

# Hit Me - Manual



## Table of contents

Requirements.....	3
Intro (Read this first!!!).....	3
Feature Overview.....	4
Ballistic Projectiles.....	4
Ballistic Path Drawing.....	5
Animation Projectiles.....	6
Animation Path Drawing.....	7
Visual Scripting.....	8
2D and 3D Support.....	8
Lots of handy APIs.....	8
Full Source Code.....	8
Tutorial: Let's spawn our first projectile.....	9
Ballistics Projectile Source.....	13
Ballistics Projectile Config.....	15
Animation Projectile Source.....	18
Animation Projectile Config.....	20
Animation Projectiles With Physics.....	29

<b>Target Movement Prediction</b> .....	<b>30</b>
<b>Config Assets</b> .....	<b>31</b>
<b>Alignment Components</b> .....	<b>32</b>
None.....	32
LookAt.....	32
AlignWithVelocity.....	33
AlignWithAnimationVelocity.....	33
Mixing Alignments (AlignLookLerp).....	34
<b>Destruction</b> .....	<b>35</b>
Destruction Component.....	35
Destruction Config.....	35
<b>Events</b> .....	<b>36</b>
<b>Projectile Registry</b> .....	<b>37</b>
<b>Frequently Asked Questions</b> .....	<b>38</b>
My targets are very far away and I can not hit them accurately.....	38
I need to predict bouncing and reflections (not supported).....	38
I need to simulate air resistance and drag (not supported).....	38
My animated projectile stops animating in mid air?.....	38
My animated projectile start falling directly from the source?.....	38
My animated projectiles stop or move weird in my 2D project.....	39
My ballistic projectiles do not fly in my 2D project, why?.....	39
The predicted path does not match the flight path.....	39
The ballistic projectiles do not move after spawning?.....	39
I need to spawn lots of projectiles. Is there some pooling?.....	39
Changing the config does not affect already existing projectiles!.....	39

# Requirements

**Unity 2021.2** or higher is required. If you can, please upgrade to the highest LTS version of Unity. The newer the minor version the better.

**Visual Scripting 1.7.2** or higher is required **if you want to use Visual Scripting**.

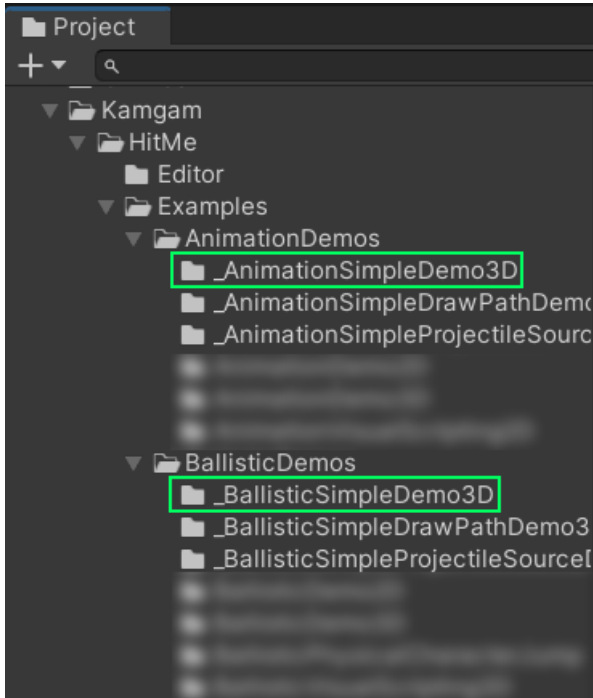
## Intro (Read this first!!!)

There are TWO kinds of projectiles this asset supports. One is the **BallisticProjectile**, the other one is the **AnimationProjectile**.

The difference is that the BallisticProjectile is just a fancy way of adding a start velocity to a rigidbody.

The AnimationProjectile on the other hand controls the projectile transform for the full duration of the animation (except when it hits something, more on that later).

The asset comes with quite a lot of demos to show you how everything is done. The demos are sorted by complexity from easy to advanced. Please start with the **Simple** demos first. They will teach you the basics.

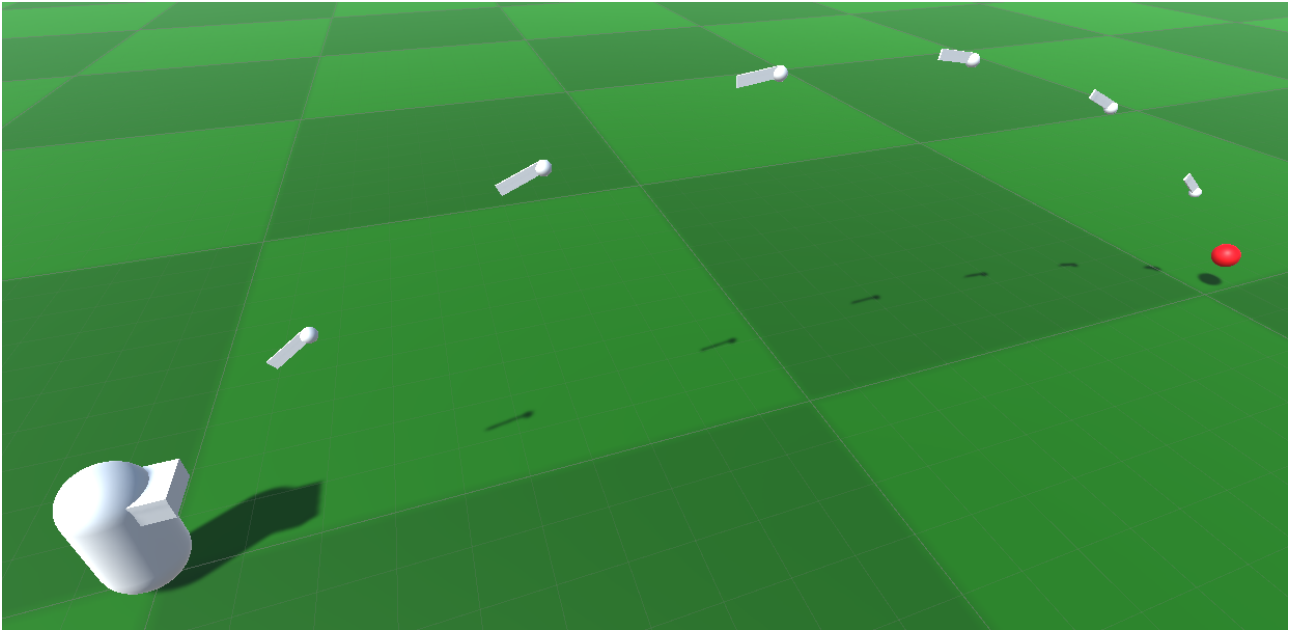


**Please notice that this is NOT a system to spawn 100s or 1000s of projectiles at the same time.** To do that you better use an optimized system (particles).

Please also notice this is **not a system to simulate air resistance, drag or reflections** (bouncing). More on that in the FAQs.

# Feature Overview

## Ballistic Projectiles



With the Ballistic API (BallisticProjectile, BallisticUtils) you can spawn a projectile and add a start velocity to reach a certain target.

<b>Projectile</b>	
Source Position	X 0 Y 0 Z 0
Source	None (Transform)
Source Offset	X 0 Y 0 Z 0
Source Offset Space	Self
Target Position	X 0 Y 0 Z 0
Target	None (Transform)
Target Offset	X 0 Y 0 Z 0
Target Offset Space	Self
<b>Ballistics</b>	
Dimensions	Physics 3D
Calculation Method	Time
Override Gravity	<input type="checkbox"/>
Gravity	X 0 Y -9.81 Z 0
Duration	2
Altitude	1
Altitude Base	Target
Allow Lower Altitude	<input checked="" type="checkbox"/>
Angle	45
Speed	20
Speed Use High Sol	<input type="checkbox"/>
Prediction Iteration	1

```

public class BallisticSimpleDemo3D : MonoBehaviour
{
    public Transform Source;
    public Transform Target;

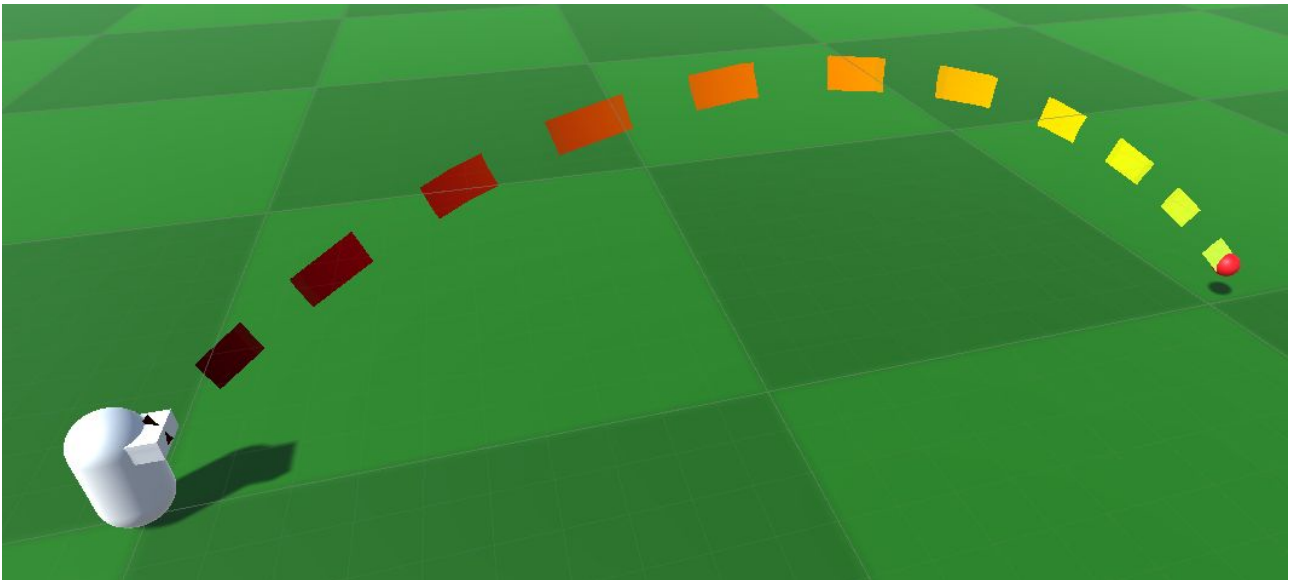
    public GameObject ProjectilePrefab;
    public BallisticProjectileConfig Config;

    public void Start()
    {
        BallisticProjectile.Spawn(ProjectilePrefab, Source, Target, Config);
    }
}

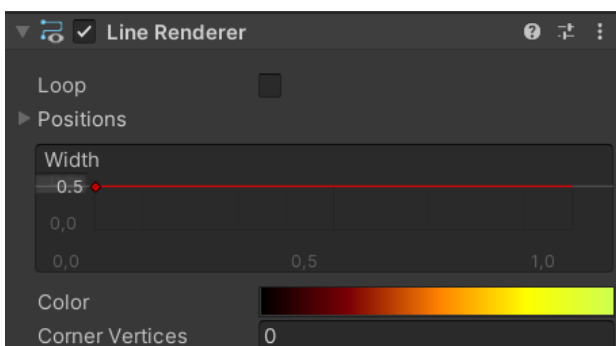
```

## Ballistic Path Drawing

Draws a preview of the path the projectile will take.



It uses the default Unity Line Renderer.



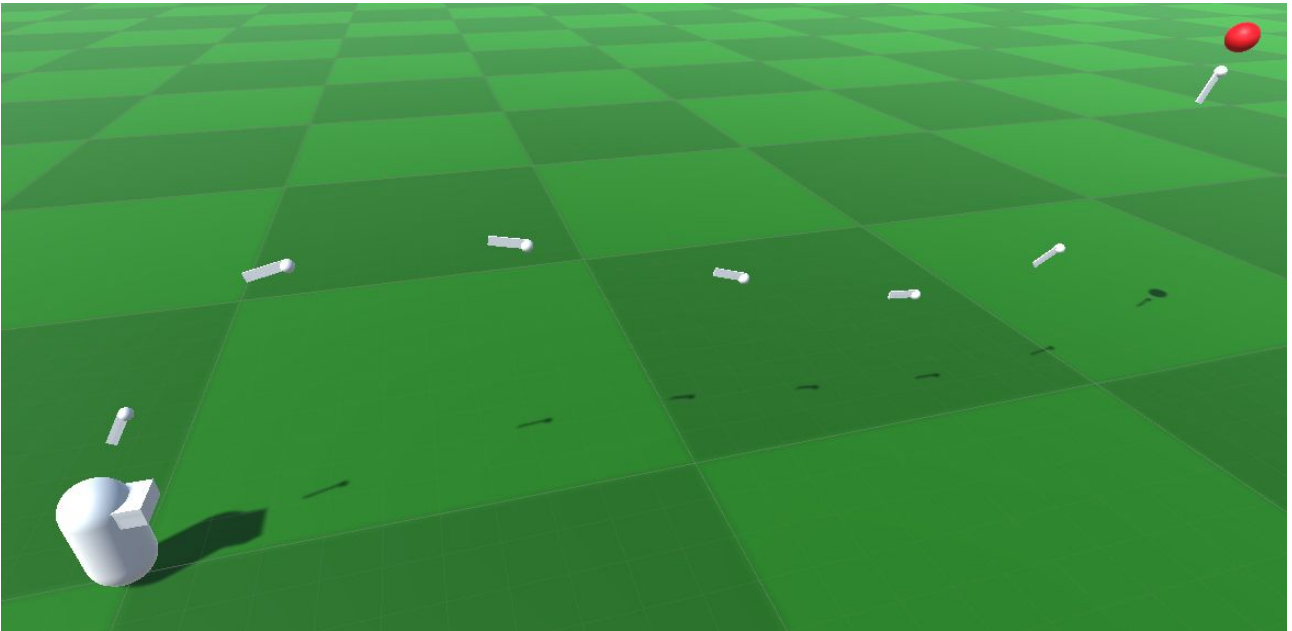
You can replace it with your own if you want. To do that implement the `IProjectileLineRenderer` interface and then use that instead of the `ProjectileLineRenderer`.

```

AnimationProjectile.DrawPath<YourRendererType>(source, target, config, prefab);

```

## Animation Projectiles



With the animation API (AnimationProjectile) you can spawn a projectile and animate it towards a target.

**Animation**

Duration

Follow Target

Follow With Adaptive

Curve Wrap

**Easing**

Easing

**Animation Curves**

Use Curves

Curve X

Curve Tile X

Curve Scale X

Curve Y

Curve Tile Y

Curve Scale Y

Curve Z

Curve Tile Z

Curve Scale Z

▼ Start Angle

Mode

Constant

► Angle Over Duration

Start Up Axis X  Y  Z

**Collisions**

Stop On Collision

Animation projectiles support PHYSICS too. You can have your projectile animated but as soon as it hits another object it can stop and let the physics engine take over (velocity is preserved). See „Animation Projectiles With Physics“ for more details on how to set that up.

This is how the code to spawn a projectile looks like:

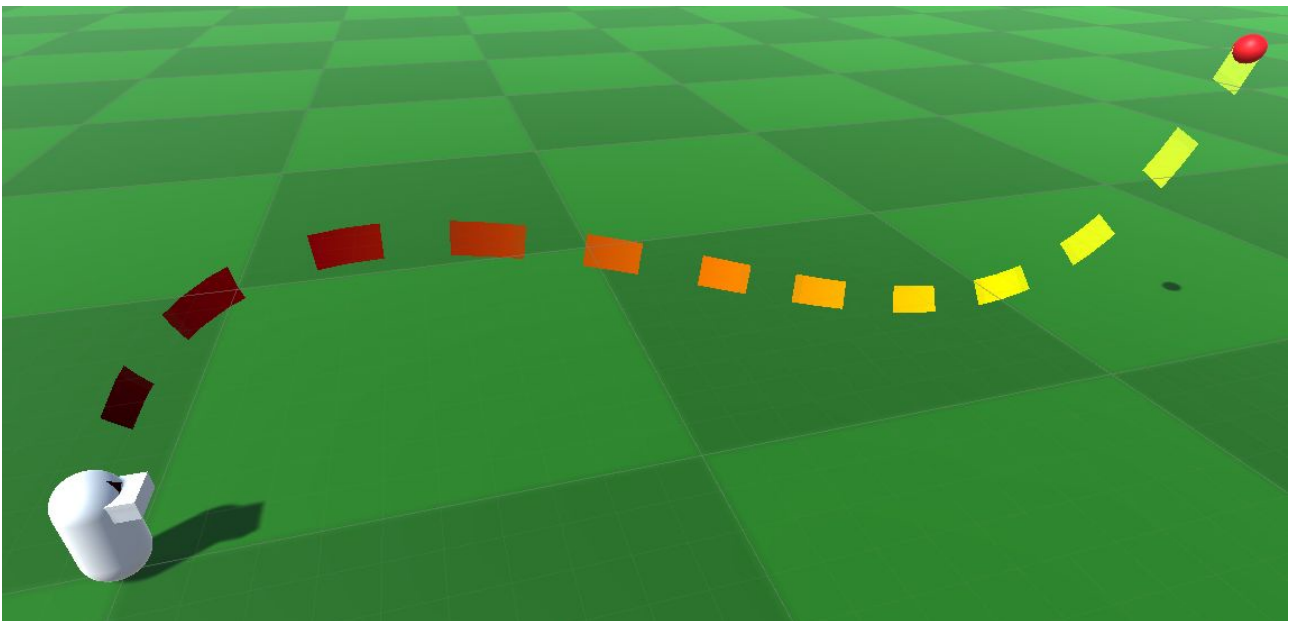
```
public class AnimationSimpleDemo3D : MonoBehaviour
{
    public Transform Source;
    public Transform Target;

    public GameObject ProjectilePrefab;
    public AnimationProjectileConfig Config;

    public void Start()
    {
        AnimationProjectile.Spawn(ProjectilePrefab, Source, Target, Config);
    }
}
```

## Animation Path Drawing

Easily draw the path of a projectile.

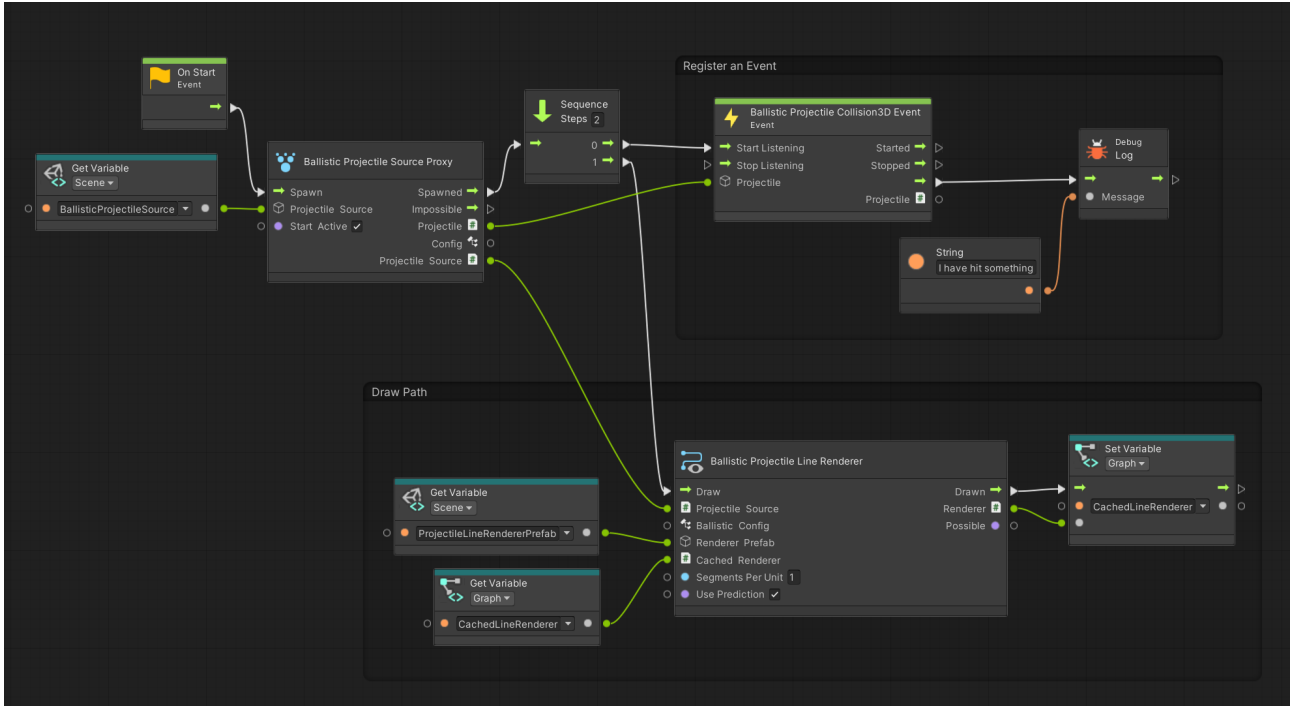


It uses the default Unity Line Renderer. You can replace it with your own if you want. To do that implement the `IProjectileLineRenderer` interface and then use that instead of the `ProjectileLineRenderer`.

```
AnimationProjectile.DrawPath<YourRendererType>(source, target, config, prefab);
```

# Visual Scripting

Spawn projectiles without writing a single line of code.

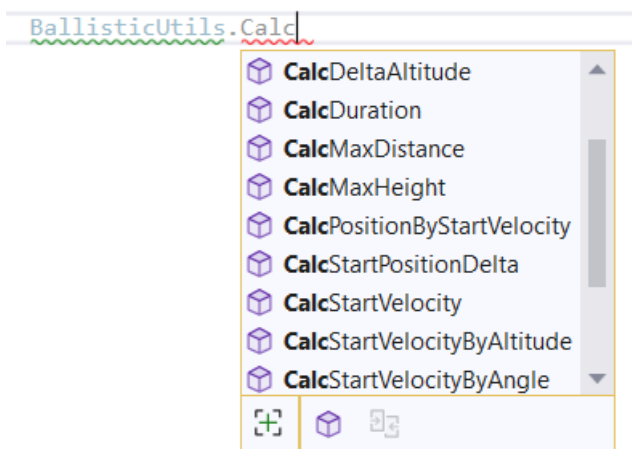


## 2D and 3D Support

It supports both 2D and 3D physics with one unified API.

## Lots of handy APIs

You need a custom solution that is not covered by the premade components? No problem! There is a low-level static API to help you out.



## Full Source Code

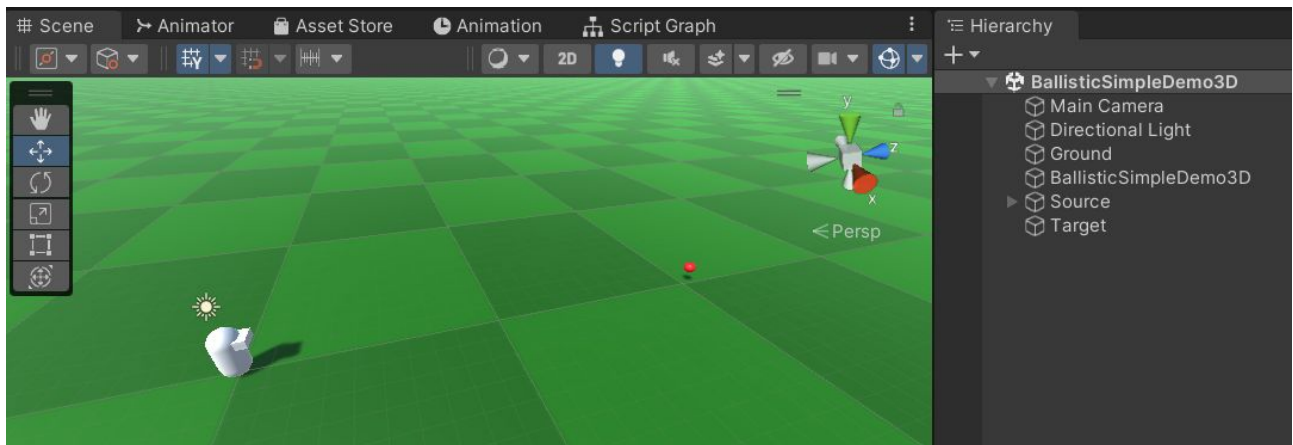
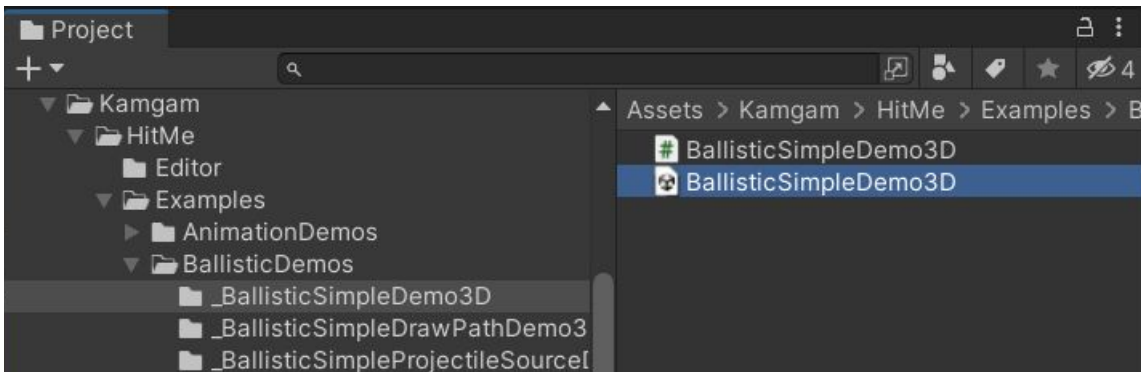
You can take a look and make changes if needed.



# Tutorial: Let's spawn our first projectile

This tutorial assumes you have opened the demo scene

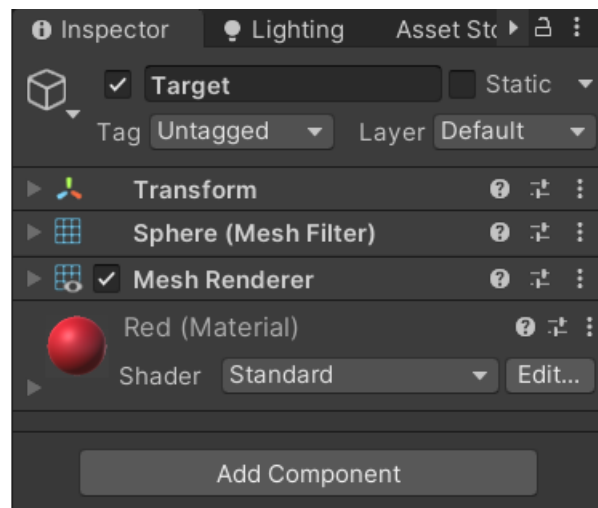
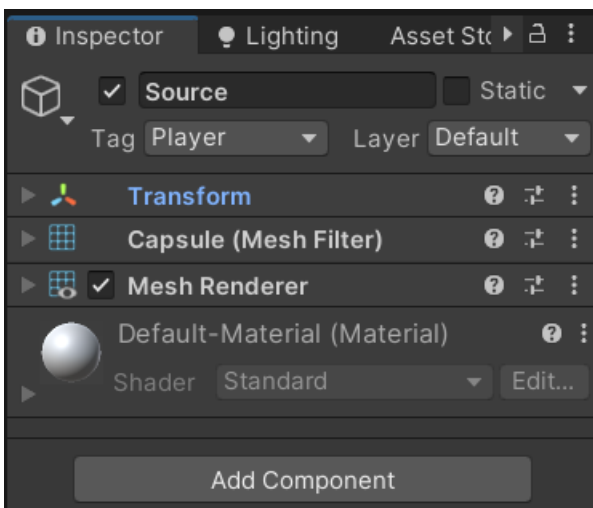
Assets/Kamgam/HitMe/Examples/BallisticDemos/\_BallisticSimpleDemo3D.



It contains a the camera, a light and a „Ground“ object with a mesh collider. The ground exists only to avoid things from falling down forever.

The interesting parts are „BallisiticSimpleDemo3D“, „Source“ and „Target“.

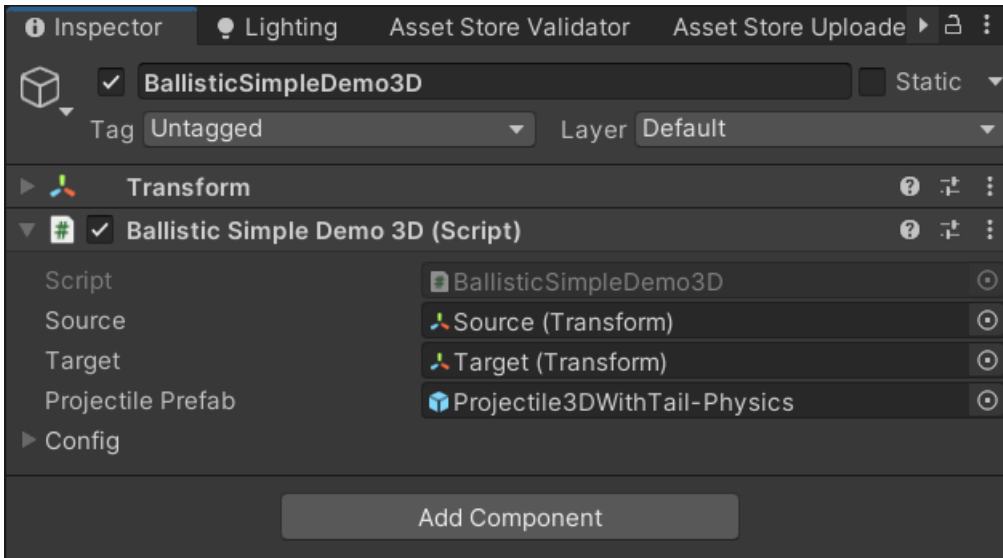
„Source“ and „Target“ are just two transforms that are used as the source position (where we will spawn the projectile) and the target position (where we want it to land).



Imagine that instead of „Source“ and „Target“ you use your character as source and an enemy as the target.

Now „BallisticSimpleDemo3D“ is where it gets serious.

We have to let our demo script know what source and target we want to use and what projectile (Projectile Prefab) we want to spawn.



The code itself is pretty basic. It waits for one second and then spawns one single projectile:

```
public class BallisticSimpleDemo3D : MonoBehaviour
{
    public Transform Source;
    public Transform Target;

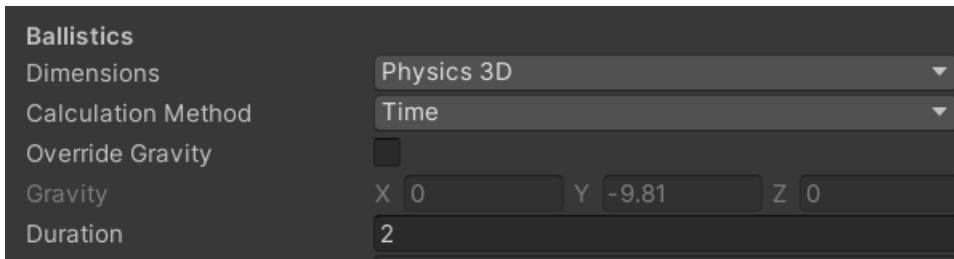
    public GameObject ProjectilePrefab;
    public BallisticProjectileConfig Config;

    public IEnumerator Start()
    {
        yield return new WaitForSeconds(1f);

        BallisticProjectile.Spawn(ProjectilePrefab, Source, Target, Config);
    }
}
```

If you unfold the „Config“ section you will see A LOT of configuration options. If you want to know more about what each option does please go to the „Ballistic Config“ section below.

All we are interested now in is this:

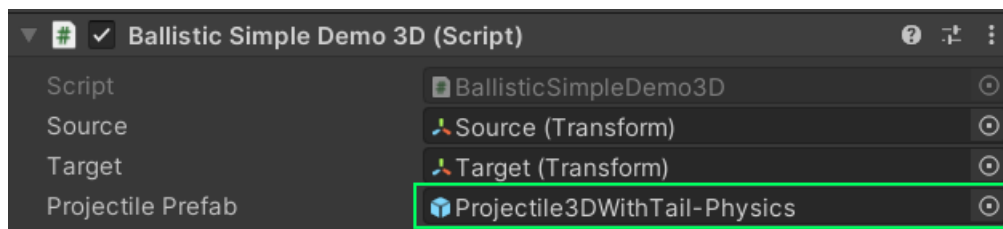


Here you can configure whether you are using 2D or 3D physics. In our case it's 3D.

The „Calculation Method“ is where the fun part begins. Right now it is set to use „Time“. That means the start velocity of the projectile is calculated in a way that ensures it will take exactly 2 seconds to reach the target. Why 2 seconds? Because that's what we have set in the „Duration“ option below.

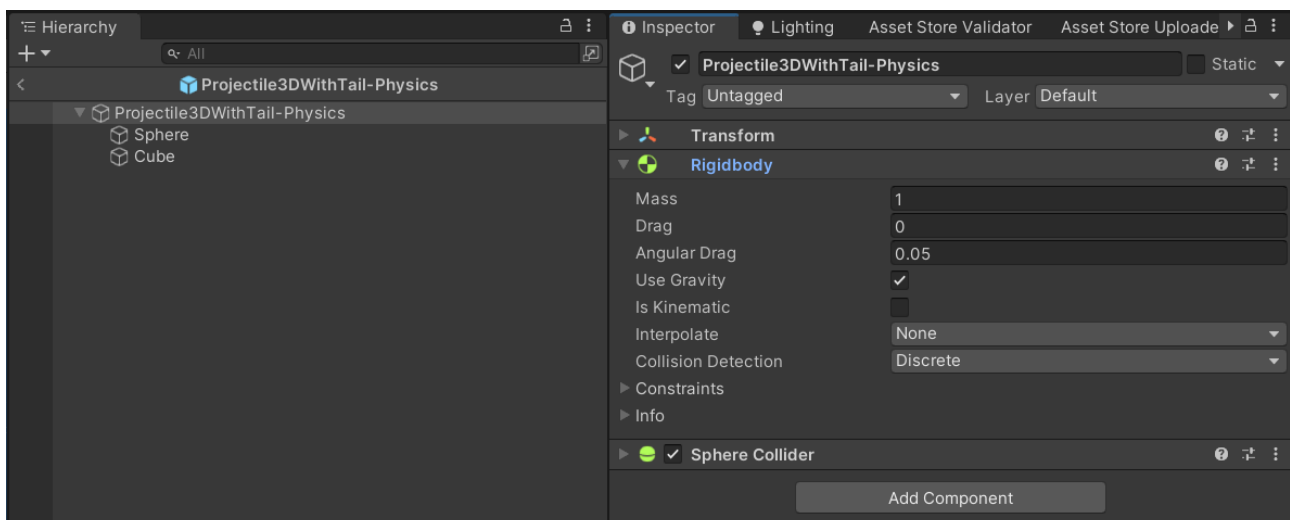
You may have noticed that most of the options are disabled. That's because they are not relevant for the „Time“ calculation method. If you change the calculation method then different options will be enabled. Give it a try!

One thing we have not yet taken a look at is the used prefab „Projectile3DWithTail-Physics“.



When we look at it we see there it is nothing special about it.

The only requirement for a **BallisticProjectile** is that it has a **Rigidbody that matches the Dimensions set in the config**. The ballistic projectiles are moved by the physics engine. If you use a prefab without a rigidbody then the projectile will be spawned but it won't move.



NOTICE: Depending on the calculation method and parameters you enter it may not be possible to reach the target. I.e.: shooting at a very distant target with a very low speed will not work. **If a target can not be reached the Spawn() method will return NULL.**

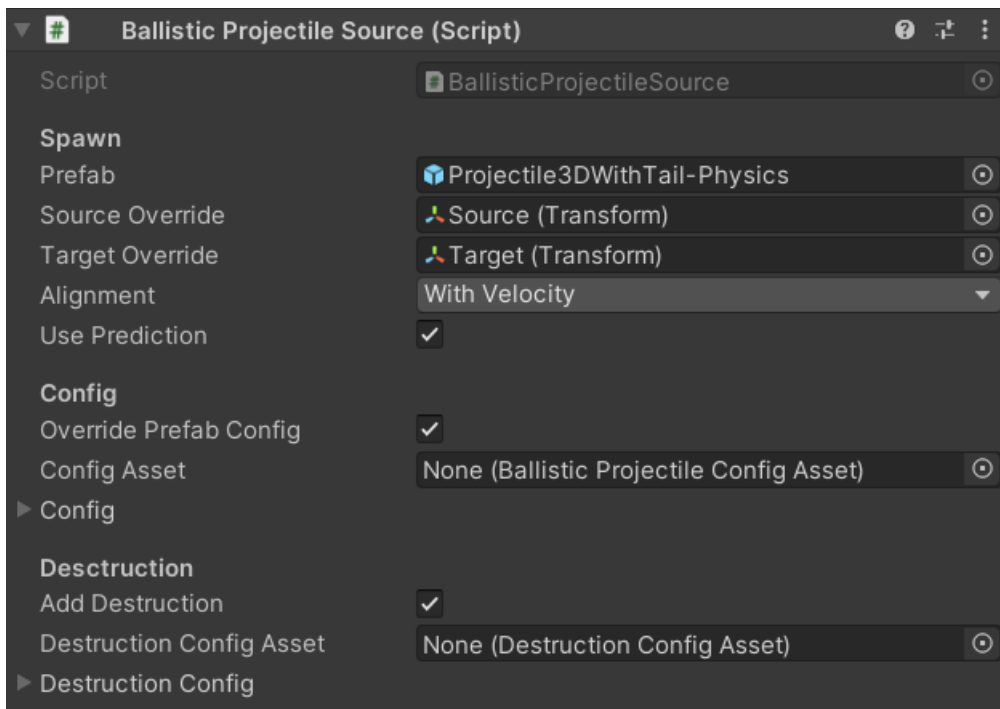
And that's it.

# Ballistics Projectile Source

While you can write your own script to spawn a projectile, as is shown in the „Simple“ examples, it is often much more convenient to use the included projectile source component.

It does cover some common requirements:

- Configure source, target and prefab.
- Load config from asset or prefab.
- Search for a movement predictor on the target and use it.
- Add alignment components to projectiles.
- Add destruction components to projectiles.



**Prefab:** The prefab that should be used to instantiate new projectiles.

**Source Override:** The source transform is used as the spawn position for new projectiles.

**Target Override:** The target transform is used as the target position where the projectile should land.

**Alignment:** One of multiple alignment options. These define how the projectile is oriented during flight. They are added as separate components to each projectile. You can find more on how they work in the „Alignment“ section below.

**Use Prediction:** If enabled then the „Target“ transform will be searched for a MovementPredictor component and if one is found it will be used to predict the target position.

**Override Prefab Config:** If enabled then the config will replace the config on the projectile prefab. Each projectile gets a COPY of the config. If the prefab has no config yet then it will be added (as a copy).

**NOTICE:** There are two ways of configuring a projectile. Either you add a Projectile component to your prefab and set a config there OR you enable the „Override Prefab Config“ option and have a copy of the config added to each new prefab instance. - Using the override is the recommended way.

**Config Asset:** You can reference a config from an asset. If set then this will override the local config.

**Config:** The local config object. If no asset is set then this config will be used for overrides.

**Add Destruction:** If enabled then a destruction component will be added to each projectile.

**Destruction Config Asset:** You can reference a destruction config from an asset. If set then this will override the local destruction config.

**Destruction Config:** The local destruction config object. If no asset is set then this config will be used.

# Ballistics Projectile Config

This configures all the values that are used to calculate the start velocity of the projectile.

<b>Projectile</b>			
Source Position	X	0	Y 0 Z 0
Source	None (Transform)		
Source Offset	X	0	Y 0 Z 0
Source Offset Space	Self		
Target Position	X	0	Y 0 Z 0
Target	None (Target)		
Target Offset	X	0	Y 0 Z 0
Target Offset Space	Self		
<b>Ballistics</b>			
Dimensions	Physics 3D		
Calculation Method	Time		
Override Gravity	<input type="checkbox"/>		
Gravity	X	0	Y -9.81 Z 0
Duration	2		
Altitude	1		
Altitude Base	Target		
Allow Lower Altitudes	<input checked="" type="checkbox"/>		
Angle	45		
Speed	20		
Speed Use High Solution	<input type="checkbox"/>		
Prediction Interaction Multiplier	1		
<b>Events</b>			
Collision Min Age	0.5		
Trigger First N Collisions	1		
Trigger Collisions After End	<input type="checkbox"/>		

**Source Position:** The start position from where the projectile is spawned.

**Source:** The start transform. If null then the „Source Position“ will be used. If not null then it will override the „Source Position“.

**Source Offset:** The offset that should be added to the source position. Very useful to make the projectiles spawn at a position relative to a transform. I.e.: spawn it outside the character collider.

NOTICE: These can be overridden by the optional „source, target, ...“ parameters of the Spawn() method. For example the ProjectileSource component „SourceOverride“ uses them.

**Source Offset Space:** The space (self or world) that is used to transform the „Source Offset“. World space may be used even if it is set to „self“ IF there is no „Source Transform“.

**Target Position:** The target position where the projectile should land.

**Target:** The target transform. If null then the „Target Position“ will be used. If not null then it will override the „Target Position“.

**Target Offset:** The offset that should be added to the target position. Very useful to make the projectile hit at a position relative to a transform. I.e.: hit the head of an enemy instead of the body.

NOTICE: These can be overridden by the optional „source, target, ...“ parameters of the Spawn() method. For example the ProjectileSource component „TargetOverride“ uses them.

**Target Offset Space:** The space (self or world) which is used to transform the „Target Offset“. World space may be used even if it is set to „self“ IF there is no „Target Transform“.

**Dimensions:** Whether to use 2D or 3D physics.

**Calculation Method:** The ballistic calculation method defining the input parameters for the start velocity calculation.

NOTICE: Depending on the calculation method and parameters you enter it may not be possible to reach the target. I.e.: shooting at a very distant target with a very low speed will not work.

**Override Gravity:** Enable to set a custom gravity.

**Gravity:** The gravity vector. Usually  $9.81 \text{ m/s}^2$  downwards.

**Duration:** The flight duration



**Altitude:** The max altitude of the projectile.

**Altitude Base:** Where the max altitude is measured from (in world units). „Highest“ means the highest value between „Source“ and „Target“ will be used. „Lowest“ means the lowest value between „Source“ and „Target“ will be used.

**Allow Lower Altitudes:** Allow lowering the altitude below the source height? If yes then a max altitude of 0.01 above the source is used for altitudes lower than the source.

**Angle:** The start direction angle upwards (around X axis in 3D, around Z axis in 2D).

**Speed:** The start speed. Useful if you want to match the speed of a rigidbody.

**Speed Use High Solution:** Using the speed as input parameter gives two solutions. One with a trajectory that has a high altitude and is curved a lot and one with a more direct strait path.

**Prediction Iteration Multiplier:** Use with care. Leave at 1 if possible. Higher values increase accuracy but lower performance.

**Collision Min Age:** This is for event handling only. It has not effect on actual collisions. It defines for how long the projectile should ignore collision events after spawning. Useful if you spawn from a source with lots of triggers.

**Trigger First N Collisions:** Usually you are only interested in the very first collision. However, with this you can set it to trigger for the first N collisions.

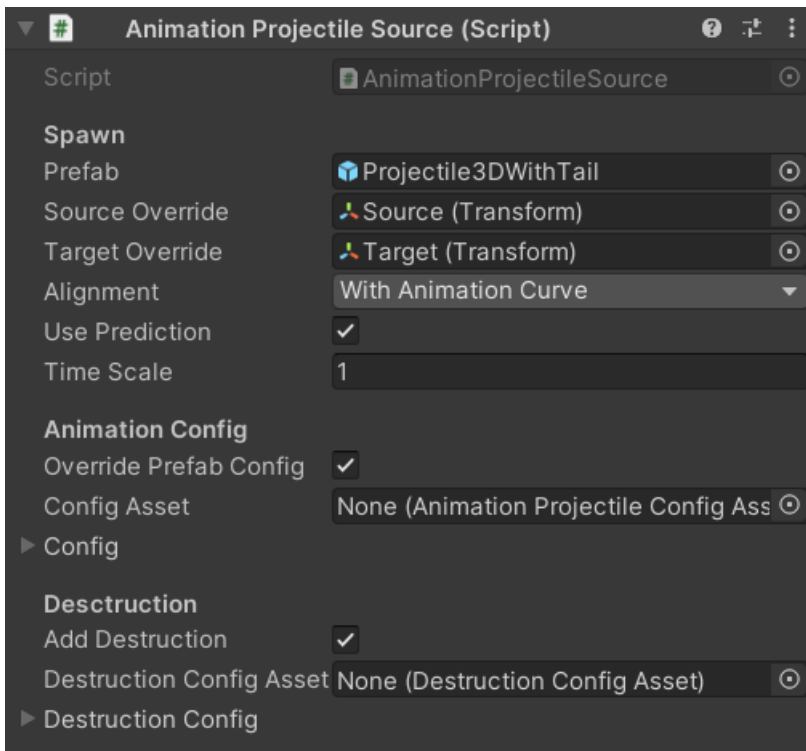
**Trigger Collisions After End:** Should any collision events be triggered after the flight has ended?

# Animation Projectile Source

While you can write your own script to spawn a projectile, as is shown in the „Simple“ examples, it is often much more convenient to use the included projectile source components.

It does cover some common requirements:

- Configure source, target and prefab.
- Load config from asset or prefab.
- Search for a movement predictor on the target and use it.
- Add alignment components to projectiles.
- Add destruction components to projectiles.



**Prefab:** The prefab that should be used to instantiate new projectiles.

**Source Override:** The source transform is used as the spawn position for new projectiles.

**Target Override:** The target transform is used as the target position where the projectile should land.

**Alignment:** One of multiple alignment options. These define how the projectile is oriented during flight. They are added as separate components to each projectile object.

**Use Prediction:** If enabled then the „Target“ transform will be searched for a MovementPredictor component and if one is found it will be used to predict the target position.

**Time Scale:** Use this to change the time scale of the animation independently of Unity time.

**Override Prefab Config:** If enabled then the config will replace the config on the projectile prefab. Each projectile gets a COPY of the config. If the prefab has no config yet then it will be added (as a copy).

**NOTICE:** There are two ways of configuring a projectile. Either you add a Projectile component to your prefab and set a config there OR you enable the „Override Prefab Config“ option and have a copy of the config added to each new prefab instance. - Using the override is the recommended way.

**Config Asset:** You can reference a config from an asset. If set then this will override the local config.

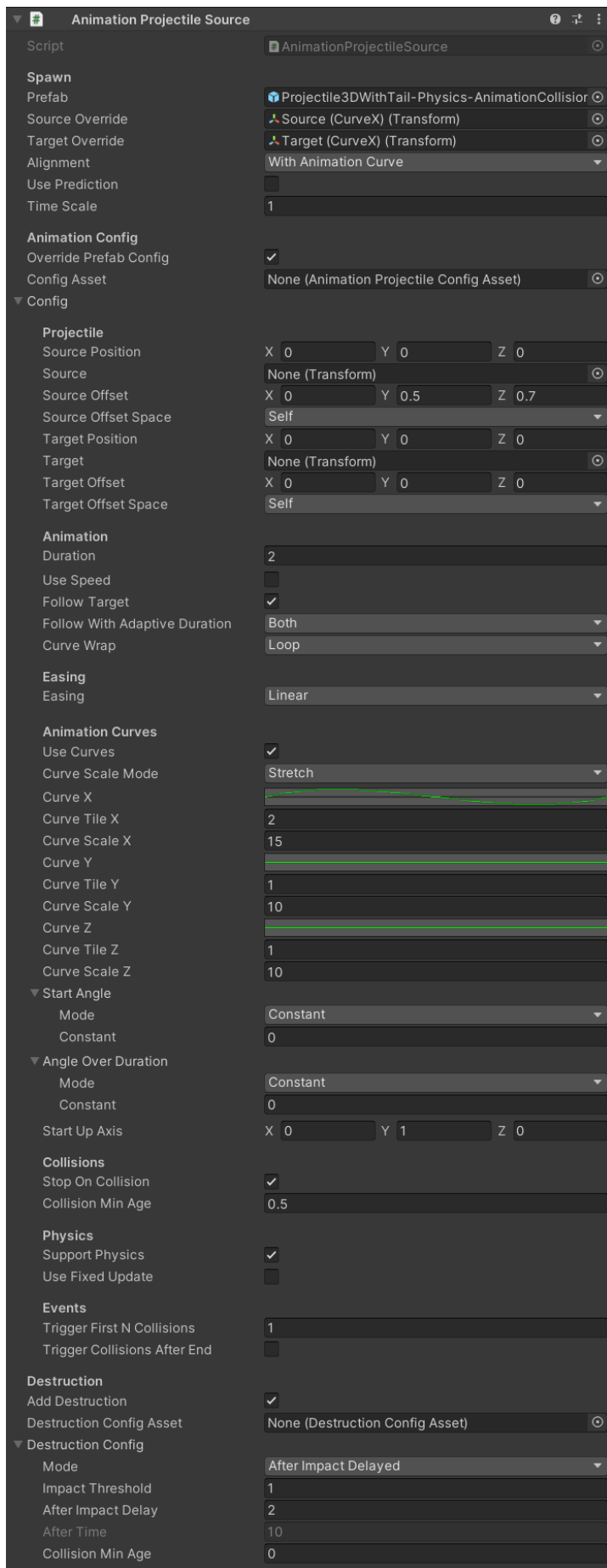
**Config:** The local config object. If no asset is set then this config will be used for overrides.

**Add Destruction:** If enabled then a destruction component will be added to each projectile.

**Destruction Config Asset:** You can reference a destruction config from an asset. If set then this will override the local destruction config.

**Destruction Config:** The local destruction config object. If no asset is set then this config will be used.

# Animation Projectile Config



This configures all the values that are used to calculate the start velocity of the projectile.

**Source Position:** The start position from where the projectile is spawned.

**Source:** The start transform. If null then the „Source Position“ will be used. If not null then it will override the „Source Position“.

**Source Offset:** The offset that should be added to the source position. Very useful to make the projectiles spawn at a position relative to a transform. I.e.: spawn it outside the character collider.

NOTICE: These can be overridden by the optional „source“ and „target“ parameters of the Spawn() method. The ProjectileSource component „SourceOverride“ uses these overrides for example.

**Source Offset Space:** The space (self or world) which is used to transform the „Source Offset“. World space may be used even if it is set to „self“ IF there is no „Source Transform“.

**Target Position:** The target position where the projectile should land.

**Target:** The target transform. If null then the „Target Position“ will be used. If not null then it will override the „Target Position“.

**Target Offset:** The offset that should be added to the target position. Very useful to make the projectiles hit at a position relative to a transform. I.e.: hit the head of an enemy instead of the body.

NOTICE: These can be overridden by the optional „source“ and „target“ parameters of the Spawn() method. The ProjectileSource component „TargetOverride“ uses these overrides for example.

**Target Offset Space:** The space (self or world) which is used to transform the „Target Offset“. World space may be used even if it is set to „self“ IF there is no „Target Transform“.

**Duration:** How long it should take to reach the target in seconds.

**Use Speed:** If enabled then the duration is based on the distance between the source and the target and the 'Speed' parameter.

Formula: duration = linear distance between source and target / Speed

NOTICE: the animation is always duration based (meaning easing will still work even if speed is used to calculate the duration).

HINT: You may want to disable 'Follow With Adaptive Duration' if this is enabled.

**Follow Source:** Enabled by default. If disabled then the animation curve will act as if the source was a fixed position in world space (i.e. it will not follow the source transform).

NOTICE: This behaviour can also be achieved by simply passing in a „source position“ and keeping the „source transform“ null.

**Follow Target:** Should the animation curve stretch to follow the target? If not then the target position will remain the same once set. If true then the projectile will follow with the curve and hit the target even if it is moving.

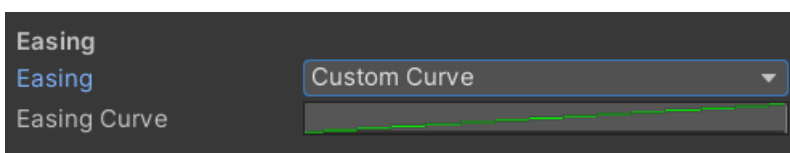
If enabled then the prediction can not be used (predictions will be ignored).

**Follow With Adaptive Duration:** Defines the shrink and expand behaviour of the curve if „Follow Target“ is on. Should the positions of already existing projectiles be dragged along with the new curve or should they remain at their current position (relative to normalized time)?

It's hard to describe. Best you just try out the values and move the target around to see how this works.

**Curve Wrap:** How the animation curves should be wrapped. Usually that's not important. However if an easing method is used that goes beyond the (normalized) time range then this is relevant as it defines how the position is calculated beyond the time range.

**Easing:** How the time will progress. This is similar to the easing you might know from tweening libraries like DoTween. You can even define a custom easing curve.



**Use curves:** This setting enables all the curve options.



All curves will be stretched along the path (source → target).

The „time“ of the curves is the normalized time of the animation duration (0 = start of animation, 1 = end of animation).

The value is the displacement on a single axis (X or Y or Z) in world units. X and Y are very useful. Z is a bit

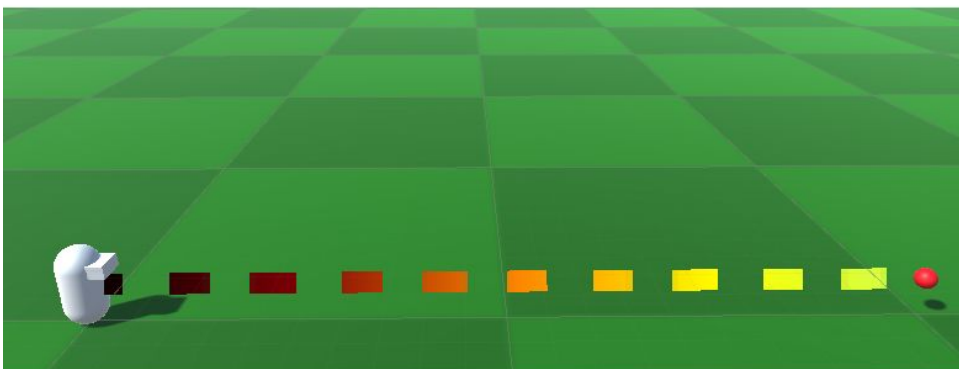
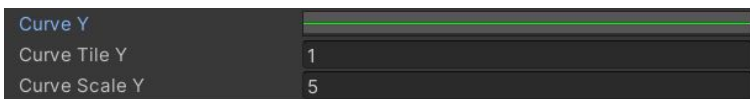
tricky to use, best leave it alone and use the easing controls instead.

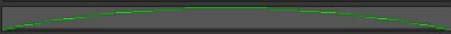
Tiling means how often the curve is repeated along the path.

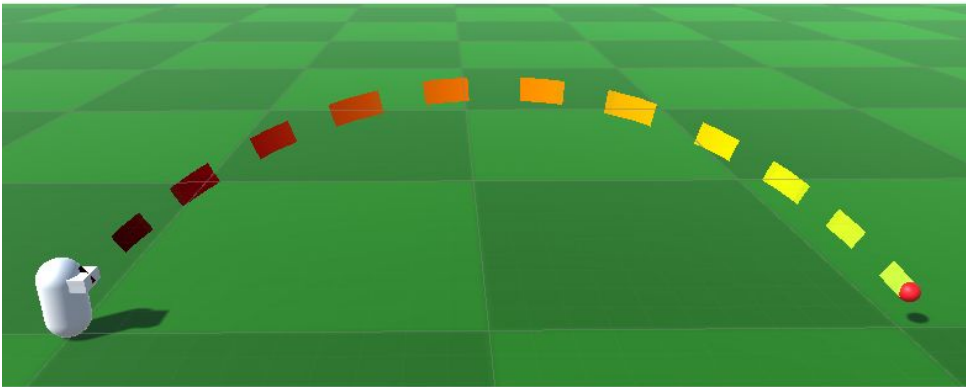
Scaling means how much the curve values (displacement along the up axis) are scaled. This way you can keep your curve values in a nice 0 to 1 range, yet scale them up as much as you need.


Each curve should start at (time: 0, value: 0) and end at (time:1, value: 0). If the value is not zero at the end then the target will be missed and it will look very odd if the curve is tiled.

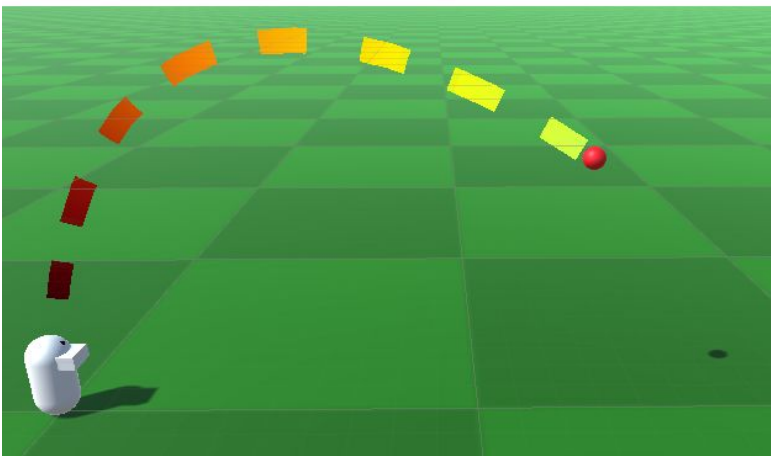
In the images below you can see the effect of a curve for the Y-axis. Notice how in the last example the overall curve changes if the path (source → target) changes, despite the animation curve staying the same. That's because the curve is applied relative to the path.



Curve Y	
Curve Tile Y	1
Curve Scale Y	5



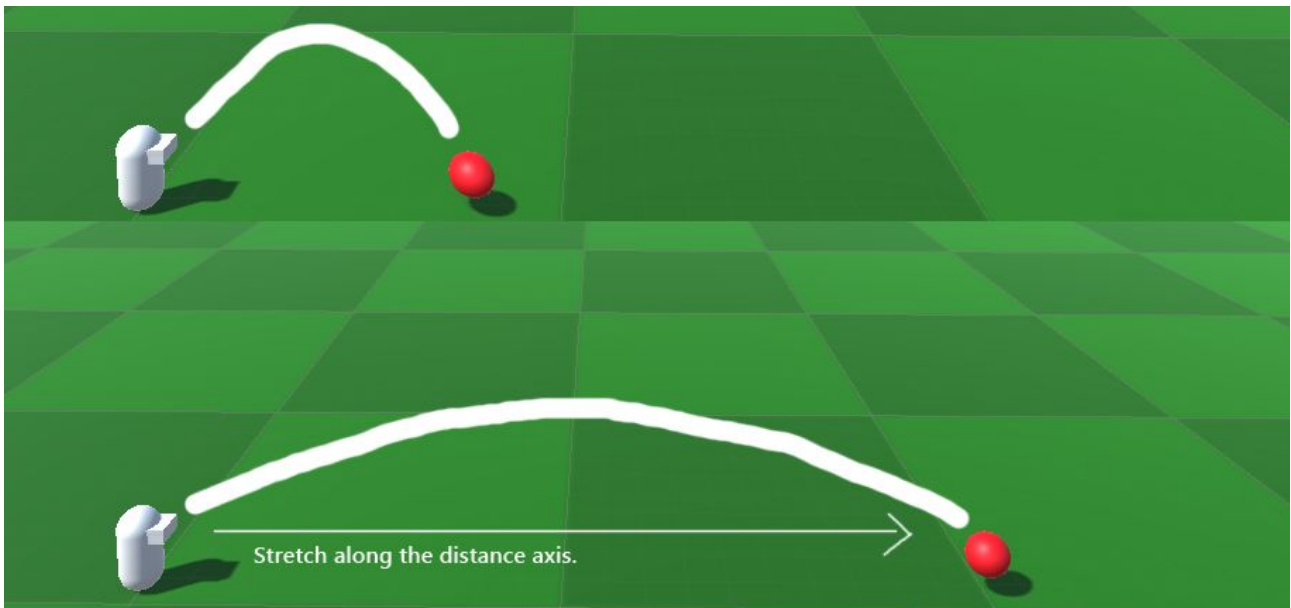
Curve Y	
Curve Tile Y	1
Curve Scale Y	5





**CurveScaleMode:** Defines how the curve will scale if the distance from source to target changes. You can check out the scale modes in the „AnimationCurveScaleDemo3D“ scene.

The default mode is Stretch:



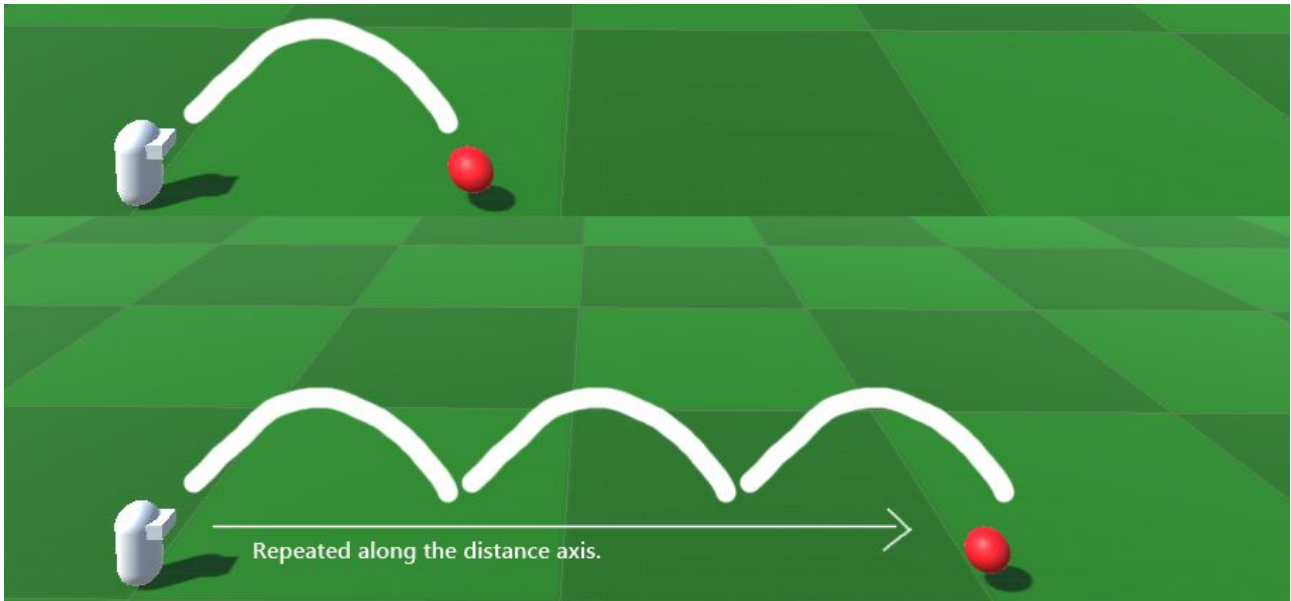
This means the curve is stretched along the distance.

The second mode is Scale:



In scale mode the curve will be scaled up based on the distance. The basis for the scale multiplies calculation is the „Curve Scale Reference Distance“ (see below).

The third mode is Repeat:



In repeat mode the curve will be repeated along the distance. The basis for the repetition multiplies calculation is the „Curve Scale Reference Distance“ (see below).

**Curve Scale Reference Distance:** The reference distance that is used to calculate the scale factor of the curve if CurveScaleMode is scale or repeat.

HINT: If you set it to a value below zero (-1 for example) then it will initialize with the current distance between source and target.

**Curve X:** The curve for the X-axis.

**Curve Tile X:** The curve tiling for the X-axis (along the path, aka time).

**Curve Scale X:** The curve scaling X-axis normal to the path and the „Start Up Axis“ (using value).

**Curve Y:** The curve for the Y-axis.

**Curve Tile Y:** The curve tiling for the Y-axis (along the path, aka time).

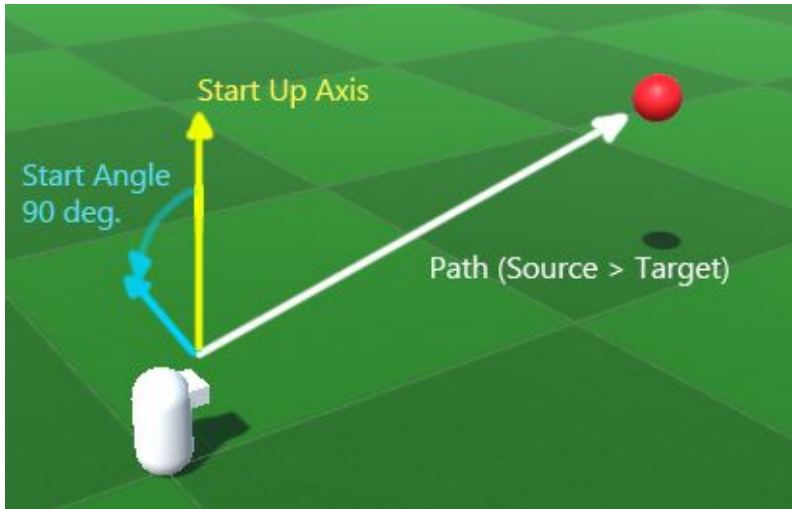
**Curve Scale Y:** The curve scaling Y-axis normal to the path along the „Start Up Axis“ (using value).

**Curve Z:** The curve for the Z-axis.

**Curve Tile Z:** The curve tiling for the Z-axis (along the path, aka time).

**Curve Scale Z:** The curve scaling Z-axis along the path (using value).

**Start Angle:** The start angle rotates the „Start Up Axis“ and with it the curves. It supports multiple modes (constant, random, curve). The rotation of the up axis is only done once, when the animation starts. The angle is in degrees.



**Angle Over Duration:** Rotates the up axis over time. The angle is in degrees. For example: If you set it to 180 then at the end of the animation the up axis will point downwards.

NOTICE: This is not degrees per second, it's the total rotation spread across the total duration.

**Start Up Axis:** Defines where the local Y-axis points to at the start. From this all the angles are calculated.

**Stop On Collision:** Should the animation stop if another object is hit?

To use this you have to add a „CollisionTrigger“ component to your projectile prefab.

If physics support is on then before stopping the current velocity will be applied to the rigidbody to ensure consistent movement after the animation.

**Support Physics:** If enabled then the rigidbody of the projectile is set to `isKinematic = true` and `useGravity = false` while animating.

To use this you should add a „CollisionTrigger“ component to your projectile prefab. More on that in the „Animation Projectiles With Physics“ section below.

If disabled then the projectile logic will ignore all physics components (rigidbodies, colliders) on the projectile and just animate the transform. If you disable it and your objects do have rigidbodies then this may lead to undefined behaviour (transform changes fighting physics simulation).

The object will use infinite force to move other objects. You probably do not want that. Enable 'StopOnCollision' to avoid that.

**Use Fixed Update:** If enabled then the object will be moved by the animation in every FixedUpdate step via Rigidbody.MovePosition().

FixedUpdate is not necessary in most cases. The recommended approach is to disable this option and enable StopOnCollision in combination with collision triggers instead.

Disabling FixedUpdate saves a bit in performance as not FixedUpdate method will be used.

**Collision Min Age:** Sometimes it is useful to ignore collisions for the first N seconds after spawning.

For example to avoid self colliding with the character or nearby objects.

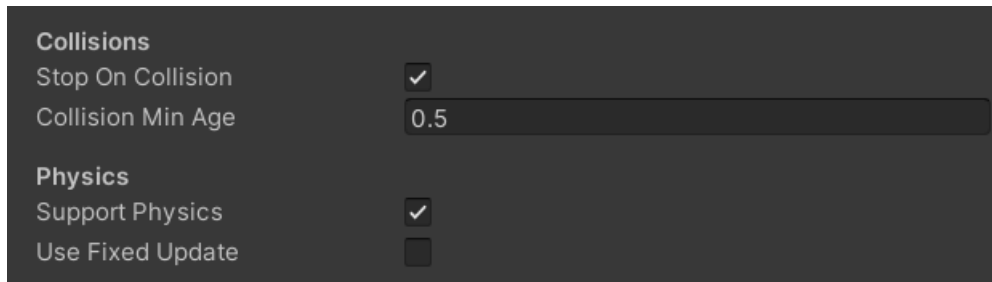
NOTICE: This is only for EVENTS. It does not prohibit collisions. It just does not report them as an event.

**Trigger First N Collisions:** Usually you are only interested in the very first collision. However, with this you can set it to trigger for the first N collisions (for example to spawn impact particles).

**Trigger Collisions After End:** Should any collision events be triggered after the flight has ended?

# Animation Projectiles With Physics

If you want to animate projectiles that have a rigidbody on them then you can do that by enabling physics support in the config. These are the recommended settings:



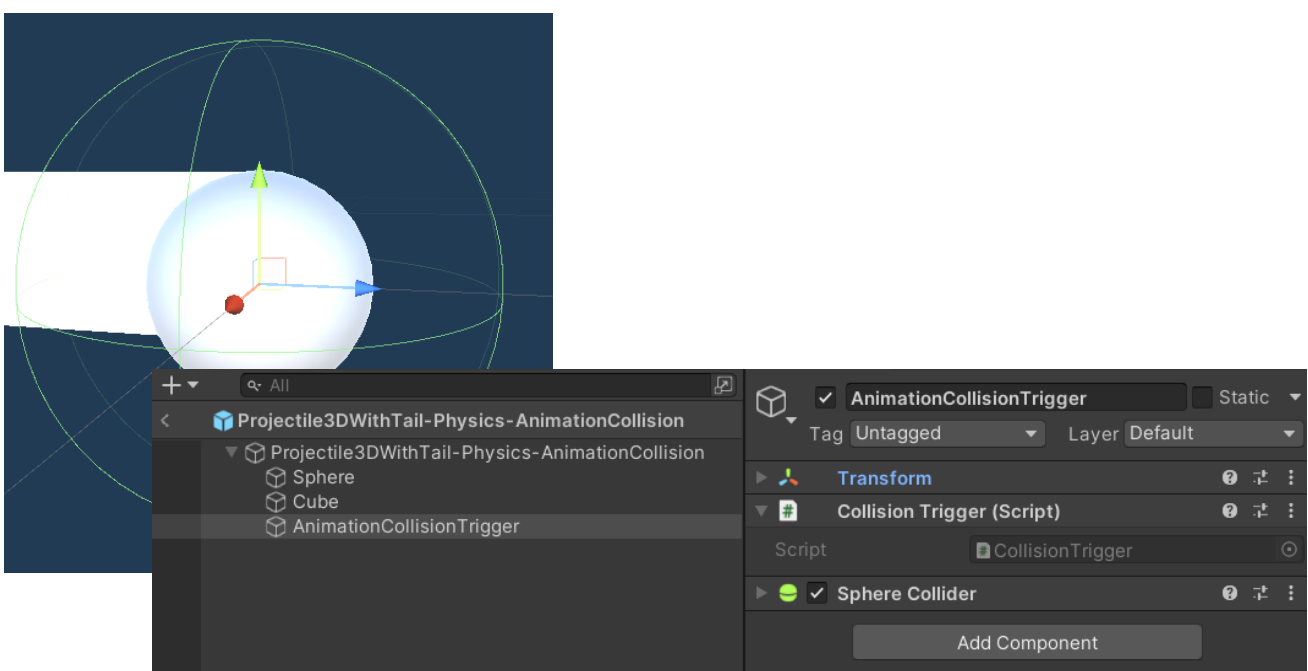
The rigidbody of the projectile is set to `isKinematic = true` and `useGravity = false` while animating.

I'd advise you to add a „CollisionTrigger“ component to your animated projectile if you are using physics. The reason is that for the physics to continue smoothly after the animation the velocity of the animation has to be copied to the physics object BEFORE any collisions happen, or else the velocity copy process would interfere with physics simulation.

The best approach is to release the control of the projectile shortly BEFORE it will hit anything and apply the velocity there. Once the physics simulation takes over it will all go smoothly. To do this we add a collision trigger to the projectile to detect these „soon to happen“ impacts.

The „Projectile3DWithTail-Physics-AnimationCollision“ is a prefab that has been prepared for this exact use case.

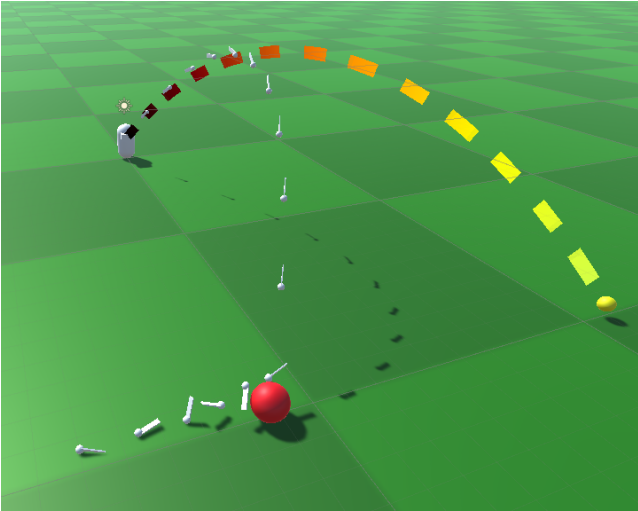
If you open it you will notice that the trigger is quite a bit bigger than the actual collider. That's to ensure the trigger „triggers“ before any collisions will happen.



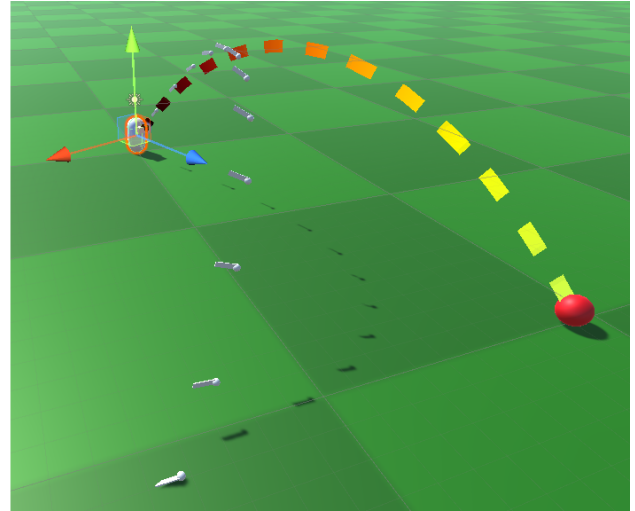
# Target Movement Prediction

Target movement prediction is done by a component on the target. It allows the projectiles to aim not at where the target is now but instead where it will be in the future.

With Prediction



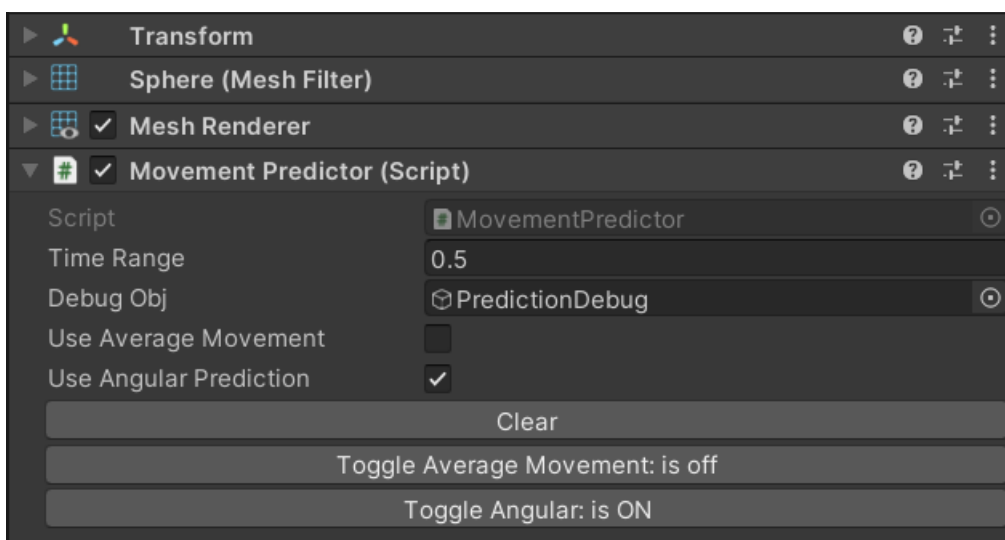
Without Prediction



Predictions are not 100% accurate. Sometimes it is even impossible to predict future positions (i.e. a player controller character). However, for scripted objects that only move in circles or straight (unaccelerated) lines it is a good approximation.

The prediction component is completely optional and not required for the system to work. I just thought it might be a nice addition.

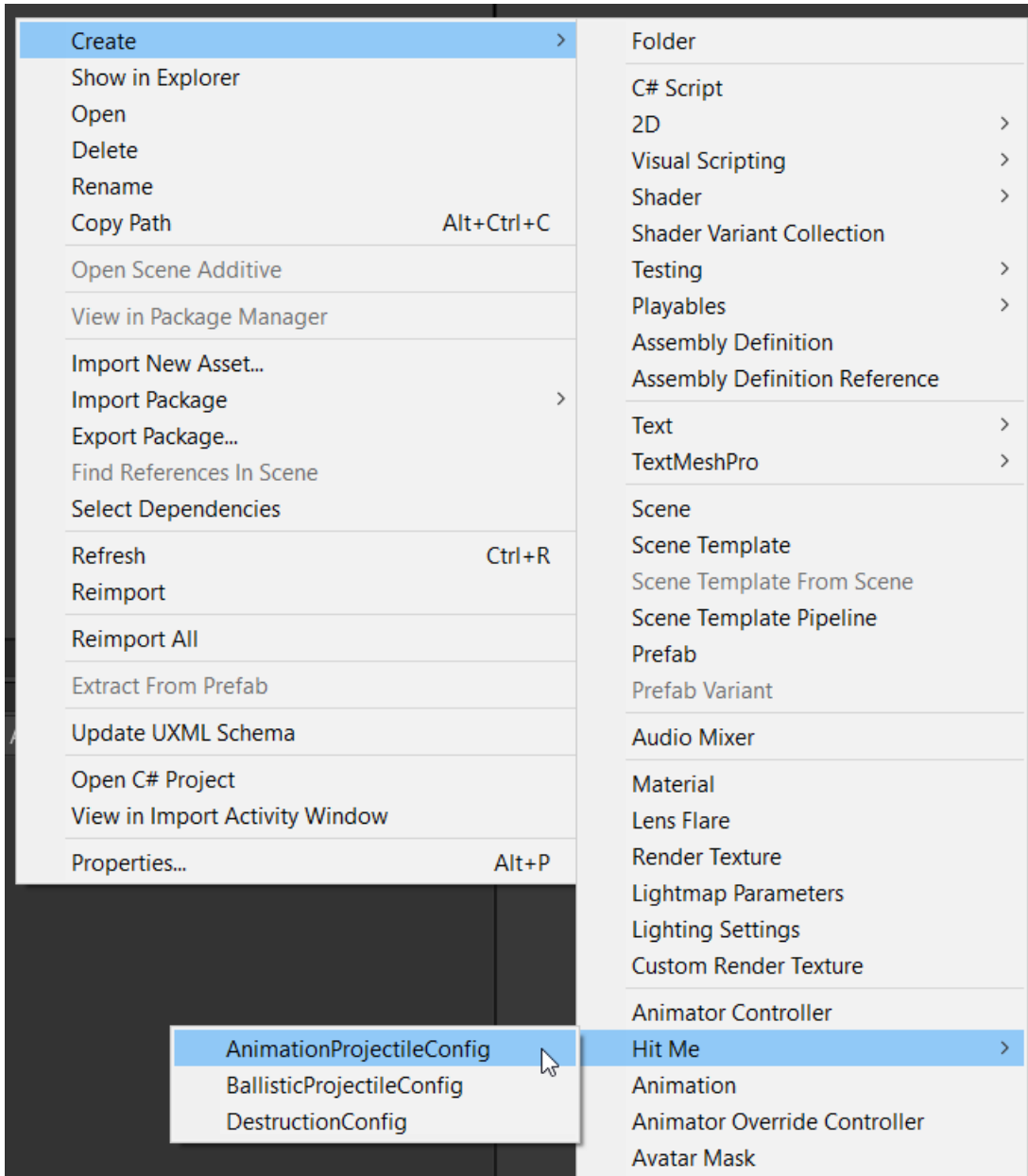
To use it simply add the „MovementPredictor“ component to your game object. Like this:



The MovementPredictor implements the **IMovementPredictor** interface so you can create your very own predictor if needed.

# Config Assets

Config assets are scriptable objects. They contain the same information as normal configurations. They are a asset files and can be easily reused in multiple locations. Use them to avoid copying your config values.

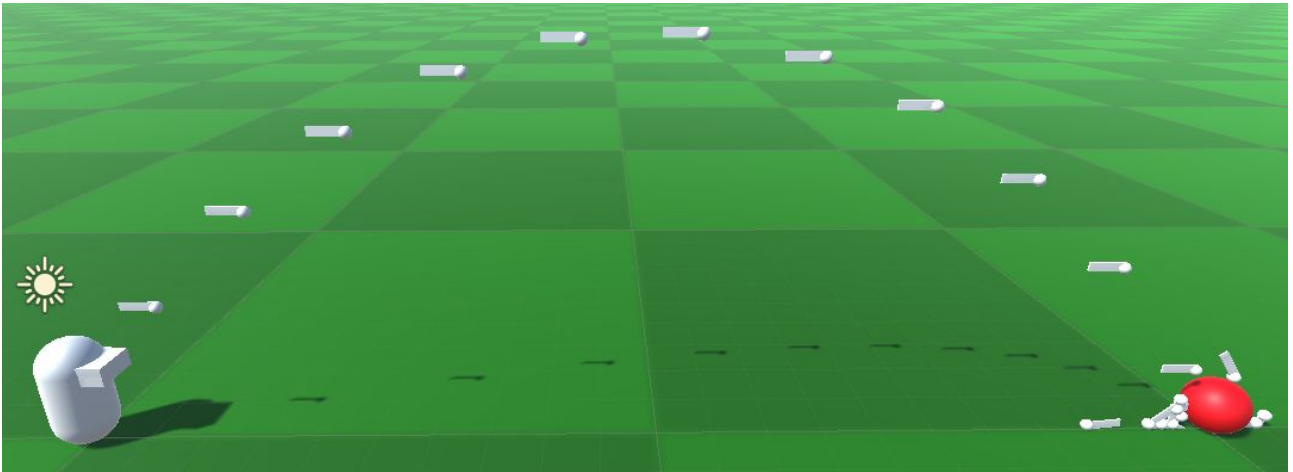


## Alignment Components

These components are added directly to a projectile game object. They continuously update the rotation of the transform.

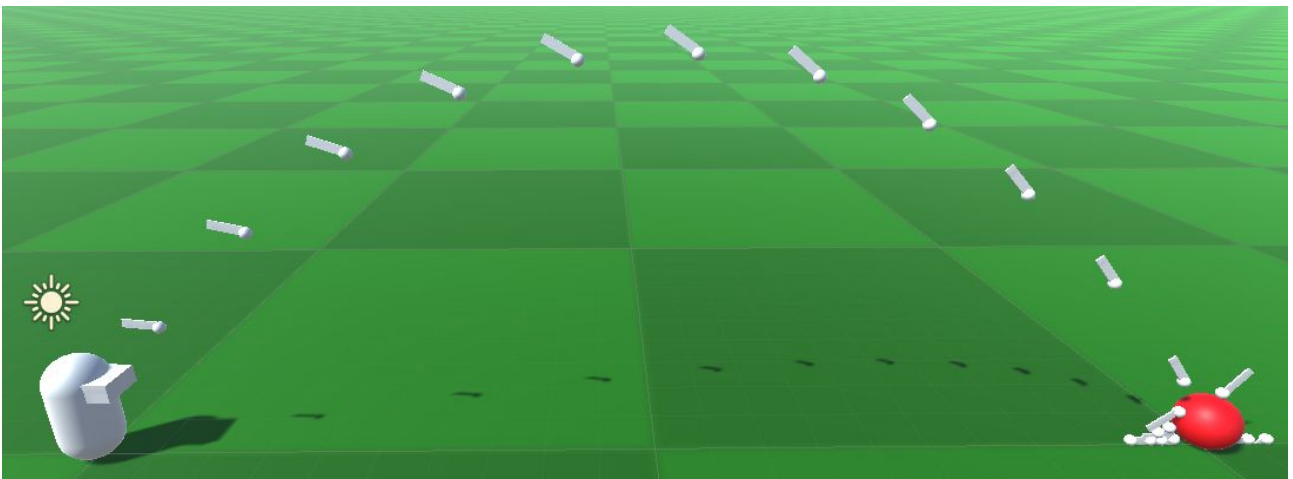
### None

No alignment means the projectiles will not be aligned (AnimationProjectiles) OR will be controlled fully by the physics (BallisticProjectile).



### LookAt

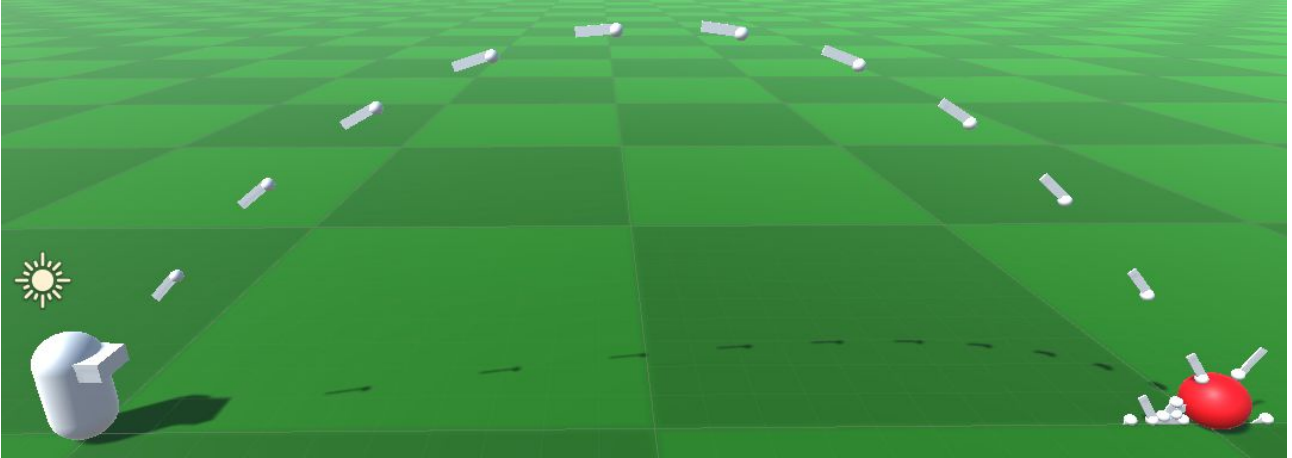
Makes the projectiles look at the target position.





## AlignWithVelocity

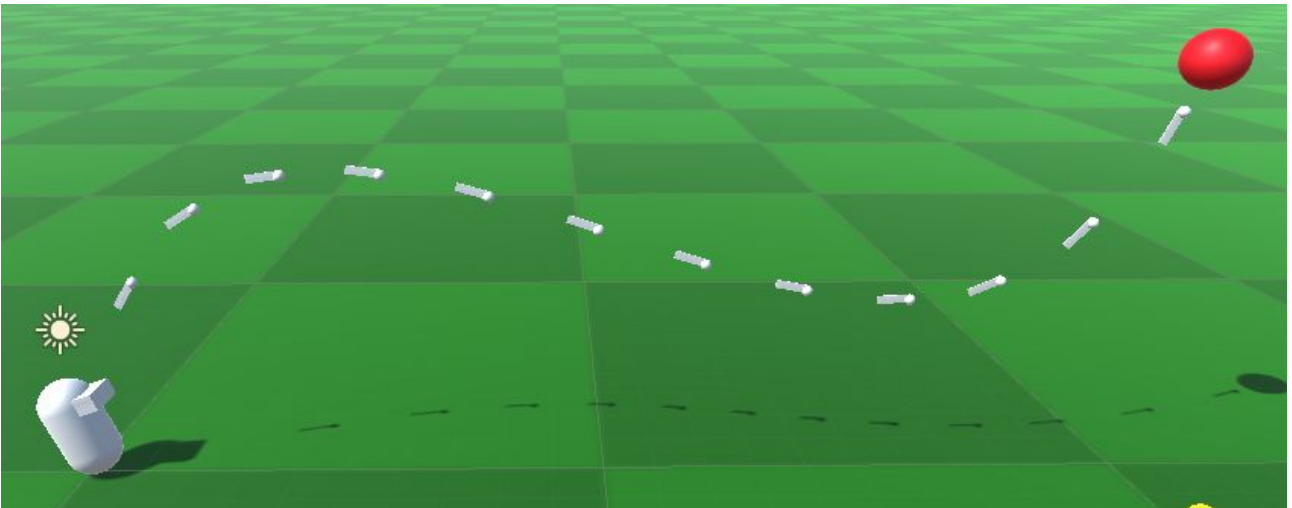
Aligns the projectiles along their current velocity direction.



## AlignWithAnimationVelocity

This is **only available for AnimationProjectiles**. It aligns the projectile with the velocity vector of the animation curve. Most of the times this gives the same result as „AlignWithVelocity“.

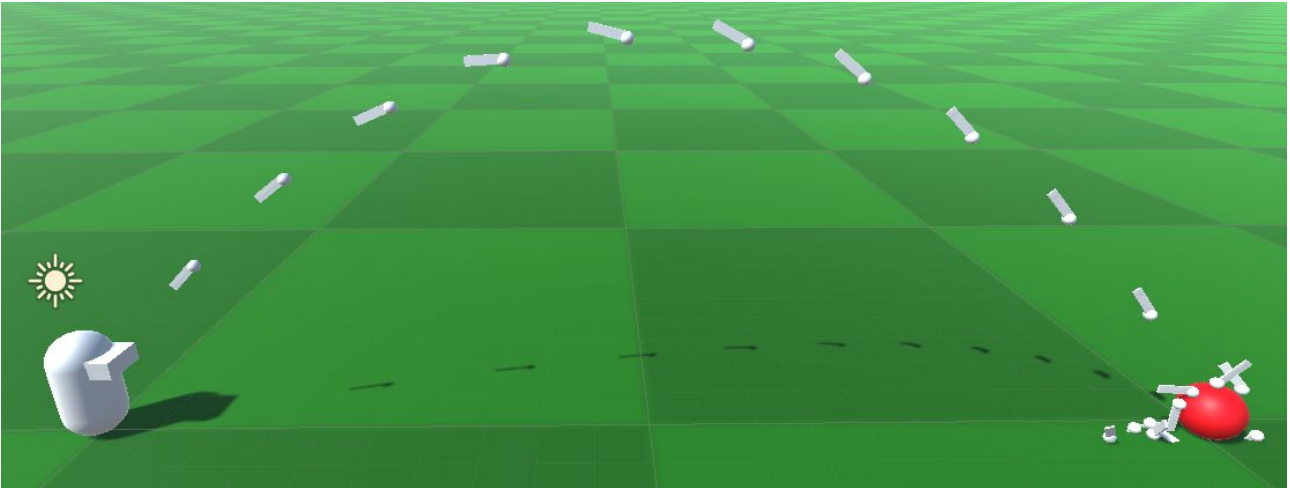
However, if the target is moving then the curve velocity direction can differ from the projectile velocity direction.



## Mixing Alignments (AlignLookLerp)

This is part of the „BallisticDemo3D“ and it mixes two alignments based on a timer. In the beginning the projectiles are aligned with the velocity and later they transition into looking at the target (you can see the change happening in the middle).

The custom alignment lerp is a demo of the „Apply“ property of the alignment components. If set to false then the alignment rotation is calculated but not applied. You can then read it via the „Result“ property, modify it and apply it later.

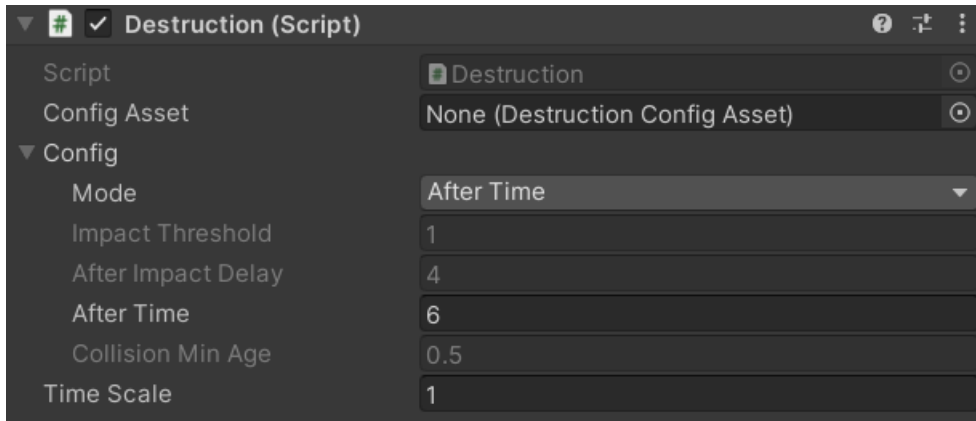


HINT: Most of the alignment components have no connection to any projectile code. They are completely independent and thus can be used for other objects too ;-)

# Destruction

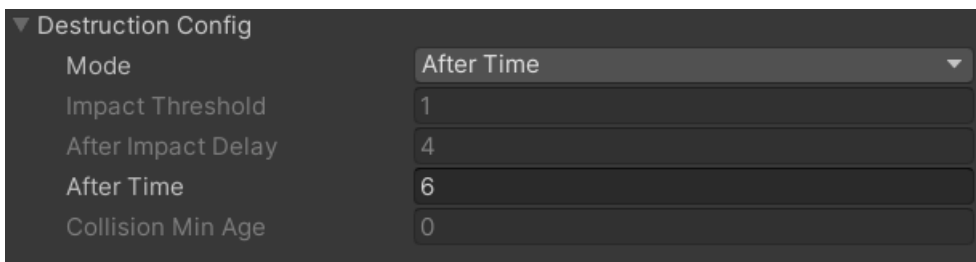
## Destruction Component

The destruction component is added to the projectile game object. It has some trigger conditions stored in a config object. Once these conditions are met it will destroy the game object.



## Destruction Config

Configures the conditions for a destruction.



**Mode:** What logic to use.

**Impact Threshold:** How many collisions does it take to trigger the „impact“ behaviour. Usually this value is 1.

**After Impact Delay:** Destroy N seconds after the impact threshold has been reached.

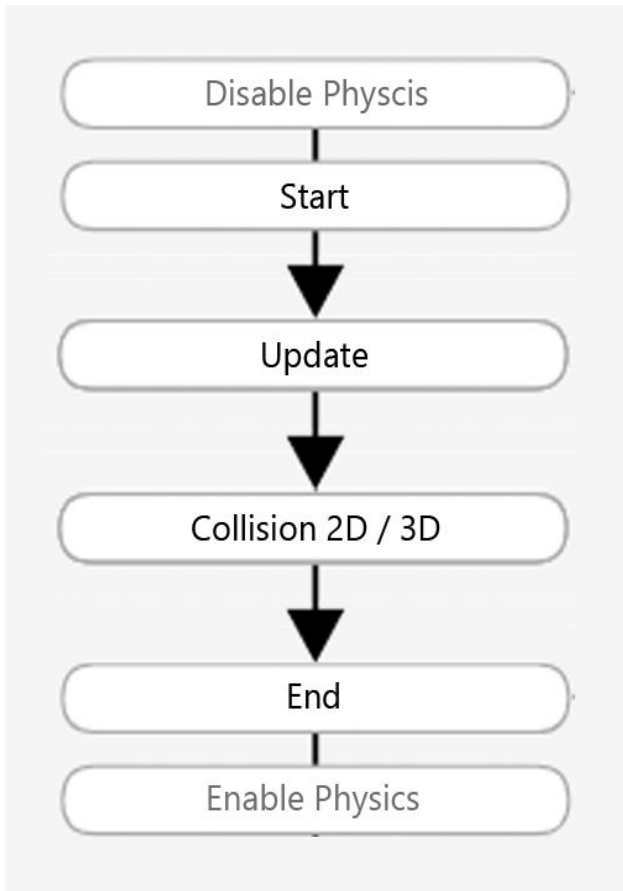
**After Time:** Destroy after a countdown has expired. Time is in (scaled) seconds.

**Collision Min Age:** This is for event handling only. It has not effect on actual collisions. It defines for how long the object should ignore collision events after Start().

HINT: The destruction component has no connection to any projectile code. It is completely independent and thus can be used for other objects too ;-)

## Events

Each projectile goes through a lifecycle of events. These are:



\*Disable and Enable Physics events are only called on AnimationProjectiles since only these disable and enable physics.

The end event is fired for BallisticProjectiles if the calculated flight time has elapsed OR if a collision occurred.

Update will be called with time = 0 and time = duration too, so you can use it to cover 100% of all possible states.

## Projectile Registry

Each projectile registers to the static „ProjectileRegistry“. You can use it to fetch spawned projectiles (use „ProjectileRegistry.Instance“ to access it).

Each projectile also has a `τ GetConfig<T>()` method allowing you to fetch the config.

## Frequently Asked Questions

Here are some common issues that have been reported.

If you can, please upgrade to the highest LTS version of Unity. The newer the version the better.

### My targets are very far away and I can not hit them accurately.

First, disable the `compensateSimulation` option.

```
// Animate the projectile.  
BallisticProjectile.Spawn(ProjectilePrefab, Source, Target, Config, compensateSimulation: false);  
▲ 2 of 4 ▼ BallisticProjectile BallisticProjectile.Spawn(GameObject prefab, Transform source, Transform target, [BallisticProjectileConfig config = null], [bool compensateSimulation = true],
```

And second, reduce the „Fixed Timestep“ value (to 0.005 for example):



Though be aware that this will decrease the performance. You should only do this as a last resort.

### I need to predict bouncing and reflections (not supported)

I am sorry, this is not supported. A common approach to solve this is to have another hidden scene in which you then run the physics simulation and record the results. Then you can use these results to generate a preview.

### I need to simulate air resistance and drag (not supported)

I am sorry, this is not supported. You will have to create your own ballistic methods for that. All predictions are for frictionless (dragless, airless) motion only.

### My animated projectile stops animating in mid air?

Probably two projectiles have collided and you have "StopOnCollision" turned on. You should disable projectiles collisions in the physics matrix.

### My animated projectile start falling directly from the source?

If they spawn on top of each other and 'Stop On Collision' is turned on then this is normal. Try disabling 'Stop On Collision' or increase the 'Collision Min Age' or remove the collider. This may also happen if you spawn them quickly and have some easing enabled that slows down movement in the beginning.

## **My animated projectiles stop or move weird in my 2D project.**

Notice that rigidbodies2D do NOT have a Z velocity. Therefore the Z velocity will always be 0 once the physics simulation takes over.

If your curve turns in the z direction then it may lead to unexpected results.

For example at the end the velocity will be copied from the animation to the rigidbody. Yet for 2D physics there is no Z velocity and thus this will always be  $z = 0$  at the end.

## **My ballistic projectiles do not fly in my 2D project, why?**

Check that in the config you have Ballistics > Dimensions set to "Physics 2D".

Check that you are not spawning them INSIDE a collider.

## **The predicted path does not match the flight path.**

Symptoms: The ballistic projectiles are at the wrong position right from the start (they are a bit too high or too low). They do not hit the target.

Maybe you are spawning them INSIDE another collider and that one pushes them outwards quickly altering the velocity and position in the process. Try adding a source offset to make them spawn outside any colliders.

NOTICE: The "Collision Min Age" is only for EVENTS. It does not prohibit collisions. It just does not report them as an event.

## **The ballistic projectiles do not move after spawning?**

You probably forgot to add a rigidbody or set the Config.Dimensions to match the type of your rigidbody (2D or 3D).

## **I need to spawn lots of projectiles. Is there some pooling?**

There is no read-made pooling class but using the BallisticUtils you can reuse objects without allocating any memory. Just be aware that resetting physics objects is a tricky business. Use the profiler to check if pooling is really needed. The memory footprint of each new projectile is rather small.

## **Changing the config does not affect already existing projectiles!**

When a projectile is spawned it gets handed a COPY of the source config. Therefore changes to the source config will not affect the existing projectiles.

Each projectile registers to the „ProjectileRegistry“. You can use it to fetch spawned projectiles (ProjectileRegistry.Instance.\*). Each projectile also has a `τ GetConfig<T>()` method allowing you to fetch the config.