MF_EasyTurret

Describes the properties of a turret, and it will control the aiming of your turret. Place this in the root GameObject of your turret. Typically, this will be aiming a weapon at a target, but can be used with cameras, or other rotating objects.

Public Interface ITurretEvents

Offers an alternate way to track turret states. In this case, the turret can send a TurretEventType to any class that implements ITurretEvents.

TurretEvent : void (*eventType* : 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. Must loose proper aim before sending this event again.
LostTargetAim: Sent upon losing proper aim to the target. Must gain proper aim before sending this event again.
GainedTarget: Sent upon gaining a new target.
LostTarget: Sent upon losing a target.

Public Methods

AddEventTarget : void (eventTarget : ITurretEvents)
AddEventTarget : void (gameObject : GameObject, eventTarget : ITurretEvents)
Adds an object implementing ITurretEvents as a receiver to a turret's TurretEvents.
gameObject is an optional parameter during runtime, but it can serve as a visual indicator in the inspector, since the editor doesn't display interface types.

AimCheck : bool (*aimTolerance* : float)
AimCheck : bool (*targetSize* : float, *aimTolerance* : float)
Returns true if the turret is aimed properly to hit the target. This takes into account intercept and ballistics.

targetSize in meters is the average size of the target. AimCheck() will use this to compute the angular size of the target.

aimTolerance is the angle is how close (in degrees) to the direction to the target location necessary to return a true result. This will be added to any given *targetSize*. Avoid using 0 due to floating point errors.

ClearAngleGoal : void ()
Clears and angle goal set by SetAngleGoal() and allows the turret to rotate as needed.

ClearTarget : Transform ()
Will clear a target. Same as calling SetTarget(null);

CheckAngleGoal : bool ()
Checks to see if the turret is pointed towards the angles set by SetAngleGoal().

GetAngleToTargetAim : float? ()
Returns the absolute angle value to the target's predicted location. Returns null if the turret has no target;

GetRotationToTargetAim : float? ()
Returns the rotation angle to the target's predicted location. Returns null if the turret has no target or no valid solution. (-180 - 180)

GetElevationToTargetAim : float? ()
Returns the elevation angle to the target's predicted location. Returns null if the turret has no target or no valid solution. (-90 - 90)

GetTarget : Transform ()
Will return the current target transform.

PositionWithinLimits : bool (*Vector3* : position)
Checks if a given *position* in world space is within the gimbal limits of the turret.

RemoveEventTarget : void (eventTarget : ITurretEvents)
Removes an object implementing ITurretEvents as a receiver of a turret's TurretEvents.

SetAngleGoal : void (*x* : float, *y :* float)
SetAngleGoal : void (*angles* : Vector2)
Sets the turret to move to the specified *angles*. *x* = elevation angle, *y* = rotation angle.
This will override any target, but is bound by rotation/elevation rates and gimbal limits.

SetTarget : void (*target* : Transform)
SetTarget : void (*target* : Transform, *targetRigidbody* : Rigidbody)
SetTarget : void (*target* : Transform, *targetRigidbody2D* : Rigidbody2D)
Sets the turret to aim at the input *target* according to *aimType* (Direct, Intercept, Ballistic, BallisticIntercept.)
If the target has no rigidbody specified the transform will be first searched for a Rigidbody. If none was found, then it will be searched for a Rigidbody2D. If still no rigidbody is found, target velocity will be calculated by tracking location, but this is not as accurate. Setting *target* to null will clear the target

Public Variables

editorTarget : Transform
Target that can be initially set in the editor. Changing this during runtime has no effect.

hasSolution : bool
Shows if ther turret has a mathematical solution to hit target, regardless if it is currently aimed correctly. A false result is usually due to insufficient shot speed.

aimType : AimType enum { Direct, Intercept, Ballistic, BallisticIntercept }
Direct: Will aim directly at target.

Intercept: Will compute linear intercept to hit a moving target.
Ballistic: Will aim in a ballistic arc to hit a target not moving relative to turret.
BallisticIntercept: Will aim in a ballistic arc to hit a target moving relative to the turret.

Intercept and BallisticIntercept modes will require the shot speed, and velocity of shooter and target. Ballistic mode requires shot speed. All ballistic calculations are made with drag equal to 0.

ballisticArc : MFnum.ArcType enum { Low, High }
Specify which ballistic arc solution to use, since ballistic aim usually has two possible solutions.

highArcIterations : int
High *ballisticArc* generally requires more iterations to hit moving targets. More = better accuracy, but more cpu usage. If both shooter and target are stationary, iterations will be ignored. Recommend starting with a value of 3. If dealing with long flight times, a higher value may be needed.

shotSpeed : float
Should be set to match the shot speed of the weapon used. This may be assigned by the weapon directly.

rotationRate : float
The maximum rotation rate of the *rotator* part.

elevationRate : float
The maximum elevation rate of the *elevator* part.

limitLeftRight : float
rotator turn arc limit: (0 to 180) Mirrored on both sides. Left and right limits are visualized in the scene view with yellow lines. If no limit exists, a dimmed gray circle appears instead.

limitUp : float
(0 - 90) The maximum angle the *elevator* part may rise. Upward limit visualized with a red line in the scene view.

limitDown : float
(-90 - 0) The minimum angle the *elevator* part may depress. Down limit visualized with a red line in the scene view.

restAngle : Vector2
You may define an angle for the turret to point when it has no target. These angles are bound by *limitLeftRight*, *limitUp*, and *limitDown*.
x is the angle of elevation (-90 to 90).
y is the angle of rotation (-180 to 180).
0, 0 points straight ahead.

restDelay : float
Time without a target before turret will move to the *restAngle*.

rotator : Transform
Transform that rotates around the y axis. Should be a child of the turret base.

elevator : Transform
Transform that rotates around the x axis. Should be a child of the rotator part.

weaponExit : Transform
The Transform where a shot will be produced. This may be changed dynamicaly in the case of multiple exits - alternately, pick a transform at the center of multiple exits. If left blank the *elevator* part will be assigned, but this should eventually be assigned by a weapon.

velocityRoot : Transform
Assign a specific transform to be used for velocity. Ideally, one that has a Rigidbody component. If blank, will assume this turrets's root transform. If no rigidbody is found, velocity will be computed from tracking position, but this is not as accurate.

rotatorSound : AudioObject
elevatorSound : AudioObject
These control the audio to be used for each part's movement sound. This sound will play at a varying pitch and volume level based on the current rotation/elevation speed as compared to its respective maximum.

For best results, this sound should be a constant looping whir or hum. Steady looping rhythmic elements work well too, as their rate will also vary with the turning speed.

eventTargets : List<TurretEventTarget>
Allows the turret for register objects implementing ITurretEvents. These objects will reviece TurretEvent items.

Public Class TurretEventTarget

A wrapper class is used because the editor doesn't serialize interfaces. The GameObjects will be searched for ITurretEvents upon Awake()

gameObject : GameObject
[Hidden] **ITurretEvents** : eventTarget

Public Class AudioObject

audioSource : AudioSource

pitchMin : float
Minimum pitch multiplier at near 0 turn rate.

pitchMax : float
Maximum pitch multiplier at fastest turn rate.

volumeMin : float
Minimum volume multiplier at near 0 turn rate.

volumeMax : float
Maximum volume multiplier at fastest turn rate.

MF_StaticIntercept

Public Class MFmath

Static Methods

Intercept : Vector3? (*shooterPosition* : Vector3, *shooterVelocity* : Vector3, *shotSpeed* : float, *targetPosition* : Vector3, *targetVelocity* : Vector3)
Calculates the Vector3 required to aim to hit a moving target at a particular shot speed. Returns null if no solution is possible.

MF_StaticBallistics

Methods to compute various ballistic functions.

Public Class MFball

Static Methods

BallisticAimAngle : float? (*targetLoc* : Vector3, *exitLoc* : Vector3, *shotSpeed* : float, *arc* : MFnum.ArcType)
Computes the angle to aim in radians to hit a target. Since nearly every solution will have two possible arcs, arc designates whether to find the lower or higher arc.

BallisticFlightTime : float? (*targetLoc* : Vector3, *exitLoc* : Vector3, *shotSpeed* : float, *aimRad* : float, *arc* : MFnum.ArcType)
Calculates the flight time of a shot fired at *aimRad* radians. Internally, the formula uses the height difference between *targetLoc* and *exitLoc*, and to hit a given height, there are two solutions. arc will be used to determine the correct solution to use.

BallisticIteration : float? (*targetLoc* : Vector3, *exitLoc* : Vector3, *shotSpeed* : float, *aimRad* : float, *arc* : MFnum.ArcType, *targetVelocity* : Vector3, *platformVelocity* : Vector3, *aimLoc_In* : Vector3, **out** *aimLoc_Out* : Vector3)

When using both ballistics and intercept to hit a target, an iterative solution is needed.

This finds the angle to aim in radians to hit a target, then finds the intercept solution based on the time of flight. Then recalculates the aim angle with the new effective shot speed based on flight time.

Both *aimLoc_In* and *aimLoc_Out* are used to incrementally get closer to an acceptable solution with every iteration.