Menu

API 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	Simulating 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.

Trajectory prediction

Method	Example use case
PositionAtTime	Implementing 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	To predict the trajectory of a projectile. (You can use **Trajectory Predictor** component instead, it has trajectory rendering implemented.)

How to use

> **Note**
>
> - `using Blobcreate.ProjectileToolkit;` in your scripts to be able to call the APIs.
> - Trajectory line materials are Built-in RP materials. If you are using SRP, see Explore the demos > In editor to see how to convert.

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.

Take `VelocityByTime(...)` for example. If you want enemies 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.

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

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

Trajectory prediction

To predict and render the trajectory of a projectile:

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);
```

For more information and examples, please refer to the *Scripting Reference* and demos.

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:

- In URP, go to Edit > Render Pipeline > Universal Render Pipeline > Upgrade Project Materials to UniversalRP Materials.

  > (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.

- In HDRP, go to Edit > Render Pipeline > Upgrade Project Materials to High Definition Materials. The skybox material won't convert, you need to replace it with your own.
- In your custom RP, manually replace these materials with the equivalent shaders in your RP.

⚠ After upgrading, if you encounter that the trajectories don't display, 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".

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.

2. Set up layers and physics

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 save 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.

💡 If you want to go back to the physics and layer settings of your project, simply apply the settings you've backed up.

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 **Trajectory Predictor** component.

For **Trajectory Predictor** component, the demo scene is 02.

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

Projectile Toolkit 1.0