

Medal of Honor (Honour) Allied Assault

Author:JCBDigger

There are several tutorials about, but these are a few extra notes, I have added myself.

This document expands on the information in the 'script files.txt' document included with the MOH:AA SDK. Where useful, extracts from that document have been included in this one.

Scripts

These are the scripts used by the maps and entities during a game. This includes the ai scripts. Most of the game scripts are located in the global folder, the specific one for each map are in the map files. The file names usually end .scr

Entities

These are the objects on which scripts act. For example, weapons, players, characters and items.

Basic Syntax

The following adds an ammobox to a map above the floor and a command is used to make sure it lands on the floor.

```
local.jcb_ammo_1 = spawn models/items/item_grenade_ammobox.tik
local.jcb_ammo_1.origin = ( -716.47 603.88 32.13 )
local.jcb_ammo_1.angle = 0
local.jcb_ammo_1 droptofloor
```

Note that properties are set using the equals = and methods are just executed after the variable name. Some methods also have parameters within brackets ().

Different types of items will have different properties and methods.

Predefined object references

1) game

Refers to the unique game object which maintains its state across levels. Only primitive values (integers/floats/strings/vectors) will persist across levels.

2) level

Refers to the unique level object which maintains its state for the duration of a level.

3) local

Refers to the thread executing the current command.

4) parm

Refers to the unique parm object which can be used to pass parameters to new threads.

Note that any use of this variable could be coded "better" by using parameters in the creation of new threads.

5) self

Refers to the object that the thread is processing for. This object is the same for all threads in a group of threads.

6) group

Refers to the object representing the group of threads the thread executing the current command belongs to.

Vectors

I'm still working on these, but I think they are as follows:

Location: (xxxx.xx yyyy.yy zzzz.zz)
Angle: (uuuu.uu rrrr.rr ?????.??)

Location: **x** = across, **y** = forward, **z** = up

Angle: **u** = elevation, up down, **r** = rotation, left right and **?** another one I don't know yet.

```
// ( +N-S +W-E +U-D )
```

To move a character in a specific direction, use the following:

```
local.destvector = local.character-name.forwardvector * 100
```

```
local.character-name.origin = local.destvector
```

Calculates the end location to move a character 100 forward.

You can use:

forwardvector

rightvector

leftvector

They produce a vector in the form (1.00 0.00 0.00) or (0.00 1.00 0.00) or (0.50 0.50 0.00) etc.

Examples

// Table, note the solid method, or you will walk through it. BUT I can't get the size right!!!

```
local.jcb_furn_1 = spawn models/furniture/cardtable.tik  
local.jcb_furn_1.origin = ( -697.09 394.62 -15.88 )  
local.jcb_furn_1.angle = 0  
local.jcb_furn_1 solid
```

// Machine gun post

```
local.jcb_mg42_1 = spawn models/statweapons/mg42_gun.tik  
local.jcb_mg42_1.origin = ( 163 1751 450 )  
local.jcb_mg42_1.angle = 90  
local.jcb_mg42_1 yawCenter ( 90 )  
local.jcb_mg42_1 maxYawOffset ( 70 )
```

// Note the maxYawOffset (70) limits the rotation of the gun. 180 = 360 degrees.

```
// Static ammo boxes need to be spawned as a script_model
local.jcb_mg42_ammo_1 = spawn script_model "model"
"models/static/mg42ammoboxwbelt.tik" "targetname" "jcb_mg42ammobox1"
"testanim" "idle"
local.jcb_mg42_ammo_1.origin = ( 204 1728 384.13 )
local.jcb_mg42_ammo_1.angle = 150
```

```
//Ammo that can be picked up
local.jcb_ammo_1 = spawn models/items/item_grenade_ammobox.tik
local.jcb_ammo_1.origin = ( -697.09 394.62 32.13 )
local.jcb_ammo_1.angle = 0
local.jcb_ammo_1 droptofloor
```

It is also possible to spawn:

```
script_object
script_origin
```

List of Useful Items

```
local.jcb_item[local.i] = spawn models/items/item_100_healthbox.tik
local.jcb_item[local.i] = spawn models/items/item_50_healthbox.tik
local.jcb_item[local.i] = spawn models/items/item_25_healthbox.tik
local.jcb_item[local.i] = spawn models/items/item_bar_weapon.tik
local.jcb_item[local.i] = spawn models/items/item_grenade_ammobox.tik
local.jcb_item[local.i] = spawn models/items/item_heavy_ammobox.tik
local.jcb_item[local.i] = spawn models/items/item_mg_ammobox.tik
local.jcb_item[local.i] = spawn models/items/item_pistol_ammobox.tik
local.jcb_item[local.i] = spawn models/items/item_rifle_ammobox.tik
local.jcb_item[local.i] = spawn models/items/item_smg_ammobox.tik
```

Model Actions

“type_idle”

This is what they do to start with.

“idle”

“patrol”

“patrolpath”

I have not tried changing this in a script yet. The first ‘info_pathnode’ object has atarget of the next ‘info_pathnode’ object for the character to move to, then that has a target of the next until one returns to thje first one and round it goes again.

"exit_patrolnode”

"bridge_generic_patrolnode2”

"cargoyard_path2_patrolnode”

“alarm2_officer_patroller_patrolnode”

"plaza_path1_patrolnode”

"frontroad_winterguy_patrolnode”

So far I have been UNABLE to create ‘info_pathnode’ objects using a script, these have to be created in the original map.

```
{
"classname" "info_pathnode"
"origin" "-1576 -3176 16"
"spawnflags" "0"
"targetname" "t28"
"target" "plaza_path1_patrolnode"
}
```

// The following example appears to indicate that the model goes to the first then the second of the patrol nodes, based on the target and targetname parameters.
I can NOT get this to work with DM maps when the actor is added in the script!

```
{
"classname" "ai_german_winter_type2"
"scale" "1.0"
"model" "human//german_winter_type2.tik"
"testanim" "idle"
"origin" "1576.00 -3032.00 328.00"
"angle" "270"
"type_idle" "patrol"
"type_attack" "turret"
"type_disguise" "salute"
"type_grenade" "grenade"
"sound_awareness" "20"
"noticescale" "20"
"fixedleash" "0"
"enemysharerange" "0"
"patrolpath" "exit_patrolnode"
"waittrigger" "0"
"accuracy" "35"
"ammo_grenade" "2"
"disguise_range" "256"
"disguise_period" "15"
"disguise_level" "1"
"gren_awareness" "10"
"gun" "StG44"
"hearing" "600"
"sight" "640"
"maxdist" "640"
"$targetname" "exit_winter_patroller"
}
```

```
{
"origin" "1576 -3304 328"
"classname" "info_pathnode"
"spawnflags" "4"
"targetname" "t156"
"target" "exit_patrolnode"
```

```

}

{
  "classname" "info_pathnode"
  "origin" "1576 -2972 328"
  "targetname" "exit_patrolnode"
  "target" "t156"
}

{
  "origin" "-4496 -4847 47"
  "classname" "info_player_start"
  "angle" "90"
}

"waittrigger" "0"

```

“Type_Attack”

Enemy models can have different types of attack actions. It is set using the method ‘Type_Attack’. This is the list that I have found so far.

```

“cover”
“alarm”
“turret”
“machinegunner”
“balcony_attack”

```

I’ve also found:

```

“weaponless”

```

This might be useful for a Zombie creature, if it was given enough health.

RANDOM

Have a look at m511a.scr

```

randomint(3)

```

I would guess this works with the 3 being = generate a number between 1 and 3. (I have not checked that it is 1 to 3, it might be 0 to 3!)

Detect a Null or Uninitialised Variable

```

if(local.my_var == NULL || local.my_var == NIL || local.my_var == "")
{
    local.my_var = 0
}

```

and it works with the current object too...

```

if(self.target == NULL || self.target == NIL || self.target == "")
{
    self.target = "An Item"
}

```

An unused array, will be set to NIL.

Arrays

Any variable can be an array, just add an element in square brackets [2].

e.g. local.myvariable[1], local.myvariable[2], this is an array with two elements.

\$player is an array in multiplayer games. Before any players has joined it has only one element \$player[1] or just \$player.

When the next player joins they become \$player[2]

To find out how many elements are in an array, use the size method.

e.g. Myvariable.size

\$player.size

Cast Variables

First make sure all the variables are initialised, with at least a 0 or empty string. NULL variables cause a multitude of errors. See above.

Console String to Integer

```
level.coop_maxplayers = int ( getcvar (sv_maxclients) )
```

Join Integer to a string

```
local.jcb_string_item = "jcb_supplies" + local.i
```

Usually this version works, but if you get an error, try the following...

```
local.jcb_string_item = ("jcb_supplies" + local.i)
```

The brackets force the string to be calculated before it is passed to the string variable. An integer is then automatically cast to a string.

All variables need to be initialised to their correct type, before either of the above will work. See the above section on detecting NULL variables.

Print Text

```
println ("a space was found at: " + local.i)
```

The brackets should force the string to be calculated before it is passed to the print command. An integer is then automatically cast to a string.

```
println "STRING FORMAT: number: " level.str_format_num_lines
```

Two strings joined with no add function.

How to get a lift to work

My Version

```
// =====
```

```

// =====
//*** "Elevator Workings"
// =====

// Called to continually test if any player is touching something
testtouchingplayer:

    for(local.i = 1;local.i <= $player.size;local.i++)
    {
        // Call the elevator if the gate is touched and the elevator is not already on that
        floor
        if($clip1 isTouching $player[local.i] && level.elevator_active < 1 &&
        level.elevator_floor != 1)
        {
            thread elevator_moveto 1
        }

        if($clip2 isTouching $player[local.i] && level.elevator_active < 1 &&
        level.elevator_floor != 2)
        {
            thread elevator_moveto 2
        }

        if($clip3 isTouching $player[local.i] && level.elevator_active < 1 &&
        level.elevator_floor != 3)
        {
            thread elevator_moveto 3
        }

        if($elevator_switch isTouching $player[local.i] && level.elevator_active < 1)
        {
            thread elevator_nextfloor
        }

        waitframe    //infinite loop protection
    }
    waitframe    //infinite loop protection

    // infinite loop
    goto testtouchingplayer

end

// =====

//*** Elevator Setup
elevatorprep:

//*** JCB CoOp

```

```

        level.elevator_active = 0      // let lift get to a floor before being called again
        level.elevator_floor = 0      // 1=Top 2=Middle, 3=Bottom, 0=Closed Off
level.elevator_next = 0      // 1=Top 2=Middle, 3=Bottom, 0=Closed Off
level.elevator_down = 1  // 1 = down -1 = up

    /*** Switch Position
    $elevator_switch_pulse bind $elevator_cab
    $elevator_switch bind $elevator_cab
    $elevator_switch anim off

    /*** Elevator Speeds
    $elevator_cab time 4
    $elevator_gate_1 time 0.75
    $elevator_gate_2 time 0.75
    $elevator_gate_3 time 0.75

    /*** Elevator Clips
    $clip1 hide
    //$clip1 notsolid
    $clip2 hide
    $clip2 notsolid
    $clip3 hide
    $clip3 notsolid

    /*** Elevator Start Down
    waitthread elevator_moveto 2

    /*** Hide
    $elevator_switch hide

thread testtouchingplayer

end

// =====

elevator_moveto local.floor:

    // Floors 1=Top 2=Middle, 3=Bottom, 0=Closed Off
    level.elevator_active = 1      // make sure the lift finishes before it can be
called again

    iprintlnbold_noloc "Elevator moving..."

    //$elevator_cab playsound elevator_start wait

    $elevator_switch anim turn
    $elevator_switch anim waittill animdone
    $elevator_switch anim on

```



```

wait 1 // delay lift move to let people on and off
$clip1 solid
$clip2 solid
$clip3 solid

$elevator_cab playsound elevator_run

switch level.elevator_floor
{
    case 0:
        break
    case 1:
        $elevator_gate_1 playsound elevator_gate
        $elevator_gate_1 moveup 70
        $elevator_gate_1 waitmove
        break
    case 2:
        $elevator_gate_2 playsound elevator_gate
        $elevator_gate_2 moveup 70
        $elevator_gate_2 waitmove
        break
    case 3:
        $elevator_gate_3 playsound elevator_gate
        $elevator_gate_3 moveup 70
        $elevator_gate_3 waitmove
        break
}

wait .5

switch local.floor
{
    case 1:
        $elevator_cab moveto $elevator_way1
        $elevator_cab waitmove
        $elevator_cab playsound elevator_stop
        wait .5
        $elevator_gate_1 playsound elevator_gate
        $elevator_gate_1 movedown 70
        $elevator_gate_1 waitmove
        $clip1 notsolid
        $elevatoractorclip notsolid
        local.liftmessage = "Elevator on the Top Floor"
        level.elevator_floor = 1 // 1=Top 2=Middle, 3=Bottom
        break
    case 2:
        $elevator_cab moveto $elevator_way2
        $elevator_cab waitmove
        $elevator_cab playsound elevator_stop
        wait .5

```

```

$elevator_gate_2 playsound elevator_gate
$elevator_gate_2 movedown 70
$elevator_gate_2 waitmove
$clip2 notsolid
$elevatoractorclip notsolid
local.liftmessage = "Elevator on the Middle Floor"
level.elevator_floor = 2      // 1=Top 2=Middle, 3=Bottom
    break
    case 3:
$elevator_cab moveto $elevator_way3
$elevator_cab waitmove
$elevator_cab playsound elevator_stop
wait .5
$elevator_gate_3 playsound elevator_gate
$elevator_gate_3 movedown 70
$elevator_gate_3 waitmove
$clip3 notsolid
$elevatoractorclip notsolid
local.liftmessage = "Elevator on the Lower Floor"
level.elevator_floor = 3      // 1=Top 2=Middle, 3=Bottom
    break
}

iprintlnbold_noloc local.liftmessage

$elevator_switch anim off

    level.elevator_active = 0      // make sure the lift finishes before it can be
called again

end

// =====

elevator_nextfloor:

if (level.elevator_floor < 2)
    level.elevator_down = 1

if (level.elevator_floor > 2)
    level.elevator_down = -1

level.elevator_next = level.elevator_floor + level.elevator_down

if (level.elevator_next > 3 || level.elevator_next < 1)
    level.elevator_next = 2 // failsafe

    $elevator_switch anim off
    $elevator_switch show
    $elevator_switch_pulse hide

```

```

thread elevator_moveto level.elevator_next

end

// =====
// =====

```

EXAMPLE SCRIPT FILES

It's worth looking at the files provided with the game for examples.

Global/items.scr – this has examples of parameters being passed to classes within the script file.

```

// from m4l2
//Sample sparks Entity
{
"testanim" "idle"
"anim" "stop"
"targetname" "sparks2"
"model" "animate/fx_trainspark.tik"
"origin" "-1886.48 2319.62 10.38"
"classname" "script_model"
"angles" "20 273 0"
}

```

Types of Attack

See script m4l1

The friendly in this script in the thread pilot_init: uses;

Typeattack weaponless

Weapon Names

These can be found in the file:

Global/weapons.scr

```

//*** US WEAPONS
case "high standard":
case "colt 45":
case "m1 garand":
case "springfield '03 sniper":
case "thompson":
case "bar":
case "bazooka":
case "shotgun":
//*** GERMAN WEAPONS
case "walter p38":

```

```
case "mauser kar 98k":  
case "mauser kar 98d sniper":  
case "mp40":  
case "stg44":  
case "panzerschrek":  
case "mg42":
```

loadout.scr

This script assigns initial weapons.

You need to set the name of an existing level when calling the loadout.scr

```
exec global/loadout.scr maps/m513.scr
```

The script should be run BEFORE the level waitTill prespawn

It must also be before any other scripts are executed. In fact it is usually the first line, except perhaps for setting a few variables!

The script allocates different initial weapons that the players have at the start, depending on the map name! map/m513 is a good choice, giving a pistol, rifle and smg.

It is NOT necessary for Multiplayer or Co-Op games, as the multiplayer weapon selection overrides any settings from the loadout.scr.

You will get lots of unnecessary error messages if you leave this in, but only one, if you leave it out.

\$player

This is a special object representing the player. In a multiplayer game it is an array starting at 1.

e.g. \$player[1]

Things \$player can do...

\$player isTouching

\$player canSee

or

isAlive \$player

note, isAlive is in front of \$player, not following.

\$world

This is a special object representing the game world.

Setting parameters

level.farplane = 1600 //this is how far away the fog is.

\$world farplane level.farplane

The maxdist parameter of characters should be no more than 82.8% of the farplane, or the character can shoot through the fog. e.g.

```
Enemy.maxdist = int ($world.farplane * .8)
```

Getting parameters

```
local.farplane = $world.farplane
```

while()

```
while ( (local.one > local.two) && (local.three == local.four) )  
{  
    //Do something here  
}
```

if()

```
if (local.thing >= level.another_thing)  
{  
    //do something here  
}  
else  
{  
    // or something else here  
}
```