MobFarm Basics 2.0 - Abstract Classes and Static Methods

MF_AbstractMobility
Base class for MF_B_MobilitySpaceArcade, MF_B_MobilityWheel

MF_AbstractNavigation
Base class for MF_B_Navigation

MF_AbstractPlatform
Base class for MF_B_Turret

MF_AbstractPlatformControl
Base class for MF_B_TurretControl

MF_AbstractSelection
Base class for MF_B_Selection

MF_AbstractTargeting
Base class for MF_B_Targeting

MF_AbstractTargetList
Base class for MF_B_TargetList

MF_AbstractWeapon
Base class for MF_B_Gun

MF_enums
Project-wide enums

MF_AbstractStatus
Base class for MF_B_Status

MF_StaticBallistics
MFball: BallisticAimAngle(), BallisticFlightTime(), BallisticIteration()

MF_StaticCompute
MFcompute: SmoothRateChange(), DistanceToStop(), FindTerminalVelocity(), Intercept()

MF_StaticUtilities
MFmath: Mod(), AngleSigned()
UtilityMF: RecursiveParentComponentSearch(), RecursiveParentTransformSearch(),
FindColliderBoundsSize(), BuildArrayFromChildren()

MF_AbstractMobility

Base class for mobility scripts. And also provides a common point of reference for certain variables used by other scripts.

Public Variables

navTarget : Transform
This is the target the mobility script will attempt to reach.

useIntercept : bool
If true, will try to compute an intercept angle to *navTarget*. If no valid intercept is found, it will simply head towards the *navTarget* position.

interceptForPlatform : MF_AbstractPlatform
If *useIntercept* is true, it will maneuver to intercept with a weapon. Designate the weapon platform to use. This weapon platform must also have a target and use intercept, as it uses the intercept calculation already performed by the platform. This is useful to aim fixed guns.

HiddenVariables

navTargetAim : Vector3
The position of *navTarget* intercept point.

myRigidbody : Rigidbody
The rigidbody, if any, of the vehicle.

velocity : Vector3
The stored velocity of the unit.

Public Methods

UnitVelocity : Vector3 ( )
Returns the velocity of the unit. A rigidbody is not necessary, but highly recommended.

ModifiedPosition : Vector3 (*pos* : Vector3)
Returns the same Vector3. However, individual mobility scripts may override this to allow a positional modification of the *navTarget* position. For example, a wheeled vehicle may not be able to reach a location in the air, so the position would be modified to the height of the vehicle.

MF_AbstractNavigation

Base class for navigation scripts. And also provides a common point of reference for certain variables used by other scripts.

Public Variables

navMode : NavType enum { Waypoint, Target, TargetOrWaypoint }
Waypoint - Will follow a list of waypoints.
Target - Will follow a target picked by a targeting script.
TargetOrWaypoint - Will follow a target. If no target, will follow a list of waypoints.

navTarget : Transform
The current target to approach.

waypointGroup : Transform
A transform assigned to this field will cause it's children to become a list of waypoints.

goalProx : float
How close to get to *navTarget*. If a waypoint, will change *navTarget* to the next waypoint in the list.

waypoints : Transform[]
The list of waypoints. This can be created by assigning a Transform to *waypointGroup*.

randomWpt : bool
If true, Instead of following waypoints in order, the next *navTarget* assigned will be chosen at random from *waypoints*.

curWpt : int
The index of the current *waypoint* assigned as *navTarget* in *NavType.Waypoint* mode.

MF_AbsractPlatform

Typically, the base class of a weapon platform, but could be used for non-weapon platforms. This also provides a common point of reference for certain variables used by other scripts.

Public Variables

target : Transform
The target of this platform.

weaponMount : GameObject
The parent object for all weapons of this platform.

aimObjectActive : bool
If true, will enable *aimObject*. If false, will disable *aimObject*.

aimObject : GameObject
The object to use as an aim indicator. If a prefab, it will be created from that prefab.

emptyAimRange : float
If not aimed at a collider, the range to display *aimObject*.

fixedAimObjectRange : bool
If true, the *aimObject* will always appear at *emptyAimRange*. This also results in better performance. However, if the camera is not the same position as *weaponMount*, this will not line up with the aim point, but rather, just a point the the aim is projected through.

useIntercept : bool
If true, platform will calculate aim to hit a moving target.

useVelocityZero : bool
If true, will not impart the unit's velocity to fired shots, and will not use the unit's velocity in calculations.

Hidden Variables

These are referenced by other scripts, and typically only set when they need to be used.
shotSpeed : float
exitLoc : Vector3
targetAimLocation : Vector3
targetRange : float
lastTargetPosition : Vector3
targetRigidbody : Rigidbody
velocity : Vector3
playerControl : bool

Public Methods

AimLocation : Vector3 ()
Adjusts aim location based on intercept settings.

AimCheck : bool (*targetSize* : float, *aimTolerance* : float)
Used to determine if the platform is aimed at the target. Takes intercept into account.

TargetWithinLimits : bool (*targ* : Transform)
Used to determine if a target is within the limits of a platform, and these limits are determined by different platform scripts.
This returns true, but may be overridden by the platform script.

MF_AbstractPlatformControl

Base class for platform control scripts. And also provides a common point of reference for certain variables used by other scripts.

Public Variables

targetingScript : MF_AbstractTargeting

weapons : WeaponData[]
An array of all weapons this platform will handle.

Hidden Variables

curWeapon : int
The weapon index that is next to fire.

Public Class WeaponData

Caches the script and tracks the burst status of a weapon.
weapon : GameObject
script : MF_AbstractWeapon (Hidden)
burst : bool (Hidden)

MF_AbstractSelection

Base class for selection scripts. And also provides a common point of reference for certain variables used by other scripts.

Public Variables

targetListScript : MF_AbstractTargetList
The location of this unit's target list.
If blank: recursively searches self and parents until a target list is found.

targetingScript : MF_AbstractTargeting
The location of this unit's targeting script.
If blank: recursively searches self and parents until a targeting script is found.

navigationScript : MF_AbstractNavigation
The location of this unit's navigation script.
If blank: recursively searches self and parents until a navigation script is found.

NoTargetList : bool
If true, will suppress the automatic searching for a target list.

NoTargetingScript : bool
If true, will suppress the automatic searching for a targeting script.

NoNavigationScript : bool
If true, will suppress the automatic searching for a navigation script.

clickObjectBase : GameObject
The base object returned from a mouse click. Also should be the location of MF_AbstractStatus, if any. This allows the clicky collider to be anywhere in the object and still return the correct base object.
If blank: assumes the same object. There is no automatic search for this object as there is no standard criteria to search for.

selectable : bool
If true, this object may be clicked to become selected.

allowClickTargeting : bool
If true, this object's target list may be edited when selected by shift or right-clicking other objects.

clickTargetPersistance : bool
If true, clicked targets are persistent in the target list. (They won't be cleared due to expired time since last detection.)

prioritizeClickTargets : bool
If true, right-clicked targets are given targeting priority.

clickTargetable : bool
When another object is selected, this object may be clicked to become a target. Does not affect other targeting methods.

selectionManager : GameObject
This prefab should be the same for all clickable objects in the scene. It will be automatically added to the scene at runtime and all clickable objects will reference it.

bracketSize : float
May specify custom bracket size. If 0: will try to determine size from the largest collider bounds dimension. If no collider found, will use the first collider found on immediate children. If still none found, it will then check all of the children's children for the first collider found. If still none found, it will then use 1.

Hidden Variables

selectionManagerScript : MF_SelectionManager
The script of the selection manager.

myID : int
The instance ID used to find a target in target lists.

MF_AbstractStatus

Base class for status scripts. And also provides a common point of reference for certain variables used by other scripts.

Public Variables

layerColliderLocation : Transform
Use to point to object holding the collider used with faction layers. This may be used by MF_Spawner to change the faction of spawned units that have a collider on a faction layer.

health : float
The health of the unit.

signature : float
Not used in MobFarm Basics. However, this might be used to influence scanner and targeting scripts.

kind : float
Not used in MobFarm Basics. This could be use for some kind of class designation used to influence scanner or targeting scripts.

Hidden Variables

damageID : float
Used to prevent an explosion from damaging the same status script more than once per frame.

MF_AbstractTargeting

Base class for targeting scripts. And also provides a common point of reference for certain variables used by other scripts.

Public Variables

target : GameObject
The current target that the targeting script has chosen.

receivingObject : GameObject
This location will be used to find MF_AbstractPlatform, MF_AbstractNavigation, and MF_AbstractWeaponControl. These scripts will grab the target valu themselves. This is only used to run methods that would restrict target choices, such as arc limits. If this is blank, it will default to check for MF_AbstractPlatform on the same game object, then recursively check parents until it is found.

dontSearchForReceivingObject : bool
Will suppress searching for MF_AbstractPlatform if *receivingObject* is blank.

Hidden Variables

platformScript : MF_AbstractPlatform
navScript : MF_AbstractNavigation
receivingControllerScript : MF_AbstractWeaponControl

These values are all attempted to be found at the location of *receivingObject*.

MF_AbstractTargetList

Base class for MF_B_TargetList. And also provides a common point of reference for certain variables used by other scripts.

Public Variables

targetClearTime : float
How long a target can go having not been detected before it is considered for removal from the list.

dataClearTime : float
How long a target can go having not been analyzed before certain ‘analyze data’ to be reset. However, MF_B_TargetList does not consider any data to be specifically ‘analyze data’. Therefore, no data will be removed. This is for compatibility with future assets.

clearInterval : float
How often the dictionary will check to see if a target should be dropped off the list due to having not been detected in awhile.

targetCount : int
The number of targets on the list.

Hidden Variables

lastUpdate : float
Used to correctly time target choosing.

iteratingDict : bool
Set to true when the dict holding the target list is currently being iterated. Mainly used when a coroutine is iterating over multiple frames.

Public Class AbstractTargetData

Holds the data about each target on a target list.
Many of the variables are nullable, to accommodate the concept of “not detected”, or “unknown”, in the case that you want scanners of different capabilities.

transform : Transform
The transform of this target.

script : MF_AbstractStatus
The status script of this entry. Could be used to lookup the most current values in that script.

clickedPriority : bool
If true, MF_BasicTargeting will prioritize these target's above all others. This will be set if a target is added to the target list by an user click, and the selected object's MF_B_Selection script has *prioritizeClickedTargets* set to true.

targetPersists : bool
Targets won't drop off the target list due to expired detection time. This will be set to true due to an user clicked target, if the selected object's MF_B_Selection script has *clickedTargetsPersistance* set to true.

dataPersists : bool
Target data won't drop off the target list due to expired detection time.

lastDetected : float?
The last time this entry was scanned.

lastAnalyzed : float?
The last time this entry was analyzed. No distinction between detection and analysis when using MF_B_Scanner.

sqrMagnitude : float?
The square magnitude may be used for range comparisons to avoid using Vector3.Distance(). This can be used to save performance, particularly in mobile applications.

range : float?
Range of the target.

auxValue1 : float?
auxValue2 : float?
Additional variables to be used however you see fit, without needing to replace this class. (Possibly use for health, threat, etc.)

Public Methods

All these methods are empty, and are intended to be overridden by individual target list scripts.
TargetCount()
ClearOld()
RemoveAnalyzeData()
ContainsKey()
ClickAdd()
ClickRemove()
GetLastAnalyzed()
SetClickedPriority()

MF_AbstractWeapon

Base class for weapon scripts. And also provides a common point of reference for certain variables used by other scripts.

Public Variables

shotSound : AudioSource
Points to the AudioSource object and will play when the gun fires.

inaccuracy : float
The error radius in degrees of fired shots.

shotSpeed : float
The velocity the shot will travel at. This value will also be given to the platform when computing an intercept path.

maxRange : float
The maximum distance the shot is to travel before expiring.

shotDuration : float
The maximum time before the shot expires.

shotSpeed, *maxRange*, and *shotDuration* are mathematically linked. You only need to supply two of the three values, and the third will be computed. If all three values are supplied, *shotDuration* will still be overwritten based on *shotSpeed* and *maxRange* to eliminate any inconsistency.

cycleTime : float
The time in seconds time between fired shots.

burstAmount : int
The number of shots fired before *burstResetTime* is triggered.

burstResetTime : float
The delay in seconds before *burstAmount* resets. The greater value of *cycleTime* and *burstResetTime* will be the delay used. This should also be less than *reloadTime*, as *reloadTime* will take precedence if they both occur on the same shot.

shotsPerRound : int
How many shots are fired per ammo point and per Shoot() command. This is like a shotgun blast.

maxAmmoCount : int
The maximum ammo load of the weapon. When this reaches 0, *reloadTime* begins and will replenish the ammo of the weapon.

unlimitedAmmo : bool
The weapon doesn't need ammo, and doesn't use ammo, but *burstAmount* and *burstResetTime* still affect the weapon.

reloadTime : float
The time in seconds it takes to refill the ammo load of the weapon.

dontReload : bool
This weapon never replenishes its ammo when it runs out.

active : bool
If set to false, this weapon won't fire, and will be skipped in rotation in MF_B_TurretControl. This can be used to create weapon toggle scripts for turrets with more than one weapon.

aimTolerance : float
Extra radius in degrees the weapon reads to the target size. This allows the weapon to fire at a target even if it's not aimed perfectly. This can compensate for weapon inaccuracy and versus targets that change direction quickly. If negative, this will cause targeting to be more restricted to the center of the target.

exits : GunExit[] class { *transform* : Transform, *particleComponent* : ParticleSystem }
Weapons may have more than one exit point. This array holds all the exit transforms of a single weapon. This is particularly useful for missile banks.

particleComponent will be used as a muzzle flash. It will accept a reference to an object in the hierarchy, or a prefab. If it's a prefab, it will be instantiated at runtime.

Hidden Variables

platformVelocity : Vector3
This is the velocity of the unit, and will be set from a control script to be imparted to fired shots.

curAmmoCount : int
Amount of remaining ammo.

curBurstCount : int
Number of shots remaining in the burst.

curExit : int
Current exit index of the weapon next to fire.

curInaccuracy : float
Current inaccuracy of the weapon.

bursting : bool
True, when the weapon is performing a full burst of fire.

delay : float
The total delay before the next shot can fire. Includes *cycleTime*, and any *burstResetTime*, or *reloadTime*, that may apply.

Public Methods

Shoot : void ()
Call this method when the weapon should shoot. It will internally call ReadyCheck() before trying to fire, and won't shoot if the weapon isn't ready.

ShootBurst : void ()
Similar to Shoot(), except it will continue to fire until *curBurstCount* is 0. The weapon will not process any other shoot commands, and ReadyCheck() will return false until the weapon has completed the burst.

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.

ReadyCheck : bool ()
Returns true if the weapon is ready to fire. Will return false if it is still cycling between shots, waiting for a reload or burst reload to finish, or if it is out of ammo.

AimCheck : bool (*target* : Transform, *targetSize* : float)
Determines if the weapon is aimed to hit the given target that has a size of *targetSize*. This will also take into account the weapon's *aimTolerance*.
Does not account for intercept or ballistics. Normally you would use the AimCheck() that appears in a platform script.

RangeCheck : bool (*target* : Transform)
Always returns true. However, this method is overridden by weapon scripts for results specific to a weapon.

MF_enums

Project-wide enums.

Namespece MFnum
Public enums

FactionMethodType : enum { Tags, Layers }
Used to determine whether to use tags or layers when detecting targets with a scanner.

FactionType : enum { Side0, Side1, Side2, Side3 }
Factions can be designated using tags or layers. These names must match the tag or layer names used. Alternately, these may be changed to match layer or tag names used to differentiate teams in your project.

MF_StaticBallistics

Methods to compute various ballistic functions. These aren't used in MobFarm Basics, but are used in other packages.

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, *target* : Transform, *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.

MF_StaticCompute

Various computations.

Public Class MFcompute
Static Methods

SmoothRateChange : float (*distance* : float, *closureRate* : float, *curRate* : float, *rateAccel* : float,
decelMult : float, *rateMax* : float)
Returns the new rate of speed required to produce smooth starts and stops.

DistanceToStop : float (*speed* : float, *accel* : float)
Finds the distance required to stop based on current speed and deceleration (acceleration)

FindTerminalVelocity : float (*thrust* : float, *rb* Rigidbody)
Finds an approximated max speed of a rigidbody under a given thrust. This accounts for the rigidbody's mass and drag. This is an approximation because Unity3D calculates the effect of drag in an unusual way.

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_StaticUtilities

Various methods and math, as well as a central location for common enums.

Public Class MFmath
Public Methods

Mod : int (*a* : int, *b* : int)
Returns the modulo. Not to be confused with the '%' operator which is not modulo, but rather, the remainder operator in C#. Mod also works with negative integers.

AngleSigned : float (*v1* : Vector3, *v2* : Vector3, *axis* : Vector3)
Finds the angle between two Vector3s relative to an axis. This returns positive and negative angles. If the *axis* is up, and *v2* is to the left, the result is negative. If it's to the right, it returns a positive angle.

Public Class UtilityMF
Public Methods

RecursiveParentComponentSearch : Transform (*name* : string, *location* : Transform)
Searches for a component of type *name* on *location*. If none found, recursively checks up the tree until the component is found or the root level is reached.
Returns the transform where the component is found, or null if not found.

RecursiveParentTransformSearch : bool (*target* : Transform, *location* : Transform)
Similar to RecursiveParentComponentSearch but specifically for searching up the tree for a transform. This is mainly used as a quick way to determine if *location* appears somewhere in the hierarchy below *target*.

FindColliderBoundsSize : float (*trans* : Transform)
FindColliderBoundsSize : float (*trans* : Transform, *checkChildren* : bool)
Finds the average dimension of the bounds of first collider found on *trans*. Be aware that bounds size in Unity is world axis-aligned. That is, it is never rotated with the object. So on an angled object, a bounds size can be larger than the side of your object.
checkChildren is optional and defaults to false. If true, when no collider is found on the object, its children will be checked until the first a collider is found. Then if still no collider found, it will check all of the children's children for the first collider.
In either case, if no collider is eventually found, it will return 0.

BuildArrayFromChildren : Transform[] (*trans* : Transform)
Will return an array of the transforms that appear as the children of *trans*.
If *trans* has no children, *trans* will instead be used as a single array element.
If *trans* is null, the array will have 0 elements.