Visualising the Conflict

Second Life Programming / Scripting

By Julio López (2011)

Contents

1. Introduction

Part of the work outlined in the AHRC proposal relating to “Visualizing the Conflict” was to create a Virtual Learning Environment in Second Life which would allow users to interact with the information on physical memorials and to engage with other users. This interaction should be based on a Google Map displaying the different memorials in its geographical context, where users could identify and select a specific memorial in order to see its 3D virtual replica and some additional information about it.

It was necessary then to integrate or simulate an interactive Google Map application in Second Life. Moreover, and in order to create a complete learning experience about a specific memorial, we were interested in having access, from the virtual world, to the information stored on a pre-existing external database compiled by CAIN. Additionally, secondary interactive tools were created to improve the user experience.

Two are the areas of the INCORE Island where the user can find interactive elements: the Reception Area and the Map Area, this last one being the central part of the island. Furthermore, small interactive pins can be found all around the island to provide instant teleport to the Reception Area.

This paper gives a brief description of how these interactive objects were scripted in Second Life and how this virtual environment communicates with the CAIN server.

2. Tools Used

Linden Scripting Language (LSL) is the language available in Second Life that gives behavior to the different in-world agents, from single primitives to avatars, and was used to script the interactions in the virtual world. One or more LSL scripts can be created inside an object, and different methods allow a script to communicate with other scripts, with other in-world agents and also with agents outside Second Life. Objects can also contain other objects which will be used during the interaction.

PHP and JavaScript where used in the external server to complete the communication CAIN – Second Life.

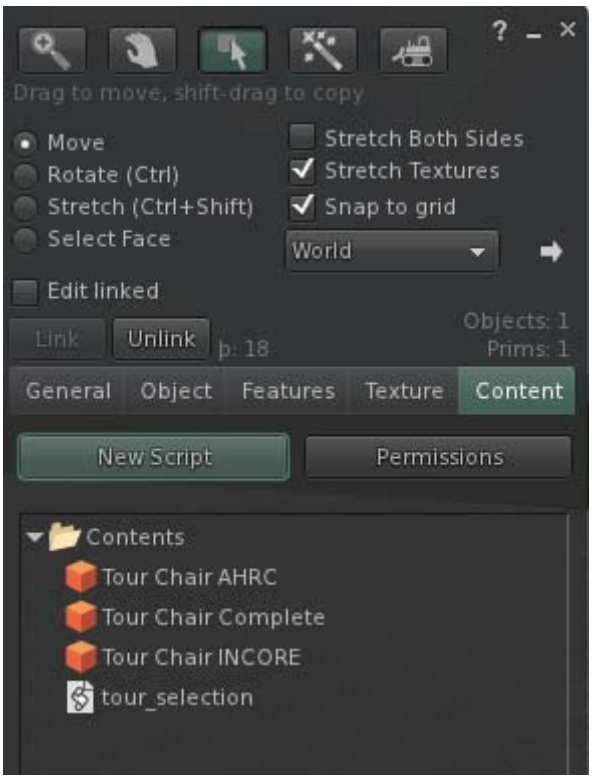

Figure 1: Objects and script in a prim's inventory

3. INCORE Island: Reception Area

This is the first area the user sees when he arrives to the INCORE Island, and two different interactive objects are available offering the user a first approach to the environment: Instant Teleport Pins and Guided Tour Chairs.

3.1. Teleport Pins

At the Reception Area, a mini-map provides the user with a complete overview of the whole island, with different pins pointing to the key zones.

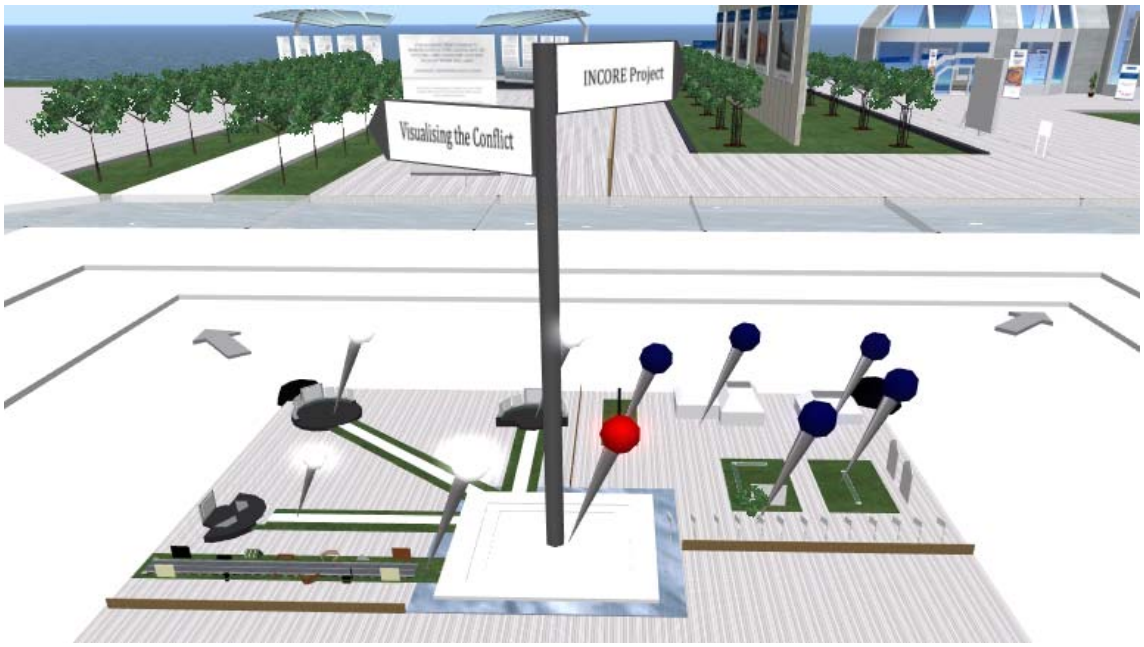

Figure 2: Mini-map at the Reception Area

When the camera is close enough, floating text becomes visible to tell the user what part of the Island each pin is pointing to. By clicking on a pin, the avatar is automatically teleported to the corresponding location of the INCORE Island.

Each teleport pin contains a script where floating text and target location are specified. To perform the teleport functionality, the LSL function *llSitTarget* was used. This function is mainly used to create chairs, benches, etc, and it sets the sit location for the prim, that is the exact location the avatar will have when sitting on that chair. In this case, the sit target will be the location we want to teleport to.

Immediately after the avatar “sits” on the target location, it is detected and forced to stand up with the function *llUnSit*, having in this way the effect of an instant teleport.

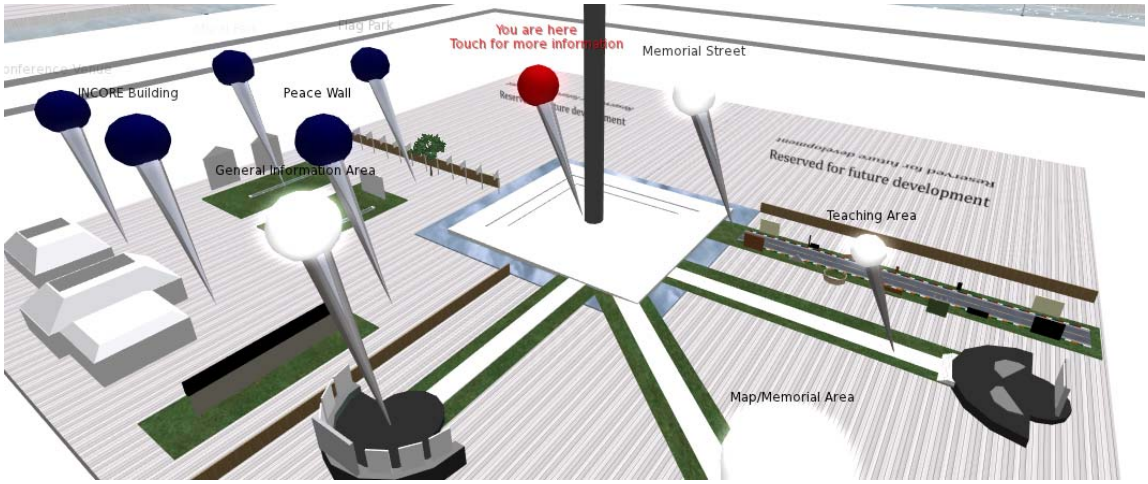

Figure 3: Floating text over the pins

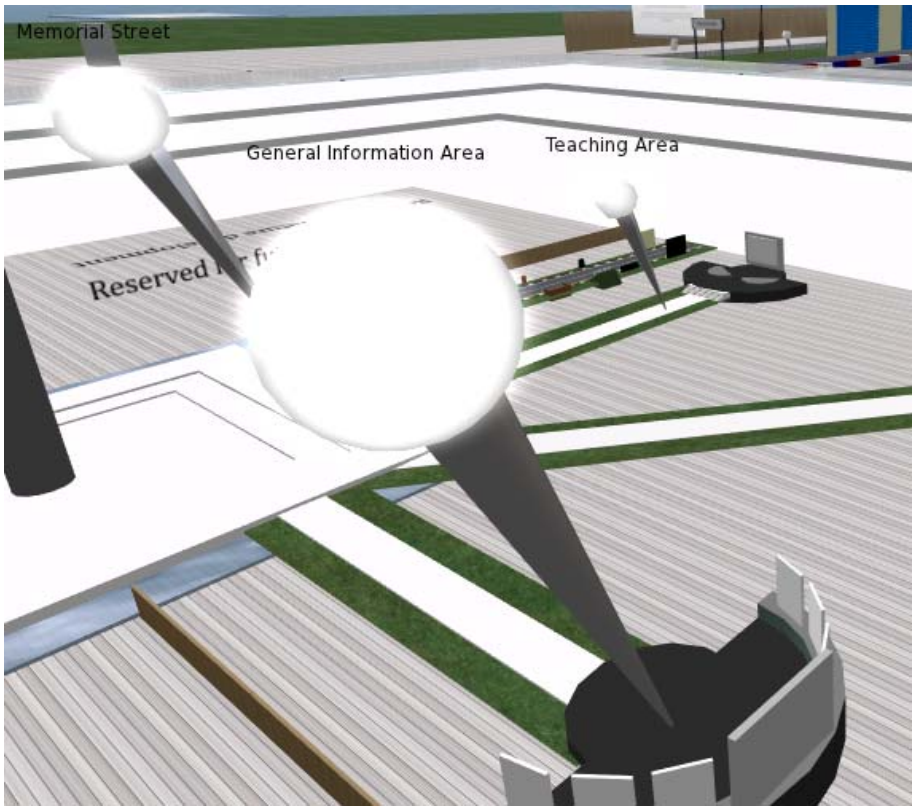

Figure 4: Floating text over the pins

```
vector tel_Location = <213, 209, 26>; //The x, y, z coordinates to teleport
string float_Text = "INCORE Building"; //label that floats above Teleport
```

Figure 5: Target location and floating text specified on the pin's script

```
vector target = (tel_Location- llGetPos()) * (ZERO_ROTATION / llGetRot());
llSitTarget(target, tel_Rotation);
llSetSitText("Teleport");
llSetText(float_Text, <1,1,1>, 1);
```

Figure 6: Code to set target location and floating text

```
changed(integer change)
{
    llSleep(0.15);

    llUnSit(llAvatarOnSitTarget());
}
```

Figure 7: After teleporting, the avatar
is forced to stand up

3.2. Tour chairs

Located at two corners of the mini-map, two black hemispheres provide the user with three different guided tour chairs. If one hemisphere is clicked, one dialog box appears giving the possibility of choosing one tour.

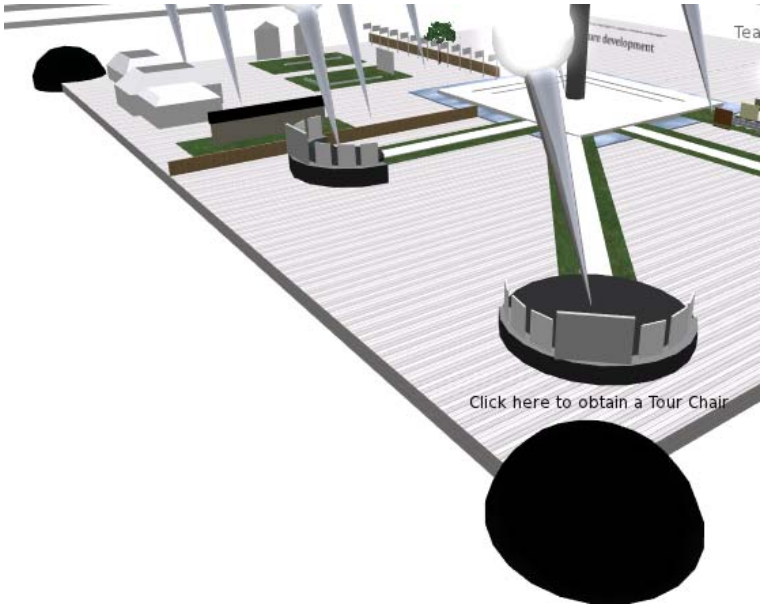

Figure 8: Guided Tours at the mini-map

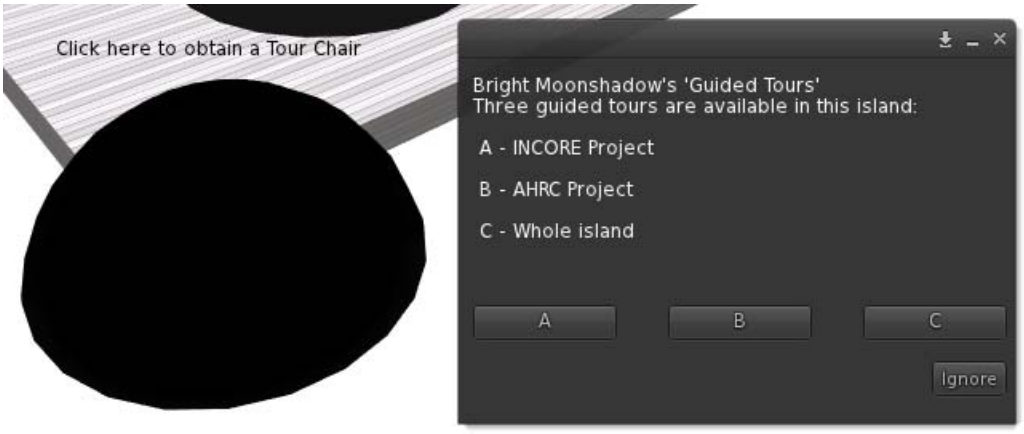

Figure 9: Dialog box to choose the Guided Tour Chair

When one option is clicked, the corresponding chair is automatically created besides the hemisphere. To do this, the three chairs need to be stored in the hemisphere's inventory with the script where the interaction is programmed (Figure 1). The function *llRezObject* is used then to create the selected chair, which is ready to start the guided tour.

```
touch_start(integer total_number)
{
    llDialog(llDetectedKey(0),"Three guided tours are available in this island: \n\n
                               A - INCORE Project\n\n
                               B - AHRC Project\n\n
                               C - Whole island",["A","B","C"],DIALOG_CHANNEL);
}

listen(integer channel, string name, key id, string message)
{
    string tour;
    if (channel == DIALOG_CHANNEL)
    {
        if (message == "A")
            tour = "Tour Chair INCORE";
        else if (message == "B")
            tour = "Tour Chair AHRC";
        else if (message == "C")
            tour = "Tour Chair Complete";
        llRezObject(tour,llGetPos() + <1,1,1.5>,<0,0,0>,rot,10);
    }
}
```

Figure 10: Tour Chairs Distributor's script

Once the chair has been created, all the user has to do is click on it. The avatar is automatically sat on the chair and the guided tour starts. Each chair has one script where coordinates, descriptions and user's rotations for the different stages of the tour are specified, and the functions *llMoveToTarget* and *llStopMoveToTarget* are used to move from one location to another.

```
list tourtarget = ["General Information Area","Teaching Area","Map Integration","Welcome Hub"];
list tourpos = [ < 220,153,25 >, < 152,227,29 >, < 211,204,26 >, < 135,136,21 >];
list tourrot = [ < 0,0,180 >, < 0,0,320 >, < 0,0,160 >, < 0,0,45 > ];
list targetrot = [ < 0,0,187 >, < 0,0,275 >, < 0,0,230 >, < 0,0,45 > ];
```

Figure 11: Locations and rotations specified in the chair's script

```
move_to(vector p,vector r)
{
    float d = 2;
    unit = (p-llGetPos())/(llVecDist(p,llGetPos())/60);
    llSetRot(llEuler2Rot(r*DEG_TO_RAD));
    do
    {
        if(llVecDist(p,llGetPos())>60)
        {
            llSetStatus(STATUS_PHYSICS | STATUS_PHANTOM, TRUE);
            llMoveToTarget(llGetPos()+unit,d);
            llSleep(d*1.3);
            llStopMoveToTarget();
            llSetStatus(STATUS_PHYSICS | STATUS_PHANTOM, FALSE);
        }
        else
        {
            llSetStatus(STATUS_PHYSICS | STATUS_PHANTOM, TRUE);
            llMoveToTarget(p,d);
            llSleep(d*1.3);
            llStopMoveToTarget();
            llSetStatus(STATUS_PHYSICS | STATUS_PHANTOM, FALSE);
        }
    }
    while (llVecDist(p,llGetPos())>2);
}
```

Figure 12: Function to move between locations

4. INCORE Island: Map Area

This is the main area of the island: the user will find here a Google Map where markers indicating the memorials modeled in Second Life are displayed. Shared Media, the new feature introduced with Viewer 2, is used to embed a real Google Map application which is hosted at CAIN and was built using the Google Map API. Two interactions (events) are supported by the markers on this Google Map: mouse-over event, i.e., put the mouse over a marker, and click event, i.e., click on a marker. Each event provides the user with a different level of information about the corresponding memorial. Five panels to the left of the Google Map display different information related to the memorial, and the empty area to the right of the map is where the 3D model is shown.

Figure 13: Map Area at the INCORE Island

The user has to interact with the Google Map in the same way he interacts with a Google Map application being displayed in a web browser. An information window appears on the map when the mouse is over the marker, allowing the user to know which the corresponding memorial is without further interactions. All this is managed by the Google Map application on the CAIN server, and the only communication with Second Life happens through Shared Media.

The communication between platforms takes place with the second interaction supported. When a marker on the map is clicked, the 3D virtual model becomes visible and the Information Panels change the content to display different material about the memorial. To make this possible, data from the CAIN server needs to be sent into Second Life and used to modify the virtual environment.

4.1. Communicating with SL

LSL provides functions that use HTTP for communicating with web servers on the outside internet, being possible for an in-world object to act as either a client or a server. The choice of the role depends on who initiates this communication.

Although the interaction in the Map Area is always started by the user on the Second Life side, this first interaction happens through the Google Map displayed using Shared Media, and it is interpreted by the server as done using a normal web browser and not from Second Life. As a response to that action, data will be sent from the CAIN server to the virtual world. This means that the data is sent in-world without detecting if it was requested from there, initiating then a new communication process for that purpose. In this scenario, the CAIN server starting the communication acts as the client, and an in-world object needs to be configured as an HTTP server.

Figure 14 below shows the high level communication between elements at the Map Area. The Data Processing Hub is at the core of this and acts as a link between the CAIN server and Second Life.

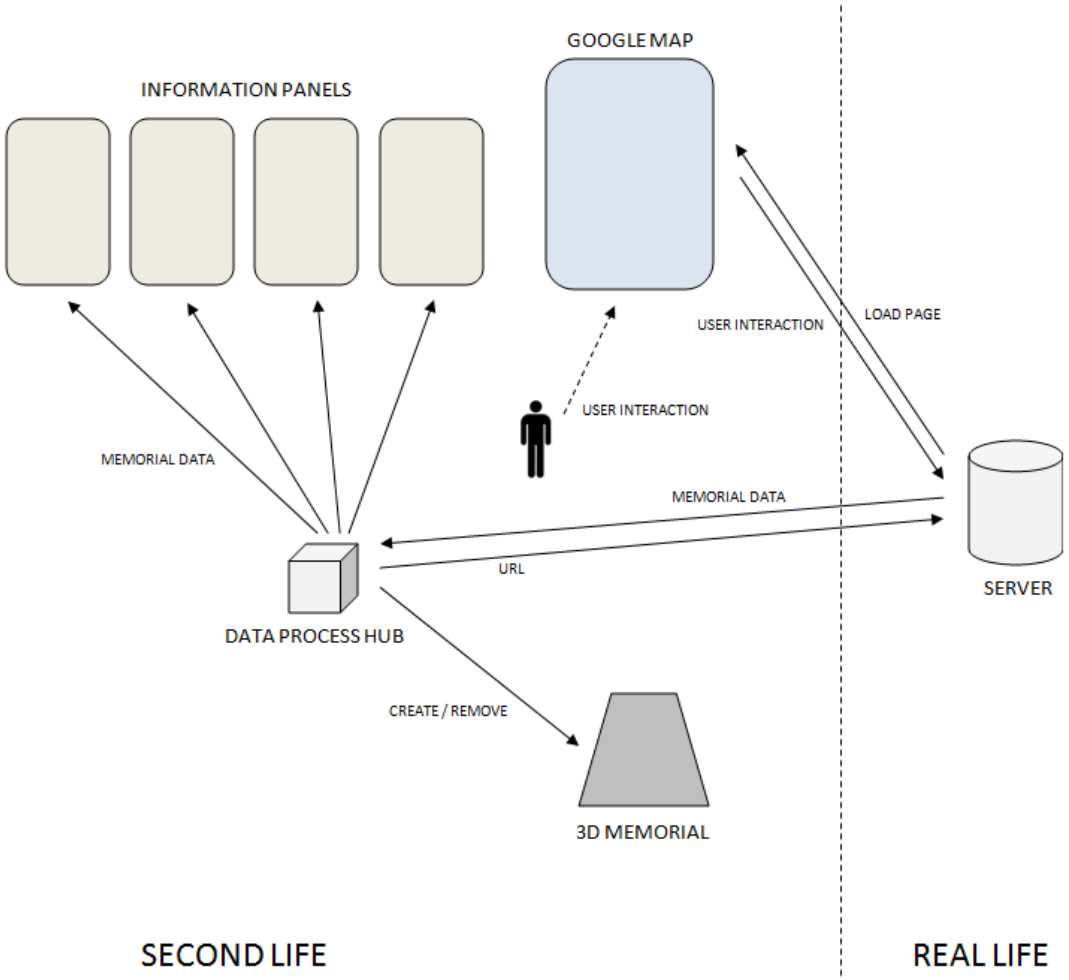

Figure 14: Communication

4.2. The Data Process Hub

This object, invisible to the user, is the central element of the Map Area, and manages the communication between Second Life and the external server. To work as an HTTP server, the Data Process Hub needs a url where data can be sent to. It is necessary then to obtain and send this url to the CAIN server, which is done with a first communication Second Life – CAIN server where the in-world object acts as a client.

When first saved, and after certain events that invalidates the url, the script in this object calls the function *llRequestURL* to request one HTTP:// url for use by this script, which is immediately sent to the CAIN server with the function *llHTTPRequest* using the POST request method. A PHP script on the CAIN server receives and saves that url, which will be used to send data about the memorials when required.

```
state_entry()
{
    llListen(RESET_URL_CHANNEL,"","","");
    inventoryObjects = getInventoryObjectsList();

    // A URL is requested
    llRequestURL();
}

// A new URL needs to be obtained after certain events
on_rez(integer n)
{
    llRequestURL();
}

changed(integer c)
{
    if (c & (CHANGED_REGION | CHANGED_REGION_START | CHANGED_TELEPORT) )
    {
        llRequestURL();
    }
}

listen(integer channel, string name, key id, string message)
{
    if ((channel == RESET_URL_CHANNEL)&&(message == "Reset"))
        llRequestURL();
}

http_request(key id, string method, string body)
{
    // When the URL is received, it is sent to the external server
    if ((method == URL_REQUEST_GRANTED))
    {
        llOwnerSay("URL: " + body);
        send_url(body);
    }
```

Figure 15: URL requested and sent to the external server

```
send_url(string url)
{
    string body = "URL=" + url;
    llHTTPRequest(PHP_LINKER, [HTTP_METHOD, "POST", HTTP_MIMETYPE, "application/x-www-form-urlencoded"], body);
}
```

Figure 16: Function to send the URL

```
$url = $_POST["URL"];

$myFile = "SecondLifeURL.txt";
$fh = fopen($myFile, 'w');

fwrite($fh, $url);
```

Figure 17: PHP code to receive and store the URL

Every time a marker on the map is clicked, an HTTP request is built by the Google Map application with data about the corresponding memorial. This request is sent to a PHP script, where a socket is created using the URL previously stored to send the data to the virtual world.

```
GEvent.addListener(marker, 'click', function(){
  realHideTooltip();
  if (isCoordRepeat(lat+lng) == false)
  {
          makeRequest(id);
  }
});
```

Figure 18: Click event for a marker on the Google Map

```
function makeRequest(id) {

        var marker = getMarkerById(id);
        marker.name = marker.name.replace(/&#39;/g,"'");
        var param = "id="+id;
        param += "&name="+marker.name;
        param += "&lat="+marker.slat;
        param += "&lng="+marker.slng;
        param += "&yaw="+marker.yaw;
        param += "&pitch="+marker.pitch;
        param += "&zoom="+marker.zoom;
        param += "&photo_id="+marker.photo_id;
        var request = makeHttpObject();
        request.open("POST","link.php",false);
        request.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
        request.setRequestHeader("Content-length", param.length);
        request.setRequestHeader("Connection", "close");
        request.send(param);
}
```

Figure 19: Request built by the Google Map application

```
$data = $id."/".$name."/".$lat."/".$lng."/";
$data .= $yaw."/".$pitch."/".$zoom."/".$photo_id;

$myFile = "SecondLifeURL.txt";
$fh = fopen($myFile, 'r');
$URL = fread($fh, filesize($myFile));
fclose($fh);

CallLSLScript($URL, $data);
```

Figure 20: PHP code to build and send the request to Second Life.
CallLSLScript opens a socket and writes on it

The data sent to that URL is received by the Data Process Hub as an HTTP request, where it is split into its different fields. Some fields are sent to the corresponding elements of the Map

Area, and one last field is used by the Data Process Hub to load the 3D virtual model of the memorial.

Once the environment has changed, the avatars on the Map Area are notified of the new memorial loaded. To do so, a sensor is launched and a Dialog Box with this notification is sent to the avatars detected.

```
http_request(key id, string method, string body)
{
    if ((method == URL_REQUEST_GRANTED))
    {
        llOwnerSay("URL: " + body);
        send_url(body);
    }
    else if (method == "POST")
    {
        if (body == "")
        {
            llShout(0,"An error has occurred. Try again later.");
        }
        else
        {
            // The message is read and split
            read_message(body,"/");

            // Each part is sent to the corresponding object
            llShout(PANELS_CHANNEL,monument_id);
            llShout(PHOTO_CHANNEL,monument_id+"/"+photo_id);
            llShout(STREETVIEW_CHANNEL,lat+"/"+lng+"/"+yaw+"/"+pitch+"/"+zoom);

            // The current memorial is removed, and the new one rezzed
            llSay(DELETE_CHANNEL, "DIE");
            llSleep(.5);

            if (hasBeenModelled(monument_id))
            {
                llSleep(.5);
                llRezObject(monument_id,llGetPos(),<0.0, 0.0, 0.0>,<0.0, 0.0, 0.0, 0.0>,10);
                llPlaySound("SHAWEE",1.0);
                MODELLED = TRUE;
            }
            else
            {
                MODELLED = FALSE;
            }

            // A sensor detects the avatars in the area to send a Dialog Box
            llSensor("",NULL_KEY,AGENT,34,PI);
        }
```

Figure 21: Data Process Hub's script. The HTTP request is received and properly managed

4.3. Memorials

A 3D virtual model of a certain memorial becomes visible when its corresponding marker on the Google Map is clicked. For a script to be able to create objects dynamically, it is necessary that those objects are stored in the prim's inventory where the script is, in this case in the Data Process Hub's inventory. These objects can be rezzed with the function *llRezObject*, using the object's name.

Each 3D memorial is stored using as the name the id that the corresponding physical memorial has on the CAIN database. This id is part of the data sent into Second Life by the CAIN server, and all the Hub's script has to do is identify that field on the message received and use it with the function *llRezObject* to create the correct 3D memorial, sending first a message to remove the previous memorial with *llSay*. To be sure that the selected memorial has a 3D replica and can be rezzed, a list of memorials modelled is checked every time a request is received; in case

the 3D model doesn't exist in the Hub's inventory, the user is notified while all the other infomative content is loaded.

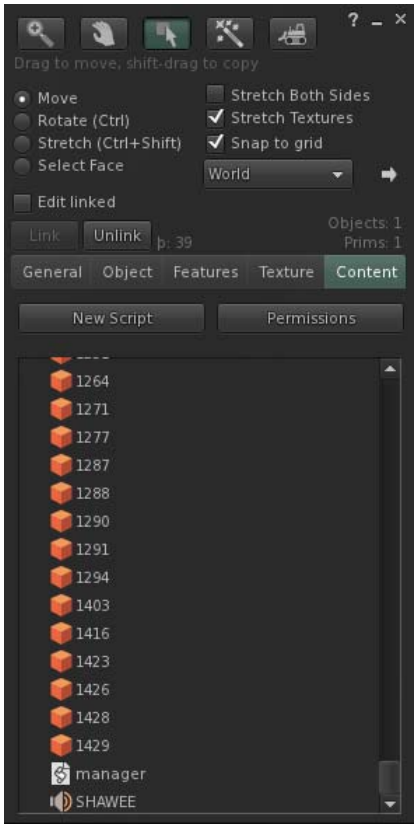

Figure 22: Memorials and script in the Data Process Hub's inventory

```
list getInventoryObjectsList()
{
    list result = [];
    integer n = llGetInventoryNumber(INVENTORY_OBJECT);

    while(n)

        result = llGetInventoryName(INVENTORY_OBJECT, --n) + result;

    return result;
}
```

Figure 23: List of memorials modeled

```
// The current memorial is removed, and the new one rezzed
llSay(DELETE_CHANNEL, "DIE");
llSleep(.5);

if (hasBeenModelled(monument_id))
{
    llSleep(.5);
    llRezObject(monument_id,llGetPos(),<0.0, 0.0, 0.0>,<0.0, 0.0, 0.0, 0.0>,10);
    llPlaySound("SHAWEE",1.0);
    MODELLED = TRUE;
}
```

Figure 24: The previous memorial is removed and the new one created

Each memorial is responsible for setting its correct position and rotation once it has been created, and for removing itself when the correct message is received. To do this, each memorial has its own script specifying this position and rotation and which uses the function *llDie* when required.

```
// Position and rotation for this memorial
vector POS = <237.639,219.583,21.536>;
vector ROT = <0.0,0.0,135.0>;


default
{
    // When created, the position is set
    on_rez(integer start_param)
    {
        llSetPos(POS);
        vector in_radians = ROT * DEG_TO_RAD;
        rotation rotat = llEuler2Rot(in_radians);
        llSetRot(rotat);
        llListen(DELETE_CHANNEL,"","","");
    }

    listen(integer channel, string name, key id, string message)
    {
        // When the message is received, this memorial is removed
        if ((channel == DELETE_CHANNEL)&&(message == "DIE"))
        {
            llDie();
        }
```

Figure 25: Memorial's script. Position is set when created, and the memorial is removed when the correct message is received

4.4. Info Panels

Four panels to the left of the Google Map display interactive content with information about the last memorial selected on the map. When a new memorial is selected, the panels change automatically the content using the information sent by the Data Process Hub, and the user is notified with this change.

Like the Google Map, these panels use Shared Media to show interactive content from the internet. Pages from the CAIN web site with different information about the memorial are displayed by three of these panels, and one last panel embeds Google StreetView with the correct Point of View (POV). To modify this content dynamically, one script in each object uses the function *llSetPrimMediaParams* with the data received from the CAIN server through the

Data Process Hub. Every time a request is receibed by the Hub, new data is sent to the panels where *llSetPrimMediaParam* is called and the new content loaded.

```
setMedia()
{
    string current_url;

    if ((lat == "")||(lng == ""))

        current_content = DEFAULT_CONTENT;

    else

        current_content = build_Content(URL, lat, lng, yaw, pitch, zoom);

    llSetPrimMediaParams(1,[PRIM_MEDIA_CURRENT_URL,current_content]);
}
```

Figure 26: Function to set the content on Shared Media

```
default
{
    state_entry()
    {
        llListen(STREETVIEW_CHANNEL,"","","");
        setMedia();
    }

    listen (integer channel, string name, key id, string message)
    {
        if (channel == STREETVIEW_CHANNEL)
        {
            read_message(message,"/");
            setMedia();
        }
    }
```

Figure 27: With a new message, this is read and the new content loaded