

Scripting in Second Life

1. Second Life Overview
2. Creating Objects
3. Linden Scripting Language (LSL)

Sariseelia Sore

Acting Principle Lecturer



LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

Scripting in Second Life

1. Second Life Overview
2. Creating Objects
3. Linden Scripting Language (LSL)



A virtual world is an online environment intended for its users to interact with others via avatars



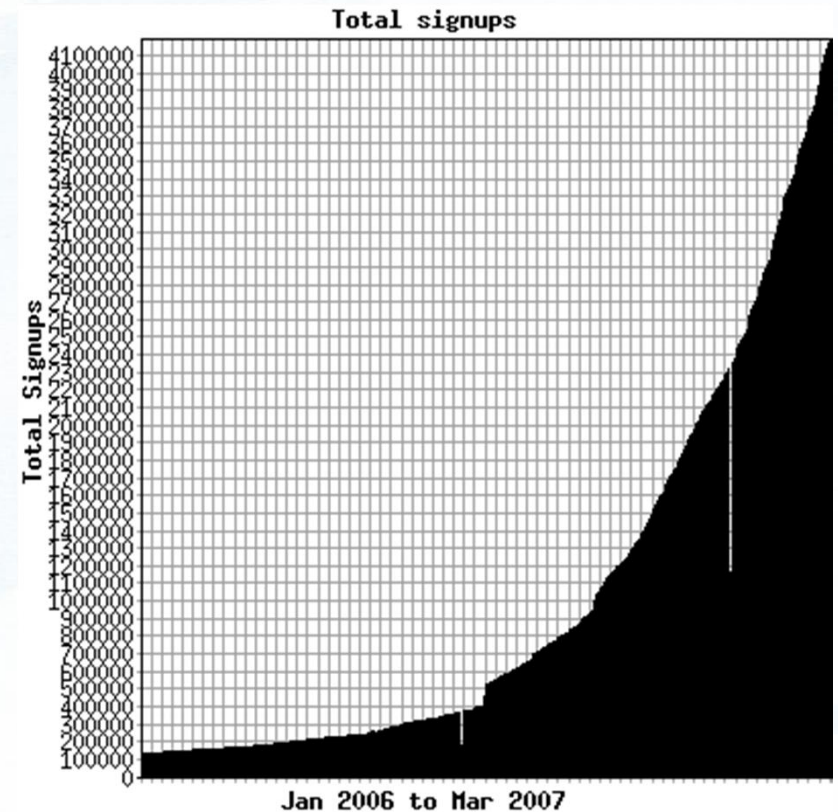


- Virtual shared 3D environment (~3D WWW)
- Origin in game world → non-academic language (*rez, prim,...*)
- The environment consists of objects (buildings, plants, etc.)
- Objects are made of primitives
- Primitives may have functionalities (Scripts)
- Browsing with virtual personalities (Avatars)





- The *best known* virtual world at the moment
- Developed by Linden Research, Inc (Philip Rosedale)
- Launched in June 2003
- Came to international attention in late 2006 and early 2007



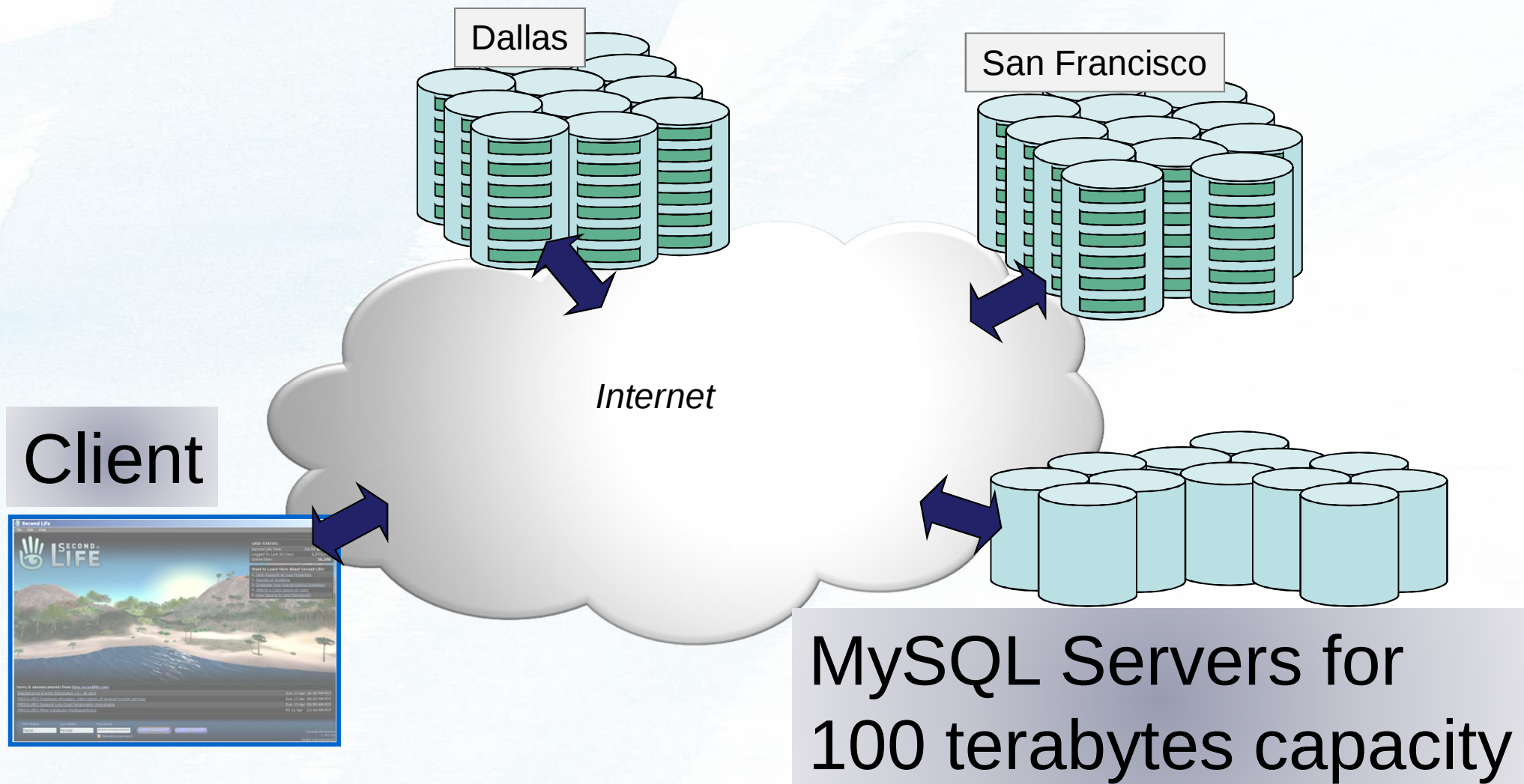
Source: http://en.wikipedia.org/wiki/Image:Graph_of_Second_Life_population.png



LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

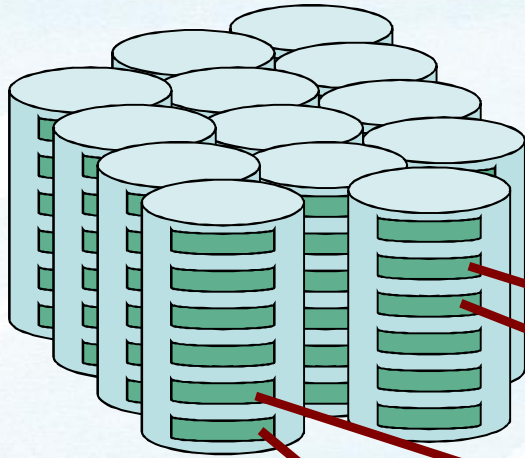
Client-Server Architecture

Thousands of Debian Servers

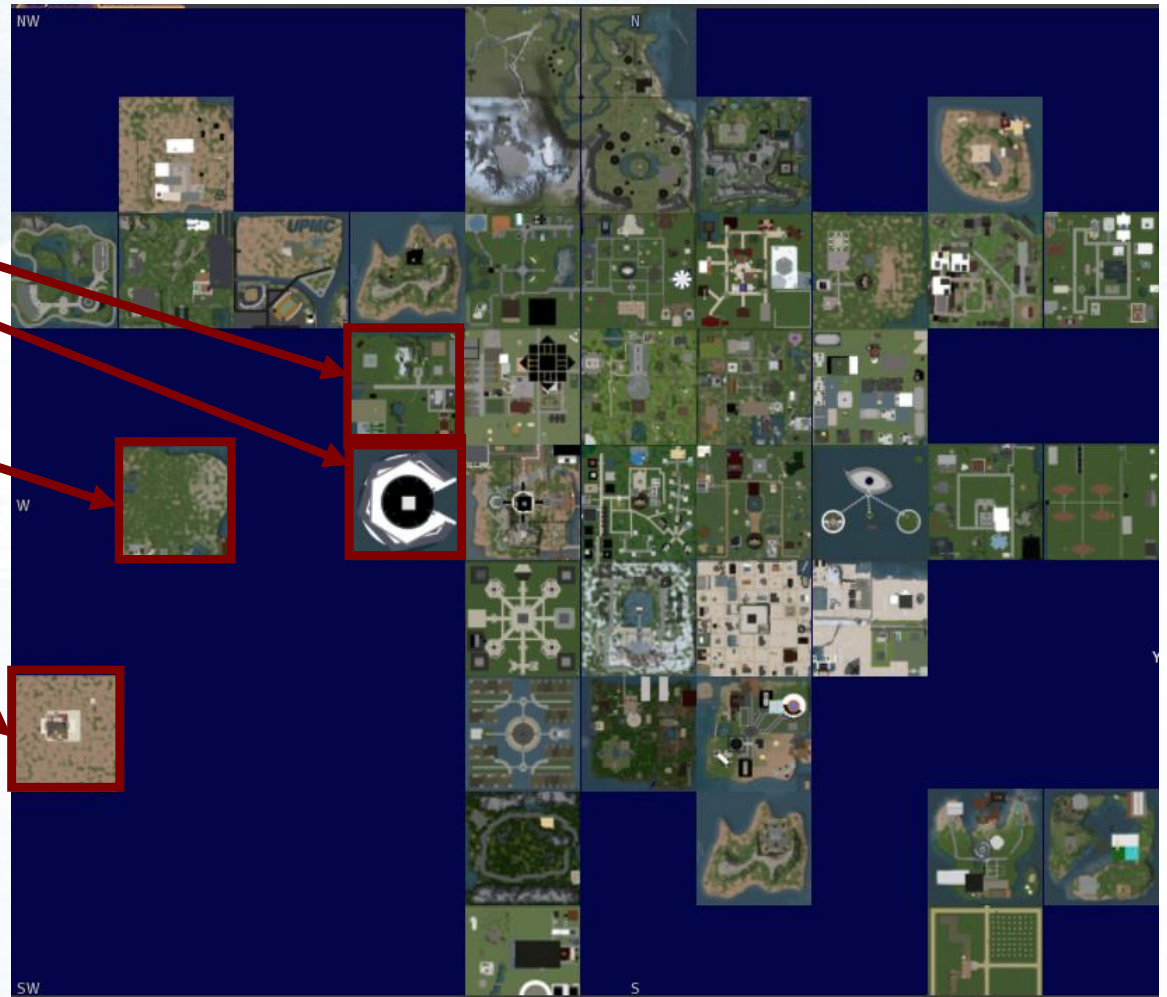


Region Simulators (Sims)

Second Life World Map



Each region
(256 m x 256 m)
is simulated by a
single named
server instance



For now, Linden Lab is the only company that runs sims

Avatars

- Resident: people providing content and contributing to the experience (*user of SL*)
- Avatar: The appearance of a resident



Moving Around



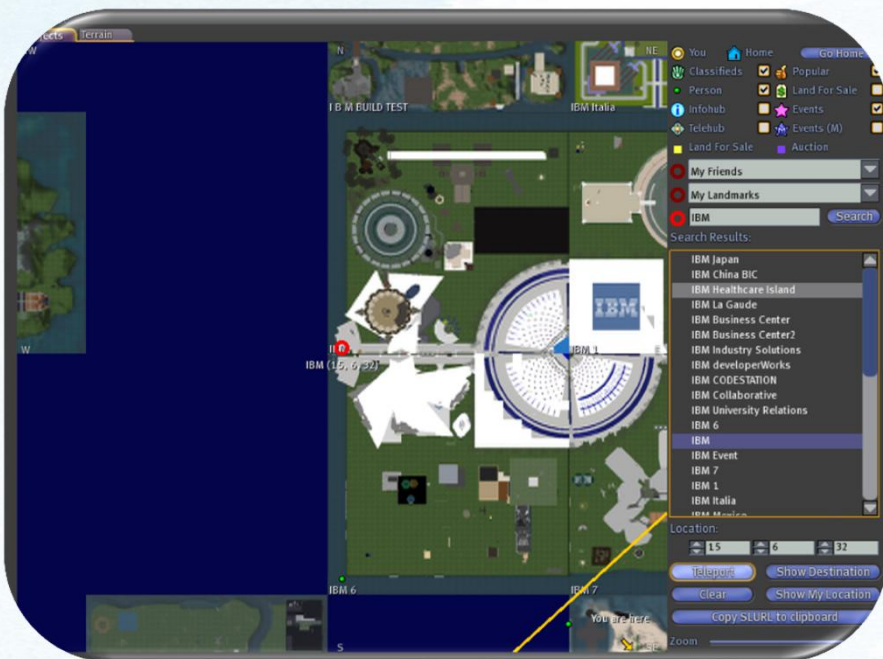
You can move **by foot**
walk, run and jump...



... **fly**...

... **ride** in vehicles...

... or **teleport** anywhere
in SL in an instant!



Communicating

private: Instant Messages (IM)

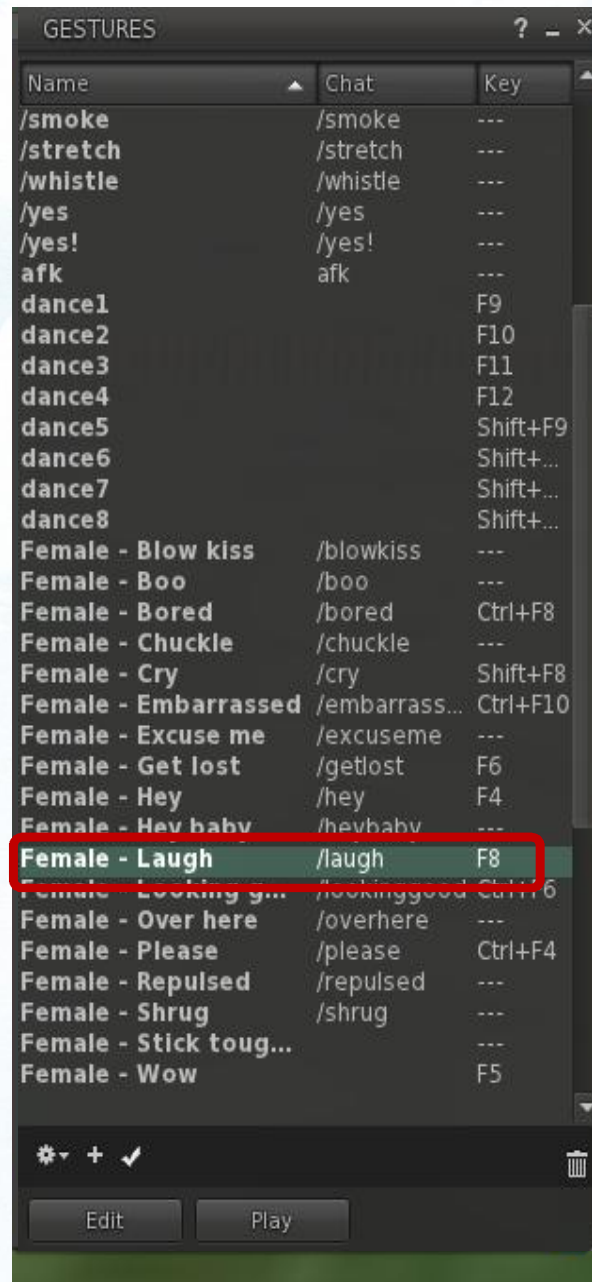
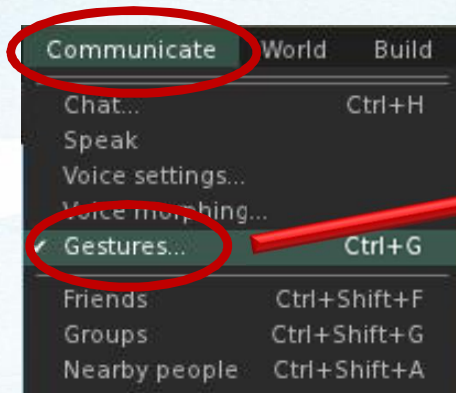
public: local chat
(whisper, talk, shout)
or group IM



verbally: text and voice

non-verbally:
poses, animations and gestures

Gestures





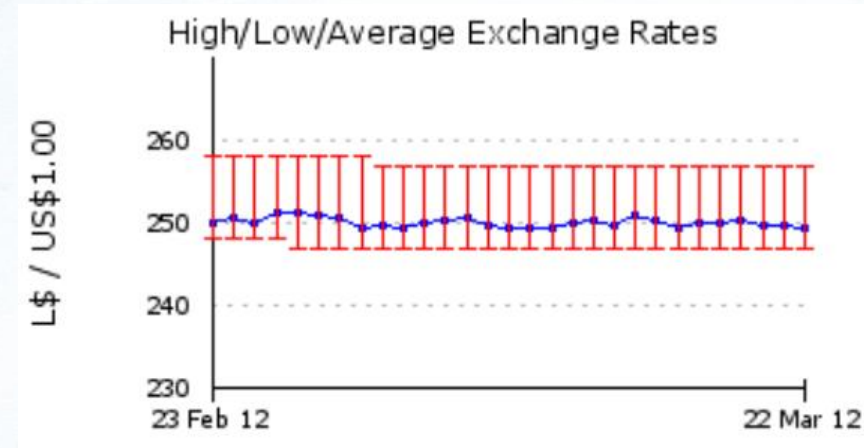
Economy



Financed by residents

Linden Lab: SL “central bank”

Linden Dollar (L\$)



<https://secondlife.com/my/lindex/market.php>

Exchange rate quite stable: L\$ 255 to 1 US\$
(23.3.12: 1 € = 1,3242 US\$)

=> 1 € ≈ 338 L\$ => 100 L\$ ≈ 30 snt

Businesses and Organizations



1. Solely in-world businesses
 - Selling virtual goods and services
2. Companies participating SL
 - Assisting and advising real-life businesses on presenting themselves in SL

Scripting in Second Life

1. Second Life Overview
2. Creating Objects
3. Linden Scripting Language (LSL)



SL is maintained by Linden Lab, but
it's created by it's residents!

“Your World. Your Imagination.”



Creating Second Life

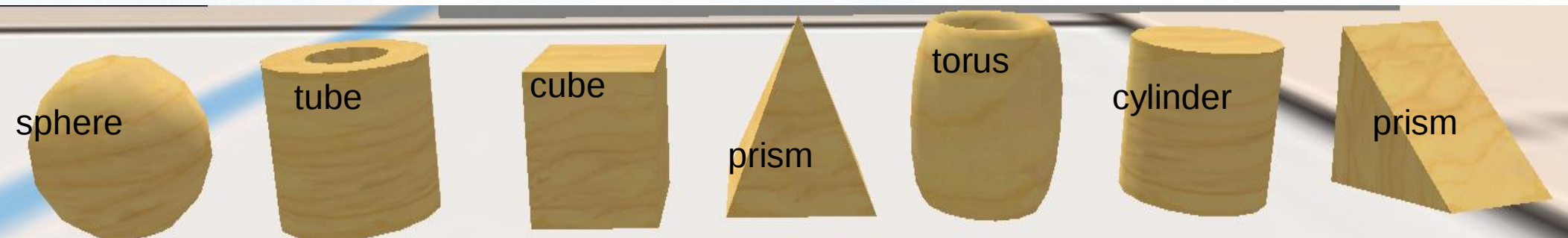
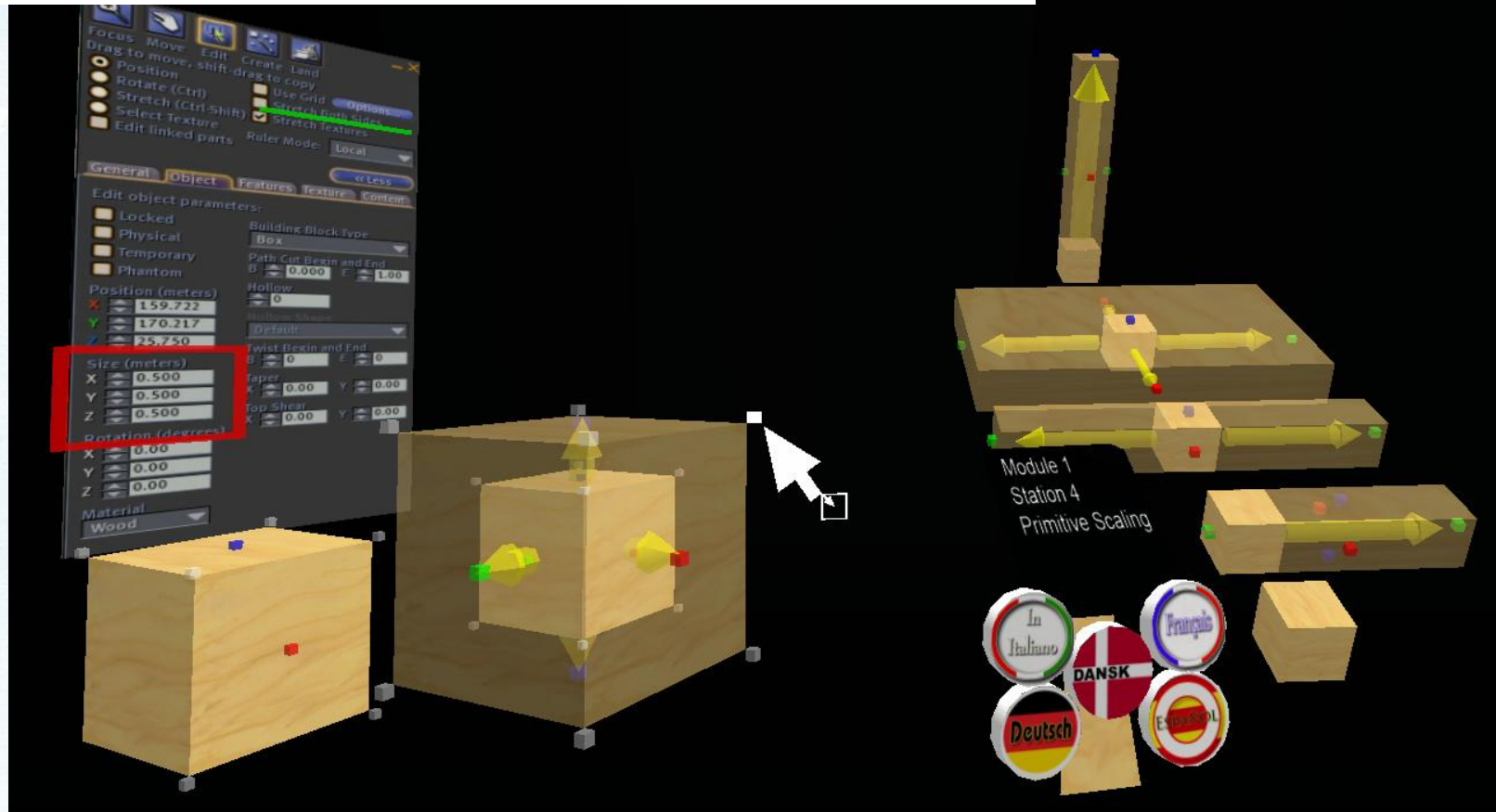
- All the objects in SL are created with a build-in 3D modeling tool
- The behavior is added with the Linden Scripting Language (LSL)



“You create it, you own it.”

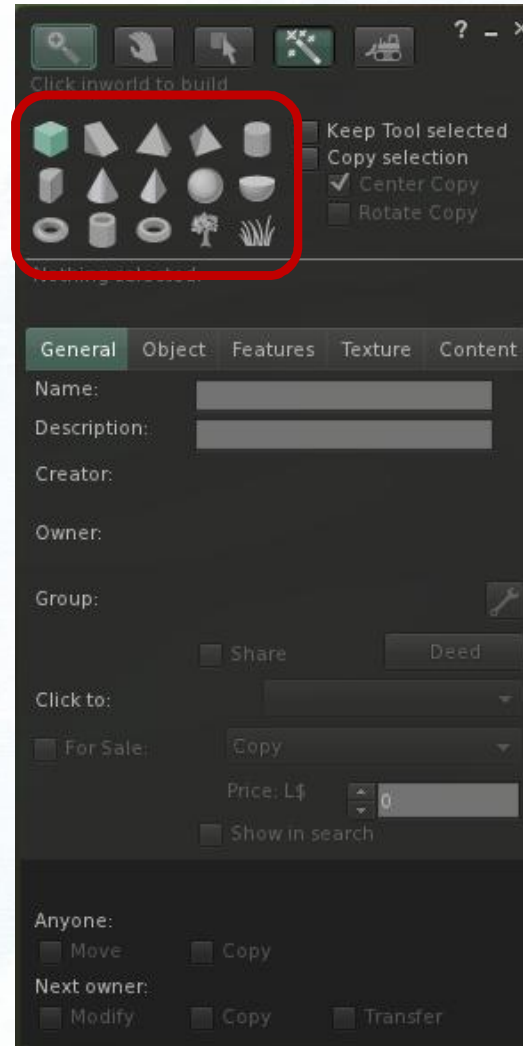
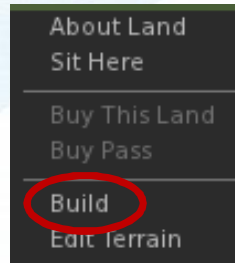
Primitives (Prims)

The Ivory Tower Library of Primitives (Natoma 210, 164, 27)



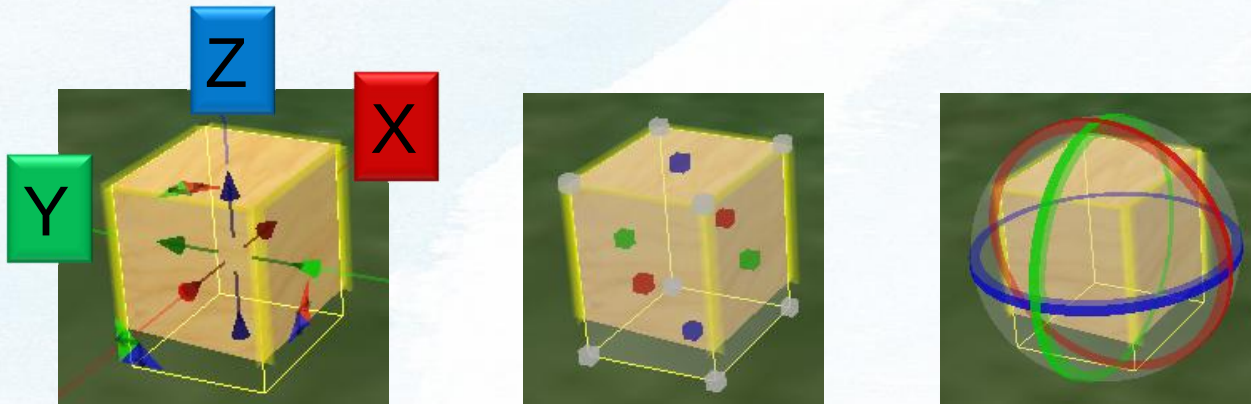
Creating Prims

1. Right-click the ground
2. Choose Build from the opened menu
3. Left-click on the prim type you want to build
4. Left-click the ground

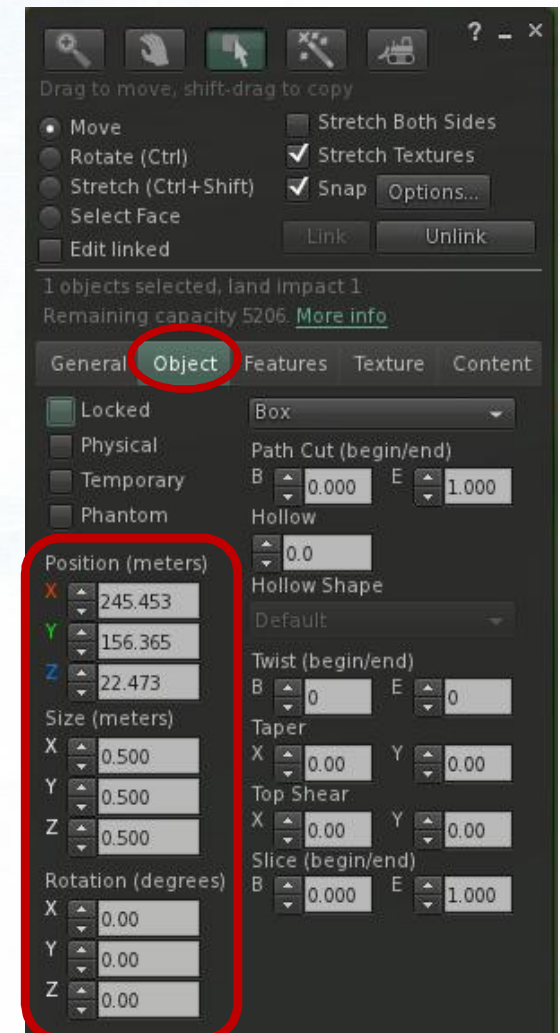


Modifying Prims

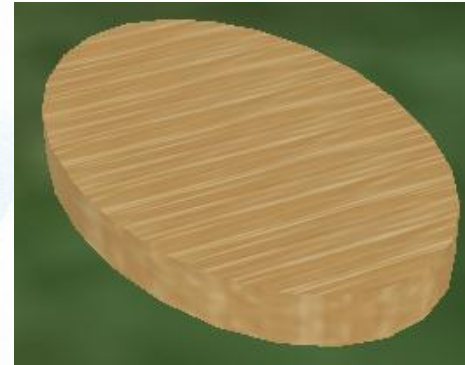
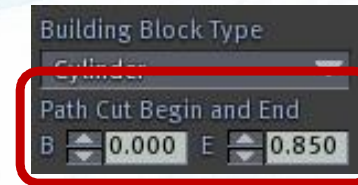
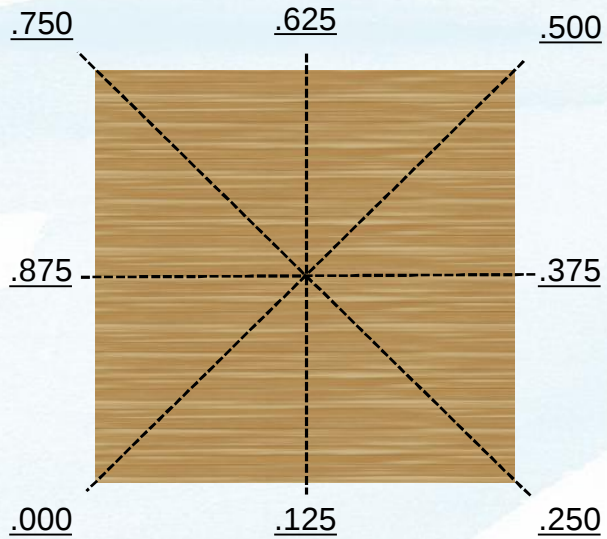
- Prims can be moved, resized and rotated using the mouse...



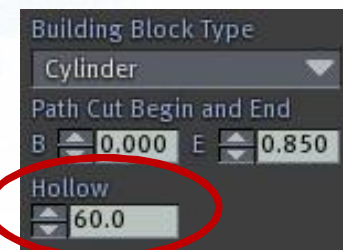
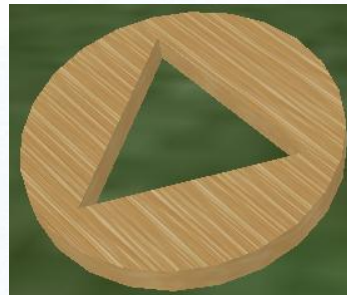
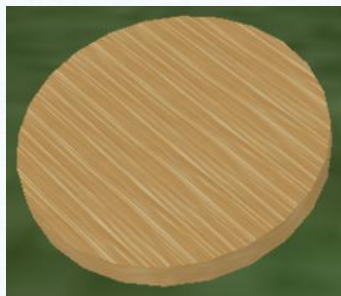
- ... **or** setting the values of x, y, and z on the Object tab of Edit window



Bath Cut

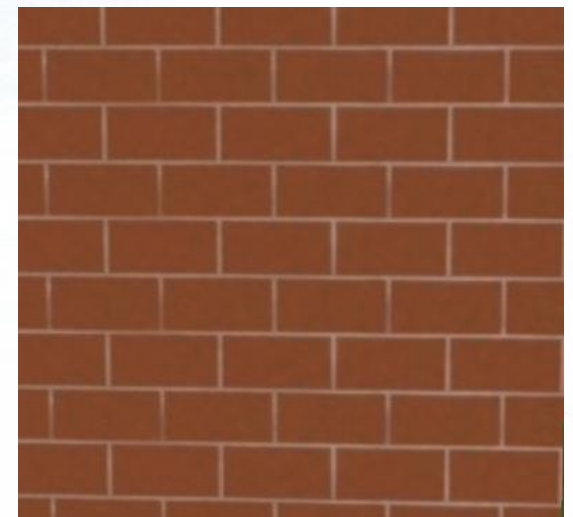
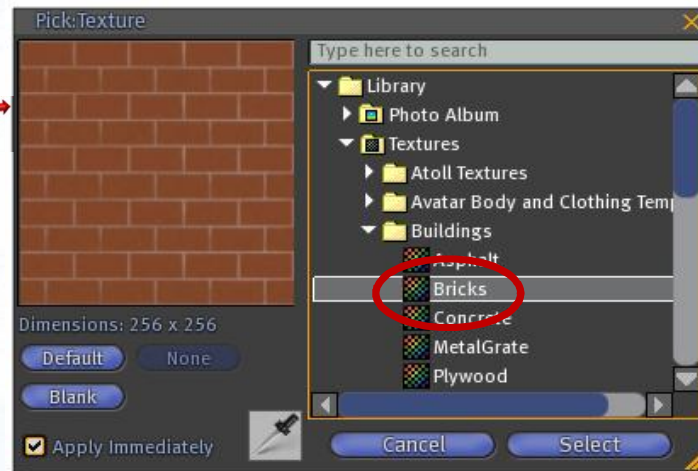
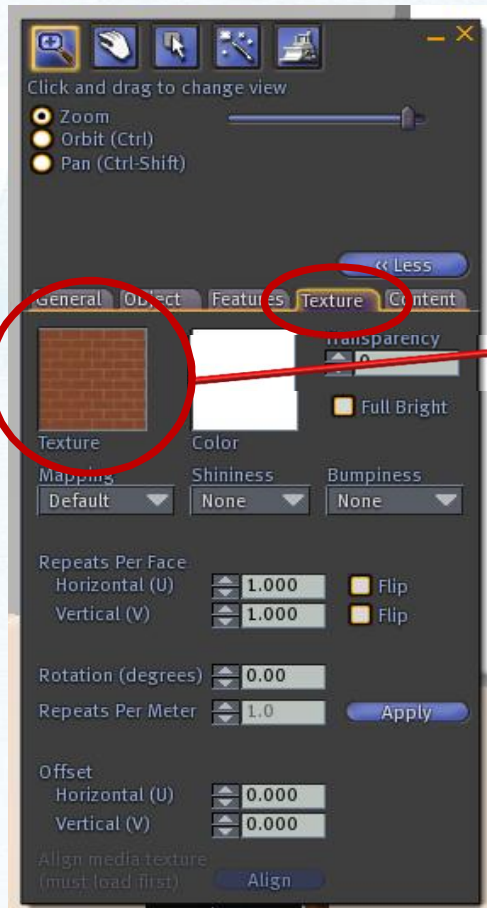


Hallow



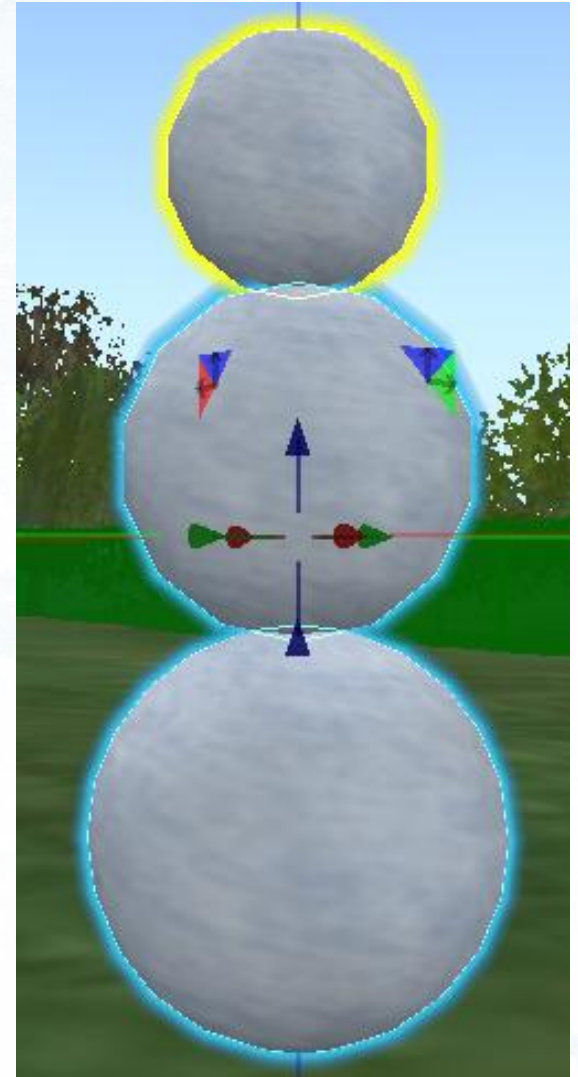
Texture

- Pretty large collection exists
- Default texture plywood
- Own textures
 - Resolution power of 2 (32x32, ...1024x1024)
 - Uploading 10 L\$
 - .tga, .bmp, .jpg, or .jpeg



Linked Prims - Objects

- Linking
 - Select the prims → Ctrl+L (Tools → Link)
 - Max 256 prims, for physical objects 31 prims
- Unlinking
 - Shift+Ctrl+L (Tools → Unlink)
- Root prim
 - The last prim selected (glow yellow)
 - Carries the most of the characteristics of the linked set (e.g. name, and scripts)



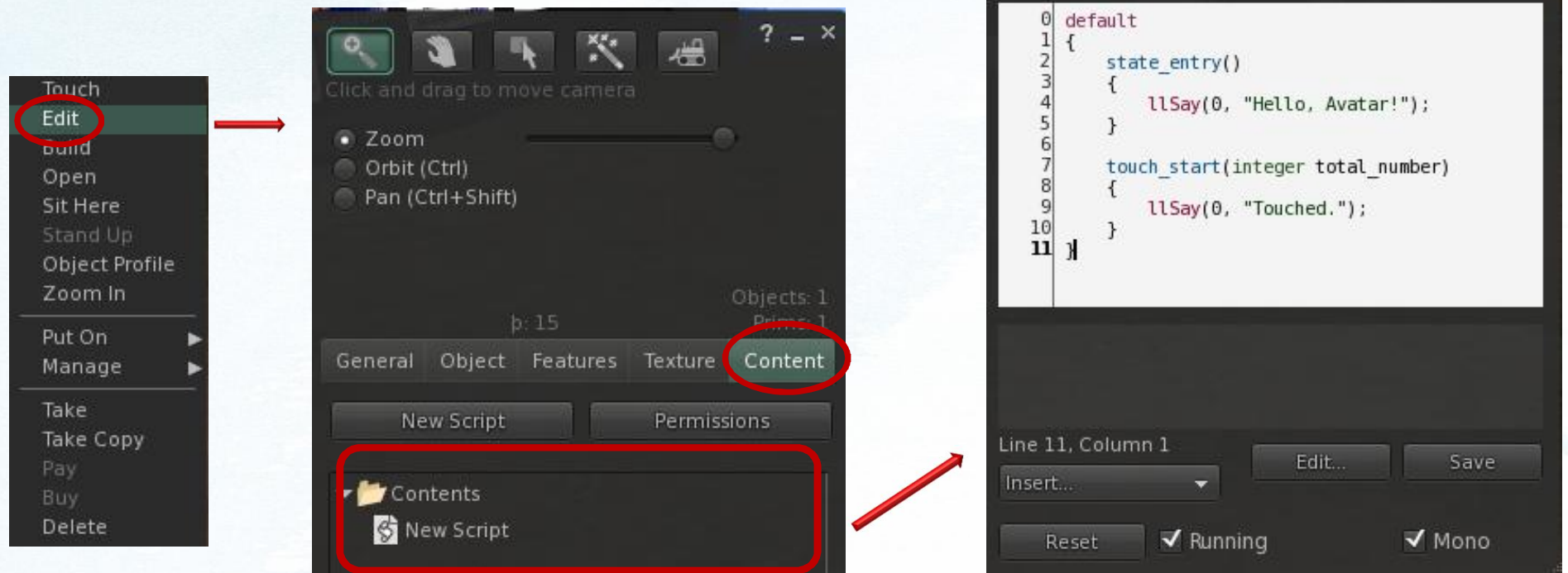
Scripting in Second Life

1. Second Life Overview
2. Creating Objects
3. Linden Scripting Language (LSL)



Behavior of Objects

- Linden Scripting Language (LSL)
- Scripts are attached to primitives
- Build-in editor
- External stand-alone editor (e.g. LSLEditor)

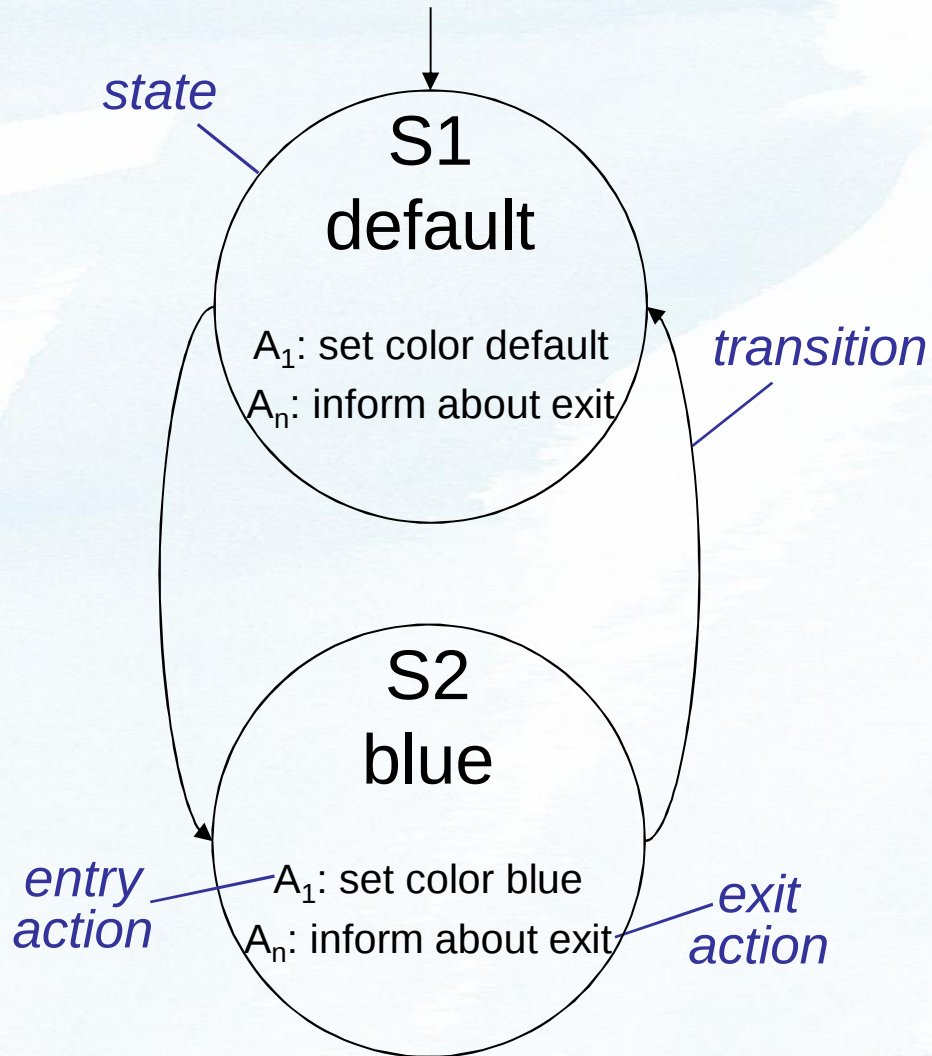


Linden Scripting Language

- Like C or Java
- About 35 ready made *event handlers*
- Hundreds of built-in *functions*
- LSL Portal:
http://wiki.secondlife.com/wiki/LSL_Portal



Finite State Machines



- A model of behavior composed of a finite number of *states*, *transitions* between those states, and *actions*



Example: *Defining States*

```
default {  
    //contents of the default state  
    //goes here  
}  
  
state blue {  
    //contents of state blue goes here  
}
```



States in LSL

- A state is a set of event handlers, which are running and waiting for events
- All scripts must have a state named *default*, the state SL puts the script in only when it's
 - first compiled and
 - whenever it's reset
- States cannot have user functions or variables inside their immediate scope, only event definitions



Several States

- A script may have several states
 - The *default* state must be defined before all others
 - Only one state per script can be active at any one time
- The states are defined with the word *state* (except for the default state) followed by the *name* of the state and the contents is enclosed in curly brackets



Changing States

- Command *state* followed by the name of the target state:
state target_state;
 1. Trigger *state_exit* and clear the event queue
 2. Change state to target state
 3. Trigger *state_entry* in the *target* state



```
vector color_default = <1.0, 1.0, 1.0>; // white
vector color_blue = <0.0, 0.0, 1.0>; // blue

default {
    state_entry() {
        llSetColor(color_default, ALL_SIDES);
    }
    touch_start(integer total_number) {
        state blue; // changes the state to state blue
    }
    state_exit() {
        llOwnerSay("state_exit() for default state");
    }
}

state blue {
    state_entry(){
        llSetColor(color_blue, ALL_SIDES);
    }
    touch_start(integer total_number){
        state default; // changes the state to default state
    }
    state_exit() {
        llOwnerSay("state_exit() for blue state");
    }
}
```

Events Trigger Event Handlers



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

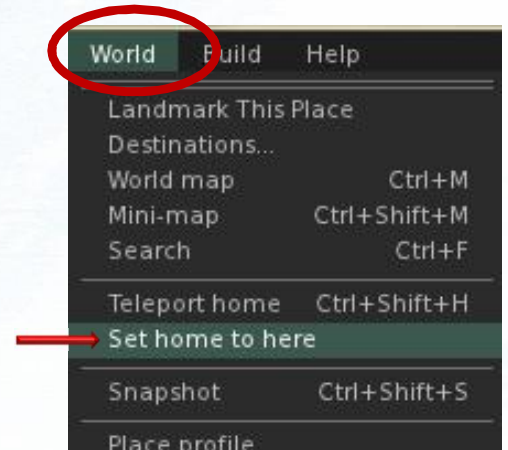
    touch_start(integer total_number) {
        llSay(0, "Please, don't touch");
    }
}
```

 Snowman Please, don't touch



Exercise 1: The First Script

1. Go to a land named IP2012 and set it to be your home
2. Create a primitive and add your first script to it:



Make the primitive to respond to a touch so that

1. the first touch will start a timer
(**llSetTimerEvent(float sec)** and **timer()**) showing the avatar every second an increasing number starting from 1, and
2. the second touch will stop the timer

LSL Portal: http://wiki.secondlife.com/wiki/LSL_Portal

Timer: <http://wiki.secondlife.com/wiki/Timer>

Data Types

- LSL supports basic data types (integer, float and string) and the following four:
 1. key,
 2. list,
 3. vector, and
 4. rotation



Constants

- Hundreds of predefined constants: PI, TRUE, ZERO_VECTOR, NULL_KEY, DEG_TO_RAD, etc.
- Not possible to define own constants
 - Variables can be used as pseudo-constants



Key

- A unique identifier that all the items in SL have
- Can be used to reference objects and agents

```
//Returns the key of the owner of the object
```

```
key owner = llGetOwner();
```

```
//Returns the key of the prim the script is attached to
```

```
key me = llGetKey();
```

```
//Returns the key of the detected avatar or object
```

```
//works within Detection events e.g. touch_start() and
```

```
//in functions called by Detection events
```

```
key touchedBy = llDetectedKey(0);
```

List

- Ordered set of values of *other data types* (two dimensional lists *NOT* possible)
- Created via comma-separated values enclosed by square brackets ([])
- The values in a list *can't be changed*

```
list lst = []; //Empty list
lst = ["apple", 7];
lst += [6.9, 11]; //lst is now ["apple", 7, 6.9, 11]
integer length = llGetListLength(lst); //4
string s = llList2String(lst, 0); //s is "apple"
lst = llList2List(lst, 1, 2); //[7, 6.9]
integer position = llListFindList(lst, [6.9]); //1
```

Vector

- Single unit of three floats: $\langle x, y, z \rangle$
- A vector can represent e.g. position and color

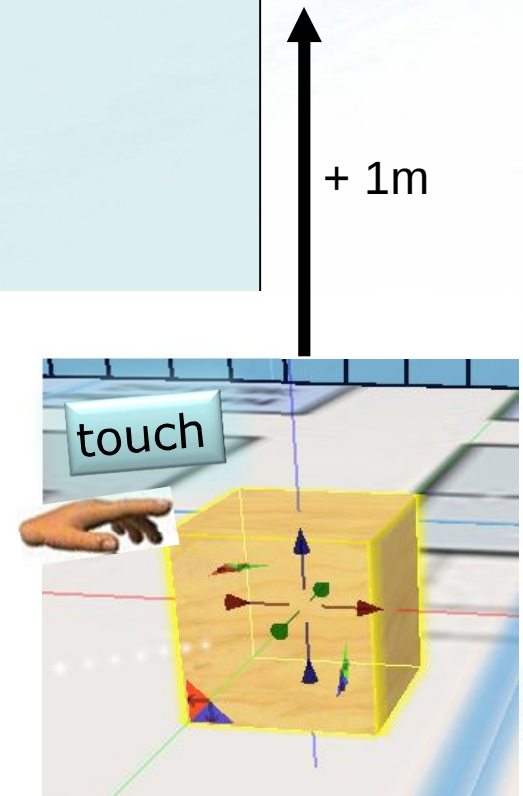
```
vector v = ZERO_VECTOR;  
//Constant ZERO_VECTOR has the value <0.0, 0.0, 0.0>  
v = llGetPos();  
//Gets the object's position in the sim  
v.x = 45.6;  
v.y = v.x + 7.0;  
//v is now <45.6, 52.6, 0.0>  
vector red = <1.0, 0.0, 0.0>;  
//The components represent red, green, and blue (rgb)
```



Setting a New Position

```
//Move the object up 1m when someone touches it  
//Stay there for 2s and return to the origin
```

```
vector startPosition;  
default {  
    touch_start(integer total_number) {  
        startPosition = llGetPos();  
        llSetPos(llGetPos() + <0,0,1>);  
        llSleep(2.0);  
        llSetPos(startPosition);  
    }  
}
```



Rotation

- Single unit of four floats: $\langle x, y, z, s \rangle$
 - Representation of the mathematical concept *quaternion*
- Represents the orientation of an object

```
rotation r = ZERO_ROTATION;  
//Constant ZERO_ROTATION: <0.0, 0.0, 0.0, 1.0>  
r = llGetRot();  
//Gets the object's rotation  
r = llEuler2Rot(<0, 45 * DEG_TO_RAD, 0>);  
//45 degrees around the y-axis
```



Rotation of an Object in Practice

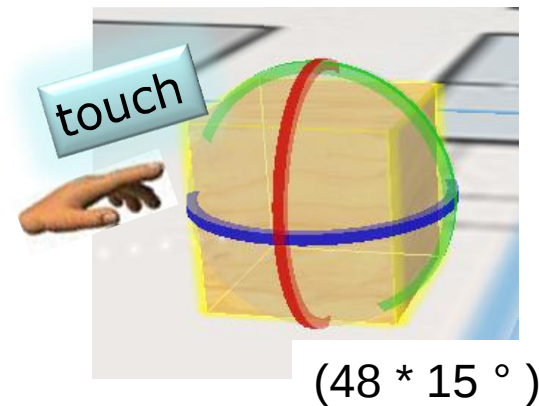
1. Define the angle of rotation in degrees (vector)
2. Change it to radians
3. Convert the vector to rotation
4. Set the new orientation (rotation) value for the object

```
vector angleInDegrees = <45, 0, 0>;  
vector angleInRadians = angleInDegrees * DEG_TO_RAD;  
rotation rot_x45 = llEuler2Rot(angleInRadians);  
llSetRot(llGetRot() * rot_x45);  
  
////////// OR //////////  
  
llSetRot(llGetRot()*llEuler2Rot(DEG_TO_RAD*<45, 0, 0>));
```

Example: *Rotating an Object*

```
//Rotate the object around z-axis when someone touches it  
//Rotate twice 360 degrees in segments of 15 degrees
```

```
rotation rot_z15;  
default {  
    state_entry() {  
        vector angleInDegrees = <0, 0, 15>;  
        vector angleInRadians = angleInDegrees * DEG_TO_RAD;  
        rot_z15 = llEuler2Rot(angleInRadians);  
    }  
    touch_start(integer total_number){  
        integer i;  
        for(i=1; i<49; i++) {  
            llSetRot(llGetRot() * rot_z15);  
        }  
        llSay(0, "Rotation stopped");  
    }  
}
```



Creating Opening Doors

The prims in SL are rotated around their center. The doors should turn around their side. To achieve the desired result the door may be done

- of two linked prims (the actual door and "hinges" on the side, which holds the rotation script) or
- one can use the path cut property to cut away half of the prim.



Exercise 2: Door

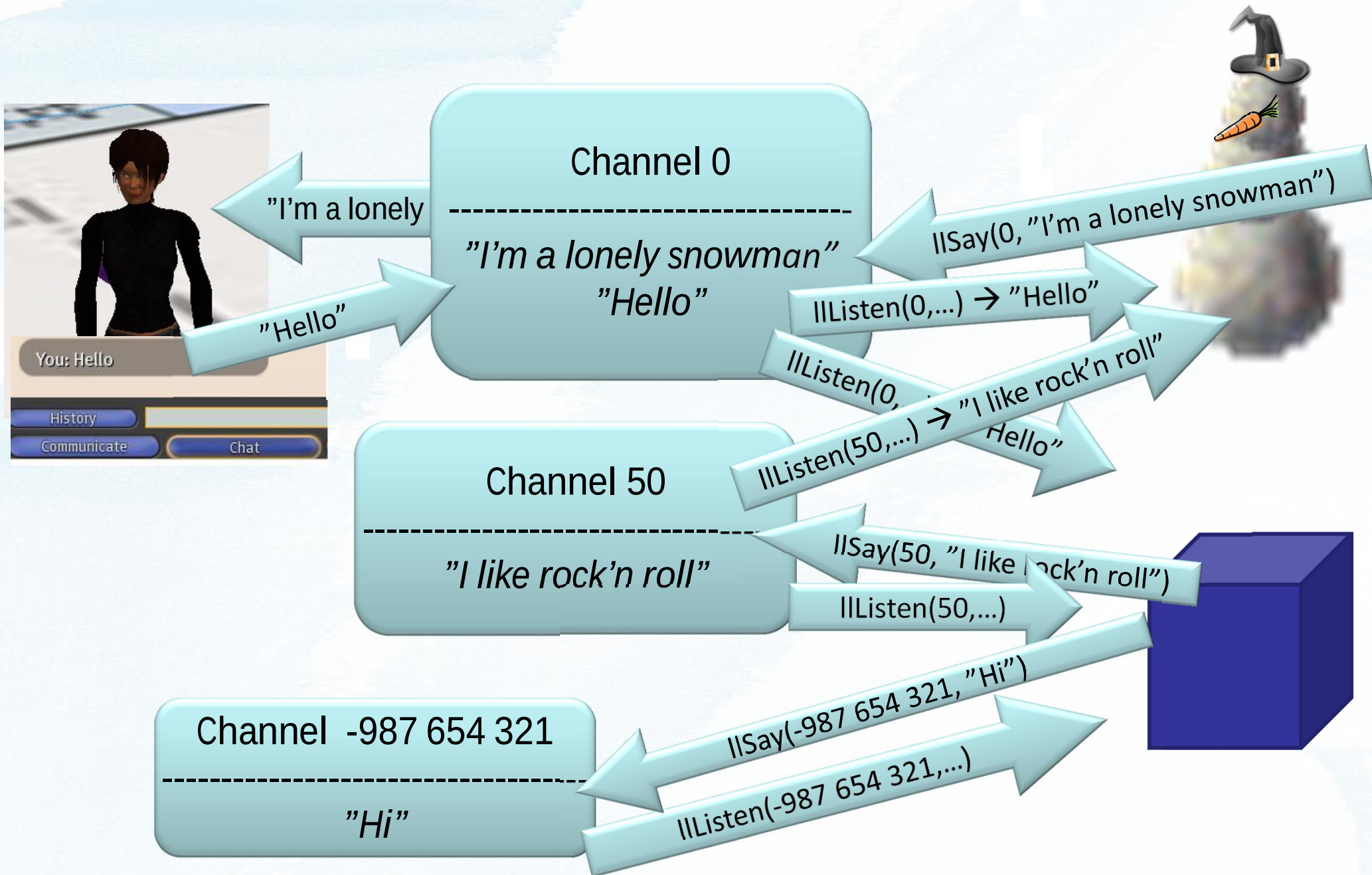


Create a wall and a door to it

- The door shall open and close when it's touched
- Use states for different positions of the door
- If the door's not closed with the touch event for five seconds, it shall close automatically
 - In this case the door informs the surrounding world that it'll be closing
 - Use *//GetTime()* function to get the time since the script was started or reset



Communicating



Talking to Channels

A channel may be any valid integer from -2,147,483,648 through 2,147,483,647

- **llWhisper**(channel, msg) – 10 meters
- **llSay**(channel, msg) – 20 meters
- **llShout**(channel, msg) – 100 meters
- **llRegionSay**(channel, msg) – current sim



Listening to Channels

Enable listening

- **llListen(channel, name, id, msg)**
 - integer channel: channel to listen to
 - string name: filter for prim/avatar name
 - key id: filter for prim/avatar UUID
 - string msg: filter for specific chat message

Event handler

- **listen(channel, name, id, msg)**



Example: *Talking Cube*



2

*Hello Tor Harbour
You said Hello to
you, little cube
Hello to you too*



1

*Hello to you,
little cube*



LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

Example: *Talking Cube*



```
default {  
    state_entry() {  
        llListen(0, "", "", "");  
    }  
  
    listen(integer ch, string name, key id, string msg) {  
        integer i;  
        llSay(0, "Hello " + name + " You said " + msg);  
        i = llSubStringIndex(msg, "Hello");  
        //returns the index of the first pattern in source  
        //returns -1, if the pattern is not found  
        if (i >= 0){  
            llSay(0, "Hello to you too");  
        }  
    }  
}
```

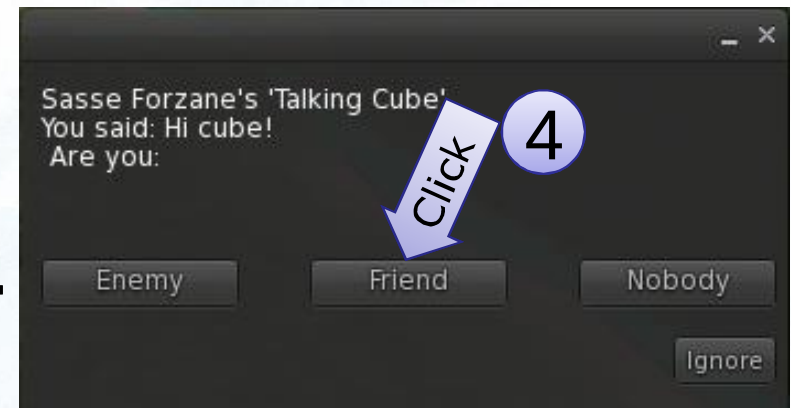
Dialog Box



*Instead of typing, just
click one option*

Channel x

Friend



Owner



Channel 0

① Tor Harbour: Hi cube!
⑤ Talking Cube:
Great to have friends!

*Talking Cube:
Tor Harbour said: Hi cube!*



Example: *Dialog Box...*



```
integer channel;
default {
  state_entry() {
    //Create random channel within range [500,1000]
    channel = 500 + (integer)(llFrand(501.0));
    llListen(0, "", "", "");
  }
  listen(integer ch, string name, key id, string msg) {
    llOwnerSay(name + " said: " + msg);
    //says msg to the owner only
    list options = ["Enemy", "Friend", "Nobody"];
    llDialog(id, "You said " + msg, options, channel);
    state dialog;
  }
}
```

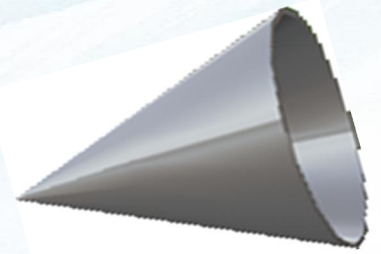
...example continues



```
state dialog {
  state_entry() {
    llListen(channel, "", "", "");
  }
  listen(integer ch, string name, key id, string msg) {
    {
      if (msg == "Friend")
        llSay(0, "Great to have friends!");
      state default;
    }
  }
}
```



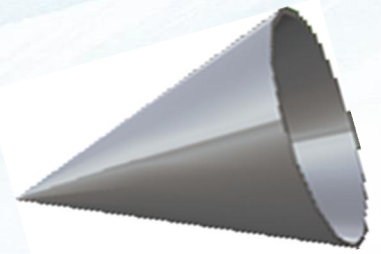
Using Sensors



- Sensors are used to detect objects or avatars
- The shape of a sensor is a cone
- A script can have only one sensor at a time
- Creating sensors
 - For a single scan: function **l1Sensor**
 - For repetitive scan: function **l1SensorRepeat**
 - Needs to be removed with the function **l1SensorRemove**
- Event handlers
 - Wanted found: **sensor**
 - Wanted not found: **no_sensor**



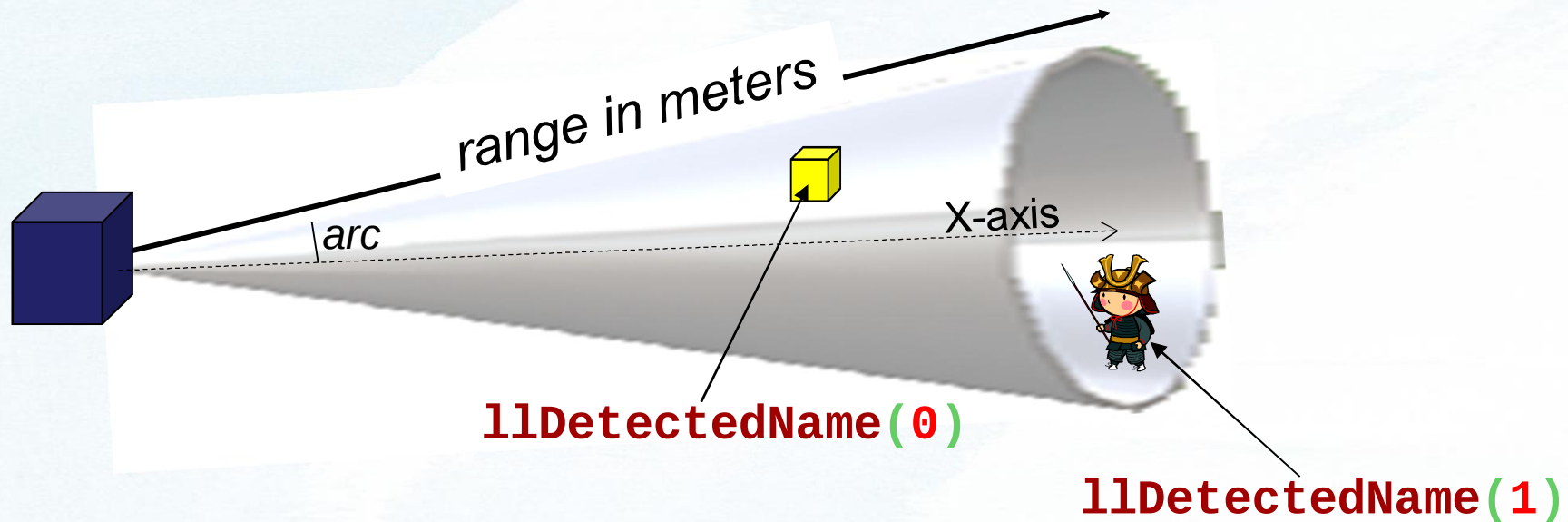
Parameters for a Sensor



- **name**: Avatar or object name
- **id**: Avatar or object ID
 - NULL_KEY is used for searching any key
- **type**: What to look for
 - AGENT: avatars
 - ACTIVE: objects with active script or moving physical objects
 - PASSIVE: objects which are not active
 - SCRIPTED: objects with active script
- **range**: Distance how far to sensor (max: 96.0 m)
- **arc**: The angel (around the x-axis), in radians, to search (range: 0.0 to PI)
- **rate**: In seconds, how often to repeat the search



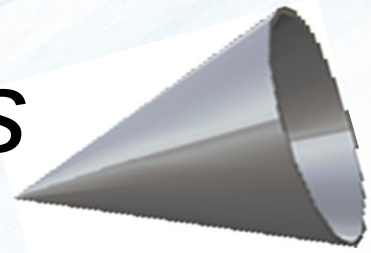
```
llSensor("Something", NULL_KEY, PASSIVE, 10.0, PI);  
//Looking for things named Something that are passive  
//and within 10 meters and pi radians  
  
llSensorRepeat("", llGetOwner(), AGENT, 5.0, PI_BY_TWO, 2.0);  
//Looking for the owner within 5 meters and pi/2 radians  
//every two seconds
```



Note

The angle is arc radians around the forward vector

Example: *Sensing Avatars*



```
default {
  state_entry() {
    llSensorRepeat("", NULL_KEY, AGENT, 3.0, PI, 8.0);
    //Range in meters, arc in radians,
    //repeat time in seconds
  }
  sensor(integer num_detected) {
    integer i;
    for (i = 0; i < num_detected; i++) {
      llOwnerSay("Sensed avatar " + llDetectedName(i) +
        " at " + (string)llDetectedPos(i));
    }
  }
  no_sensor() {
    llOwnerSay("No avatars in range.");
  }
}
```

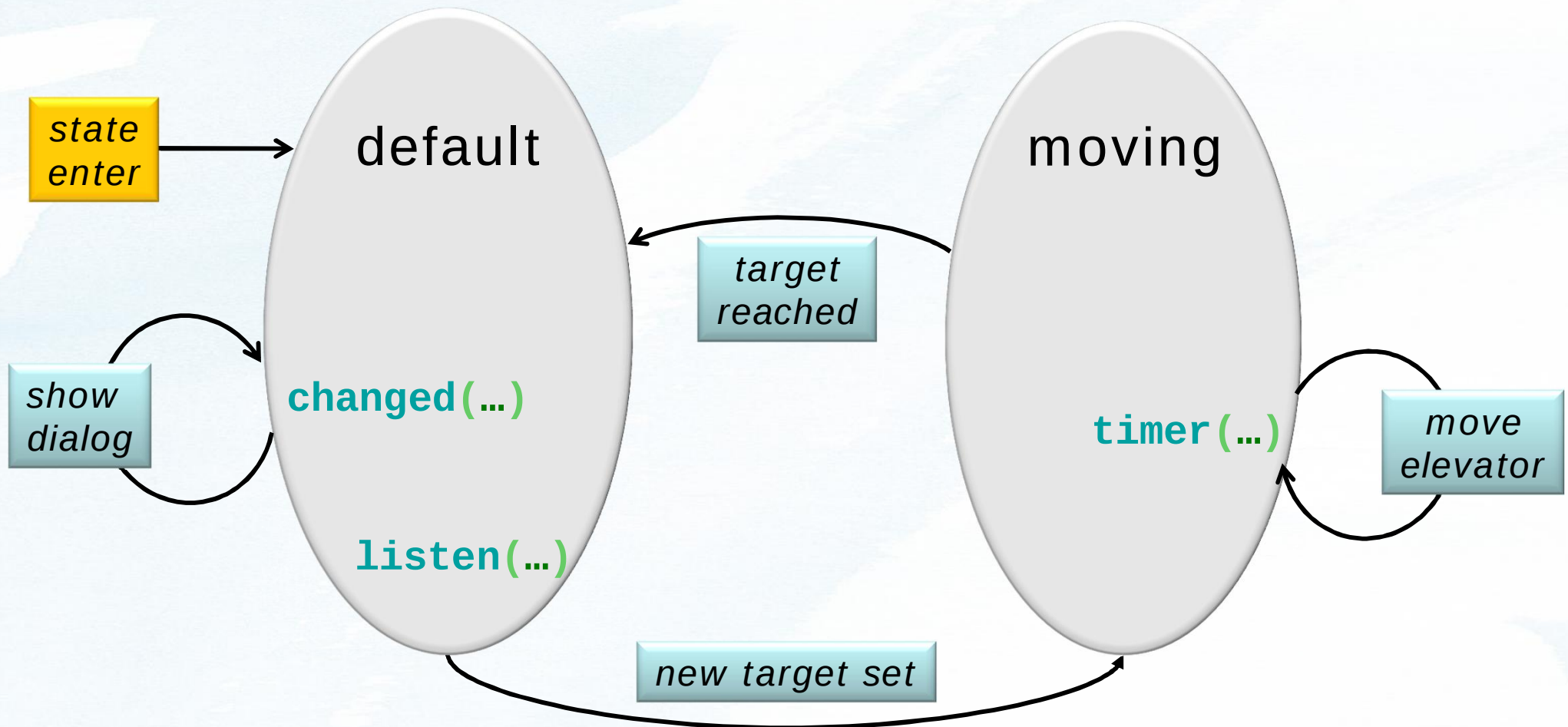
Moving Avatars

When an avatar sits on a prim, it moves along with the prim.





Elevator





Some Notices on the Elevator



- When an avatar sits on an object, the **changed(...)** event handler is triggered
- To get the key of the avatar sitting on the object, use the function **llAvatarOnSitTarget(...)**
- The sit location for a prim is set with the function **llSitTarget(...)**
- The "Sit Here" text in the right-click menu may be changed with **llSetSitText(...)**

Thank you for your attention!



FL: Sariseelia Sore

SL: Sasse Forzane

email: sariseelia.sore@lamk.fi



LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

Exercise 3: Moving Chair

1. Create a chair moving up and down according to the choice of the avatar sitting on it
2. The chair should have a floating text above it telling that it's a moving chair (*//SetText()*)
3. When an avatar comes close enough your chair, he/she is shown a dialog box asking whether he/she wants to sit on your moving chair
4. When the avatar is sitting on the chair she/he is asked if he/she wants to move up
5. When the avatar stands up from the chair the chair will return back to its original place
6. If the owner comes close to the hot-air balloon, it'll tell him/her who's sitting on it