Table of Contents

Scripting Reference

Targeting

VelocityByA

Computes the launch velocity by the given start point, end point, and coefficient *a* of the quadratic function $f(x) = ax^2 + bx + c$ which determines the trajectory of the projectile motion.

```
public static Vector3 VelocityByA(Vector3 start, Vector3 end, float a)
```

`start`: The starting point of the projectile motion.

`end`: The target point you want the projectile motion to hit or pass through.

`a`: The *a* coefficient of the quadratic function $f(x) = ax^2 + bx + c$. It determines the shape and speed of the trajectory, for example, -0.2f makes the trajectory curvier and slower while -0.01f makes it straighter and faster. Should always be negative.

VelocityByAngle

Computes the launch velocity by the given start point, end point, and launch angle in degrees.

```
public static Vector3 VelocityByAngle(Vector3 start, Vector3 end, float
elevationAngle)
```

`start`: The starting point of the projectile motion.

`end`: The target point you want the projectile motion to hit or pass through.

`elevationAngle`: The launch angle in degrees. 0 means launch horizontally. Should be from -90f (exclusive) to 90f (exclusive) and greater than the elevation angle formed by `start` to `end`.

VelocityByTime

Computes the launch velocity by the given start point, end point, and time in seconds the projectile flies from `start` to `end`. The projectile object will be exactly at the end point `time` seconds after launch.

```
public static Vector3 VelocityByTime(Vector3 start, Vector3 end, float time)
```

`start`: The starting point of the projectile motion.

`end`: The target point you want the projectile motion to hit or pass through.

`time`: The time in seconds you want the projectile to fly from `start` to `end`.

VelocityByHeight

Computes the launch velocity by the given start point, end point, and max height of the projectile motion.

```
public static Vector3 VelocityByHeight(Vector3 start, Vector3 end, float
heightFromEnd)
```

`start`: The starting point of the projectile motion.

`end`: The target point you want the projectile motion to hit or pass through.

`heightFromEnd`: The height measured from the `end` point (for example, 1f means the max height of the trajectory is 1 meter above the end point). The algorithm automatically clamps the value if it is lower than the `y` value of `start` or `end`.

AnglesBySpeed

Computes the two angle results by the given start point, end point, and launch speed. Returns `false` if out of reach.

```
public static bool AnglesBySpeed(Vector3 start, Vector3 end, float speed, out
float lowAngle, out float highAngle)
```

`start`: The starting point of the projectile motion.

`end`: The target point you want the projectile motion to hit or pass through.

`speed`: The launch speed of the projectile object.

`lowAngle`: The lower angle that satisfies the conditions, or 0 if the method returns false.

`highAngle`: The higher angle that satisfies the conditions, or 0 if the method returns false.

> 💡 **Note**
>
> If `AnglesBySpeed` or `VelocitiesBySpeed` returns `true`, then there are always two effective and different `out` results, this is mathematically correct. One extreme case is that when the `start` and the `end` form exactly the maximum range that the `speed` can reach, the two `out` results will be the same. No matter whether the return value is true or false, any value originally supplied in `out ...` will be overwritten.

VelocitiesBySpeed

Computes the two velocity results by the given start point, end point, and launch speed. Returns `false` if out of reach. This is an extended version of `AnglesBySpeed`. It is more convenient than `AnglesBySpeed` when the rotation is not separated into y axis and x axis.

(For example, cannon's rotation is separated, base => y, barrel => local x, while an archer using a bow the rotation can be `Slerp(...)` directly between two directions.)

```
public static bool VelocitiesBySpeed(Vector3 start, Vector3 end, float speed,
out Vector3 lowAngleV, out Vector3 highAngleV)
```

`start`: The starting point of the projectile motion.

`end`: The target point you want the projectile motion to hit or pass through.

`speed`: The launch speed of the projectile object.

`lowAngleV`: The lower-angle velocity that satisfies the conditions, or (0, 0, 0) if the method returns false.

`highAngleV` The higher-angle velocity that satisfies the conditions, or (0, 0, 0) if the method returns false.

(new in 1.1) - ElevationalReach

Overload 1

Computes how far a projectile that uses the given `speed` at `start` can reach at the given elevation `endElevation`. Returns -1f if can't reach the elevation.

```
public static float ElevationalReach(Vector3 start, float endElevation, float
speed)
```

`start`: The starting point of the projectile motion.

`endElevation`: The elevation (y) of the target point you want the projectile motion to hit or pass through.

`speed`: The launch speed of the projectile object.

Overload 2

Computes how far a projectile that uses the given `speed` at `start` can reach at the given elevation `endElevation`, and outputs the corresponding launch angle. Returns -1f if can't reach the elevation.

```
public static float ElevationalReach(Vector3 start, float endElevation, float
speed, out float angle)
```

`start`: The starting point of the projectile motion.

`endElevation`: The elevation (y) of the target point you want the projectile motion to hit or pass through.

`speed`: The launch speed of the projectile object.

`angle`: The angle that satisfies the conditions.

Prediction

PositionAtTime

Computes the position of the projectile at the given time counted from the moment the projectile is at `origin`.

```
public static Vector3 PositionAtTime(Vector3 origin, Vector3 originVelocity,
float time, float gAcceleration)
```

`origin`: Launch position, or the position of the projectile at a certain time (usually current).

`originVelocity`: The velocity of the projectile when it is at `origin`.

`time`: The time counted from the moment the projectile is at `origin`.

`gAcceleration`: Gravitational acceleration, equals the magnitude of gravity (normally equals Physics.gravity.y).

Positions

Computes the trajectory points of the projectile and stores them into the buffer.

```
public static void Positions(Vector3 origin, Vector3 originVelocity, float
distance, int count, float gAcceleration, Vector3[] positions)
```

`origin`: Launch position, or the position of the projectile at a certain time (usually current).

`originVelocity`: The velocity of the projectile when it is at `origin`.

`distance`: To calculate the positions to how far, from `origin` and ignoring height.

`count`: How many positions to calculate, including the origin and end.

`gAcceleration`: Gravitational acceleration, equals the magnitude of gravity (normally equals Physics.gravity.y).

`positions`: The buffer to store the calculated positions.

(new in 1.1) - VerticalFlightTest

Tests if a projectile at `start` can use the vertical velocity (y) of `startVelocity` to hit the elevation (y) of `end`, if true, outputs the time of flight based on the vertical speed. Horizontal speed is ignored.

```
public static bool VerticalFlightTest(Vector3 start, Vector3 end, Vector3
startVelocity, out Vector2 timesOfFlight)
```

`start`: The starting point of the projectile motion.

`end`: The target point you want the projectile motion to hit or pass through.

`startVelocity`: The velocity at the start point, or launch velocity.

`timesOfFlight`: The time results that a projectile fly from `start` to `end` with the launch velocity `startVelocity`.

(new in 1.1) - FlightTest

Tests if a projectile at `start` can use `startVelocity` to hit `end`, and outputs the time of flight.

```
public static bool FlightTest(Vector3 start, Vector3 end, Vector3 startVelocity,
FlightTestMode testMode, out float timeOfFlight)
```

`start`: The starting point of the projectile motion.

`end`: The target point you want the projectile motion to hit or pass through.

`startVelocity`: The velocity at the start point, or launch velocity.

`testMode`: FlightTestMode (Enum).

`timeOfFlight`: The time that a projectile fly from `start` to `end` with the launch velocity `startVelocity`.

> 💡 What's the Difference between `FlightTest` and `VerticalFlightTest`?
>
> `VerticalFlightTest` focuses on the vertical value `y` of the velocity and end point, x and z values are ignored. It is good for when: 1) you don't know the x and z values of the end point, or 2) the start and end points are very close, or equal, on the xz-plane, which will cause computer precision issues using the horizontal-based one (`FlightTestMode.Horizontal`).
>
> `FlightTest` is a superset of `VerticalFlightTest`, when you set the `testMode` to `FlightTestMode.VerticalA` or `FlightTestMode.VerticalB`, it invokes `VerticalFlightTest`.

Components (MonoBehaviour)

Trajectory Predictor

This is a component that let you easily predict and render trajectories, it wraps `Positions(...)` and has trajectory rendering implemented. See *Manual* > *How to use* > *Trajectory prediction* for the concrete usage.