Second Life Scripting Basics

• Second Life has its own programming language, in much the same way that web browsers have JavaScript

• This language, called Linden Scripting Language or LSL for short, can imbue rudimentary “behavior” on the objects that you create in the environment

• The language also has “hooks” for determining when certain events take place

• With events as “stimuli” and scripted behavior as “responses,” LSL can be used to create simple agents

Getting Started

• Go someplace where you can build objects

• Right-click on the ground, and choose Create

• Choose an object to create, then click on the ground to instantiate or rez it at that location

• Click on the Content tab if it isn’t already selected

• Click on the New Script… button

• Double-click on the New Script icon that appears within the Contents folder
States, Events, and Functions

• A typical script consists of a number of states — a state is some “situation” for a Second Life object.

• For each state, the object can be told if certain events take place — for example, whether it has been touched, sent a message, or changed, to name a few.

• The script then handles an event; handling ranges from changing state to performing some computation, or otherwise interacting with the 3D environment, typically through one or more available functions.

• The standard “starter” script, provided whenever a new script is created, looks like this (some lines are compressed to save space on this handout):

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

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

• The script shows one state (default), two event handlers (state_entry and touch_start), and the use of one built-in function (llSay).

• Choose Scripting Guide… from the Help menu to see a rudimentary list of available events and functions, or access the LSL Portal at http://wiki.secondlife.com/wiki/LSL_Portal to learn more.
Planning Out an SL Agent

• A basic approach to putting together a Second Life (SL) agent is to:

  ◇ Figure out what states this agent can have (in particular, give each state a name)
  ◇ Plan what events move the agent from one state to another (keep that list of events from the Scripting Guide handy to see what the object can detect)
  ◇ Determine what the agent will do, in terms of LSL functions in response to events (this can happen whether or not the agent’s state changes too; again, the Scripting Guide helps here)

• When you have a general idea for what you want to happen, start by listing the states in your script; for each state, list the events to be handled when in that state; finally, fill in the code that handles each event

• For example, the script below makes an object get “irritated” when touched once, then “mad” when touched again, “saying” so each time; if it isn’t touched for five seconds, it returns to its default state:

```lsl
default {
  touch_start(integer total_number) {
    state_entry();
    llSay(0, "I'm irritated.");
    llSetTimerEvent(5.0);
  }
}

state irritated {
  state_entry() {
    llSay(0, "I'm irritated.");
    llSetTimerEvent(5.0);
  }
  touch_start(integer total_number) {
    state mad;
  }
  timer() {
    llSetTimerEvent(0.0);
    llSay(0, "Going back to normal...");
    state default;
  }
}

state mad {
  state_entry() {
    llSay(0, "I'm mad!");
    llSetTimerEvent(5.0);
  }
  touch_start(integer total_number) {
    llSay(0, "I'm still mad!");
    // Stay mad for 5 more seconds.
    llSetTimerEvent(5.0);
  }
  timer() {
    llSetTimerEvent(0.0);
    llSay(0, "Going back to normal...");
    state default;
  }
}
```

Note how the passage of time is another event that can be detected.
Things to Do

Second Life’s functions (all of which start with two lowercase Ls, such as llSay) define the range of things that you can get your agent to do; some examples:

- **llTargetOmega** spins the object along some axis and speed

- **llPlaySound** will make the object emit a sound; you can grab sounds from the Library section of the Inventory window; place these in the Contents folder alongside your script to make them available

- **llSetAlpha** sets an object’s transparency or alpha value

- **llSensor** and **llSensorRepeat** perform “scans” of an object’s vicinity for other objects (including avatars); when an object is detected, a sensor event is triggered — useful for making objects do something when someone walks up to them

- **llGetPos** and **llSetPos** read and change an object’s position, respectively; positions are represented by vectors, or \(<x, y, z>\) triplets

- **llListen** makes the object “listen” for chat messages; when an appropriate message is detected, a listen event is triggered — this is the foundation for creating agents that respond to messages from other agents

- **llDialog** pops a visible message with button choices — useful for menus or multiple-choice questions
But Wait, There’s More!

This handout only scratches the surface of what can be done in LSL; check the Scripting Guide as well as the LSL Portal for information on:

- Animations
- Interacting with inventory
- Monetary transactions
- Network activities: URLs, e-mail, instant messages
- Vehicles