NetLogo 4.0.2 用户手册

（简体中文版）

翻译：张发

2008 年 3 月---- Monkey 年 Horse 月

译者说明

我为什么要翻译 NetLogo 用户手册？

这几年我对复杂系统很感兴趣，了解、使用过一些复杂系统仿真工具。平心而论，NetLogo 并不是特别强大，但与其他工具相比非常容易使用。对于许多从事复杂系统研究的人来说，用它作为一个工具搞点研究是比较省事的。

以前我并没有要翻译 NetLogo 学习资料的想法。我本来认为做学术研究的人读点软件文档不成问题，而不做学术研究的人也用不着学习 NetLogo。后来我发现情况并非如此，有的学生使用 NetLogo 做东西，向我抱怨说英文帮助看着费劲，因此影响了研究进展。我想也许这是事实，作为中国人看中文总比看英文容易点吧。

因此本项目就是让那些时间宝贵，看英文不是那么顺畅的人学习 NetLogo 使用的。当然如果是从事学术研究的人，我的忠告是：还是要多看英文！

读者的法律责任

联系方式

如果本译作对你有所帮助，请考虑以下两点：

（1）如果你发现译文有错误或不当之处，望不吝赐教，本人将根据你的建议做出修改。

（2）如果你愿意参与翻译工作，请与我联系。我将根据翻译进展和你协调，以免重复翻译。你参与翻译的部分将标明你的个人信息。

我的电子邮件：Richter2000@163.com

致谢

这项不打粮食的工作之所以得以进行，需要衷心感谢以下人员：

（1）我的一个好朋友让我萌生了启动这项工作的想法（虽非直接，但确有关系）。

（2）感谢电视节目制作人员，他们那些充斥荧屏的不太吸引人的作品，让我能够放弃每天晚上 2-3 个小时的电视时间，用来从事这项工作，心里也不是那么痛苦。

（3）感谢我的父母，他们赐给我一个基本够用的脑袋，尤其是脑袋里那副质量过硬的牙齿。当我感到难以继续时，有牙可咬，还不至于咬坏！

词汇对照表（部分）

英文	中文
agent	主体
agentset	主体集合
turtle	海龟
patch	瓦片
interface	界面
procedure	例程
link	链
observer	观察者

NetLogo 简介

NetLogo 是一个用来对自然和社会现象进行仿真的可编程建模环境。它是由 Uri Wilensky 在 1999 年发起的，由连接学习和计算机建模中心（CCL）负责持续开发。

NetLogo 特别适合对随时间演化的复杂系统进行建模。建模人员能够向成百上千的独立运行的“主体”(agent)发出指令。这就使得探究微观层面上的个体行为与宏观模式之间的联系成为可能，这些宏观模式是由许多个体之间的交互涌现出来的。

NetLogo 可以让学生运行仿真并参与其中，探究不同条件下他们的行为。它也是一个编程环境，学生、教师和课程开发人员可以创建自己的模型。NetLogo 足够简单，学生和教师可以非常容易的进行仿真，或者创建自己的模型。并且它也足够先进，在许多领域都可以做为一个强大的研究工具。

NetLogo 有详尽的文档和教学材料。它还带着一个模型库，库中包含许多已经写好的仿真模型，可以直接使用也可修改。这些仿真模型覆盖自然和社会科学的许多领域，包括生物和医学，物理和化学，数学和计算机科学，以及经济学和社会心理学等。几个用 NetLogo 实现的基于模型的探究性课程正在开发。

NetLogo 提供了一个课堂参与式仿真工具，称为 HubNet。通过联网计算机或者一些如 TI 图形计算器这样的手持设备，每个学生可以控制仿真模型中的一个主体。详情见链接。
NetLogo 是一系列源自 StarLogo 的多主体建模语言的下一代。它基于我们的产品 StarLogoT，增加了许多显著的新特征，重新设计了语言和用户界面。NetLogo 是用 Java 实现的，因此可以在所有主流平台上运行（Mac,Windows,Linux 等）。它作为一个独立应用程序运行。模型也可以作为 Java Applets 在浏览器中运行。

产品特性：

你可以通过下面列表了解 NetLogo 的特点和所提供的功能。

系统：

跨平台：
可以在 Mac,Windows,Linux 等平台运行

语言：
完全可编程
简单语言结构
对 Logo 语言进行扩展支持主体
移动主体（海龟）在由静态主体（瓦片）组成的网格上移动

主体之间可以创建链接，形成聚集、网络和图
内置大量原语
双精度浮点数（IEEE 754）
运行过程在不同平台上完全可复现

环境：
用 2 维或 3 维模式查看模型
可伸缩、可旋转矢量图形
海龟和瓦片标签
可以进行运行中（on-the-fly）交互的命令中心
界面构建，包括按钮、滑动条、开关、选择器、监视器、文本框、注解、输出区
快进滑动条使你可以对模型进行快进和慢放
强大灵活的绘图系统
信息 Tab 页面用来解释模型
HubNet:使用联网设备进行参与式仿真
主体监视器用来监视和控制主体
输出输入功能（输出数据，保存、恢复模型状态，制作电影）
行为空间（BehaviorSpace）工具用来从多次运行中收集数据。
系统动力学建模

Web:
模型可以存为 applet 嵌入 web 页（注释：有些功能 applets 不能使用，例如有些扩展和 3 维视图）

版权信息

第三方许可证

更新历史：

用户反馈对我们设计和改进NetLogo非常有价值。我们希望听取你的意见。请把评论、建议和问题发送到feedback@ccl.northwestern.edu，Bug报告发送到bugs@ccl.nrthwestern.edu

版本 4.0.2（2007 年 12 月）

系统需求

NetLogo 可以运行在目前几乎所有计算机上。

如果你的 NetLogo 不能正常运行，发送错误报告到 bugs@ccl.northwestern.edu

系统需求：应用程序

Windows

NetLogo 可以运行在 Windows Vista,XP,2000,NT,ME 和 98 上

NetLogo 安装程序安装 Java 1.5.0，由 NetLogo 独占使用，不影响计算机上的其他程序。

Mac OS X

强烈推荐 Mac OS X 10.4(或以上)，10.3 或 10.2 也支持。

请运行软件更新以确保有最新的 Java。

其他平台

NetLogo 可以运行在安装了 Java 虚拟机 1.4.1 以上的任何平台上。1.5.0_13 以上更好。

通过运行提供的脚本程序 netlogo.sh 启动 NetLogo

系统需求：保存 Applets

NetLogo 模型存为 Java Applet 后可以运行在任何安装了 Java 1.4.1 以上的浏览器中。

系统需求：3 维视图

少数情况下一些老的、性能差的系统不能成功使用 3 维视图。试试看。

一些系统能使用 3 维视图但不能切换到全屏模式，这与图形卡有关。例如 ATI Radeon IGP 345 和 Intel 82845 可能不能工作。

Windows 用户关于 Java 的技术细节

多数 Windows 用户应选择捆绑了 Java 的 NetLogo 下载包。

有两个可能的原因使用没有捆绑 Java 的其他下载包：

1. 希望下载包较小，少占用硬盘空间
2. 因为某些特别的技术原因，你需要使用其他 Java 版本

如果你认为其他下载包适合你，请阅读下面的详细技术信息。

即使你已经安装了 Java，它也可能不能与 NetLogo 一起工作。

为了获得最佳性能，NetLogo 使用了 Java 虚拟机的一个特别选项“server”。JRE 默认

安装时没有这个选项，只有 JDK 有这个选项。

如果你不是 Java 开发人员，你可能使用的是 JRE，而非 JDK。

因此，如果你要用自己的 Java 虚拟机运行 NetLogo，你有两种选择：

1.确保你有完全的 JDK 而非 JRE。

2.或者你能编辑一个配置文件，让 NetLogo 与 JRE 一起工作。

我们不推荐选项 2，因为没有“server”选项使 NetLogo 运行特别慢。

如果你非要用选项 2，你就应这样做。你必须告诉 NetLogo 不要使用“server”虚拟机选项。首先，使用本页的下载包安装 NetLogo，然后使用文本编辑器如 NotePad 打开 NetLogo.4.0.2.lax，这个文件在 NetLogon 安装目录里。在附加 java 选项里去掉-server 选项。将这一部分：

```
# LAX.NL.JAVA.OPTION.ADDITIONAL
# ------------------------------
# don't load native libs from user dirs, only ours, also run server not client VM
lax.nl.java.option.additional=-Djava.ext.dirs= -server -Dsun.java2d.noddraw=true
```

改为：

```
# LAX.NL.JAVA.OPTION.ADDITIONAL
# ------------------------------
# don't load native libs from user dirs, only ours, also run server not client VM
lax.nl.java.option.additional=-Djava.ext.dirs= -Dsun.java2d.noddraw=true
```

再说一次，使用这种方法，NetLogo 性能会变差。

已知问题

联系我们

模型实例：聚会（Party）

这一部分让你思考什么是计算机建模以及如何使用它，也让你对 NetLogo 软件有所了解。我们推荐初学者从这里开始。

聚会

你是否参加过聚会，注意过人们是怎样聚集成小组的吗？你也可能注意到人们并非一直呆在一个小组里，而是走来走去。当个人走来走去时，小组就发生变化。如果你长期观察这种变化，你应该注意到模式的形成。

例如，在社交场合人们倾向于展示出与工作或家庭中不同的行为。那些在工作中信心满满的人可能在社交场合变得羞怯，而那些在工作中安静保守的人却可能与朋友发起聚会。

聚集模式也取决于聚会的性质。在某些场合，人们接受训练组织成混合小组，例如聚会游戏或校园活动。但在非结构化的气氛里，人们以更加随机的方式形成小组。

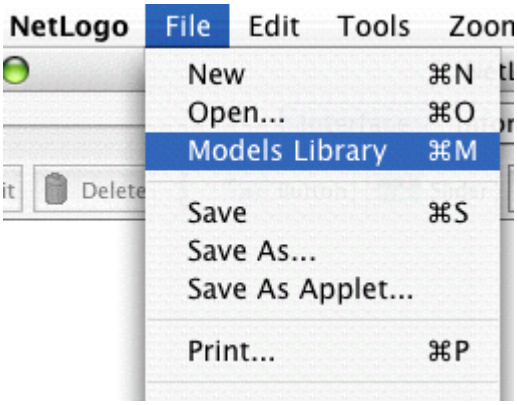
这种分组行为有没有什么模式呢？

让我们使用计算机对聚会中人们的行为建模，更详细的考察这个问题。NetLogo 的“Party”模型从性别这个特殊角度考察这个问题：为什么这些小组多数是男性，或多数是女性？

我们使用 NetLogo 研究这个问题。

操作步骤：

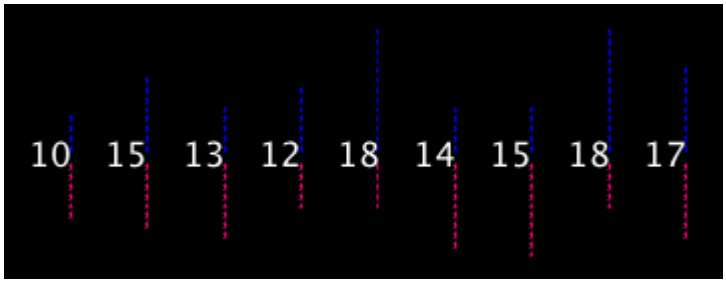
1. 启动 NetLogo
2. 在 File 菜单中选择 "Models Library"



3. 打开文件夹 "Social Science"
4. 点击模型 "Party".
5. 按下"open" 按钮
6. 等待模型加载
7. (可选)放大 NetLogo 窗口，这样能看多更多内容

8. 按下”Setup" 按钮

在视图中你可以看到粉线和蓝线，还有数字。



这些线表示聚会上男女混合的小组。男性用蓝色表示，女性用粉色。数字是每个小组的人数。

所有小组的人数相同吗？

所有小组的每种性别的人数相同吗？

例如你邀请了 150 人参加聚会，你想知道人们怎样扎堆。假设人们分成了 10 组。

你怎么思考分组情况？

这里我们使用计算机仿真，而不是去问你那 150 个亲密朋友。

操作步骤

1. 按下 "go" 按钮 (再次按下 "go" 会停止模型运行)
2. 观察人们的移动直到模型停止
3. 看图形输出了解发生了什么

现在每组有多少人？

开始时你可能认为将 150 人分成 10 组的结果是每组大约 10 人。从模型运行得知，人们并没有均等的分成 10 组——相反，有些组人数特别少，而有些组人数却特别多。另外，随着时间发展，从所有小组男女都有转变为所有小组均由同性组成。

这怎么解释？

对这个问题有很多可能的回答。本模型的设计者认为聚会上的小组不是完全按随机方式形成的。小组如何形成取决于个体的行为。模型设计者关注一个特殊变量“tolerance”（容忍度）：

这里将容忍度定义为个体感到舒服的异性的比例。如果小组中异性比例超过容忍度，他们就觉得不舒服，因此离开这一组去寻找别的小组。

例如，如果容忍度水平设为 25%，那么一个男性只有在女性比例少于 25%的小组里才感到舒服。同样女性只有在男性少于 25%的小组里才感到舒服。

当个体变得不舒服时选择离开，移动到别的小组，这可能又让这个组中的某些人不舒服。这种链式反应不断进行，直到聚会上的所有人都感到舒服。

注意这个模型中，容忍度不是固定的。用户可以用滑动条改变容忍度，重新运行模型，看看结果如何。

怎样重新启动模型：

1. 如果 "go" 按钮已按下 (黑色)，说明模型还在运行。再次按下该按钮停止运行。
2. 通过拖动红色手柄调整 "tolerance" 滑动条，设置新值
3. 按下"setup" 按钮重设模型
4. 按下"go" 按钮再次启动模型

挑战

作为聚会的主人，你希望看到各组里都是男女混合。调整容忍度滑动条，让每组都男女混合。

为保证 10 个小组都是男女混合，容忍度水平要设成多少?

看看你的预测结果吧。

你能想到可能影响每组中男女比例的其他因素或变量吗？

进行预测，用模型检验你的想法。放开手脚同时操作多个变量。

当你检验假设的时候，你会从数据中注意到模式的涌现。例如，如果保持聚会人数不变，但逐渐增加容忍度水平，更多的组会成为男女混合组。

容忍度水平必须是多少才能得到混合组？

容忍度水平与混合组的比例有什么关系？

用模型思考

用 NetLogo 对聚会这样的情景建模使你可以对系统进行快速、灵活的试验，而在现实情况下这是很困难的。建模也给了你少受偏见的影响去观察各种情景的机会，因为你可以检查系统内部的动态。你会发现随着你建模越来越多，对许多现象的原有的想法会遇到挑战。例如 Party 模型一个令人惊讶的结果是:即使容忍度水平相对较高,大量性别分离仍然会发生。

这是关于“涌现”现象的一个经典例子，这里小组模式是许多个体交互的结果。“涌现”思想可以应用在几乎任何领域。

你能想到别的涌现现象吗?

要想获得更多的例子，对这个概念有更深的理解，你可以探索 NetLogo 模型库，里面有许多模型，演示了各种系统中的这类思想。

要更详细了解关于涌现的讨论以及NetLogo如何帮助学习者进行探索，参见"Modeling Nature's Emergent Patterns with Multi-agent Languages" (Wilensky, 2001).

下一步做什么？

用户手册的 教学 1：运行模型 讲述模型库的更多细节。

如果你想学习怎样在更深的层次上探索模型，教学 2：命令 将引导你了解 NetLogo 建模语言。

最后，你将学习 教学 3：过程，你将学习怎样替换、扩展模型，增加新行为，以及如何建造自己的模型。

教学# 1：模型（Models）

如果你已经读过 模型实例：聚会 部分，你应该简单了解了怎么与 NetLogo 模型交互。这一部分更深入的了解一些功能，这些功能在你探索模型库时会用到。

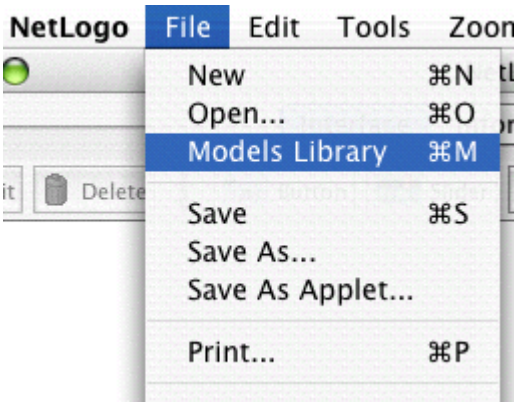
在整个教学过程中我们会请你做一些预测，预测修改模型后会出现什么后果。记住后果往往令你惊讶。我们认为这种惊讶非常令人激动，提供了特棒的学习机会。

有些人发现把这些教学材料打印出来放在手边，对学习很用帮助。打印出来后，你的计算机屏幕上有更大的空间显示你要查看的 NetLogo 模型。

模型实例:狼吃羊（ Wolf Sheep Predation）

我们要打开一个模型实例详细探索。我们试试一个生物模型：狼吃羊，这是一个掠食-食饵种群模型。

- 从文件菜单打开模型库



- 从 Biology 部分选择"Wolf Sheep Predation" 按下"Open"

界面标签页填满了许多按钮、开关、滑动条和监视器。这些界面元素使你可以与模型交互。按钮是蓝色的，用它们设置、启动、停止模型。滑动条和开关是绿色的，它们用来修改模型配置。监视器和绘图是浅褐色的，它们用来显示数据。

如果你想让窗口大一些，让所有元素都能容易的看到，你可以使用窗口顶部的 zoom 菜单。

当你第一次打开模型时，你会看到视图是空的（全黑）。要让模型开始，你需要先设置它。

- 按下 "setup" 按钮

视图中出现什么？

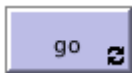
- 按下 "go" 按钮开始仿真

模型运行时，狼群和羊群发生什么？

- 按下"go" 按钮停止运行

控制模型：按钮

按钮按下后模型就会通过执行一个动作做出响应。按钮分为“一次性”（once）和“永久性”(forever)两种。可以通过按钮上的一个符号区分二者。永久性按钮的右下角有两个箭头，就像这样：



一次性按钮没有箭头，就像这样:

一次性按钮执行动作一次然后停止。当动作完成后，按钮弹起。

永久性按钮不断的执行一个动作。当你想让动作停止时，再次按下按钮。它会完成当前动作，然后弹起。

大多数模型，包括狼吃羊模型，有一个一次性按钮称为“setup”和一个永久性按钮称为“go”。许多模型还有一个一次性按钮称作“go once”或“step once”,它们很像 go 按钮，但区别在于它们只执行一步（时间步长）。使用这样的一次性按钮能让你更仔细的查看模型的运行过程。

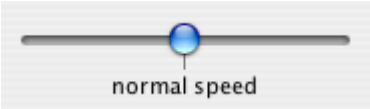
停掉永久按钮是终止模型的正常方式。通过停止永久性按钮暂停模型运行，然后再次按下按钮让模型继续，这非常安全。你也可以使用 Tools 菜单的“Halt”停止模型运行，但是只有当模型因某种原因卡住时才应该这样做。使用“Halt”可能会让模型在某次行动的中间停住，这可能导致模型乱套。

- 如果你愿意的话，试试狼吃羊模型的 "setup" 和 "go" 按钮

如果使用同样的设置多次运行模型，结果会有不同吗？

控制速度:速度滑动条

速度滑动条控制模型运行速度，就是海龟的移动速度、瓦片颜色改变的速度，等等



滑块左移使模型速度变慢，每个时间步之间的暂停时间更长。这样更容易观察发生了什么。你甚至可以让模型运行的极慢，能看到每个海龟做什么。

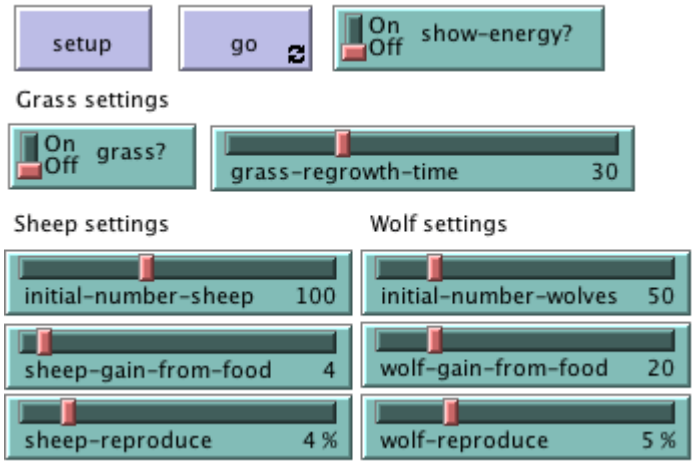
滑块右移使模型速度变快。NetLogo 可能会跳帧，意思是说不再是每个时间步都进行视图刷新。显示世界状态要耗费时间，因此少显示这些一般意味着运行更快。

注意如果滑块太靠右的话，视图更新太频繁看起来却好像变慢了。实际没有变慢，你可以从时钟显示确认这一点。只有视图更新频率降低了。

调整设置：滑动条和开关

模型的配置给了你尝试不同场景或假设的机会。修改配置运行模型，观察这些改变所引起的反应，使你能更深入的了解所模拟的现象。开关和滑动条用来修改模型配置。

下面是狼吃羊模型中的开关和滑动条：



我们来试试这些行为的效果。

- 如果狼吃羊模型还没打开，现在打开它
- 按下"setup" 和 "go" ，运行大约 100 时间步 (注意：图的右上有时钟读数)
- 按下"go" 停止

羊群怎么变化？

我们看看如果改变下面的设置的话，羊群怎么变化。

- 打开"grass?" 开关
- 按下"setup" 和 "go" ，运行与上次差不多相同的时间

这个开关对模型有什么作用？结果和上次一样吗？

像按钮一样，开关也有与它相连的信息。这些信息采用开/关格式。开关发出特别的指令，这些指令对模型并非必要，但为模型增加了附加的维度。打开"grass?"影响模型结果。本次运行之前，草的增长率为常数。在掠食-食饵关系中这是不真实的，因此通过设置和打开草的增长率，我们能够对三个因素建模：羊、狼和草。

另一种配置类型是滑动条。

滑动条是不同于开关的一种配置类型。开关有两个值：开或关。滑动条是一个可调的数值范围。例如"initial-number-sheep"滑动条最小值为 0，最大值为 250。模型运行时可以有 0 只羊，也可以有 250 只羊，或者中间的任何一个数值。试试看。当你从左到右移动滑块时，滑动条右侧的数字变化，这就是当前值。

我们研究一下狼吃羊模型的滑动条。

- 阅读信息标签页中的说明，了解每个滑动条表示什么.

信息标签页提供了模型的指导和洞察。在这一页你会找到模型的解释，尝试建议以及其他信息。你可以在运行模型前读它，也可以先试试模型，再返回来读它。

如果仿真开始时羊更多而狼更少，会怎么样？

- 关掉"grass?" 开关
- 设置"initial-number-sheep" 滑动条为 100.
- 设置"initial-number-wolves" 滑动条为 20.
- 按下 "setup" 和 "go".
- 模型运行约 100 时间步

尝试重复运行模型几次。

羊群数量发生了什么变化？

结果令你惊讶吗？调整哪些其他开关、滑动条能帮助羊群?

- 设置 "initial-number-sheep" 为 80 ，"initial-number-wolves" 为 50.(这与你第一次打开模型时接近)
- 设置"sheep-reproduce" 为 10.0%.
- 按下"setup" 和 "go".
- 模型运行约 100 时间步

本次运行狼群怎么样？

当你打开模型时，所有的滑动条和开关采用默认配置。如果你打开一个新模型或退出程序，你的更改不会保存，除非你选择保存它们。

(注意：除了滑动条和开关，一些模型还有第三类配置元素，选择器（Chooser）。但本模型没有。)

收集信息：绘图和监视器

建模的一个目的是对那些难以在实验室中进行研究的问题收集数据。NetLogo 主要有两

个显示数据的方式：绘图和监视器。

绘图（Plots）

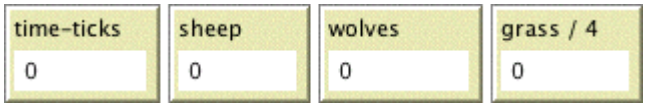
狼吃羊中的图有三条线：羊、狼和草/4（草除以 4 的原因是为了别使图形太高）。这些线显示了随着时间推进模型中发生了什么。要想知道每条线代表什么，在图形窗口的右上角单击”Pens”，打开画笔图例。一个关键字说明了每条线是什么。在本例中就是种群数量。

当图快被充满时，水平轴增加，以前的数据被压缩只占一部分空间，更多的空间用来绘制将来的图形。

如果你想保存图上数据以备查看或在另一个程序里进行分析，使用 File 菜单的"Export Plot"。这些数据就被保存，数据格式可以被电子表格，如 Excel,或数据库程序识别。也可以通过 Ctr+单击（Mac）或右击(Windows)弹出快捷菜单，然后选择"Export..."。

监视器（Monitors）

监视器是模型显示信息的另一种方法。下面是狼吃羊模型中的监视器：



监视器"time-ticks"告诉我们仿真时间。其他的监视器告诉我们狼、羊、草的数量。（记住，草的数量除以 4，为了别使图形太高）

当模型运行时监视器中的数值不断更新，而图形能显示模型整个运行过程中的数据。

注意 NetLogo 还有另一种监视器，叫做"agent monitors"，在教学 2 里介绍。

控制视图

如果观察界面标签页，会看到沿工具条上边缘有一条控件。这些控件改变视图的不同方面。

试试这些控件的效果。

- 按下 "setup" 和 "go" 启动模型
- 模型运行时，将速度滑动条向左移动

发生什么？

如果模型运行的太快，你可以使用它看清细节。

- 移动速度滑动条到中间
- 右移速度滑动条
- 现在试试勾选或不勾选“view updates”选择框

发生什么？

如果你不耐烦，想让模型运行的更快，可以快进也可以关闭视图更新。快进（速度滑动条右移）关闭视图更新，因此模型运行的更快，这是因为更新视图需要时间。

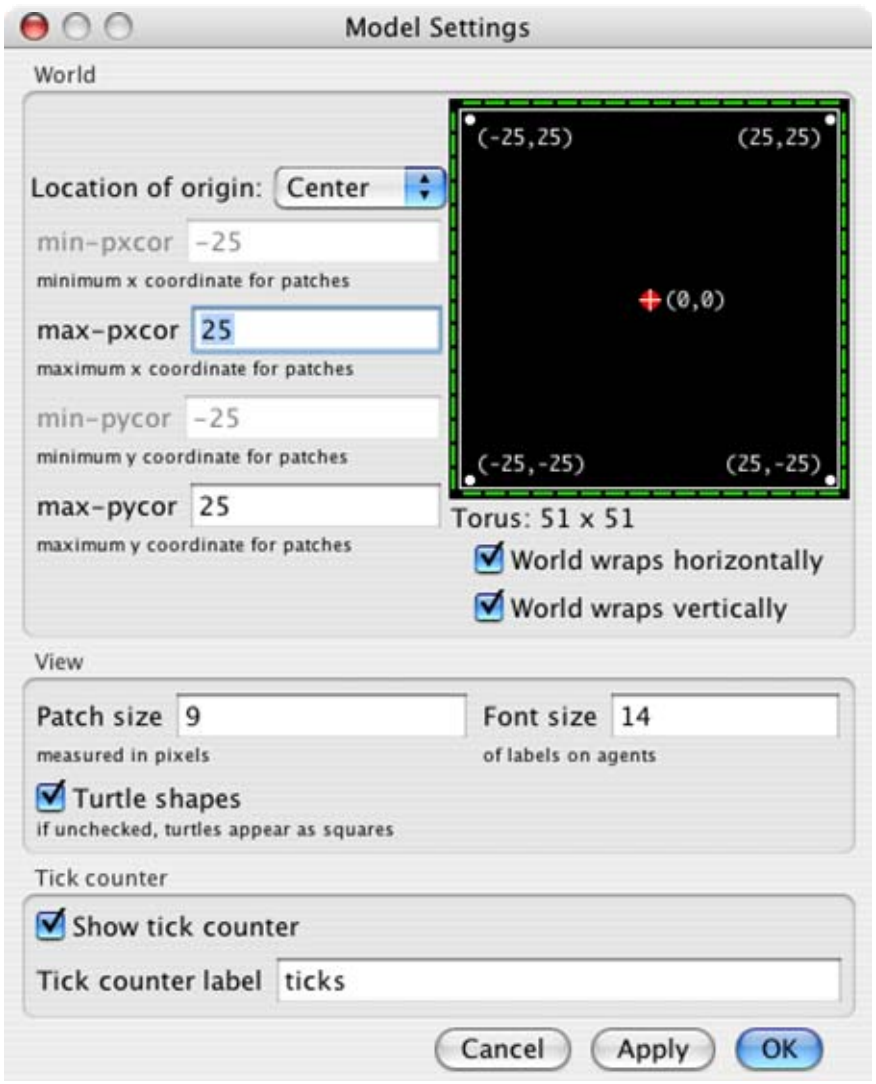
当视图更新完全关闭后，模型继续在后台运行，绘图和监视器也一直在更新。如果你想看看在发生什么，你需要重新勾选视图更新选项。当视图更新关闭后，多数模型运行速度更快。

视图的尺寸由 5 个设置共同决定：最小 X、最大 X，最小 Y、最大 Y 和瓦片尺寸。现在让我们看看当我们改变狼吃羊模型视图的尺寸时发生什么。

有更多的关于世界和视图的设置，因为工具条面积有限，没有放上。"Settings..."按钮可以让你获得其他设置。

- 按下工具条上的"Settings..." 按钮

会打开一个对话框，其中包括所有视图设置：



当前的 max-pxcor, min-pxcor, max-pycor, min-pycor 和 Patch size 是多少?

- 按下 "cancel" 按钮取消所做的改变。
- 将鼠标指针靠近视图，但不要进入视图窗口

注意到鼠标指针变成了十字型

- 按着鼠标按钮在视图上拖动

现在视图被选中，看到视图被灰框环绕，你可以确知这一点。

- 拖动黑色方块型“手柄”。手柄在视图的边上和角上。
- 在界面标签页的白色背景的任何地方单击，反选视图
- 再次按下"Settings..." 按钮，看看设置

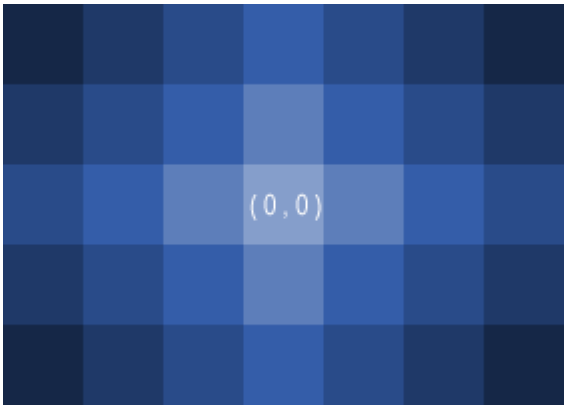
哪些数字改变了？

哪些数字没有改变？

NetLogo 世界是由“瓦片”构成的二维网格，瓦片是网格中的一个方格。

在狼吃羊模型中，当"grass?"开关打开时瓦片很容易看到，因为一些瓦片是绿色的，一些是褐色的。

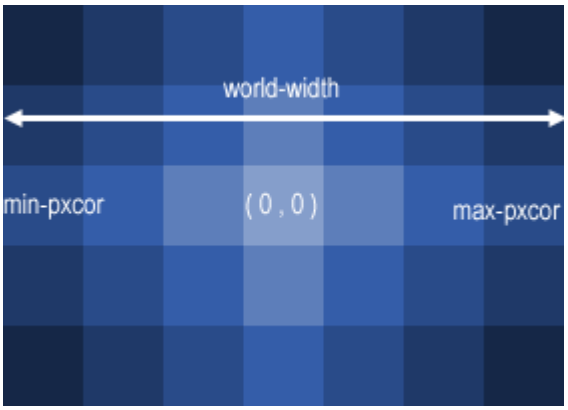
可以把瓦片想象成地板上铺的方形瓷砖。默认情况下房子正中的一片瓷砖标记为(0,0)，意味着如果我们在水平和垂直方向画等分线，交叉点这在此处。这样我们就有了一个在房间中定位对象的坐标系统：

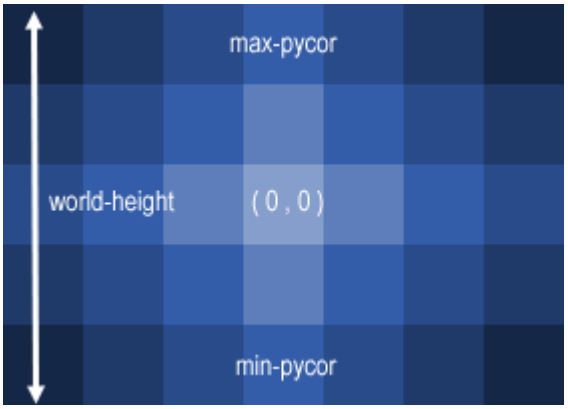


瓷砖(0,0)与房间右侧之间有多少瓷砖？

瓷砖(0,0)与房间左侧之间有多少瓷砖？

在 NetLogo 中，从右到左的瓷砖数称为世界宽度（world-width）。从顶到低的瓷砖数称为世界高度（world-height）。这些数字由顶、低、左、右边界(top, bottom, left and right)来定义。





在这些图中，max-pxcor 是 3，min-pxcor 是 -3, max-pycor 是 2 ，min-pycor 是 -2. 当你改变瓦片大小时，瓦片（瓷砖）的数量不变，只是屏幕上瓦片的大小变化了。
让我们看看改变世界的最小、最大坐标的效果。

- 使用仍在打开的 Settings 对话框, 改变 max-pxcor 为 30，max-pycor 为 10。注意 min-pxcor 和 min-pycor 也变了，这是因为默认原点 (0,0)在世界的中心。

视图的形状发生了什么变化?

- 按下 "setup" 按钮

现在可以看到你创建的新瓦片

- 再次按下"Settings..." 按钮
- 将瓦片大小设为 20，按下"OK".

视图的大小发生了什么变化？它的形状变了吗？
编辑视图也可以让你改变其他设置，包括标签字体大小\视图是否使用形状（shape）。随便试试这些设置吧。
当你探索完狼吃羊模型后，你可能想花点时间探索一下模型库中的其他模型。

模型库

模型库包括五部分： Sample Models, Perspective Demos, Curricular Models, Code Examples, HubNet Computer Activities.

模型样例（Sample Models）

Sample Models 部分是分科目组织的，目前有 210 多个模型。我们一直在增加模型，因此过段时间后能看到新加的模型。

有些文件夹下包含"(unverified)"子文件夹。这些模型是完整、可用的，但模型的内容、精度、代码质量等仍在评审之中。

透视演示（Perspective Demos）

这些模型在 Sample Models 中也有。但是略作修改，用来演示 NetLogo 的透视功能。

课程模型（Curricular Models）

这些模型是西北大学 CCL 开发的在学校使用的课程。有些模型在 Sample Models 中也有，有些没有。看看信息标签页，了解更多的信息。

代码例子（Code Examples ）

这是 NetLogo 特别功能的一些简单演示。当你以后扩展现存模型或新建模型时很有用。例如，你想在模型中增加直方图，可以看看"Histogram Example"，看看怎么做。

HubNet 计算机活动(HubNet Computer Activities)

这一部分包括教室中使用的参与式仿真。要了解HubNet的更多信息，参见 HubNet Guide.

下一步?

如果想在更深的层次上探索模型，教学 2：命令 将引导你了解 NetLogo 建模语言。

在 教学 3：例程 中，你学习怎样替换现有模型，以及如何构建你自己的模型。

教学# 2：命令（Commands）

在教学#1，你有机会查看了一些 NetLogo 模型，我们引导你打开、运行模型、按下按钮、改变滑动条和开关值，以及使用绘图和监视器从模型收集信息。在这一部分，焦点从观察模型转换到操纵模型。你将开始看模型的内部运转，能够改变它们的样子。

模型实例：基本交通模型(Traffic Basic)

- 到模型库去 (File 菜单).
- 在"Social Science"部分，找到并打开 Traffic Basic
- 运行模型两分钟，感受一下
- 如果有什么问题，去信息标签页查查

在这个模型里，你会注意到一系列蓝车里有一辆红车，车流同向移动。这些车时不时的会挤成一堆，无法移动。这是关于幽灵式阻塞的模型，即有时交通流会出现阻塞，但却找不到任何明显的原因，例如事故、断桥、侧翻的卡车等。要形成交通阻塞，没有“明显原因”（centralized cause）是需要的。

你可以改变配置运行几次，对模型有个全面理解。

当你使用这个 Traffic Basic 模型时，有没有注意到需要给模型增加什么?

看看这个模型，你会注意到环境太简单了，就是黑色背景、白色街道、一些蓝车和一辆红车。需要对模型做点改变：改变车的形状和颜色、加上房子或路灯、新建信号灯、或者再创建一条车道。这些建议有些是装饰性的，只是改善模型的观感，另外一些是行为性的。在本教程里我们主要关注较简单的、装饰性的改变。（教学#3 深入介绍行为性改变，那需要在例程页面中进行修改）

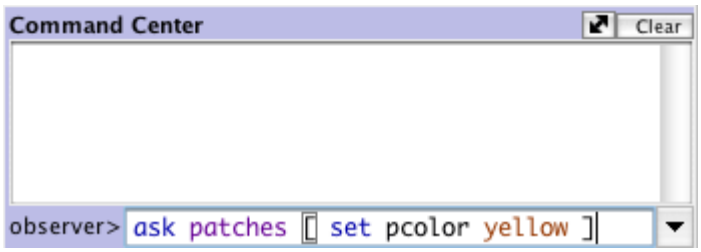
我们使用命令中心做这些简单改变。

命令中心（The Command Center）

命令中心位于界面Tab页,在这里你可以向模型发出命令或指令。命令就是你可以发给NetLogo主体（海龟、瓦片、链、观察者）的指示。（参见界面指南Interface Guide，了解命令中心的各个部分）

在 Traffic Basic 中:

- 按下"setup" 按钮
- 找到命令中心
- 在命令中心底部的白框里按一下鼠标
- 输入下面所示的文本



- 按回车键

视图发生什么变化?

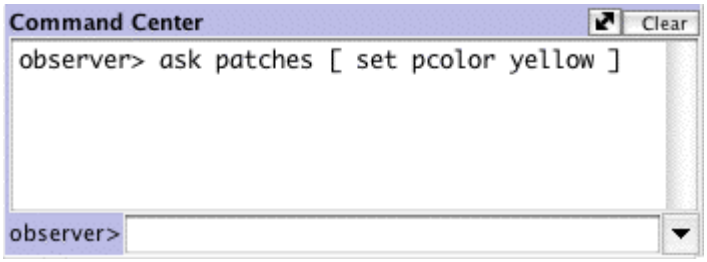
你注意到视图背景变成了黄色，街道消失了。

为什么车没有变成黄色?

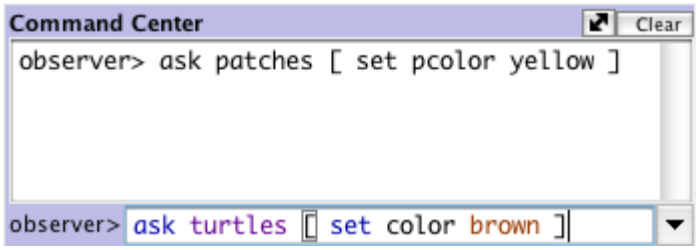
回头看看刚才输入的命令，我们只是请求瓦片改变颜色。在这个模型里，车辆用另外一种称为”海龟”的主体表示。因此车辆并没有接收指令，也就没有改变。

命令中心发生了什么?

你可能没注意到你刚才输入的命令现在显示在命令中心的中间部分白框里，就像下面这样：

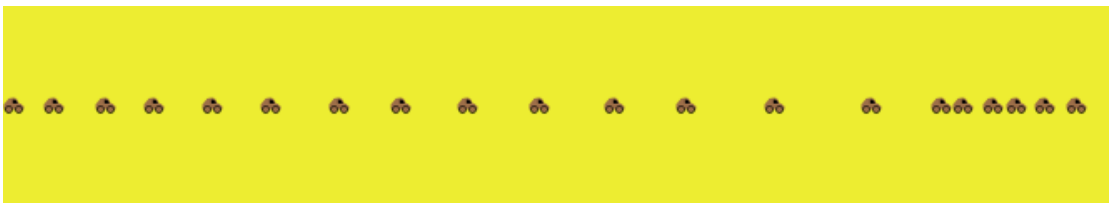


- 在命令中心底部的白框中输入下面的文本:



结果和你想的一样吗?

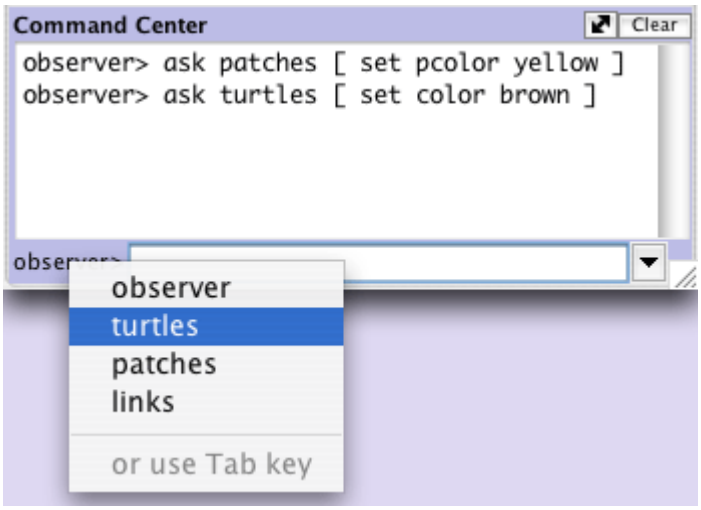
视图里是黄色背景，中间是一串灰色的车:



NetLogo是由海龟、瓦片和观察者组成的二维世界。瓦片构成背景，海龟在背景上移动，观察者（observer）是观察着所有事情的一个生命体。（关于世界的细节，参考 NetLogo 编程指南NetLogo Programming Guide）

在命令中心，我们可以给海龟、瓦片和观察者发出命令。我们通过命令中心左下角的弹出式菜单进行选择,也可以用 Tab 键在选项之间循环。

- 在命令中心,单击左下角的"observer>":



- 在弹出菜单中选 "turtles"
- 输入 set color pink ，回车
- 按下 tab 键直到在左下角看到"patches>"
- 输入 set pcolor white，回车.

现在视图看起来怎么样?

你注意到这两条命令和前面的 observer 命令的区别了吗?

观察者（observer）俯视着世界，因此使用 ask 向瓦片或海龟发出命令。正如第一个例子那样(observer> ask patches [set pcolor yellow])，observer 必须请求（ask）瓦片把它的颜色 pcolor 设为黄色。但在第二个例子中，命令直接发给了一组主体(patches> set pcolor white)，你只需直接给出命令。

- 按下"setup".

发生什么?

为什么视图变回了原样，还是黑背景白路？因为按下"setup"后，模型重新按例程页中的内容配置模型。命令中心一般不用来对模型做永久性修改,而是用来对当前模型进行定制，让你能操纵模型，回答探究模型时冒出来的“what if”问题。(例程页（Procedures tab）在下个教学里解释，也可参考编程指南Programming Guide.)

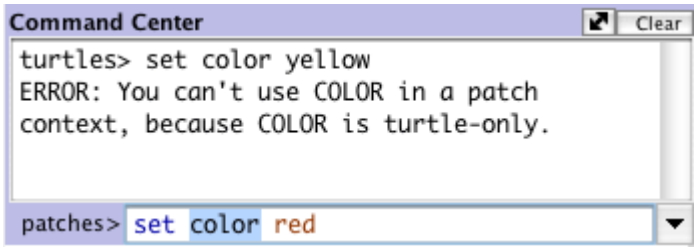
我们已熟悉了命令中心，再看看 NetLogo 中关于颜色的一些细节。

操纵颜色

你可能注意到在上面我们使用了两个不同的词来改变颜色：color与 pcolor.
color 与 pcolor 有何区别？

- 在命令中心的弹出菜单中选 "turtles" (或使用 tab 键).
- 输入 set color blue，回车

车辆发生什么变化?
思考一下你做了什么让车变成了蓝色，试试把瓦片变成红色。
如果想让瓦片变成红色，出现一条错误信息：



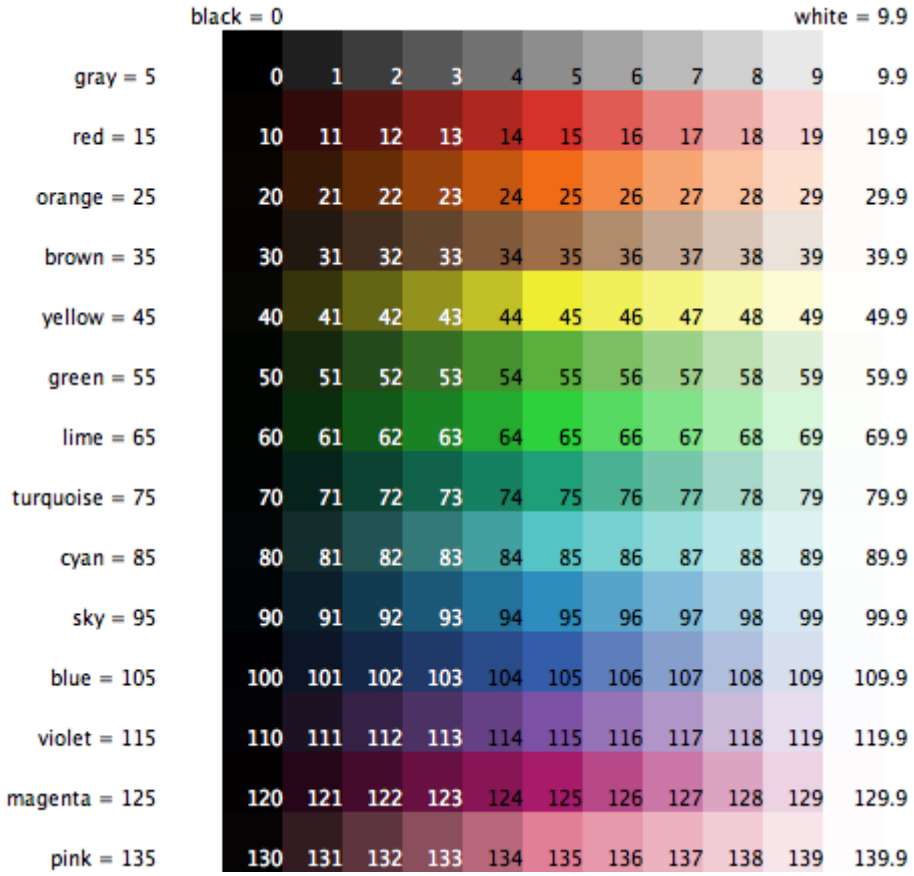
- 输入 set pcolor red，回车.

color 和pcolor是变量（variables）。有些命令和变量是海龟专用的，有些是瓦片专用的。例如，color是一个海龟变量，而pcolor是一个瓦片变量。

继续尝试，使用set命令和这两个变量改变海龟和瓦片颜色。

为了能对海龟和瓦片做更多的颜色改变，也就是车辆和背景，我们需要了解 NetLogo 如何处理颜色。

在NetLogo所有颜色对应一个数值。在这些练习里我们使用了颜色名，只是因为NetLogo认识16个不同的颜色名。这并不意味NetLogo只能分辨16种颜色，这些颜色之间的中间色也可使用。下面是NetLogo颜色空间的一张图：



为得到一个没有名字的颜色，你需要使用一个数值，或者在颜色名上加上或减去一个数。例如，输入 set color red 与输入 set color 15 效果完全一样。要得到一个更浅或更深的颜色，只需使用一个比该颜色更小或更大的一个数。如下所示：

- 在命令中心的弹出菜单选 "patches" (或使用 tab 键).
- 输入 set pcolor red - 2 ("-" 两侧的空格很重要)

通过在 red 上减去一个数，得到更深的颜色。

- 输入 set pcolor red + 2

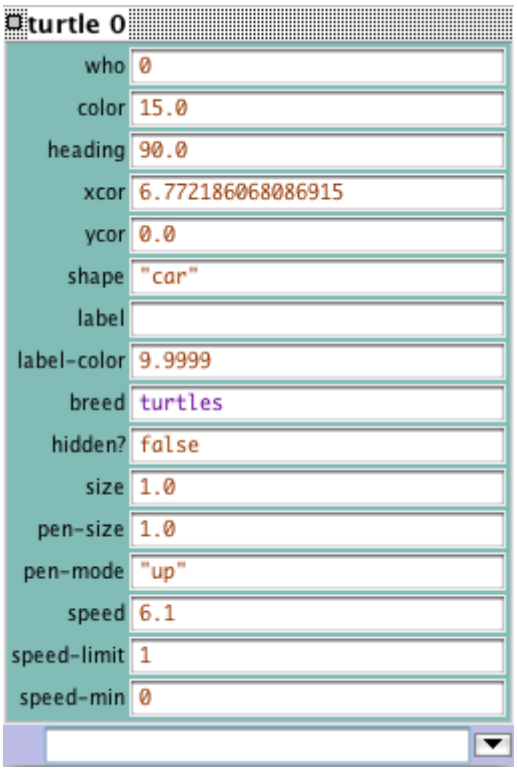
通过在 red 上加上一个数，得到更浅的颜色。
图上任何颜色均可采用这种方法。

主体监视器（Agent Monitors）和主体命令器（Agent Commanders）

上面我们使用 set 命令改变所有车辆的颜色。你是否记得，最初的模型中一群蓝车里有一辆红车。现在看看怎样只改变一辆车的颜色。

- 按下 "setup" 让红车再次出现
- 如果使用 Macintosh，按下 Ctrl 在红车上单击。其他操作系统的话，在红车上右击
- 如果有别的海龟与红车太近，你可能看到在菜单底部列出多个海龟。将鼠标移动到海龟菜单项上，注意到菜单和视图中海龟都加亮了。在红色海龟项的子菜单中选 "inspect turtle"。

关于那辆车的一个海龟监视器出现了:



仔细看看海龟监视器，可以看到属于红车的所有变量。变量是存储数值的，可以改变。还记得我们说过颜色在计算机里都是用数字表示的吗？对主体也一样。例如，每个主体都有一个 ID 号，叫做"who number"。

再看看海龟监视器。

海龟的 who number 是多少?

海龟颜色是什么？

海龟形状是什么？

这个监视器显示该海龟的 who number 是 0，颜色 15（红色），形状是“ car”。

除了右击（mac 上是 Ctrl+单击）外，还有两个方法打开海龟监视器。方法 1 是从 Tools 菜单选择"Turtle Monitor"，然后在“who”域中输入要查看的海龟的 ID，回车。另一种方法是在命令中心输入 inspect turtle 0（或其他 ID）。

要关闭海龟监视器，只需在窗口左上角关闭标志（ Macintosh）或右上角关闭标志（其他操作系统）上单击。

现在了解了主体监视器，有三种方式改变一个海龟的颜色。

第一种是使用主体监视器底部的主体命令器（Agent Commander）。在这输入命令，就

像在命令中心一样，只是在这输入的命令只由这个海龟执行。

- 在海龟 0 主体监视器中的主体命令器 中输入 set color pink.

视图发生什么变化?
海龟监视器有什么变化吗?

第二种是直接改变海龟监视器中的 color 变量

- 在海龟监视器中选择 "color" 右侧的文本.
- 输入新颜色，如 green + 2.

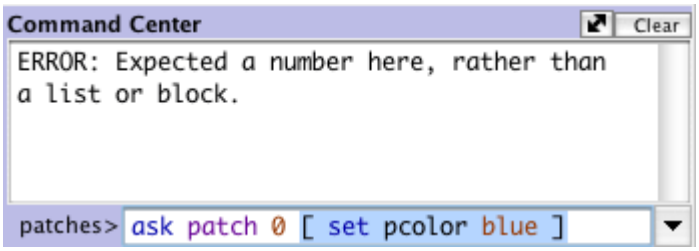
发生什么?

第三种是使用观察者(observer)改变海龟或瓦片的颜色。因为 observer 俯视着 NetLogo 世界，它可以发出命令，影响单个或一组海龟。

- 在命令中心的弹出菜单，选择 "observer" (或使用 tab 键).
- 输入 ask turtle 0 [set color blue] ，回车

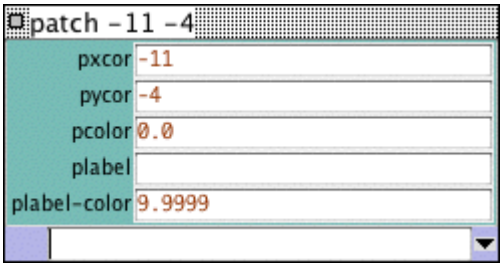
发生什么?

除了海龟监视器（Turtle Monitors）,也有瓦片监视器（Patch Monitors）。 瓦片监视器与海龟监视器很相似。
你能使用瓦片监视器改变单个瓦片的颜色吗?
如果你让观察者（observer ）请求瓦片 0 改变颜色（ask patch 0 [set pcolor blue]），会出现一条错误信息。



要让某个海龟做什么，我们使用 who number。但瓦片没有 who number，需要其他方法。
记住，瓦片存在于一个坐标系统中。要在图上画个点需要两个数：x 坐标和 y 坐标。瓦片的定位方式与此相同。

- 随便选一个瓦片，打开瓦片监视器



监视器表明这个瓦片的pxcor变量是-11，pycor 是 -4。在坐标平面上，这个点处于左下象限。

使用坐标让这个特定的瓦片改变颜色。

- 在瓦片监视器的底部，输入 `set pcolor blue`，回车.

在海龟或瓦片的监视器中输入命令只对这个海龟或瓦片管用。

在命令中心也可操作单个瓦片：

- 在命令中心输入 `ask patch -11 -4 [set pcolor green]` ，回车

下一步?

此时也许你想打开模型库的其他模型，试试刚学的这些技术。

在 教学#3:例程 你将学习怎样替换、修改已有模型，或构建自己的模型。

教学 #3：例程（Procedures）

本教程带你一步步建立一个完整的模型，对每一步做出解释。

主体和例程

在教学#2 你学习了怎样使用命令中心和主体监视器查看和修改主体，让他们执行动作。现在准备进入 NetLogo 模型的真正核心：例程页。

你已经使用了 NetLogo 中可以执行命令的主体类型：瓦片、海龟、链和观察者。瓦片是静止的，组成网格。海龟在网格上移动，链链接两个海龟。观察者俯视在进行的所有事情，做那些海龟、瓦片和链自己不能做的事情。

所有这四种主体都能执行 NetLogo 命令。前三种主体还能运行例程（procedures）。一个例程包括一系列 NetLogo 命令，你将它们定义为一个单一的新命令。

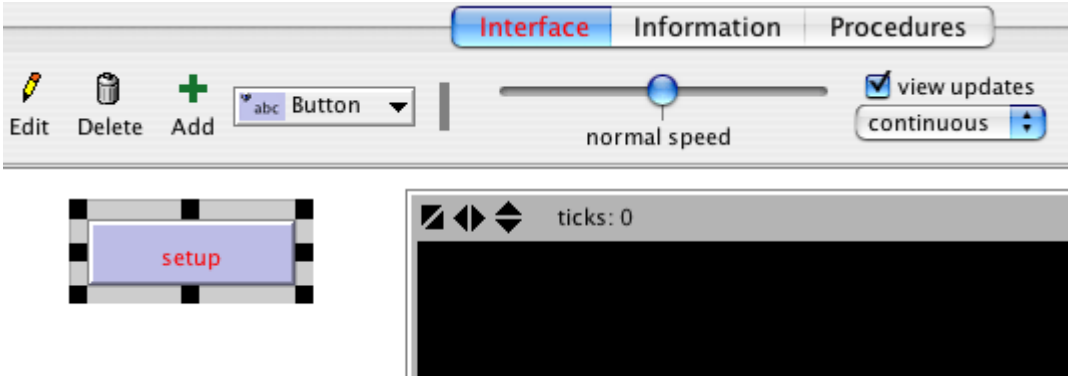
你将学习编写例程，让海龟移动、进食、繁殖和死亡。还将学习如何制作监视器、滑动条和绘图。我们要建立一个简单的生态系统模型，与教学#1 的狼吃羊模型部分相似。

制作 setup 按钮

要开始一个新模型，在 File 菜单中选择 New。然后从创建一个 setup 按钮开始：

- 在界面页上部的工具条上单击 "Button" 图标
- 在界面页的空白区域，定位到你想放置按钮的地方单击
- 编辑按钮的对话框出现了。在标签为"Commands"的文本域中输入 setup
- 按下 OK 按钮，对话框关闭

现在有了一个 setup 按钮。按下按钮就执行一个名为"setup"的例程。例程就是一系列的 NetLogo 指令，我们给定一个新名字。现在还没有定义例程（一会就做）。由于按钮指向的例程现在还不存在，按钮变成了红色：



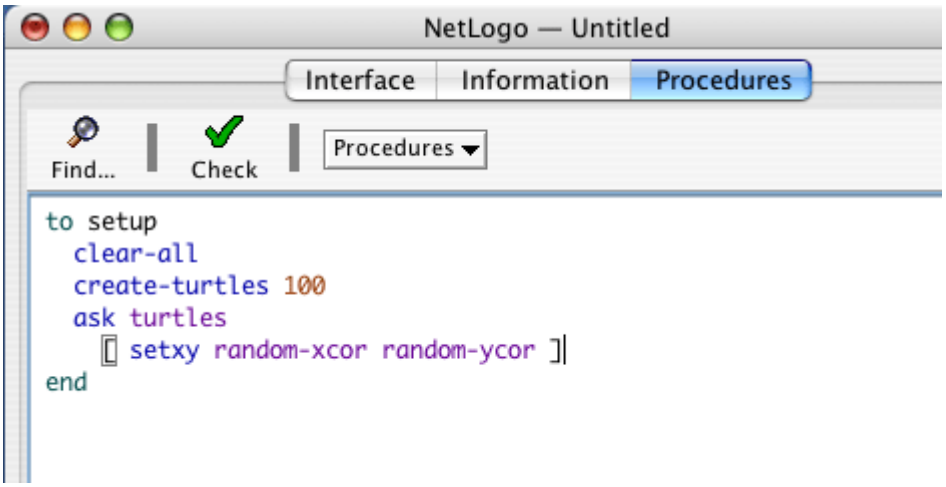
要想看实际的错误信息，单击该按钮。

现在创建"setup"例程，这样错误信息就会消失。

- 切换到 Procedures 页
- 输入下面的代码：

```
to setup
  clear-all
  create-turtles 100
  ask turtles [ setxy random-xcor random-ycor ]
end
```

做完后，Procedures 页就像下面的样子：



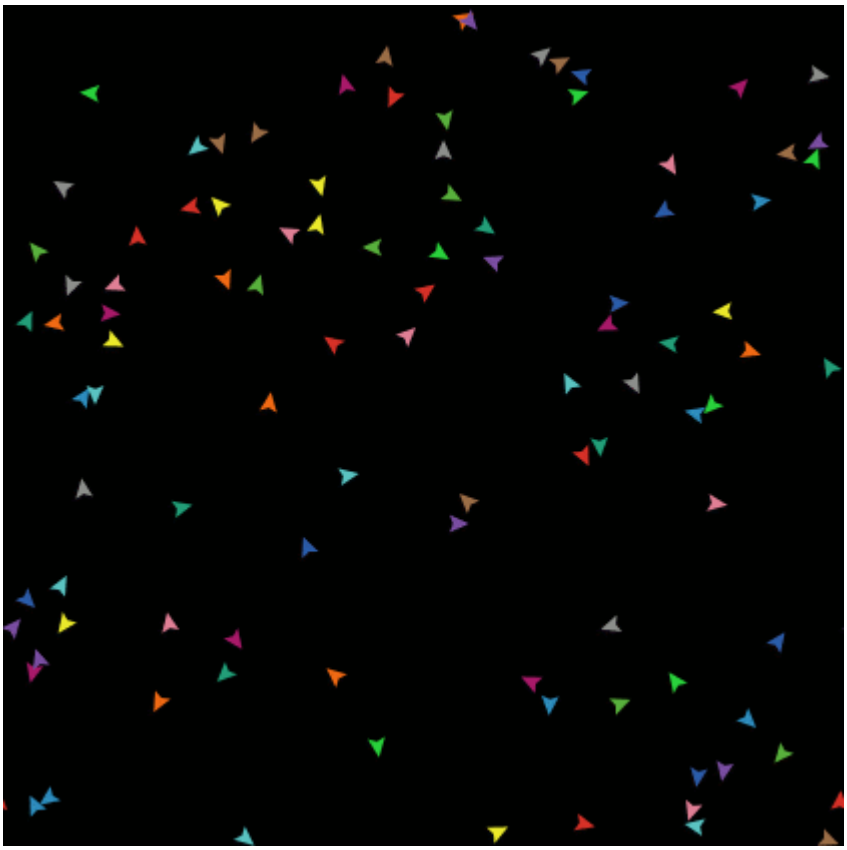
注意每行缩进量不同。多数人觉得代码这样缩进很有用，但这不是强制的，只是使得代码易读易改而已。例程以to开始，以end结束。所有的例程都要用这两个词开始和结束。

看看你输入了什么，每行是干什么的？

- to setup 开始定义一个名为"setup"的例程
- clear-all 将世界重设为初始、全空状态。所有瓦片变黑，你已经创建的海龟消失。基本上是将过去一笔勾销，为新模型运行做好准备。
- create-turtles 100 创建 100 个海龟。这些海龟都在原点，即瓦片 0,0 的中心。
- ask turtles [...] 告诉每个海龟独立地去运行方括号中的命令 (NetLogo中每条命令都是由某些主体执行的。 ask 也是一条命令。在这里是observer运行这条 ask 命令，这条命令又引起海龟运行命令)
- setxy random-xcor random-ycor 是一条使用"reporters"的命令。reporter与命令不同，它只报告一个结果。首先每个turtle 运行reporter random-xcor，它返回 X 坐标范围内的一个随机数，然后每个turtle运行 reporter random-ycor，返回Y坐标范围的一个随机数。最后每个turtle 使用前面的两个数做输入参数运行setxy 命令，这使得turtle移动到相应的坐标处。
- end 结束"setup" 例程的定义。

输入完成后，切换到界面页，按下前面制作的 setup 按钮 ，你将看到海龟分散在世界

内：



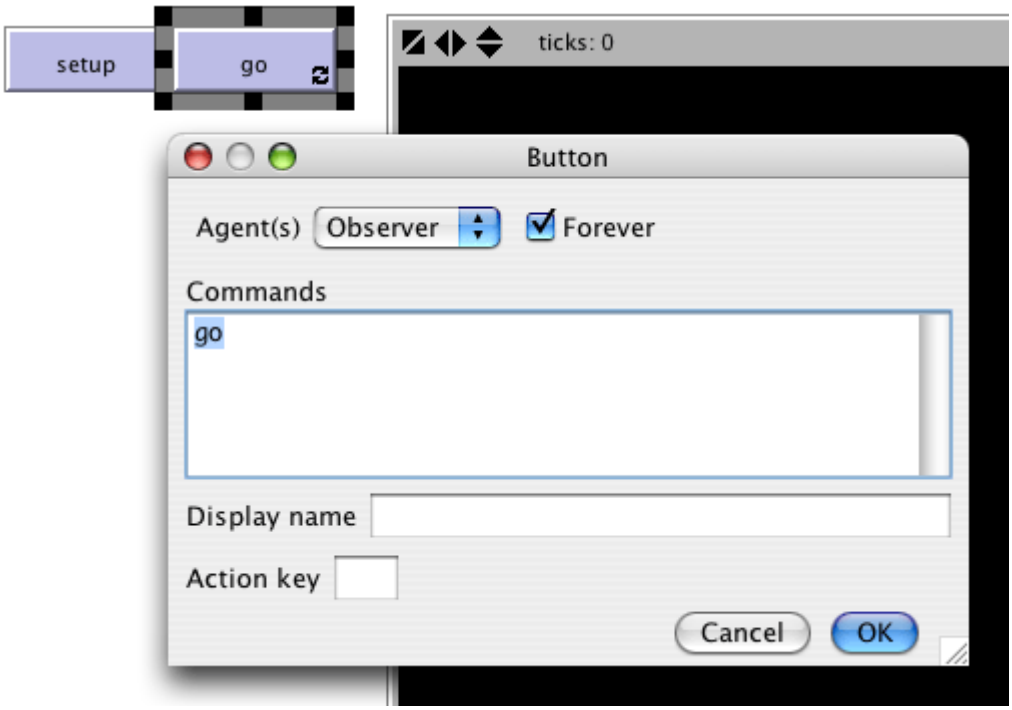
多按 setup 几次，看看海龟的散布有何不同。 注意有些海龟会叠压在一起。

稍微想想要达到这样的结果需要做哪些事情。你要在界面页里做一个按钮，还要创建该按钮要使用的一个例程。只有这两步都做了，按钮才能工作。在本教程的剩余部分，经常需要做类似两步或更多步，为模型增添新功能。在增加新功能时，当你觉得该做的步骤都完成了，但就是不能正常工作时，那就继续向前读，看看是否还要其他步骤。读过几节后，返回来看看是否遗漏了某个步骤。

制作 go 按钮

现在做个 "go"按钮。步骤与 setup 按钮的一样，除了下面几点：

- 在命令部分输入 go，而非 setup.
- 在编辑对话框里勾选 "forever"



"forever" 勾选项使得按钮按下后保持按下状态，因此命令会不断重复执行，而不是只执行一次。

- 在 Procedures 也增加 go 例程:

```
to go
  move-turtles
end
```

move-turtles是什么？它是一个象clear-all那样的原语（NetLogo内嵌的）吗？ 非也，它是一个需要你添加的例程。至今你已经有两个自己添加的例程了：setup 和go

- 在 go 例程后面增加 move-turtles 例程:

```
to go
  move-turtles
end

to move-turtles
  ask turtles [
    right random 360
    forward 1
  ]
end
```

注意 move-turtles 中间的连字符两侧没空格。教学#2 中用过的 red - 2 有空格，是为了做减法操作。此处我们要的是 move-turtles，没空格。"-" 将 "move" 和"turtles" 组成一个词。

例程 move-turtles 中的命令：

- ask turtles [...] 每个 turtle 运行[ ]中的命令
- right random 360 是使用reporter的命令。首先每个turtle 在 0 和 359 之间随机选一个整数 (random 不会返回你给它的数) ，然后turtle 右转这个度数。
- forward 1 让 turtle 前进 1 步。

为什么我们不把所有这些命令都写在 go 例程里，而是分为几个例程？确实可以这样做，但是在创建你的项目的过程中，你很可能会增加更多的东西。最好保持 go 例程尽量简单，这样更容易理解。最后还会包括许多其他的东西，比如计算，画图等。这些事情由相应的例程完成，每个例程有各自的名字。

界面页中的 go 按钮是永久性的，意味着将不断执行命令，直到你关掉它（重新单击它）。按下 setup 按钮，创建海龟，然后按下 go 按钮。观察模型。关闭它，你会看到所有海龟停住了。

注意当海龟越过世界边缘时，它要回绕（wrap），即出现在另外一边。（这是默认行为，可以改变，详情参加编程指南的Topology部分。）

试试命令

我们建议你试试其他海龟命令。

在命令中心输入命令 (如 turtles> set color red)，或在 setup, go, move-turtles 中添加命令。

注意在命令中心输入命令时，你必须使用弹出菜单选择 turtles>, patches>, or observer>，具体选择取决于哪个主体将执行命令。这就像 ask turtles 或 ask patches 一样，只是不用打字而已。你可以使用 tab 键在主体类型之间切换，这比用菜单更方便。

可以试试在命令中心输入 turtles> pen-down，然后按下 go 按钮。

在 move-turtles 例程中，试试将 right random 360 改为 right random 45。

玩吧。很容易，并且结果立现 —— 这是 NetLogo 许多优点之一。

试验够了吧，准备继续改进模型。

瓦片和变量

现在我们有 100 个海龟，它们漫无目的的移动，对周围的事物毫无知觉。下面我们给海龟一个好点的背景，让模型稍微有趣一些。

- 回到 setup 例程，修改例程如下：

```
to setup
```

```
  clear-all
  setup-patches
  setup-turtles
end
```

- 新的 setup 引用了两个新例程，定义 setup-patches：

```
to setup-patches
  ask patches [ set pcolor green ]
end
```

例程setup-patches将开始时所有瓦片颜色定义为绿色。（海龟的颜色变量是color，瓦片的是pcolor）

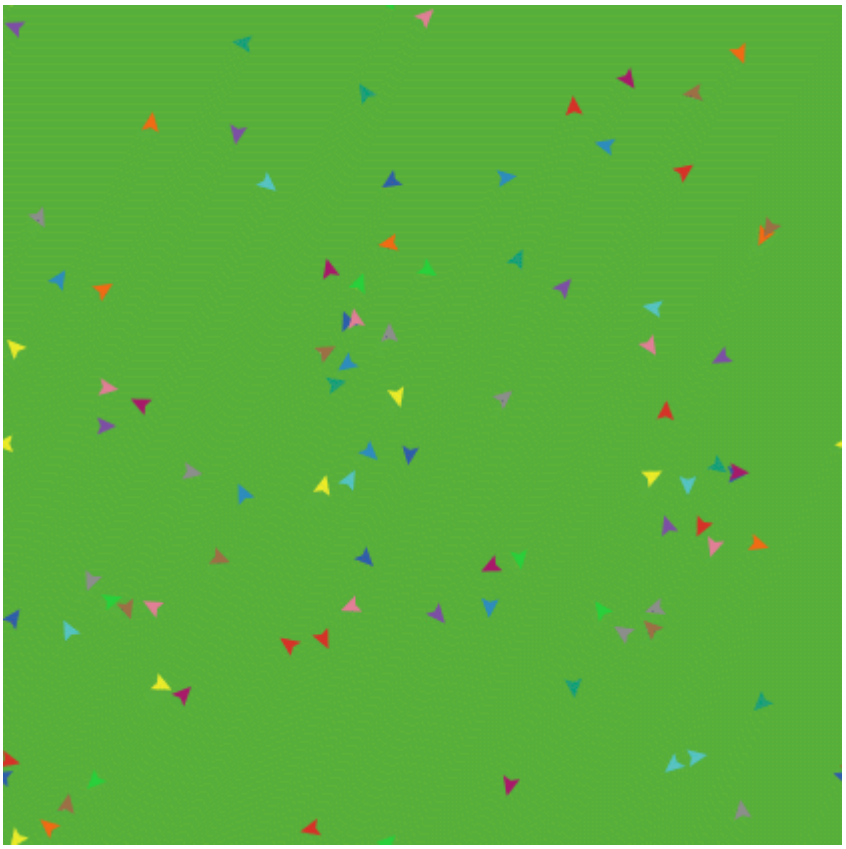
- 加上这个例程

```
to setup-turtles
  create-turtles 100
  ask turtles [ setxy random-xcor random-ycor ]
end
```

你是否注意到新的 setup-turtles 与老的 setup 包含许多相同的命令？

- 切换回 Interface 页
- 按下 setup 按钮

瞧！ 由海龟和绿色瓦片构成的 NetLogo 风景 ：



看过新 setup 例程的效果后，你会发现重新读读这个例程很有帮助。

海龟变量

目前海龟在可以地表上移动，但什么都不做。现在在海龟和瓦片之间加上一些交互。

我们让海龟吃“草”（绿色瓦片），繁殖、死亡。草被吃掉后要逐渐恢复。

我们需要一种控制海龟繁殖和死亡的方式。我们通过跟踪海龟有多大能量（energy）来决定。要这样的话需要增加一个新的海龟变量。

你已经见过一些内置的海龟变量，如color。要添加新的海龟变量，需要在例程页的顶部加上turtles-own 声明，这个声明必须在所有例程之前，变量名为energy：

```
turtles-own [energy]

to go
  move-turtles
  eat-grass
end
```

使用这个新定义的变量(energy)，允许海龟吃草。

- 切换到 Procedures 页
- 重写 go 例程如下:

```
to go
  move-turtles
  eat-grass
end
```

- 加上新例程 eat-grass :

```
to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
      set energy (energy + 10)
    ]
  ]
end
```

我们第一次使用if 命令，仔细看看代码。当每个海龟执行这些命令时，比较它所处的瓦片的颜色(pcolor)与绿色是否相同。（海龟能直接访问所处瓦片的变量），如果瓦片是绿色则返回true，这时才执行[ ]中的命令（否则跳过）。这些命令让海龟将瓦片改为黑色，海龟能量值增加 10。瓦片变黑表明该处的草被吃掉，因为吃了草，海龟能量增加。

下面，让海龟移动时消耗一些能量：

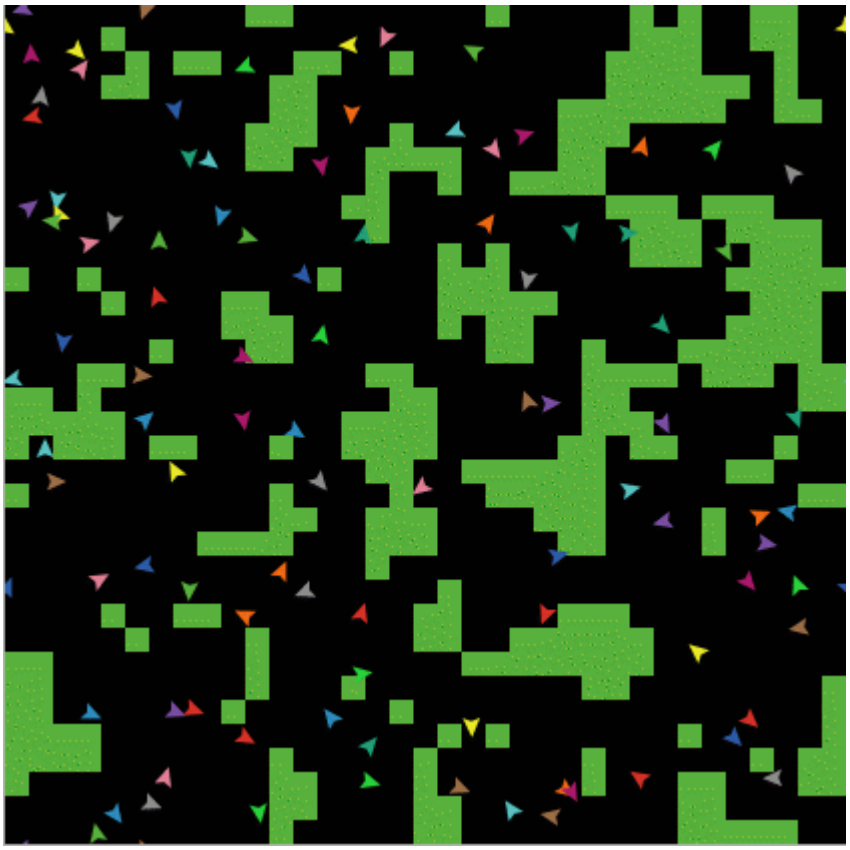
- 重写 move-turtles 如下:

```
to move-turtles
  ask turtles [
    right random 360
    forward 1
    set energy energy - 1
  ]
end
```

当海龟移动时，每步减少 1 个单位的能量。

- 切换到 Interface 页 ， 按下 setup 和 go 按钮

你将看到当海龟走到瓦片上时，瓦片变为黑色。



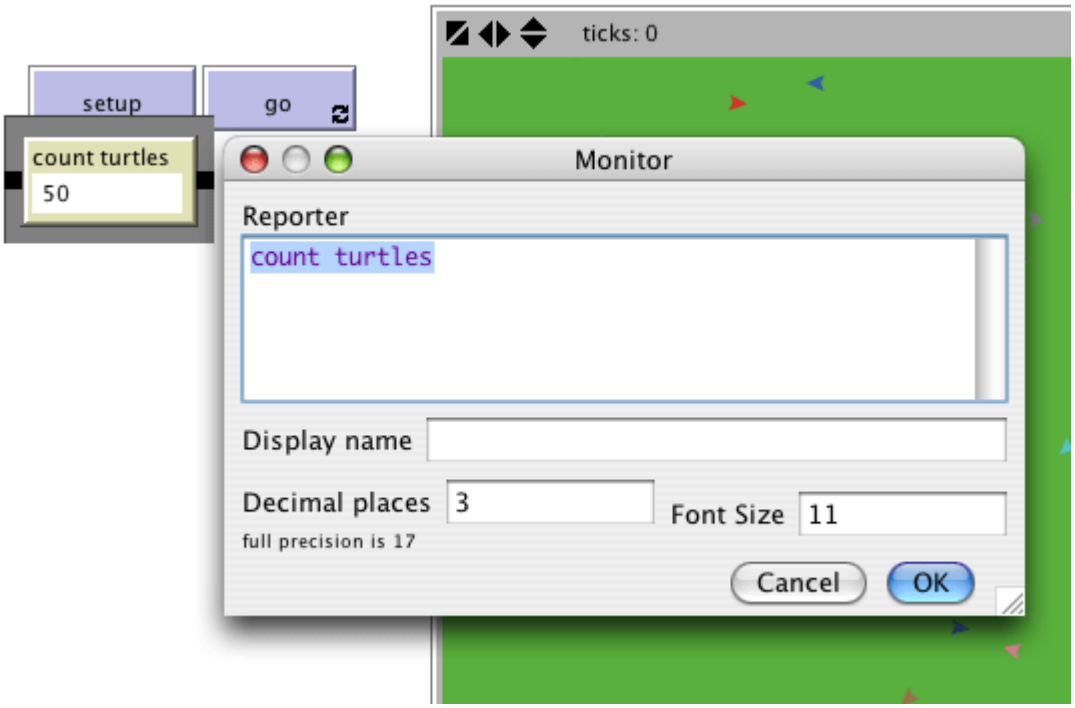
监视器（Monitors）

下面使用工具条在 Interface 页中创建 2 个监视器。（像使用按钮、滑动条一样，使用工具条上的监视器图标）。先做第一个监视器。

- 使用工具条上的监视器图标，在界面页空白处创建一个监视器。

出现对话框

- 在对话框中输入: count turtles （见下图).
- 按 OK 关闭地对话框



turtles 是一个"agentset",即所有海龟的集合。count告诉我们这个集合中有多少主体。

制作第二个监视器：

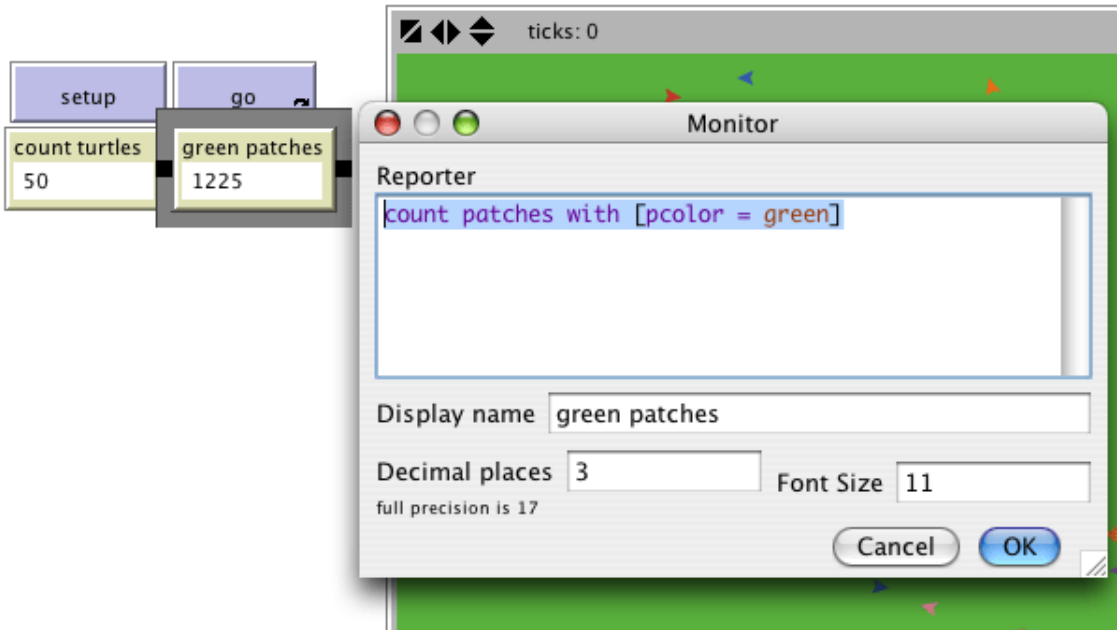
- 使用工具条上的监视器图标，在界面页空白处创建一个监视器。

出现对话框。

在对话框的 Reporter 部分输入：`count patches with [pcolor = green]` (见下图).

在 Display name 部分输入： `green patches`

按 OK 关闭对话框



此处我们再次使用count查看一个agentset中有多少主体。patches是所有瓦片的集合，但我们并不想知道总共有多少瓦片，而是想知道有多少绿色的。这就是with要做的，它创建一个较小的agentset，只有满足[ ]中的条件的主体才会包含进来，条件是pcolor = green，因此得到的是绿色瓦片。

现在有两个监视器报告有多少海龟，有多少绿色瓦片，帮助我们跟踪模型运行。模型运行时，监视器中的数字自动变化。

使用 setup 和 go 按钮，观看监视器数值的变化。

开关和标签（Switches and labels）

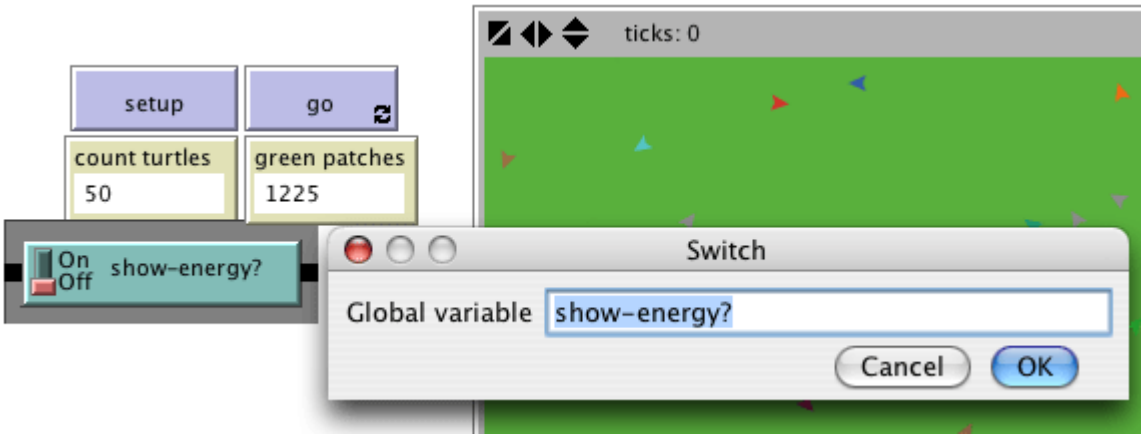
海龟不仅是将瓦片变黑，它们还获得、损失能量。模型运行时试试使用海龟监视器查看一个海龟的能量变化。

如果能在任何时候看到所有海龟的能量就更好了。现在就这样做，并且增加一个开关能控制这些额外信息显示与否。

- 在界面页 的工具条上选择开关图标，在空白处单击，创建一个开关。

出现对话框。

- 在对话框的 Global variable 部分输入 show-energy? 别忘了包括问号 (见下图)



- 返回 'go' 例程
- 重写 eat-grass 例程如下：

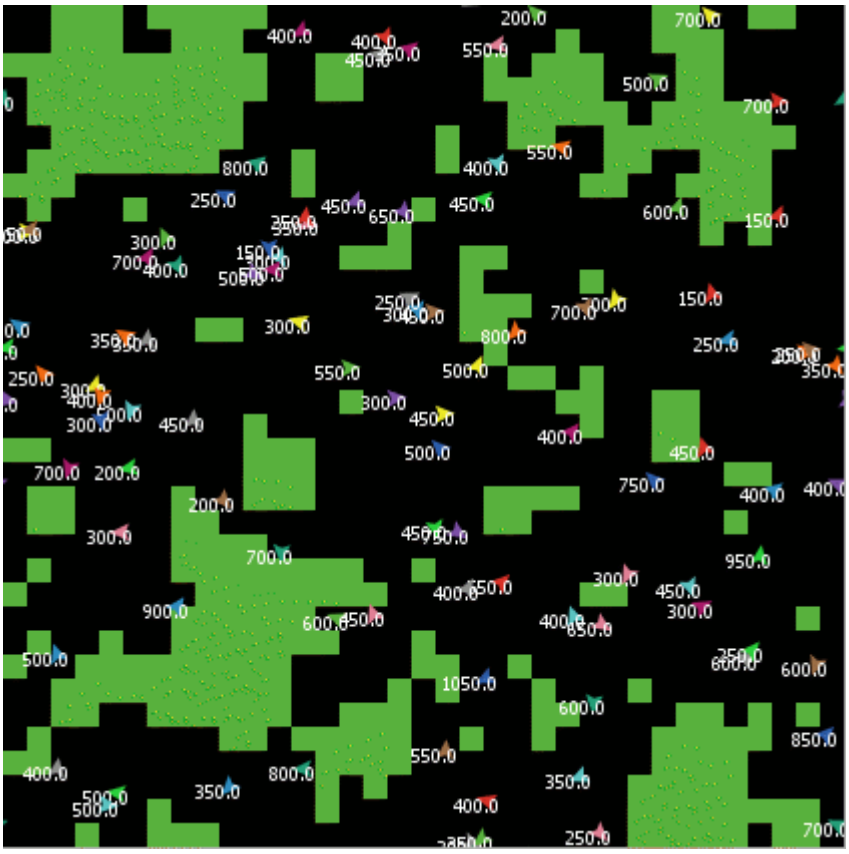
```
to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
      set energy (energy + 10)
    ]
  ifelse show-energy?
    [ set label energy ]
    [ set label "" ]
  ]
end
```

例程eat-grass用了ifelse命令，仔细看代码。每个海龟当运行这些新命令时检查show-energy?的值（由开关决定）。如果开关打开，比较结果为true，海龟执行第一个[ ]中的命令。这时将能量值赋给海龟标签。如果比较结果为false（开关关闭），海龟执行第二个[ ]中的命令，这时移去文本标签（通过将海龟标签设为空）。

（在 NetLogo 中几个字符称为字符串（string）。字符串是在双引号之间的一串字母和字符。此处两个双引号之间什么也没有，这是一个空串。如果海龟的标签是空串，就表示没有任何文本。）

- 测试一下。在界面页中运行模型（使用 setup 和 go），来回拨动 show-energy? 开关。

当开关打开时，能看到海龟的能量因吃草而增加，移动时减小



更多例程

现在海龟在吃草，再让它们繁殖和死亡，也让草能恢复。现在增加三个例程，分别负责这三种行为。

- 切到 Procedures 页
- 重现 go 例程如下：

```
to go
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
end
```

- 增加例程 reproduce, check-death 和 regrow-grass 如下：

```
to reproduce
  ask turtles [
    if energy > 50 [
```

```
      set energy energy - 50
      hatch 1 [ set energy 50 ]
    ]
  ]
end

to check-death
  ask turtles [
    if energy <= 0 [ die ]
  ]
end

to regrow-grass
  ask patches [
    if random 100 < 3 [ set pcolor green ]
  ]
end
```

这些例程都使用了if命令。当繁殖时，每个海龟检查它的energy值，如果大于 50 执行第一个[ ]中的命令。在这里energy减少 50，然后孵出（hatches）一个energy为 50 的新海龟。hatch命令是NetLogo的一个原语，形如hatch *number* [*commands*]，海龟创建*number*个新海龟，每个都与母体相同，并且请求这些新海龟执行*commands*，你可以使用*commands*让这些新海龟有不同的颜色、方向等。这里运行一条命令，将新海龟的energy设为 50。

当每个海龟运行check-death时，它检查energy是否小于等于 0。如果是真，则海龟被告知去死die（这是NetLogo的一个原语）。

当每个海龟运行 regrow-grass 时,它检查随机产生的 0-99 之间的整数是否小于 3。如果是，瓦片颜色设为绿。对每个瓦片来说，发生的次数（平均）是 3%，因为在 100 个可能的数中，有三个数（0，1，2）小于 3。

- 切换到 Interface 页，按下 setup 和 go

现在能看到模型的一些有趣行为。一些海龟死掉，一些新海龟出现（孵出），一些草恢复。这正是我们要做的。

如果继续看模型的监视器，会发现 **count turtles** 和 **green patches** 监视器都有振荡。振荡模式能预测吗？这些变量之间有关系吗？

如果有更容易的跟踪模型行为的方式就更好了。NetLogo 可以为我们画图，这是下面要讲的。

画图（Plotting）

要想画图的话，需要在界面页创建一个 plot，做一些设置。然后在例程页增加一些例程，这些例程为我们更新绘图。

先做例程页中的工作。

- 修改 setup，调用将要增加的新例程 do-plots

```
to setup
  clear-all
  setup-patches
  setup-turtles
  do-plots
end
```

- 还要修改 go ，调用 do-plots

```
to go
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  do-plots
end
```

- 增加新例程。我们要画的是海龟的数量和绿瓦片的数量。在每个时间步 (go 例程的一次运行)这些值加到图中。

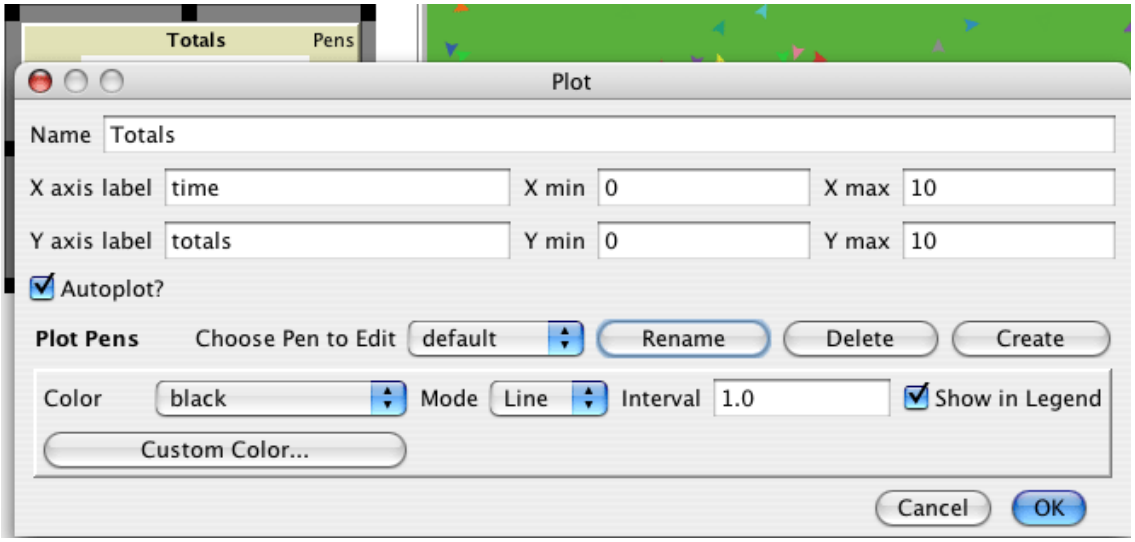
```
to do-plots
  set-current-plot "Totals"
  set-current-plot-pen "turtles"
  plot count turtles
  set-current-plot-pen "grass"
  plot count patches with [pcolor = green]
end
```

注意使用plot命令在图上增加新点。然而在这样做之前，需要告诉NetLogo两件事。第一，需要指明要使用哪个图（因为后面的模型有多个图）。第二，要指定使用哪支笔（pen）画图（本图使用两支笔）。

plot命令告诉画笔移动到新点，新点的X 坐标是先前的X坐标加 1，Y坐标就是plot命令中给定的值（第一种情况是海龟数量，第二种情况是绿瓦片数量）。当画笔移动时就画出线。

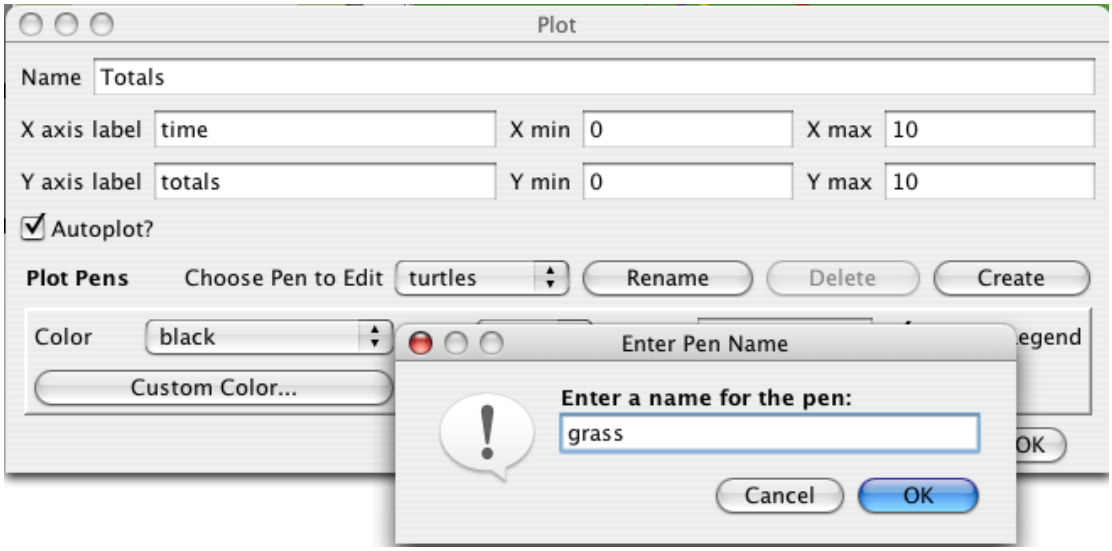
为了让 set-current-plot "Totals"工作，必须在界面页中为你的模型增加一个图（plot），然后进行编辑，让它的名字与例程中用到的相同。即使名字中多个空格也会出错---两处必须完全相同。

- 在界面页上使用 plot 图标创建一个 plot
- 设置图名为"Totals" (见下图)
- 设置 X 轴标签为"time"
- 设置 Y 轴标签为"total"



下一步创建两支笔（pen）。

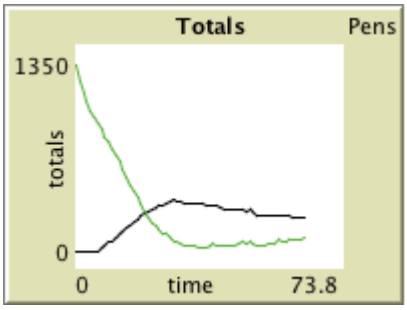
- 在 Plot 对话框中，按下'Create' 按钮创建一支新笔
- 输入笔的名字"turtles"，在"Enter Pen Name"对话框中按下 OK (见下图)
- 在 Plot 对话框中，按下'Create' 按钮创建第二支新笔
- 输入笔的名字"grass" ，在"Enter Pen Name"对话框中按下 OK (见下图)
- 选择笔的颜色，变为 green.
- 在 Plot 对话框中选择 OK



注意在创建图的时候，也可设置 X,Y 轴的最大最小值。让"Autoplot?"勾选着，如果图超过坐标轴的设定范围，坐标轴会自动增长，使你能看到所有数据。

- 再次按 Setup 和 go，运行模型

你会看到模型运行时就能绘图。图的样子与下图相似，尽管可能不完全一样。
记住我们让"Autoplot?"打开。这使得没有空间时，图能自动调整。



如果你忘了哪支笔是干什么的，单击图的右上角处 Pens 标签。你试试运行模型几次，看看这些图哪些部分相同哪些不同。

时钟计数器（Tick counter）

在比较同一模型多次运行得到的图时，如果每次运行长度相同会很有用。学会怎样让模型在特定的时刻停止或启动很有帮助，因为我们可以让模型在同一时刻停止。跟踪 go 例程运行了多少次是实现这一技术的关键。

要跟踪这一点，使用 NetLogo 内建的时钟计数器（tick counter）

- 修改 go 例程

```
to go
  if ticks >= 500 [ stop ]
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  tick
  do-plots
end
```

- 按下 setup ，再运行模型

图形和模型不会一直运行下去，当界面页工具条上的时钟计数器达到 500 时，模型自动停止。

tick命令将时钟计数器加 1， ticks是一个报告器，返回时钟计数器当前值。每次重新运行时clear-all将时钟计数器清 0.

注意tick在do-plots之前。如果绘图代码使用时钟计数器就要这样，它会查看新值，而非旧值。（本教程实际没写这样的代码，但是一般说来将tick放在主体动作之后、绘图之前是个好做法）

现在模型使用了 ticks，你可能想使用界面页顶部的菜单从连续更新（continuous updates）变为基于时钟更新（"tick-based" updates）。这意味着 NetLogo 仅在时钟点上更新（重画）视图（主体显示区），tick 中间不更新。这样模型运行的更快一些，并保证稳定观感（因为更新的时间间隔固定）。关于视图更新的全部讨论见编程指南。

更多细节

首先，可以有可变数量的海龟，而非总是 100 个。

- 增加一个滑动条 'number', 改变最大最小值
- 在例程 setup-turtles 中，不使用 create-turtles 100，而是：

```
to setup-turtles
  create-turtles number
  ask turtles [ setxy random-xcor random-ycor ]
end
```

测试一下。比较初始时刻海龟更多或更少时图的变化.

其次，如果能调整海龟吃草获得的能量和移动时消耗的能量不是更好吗？

- 增加滑动条 energy-from-grass.
- 增加滑动条 birth-energy
- 修改 eat-grass 例程：

```
to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
      set energy (energy + energy-from-grass)
    ]
    ifelse show-energy?
      [ set label energy ]
      [ set label "" ]
  ]
end
```

- 修改 reproduce：

```
to reproduce
  ask turtles [
    if energy > birth-energy [
      set energy energy - birth-energy
      hatch 1 [ set energy birth-energy ]
    ]
  ]
end
```

最后，如果要改变草的恢复率应该增加什么滑动条？能增加什么海龟移动规则，或让孵化只发生在特定时刻，试着写出来。

下一步?

现在有了一个简单生态系统模型。瓦片长草，海龟移动、吃草、繁殖、死亡。你创建了界面，包括按钮、滑动条、开关、监视器和绘图。甚至写了一些例程让海龟做事。

教学到此结束。

如果要看更多的NetLogo文档，界面指南Interface Guide让你了解NetLogo所有界面元素的功能。要写例程，看编程指南Programming Guide。所有原语在NetLogo词典NetLogo Dictionary。

如果愿意，你还可以继续试验、扩展本模型，试验主体不同的变量和行为。

另外，还可以再看教学#1 的狼吃羊模型。你看到羊走来走去，消耗资源，资源随机补充，一定条件下繁殖，没有资源时死亡。该模型还有另一类生物—狼。增加狼需要另外的例

程及新的原语。狼和羊是两种不同的物种（breeds），研究该模型，了解怎样使用 breeds。

另外，还可以查看其他模型（包括模型库中的 Code Examples），甚至继续建立自己的模型。你甚至不必建模，只是看着瓦片和海龟形成模式就很有趣，或试着创建个游戏玩玩，等等。

希望你学到了一些东西，包括 NetLogo 语言及如何建模。上面创建的所有例程列在下面。

附录：完整代码

这些完整代码在 NetLogo 模型库里也有，在 Code Examples 部分，名为 "Tutorial 3"。

注意代码有注释，注释由分号开始。使用注释帮助你理解模型。

在例程页，注释是灰的，容易区别。

```
turtles-own [energy] ;; for keeping track of when the turtle is ready
                     ;; to reproduce and when it will die

to setup
  clear-all
  setup-patches
  setup-turtles
  do-plots
end

to setup-patches
  ask patches [ set pcolor green ]
end

to setup-turtles
  create-turtles number    ;; uses the value of the number slider to create turtles
  ask turtles [ setxy random-xcor random-ycor ]
end

to go
  if ticks >= 500 [ stop ]  ;; stop after 500 ticks
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  tick                    ;; increase the tick counter by 1 each time through
  do-plots
end
```

```
to move-turtles
  ask turtles [
    right random 360
    forward 1
    set energy energy - 1  ;; when the turtle moves it looses one unit of energy
  ]
end

to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
           ;; the value of energy-from-grass slider is added to energy
      set energy (energy + energy-from-grass)
    ]
  ifelse show-energy?
    [ set label energy ] ;; the label is set to be the value of the energy
    [ set label "" ]     ;; the label is set to an empty text value
  ]
end

to reproduce
  ask turtles [
    if energy > birth-energy [
    set energy energy - birth-energy  ;; take away birth-energy to give birth
    hatch 1 [ set energy birth-energy ] ;; give this birth-energy to the offspring
    ]
  ]
end

to check-death
  ask turtles [
    if energy <= 0 [ die ] ;; removes the turtle if it has no energy left
  ]
end

to regrow-grass
  ask patches [ ;; 3 out of 100 times, the patch color is set to green
    if random 100 < 3 [ set pcolor green ]
```

```
  ]
end

to do-plots
  set-current-plot "Totals" ;; which plot we want to use next
  set-current-plot-pen "turtles" ;; which pen we want to use next
  plot count turtles ;; what will be plotted by the current pen
  set-current-plot-pen "grass" ;; which pen we want to use next
  plot count patches with [pcolor = green] ;; what will be plotted by the current pen
end
```