MobFarm Basics 2.0 - Script Reference

MF_B_Gun
Contains the properties of a gun style weapon. Place this at the root level of your weapon object.

MF_B_Hardpoint
This is a platform for fixed weapons. This is really an empty script that inherits MF_AbstractPlatform, and the base class already has everything it needs.

MF_B_HardpointControl
Controls the behavior of a fixed platform, and coordinates its weapons. Since the weapons aren't aimed, the unit must aim them by maneuvering.

MF_B_MobilitySpaceArcade
Provides the parameters for a simple arcade-style flying vehicle.

MF_B_MobilityWheel
Provides the properties for a simple wheeled vehicle.

MF_B_MouseLook
Allows the use of the mouse to control the camera first-person style. Includes zoom functionality.

MF_B_Navigation
In conjunction with a mobility script, allows the unit to navigate to a target or waypoints. You can also designate a group of waypoints, by arranging them as children of a game object, and assigning that object in waypointGroup.

MF_B_Projectile
A simple bit of code for the projectiles fired by the weapons. It uses raycast to check ahead for colliders to determine hits to compensate for fast movement speeds. It will destroy itself after a fixed time, or if it detects a hit. The duration is supplied by the weapon script upon firing. Damage can be given a blast radius, and damage is applied to a hit target if it has an MF_AbstractStatus script. It will also display an fx object upon a hit.

MF_B_Scanner
Will detect targets within a particular range using tags or layers to define teams, and store them in a list of potential targets in MF_B_TargetList. MF_B_Targeting will then search the list for an appropriate target.

MF_B_Selection
Allows units to be selected and provides a visual representation of navigation goal, targeting, and target lists.

MF_B_SelfDestruct
Destroys its GameObejct after a designated amount of time.

MF_B_Spawner
An interface for the automatic spawning of prefabs.

MF_B_Status
Controls health and other properties of a unit. If health <= 0 then it will destroy the object.

MF_B_Targeting

Holds the logic for choosing targets from a target list.

MF_B_TargetList
Holds the targets found using MF_B_Scanner. This will typically be at the root level of the game unit, as more than one scanner may contribute to it, and more than one turret may access it, as in the case of a complex unit with multiple scanners and/or turrets.

MF_B_Turret
The main control for turret motion. It also has functions to 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_B_TurretControl
Controls the behavior of MF_B_Turret, and coordinates its weapons.

MF_SelectionManager
Used with a prefab to hold default visual indicators and keep track of the currently selected object.

MF_SelectionRepeater
Used for complex objects to allow clicking on colliders that may otherwise not register as a click for the main collider containing MF_B_Selection.

MF_UtilityKeys
A few keybinds to aid testing and development. Can change timeScale, plus pause, and step functionality. These can be a little more convenient than the editor buttons.

MF_B_Gun : MF_AbstractWeapon

Place this at the root level of a weapon GameObject. This controls all the properties of a weapon.

Inherited Variables

shotSound, inaccuracy, shotSpeed, maxRange, shotDuration, cycleTime, burstAmount, burstResetTime, shotsPerRound, maxAmmoCount, unlimitedAmmo, reloadTime, dontReload, active, aimTolerance, exits, platformVelocity, curAmmoCount, curBurstCount, curExit, curInaccuracy, bursting, delay

Inherited Methods

Shoot(), **ShootBurst**(), **ReadyCheck**(), **AimCheck**()

Override **DoFire** : void ()
Use if you've already called ReadyCheck(), and want the weapon to fire without first checking if it's ready. This will cause the weapon to shoot even if it shouldn't be able to fire, such as being out of ammo.

Override **RangeCheck** : bool (*target* : Transform)
Determines if a target is within the *maxRange* of this weapon. If this weapon's *shot* prefab's rigidbody *useGravity* is true, then the ballistic arc will be considered in the range calculation.

Public Variables

shot : GameObject
The prefab of your projectile to be fired. This should have a rigidbody component in its root object.

gravity : gravityUsage enum { Default, Gravity, No_Gravity }
Provides an easy way to change shot gravity usage.

Default: The gravity setting of the projectile rigidbody won't be changed.
Gravity: Projectile rigidbody gravity setting will be set to true.
No_Gravity: Projectile rigidbody gravity setting will be set to false.

MF_B_Hardpoint : MF_AbstractPlatform

This is a platform for fixed weapons. This is really an empty script that inherits MF_AbstractPlatform, and the base class already has everything it needs.

Inherited Variables

target, weaponMount, aimObjectActive, aimObject, emptyAimRange, fixedAimObjectRange, useIntercept, useVelocityZero, shotSpeed, exitLoc, targetAimLocation, targetRange, lastTargetPosition, targetRigidbody, velocity, playerControl

Inherited Methods

TargetWithinLimits(), **AimCheck**(), **AimLocation**()

MF_B_HardpointControl : MF_AbstractPlatformControl

Controls the behavior of a fixed platform, and coordinates its weapons. Since the weapons aren't aimed, the unit must aim them by maneuvering. You'll probably want the navigation target and weapon target to be the same to make the unit aim properly.

Inherited Variables

targetingScript, weapons, curWeapon

Public Variables

target : GameObject
Current target of the platform. This will either be supplied by the targeting script, or be assigned directly. However, if the platform is choosing its own targets, this may immediately be overwritten.

controller : ControlType enum { AI_AutoTarget, None }
AI_AutoTarget: AI will use a scanner to pick targets.
None: Platform is deactivated.

alternatingFire : bool
When using multiple weapons, you can choose to have them all fire at once, or alternate shots between them. If alternating, the delay before the next weapons fires is the *cycleTime* of the weapon at index 0 divided by the number of weapons. This will produce odd results if you have weapons with different rates of fire.

fullBurst : bool
Upon firing, weapons will finish a full burst according to their burst fire setting.

checkLOS : bool
Uses Raycast to check line of sight before firing.

losCheckInterval : float
Increase time between Raycast line of sight checks to improve performance. 0 = before every shot.

checkTargetSize : bool
This script tries to determine the target size when choosing whether or not to fire. This way, it will fire even if not aimed directly at the target center. It approximates the size of the target using its largest dimension of any root collider component. If no collider is found, it will look for a collider in the root child index 0. If still none found, it will use *targetSizeDefault*.

targetSizeDefault : float
If no target size can be found, you can set a default size for the the weapons to fire at.

MF_B_MobilitySpaceArcade : MF_AbstractMobility

This describes a simple vehicle that flies like an arcade style space ship. Rigidbody not required, but recommended.
To create a proper space arcade vehicle, this script should be on the object at the vehicle's root level. Next, a child object is needed to control the roll of the ship, and all other objects of the ship should appear as children to the roll object.

Inherited Variables

navTarget, useIntercept, interceptForPlatform, navTargetAim, myRigidbody, velocity

Inherited Methods

UnitVelocity()

Override **ModifiedPosition** : Vector3 (*pos* : Vector3)
Will return the position but with the y value made to be within the range of *heightRange.*

Public Variables

rollTransform : Transform
The transform of the roll object. This is the part of the vehicle that controls the bank angle.
If blank, the first child object transform will be assigned as the *rollTransform*.

turnRate : float
How fast the vehicle can turn in deg/sec.

slowDownForTurns : bool
If true, if the navTarget is more than 10 degrees away from the forward direction of the vehicle, it will begin to reduce thrust as the angle increases to 90 degrees, where thrust will be 0%.

bankRate : float
When the vehicle turns, it will begin to bank. This is the rate of bank in deg/sec.

bankAngleMax : float
The maximum angle the vehicle will bank when at top speed. If speed is less than top speed, the maximum bank angle will be reduced to ½ *bankAngleMax* as speed approaches 0.

maintainHeightRange : bool
If true, will attempt to stay at within y axis height of *heightRange*.

heighRange : Vector2
Position range along the y axis in world space used for *maintainHeight*. x = low limit, y = high limit.

thrusters : ThrusterArcadeObject[]
An array of thrusters to provide forward movement.

Public Class ThrusterArcadeObject

maxThrust : float
The maximum force this thruster can apply. All force is applied at the center of the vehicle.

particle : ParticleSystem
An optional particle system to use for this thruster.

particleLow : float
The particle speed when at 0% thrust.

particleHigh : float
The particle speed when at 100% thrust.

Public Methods

Steer : void (goal : Vector3)
Will turn and bank the vehicle towards *goal*.

Move : void (percent : float)
Will set the thrusters to *percent* of thrust.

MF_B_MobilityWheel : MF_AbstractMobility

This describes a simple wheeled vehicle.

Inherited Variables

navTarget, useIntercept, interceptForPlatform, navTargetAim, myRigidbody, velocity

Inherited Methods

UnitVelocity()

Override **ModifiedPosition** : Vector3 (*pos* : Vector3)
Will return the position but with the y value made to be the same as the vehicle y value.

Public Variables

throttleRate : float
The rate at which the throttle setting may change in percent/sec. Use to allow gradual acceleration of wheels to prevent slippage.

wheels : WheelObject[]
An array of this vehicle's wheels.

Public Class WheelObject

wheelCollider : WheelCollider
This wheel’s WheelCollider.

torque : float
The max torque of this wheel.

steerAngleMax : float (0 to 90)
The maximum angle this wheel may steer. 0 = no steering. Negative numbers will cause this wheel to steer in the reverse direction. This could be used to give rear wheels the ability to steer.

steerRate : float
The rate that this wheel can change steering angle in deg/sec.

wheelObject : Transform
An optional transform of this wheel’s graphical elements.
If blank, will try to assign the first child of the wheelCollider object as the *wheelObject*.

Public Methods

Steer : void (goal : Vector3)
Will cause steerable wheels to steer towards *goal*.

Move : void (percent : float)
Will set the throttle goal to *percent.* The torque of each wheel will be multiplied by the current throttle setting.

MF_B_MouseLook

A simple script to turn a camera with the mouse, FPS style.
Press ‘~’ to activate/deactivate mouse aiming. (Escape has a reserved use in the editor.)
Press ‘z’ to toggle the zoom level.

Public Variables

sensitivity : float
Sensitivity of mouse input to turn camera. Higher equals faster turning.

elevationLimit : float
In degrees. Limits the up-down angle of the camera.

zoom : float
The magnification factor when zoomed in.

hideCursor : bool
If true, will hide the cursor when aiming the camera.

centerCursor : bool
If true, will keep the cursor centered when aiming the camera.

MF_B_Navigation : MF_AbstractNavigation

A simple navigation script. It will allow a mobility script to follow a target supplied by a targeting script, or a series of waypoints.

Inherited Variables

navMode, navTarget, waypointGroup, goalProx, waypoints, randomWpt, curWpt

Public Variables

stopAtLastWpt : bool
If true, will stop when the last waypoint in the list is reached.

stopAtTarget : bool
If true, will stop when the target is reached

targetingScript : MF_AbstractTargeting
The location of a targeting script. If this is blank, it will check the same game object. This is used to determine which target to follow when using *NavMode.Target* or *NavMode.TargetOrWaypoint.*

mobilityScript : MF_AbstractMobility
The location of a mobility script. If this is blank, it will check the same game object, then recursively check parents until one is found.

MF_B_Projectile

A simple rigidbody projectile. Will use raycast to check path ahead for a hit based on velocity to compensate for fast speeds that would otherwise skip over a target. This projectile does use a collider, so this is the only type of collision detected.
When a collider is hit, it will search up the target's tree looking for an MF_AbstractStatus script. If one is found, it will apply damage.

Public Variables

damage : float
Amount of damage this projectile does when hitting a collider.

damageRadius : float
When a hit is made, will check for other colliders within *damageRadius*. Colliders not be blocked by other colliders will also get damage applied, but a single status script will not be damaged more than once.

hitObject : GameObject
The prefab to create when a hit is made.

fxRadius : float
The radius of the *hitObject*. *fxRadius* * 2 is applied to the scaling of the prefab. If *fxRadius* = 0, then the prefab's scaling will not be changed.

Hidden Variables

duration : float
The maximum time this projectile can remain in the scene. This value is supplied by the weapon that fires it.

MF_B_Scanner

This script will scan for enemies and store them in a dictionary. Units should be classified into factions either using tags or layers. It's possible to use multiple scanners all providing data to the same target list.

Public Variables

targetListScript : MF_B_TargetList
The location of MF_B_TargetList, that this scanner will provide targets for. If this is blank, it will check the same game object, then recursively check parents until one is found.

factionMethod : FactionMethodType enum { Tags, Layers }
The scanner will evaluate targets by using tags or layers. Which one to use will depend on your game.
Tags: Objects are first gathered by tag, and then individually checked for range to the scanner.
Layers: Colliders are checked by range using OverlapSphere only on layers deemed targetable. If layers are used, the object on the scan layer must have a collider.
In both cases, the root GameObject of the scanned part will be stored in *targetList of* MF_B_TargetList.

targetableFactions : FactionType[] enum { Side0, Side1, Side2, Side3 }
List what factions this scanner should deem as targetable. Typically, you'll want these to be enemies of the scanning unit. Faction names must match either the tags names or the layer names.
FactionType enum definition is located in MF_StaticUtilities.

targetRootObject : bool
If true, the scanner will return the root object of the detected part. Otherwise the detected object itself will be the stored target.

detectorRange : float
Maximum range the scanner can see.

detectorInterval : float
How often in seconds the scanner refreshes the target list. Lower times result in quicker updates, but use more processing power. 0 equals every frame.

requireLos : bool
Requires a clear line of sight to detect targets.

losMinRange : float
Range at which targets will not be blocked by colliders. This is usefull to prevent collider geometry that is part of the scanning unit from blocking line of sight checks.

MF_B_Selection : MF_AbstractSelection

Attached to a collider, this allows a unit to be selected and/or targeted by the player.
If not the root object, and the root object has a rigidbody, a kinematic rigidbody will need to be added to this object as well, or clicks won't be registered. (A Unity limitation)

Additional colliders may be added to the clickable areas of this object with MF_SelectionRepeater. (See below)

Inherited Variables

targetListScript, targetingScript, navigationScript, NoTargetList, NoTargetingScript, NoNavigationScript, clickObjectBase, selectable, allowClickTargeting, clickTargetPersistance, prioritizeClickTargets, clickTargetable, selectionManager, bracketSize, selectionManagerScript, myID

Public Variables

If any of these are left blank they will use the default defined in the selection manager.

detectedMark : GameObject
A prefab of the bracket to display when this object has been detected by the selected object. If left blank, the default defined in the selection manager will be used.

analyzedMark : GameObject
A prefab of the bracket to display when this object has been analyzed by the selected object. If left blank, the default defined in the selection manager will be used.

weaponTargetMark : GameObject
A prefab of the bracket to display when this object is the weapon target of the selected object. If left blank, the default defined in the selection manager will be used.

navTargetMark : GameObject
A prefab of the bracket to display when this object is in the current navigation target of the selected object. If left blank, the default defined in the selection manager will be used.

selectedMark : GameObject
A prefab of the bracket to display when this object is the selected object. If left blank, the default defined in the selection manager will be used.

MF_B_SelfDestruct

A script to automatically destroy an object after a set amount of time.

Public Variables

destructTime : float
Time after creation this object will be destroyed.

MF_B_Spawner

A script to automatically spawn units.

Public Variables

initialSpawnAmount : int
Number of units spawned when the scene starts.

unitsPerTimedSpawn : int
Number of units spawned at every spawn interval.

spawnInterval : float
Time between spawns.

maxTimedSpawns : int
Maximum number of spawns. -1 = unlimited.

spawnRandomRadius : float
Units spawns at a random location within radius.

sameHeight : bool
Keeps a spawned unit at the same height of the spawn point, no matter the *spawnRandomRadius*.

spawnPointsGroup : Transform
Group of spawn points each unit can spawn at. Randomly chosen.

waypointGroups : Transform[]
Groups of waypoints given to each object's navigation script. Randomly chosen if more than one group.

spawnTypes : SpawnType[]
Different unit types to spawn.

Public Class SpawnType

chance : float
Spawn chance. Chance to spawn is the listed chance out of the sum of all listed chances of the group.
Example: If 3 types are listed with chances of [10, 25, 15] The first has a 20% chance to be picked. (10 / 50)

unit : GameObject
Prefab to be spawned.

overrideFaction : bool
If true, will override the default faction of the prefab. This will change the tag on the root level of the unit, and the layer of the collider object referenced in MF_AbstractStatus.layerColliderLocation to *faction*.

faction : FactionType enum { Side0, Side1, Side2, Side3 }
If *overrideFaction* is true, this is the faction the unit becomes.

spawn : Transform
Optional spawn point to override *spawnPointsGroup.*

wpt : Transform
Optional waypoint group to override *waypointGroups*.

randomWpt : bool
Spawned object begins with MF_AbstractNavigation.randomWpt set to true.

MF_B_Status : MF_AbstractStatus

Describes the health and status of an object. This is used with weapons and could be referenced by scanners and the like.

Inherited Variables

layerColliderLocation, health, signature, size

Public Variables

blastObject : GameObject
The prefab that will be created when the object is destroyed by health being 0 or less.

fxRadius : float
The radius of the *blastObject*. *fxRadius* * 2 is applied to the scaling of the prefab. If *fxRadius* = 0, then the prefab’s scaling will not be changed.

health : float (Override)
When *health* changes, if it is 0 or less, the object will be destroyed and the *blastObject* will be created.

MF_B_Targeting : MF_AbstractTargeting

Searches a list of targets on MF_B_TargetList provided by MF_B_Scanner for a target matching a criteria. This target can then be used for navigation or weapons.

Inherited Variables

target, receivingObject, dontSearchForReceivingObject, platformScript, navScript, receivingControllerScript

Public Variables

targetListScript : MF_B_TargetList
If left blank, will search the same game object, then recursively check parents until MF_B_TargetList is found.

priority : PriorityType enum { Closest, Furthest, MostHealth, LeastHealth,
MostHealthPercent, LeastHealthPercent }
How the turret chooses the target from the list to fire at. If any values result in a tie, it keeps the first result found of the tied values.

keepCurrentTarget : bool
If true, once a target is chosen, target won't change until it no longer appears on the target list. Such as when it goes out of the scanner’s range, or when it has died.

checkWeapRange : bool
Will only choose targets that are within the current weapon’s max range. This will account for shots that use ballistic arcs.

checkArcLimits : bool
Will only choose targets that are within the turret’s arc of rotation and elevation.

MF_B_TargetList : MF_AbstractTargetList

Holds the list of targets provided by MF_B_Scanner. This will be used by MF_B_Targeting to choose a target. You may have multiple scanners or targeting scripts accessing the target list.

Inherited Variables

targetClearTime, dataClearTime, clearInterval, targetCount, lastUpdate, iteratingDict

Inherited Methods

RemoveAnalyzeData()

Override **TargetCount** : int ()
Returns the number of targets on *targetList*.

Override **ClearOld** : void ()
Clears old targets and/or target data based on *targetClearTime* and *dataClearTime*.

Override **ConatinsKey** : bool (*key* : int)
Returns true if *targetList* contains the key.

Override **ClickAdd** : void (*key* : int, *transform* : Transform, *script* : MF_AbstractStatus, *clickedPriority* : bool, *targetPersists* : bool, *sqrMagnitude* : float?)
Adds a target to the list due to a mouse click from MF_B_Selection.

Override **ClickRemove** : void (*key* : int)
Removes a target due to a mouse click from MF_B_Selection.

Override **GetLastAnalyzed** : float? (*key* : int)
Returns the time a target was last analyzed. Returns null if it has never been analyzed.

Override **SetClickedPriority** : void (*key* : int, *cp* : bool)
Sets the value of the targets clickedPriority to *cp*.

Public Variables

targetList : Dictionary { int, TargetData }
The actual dictionary array that holds the targets

Public Class TargetData : AbstractTargetData

Contains no additional fields, only the ones it inherits from AbstractTargetData in MF_AbstractTargetList.

MF_B_Turret : MF_AbstractPlatform

Describes the properties of a simple turret.

Inherited Variables

target, weaponMount, aimObjectActive, aimObject, emptyAimRange, fixedAimObjectRange, useIntercept, useVelocityZero, shotSpeed, exitLoc, targetAimLocation, targetRange, lastTargetPosition, targetRigidbody, velocity, playerControl

Inherited Methods

AimCheck()
AimLocation()

Override **TargetWithinLimits** : bool (*targ* : Transform)
Returns true if *target* is within the turning limits of the turret.

Public Variables

rotationRate : float
elevationRate : float
The speed in deg/sec. of the turret rotation and elevation. This will affect the pitch of rotator and elevator sounds.

limitLeft : float (0 to -180)
limitRight : float (0 to 180)
limitUp : float (0 to 90)
limitDown : float (0 to -90)
Each of these allow you to place traversal limits on the *rotator* and *elevator* parts. Movement tracking left and down use negative numbers. Tracking to the right and up use positive numbers.

restAngle : Vector2
You may define an angle for the turret to point when it has no target.
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 *restAngle*.

rotator : Transform
The object that rotates around the y axis.

elevator : Transform
The object that rotates around the x axis.

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/rate 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.

Public Class AudioObject

audioSource : AudioSource
The audio source used for audio.

pitchMax : float
The maximum pitch the audio can be played at. This is reached when turning at full speed.

pitchMin : float
The minimum pitch the audio can be played at. This is the pitch that is approached as the turning motion slows.

MF_B_TurretControl : MF_AbstractPlatformControl

Controls the behavior of MF_B_Turret, and coordinates its weapons.

Inherited Variables

targetingScript, weapons, curWeapon

Inherited Classes

WeaponData

Public Variables

target : Transform
Current target of the turret. This will either be supplied by the targeting script, or be assigned directly. However, if the turret is choosing its own targets, this may immediately be overwritten.

controller : ControlType enum { AI_AutoTarget, None }
AI_AutoTarget: AI will use a scanner to pick targets.
None: Turret is deactivated.

alternatingFire : bool
When using multiple weapons, you can choose to have them all fire at once, or alternate shots between them. If alternating, the delay before the next weapons fires is the *cycleTime* of the weapon at index 0 divided by the number of weapons. This will produce odd results if you have weapons with different rates of fire.

fullBurst : bool
Upon firing, weapons will finish a full burst according to their burst fire setting.

checkLOS : bool
Uses Raycast to check line of sight before firing.

losCheckInterval : float
Increase time between Raycast line of sight checks to improve performance. 0 = before every shot.

checkTargetSize : bool
This script tries to determine the target size when choosing whether or not to fire. This way, it will fire even if not aimed directly at the target center. It approximates the size of the target using its largest dimension of any root collider component. If no collider is found, it will look for a collider in the root child index 0. If still none found, it will use *targetSizeDefault*.

targetSizeDefault : float
If no target size can be found, you can set a default size for the the weapons to fire at.

MF_SelectionManager

Default values to use for MF_B_Selection.

Public Variables

detectedMark : GameObject
A prefab of the bracket to display when this object has been detected by the selected object.

analyzedMark : GameObject
A prefab of the bracket to display when this object has been analyzed by the selected object.

weaponTargetMark : GameObject
A prefab of the bracket to display when this object is the weapon target of the selected object.

navTargetMark : GameObject
A prefab of the bracket to display when this object is in the current navigation target of the selected object.

selectedMark : GameObject
A prefab of the bracket to display when this object is the selected object. If left blank, the default defined in the selection manager will be used.

selectedObject : GameObject
Visible for debug purposes, and this should not be changed. This is the currently selected object in the scene.

Hidden Variables

selectedObjScript : MF_AbstractSelection

MF_SelectionRepeater

This simply allows more colliders of an object to respond to player clicks. It sends clicks to the main selection script object.

Public Variables

selectionScript : MF_AbstractSelection
The location of the selection script of the unit.
If none provided, this object and its parents will be recursively searched until an MF_AbstractSelection script is found.

MF_UtilityKeys

Some helpful keybinds to aid development. These can be changed.
‘p’ will pause or unpause the game.
‘f’ will play a single frame with every press.
‘[‘ will halve the game speed.
‘]’ will double the game speed.

This can easily be added onto for your own purposes.

Public Variables

pause : string
The key used to pause or unpause the game.

frame : string
The key used to advance a frame.

timeDecrease : string
The key used to halve *playScale*.

timeIncrease : string
The key used to double *playScale*.

playScale : float
The speed of the game when not paused.