

# **The MCS Movement Control Script**

## **A LUA Script for Game Guru**

### **By GoDevil**

#### **What is MCS?**

The purpose of the MCS (Movement Control Script) is to provide Game Guru (GG) developers with a reusable script designed to move entities during game play.

This script does not require complex coding by the user, rather, the game developer enters data (direction, speed, ascend/descend, turns, sounds, etc.). During game play, the data is used by the MCS's internal programming to control the movement (and other factors) of the entity, to which the script is attached.

Finally, by making copies, renaming the script, and entering new data, the same basic MCS script can be used again and again, with different entities, adding life and movement, as well as new functionality to your game development projects.

***NOTE: MCS does NOT allow the player to control the movement of the entities. Once the script is activated it follows the data based instructions, step by step, until the “Sequence” is completed***

#### **How a LUA script works in Game Guru.**

In order for a LUA script to work requires three elements.

Data

Operational Code

Game Engine (that uses the operational code and data)

## **The Game Guru Game Engine**

The Game Guru Game Engine cannot be changed. No need for the user to know the engine connections. These are built in to the MCS. It is important to have a basic understanding of the way in which the game engine functions as it relates to LUA scripts.

Almost all LUA scripts are attached to an entity (3D model in Game Guru). This is done using the properties function in the GG editor. Once the game begins the game engine cycles through all of the LUA scripts, and executes each in turn. This cycle happens over and over during game play. Thus, with each game cycle the LUA scripts are activated effecting the play of the game.

Let's say I have a script that is going to move an entity from one location to another. As the game cycles, the movement script is activated over and over again, ultimately moving the entity to the preset location on or above the terrain.

Each GG game cycle moves the entity a certain distance and this continues until the entity has reached its goal. At this point the script is designed to tell the engine that it has arrived at its location, and as the cycles continue there is no more movement of the entity.

## **MCS Operational Code**

Like the game engine, the GG developer should not attempt to change any of the MCS script's operational code. This has been completed and should not be altered.

There are two important pieces of the operational code that you will need to change. When you make a copy of the basic script to use with another entity, you must rename the script when you save it. Inside the operational script are two lines of code where you must enter this new file name "exactly" as it is saved. Instruction for this process will be covered later.

## **MCS Data**

Planning your movement and entering the data so the MCS script will execute properly, is the function of the GG developer. This data is derived from the game itself and what you plan to accomplish. Let's for a moment look at the type of data you will need to enter.

- Move and entity forward or backward in a desired direction
- Move the entity forward until it reaches it's final destination
- Control the speed at which the entity will move
- Turn the entity and continue to move in the new direction
- Fly the entity up and down
- Establish the minimum and maximum height for the entity
- Play a sound while the entity is moving
- Set a delay so the movement does not begin immediately

In addition, data is entered to set the script into one of four basic operational formats.

- Move the entity by itself after being activated by the player pressing the “E” key

- Move the entity by itself, activated automatically

- Move the entity and the player at the same time. activated by the player pressing the “E” key

- Move the entity, activated by player proximity to the entity.

## **Player / Entity Combined Movement**

A couple of years ago the GG developers finally gave us a way to control the Player/Camera. Effectively, the camera can be separated from the player. In doing so the MCS script links the entity and camera location so they move together.

You (the player) lose a little control. You cannot move the player separately with the arrow keys or W,S,X controls. You can turn your head in all directions (with the mouse) so you can observe during the movement. You can also draw and fire a weapon in this mode.

Also, when the camera arrives at the MCS script's final location, the Player/Camera are reunited to the normal player functions. Thus, you can move the player and entity from one location to another. Once the movement is completed the MCS script returns the player to normal operation at the new location.

To understand this I have put together a video that demonstrates all three of the above noted modes. <https://youtu.be/yMp-i-scflQ>

## **Planning Your MCS Script; Sequence and Steps**

There are two main ideas to the MCS.

### **Sequence:**

The MCS script can create a movement Sequence with up to 16 Steps, or as few as 1 step. Each step is activated, one after the other, until the Sequence is complete.

### **Steps:**

Each Step provides data inputs for the variety of information (direction, speed, etc) needed to complete that step. This input process is repeated for each step.

Here's a simple movement plan.

I want the entity to fly upward and level off and continue forward to a preset new location.

Once it arrives at the new location I want it to turn to the left and continue the next preset location.

Then I want it turn right, and continue to the final preset location

At this final location the entity will land vertically on the terrain.

Overall this plan is referred to as a **Sequence**. Within the sequence are four distinct **Steps**...

Move to first location  
Turn and move to 2<sup>nd</sup> location  
Turn and move to 3<sup>rd</sup> location  
Land Vertically on the terrain

For each one of these steps the developer has the ability to enter all of the movement data described above for that specific step.  
(Direction, Speed, Length, Height, etc.)

As the MCS script is running, and as one step is completed, the sequence moves automatically to the next step. This step-by-step process continues until all of the steps are completed. That may be three steps, ten steps, or just one step.

This data entry system gives the developer a great deal of versatility in the type of movement scripts they can create just by entering data.

## Entering Data

The following is a section from the MCS script where the data for step #1 will be entered

```
--Movement Step #1 (1-20)
  ObjcontE1[1] = 8      -- Direction code for this step 1-8
  ObjcontE1[2] = 6      -- Forward speed value 0 - 15
  ObjcontE1[3] = 3      -- Ascend and descend speed -3 to +3
  ObjcontE1[4] = 25000  -- Number of game cycles for this step
  ObjcontE1[5] = 25     -- Delay before this step begins
  ObjcontE1[6] = 16500  -- Max Altitude
  ObjcontE1[7] = 0      -- Leave set to zero Not in use at this time
  ObjcontE1[8] = 0      -- X Position Target .. If not in use, set to zero
  ObjcontE1[9] = 45000  -- Z position Target If not in use, set to zero
  ObjcontE1[10] = 3     -- Minimum value above terrain height
  ObjcontE1[11] = 1     -- No Entity Rotation = 0 Rotation Steps (1-7)
                        --example turn from Dir 1 to Dir 3 = 2 Rotation Steps
  ObjcontE1[12] = -1    -- No Rotation = 0 Rotate Right enter 1 Rotate Left enter -1
  ObjcontE1[13] = 4     -- Sound Number (0-3) or 4 = no sound
  ObjcontE1[14] = 0     -- Sound Vol 0 - 100
  ObjcontE1[15] = 0     -- entry points 7, and 15 - 19 are blank for future expansion
  ObjcontE1[16] = 0
  ObjcontE1[17] = 0
  ObjcontE1[18] = 0
  ObjcontE1[19] = 0
  ObjcontE1[20] = 21    -- ObjcontE1 number pointer DO NOT CHANGE !!!
```

This step includes data I used in one of my scripts. Let's take a look at each of these elements.

### Directional Code:

A value between 1 and 8 that defines the movement direction. This will be discussed shortly.

### Forward Speed Value:

A value for each cycle of the game engine, how far forward/backwards the entity moves. The higher the number the faster the speed. Numbers that are too high will create choppy movements. Negative numbers can be used so the entity will move backwards.

**Ascend / Descend Value;**

A value that defines up and down movement for each cycle of the game engine. Fly up, level off. Next Step you can descend.

**Game Cycles for this Step**

There are two ways to control the length of a Step. It will run for a specific number of game cycles, or end based upon reaching a specific location on the terrain (see below).

**Delay Before Step Begins**

A value that pauses the execution of the step until a certain number of game cycles are reached. At that point the step is activated and completed without further delay. Each step can have its own delay, so as one step is completed there can be a delay before the next step begins.

**Maximum Altitude:**

A value which will restrict the entity from flying above a certain height. For example, you enter +3 ascend. However, when the max height limit is reached the ascend value is ignored, and the entity continues level flight at the max level. This Max Altitude value can be changed for each step in the sequence.

**X Position and Z Position Targets**

Noted above, Game Cycles value can be used to control the length of the step. However, let's say I want the entity to move to a specific X or Z terrain coordinate. When it reaches the coordinate the step ends, even if the cycles value has not been reached.

*Note: You can only identify an X or a Z coordinate. The other is simply set to zero.*

*Note: Even if you will be using the X or Z coordinate, you need to make sure that the cycles counter value is set to a high enough number so there is enough time for the entity to reach their X or Z coordinate.*

*In the example above, a Z Position Target has been set for a terrain coordinate of 45000, along with a cycles counter value of 25000. Regardless of the cycles counter, once the Z 45000 is reached the step ends. However, if the cycles counter was set to a lower number, the cycles counter could run out before the Z 45000 was reached. In this case the step would end before you arrived at your desired location.*

### **Minimum Value Above Terrain**

A value that establishes the lowest level the entity can descend above the terrain. This is the opposite of Maximum Altitude. It also allows the developer to have the entity follow the ups & downs of the terrain, such as a car would do when driving over a bumpy road.

### **Turning: Entity Turn Increments and Rotation** (data steps 11 and 12)

At the beginning of each step the operation code examines these two turning values. If a turn is indicated these two values determine..

The number of incremental steps  
needed to complete the turn.

Clockwise or counter-clockwise direction of the turn.



In the above example, the starting direction is set to 8. However a turn is going to be activated because Data Point 11 is set to 1, and Data Point 12 is set to negative one (-1). This means that the entity will turn one step, and will do so in a counter clockwise direction.

*Note: When the #11 data point is set to zero there is no turn.*

*Note: The directional code (first data entry point) operates separately from these turning codes. When a step is activated the movement will take place in that specified direction regardless of the settings for the turn codes. Thus, you can change direction without turning the entity, or turn the entity without changing direction. This allows for sideways movement like a horizontal elevator or the turning of a gun turret.*

It is important to note that the value used for data point 11 **IS NOT** a directional code. It is a number that tells the engine how many steps are required to complete the turn. This can be a value between 1 and 7. This will be explained in the next section.

Data entry point 12 is set to -1. This tells the script it will turn the entity counter clockwise. +1 value would turn in a clockwise direction.

## **Sound Number and Sound Volume**

These are values that allow the script to activate a sound while the sequence is playing, and control the sound's volume.

*Note: The sound can only be activated by the **first step** of the movement sequence. Entering sound settings in other steps does nothing. The sound begun in Step #1 continues until the sequence is complete, or the sound file ends.*

## **Enhanced Sound Controls**

The MCS offers some additional controls for using sound. Even though the MCS is an entity movement system, it can be set-up so the entity does not move, but the sound can still be activated. Using this in conjunction with the Proximity Start setting, allows the desired sound to be activated when the player comes near the MCS entity.

In addition, the MCS has a special built-in global control (the brain child of Amen Moses .. Thank You). The player has activated a sound. The player continues moving while the sound is playing when the player encounters another MCS entity which starts a second sound.

Once the 2<sup>nd</sup> sound begins, the script reaches out and stops the 1<sup>st</sup> sound from playing. Thus, two sounds will not be playing over one another. If the player returns to the 1<sup>st</sup> MCS entity, the 1<sup>st</sup> sound plays and the 2<sup>nd</sup> sound is stopped automatically.

## **Data Entry Points Not In Use**

Data Entry points 7, and 15 – 19 are set to zero and are currently there for future expansion.

## **The Number of Steps in the Sequence**

The MCS script contains the data entry points for 16 steps. Depending upon your movement plan a Sequence can be set for one to sixteen steps. (see below)

## **Overall Sequence Data Entry Points**

Once all of the step data has been completed, there are just a few additional pieces of data to enter. They look like this,

```

-- ***** BASIC SEQUENCE VALUES Entered by User *****
-- *****
local StepTotal = 1      -- Set the number of steps in this sequence

local StartCode = 1      -- 1 = Cam/Entity E Start  2 = Entity only E Start
                        --3 = Entity Only auto Start
                        -- 4 = Entity Only Proximity to Player activates sequence

local ProxVal = 150      -- if StartCode = 4 the ProxVal equals the distance of the
                        --player at which the sequence will begin. 150 default up to 800

local EntOnlyFinalState = 1  -- = 1 no repeat entity remains inactive
                        -- = 2 no repeat entity destroyed
                        -- = 3 entity returned to original position, Sequence
                        -- restarts at step 1

local SndLoop = 1         -- 1 = Play sound once  2 = Loop sound
                        --NOTE if no sound set to 1

local CamPosAdjX = 0; local CamPosAdjY = 20; local CamPosAdjZ = 0
                        -- enter +/- numbers to adjust Camera
                        -- to Entity position (StartCode = 1)

PlyrDismountX = 0; PlyrDismountZ = 0  --determines the X and Z distance
                        --from the entity where the Player/Camera
                        --reunite and return to play (StartCode = 1)

*****

```

### **Step Total:**

Number of steps in the sequence.

### **Start Code:**

Determines how the script will begin;

- 1= Camera and Entity move together with "E" start
- 2= Entity moves alone with "E" start
- 3= Entity moves automatically
- 4= Entity activated by Proximity to player.

*Note: The term “E Start” refers to the Game Guru use of the “E” key to activate the Sequence. Codes 1 and 2, can be activated when the player gets close to the Entity and is prompted to press the E key. Code 3, Entity movement starts automatically. Code 4, the Sequence is activated based upon the proximity to the player to the entity.*

## **Entity Only Final State**

When the movement sequence ends the developer can choose one of three final states for the entity;

The entity remains where the sequence ends (inactive)

The entity is removed or destroyed. (vanishes)

The entity is returned to its original start position and the sequence can be used again.

*Note: If the Player/Camera is in ride-a-long mode ( StartCode = 1) when the Sequence ends, the player is decoupled from the Entity, returning the player to normal game mode at the new location.*

## **Camera Position Adjustments (x,y,z)**

When the camera is paired with the entity, (StartCode = 1) that pairing is based upon the 3D center (origin point) of each. Linking the position of the camera and entity, you often find the camera and entity in an odd alignment.

To compensate use these variables

**(CamPosAdjX = 0 ; CamPosAdjY =35 ; CamPosAdjZ = 0)**

These values adjust the position of the camera relative to the entity, during the combined movement sequence. You may have to run a few tests to find the best settings.

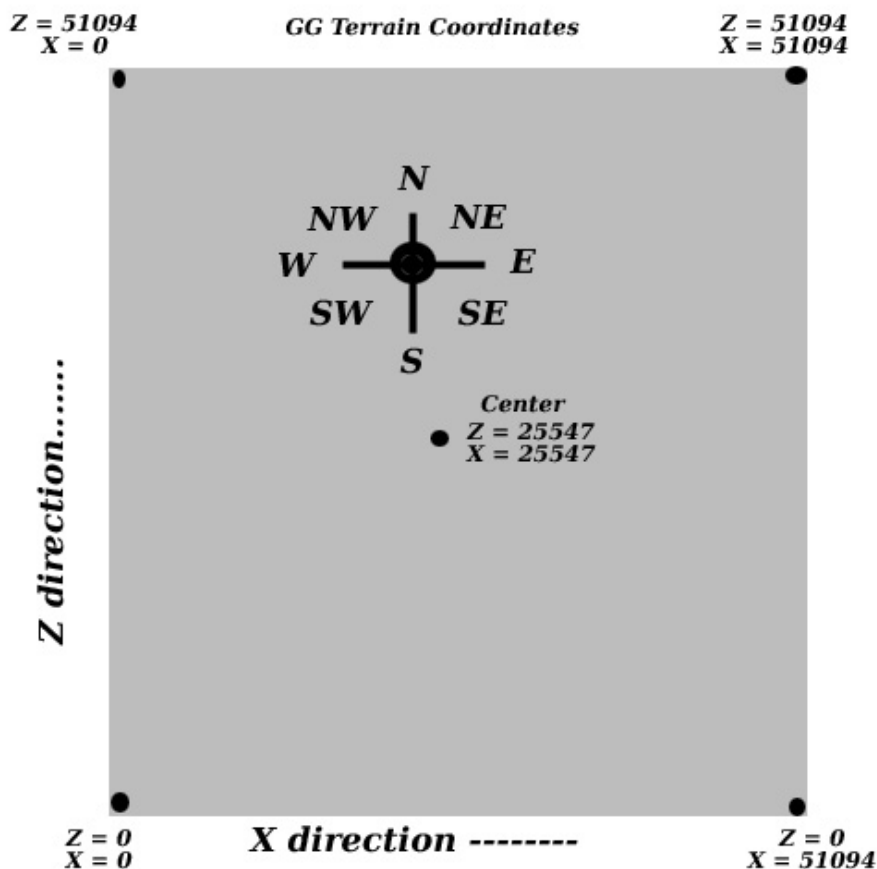
## **PlyrDismount X & Z**

When a Player ride-a-long sequence ends (StartCode = 1), the Player and Camera are reunited, and normal game play resumes at the new location. The PlyrDismountX and PlyrDismountZ represent additional distances, away from the entity, where game play resumes. This let's the developer determine where play will resume. Positive and negative values can be entered.

## Understanding Movement Direction Codes with MCS

In order to set the directional data it is important to understand the Game Guru's terrain layout.

The GG terrain uses terrain coordinates to determine where on the terrain an entity is located. These are defined in the script as X, Y, and Z coordinates. The Y coordinate represents up and down. The X and Z coordinates represent; forward and back (Z), left and right (X).



Using the MCS script there are 8 directions which can be set. Values 1 – 8 represent the following directions for the MCS script.

--	MCS Directional Codes							
--Direction	N	NE	E	SE	S	SW	W	NW
--Direction Code	1	2	3	4	5	6	7	8

If you want the Entity to move in a particular direction, you need to determine the correct directional code (data entry point 1), and determine if and how you want the entity to turn (data entry points 11 & 12).

### **Finding X Y Z Coordinates on the Game Guru Classic Terrain**

In Game Guru Classic there is no (usable) read-out for XYZ coordinates in the editor. How does one find the coordinate values for use in MCS?

Enclosed with MCS are two small scripts titled; **aaplayerxyz.lua**, and **aaentityxyz.lua**. These scripts can be attached to an entity like a box. When activated in the game test mode, a running display of the players XYZ coordinates or the entities XYZ is displayed across the lower edge of the display.

*Note: You can only use one of these scripts at a time.*

Move the player around the terrain to the places you want your movement to go. Note the X/Y/Z values and input them into the MCS script as required.

## **MCS Pristine Files**

the MCS script comes as a pristine file ready for use. It is highly recommended that you make a copy of the pristine file and put it away somewhere just in case. Be sure to rename new copies before beginning the data entry process.

## **Making Copies and Renaming a LUA Script**

**KEEP THE PRISTINE FILE SEPARATE**

Saving and renaming a .lua script is a two step process.

First, save the pristine file under a different name.

Remember: .lua file names  
must be in all lower case.

MyNewFile.lua	WRONG
mynewfile.lua	RIGHT!

Second, you **MUST** also change two lines of code inserting the new file name before these statements;

```
_init(e)  
_main(e)
```

Scroll down to (approx) LINE NUMBER 489 where you will find the \_init(e) and \_main(e) statements. Change the old file name to match the new file name you just saved.

oldfilename_init(e)	to	mynewfile_init(e)
oldfilename_main(e)	to	mynewfile_main(e)

--Save the file and you have a clean copy of the MCS ready for use.



## Sample Tutorial MCS Script

Included is a file titled; aamcsv6tutorial01.lua

*Like the pristine files, I strongly suggest making a back-up copy of the tutorial file as well. Remember to change the `_init(e)` and `_main(e)` statements with the new name.*

Sometimes the best way to learn something is to just jump-in and see how it works. This MCS file has a 4 step sequence already programmed. This is the four step program we discussed earlier.

- Move (fly) to first location
- Turn and move to 2<sup>nd</sup> location
- Turn and move to 3<sup>rd</sup> location
- Land Vertically on the terrain

Attach this file to the entity you want to move and fire up Game Guru's test engine and see what it does. Use a Entity like a barrel on an open flat terrain.

Go back to the data inputs in your text editor and change things. Increase the speed. Change the ascend/descend speed. Change the max altitude, etc. Save the altered file and test it again.

The following is the 4 step data entry points for the tutorial script, as well as the Basic Sequence settings..

```
-- ***** BASIC SEQUENCE VALUES Entered by User *****
-- *****
```

```
local StepTotal = 4      -- Set the number of steps in this sequence

local StartCode = 2      -- 1 = Cam/Entity E Start  2 = Entity only E Start
                        --3 = Entity Only auto Start
                        -- 4 = Entity Only Proximity to Player activates sequence

local ProxVal = 150      -- if StartCode = 4 the ProxVal equals the distance of the
                        --player at which the sequence will begin. 150 default up to 800

local EntOnlyFinalState = 1  -- = 1 no repeat entity remains inactive
                        -- = 2 no repeat entity destroyed
                        -- = 3 entity returned to original position, Sequence
                        -- restarts at step 1

local SndLoop = 1          -- 1 = Play sound once  2 = Loop sound
                        --NOTE if no sound set to 1

local CamPosAdjX = 0; local CamPosAdjY = 20; local CamPosAdjZ = 0
                        -- enter +/- numbers to adjust Camera
                        -- to Entity position (StartCode = 1)

PlyrDismountX = 0; PlyrDismountZ = 0  --determines the X and Z distance
                        --from the entity where the Player/Camera
                        --reunite and return to play (StartCode = 1)
```

```
*****
```

These basic sequence settings are important. In this case we have 4 steps. The start code = 2 so the entity will only begin it's movement when the player gets close and is prompted to press the "E" key. This activates the 4 step sequence which will continue to run until conclusion.

The EntityOnlyFinalState = 1. The sequence plays and where ever the entity ends up, that's where it stays! Test out the other options. Since there is no sound in this sequence SndLoop is left to it's default =1

The CamPosAdj and PlyrDismount variables are not used in this test. These are used when StartCode = 1, where the player and entity will ride together. See information above.

## --Data Entry Points for each Step of movement Sequence

\*\*\*\*\*

### --Movement Step #1 (1-20)

ObjcontE1[1] = 3	-- Direction code for this step 1-8 (See above)
ObjcontE1[2] = 2	-- X/Z speed value 0 - 15
ObjcontE1[3] = .5	-- Ascend and descend speed -3 to +3
ObjcontE1[4] = 600	-- Number of game cycles for this step
ObjcontE1[5] = 50	-- Delay before this step begins values 0 - 1000
ObjcontE1[6] = 800	-- Max Altitude
ObjcontE1[7] = 0	-- Not in use leave value set to zero
ObjcontE1[8] = 0	-- X Position Target not in use, set to zero **only one X or Y value per step**
ObjcontE1[9] = 0	-- Z position Target not in use, set to zero **only one X or Y value per step**
ObjcontE1[10] = 10	-- Minimum height above the terrain
ObjcontE1[11] = 0	-- No Rotation = 0 Rotation Steps (1-7) example turn from Dir1 to Dir3 = 2 Rotation Steps
ObjcontE1[12] = 0	-- No Rotation = 0 Rotate Clockwise enter 1 or Rotate Counter Clockwise enter -1
ObjcontE1[13] = 4	-- Sound Play 4=No Sound or enter Sound Number 0-3 Sound linked in GG properties
ObjcontE1[14] = 0	-- Sound Volume 80 - 100
ObjcontE1[15] = 0	--
ObjcontE1[16] = 0	
ObjcontE1[17] = 0	
ObjcontE1[18] = 0	
ObjcontE1[19] = 0	
ObjcontE1[20] = 21	-- ObjcontE1 number pointer DO NOT CHANGE !!!

### Comment:

The Entity will first move to the East (direction code = 3). Be sure to place the entity so the front is facing in the code 3 direction. The entity will begin with a forward speed of 2, meaning that the entity will move forward two grid values with every every game cycle. The entity will also ascend at a rate of  $\frac{1}{2}$  (.5) grid values with each game cycle.

Please note entry point 6 “Max Altitude”. This works in conjunction with Ascend and descend. When the entity reaches the Max Altitude, it stops ascending and establishes level flight at the Max Altitude value, ignoring the ascend value.

The first step is set for a length 600 game cycles. There is also a delay value. This delays the execution of the step for a number of game cycles, before executing the step (all 600 cycles). There are no X or Y target values (see manual). No turn and no sound.

## --Movement Step #2 (21-40)

ObjcontE1[21] = 1      -- Direction code for this step 1-8 See above)  
ObjcontE1[22] = 2      -- X/Z speed value 0 - 15  
ObjcontE1[23] = 0      -- Ascend and descend speed -3 to +3  
ObjcontE1[24] = 600      -- Number of game cycles for this step  
ObjcontE1[25] = 0      -- Delay before this step begins values 0 - 1000  
ObjcontE1[26] = 800      -- Max Altitude  
ObjcontE1[27] = 0      -- Not in use leave value set to zero  
ObjcontE1[28] = 0      -- X Position Target not in use, set to zero  
                         \*\*only one X or Z value per step\*\*  
ObjcontE1[29] = 0      -- Z position Target not in use, set to zero  
                         \*\*only one X or Z value per step\*\*  
ObjcontE1[30] = 10      -- Minimum height above the terrain  
ObjcontE1[31] = 2      -- No Entity Rotation = 0 Rotation Steps (1-7)  
                         example turn from Dir1 to Dir3 = 2 Rotation Steps  
ObjcontE1[32] = -1      -- No Entity Rotation = 0 Rotate Right enter 1  
                         or Rotate Left enter -1  
ObjcontE1[33] = 0      -- Sound set in step 1 only  
ObjcontE1[34] = 0      -- Sound set in step 1 only  
ObjcontE1[35] = 0  
ObjcontE1[36] = 0  
ObjcontE1[37] = 0  
ObjcontE1[38] = 0  
ObjcontE1[39] = 0  
ObjcontE1[40] = 41      -- ObjcontE1 number pointer DO NOT CHANGE !!

## Comments:

OK, we've completed the first step. Note that step 2 does not have a delay so execution goes from one step to the next without delay. In Step 2 we change the direction code to north (DC = 1). There is no ascend value so the entity will fly level at this level.

## Entry Point 31 and 32

As soon as step 2 begins the entity will change direction. The at the same rate forward of 2 units per cycle. Using entry points 31 and 32 you can have the entity turn. In this case turn in the new direction. EP(31) tells the engine to turn 2 steps of the eight steps in the directional codes. (from 3 to 1).

--	MCS Directional Codes							
--Direction	N	NE	E	SE	S	SW	W	NW
--Direction Code	1	2	3	4	5	6	7	8

EP(31) is a flag to tell the engine that two steps (in this case) is needed for this turn. EP(32) is a flag telling the engine which way to turn the entity clockwise (=+1) or counter clockwise (= -1) In this case the turn is counter clockwise.

**--Movement Step #3 (41-60)**

ObjcontE1[41] = 2      -- Direction code for this step 1-8 See above)  
ObjcontE1[42] = 2      -- X/Z speed value 0 - 15  
ObjcontE1[43] = 0      -- Ascend and descend speed -3 to +3  
ObjcontE1[44] = 400      -- Number of game cycles for this step  
ObjcontE1[45] = 0      -- Delay before this step begins values 0 - 1000  
ObjcontE1[46] = 800      -- Max Altitude  
ObjcontE1[47] = 0      -- Not in use leave value set to zero  
ObjcontE1[48] = 0      -- X Position Target not in use, set to zero  
                            \*\*only one X or Z value per step\*\*  
ObjcontE1[49] = 0      -- Z position Target not in use, set to zero  
                            \*\*only one X or Z value per step\*\*  
ObjcontE1[50] = 10      -- Minimum height above the terrain  
ObjcontE1[51] = 1      -- No Entity Rotation = 0 Rotation Steps (1-7)  
                            example turn from Dir1 to Dir3 = 2 Rotation Steps  
ObjcontE1[52] = 1      -- No Entity Rotation = 0 Rotate Right enter 1  
                            or Rotate Left enter -1  
ObjcontE1[53] = 0      -- Sound set in step 1 only  
ObjcontE1[54] = 0  
ObjcontE1[55] = 0  
ObjcontE1[56] = 0  
ObjcontE1[57] = 0  
ObjcontE1[58] = 0  
ObjcontE1[59] = 0  
ObjcontE1[60] = 61      -- ObjcontE1 number pointer DO NOT CHANGE !!

**Comments:**

Another smaller turn to direction DC = 2. Same speed, same altitude, and a length of only 400 on this third leg. Again a turn of one step (1 to 2), and this time the turn is clockwise.

**--Movement Step #4 (61-80)**

ObjcontE1[61] = 2      -- Direction code for this step 1-8 See above)  
ObjcontE1[62] = 0      -- X/Z speed value 0 - 15  
ObjcontE1[63] = -.5      -- Ascend and descend speed -3 to +3  
ObjcontE1[64] = 500      -- Number of game cycles this step  
ObjcontE1[65] = 100      -- Delay before this step begins values 0 - 1000  
ObjcontE1[66] = 800      -- Max Altitude  
ObjcontE1[67] = 0      -- No in use leave set to zero  
ObjcontE1[68] = 0      -- X Position Target not in use, set to zero  
                         \*\*only one X or Z value per step\*\*  
ObjcontE1[69] = 0      -- Z position Target not in use, set to zero  
                         \*\*only one X or Z value per step\*\*  
ObjcontE1[70] = 10      -- Minimum height above the terrain  
ObjcontE1[71] = 3      -- No Entity Rotation = 0 Rotation Steps (1-7)  
                         example turn from Dir1 to Dir3 = 2 Rotation Steps  
ObjcontE1[72] = 1      -- No Entity Rotation = 0 Rotate Right enter 1  
                         or Rotate Left enter -1  
ObjcontE1[73] = 0      -- Sound set in step 1 only  
ObjcontE1[74] = 0  
ObjcontE1[75] = 0  
ObjcontE1[76] = 0  
ObjcontE1[77] = 0  
ObjcontE1[78] = 0  
ObjcontE1[79] = 0  
ObjcontE1[80] = 81      -- ObjcontE1 number pointer DO NOT CHANGE !!

**Comments**

At this point the entity is hovering. There is a short delay and then the entity begins a vertical descent to the terrain. While descending there is a three step rotation so the upon landing the entity will be facing south (DC =5). Just to make it look flashy!

That's it. The 4 steps are complete, The entity now rests in it's final location.

## Using MCS for Sound Triggers and Control

MCS has a robust sound system. It may seem funny to use a movement script to control sounds. However, the step controls do not require a value for either movement or ascend/descend, so the entity to which the script is attached, just sits there and controls the playing of sounds.

In the Basic Sequence Values you can determine if the sound will play once or loop..

**local SndLoop = 2                  - 1 = Play sound once    2 = Loop sound**  
**--NOTE if no sound set to 1**

You can use the Proximity StartCode = 4. When the player comes near the entity the playing of the sound is activated.

**local StartCode = 2**      -- 1 = Cam/Entity E Start    2 = Entity only E Start  
                              --3 = Entity Only auto Start  
                              -- 4 = Entity Only Proximity to Player activates sequence

You can only begin a sound in step 1. If entry point 13 = 4 (the default) no sound will play. If you plan to play a sound then EP-13 equals the slot number (0,1,2,3) in the entities properties where the sound file is referenced. In this case Slot #1, with a volume set to 100 (max).

**ObjcontE1[13] = 1**      -- Sound Play 4=No Sound or enter Sound Number 0-3  
                                  Sound linked in GG properties  
**ObjcontE1[14] = 100**      -- Sound Volume 80 – 100  
**Note: Anything below 80 is basically zero volume.**

## How Can I use this?

Let's say you set-up an entity with a sound script of one step. Set to the Proximity start, when the player comes close the sequence is executed, and a background sound plays or loops.

You continue on and down the way the player trips a second script that begins a different sound. The problem is that sound one may still be playing. We want to stop the first sound from playing, so the 2<sup>nd</sup> sound can be heard. Not such an easy task.

Each MCS script has within a Global Variable which keeps track of the last sound one of these scripts started. If a new script sequence starts before it plays the new sound, it checks if the last sound is playing. If it is, the earlier sound is stopped, and the new sound plays.

In this way you can use the MCS to begin background music, or other sounds (.wav or .ogg sound formats). Thus, as the player moves through the map MCS sound entities can be scattered to fit the situation, but will only play one sound at a time.

### **MCS Guarantee & Support**

The MCS scripts are provided free as a public domain entities. You may use them, give them to others, modify them (at your own risk). You may not sell them!

THIS SOFTWARE COMES WITH NO GUARANTEE OF ITS OPERATION, NOR WILL THERE BE ANY SUPPORT BEYOND THE PROVIDED TRAINING MATERIALS.