

Creando contenidos en Second Life

ÉSTA ES TU (2DA.) VIDA



Lejos de ser una moda pasajera, Second Life se está convirtiendo en un nuevo escaparate que empresas, organizaciones y hasta partidos políticos usan como nuevo medio para el marketing. El secreto de este mundo virtual está en las posibilidades que ofrece a sus usuarios de modificar el entorno y crear nuevos contenidos.

POR ALBERTO GARCÍA SERRANO

Second Life es un mundo virtual en 3D en el que podemos movernos e interactuar con otros residentes llamados avatares. La versión para Linux se halla aún en fase alfa, sin embargo llevo ya algún tiempo usándola y es totalmente estable. Lo que diferencia a Second Life de otros mundos virtuales es que permite a sus residentes moldearlo a su gusto. Tanto es así que posee su propio lenguaje de programación, gracias al cual podremos modelar cualquier comportamiento en los objetos que vayamos creando. LSL es el acrónimo de Linden Scripting Language y tiene una sintaxis similar a C++ o Java. Sin embargo, el paradigma de programación que utiliza es bien distinto. Más que un lenguaje orientado a objetos, podemos decir que es un lenguaje orientado a eventos. En realidad es similar a una máquina de estados en la que definimos estados y respondemos a los eventos pasando de un estado a otro. En el cuadro 1 podemos ver con más exactitud cómo funciona.

Nuestro Primer Script

Un script LSL va asociado a un objeto, así que comenzaremos creando uno. Para ello debemos estar conectados a Second Life.

Buscaremos un lugar donde poder crear objetos, por ejemplo, en la SandBox de Help Island Public (142, 105, 26). Elegiremos un lugar apartado y pulsaremos el botón derecho del ratón sobre el suelo. En el menú de disco seleccionamos *Crear*. Veremos una ventana como la de la Figura 1. Seleccionaremos *prim cubo* y lo soltaremos sobre el suelo. Nos resta asociar un script LSL al cubo. Para ello pulsamos el botón *Más >>* y después la pestaña *Contenido*. Aquí pulsamos el botón *Nuevo Script*. SL acaba de crear el script del Listado 1. Para verlo, haremos doble click sobre su nombre que, por defecto es *New Script*. La Figura 2 muestra la ventana de edición de scripts.

Este script tiene un solo estado: *Default*. Un estado se define de la siguiente manera:

```
Default {
    // Eventos que puede manejar
    // el objeto en este estado.
}
```

Listado 1: El Script por Defecto

```
01 default
02 {
03     state_entry()
04     {
05         llSay(0, "Hello,
06         Avatar!");
07     }
08     touch_start(integer
09     total_number)
10     {
11         llSay(0, "Touched.");
12     }
```

Listado 2: Estructuras de Control

```

01 // condicional: if
02 if (i==1) {
03     llSay (0, "i vale 1");
04 } else {
05     llSay(0, "i no vale 1");
06 }
07
08 // Repetitiva: for
09 for (contador = 1; contador <
10     10; contador++) {
11     llSay (0, "cuenta: " +
12         (string)contador);
13 }
14
15 // Repetitiva: while
16 integer contador = 1;
17 while (contador < 10) {
18     llSay (0, "cuenta: " +
19         (string)contador);
20     contador++;
21 }
22
23 // Repetitiva: do - while
24 integer contador = 1;
25 do {
26     llSay (0, "cuenta: " +
27         (string)contador);
28     contador++;
29 } while (contador < 10);

```

Dentro de las llaves definiremos los eventos a los que queremos que nuestro objeto sea capaz de responder. Nuestro script tiene dos eventos definidos dentro del estado *Default*: *state_entry* y *touch_start*. El primero es un evento que se dispara de forma automática cuando el objeto entra en el estado. El segundo se dispara cuando un avatar cualquiera toque nuestro objeto. Analicemos con más detalle cada evento.

```

state_entry()
{
    llSay (0, "Hello, Avatar!");
}

```

Cuando un avatar se acerque al objeto, éste entrará en el estado *Default*, y por lo tanto se lanzará el evento *state_entry*.

En el caso concreto que estamos tratando, la única acción que se realiza es mandar el mensaje "Hello, Avatar!" como un mensaje de chat. Para eso usamos la función *llSay*. Esta función tiene el siguiente formato:

```
llSay (canal, "mensaje");
```

La función *llSay* envía un mensaje de chat que puede "oírse" en unos 20 metros a la redonda. Tiene dos parámetros, el primero es el canal por el que queremos transmitir el mensaje y el segundo, el mensaje que queremos transmitir.

El canal 0 es el canal público, es decir, que cualquiera podrá leer el mensaje. El resto de canales, que van de 1 al 2.147.483.648, son privados, y suelen usarse para comunicar unos objetos con otros y hacerlos funcionar de forma sincronizada.

Veamos ahora el evento *touch_start*.

```

touch_start (integer total_number)
{
    llSay(0, "Touched.");
}

```

La única diferencia con el evento anterior es que *touch_start* recibe un parámetro llamado *total_number*, que es de tipo *integer*. Este parámetro indica el número

de agentes (avatares) que están tocando el objeto.

El Lenguaje LSL

Como cualquier otro lenguaje, LSL tiene varios tipos de datos para declarar variables.

- *Integer* Representa un número entero entre -2.147.483.648 y 2.147.483.647. Ejemplo: *integer foo = 10;*
- *Float* Representa un número decimal en punto flotante. El mayor positivo o menor negativo representable con un *float* es +/-3.4028235e38, mientras que el menor positivo o el mayor negativo es +/-1.17549351e-38. Ejemplo: *float bar = 10.5;*
- *String* Almacena una cadena de caracteres. Ejemplo: *string saludo = "Hola";* *saludo = saludo + " mundo";*
- *List* Permite almacenar colecciones de cualquiera de los tipos soportados. Es similar a los arrays. Ejemplo: *list unaLista = ["soy", "una", "lista"];*
- *Vector* Representa un vector tridimensional en formato <X,Y,Z>. Puede representar una posición, una velocidad, una aceleración o un color.

```

vector p = <1.0, 2.0, 3.0>;
vector pos;
pos.x = 1.0;
pos.y = 2.0;
pos.z = 3.0;

```

Listado 3: Estados y Funciones Propias

```

01 string estado;
02
03 setEstado(string _estado) {
04     estado = _estado;
05 }
06
07 default {
08     state_entry() {
09         state apagado;
10     }
11 }
12
13 state encendido {
14     state_entry() {
15         llSay(0, "Estoy
16         encendido");
17         setEstado("encendido");
18     }
19 }
20
21 touch_start(integer total_number){
22     state apagado;
23 }
24
25 state apagado {
26     state_entry() {
27         llSay(0, "Estoy apagado");
28         setEstado("apagado");
29     }
30 }
31
32 touch_start(integer total_number){
33     state encendido;
34 }

```

Listado 4: Script de Teletransporte

```

01 // Coordenadas de destino          total_number)
02 vector target = <150,146,21>;      13 {
03 default                             14     llSay(0,"Pulse el boton
04 {                                   15     derecho del raton y seleccione
05     state_entry()                   16     Ir.");
06     {                               17
07         llSetSitText("Ir");         18 }
08     rotation                         19
09     my_rot=llGetRot();               20 changed(integer change)
10                                     21 {
11                                     22     llSitTarget((target-llGetPos()
12     touch_start(integer              23     )/my_rot,ZERO_ROTATION/my_rot)

```

- *Rotation* Representa una rotación en formato <X,Y,Z,S>. Tiene formato de quaternions, donde los tres primeros números forman un vector que indica en qué ejes se aplicará la rotación, y el cuarto es el ángulo de rotación en sí. En cualquier caso, si este vector no está normalizado, internamente se realizará una normalización (un vector está normalizado cuando su longitud, también llamado módulo, es 1, lo que facilita y agiliza las operaciones matemáticas

realizadas sobre él) . Ejemplo: *rotation rot = <0.1, 2.5, 3.6, 1.0>*

- *Key* Identificador único (UUID) que tiene cada objeto dentro del mundo. Incluido nuestro avatar. Ejemplo: *key objeto = "00000000-0000-0000-0000-0000000000"*;

Desde el punto de vista de las estructuras de control, las de LSL son similares a las de cualquier lenguaje como C o Java. Disponemos de las estructuras clásicas de control: *if, for, while* y *do - while*.

Listado 6: Script de Movimiento

```

01 vector centroRotacion;              14     integer i;
02 default                             15     while (TRUE) {
03 {                                   16     {
04     state_entry()                   17         vector actPos =
05     {                               18         llGetPos();
06         vector iniPos =             19         vector actOffset =
07         llGetPos();                 20         actPos - centroRotacion;
08         centroRotacion = iniPos +   21         vector rotOffset =
09         < 3, 3, 3 >;                 22         actOffset * rotZ;
10     }                               23         vector newPos =
11     touch_start(integer              24         centroRotacion + rotOffset;
12     total_number)                   25     }
13     {                               26     llSetPos( newPos );
14     rotation rotZ =                 27     }
15     llEuler2Rot( < 0, 0, 15 *      28     DEG_TO_RAD > );

```

Listado 5: Script de Rotación

```

01 default {
02     state_entry() {
03
04         llTargetOmega(<0,0,1>,0.2,1.0
05     );

```

En el Cuadro 2 podemos ver ejemplos de estas estructuras.

Crear Funciones y Estados

Las funciones en LSL se declaran de forma similar a C. Declaremos los parámetros de entrada entre paréntesis y el de salida lo devolveremos con *return*.

También podemos crear estados propios usando la palabra reservada *state*. Vamos a verlo con el ejemplo del Listado 3, que usa estados y funciones propias en el que, cada vez que se toque el objeto, cambiará entre el estado encendido y apagado. En este caso hemos definido dos estados en el objeto: *encendido* y *apagado*. También usamos *state* para cambiar de un estado a otro, tal y como se observa en el estado *default* del Listado 3. Nada más entrar en él, hacemos la transición al estado *apagado*. Puede consultar el listado completo de funciones, eventos y estados en [1].

Algunos Ejemplos

Vamos a ver algunos ejemplos. El primero, que se puede ver en el Listado 4, es un script de teletransporte. Puede usarse para transportar nuestro avatar a un punto cercano. En la Figura 3 se puede observar un directorio que, usando este script, permite transportar al visitante a una planta concreta de un edificio. Se utilizan las funciones *llSitTarget* y *llUnSit*, que permiten sentar o levantar al avatar. En este caso se usa un pequeño truco que consiste en sentar al avatar en el punto al que queremos viajar y seguidamente levantarlo. Se hace así porque LSL no posee funciones de teletransporte entre distancias cortas (aunque sí para lugares distantes).

El Listado 5 muestra otro sencillo pero interesante script que usa la función *llTargetOmega* para hacer girar un objeto sobre sí mismo.

Por último, en el Listado 6 podemos observar un script que nos muestra cómo hacer que un objeto se mueva en el espacio gracias a la función `llSetPost`. Este script hace orbitar al objeto alrededor de su eje Z.

Creación de Contenidos

Son múltiples los contenidos demandados por los residentes de Second Life. Algunos de ellos son muy valorados y se pagan bien (Second life tiene su propia moneda, llamada Liden Dollar, que puede cambiarse por dinero real). Vemos algunos ejemplos.

Los vehículos son bastante valorados. Sobre todo aquellos que imitan a vehículos reales (con marcas y modelo concretos). Para crear un vehículo hay que definir sus propiedades físicas, que no serán iguales en un coche, un barco o un avión. Podemos obtener más información sobre programación de vehículos en [4].

El aspecto del avatar también puede ser modificado. En este sentido, la ropa cobra un papel importante. Podemos diseñar y crear ropas para nuestro avatar. Para ello usaremos unos patrones prediseñados que pueden ser descargados de la página oficial de Second Life. Con esos patrones y algún programa de diseño gráfico que soporte capas, como The Gimp, podremos comenzar a diseñar nuestra vestimenta. Para obtener más información véase [5].

Una vez dentro de Second Life, la comunicación con los demás es importante, y no sólo la verbal, que llevaremos a cabo mediante chat, sino también la



Figura 1: Ventana de creación y edición de objetos.

gestual. Por defecto, nuestro avatar tiene una serie de animaciones predefinidas, llamadas gestos. Además de estos gestos podemos crear nuevas animaciones usando por ejemplo Blender, aunque hay programas mucho más sencillos diseñados especialmente para Second Life como *qAvimator*, que puede ser descargado de [6] y tiene versión para Linux.

Conclusiones

Algunos engloban ya a Second Life en lo que se ha dado en llamar la WEB 3.0. Las empresas empiezan a ver su potencial como escaparate al mundo. La interactividad y las posibilidades de Second Life están más allá de las de la WEB actual, y aún sólo logramos imaginar vagamente las posibilidades que nos ofrece este uni-

verso virtual, que no sólo se reducen al plano mercadotécnico, sino que ofrecen un importante potencial en el mundo de la educación. Ya se está diseñando material didáctico basado en mundos virtuales 3D y ofreciendo cursos que se desarrollan íntegramente dentro de Second Life. Hay, de hecho, un intento de integrar la plataforma Moodle con este universo virtual, llamado Soodle, y del que puede obtener más información en [7]. Empresas como Avata.es [2] ya han comenzado a prestar servicios enfocados a este nuevo mercado. ■

Cuadro 1: Eventos, Estados y Funciones

Hemos dicho al principio del artículo que LSL es un lenguaje orientado a eventos. Esto quiere decir que nuestro programa no tiene un comportamiento determinista a priori. En otras palabras, nuestro programa hará unas cosas u otras dependiendo de lo que ocurra a su alrededor (por ejemplo, que un avatar toque el objeto).

Estos eventos desencadenan acciones que modifican el comportamiento de los objetos. Estos comportamientos se modelan mediante estados.

Podemos definir los estados que puede tener un objeto y cambiar de uno a otro. Por ejemplo, si creamos una puerta que se abre automáticamente cuando alguien quiere pasar, podemos definir dos estados: *Abierta* y *Cerrada*. En un principio nuestra puerta está cerrada, pero si alguien se acerca la puerta pasará al estado Abierta. En Second Life todos los objetos deben tener al menos un estado por defecto llamado *Default*.

Según el estado en que se encuentre el objeto realizará diferentes acciones. Estas acciones se ejecutan mediante las *Funciones predefinidas*. En LSL hay cerca de 300 funciones predefinidas, cuya misión es permitir a nuestros objetos interactuar con el resto del mundo. Por ejemplo, hay funciones para hacer que nuestro objeto emita mensajes a través del chat o animarlo y darle movimiento. LSL también nos permite definir funciones propias.

RECURSOS

- [1] Listado completo de funciones, eventos y estados: http://wiki.secondlife.com/wiki/LSL_Portal
- [2] Educación en Second Life: <http://www.avata.es>
- [3] *Exprime Second Life*. Ed. Anaya Multimedia. ISBN: 978-84-415-2284-8
- [4] Programación de vehículos para Second Life: http://wiki.secondlife.com/wiki/Category:LSL_Vehicle
- [5] Cómo crear ropa en Second Life: http://wiki.secondlife.com/wiki/Clothing_Tutorials
- [6] Programa de creación de animaciones *qavimator*: <http://www.qavimator.org/>
- [7] Integración de Moodle con Second Life (Soodle): <http://www.sloodle.org/>