《基于主体建模》
实验教学大纲

课程名称：基于主体建模

英文名称：Agent Based Modeling

课程代码：

课程性质：必修

实验指导书名称：基于主体建模实验教程

适用专业：系统科学、管理科学与工程

一.实验总学时（课外学时/课内学时）：40　　总学分：2

必开实验个数：10　　选开实验个数：10

二.实验的地位、作用和目的

“基于主体建模”是系统科学专业和管理科学与工程专业硕士研究生的必修课《计算机建模与工具》的三大板块之一，是研究分散系统运行机制的一类建模方法。

“基于主体建模”实验所用的主要工具是美国麻省理工大学多媒体实验室开发的 Starlogo。Starlogo 是一个可编程的建模环境，用来探索分散系统的工作原理。分散系统是这样一类系统，它们不需要组织者而能自我组织，不需要协调者而能自我协调。

通过对本实验课程的学习，学生可以进一步加深对计算机建模理论和方法的理解，熟练掌握 Starlogo 这一重要的建模工具，能够运用基于主体的建模方法分析和解决现实问题。在对复杂分散系统研究建模的过程中，培养学生分析问题、提炼模型、解决问题的能力，训练学生的抽象思维，加强学生对计算机建模工具的掌握，为进一步从事科学研究打下坚实的基础。

三.基本原理及课程简介

计算机建模的基本思想是对现实世界进行抽象，剔除与主题无关的次要因素，抓住主要因素，提炼规则，用计算机来模拟主体在现实世界中依据一定规则运动的情况，观察结果，总结规律。

基于主体建模是计算机建模的一个重要组成部分，它所研究的分散系统是一类没有组织者和协调者，而系统整体却呈现出有组织的协调的形态的系统。在现实世界中存在很多的分散系统，例如：鸟群、蚁群、交通运输以及市场经济。之所以说它们是分散系统是因为在这些系统中，不存在集中的控制，每一个个体都是按照一定的规则在运转，而整体却呈现出有序模式。随着人们对分散系统的认知程度的逐渐加深，越来越多的研究者选用分散模式来创立组织机构、构造科学技术、甚至是架构关于世界的理论基础。

实验所用的建模工具 Starlogo 采用基于主体的建模方法。“主体”是一只只的海龟（turtles），我们可以并行地控制数千只海龟，同时也可以为它们制定不同的行为模式。而这些主体所处的“环境”则是用点（patch）来表现的，数千个点拼成一块大的背景（canvas），代表着主体所处的大环境系统。

StarLogo 允许对海龟和点进行编程，这使得海龟和海龟所处的环境都具有了自己的变化方式。海龟和点之间是可以彼此交互的，例如：我们可以编程让海龟在它的世界里到处“闻”，根据在它所在的点上闻到的“气味”来决定它的行为。海龟和海龟之间也可以交互，例如：让海龟们朝着一个方向前进，如果前面的海龟离自己很近，就走慢一点，以免撞到，否则，就走快一点追上。在这里，海龟和点之间的交互作用体现了主体与环境的关系，海龟与海龟之间的相互影响则体现了主体之间的关系。

四.实验基本要求

1、能够从计算机可以接受的概念模式入手，把实际问题转化为计算机可以识别的形式。

2、了解计算机建模的基本原理，掌握基于主体建模的研究对象、研究特点及建模方法。

3、熟悉 Starlogo 建模环境，掌握 Starlogo 建模语言，能够运用 Starlogo 对实际问题建模。

4、在基础性实验中，能够完成对 Starlogo 某一功能特性的实

验要求，掌握基本问题的建模思路、建模方法和建模过程。

5、在综合性实验中，能够独立完成对一个复杂问题的系统分析、建模过程及编码实现，对模拟过程中出现的问题能够独立排除。

6、能够根据模拟结果，对所提出的问题进行初步分析。

7、撰写简明扼要、图表清晰，结论正确、分析科学的实验报告。

五．实验具体要求

基于主体建模实验采用三段式教学模式，教学过程由理论知识准备、基础性实验及综合性实验三个环节组成。

（一）理论知识讲座

通过教师讲解，使学生了解基于主体建模的基本原理、建模方法和建模过程，初步掌握实验建模工具 Starlogo 的界面操作、变量设定、流程控制等基本特性。了解本实验课程的性质、任务、要求、课程进度安排、实验考核内容及实验报告要求等，为进行实验做好准备。理论知识准备时间为 1—2 次课。

（二）基础性实验

此阶段包括 10 个实验的操作训练，这是本实验课程的中心环节。要求做好：

1.预习

学生须认真阅读实验指导书，了解实验的目的和原理，明确本次实验中需要了解和掌握的 Starlogo 建模工具的基本特性和功能，在此基础上写出实验预习报告，内容包括：实验目的和基本原理，测试模块的特性说明，简单的实验步骤。

2．实验

学生进入实验室后首先熟悉 Starlogo 的运行环境。教师检查学生的预习情况并做好记录，包括预习报告和对实验的理解，不合格者不得进行实验。

指导教师讲解实验难点和注意事项，通过提问的方式引导学生深入思考与实验现象有关的一些问题，着力培养学生观察实验、综合考虑问题的能力，使学生学会分析和研究问题的方法。学生独立或按学号编组进行实验，注意实验中的独立操作和相互配合。要求学生在实验中勤于动手，敏锐观察，细心操作，开动脑筋，分析钻研问题，熟练掌握 Starlogo 的功能特性及编程要点。实验结束后经教师检查并签名，实验结果和程序代码才有效。

3．书写实验报告

认真写出实验报告是本课程的基本训练。它将使学生在实验数据处理、作图、误差分析、问题归纳等方面得到训练和提高。实验报告的质量，在很大程度上反映了学生的实际水平和能力。

实验报告的内容大致包括：实验目的和原理、实验测试模块的功能说明、实验结果的展示和分析以及应提交的带注释的源程序代码等。实验报告的讨论可以包括对实验现象的分析和解释、对实验结果的误差分析、对实验的改进意见、心得体会等。

实验结束的两周内完成实验报告，于下次实验时提交报告，其中必须附有教师签名的原始记录表。

（三）综合性实验

综合性实验又分为两种类型：

类型 1：基础性综合实验。开设 5 个与实际应用关系密切、具有一定复杂性和综合程度的实验，学生选做其中的 1 个。该类型实验由教师给出实验题目和要求，学生选择实验题目后需要查找资料，研读文献，钻研有关理论知识，在此基础上选择实验方法，提出实验方案，与指导教师讨论实验方案的可行性，然后预约时间进行实验。

类型 2：研究性综合实验。该类型实验均由教师的科研项目转化而来，目前开发有 5 个实验供学生选择，为实验兴趣浓厚和学有余力的学生提供更多的实验空间，对培养学生的创新意识和科研能力大有裨益。研究性改变了传统实验中固定实验步骤，照方抓药式的被动性，使学生有了很大的自由发挥空间；其次，实验引入了挫折—激励教学机制，允许多次失败，但要求最终一定要成功。

综合实验室每天都对学生开放，学生只要有空，都可以预约时间，到实验室做实验。

六 . 考 核 与 报 告

学生在课下做好预习，写出预习报告，实验前指导教师检查预习情况，根据学生预习报告和回答问题情况给出预习成绩。

实验进行中，教师考察学生的实验操作技能，根据学生实验操作、遵守实验

规则、实验纪律情况以及实验结果展示情况给出实验操作分。

实验结束后学生书写实验报告，并于下一次实验时提交，其中必须附有教师签名的原始记录表。教师根据学生实验报告书写的完整性、实验结果的合理性、对实验的理解和体会等给出报告成绩，包含报告完整、字迹工整(30%)、测试说明正确(10%)、程序代码正确(40%)、实验结果正确(20%)。

期末的最后两次实验作为考核实验，考查学生独立从事实验研究的能力，给出考核成绩。

本实验课程的最后成绩：预习分 20%+操作分 30%+实验报告分 40%+实验考核分 10%

缺少实验或报告者须在实验教学结束前补做和补交，否则不予通过。成绩不合格者下学年重修。

综合实验不安排课内学时，学生利用课外时间到开放实验室进行实验。实验之前写出完整的预习报告，详细列出实验目的、依据的原理、实验步骤、测试说明。实验结束后，将实验结果整理成小论文的形式提交。教师根据学生实践能力、实验水平、实验结果的创造性大小来评定实验成绩。

七．实验指导书（实验教材）

1.《计算机模拟》 方美琪 中国人民大学出版社

2. http://education.mit.edu/starlogo/

八.实验项目与内容提要

序号	实验项目名称	内容提要	实验学时	每组人数	实验类型	实验要求
1	Starlogo 中的 Turtle 控制	1. 学习使用命令创建和消除 Turtle 2. 学习设置 Turtle 属性、状态和动作 3. 了解各种返回值命令	2	1	基础	必做
2	Starlogo 中的 Turtle 计数问题	1. 了解 Starlogo 中求 Turtle 的总和、平均值、最大值、最小值等命令 2. 学习求解有条件约束的计数问题	2	1	基础	必做
3	Starlogo	1. 了解基本运算符的使用方法	2	1	基	必

	中的数学函数运算	2. 学习三角函数的使用方法 3. 掌握函数运算方法			础	做
4	Starlogo中的流程控制	1. 了解Starlogo中流程控制的概念、特点及应用 2. 学习流程控制的结构 3. 熟练掌握各种典型的流程控制语句	2	1	基础	必做
5	Starlogo中的运动与位置指令	1. 了解Starlogo中运动与位置指令的概念和分类 2. 熟练掌握各类指令的内容和使用方法	2	1	基础	必做
6	Starlogo中的监视器功能	1. 熟练操纵turtles的运动过程 2. 了解patches赋值 3. 掌握监视器的性质和功能	2	1	基础	必做
7	Starlogo中的Turtle与环境的交互	1. 了解如何控制turtles的状态变化 2. 学习turtles如何与环境进行交互	2	1	基础	必做
8	Starlogo中的条件判断语句	1. 学习turtle和patch之间值传递的过程 2. 学习使用条件判断语句	2	1	基础	必做
9	Starlogo中的概率运算过程	1. 学习turtle之间状态的交互影响 2. 了解模型中概率的使用方法	2	1	基础	必做
10	Starlogo中的阈值概念及使用方法	1. 了解阈值的概念和使用方法 2. 掌握距离计算的一般公式	2	1	基础	必做
11	Traffic	1. 学习设置和控制turtle和patch的多重属性 2. 掌握复杂的条件控制语句	2	3	综合	选做
12	Painted Turtles	1. 学习如何用Starlogo画图 2. 熟练掌握hatch语句的使用方法 3. 了解有关颜色的常识问题	2	3	综合	选做
13	Yellow Brick Road	1. 学习turtle的颜色识别功能 2. 掌握case语句的使用方法	2	3	综合	选做
14	Predator Prey	1. 学习控制系统时间来模拟生命周期	2	3	综合	选做

	Grass	2. 熟练掌握循环语句的使用方法				
15	Enzyme	1. 学习多条件控制下的模拟过程 2. 学习模型抽象和类比的功能	2	3	综合	选做
16	自行车失窃模型	1. 研究自行车失窃问题的原因 2. 研究自行车失窃问题的规律 3. 找出解决自行车失窃问题的方法	2	5	综合	选做
17	信息发布利弊分析模型	1. 了解大系统的脆弱性 2. 分析信息发布的利弊 3. 权衡利弊，决定是否发布信息	2	5	综合	选做
18	行业竞争模型	1. 熟悉 EC-Game 虚拟社区 2. 了解电子商务 B-B 模式 3. 建立消费者、生产商、供应商和销售商之间的行业竞争模型	2	5	综合	选做
19	银行排队模拟系统	1. 了解目前银行排队现状 2. 分析银行排队情现状的产生原因及影响其结果的关键因素 3. 建立模拟系统解决银行排队问题	2	5	综合	选做
20	信息搜寻与选择模型	1. 了解价格离散的概念、原因和意义 2. 了解信息搜寻与选择的原理 3. 建立信息搜寻与选择模型	2	5	综合	选做

《Swarm》课程实验教学大纲

课程名称：

英文名称：

课程代码：

一.实验总学时 7（课外学时 3/课内学时 4）：　　总学分：2

必开实验个数：2　　选开实验个数：0

二.实验的地位、作用和目的

使学生熟悉计算机建模软件工具（Swarm）的安装、配置和使用方法，为今后的研究做好技术储备。

三.基本原理及课程简介

在科学研究中，特别是在复杂系统的研究中，计算机程序已经开始和科学仪器一同发挥重要的作用。美国圣菲研究所（SFI）为此开发的软件工具集 Swarm，因其强大的功能和开放的设计理念，已被许多科研机构和高等院校用于应用研究或教学。本课程是针对系统理论专业的基础实验课，将对 Swarm 工具的由来、组成结构和设计理念等做详细介绍，指导学生完成 Swarm 的安装配置并通过具体设计一个简单程序学习其使用方法。

四.实验基本要求

1，背景知识讲座

教师讲解、操作及程序演示

2，实验部分

学生应认真阅读实验指导材料，精心准备，并记录实验步骤全过程，以电子文件（如代码，屏幕截图，PPT，电子表格）的方式

记录实验各个关键步骤。

五.考核与报告

学生须在课下做好预习准备。

实验结束后，学生须以电子文件（如代码，屏幕截图，PPT，电子表格）的方式提交实验报告，对实验目的、实验复杂度、依据的原理、实验步骤、数据表格及代码注释进行详细整理。

本实验课程的最后成绩：实验结果分 70%+实验报告分 30%，由教师根据学生实践能力、实验水平、实验结果的创造性大小来评定。

缺少实验或报告者须在实验教学结束前补做和补交，否则不予通过。成绩不合格者下学年重修。

六.实验仪器设备配置

硬件：

IBM-PC 兼容机

主要参数：Intel Celeron 2.0G 256DDR

软件：

Windows 2000/XP Swarm 2.1.1 Swarm 2.2

七.实验指导书（实验教材）

八.实验项目与内容提要

序号	实验项目名称	内容提要	实验学时（课内/外）	每组人数	实验类型	实验类别	实验要求	备注
1	Swarm开发环境的配置	1，在Windows 2000中配置Swarm2.1.1 2，在Windows XP中配置Swarm2.1.1 3，在Windows 2000中配置Swarm2.2 4，在Windows XP中配置Swarm2.2	2(1/1)	2	验证	基础	必修	任选其一，复杂度递增 课堂讲解、课外实现各1学时
2	简单模拟程序设计	用Swarm设计一个简单的程序，模拟部分到总体的涌现现象。	5（3/2）	2	设计	专业基础	必修	参考：Heatbug（热虫模型） 课堂讲解、课外准备、课上提交分别1、2、2学时

实验报告内容的基本要求

1．实验名称、学生姓名、班级和实验日期；

2．实验目的和要求；

3．实验设备；

4．实验原理；

5．实验步骤；

6．实验原始记录；

7．实验结果分析，讨论实验指导书中提出的思考题，心得与体会；

实验教学课程表格式

中国人民大学

ＸＸＸＸ至ＸＸＸＸ学年第Ｘ学期

实 验 课 程 表

实验课程名称	课程代码	实验代码	学分	指导教师	优选类别	年级	起始周	结束周	开课院系	上课时间	上课地点	备注

《分形》

实验教学大纲

课程名称：分形

英文名称：fractal

课程代码：

课程性质：必修

实验指导书名称：分形实验教程

适用专业：系统科学、管理科学与工程

一.实验总学时（课外学时/课内学时）：20　　总学分：2

必开实验个数：5　　选开实验个数：5

二.实验的地位、作用和目的

分形作为系统科学研究的一个重要课题，是系统科学专业和管理科学与工程专业硕士研究生的一门必修课程。分形是现代数学的一个新分支，但其本质却是一种新的世界观和方法论。分形集是一类不能用经典几何方法描述的“不规则”集合，它们基本满足分形的经典性质，但是在自相似的程度上可以有很大差别。分形实验有助于拓展学生的科学视野，增强学生的数学素养，加深对系统科学的理解。分形实验的目的是理论与实践相结合，将分形的原理应用于现实生活中，生成有趣的分形图案，提高学生对科学世界以及数学领域的兴趣，为进一步从事科学研究打下坚实的基础。

三.基本原理及课程简介

被誉为大自然的几何学的分形理论，是现代数学的一个新分支，但其本质却是一种新的世界观和方法论。它与动力系统的混沌理论交叉结合，相辅相成。它承认世界的局部可能在一定条件下，在某一方面表现出与整体的相似性，它承认空间维数的变化既可以是离散的也

可以是连续的，因而拓展了视野。

分形集是一类不能用经典几何方法描述的“不规则”集合，它们基本满足分形的经典性质，但是在自相似的程度上可以有很大差别。分形几何就是研究所谓“简单”空间上这样一类“复杂”子集的一门新兴数学分支。

我们可以根据分形集合生成算法的特征对分形进行分类，一般分为线性分形和非线性分形两大类。这两类分形，都有无穷的算法表述，因而也都包含无穷的分形图形。线性分形是最基本的一种，从数学上说，就是实现这些分形的算法中仅含有一次项，如 Koch 曲线、Peano 曲线及迭代函数系统(IFS)，这些迭代变换使直线保持平直，仅改变其长度、位置和方向。非线性分形则内容丰富得多，变化也多，如 Julia 集。

我们也可以根据分形产生算法中随机性加入的影响，把分形分为确定性分形和随机性分形两类。对确定性分形，其算法的规则是确定的，在算法中也没有随机性的加入，或者虽然有随机性的加入，但并不影响分形图形的形状，即算法的多次重复仍然产生同一个分形图，如 IFS；对随机性分形，虽然其产生的规则也是一定的，但随机因素的影响，可以使每次生成过程产生的分形虽然可以具有一样的复杂度，但形态都会有所不同，它不具有可重现性，如 Brown 运动。

分形的发展很大程度上依赖于计算机科学技术的进步，这也对纯数学的传统观念提出了挑战。计算机技术不仅使分形领域的一些新发现成为可能，同时因其图形直观的表现方式也极大地激发了科学家和人们的兴趣与认识，推动了分形理论的发展。

四.实验基本要求

1、了解分形的基本概念和发展历史。

2、掌握分形的数学原理和基本生成方法。

3、了解分形在现实生活中的应用领域。

4、在基础性实验中，掌握几种经典分形图形的生成算法，能够根据要求生成相应的分形图形。

5、在综合性实验中，能够独立完成对一个复杂系统分形结构的模拟，对模拟过程中出现的问题能够独立排除。

6、能够根据模拟结果，对所提出的问题进行初步分析。

7、撰写简明扼要、图表清晰，结论正确、分析科学的实验报告。

五．实验具体要求

分形实验采用三段式教学模式，教学过程由理论知识准备、基础性实验及综合性实验三个环节组成。

（一）理论知识讲座

通过教师讲解，使学生了解分形的基本概念和发展历史，初步掌握分形的数学原理和基本生成方法。了解本实验课程的性质、任务、要求、课程进度安排、实验考核内容及实验报告要求等，为进行实验做好准备。理论知识准备时间为1—2次课。

（二）基础性实验

此阶段包括5个实验的操作训练，这是本实验课程的中心环节。要求做好：

1.预习

学生须认真阅读实验指导书，了解实验的目的和原理，明确本次实验中需要了解和掌握的分形图形生成的数学原理，在此基础上写出实验预习报告，内容包括：实验目的和基本原理，分形图形生成的数学原理，简单的实验步骤。

2．实验

学生进入实验室后首先熟悉分形程序的运行环境。教师检查学生的预习情况并做好记录，包括预习报告和对实验的理解，不合格者不得进行实验。

指导教师讲解实验难点和注意事项，通过提问的方式引导学生深入思考与实验现象有关的一些问题，着力培养学生观察实验、综合考虑问题的能力，使学生学会分析和研究问题的方法。学生独立或按学号编组进行实验，注意实验中的独立操作和相互配合。要求学生在实验中勤于动手，敏锐观察，细心操作，开动脑筋，分析钻研问题，熟练掌握分形图形的数学原理及生成算法。实验结束后经教师检查并签名，实验结果和程序代码才有效。

3．书写实验报告

认真写出实验报告是本课程的基本训练。它将使学生在实验数据处理、作图、误差分析、问题归纳等方面得到训练和提高。实验报告的质量，在很大程度上反映了学生的实际水平和能力。

实验报告的内容大致包括：实验目的和原理、分形图形生成的数学原理、实验结果的展示和分析以及应提交的带注释的源程序代码等。实验报告的讨论可以包括对实验现象的分析和解释、对实验结果的误差分析、对实验的改进意见、心得体会等。

实验结束的两周内完成实验报告，于下次实验时提交报告，其中必须附有教师签名的原始记录表。

（三）综合性实验

综合性实验的开设是为了使学生在掌握分形的基本原理的基础上，进一步提高学生解决现实具体问题的能力。教师可以根据本专业的实际情况选开其中部分实验。

综合性实验又分为两种类型：

类型 1：基础性综合实验。开设 3 个与实际应用关系密切、具有一定复杂性和综合程度的实验，学生选做其中的 1 个。该类型实验由教师给出实验题目和要求，学生选择实验题目后需要查找资料，研读文献，钻研有关理论知识，在此基础上选择实验方法，提出实验方案，与指导教师讨论实验方案的可行性，然后预约时间进行实验。

类型 2：研究性综合实验。该类型实验均由教师的科研项目转化而来，目前开发有 2 个实验供学生选择，为实验兴趣浓厚和学有余力的学生提供更多的实验空间，对培养学生的创新意识和科研能力大有裨益。研究性改变了传统实验中固定实验步骤，照方抓药式的被动性，使学生有了很大的自由发挥空间；其次，实验引入了挫折—激励教学机制，允许多次失败，但要求最终一定要成功。

综合实验室每天都对学生开放，学生只要有空，都可以预约时间，到实验室做实验。

六.考核与报告

学生在课下做好预习，写出预习报告，实验前指导教师检查预习情况，根据学生预习报告和回答问题情况给出预习成绩。

实验进行中，教师考察学生的实验操作技能，根据学生实验操作、遵守实验规则、实验纪律情况以及实验结果展示情况给出实验操作分。

实验结束后学生书写实验报告，并于下一次实验时提交，其中必须附有教师签名的原始记录表。教师根据学生实验报告书写的完整性、实验结果的合理性、对实验的理解和体会等给出报告成绩，包含报告完整、字迹工整(30%)、数学原理正确(10%)、程序代码正确(40%)、实验结果正确(20%)。

期末的最后两次实验作为考核实验，考查学生独立从事实验研究的能力，给出考核成绩。

本实验课程的最后成绩：预习分 20%+操作分 30%+实验报告分 40%+实验考核分 10%

缺少实验或报告者须在实验教学结束前补做和补交，否则不予通过。成绩不合格者下学年重修。

综合实验不安排课内学时，学生利用课外时间到开放实验室进行实验。实验之前写出完整的预习报告，详细列出实验目的、依据的原理、实验步骤、算法说明。实验结束后，将实验结果整理成小论文的形式提交。教师根据学生实践能力、实验水平、实验结果的创造性大小来评定实验成绩。

七.实验指导书（实验教材）

1．分形论——奇异性探索　林鸿益、李映雪　北京理工大学出版社

2．分形对象-形、机遇和维数　B.曼德尔布洛特　世界图书出版公司

3．分形——大自然的艺术构造　汪富泉等　山东教育出版社

八.实验项目与内容提要

序号	实验项目名称	内容提要	实验学时	每组人数	实验类型	实验要求
1	Mandelbrot 集合	1．了解 Mandelbrot 集合的生成原理 2．用计算机生成 Mandelbrot 集合的分形图形	2	1	基础	必做
2	Julia 集合	1．了解 Julia 集合的生成原理 2．用计算机生成 Julia 集合的分形图形	2	1	基础	必做
3	Newton/Nova 分形	1．了解 Newton 分形的原理及 Nova 分形的改进 2．用计算机生成 Newton/Nova 分形图形	2	1	基础	必做
4	英国的海岸线有多长	1．了解海岸线度量的数学模型 2．用计算机生成海岸线的分形图形	2	1	基础	必做
5	H-分形	1．了解 H-分形的应用背景 2．掌握 H-分形的生成原理 3．用计算机生成 H-分形的图形	2	1	基础	必做
6	L 系统	1．了解 L 系统的生成原理 2．用计算机生成 L 系统的分形图形	2	3	综合	选做
7	IFS 系统	1．了解用 IFS 系统创作分形图形的原理 2．用 IFS 系统生成分形图形	2	3	综合	选做
8	分形图像压缩技术	1．了解分形图像压缩技术的原理 2．掌握分形图像压缩技术的几种典型算法	2	3	综合	选做
9	分形音乐	1．了解分形音乐生成原理 2．用计算机生成一段分形音乐	2	3	综合	选做
10	资本市场的分形结构及其自组织临界性研究	1．了解资本市场的分形结构 2．了解资本市场分形结构的研究意义及应用前景	2	5	综合	选做

系统动力学实验教学大纲

课程名称：系统动力学建模

英文名称：Systematic Dynamics Modeling

课程代码：

实验指导书名称：系统动力学实验教程

开课专业：系统理论

一．实验总学时（课外学时/课内学时）：　　总学分：

必开实验个数：　　选开实验个数：

二.实验的地位、作用和目的

“系统动力学建模”是系统理论专业硕士研究生的必修课《计算机建模与工具》的三大版块之一，是研究某一类复杂系统问题（主要是可用微分方程组表示的动态连续系统）的一种方法学。它以 DYNAMO 语言作为建模语言，这使得系统动力学的建模方法具有更为深刻的实际意义。通过该课程及实验课的教学使学生能够掌握对一类复杂系统进行建模和模拟的方法，为研究复杂系统，利用模型进行分析和决策实际问题奠定良好的理论和实践基础。依据教学大纲的要求，通过实验教学环节，使学生在课堂理论学习的基础上，通过实验来理论联系实际。

三、基本原理及课程简介

系统动力学的基本思想是充分认识系统中的反馈和动态性，并按一定的规则逐步的建立系统动力学的结构模式。动态性，是指系统所包含的量具有随时间而变化的特征。比如，企业雇用员工的变动、股票市场上股票价格和交易额的波动、城市中税收和生活标准的变化、甚至糖尿病的血糖指标的变化，这些都是动态问题。它们可以用变量随时间变化的图形来表示。系统动力学中的动态性，不是随机的不稳定的动态性，而是可以预期的，有一定规律的动态性。同时，某个变动经常在时间上表现出一定的延迟。从这个意义上，也可以说系统动力学的两个基本观点是：反馈和延迟。

系统动力学把现实生活中的复杂系统映射成系统动力学流图，DYNAMO 语言则把系统流图模型送入计算机并计算出数字结果。

用系统动力学的观点来研究一个问题，大致可以分为以下几个阶段：（1）问题的识别和定义。（2）系统的概念化。（3）模型格式化（模型的建立）。（4）模型行为的分析（计算机模拟）。（5）策略分析。（6）模型的使用或执行。每个阶段的起点和终点和整个过程的起点和终点都是对这一系统及其问题的不断深入的理解。因此，它是一个环，或者网，而不是线性的序列。一个循环之后又可以开始新的一个循环。可以不断反复迭代。

系统动力学研究所得出的策略建议，不仅来自模型的最终计算结果，而且还来自模拟过程中通过各阶段迭代所得到的认识。系统动力学研究最终应该能够提出切实可行的策略建议。因为模型只是达到目的的工具，最终的目的是提高对现实世界的认识，增加对客观规律的理解。

四、实验基本要求

- 能够从计算机能接受的概念模式入手，把实际中要解决的问题转化为计算机可以识别的形式。
- 掌握因果关系图和系统动力学流图的画法，熟练掌握系统动力学方程的写法。
- 独立完成一个复杂系统的系统分析、编码实现，对模拟过程中出现的问题能独立排除。
- 撰写简明扼要、图表清晰，结论正确、分析科学的实验总结报告。

五、实验具体要求

系统动力学实验采用三段式教学模式，教学过程由理论知识课、基础性实验、综合性实验三个环节组成。

（一）理论知识讲座

通过教师的讲解，使学生了解课程的性质、任务、要求、课程安排和进度、实验原理、实验考核内容和实验守则等，为上机实验做好准备。

（二）基础性实验

此阶段包括 16 个实验的上机练习，这是本实验课程的中心环节。要求做好：

1.预习

学生须认真阅读实验指导书，了解实验的目的和原理，明确本次实验中使用的工具平台和实验方法等，在此基础上写出预习报告，内容包括：实验目的和基本原理，简单的实验步骤，原始数据记录表格。

2．实验

指导教师讲解实验难点和注意事项，通过提问的方式引导学生深入思考与实验过程有关的一些问题，着力培养学生综合考虑问题和将实际问题转化为计算机模型的能力，使学生学会分析和研究问题的方法。要求学生在实验中勤于动手，敏锐观察,开动脑筋，分析钻研问题，准确记录数据，发现并分析解决实验中出现的问题。

3．书写实验报告

认真写出实验报告是本课程的基本训练。它将使学生在实验数据处理、作图、误差分析、问题归纳等方面得到训练和提高。实验报告的质量，在很大程度上反映了学生的实际水平和能力。

实验报告的内容大致包括：实验目的和原理、实验的软件平台、对实际问题的抽象和分析以及形式化、建模流程步骤和结果的评价检验。实验报告的讨论可以包括对实际问题的分析和解释、对实验参数的取舍和赋值、对实验的改进意见、心得体会等。

实验结束的两周内完成实验报告，于下次实验时提交报告。

（三）综合性实验

综合性实验又分为两种类型：

类型 1：基础性综合实验。开设 4 个与社会关系密切、具有一定复杂性和综合程度的实验，学生选做其中的 1 个。该类型实验由教师给出实验题目和要求，学生选择实验题目后需要查找资料，研读文献，钻研有关理论知识，在此基础上选择实验方法，提出实验方案，与指导教师讨论实验方案的可行性，然后预约时间进行实验。

类型 2：研究性综合实验。该类型实验由教师的科研项目转化而来，为实验兴趣浓厚和学有余力的学生提供更多的实验空间，对培养学生的创新意识和科研能力大有裨益。研究性改变了传统实验中固定实验步骤，照方抓药式的被动性，使学生有了很大的自由发挥空间。

综合实验室每天都对学生开放，学生只要有空，都可以到实验室做实验。

六、考核与报告

实验课内考核为 20 分，评分标准如下：

1 ）实验操作（ 6 分）

实验操作方法正确，能熟练操作 Versim 工 具。 实验结果正确，计 6 分（有问题酌情扣分）。

2 ）开发文档（ 10 ）

要求能够独立完成系统仿真各阶段的文档，文档包括系统调研文档、系统分析文档、系统设计文档。

系统开发文档整体完备，计 4 分（有问题酌情扣分）

系统分析文档正确、详细，计 3 分（有问题酌情扣分）

系统设计文档正确、详细，计 3 分（有问题酌情扣分）

3) 实验总结报告（ 4 分）

内容全面，字迹清晰工整。对开发中出现的问题分析正确，并能解决问题，计 4 分（有问题酌情扣分）。如果实验总结报告有雷同均以 0 分计。

4 ）无故缺席实验课，实验成绩以 0 分计，并取消该门理论期末考试资格。

5 ）特殊情况（事假、病假）必须由本人提出申请，学院主管领导批准，待期末考试前统一补做。

七、实验教材

1.《计算机模拟》 方美琪 中国人民大学出版社

2.《管理与社会经济系统仿真》 宣慧玉 高宝俊 武汉大学出版社

八、实验模拟工具和平台

1. Ventana.Vensim.PLE.v5.4B.CR.Final

2. DYNAMO 中文版 （可选）

九、实验项目

序号	实验项目名称	内容提要	实验学时	实验性质	必开/选开
1	蒙特卡洛法求面积	1. 用计算机模拟求积分	1	验证	必开
2	中子穿墙实验	1. 用计算机模拟随机过程	1	验证	必开
3	空调机调节室温实验	1. 认识系统中的反馈关系 2. 学会画因果关系图 3. 学会画系统动力学流图	3	基础	必开
4	细菌生长实验（一）	1. 认识一阶正反溃系统 2. 熟练因果关系图的画法	2	基础	必开
5	热风调节实验	1. 认识二阶系统	2	基础	必开

		2. 认识系统的延迟 3. 画出加入适当控制参数的流图			
6	定货模型	1. 画因果关系图 2. 加深对反馈和延迟的认识	1	基础	必开
7	多实物流定货模型	1. 学习把现实问题归纳为计算机模型	1	基础	必开
8	细菌生长实验（二）	1. 学习写系统动力学方程	2	基础	必开
9	食物链模型	1. 练习写系统动力学方程 2. 学习表函数和正余弦函数的使用	4	基础	必开
10	河流中的农药净化模型	1. 学习一阶物流延迟函数的原理和应用 2. 学习脉冲函数	1	基础	必开
11	城市建房模型	1. 学习一阶物流延迟	1	基础	选开
12	定货购物商店模拟	1. 巩固系统动力学流图的画法 2. 熟练书写系统动力学方程 3. 学习三阶物流延迟	3	基础性综合	必开
13	商店定货策略的比较	1. 学习分析复杂系统各主体的因果关系 2. 学习信息延迟 3. 学习对模拟结果的分析	3	基础性综合	必开
14	工厂生产模型	1. 学习分析复杂系统各主体的因果关系 2. 学习信息延迟 3. 学习对模拟结果的分析	3	基础性综合	选开
15	供应链牛鞭效应模型	1. 模拟牛鞭效应曲线 2. 学习开关变量的使用 3. 模型结果的比较分析	3	基础性综合	选开
16	电信	1. 强化对于复杂系统中因果关系的分析 2. 了解对原始数据的收集和处理方法 3. 学习写系统设计文档	4	研究性综合	选开

系统动力学建模

实验指导书

中国人民大学经济科学实验室

2006 年 **9** 月

前　　言

《系统动力学建模》课程是硕士研究生系统理论专业的重要专业课程之一。计算机模拟技术的迅速发展和在当今信息社会中的广泛应用，给《系统动力学建模》课程的教学提出了新的更高的要求。

由于系统动力学建模是一门实践性较强的技术，课堂教学应该与实践环节紧密结合。将《系统动力学建模》课程建设成世界一流的课程，是近期《系统动力学建模》课程努力的方向。本学期我们重新编写了实验指导书，调整了实验安排，加大了实践力度。希望同学们能够充分利用实验条件，认真完成实验，从实验中得到应有的锻炼和培养。

希望同学们在使用本实验指导书及进行实验的过程中，能够帮助我们不断地发现问题，并提出建议，使《系统动力学建模》成为具有世界一流水平的课程。

本学期授课教师为方美琪教授。

目　　录

实验要求

“系统动力学建模”是系统理论专业硕士研究生的必修课《计算机建模与工具》的三大版块之一，是研究某一类复杂系统问题（主要是可用微分方程组表示的动态连续系统）的一种方法学。它以 DYNAMO 语言作为建模语言，这使得系统动力学的建模方法具有更为深刻的实际意义。通过该课程及实验课的教学使学生能够掌握对一类复杂系统进行建模和模拟的方法，为研究复杂系统，利用模型进行分析和决策实际问题奠定良好的理论和实践基础。

在《系统动力学建模》的课程实验过程中，要求学生做到：

- 能够从计算机能接受的概念模式入手，把实际中要解决的问题转化为计算机可以识别的形式。
- 掌握因果关系图和系统动力学流图的画法，熟练掌握系统动力学方程的写法。
- 独立完成一个复杂系统的系统分析、编码实现，对模拟过程中出现的问题能独立排除。
- 撰写简明扼要、图表清晰，结论正确、分析科学的实验总结报告。

实验的验收将分为两个部分。第一部分是上机操作，包括检查程序运行和即时提问。第二部分是提交书面的实验报告。此外，针对以前教学中出现的问题，建模实验将采用阶段检查方式，每个实验都将应当在规定的时间内完成并检查通过，提前完成会有加分，过期完成会有较大额度的扣分，直至扣到0 分，以避免期末集中检查方式产生的诸多不良问题，希望同学们抓紧时间，合理安排，认真完成。

实验一 蒙特卡洛法求面积

1. 实验目的

本实验是计算机模拟的入门实验，目的在于使同学们了解一种全新的计算机建模思想来代替传统的数学模型方法解决不了的问题。传统的做法是把变动的机制用数学解析式表达出来。这种做法只把可以量化的几可以用数学解析式表时的那部分知识、信息表示出来。而计算机模拟则只凭经验数据，直接模仿客观现象，不仅利用数量关系，还利用逻辑关系描述复杂的现象。

2. 实验内容

用蒙特卡洛法求给定一条曲线 y=f(x)与直线 x=a 和 x=b 以及坐标轴 x 所围成的图形的面积。

3. 模拟实现及步骤

蒙特卡洛法是把求曲线下所围的面积化成一个随机数是否落在曲线下的实验。

（1） 划定一个能包含所求面积的矩形区域，找到 y 的值域。

（2） 将区间划分成若干等份（通常大于 200）。

（3） 每个 x_i 对应一个 0 到 1 的随机数 R_i，若 R_i 大于 $f(x_i)$,表示随机数落在曲线 $f(x_i)$之上；反之，随机数落在曲线之下。

（4） 从落在曲线之下的点数 n 与总份数之比可以求出面积。

4. 注意事项

用看均匀随机落下的点是否落在曲线之下来模拟曲线的面积。这两件事的实质不一样，但现象上类似。用类比、用模拟的办法解决数学问题显然是个好方法。

实验二 中子穿墙实验

1. 实验目的

本实验的目的在于加深同学们对于用计算机模拟的方法去解决一些不好解决的数学问题的认识和理解。

2. 实验内容

1. 用计算机模拟随机过程，求出中子穿越铅墙的百分比。利用程序模拟这个过程，把所有与过程有关的活动、事件、随机量都在程序中反映出来，使得在过程中可以量化的或不可量化的信息全部利用上。其 BASIC 程序如下：

```
INPUT “请输入被试验的中子数？”; N
M = 0
FOR I = 1 TO N
X =1.0
PI = 3.1416
FOR K = 1 TO 10
Y = PI * RND(1)
X = X + COS(Y)
IF X > 5.0 THEN 120
IF X < 0.0 THEN 130
NEXT K
GOTO 130
M = M + 1
NEXT I
PRINT “对于”; N; “个中子，其穿过铅墙的百分比为”; M/N
END
```

3. 习题

请用 java 或 c 语言实现上述随机过程。

实验三　空调机调节室温模拟

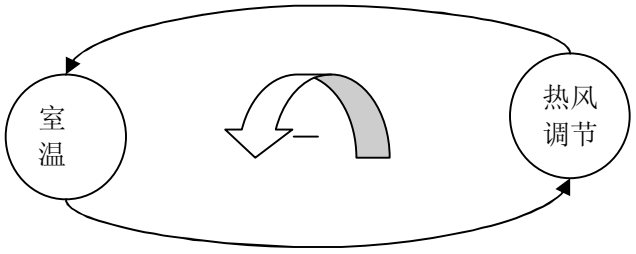
1. 实验目的

本实验的目的在于使学生接触和理解系统动力学的基本思想之一——反馈，学会画因果关系图，并按一定的规则逐步建立系统动力学流图，体会把现实的系统转化成为模型。

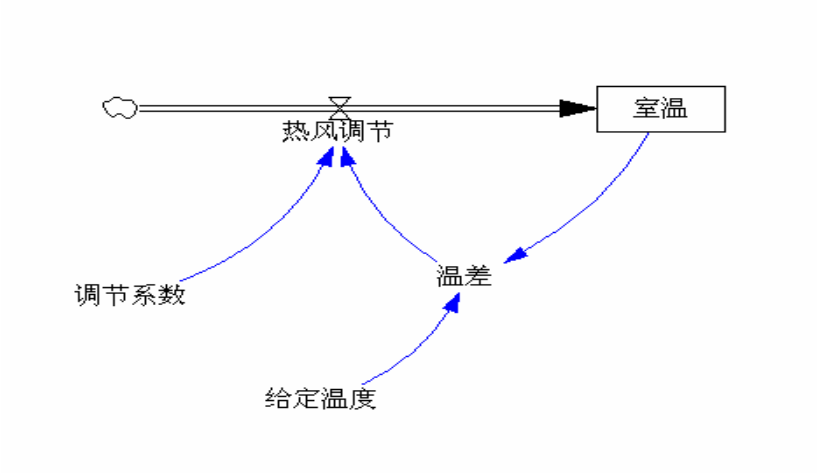
2. 实验内容

1. 抽象出室温和热风调节之间的因果关系，标明正反馈和负反馈。
2. 画调温过程的系统动力学流图。

3. 实验结果参考



因果关系图



系统动力学流图

4. 注意事项

1．正确区分正反馈和负反馈。

2．注意流图中元素的表示方法。

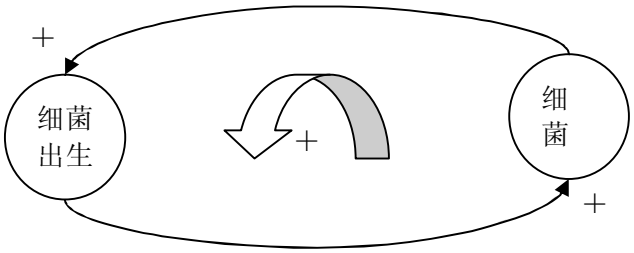
实验四 细菌生长实验（一）

1. 实验目的

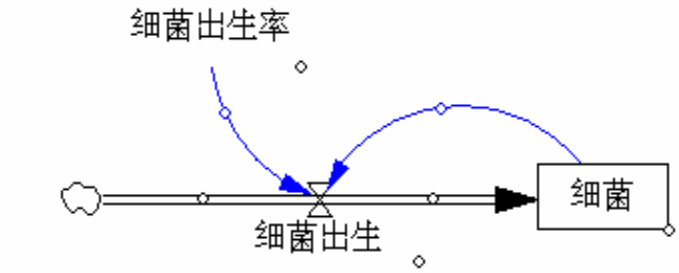
如果一阶习题中有正反馈，此系统就称为一阶正反馈系统。本实验的目的在于使学生认识一阶正反馈系统，熟练因果关系图的画法。

2. 实验过程

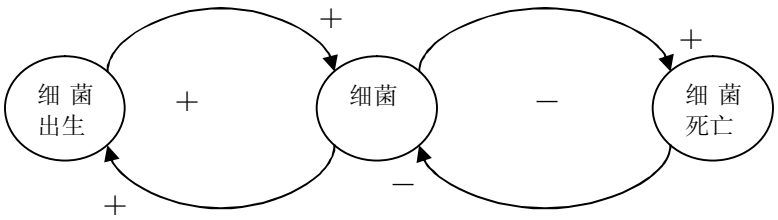
细菌生长是一种典型的一阶正反馈系统，细菌出生的速度越快，细菌的数量越多；而细菌的数量越多，细菌出生的速度也越快，它们之间的因果关系图如下：



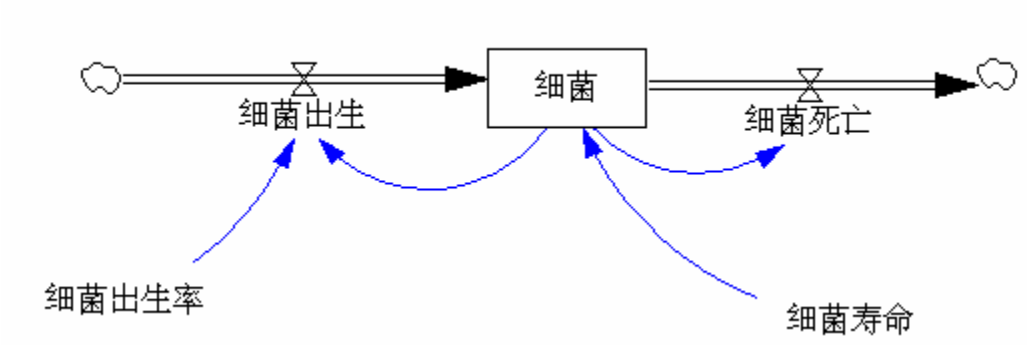
其中，细菌数量是通过细菌出生率来对细菌出生产生影响的。上面的因果关系图对应流图如下：



如果仔细考虑，细菌数量变化的动态描述还应作些补充。细菌除了繁殖使之增加外，细菌还会死亡，因此细菌数量与细菌出生和细菌死亡之间呈现下面的因果关系。



把这种因果关系图转换成下面的系统流图：



3. 注意事项

1．本例中一个存量与两个速率对应。进入的箭头表示细菌出生使细菌数量增加，出去的箭头表示细菌死亡使细菌减少。

2．细菌寿命的倒数是死亡率。

2．注意和区分反馈及系统动态变化模式的不同形式。

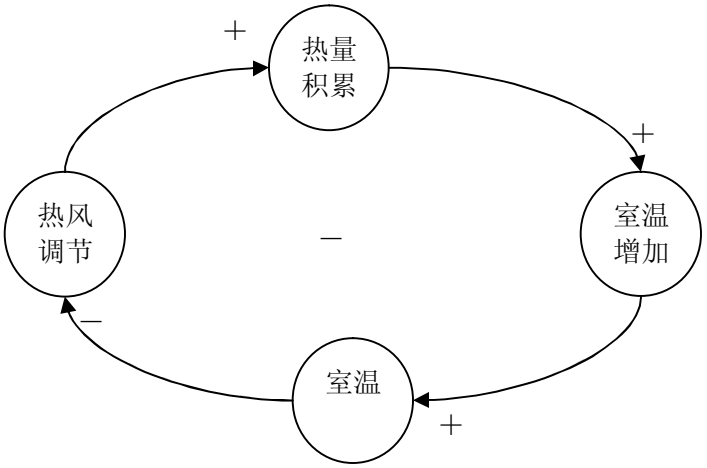
实验五 热风调节实验

1. 实验目的

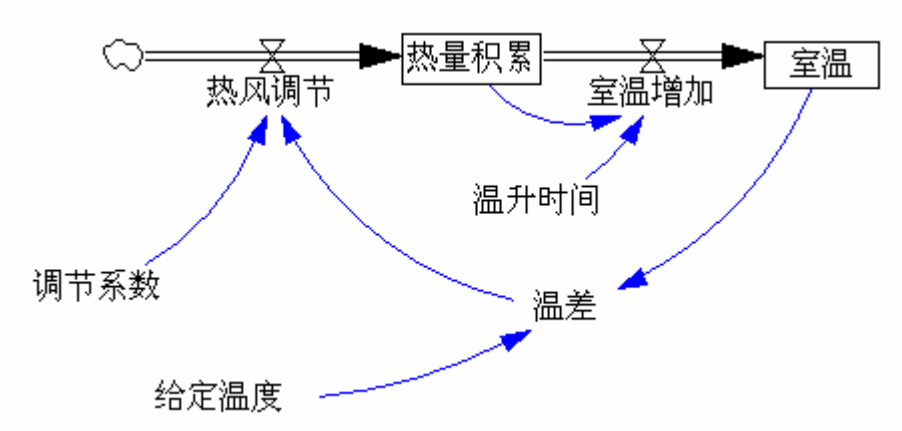
本实验旨在使学生认识二阶系统，认识系统动力学的另外一个特点——延迟，画出加入适当参数的系统动力学流图。

2. 实验步骤

(1) 回顾室温调节的过程，画出因果关系图。



（2）把因果关系图转换成流图。



3. 注意事项

在这个系统中，热风调节以后，热量要经过一定的时间才会积累起来，室温才会升高。所以如果室温低于给定温度，热风就会增加，但这些热风过了一定时间才会影响室温，那么这个影响开始产生作用前，室温仍低于给定温度，热风就仍然开放着。当室温等于期望温度时，虽然不再送热风，但是刚才的热风会积累热量，热量积累在时间上的延迟就会使温度继续升高，因而出现了温度高于给定温度的情况。因此这个系统中会出现室温在给定温度上下波动的情况。

4. 习题

在同一个坐标系中画出此系统中热风调节、热量积累、室温和给定温度四个量大致的变化趋势和关系图。

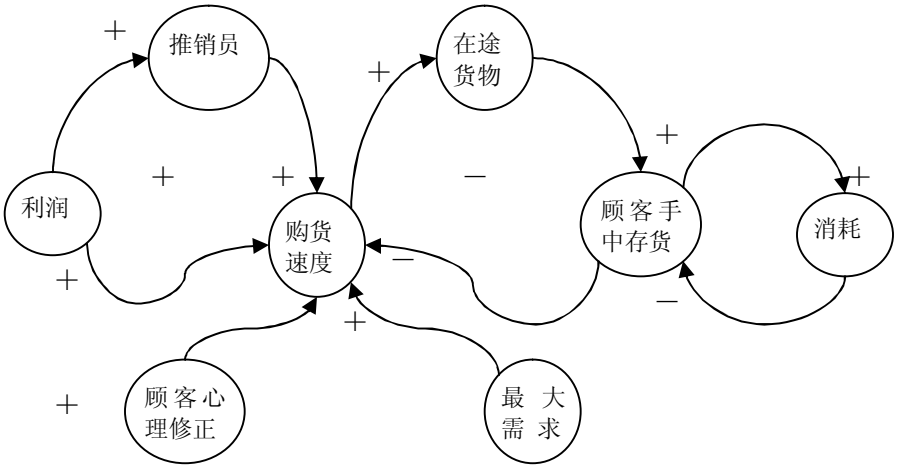
实验六 多实物流定货模型

1. 实验目的

了解定货中“延迟”的作用，分析较为复杂的系统中各个元素间的相互关系；充分认识反馈对整个实物流的影响。

2. 实验步骤

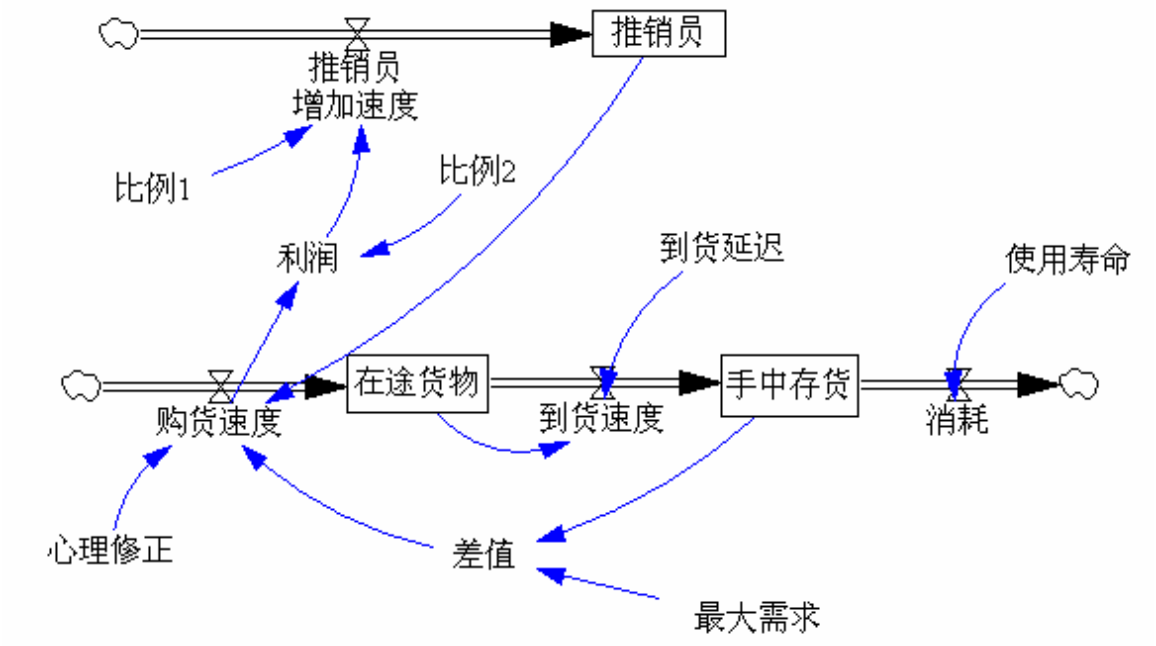
（1）分析某种商品分布情况动态及推销员数量的设置，画因果关系图。



其中，购货速度转化为顾客手中的存货，要经过延迟。这集中表示为在途货物。由于顾客的最大需求是固定的，所以最终顾客手中的存货又制约了购货速度。这样从“购货速度”到“在途货物”，再到“顾客手中存货”，返回“购货速度”，这之间存在一个负反馈环。

（2）确定流图中的主要实物流及信息链。此例中一条实物流应是反映推销员数量的，另一条是反映商品流通过程的。

（3）画系统动力学流图。



3. 注意事项

得到系统流图是认识系统动态的关键一步。根据系统的原理，采用分解和综合的办法来认识系统。首先根据研究的目标找出我们关心的对象，即确定系统的边界。然后把系统分解成其组成要素。再按上面的几个原则找反馈环，认识延迟的存在，鉴别出存量和速率，找出实物流和信息流。最后将它们正确的组合成一个有机的整体。

4. 习题

找出并分析系统中所有的反馈环。

实验七 细菌生长实验（二）

1. 实验目的

在实验四的基础上编写系统动力学方程，并在 DYNAMO 上进行模拟分析。

2. 实验步骤

（1）写存量方程

L 细菌.K = 细菌.J + DT * (细菌出生.JK – 细菌死亡.JK)

（2）写速率方程

R 细菌出生.KL = 细菌出生率 * 细菌.K

R 细菌死亡.KL = 细菌.K / 细菌寿命

（3）写常量方程和初值方程

C 细菌出生率 = 0.8

C 细菌寿命 = 2

N 细菌 = 1000

（4）说明语句

SPEC DT =1, LENGTH = 20, PRTPER = 1, PLTPER = 1

（5）在 DYNAMO 上模拟，得到细菌数量曲线。

3. 注意问题

书写方程式应按照从存量到速率，再到其他变量和常量的顺序，并注意时间下标。

实验八 食物链模型

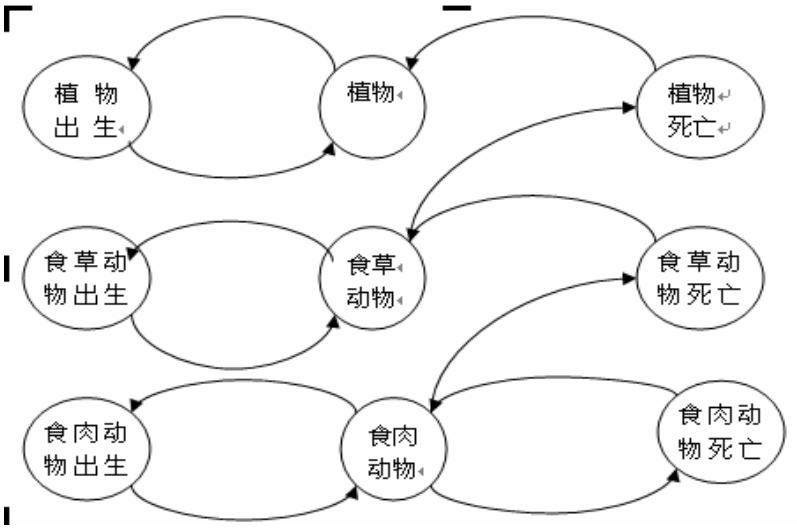
1. 实验目的

本实验的目的在于让学生练习写系统动力学方程，学习表函数和正余弦函数的使用。

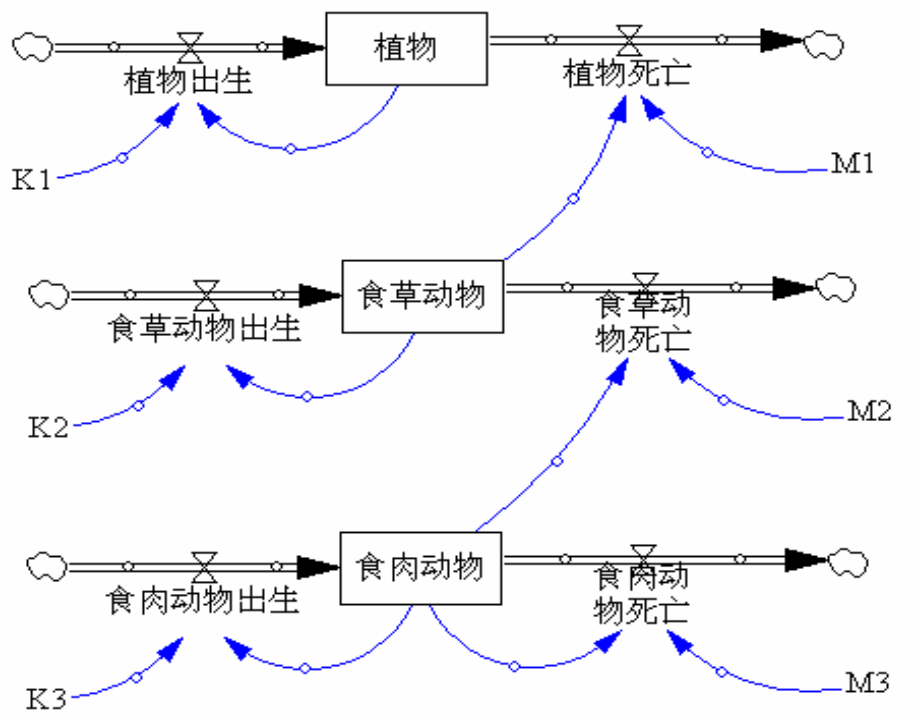
2. 实验步骤

（1）确定系统的边界：在一个封闭的海岛上，有植物、食草动物和食肉动物构成一个简单的生态系统。

（2）粗略分析三者之间的关系，得到下面的因果关系图。



（3）画出系统动力学流图。



（4）写 DYNAMO 方程

L 植物.K＝植物.J＋DT*（植物出生率.JK－植物死亡率.JK）

R 植物出生率.KL=植物.K*K1

R 植物死亡率.KL=植物.K*M1

L 食草动物.K＝食草动物.J＋DT*（食草动物出生率.JK－食草动物死亡率.JK）

R 食草动物出生率.KL=食草动物.K*K2

R 食草动物死亡率.KL=食草动物.K*M2

L 食肉动物.K＝食肉动物.J＋DT*（食肉动物出生率.JK－食肉动物死亡率.JK）

R 食肉动物出生率.KL=食肉动物.K*K3

R 食肉动物死亡率.KL=食肉动物.K*M3

N 植物＝10,000

N 食草动物＝1,000

N 食肉动物＝100

C K1＝3

C K2＝2

C K3＝1

C M1=15

C M2=20

C M3=0.5

(5) 细化数量关系，补充细节，对出生率引入噪音函数和正弦函数。

植物出生率受到气候影响和其他不可预知的因素的影响，因此，植物出生率不是一个常数，其影响因素如下面的方程所示：

A 植物出生率.K=平均出生率常数+气候影响.K+随机噪音.K

C 平均出生率常数＝3

A 气候影响.K=SIN（6.28*TIME.K/10）

A 随机噪音.K=0.1*NOISE（）

对食肉动物：

A 食肉动物寿命.K=平均寿命＋瘟疫影响.K

A 瘟疫影响.K=0.5*COS(6.28*TIME.K/20)

C 平均寿命＝2

植物作为食草动物的食物，影响食草动物的出生率。运用表函数，食草动物出生率和食肉动物出生率可以如下表示：

食草动物出生率的方程：

A 食草动物出生率.K=Y.K

A Y.K=TABLE（TY,食草动物所占的植物数.K，0，15，5）

T TY＝0，1，3，4

A 食草动物所占有的植物数.K=植物.K/食草动物.K

食肉动物出生率的方程：

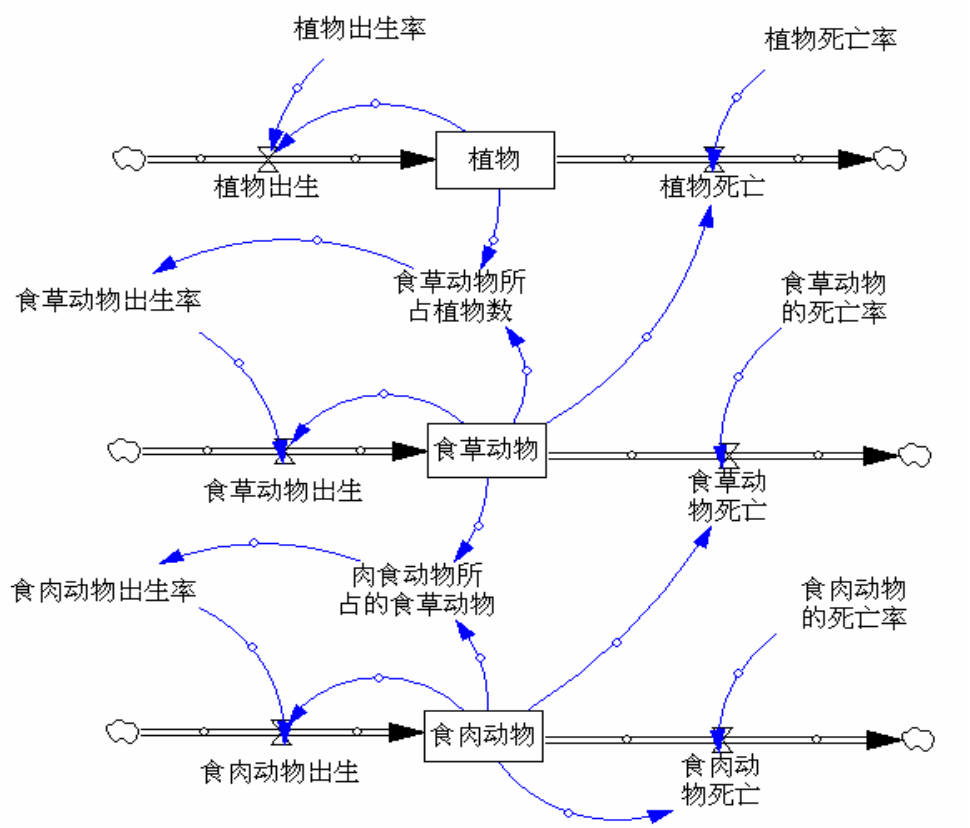
A 食肉动物出生率.K=Z.K

A Y.K=TABLE（TZ,食肉动物所占的食草动物数.K，0，15，5）

T TZ＝0，0.5，2，2.5

A 食肉动物所占有的植物数.K=食草动物.K/食肉动物.K

（6）画出细化后的系统流图。



（7）写 DYNAMO 方程

L 植物.K=植物.J+DT*(植物出生.JK-植物死亡.JK)

R 植物出生.KL=植物.K*植物出生率.K

A 植物出生率.K=平均植物出生率+气候影响.K+噪音.K

A 气候影响.K=SIN(6.28*TIME.K/10)

A 噪音.K=0.1*NOISE()

R 植物死亡.KL=食草动物.K*植物死亡率

L 食草动物.K=食草动物.J+DT*(食草动物出生.JK-食草动物死亡.JK)

R 食草动物出生.KL=食草动物.K*食草动物出生率.K

A 食草动物出生率.K=TABLE(TY,食草动物占有植物数.K,0,15,5)

T TY=0,1,3,4

A 食草动物占有植物数.K=植物.K/食草动物.K

R 食草动物死亡.KL=食肉动物.K*食草动物死亡率

L 食肉动物.K=食肉动物.J+DT*(食肉动物出生.JK-食肉动物死亡.JK)

R 食肉动物出生.KL=食肉动物.K*食肉动物出生率.K

A 食肉动物出生率.K=TABLE(TZ,食肉动物占有食草动物.K,0,15,5)

T TZ=0,0.5,2,2.5

A 食肉动物占有食草动物.K=食草动物.K/食肉动物.K

R 食肉动物死亡.KL=食肉动物.K*食肉动物死亡率.K

A 食肉动物死亡率.K=1/寿命.K

A 寿命.K=平均寿命+瘟疫影响.K

A 瘟疫影响.K=0.5*COS(6.28*TIME.K/20)

C 平均植物出生率=3

C 植物死亡率=0.5

C 食草动物死亡率=1.2

C 平均寿命=2

N 植物=10000

N 食草动物=1000

N 食肉动物=100

（8）在 DYNAMO 上模拟运行。

4. 习题

（1）画出植物出生率、食草动物出生率和食肉动物出生率的细致流图。

（2）查阅文献，尝试添加或改变出生率影响因素的数量关系，看其对最终模拟结果的影响。

实验九 河流中的农药净化模型

1. 实验目的

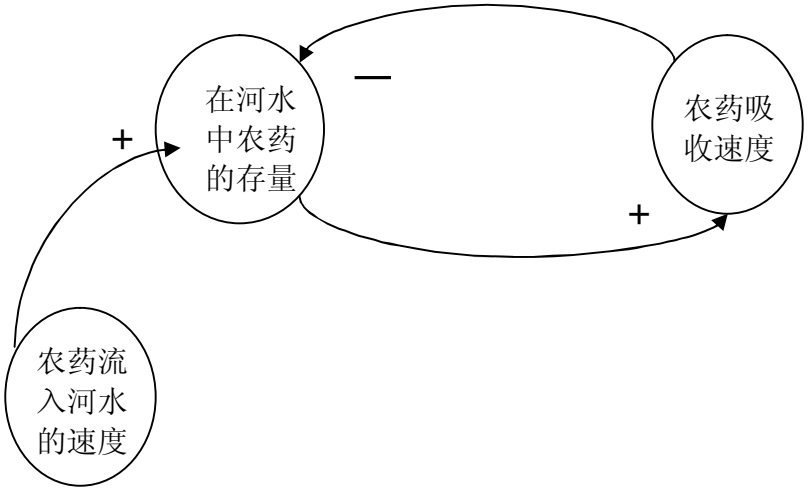
各种系统中都存在延迟。一个原因在一定的时间间隔之后才会产生效果的现象称为延迟。本实验的目的在于使学生了解一阶物流延迟的概念及延迟宏函数的写法，掌握脉冲函数的用法。

2. 实验内容

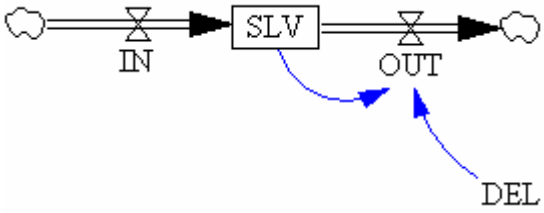
（1）确定所研究问题的背景和界限：假设某工厂每周向河里倒一次农药残渣，两天农药就

自然的被吸收净化。开始时河水无农药，工厂每隔 7 天向河里到一次农药残渣，每次倒入 420 公斤。由于农药的吸收是花时间的，所以它在水中的含量不是立即减少，而是逐渐减少。

（2）画出因果关系图，注意其中的反馈关系



（3）画出一阶物流延迟的系统流图。



（4）写出一阶物流延迟的动力学方程。

L　SLV.K = SLV.J + DT * (IN.JK - OUT.JK)

R　OUT.KL = SLV.K / DEL

N　SLV = IN * DEL

（5）用一阶延迟宏函数写出完整的动力学方程。

A　DELAY1.K = SLV.K / DEL

L　SLV.K = SLV.J + DT * (IN.JK – DELAY1.J)

N　SLV = DEL * IN

3. 注意问题

（1）注意延迟过程中存量初始值的赋值，此例中河水中的农药量初值是在没有倾倒农药时（即没有发生阶跃）时的平衡状态。

（2）注意用宏函数代替速度输入时的时间下标，DELAY1 的下标一般为 J。

实验十 城市建房模型

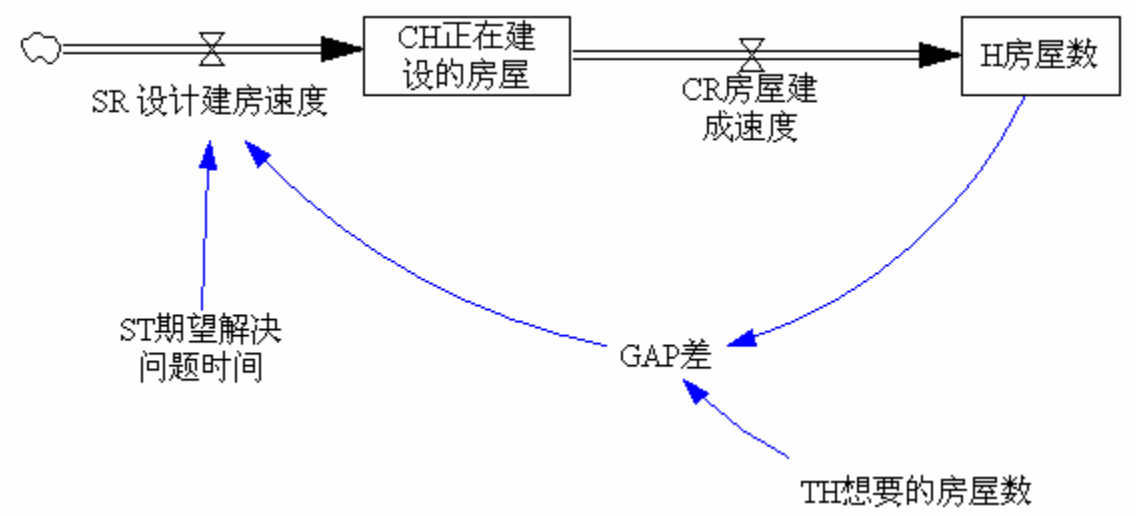
1. 实验目的

本实验的目的在于巩固一阶延迟宏函数 DELAY1 的应用，学会将实际问题中的延迟用模型可识别的语言表达出来。

2. 实验步骤

（1）了解城市建房问题的背景。设城市建设者要决策一个期望解决问题的时间。该城市已有一定数量的房屋，但还有一定数量的居民户没有房屋住。总户数与房屋数之差为需要的房屋数。城市建设者决定用 10 年来解决所有人的住房问题。从设计建房到真正建成还有一段时间，要征地、备料、建筑等，因此有一些正在建而未建成的房屋。

（2）画城市建房的系统动力学流图。



（3）写 DYNAMO 方程

L CH.K = CH.J + DT * (SR.JK – CR.JK)

R SR.KL = GAP.K / ST

C ST = 10

R CR.KL = CH.K / CT

L H.K = H.J + DT * CR.JK

A GAP.K = TH – H.K

C　CT = 2

C　TH = 5000

N　CH = 0

N　H = 3000

把 CH、CT、CR 三个元素简化成一个元素，用 DELAY1 来代替，方程可写为：

L　CH.K = CH.J + DT * DELAY1 (SR.JK, CT)

C　CT = 2

R　SR.KL = GAP.K / ST

A　GAP.K = TH – H.K

C　ST = 10

C　TH = 5000

N　H = 3000

3. 习题

画出用一阶延迟宏函数 DELAY1 代替 CH、CT、CR 三个元素后的系统动力学流图。

实验十一 定货购物商店模拟

1. 实验目的

（1）巩固系统动力学流图的画法

（2）熟练书写系统动力学方程

（3）学习三阶物流延迟

2. 实验步骤

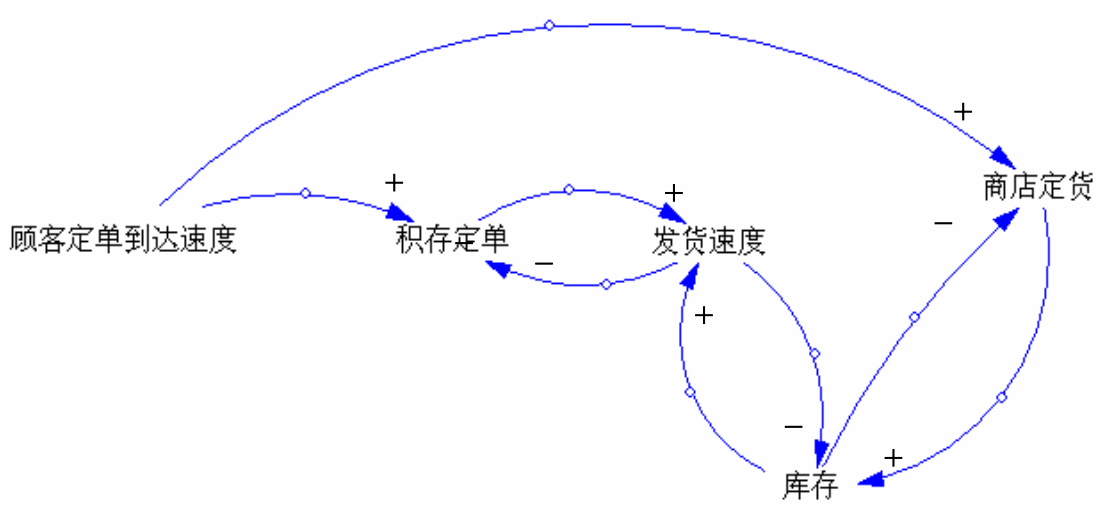
（1）了解商店定货的业务流程。

商店经理希望了解商店中定货单、库存变化的动态，例如，当外界对商店货物需求发生了一个突然增加后，定货单如何变化，库存如何变化。定货商店业务处理流程：顾客定货后，由店员处理这些定单。按照定货单到商店的仓库取货，并将货发送给顾客。发货结

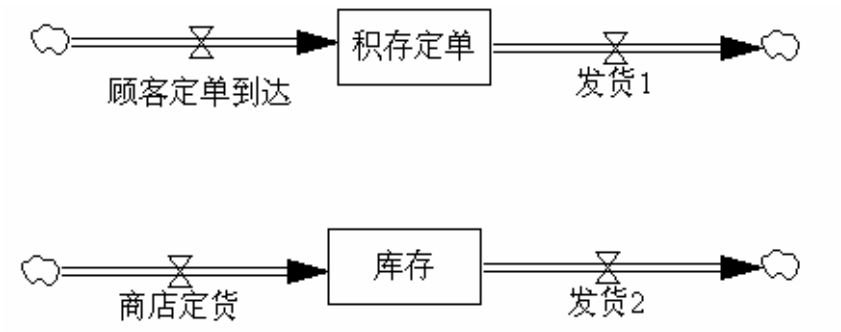
束后，这笔业务就算完成了。为了保持商店的仓库总有货物，商店就必需去定货，有着些定货来补充仓库中的库存。

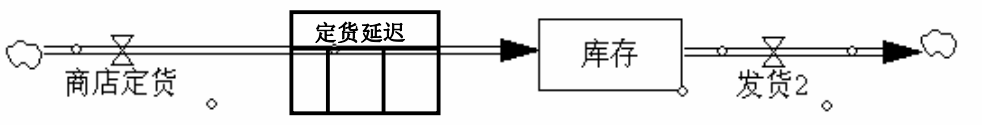
（2）确定基本框架

- 存量：积存定单、库存
- 速率：顾客定单到达速度、发货速度、商店定货速度



- 两条实物流：顾客定单流及商店货物流





（2）确定速率、补充细节

① 顾客定货速率是一个外部不可控因素，模型将其假设为一个阶跃变化是所引起的系统状态的动态变化。

R 顾客定货.KL=正常值＋STEP(100,5)

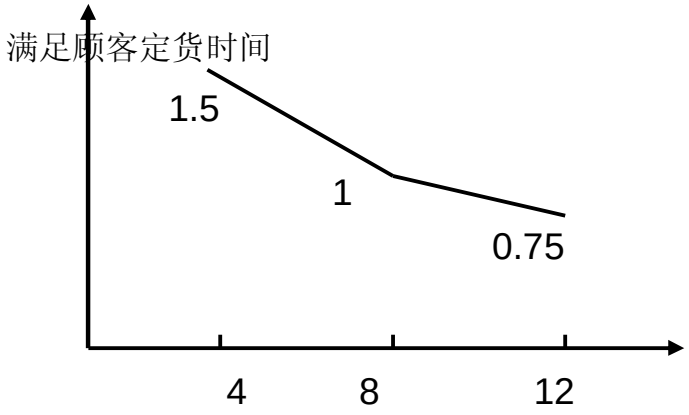
C 正常值＝1000

② 发货速度与积存定单成正比，与满足顾客定货时间成反比。而满足顾客定货时间与实际库存成反比，与平均的顾客定货成正比。平均的顾客定货是顾客定货的一阶信息延迟。

A 平均的顾客定货.K=SMOOTH(顾客定货.JK,平均时间）

C 平均时间＝8

满足顾客定货时间＝a/(实际库存/平均的顾客定货),比例系数 a 可用表函数来确定：



R 发货.KL=积存定单.K/满足顾客定货时间.K

A 满足顾客定货时间.K＝TABLE(TY,实际库存.K/平均的顾客定货.K,4,12,4)

T TY=1.5,1,0.75

③ 商店定货速率是商店的定货策略，此商店经理的策略是将定货速率分成两部分。一部分是与前 8 周内的平均顾客定货一致，另一部分是调节实际库存到一个期望库存。调节时间是 4 周。期望库存是希望维持 8 周的平均顾客定货量。

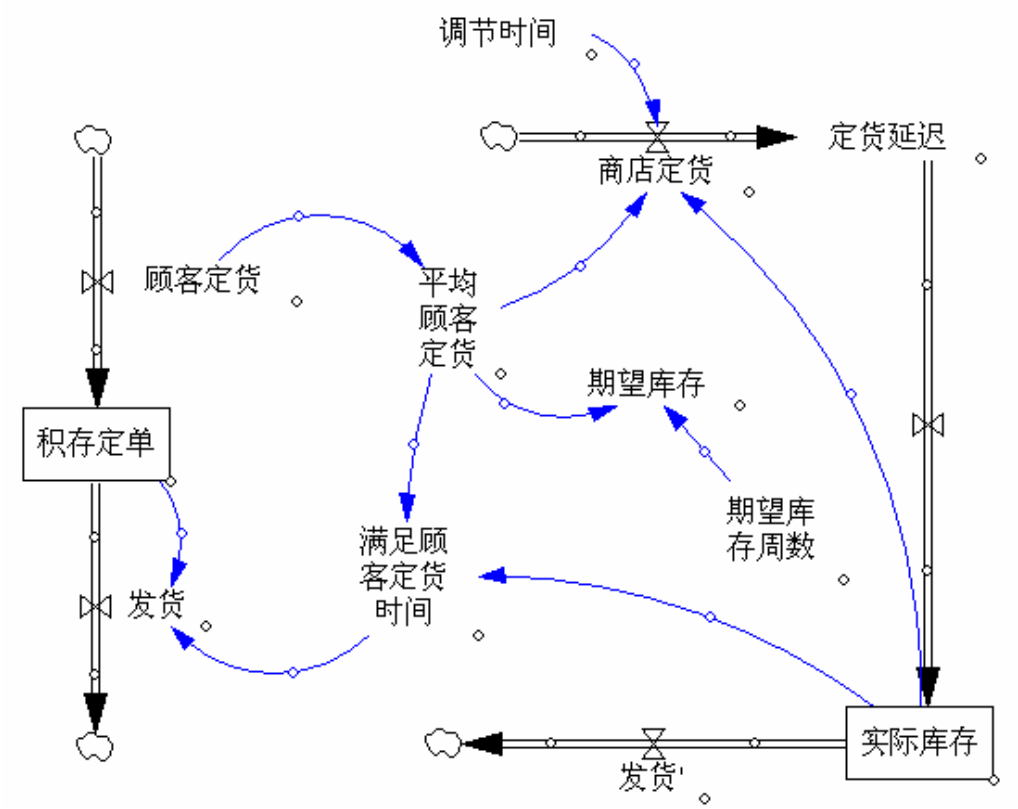
R 商店定货.KL=平均顾客定货.K+(期望库存.K-实际库存.K)/调节时间

A 期望库存.K=平均顾客定货.K*期望库存周数

C 期望库存周数＝8

C 调节时间＝4

（3）画出总的系统流图并写出对应的 DYNAMO 方程



L 积存定单.K=积存定单.J+DT*(顾客定货.JK-发货.JK)

N 积存定单=满足顾客定货时间*顾客定货

R 发货.KL=积存定单.K/满足顾客定货时间.K

A 满足顾客定货时间.K=TABLE(TDFR,X.K,4,12,4)

A X.K=实际库存.K/平均顾客定货.K

T TDFR=1.5,1,0.75

A 平均顾客定货.K=SMOOTH(顾客定货.JK,平均时间)

C 平均时间=8

A 期望库存.K=库存周数*平均顾客定货.K

C 库存周数=8

R 定货延迟.KL=DELAY3(商店定货.JK,定货延迟时间)

R 商店定货.KL=平均顾客定货.K+(期望库存.K-实际库存.K)/调节时间

C 调节时间=4

C 定货延迟时间=6

L 实际库存.K=实际库存.J+DT*(定货延迟.JK-发货.JK)

N 实际库存=期望库存

R 顾客定货.KL=正常值+STEP(修正值,5)

C 正常值=1000

C 修正值=100

3. 注意问题

（1）注意区分何时用一阶延迟，何时用三阶延迟。

（2）建模前要确定系统中各种量的单位，同一种物品单位要一致。

4. 思考题

试归纳用系统动力学思想建模的一般思路和步骤。

实验十二 牛鞭效应模型

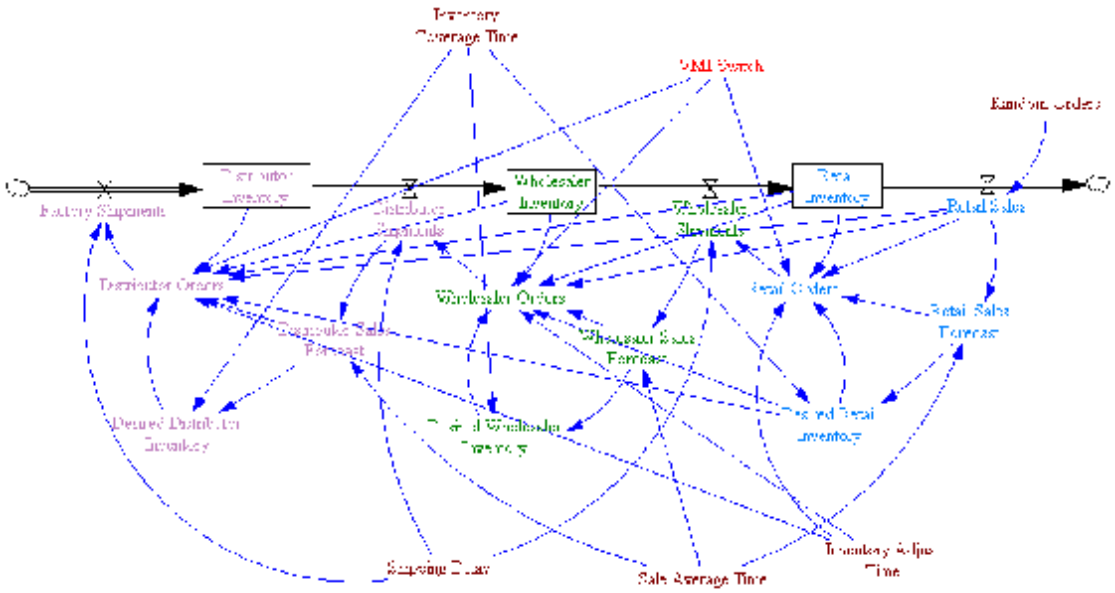
1. 实验目的

供应链的信息流从末端（最终客户）向源端（原始生产商）传递时，需求信息的波动会越来越大，这种信息扭曲的放大作用在图形上很像一条甩起来的牛鞭，因此被形象地称为牛鞭效应。本实验的目的在于 1)用计算机模拟分析研究供应链中的牛鞭效应问题；2）验证消除牛鞭效应的策略；3）学习开关变量的使用。

2. 实验步骤

（1）分析信息流动的过程，找出相关的元素和存量。

（2）分析经销商、批发商和零售商以及定货速度、销售速度和库存等之间的因果关系，画动力学流图。



（3）分析数量关系，写方程。

这是一个三阶段的供应链，考察零售商（Retailer）、批发商（Wholesaler）和分销商（Distributor）的库存与订单情况，其中常量 Inventory Coverage Time 是指各个企业应该持有多少周的库存，Shipping Delay 表示下游企业从发出订单到接收上游企业的订货需要多长时间，Sale Average Time 给出了各个企业在进行订单预测时从多少周以前的销售情况考察起，Inventory Adjust Time 则表示企业调整库存所需要的时间，我们在模型中将 Inventory Coverage Time 和 Shipping Delay 设为 3，Sale Average Time 设为 6，Inventory Adjust Time 设为 4。

Random Orders 和 VMI Switch 是布尔常量，Random Orders 为 1 表示客户的订单随机生成，否则订货量前 4 周为 1000，之后增加到 1400 并保持不变了，所以有：

Retail Sales = 1000 + IF THEN ELSE (Random Order = 1, (RANDOM NORMAL (-200, 200, 0, 50, 4)), 400)

VMI Switch 为 1 表示使用 VMI 策略（VMI 策略的具体内容将在本文的第三部分论述），否则不使用 VMI 策略。

零售商（Retailer）、批发商（Wholesaler）和分销商（Distributor）使用的库存以及销售预测策略均是相同的，所以我们仅仅来看其中批发商模型的情况：

批发商的出货速率是一个三阶信息延迟，由零售商的订单及刚才我们定义的 Shipping Delay 决定：

Wholesaler Shipments= SMOOTH3(Retail Orders, Shipping Delay)

程序刚开始运行时，批发商的初始库存 Wholesaler Inventory 为 Inventory Coverage Time*1000=3000，以后的库存按以下公式计算：

Wholesaler Inventory=Distributor Shipments-Wholesaler Shipments

批发商使用指数平滑来预测销售量：

Wholesaler Sales Forecast=SMOOTH (Wholesaler Shipments, Sale Average Time)

期望库存等于销售预测和库存持有时间的乘积：

Desired Wholesaler Inventory=Wholesaler Sales Forecast*Inventory Coverage Time

订单量在不使用 VMI 策略时按照以下公式计算：

Wholesaler Orders= Wholesaler Sales Forecast+(Desired Wholesaler Inventory-Wholesaler Inventory)/Inventory Adjust Time

该系统动力学模型中，零售商和分销商的各个存量及速率的计算公式与批发商完全相同，在这里便不再赘述。

（4）用 VENSIM 模拟该实验，分析结果。

首先设定顾客的需求在第四周发生一次变化，即订单量提高 40%，然后一直保持不变，共模拟 100 周的运行情况，来看看会产生什么样的结果

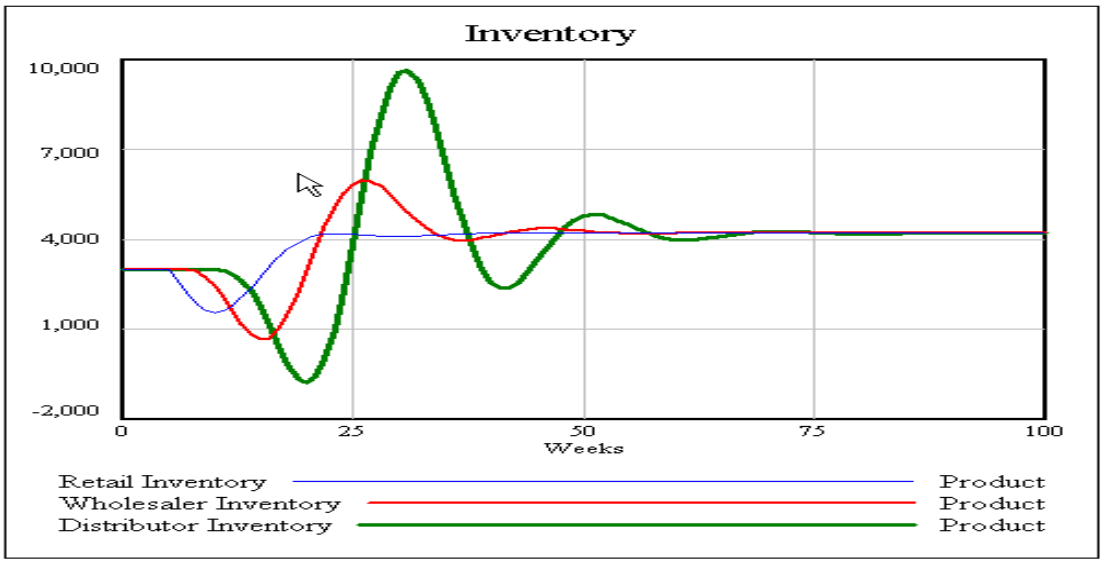


图 1

图 2

图 1 表示的是零售商（兰色曲线）、批发商（红色曲线）和分销商（绿色曲线）在这 100 周里库存的变动情况，图 2 表示的是他们向上游企业发出的订单的变动情况，不难发现，零售商库存与订单量的波动还不算太大，到了批发商则有了比较大的波动,等这种波动传到分销商时，牛鞭效应已表现得淋漓尽致。需要注意的是，客户的需求在这 100 周的时间里只变动了一次，其他时间都维持固定水平，可这小小的波动竟然被上游企业放大到如此程度。

再来考虑让顾客的需求完全随机变化的情况，看看处在上游的零售商、批发商和分销商将遇到怎样严重的库存与订单危机，这次模拟的时间同样设置为 100 周：

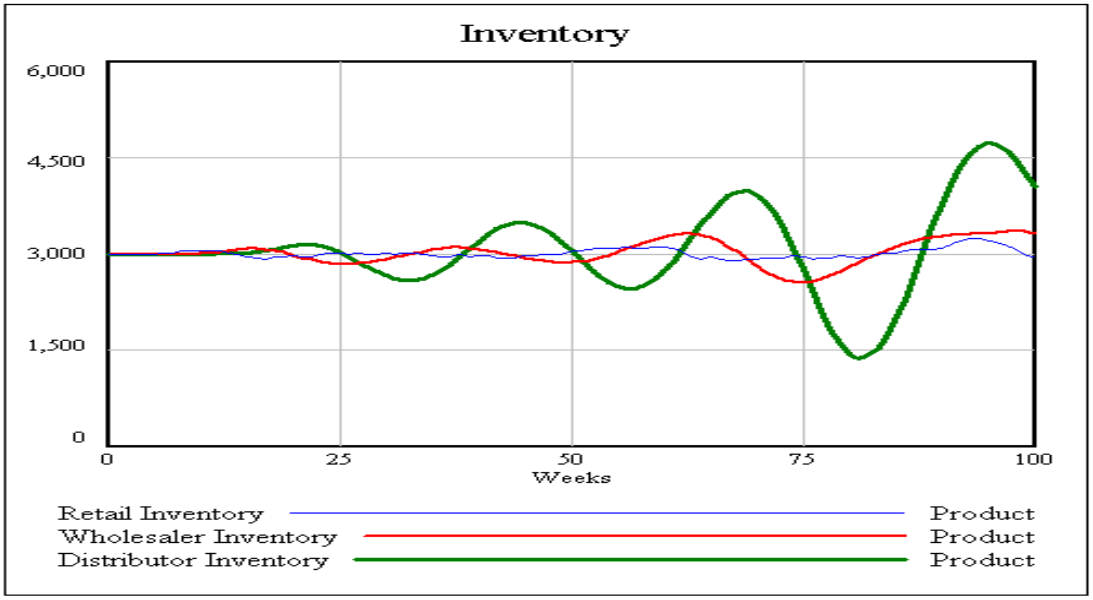


图 3

图 4

如果顾客的需求真的完全随机变化，分销商的库存与订单波动将大到无法忍受的程度。它上游的制造商更会深受其苦，为了应付这种巨大的信息扭曲而不得不维持一个相当高的库存水平，否则下游客户的满意度将大大降低——而这将付出巨额成本。

（4）模拟采用 VMI 策略后的供应链。

若使用 VMI 策略，则零售商（Retailer）、批发商（Wholesaler）和分销商（Distributor）的订单量分别由以下公式决定：

Retail Orders= Retail Sales+(Desired Retail Inventory-Retail Inventory)/Inventory Adjust Time

Wholesaler Orders= Retail Sales+(Desired Retail Inventory*2-Retail Inventory-Wholesaler Inventory)/Inventory Adjust Time

Distributor Orders= Retail Sales+(Desired Retail Inventory*3-Retail Inventory-Wholesaler Inventory-Distributor Inventory)/Inventory Adjust Time

即零售商将自己的销售情况与上游的批发商和分销商共享，同时下游企业均将自己的库存信息告诉上游。

使用 VMI 策略后（即布尔常量 VMI Switch 置为 1），计算机模拟的结果如下：

首先仍然设定顾客的需求在第四周发生一次变化，即订单量提高 40%，然后一直保持不变，共模拟 100 周的运行情况，如图 5 和图 6：

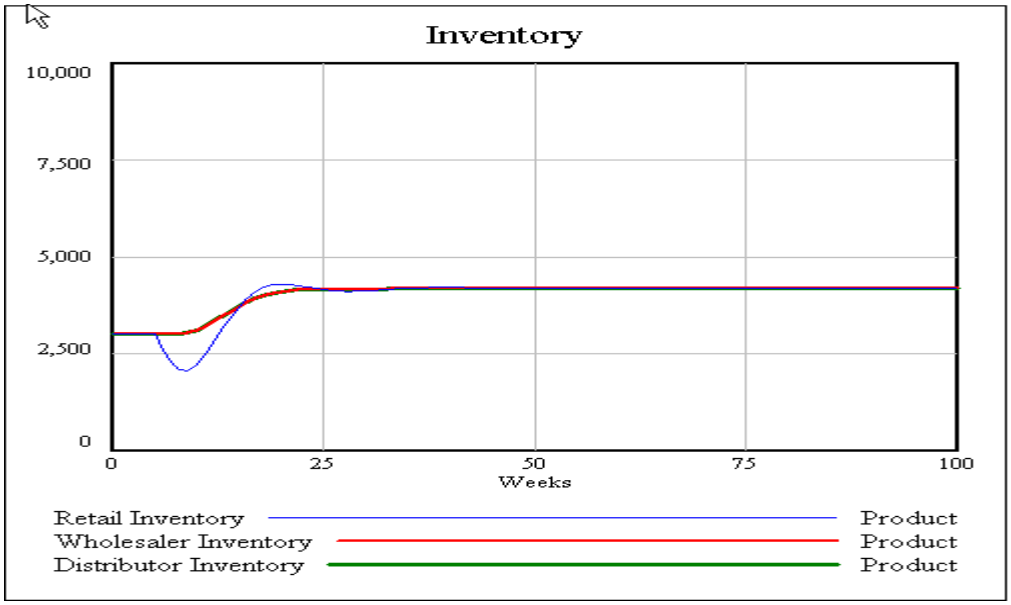


图 5

图 6

图 5 表示的是零售商（兰色曲线）、批发商（红色曲线）和分销商（绿色曲线）在这 100 周里库存的变动情况，而图 6 则表示了他们向上游企业发出的订单的变动情况，与前面的图 1、图 2 比起来，我们很容易发现，牛鞭效应大大降低了。假如客户需求完全随机变动，情况是否仍然乐观呢？请看图 7 和图 8

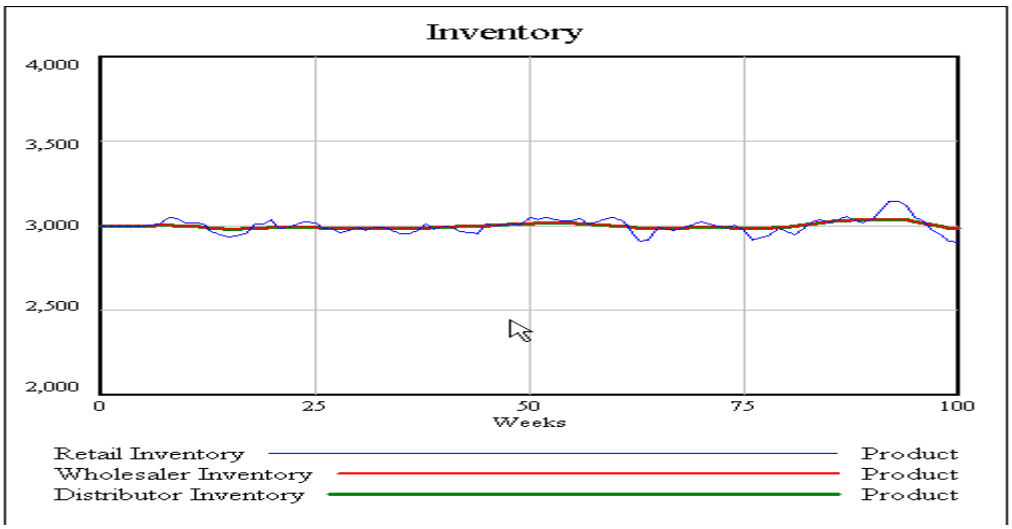


图 7

图 8

同样，图 7 表示的是零售商（兰色曲线）、批发商（红色曲线）和分销商（绿色曲线）在这 100 周里库存的变动情况，而图 8 则表示了他们向上游企业发出的订单的变动情况，和前面的图 3、图 4 相比，我们仍然可以欣喜地发现，即使在客户需求完全随机变化的糟糕情况下，VMI 策略还是可以有效地降低牛鞭效应，使供应链上的各个企业均从中获利。

3. 思考题

（1）为什么 VMI 策略可以消除牛鞭效应？

（2） 如何设计可是使得用一个模型来方便的模拟传统策略和 VMI 策略，而不是构建两个模型？

《基于主体建模》
实验教学大纲

课程名称：基于主体建模

英文名称：Agent Based Modeling

课程代码：

课程性质：必修

实验指导书名称：基于主体建模实验教程

适用专业：系统科学、管理科学与工程

一.实验总学时（课外学时/课内学时）：40　　总学分：2

必开实验个数：10　　选开实验个数：10

二.实验的地位、作用和目的

“基于主体建模”是系统科学专业和管理科学与工程专业硕士研究生的必修课《计算机建模与工具》的三大板块之一，是研究分散系统运行机制的一类建模方法。

“基于主体建模”实验所用的主要工具是美国麻省理工大学多媒体实验室开发的 Starlogo。Starlogo 是一个可编程的建模环境，用来探索分散系统的工作原理。分散系统是这样一类系统，它们不需要组织者而能自我组织，不需要协调者而能自我协调。

通过对本实验课程的学习，学生可以进一步加深对计算机建模理论和方法的理解，熟练掌握 Starlogo 这一重要的建模工具，能够运用基于主体的建模方法分析和解决现实问题。在对复杂分散系统研究建模的过程中，培养学生分析问题、提炼模型、解决问题的能力，训练学生的抽象思维，加强学生对计算机建模工具的掌握，为进一步从事科学研究打下坚实的基础。

三.基本原理及课程简介

计算机建模的基本思想是对现实世界进行抽象，剔除与主题无关的次要因素，抓住主要因素，提炼规则，用计算机来模拟主体在现实世界中依据一定规则运动的情况，观察结果，总结规律。

基于主体建模是计算机建模的一个重要组成部分，它所研究的分散系统是一类没有组织者和协调者，而系统整体却呈现出有组织的协调的形态的系统。在现实世界中存在很多的分散系统，例如：鸟群、蚁群、交通运输以及市场经济。之所以说它们是分散系统是因为在这些系统中，不存在集中的控制，每一个个体都是按照一定的规则在运转，而整体却呈现出有序模式。随着人们对分散系统的认知程度的逐渐加深，越来越多的研究者选用分散模式来创立组织机构、构造科学技术、甚至是架构关于世界的理论基础。

实验所用的建模工具 Starlogo 采用基于主体的建模方法。“主体”是一只只的海龟（turtles），我们可以并行地控制数千只海龟，同时也可以为它们制定不同的行为模式。而这些主体所处的“环境”则是用点（patch）来表现的，数千个点拼成一块大的背景（canvas），代表着主体所处的大环境系统。

StarLogo 允许对海龟和点进行编程，这使得海龟和海龟所处的环境都具有了自己的变化方式。海龟和点之间是可以彼此交互的，例如：我们可以编程让海龟在它的世界里到处“闻”，根据在它所在的点上闻到的“气味”来决定它的行为。海龟和海龟之间也可以交互，例如：让海龟们朝着一个方向前进，如果前面的海龟离自己很近，就走慢一点，以免撞到，否则，就走快一点追上。在这里，海龟和点之间的交互作用体现了主体与环境的关系，海龟与海龟之间的相互影响则体现了主体之间的关系。

四.实验基本要求

1、能够从计算机可以接受的概念模式入手，把实际问题转化为计算机可以识别的形式。

2、了解计算机建模的基本原理，掌握基于主体建模的研究对象、研究特点及建模方法。

3、熟悉 Starlogo 建模环境，掌握 Starlogo 建模语言，能够运用 Starlogo 对实际问题建模。

4、在基础性实验中，能够完成对 Starlogo 某一功能特性的实

验要求，掌握基本问题的建模思路、建模方法和建模过程。

5、在综合性实验中，能够独立完成对一个复杂问题的系统分析、建模过程及编码实现，对模拟过程中出现的问题能够独立排除。

6、能够根据模拟结果，对所提出的问题进行初步分析。

7、撰写简明扼要、图表清晰，结论正确、分析科学的实验报告。

五．实验具体要求

基于主体建模实验采用三段式教学模式，教学过程由理论知识准备、基础性实验及综合性实验三个环节组成。

（一）理论知识讲座

通过教师讲解，使学生了解基于主体建模的基本原理、建模方法和建模过程，初步掌握实验建模工具 Starlogo 的界面操作、变量设定、流程控制等基本特性。了解本实验课程的性质、任务、要求、课程进度安排、实验考核内容及实验报告要求等，为进行实验做好准备。理论知识准备时间为 1—2 次课。

（二）基础性实验

此阶段包括 10 个实验的操作训练，这是本实验课程的中心环节。要求做好：

1.预习

学生须认真阅读实验指导书，了解实验的目的和原理，明确本次实验中需要了解和掌握的 Starlogo 建模工具的基本特性和功能，在此基础上写出实验预习报告，内容包括：实验目的和基本原理，测试模块的特性说明，简单的实验步骤。

2．实验

学生进入实验室后首先熟悉 Starlogo 的运行环境。教师检查学生的预习情况并做好记录，包括预习报告和对实验的理解，不合格者不得进行实验。

指导教师讲解实验难点和注意事项，通过提问的方式引导学生深入思考与实验现象有关的一些问题，着力培养学生观察实验、综合考虑问题的能力，使学生学会分析和研究问题的方法。学生独立或按学号编组进行实验，注意实验中的独立操作和相互配合。要求学生在实验中勤于动手，敏锐观察，细心操作，开动脑筋，分析钻研问题，熟练掌握 Starlogo 的功能特性及编程要点。实验结束后经教师检查并签名，实验结果和程序代码才有效。

3．书写实验报告

认真写出实验报告是本课程的基本训练。它将使学生在实验数据处理、作图、误差分析、问题归纳等方面得到训练和提高。实验报告的质量，在很大程度上反映了学生的实际水平和能力。

实验报告的内容大致包括：实验目的和原理、实验测试模块的功能说明、实验结果的展示和分析以及应提交的带注释的源程序代码等。实验报告的讨论可以包括对实验现象的分析和解释、对实验结果的误差分析、对实验的改进意见、心得体会等。

实验结束的两周内完成实验报告，于下次实验时提交报告，其中必须附有教师签名的原始记录表。

（三）综合性实验

综合性实验又分为两种类型：

类型 1：基础性综合实验。开设 5 个与实际应用关系密切、具有一定复杂性和综合程度的实验，学生选做其中的 1 个。该类型实验由教师给出实验题目和要求，学生选择实验题目后需要查找资料，研读文献，钻研有关理论知识，在此基础上选择实验方法，提出实验方案，与指导教师讨论实验方案的可行性，然后预约时间进行实验。

类型 2：研究性综合实验。该类型实验均由教师的科研项目转化而来，目前开发有 5 个实验供学生选择，为实验兴趣浓厚和学有余力的学生提供更多的实验空间，对培养学生的创新意识和科研能力大有裨益。研究性改变了传统实验中固定实验步骤，照方抓药式的被动性，使学生有了很大的自由发挥空间；其次，实验引入了挫折—激励教学机制，允许多次失败，但要求最终一定要成功。

综合实验室每天都对学生开放，学生只要有空，都可以预约时间，到实验室做实验。

六 . 考 核 与 报 告

学生在课下做好预习，写出预习报告，实验前指导教师检查预习情况，根据学生预习报告和回答问题情况给出预习成绩。

实验进行中，教师考察学生的实验操作技能，根据学生实验操作、遵守实验

规则、实验纪律情况以及实验结果展示情况给出实验操作分。

实验结束后学生书写实验报告，并于下一次实验时提交，其中必须附有教师签名的原始记录表。教师根据学生实验报告书写的完整性、实验结果的合理性、对实验的理解和体会等给出报告成绩，包含报告完整、字迹工整(30%)、测试说明正确(10%)、程序代码正确(40%)、实验结果正确(20%)。

期末的最后两次实验作为考核实验，考查学生独立从事实验研究的能力，给出考核成绩。

本实验课程的最后成绩：预习分 20%+操作分 30%+实验报告分 40%+实验考核分 10%

缺少实验或报告者须在实验教学结束前补做和补交，否则不予通过。成绩不合格者下学年重修。

综合实验不安排课内学时，学生利用课外时间到开放实验室进行实验。实验之前写出完整的预习报告，详细列出实验目的、依据的原理、实验步骤、测试说明。实验结束后，将实验结果整理成小论文的形式提交。教师根据学生实践能力、实验水平、实验结果的创造性大小来评定实验成绩。

七．实验指导书（实验教材）

1.《计算机模拟》 方美琪 中国人民大学出版社

2. http://education.mit.edu/starlogo/

八.实验项目与内容提要

序号	实验项目名称	内容提要	实验学时	每组人数	实验类型	实验要求
1	Starlogo 中的 Turtle 控制	1. 学习使用命令创建和消除 Turtle 2. 学习设置 Turtle 属性、状态和动作 3. 了解各种返回值命令	2	1	基础	必做
2	Starlogo 中的 Turtle 计数问题	1. 了解 Starlogo 中求 Turtle 的总和、平均值、最大值、最小值等命令 2. 学习求解有条件约束的计数问题	2	1	基础	必做
3	Starlogo	1. 了解基本运算符的使用方法	2	1	基	必

	中的数学函数运算	2. 学习三角函数的使用方法 3. 掌握函数运算方法			础	做
4	Starlogo中的流程控制	1. 了解Starlogo中流程控制的概念、特点及应用 2. 学习流程控制的结构 3. 熟练掌握各种典型的流程控制语句	2	1	基础	必做
5	Starlogo中的运动与位置指令	1. 了解Starlogo中运动与位置指令的概念和分类 2. 熟练掌握各类指令的内容和使用方法	2	1	基础	必做
6	Starlogo中的监视器功能	1. 熟练操纵turtles的运动过程 2. 了解patches赋值 3. 掌握监视器的性质和功能	2	1	基础	必做
7	Starlogo中的Turtle与环境的交互	1. 了解如何控制turtles的状态变化 2. 学习turtles如何与环境进行交互	2	1	基础	必做
8	Starlogo中的条件判断语句	1. 学习turtle和patch之间值传递的过程 2. 学习使用条件判断语句	2	1	基础	必做
9	Starlogo中的概率运算过程	1. 学习turtle之间状态的交互影响 2. 了解模型中概率的使用方法	2	1	基础	必做
10	Starlogo中的阈值概念及使用方法	1. 了解阈值的概念和使用方法 2. 掌握距离计算的一般公式	2	1	基础	必做
11	Traffic	1. 学习设置和控制turtle和patch的多重属性 2. 掌握复杂的条件控制语句	2	3	综合	选做
12	Painted Turtles	1. 学习如何用Starlogo画图 2. 熟练掌握hatch语句的使用方法 3. 了解有关颜色的常识问题	2	3	综合	选做
13	Yellow Brick Road	1. 学习turtle的颜色识别功能 2. 掌握case语句的使用方法	2	3	综合	选做
14	Predator Prey	1. 学习控制系统时间来模拟生命周期	2	3	综合	选做

	Grass	2. 熟练掌握循环语句的使用方法				
15	Enzyme	1. 学习多条件控制下的模拟过程 2. 学习模型抽象和类比的功能	2	3	综合	选做
16	自行车失窃模型	1. 研究自行车失窃问题的原因 2. 研究自行车失窃问题的规律 3. 找出解决自行车失窃问题的方法	2	5	综合	选做
17	信息发布利弊分析模型	1. 了解大系统的脆弱性 2. 分析信息发布的利弊 3. 权衡利弊，决定是否发布信息	2	5	综合	选做
18	行业竞争模型	1. 熟悉 EC-Game 虚拟社区 2. 了解电子商务 B-B 模式 3. 建立消费者、生产商、供应商和销售商之间的行业竞争模型	2	5	综合	选做
19	银行排队模拟系统	1. 了解目前银行排队现状 2. 分析银行排队情现状的产生原因及影响其结果的关键因素 3. 建立模拟系统解决银行排队问题	2	5	综合	选做
20	信息搜寻与选择模型	1. 了解价格离散的概念、原因和意义 2. 了解信息搜寻与选择的原理 3. 建立信息搜寻与选择模型	2	5	综合	选做

Swarm 实验指导书

实验一　Swarm 开发环境的配置

一、实验目的

随着国内越来越多的研究者开始接受复杂性思维和多主体建模的研究方法，应用多主体仿真建模平台 Swarm 的研究也越来越多。在实践中，很多初学者或是非信息相关专业的研究者经常对 Swarm 的系统构成、原理、安装和配置等感到困惑，本实验针对这种情况给出一些问题的解决方法。

二、实验内容

在 Windows 平台上配置 Swarm 的开发和运行环境

三、实验仪器、设备及材料

硬件：

IBM-PC 兼容机

主要参数：Intel Celeron 2.0G 256DDR

软件：

Windows 2000/XP　Swarm 2.1.1　Swarm 2.2

Swarm 2.1.1 版可以在 Windows 95/98/Me/NT/2000/XP 中运行，满足操作系统硬件需求即可安装，但 Swarm 运行时构建 Cygwin 和 Java 虚拟机环境会消耗大量系统资源，为了达到更好的运行效果，高配置的计算机系统还是很有必要的。根据经验，计算机配置 CPU 最好在 Pentium III 以上，内存至少 256M，如果同时需要运行 IDE（集成开发环境，Integrated Develop Environment，IDE），则内存需要更多。

四、实验原理

Swarm 建模环境的架构有多种选择。Swarm 库本身是在 Unix 系统中用 Object-C 语言开发的，所以在 Unix/Linux 下用 Object-C 设计模型是最为直接的一种方式，这样一个建模环境的架构最为简单，一般 Unix/Linux 的缺省安装都已经提供了 Object-C 编译器，可以下载 Swarm 的源程序包，在自己的系统上编译生成库，然后用同样的方式设计模型程序。这种方式的优点在于，可以深入了解 Swarm 的内核，必要时甚至可以修改库中的代码，由于模型与库之间的联接是无缝，可以生成一个独立发布的可执行程序，模型执行效率高，模型的

规模只受内存和 Unix/Linux 对进程的配额限制。大约每 128M 可支持 1 万个 Swarm 主体，可以参考这个数字选择配置系统的内存。

对熟悉 Java 语言的模型设计者来说，如果仍然在 Unix/Linux 中设计模型，情形则稍有不同。Swarm 基本库中有一个为 Java 用户准备的 Java 本地化接口（JNI） 库 swarm.jar，模型设计者只需要通过这个接口来调用 Swarm 库中的各种基本构件。由于最终还是用到了平台有关的本地化库的代码，所以采用 Java 设计模型并不能享有 Java 的各种优良特性，这种选择唯一的好处是，对 Java 用户来说掌握模型的设计更容易了，代价是模型的规模受 Java 虚拟机的内存限制。在设计一个主体数量非常多的大规模的模型时，如果没有超越机器的内存限制而受限于 Java 虚拟机，可以重新配置 Java 虚拟机增加它的内存配额。

对于 Windows 下的用户来说，情形则更复杂一些：必须在 Windows 之上安装一个 Linux 模拟环境 Cygwin，才能调用原来在 Unix 下开发的 Swarm 库。在 Swarm-2.1.1 版中，提供了一个 Windows 下的安装包，其中集成了 Cygwin，用户就无须自己再安装 Cygwin。在 Windows 程序组中有一个 Terminal 的选项，打开它就是一个 Cygwin 的 Bash 环境，可以使用 Linux 下的常用命令，还可以用命令行的方式使用集成的 Object C 编译器。如果下载了没有集成 Cygwin 的版本，自行安装 Cygwin 时，需要注意的是除了 Cygwin 缺省安装外，还需要安装一些图形显示的库，如 Tcl/Tk 语言库和 Blt24 库等，这些库是 Swarm 图形界面所必须的。

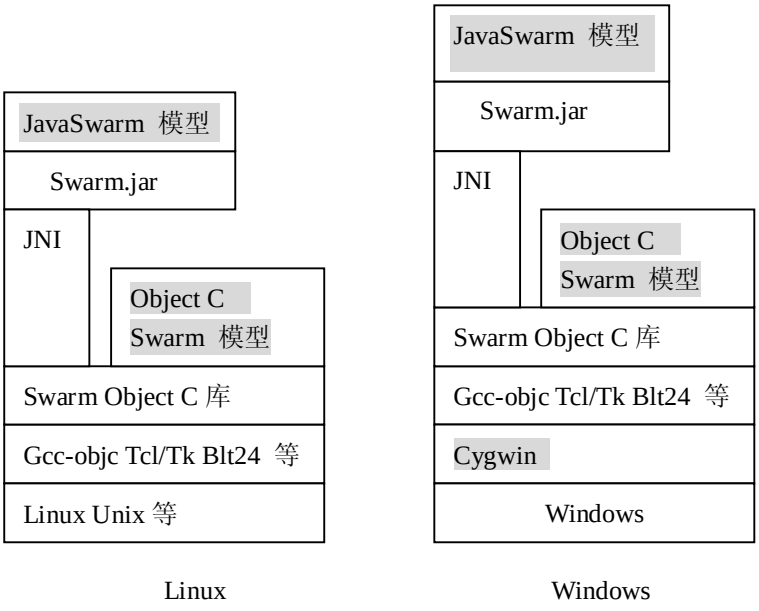


图 1　Swarm 的体系结构

在 Windows 下开发模型时，模型的规模要受 Cygwin 的内存限制，因为 Cygwin 也相当于在 Windows 中的虚拟机，缺省的内存配额只有 128M。

从上面的分析看出，用 Java 语言和在 Winodws 之上用 Cygwin 设计模型都会对模型的规模造成一定的限制，而且，由于 Java 和 Cygwin 都不是直接调用原来 Swarm 基本库的二进制代码，所以在模型效率上也会有很多损失。如果希望在 Windows 系统上用 java 设计模型，从模型的角度来说，因为是在 Cygwin 虚拟机中的 Java 虚拟机内运行，所以应该是最坏的组合，然而，对模型设计者来说，容易学习更重要，所以仍然是最常选择的模型设计环境。而另外一方面，这种两层嵌套机制给模型设计者带来了许多迷惑，很多初学者常常困扰于环境的配置而不能入门。最好的选择依然是 Swarm 2.1.1 的 Windows 安装包，事实上，这个安装包不仅仅集成了 Cygwin，也自带了一个 Cygwin 下的 Java：Kaffe Java，模型设计者可以在这个 Cygwin 下直接用 Javacswarm 脚本编译模型，用 Javaswarm 脚本运行模型。

下面我们以 Swarm 2.1.1 版为例分步骤介绍 Windows 平台下 Swarm 的安装和配置。

五、实验步骤

以下过程以中文版 Windows XP 操作系统为例，假设在 C:\model 目录中存在编写好的模型示例程序。

（一） 安装 Swarm 2.1.1

在“我的电脑”中找到安装文件 swarm-2.1.1.exe，双击，出现下面的窗口：

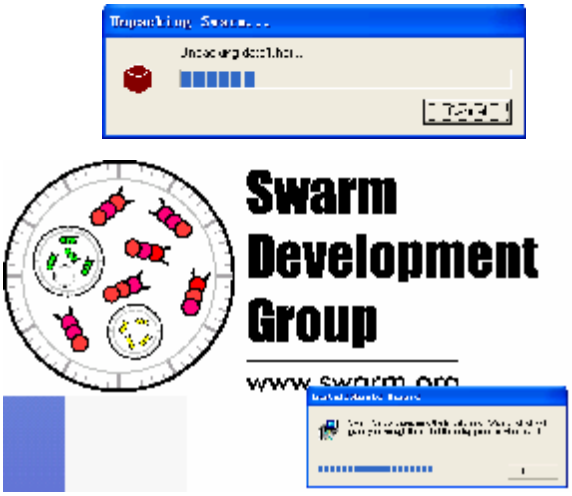


图 2

点击“下一步”，是版权信息，继续。

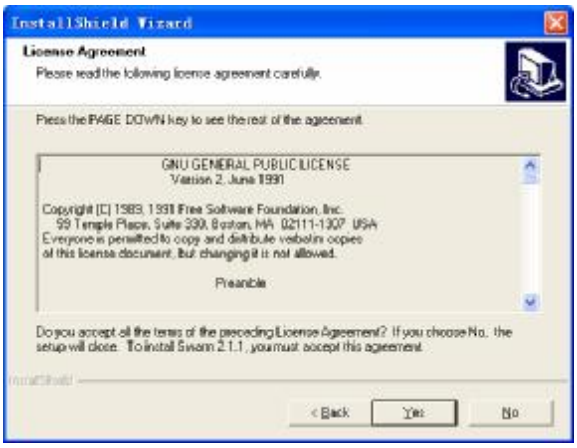


图 3

选择安装的位置，默认是“C:\Swarm-2.1.1”。如果所给的目录不存在，安装程序将自动创建这个目录。

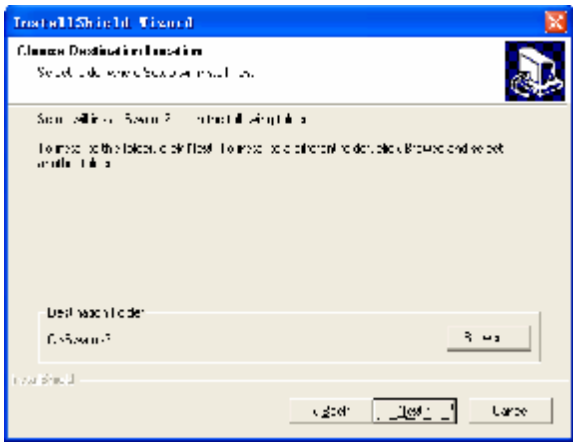


图 4

再次点击下一步，安装程序开始复制文件，稍后安装完成。

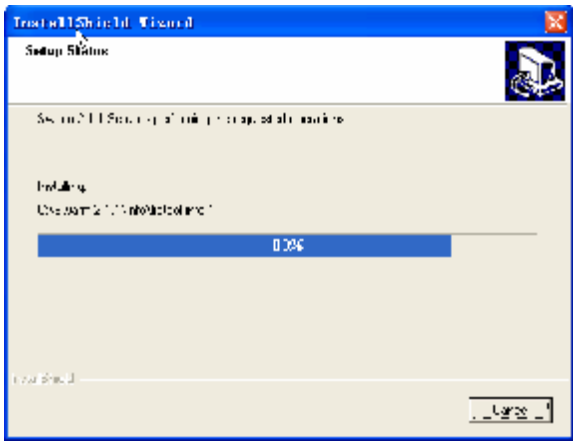


图 5

安装完毕后，系统环境变量将在“用户变量”中将增加“SWARMDIR”一条，内容是 Swarm 安装的目录。

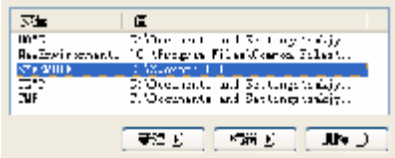


图 6

Swarm 2.1.1 的 Windows 安装包完整地提供了建构环境的必备软件和库；包括一个集成的 Cygwin、必须的 tcl/tk80、blt24 库等，方便 Object C 编写模型的 Swarm 库，方便 Java 编写模型的 javaswarm.dll 和 swarm.jar，以及 Object C 编译器等。

（二） 在 Cygwin 环境下使用 Swarm

为了方便 Java 模型开发者调试，Swarm 2.1.1 的 Windows 版安装后，在安装路径内还包括一个集成在一起的 Java 虚拟机。为方便配置 Java Swarm 环境，还提供了两个脚本文件：javaswarm（执行）与 javacswarm（编译）。这两个文件以批处理的方式设置了必要的系统参数和路径，如：java class 库路径、jre 路径、必要的 dll 路径等；javaswarm 启动自带的 Javaw.exe 解释执行作为参数传入的 Java 程序。这样用户就可以再编译好 Java 代码的模型后，到 Cygwin 的 bash 下用脚本文件 javaswarm 执行模型。典型的步骤是：

在“开始”→“所有程序”（Windows2000 及以前版本中是“程序”）→“Swarm”中点击“terminal”，打开 Terminal 窗口，这实际上是 Cygwin 的一个 bash 环境，如果可能的话，也可以用自己安装的 Cygwin 替代。

图 7

在窗口中输入：

cd c:

cd model

以“javacswarm *.java”编译程序，如果没有错误，会在 C:\model 目录中生成一系列文

件，其中主要是类文件（*.class）；否则会显示错误信息。

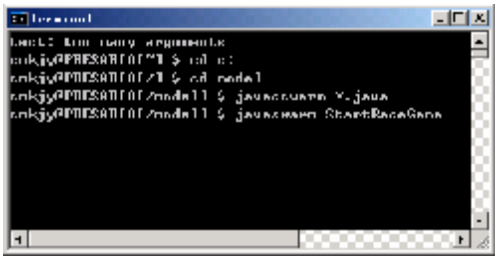


图 8

经过调试编译无误后，请在桌面右键→“属性”→“主题”中将当前主题改为“Windows 经典”，因为图形界面接口与 Windows XP 的默认风格有冲突，在 Windows XP 下一些模型程序可能无法运行。如图所示：

图 9

用命令“javaswarm StartRaceGame”运行包含 main 方法的主类，这时会出现程序面板窗口。此处可以修改模型的各种参数，点击“Start”，模型开始运转。

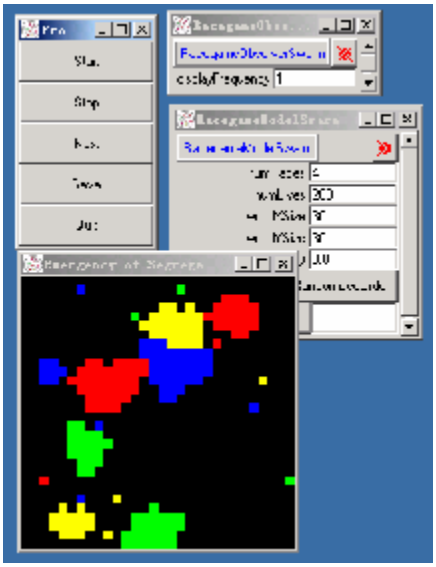


图 10

运行结束后，在面板上点击“Quit”结束程序，在 terminal 窗口输入“exit”退出。

（三） 在 Windows 命令行中使用 Swarm

在 Windows 命令行中使用 Swarm 首先需要安装一个 JDK，2.1.1 版本的 Swarm 发布时 JDK 的版本为 1.2，所以从兼容性角度来说最好用此版本的 JDK 进行编译，但模型可以正常运行在高版本的环境中（安装过程略），我们假设其安装路径为 C:\j2sdk1.4.2。

下面修改一个环境变量，具体方法是：右键单击“我的电脑”→“属性”→“高级”→“环境变量”，在“系统变量”中寻找“Path”一项，把它的内容中加入 Swarm 中 bin 目录的路径、以及 JDK 的 bin 目录路径，这里就是添加“C:\Swarm-2.1.1”和“C:\j2sdk1.4.2\bin”，路径之间用分号分隔；如果没有找到“Path”，需要手动建立这个变量。

图 11

在“开始”→“所有程序”（Windows2000 及以前版本中是“程序”）→“附件”中点击“命令提示符”。或者在“开始”→“运行”中输入“cmd”，确认后也可以打开“命令提示符”。

在窗口中输入：

c:

cd\model

编译程序：

javac -classpath .;C:\swarm-2.1.1\share\swarm\swarm.jar *.java

运行程序

java -classpath .;C:\swarm-2.1.1\share\swarm\swarm.jar StartRaceGame

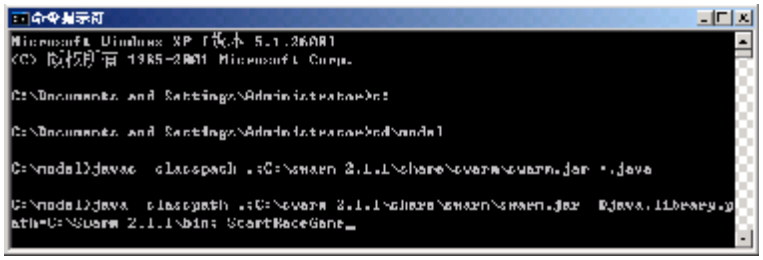


图 12

（四） 在集成开发环境（IDE）中使用 Swarm

Swarm 支持 SUN JDK 1.2 或以上版本。因此，常用的 IDE 如 Jbuilder，Visual Café，Eclipse 等都可以用来编写和调试基于 Java 的 Swarm 程序。我们以 Eclipse 3.0 为例，在假设用户已经建立了一个工作空间（Workspace）和工程（Project）swarm_test 的情况下，介绍与 Swarm 有关的配置方法。关于 Eclipse 安装与使用的一般操作，请参考相关书籍和网上资源。

首先，在工程中建立描述模型的程序文件（*.java），或者导入（Import）现有的模型。此时程序中对 Swarm 类库的引用并没有在开发环境中体现，运行编译时会出现“XXX cannot be resolved”的错误信息。

选择当前的工程 swarm_test，在工程（Project）菜单选择特性（Properties），出现 swarm_test 特性（Properties for swarm_test）对话框。

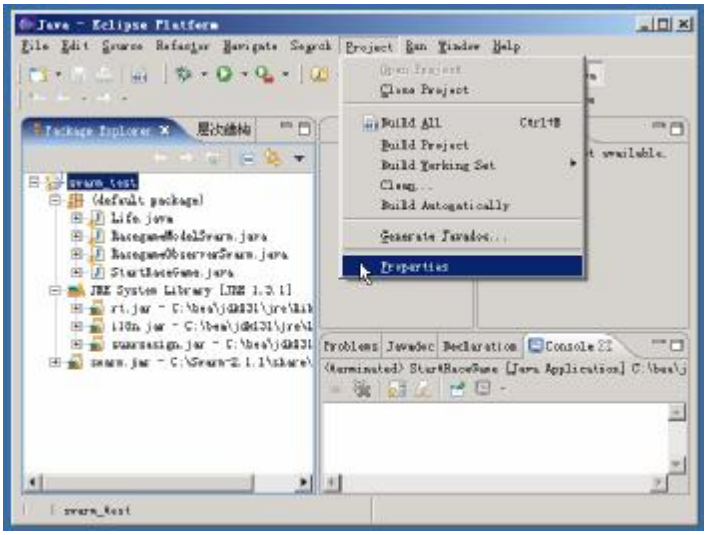


图 13

在左边选择设置 Java 构建路径（Java Build Path），右边选择库（Libraries），用添加外部 JARs（Add External JARs）按钮把 swarm.jar 添加到工程中，按前面所述的默认安装方法，此文件应该在 C:\Swarm-2.1.1\Share\swarm\目录中。

如果希望更改缺省 Java 运行库，可以点击 JRE 库（JRE System Library）按钮选择，推荐选择与 Swarm 兼容性最佳的 1.2 版（这里我们选择的是 1.3.1 版）。最后保存设置。

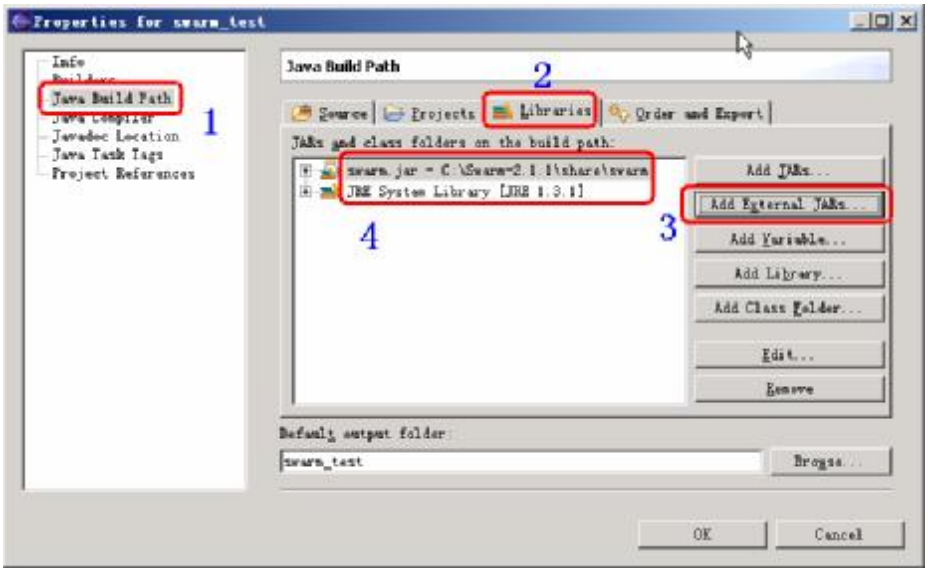


图 14

如果希望在 Eclipse 集成环境中直接运行模型，则需要设置所需的动态链接库路径。在左侧的包资源管理器（Package Explorer）中选中包含 main 方法的主类（这里是 StartRaceGame.java），在运行（Run）菜单选择运行（Run...）打开运行配置对话框。

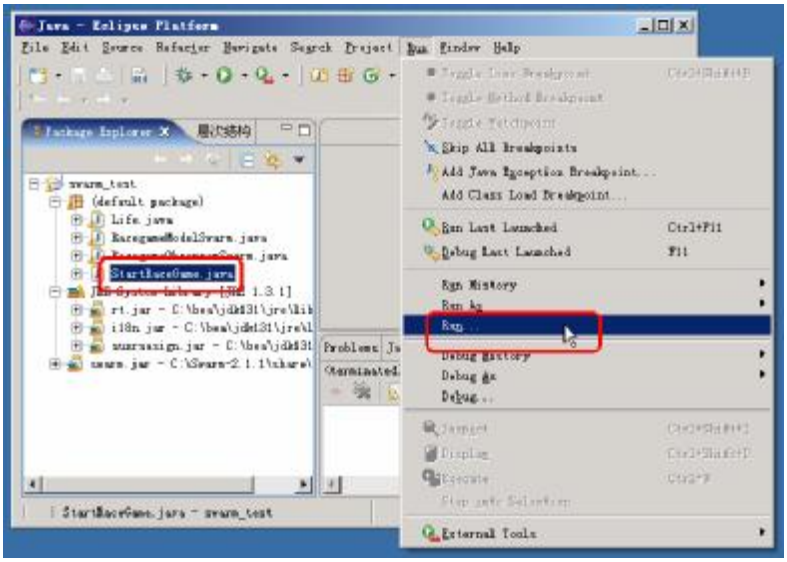


图 15

在右边第二项自变量（Arguments）中的 VM 自变量（VM Arguments）中添加动态链接库路径。按前面所述的默认安装方法，这里需要添加的内容应该是“-Djava.library.path=C:\Swarm-2.1.1\bin;”。这样，点击最下面的运行（Run）按钮，模型就可以正常地运行了。

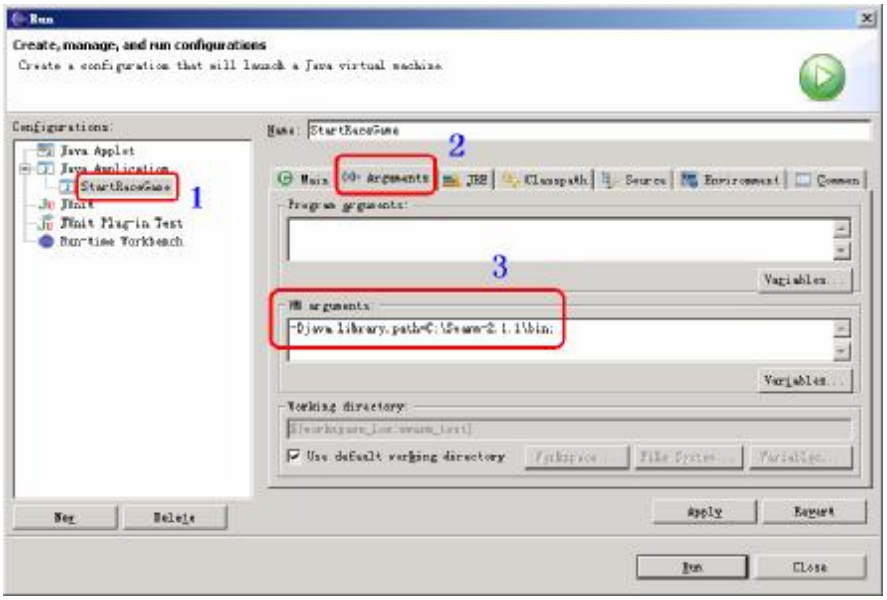


图 16

六、实验报告要求

对实验目的、实验复杂度、依据的原理、操作步骤以及遇到的各种问题进行详细整理。以电子文件（如代码，屏幕截图，PPT，电子表格）的方式提交。

语言描述准确，图示清晰。

七、实验注意事项

a) 非 NT 内核 Windows 中的环境变量配置

在 Windows 95/98/Me 这些非 NT 内核的 Windows 中，系统属性窗口中是无法进行环境变量配置的，这时需要修改自动批处理文件 autoexec.bat，此文件在安装 Windows 的硬盘分区的根目录下（通常是 C:\），修改之前应该去掉可能的隐藏或只读属性。

配置环境变量其实就是将 JDK 和 javaswarm.dll 放到系统的查找路径下。这里，Swarm 安装在 C:\Swarm-2.1.1，那么在 autoexec.bat 文件中加入以下命令行：

SET PATH = C:\j2sdk1.4.2\bin; C:\Swarm-2.1.1\bin;%PATH%

还需要修改 CLASSPATH 变量。在 autoexec.bat 文件中加入以下命令行：

SET CLASSPATH = C:\Swarm-2.1.1\share\swarm\swarm.jar;%CLASSPATH%

以上工作结束后，重新启动 Windows，就可以编译运行 java 写成的 swarm 程序了。

b) 修改虚拟机的内存配额

在设计一个主体数量非常多的大规模模型时，如果内存的使用没有超越机器内存限制而只受限于 Java 虚拟机，可以重新配置 Java 虚拟机增加它的内存配额。具体方法是：

在 Windows 注册表中的下面查找主键“HKEY_CURRENT_USER\Software\Cygnus Solutions\Cygwin\”或“HKEY_LOCAL_MACHINE\Software\Cygnus Solutions\Cygwin\”，在下面创建一个新键“heap_chunk_in_mb”，类型为 DWORD，然后就可以修改这个键值来设置 Cygwin 的内存配额。数值是 16 进制的内存大小，默认单位是 M，如 100 表示 256M。前面两个主键第一个修改只对 Windows 当前用户有效；后一个设置对本机所有用户有效。

八、思考题

- Windows 中如何安装和配置 Swarm 2.2
- 如何在其它 IDE 中配置 Swarm

实验二　简单模拟程序设计——种族聚集

一、实验目的

掌握基于主体模拟程序的设计方法，体会其设计思路

实现对涌现现象的计算机模拟

二、实验内容

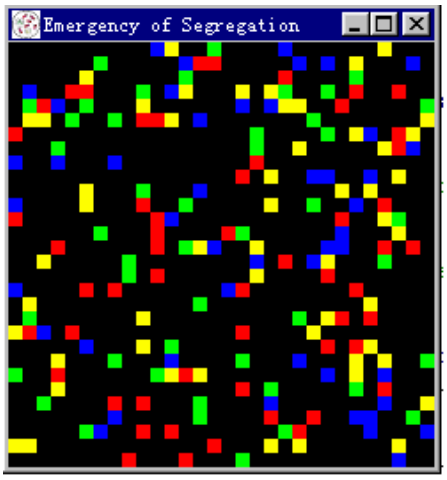
你是什么样的人？你喜欢和相同类型的人一块生活吗？物以类聚，人以群分。相同兴趣爱好的人们总是乐于聚集，在硅世界的生命是否会有同样的行为模式呢？

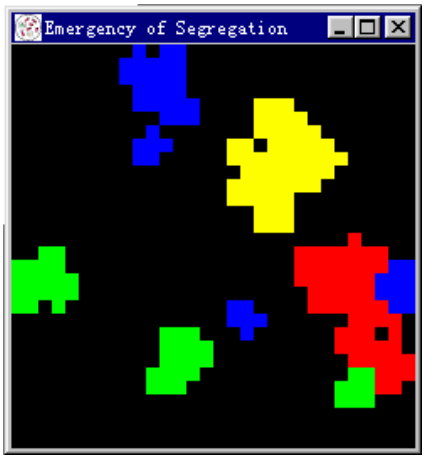
假设在一个硅世界中（以二维格栅图表示）生活的个体，其行为模式十分简单：

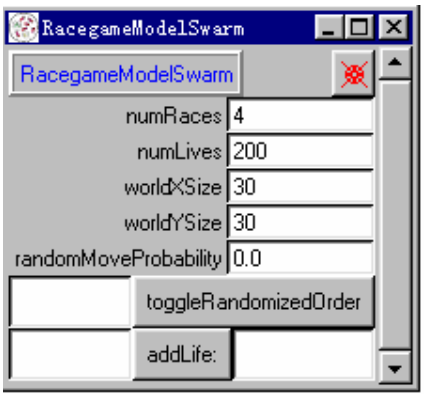
如果周围 8 个邻居中，与自己类型相同的“人”超过 3 个，则维持当前位置不变；

如果不足 3 个，则随机选择一处无人的地方，搬往居住；转 1。

假设有 4 种不同类型的个体，分别使用蓝、绿、红、黄不同的颜色表示，个体总数为 200，“世界”大小为 30 x 30，个体分布如下：







用 Swarm 设计一个简单的程序，要求模拟上述规则，观察部分到总体的涌现现象。

三、实验仪器、设备及材料

由实验一配置好的 Swarm 开发软硬件环境

四、实验原理

在复杂适应系统中，个体根据自身能力局限范围内所能获得的、关于环境和其他个体的信息，调整自己的状态，以其提高个人“效用”，从而在群体层次上展现出一定的宏观规律。种族聚集实验模型，正是这一思想的验证。

五、实验步骤

1. 设计
2. 编写程序代码
3. 调试程序代码
4. 收集并分析实验结果

六、实验报告要求

对实验目的、实验复杂度、依据的原理、程序设计、运行结果分析以及遇到的各种问题进行详细整理。以电子文件（如代码，屏幕截图，PPT，电子表格）的方式提交。

语言描述准确，图示清晰。

七、实验注意事项

本实验以学生自学完成为主，程序代码请参考 Heatbug（热虫模型）

实验三　简单模拟程序设计——牛奶配送

一、实验目的

掌握基于主体模拟程序的设计方法，体会其设计思路

通过演化算法（遗传算法）找出最有效的实现模式

二、实验内容

在现实生活中常常会遇到任务分配的问题。如果一项任务由多人完成，那么如何在这多人之间分配任务是一个非常现实的问题。通常的情况下，由于这些人的自身条件不同，因此完成不同工作的效率也不相同。此时，有必要将任务的不同内容具体分配到不同的执行者，以达到最高的工作效率。

这里我们假设有一个牛奶公司负责一个特定社区内居民的牛奶供应。该公司拥有若干送牛奶的工人，分别居住在社区内不同的位置，他们负责将牛奶发送到居民家中。每天，牛奶公司首先用汽车将牛奶（成批）送到这些工人家里，然后这些工人再将牛奶送给居民。如果不进行特定的任务分配，这意味着对于某个给定的居民家庭来说，由哪个工人将牛奶送来是随机的，此时的工作效率明显是很低的。因为每个工人在完成自己的任务时所走的距离可以看作成本，如果任务分配得当——例如每个工人负责为离自己居住位置相对较近的家庭送牛奶——可以减少这些工人完成每天的工作所需要走的距离，也就是实现较高的工作效率。

假设社区可以表示为一个长度为 m，宽度为 n 的矩形区域。并且假设牛奶公司的客户均匀分布在社区内。假设公司有 k 个工人，他们居住的位置分别表示为 $(x_1,y_1),(x_2,y_2),....(x_k,y_k)$。如图所示，在一个长度为 6，宽度为 4 的社区内，由 3 个工人负责完成任务，实验目的是找到使得所有工人每天行走的距离最小的解（初始社区分布及工人居住位置均可自由指定，或随机）。

1	2	3	2	1	2
1	2	1	3	2	3
3	2	1	3	1	2
1	3	2	3	1	3

三、实验仪器、设备及材料

由实验一配置好的 Swarm 开发软硬件环境

四、实验原理

按照遗传算法的要求，我们需要用一个代码串来表示一种任务分配的模式。一个任务分配的模式就是该问题的一个解，这里我们用一个长度为 $m * n$ 的 k 进制数字串来表示一个候选解，数字串中的每一位数字代表了负责为社区内相应位置的居民送牛奶的工人的代码。

如上图，本实验的候选解表示为一个 3 进制的 24 位的数字串：123212121323231312132313。遗传算法所要求的适应度函数认为与工人完成任务所需要走过的全部距离之和成反比。对于每一个候选解来说，该距离之和的值越小，则适应度越高。每个工人在完成任务时选择路线的原则是从自己居住的位置开始，不断寻找下一个离当前位置最近的、由自己负责的居民。例如，图中第一个工人所选择的路线可能是（1，1），（2，1），（4，1），（3，3），（3，2），（1，5），（3，5），（4，5），其所经过的距离为 13。

在模型初始化阶段，按照给定的参数生成 s 个随机的候选解，分别计算它们的适应度，也就是工人所走的距离总和。然后在每一轮的迭代过程中，不断演化生成新的可行解。生成新的可行解的方法有三种：复制、交叉和突变。复制指的是从原有的一组候选解中挑选出两个加入到新的候选解组中。交叉是指按照一定的概率（交叉率）从原有的一组候选解中挑选出两个并随机选择一个位置，在该位置上将两个解进行交叉互换，生成两个新的候选解。突变是指按照一定的概率（突变率）在新生成的候选解的每一位上发生改变，用其它的有效数字代替该位上的数字。注意交叉率一般不大于 0.2，突变率一般不大于 0.05。原有的候选解被选中进行复制或交叉的概率是根据各个候选解的适应度而分配的。适应度越高的候选解被选中的概率也就越大。适应度较高的候选解中含有具有较高适应性的“基因”——代码段，而通过复制、交叉和突变的方法可以使这些基因保留下来并组合成新的具有更高适应性的解。因此，每一次迭代过程之后生成的新的可行解的平均适应度会不断提高。经过反复的迭代过程，有可能找到具有最大适应的解。

这个实验的目的就是通过演化的算法（遗传算法）找出最有效的任务分配模式，详细请参考遗传算法相关内容。

五、实验步骤

1. 设计
2. 编写程序代码

3. 调试程序代码

4. 收集并分析实验结果

六、实验报告要求

对实验目的、实验复杂度、依据的原理、程序设计、运行结果分析以及遇到的各种问题进行详细整理。以电子文件（如代码，屏幕截图，PPT，电子表格）的方式提交。

语言描述准确，图示清晰。

七、实验注意事项

实验原理请参考遗传算法相关内容

程序代码请参考 Heatbug（热虫模型）

八、思考题

- 给出不同参数下的结果及分析，如 m = 10，n = 10，k = 2，s = 300，交叉率 = 0.2，突变率 = 0.03，迭代次数定为 5000 次。

附录　Heatbug（热虫模型）代码参考

1. StartHeatbugs.java

```
// Java Heatbugs application. Copyright ?1999-2000 Swarm Development Group.
// This library is distributed without any warranty; without even the
// implied warranty of merchantability or fitness for a particular
// purpose.    See file COPYING for details and terms of copying.

import swarm.Globals;

public class StartHeatbugs {
    /** The main() function is the top-level place where everything
        starts.    For a typical Swarm simulation, in main() you create
        a toplevel Swarm, let it build and activate, and set it to
        running.    */
    public static void main (String[] args) {
        // Swarm initialization: all Swarm apps must call this first.
        Globals.env.initSwarm ("jheatbugs", "2.1", "bug-swarm@swarm.org", args);

        // swarmGUIMode is set in initSwarm(). It's set to be `false'
        // if you typed `heatbugs --batchmode' or `heatbugs
        // -b'. Otherwise, it's set to `true'.

        if (Globals.env.guiFlag) {
            // We've got graphics, so make a full ObserverSwarm to get
            // GUI objects
            HeatbugObserverSwarm topLevelSwarm =
                new HeatbugObserverSwarm (Globals.env.globalZone);
            Globals.env.setWindowGeometryRecordName (topLevelSwarm,
"topLevelSwarm");

            topLevelSwarm.buildObjects ();
            topLevelSwarm.buildActions ();
            topLevelSwarm.activateIn (null);
            topLevelSwarm.go ();
            topLevelSwarm.drop ();
        }
        else {
            HeatbugBatchSwarm topLevelSwarm =
                (HeatbugBatchSwarm) Globals.env.lispAppArchiver.getWithZone$key
                (Globals.env.globalZone, "batchSwarm");
            topLevelSwarm.buildObjects ();
            topLevelSwarm.buildActions ();
            topLevelSwarm.activateIn (null);
```

```
                topLevelSwarm.go ();
                topLevelSwarm.drop ();
            }
        }
}
```

2. HeatSpace.java

```
// Bits to support a specialization of diffusion objects: "heat space".
// Most of the real work is done in Diffuse, which implements a CA.
// These functions simplify and stereotype access to the space variable,
// making the Heatbug code higher level.

import swarm.Globals;

import swarm.defobj.Zone;

import swarm.space.Diffuse2dImpl;

import java.util.List;
import java.util.ArrayList;

/** The HeatSpace is a simple object to represent the heat in the
  * world: a spatial variable. HeatSpace inherits most of its
  * behaviour from the "Diffuse2dImpl" space Object. Inherit from
  * Diffuse2dImpl, don't add any new variables */
public class HeatSpace extends Diffuse2dImpl {

    /** constant: maximum heat.    This could just be used from the
        Diffuse2d object's max states. */
    public static final int maxHeat = 0x7fff;

    // used in findExtremeTypeX$Y
    static final int cold = 0, hot = 1;

    public int sizeX, sizeY;

    public HeatSpace (Zone aZone, int worldXSize, int worldYSize,
                      double diffuseConstant, double evaporationRate)
    {
        super (aZone, worldXSize, worldYSize, diffuseConstant,
               evaporationRate);
    sizeX = worldXSize;
    sizeY = worldYSize;
    }
```

```
/**
 * Add heat to the current spot. This code checks the bounds on
 maxHeat, pegs value at the top. */
public Object addHeat$X$Y (int moreHeat, int x, int y) {
      int heatHere;

      heatHere = getValueAtX$Y (x, y);     // read the heat
      if (moreHeat <= maxHeat - heatHere)      // would add be too big?
            heatHere = heatHere + moreHeat;      // no, just add
      else
            heatHere = maxHeat;                // yes, use max
      putValue$atX$Y (heatHere, x, y);      // set the heat
      return this;
}

/**
 *    Search the 9 cell neighbourhood for the requested extreme
 * (cold or hot) The X and Y arguments are used both as input
 * (where to search from) and as output (pointers are filled with
 * the coordinate of the extreme).    Note that wraparound edges
 * (boundary conditions) are implicitly in the code - look at the
 * call to getValueAtX$Y. */

public int findExtremeType$X$Y (int type, HeatCell hc) {
   int bestHeat;
   int x, y;
   List heatList;
   HeatCell cell, bestCell;
   int offset;
   int px = hc.x;
   int py = hc.y;

      // prime loop: assume extreme is right where we're standing
   bestHeat = getValueAtX$Y (px, py);

   // Now scan through the world, finding the best cell in the 8
      // cell nbd.    To remove the bias from the choice of location,
   // we keep a list of all best ones and then choose a random
   // location if there are points of equal best heat.
   heatList = new ArrayList ();

   for (y = py - 1; y <= py + 1; y++) {
for (x = px - 1; x <= px + 1; x++) {
```

```
    int heatHere;
    boolean hereIsBetter, hereIsEqual;

    heatHere = getValueAtX$Y ((x + sizeX) % sizeX,
                              (y + sizeY) % sizeY);

    hereIsBetter = (type == cold) ? (heatHere < bestHeat)
      : (heatHere > bestHeat);

    hereIsEqual = (heatHere == bestHeat);

    if (hereIsBetter) {                  // this spot more extreme

      cell = new HeatCell (x, y);

      // this heat must be the best so far, so delete all the
      // other cells we have accumulated
      heatList.clear ();
      heatList.add (cell);

      // now list only has the one new cell

      bestHeat = heatHere;      // update information
    }

    // if we have spots of equal best heat - then we add to the
    // list from which we can choose randomly later
    if (hereIsEqual) {
      cell = new HeatCell (x, y);
      heatList.add (cell); // add to the end of the list
    }
  }

    }
    // choose a random position from the list
    offset = Globals.env.uniformIntRand.getIntegerWithMin$withMax
  (0, (heatList.size () - 1));

    // choose a point at random from the heat list
    bestCell = (HeatCell) heatList.get (offset);

    // Now we've found the requested extreme. Arrange to return the
    // information (normalize coordinates), and return the heat we found.
```

```
            hc.setX ((bestCell.x + sizeX) % sizeX);
            hc.setY ((bestCell.y + sizeY) % sizeY);

            // clean up the temporary list of (x,y) points
            heatList.clear ();

            return getValueAtX$Y (hc.x, hc.y);
        }
}
```

3. HeatCell.java

```
/**
 * Wrapper object for an (x,y) co-ordinate */
public class HeatCell
{
    public int x, y;

    public HeatCell (int theX, int theY)
    {
        x = theX;
        y = theY;
    }
    public Object setX (int theX)
    {
        x = theX;
        return this;
    }
    public Object setY (int theY)
    {
        y = theY;
        return this;
    }

    public int getX ()
    {
        return x;
    }
    public int getY ()
    {
        return y;
    }
}
```

4. HeatbugObserverSwarm.java

```
import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import swarm.gui.Colormap;
import swarm.gui.ColormapImpl;
import swarm.gui.ZoomRaster;
import swarm.gui.ZoomRasterImpl;

import swarm.analysis.EZGraph;
import swarm.analysis.EZGraphImpl;

import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;

import swarm.space.Value2dDisplay;
import swarm.space.Value2dDisplayImpl;
import swarm.space.Object2dDisplay;
import swarm.space.Object2dDisplayImpl;

import java.util.List;

/**
  The HeatbugObserverSwarm is a swarm of objects set up to observe a
  Heatbugs model when the graphical interface is running. The most
  important object is the heatbugModelSwarm, but we also have graphical
  windows and data analysis */
public class HeatbugObserverSwarm extends GUISwarmImpl {
    /** one parameter: update freq */
    public int displayFrequency;

    /** ActionGroup for sequence of GUI events */
    public ActionGroup displayActions;
```

```
/** the single Schedule instance */
public Schedule displaySchedule;

/** the Swarm we're observing */
public HeatbugModelSwarm heatbugModelSwarm;

/* Lots of display objects. First, widgets */

/**    allocate colours */
public Colormap colormap;
/**    2d display widget */
public ZoomRaster worldRaster;
/** graphing widget */
public EZGraph unhappyGraph;

/* Now, higher order display and data objects */

/** display the heat */
public Value2dDisplay heatDisplay;
/** display the heatbugs */
public Object2dDisplay heatbugDisplay;

/** Constructor for class */
public HeatbugObserverSwarm (Zone aZone) {
   super(aZone);

   // Fill in the relevant parameters (only one, in this case).
   displayFrequency = 1;

   // Now, build a customized probe map using a `local' subclass
   // (a special kind of Java `inner class') of the
   // EmptyProbeMapImpl class.    Without a probe map, the default
   // is to show all variables and messages. Here we choose to
   // customize the appearance of the probe, give a nicer
   // interface.

   class HeatbugObserverProbeMap extends EmptyProbeMapImpl {
      private VarProbe probeVariable (String name) {
         return
            Globals.env.probeLibrary.getProbeForVariable$inClass
            (name, HeatbugObserverSwarm.this.getClass ());
      }
      private MessageProbe probeMessage (String name) {
         return
```

```
            Globals.env.probeLibrary.getProbeForMessage$inClass
            (name, HeatbugObserverSwarm.this.getClass ());
      }
      private void addVar (String name) {
         addProbe (probeVariable (name));
      }
      private void addMessage (String name) {
         addProbe (probeMessage (name));
      }
      public HeatbugObserverProbeMap (Zone _aZone, Class aClass) {
         super (_aZone, aClass);
         addVar ("displayFrequency");
         addMessage ("graphBug:");
      }
   }

   // Install our custom probeMap class directly into the
   // probeLibrary
   Globals.env.probeLibrary.setProbeMap$For
      (new HeatbugObserverProbeMap (aZone, getClass ()), getClass ());
}

public Object _worldRasterDeath_ (Object caller) {
   worldRaster.drop ();
   worldRaster = null;
   return this;
}

public Object _unhappyGraphDeath_ (Object caller) {
   unhappyGraph.drop ();
   unhappyGraph = null;
   return this;
}

/**
    Create the objects used in the display of the model. This code
    is fairly complicated because we build a fair number of
    widgets. It's also a good example of how to use the display
    code. */
public Object buildObjects () {
   //int i;

   super.buildObjects ();
```

```
// First, we create the model that we're actually observing. The
// model is a subswarm of the observer.

heatbugModelSwarm = new HeatbugModelSwarm (getZone ());

// Now create probe objects on the model and ourselves. This gives a
// simple user interface to let the user change parameters.

Globals.env.createArchivedProbeDisplay (heatbugModelSwarm,
                                        "heatbugModelSwarm");
Globals.env.createArchivedProbeDisplay (this, "heatbugObserverSwarm");

// Instruct the control panel to wait for a button event: we
// halt here until someone hits a control panel button so the
// user can get a chance to fill in parameters before the
// simulation runs
getControlPanel ().setStateStopped ();

// OK - the user has specified all the parameters for the
// simulation.    Now we're ready to start.

// First, let the model swarm build its objects.
heatbugModelSwarm.buildObjects ();

// Now get down to building our own display objects.

// First, create a colormap: this is a global resource, the information
// here is used by lots of different objects.
colormap = new ColormapImpl (getZone ());

// Colours [0,64) are assigned to the range Red [0, 1), for
// heat display.
for (int i = 0; i < 64; i++)
   colormap.setColor$ToRed$Green$Blue
      ((byte) i, (double) i / 63.0, 0, 0);

// Colour 64 is set to green, to display heatbugs
colormap.setColor$ToName ((byte) 64, "green");

// Colour 65 is set to white, used in this case below on
// probed heatbug.
colormap.setColor$ToName ((byte) 65, "white");

// Now go in to the heatbugs in the model and set their
```

```
// colours to green (64)
List heatbugList = heatbugModelSwarm.getHeatbugList ();

for (int i = 0; i < heatbugList.size (); i++) {
   Heatbug bug = (Heatbug) heatbugList.get (i);
   bug.setBugColor ((byte) 64);
}

// Next, create a 2d window for display, set its size, zoom
// factor, title.
worldRaster = new ZoomRasterImpl (getZone (), "worldRaster");
try {
   worldRaster.enableDestroyNotification$notificationMethod
      (this,
       new Selector (getClass (), "_worldRasterDeath_", false));
} catch (Exception e) {
   System.err.println ("Exception _worldRasterDeath_: "
                       + e.getMessage ());
}

worldRaster.setColormap (colormap);
worldRaster.setZoomFactor (4);
worldRaster.setWidth$Height
   ((heatbugModelSwarm.getWorld ()).getSizeX (),
    (heatbugModelSwarm.getWorld ()).getSizeY ());
worldRaster.setWindowTitle ("Heat World");
worldRaster.pack();                    // draw the window.

// Now create a Value2dDisplay: this is a special object that
// will display arbitrary 2d value arrays on a given Raster
// widget.
heatDisplay = new Value2dDisplayImpl
   (getZone (), worldRaster, colormap, heatbugModelSwarm.getHeat ());

heatDisplay.setDisplayMappingM$C (512, 0); // turn [0,32768) -> [0,64)

// And also create an Object2dDisplay: this object draws
// heatbugs on the worldRaster widget for us, and also
// receives probes.
try {
   heatbugDisplay = new Object2dDisplayImpl
      (getZone (), worldRaster, heatbugModelSwarm.getWorld (),
       new Selector (Class.forName ("Heatbug"), "drawSelfOn", false));
} catch (Exception e) {
```

```
    System.err.println ("Exception drawSelfOn: " + e.getMessage ());
}

heatbugDisplay.setObjectCollection
    (heatbugModelSwarm.getHeatbugList ());

// Also, tell the world raster to send mouse clicks to the
// heatbugDisplay this allows the user to right-click on the
// display to probe the bugs.
try {
    worldRaster.setButton$Client$Message
        (3, heatbugDisplay, new Selector (heatbugDisplay.getClass (),
                                          "makeProbeAtX$Y", true));
} catch (Exception e) {
    System.err.println ("Exception makeProbeAtX$Y: "
                        + e.getMessage ());
}

// Create the graph widget to display unhappiness.
unhappyGraph = new EZGraphImpl
    (getZone (),
     "Unhappiness of bugs vs. time",
     "time", "unhappiness",
     "unhappyGraph");

//Globals.env.setWindowGeometryRecordName (unhappyGraph, "unhappyGraph");

// instruct this _unhappyGraphDeath_ method to be called when
// the widget is destroyed
try {
    unhappyGraph.enableDestroyNotification$notificationMethod
        (this, new Selector (getClass (),
                             "_unhappyGraphDeath_",
                             false));
} catch (Exception e) {
    System.err.println ("Exception _unhappyGraphDeath_: "
                        + e.getMessage ());
}

// create the data for the average heatbug unhappiness
try {
    unhappyGraph.createAverageSequence$withFeedFrom$andSelector
        ("unhappiness", heatbugModelSwarm.getHeatbugList (),
         new Selector (Class.forName ("Heatbug"), "getUnhappiness",
```

```
                                    false));
        } catch (Exception e) {
            System.err.println ("Exception getUnhappiness: "
                                        + e.getMessage ());
        }
        return this;
    }

    public Object _update_ ()    {
        if (worldRaster != null) {
            heatDisplay.display ();
            heatbugDisplay.display ();
            worldRaster.drawSelf ();
        }

        if (unhappyGraph != null)
            unhappyGraph.step ();
        return this;
    }

    /**
       Create the actions necessary for the simulation. This is where
       the schedule is built (but not run!)    Here we create a display
       schedule - this is used to display the state of the world and
       check for user input. This schedule should be thought of as
       independent from the model - in particular, you will also want
       to run the model without any display.    */
    public Object buildActions () {
        super.buildActions();

        // First, let our model swarm build its own schedule.
        heatbugModelSwarm.buildActions();

        // Create an ActionGroup for display: a bunch of things that
        // occur in a specific order, but at one step of simulation
        // time. Some of these actions could be executed in parallel,
        // but we don't explicitly notate that here.
        displayActions = new ActionGroupImpl (getZone());

        // Add the methods to the ActionGroup to draw the display of
        // the world
        try {
            displayActions.createActionTo$message
                (this, new Selector (getClass (), "_update_", false));
```

```
      // Schedule the update of the probe displays
      displayActions.createActionTo$message
        (Globals.env.probeDisplayManager,
          new Selector (Globals.env.probeDisplayManager.getClass (),
                        "update", true));

      // Finally, schedule an update for the whole user
      // interface code.    This is crucial: without this, no
      // graphics update and the control panel will be
      // dead. It's best to put it at the end of the display
      // schedule
      displayActions.createActionTo$message
        (getActionCache (), new Selector
          (getActionCache ().getClass (), "doTkEvents", true));
    } catch (Exception e) {
      System.err.println ("Exception in setting up displayActions : "
                          + e.getMessage ());
    }

    // And the display schedule. Note the repeat interval is set
    // from our own Swarm data structure. Display is frequently
    // the slowest part of a simulation, so redrawing less
    // frequently can be a help.

    // note frequency!
    displaySchedule = new ScheduleImpl (getZone (), displayFrequency);

    // insert ActionGroup instance on the repeating Schedule
    // instance
    displaySchedule.at$createAction (0, displayActions);

    return this;
  }

  /**
     activateIn: - activate the schedules so they're ready to run.
     The swarmContext argument has to do with what we were activated
     *in*.    Typically the ObserverSwarm is the top-level Swarm, so
     it's activated in "null". But other Swarms and Schedules and
     such will be activated inside of us.    */
  public Activity activateIn (Swarm swarmContext) {
    // First, activate ourselves (just pass along the context).
    super.activateIn (swarmContext);
```

```
    // Activate the model swarm in ourselves. The model swarm is a
    // subswarm of the observer swarm.
    heatbugModelSwarm.activateIn (this);

    // Now activate our schedule in ourselves. This arranges for
    // the execution of the schedule we built.
    displaySchedule.activateIn (this);

    // Activate returns the swarm activity - the thing that's ready to run.
    return getActivity();
  }

  public Object graphBug (Heatbug aBug) {
    if (unhappyGraph != null)
      try {
        unhappyGraph.createSequence$withFeedFrom$andSelector
          ("Bug", aBug, new Selector (aBug.getClass (),
                                      "getUnhappiness", false));
      } catch (Exception e) {
        System.err.println ("Exception graphBug: " + e.getMessage());
      }
    return this;
  }
  public void drop () {
    if (unhappyGraph != null)
      unhappyGraph.disableDestroyNotification ();
    if (worldRaster != null)
      worldRaster.disableDestroyNotification ();
    super.drop ();
  }
}
```

5. HeatbugModelSwarm.java

```
import swarm.Globals;
import swarm.Selector;
import swarm.defobj.Zone;
import swarm.defobj.SymbolImpl;

import swarm.defobj.FArguments;
import swarm.defobj.FArgumentsImpl;
import swarm.defobj.FCall;
import swarm.defobj.FCallImpl;
```

```
import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;
import swarm.activity.FActionForEach;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;
import swarm.objectbase.VarProbe;
import swarm.objectbase.MessageProbe;
import swarm.objectbase.EmptyProbeMapImpl;

import java.util.LinkedList;
import java.util.List;

import swarm.space.Grid2d;
import swarm.space.Grid2dImpl;

/**
 * The HeatbugModelSwarm encapsulates all the objects used in the
 * simulated heatbug world itself (but not the user interface objects)
 * */
public class HeatbugModelSwarm extends SwarmImpl
{
  // simulation parameters
  public int numBugs;
  public double evaporationRate;
  public double diffuseConstant;
  public int worldXSize, worldYSize;
  public int minIdealTemp, maxIdealTemp;
  public int minOutputHeat, maxOutputHeat;
  public double randomMoveProbability;

  public boolean randomizeHeatbugUpdateOrder;

  /** ActionGroup for holding an ordered sequence of action */
  public ActionGroup modelActions;
  /** the single Schedule */
  public Schedule modelSchedule;

  /** list of all the heatbugs */
  public List heatbugList;
```

```
/**    the 2d world    */
public Grid2d world;
/** the 2d heat space */
public HeatSpace heat;

FActionForEach actionForEach;

// These methods provide access to the objects inside the
// ModelSwarm.    These objects are the ones visible to other
// classes via message call.    In theory we could just let other
// objects use Probes to read our state, but message access is
// frequently more convenient.

public List getHeatbugList () {
   return heatbugList;
}

public Grid2d getWorld () {
   return world;
}

public HeatSpace getHeat () {
   return heat;
}

public boolean toggleRandomizedOrder () {
   randomizeHeatbugUpdateOrder = !randomizeHeatbugUpdateOrder;
   syncUpdateOrder ();
   return randomizeHeatbugUpdateOrder;
}

public void syncUpdateOrder () {
   if (actionForEach != null)
      actionForEach.setDefaultOrder
         (randomizeHeatbugUpdateOrder
           ? Globals.env.Randomized
           : Globals.env.Sequential);
}

/**
    * This method isn't normally used, but is convenient when running
    * probes: it lets you easily clone a heatbug and drag it into the
    * model. */
public Object addHeatbug (Heatbug bug) {
```

```
    heatbugList.add (bug);
    return this;
  }

  public HeatbugModelSwarm (Zone aZone) {
    super (aZone);

    // Now fill in various simulation parameters with default values.
    numBugs = 100;
    evaporationRate = 0.99;
    diffuseConstant = 1.0;
    worldXSize = 80;
    worldYSize = 80;
    minIdealTemp = 17000;
    maxIdealTemp = 31000;
    minOutputHeat = 3000;
    maxOutputHeat = 10000;
    randomizeHeatbugUpdateOrder = false;
    randomMoveProbability = 0.0;

    // Now, build a customized probe map using a `local' subclass (a
    // special kind of Java `inner class') of the EmptyProbeMapImpl
    // class. Without a probe map, the default is to show all
    // variables and messages. Here we choose to customize the
    // appearance of the probe, to display a nicer interface.

    class HeatbugModelProbeMap extends EmptyProbeMapImpl {
      private VarProbe probeVariable (String name) {
        return
          Globals.env.probeLibrary.getProbeForVariable$inClass
          (name, HeatbugModelSwarm.this.getClass ());
      }
      private MessageProbe probeMessage (String name) {
        return
          Globals.env.probeLibrary.getProbeForMessage$inClass
          (name, HeatbugModelSwarm.this.getClass ());
      }
      private void addVar (String name) {
        addProbe (probeVariable (name));
      }
      private void addMessage (String name) {
        addProbe (probeMessage (name));
      }
      public HeatbugModelProbeMap (Zone _aZone, Class aClass) {
```

```
            super (_aZone, aClass);
            addVar ("numBugs");
            addVar ("diffuseConstant");
            addVar ("worldXSize");
            addVar ("worldYSize");
            addVar ("minIdealTemp");
            addVar ("maxIdealTemp");
            addVar ("minOutputHeat");
            addVar ("maxOutputHeat");
            addVar ("evaporationRate");
            addVar ("randomMoveProbability");
            addMessage ("toggleRandomizedOrder");
            addMessage ("addHeatbug:");
        }
    }

    // Now, install our custom probeMap class directly into the
    // probeLibrary
    Globals.env.probeLibrary.setProbeMap$For
        (new HeatbugModelProbeMap (aZone, getClass ()), getClass ());
}

/**
     * Now it's time to build the model objects. We use various
     * parameters inside ourselves to choose how to create things.
     */
public Object buildObjects ()
{
    int i;

    // allow our parent class to build anything.
    super.buildObjects();

    // First, set up objects used to represent the
    // environment.    The heatspace agent represents the
    // spatial property of heat.    It is initialized via
    // various model parameters.

    heat = new HeatSpace (getZone (), worldXSize, worldYSize,
                                    diffuseConstant, evaporationRate);

    // Now set up the grid used to represent agent position

    world = new Grid2dImpl (getZone (), worldXSize, worldYSize);
```

```
// Create a list to keep track of the heatbugs in the model.

heatbugList = new LinkedList ();

// Create heatbugs themselves. This is a fairly complex
// step, as is appropriate: the heatbugs are essential
// aspects of the simulation.

// First, a quick hack. During creation we might put
// several heatbugs in the same square. This is a design
// flaw, but it's one that's not fatal, so we ask the
// world object not to warn us about it. This is not an
// example to be emulated :-)

world.setOverwriteWarnings (false);

// Now a loop to create a bunch of heatbugs.

for (i = 0; i < numBugs; i++) {
   Heatbug hbug;
   int idealTemp, outputHeat;

   // Choose a random ideal temperature, output heat
   // from the specified range (model parameters).
   idealTemp =
      Globals.env.uniformIntRand.
      getIntegerWithMin$withMax (minIdealTemp, maxIdealTemp);
   outputHeat =
      Globals.env.uniformIntRand.
      getIntegerWithMin$withMax (minOutputHeat, maxOutputHeat);


   // Create the heatbug, with a standard Java constructor
   hbug =    new Heatbug (world, heat);

   // Add the bug to the end of the list.
   heatbugList.add (hbug);

   // Now initialize the rest of the heatbug's state.

   hbug.setIdealTemperature (idealTemp);
   hbug.setOutputHeat (outputHeat);
   hbug.setRandomMoveProbability (randomMoveProbability);
```

```
        // random position
        hbug.setX$Y ((Globals.env.uniformIntRand.
                        getIntegerWithMin$withMax (0, (worldXSize-1))),
                      Globals.env.uniformIntRand.
                      getIntegerWithMin$withMax (0, (worldYSize-1)));
    }
    world.setOverwriteWarnings (true);          // ok, done cheating.

    return this;
}

/**
     * Here is where the model schedule is built, the data structures
     * that define the simulation of time in the mode. The core is an
     * actionGroup that has a list of actions. That's then put in a
     * Schedule.    */
public Object buildActions () {
    super.buildActions();

    // Create the list of simulation actions. We put these in
    // an action group, because we want these actions to be
    // executed in a specific order, but these steps should
    // take no (simulated) time. The M(foo) means "The message
    // called <foo>". You can send a message To a particular
    // object, or ForEach object in a collection.

    // Note we update the heatspace in two phases: first run
    // diffusion, then run "updateWorld" to actually enact the
    // changes the heatbugs have made. The ordering here is
    // significant!

    // Note also, that with the additional
    // `randomizeHeatbugUpdateOrder' Boolean flag we can
    // randomize the order in which the bugs actually run
    // their step rule.    This has the effect of removing any
    // systematic bias in the iteration throught the heatbug
    // list from timestep to timestep

    // By default, all `createActionForEach' modelActions have
    // a default order of `Sequential', which means that the
    // order of iteration through the `heatbugList' will be
    // identical (assuming the list order is not changed
    // indirectly by some other process).
```

```
modelActions = new ActionGroupImpl (getZone ());

try {
   modelActions.createActionTo$message
      (heat, new Selector (heat.getClass (), "stepRule", false));
} catch (Exception e) {
   System.err.println ("Exception stepRule: " + e.getMessage ());
}

try {
   Heatbug proto = (Heatbug) heatbugList.get (0);
   Selector sel =
      new Selector (proto.getClass (), "heatbugStep", false);
   actionForEach =
      modelActions.createFActionForEachHomogeneous$call
      (heatbugList,
        new FCallImpl (this, proto, sel,
                       new FArgumentsImpl (this, sel, true)));
} catch (Exception e) {
   e.printStackTrace (System.err);
}

syncUpdateOrder ();

try {
   modelActions.createActionTo$message
      (heat, new Selector (heat.getClass (), "updateLattice", false));
} catch (Exception e) {
   System.err.println("Exception updateLattice: " + e.getMessage ());
}

// Then we create a schedule that executes the
// modelActions. modelActions is an ActionGroup, by itself it
// has no notion of time. In order to have it executed in
// time, we create a Schedule that says to use the
// modelActions ActionGroup at particular times.    This
// schedule has a repeat interval of 1, it will loop every
// time step.    The action is executed at time 0 relative to
// the beginning of the loop.

// This is a simple schedule, with only one action that is
// just repeated every time. See jmousetrap for more
// complicated schedules.
```

```
        modelSchedule = new ScheduleImpl (getZone (), 1);
        modelSchedule.at$createAction (0, modelActions);

        return this;
    }

    /**
     * Now set up the model's activation. swarmContext indicates where
     * we're being started in - typically, this model is run as a
     * subswarm of an observer swarm. */
    public Activity activateIn (Swarm swarmContext) {
        // First, activate ourselves via the superclass
        // activateIn: method.    Just pass along the context: the
        // activity library does the right thing.
        super.activateIn (swarmContext);

        // Now activate our own schedule.
        modelSchedule.activateIn (this);

        // Finally, return our activity.
        return getActivity ();
    }
}
```

6. HeatbugBatchSwarm.java

```
import swarm.Globals;
import swarm.Selector;

import swarm.defobj.Zone;

import swarm.activity.Activity;
import swarm.activity.ActionGroup;
import swarm.activity.ActionGroupImpl;
import swarm.activity.Schedule;
import swarm.activity.ScheduleImpl;

import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;

import swarm.analysis.EZGraph;
import swarm.analysis.EZGraphImpl;
```

```
public class HeatbugBatchSwarm extends SwarmImpl {

    /** Frequency of fileI/O */
    public int loggingFrequency;

    /** When to Stop the Sim */
    public int experimentDuration;

    public ActionGroup displayActions;
    public Schedule displaySchedule;
    public Schedule stopSchedule;

    /** the Swarm we're observing */
    public HeatbugModelSwarm heatbugModelSwarm;

    /** The EZGraph will be used in FileI/O mode rather than the usual
        Graphics mode... */
    public EZGraph unhappyGraph;

    public HeatbugBatchSwarm (Zone aZone) {
        super (aZone);
    }

    public Object buildObjects () {
        super.buildObjects();

        // But since we don't have any graphics, we load the
        // object from the global `lispAppArchiver' instance which
        // is created automatically from the file called
        // `heatbugs.scm'

        // `modelSwarm' is the key in `heatbugs.scm' which
        // contains the instance variables for the
        // HeatbugModelSwarm class, such as numBugs etc.

        heatbugModelSwarm =
          (HeatbugModelSwarm)
          Globals.env.lispAppArchiver.getWithZone$key (getZone (),
                                                       "modelSwarm");

        // Now, let the model swarm build its objects.
        heatbugModelSwarm.buildObjects ();

        // Finally, build some data analysis objects. In this case
```

```
        // we're just going to create an EZGraph (with graphics turned
        // off and fileI/O turned on) collect some statistics (the
        // average) over the collection of heatbugs (which we get from
        // the heatbugModelSwarm).

        // If the user sets loggingFrequency to 0 s/he does not
        // require the logging of results at all. Consequently, some
        // objects will not be created -> the schedule will also be
        // simplified. This sort of switch is useful when the Sim
        // could potentially log many different aspects of the
        // model...
        if(loggingFrequency > 0) {
            unhappyGraph =    new EZGraphImpl (getZone (), true);

            try {
                unhappyGraph.createAverageSequence$withFeedFrom$andSelector
                    ("unhappiness.output",
                     heatbugModelSwarm.getHeatbugList (), new Selector
                         (Class.forName("Heatbug"), "getUnhappiness",
                          false));
            } catch (Exception e) {
                System.err.println ("Exception batch getUnhappines: "
                                    + e.getMessage ());
            }
        }
        // All done - we're ready to build a schedule and go.
        return this;
    }

    /** Create the actions necessary for the simulation. This
        is where the schedule is built (but not run!) */
    public Object buildActions () {
        super.buildActions();

        // First, let our model swarm build its own schedule.
        heatbugModelSwarm.buildActions();

        if (loggingFrequency > 0) {

            // Create an ActionGroup for display. This is pretty
            // minimal in this case. Note, there's no doTkEvents
            // message - no control panel!
            displayActions = new ActionGroupImpl (getZone ());
```

```
            // Now schedule the update of the unhappyGraph, which will
            // in turn cause the file I/O to occur...
            try {
                displayActions.createActionTo$message
                    (unhappyGraph,
                     new Selector (unhappyGraph.getClass (), "step", true));
            } catch (Exception e) {
                System.err.println ("Exception batch unhappyGraph step: "
                                    + e.getMessage ());
            }

            // the displaySchedule controls how often we write data out.
            displaySchedule =
                new ScheduleImpl (getZone (), loggingFrequency);

            displaySchedule.at$createAction (0, displayActions);
        }

        // We also add in a "stopSchedule", another schedule
        // with an absolute time event - stop the system at
        // time .
        stopSchedule = new ScheduleImpl (getZone (), true);

        try {
            stopSchedule.at$createActionTo$message
                (experimentDuration, this, new Selector
                    (getClass (), "stopRunning", false));
        } catch (Exception e) {
            System.err.println ("Exception stopRunning: "
                                + e.getMessage ());
                                }
        return this;
    }

    /** activateIn: - get the Swarm ready to run. */
    public Activity activateIn (Swarm swarmContext) {

        // First, activate ourselves (just pass along the context).
        super.activateIn (swarmContext);

        // We need to activate the model swarm.
        heatbugModelSwarm.activateIn (this);

        // Now activate our schedules in ourselves. Note that we just
```

```
        // activate both schedules: the activity library will merge
        // them properly.
        stopSchedule.activateIn (this);

        if (loggingFrequency > 0)
            displaySchedule.activateIn (this);

        // Activate returns the swarm activity - the thing that's
        // ready to run.
        return getActivity ();
    }

    /** the HeatbugObserverSwarm had a go method inherited from
        GUISwarm, but we have to define our own here. It's pretty
        simple. There's also a friendly message printed out here just
        in case someone is confused when they run heatbugs and see no
        graphics. */
    public Object go () {
        System.out.println ("You typed `heatbugs -b' or    `heatbugs --batch'"
                            + " so we're running without graphics.");

        System.out.println ("Heatbugs is running for "
                            + experimentDuration + " timesteps");

        if (loggingFrequency > 0)
            System.out.println ("It is logging data every "
                                + loggingFrequency +
                                " timesteps to: unhappiness.output");

        (getActivity ().getSwarmActivity ()).run ();

        if (loggingFrequency > 0)
            // Close the output file.
            unhappyGraph.drop() ;

        return getActivity ().getStatus ();
    }

    /** The termination method. When this fires we just terminate
        everything that's running and close our output file(s) by
        dropping the EZGraph which "owns" the sequence(s) we are
        logging. */
    public Object stopRunning () {
        // Terminate the simulation.
```

```
            System.out.println("quitting at " + Globals.env.getCurrentTime ());
            Globals.env.getCurrentSwarmActivity ().terminate ();
            return this;
        }
}
```

7. Heatbug.java

```
import swarm.Globals;
import swarm.space.Grid2d;
import swarm.space.Grid2dImpl;
import swarm.gui.Raster;

/**
 * Heatbugs are agents in a 2d world with simple behaviour: if too
 * cold, move to warmer spot if too warm, move to cooler spot.    and
 * some occasional exceptions if the spot is occupied, try to move to
 * an unoccupied spot.    randomMoveProbability chance of moving to a
 * random spot */
public class Heatbug {
  /** my current unhappiness */
  public double unhappiness;
  /**    my spatial coordinates */
  public int x, y;
  /**    my ideal temperature */
  public int idealTemperature;
  /** how much heat I put out */
  public int outputHeat;
  /** chance of moving randomly */
  public double randomMoveProbability;
  /** the world I live in */
  public Grid2d world;
  /** how big that world is */
  public int worldXSize, worldYSize;
  /** the heat for the world */
  public HeatSpace heat;
  /** my colour (display) */
  public byte bugColor;

  /* Scratch cell for extracting return values */
  public HeatCell scratchHeatCell;

  /**
   * these methods are used to initialize the object's state. First,
   * methods that have to be sent to create an object. */
```

```
/**
 * Constructor for Heatbug
 */
public Heatbug (Grid2d w, HeatSpace h) {
   // Strictly speaking, this check isn't necessary. But we intend these
   // parameters to be immutable once set, so to be extrasafe we check:
   // it could catch an error later.

   if (world != null || heat != null)
      System.err.println ("You can only set the world/heat " +
                                    "of a heatbug at creation time");

   world = w;
   heat = h;

   // make sure the user set up world and heat.

   if (world == null || heat == null)
      System.err.println ("Heatbug was created without a world or heat");

   // Cache the worldSize for speed of later access. Note how we
   // do this in createEnd - it could also have been done when
   // setWorld:Heat: was called, but this is a good place to do
   // it, too. If an object needed to allocate extra memory, this
   // is the right place to do it.

   worldXSize = world.getSizeX ();
   worldYSize = world.getSizeY ();

   // Someday, it'd be good if the space library to be powerful
   // enough that the heatbugs never need to be aware how big
   // their world is.

   scratchHeatCell = new HeatCell (0, 0);
}

/**
      Methods for reading/writing a Heatbug's state during runtime.
      The probe mechanism is the lowlevel way of getting at an
      object's state - you're also allowed (but not required) to
      write methods to access the state as you find it is necessary
      or convenient.    Note the naming convention: for a variable
      named "fooBar" methods are -(sometype) getFooBar; -setFooBar;
```

```
      this naming convention will be important for a later version
      of probe.    (probe will preferentially use these methods
      instead of direct access).    */
public double getUnhappiness () {
   return unhappiness;
}

/**
     Simple set methods for Heatbug state. Some of these are
     probably not going to normally change in a heatbugs lifetime,
     but there's no reason they couldn't change.    */
public Object setIdealTemperature (int i) {
   idealTemperature = i;
   return this;
}

public Object setOutputHeat (int o) {
   outputHeat = o;
   return this;
}

public Object setRandomMoveProbability (double p) {
   randomMoveProbability =    p;
   return this;
}

/**
     This method is a bit dangerous: we blindly put ourselves on top
     of the grid no matter what's underneath us: because Grid2d only
     allows one object per square, we could be destroying data. This
     is poor design, but fortunately doesn't kill us in this
     particular app. If some other object really needed to find all
     objects based on looking in the grid, it would cause
     problems. (But note, in heatbug creation, how we tell Grid2d to
     turn off its warnings about overwrites) */
public Object setX$Y (int inX, int inY) {
   x = inX;
   y = inY;
   world.putObject$atX$Y (this, x, y);                // yikes!
   return this;
}


/**
```

```
    All of the previous code is basic Swarm object programming. The
    real simulation code follows.    Heatbug behaviour is actually
    implemented here. The notion of a "step" method is a nice
    simplification for basic simulations.
*/
public void heatbugStep () {
  int heatHere;
  int newX, newY;
  int tries;

  // find out the heat where we are sitting.

  heatHere = heat.getValueAtX$Y (x, y);

  // update my current unhappiness value: abs(ideal - here);

  if (heatHere < idealTemperature)
    unhappiness = (double) (idealTemperature - heatHere) / HeatSpace.maxHeat;
  else
    unhappiness = (double) (heatHere - idealTemperature) / HeatSpace.maxHeat;

  // now ask the heatspace to tell us where the warmest or
  // coldest spot is The method call returns values back into
  // newX and newY.

  scratchHeatCell.x = x;
  scratchHeatCell.y = y;

  heat.findExtremeType$X$Y (((heatHere < idealTemperature)
                             ? HeatSpace.hot
                             : HeatSpace.cold),
                            scratchHeatCell);
  newX = scratchHeatCell.x;
  newY = scratchHeatCell.y;

  // After choice of ideal spot is made, there's a chance of
  // random move.    (Note the normalization of coordinates to [0,
  // worldSize). The current space library does not enforce
  // boundary conditions.)
  if ((Globals.env.uniformDblRand.getDoubleWithMin$withMax (0.0, 1.0))
      < randomMoveProbability)
    {
      // pick a random spot
      newX =
```

```
        x + Globals.env.uniformIntRand.getIntegerWithMin$withMax (-1, 1);

      newY =
        y + Globals.env.uniformIntRand.getIntegerWithMin$withMax (-1, 1);

      // normalize coords
      newX = (newX + worldXSize) % worldXSize;
      newY = (newY + worldYSize) % worldYSize;
    }

// Part of the heatbug simulation is that two bugs cannot be
// in the same spot. The code to enforce that is done here: if
// the site we want is occupied by another heatbug, move
// randomly. Note that this code does not parallelize
// properly, it requires that each bug be set a "step" method
// in sequence. This is a design flaw in heatbugs: proper
// conflict resolution is difficult.    Also note we only look
// for 10 random spots - if we don't find an unoccupied spot
// by then, assume it's too crowded and just don't move.

if (unhappiness == 0)
    {
      // only update heat - don't move at all if no unhappiness
      heat.addHeat$X$Y (outputHeat, x, y);
    }
else {
    tries = 0;

    // only search if the current cell is neither the optimum
    // or randomly chosen location - else don't bother
    if ( (newX != x || newY != y) )
      {
        while ( (world.getObjectAtX$Y (newX, newY) != null)
                  && (tries < 10) )
          {
            int location, xm1, xp1, ym1, yp1;
            // choose randomly from the nine possible
            // random locations to move to
            location =
              Globals.env.uniformIntRand.getIntegerWithMin$withMax (1,8);

            xm1 = (x + worldXSize - 1) % worldXSize;
            xp1 = (x + 1) % worldXSize;
            ym1 = (y + worldYSize - 1) % worldYSize;
```

```
            yp1 = (y + 1) % worldYSize;

            switch (location)
              {
              case 1:
                newX = xm1; newY = ym1;      // NW
                break;
              case 2:
                newX = x ; newY = ym1;       // N
                break;
              case 3:
                newX = xp1 ; newY = ym1;    // NE
                break;
              case 4:
                newX = xm1 ; newY = y;       // W
                break;
              case 5:
                newX = xp1 ; newY = y;       // E
                break;
              case 6:
                newX = xm1 ; newY = yp1;    // SW
                break;
              case 7:
                newX = x ; newY = yp1;       // S
                break;
              case 8:
                newX = xp1 ; newY = yp1;    // SE
              default:
                break;
              }

            tries++;                    // don't try too hard.
          }
        if (tries == 10)
          {
            // no nearby clear spot, so just don't move.
            newX = x;
            newY = y;
          }
      }
// Phew - we've finally found a spot to move ourselves, in
// (newX, newY). Update heat where we were sitting.

heat.addHeat$X$Y (outputHeat, x, y);
```

```
            // Now move ourselves in the grid and update our coordinates.

            world.putObject$atX$Y (null, x, y);
            x = newX;
            y = newY;
            world.putObject$atX$Y (this, newX, newY);
        }

        // all done moving! Return this.
    }

    /**
        Extra bits of display code: setting our colour, drawing on a
        window.    This code works, but it'd be better if there were a
        generic object that knew how to draw agents on grids.    */
    public Object setBugColor(byte c) {
        bugColor = c;
        return this;
    }

    public Object drawSelfOn (Raster r) {
        r.drawPointX$Y$Color (x, y, bugColor);
        return this;
    }
}
```

《分形》实验教学指导书

课程名称：分形

英文名称：fractal

课程代码：

课程性质：必修

实验指导书名称：分形实验教程

适用专业：系统科学、管理科学与工程

一.实验总学时（课外学时/课内学时）：20　　总学分：2

必开实验个数：5　　选开实验个数：5

二.实验的地位、作用和目的

分形作为系统科学研究的一个重要课题，是系统科学专业和管理科学与工程专业硕士研究生的一门必修课程。分形是现代数学的一个新分支，但其本质却是一种新的世界观和方法论。分形集是一类不能用经典几何方法描述的“不规则”集合，它们基本满足分形的经典性质，但是在自相似的程度上可以有很大差别。分形实验有助于拓展学生的科学视野，增强学生的数学素养，加深对系统科学的理解。分形实验的目的是理论与实践相结合，将分形的原理应用于现实生活中，生成有趣的分形图案，提高学生对科学世界以及数学领域的兴趣，为进一步从事科学研究打下坚实的基础。

三.基本原理及课程简介

被誉为大自然的几何学的分形理论，是现代数学的一个新分支，但其本质却是一种新的世界观和方法论。它与动力系统的混沌理论交叉结合，相辅相成。它承认世界的局部可能在一定条件下，在某一方面表现出与整体的相似性，它承认空间维数的变化既可以是离散的也可以是连续的，因而拓展了视野。

分形集是一类不能用经典几何方法描述的“不规则”集合，它们基本满足分形的经典性质，但是在自相似的程度上可以有很大差别。分形几何就是研究所谓“简单”空间上这样一类“复杂”子集的一门新兴数学分支。

我们可以根据分形集合生成算法的特征对分形进行分类，一般分为线性分形和非线性分形两大类。这两类分形，都有无穷的算法表述，因而也都包含无穷的分形图形。线性分形是最基本的一种，从数学上说，就是实现这些分形的算法中仅含有一次项，如 Koch 曲线、Peano 曲线及迭代函数系统(IFS)，这些迭代变换使直线保持平直，仅改变其长度、位置和方向。非线性分形则内容丰富得多，变化也多，如 Julia 集。

我们也可以根据分形产生算法中随机性加入的影响，把分形分为确定性分形和随机性分形两类。对确定性分形，其算法的规则是确定的，在算法中也没有随机性的加入，或者虽然有随机性的加入，但并不影响分形图形的形状，即算法的多次重复仍然产生同一个分形图，如 IFS；对随机性分形，虽然其产生的规则也是一定的，但随机因素的影响，可以使每次生成过程产生的分形虽然可以具有一样的复杂度，但形态都会有所不同，它不具有可重现性，如 Brown 运动。

分形的发展很大程度上依赖于计算机科学技术的进步，这也对纯数学的传统观念提出了挑战。计算机技术不仅使分形领域的一些新发现成为可能，同时因其图形直观的表现方式也极大地激发了科学家和人们的兴趣与认识，推动了分形理论的发展。

四.实验基本要求

1、了解分形的基本概念和发展历史。

2、掌握分形的数学原理和基本生成方法。

3、了解分形在现实生活中的应用领域。

4、在基础性实验中，掌握几种经典分形图形的生成算法，能够根据要求生成相应的分形图形。

5、在综合性实验中，能够独立完成对一个复杂系统分形结构的模拟，对模拟过程中出现的问题能够独立排除。

6、能够根据模拟结果，对所提出的问题进行初步分析。

7、撰写简明扼要、图表清晰，结论正确、分析科学的实验报告。

五．实验具体要求

分形实验采用三段式教学模式，教学过程由理论知识准备、基础性实验及综合性实验三个环节组成。

（一）理论知识讲座

通过教师讲解，使学生了解分形的基本概念和发展历史，初步掌握分形的数学原理和基本生成方法。了解本实验课程的性质、任务、要求、课程进度安排、实验考核内容及实验报告要求等，为进行实验做好准备。理论知识准备时间为1—2次课。

（二）基础性实验

此阶段包括5个实验的操作训练，这是本实验课程的中心环节。要求做好：

1.预习

学生须认真阅读实验指导书，了解实验的目的和原理，明确本次实验中需要了解和掌握的分形图形生成的数学原理，在此基础上写出实验预习报告，内容包括：实验目的和基本原理，分形图形生成的数学原理，简单的实验步骤。

2．实验

学生进入实验室后首先熟悉分形程序的运行环境。教师检查学生的预习情况并做好记录，包括预习报告和对实验的理解，不合格者不得进行实验。

指导教师讲解实验难点和注意事项，通过提问的方式引导学生深入思考与实验现象有关的一些问题，着力培养学生观察实验、综合考虑问题的能力，使学生学会分析和研究问题的方法。学生独立或按学号编组进行实验，注意实验中的独立操作和相互配合。要求学生在实验中勤于动手，敏锐观察，细心操作，开动脑筋，分析钻研问题，熟练掌握分形图形的数学原理及生成算法。实验结束后经教师检查并签名，实验结果和程序代码才有效。

3．书写实验报告

认真写出实验报告是本课程的基本训练。它将使学生在实验数据处理、作图、误差分析、问题归纳等方面得到训练和提高。实验报告的质量，在很大程度上反映了学生的实际水平和能力。

实验报告的内容大致包括：实验目的和原理、分形图形生成的数学原理、实验结果的展示和分析以及应提交的带注释的源程序代码等。实验报告的讨论可以包括对实验现象的分析和解释、对实验结果的误差分析、对实验的改进意见、心得体会等。

实验结束的两周内完成实验报告，于下次实验时提交报告，其中必须附有教师签名的原始记录表。

（三）综合性实验

综合性实验的开设是为了使学生在掌握分形的基本原理的基础上，进一步提高学生解决现实具体问题的能力。教师可以根据本专业的实际情况选开其中部分实验。

综合性实验又分为两种类型：

类型 1：基础性综合实验。开设 3 个与实际应用关系密切、具有一定复杂性和综合程度的实验，学生选做其中的 1 个。该类型实验由教师给出实验题目和要求，学生选择实验题目后需要查找资料，研读文献，钻研有关理论知识，在此基础上选择实验方法，提出实验方案，与指导教师讨论实验方案的可行性，然后预约时间进行实验。

类型 2：研究性综合实验。该类型实验均由教师的科研项目转化而来，目前开发有 2 个实验供学生选择，为实验兴趣浓厚和学有余力的学生提供更多的实验空间，对培养学生的创新意识和科研能力大有裨益。研究性改变了传统实验中固定实验步骤，照方抓药式的被动性，使学生有了很大的自由发挥空间；其次，实验引入了挫折—激励教学机制，允许多次失败，但要求最终一定要成功。

综合实验室每天都对学生开放，学生只要有空，都可以预约时间，到实验室做实验。

六.考核与报告

学生在课下做好预习，写出预习报告，实验前指导教师检查预习情况，根据学生预习报告和回答问题情况给出预习成绩。

实验进行中，教师考察学生的实验操作技能，根据学生实验操作、遵守实验规则、实验纪律情况以及实验结果展示情况给出实验操作分。

实验结束后学生书写实验报告，并于下一次实验时提交，其中必须附有教师签名的原始记录表。教师根据学生实验报告书写的完整性、实验结果的合理性、对实验的理解和体会等给出报告成绩，包含报告完整、字迹工整(30%)、数学原理正确(10%)、程序代码正确(40%)、实验结果正确(20%)。

期末的最后两次实验作为考核实验，考查学生独立从事实验研究的能力，给出考核成绩。

本实验课程的最后成绩：预习分 20%+操作分 30%+实验报告分 40%+实验考核分 10%

缺少实验或报告者须在实验教学结束前补做和补交，否则不予通过。成绩不合格者下学年重修。

综合实验不安排课内学时，学生利用课外时间到开放实验室进行实验。实验之前写出完整的预习报告，详细列出实验目的、依据的原理、实验步骤、算法说明。实验结束后，将实验结果整理成小论文的形式提交。教师根据学生实践能力、实验水平、实验结果的创造性大小来评定实验成绩。

七.实验指导书（实验教材）

1．分形论——奇异性探索　林鸿益、李映雪　北京理工大学出版社

2．分形对象-形、机遇和维数　B.曼德尔布洛特　世界图书出版公司

3．分形——大自然的艺术构造　汪富泉等　山东教育出版社

八.实验项目与内容提要

序号	实验项目名称	内容提要	实验学时	每组人数	实验类型	实验要求
1	Mandelbrot 集合	1．了解 Mandelbrot 集合的生成原理 2．用计算机生成 Mandelbrot 集合的分形图形	2	1	基础	必做
2	Julia 集合	1．了解 Julia 集合的生成原理 2．用计算机生成 Julia 集合的分形图形	2	1	基础	必做
3	Newton/Nova 分形	1．了解 Newton 分形的原理及 Nova 分形的改进 2．用计算机生成 Newton/Nova 分形图形	2	1	基础	必做
4	英国的海岸线有多长	1．了解海岸线度量的数学模型 2．用计算机生成海岸线的分形图形	2	1	基础	必做
5	H-分形	1．了解 H-分形的应用背景 2．掌握 H-分形的生成原理 3．用计算机生成 H-分形的图形	2	1	基础	必做
6	L 系统	1．了解 L 系统的生成原理 2．用计算机生成 L 系统的分形图形	2	3	综合	选做
7	IFS 系统	1．了解用 IFS 系统创作分形图形的原理 2．用 IFS 系统生成分形图形	2	3	综合	选做
8	分形图像压缩技术	1．了解分形图像压缩技术的原理 2．掌握分形图像压缩技术的几种典型算法	2	3	综合	选做
9	分形音乐	1．了解分形音乐生成原理 2．用计算机生成一段分形音乐	2	3	综合	选做
10	资本市场的分形结构及其自组织临界性研究	1．了解资本市场的分形结构 2．了解资本市场分形结构的研究意义及应用前景	2	5	综合	选做

实验一：分形中的 Mandelbrot 集合（实验代码 1）

一、实验目的

了解分形中的 Mandelbrot 集合的生成原理，并能使用计算机生成 Mandelbrot 集合的各种分形图形。

二、 实验内容

1. 掌握分形几何的基本定义，并能根据定义举出身边的分形几何图形。
2. 观察 Mandelbrot 集合的基本分形图形，找出几个图形的基本特征与生成规律。如图 1，图 2，图 3。

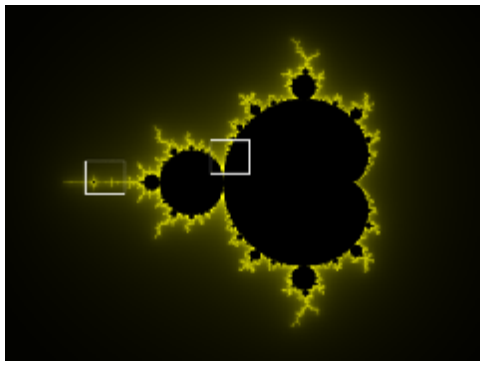


图 1　Mandelbrot 集合

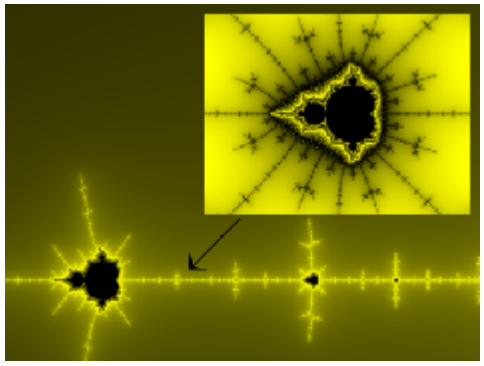


图 2 Mandelbrot 集合局部放大

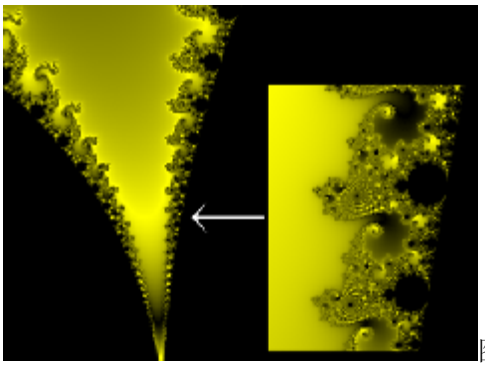


图 3　Mandelbrot 集合局部放大

3. 结合图形以及分形图形生成的基本原理，掌握 Mandelbrot 集合的基本生成过程以及生成原理。
4. 通过 Mandelbrot 集合生成的 4 个步骤，用计算机模拟 Mandelbrot 集合的生成过程。

假设监视器的分辨率是 a*b 点，可显示的颜色为 K+1 种，分别以数字 0，1，2，…．K 表示，且以 0 表示黑色。

步骤 0：

选定 P1=-2.25, P2=0.75,q1=-1.5,q2=1.5,M=100.

令⊿P=P2-P1/a-1，⊿q=q2-q1/b-1

对所有点（Np，Nq），Np=0，1，…a-1 及 Nq=1,2,…b-1,完成如下循环

步骤 1：

P0=P1+Np．⊿P, q0=q1+Nq．⊿q

令 k=0，　x0=y0=0

步骤 2：

用式 Yk+1=2XkYk+q 从（Xk，Yk）得出（Xk+1，Yk+1），计数 k：=k+1

步骤 3：

计算 r=Xk* Xk+Yk* Yk

如果 r>M,则选择颜色 k，转至步骤 4；如果 k=M，则选择颜色 0（黑色），转至步骤 4；如果 r<=M，且 k<K，转至步骤 2；

步骤 4：

对于点（Np，Nq）着颜色 k，转至下一个点（步骤 1）

利用上述过横，灵活地选取画面窗口，可以观察到 Mandelbrot 集合的精细结构。

5. 根据 Mandelbrot 集的算法，带入不同的参数值，反复实现计算机的图形

显示。

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 VISUAL BASIC6.0 程序运行环境

四、实验报告要求

1．提交对于基本分形图形的理论解释，列出几种基本分形图形的基本特征与生成原理

2．列出 Mandelbrot 集的图形生成过程。

3．对于 Mandelbrot 集的图形生成步骤，代入不同参数进行计算，比较几种结果的图形差异。

实验二：分形中的 Julia 集合（实验代码 2）

一、实验目的

了解分形中的 Julia 集合的生成原理，并能使用计算机生成 Mandelbrot 集合的各种分形图形。

二、实验内容

1. 深入掌握分形的基本原理，理解分形几何图形的生成过程。
2. 观察 Julia 集合的基本形态和精细的结构，找出几个图形的基本特征与生成规律。如图 4，图 5，图 6。

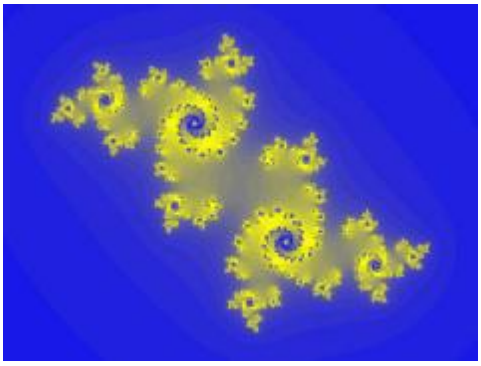


图 4 象尘埃一样的结构

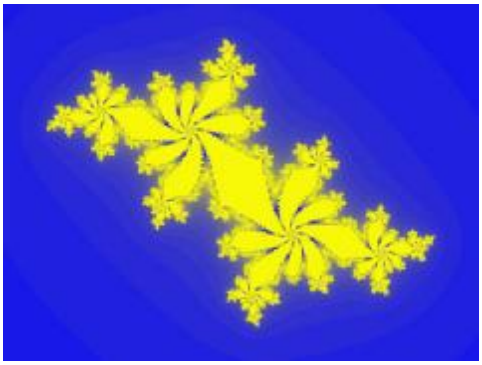


图 5 稳定的固态型

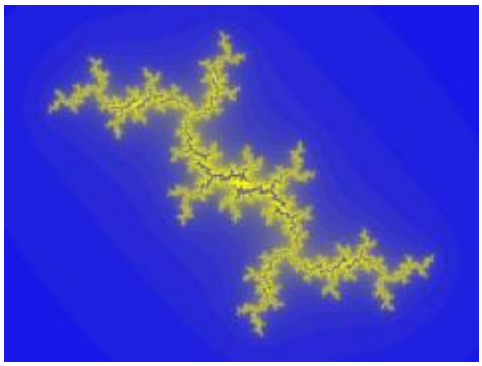


图 6 象树枝状

3. 结合几种 Mandelbrot 集以及图形生成的基本过程，掌握 Mandelbrot 集合的基本生成过程以及生成原理。
4. 通过 Mandelbrot 集合生成的 4 个步骤，用计算机模拟 Mandelbrot 集合的生成过程。

假设监视器的分辨率是 a*b 点，可显示的颜色为 K+1 种，分别以数字 0，1，2，….K 表示，且以 0 表示黑色。

步骤 0：

选定参数 u=p+iq，Xmin=Ymin=-1.5, Xmax=Ymax=1.5.这就是说，图形在指定范围内显示。又取 M=100，这是为了在计算机上作控制：复数的模超过 M 就被认为是“无穷”，自然，M 的值可以换成其他大的实数。

令⊿X=Xmax-Xmin/a-1，⊿Y=Ymax-Ymin/b-1

对所有点（Nx，Ny），Nx=0，1，…,a-1 及 Ny=0，1，…,b-1 完成下面的循环

步骤 1：

令 X0=Xmin+Nx* ⊿X ， Y0=Ymin+Ny*⊿Y，k=0

步骤 2：

由迭代过程 Yk+1=2XkYk+q 从（Xk，Yk）得出（Xk+1，Yk+1），计数 k：=k+1

步骤 3：

计算 r=Xk* Xk+Yk* Yk

如果 r>M,则选择颜色 k，转至步骤 4；如果 k=M，则选择颜色 0（黑色），转至步骤 4；如果 r<=M，且 k<K，转至步骤 2；

步骤 4：

对于点（Np，Nq）着颜色 k，转至下一个点（步骤 1）

5. 根据 Julia 集的算法，带入不同的参数值，反复实现计算机的图形显示。
6. 比较 Mandelbrot 集和 Julia 集生成原理，找出图形生成过程中的差异。

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 VISUAL BASIC6.0 程序运行环境

四、实验报告要求

1. 列出 Julia 集的图形生成算法。

2. 对于 Julia 集的图形生成步骤，代入不同参数进行计算，比较几种结果的图形差异。

3. 比较 Mandelbrot 集和 Julia 集的异同，列出图形生成步骤的差异以及典型图形的差异。

实验三：Newton/Nova 分形（实验代码 3）

一、实验目的

了解 Newton 分形的原理及 Nova 分形的改进，并能使用计算机生成 Newton/Nova 分形图形。

二．实验内容

1. 了解 Newton 分形产生的基本背景及方法。
2. 观察基本 Newton 分形图形，找出图形的基本特征与生成规律。如图 7

图 7 Newton 分形

3. 掌握 Nova 分形图形产生的基本过程与方法。
4. 观察基本 Nova 分形图形，找出图形的基本特征与生成规律。如图 8

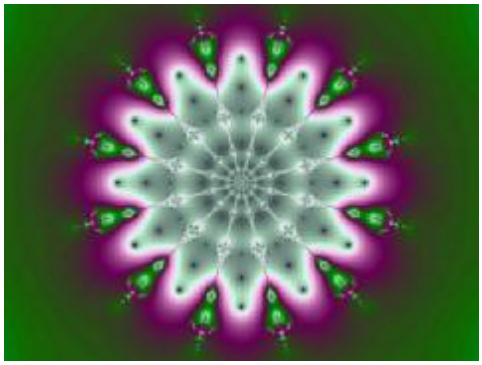


图 8 Nova 分形

5. 比较 Newton 与 Nova 分形图形的差异，以及与 Julia 集的异同，自己更改 Julia 集合的参数，来产生不同的分形图形。

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 VISUAL BASIC6.0 程序运行环境。

四、实验报告要求

1. 列出 Newton 与 Nova 分形图形产生的理论背景
2. 给出 Newton 分形产生的方法
3. 给出 Nova 分形产生的方法
4. 比较比较 Newton 与 Nova 分形的差异。

实验四：英国的海岸线有多长（实验代码 4）

一、实验目的

了解分形理论中的自相似原则和迭代生成原则，掌握海岸线度量的数学模型，并能使用计算机生成海岸线的分形图形。

二、实验内容

1. 理解海岸线度量问题产生的理论背景。
2. 掌握自相似原则和迭代生成原则。
3. 根据自相似原则分析海岸线度量问题的基本特征和规律
4. 学习海岸线度量的数学模型，更加深入的理解分形的迭代生成原则。
5. 根据海岸线度量的数学模型，改变不同的参数设置，观察结果变化。

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 VISUAL BASIC6.0 程序运行环境。

四．实验报告要求

1. 列出海岸线度量问题产生的理论背景
2. 解释自相似原理以及举出生活中自相似图形的例子。
3. 列出海岸线度量的数学模型，并分析不同参数变化对于结果的影响。

实验五：H-分形（实验代码 5）

一、实验目的

了解 H-分形的应用背景，掌握 H-分形的生成原理，并能使用计算机生成 H-分形的图形。

二、 实验内容

1. 了解 H-分形产生的理论以及应用背景
2. 掌握 H-分形图形产生的基本方法及生成原理
3. 根据 H-分形图形产生的步骤，观察 H-分形图形，并自己模拟绘制 H-分形图形。

用图形表示二进小数：

我们的任务是将 0.0,0.10.01,0.11,0.001,0.011,0.101,0.111 在平面上表示出来。如图 9

步骤 1：选定 1 点，设想你站在此处，而且是面向正北方向。规定：当你“向左转、一步走、连直线”之后，你所在的位置是 0.0,而“向右转，一步走，连直线”之后的位置是 0.1

步骤 2：重复步骤 1，步长改为步骤 1 的一半。记住在 0.0 位置时，脸面朝向西。在 0.1 位置时，脸面朝向东。

步骤 3：记住面向，步长再缩短为前次的一半，重复动作“向左转、一步走、连直线”（落脚处对应的数，是前次的小数之尾巴加个 0），“向右转，一步走，连直线” （落脚处对应的数，是前次的小数之尾巴加个 1）。

步骤 4：按照这个规律进行若干次下去，即得到 H-分形的基本图形

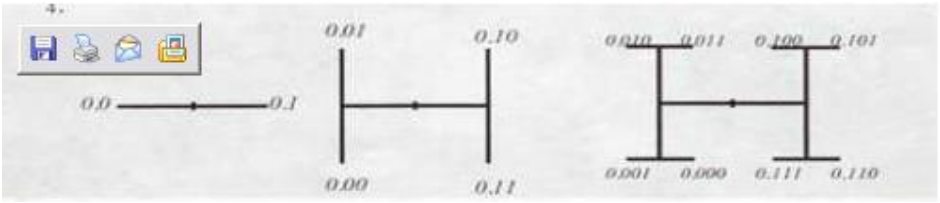


图 9

4．改进 H 分形图形，观察 H-分形的变形图形，（如图 10）分析其变化规律及基本原理

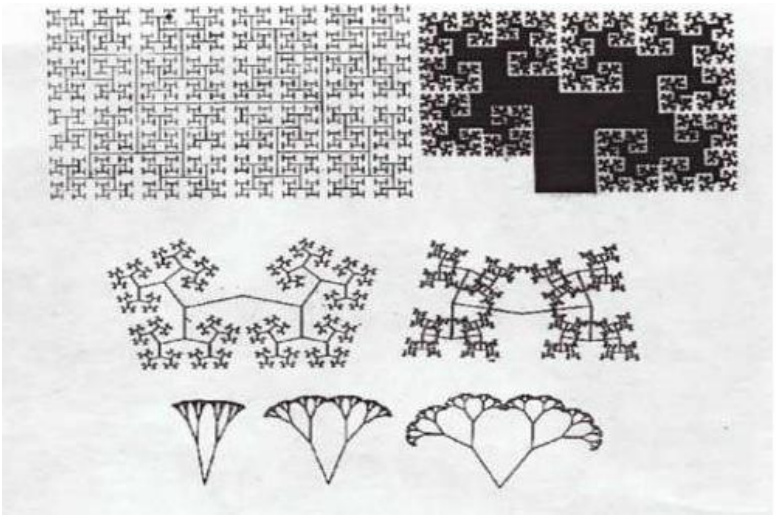


图 10

5．使用计算机模拟 H-分形图形产生过程。

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 VISUAL BASIC6.0 程序运行环境。

四、实验报告要求

1．给出 H-分形产生的理论以及应用背景
2．列出 H-分形的应用范围
3．列出 H-分形基本图形的生成方法。
4．根据 H-分形图形的生成算法，给出 H-分形的变形图形的生成方法。
5．列出使用计算机模拟 H-分形图形产生过程中所遇到的问题。

实验六：L 系统（实验代码 6）

一、实验目的

了解简单 L 系统的概念以及生成原理，用计算机生成 L 系统的分形图形。

二、实验内容

1. 通过例子，了解简单 L 系统的主要思想
2. 掌握简单 L 系统的两条普遍规则
3. 通过 L 系统的基本观念，观察 L 系统绘制的几种图形，如图 11

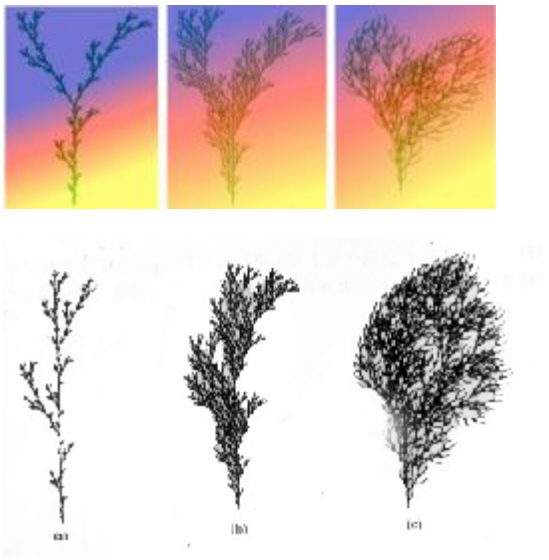


图 11 利用 L 系统绘制的树木图形

4、 了解 L 系统应用的范围和主要用途

5、 使用计算机模拟绘制 L 系统图形

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 VISUAL BASIC6.0 程序运行环境。

四、实验报告要求

1. 结合所给例子，叙述 L 系统的主要思想。
2. 给出 L 系统产生的背景条件和理论基础。
3. 列出绘制 L 系统的基本方法
4. 列出 L 系统所应用的范围
5. 列出自己模拟绘制 L 系统图形所遇到的问题。

实验七：IFS 系统（实验代码 7）

一、实验目的

掌握 IFS 基本算法思想，了解用 IFS 系统创作分形图形的原理，并且能够用 IFS 系统生成分形图形。

二、实验内容

1. 了解 IFS 系统基本概念和产生背景
2. 掌握 IFS 系统基本算法思想
3. 观察使用 IFS 系统绘制的基本图形，比较分析不同的参数设置对绘制图形形态的影响。如图 12

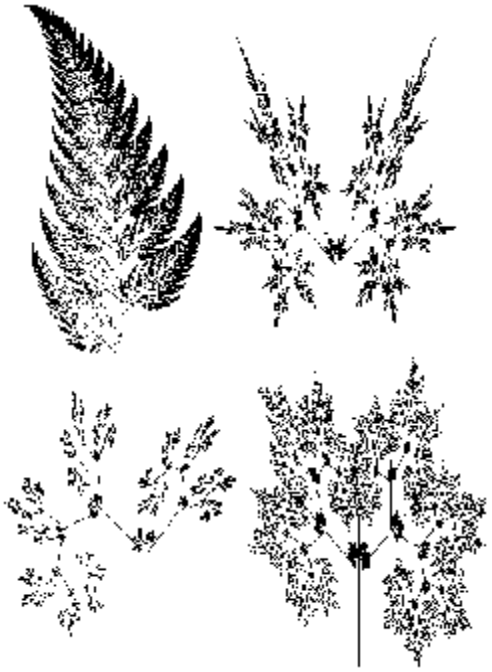


图 12 用 IFS 方法生成的植物形态

4. 参考用 IFS 方法生成植物形态的基本算法，自己可以自己改变参数，试验着生成各种精美的分形植物形态。

通用程序算法：Procedure AFF(a,b,c,d,e,f,S,T:real);

```
Var lins:real;
Begin
  lins:=a*S+b*T+e;
  y:=c*S+d*y+f;
  x:=lins;
End;
```

程序中不断调用此过程 AFF，即可完成迭代。

5. 运行 IFS 的计算机程序，并试着修改参数值，实现不同的图形形态。

```
{IFS 生成植物形态 PASCAL 程序}
Program FractalPlant28;
```

```
Uses Graph,Crt;
var
x,y,E,u:real;
Gd,Gm:integer;
Begin
  Gd:=VGA;{用 VGA 方式 640×480}
  Gm:=VGAHi;
  Initgraph(Gd,Gm,'D:\\PASCAL');{初始化图形}
  if GraphResult <> grOK then Halt(1);{若出错则停机}
  x:=0;y:=0;{赋初值}
  Randomize;{初始化随机数发生器}
  Repeat
    E:=Random(100);{产生一个介于 0 至 100 之间的随机数}
    if E < 1 then
    Begin
      x:=0;y:=0.16 * y;{用 R_1 迭代}
    End;
  if (E >=1) and (E < 86) then
    Begin
      u:=0.85*x+0.04*y;y:=-0.04*x+0.85*y+1.6;{用 R_2 迭代}
      x:=u;
    End;
  if (E >= 86) and (E < 97) then
    Begin
      u:=0.2*x-0.26*y;y:=0.23*x+0.22*y+1.6;{用 R_3 迭代}
      x:=u;
    End;
  if E >=97 then
    Begin
      u:=-0.15*x+0.28*y;y:=0.26*x+0.24*y+0.44;{用 R_4 迭代}
      x:=u;
    End;
  PutPixel(Round(50*x)+300,500-Round(50*y),2);{描点，绿色}
  Until KeyPressed;{按任意键退出}
  CloseGraph;{关闭图形方式}
End.
```

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 PASCAL 程序运行环境。

四、实验报告要求

1. 列出 IFS 系统基本概念和产生背景

2．列出 IFS 系统绘制不同图形的基本原理和方法

3．比较不同程序参数设置对于所绘制图形形态变化的影响

4．使用给出 IFS 程序绘制植物图形时所出现的问题

实验八：分形图像压缩技术（实验代码 8）

一、实验目的

了解分形图像压缩技术的原理，掌握分形图像压缩技术的几种典型算法，并且能根据简单的算法编制程序。

二、实验内容

1．了解分形图象压缩技术的基础：分形变换的基本原理

2．进一步了解局部迭代函数系统的理论以及它在分形图形压缩上的应用。

3．了解几种典型分形图象压缩技术的算法，如照相机算法，局部 IFS 的拼贴方法。

4．细化简单的算法，编制成计算机程序，在计算机上模拟。

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 PASCAL 程序运行环境。

四、实验报告要求

1．叙述分形变换的基本原理

2．给出局部迭代函数系统的基本理论以及在图形压缩上是如何应用的。

3．给出 1 到 2 种分形图象压缩技术的基本算法

4．给出 1 种分形图形压缩技术的计算机程序。

实验九：分形音乐（实验代码 9）

一、实验目的

在了解分形理论基本应用的基础上，了解分形音乐生成原理，并且能够使用计算机生成一段分形音乐。

二、实验内容

1．了解分形理论的应用范围

2．了解分形艺术产生的背景以及基本概念

3．观察几种分形艺术的图形

4．了解分形音乐生成的基本原理

5．使用计算机模拟生成一段分形音乐。

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 PASCAL 程序运行环境。

四、实验报告要求

1．列出分形艺术的基本概念以及构成。

2．给出几种典型的分形艺术图形或者音乐的生成原理以及例子

3．列出分形艺术应用的范围

4．给出一段分形音乐的计算机程序

实验十：资本市场的分形结构及其自组织临界性研究（实验代码 10）

一、实验目的

通过分形理论的基本特点，了解资本市场的分形结构，并进一步了解资本市场分形结构的研究意义及应用前景。

二、实验内容

1. 再次回顾分形理论的基本内容以及原理
2. 了解资本市场的基本特征与构成
3. 分析资本市场的分形结构
4. 分析资本市场的分形析结构对于研究的意义
5. 展望资本市场分形析结构的应用前景。

三、实验仪器、设备及材料

每个学生配备上机电脑一台，并安装基本图形浏览软件，安装 PASCAL 程序运行环境。

四、实验报告要求

1. 叙述基本分形理论
2. 叙述资本市场的基本特点
3. 列出资本市场的基本分形结构特征
4. 列出资本市场的分形结构对于研究的意义
5. 列举资本市场分形结构研究在未来的意义