Basic Concepts of LUA scripting

Based on an original document by Joel Segal - with thanks.

GameGuru is a software package which uses the Microsoft Direct X system to simulate a 3D environment in your computer. Using models, scripts, and other processes of the GameGuru Engine, a user/developer can create a 3D game. (We actually like to think of it as a Sandbox game that can also make proffesional quality games with.).

Using the expanding command structure in GameGuru and a Scripting language called LUA, you can create more complex action/strategy games. The purpose of this guide is to explore at a very basic level the use of the LUA language with GameGuru.

Keep in mind that GameGuru at the time of this writing is still in the beta development cycle, so not all of the structure of GameGuru has yet to be completed.

Basic LUA Concepts

Think of the LUA scripting language as a program inside a program. The main program is the 3D Engine of GameGuru. Ultimately, everything you see on the screen, from models, to character's, to movement, to lighting, etc., are controlled by the GameGuru engine. Built-in to the engine are command structures (called Common Functions) which can communicate with the outside world and through which curtain processes of the GameGuru engine can be controlled.

It would be impossible to build the GameGuru engine to take into consideration every possible twist and turn game creators might dream-up.

LUA is the "inside" programming language that communicates with the GameGuru engine. The LUA scripting commands can be specific to the story, strategy, and actions of the "Game" being developed, providing control and activation of dozens, even hundreds of activities, which will be performed by the Engine, as requested by commands from a LUA script.

The real power of the LUA language is its ability to carry-out it's own set of control and simulation code, controlling game play by communicating with GameGuru beyond the control of the Engine. This allows for the development of very complex simulations which can be played-out in full real-time 360 degree 3D.

LUA is a very complete yet compact language, allowing the script writer a wide range of useful possibilities. (I hate to use the word "programmer", as it seems to frighten people, where "Script Writer" seems so much softer.)

"Hey...They Said There Would Not Be Any Coding"

OK, so GameGuru is advertised as the "Easy To Use Game Engine, and it can be. Using basic entities, and characters, and with only a little knowledge of how the Engine and LUA scripts work, one can select, build, and populate a 3D environment, and build a

working game with no major script-writing. So, if all you want is to set-up an environment and shoot AI controlled zombies all day, have at it, GameGuru's got you covered.

"Two Other Ways To Think About LUA"

First, LUA can become a very powerful tool if you learn a little about the scripting process. With this knowledge you can make modifications to existing scripts. Changing values, adding additional display data (Prompt command), modifying the code so the script does something you want.

Second, is the jump in with both feet, LUA deep-end, or finally breaking down and learning to script. Creating your own scripts, and using LUA to control curtain aspects of the game. For example, the ability to build a simple data structure (a Table in LUA speak), and use that data in the conduct and evolving strategy of your game.

I suggest that most people get started with step one, which ultimately makes step 2 a bit easier, rather than diving into LUA coding-land. Those who have other coding disciplines (C, Basic, other scripting languages) will find LUA to be pretty straight forward and relatively easy to construct. (I'm from the DARK-BASIC side of the force.. There is no spoon!)

The point is, beyond shooting zombies all day, GameGuru has (or will have) deeper levels of control, allowing the game creator to control the game, not just fighting against an AI script running inside the GameGuru engine.

Back To The Basics

A Lua program (you create or modify) is called a "Script". It is a simple text file, with the naming convention of "name.lua".

TID-BIT: The version of LUA running with GameGuru requires that all LUA script file names be in all lower case or numbers. You'll see later why this is very important.

Name.lua --NO GOOD, **name.lua** --Correct

NotePad, or NotePad+ (which is free and designed for LUA style scripting), are all that is needed to write and/or modify LUA scripts.

Most LUA scripts are assigned or attached to specific entities that are part of the 3D game. For the most part these Entities are Direct X models which populate the 3D world. A Lua script is attached as a property of each Entity before the game is run. Some times these are default scripts that do nothing, while other scripts can have very profound effects on the Entity to which it is attached or the course of game play.

The Play of the Game

The 3D environment in which the game will be played has been populated with dozens of Entities and Characters. Each Entity and each Character has a LUA script attached to them. These scripts can be attached to the Entity via the "FPE" file, or scripts can be changed in GameGuru's Properties Editor.

In part, the function of the game engine is to organize and run these LUA scripts, which ultimately constitutes the major part of the game play. In programmer terms it's called a;

"Do Loop". Once everything is in place and working, the game begins. One of the processes of the engine is to take a quick look at each and every LUA script. This process is on-going from one script to the next, repeated (looped) over and over again at very high speed. This looping through the LUA scripts, plus the integration of commands coming from the mouse and keyboard (like move forward, look left, etc.) represents a large part of the process of playing the game.

Scripts for static objects (like buildings that do not move, or parked cars, etc.) do nothing and their LUA scripts are all but ignored, while other scripts can provide the engine with commands based upon some aspect of the play of the game (usually related to

the Entity or character the script is attached too)

TID-BIT: GameGuru has two types of Entities, Static and Non-Static. In order for a script to operate the Entity to which it is attached must be Non-Static. This does not mean that the entity will move. For example I could attach a LUA script to a building that feeds back a screen Prompt() providing the current XYZ coordinates of the Player. There is no intention to move the building, but in order for the script to work, the building must be set-up as Non-Static. This too can be done through the FPE file or using the Properties Editor.

The Loop Continues

Based upon these conditions, the script either issues a command(s) or does not. When finished that script ends for the moment. A few micro-seconds later the game loop comes back around and depending upon any game condition changes since the last go round, the script is once again run, issuing or not issuing commands, and it ends again. The looping process continues until the game's over. Thus a LUA script is looked at and run by the GameGuru Engine thousands of time a second.

It is this looping environment that provides the script writer with timing and circumstantial information, used by the script to decide how and when to issue a command or request to the engine. (move a vehicle, the location of the player, an explosion, etc)

GameGuru and LUA - the hard and fast rules.

Scripts and Functions

Most scripts are in the form of "Functions". A Function is a common programming practice. The Function is a structured piece of coding. Encased within the Function is the coding that makes the script work. A single LUA script can contain many Functions, but 99% of the time there are just two Functions;

1) Function name_init(e)

This "init" Function is used at the very beginning of the game as the Engine sorts out the LUA scripts it will be running (looping). It is also an opportunity for the script writer to set-up special values which the remainder of the script (and other scripts) can use during actual game play.

2) Function name_main(e)

The "main" Function provides the code that will be looped through during game play.

TID-BIT: In GameGuru every Script MUST include the "init" and "main" Functions (there are some scripts that do not need these basic function set-ups as we will discuss later). The name of the LUA script must be EXACTLY the same of the name used in both required Functions;

File Name... name.lua (remember all lowercase)

Function name_init(e)
end

Function name_main(e)
end

OK, So You Caught Me

There I go pushing in some of that coding stuff again. Two things of note here;

1) Your introduction to “e”. Not to worry, “e” is referred to the the “Entity Index”. It is a universal value assigned and used by the engine to identify the entity under consideration in the script. You don’t need to know what “e” is, you just need to get it in the right places so everything works smoothly.

2) “end”. Every Function must have an “end” command. This concludes the Function and allows the engine to move-on to the next script in the looping process. In LUA you will find “end” to be both friend and trouble maker. LUA uses a lot of “if/then” (conditional) statements, and each if/then statement also requires an “end” command. It is not uncommon to have a script with dozens of “end” commands. As you learn LUA, learning to place your “end” commands properly is important.

Let’s Think About A Simple Function

You have an entity with a Lua script. The player moves towards this entity. Each time the script is looped a simple math equation determines how far the Player is from the entity. When the player gets close, almost magically a message appears on the screen saying “You Have Arrived”.

Inside this script’s main Function is a coding sequence like this..

_____

-- Main House Script

Function house1_init -- required but in this case does nothing

end

Function house1_main(e)

PlayerDist = GetPlayerDistance(e) -- this line calls to another Function

if PlayerDist < 100 then

Prompt(“You Have Arrived”)

end

end

_____

TID-BIT: In LUA code any text which is preceeded by -- (two dashes) is called a Remark. Remarks are there for the script writer so you can “remember what the hell you were doing two months ago”. During operation the remarks are ignored. Let’s look at the above function step by step.

First we set-up the function with a name which is the same name as the Lua script (house1.lua), which is attached to a specific Entity (represented by (e)). Note the two “end’ commands, one for the if/then statement, and the other to close the main function.

Second, the script puts out a call to a different Function called; GetPlayerDistance(e). This function is located in a different script, but it can be called from any script in the game when needed.

What returns from the external Function “GetPlayerDistance(e)” is a value now represented by the variable PlayerDist. (We’ll discuss the secret location of this and other Common Functions in a little while.)

Next we put that distance to the test to see if the Player is close enough to the entity to receive our Prompt. “IF” the value of PlayerDist is less than (<) 100 units then show the prompt. However, if the value of PlayerDist is greater than 100, the prompt statement is passed over, the function ends, and the game loop continues. When it next comes around, and the Player is still outside the 100 unit distance, No Prompt For You! If the player moves inside the 100 unit distance, the Prompt appears. If I go back out of the area, the Prompt disappears.

"Some LUA scripts are different"

In the GameGuru Scriptbank folder is a script called “global.lua”. This script is not attached to an entity, and serves as a central LUA depository for the declaration of variables, in particular “Global Variables”, Table Variables (name = {} set-up a table), as well as providing a centralized home for dozens of “Common Functions” such as the GetPlayerDistance(e) function. These common functions represent the interaction between the script and the Engine .

It is through this global script that most of the connections reside which allow the LUA scripts to control game play, by sending commands and values to one of the external common functions in global.lua. The Prompt function for example, prints a prompt on the screen. The actual printing of the message is accomplished by the GameGuru Engine, but the trigger to display the prompt and the message to be displayed are provided by the LUA script. The LUA command Prompt(“Hi this is a prompt”), calls the global.lua Function Prompt() and the engine displays your text.

A Script writer can create and include their own “myglobal.lua file in addition to the global.lua file. This script can declare other global variables and common functions specific to your game. With these tools one can create sophisticated script systems that will make your gaming experience challenging and enjoyable.

LUA Script example.

Here is a LUA script that displays a prompt which provides the X Y Z locations of the player. Since the player does not have it’s own script, this script is attached to another Entity in the 3D environment, say a building.

_____

```
-- Script to display current location of player

function aaplayerxyzprompt_init(e)

end

function aaplayerxyzprompt_main(e)

x1=math.floor (g_PlayerPosX)

y1=math.floor (g_PlayerPosY)

z1=math.floor (g_PlayerPosZ)

Prompt("X= "..x1.." Y= "..y1.." Z= "..z1)

end
```

_____

We’ll take this script step by step shortly, but let’s first look at the attachment process.

The GameGuru File System

Like many software programs GameGuru has a file system which is specific to the engine. These files are organized and named in such a way, and placed in a specific location, so that the engine can find what it needs. This file system can be found in the program folder where you installed GameGuru. It is a folder called “Files”.

Inside the Files folder are a group of “Bank” folders that hold different types of files. The scriptbank holds the LUA scripts. The mapbank holds the map (levels) files. The entitybank holds the models and textures which are available to be loaded into your 3D environment.

In order to make a entity ready to load, a specific entity requires a minimum 3 files in order to be successfully loaded and displayed.

1) The dot X (.x) file which contains the actual mesh of the model. Train.x

2) A texture file using the “_D.dds” format. Train_D.dds.

(please note the _D.dds format as this is important to GameGuru)

3) The FPE file Train.fpe.

The FPE File explained

Every entity that will end-up in your 3D environment requires an FPE file. If you look into the entitybank folder to a specific entity you will find an FPE file for each entity. Like a LUA script, the FPE is a simple text file saved using the .fpe form.

The FPE file is a group of initial instructions that tells the GameGuru engine how to set-up your entity in the 3D environment. Here is a typical FPE file.

;header
desc = DodgeChargerC2

;visualinfo
textured = LPVcar2_D.dds

effect = effectbank\GameGuru\entity_basic.fx
castshadow = 1

;orientation
model = DodgeChargerC2.x
offx = 0
offy = 0
offz = 0
rotx = 0
roty = 0
rotz = 0
defaultstatic = 1
materialindex = 3
collisionmode = 1

;statistics
strength = 25
explodable = 0

debrisshape = 0

;ai
aiinit = appear.lua
aimain = default.lua
aidestroy = disappear.lua

;spawn
spawnmax = 0
spawndelay = 0
spawnqty = 0

There are many additional command settings that can be used in an FPE file. For now note the following...

1) The use of the desc line should also be the name of the FPE file. It is important that the name of the FPE file be the same as the name of the model .x file, in this case DodgeChargerC2, the same as DodgeChargerC2 (.x). (the FPE file is not subject to the lowercase bias of LUA scripts as far as naming is concerned.)

2) The name of the Texture file using the _D.dds form. Note that the Texture file, the FPE file and the .X file all need to be in the same folder inside the entitybank folder, in order to load the entity into the environment.

3) Under the “ai” section, note the aimain script =. This is the LUA script which can be attached to the entity in the FPE file. Also note that a specific script can also be attached to the entity using the Properties Editor in GameGuru. (to be discussed shortly)

4) Note the “effect” line... entity_basic.fx. This too is a type of script which defines the type of entity to be loaded, and how GameGuru will work with that entity. There are two of these scripts; entity_basic.fx and character_basic.fx.

5) Note the “defaultstatic=:”. Remember before when we discussed Static and Non-Static entities. This is where 1= static mode, and 0=non-static mode. This too can be changed in the Property Editor.

OK, Let’s Return To The Prompt LUA Script

_____

```
1 -- Script to display current location of player

2 function aaplayerxyzprompt_init(e)

3 end

4 function aaplayerxyzprompt_main(e)

5 x1=math.floor (g_PlayerPosX)

6 y1=math.floor (g_PlayerPosY)

7 z1=math.floor (g_PlayerPosZ)

8 Prompt("X= "..x1.." Y= "..y1.." Z= "..z1)

9 end
```

_____

OK line by line

1) Just a remark so I know what the script is about (not required)

2) Establishes the “init” function (discussed earlier)

3) The end command to close the init function

4) Establishes the main function which is run each time the script is looped by the engine.

5) Sets the value of the players X location by referencing a global variable (established by the global.lua script discussed earlier) (g_PlayerPosX). In this case the value is held by the variable with the name “x1“. The “math.floor” statement is a call to the LUA library of built-in commands. This command rounds-down the values of g_PlayerPosX. IE if the value of the player’s x location is 2367.8904, the math.floor command rounds-down the number to 2367. Integer numbers are easier to read on the screen than double precision numbers.

6 & 7) these are the same as line 5, used to establish the values for Y and Z (y1 and z1). These three number represent the exact location of the player within the 3D environment, which in GameGuru is some 51000 x 51000 units (x & z). Y is the up/down location.

8) This line displays the Prompt() in the monitor. In this case the display would look something like this, running across the lower center of the screen.

X= 21563 Y=125 Z=15674

These values can be used to take measurements within the environment, or can be used in other scripts to define a specific location.

9) This is the “end” command for the _main function. and the script ends.

TID-BIT: the Prompt display of the values lasts for only a short time, and this display is refreshed each time the script is looped. Thus, if I move around, the values change to represent my new position. There are other Prompt related commands such as PromptDuration(). This command displays the prompt, but for a specific amount of time.

The GameGuru File system.

Like many software programs GameGuru has a file system which is specific to the engine. These files are organized and named in such a way, and placed in a specific location, so that the engine can find what it needs. This file system can be found in the program folder where you installed GameGuru. It is a folder called “Files”.

Inside the Files folder are a group of “Bank” folders that hold different types of files. The scriptbank holds the LUA scripts. The mapbank holds the map (levels) files. The entitybank holds the models and textures which are available to be loaded into your 3D environment.

In order to make a entity ready to load, a specific entity requires a minimum 4 files in order to be successfully loaded and displayed. The example below is the from the very first object that if provided with GameGuru, the Barrel (Acid).

1) The dot X (.x) file which contains the actual mesh of the model. E.g. barrel.x

2) A texture file using the “_D.dds” format. E,g, barrell_acid_D.dds

(please note the _D.dds format as this is important to GameGuru)

3) A .BMP file, this is the in editor image. E.g. Barrel (Acid).bmp

4) The FPE file E.g.Barrel (Acid).fpe. This MUST have the same name as the BMP.

The FPE File explained

Every entity that will end-up in your 3D environment requires an FPE file. If you look into the entitybank folder to a specific entity you will find an FPE file for each entity. Like a LUA script, the FPE is a simple text file saved using the .fpe form.

The FPE file is a group of initial instructions that tells the GameGuru engine how to set-up your entity in the 3D environment. Here is a typical FPE file.

```
;header
desc = Barrel (Acid)

;visualinfo
textured = barrell_acid_D.dds
effect = effectbank\reloaded\entity_basic.fx
castshadow = 0

;orientation
model = barrel.x
offx = 0
offy = 0
offz = 0
rotx = 0
roty = 0
rotz = 0
defaultstatic = 0
materialindex = 2

;statistics
strength = 25
explodable = 0
debrisshape = 1

;ai
aiinit = appear.lua
aimain = default.lua
aidestroy = disappear.lua

;spawn
spawnmax = 0
spawndelay = 0
spawnqty = 0
```

There are many additional command settings that can be used in an FPE file. For now note the following...

1) The use of the desc line should also be the name of the FPE file. (the FPE file is not subject to the lowercase bias of LUA scripts as far as naming is concerned.). This is also the name that will be given to the entity in the editor.

2) The name of the Texture file using the _D.dds form. Note that the Texture file, the FPE file and the .X file all need to be in the

same folder inside the entitybank folder, in order to load the entity into the environment.

3) Under the “ai” section, note the aimain script =. This is the LUA script which can be attached to the entity in the FPE file. Also note that a specific script can also be attached to the entity using the Properties Editor in GameGuru. (to be discussed shortly)

4) Note the “effect” line... entity_basic.fx. This is a shader file and defines an effect that will be a and how GameGuru will work with that entity. There are two of these scripts; entity_basic.fx and character_basic.fx.

5) Note the “defaultstatic=:”. Remember before when we discussed Static and Non-Static entities. This is where 1= static mode, and 0=non-static mode. This too can be changed in the Property Editor.

OK, Let’s Return To The Prompt LUA Script

_____

```
1 -- Script to display current location of player

2 function aaplayerxyzprompt_init(e)

3 end

4 function aaplayerxyzprompt_main(e)

5 x1=math.floor (g_PlayerPosX)

6 y1=math.floor (g_PlayerPosY)

7 z1=math.floor (g_PlayerPosZ)

8 Prompt("X= "..x1.." Y= "..y1.." Z= "..z1)

9 end
```

_____

OK line by line

1) Just a remark so I know what the script is about (not required)

2) Establishes the “init” function (discussed earlier)

3) The end command to close the init function

4) Establishes the main function which is run each time the script is looped by the engine.

5) Sets the value of the players X location by referencing a global variable (established by the global.lua script discussed earlier) (g_PlayerPosX). In this case the value is held by the variable with the name “x1“. The “math.floor” statement is a call to the LUA library of built-in commands. This command rounds-down the values of g_PlayerPosX. IE if the value of the player’s x location is 2367.8904, the math.floor command rounds-down the number to 2367. Integer numbers are easier to read on the screen than double precision numbers.

6 & 7) these are the same as line 5, used to establish the values for Y and Z (y1 and z1). These three number represent the exact location of the player within the 3D environment, which in GameGuru is some 51000 x 51000 units (x & z). Y is the up/down location.

8) This line displays the Prompt() in the monitor. In this case the display would look something like this, running across the lower center of the screen.

X= 21563 Y=125 Z=15674

These values can be used to take measurements within the environment, or can be used in other scripts to define a specific location.

9) This is the “end” command for the _main function. and the script ends.

TID-BIT: the Prompt display of the values lasts for only a short time, and this display is refreshed each time the script is looped. Thus, if I move around, the values change to represent my new position. There are other Prompt related commands such as PromptDuration(). This command displays the prompt, but for a specific amount of time.

Ok, I have my three files (.x, FPE, and _D.dds), and I have my Lua Script. I could attach my script to the entity through the FPE file by changing the file name in the aimain line to aaplayerxyzprompt.lua. I could also change the script using the Properties Editor in GameGuru.

Here is where the question of static vs non-static entities becomes important. If I attach this script to a “static” entity nothing will happen during game play. Thus, I must make sure that the entity is non-static. Here again I can change the “defaultstatic” value in the FPE from 1 to zero. Zero means that the entity is treated as non-static by GameGuru, and thus when the game is run, the prompt display works as intended.

Script activation limits

TID-BIT: So you have your script running and there is your display. You continue to move around in your 3D environment and as expected the values change as you move. Then suddenly, the prompt disappears. Why?

Built into GameGuru is a system that limits scripts from being action if the player is more than 3000 units away from the entity. This ensures that scripts that don't need to be run are suspended, which in turn aids performance.

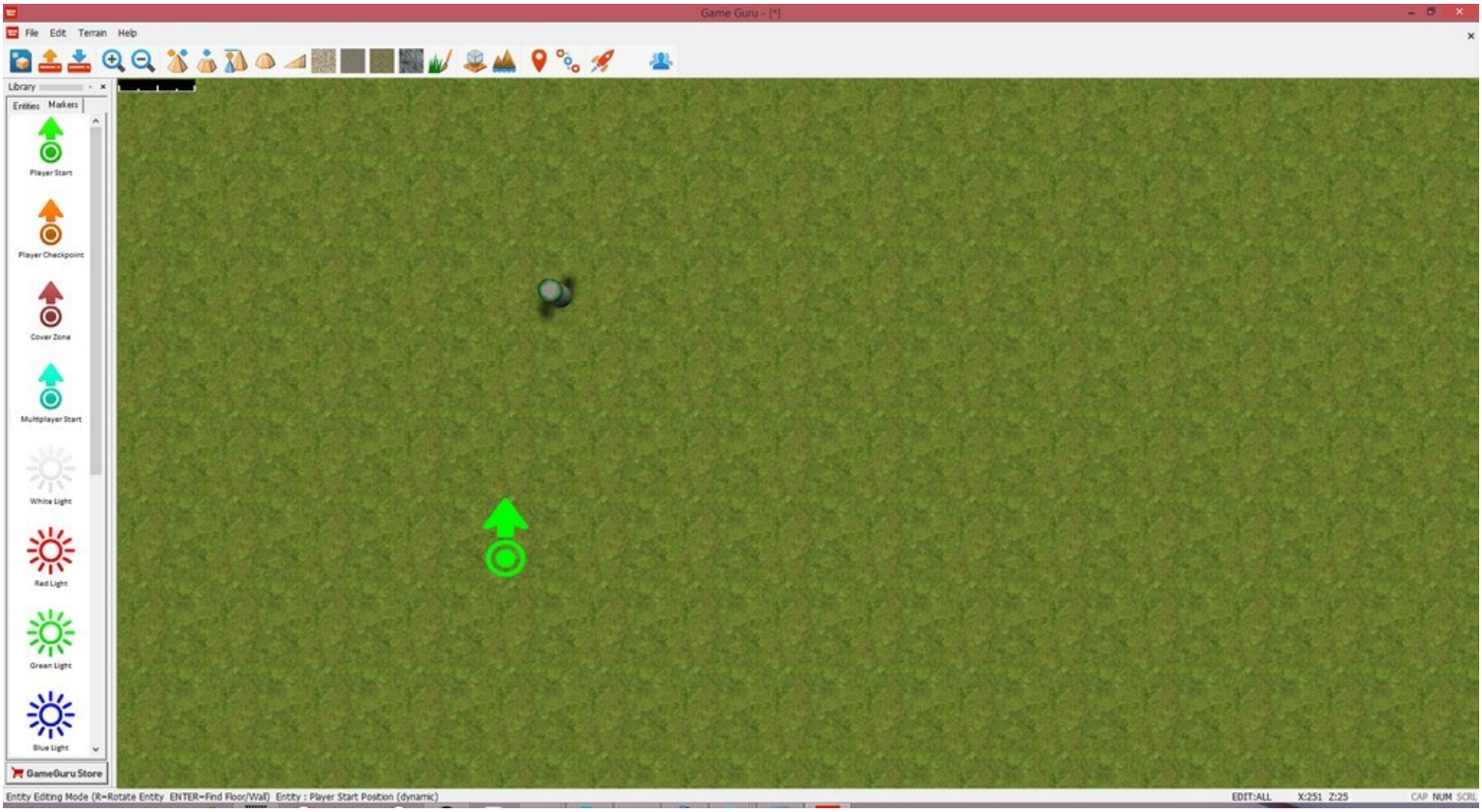
So, in the example above, if your display script exceeds 3000 units in any direction, the prompt display is lost.

However, there is another setting that can override this limitation by keeping the script Always On, regardless of distance. We’ll discuss this setting in a moment when we look into the mysterious Properties Editor inside the GameGuru Editor.

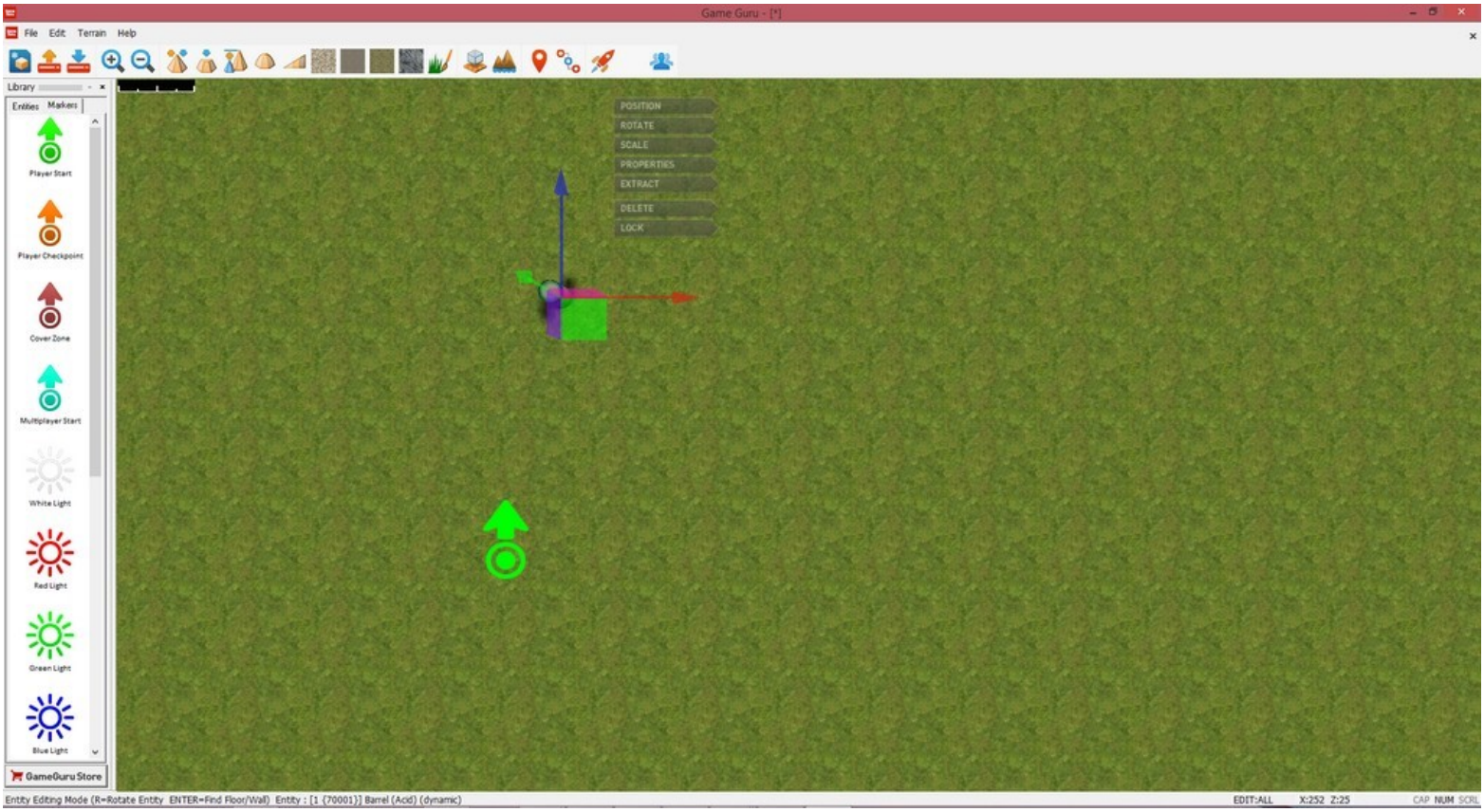
Adding a script to an entity

The Properties Editor.

Here is the typical editor for GameGuru. This is where you add entities and shape your 3D environment. I’ve placed a car on the terrain, as well as a Start Marker (green) which establishes the location where the game will start when run. You can find more information on this in our other tutorials.

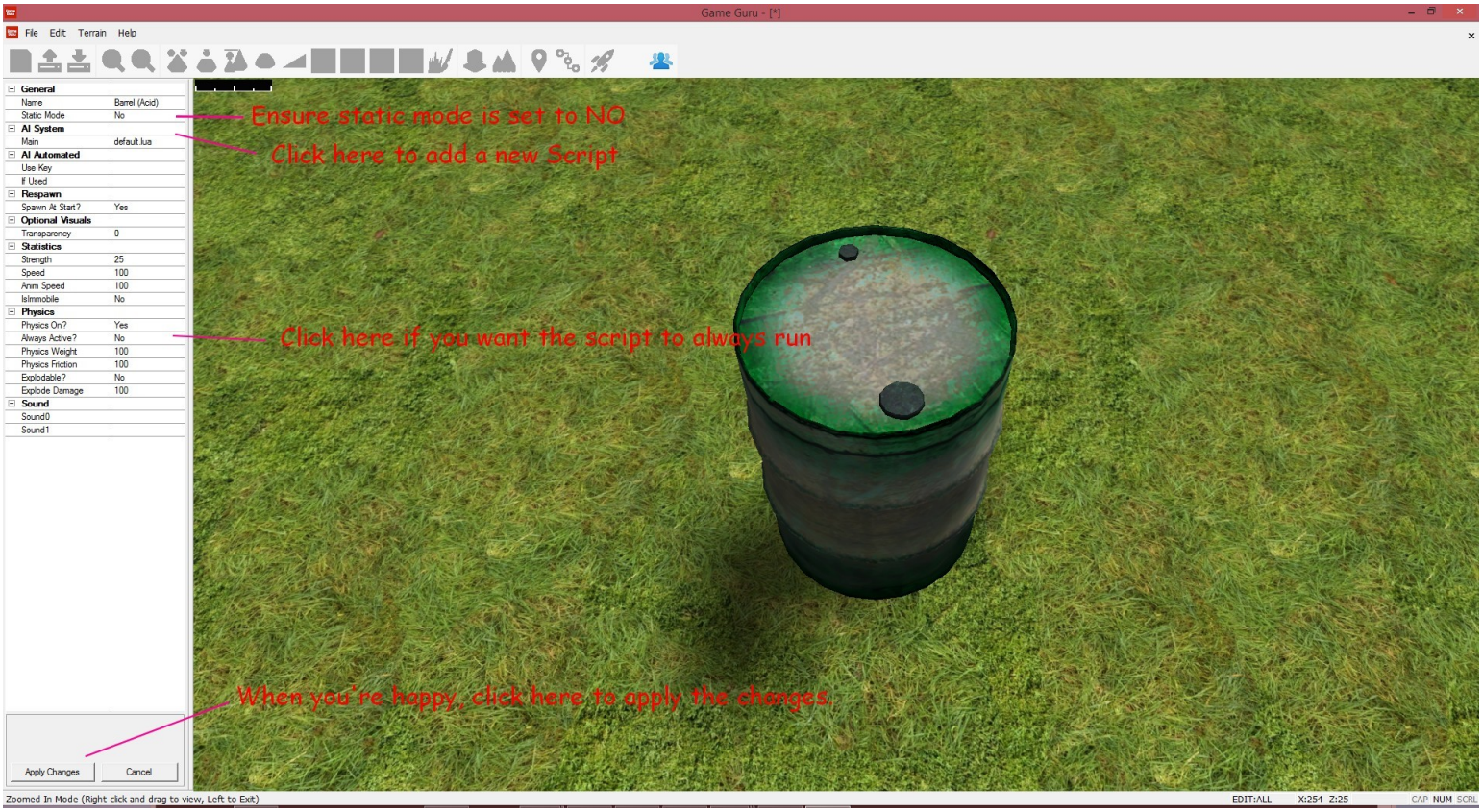


If you left click on the entity, a set of editing choices are displayed (The Widget). One of the choices is “PROPERTIES”



When you select properties the screen display changes (see screen shot below). To the left are the properties for this entity. Make sure the static mode is set to NO, that the proper LUA script is selected, and should you want the script always running,

that the Always Active property is set to YES.



Finally, remember to click Apply Changes, so that your changes are set with the entity.

Creating your first script.

For all you budding scripters we'll now look at all of the LUA commands unique to GameGuru

You'll find a default script, the one to pick up health in this case to give scripters an idea of layout. The script ships with GameGuru and can be found in your Scriptbank folder.

In the code below, four important things to note are:-

1) A script must be saved under the same name as it is referenced by in the script function. In the example below that would be health.lua

2) A script may contain remarks (words or sentences that are not scripting commands) and these must use the prefix --

3) All scripts MUST start with

function NAME_init(e)
end

In the example below is

function health_init(e)
end

4) All main script functions must start and end with

function NAME_main(e)

```
end
```

In the example below, this is

```
function health_main(e)
some code
end

-- LUA Script - precede every function and global member with lowercase name of

script + '_main'

-- Player Collects Health

function health_init(e)

end

function health_main(e)

PlayerDX = g_Entity[e]['x'] - g_PlayerPosX

PlayerDY = g_Entity[e]['y'] - g_PlayerPosY

PlayerDZ = g_Entity[e]['z'] - g_PlayerPosZ

PlayerDist=math.sqrt(math.abs(PlayerDX*PlayerDX)+math.abs(PlayerDY*PlayerDY)+math.abs

(PlayerDZ*PlayerDZ))

if PlayerDist < 80 then

PlaySound(e,0)

AddPlayerHealth(e)

Destroy(e)

end

end
```

Another very useful addition to the LUA scripting is the new suffix _exit.

“The _exit function in an optional routine that can be added to any script. When an entity is destroyed in game, and goes through the process of being removed from the game, the _exit function for that entity will be called, if it exists. This presents an opportunity to have a custom way for your character, object etc. to go out with a bit of style. Perhaps you would like to play a custom death sound, damage the player if he is nearby etc.”

Finally, here is a list of all the current commands, we'll keep this up to date as the scripting expands.

LUA commands - GameGuru Global variables

Global variable within GameGuru are a fast way of returning information about entities or input when in game. These can be very powerful and allow access to a lot of internal figures that can turn a great script into an amazing one.

Player Globals

g_PlayerPosX - Returns the players X position in the 3d world
g_PlayerPosY - Returns the players Y position in the 3d world
g_PlayerPosZ - Returns the players Z position in the 3d world
g_PlayerAngX - Returns the players X angle/rotation
g_PlayerAngY - Returns the players Y angle/rotation
g_PlayerAngZ - Returns the players Z angle/rotation
g_PlayerObjNo - Returns the ObjNo assigned to the player, this is mainly for internal use.
g_PlayerHealth - Returns the players current health
g_PlayerLives - Returns the players current number of lives
g_PlayerFlashlight - Returns true if the player flashlight is active
g_PlayerGunCount - Returns the number of guns currently carried by the player
g_PlayerGunID - Returns the ID of the current player weapons use
g_PlayerThirdPerson - Returns true if the player is in 3rd person view

Timers

g_Time - Returns the amount of time actually played since the test or standalone game launched, not including time the game is paused.
g_TimeElapsed - Returns the time passed in the current frame

Input

g_KeyPressE - Returns true if the E key has been pressed
g_Scancode - Returns the current scancode
g_InKey = Returns true if X key is being pressed, e.g. g_InKey(S) would return true if S was pressed

Entities

g_Entity - Returns the entity number

Multiplayer and Co-op

mp_isServer - Returns true if the machine is hosting
mp_playerNames - Returns names of MP players
mp_playerKills - Returns the players kills
mp_playerDeaths - Returns the times the player has been killed
mp_playerConnected - Returns true if the player is connected
mp_playerTeam - Returns the players team
mp_isConnectedToSteam - Returns true if the player is connected to Steam
mp_gameMode = 0
mp_servertimer - Returns the current server time
mp_showscores - Shows the current scores
mp_teambased - Returns true if the game is team based
mp_friendlyfireoff - 1 =Friendly fire allowed, 0 = Friendly fire disallowed
mp_me = 0;
mp_coop - Returns true if the game is co-op
mp_enemiesLeftToKill - Returns the number of characters left to kill

LUA commands - TEXT

Prompt(str)

Displays text (str) at the base and centre of the screen

PromptDuration(str,v)

Displays text (str) for V millisecs at the base and centre of the screen.

PromptTextSize(v)

Sets the prompt size (V) – this has a value of between 1 and 5 with 1 being the smallest.

PromptLocal(e,str)

Displays text (str) at the current entities (e) location. (e) references the current entitiy, no number is required. E.g. PromptLocal(e,"hello")

Text(x,y,size,txt)

Displays text (txt) using (size) at the relative screen position X,Y which have a value between 1-100. This is percentage based with 100 being the full width or height of the screen.

TextCenterOnX(x,y,size,txt)

Displays text (txt) using (size) centred at the relative screen position X,Y which have a value between 1-100. This is percentage based with 100 being the full width or height of the screen.

LUA Commands - VISUALS

SetFogNearest(v)

Sets the fog near distance to (V)

SetFogDistance(v)

Sets the fog near furthers far distance to (V)

SetFogRed(v)

Sets the red colour value of the fog. This has a range of 0-255

SetFogGreen(v)

Sets the green colour value of the fog. This has a range of 0-255

SetFogBlue(v)

Sets the blue colour value of the fog. This has a range of 0-255

SetFogIntensity(v)

Sets the thickness value of the fog. This has a range of 0-100

SetAmbienceIntensity(v)

Sets the Ambience Intensity to V. This has a range of 0-100

SetAmbienceRed(v)

Sets the red colour value of the overall ambient light. This has a range of 0-255

SetAmbienceGreen(v)

Sets the green colour value of the overall ambient light. This has a range of 0-255

SetAmbienceBlue(v)

Sets the blue colour value of the overall ambient light. This has a range of 0-255

SetSurfaceRed(v)

Sets the red colour value of light applied to the surface of entities. This has a range of 0-255

SetSurfaceGreen(v)

Sets the green colour value of light applied to the surface of entities. This has a range of 0-255

SetSurfaceBlue(v)

Sets the blue colour value of light applied to the surface of entities. This has a range of 0-255

SetPostVignetteRadius(v)

Adjusts the radius of the Vignette effect

SetPostVignetteIntensity(v)

Sets the intensity of the vignette effect

SetPostMotionDistance(v)

Set the distance affected when motion blur is applied.

SetPostMotionIntensity(v)

Set the intensity of motion blur

SetPostDepthOfFieldDistance(v)

Sets the distance Depth of Field is applied

SetPostDepthOfFieldIntensity(v)

Sets the Depth of Field intensity

LUA Commands - LEVEL CONTROL

JumpToLevelIfUsed(e)

Ends the level and loads the level defined in the entities IfUsed field in the propeties menu.

JumpToLevel(levelname)

Ends the level and loads the level defined by (levelname)

FinishLevel()

End the level, showing the game complete screen

LUA Commands - GLOBAL

HideTerrain()

Hides the terain, ideal for indoor scenes that don't use it.

ShowTerrain()

Displays previously hidden terrain.

HideWater()

Hides the water plane.

ShowWater()

Displays previously hidden water plane.

HideHuds()

Hides all onscreen huds,

ShowHuds()

Displays previously hidden huds.

Control the occluder from script! Set the occlude from 0 (off) to 100 (max). 10 and under for minimal popping

LUA Commands - PLAYER CONTROL

FreezePlayer()

Freeze all player movement and mouse looking

UnFreezePlayer()

Unfreeze the player and allow movement and mouse looking.

SetPlayerHealth(v)

Set the player health to (v)

AddPlayerWeapon(e)

If entity is a weapon assign it to the next available weapon slot.

AddPlayerAmmo(e)

If entity is ammo assign it to the appropriate ammo pool.

AddPlayerHealth(e,v)

Add (v) points of health to the player as defined by the quantity value in the entity FPE.

SetPlayerLives(e,v)

Set the players lives to (v).

RemovePlayerWeapons(e)

Remove all player weapons.

AddPlayerJetPack(e,fuel)

Adds the jetpack ability to the player and sets (fuel) amount.

HurtPlayer(e,v)

Cause V damage to player

SetFlashLightKeyEnabled(V)

V=1 for on allowing the flashlight to be used

V= 0 for off preventing the flashlight from being used.

SetFlashLight(V)

V=1 switches the flashlight on,

V=0 switches the flashlight off

SetPlayerWeapons(0)

Disables player weapons

SetPlayerWeapons(1)

Restores player weapons

Subtract (v) health from the player.

TransportToIfUsed(e)

Moves the player to the location of the entity named in the ifused field.

LUA Commands - PARTICLES

StartParticleEmitter(e)

Emit particles from the entity – This is currently a basic emitter.

StopParticleEmitter(e)

Stop the entity emitting particles.

LUA Commands - ENTITY CONTROL - Part 1

SetEntityHealth(e,v)

Set entity health to (v)

StartTimer(e)

Starts an entity dependant timer.

GetTimer(e)

Returns the current entity time.

Destroy(e)

Destroys the current entity, removing it from the current game.

CollisionOn(e)

Turn an entity collision on (for example for a closed door).

CollisionOff(e)

Turn an entity collision off (for example for an open door).

GravityOff(e)

Turn off entity Gravity.

GravityOn(e)

Turn on entity Gravity.

LookAtPlayer(e)

Forces the entity to face the player.

RotateToPlayer(e)

Rotates the entity to face the player.

RotateToPlayerSlowly(e,v)

Rotates the entity to face the player at (v) speed

Hide(e)

Makes the entity invisible

Show(e)

Makes the entity visible

Spawn(e)

Forces the entity to spawn if not set to spawn at start, except for characters in multiplayer mode

SetActivated(e,v)

Sets the entity as active (v=1) or inactive. (v=0)

ActivateIfUsed(e)

Activates the enttiy named in the IfUsed properties field.

Collected(e)

Flag the entity as collected (e.g a key)

MoveUp(e,v)

Move the entity up and (v) speed

MoveForward(e,v)

Move the entity forward and (v) speed

MoveBackward(e,v)

Move the entity backward and (v) speed

SetPosition(e,x,y,z)

Move the entity to map postion x,y,z

ResetPosition(e,x,y,z)

Reposition a physics entity

SetRotation(e,x,y,z)

Rotates the entity to angle x,y,z

LUA Commands - ENTITY CONTROL - Part 2

ModulateSpeed(e,v)

Modualate the speed of all entity actions by (v)

RotateX(e,v)

Rotate the X angle of the entity by (v)

RotateY(e,v)

Rotate the Y angle of the entity by (v)

RotateZ(e,v)

Rotate the Z angle of the entity by (v)

Scale(e,v)

Scales an entity to (v) percentage

SetAnimation(e)

Set an animation index value for later use

SetAnimationFrames(a,b)

Set the entity animation frame to the range (a) to (b)

PlayAnimation(e)

Play the current animation range once.

LoopAnimation(e)

Loop the current animation range.

StopAnimation(e)

Stop all animations for the current entity

SetAnimationSpeed(e,v)

Set the animation speed of the entity

SetAnimationFrame(e,v)

Set the entities animation frame to (v)

GetAnimationFrame(e)

Get the animation frame number from the entity

SpawnIfUsed(e)

Spawns the entity named in the IfUsed field of the current entity

LUA Commands - SCRIPTS

Include ("xxx")

Use within the init function to ensure the script called (xxx) is pre loaded

SwitchScript(e,str)

Switch the entities script to script named (str)

LUA Commands - CHARACTER and AI CONTROL

CharacterControlUnarmed(e)

Switch character to unarmed state

CharacterControlLimbo(e)

Switch character to limbo state

CharacterControlArmed(e)

Switch character to armed state

CharacterControlFidget(e)

switch character to fidget state

CharacterControlDucked(e)

switch character to crouched state

CharacterControlStand(e)

switch character to stood state

SetCharacterToWalk(e)

set the character to walk when moving

SetCharacterToRun(e)

set the character to run when moving

SetCharacterToStrafeLeft(e)

Set the character to strafe left

SetCharacterToStrafeRight(e)

Set the character to strafe right

SetCharacterVisionDelay(e,v)

Set the speed at which the entity will react after seeing the player

SetAttachmentVisible(e,1).

1 sets the entities attachment to be visible (such as their weapon), 0 switches it off

LockCharacterPosition(e,v)

Lock a character in place. Parameter reserved for future use.

UnlockCharacterPosition(e,v)

UmLock a character. Parameter reserved for future use.

FreezeAI()

Freeze all AI

UnFreezeAI()

Unfreeze all AI

FireWeapon(e)

If the character has a weapon, fire it. If not, trigger melee attack.

LUA Commands - ZONES

Checkpoint(e)

Trigger player to record checkpoint for later use

GetPlayerInZone(e)

Return 1 if the player is inside entity zone area

WinZone(e)

Trigger player to win this level and move to level specified in IfUsed property.

LUA Commands - SOUND and MUSIC

PlaySound(e,v)

Play the sound stored in the specified slot (v) as defined in entity properties. (v) can be 0 or 1.

PlaySoundIfSilent(e,v) - Plays a sound if NOT currently playing

Play the sound stored in the specified slot (v) as defined in entity properties if not playing which can be 0 or 1.

PlayNon3DSound(e,v)

Play the sound stored in the specified slot (v) as defined in entity properties ignoring location and distance which can be 0 or 1.

LoopSound(e,v)

Play and loop the sound stored in the specified slot as defined in entity properties which can be 0 or 1.

StopSound(e,v)

Stop the sound that is playing which can be 0 or 1.

SetSound(e,v) to be used before functions such as SetSoundVolume(vol) which do not specify a sound to use

SetSoundSpeed(freq)

Set currently set sound frequency

SetSoundVolume(vol)

Set the currently set sound volume (1 – 100)

SetCharacterSound(e,str)

Set the character sound to (str) as defined in audiobank folder

PlayCharacterSound(e,str)

Play the character sound (str) as defined in audiobank folder

PlayCombatMusic(playTime,fadeTime)

Plays the combat music as defined in the audiobank

PlayFinalAssaultMusic(fadeTime)

Plays the assault music as defined in the audiobank

DisableMusicReset(v)

Call at start of game to suspend normal music restart behaviour by setting (v) to 1.

LUA Commands - LIGHTS

HideLight(e)

Hide a light if entity is a dynamic light. Has no effect on static lights.

ShowLight(e)

Show a light if entity is a dynamic light. Has no effect on static lights.

LUA Commands - IMAGES

LoadImages(str,v)

Specify an image inside 'scriptbank\images\"' to load images at slot (v) onwards. Images must be named 000,001,002, etc.

SetImagePosition(x,y)

Set the X and Y position of where the image should be drawn

ShowImage(i)

Select the image slot number to be drawn

HideImage(i)

Hide the image from being drawn

SetImageAlignment(i)

Use 0 to position image using center hotspot, 1 is top left hotspot

Panel(x,y,x2,y2)

Draw a panel from X,Y to X2, Y2

LUA Commands - SPRITES

Sprites are images that can be positioned, rotated, resized and manipulated in many ways. This powerful set of commands allow many advanced features to be coded. Sprites share image files with the image commands.

LoadImage: myImage = LoadImage ("myFolder\\myImage.png") -- Anywhere inside GG Files folder (however it is best to put into scriptbank images folder for making standalones)

CreateSprite: mySprite = CreateSprite (myImage)

DeleteSprite: DeleteSprite (mySprite) - Remove the sprite from memory

***SetSpritePosition**: SetSpritePosition (mySprite , x , y) - Sprite positions are percentage based, so SetSpritePosition(mysprite,50,50) would position the sprite in the centre of the screen.

SetSpriteSize: SetSpriteSize (mySprite , sizeX , sizeZ) -- passing -1 as one of the params ensure the sprite retains its aspect ratio

SetSpriteDepth: SetSpriteDepth (mySprite , 10) (0 is front, 100 is back) - Set the order in which the sprite will be drawn.

SetSpriteColor: SetSpriteColor (mySprite , red , green, blue, alpha) - Sets the sprite RGB values and alpha, each value has a range of 0-255.

SetSpriteAngle: SetSpriteAngle (mySprite , 180) - Sets the sprite rotation in degrees.

SetSpriteOffset: SetSpriteOffset (mySprite , 32 , 32) (would be the centre for a 64x64 image assigned to a sprite)

SetSpriteImage: SetSpriteImage (mySprite , myImage) - Assigns a new image to the previously created sprite.

LUA Commands - MULTI-PLAYER

MP_IsServer()

When developing a multiplayer script, you will need to have code for both the client and the server. Some may be shared (such as a function for displaying the score) but some will need to be server or client only. This command enables you to do just that. a return value of 1 indicates this script is running on the server, 0 means this is a client machine.

SetMultiplayerGameMode(mode)

A game cycle will typically be made up of more than one mode. The simplest example would be:

1. Starting the game, setting scores to 0, then heading to mode 2.

2. The main game itself, runs until somebody has deemed to have won, then switches to 3

3. Displays the ending scores, stops play and counts down to the next game. heads back to 1

The server is in charge of settings these modes and does so via this command to inform all clients of the mode. It is wise for the server to keep a variable with the mode locally so it doesn't loose track of the mode itself.

GetMultiplayerGameMode(mode)

This command enables clients to check which mode the game is in currently. This will have been set by the server previously via SetMultiplayerGameMode(mode)

SetServerTimer(t)

Enables the server to set a timer (perhaps to trigger the end of the game) That can be picked up by clients via GetServerTimer().

GetServerTimer(t)

Returns the time the timer was set by the server using SetServerTimer()

GetServerTimerPassed()

Returns how much time has passed since the server first set the timer up.

ServerRespawnAll()

This command is the companion to ServerEndPlay() and will allow for play to carry on, by first triggering everyone to spawn in. This will be used to kick off a new round after the completion of the previous round.

ServerEndPlay()

When a game has been won, this command can be used to stop any further gameplay taking place. The camera will change to a top down view of the current level and presents the perfect time to show scores and perhaps set a countdown to the next round. If anyone is dead they will spawn back in still, but then switch to the top down view along with everyone else.

GetShowScores()

If a player presses tab in game to show the scores, GetShowScores() will return a 1. After a short delay of a player pressing tab it is set back to 0. Use this command when game play is in full swing to show the current scoring of all playings.

```
-- Showing the current scoreboard
if GetShowScores() == 1 then
multiplayer_firstto10_results()
end
```

SetServerKillsToWin(v)

This command allows the LUA script to inform the game engine how many kills are needed to win (if any). This is needed so the game engine knows when to stop updating kills in a match. For example when someone has won a game, any kills after that need to be not counted. This command enables that functionality.

Typically you would set the kills needed to win inside the lua script so the server knows how many kills will be needed for victory:

```
GAME_KILLS_TO_WIN = 10
```

Then the server can inform everyone else of the kills needed, so everyones version of the game knows when to stop counting kills:

```
SetServerKillsToWin(GAME_KILLS_TO_WIN)
```