PyroCaster component and sync it to Collider Detection so that the list is automatically updated. Then start casting from there.

Write the following script and add it to PyroCaster as described above. After selecting the SightDetector, press the Play button and as soon as the Collider is detected, your List will be updated.

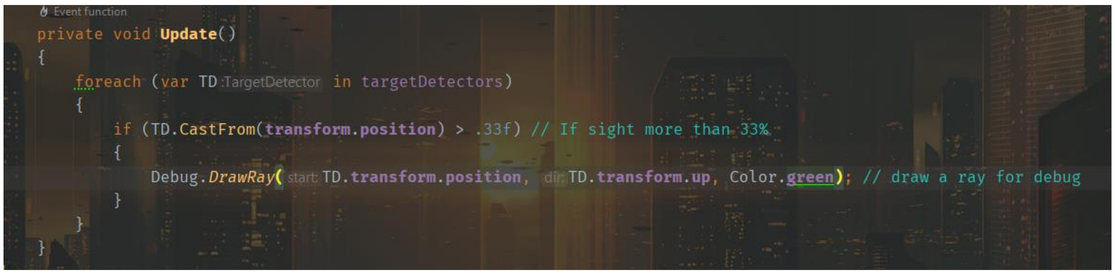


```
public class PyroCasterSight : MonoBehaviour
{
    public SightDetector sightDetector;  ⚙ Unchanged
    public List<TargetDetector> targetDetectors;  ⚙ Serializable
    ⚙ Event function
    void Start()
    {
        sightDetector.SyncDetection(targetDetectors);
    }
}
```



```
Pyro Caster Sight (Script)
Script                 PyroCasterSight
Sight Detector         Sight Detector (S
Target Detectors       1
    Element 0          FloorAgent (Target D
```

5. The following code starts Casting manually and returns the correct visibility. Here, each TD object is the one that is displayed more than the specified value.



```
Event function
private void Update()
{
    foreach (var TD:TargetDetector in targetDetectors)
    {
        if (TD.CastFrom(transform.position) > .33f) // If sight more than 33%
        {
            Debug.DrawRay( start: TD.transform.position,  dir: TD.transform.up, Color.green); // draw a ray for debug
        }
    }
}
```

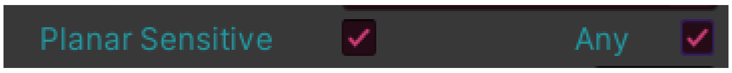
6. This solution gives you access to the found object, but easier solutions with simple components will also be developed in the future.

PLANERS:

planar is a tool to guide the direction of the RaySensor and in this version it is offered as an **experimental version.**

HOW TO SETUP PLANAR:

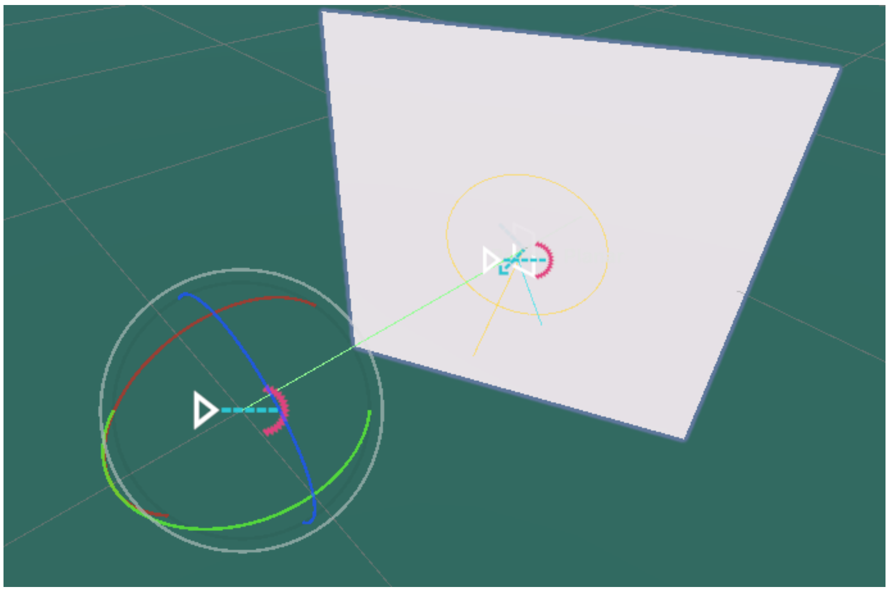
1. Create a new scene and a Basic Ray from the Raycast Pro panel.


Planar Sensitive ✓ Any ✓

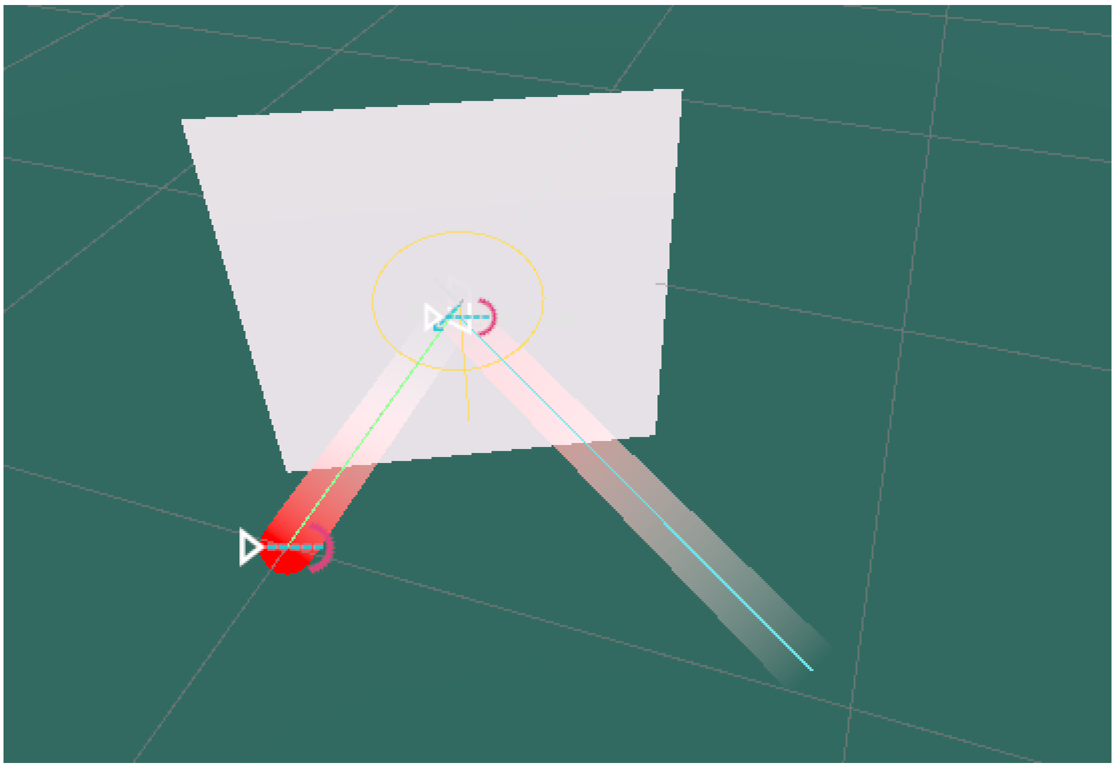
Make sure you tick Planar Sensitive, the Any option means accepting all planers. Otherwise, you have to select planar manually, which of course has better performance.

2. Now select a Reflect Planar from the panel and place it in front of the Ray as shown below. Then turn it slightly until a reflected ray has been seen.

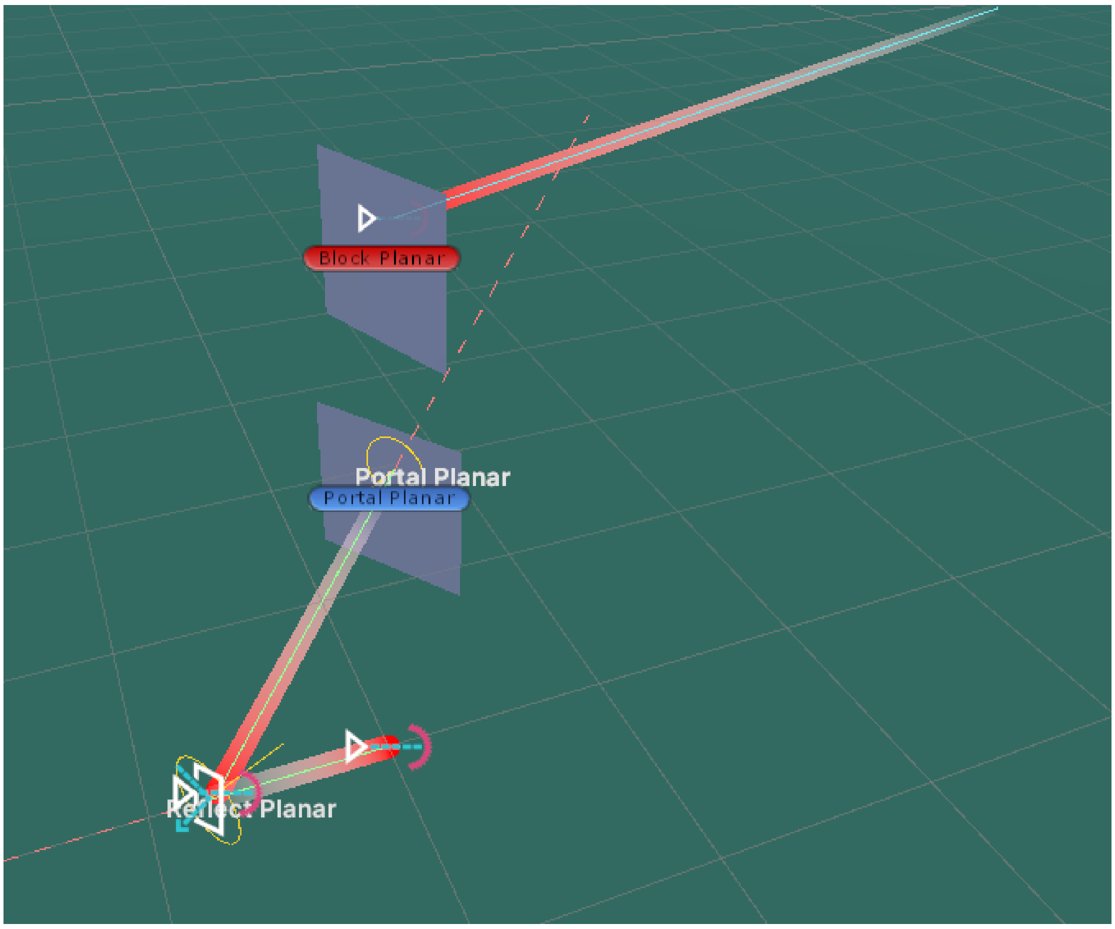
Hint: planers cannot work in Real-time in the Editor due to the creation of clones, but by pressing the play button, the process of creating clones starts.



3. Stop again and play after setting a **Liner** to Cut On Hit mode. The clone automatically tries to copy the parameters of the Ray behind it.

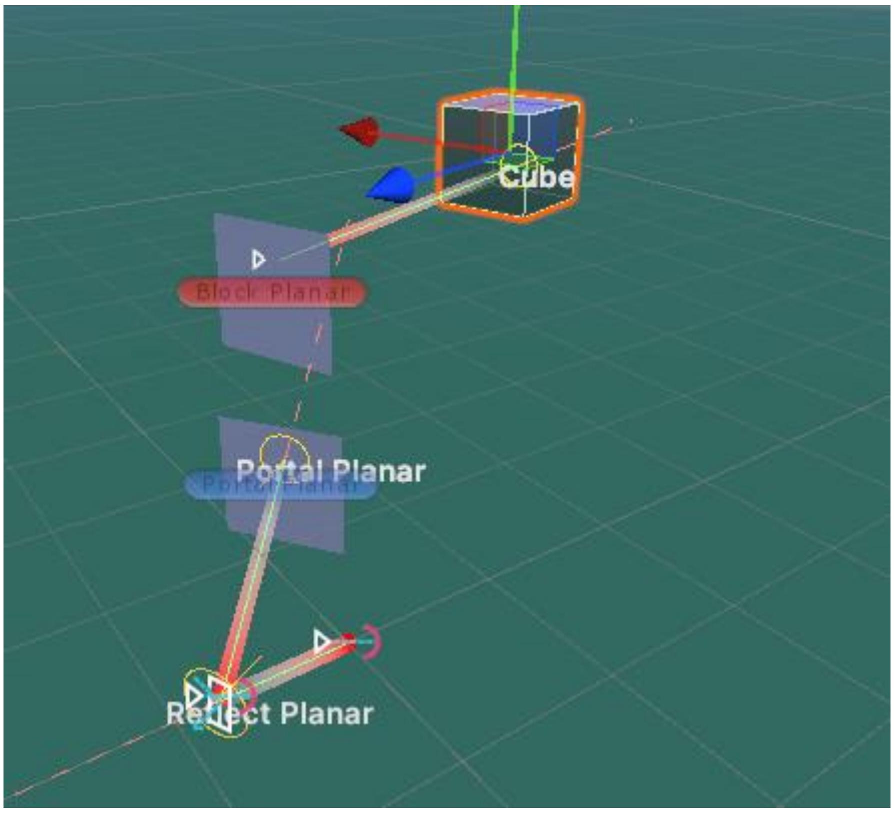


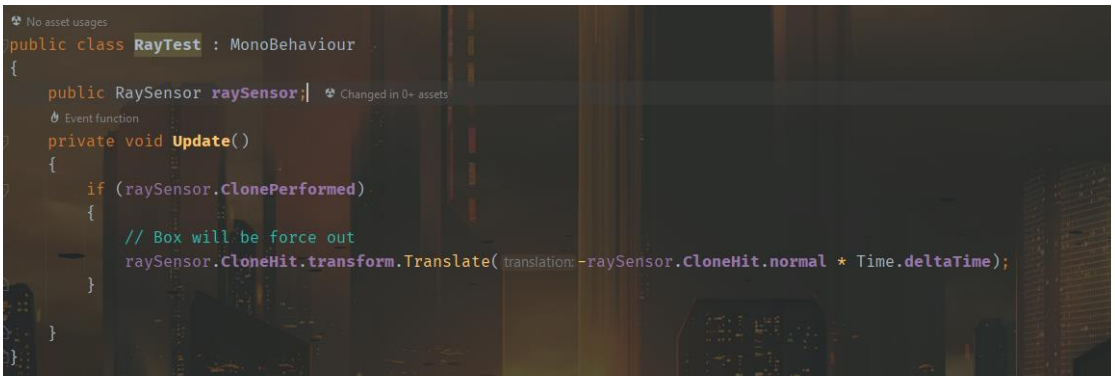
4. Then expand the process and extend the Ray's path by placing a Planar Portal further along. In its first parameter, Portal Planar asks you for the outer location, it is important that the dimensions of the exit object are also flat, otherwise you may experience strange behavior. Use **Block Planar** for output, which is used as an auxiliary Planar.



5. Next you need to access the Clone collision point, there are two ways. An easy way for non-coders is to use Stamp, which you can use as collision coordinates. The second way is Script, which we are currently testing second way.

First create a Box like below in the Wall layer and add this layer to the Ray as well, by adding the following code it will push it back when **CloneHit** hits.





```
No asset usages
public class RayTest : MonoBehaviour
{
    public RaySensor raySensor;    Changed in 0+ assets
    Event function
    private void Update()
    {
        if (raySensor.ClonePerformed)
        {
            // Box will be force out
            raySensor.CloneHit.transform.Translate( translation: -raySensor.CloneHit.normal * Time.deltaTime);
        }

    }
}
```

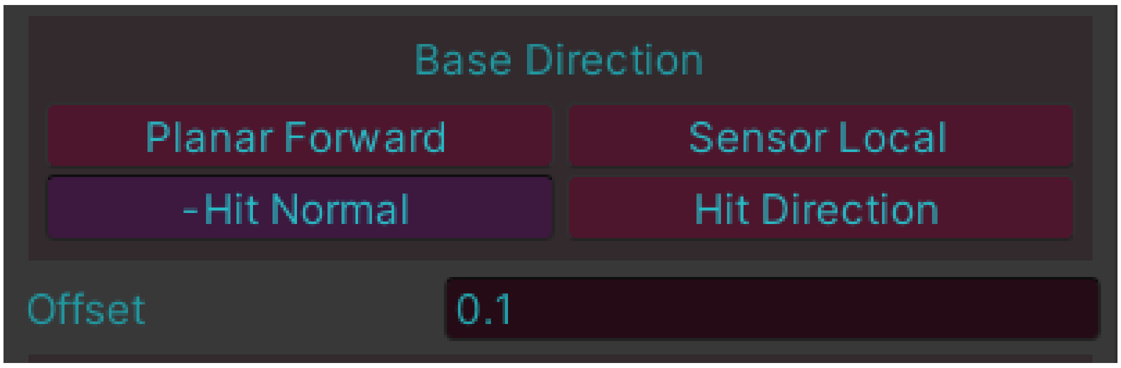
ClonePerformed: Use this Property when you need the Clone detection condition. Also, in if, it causes Error missing Reference to be solved.

CloneHit: This Property returns the clone RaycastHit, use it for access clone hit data.

LastClone: This Property returns the last CloneRay itself.

BASE DIRECTION:

It is to produce Clone Ray, which can be based on the following formulas. Please be careful when using this option because it seems a bit complicated.



Offset 0.1

LENGTH CONTROLL:

Continues: This option is the most common possible mode you can see, the Ray takes its remaining length out of the planar, which is normal.

Constant: As it is known, certain length of Ray is out of planar.

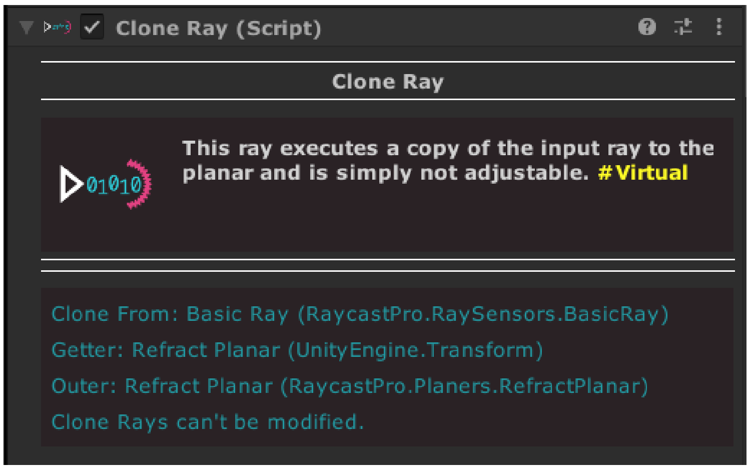
Sync: The same length of input Ray comes out of the Planar.

OUTER TYPE:

This option specifies the type of Ray output, in the case of single-model Rays such as Box and Pipe, the output can be a Reference like themselves, but **PathRays** must be cloned due to their different structure. It is better to keep this option on Auto for now.

CLONE RAY:

CloneRay is an array copy of ray points that carries information about its parent ray. You can access its information from the **CloneHit** & **LastClone** property.



This ray executes a copy of the input ray to the planar and is simply not adjustable. #Virtual

Clone From: Basic Ray (RaycastPro.RaySensors.BasicRay)
Getter: Refract Planar (UnityEngine.Transform)
Outer: Refract Planar (RaycastPro.Planers.RefractPlanar)
Clone Rays can't be modified.

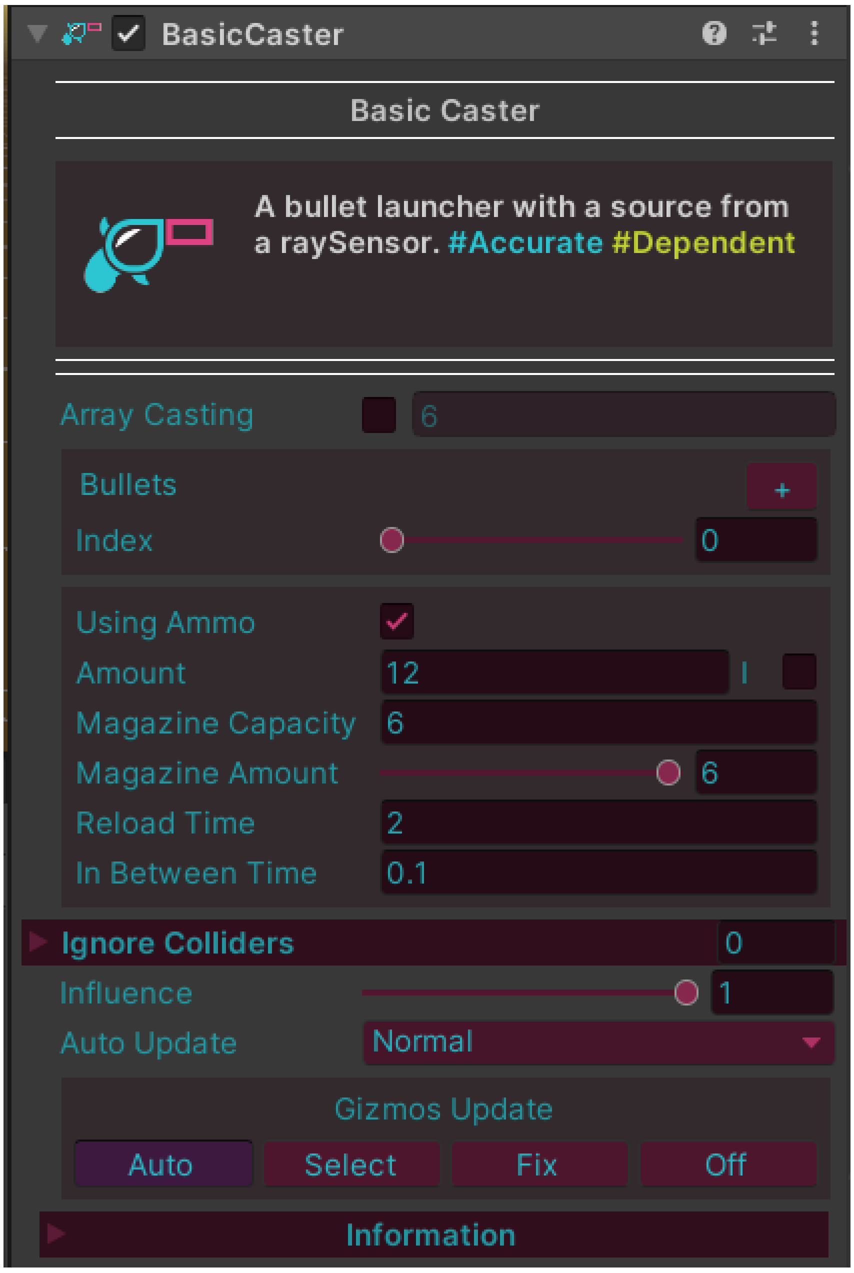
The **CloneHit** property returns the resulting Hit from the planar and **LastClone** for Get final clone information.

CASTERS:

Casters have the task of sending a bullet along a ray by taking a source, so they are dependent on a **RaySensor** and work with **Bullets**, before making them, make sure you have prepared the bullets in prefab and RaySensor that you need.

BASIC CASTER:

This Caster model is only used for simple tasks and quick Bullet Casting, in any case, it can only mount Basic Bullet and fortunately, it does not need RaySensor, and it shoots in its **forward** direction in 3D and to the **right** in 2D.



BasicCaster

Basic Caster

A bullet launcher with a source from a raySensor. #Accurate #Dependent

Array Casting ☐ 6

Bullets +

Index 0

Using Ammo ✓

Amount 12

Magazine Capacity 6

Magazine Amount 6

Reload Time 2

In Between Time 0.1

Ignore Colliders 0

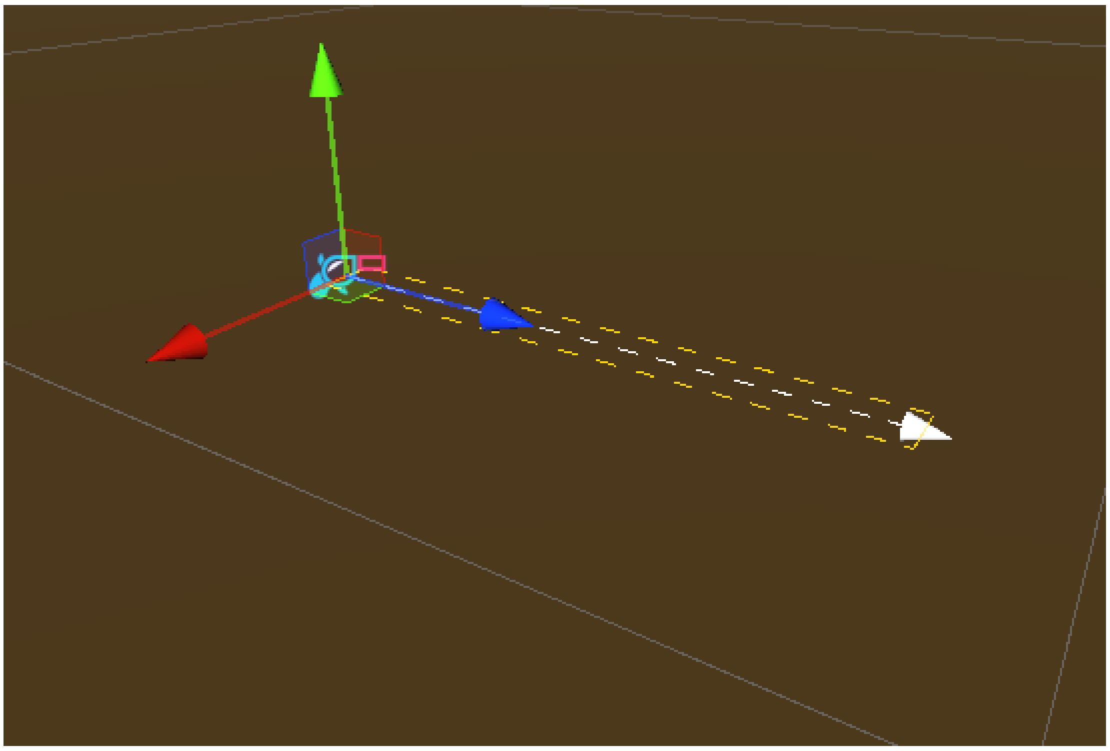
Influence 1

Auto Update Normal

Gizmos Update

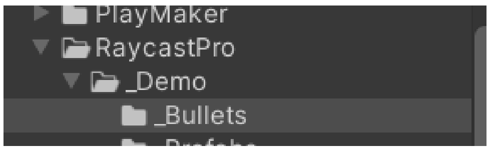
HOW TO SETUP BASIC CASTER:

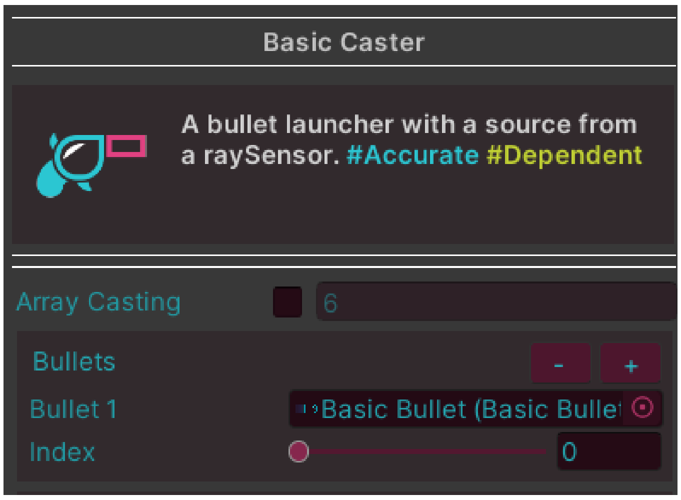
1. First, make a simple Caster from the RaycastPro panel.



2. Now that's enough, put a bullet on it. You can use Panel to make bullets, but it is better to make Prefab bullets.

For the simplicity of the test, I put a Folder called **Bullets** in the plugin. Use it and add Basic Bullet to Caster.





Basic Caster

A bullet launcher with a source from a raySensor. #Accurate #Dependent

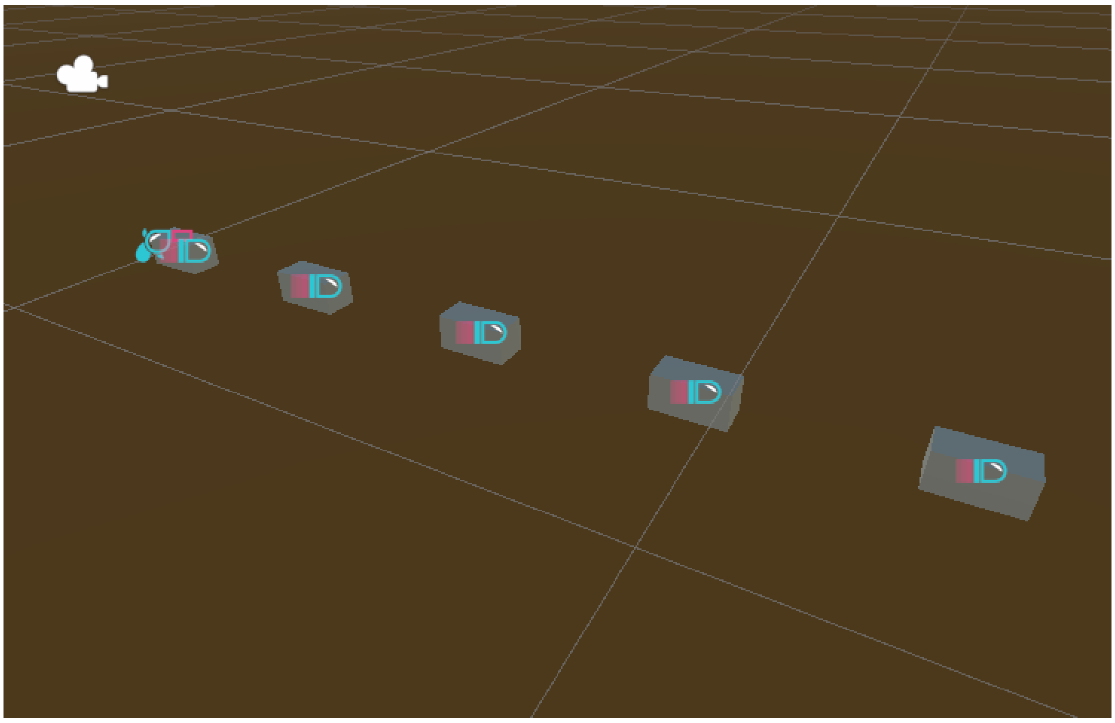
Array Casting 6

Bullets - +

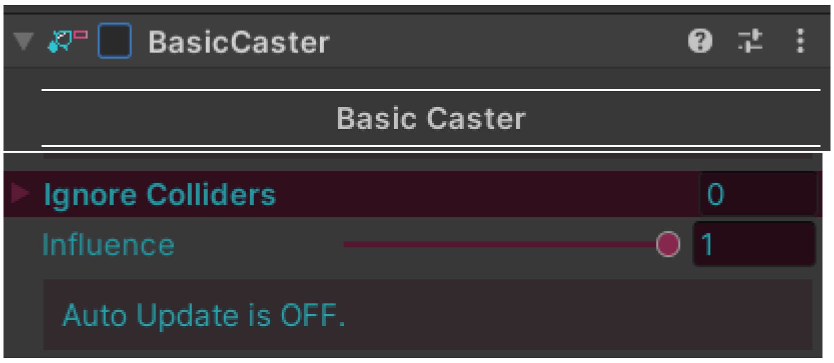
Bullet 1 Basic Bullet (Basic Bullet

Index 0

3. Now, by pressing the Play key, the shooting starts.



4. But you need to be able to use it whenever needed inside the code. This is also very simple. Disable the Caster component, which causes it to stop processing forever.



BasicCaster

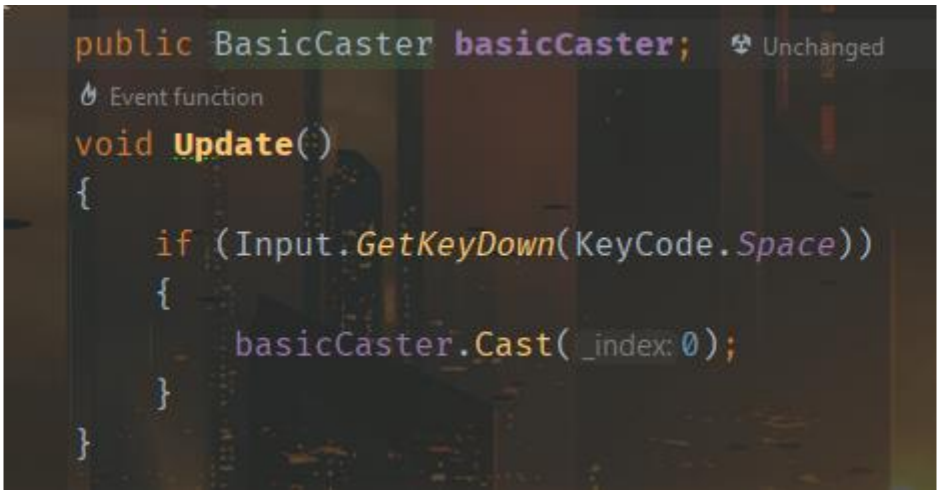
Basic Caster

Ignore Colliders 0

Influence 1

Auto Update is OFF.

5. Create a simple script and get the BasicCaster and call it whenever needed with the Cast method. Pay attention that the **_index** parameter refers to the bullet number, which is the first zero.



```
public BasicCaster basicCaster;  Unchanged
 Event function
void Update()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        basicCaster.Cast( _index: 0);
    }
}
```

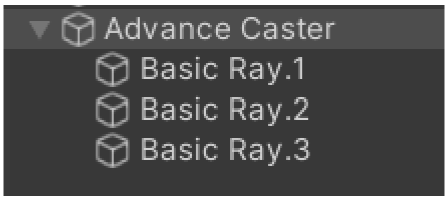
6. Press the space key, so easily the bullet you set will be fired.

ADVANCE CASTER:

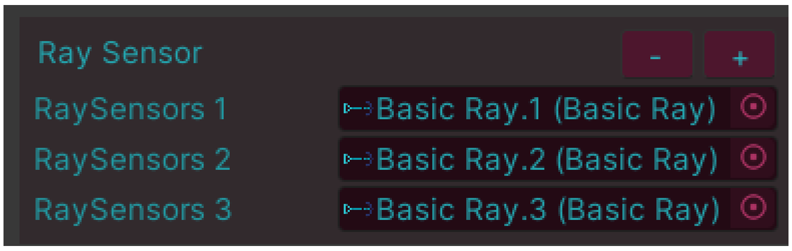
If you have understood the above tutorial well, now you can use **AdvanceCaster** to make complex weapons, which will make your work very fast and accurate and still with the **ArrayCasting** standard. I will explain further.

SETUP ADVANCE CASTER:

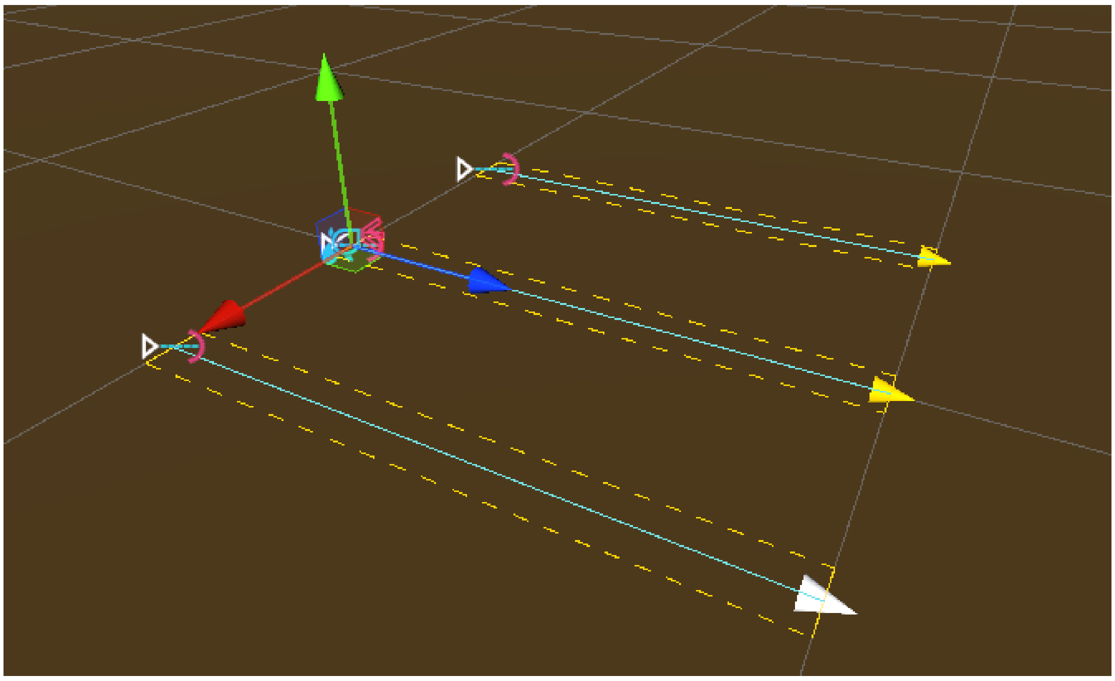
1. To start, make an Advance Caster like the one below with 3 simples BasicRays that each one in its own place. Also insert RaySensors into Caster to see a gizmo like below.



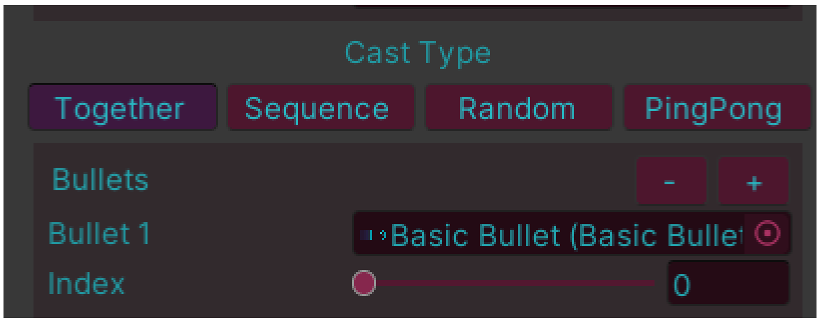
```
▼ Advance Caster
    Basic Ray.1
    Basic Ray.2
    Basic Ray.3
```


Ray Sensor

RaySensors 1	Basic Ray.1 (Basic Ray)
RaySensors 2	Basic Ray.2 (Basic Ray)
RaySensors 3	Basic Ray.3 (Basic Ray)



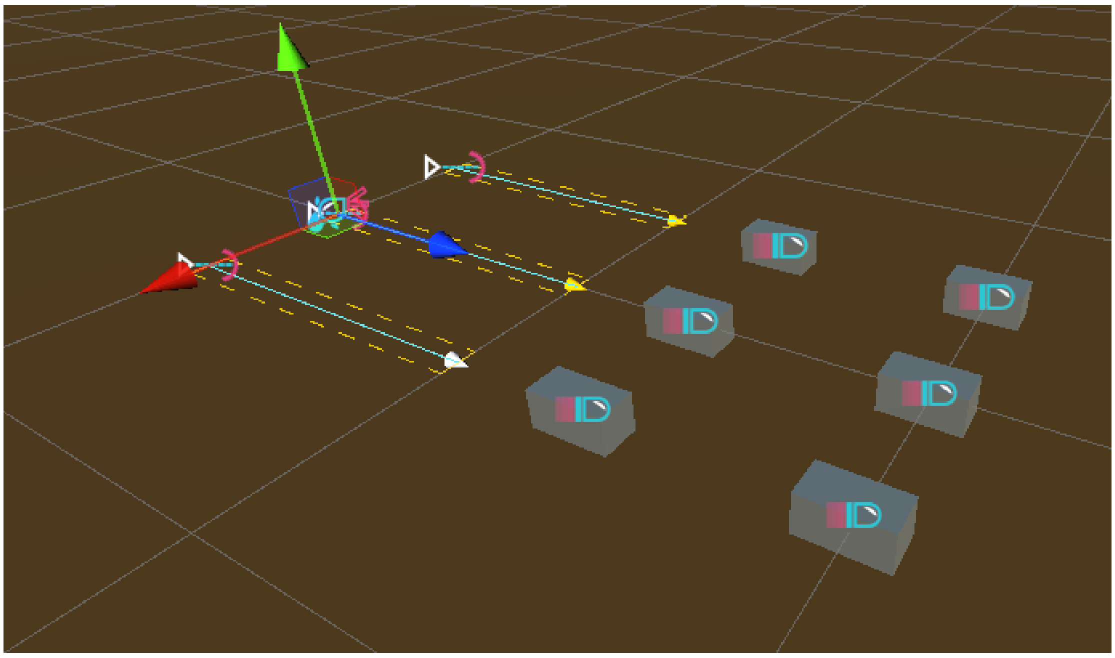
2. Now put a **BasicBullet** in the magazine as in the previous example and set the **CastType** to **Together**. By pressing the play key, the triple shot starts, and six bullets are fired from the caster each period of time.


Cast Type

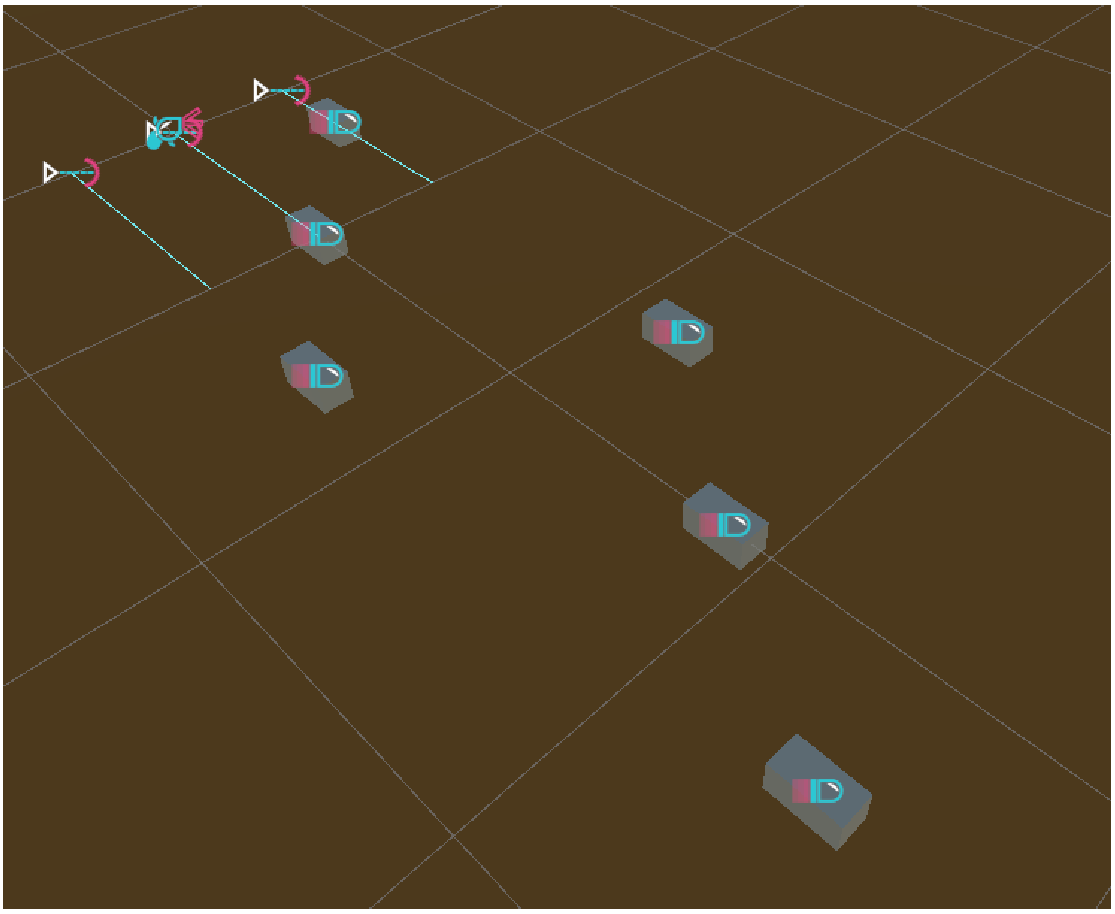
[✓ Together] Sequence Random PingPong

Bullets

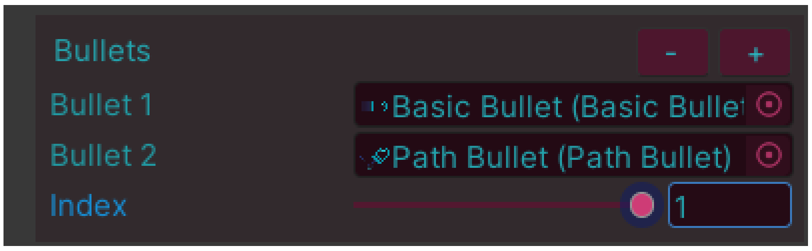
Bullet 1	Basic Bullet (Basic Bullet
Index	0



3. With the success of the first test, change the **CastType** mode and you can see that the Sequence works as follows. It is clear how the other modes work, but we will go further and try with different bullet models and RaySensor.



5. Now add the **PathBullet** to the magazine and change the index to 1 to fire this bullet model. If the **Render Pipeline** is standard, the color of the shooting bullet should be Cyan, but if you pay attention, these bullets do not go out of the path of the Ray. The reason is that the **PathBullet** always moves along the path of Ray no matter what type of that. Now change the size and direction of Ray, randomly and see the result.

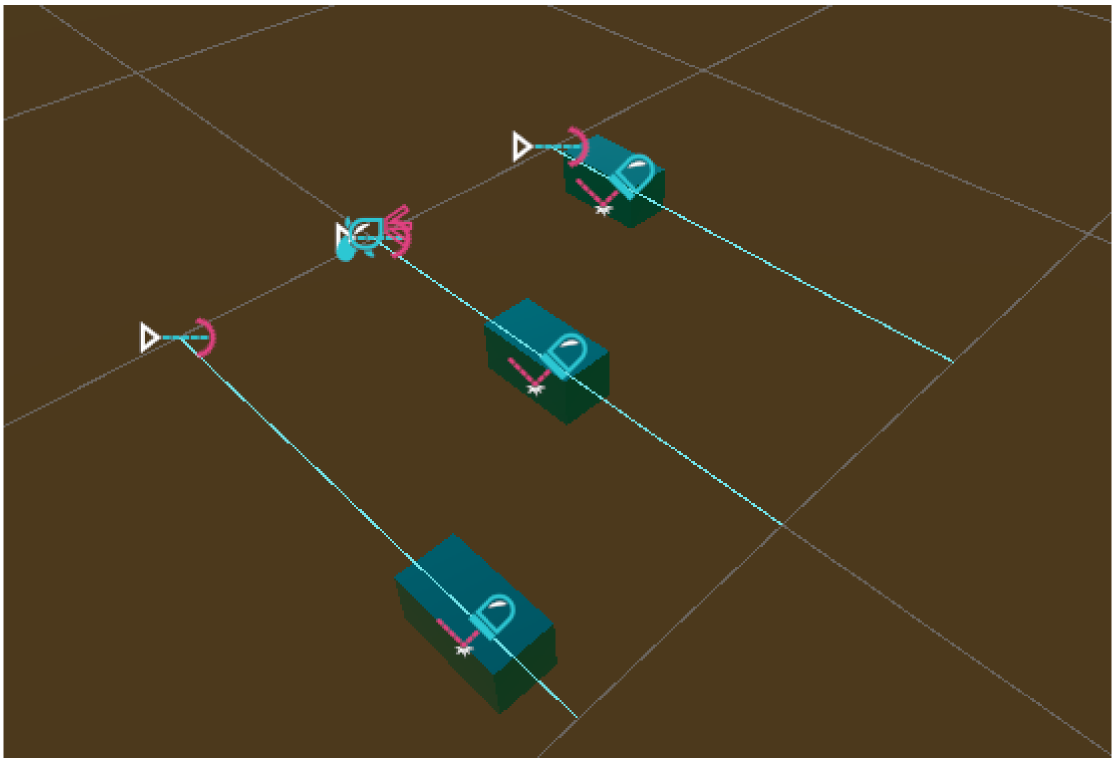


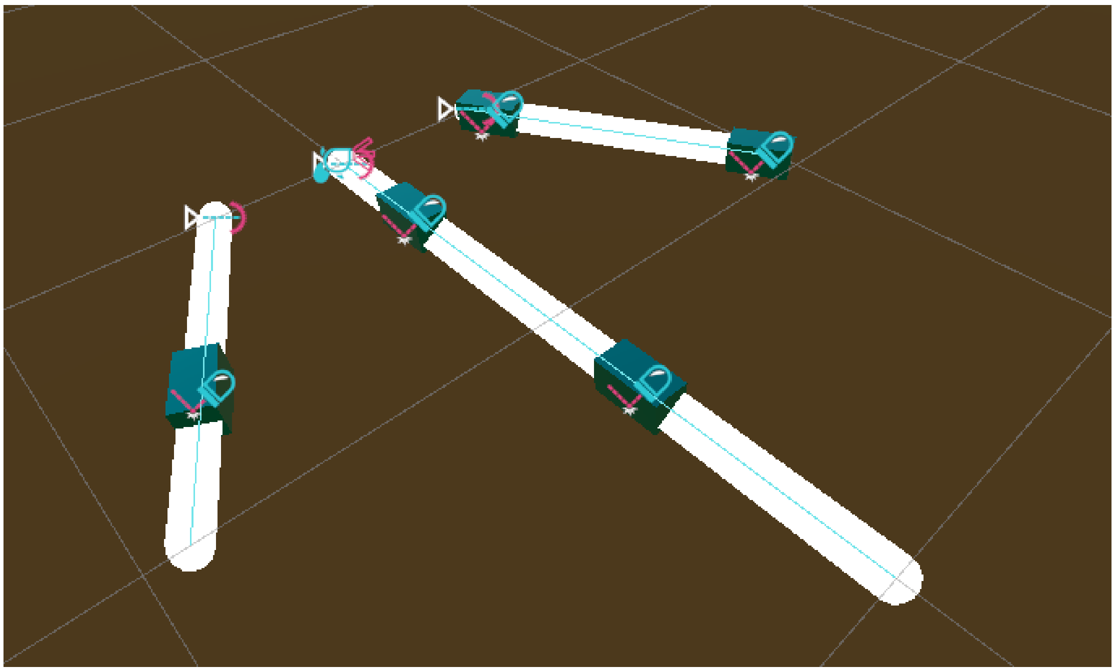
Bullets

Bullet 1 Basic Bullet (Basic Bullet

Bullet 2 Path Bullet (Path Bullet)

Index 1

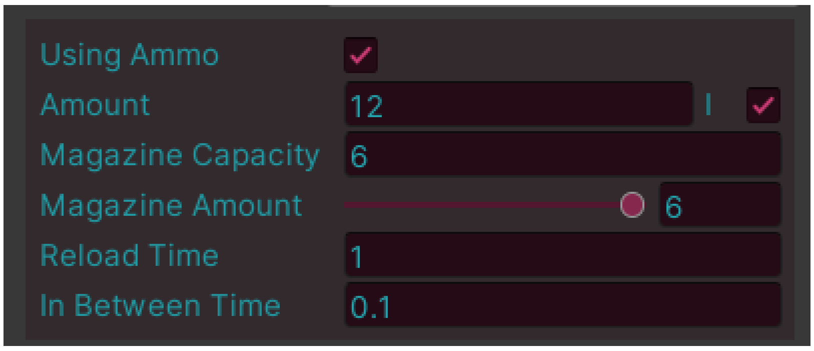




6. My strange shotgun is ready. In the following, I will also teach the function of each type of bullet, but it is necessary to see other common parameters between casters first.

AMMO:

This method has been optimized by deactivating the instantiated bullets and reusing them to reduce garbage.



Using Ammo	✓
Amount	12 I ✓
Magazine Capacity	6
Magazine Amount	6
Reload Time	1
In Between Time	0.1

Using Ammo: If you tick Ammo, the projectile firing system will work in the normal way of the gun. Otherwise, it works unconditionally, and you can code manually.

Amount: Obviously, this value shows the number of all the beams. Option I in front of this parameter is the infinite number of bullets.

Magazine Capacity: The amount of magazine that is available in each firing period until Reload Time.

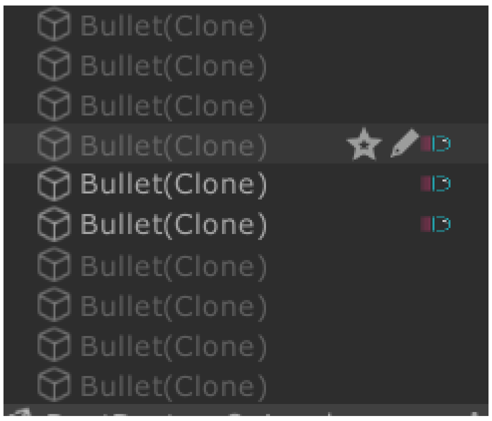
Magazine Amount: The current volume and the initial amount of the magazine, which cannot exceed the total volume of the magazine.

Reload Time: The pause time between each magazine filling cycle.

In Between Time: The time between firing each shot.

ARRAY CASTING (POOL MANAGER):

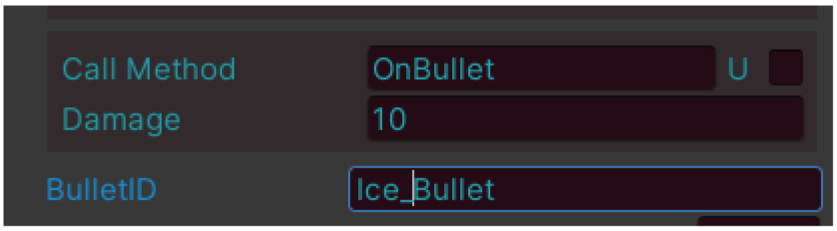
This method has been optimized by deactivating the instantiated bullets and reusing them to reduce garbage production. If your game uses a lot of bullets, use this option.

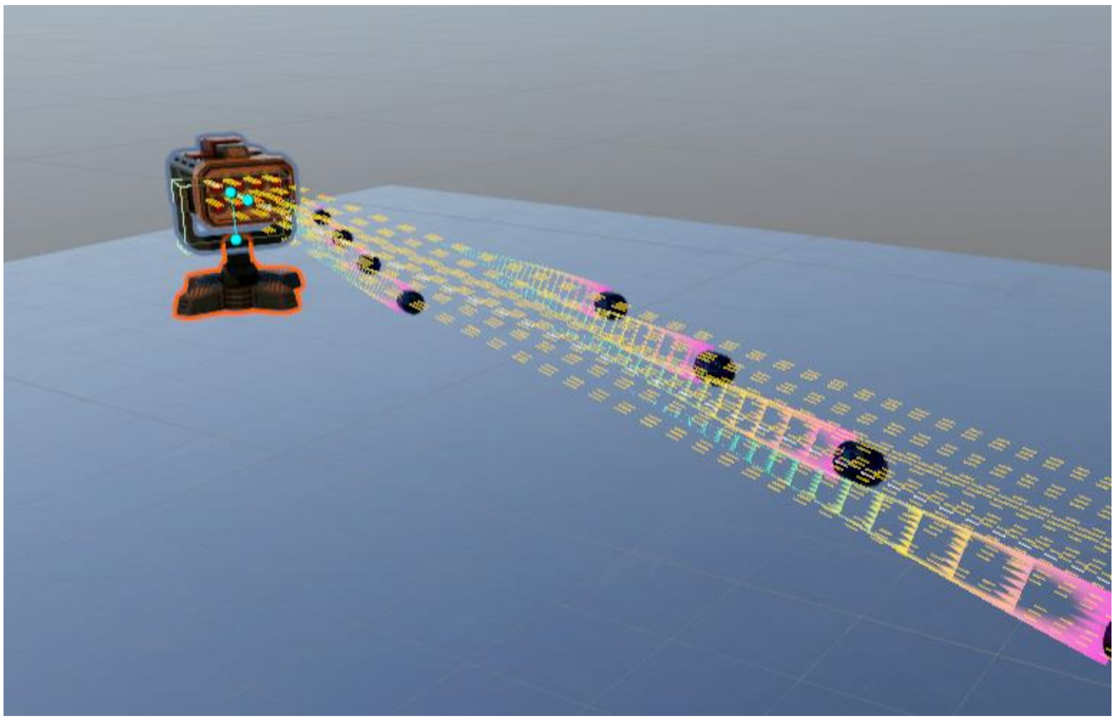

Bullet(Clone)
Bullet(Clone)
Bullet(Clone)
Bullet(Clone)
Bullet(Clone)
Bullet(Clone)
Bullet(Clone)
Bullet(Clone)
Bullet(Clone)
Bullet(Clone)

production. If your game uses a lot of bullets, use this option.

Hints To Use:

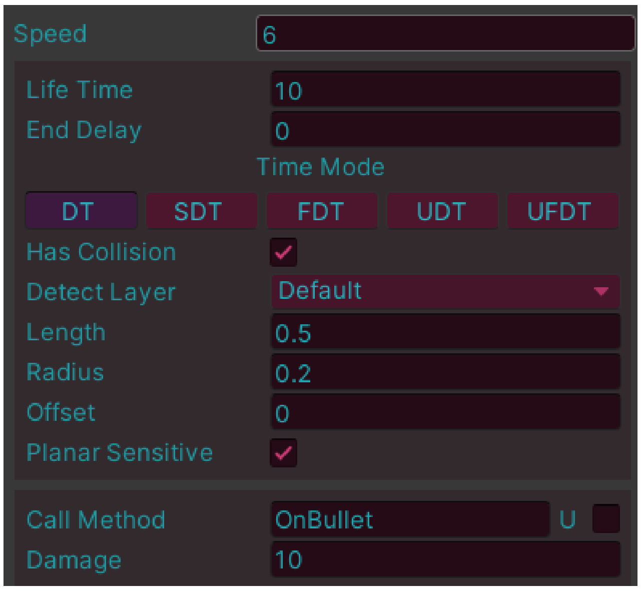
1. Array Casting is currently not modifiable in-game, so make sure the Array range is as large as you want.
2. In order to be able to change different bullets when shooting in the same way, you must use different Bullet ID.


Call Method OnBullet U
Damage 10
BulletID Ice_Bullet



BULLETS:

Currently, five bullet models, both in 3D and 3D, are supported in this package, which makes almost 99% of the shooting system workable. After a brief description of the general parameters, we will examine each bullet separately.



Speed	6
Life Time	10
End Delay	0

Time Mode

[✓ DT] SDT FDT UDT UFDT

Has Collision	✓
Detect Layer	Default
Length	0.5
Radius	0.2
Offset	0
Planar Sensitive	✓
Call Method	OnBullet U
Damage	10

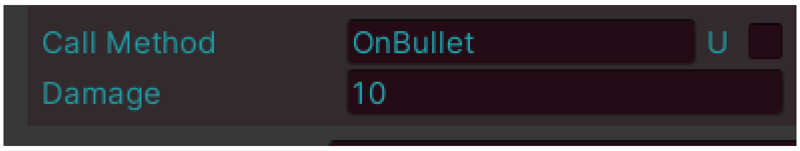
Speed: This parameter is very clear, whatever value determines the speed; it travels the same unit per second.

LifeTime: Bullet life is in seconds, which triggers OnEnd when it runs out.

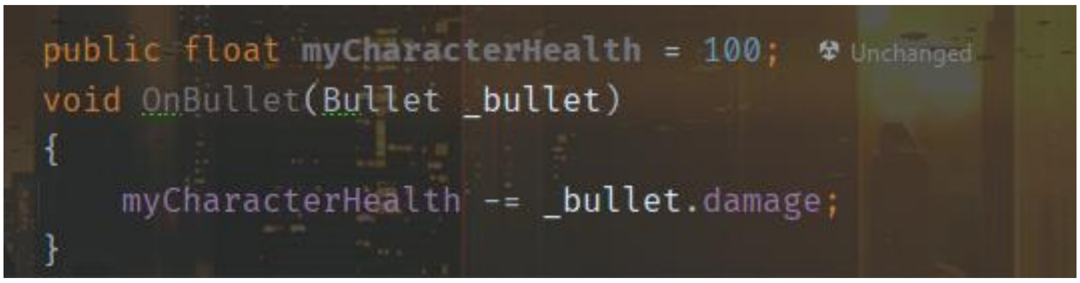
End Delay: This option is a pause before disabling or destroying the bullet, its use is when you want it to remain stable for a few seconds before death.

Collision Ray: By inserting a Ray Sensor into the body of the bullet, you can use it as a Collision. Keep in mind that this Ray should be a Child so that you can take advantage of its Planar Sensitive.

Call Method: This is the easiest way for you to pass an object-oriented script inside the target object with enough bullet information. Just write the desired method name on hit target object.



Call Method	OnBullet
Damage	10



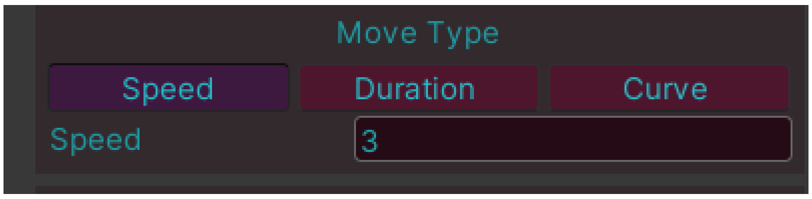
```
public float myCharacterHealth = 100;
void OnBullet(Bullet _bullet)
{
    myCharacterHealth -= _bullet.damage;
}
```

INSTANT BULLET:

As its name suggests, this bullet hits the target as soon as it is fired.

PATH BULLET:

PathBullet has an automatic system that always follows the path of the Ray, and it doesn't matter if your Ray is PathRay or not. In addition to the speed, there are two other ways to control the movement of the bullet.



Speed: 3

Local: In the Local parameter, when the bullet is casted, it bakes the ray path at the beginning of the cast, and you will no longer see the path of the bullet being disturbed by the movement of the caster. But if this tick is not present, the bullet will always move on the Path, even if the Path itself is moving.

RigidBody: If you add a *RigidBody* to the bullet prefab, the bullet will use ***RigidBody.SetPosition*** instead of the **transform.Translate** algorithm, which will have a physical effect on the Colliders, but it can continue its way without affect by them.

PHYSICAL BULLET:

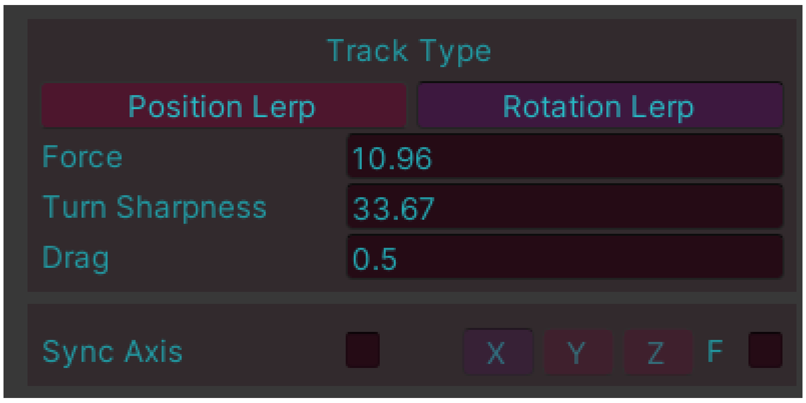
This bullet itself has a RigidBody, which is thrown with the initial force when fired.

TRACKER BULLET:

Tracker Bullet is a bullet that can follow its Caster **Track Target** based on two different algorithms.

Position Lerp: the first algorithm is based on the location that will always reach the destination after a certain period, for example, I can use the Projection type of ranged heroes in Dota2 or other strategy games.

Rotation Lerp: The second algorithm, which is based on rotation, acts like a ballistic missile and tries to change direction towards the target, but there is no guarantee that it will always hit it.



Track Type

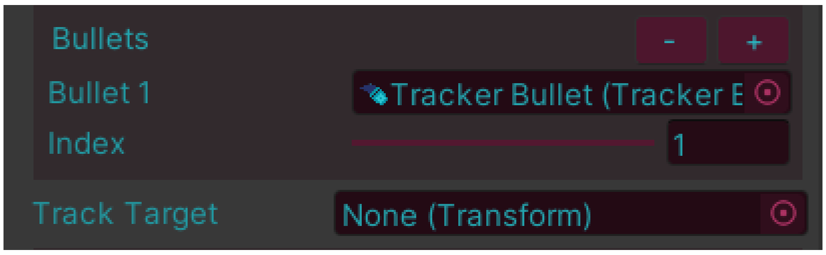
Position Lerp | Rotation Lerp

Force 10.96

Turn Sharpness 33.67

Drag 0.5

Sync Axis X Y Z F



Bullets - +

Bullet 1 Tracker Bullet (Tracker E

Index 1

Track Target None (Transform)

Force: Initial throwing power.

Turn Sharpness: The power to change the curvature of the following direction.

Drag: The amount of friction that decrease bullet force to zero.