Understanding and Using the MCS LUA Script in Game Guru
By GoDevil

What is MCS?

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

Further, this script does not require complex coding by the user. The movement and control coding has already been completed. All the user/developer does is to enter data that the script will use to complete the desired movement tasks.

Finally, by making copies, renaming the script, and entering new data, the same script can be used over and over with different entities, to add life, movement, and a new level of camera control to your game.

NOTE: MCS does NOT allow the player to control the movements defined in the scripts data. Once the script is activated it follows the instructions, step by step, from the data the developer entered. During operation the arrow keys do not function. More on this later.

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. 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 script in turn. This cycle happens over and over again during game play.

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 object to the preset location in the 3D game.

Each cycle moves the entity a curtain distance and this continues until the entity has reached its goal. At this point the script is designed to the tell the engine that it has arrived at it's 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 (unless you know what you're doing).

There are two pieces of the operational code that you will need to change. When you make a copy of the basic script to use in the movement of 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.

Using the MCS script the game developer's job is to plan and enter the DATA for your movement script(s). If this data is entered correctly the script is capable of the following functions.

Move and entity forward 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 by itself or while moving forward
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 three basic operational formats.

Move the entity by itself after being activated by the player

Move the entity by itself activated automatically

Move the entity and the player at the same time.

Player / Entity Combined Movement
Regarding this last format, it is important to understand how this player ride-a-long can happen. 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) loose 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 scripts final location, it is automatically reset to the normal player functions. Thus, you can move the player or fly the player from one location to another. Once the movement is completed the MCS script returns to normal player/camera operation at the new location. This function can be used for driving, flying, boats, elevators, even horizontal elevators.

To understand this I have put together a video that demonstrates all three of the above noted modes. (Put video link here)

Planning Your MCS Script; Sequence and Steps

Here's a simple movement plan. I want the entity to move from it's starting position to a new location. Once it arrives at the that location I want it to turn 90 degrees to the left and continue in this new direction until reaching another preset location. Then I want it turn right 45 degrees, and continue to a new final location.

Overall this plan is referred to as a **Sequence**. Inside that sequence are three distinct **Steps**...

Move to first location
Turn and move to 2nd location
Turn and move to 3rd and final location

The MCS script provides data entry points for a Sequence of up to 16 Steps. For each one of these steps the developer has the ability to enter all of the movement data functions 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 movement process continues until all of the steps are completed. That may be three steps, ten steps, or just one step. One of the data pieces you will enter is the number of steps in the sequence that tells the

operational code when the sequence is complete.

This gives the developers a great deal of versatility in the type of movement scripts they can create just by entering data. There is even a drop-off function, where the transport (for example) takes the player to a new location and drops him/her off at the new location, resetting the player to normal game mode.

In such a script the entity's movement can continue even after the player is no longer attached. After a short delay the entity (Transport) can continue it's journey without the play attached.

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] = 9         -- Sound Number (0-7)  or 9 = 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 !!!
```