实验课 3：智能 Agent 感知-推理-行为实验

一、实验大纲

1. LSL 感知函数事件处理实习
   - 掌握听说脚本方法
   - 掌握 Sensor 感知事件处理机制
   - 掌握控制物体运动和外观行为脚本方法
2. 智能 Agent 感知-推理-行为实验
   - 智能 Agent 理论
   - LSL 智能 Agent 脚本框架
   - 实验实现基于规则的反射性 Agent

二、LSL 感知函数事件处理实习

2.1 LSL 脚本语言听/说操作方法

听说函数使用：

llWhisper
llSay
llShout
llListen
llListenControl
llListenRemove

llWhisper

llWhisper(integer channel, string text);
Whisper text on channel. Channel 0 is the public chat channel that all avatars see as chat text. Channels 1 – 2,147,483,648 are private channels that aren't sent to avatars but other scripts can listen for.

llSay

llSay(integer channel, string text);
Say text on channel. Channel 0 is the public chat channel that all avatars see as chat text. Channels 1 – 2,147,483,648 are private channels that aren't sent to avatars but other scripts can listen for.

llShout

llShout(integer channel, string text);
Shout text on channel. Channel 0 is the public chat channel that all avatars see as chat text. Channels 1 – 2,147,483,648 are private channels that aren't sent to avatars but other scripts can listen for.

llListen

integer llListen(integer channel, string name, key id, string msg);
Sets a listen event callback for msg on channel from name (name, id and/or msg can be empty) and returns an identifier that can be used to deactivate or remove the listen. Channel 0 is the public chat channel that all avatars see as chat text. Channels 1 – 2,147,483,648 are private channels that aren't sent to avatars but other scripts can listen for.

llListenControl

llListenControl(integer number, integer active);
Makes a listen event callback active or inactive.

llListenRemove

llListenRemove(integer number);
Removes a listen event callback.

示例脚本文件：

《0701_Using_The_Chat_Functions.txt》
《0702_Using_llOwnerSay.txt》
《0703_Listening_To_All_Messages_on_a_Channel.txt》
《0704_Listening_For_A_Specific_Message_From_A_Specific_Key.txt》
《0705_Removing_A_Listener.txt》
《0706_Turning_A_Listener_On_And_Off.txt》

2.2 LSL 脚本语言 Sensor 处理机制

LSL 脚本提供了一个 Sensor 事件处理机制，允许物体通过脚本感知本身状态、感知周围其他的物体或脚本、感知到周围的化身 Avatars、以及周围土地的一些属性。llSensor

方法主要用于感知周围其他对象，通过一系列 llDetected… functions 获取感知对象的属性。

llSensor

llSensor(string name, key id, integer type, float range, float arc);
Performs a single scan for name and id with type (AGENT, ACTIVE, PASSIVE, and/or SCRIPTED) within range meters and arc radians of forward vector (name and/or id can be empty or 0). A range of 0.0 does perform a scan.

llSensorRepeat

llSensorRepeat(string name, key id, integer type, float range, float arc, float rate);
Sets a callback for name and id with type (AGENT, ACTIVE, PASSIVE, and/or SCRIPTED) within range meters and arc radians of forward vector (name and/or id can be empty or 0) and repeats every rate seconds. Passing a range of 0.0 cancels the sensor.

llSensorRemove

llSensorRemove();
Removes the sensor.

llDetectedName

string llDetectedName(integer number);
Returns the name of detected object number (returns empty string if number is not valid sensed object).

llDetectedKey

key llDetectedKey(integer number);
Returns the key of detected object number (returns invalid key if number is not valid sensed object).

llDetectedOwner

key llDetectedOwner(integer number);
Returns the key of detected object's owner number (returns invalid key if number is not valid sensed object).
integer llDetectedType(integer number);
Returns the type (AGENT, ACTIVE, PASSIVE, SCRIPTED) of detected object (returns 0 if number is not valid sensed object). Note that the return value is a bit field, so comparisons need to be an AND check, ie:

```
integer type = llDetectedType(0);
if (type & AGENT)
{
        // do stuff
}
```

llDetectedPos

vector llDetectedPos(integer number);
Returns the position of detected object number (returns <0,0,0> if number is not valid sensed object).

llDetectedVel

vector llDetectedVel(integer number);
Returns the velocity of detected object number (returns <0,0,0> if number is not valid sensed object).

llDetectedGrab

vector llDetectedGrab(integer number);
Returns the grab offset of detected object number (returns <0,0,0> if number is not valid sensed object).

llDetectedRot

rotation llDetectedRot(integer number);

Returns the rotation of detected object number (returns <0,0,0,1> if number is not valid sensed object).

llDetectedLinkNumber

integer llDetectedLinkNumber(integer number);
Returns the link position (0 for a non-linked object, 1 for the root of a linked object, 2 for the first child, &c) of the triggered event for touches.

sensor(integer total_number)

This event is raised whenever objects matching the constraints of the llSensor command are detected. The number of detected objects is passed to the script. Information on those objects may be gathered via the llDetected* library functions.

no_sensor()

This event is raised when sensors are active (via the llSensor library call) but are not sensing anything.

示例脚本文件：

《0718_Visualizing_Sensors.txt》
《0719_Using_Sensors_To_Find_Owner.txt》
《mat_script.txt》

2.3 LSL 脚本控制物体运动和行为方法

```
setupPrim()
{
    vector dimple = <0.0, senseArc / PI, 0.0>;
    llSetPrimitiveParams([
        PRIM_TYPE,
            PRIM_TYPE_SPHERE,     // type
                0,                          // hole shape
                <0.0, 1.0, 0.0>,      // cut
                0,                          // hollow
                ZERO_VECTOR,            // twist
                dimple,                   // dimple
        PRIM_SIZE,
            2 * <senseRange, senseRange, senseRange>,
        PRIM_PHANTOM,
            TRUE,
        PRIM_COLOR,
            ALL_SIDES,
            <1.0, 1.0, 1.0>,
            0.5
```

]);

示例脚本文件：

《chair_script.txt》

llGet 和 llSet 函数
llSetStatus
llGetStatus
llSetScale
llGetScale
llSetColor
llGetColor
llSetAlpha
llGetAlpha
llSetTexture
llGetTexture
llScaleTexture
llOffsetTexture
llRotateTexture
llSetTextureAnim
llSetPos
llGetPos
llGetLocalPos
llSetRot
llGetRot
llGetLocalRot
llSetText
llSetForce
llGetForce

llSetStatus

llSetStatus(integer status, integer value);
Sets status to value.

llGetStatus

integer llGetStatus(integer status);
Gets value of status.

llSetScale

llSetScale(vector scale);
Sets the scale.

llGetScale

vector llGetScale(key name);
Gets the scale.

llSetColor

llSetColor(vector color, integer face);
Sets the color. If face == ALL_SIDES, set the color to all faces.

llGetColor

vector llGetColor(integer face);
Gets the color.

llSetAlpha

llSetAlpha(float alpha, integer face);
Sets the alpha. If face == ALL_SIDES, set the alpha to all faces.

llGetAlpha

float llGetAlpha(integer face);
Gets the alpha.

llSetTexture

llSetTexture(string texture, integer face);
Sets the texture from object inventory of face. If face == ALL_SIDES, set the texture to all faces.

llGetTexture

string llGetTexture(integer face);
Gets the texture of face if in object inventory.

llScaleTexture

llScaleTexture(float scale_s, float scale_t, integer face);
Sets the texture s and t scales of face. If face == ALL_SIDES, scale the texture to all faces.

llOffsetTexture

llOffsetTexture(float offset_s, float offset_t, integer face);
Sets the texture s and t offsets of face. If face == ALL_SIDES, set the texture offsets for all faces.

llRotateTexture

llRotateTexture(float rot, integer face);
Sets the texture rotation of face. If face == ALL_SIDES, rotate the texture of all faces.

llSetTextureAnim

llSetTextureAnim(integer mode, integer face, integer sizex, integer sizey, float start, float length, float rate);
Animates a texture by setting the texture scale and offset. Mode is a mask, consisting of:
ANIM_ON – texture animation is on.
LOOP – loop the texture animation.
REVERSE – play animation in reverse direction.
PING_PONG – play animation going forwards, then backwards.
SMOOTH – slide in the X direction, instead of playing separate frames.
ROTATE – Animate texture rotation instead.
SCALE – Animate the texture scale instead.
Only ONE texture animation can be active at a time. You can only do traditional animation, ROTATE, or SCALE at a time, you can't combine masks.
In the case of ROTATE or SCALE, sizex and sizey are ignored, and start and length are used as the start and length values of the animation. For rotation, start and length are in radians.
face is which face to animate. If face == ALL_SIDES, all textures on the object are animated. You can only have one texture animation on an object, calling llSetTextureAnim more than once on an object will reset it.
sizex and sizey describe the layout of the frames within the texture. sizex is how many horizontal frames, sizey is how many vertical frames.
start is the frame number to begin the animation on. Frames are numbered from left to right, top to bottom, starting at 0.
length is the number of frames to animate. 0 means to animate all frames after the start frame.
rate is the frame rate to animate at. 1.0 means 1 frame per second, 10.0 means 10 frames/second.

llSetPos

llSetPos(vector pos);
Sets the position (if the script isn't physical). If the object is a child, the position is treated as root relative and the linked set is adjusted.

llGetPos

vector llGetPos();
Gets the position.

llGetLocalPos

vector llGetLocalPos();
Gets the local position of a child object.

llSetRot

llSetRot(rotation rot);
Sets the rotation (if the script isn't physical). If the object is a child, the rotation is treated as root relative and the linked set is adjusted.

llGetRot

rotation llGetRot();
Gets the rotation.

llGetLocalRot

rotation llGetLocalRot();
Gets the local rotation of a child object.

llSetText

llSetText(string text, vector color, float alpha)
Sets text that floats above object to text, using specified color and alpha.

llSetForce

llSetForce(vector force, integer local);
Sets the force, in local coordinates if local == TRUE (if the script is physical).

llGetForce

vector llGetForce();
Gets the force (if the script is physical).

llTarget

integer llTarget(vector position, float range);
Set positions within range of position as a target and return an ID for the target.

llTargetRemove

llTargetRemove(integer tnumber);
Remove target number tnumber.

llRotTarget

integer llRotTarget(rotation rotation, float error);
Set rotations within error of rotation as a rotational target and return an ID for the target.

llRotTargetRemove

llRotTargetRemove(integer tnumber);
Remove rotational target number tnumber.

llMoveToTarget

llMoveToTarget(vector target, float tau);
Critically damp to target in tau seconds (if the script is physical). Good tau values are greater than 0.2. A tau of 0.0 stops the critical damping.

llStopMoveToTarget

llMoveToTarget();
Stops critically damped motion.

三、智能 Agent 感知-推理-行为实验

3.1 实验目的

学习智能Agent理论，实验智能Agent感知-推理-行为机制。本次实验为基础性实验，要求通过LSL脚本，操纵虚拟机器人，实现简单的智能Agent感知-推理-行为框架。

3.2 实验原理

3.2.1 智能 Agent 理论研究背景

Agent理论、技术和应用的研究近年来在人工智能、软件工程、信息系统、分布计算、人机交互等领域非常活跃，它日趋与软件工程、信息系统、面向服务计算、数据挖掘、网格计算、对等计算、自治计算和通讯、模拟仿真、自适应和自组织系统等方面的研究相融合，通过交叉和借鉴其它学科知识，为复杂、灵活和具有智能特征的计算机系统开发提供基础理论和关键技术。智能Agent技术不仅是分布智能的研究热点，而且可能成为下一代软件开发的重要突破点。事实上，对智能Agent的研究已经成为人工智能学科的核心内容，有人认为是人工智能研究的初始目标和最终目标。

智能Agent的研究可追溯到70年代分布式人工智能的研究(DAI:Distribute Artifical Intelligence),主要分两条研究路线：一条围绕经典人工智能展开,主要研究智能Agent的拟人行为,多智能Agent的协商模型等,其研究方向可分为智能Agent理论,智能Agent体系结构,智能Agent语言,多智能Agent系统等,一些计算机科学家称之为强定义的智能Agent；另一条从90年代左右到现在,以应用为主,将经典人工智能关于智能Agent的强定义弱化,拓宽了智能Agent的应用范围,新的研究方向主要包括智能Agent人机界面,基于智能Agent的软件工程(AOSE)等。

3.2.2 智能 Agent

智能Agent，也叫智能Agent，智能Agent。在计算机科学和人工智能领域中， 智能Agent可以看作是一个自动执行的实体，它通过传感器感知环境，通过效应器作用于环境[160]。若智能体是人，则传感器有眼睛、耳朵和其它器官，手、腿、嘴和身体的其它部分是效应器。若智能体是机器人，摄像机等是传感器，各种运动部件是效应器。一般智能Agent可以用下图表示。

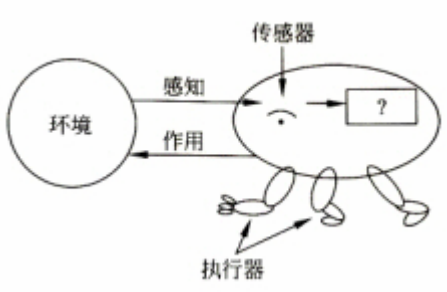


图 智能 Agent 与环境交互作用

智能Agent可根据智能的强弱定义性进行分类。弱定义包括自主性(Autonomy)，社会性(Social ability)，反应性(Reactivity)等人类特有的性质。强定义在弱定义的基础上加入

知识(Knowledge)、信念(Belief)、意图(Intention)、责任(Obligation)等精神概念，有研究者称之情感(Emotional) Agent。其它很多词汇也常被研究者用来描述Agent，如移动性(Mobility)，诚实(Veracity)，善良(Benevolence)，理性(Rationality)，长寿(Longevity)，前瞻性(Pro-active, Goal-directed) 等。

通常认为一个智能Agent需要具有下述特性：

(1) 自治性。这是一个智能Agent的基本特性，即可以控制它自身的行为。智能Agent的自治性体现在：智能Agent的行为应该是主动的、自发的；智能Agent应该有它自己的目标或意图（intention）；根据目标、环境等的要求，智能Agent应该对自己的短期行为作出计划。

(2) 交互性，即对环境的感知和影响。无论智能Agent生存在现实的世界中（如机器人、Internet上的服务智能Agent等）还是虚拟的世界中（如虚拟商场中的智能Agent等），它们都应该可以感知它们所处的环境，并通过行为改变环境。一个不能对环境作出影响的物体不能被称为智能Agent。

(3) 协作性。通常智能Agent不是单独地存在，而是生存在一个有很多个智能Agent的世界中。智能Agent之间的良好有效的协作可以大大提高整个多智能Agent系统的性能。

(4) 可通信性。这也是一个智能Agent的基本特性。所谓通信，指智能Agent之间可以进行信息交换。更进一步，智能Agent应该可以和人进行一定意义下的“会话”。任务的承接、多智能Agent的协作、协商等都以通信为基础。

(5) 长寿性（或时间连贯性）。传统程序由用户在需要时激活，不需要时或者运算结束后停止。智能Agent与之不同，它应该至少在“相当长”的时间内连续地运行。这虽然不是智能Agent的必须特性，但目前一般认为它是智能Agent的重要性质。

另外，有些学者还提出智能Agent应该具有自适应性、个性等特性。在实际的应用中，智能Agent经常需要在时间和资源受到一定限制的情况下作出行动。所以，对于现实世界中的智能Agent，除了应该具有智能Agent的一般性质外，还应该具有实时性。

智能Agent的理论主要关于智能Agent的认知模型和形式化描述。智能Agent的体系结构讨论应由哪些模块组成，它们之间如何交互信息，智能Agent感知到的一些信息如何影响它的行为和内部状态，如何将智能Agent的这些模块用软件或硬件的方式组合起来形成一个有机的整体。根据人类思维的不同层次，可以把Agent分为：

（1）反应式Agent

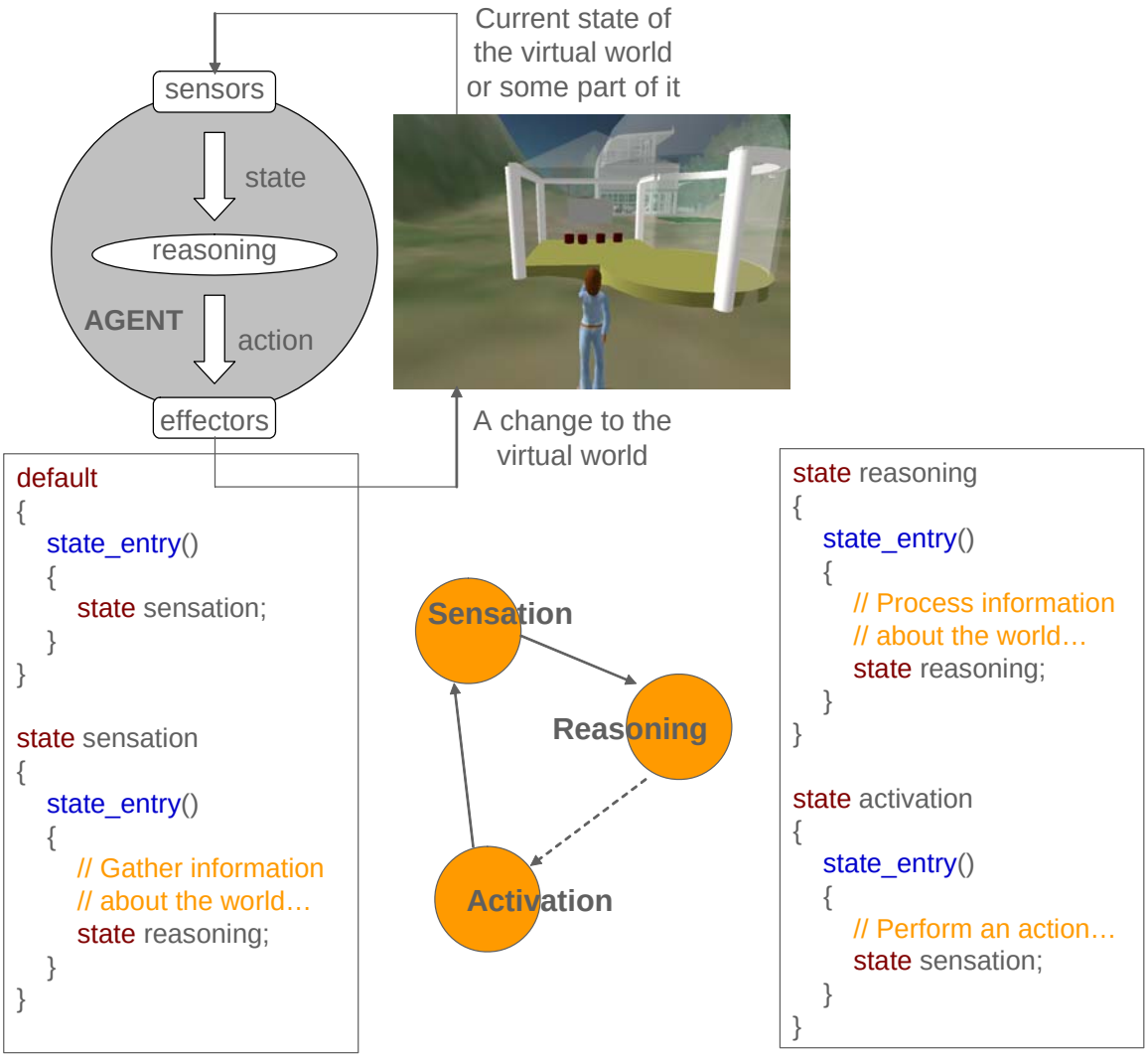
反应式（reflex或reactive）Agent只简单的对外部刺激产生响应，没有内部状态。

（2）慎思式Agent

慎思式（deliberative）Agent又称为认知式（cognitive）Agent，是一个具有现实符号模型的基于知识的系统。

慎思式智能Agent（Deliberative agent），也称作认知智能Agent（Cognitive agent），是一个显式的符号模型，包括环境和智能行为的逻辑推理能力。它保持了经典人工智能的传统，是一种基于知识的系统（Knowledge-based system)。环境模型一般是预先实现的，形成主要部件知识库。反应式智能Agent（Reactive agent）是不包含用符号表示的世界模型，并且不使用复杂的符号推理的智能Agent。目前，多智能Agent系统的研究非常活跃。多智能Agent系统试图用智能Agent来模拟人的理性行为，主要应用在对现实世界和社会的模拟、机器人和智能机械等领域。而在现实世界中生存、工作的智能Agent，要面对的是一个不断变化的环境。在这样的环境中，智能Agent不仅要保持对紧急情况的及时反应，还要使用一定的策略对中短期的行为作出规划，进而通过对世界和其它智能Agent的建模分析来预测未来的状态，以及通过通讯语言实现和其他智能Agent的协作或协商。

3.2.3 智能 Agent 感知-推理-行为框架



Agent framework.txt

```
//========================================
// Description: A simple agent framework
//
// Author: Kathryn Merrick
//========================================
default
{
	state_entry()
	{
		state sensation;
	}
}
state sensation
{
	state_entry()
	{
		// Gather information about the world
		// Store information in global variables
		llOwnerSay("DEGUG: Starting sensation process.");
		state reasoning;
	}
}
```

```
state reasoning
{
    state_entry()
    {
        // Process information stored in global variables
        llOwnerSay("DEGUG: Starting reasoning process.");
        state activation;
    }
}
state activation
{
    state_entry()
    {
        // Perform an action in the world
        llOwnerSay("DEGUG: Starting activation process.");
        state sensation;
    }
}
```

3.3 实验

实验 1：实现智能设备，根据环境状态自动切换状态。

实验步骤：

- 创建智能设备的三维形状和纹理；
- 用 llSensor 等函数设计感知对象和事件；
- 根据感知对象和时间的属性改变自身的形状/颜色等属性特征。

示例脚本文件：

《table_script.txt》：根据碰撞检测事件，自动切换情感状态；

《light_switch_script.txt》+《light_script.txt》：两个物体通过脚本对话；

实验 2：实现基于规则的简单反射性 Agent

实验步骤：

- 创建 Agent 的三维物体 Prim；
- 在感知 Sensing 状态里添加一个对象 Sensor，将 Sensor 的对象放在一个列表里 List。（Use llSensor() and the sensor() event.）
- 在推理 Reasoning 状态里，设计规则，让物体在感知对象集里，识别一类对象作为目标，并向它们移动；同时尽量远离不是目标对象的物体。
- 在行为 Action 状态里，执行推理规则设定的行动计划，（使用 llMoveToTarget()函数）。
- 复制拷贝生成多个物体对象，检查它们如何根据规则运动的。（这也是一个 agent 程序可用作多个角色对象的例子）。

实验进阶：（可选）

在实验2的基础上，修改规则部分，增加一些模糊逻辑规则，支持“远近”、“多少”等模糊规则。