EasyTurret

Streamlined. Flexible. Ballistic.

Getting Started

EasyTurret is a streamlined turret asset that's easy to integrate into any project.

Access EasyTurret classes with the namespace:
using MobFarm;

You may wish to start with the included demo scene, or head to the **tutorial** section.

Demo Scene

Within the Demo folder are several assets that comprise the DemoScene. These assets are completely independent from core EasyTurret functionality. This folder and its contents may be deleted.

The demo includes a simple gun script that shows how to interface with the turret script.

This scene has a few primitive tank-like objects showing different turret settings firing on a small moving cube target.

The close moving turret uses low arc ballistic intercept, the middle turret uses a slow shot speed with linear intercept and no gravity, and the furthest turret uses high are ballistic intercept.

Feel free to experiment with different settings.

If you input results that cause a turret to not fire, check the following:

1. Make sure the target is in range of the weapon.
2. Make sure the target is within any turret arc limits.
3. Check the turret has a valid firing solution. (Check the Firing Solution field in the inspector of MF_EasyTurret during runtime) If there is no solution, this is usually due to insufficient shot speed.

If the shots are missing the target:

1. Check if the target is changing directing of speed after the shot was fired - this can cause a miss.
2. If using high arc ballistic intercept, try increasing High Arc Iterations.
3. Shots may be expiring before they hit - increase Max Range on the MF_DemoGun script.

Script Reference

- **MF_EasyTurret**
  The main script for turret motion. Has various methods to access turret control, such as to set targets, query if the turret is aimed at the target, or if a given target is within the turret's range of motion. This should be placed at the root level of a turret object.
- **MF_StaticBallistics**
  Contains calculations related to ballistic arc functions.
- **MF_StaticIntercept**
  Calculations to find the linear intercept point to hit a moving target.
- **MF_EasyTurret_Editor**
  Custom editor that includes turret arc visualization in the scene view and supports multi-object editing.

Demo Scrips
These scripts are not necessary for MF_EasyTurret to work, and may be deleted. They simply provide example functionality to the demo scene.

- **MF_DemoMovement**
  Moves an object between waypoints with variable speed and turn rates.
- **MF_DemoGun**
  A simple gun to interface with MF_EasyTurret and shoot projectiles.
- **MF_DemoProjectile**
  A rigidbody projectile with a trail shot by MF_DemoGun.

Controlling the turret

To integrate EasyTurret into your project, you'll need a script to designate targets and coordinate with your weapon.

Usually, this simply involves using another script to invoke various methods on the turret. However, EasyTurret can also be configured to send events to another script using the interface ITurretEvents. Interface method described below.

For complete details, see **MF_EasyTurret** script reference.

To properly aim to hit a target, the turret needs the shot speed of the weapon. (Except direct aim mode does not use shot speed.) This can be entered manually or set via script. For the turret to compute intercept, you'll also need velocity data of the shooter and target.

To access MF_EasyTurret and other classes, add at the top of your script:

```
using MobFarm;
```

Set a field to reference an MF_EasyTurret script:

```
MF_EasyTurret turretScript;
```

Then in Awake() cache a reference to the turret script of the turret you wish to access:

```
void Awake ( ) {
        turretScript = myTurret.GetComponent<MF_EasyTurret>();
}
```

Use turret methods thus:

```
turretScript.SetTarget( target, targetRigidbody );
```

Some methods to get you started:

SetTarget()
Set the turret's target:

```
turretScript.SetTarget( target, targetRigidbody )
turretScript.SetTarget( target, targetRigidbody2D )
```

target: the transform of the target. Set to null to clear a target.

targetRigidbody: The rigidbody of the target. This is used to find the target's velocity data for intercept calculations. Without a rigidbody, velocity calculations are much less accurate.

GetTarget()
Returns the transform of the current target:

```
turretScript.GetTarget()
```

PositionWithinLimits()
Bool test if a position is within the turret's gimbal limits:
turretScript.PositionWithinLimits(position)

position: the vector3 location of the position to test.

This can be used to choose if a particular target should be passed to the turret.

AimCheck()
Bool test if turret is aimed to hit its current target, if any:
turretScript.AimCheck(aimTolerance)

aimTolerance: This angle is how close (in degrees) to the direction to the target location necessary to return a true result. Avoid using 0 due to floating point errors.

This can be used to decide when to fire a weapon.

Setting up your own turret

If you have a turret asset that already has a proper hierarchy for rotation and elevation, set-up will be easy.

1. Place the MF_EasyTurret script at the turret root.
2. Assign the turret's rotator, elevator, and weapon exit parts on the script.

Below is if the turret asset doesn't already have the proper hierarchy.

A blank turret prefab with the proper hierarchy has been provided to make things easier.

1. From the Prefabs folder drag the Blank Turret Template prefab into your scene.

2. In the inspector, you can see the object hierarchy:
   Blank Turret Template
   Rotator
   Elevator

The root object, Blank Turret Template, has the MF_EasyTurret script attached. (You may wish to right click the object and select Prefab > Unpack to un-link it from the template.)

Next, we'll add mesh objects to represent the turret parts.
Instead of placing mesh components directly to the parts, it is highly recommended to give these their own GameObjects placed as children of the part. This will allow you to freely move, scale, or rotate the mesh object.

1. Place your base mesh as a child of the Blank Turret Template, and move, scale, or rotate it as desired.

2. If needed, move the Rotator object so that it's pivot represents the center of rotation for your turret. It need not be centered with the root object, but don't change its rotation.

3. Place the mesh associated with the rotation part as a child of of the Rotator. Move, scale, or rotate the mesh object as desired. Match the direction you want to be forward to the positive z axis.

4. Move the elevator object so that its pivot is where its elevation movement should be centered. This need not be centered with the Rotator, but don't change its rotation.

5. Place the mesh associated with the elevator part as a child of the Elevator. Move, scale, or rotate the mesh object as desired. Match the direction you want to be forward to the positive z axis.

6. Now drag your weapon object as a child of the Elevator object. Keep the weapon's rotation at (0,0,0)

   In the inspector, your object hierarchy should look something like this:
   Blank Turret Template
   GameObject (with mesh)
   Rotator
   GameObject (with mesh)
   Elevator
   GameObject (with mesh)

Weapon Object

This completes the basic set-up of the turret parts.

Using interface ITurretEvents

If you want EasyTurret to send TurretEvents to other scripts, you can use use an interface.

You'll first need to set up your script to handle sent events, then add it to a turret's list of items to send events to.

Set up your class to inherit the interface:

```
public class MyScript : Monobehaviour, ITurretEvents {
```

Then implement the TurretEvent() method:

```
public void TurretEvent ( eventType ) { }
```

In this method you can check for any or all particular events of TurretEventType:
enum **TurretEventType** { AimedAtTarget, GainedTargetAim, LostTargetAim, GainedTarget, LostTarget }

AimedAtTarget: Sends this eventType every frame the turret is aimed to hit its current target. (within 0.5 degrees)
GainedTargetAim: Sends this event upon reaching the aim to hit its target. Turret must loose proper aim before sending this event again.
LostTargetAim: Sent upon loosing proper aim to the target. Turret must gain proper aim before sending this event again.
GainedTarget: Sent upon gaining a new target.
LostTarget: Sent upon loosing a target.

For example, to have some action upon the turret gaining the proper aim to hit its target:

```
public void TurretEvent ( eventType ) {
        if ( eventType == TurrtEventType.GainedTargetAim ) {
        // do a thing here
        }
}
```

Finally, in the editor of MF_EasyTurret, drag and drop the object with the script inheriting ITurretEvents into the list of Event Targets.

Upon Awake() the turret will assign these items as places to send TurretEvents.

Alternately, you can use:

```
AddEventTarget ( eventTarget )
RemoveEventTarget ( eventTarget )
```

to add or remove an object during runtime.