Contents

API example use cases

Targeting

Projectile Toolkit provides various targeting algorithms to meet the needs of different scenarios, here are some example use cases:

Method	Example use case
VelocityByA	This method automatically adapts max height of the trajectory according to the distance to the target. Great for human-like throwing/jumping behavior, and projectile launch calculation in 3D top-down shooters.
VelocityByAngle	Launch projectiles to hit a target with a specific elevation angle.
VelocityByTime	Let archers to accurately hit moving targets.
VelocityByHeight	Use in animations, or achieve realistic Off-Mesh Link / NavMesh Link movement, or achieve jump pad mechanism.
AnglesBySpeed	Simulate weapons that has a specific launch speed, such as cannon and mortar.
VelocitiesBySpeed	An extended version of AnglesBySpeed. It is more convenient than AnglesBySpeed when the rotation is not separated into y axis and x axis, such as hand-held mortar and bow.
ElevationalReach	Display the maximum distance a weapon can attack.

Prediction

Method	Example use case
PositionAtTime	Implement anti-ballistic missile (use this method to predict the position of the hostile projectile after *x* seconds, and use VelocityByTime(..., *x*) to launch the anti-ballistic missile).
Positions	Predict the trajectory of a projectile. (You can use **TrajectoryPredictor** component instead, it has trajectory rendering implemented.)
VerticalFlightTest	Predict the flight time of a projectile when the x and z coordinates of the target are unkown, but the elevation of the target is known.
FlightTest	Test if a certain velocity will allow a projectile to hit the target.

How to use

> **Important**
>
> Add `using Blobcreate.ProjectileToolkit;` in your scripts to be able to call the APIs.

Targeting

To launch a projectile to hit a target:

1. Add a *Collider* and a *Rigidbody* to your prefab if it doesn't have one (you can set Interpolate to Interpolate and set Collision Detection to Continuous Dynamic to get the best result),
2. In your code, instantiate the prefab,
3. Call one of the targeting algorithms to calculate the launch velocity, and apply it using `AddForce(...)`.

The targeting algorithms are all static methods so the integration is very flexible.

Example code

Take `VelocityByTime(...)` for example, if you want AI units to accurately striking moving objects:

```
[SerializeField] Rigidbody projectilePrefab;
[SerializeField] Transform launchPoint;
[SerializeField] float timeOfFlight;
...
// In your own method:
var myRigid = Instantiate(projectilePrefab, launchPoint, launchPoint.rotation);
var v = Projectile.VelocityByTime(myRigid.position, predictedPos, timeOfFlight);
myRigid.AddForce(v, ForceMode.VelocityChange);
```

View `Defender.cs` for the whole implementation of this (how to calculate `predictedPos`, etc.).

Additionally, you can add a *SimpleExplosive* component to your prefab, it handles collision events, explosion VFX, explosion force, and damage. Add the following logic below the above lines to make it work properly (for the above example, `targetPosition` is `predictedPos`):

```
myRigid.GetComponent<ProjectileBehaviour>().Launch(targetPosition);
```

For 2D

In 2D, everything is the same, except that `Rigidbody2D` does not have `ForceMode2D.VelocityChange` mode, so we use `ForceMode2D.Impulse` instead:

```
myRigid2D.AddForce(myRigid2D.mass * v, ForceMode2D.Impulse);
```

Trajectory prediction

There are 3 approaches to do trajectory prediction, here is a table that compares their pros and cons:

	Projectile.Positions(...)	**TrajectoryPredictor**	**PEBTrajectoryPredictor**
Description	A low-level method that returns an array of Vector3 representing points of the trajectory.	A component that handles calculation and rendering of trajectories.	A component that handles calculation and rendering of trajectories, which invokes physics engine.
Calculation	Algorithmic; Non-iterative*	Algorithmic; Non-iterative*	Physics-engine based; Iterative
Collisions and Bounces	No	No (collision is planned)	Yes
Aerodynamic Move	No	No (planned)	Yes
Performance	High	High	Medium
Usage	Link	Shown below	Link

* For example, you can calculate just 10 points to represent a 10-km trajectory, those 10 points will be accurate. This can't be done with *PEBTrajectoryPredictor*.

> **Important**

> ⚠️ If you encounter that the trajectory is not showing, search the following materials in the Project panel and click on them one by one, and the problem should be fixed: "Dash Line", "Slash Line", "SquareParticle". (This is a Unity Editor bug.)

How to use *TrajectoryPredictor*

1. Drag and drop the prefab "Trajectory Predictor.prefab" in folder "Blobcreate/Projectile Toolkit/Prefabs" into your scene,
2. In your script, add the following logic:

```
[SerializeField] TrajectoryPredictor tp;
...
// Update() or your own method
void Update()
{
    // Call Render to update the positions of the line.
    tp.Render(launchPosition, launchVelocity, distanceOrEnd);
}
```

Or if you want to predict the trajectory of a moving rigidbody:

```
tp.Render(myRigid.transform.position, myRigid.velocity, distanceOrEnd);
```

> ⓘ **Note**
>
> "distanceOrEnd" here means that you can either put "distance (float)", or "end (Vector3)" point of the trajectory if known, to control the length of the predicted trajectory line. In other words, to calculate the trajectory to how far.

Explore the demos

Online

Click here to play the online version (WebGL).

In editor

You can explore the demos under the folder "Blobcreate/Projectile Toolkit/Demos".

Some setup is required:

1. Upgrade materials

The materials are Built-in Render Pipeline materials. If you are using Scriptable Render Pipeline, you can convert the materials easily:

- URP: for newer versions: URP 12.0+, for older versions: URP

  (Optional): import native URP unlit materials from "Blobcreate/Projectile Toolkit/Materials/URP Unlit Materials.unitypackage". Double click on it and import, the corresponding materials will be overriden and updated.
- HDRP: for newer versions: HDRP 12.0+, for older versions: HDRP
- Custom RP: manually replace these materials with the equivalent shaders in your RP.

> **Important**
>
> ⚠️ If you encounter that the trajectory is not showing, search the following materials in the Project panel and click on them one by one, and the problem should be fixed: "Dash Line", "Slash Line", "SquareParticle". (This is a Unity Editor bug.)

2. Visual setup (optional):

Post-processing: you can create a post-processing volume and assign your profile to it. A profile called "URPPostProcessing" is provided for use in URP.

If your project uses URP but there are rendering problems with demo scenes you can use "PTK-URP-HighQuality" render pipeline asset.

3. Set up layers and physics

(These settings are for the demo scenes, they are not required for your own scenes/projects.)

Back up your layer settings and physics settings, and apply the layer preset "PTKLayers" and physics preset "PTKPhysics" under ".../Demos/Other Assets/Settings". Detailed steps:

1. (At the top right of Unity editor) select "Layers > Edit Layers...",
2. Click the second icon in the top right of the inspector,
3. Click "Save current to..." button to back up your current layer settings,
4. Click that second icon again and then choose "PTKLayers" in the pop up window.

Setting up the physics is similar, select "Edit > Project Settings...", select "Physics", click the second icon in the top right, back up your current settings, and apply "PTKPhysics" preset.

> **Tip**
>
> Backups can be important if your project already have custom layers or layer collision matrix.

Relation

Which scene demonstrates which algorithm? The relation is shown in the table below:

Method	Scene Index(es) / `class(es)`
VelocityByA	02 / `JumpAttacker` `ProjectileLauncher`
VelocityByAngle	03 / `CannonLike`
VelocityByTime	02 / `Defender`
VelocityByHeight	00 / `JumpTester`, 01 / `NMJump`, 02 / `JumpAttacker`
AnglesBySpeed	03 / `CannonLike`
VelocitiesBySpeed	03 / `CannonLike`
PositionAtTime	Demo coming soon
Positions	Used in *TrajectoryPredictor* component.
ElevationalReach	03 / `CannonLike`
VerticalFlightTest	Used in FlightTest(...) method.
FlightTest	03 / `CannonLike`

For *TrajectoryPredictor* component, the demo scene is 02.

For other features, you can find the corresponding demo scenes by their names.

> **Tip**
>
> The script filenames of the classes are `class` + `.cs`. You can find them under the folder "Blobcreate/Projectile Toolkit/Demos/Scripts".

Newer feature sets 🥳

(The titles in italics mean they are also the primary class names.)

– *TrajectoryPredictor*

This is a component that handles calculation and rendering of trajectories, it wraps `Projectile.Positions(...)` and has trajectory rendering implemented. See *How to use > Trajectory prediction* for the concrete usage.

Other features have dedicated documentation pages:

– *PEBTrajectoryPredictor*