实验2：虚拟世界脚本语言实习

1 LSL 脚本入门指南

1 起步

1.1 简单介绍

1.1.1 关于林登脚本语言

林登脚本语言（*LSL, Linden Scripting Language*），是一种简单易学而又具有强大功能的编程语言。在虚拟世界的虚拟世界中，我们运用林登脚本语言来编辑动作以及其它各种功能。

林登脚本语言的存在，使得程序设计人员和用户都可以建造自己的交互内容。伴随着林登脚本语言的发展，虚拟世界变得更加有趣，更加鲜活。

拥有了林登脚本语言，虚拟世界中的每一位用户就拥有了和程序开发人员同样强大的工具， 你可以自由的使用支配它，当然，随之而来的也有相应的责任和义务。因为通过创造脚本，你还可以制造混乱，甚至对他人造成损害，所以请牢记，在使用林登脚本语言时要遵守我们的社区规范。

通过林登脚本语言，可以在虚拟世界中的任何物体上添加“行为”，包括可以给门上密码锁，使火焰燃烧起来，为物体设计运行轨道等等，任何你能够想想到的都可以通过脚本语言来实现。

只要你学习了最基本的脚本语言，就可以在虚拟世界中充分发挥想象的空间。当你能够熟练使用的时候，你可以尝试为他人编写脚本。

1.1.2 如果我没有编程基础怎么办？

没问题。虽然觉有编程和数学基础的人可能会上手更快，并且编写出更复杂的脚本。不过一个完全没有编程经验的新手，也完全可以学会最基础的脚本语言，并且在短时间内创造出你自己的简单程序。

即使是那些不愿意花时间来学习脚本语言的用户，也可以通过修改现有的脚本来满足自己的需要。例如，修改门锁密码的工作就很简单。

1.1.3 如何学习脚本语言？

就从现在开始吧：

首先，阅读这一章的第一部分，尤其是那些样例脚本。

然后，阅读这一章的第二部分，你将会更好的了解林登脚本语言的结构和功能。

接下来，尝试自己编写一些脚本，看看会有什么结果。

除此之外，下面的这些资源将对你有所帮助：

- 脚本语言资料——在帮助菜单中，它们包含了每个功能的各个细节，你可以通过关键字来搜索，当然也可以将这些文献打印出来。
- 用户——很多用户都可以熟练的使用脚本语言，并且乐于向你提供帮助。
- 论坛——虚拟校园的工作人员和用户们会在论坛上讨论问题，交流观点，提供帮助。
- 编程讨论会——一些有经验的程序员会举办讨论会。可以在网上搜索他们的日程及地点。
- 技术支持——你可以向林登工作室的技术支持寻求帮助。

1.1.4 关于这份使用指南

这份使用指南写给那些对林登脚本语言具有零基础的初学者。它将教会你如何创建自己的简单脚本，而不是单纯的向你灌输学术术语。

1.2 寻找脚本

你可以在虚拟世界中的任何地方找到脚本！

那些你身边的物体就有可能具有脚本。右键点击它们，选择 Edit，在 Content 栏目中可以看到该物体是否具有脚本。当然，一些他人拥有的物品也许你无法察看，你可以查看那些属于你自己的物品和公共物品。另外，在你的 Inventory 目录中有脚本文件夹（script folder）。论坛也是一个获取脚本的好地方。

1.3 打开脚本

如果需要打开你的 Invertory 目录中的脚本，通过双击鼠标打开即可。

如果需要打开物体中的脚本，请按照以下步骤操作：

a) 右键点击物体，选择 Edit；
b) 点击 Content 栏；
c) 如果没有显示的脚本，双击打开 Content 文件夹；
d) 双击打开脚本。

1.4 将脚本安装到物体

将脚本安装到物体，只需将脚本从 Inventory 文件夹中拖拽到物体上即可，或者将脚本拖拽到物体的 Content 文件夹中。

下面我们来尝试一下：

a) 创建一个简单物体。（可以是球，盒子，圆锥等任何物体）；
b) 打开 Inventory 文件夹，将“Rotation Script”拖拽到物体上；
c) 如果你的 Inventory 文件夹中没有这个脚本，右键，选择“New Script”，命名为“Rotation Script”，然后双击打开， 输入以下内容：

```
default
{
    state_entry()
    {
```

```
        llTargetOmega(<0,0,1>,PI,1.0);
    }
}
```

d) 关闭物体的属性栏，现在，你可以用看到物体在旋转了！
e) 察看刚才我们所添加的脚本，运用 1.3 中的步骤。

1.5 修改脚本

1.5.1 为什么要从修改脚本学起？

修改脚本可以让你更快的了解脚本的工作原理，而且远比盲目的着手编写脚本要容易得多。

1.5.2 一个简单的修改

双击打开上一节中介绍的 Rotation Script
这个简单的脚本只包含有一个函数：**llTargetOmega**
llTargetOmega(<0,0,1>,PI,1.0)
函数中包含的数字是函数的参数，它决定着函数的功能。
我们在林登脚本语言资料库中搜索 llTargetOmega，会得到如下资料：

llTargetOmega
llTargetOmega(vector axis, float spinrate, float gain);
Attempt to spin at spinrate with strength gain. A gain of 0.0 cancels the spin.

如果你以前接触过编程，对上面函数中的向量、浮点等都不会陌生。我们可以大致猜测函数的三个参数的作用：

a) 改变向量会改变旋转的方向，三个数字分别代表 x,y,z 轴；
b) 改变第二个数会改变旋转的速度；
c) 第三个数字，与使得物体旋转的力或者扭矩有关，当它的值为 0 时旋转会停止。

1.5.3 谨慎修改

因为我们对脚本的修改可能会导致混乱，所以在修改之前要做好恢复如初的准备。当我们把脚本从 Inventory 文件夹中拖拽至物体上时，文件夹中还保留着最初的版本。因此，我们要牢记的是，编辑物体上的脚本，而不是脚本文件夹中的，而当你设计出自己满意的脚本时，可以把它重新命名后再放回到文件夹中。

1.5.4 尝试做些改动

a) 在脚本编辑窗口中，将向量改为<1,0,0>；
b) 点击保存；
c) 关闭编辑状态，你将看到物体在围绕 x 轴旋转！
d) 将向量改为<.5,2,1>以及<0,0,-1>，看看出现的效果；
e) 复制一个同样的物体，修改其中一个的旋转速度。

1.6 脚本举例解析——Hello Avatar

Hello Avatar 是一个非常简单的，单状态（one-state）脚本。

```
default
{
        state_entry ()
        {
                llSay (0, “Hello, Avatar!”);
        }

        touch_start (integer total_number)
        {
                llSay (0, “Touched.”);
        }
}
```

在脚本的第一行是这样一句：

default

“Default”设定了初始状态，所有的脚本都以这样一句话开始，即使是简单到只有一个状态的脚本也不例外。

下一行：

state_entry()

说明了当脚本达到这个状态时，执行下面两个括号中的内容：

```
{
        llSay (0, “Hello, Avatar!”);
}
```

我们在资料库中查询 llSay 函数会看到如下说明：

llSay

llSay(integer channel, string text)

第一个整数，决定了脚本用来交流的通道。通道 0 是公共通道：任何发送到这个通道的信息都会出现在聊天窗口中。通道 0 至 2,147,483,648 是非公共通道，你通过这些通道发送的信息将不会显示在聊天中。后面的字符串是这个脚本将要发送到 0 通道的信息，你可以在里面写上你想说的话。

下一个句柄（handle）以及它所包含的内容是：

```
touch_start (integer total_number)
{
        llSay (0, “Touched.”);
}
```

touch_start 函数可以检测一个物体是否可以触摸或点击。当它被点击时，将执行括号中的内容，在这里，是显示“Touched”。

1.7 样例脚本注释

这里有一些添加了注释的脚本。仔细阅读并尝试使用它们，将会帮助你快速掌握脚本语言。

```
// hello world
// all scripts need a default state
default
{
    // the state_entry event handler is called when the
    // state (in this case, the default state) is entered
    state_entry()
    {
        // llSay is a library function call that says the string
        // on channel 0 (the channel avatars chat and listen on
        llSay(0, "Hello, world!");
    }
}
```

```
// hello world on rez from inventory
default
{
    // the on_rez event handler is called whenever an object is
    // rez-ed out of inventory or by a script call
    // start_param is 0 is rez-ed from inventory, but can be set
    // when rez-ed by a script call
    on_rez(integer start_param)
    {
        llSay(0, "Hello, world!");
    }
}
```

```
// hello world when clicked on by an avatar
default
{
    state_entry()
    {
        llSay(0, "Hello, world!");
    }
    // the touch_start handler is called when an avatar clicks on the
    // object total_number is the total_number of avatars that
    // started clicking on the object since the last touch_start
    // handler was called
    touch_start(integer total_number)
```

```
    {
        llSay(0, "I've been touched");
    }
}


// change the color of an object when clicked on
// global variables
// global variables are used to store data that is available inside
// any function or event handler
// initialize this variable to false
integer COLOR_ON = FALSE;
// global functions
// global functions can be called by other global functions or by
// event handlers
// change the color of the object based on the COLOR_ON global variable
set_color(integer color)
{
    // if statements use the same syntax as C
    if (color == TRUE)
    {
        // colors are set via vectors where the first value corresponds to
        // red, the second green, the third blue
        // this call sets all the sides of the object to white
        llSetColor(<1,1,1>, ALL_SIDES);
    }
    else
    {
        // otherwise, set the color to black
        llSetColor(<0,0,0>, ALL_SIDES);
    }
}
default
{
    touch_start(integer total_number)
    {
        // lsl supports C style Boolean and bit operations,
        // in this case
        // logical NOT
        COLOR_ON = !COLOR_ON;
        // tell the world what the setting is, using a type cast
        // convert the integer to a string and using the +
        // operator to concatentate
        // the string
```

```
            llSay(0, "COLOR_ON set to " + (string)COLOR_ON);
            // actually change the color
            set_color(COLOR_ON);
        }
}


// change the color of an object when the owner tells it "on" or "off"
// use the same global variable and global function
integer COLOR_ON = FALSE;
        set_color(integer color)
        {
            if (color == TRUE)
            {
                llSetColor(<1,1,1>, ALL_SIDES);
            }
            else
            {
                llSetColor(<0,0,0>, ALL_SIDES);
            }
        }
default
{
        state_entry()
        {
            // llListen library function call to cause the script to
            // listen for chat from its owner
            // the first empty string is an optional name argument,
            // allowing the listen to be set to listen to objects and
            // avatars a certain name
            // the second empty string specifies the text to listen
            // for but since we want to get both on
            // and off we leave that blank as well
            // if the llGetOwner() was changed to an empty string then
            // all chat on channel 0 would be heard
            llListen(0, "", llGetOwner(), "");
        }
        // the listen event is called when the object hears chat that
        // meets the condition specified
        // by the llListen library function
        // scripts can have more than one listen active at a time
        listen(integer channel, string name, key id, string message)
        {
            // change COLOR_ON based in the message text
```

```
        if (message == "on")
        {
            COLOR_ON = TRUE;
        }
        else if (message == "off")
        {
            COLOR_ON = FALSE;
        }
        llSay(0, "COLOR_ON set to " + (string)COLOR_ON);
        set_color(COLOR_ON);
    }
}


// a door that moves up and down when clicked on
// this script causes an object to move up and down when clicked on
// it is set into position by the owner chatting "ready" at it
// we need a global to store whether the door is open or closed
integer CLOSED = TRUE;
default
{
    state_entry()
    {
        // we only need to listen for the command "ready"
        llListen(0, "", llGetOwner(), "ready");
    }
    listen(integer channel, string name, key id, string message)
    {
        // when the door is told ready, assume that it is in the
        // proper closed position
        CLOSED = TRUE;
    }
    touch_start(integer total_number)
    {
        // we need to know where we are so that we can decide where
        // to move to
        // llGetPos() is a library function that returns our
        // to move to current position
        vector our_position = llGetPos();
        // we also need to know how big we are so we know how much
        // to move
        vector our_scale = llGetScale();
        // we'll move along the up (z) axis, so let's store that
        // vectors access x,y,and z values via .x, .y, and .z
```

```
        float z_scale = our_scale.z;
        // if we're closed, move, otherwise move down
        // also, set CLOSED correctly
        if (CLOSED)
        {
            our_position.z += z_scale;
            CLOSED = FALSE;
        }
        else
        {
            our_position.z -= z_scale;
            CLOSED = TRUE;
        }
        // now set the object to the new position
        llSetPos(our_position);
        // set the closed variable to the new setting
    }
}
```

1.8 提示

- 经常保存你的文件！
- 一个物体通常会包含多个脚本，如果你希望一个物体完成很多功能，编写和调试多个简单的脚本会比一个复杂的脚本要容易得多！

2 林登脚本语言规则

2.1 概述

林登脚本语言（*LSL, Linden Scripting Language*），在语法上类似于 C 语言和 Java。它运用的是事件触发模式，也就是说，事件会引发特定代码的运行。事件句柄是是那些对事件进行设置、复位或其他反应的函数命令。而库函数（任何以 ll 开头的单词）会调用和预编译你程序中有用的模块。

接下来，你会了解到林登脚本语言中一些基本概念及组成部分。如果你希望掌握更多，可以通过帮助菜单来查看完整的脚本语言资料，或者打开安装目录下的 html 文件进行浏览。（默认位置为 C:\Program Files\Hippo_OpenSim_Viewer\lsl_guide.html）

2.2 制作脚本语言的步骤

制作脚本语言最基本的步骤包括：

- 编写
- 保存
- 测试
- 改进

在保存的过程中，会自动进行编译。如果你的脚本种有错误，将会得到提示。

2.3 注释（Comments）

注释是代码中的文本信息，它们并不具备任何的程序功能，但是好的注释能够清晰的说明脚本的作用，便于代码的阅读和修改。尤其是当你需要对自己很久前编写的代码进行升级或改进时，注释就显得尤为重要。

注释通常以双斜杠开头：//

当你需要写很长的注释时，要在每一行前加上双斜杠，不然编译器将无法分辨注释与代码。

例如：

```
// This is a comment.

// This is a really, really, really, really, really, really, really,
// really, really long comment, but works, because it has the double
// slashes on each and every line.

// This starts out as a comment, but, since the slashes are missing on
the second line, it actually stops the program from working.
```

2.4 函数（Functions）

在林登脚本语言中，有 200 个左右的函数可供你使用。这些函数已经经过编译和测试，直接运用它们可以节省你很多的时间。在帮助菜单的脚本函数资料中，有所有这些函数的功能及使用介绍。你也可以定义你自己的函数，只要函数名不和已有的保留字、常量名称以及函数名称冲突即可。

可调用的库函数通常以”ll”开头，正如我们在上一部分已经讲述过的：

llTargetOmega(vector axis, float spinrate, float gain);

llSay(integer channel, string text)

在函数的参量中，经常出现的有以下几类：

- Float——浮点型，可以表示小数。
- Integer——整型，只能表示整数。
- String——字符串，可以是单词、短语或句子。
- Vector——向量，由三个浮点型组成。

2.5 状态（States）

林登脚本语言是由状态组成的。状态，指的是一种特定的条件或模式。例如，门就可能具有两种状态，开和关。每一个脚本都有它的初始状态，一般简单的脚本只具有一种状态，就是默认初始状态（**default state**），而复杂的脚本可以包含多个状态。

2.6 事件（Events）

事件，顾名思义就是事情的发生。一个事件可以是：

- 进入到初始或其它状态
- 一个人的靠近
- 一次鼠标点击
- 一次键盘敲击
- 两个物体的碰撞

……

2.7 句柄（Handles）和花括号（Brackets）

句柄（事件句柄），通过探测相应事件来执行它所包含的代码，也就是它后面的花括号中所包含的部分。句柄不是一次性的，对每一次特定事件的发生句柄都会做出相应的反应。花括号的功能是组织代码段，就是将代码包含在一对花括号内，花括号总是成对出现和一一对应的。

2.8 全局与局部（Global and Local）

变量和函数可以是全局的——作用于脚本中的任何地方，也可以是局部的——只作用于脚本中的某一部分。对于简单的脚本来说，全局和局部没有过多的影响。不过对于那些具有很多相互关联的函数的复杂脚本来说，全局化会使脚本更加清晰。

2.9 编程规则

脚本语言的编写有一些默认的规则：

- 注释可以出现在脚本中的任何位置，任何你认为需要的位置。
- 全局变量和全局函数通常写在脚本的前几行。
- 接下来，是各个不同的状态和你需要句柄来处理的事件，通常以默认初始状态（Default）开始。

3. LSL 脚本资源

《Beginners_Script_Guide.pdf》

《lsl guide book.pdf》

《Using the Linden Script Language.mht》：一些有趣的例子

《Linden Scripting Language Reference》：手册和函数