

目錄

Introduction	1.1
Part 1: Scratch入门	1.2
Scratch ABC	1.2.1
Scratch界面和基础使用	1.2.1.1
Scratch积木和应用原则	1.2.1.2
Scratch疑难问题举例	1.2.1.3
Part 2: 程序逻辑及Scratch高级技能	1.3
Scratch的形象世界	1.3.1
简单运动控制	1.3.1.1
画笔和运动结合的威力	1.3.1.2
外观控制和简单序列帧动画	1.3.1.3
Scratch的抽象世界	1.3.2
变量可以实现哪些事	1.3.2.1
布尔表达式和逻辑运算 另一种加减乘除	1.3.2.2
列表 有序世界的基础	1.3.2.3
循环 批量化的天使	1.3.2.4
计算机科学的发展史就是一部抽象史 函数	1.3.2.5
抽象世界深处	1.3.3
问题还没开始处理就已经被解决 递归	1.3.3.1
排序和查找 有序和贪婪	1.3.3.2
避免耦合 队列	1.3.3.3
堆栈和树的关系	1.3.3.4
降维打击 二维数组的实现	1.3.3.5
上知天文下知地理的预测 回归	1.3.3.6
风险 加锁和竞争	1.3.3.7
从开关到王者荣耀 计算机组成	1.3.3.8
Part 3: 数学和物理概念可视化应用	1.4
数学黑魔法	1.4.1
数论 高斯的光芒	1.4.1.1
指数对数和苹果树	1.4.1.2

太阳和尼罗河 进制	1.4.1.3
赌神必备 概率和随机	1.4.1.4
黑客的帝国 矩阵	1.4.1.5
莱布尼茨我不服 微积分	1.4.1.6
让暴躁的世界平缓 简单滤波器	1.4.1.7
摊开的地球仪 卡诺图和逻辑化简	1.4.1.8
田忌的小算盘 规划问题	1.4.1.9
以牙还牙的汉莫拉比法典 博弈论	1.4.1.10
物理白魔法	1.4.2
经典的力和加速度	1.4.2.1
撬地球的棍子 简单机构	1.4.2.2
最接近魔法的学问 电磁	1.4.2.3
Part 4: 游戏设计专题	1.5
你真正了解游戏么 上帝视角的游戏设计	1.5.1
游戏类型和发展史	1.5.1.1
游戏引擎是什么	1.5.1.2
游戏核心 是什么让游戏有趣	1.5.1.3
游戏为谁设计 四象限人格论	1.5.1.4
地牢的迷思 游走在现实和虚幻的边界	1.5.1.5
超高难度游戏为什么有死忠粉 心流体验	1.5.1.6
大巧造物 游戏设计专题	1.5.2
行走在确定和随机之间 随机专题	1.5.2.1
撒豆成兵千军万马 克隆专题	1.5.2.2
掌控了时间的魔法师们 状态回溯和时间操控专题	1.5.2.3
一瞬间天翻地覆 逐帧渲染和特效合成专题	1.5.2.4
提前规划 妥善抽象 合作开发和设计模式专题	1.5.2.5

LEARN SCRATCH



MAGIC WAY

Scratch魔法学

本书将包含四个部分：

1. Scratch入门
2. 程序逻辑及Scratch高级技能
3. 数学和物理概念可视化应用
4. 游戏设计专题

Learn Scratch Magic Way

This book will contain four parts.

1. Scratch ABC
 2. Advance skills of programming
 3. Math and physics applications
 4. Game design topics
-

目标读者

1. 无编程基础的Scratch K-6 初中以下学生
2. 有编程基础想学习Scratch高级用法的7-12年级初高中学生

3. 想使用Scratch制作交互式教具的数学物理老师
4. 全年龄段想学习游戏设计的开发者

为什么写这本书

Scratch的编辑器十分类似于Unity等主流游戏编辑器，sprites & scripts的概念也是基本的游戏编辑器概念。十分适合制作各类交互式的动画、游戏和课堂教具。本书的作者也曾是一名独立游戏策划和开发者，后从事了面向小学生的Scratch的课程设计。在设计课程的过程之中有许多原创的点子和游戏原型迸发出来，系统的编排之后均收录于这本书之中。市面上现有的Scratch教材多是介绍Scratch的基本概念和应用，对于Scratch这个工具可以发挥的极限远没有触及。所以本书在第四部分中，讲述了许多Scratch可以承载的复杂游戏设计模式。

本书的难度如何

1. 本书并不是一本Scratch的从零开始到精通的教材，面向儿童读者也应该是参加过或者正在参加Scratch培训的。
2. 笔者尽量用容易理解的语言描述有一定深度的思想，小学和初中学生可以把此书当做可以自己动手操作的科普教材阅读。
3. 此外书中对于Scratch各种细节和设计原则的讨论，非常适合参加了Scratch培训想要进一步提升自己实战能力的孩子们。
4. 书中的许多代码和实现方式更是方便孩子们直接使用，对于竞赛或者是打开设计思路应该有一些帮助。
5. 另外本书很方便没有程序基础的老师和大朋友来阅读，尽量覆盖了Scratch会遇到的各类疑难问题，并且给出了许多可用的原型。

什么是程序

在普遍的介绍里面，程序都被解释为一种和计算机对话的通用语言。但是作者在这里要继续深入一下这个话题。程序是一种控制数据的方法。这里包含两件事情，就如同阴阳一样，阴就是数据，阳就是控制或操纵数据的方法。

对于数据，我们把它们存进了各种变量的盒子里面，这样还觉得不够方便，还要发明很多存储数据的结构。这样我们就能很方便的找到并且控制数据。

对于控制数据的方法，人类掌握的手段很有限，无非是判断和循环。但是这两种手段的组合威力无穷，甚至人工智能都是依靠这两个手段完成的。

为什么Scratch要借鉴游戏编辑器

游戏对于人类来说，既是魔鬼也是天使。人类在现实世界之中学习技能时往往很难知道自己到底到了什么地步，就像武侠小说中的武痴总喜欢找人切磋，无非是想给自己一个能力反馈。然而在游戏之中，任何一个动作都会得到反馈，小到按钮物品音效，大到世界排行榜，我们也可以清楚的知道自己还有多久升级。这种反馈密度带来的落差，让我们更加喜欢游戏，甚至沉迷。但是从另一个角度来看，我们在学习任何技能的时候，何尝不能给自己多设置一些反馈的点，用游戏的原理指导自己在现实世界进步呢。物本无善恶，心有不坚定，仅此而已。

对于儿童来说，即时的反馈对于他们坚持下去的意义更为重大。随着年龄增长，儿童才逐渐可以沉得下心。所以Scratch做成一个类游戏编辑器，正是为了在学习枯燥程序逻辑的同时，给儿童提供足够反馈。

此外，游戏设计的世界，对于一般人来说是完全未曾了解过的。即使是一名资深的游戏玩家，如果让他们设计游戏，作品往往只有自己觉得有趣。而且这种有趣也不能持续很久。本书的第四部分会讨论有关游戏设计的原则和通用模式，并且会以案例来带读者学习。帮助读者做出更加有趣的作品。

声明

本书不涉及任何对Scratch原有代码底层的更改，使用版本为Scratch 2.0 offline
(<https://scratch.mit.edu/download>)

本书中涉及的代码均为作者团队原创，如遗漏了引用，请及时联系作者更改

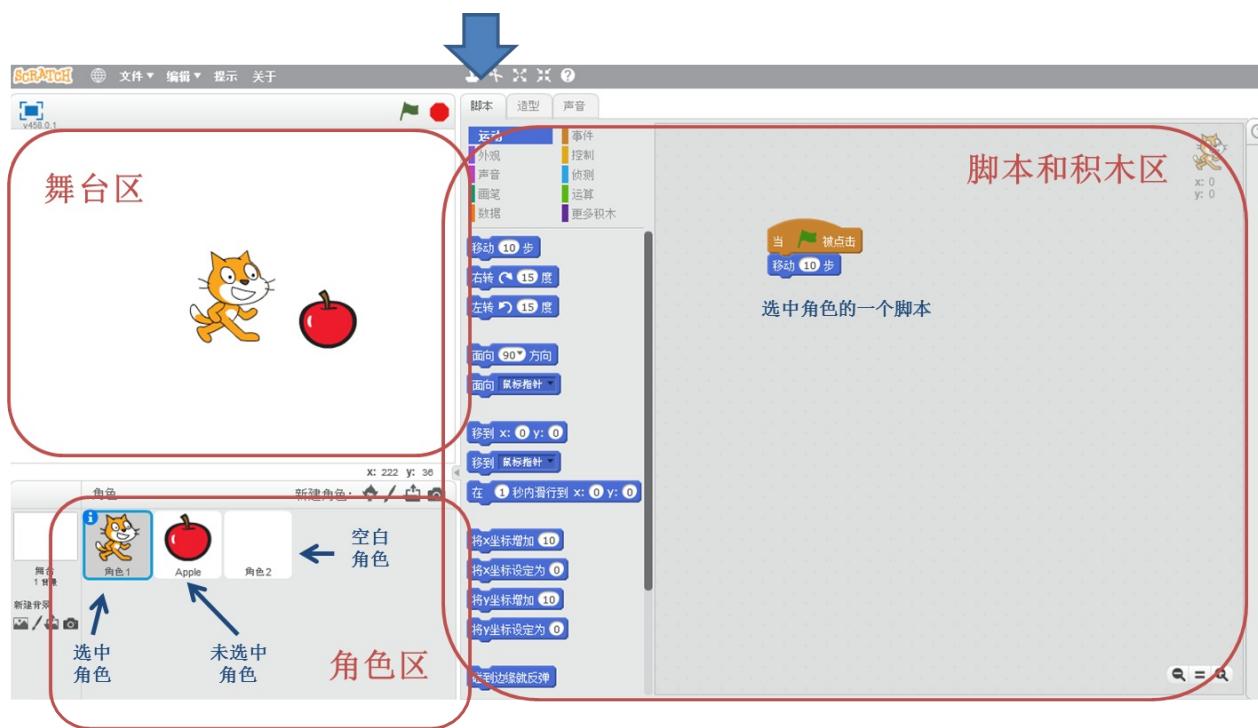
Scratch界面简介

首先要从Scratch的原理说起。Scratch的基本原理是“精灵挂脚本”。精灵在中文版的翻译中是“角色”。Scratch打开时默认的那只猫就是一个角色，一切程序都围绕着角色来进行。

1. 角色演出的地方称为舞台
2. 角色身上拿着的演出剧本称为脚本
3. 角色还自带各种演出的衣服称为造形
4. 角色甚至还拿着一个音乐播放器，可以播放已有音乐，或者录音之后播放

导演一般都站在舞台上，所以舞台上也是可以放置脚本的，用来指挥角色活动。角色身上也会自己携带自己的脚本，严格按照剧本内容执行。

这里要请初学者注意的是，随时关注你选中了哪个角色，防止把脚本放到了错误的角色身上。



舞台区角色区和积木脚本区

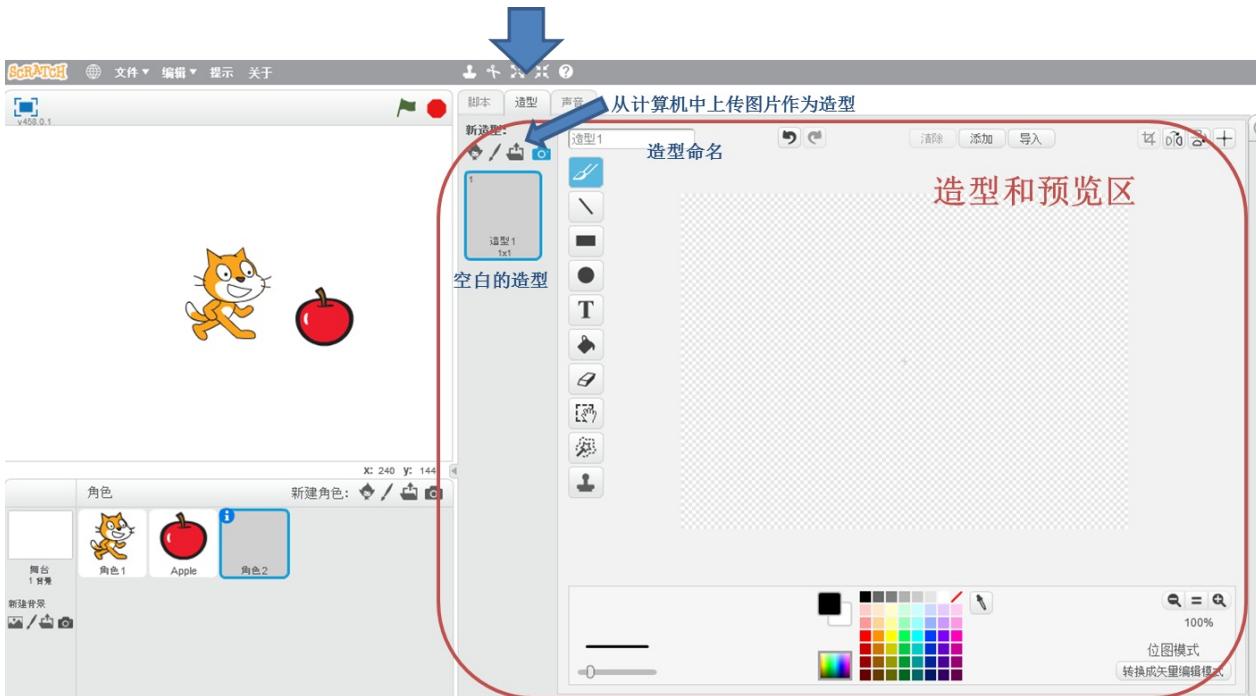
新建角色

在图1中框出的角色区右上角的四个功能按键有四大功能：

1. 新建一个角色库中自带的角色
2. 新建空白角色

3. 上传已有角色
4. 用摄像头照相制造角色

作为高端玩家，我们应该果断选择新建空白角色



造型和预览区

在接下来的空白角色的造型预览区中，我们选择从计算机中打开图片作为角色。这时候我们就可以使用自己定制的角色来做后面的开发了。

图片格式

需要注意的一点是，导入的图片最好是**PNG**格式，这样可以保存透明的背景色。如果上传了**jpg**图片，你将会看到一个人物带着一个背景方框在舞台上走来走去。

透明背景

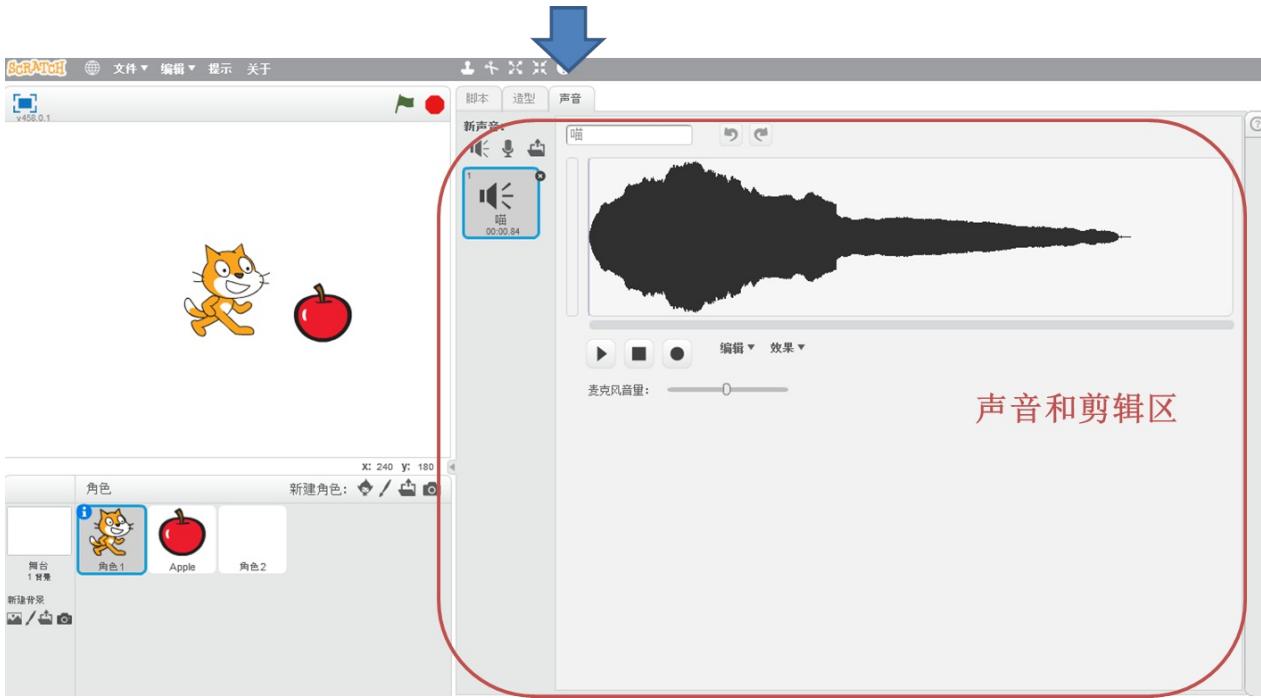
如果你是一名新手，如何区别背景色呢？只需要选中造型，在右侧的造型预览区中观察图片的周围是否是由菱形方格组成的背景即可。和**photoshop**等工具一样，菱形方格代表了透明色。

我们要如此强调透明颜色的原因是，Scratch对于角色之间的碰撞的检测是要考虑透明色的。换句话说如果两个角色都是透明的，那么他们永远不会相撞，就像一只幽灵一样。只有不透明的部分会产生碰撞。

导出

另外角色身上的造型是可以被右键选中导出为图片的。如果我们发现了别人的造型十分喜欢，只要对方是开源的作品，我们就可以用右键保存到电脑使用它的造型。

更进一步的，我们还可以在角色区中右键导出角色，导出的角色可以方便我们使用在另外的脚本之中。这个功能在本书后会详细介绍。



声音和剪辑区

剪辑和录音

在角色身上还可以携带不同的声音。和造型一样，也可以选择Scratch自带的声音库中的音乐。当然也可以从计算机中上传一段音乐，可以是mp3或者wav格式。我们也可以在右面的声音波形剪辑区，使用鼠标左键选中一段音乐，用backspace键删除它，用来截断我们不想要的音乐。当然下方还有功能更为强大的效果按钮供我们剪辑使用。

剪辑区下的圆点录音按钮可以供我们录制自己喜欢的声音来作为配音使用。

角色属性盘

角色被选中之后，左上角有一个蓝色的“i”字母。意为information，也就是角色的基本信息。左键点开它可以看到，其中旋转模式和是否显示我们会在后续章节之中提到。



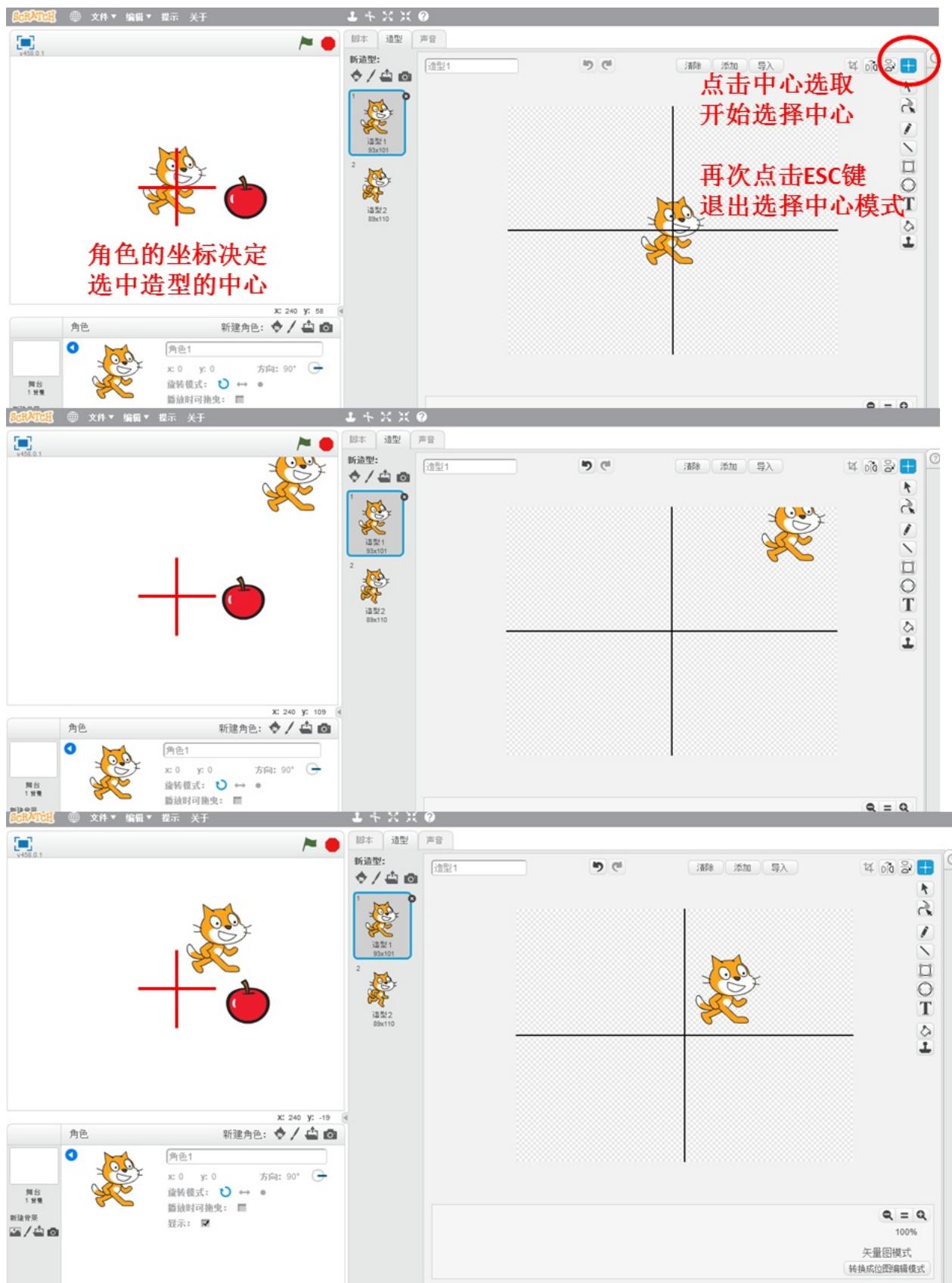
属性盘

1. 角色的坐标
2. 面向方向
3. 是否显示
4. 在播放时是否可以拖拽
5. 旋转模式

坐标

角色的坐标的确定有些隐晦。相当于用一个钉子把一张剪影画（角色造型）钉在了舞台上，这个钉子钉下的位置就是角色的坐标。

那么如何观察这个钉子钉在了什么地方呢？



造型中心和坐标对应关系

在造型区的右上角有一个十字星的图标，作用是更改角色的中心，也就是更改钉子在造型身上的位置。

1. 首先点击十字星图标，进入中心选择模式（造型区内有一个大的十字线）
2. 然后在角色造型区内点选新的中心位置
3. 点击ESC键退出中心选择模式（造型区内大十字线消失）

在上图所示的操作中，上图表示进入了中心选择模式，中图是选择了新的中心，下图是选择了另一个新的中心。不难观察到的是，角色在舞台上面的坐标(0,0)所对应的的红色十字星一直没有改变，是和角色的中心一一对应的。

方向

在角色属性盘中，可以看到右上侧方向边上有一个很小的控制器。用鼠标左键选中蓝色操纵杆拖动旋转，直接更改角色面对的方向。



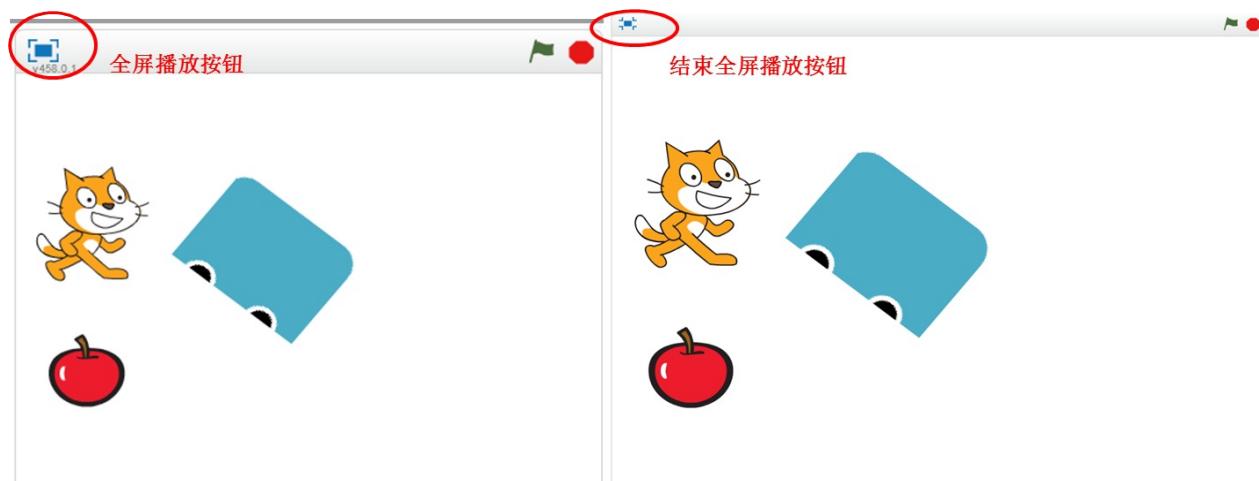
方向操纵杆

在播放时是否可以拖拽

首先要介绍Scratch的播放功能，在Scratch之中运行程序可以有两种模式

1. 编辑器模式
2. 播放器模式

打开和结束播放器模式的方式如下图



打开和关闭全屏播放器模式

编辑器模式之中，如果想使用鼠标去点击角色，Scratch会默认角色拖拽为第一优先级。换句话说，本来想点一下舞台上的角色，结果却会把角色拖动起来。这样的设置意义在于可以方便我们在舞台上面使用鼠标拖拽排布角色。

而全屏播放器模式之中，角色是默认不可以被鼠标随便拖动的。这样就避免了执行程序时候的各种错误。当然也可以把播放时可拖拽的开关打开，这样在播放时我们也可以随意拖拽角色实现一些功能。

Scratch是积木式语言（[Blockly](#)），依靠积木的可连接性来组成代码，防止简单语法错误的发生。Scratch分为十种积木，其中形象部分四种：动作、外观、声音、画笔。抽象部分六种：事件、侦测、运算、数据、控制、更多积木。本章将分别介绍所有积木的使用方式和一些易错细节，最后会展开描述所有积木的潜在使用场景。

注意：本章建议在看完全书之后再次阅读，或已经学习过Scratch一段时间的读者阅读。你将获得对Scratch更加系统和深化的理解。

动作



动作积木总览

Scratch的动作积木可以实现的功能归根结底只有两类：

1. 移动
2. 转向

其中移动类又分为了

1. 直接移动和滑动
2. 增量移动
3. 移动到某角色或者鼠标指针
4. 向面向方向移动

转向类分为了

1. 直接转向
2. 增量转向
3. 面向某角色或鼠标指针

直接移动或者转向的模式

相当于指挥角色。直接安排角色到达某个地方面向某个角度。

增量的移动转向或者向面向方向移动，相当于编写角色的内部逻辑。例如设定某小车遇到红色左转30°遇到蓝色右转30°。

外观

变量控制造型

将造型切换为 d

切换造型背景

将背景切换为 背景1

说 Hello! 2 秒

说话思考

图形特效

将 颜色 ▾ 特效增加 25

移至最上层

层叠关系设置

将 颜色 ▾ 特效设定为 0

下移 1 层

角色大小增加

清除所有图形特效

显示

显示隐藏

角色大小设定

将角色的大小增加 10

将角色的大小设定为 100

隐藏

外观积木总览

外观控制包括了

1. 角色身上造型切换
2. 外观特效
3. 角色大小设定
4. 说话思考
5. 层叠关系设定
6. 显示隐藏

造型切换

其中造型切换可以传入变量控制，其中变量可以是两种形式：

1. 数字123456...对应了造型的编号
2. 字符串，例如：“造型1”，对应了造型的名称

Scratch都可以自动的识别并找到对应的造型。如果好事者非要把编号为2的造型名称写成1，想看看究竟造型编号和造型名称哪个优先级更高。那么答案是造型名称更高，在“将造型切换为”积木中传入变量1时，名字为“1”的2号造型会显示出来。

外观特效

角色外观的特效分为了好多种类：颜色、鱼眼、旋转、像素化、马赛克、亮度、虚像

其中最有用的是颜色特效，他会在HSB的色轮上，把造型现有的所有颜色都增加固定的数据。例如造型现在是红色，增加颜色特效120会把角色变成蓝色。造型身上如果还有绿色，则会被转换为红色。

我们如果不制造更多的造型，而想制造一些新鲜感的话，可以对角色的颜色特效进行设定，用一个红色的角色做基础款造型。这样方便我们知道增加颜色特效之后，造型大致会变成什么颜色。

但是这里有个警告：对于不了解Scratch性能的初学者，请减少颜色特效的使用，这几乎是最能拖慢Scratch运行速度的语句。这相当于给正常走路的人背了一大桶水，又要走路又要负重。

第二有用的特效是亮度和虚像，亮度的正值可以增加亮度直到100会变成纯白，负数值直到-100会变成纯黑，虚像特效从0到100分别对应了实体和绝对看不到。

1. 虚像设为100时，角色完全不可见，但是仍旧可以识别碰撞
2. 如果也不想角色识别碰撞，可以直接将角色隐藏

这两种用法会在不同场景派上用场

其他的外观特效使用较少，请读者自行尝试功能，但是要注意的是在使用过外观特效之后一定要使用清楚全部特效，否则很可能会因为设置了虚像而找不到角色。

大小设定

Scratch的舞台大小是480*360，导入计算机图片作为造型时可以传入不同大小的图片，如果大于Scratch的舞台，软件会自动帮我们把图片缩放到合适尺寸。但是这样可能会造成清晰图片的锯齿化。所以，请合理选择图片的大小。

角色身上的第一个脚本，我们要养成习惯对角色的大小进行设定。而增量大小设定积木可以配合循环模块来动态变化角色的外观。例如可以将角色做成心跳的闪动效果。



心跳效果实现和变种

说话思考

对于高端玩家来说，说话思考积木，大多数情况下应该只用在调试过程中。可以将变量或者积木块放入说话和思考框之中，让程序在运行过程之中显示中间结果，方便我们调试程序。相当于文本语言中的print，请收好你们说的hello world的冲动。

层叠关系设定

Scratch的层叠关系设定十分迷惑。

设想我们有多个角色，每一个人身上都有一段“当绿旗被点击”“置于最上层”的积木脚本。点击绿旗之后会发生什么情况呢？

大多数人会说不知道，求是的说，Scratch会把你创建顺序在先的“当绿旗被点击”之下的脚本先执行。这是由于这个“当绿旗被点击”积木在Scratch内部被首先注册出来，所以就会首先运行。这种多个脚本同时执行，说穿了只是计算机运行速度很快的假象。在Scratch内核，程序执行还是有顺序的而并非真的并发。

再来设想我们的多个角色身上“当绿旗被点击”的积木下有的跟着“置于最上层”积木，有的跟着“下移1层”，有的跟着“下移2层”....当我们点击绿旗让程序运行，会发生什么？

答案是，虽然我们知道了先拿出来的“当绿旗被点击”积木首先运行，但是由于层级关系太复杂，还是很混乱无法预测。这就是Scratch层叠关系设置欠合理的地方。

所以我们在使用层叠关系时候应该坚持一些原则：

1. 例如尽量不使用多于3层的层叠关系
2. 如果一定要使用多层次关系，那么在设定层级关系时请使用微小的等待时间差距来自己控制层级顺序
3. 对于一直要处在最上层或者最底层的角色，请使用重复执行移到最上层或下移10层的脚本控制角色

4. 更好的层级控制方法是在舞台上使用发送消息并等待积木，实现多角色之间的顺序设置
层级

层级关系设置原则

显示隐藏

在角色的属性盘之中，我们也可以设定角色是否显示。且Scratch的显示隐藏还有各类特效和画笔都是记忆状态，换句话说如果某次运行我们让角色隐藏了，那么下次再运行程序，角色还是被隐藏的。这次画的画，如果没有在下次开始时清空就会存留下去。

所以我们如果要使用隐藏积木，请一定记住让显示积木跟在“当绿旗被点击”积木后面。成对使用积木的规律我们要遵守以下三条：

1. 隐藏和显示
2. 外观特效和清除特效
3. 画笔图章和清空

事实上更加广义的来说，这是一个将程序初始化的过程。任何的程序都需要妥善的初始化，你需要做两件事情：

1. 将所有的状态都清空或者设为初始值
2. 将所有定义的变量都赋予一个初始的数值

把所有角色都置于你已知的控制范围内，这是防止程序逐渐庞大后出现奇怪问题的预防针。

声音

Scratch声音控制分为两类

1. 角色携带的音轨播放
2. Scratch合成音符播放



声音积木

播放声音

我们可以把角色携带的声音用积木块播放出来，这主要作用是播放音效，例如吃到金币的音效，被打击的音效等等。我们还可以选择

1. “播放声音”：打开声音就让脚本继续向下执行
2. “播放声音并等待”：打开声音不放完就不往下执行脚本

一般情况下要使用第一种，这样让角色播放音效的时候相当于打开了音乐就继续去做别的事情，不耽误后面的程序运行。但是要注意的是，如果要播放游戏结束音效，我们打开声音后面跟着一个“结束全部”积木，那么声音就听不到了，因为播放声音了之后马上结束了所有脚本的运行。

但是我们又不想播放声音并等待，因为那样会造成角色阵亡了还可以乱动，直到结束音乐播放完毕才可以停下来脚本。这种问题我们有两种解决方案可以参考

1. 使用“播放声音”+“消息”+“结束角色身上其他脚本”积木，消息通知舞台上所有其他人停止。
2. 使用“播放声音并等待”+“消息”+“结束全部”，通知本角色身上关于运动的脚本停止。

播放音符

Scratch的播放音符可以实现打击乐器和乐音乐器两种效果，通过数字编号0到200对应了十二平均律的各种音高。这样我们就可以用变量来控制音符的播放了，这样可以播放我们自己的乐谱。然而Scratch的音符是通过正弦波频率来合成的，低于40编号的音符听上去很奇怪，而且无法实现连续音符的过渡，所有的音符都是单独的，并不能实现什么优美得音乐效果。所以如果想播放音乐，还是使用外部导入音乐来的方便。

画笔

Scratch的画笔是角色坐标和外观的延伸。

1. 落笔抬笔指的是在角色中心，也即是角色坐标位置落笔

2. 图章指的是将角色现在的造型印在背景上

取色器

画笔颜色控制

画笔亮度控制

画笔粗细控制

画笔积木

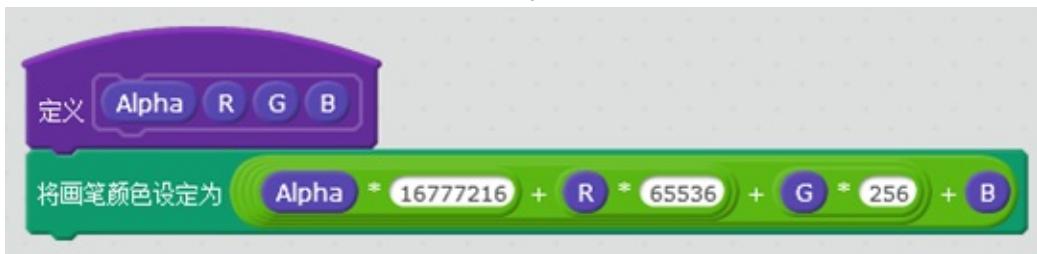
需要说明的是：

3. 画笔和图章都直接画在了舞台背景之上，如果有其他角色，将会覆盖画笔的内容。因为舞台在Scratch之中是默认最底层的，不可能有比舞台还底层的东西。
4. 画笔图章和角色隐藏与否无关，隐藏的角色依旧可以画笔和使用图章。
5. 画笔和图章之间是可以互相覆盖的，后画的覆盖先画的。

对于画笔来说，Scratch 3.0之中可以完整的设置色相、饱和度和明度，也即是HSB颜色空间。在Scratch 2.0中我们只能设置画笔颜色为0到200为红橙黄绿青蓝紫的色相，饱和度是固定的。

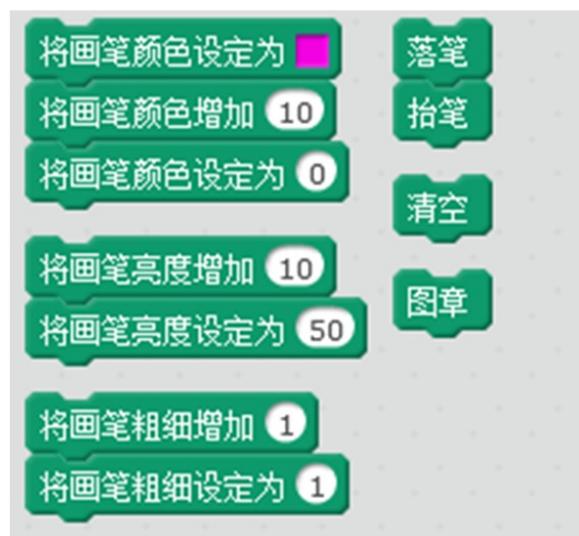
作为代替，我们可以使用取色器设定画笔颜色积木块来设置其他颜色。不过麻烦在于我们没有办法取Scratch软件外的颜色，除非我们把色卡做成一个角色造型，导入到Scratch之中，然后使用取色器取用颜色。

其实Scratch留有一个RGB的后门的，取色器设定画笔颜色积木块可以放入一个变量。我们可以用这样的方式来控制RGB和透明度Alpha。



取色器的RGB后门

应用上，画笔大多是和运动控制结合使用的。然而我们总是局限于运动物体速度过慢的情况，这种情况下我们要使用自定义模块的运行不刷新屏幕模式来实现快速画笔。该模式将在自定义模块部分详细介绍。另：截止到写作时间为止，未发布的Scratch 3.0项目之中，使用



抬笔落笔控制

清空一切笔迹图章

外观图章

自定义模块是默认不刷新屏幕的。

事件

Scratch之中圆顶的积木称之为脚本入口，也即是当他们等待的条件满足时，就开始下面连接的脚本。Scratch有六大脚本入口

1. 绿旗入口
2. 切换背景入口
3. 角色点击入口
4. 按键入口
5. 声控入口
6. 消息入口



事件积木

其中前五个入口是可以由人为发起的，最后一个是角色之间互发消息发起的。这里面最常见的绿旗入口我们几乎每天都要用到，但是我们真的了解绿旗入口么？

绿旗入口

我们究竟需要几个绿旗入口？前面在讲解层叠关系时说到了绿旗入口的Scratch内核注册顺序问题。虽然说计算机内部都是顺序执行的，但是我们还是在编写并发程序，也即是说许多绿旗入口在你一次点击之下会集体开始执行。在这种可怕的无法预测的情况下，我们需要做到的是防止两个绿旗入口下的脚本互相干扰。例如一段代码把一个变量改变了一个数值，另一段代码又改回去，一段说要停止程序，另一段却又开始等等。合理的拆分程序到不同的绿旗入口之下是一种必要的程序逻辑训练，如果有条件和耐心，不断地重新编写一个程序，以求更加的清晰和优化，这会极大地有助于编程思维能力的提升。

笔者的计算机课程老师经常教育笔者：如果发现程序错了，全部重新写一次，那将比你试图修改要快的多。而且你将有机会重新审视自己这段程序的逻辑，是最好的训练之一。

我们要坚持的原则是：

1. 把互相关联的东西放在不同的绿旗入口之下。
2. 检查程序流向是否会互相交叉干扰，同一个消息只有一个发送者，同一个变量同时只有一个更改者。

好事者又要提问了，如果同时一定有好多角色可以操作到某个变量，应该怎么办？

答案是，应该加锁，这个过程将在后续章节之中介绍。

切换背景入口

切换背景入口是为了不同关卡的场景切换而生，进入规定关卡则执行该入口下的内容。建议在设计复杂的多关卡游戏时，只在总控制部分放置切换背景更改状态的脚本。如果每个角色身上都带有此类脚本，而背景又经常切换的时候，逻辑将变得及其复杂，调试起来十分痛苦。

角色点击入口

角色的点击入口又要牵涉前文提到过的拖拽角色优先级最高。我们可以尝试简单的放下：当角色被点击则角色大小增加10。在编辑器状态下点击角色，角色将被拖拽起来而不会执行增加大小的积木。在播放器状态下，点击角色可以看到角色增加了大小。

而且更重要的一点是，这个入口是一次性激发的，也即是如果在播放器状态下对角色点着鼠标不放，角色大小只会增加一次。要想执行第二次必须松开鼠标按键再次点击。这种防止被多次触发的机制也让该入口成为Scratch最安全的入口之一，当然前提是在播放器状态下。

按键入口

刚才我们讨论了角色被点击入口，大家可能还不能理解入口被多次触发带来的困扰。现在麻烦就要来了。

假设我们要写一段脚本，当向上键被按下时，角色的y坐标增加10。使用如下代码，当我们长按向上键时，可以观察到角色首先向上移动了一下，接着开始快速向上移动，中间发生了一



个小的卡顿。好像向上窜了一下才往前移动，控制起来十分不流畅。

按键入口控制角色移动

为什么会发生这种奇怪的现象？

这是因为Scratch对于积木块执行速度的有特殊的控制。Scratch既想让按键可以单独的控制某个动作，不至于因为单独按键发生了连续执行的效果，又想让按键可以产生连续的控制这个动作。

这种矛盾的做法让按键入口很难作为角色移动的控制。一般来说我们要使用重复执行和侦测



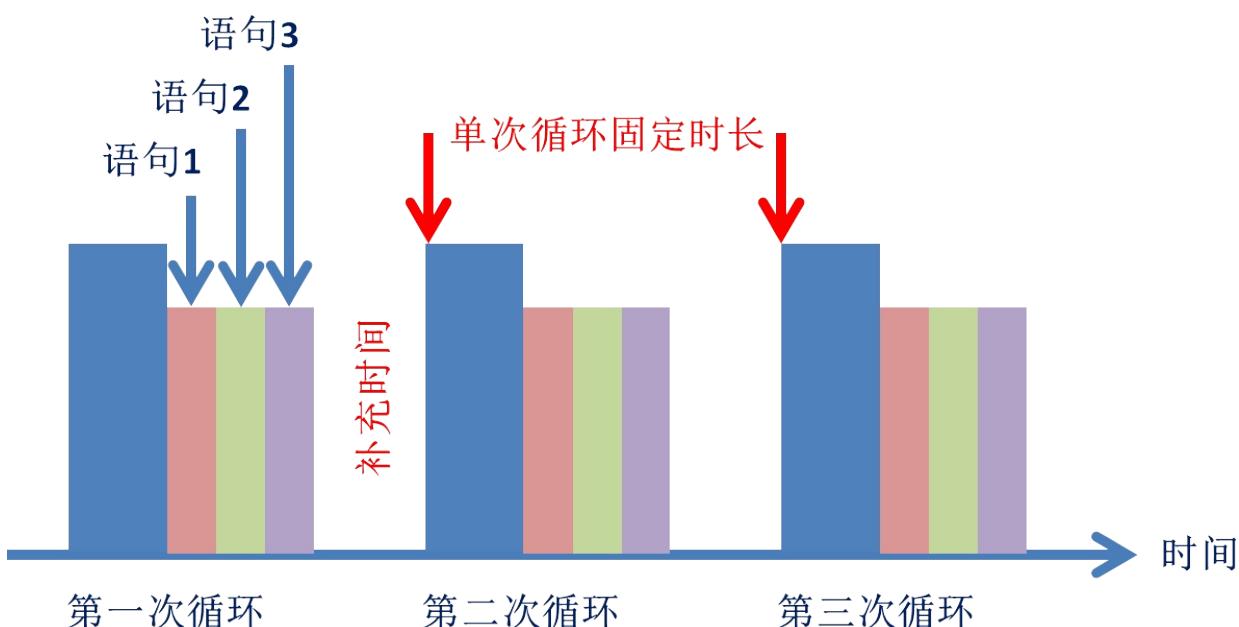
判断按键来实现稳定的角色移动控制。代码如下

重复执行加侦测判断控制角色移动

为什么重复执行加侦测判断的方案就可以稳定的控制角色移动呢？这要从游戏引擎说起了，任何一款用作游戏开发的引擎必然会自带一个执行速度固定语句。

我们要实现角色以一个固定速度移动，我们只需要让他固定时间长度往前移动固定步数即可。设想这个语句的执行时间不固定，角色的速度也会不固定，这样就没法实现最简单的匀速运动。

熟悉游戏开发的朋友可能知道，在Unity编辑器之中有一个FixUpdate()语句，可是实现固定时间长度的运行。高级的物理引擎也都是基于这个语句实现的。在Scratch之中，为了不把这个问题展示给初学者引起不必要的迷惑，开发者们聪明的将重复执行积木变成了这个固定时间长度语句。也就是循环中间装入不是很多语句的时候，每次循环中间的语句执行完毕的时候还会等一小下，把一次循环的总时长补到大约是0.03s左右（测量并不准确）。



重复执行时间示意

因此，我们把“重复执行”积木之中放入一个“移动10步”积木，就可以实现角色的移动。究其根本是因为重复执行锁定了每一次执行的时间。

好事者又想问，如果我们往重复执行里面放入许多耗时极长的语句，会不会挤爆重复执行语句？

答案是不会，一旦重复执行之中包含的语句超过了重复执行锁定的时长，重复执行的时间就被撑长了，撑到执行完中间包含的所有语句。

声控入口

Scratch可以调用麦克风识别响度数值，如果响度大于某数值则该入口开启。声控入口和角色点击入口一样是一个单次入口，也即是开启之后必须等响度小于这个数值之后才能第二次开启。不会产生多次开启的情况，也是一个比较安全的入口。

消息入口

消息入口也称为消息接收器，是六个入口之中唯一一个不能靠人为开启，只能靠程序控制的入口。角色的消息既可以发给别的角色，也可以发给自己，还可以发给舞台上的脚本。这部分内容在后续内容将会详尽介绍。

侦测

Scratch的侦测积木分为以下几类：

1. 距离和坐标侦测类
2. 角色信息侦测类
3. 视频响度侦测类
4. 询问回答和计时器
5. 碰撞侦测类
6. 按键侦测类

	数值	布尔表达式	
距离检测	到 [鼠标指针] 的距离	碰到 [鼠标指针] ?	是否碰撞角色或鼠标
鼠标坐标检测	鼠标的x坐标	碰到颜色 [■] ?	是否碰撞颜色
	鼠标的y坐标	颜色 [■] 碰到 [■] ?	是否颜色互撞
角色信息检测	x坐标 对于 角色1	鼠标键被按下?	是否按下鼠标
视频相关检测	视频 动作 对于 当前角色	按键 [空格] 是否按下?	是否按下按键
	将摄像头 开启		
	将视频透明度设置为 50 %		
响度检测	响度		
计时器	计时器		
询问和输入	询问 [What's your name?] 并等待		
	回答		

侦测类积木

侦测积木主要会产生两类结果。一类是数字，例如：坐标值、距离值、视频变动剧烈程度、响度、计时。另一类是布尔值，也就是真或者假，例如：是否按下了鼠标按键、是否按下了某键盘按键、是否碰撞某角色某颜色等。

距离和坐标侦测类

该类积木主要作用是获得和某角色或者鼠标指针的距离，获得鼠标指针的位置。可以方便的实现一些例如防御力场之类的效果，例如距离某角色50步就无法继续前进。也可以获得和鼠标之间的距离用来实现鼠标操纵。

角色信息侦测类

该积木可以实现在某角色身上的脚本中获知其他角色状态的功能。例如某boss想根据玩家角色的坐标值做出一些反应，就要依靠这块积木获得玩家角色的坐标值。

如果不使用这块积木，我们仍旧有替代方案。那就是把玩家角色的坐标值赋值给一个全局变量，然后所有人都可以看到这个变量。但是，这难道不是多此一举么？为什么要绕一个弯做这样的事情？

其实这种做法是有使用场景限制的，如果某个数值有许多人都要读取，那么最好把它做成一个全局变量。这样当我们要改写程序的时候，只需要改变这个全局变量就好，只需要动这个地方。假如我们在每一个用到这个数值的地方都用侦测角色信息的积木这样使用，那么我们如果要改动就要改动好多地方，这时候疏漏和麻烦就找上门了。

不要一味地使用角色信息侦测积木，合理的使用变量可以方便我们做出易于更改维护的程序

视频响度侦测类

这是一个帮助文档也稀里糊涂解释的东西，它包含两个内容：

1. 视频动作(motion)对于角色
2. 视频方向(direction)对于角色

视频动作其实是角色对应视频区域之中，画面活动频繁程度的一个数值。角色对应区域画面如果没有任何移动，那么该数值将会很低，甚至接近零。为什么不会真的变成零呢？这是因为摄像头和环境都有微小的抖动，肉眼判断不出的运动计算机可以感知到。如果该区域画面之中活动剧烈，这个数值将会激增。Scratch官方以此为根据做了一个切水果的游戏，就是检测落下水果对应视频区域画面活动剧烈程度，判断是否被人手在视频中切中。

好事者又来提问了，假如我们让一个苹果从舞台上方逐渐掉落，那么苹果背后的视频和苹果发生相对运动，在苹果的眼中视频动作应该很高才对，是这样的么？

答案是不会的，我们可以简单地实现一下这个场景，就会发现这个数值根本就在0左右没有动。原因是Scratch的运行和拍电影一样是一帧一帧执行的，只是速度很快的时候我们肉眼无法分辨觉得它在运动。每一次视频动作检测都在角色移动了微小一步之后发生，这时角色已经停下，下次微小移动还没发生，当然不会和视频有相对运动。



角色移动中的视频动作检测

视频方向这个概念就更加迷惑了，这个数值好像一直在狂跳从没稳定过。其实这个视频方向指的是角色对应视频区域里面视频运动方向。如果不理解的话，我们来做个试验。

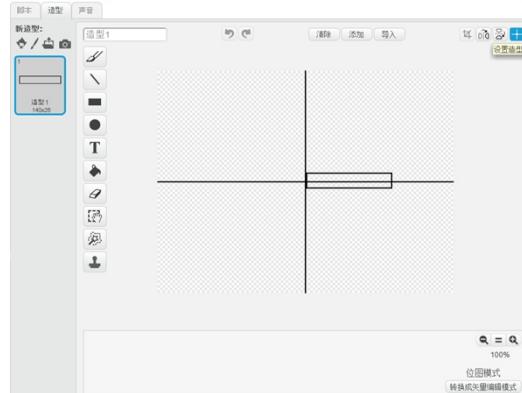
角色1大圆球



角色2方框



方框中心设置在边上
指向默认角度90°



视频方向检测实验

- 首先画一个占据舞台的大圆球作为角色1，如果没有完全占据则调整外观角色大小。首先将圆球设置外观特效为虚像80，打开摄像头，然后把视频方向对于角色1的赋值给变量dir。
- 画一个指向默认方向90°的方框，中心设置见上图。让该方框的方向重复执行始终指向dir。
- 运行观察，在大圆盘内挥手观察角色2指向的方向。

我们可以见到，手臂在视频中朝向哪里挥动，角色2就像指针一样指向了哪里。但是值得注意的一点是，这个方向十分不稳定，跳动还是有些剧烈，不适宜直接使用。本书将在后续章节滤波器之中讲述如何让该数值稳定下来以便使用。

此外关于视频处理要补充的一点是：无论我们将视频透明度设置成多少，哪怕设置成100完全看不到视频图像，动作和方向检测依然不受影响的生效。

相对检测相对于视频检测就容易理解的多，无非是麦克风输入的声音大小会变成一个响度数值。在之前的响度入口之中也可以直接使用。在此不做详述。

询问回答和计时器

熟悉文本语言的读者应该一眼就瞧出来询问回答就是C语言中的scan，为了输入变量而生。一般的用法是：

1. 向询问框之中放入字符串变量，例如列表中存储许多问题，抽出一条放入询问框
2. 询问之后程序会等待用户作答，按Enter键之后，这个答案就被存储进了回答变量之中
3. 将回答变量用在别的地方，例如判断是否答案正确

计时器变量大家十分熟悉，但是却并不一定知道如何把他玩出花活。这里就举两个例子帮助大家打开思路，例子实现在后续章节介绍

1. 使用计时器和自定义的变量配合实现蓄力攻击的效果
2. 使用计时器实现连击效果，要知道连击的触发只在一段时间内有效

另一件值得注意的事情是，Scratch的计时器和等待1秒积木，归根结底和其他语言之中的wait(),delay()差不多，并不是真实世界的时间。其实Scratch是留了最精确的时间取值点的，那就是一个不起眼的“自两千年至今的时间”积木。

1. “等待1秒”积木的运行和Scratch的软件运行有关，软件卡顿则计时卡顿，所以最不准确。
2. “计时器”积木的运行和CPU的运行有关，计算机卡顿则计时卡顿，所以第二不准确。
3. “自两千年至今的时间*1000”（或者1000以上）所获得时间是获取网络时间，最为精确。

碰撞侦测类

这类积木可以说是最常用到的积木了，角色之间发生交互，被子弹击中等都要靠它们来实现。是否碰到某角色和鼠标大家凭直觉都会使用，就不在此赘述。这里要主要说的是碰到颜色的应用场景。

Scratch提供了两类颜色碰撞侦测，角色碰到某颜色，角色的某颜色碰到某颜色。这里依靠取色器规定颜色，当然也可以用我们之前介绍画笔取色器时说到的RGB后门来规定颜色。颜色碰撞有两类用法：

1. 区分碰到一个物体上的不同部分。我们称之为一化为多
2. 用许多角色共有一种颜色碰撞检测语句，来代替多条碰撞角色检测语句，我们称之为多归为一

一化为多的举例：由于Scratch没有自带物理引擎，许多角色间碰撞和反弹实现起来总是比较尴尬，补充方法就是依靠颜色检测来实现。假如我们知道是撞到了一个角色的侧面还是撞到了顶面。这时候就只能画上一些可以区别的颜色，通过颜色就可以判断是撞到了哪个部分。

多归为一的举例：假如舞台上有好几种敌人，玩家角色碰到敌人会阵亡。那么最好在设计的时候就把所有的敌人都标记上共同的一个颜色，只要在玩家角色上检测是否碰到该颜色，就知道是否碰到了敌人。否则我们就要写出碰撞所有敌人的语句十分麻烦。

此外，角色的某颜色碰到某颜色，适用于判断角色的某个部分是否碰到另外角色的某部分。

按键侦测类

该类积木的使用场景均在键盘和鼠标控制之中，后续章节会主要介绍。

运算

Scratch的运算分为三类

1. 数字运算
2. 字符串运算
3. 布尔值运算



运算积木

数字运算

Scratch的数字运算没有括号，每一个积木就相当于一个积木，所以请大家关注一下运算的优先级关系。尤其是多层嵌套之后往往会有花眼。这时候的解决办法就是，多设置几个中间变量打散计算步骤，这样也可以检查计算中的数值出现的问题。

除了四则运算之外，Scratch提供了两个方便的运算积木，求余和四舍五入。在文本语言之中，求余可以方便我们获得数字的某一位或者某几位。然而在Scratch之中，数字和字符串是界限不明显的。我们可以把数字当成字符串，接着使用字符串运算的提取某一位字符来实现获得数字的某一位。

Scratch还给我们提供了包括乘方开方、上下取整、反三角函数和指数对数在内的一个较为强大的科学计算积木，在后续的数学章节之中我们会着重介绍这些工具。

字符串运算

Scratch之中的字符串工具可以使用在许多地方，包括使用字符串链接工具和变量组合来实现对于角色外观声音背景等的切换。读者可以自行尝试，所有可以从本地上传的例如音乐图片等均可以通过字符串变量来控制切换。

例如我们向一个角色上传了十段音乐，命名分别为：音乐1、音乐2、音乐3……我们可以使用



如下的积木来实现批量播放音乐。

字符串连接批量播放音乐

布尔值运算

首先要说的是什么是布尔值，为什么要有布尔值。

布尔值只有两个，一个是True另一个是False。布尔值的产生是为了让计算机学会应对真实世界情况。小到游戏中一个角色被子弹击中会扣血，大到火箭发射按钮，全部都是由布尔值操纵的。

设想一种情况，我们想让计算机在周围声音很大时作出一些反应，那么步骤应该是：

1. 真实世界被传感器处理，例如我们可以获得一个响度数值，这个响度数值有时大有时小。
2. 我们要设定一个值50，然后写一个布尔单元：如果大于50就为True，如果小于等于50就为False。
3. 计算机就会在现实世界声音很大后作出一些反应。

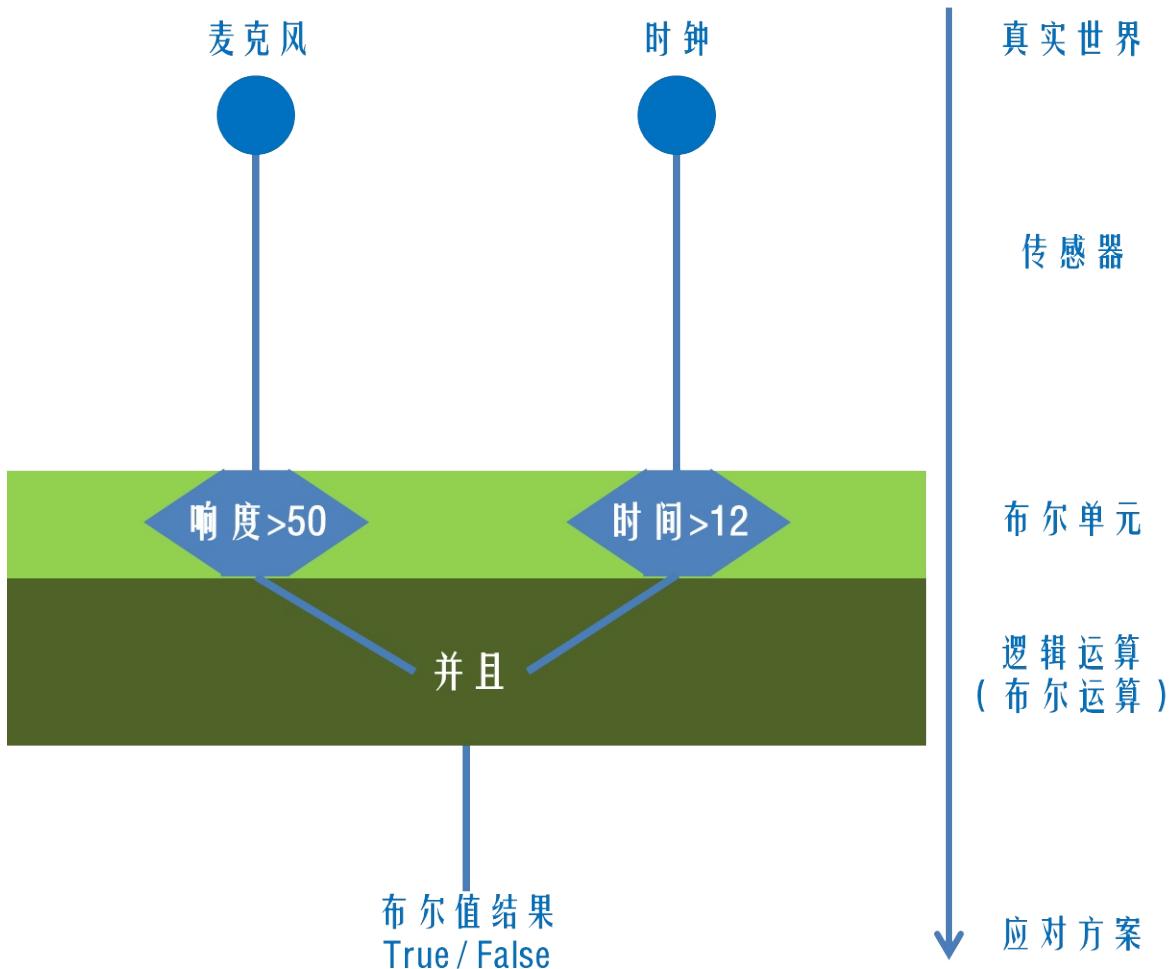
但是现实世界往往更为复杂，让计算机作出判断的条件也更多，甚至条件之间要互相影响。比如我们想让计算机在中午12点之后对于周围声音很大作出反应，那么步骤将被改写为：

1. 真实世界被传感器处理，例如我们可以获得一个响度数值，这个响度数值有时大有时小。
2. 另外有一个时钟传感器，可以获得时间。
3. 设定“声音大于50”布尔单元，设定“过了12点”布尔单元
4. 进行布尔运算得到：“声音大于50”并且“过了12点”
5. 计算机将会对过了12点之后的声音很大作出反应

这样我们处理问题的完整流程应该是：

1. 真实世界
2. 传感器获得数值

3. 设置布尔单元处理单个传感器
4. 对于多个布尔单元进行布尔运算得到一个结果
5. 让计算机对于计算结果布尔值进行反应



布尔运算的意义

数据

Scratch自带两种数据结构，变量和列表。一共有以下几类积木：

1. 变量赋值、变量增值
2. 列表增删改查，列表长度信息



数据积木

变量

变量在创建时会被询问作用域，也即是询问是对舞台上所有角色有效，还是只对当前角色有效。对所有角色有效的变量我们称之为全局变量，只对于角色自己有效的变量我们称之为私有变量。不同角色的私有变量可以重名，这就省去了我们命名的麻烦。

如果想给每一个克隆出来的角色加上一个血量值，那么我们必须用到私有变量。如果不小心设置了全局变量，那么一个角色死亡时，所有的克隆角色都将死亡，这显然是不合理的。

还应注意的是，在以往的Scratch版本中，我们在导出角色到计算机，或者是复制角色的时候。角色身上的脚本使用的变量全部会被保存下来。我们再把这个角色导入另一个程序之中，运行的同时，角色身上原有的，所有当前程序没有的全局变量，统统会被Scratch自动创建出来。这是一件很危险的事情，在我们编写复杂程序的时候，很容易引起变量间的混乱。而在最新版的Scratch中，这个问题已得到解决，复制角色的时候，只有私有变量会保存下来。

不管怎样，我们要尽量的使用私有变量，可以把私有的变量都设为私有，这样在复制或者导入导出角色时，会减少许多因为全局变量带来的麻烦。

列表

Scratch的列表的操作十分简单，无非是增删改查四类，再加上一个判断列表长度。值得注意的是，列表并不会在程序运行时自动更新，这种特性有些类似于数据库。如果要刷新列表，请使用删除全部列表数值积木。

列表创建之后，会在舞台上显示，这时在舞台上右键列表可以导入导出数据。列表支持txt格式，以换行符区别列表不同的项目。换句话说，我们创建一个txt，在里面写入数据，回车换行就可以把数据导入到Scratch的列表之中。

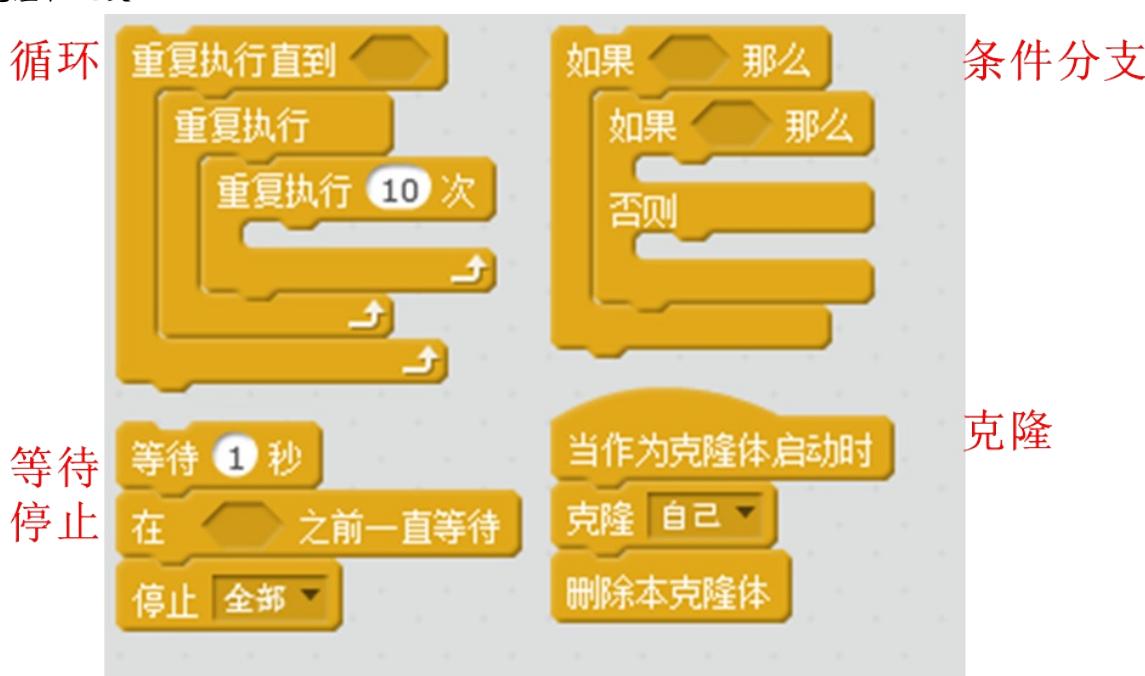
这种导入有什么用武之地呢？这里列举几个场景供大家参考：

1. 用别的语言写好的曲线坐标方程，可以通过x、y两个列表导入Scratch绘制复杂图形
2. 乐谱可以通过列表导入实现Scratch音乐播放
3. 语言问答库可以通过列表导入之后制作问答机器人

控制

Scratch的控制积木分为四类

1. 循环类
2. 条件分支类
3. 等待停止类
4. 克隆相关类



控制积木

循环

Scratch提供了三种循环积木，分别是

1. 无限重复积木
2. 计数循环积木

3. 条件跳出循环积木

无限循环积木也称之为“死循环”，应用极其广泛。任何一个系统在启动的时候都进入了一个死循环，无条件的重复一些语句，例如监听某个按键是否按下，判断是否碰撞某个角色，一直重复某种运动等等。在前面按键入口的地方曾经叙述过Scratch死循环的运行最短时间固定特性，在此不重复讲述。

要注意的原则是：尽量少让消息入口下面使用死循环，可能会产生一些令人迷惑的结果。

计数循环积木，可以循环规定次数，然后继续后面的积木运行。这个积木值得注意的是，我们是否可以动态的更改循环次数？例如我们用一个变量*i*来规定循环次数，在该循环之中不停地改变*i*的数值，那么循环的次数会根据*i*来改变么？

答案是不会，Scratch的计数循环只会记住第一次传入的*i*的数值，并严格执行这么多次循环然后跳出循环。这是为了防止儿童在编程时动态的改变循环次数产生错误。Scratch里面不可以新建变量和列表，原因也是如此，这样可以防止儿童在使用时产生程序崩溃。

条件跳出循环积木每次执行前都会判断条件是否成立，如果条件不成立就不进入这一次循环。这个积木很好的补充了Scratch没有break语句的缺陷。

条件分支

条件分支积木十分简单，只有if和if else两块，不做赘述。

等待和停止类

等待？秒是Scratch最常用的控制运行节奏的语句，经常和重复执行动作配合使用。但是要注意的是，因为循环执行最短时间固定的特性，等待？秒如果设置的时间过于小，将变得毫无意义。因为Scratch还是会自动补全时间到循环的固定最短时长。

需要注意的是等待？秒积木也会自动补全时间差，该积木有一个最短运行时间大约是0.033s，和循环最短时间一致，具体应用细节请参考后续章节scratch疑难问题举例

在条件到达之前一直等待积木十分有用，可以方便的帮助我们消除某些二值条件的重复影响。

例如，当我们按下空格的时候要执行一个动作，不论我们是用按键入口方法实现，还是用重复执行+条件判断按键是否按下来实现。当我们长按空格的时候，动作会被病态的重复好多次。这时候我们就要依靠等待“空格键按下不成立”的来实现长按屏蔽。只有当空格键抬起，这条等待语句才可以继续向下执行循环。



防止空格长按的正确和错误写法

停止积木有三种：

1. 停止全部，等同于绿旗边上的红灯，结束全部舞台上的程序
2. 停止当前脚本，停止本段脚本
3. 停止角色其他脚本，停止角色身上除了本段脚本之外的所有脚本

需要注意的是：停止全部有时候并不能真的停下我们全部的程序，原因可能是程序有绿旗入口之外的别的入口。虽然程序被停止了，但是入口又被按键声音等激发打开。妥善的停止程序运行是一个很细致的工作，有时候游戏角色阵亡，但是还有一系列特效和动作没有播放，这时候不能贸然停止全部。要用消息等待等方式确保所有舞台其他角色都停止运行，然后再停止全部。

克隆相关类

Scratch和一系列游戏编辑器一样，都需要克隆功能，来制造批量的敌人、子弹、背景等角色。克隆积木分为

1. 克隆动作

2. 克隆体入口
3. 删除克隆体

克隆动作可以由本角色发出，克隆自己。也可以由别的角色发出克隆某角色。推荐的方法是，使用舞台作为克隆的总控制器。防止脚本理解和管理混乱。

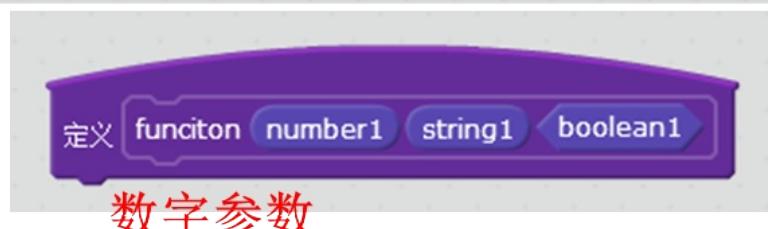
克隆体入口是决定克隆体和母体区别的关键，母体并不会运行该入口下的脚本，只有成为了克隆体才会执行该入口以下的脚本。通常的做法是，将母体大小和造型等设置完毕后隐藏，在克隆体启动入口下将克隆角色移动到规定位置后显示。克隆之中的细节问题将在后续的克隆专题之中详尽描述。

更多积木

Scratch的更多积木之中涉及函数制作和硬件相关两种，本书不牵涉任何硬件的内容，单独介绍自定义模块（函数）制作。



定义条



布尔参数

数字参数

字符串参数

自定义模块定义

自定义积木和文本语言的函数功能类似，只是没有返回值的存在。意义在于打包某功能之后重复使用。可以设置参数，这里包括了数字参数、字符串参数、布尔参数三类。分别对应Scratch的三种变量类型。

好事者又来提问，如果我们定义两个自定义积木，分别定义他们的参数。他们都包含一个叫做num的参数，那么这个从定义条上拖出来的参数可以互换着使用么？

答案是，可以的。Blockly的底层还是文本语言，重名的不同函数参数可以互相放置，不会产生问题。

自定义积木最关键的使用方法是运行时不刷新屏幕，一旦勾选了这个选项，入口下的所有循环都被取消了固定的最小运行时间，运行速度大约会提升1000倍，所有程序跑的飞一样快。但是一旦不小心往这里面放置了很多语句，电脑就会被卡住。这也是风险所在。

还需要注意的是Scratch自带的加速模式（编辑菜单下）并不建议使用，因为很可能会让计算机死机。

由于Scratch是一款面向少儿的编辑器，为了避免儿童的随意使用造成软件崩溃，引入了许多不够完善的特性。刚接触时往往會陷入一些奇怪的疑难问题之中。本章将会从以下几个方面介绍日常使用Scratch会碰到的边角问题：

1. 绘图和外观的坑
2. 看不见摸不着的事件控制
3. 不知道该用什么变量的克隆体
4. Scratch的谜之容错
5. 拖慢脚本的罪魁祸首

1. 绘图和外观的坑

1. 任何图片都不能离开舞台
2. 矢量图和位图切边
3. 角色和克隆体的复杂层级关系
4. 默认角度90°
5. 消失的几种姿势

任何角色都不能离开舞台

为了防止不小心把角色移动出舞台找不到，scratch做了几个限制。首先任何角色在移动到舞台边缘的时候会强制留下一个边缘，刚好可以用鼠标拖拽它回到舞台之中。例如我们使用坐标移动积木让某角色移动到x1000,y0。这样角色实际上的x坐标在250多就已经卡住了，刚好在舞台上漏了一个角。

对于角色的限制不光在于位置，还包括了角色的外观大小。我们让角色大小设定为极大例如1000，在外观积木盒子最下方的大小变量之上用鼠标勾选之后，可以观察到，角色根本没有变成1000，而是卡在了几百就不在增加。

相应的，如果我们给角色设定一个1的大小甚至-1000的大小，角色大小就被锁定成了5。这是为了防止角色太小无法画在舞台上。

理解了scratch的这个角色不能离开舞台的机制之后，我们就可以考虑一些问题。如果想让一个克隆体从画面外移动进画面，这个动作可以做到么？答案是不能。因为角色一旦设置一个画面外的位置，scratch就会强制把角色的一个小角拉到舞台上，这样画面边界一定会留一点角色。根本无法让角色完全从舞台外移入舞台。

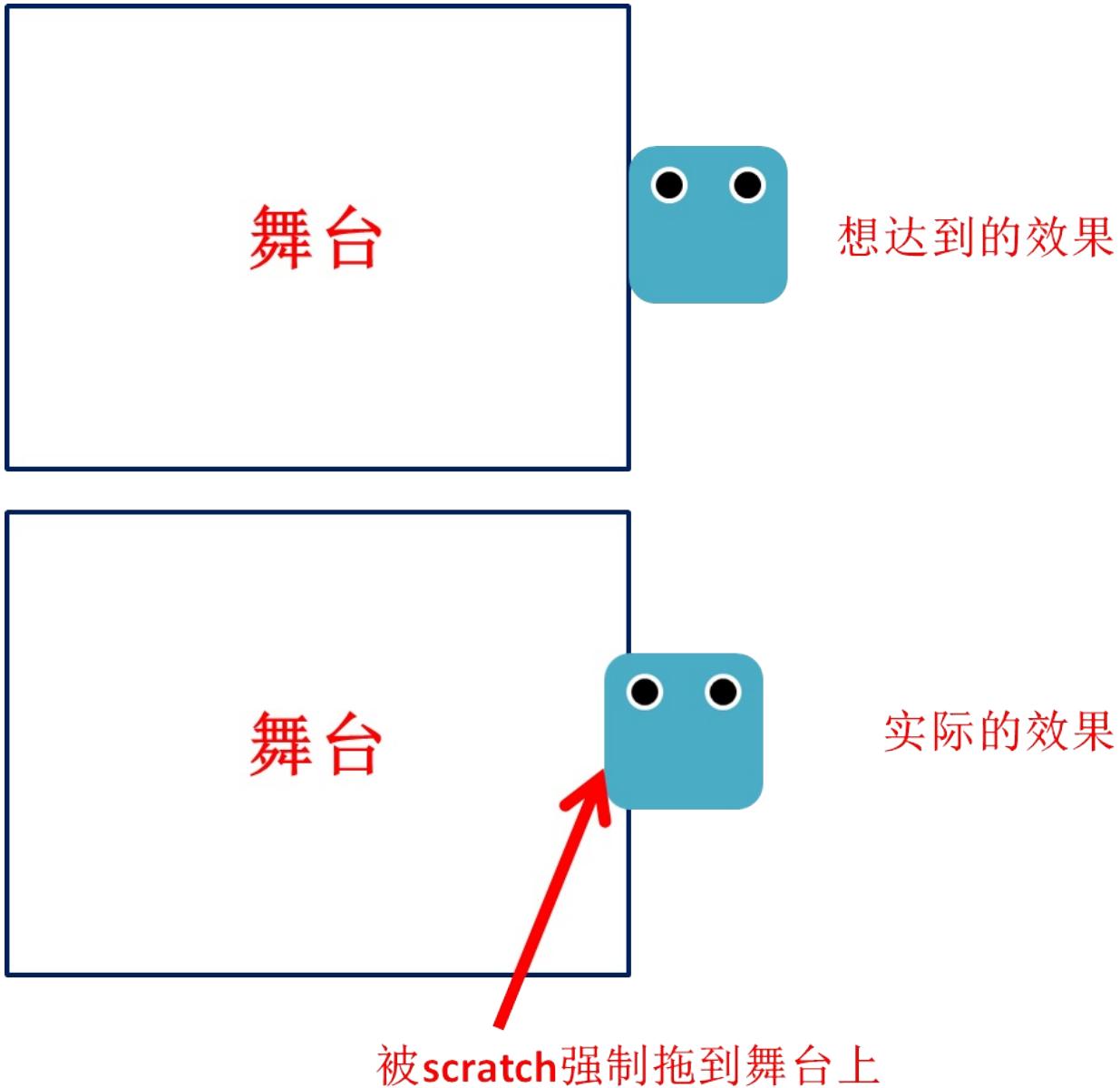


图 不能完全从舞台外移动进舞台

如果我们一定要让角色从舞台外移动进舞台，就要把入口这个边界用一个不透明的角色遮盖住。这样在视觉效果中，就看不到这里的异常了。例如我们想用scratch做一个横版卷轴游戏，类似超级马里奥。地图的卷动就需要从舞台外移动进来，这时候就需要把边界遮住防止玩家看到这里堆了奇怪的东西。

矢量图和位图切边

首先要界定什么是矢量图什么是标量图（位图）。我们做如下界定：

1. 矢量图是指软件自己知道如何可以画出来的图，可以一条线一条线的画出来
2. 标量图是软件只拿到了一副画，这幅画被毁掉了软件也不会补充回来了

关于矢量和标量的定义和意义大家如果有兴趣请参看part3的数学概念

scratch对于外形自带了两种编辑模式可以互相切换，不过非常不建议各位读者使用这些编辑功能，因为十分不完善的编辑功能影响体验。这里要强调的是几个原则：

1. 从本地导入的图片，即时原来是所谓的“矢量图”格式，例如png等。导入scratch之后也默认变成了一个位图，因为它并不是scratch自己画的
2. scratch承认的矢量图，只有两类。一类是造型之中切换矢量图编辑模式自己绘制的图片，另一类是scratch角色库之中自带的角色
3. scratch承认的矢量图在造型之中编辑。如果不小心被我们在编辑中移动出画布，还可以移动回来。但是本地导入的图片如果不小心移动出编辑画面就会发生切割丢失现象。

为什么会发生切割丢失现象呢？原因是图片移出了编辑区之后都会发生切割，但是矢量图scratch知道怎么画回来，而位图scratch不知道怎么画于是就丢失了。如何避免这个情况发生呢？这里我们要首先使用位图编辑的选择工具选中图片的主要内容，通过缩放和拖拽来设置图片中心。防止在操作过程中让图片移出编辑区，这样就不会发生切割丢失现象了。

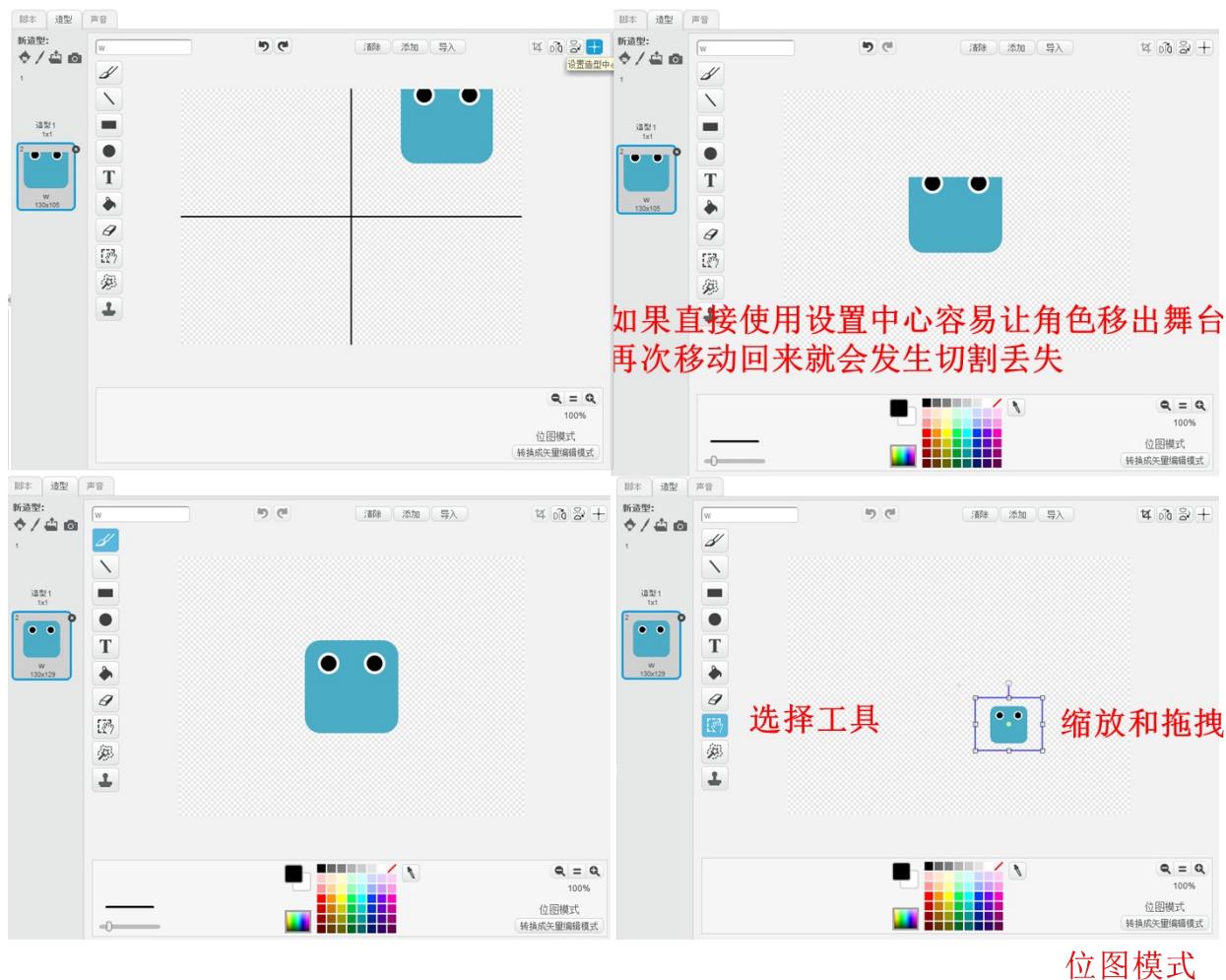


图 如何避免设置中心时候发生切割丢失现象

角色和克隆体的复杂层级关系

在scratch之中有一种复杂的层级关系和管理原则

1. 角色在舞台上被拖拽就会默认移到顶层。
2. 舞台无论如何都在最下层，画笔和图章都会绘制在舞台上。换句话说任何不透明角色都可以挡住画笔和图章。
3. 任何新创建的角色都会默认移到顶层。
4. 角色的克隆体默认在角色下一层。
5. 角色再次克隆的克隆体再角色和前一个克隆体之间。因为首先它是新生成的物体放在最顶层，其次执行角色本体在克隆体之上的操作。

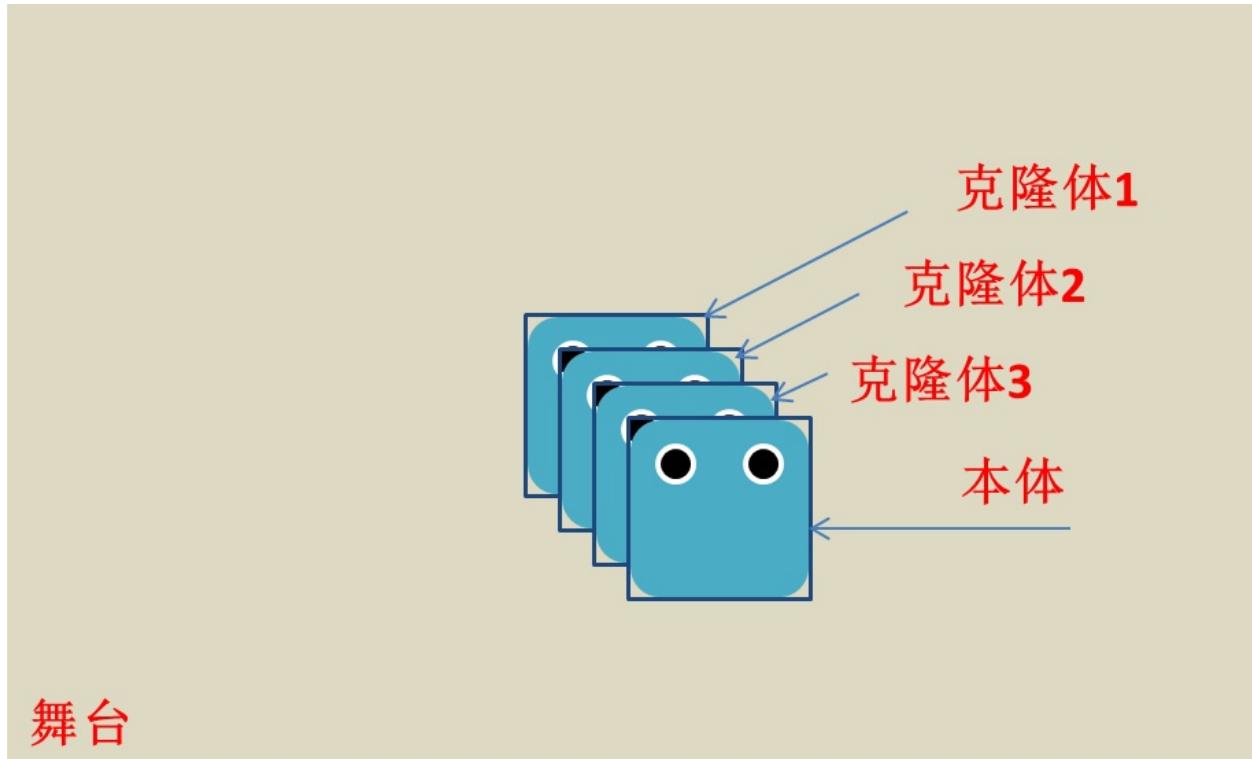


图 角色和克隆体层级关系

默认角度90°

scratch之中角色的默认角度是90°，我们在导入图片制作角色时一定要注意尽量让俯视图片的头部方向朝右。例如我们要做一个火箭，火箭的头部如果朝上，后果就是我们用移动积木和方向积木来控制火箭时，火箭永远都是横着飞，这是我们尽量要避免的。但是如果是横版角色，例如scratch初始化的那只猫，就不存在这个问题了，因为它只有左右两个方向。

消失的几种姿势

scratch之中可以通过三种方式让一个角色从我们眼前消失：

1. 隐藏，最常用的方法，但是要注意隐藏是一个下次运行仍旧会保存的状态，要和显示积木配合使用。隐藏了之后无法判断碰撞其他角色，但是请注意，这时候仍旧可以在隐藏角色身上判断碰撞边缘和鼠标指针。这是scratch的特殊细节。

2. 透明度特效，在外观特效里面可以用透明度特效让角色消失，这时候我们仍旧可以让角色判断和别的角色的碰撞，因为他还在舞台上。这样做的副作用是带来运行速度的减慢，如果有大规模的克隆体请尽量避免使用这种方法防止运行卡顿。
3. 选取角色图片和背景颜色相同，角色自然就隐藏起来了。有些区域我们想看到角色就把该区域的背景换成不同颜色即可。这种方法十分巧妙，并不增加scratch的运行负担而且仍旧可以正常判断碰撞。

2. 看不见摸不着的事件控制

1. 鼠标点绿旗的迷惑
2. 消息等待
3. 等待0秒？等待-1秒？最小等待时间是多少0.033平均
4. 藏起来就不刷新屏幕的列表和变量 没有anyone刷新屏幕就飞快
5. 死循环嵌套的后果
6. 计数循环变计数能否进行
7. 拖拽优先级最高
8. 并行的条件竞争 碰到边缘反弹和等待碰到边缘
9. 真实时间的获取

鼠标点绿旗的迷惑

在scratch之中，绿旗是我们最常使用的入口。但是如果我们让角色在鼠标按下的时候做出一些动作（例如发射子弹），这时候问题就来了。一旦我们点击绿旗开始还没准备好真正开枪，scratch会马上发射一个子弹。这是因为scratch在绿旗入口的瞬间抓到了鼠标按下的事件。为了解决这个怪异的行为，我们必须使用一个称之为锁的东西。scratch给我们提供了很方便的实现锁的工具，那就是“等到直到”积木。我们只需要在绿旗入口下跟一句：等到直到鼠标键按下不成立。这样在我们点击绿旗之后，程序会等待直到我们把鼠标键松开，才继续向下运行。就好像一个锁把程序运行的脚步锁住。这个概念我们将会在后文经常用到。在真正的文本语言之中，锁的实现是用变量记录状态做的。这种方法略显复杂。在scratch之中经常使用的就是等待直到积木。

绿旗点击锁

消息等待

在scratch之中，我们经常会接触到消息发送接受。最常见的错误就是在消息等待过程之中使用死循环。“广播消息并等待”积木将会等待所有的消息接收者执行完毕，如果有一个接受者身上带有一个死循环重复执行。那么它将永远无法执行完毕，广播消息并等待的积木也将天长地久的等下去无法继续运行后面的内容。

等待0秒？等待-1秒？最小等待时间是多少**0.033**平均

在scratch之中我们最容易搞不清楚的就是等待1秒积木。其实等待积木和任何重复执行积木都在一次运行或者一次循环之后会将运行时间补充到0.33秒。如果我们将等待积木写等待0秒或者等待-1秒，再测量其运行时间，结果都将是0.33秒左右。



等待0秒

藏起来就不刷新屏幕的列表和变量 没有**anyone**刷新屏幕就飞快

是不是scratch的所有重复执行都会补充时间到0.33秒呢？这里有例外的。如果我们把重复执行之中只放一个操作列表的语句，例如加入一个元素。然后我们先将列表隐藏，点击绿旗运行大约1秒时我们打开列表显示观察会发现，列表的元素有好几十万。这是因为重复执行积木中没有任何要刷新屏幕的积木，所以默认全速运行，也不会补充时间。如果我们把列表显示出来，这时候再运行，发现列表元素增加的过程慢了许多。这是因为列表的显示和更新就是刷新屏幕的动作，重复执行积木会补充时间。



先隐藏运行后再显示

列表和重复执行刷新屏幕

死循环嵌套的后果

新手经常犯的错误还包括死循环嵌套死循环。外层的死循环运行第一次的时候就进入了内层的死循环，从此陷入了内层的死循环再也无法出来。也就是说，外层循环变成了一个摆设，因为就运行了一次根本不会循环。请大家一定要注意。



死循环嵌套

计数循环变计数能否进行

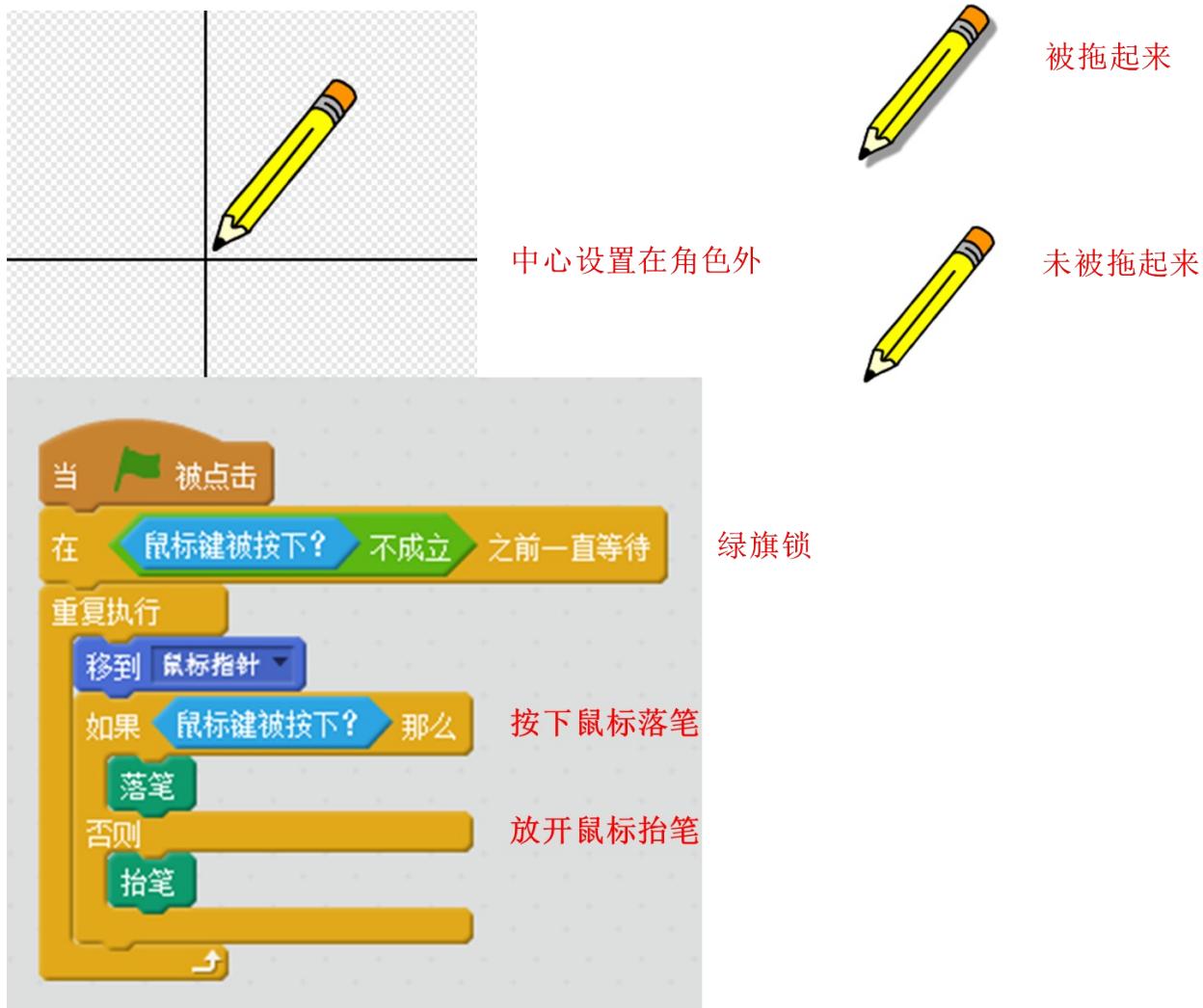
假如我们用一个变量来控制计数循环，如果这个变量在循环之中被改变了，那么循环次数会动态的改变么？答案是，不会。计数循环在第一次运行时就已经被锁定了运行次数，无论我们怎么改这个变量，循环次数是不会发生改变的。这是为了防止出现一些运行错误。



变量控制计数循环

拖拽优先级最高

在scratch的编辑模式（就是平常的模式，相对于全屏播放模式来说的），运行程序时，拖拽的优先级是高于点击判断的。如果我们程序中设定了点击角色之后有一些操作，那么点击角色之后会首先把角色拖拽起来，而不是执行这些操作。假如我们要做一个画笔，点击鼠标之后落笔开始画。点击画笔之后首先会拖拽起来画笔图案，而不是首先落笔。解决这样的问题有两个办法，第一个是永远在播放模式下运行，且不去勾选播放模式可以拖动的选项，这样就不会出现拖动的情况。第二种是将画笔图案的中心设置在图案外，这样在点击和移动到鼠标时，鼠标不会碰到图案，从而不会产生拖拽。巧妙地解决问题。



图案中心设置解决拖拽问题

并行的条件竞争 碰到边缘反弹和等待碰到边缘

在scratch之中，每一个程序入口下的脚本都是并行的。这样就会产生竞争，如果两个并行的脚本同时操纵一个角色，就会改来改去发生错误。例如我们在角色身上写两个脚本，一个让角色重复执行移动，碰到边缘就反弹。另一个脚本重复执行判断是否碰到边缘，碰到边缘就让角色说：“碰到边缘”。这样的运行，偶尔角色会说碰到边缘，偶尔不会说。就是因为发生了竞争，碰到边缘的时候：A) 如果碰到边缘反弹的积木先执行，角色脱离了边缘才进行碰到边缘检测说话，这时候不会说话。B) 如果碰到边缘检测说话先执行，那么角色就会说话。A和B两种情况我们并不知道哪个会先发生，这就是并发竞争。需要注意的是，这种竞争产生的bug偶尔会出现偶尔不会出现，十分难以察觉，属于最难搞定的bug之一。



并发竞争脚本

真实时间的获取

在scratch之中，有许多和时间有关的积木，我们来看一下他们之间的区别。

1. 等待1秒积木
2. 计时器
3. 自2000年至今的天数

最不准确的积木就是等待1秒，前面我们已经说过，该积木会带来三十分之一秒的刷新延迟。本身也不适合记录精确时间。计时器看上去比等待1秒积木靠谱的多，但是依旧有一个问题，如果我们的程序有些庞大，运行速度整体变慢，计时器也会变得很卡，这时候时间也不做准。最准确的时间是自2000年至今的天数，这个积木块用的是计算机时间，我们把这个积木乘以100000，就会得到秒级别的真实时间。当然我们可以获得更小的时间分辨率，但是意义不大。请大家自己探索。

3. 不知道该用什么变量的克隆体

1. 私有变量和私有改动
2. 消息入口的继承
3. 使用消息指数克隆
4. 网格锁定
5. 克隆体整理
6. 克隆体克隆别的克隆体的队列设置

scratch的克隆体是不大容易理解的概念，其中经常使用的一些私有和公有变量的概念更是抽象。下面我们来例举一些常见的使用方法。

私有变量和私有改动

在克隆体之中，我们为了独立控制每个克隆体，经常会给克隆体设置私有变量。实际上我们是把私有变量设置在母体身上，当克隆动作执行时，克隆体身上也复制了这个私有变量，但是这个私有变量就是属于克隆体的，和母体再无关系。

实际上克隆会复制以下事情：

- 1.母体角色当前的外观造型
- 2.母体角色当前位置坐标
- 3.母体身上全部的脚本
- 4.母体身上全部的私有变量和私有列表

前两条容易理解，第三条说克隆体复制了母体的全部脚本，那么当作为克隆体启动这个入口的意义在哪里？其实克隆体也复制了母体身上当绿旗被点击入口的脚本，只是绿旗这时候没有被点击所以不满足这个入口条件不会运行，而作为克隆体启动入口这时候刚好满足条件可以运行。如果母体身上带有一些消息入口，克隆体同样会复制下来，当消息广播时同样可以运行。我们可以通过消息来控制克隆体。第四条说母体身上全部的私有变量和私有列表。那么公有的呢？答案是公有的不需要刻意复制，谁都可以自由使用。而私有变量如果不复制，在别的角色身上是看不到没法用的。

另一个关键的事情是：积木盒子之中角色的xy坐标、方向、造型编号等可以打钩显示的变量，全都默认是私有变量。请大家注意区分，私有变量显示时有“XXX：变量名”，而公有变量只有变量名。

消息入口的继承

刚才提到了克隆体可以复制母体的消息入口，这个机制可以很方便的实现克隆体的控制。例如我们要某个消息可以把所有的克隆体清除掉，那么简单的在母体身上写下脚本：接收到消息删除本克隆体，这样就可以做到。这里的消息可以帮助我们实现许多集体效果。例如我们想通过一个消息让所有的克隆体震动一下，做出一个炫酷的效果。我们可以让消息入口下跟一个x坐标增加-20，等待0.1秒后x坐标增加20。其他用法请大家自己探索。

使用消息指数克隆

如果我们想通过很少的代码大量的克隆角色，常用的方法有：

- 1.重复执行+克隆某角色（运行速度慢有卡顿感）
- 2.把重复执行+克隆做成一个运行时不刷新屏幕的自定义积木
- 3.让克隆体克隆自己

这里主要说一下第三个方法。学过幂指数就应该知道。如果我们让克隆体作为克隆体产生时克隆自己，克隆体将会以2的指数来增长。scratch为了防止克隆体太多拖慢速度，会限制同时存在的角色总数目为300个。

我们还可以用消息来控制角色的克隆过程，为了区分克隆出来的人，我们让本体接收到消息时左转移动并克隆自己，让克隆体启动时向右移动。这样克隆体也继承了这个消息入口，再次发送这个消息时，刚才的克隆体也会分裂出下一代的克隆体。



舞台脚本



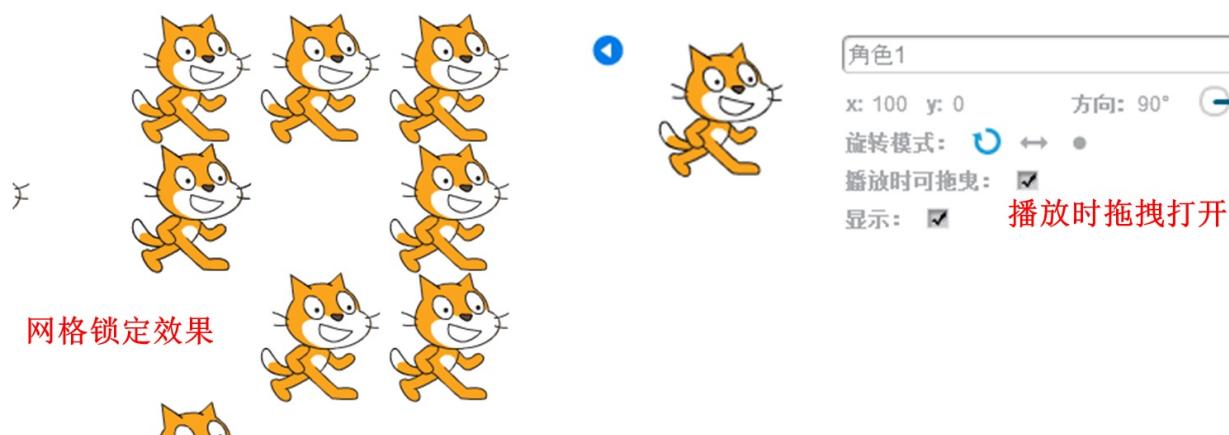
角色脚本

本体左转

二叉树克隆

网格锁定

如果我们想让克隆体在播放时可以被拖动，但是放开鼠标按键的时候我们想让角色固定在我们约定的网格上。例如我们想做一个拼图，希望拖拽放手时拼图块可以自动对齐。这时候我们就要用坐标取整的方法了。首先打开角色的播放时可以被拖动开关，然后在克隆体身上加上取整移动。



坐标取整

克隆体整理

有时候我们想用克隆体做一个整齐的砖墙，这种位置有规律的克隆体应该怎么做呢？

基本的思路是：

1. 生成位置变量
2. 重复执行克隆角色
3. 更新生成位置变量

这里面有许多讲究和细微差别，首先如果我们用重复执行来克隆多个角色，重复执行本身是有0.33秒的固定刷新时间的，如果我们要克隆一百块的砖墙，就会耗费三秒钟。如果我们想快速的刷新就一定要使用自定义积木把重复执行克隆角色的动作包进去，并且运行时不刷新屏幕。这时候问题就来了，例如我们要克隆一排角色，每隔10步摆放一个。如下图所示的积木



自定义积木重复执行 克隆体外更新位置变量

这样运行的结果是，克隆出来的十个角色都叠在了一起，并没有按我们预想的排成10步为间隔的一排。这究竟是什么原因呢？其实scratch执行一次克隆角色到作为克隆体积木启动之间是有一个比较大的时间差的，我们可以用如下的积木去测量这个时间差。



作为克隆体启动入口执行时间

可以观察到克隆动作到作为克隆体启动之间大概平均会耗费0.003秒左右，大概是重复执行刷新屏幕时间的十分之一左右，而且这个时间差并不准确，每次执行会变动。但是无论如何，这个时间和计算机全速运行的时间相比是很长的。换句话说我们运行时不刷新屏幕的全速运行重复执行克隆动作，每一个克隆动作开始时刻的差别很小，几乎可以认为是同时开始，更新变量posi的值瞬间已经被算好了结果是100。耗费0.003秒左右所有克隆体的作为克隆体被启动的入口才慢吞吞的开始执行，他们能读取到的变量posi值已经等于100了。所以所有克隆体都把自己移动到了100的位置。我们该如何解决这个问题呢？很简单，更新变量放在作为克隆体启动入口下。因为所有的克隆体本身就是一个顺序在，虽然每次的克隆动作启动几乎是同时的，但是还是有顺序差别。让他们每次作为克隆体启动时更新位置变量，就可以让角色按我们预想的顺序间隔排列了。



自定义积木重复执行 克隆体内更新位置变量

上面讨论的是克隆一行角色，是一维情况。如果我们想克隆一个二维的砖墙，应该怎么做呢？如果不考虑运行速度，为了代码的简约，我们会采用如下方法来实现：

舞台上的脚本

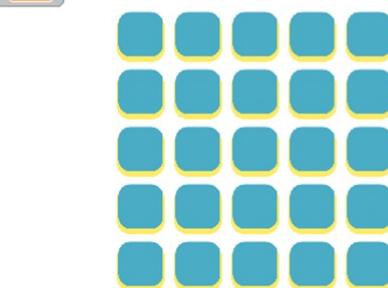


请注意等待0秒的积木
不加此积木会出现什么情况?
请大家自己尝试

双变量 克隆体外更新

如果我们想追求运行速度，用自定义积木不刷新的方法来实现，就一定要用克隆体内更新变量的方法来实现。又因为我们有两个变量要更新，这里我们就要加入一个“算式”来计算。

角色身上脚本

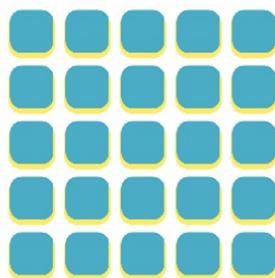


运行结果

舞台上的脚本



i 25



运行结果

角色身上的脚本



双变量 克隆体内更新 算式

可以看到算式十分的麻烦而且每次更改十分不便。这时候我们就需要进一步抽象，使用列表先生生成位置，所有的角色更新变量只对应了列表的编号。这样只需要用一个变量就解决了所有的问题，而且在我们改动了生成的规则时，不需要再更改每一个角色身上的“算式”。这样比较容易修改。

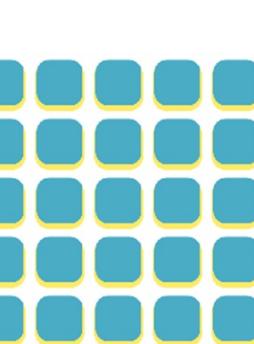
舞台脚本



运行不更新快速更新列表

运行不更新快速克隆

角色脚本



运行结果

单变量 克隆体内更新 列表编号更新

克隆体克隆别的克隆体的队列设置

如果我们要让一个克隆体克隆另一个克隆体，例如被克隆出来的敌机，要克隆发射子弹。最直接的思路是敌机更新一个xy的位置变量，接着马上克隆子弹。子弹作为克隆体启动时就移动到xy这个位置。但是这里面存在着并行的竞争问题，如果两个克隆敌机在很短时间内几乎同时发射了子弹。A敌机先更新xy变量，并克隆子弹，子弹作为克隆体启动的时间内，xy被B敌机更新。本该从A射出的子弹结果却从B射出。解决这样的竞争我们需要用到一个排队的思想。敌机每次发射时先把xy这个发射位置写入一个数组，舞台上的脚本一旦发现数组之中有元素就按顺序克隆一个子弹，之后把这一条发射位置删掉。这样提出发射要求和处理发射要求就被分开了。就好像我们去买东西，要买东西的人（要发射子弹的敌机）都要排队，而售货员（舞台）负责卖给你东西（克隆子弹）。这样就不会出现竞争的问题。



队列克隆子弹发射

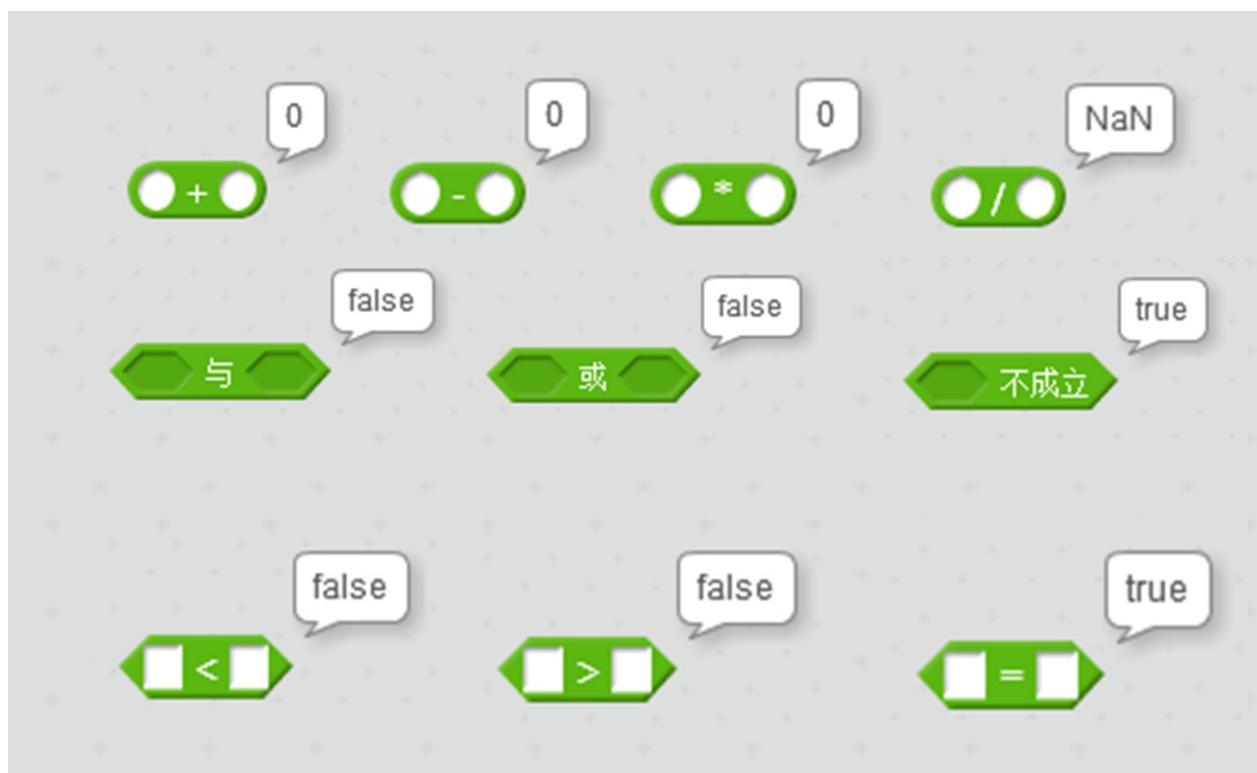
需要注意的是，克隆体克隆别的克隆体时，要尽量谨慎的使用运行不刷新的自定义积木。否则会导致scratch运行卡顿。

4. Scratch的谜之容错

1. 不写入数字会如何
2. 写入小数的列表提取
3. 造型编号0的问题 颜色和角度的小于零
4. 积木空缺填补参数 面向角色的积木块可以加入变量
5. 利用容错能做什么

不写入数字会如何

在许多计算机语言中，都存在默认值问题。也就是当我们定义了一个变量，又没有给变量赋值。这时候变量值是多少呢？为了防止发生错误，我们就要给一个默认的数值。scratch之中，数值变量的默认值是0，布尔变量的默认值是false。在运算积木盒子里面，所有的空格都是默认值。例如加法积木，我们如果不填入内容就默认是0，与或非积木不填入的空格也默认是填入了false。也就是说什么都不填的与或双击运行结果是false，不成立积木双击运行结果是true（因为默认填入的是false）。



直接运行积木结果

除法积木和余数积木因为分母不能为零，所以什么都不写入直接双击运行结果是NaN代表了没有结果返回。比较复杂的是大小判断积木。大于小于积木在什么都不填入的情况下双击运行结果是false，而等于判断积木在什么都不填入时双击结果是true。

写入小数的列表提取

我们可以在取出列表某一项元素时使用小数，例如取得列表的1.345项，这时候scratch会把1.345向下取整为1，然后取出第一项。如果我们试着访问列表的第0项或者负数项，又或者是访问超过列表项目数的项，返回的结果都是空白。

造型编号0的问题 颜色和角度的小于零

在scratch之中，有一些可以循环访问的东西。包括：

1. 角色的造型编号
2. HSV的颜色数值
3. 角色方向

例如角色造型编号如果写入0，则是切换到角色最后一个造型，以此类推的-1是倒数第二个造型。而颜色超过了200则相当于又从零开始，小于零相当于从200倒数。方向也是同样。这些可以避免许多使用时的错误。

积木空缺填补参数 面向角色的积木块可以加入变量

在将造型切换为积木中，造型不止可以通过下来菜单来选择，还可以放一个变量来控制。这个变量可以是数字也可以是字符串。如果是数字的话，对应的就是造型的编号，如果是字符串对应的就是造型名称。同时有数字和字符串时，造型编号会优先于造型名称。

利用容错能做什么

如果我们要分辨一个变量是数字还是字符串，我们就可以用变量乘以1，观察变量是否等于变量本身。scratch之中字符串乘以1的容错结果是0，而数字乘以1会得到数字本身。这样我们就可以简单的判断变量是否是数字了。

5. 拖慢脚本的罪魁祸首

在scratch之中，我们要尽量避免三种用法：

1. 特效加循环
2. 外形复杂多角色移动
3. 克隆大量角色

原因是这些动作会耗费大量的计算机资源，如果我们还同时使用自定义积木的运行时不刷新屏幕选项，scratch会直接卡死崩溃，请尽量避免。

1. 直线运动的两种方式 坐标描述和面向描述
2. 偏心旋转和步进旋转
3. 高级的步进旋转讲解和三角函数计算公式
4. 随机运动的实现
5. 撞墙了怎么办？回头和回头前冲
6. 如何把随机运动物体限定在画面中间 幽灵追随
7. 永远撞不到的物体 图章轨迹刷新

1. 运动的两种方式 坐标描述和面向描述

scratch的运动可以通过两种方式来控制：一个是直接告诉scratch，角色要移动到哪个坐标、哪个其他角色（的坐标）。我们称之为“上帝式移动”。另一个是在角色身上告诉角色重复执行往前移动，然后改变角色的方向。我们称之为“驾驶式移动”。这两类运动方式使用的时候当然要按着场景来区别。顾名思义当我们想让角色直接移动到某位置时，就可以使用上帝式移动。而在代入游戏角色时，就要使用驾驶式移动了。

2. 偏心旋转和步进旋转

scratch的旋转运动可以用两种方法实现：一种就是设置角色的中心在图形的外侧或者外部，这时候我们让角色重复执行左转或者右转时，角色因为中心在外部从而看上去像角色在绕着某中心做圆周运动。另一种是让角色重复执行向前移动n步向左或向右转m角度。这种方式就好比祖冲之的割圆术将圆割成了多边形。因为是一步一步的旋转，我们将其称为步进旋转。

3. 高级的步进旋转讲解和三角函数计算公式

为了实现角色按着约定半径进行圆周运动，我们需要知道步进n和每步旋转角m还有圆周半径r之间的关系。初中几何告诉我们， $r=n/2\sin(m/2)$ 。在这里我们可以用一个自定义积木块来实现这个公式的计算，这里不用设置刷新屏幕也可以，因为这个自定义积木并没有包含重复执行等刷新屏幕的语句。这里我们实现了输入变量n和r来计算每步旋转角度，这样做的原因是因为我们常常需要控制步进来让运动看起来更加平滑，而角度不够直观，我们也很少去控制。这个自定义积木还可以有很多应用，例如实现蛇形运动等，留给读者自己去实现。

4. 随机运动的实现

在计算机中我们应用的最多的就是随机，而在随机之上叠加的各种规则，可以让程序看起来更加的自然，甚至更加智能。前面几种基础运动，直线和曲线运动，我们可以给他们加上：

1随机的步进

2随机次数的步进（通过随机数和重复执行n次来实现）

3随机的角度

4随机的掉头，也即是随机乘以-1

5随机的游走，在一个变量x上不断叠加+1或者-1（不限制在+1，什么数字都可以）。这样做可以实现一种游动的效果，在随机过程专业课之中称之为随机游走。

这五种基本的套路可以互相组合，形成很复杂的随机运动效果。

5. 撞墙了怎么办？回头和回头前冲

如果我们让一个角色随机的在舞台上乱晃，就像没头苍蝇一样。那么就有一个问题，角色撞墙了会发生什么？前面的内容已经说过，scratch是不允许任何角色的图案脱离舞台的。如果一个角色在随机的乱晃，撞墙了他也绝对不会脱离舞台，而且很可能会在舞台边上晃半天也不会回到舞台中心。这种情况很不美观，也不方便我们制作作品。我们要使用一个判断来检测角色是否撞到了舞台边上，然后让角色回头，也就是重复执行判断角色碰到边界就反弹。但是前面疑难杂症部分也描述过，这个积木块是给初学者用的，存在很多不确定问题。最好使用侦测积木侦测到碰到边缘就旋转180度来的妥当，这里要注意的是，在旋转之后，角色仍旧有可能还碰到边缘，重复执行的检测又一次执行了180度翻转。看上去角色就在边界转来转去抖个不停。这也是常见的使用错误。正确的做法是，在旋转180度积木后跟一句向前移动。这样在角色转身之后有一个前冲的效果，不至于再次碰到边缘。问题又来了，如果我们使用如下的积木来实现一个随机运动撞墙回头的角色，为什么回头前冲之后，角色还是会经常在墙边不肯回到中心呢？

（这里用一个随机转向的例子）

我们来脑内设想一下这个过程，角色在一次循环之后有二分之一的概率会面向墙壁运动，我们应该让角色的转向随机的慢一些，这样看上去不会像没头苍蝇一样乱撞。这里我们通过一个随机10-20次的重复执行计数循环来实现控制角色移动方向，在碰到边界回头时，改变角色的方向变量减去180，使用增量来控制角色。可以观察到角色更像一个智慧生物在运动了。需要注意的是这种用增量来控制角色的思想，就是随机游走的思想，可以让角色的运动看上去不突兀。

（更改代码）

6. 如何把随机运动物体限定在画面中间 幽灵追随

如果我们仍旧强迫症不可解，就是要一个速度很均匀的运动在舞台中央不会碰到墙壁的角色。该如何实现呢？这里提供一种方案供大家思考：我们用两个角色，一个十分快速的随机移动，我们把它隐藏起来，起名叫做“幽灵”。另一个主角色，我们让他重复执行匀速运动，且每一次重复执行的时候都面向幽灵。因为幽灵速度极快的在舞台上飞动乱撞，而主角只是方向在变化，而速度恒定。且因为是面向飞动的幽灵，方向变动也不是特别突兀。这样就实现了十分逼真的类生物移动效果。主角会在舞台中央匀速但是偶尔转向的移动。需要注意的是：幽灵本身因为隐藏起来了，他还可以通过碰撞边缘的侦测模块识别是否撞墙。在scratch之中，是否碰到边缘和是否碰到鼠标指针是两个特殊的判断，即时我们把角色隐藏起来，仍然可以使用。如果在后续版本的scratch之中这个功能产生了改变，还请读者自己设立一个变量来实时观察这个状态是否仍旧如此对于隐藏不敏感。在文本语言之中，我们所熟知的是面向过程和面向对象两种编程模式。但是在scratch之中，我们其实是面向角色来编程更为方便。在幽灵追随这个例子之中，我们使用了一个幽灵跟随就省去了不少编写复杂随机逻辑的工作。后续也会有很多例子使用幽灵这一概念来帮助我们完成一些写在一个角色身上会很麻烦的事情。

7. 永远撞不到的物体 图章轨迹刷新

如果我们想做一个永远无法被碰撞到的角色，该如何实现呢？这里提供一个解决思路，我们将角色隐藏，在角色运动的重复执行之中开始将舞台画笔清空，角色移动之后，使用图章积木将角色画在画笔上。这样，别的角色再也不会误碰到该角色，而该角色也没有被隐藏消失，我们仍旧可以看到他的活动。大家一定会问这样做的意义在哪里。原因是有时候我们为了某些互相产生复杂逻辑条件的碰撞（例如碰到角色且碰到红色且碰到别的角色不成立），这时候让一个角色不可触碰，逻辑条件会简约不少，不失为一个优雅的好方法。

1. 色轮和色环
2. 旋转和画笔结合
3. 斐波那契曲线
4. 彩虹旋涡
5. 闪电和雷暴
6. 画笔贪食蛇
7. 流星和星轨
8. 残影和运动轨迹绘制 抛射游戏

1. 膨胀效果
2. 颜色等闪动效果
3. 序列帧动画制作和导入
4. 虚像隐藏的意义
5. 旋转和传送门效果

1. 方便记录 变量计分 第一次应用
2. 方便调整 变量规定 步进 上线 血量等
3. 方便控制 变量控制角色属性
4. 方便逻辑 计数变量遍历列表
5. 方便存储 存储数据信息

变量作为几乎所有编程语言的最基础概念，是我们进入程序世界的时候最先要了解的。一般我们把变量解释为一个写了变量名的盒子，盒子里面装的东西称之为数据，数据是可以被替换的，但是盒子还是盒子，只能装一个东西。例如装一个数字，装一个单词（字符串）等等。在真实的计算机世界中，变量（寄存器变量除外）其实对应了内存之中的一个地址。内存就好比一个巨大无比的仓库。我们一旦命名了一个叫做a的变量，计算机就依照某种规则给我们的a分配了一个格子，然后把地址和变量名挂钩对应起来。当然地址是一串数字，我们人类不会喜欢看也记不住，我们就只要记住a就行了。当我们把a删除掉，计算机就把分配的格子和a的对应关系忘记掉，以便下次继续分配给别的变量使用。变量究竟有哪些作用，为什么几乎所有的编程语言都会有这个东西呢？

1. 方便记录

最直观的感觉是变量可以帮我们记录事情，例如记录游戏的得分，记录玩家的血量，记录敌人的数目，记录各种事情。说白了就是让电脑这个好脑子帮我们记住事。

2. 方便调整

我们可以用很多变量来实现各种复杂的功能，例如加分减分，规定血量上限，规定速度上限等等。这些事情往往是几个变量相互作用，一些用来记录，另一些变量则负责做出规定。在一些计算机语言之中，负责做出规定的这些不变的值，也被称之为“常量”，也即是恒常的量。这时候我们一定很疑惑，数学之中的方程式我们很熟悉， $3x+5=y$ ，x和y是变量，5是常数量，为什么计算机里面5这种常量还要放在变量里面呢？这里要注意：计算机和数学的概念不同，计算机所有的量本质上讲都是变量，即使是5这样的常数，即使是1000这种血量上限，即使是50这种速度上限。我们难保不会下次心血来潮给他改一个数值。所以计算机索性把所有的量都弄成了变量。请大家不要和数学的说法混淆。

3. 方便控制

我们在所有的游戏之中都会对人物做出很多属性的设定，例如血量多少，跑多快，攻击力如何等等。这时候我们经常使用一组变量来实现对人物的控制。这样的规定还有一种类似模板的功效，我们再创建别的人物也可以用这一套变量来描述。这样在设计时候会清晰不少。在

文本语言之中，我们常常听到一个说法是面向对象。究竟什么是面向对象呢？其实和刚说到的模板很类似，例如我们要给游戏设计100个怪物，他们的血量速度和攻击力都不一样。但是他们都有一些共性，因为我们用了同一套变量同一个模板来描述他们。这套模板就被称之为类（class）而每一个怪物拥有独一无二的变量数值，他们就被称之为类的一个实例。这就是面向对象的基础解释。

4. 方便逻辑

在后面的控制流中，我们常常使用变量来控制循环的逻辑。具体用法很简单，就好比用一个变量来帮我们计数，每次循环结束它就被加了1，而它从0开始就恰好是循环的一个编号。这种用法可以方便我们把数组的每一个元素按顺序拿出来。我们称这种方法为“遍历”。也许我们会觉得这种遍历不是理所应当的，为什么还要单独的说呢？那么我们可以试验几个好玩的事情，例如把列表的所有偶数项拿出来求和，把列表的素数项求和。这时候我们就发现，一个技术的变量是多么的方便。

5. 方便存储

在计算机科学之中，我们常常把数据是如何存储的称之为“数据结构”。变量就是最简单的一个数据结构，列表是更加方便的一个数据结构，他把变量编号放在了一起，我们可以更加方便有序的使用这些变量。当然还有更加高级的一些数据结构，他们都有特定的用途。这些将在后文有所涉及。

1. 布尔值和布尔
2. 布尔表达式和布尔运算（逻辑运算）
3. 真值表
4. 逻辑运算和条件分支嵌套的互相转换

1. 列表和字典的区别
2. 列表存语言 聊天机器
3. 列表存随机 不重复随机列表
4. 列表存方向 贪食蛇 和重复蛇
5. 列表和字符串的异同

1. 循环100000次
2. 多重循环的意义 扫描屏幕
3. 数独游戏
4. 循环的展开 水仙花数
5. 循环fix
6. 循环监听
7. 循环刷新序列帧

1. 循环的好朋友 函数 为了复用而抽象
2. 函数的参数 意义何在？更好的复用
3. 函数的刷新屏幕
4. 函数的嵌套
5. 拒绝长脚本 提倡并列式
6. 造好的轮子要不要重新造 Scratch的角色复用

1. 递归三要素 递归方式 停止条件 逻辑不重叠 德罗斯特效应 谷歌递归
2. 递归和顺序的区别 阶乘 最大公约数 首先假设问题已经被解决
3. 分型和递归 科赫雪花 H树 随机树 蕨类植物 龙形曲线 C形曲线
4. 汉诺塔和二进制三进制
5. 斐波那契和兔子
6. 树遍历

1. 排序的八种姿势
2. 排序算法的复杂度
3. 排序算法的使用场景
4. 查找的七种武器
5. 查找长度和应用场景

1. 良好秩序从排队抓起 队列的意义
2. 如果有有限级更高的特权怎么办？优先队列和堆
3. 网络资源分配和CPU资源分配

1. 为什么要后入先出？从摊开的书本开始
2. 函数递归的堆栈表示
3. 递归 树和堆栈
4. 屏幕跳转
5. 带优先级的计算器

1. 二维数组的本质是一维数组
2. 舞台空间也可以是一种存储？存储只是一种思想
3. 多维数组怎么办？
4. 二维数组和平面的连续地形绘制

1. 真实世界的偏差 线性回归
2. 回归核心和预测

1. 竞争的本质是时序规则
2. 敌人的竞争 你更新我刷新
3. 自己和自己的竞争 按键瞬间多次激发
4. 时序图 上升沿和下降沿控制 正锁反锁
5. 蓄力和打断
6. 格斗游戏的硬直和解除
7. 连招系统和时序控制

1. 计算机的原子二值电路和开关 为什么不搞三值电路
2. 三极管推拉制造逻辑门电路
3. 逻辑门电路到锁存器 如何存下01000101010
4. 锁存器到加法器 实现基本计算
5. 计算机结构和CPU指令集
6. 驱动和操作系统
7. 软件和通讯
8. 互联网和云计算区块链

1. 正负数 虚实数 有理数
2. 素数和加密
3. 梅森素数和哥德巴赫
4. 数列
5. 余数和同余 奇偶校验
6. 函数
7. 映射和坐标变换

1. 指数对数
2. 复利和稳定复利
3. 树结构的必要性 算法的极限为什么是对数
4. 阶乘和箭号 大的不可思议

1. 二进制
2. 三进制
3. 八进制
4. 十进制
5. 十二进制
6. 十六进制
7. 六十进制
8. 三百六十度

1. 排列组合
2. 大数定理和中心极限定理
3. 概率和分布 为什么正态分布很多
4. 概率陷阱
5. 条件概率和贝叶斯教主
6. 马尔可夫链
7. 鞑伦和赌博如何必胜
8. 即使必胜也要有凯利公式
9. 观察者和实验者错觉 小概率黑天鹅事件

1. 物体和空间
2. 空间的基准 基向量 空间维度
3. 换一组基向量会怎样？空间还是那个空间么？
4. 矩阵是什么？矩阵就是描述一个物体在空间中的位置
5. 物体瞬移了 乘以一个变换矩阵 变换矩阵是什么？是另一组基描述了物体的位置 运动是相对的
6. 不同的基描述同一个变换矩阵究竟有什么共性？本征值
7. 空间解析几何 点线面
8. 有用的旋转变换
9. 逆矩阵解方程组和谷歌看家法宝 pagerank
10. 最小二乘法和定位算法

1. 函数的图像
2. 速度的速度的速度的速度 导数
3. 差值和逼近
4. 聚沙成塔 面积体积各种积

1. 三角函数组成世界？
2. 傅里叶祖师爷
3. 广播如何分频道
4. 模拟滤波器和数字滤波器
5. 噪声之中淹没了一架战斗机 匹配滤波
6. 稳定一下情绪 滑动滤波窗
7. 图像的降噪 二维滤波器
8. 预测未来的卡曼滤波器

1. 逻辑和真值表
2. 逻辑也是一种空间？逻辑运算
3. 卡诺图和贯通性

1. 规划问题 田忌赛马 状态图
2. 背包问题和穷举
3. 递归和规划

1. 合作还是不合作 动和静 全知还是蒙在鼓里
2. 囚徒的均衡分析 制造概率均衡点
3. 规则塑造人性
4. 公共知识和信息对称武器
5. 先手必胜的游戏是否有意义
6. 博弈实验和社会人格延续 以牙还牙

1. 三角函数和反三角函数
2. 牛顿第二定律 力的分解和加速度用Scratch实现
3. 多个力怎么办？分解之后合成

1. 刚体表面触碰效果实现
2. 杠杆和力量传递
3. 变速 齿轮
4. 旋转变直线伸缩 齿条
5. 旋转往复 凸轮 曲柄摇臂 急回机构
6. 极度减速 蜗杆
7. 间歇运动机构
8. 来了就不许走 棘轮和棘轮效应 捲纵器

1. 电荷和洛伦兹力 高能粒子回旋加速器
2. 霍尔元件是什么
3. 电子枪和扫描
4. 法拉第的发电机 电线里面发生了什么
5. 麦克斯韦干了啥 为啥最伟大
6. 电磁波和电路里面的电是一码事么？
7. 微波炉 WiFi 和高压电线哪个更危险？尺度 波长 谐振和能量交换
8. 电也是守恒的？电路原理 欧姆定律弱爆了 我们用基尔霍夫定律

1. 容纳故事的RPG 暗黑支配的世界
2. 异类RPG文字冒险 我才是鼻祖
3. 动作类游戏 人物外动作 跳跃的马里奥
4. 人物动作 格斗和拳皇
5. 解谜类 机械迷城 谜画之塔 gorogra 三位一体
6. 沙盒游戏 自由不是放纵
7. mmo社群和副本 信仰级别的EVE
8. APM和帕金森 策略类游戏和电子竞技 红色警戒 帝国时代你居然9. 是微软做的 刀塔和LOL
9. 手游和刀塔传奇 天下一家万马齐喑
10. killtime休闲的三消和智龙迷城
11. 横空出世的supercell 游戏机制的创新才是真英雄

1. 完善抽象的轮子库
2. 物理引擎的基本构造
3. 主流引擎的优劣
4. 扩展和跨平台的优势
5. 图像渲染和声音编辑
6. 动画和状态机
7. Scratch作为简单引擎
8. 实现简单重力引擎和刚体碰撞引擎

反馈 和 参照 制衡

1. 视觉特效反馈
2. 数值反馈
3. 音效反馈
4. 心理反馈 Ludo平衡 敌人让我们有意义
5. 心理参照系 赋予积累意义 在99级台阶之上开始游戏
6. 多人游戏的时间线均衡
7. 机关和出口 嵌套多层目标 被掩藏的路径
8. 游戏世界的资源产出和回收 经济体维稳

1 触发效果 按钮 物品焦点 鼠标效果 加分减分效果 出击和受击 打击感 速度感 拖尾和残影 力量感 重量 参照物的对比效果 场效果 群体反应效果 属性效果 专场 冰 火 毒 神圣 邪恶 神秘（闪动） 自然力量（生长） 机械

2 线性数值 非线性数值和时间成长 指数数值和增长锁定 战斗核心公式 乘法公式和加法公式 战斗力核算 预测一切 道具的影响控制 挖掘各种方面的数值 让玩家感受到数值的影响 慌张感 和数值的准确关联

3 选择纯净的音乐并控制音量层次 高级感 鼓点和心跳控制节奏 按钮和UI的音效 游戏长效基调音乐 引发直接感情的音效

4 成对的创造 机遇和冒险 人的心理能量是怎么消耗的 决定的次数和感觉加权求和 均衡的嵌套和面向均衡设计游戏 设计时关注心理的敌人而不是游戏中的敌人 玩家对于决定后效的积累 学习 技能训练

5 前有糖后有虎 完成感驱动为什么优于目标感驱动 有效划分任务 随时反馈 世界广播和高等游行 展示别人的完成感 积累与消耗

6 瞬间产生压倒优势一定不好么 可见优势线性积累会让弱势者放弃游戏 局势反转的设计余地 属性道具和功能道具 必胜策略的避免 游戏平衡的关键在于可容纳的复杂度 确定性是石头 随机性是水 用随机来掩盖石头的不平

7 机关就是掩盖路径避免无聊的一种方法 先要有路径 机关的可重复性和时间属性 机关之间的嵌套和预先解锁路径设计 用视错觉来掩盖路径 神作的内核 机关和角色的交互 机关和动作战斗等元素的结合

8 什么是资源 开采权有限 有消耗 资源的流通 货币的拆解和促进交换的属性 多资源游戏的资源属性区别 资源漏斗和资源冲刷效应（指向玩家行动） 游戏经济崩溃的几种姿势 某资源一支独大 其他资源起不到辅助效果 通胀 玩家线下绕行自定体系

1. 四象限人格论
2. 生来为了攻克 真正的leader
3. 社交关系维护
4. 反社会人格引导
5. 探索搜集和程序内容丰富度

1. 程序内容生成
2. 地牢地形生成
3. 探索的惩罚和均衡
4. 过程化生成 四两拨亿斤
5. 分型和超大规模渲染
6. 捆住随机的野马 规则的金字塔层级设定和完善

1. 硬核游戏和训练机制
2. 文化硬核和玩法硬核 短期和长期
3. 心流体验的维持
4. 心流和游戏寿命

1. 随机控制位置
2. 随机控制抖动效果
3. 随机控制概率
4. 随机+列表实现概率分布
5. 随机和战斗核心龙与地下城多面骰子
6. 随机和地形构造
7. 随机和外观构造
8. 随机和DNA
9. 随机和对话生成器
10. 随机和蒙特卡洛方法

种植卡牌的克隆 单击克隆 再单击防止

1. 克隆和私有变量 我和母体不一样
2. 克隆体和别的物体相互作用 子弹和锁
3. 克隆体和克隆体之间的相互作用 消息和延迟消灭
4. 克隆体自克隆的嵌套 递归的形象展示
5. 克隆体被克隆后克隆另一个克隆体 塔防和队列
6. 克隆消灭模式和克隆隐藏模式
7. 克隆应用举例 卷轴游戏 卷轴实现 塔防游戏 弹幕游戏 消砖游戏

1. 时空幻境
2. 状态记录与列表
3. 反向刷新状态贪食蛇
4. 重力引擎的实现
5. 时间倒流的实现 各元素状态记录
6. 时间倒流的特例与谜题
7. 时间线和物理位置关联
8. 多时间线的统一展示
9. 时间倍数缩放的实现
10. 时间领域的距离判断实现

1. 粒子特效
2. 节奏爆炸特效
3. 拖尾特效
4. 弹幕的一百种姿势
5. 单尾电系特效和树状电系特效
6. 帧动画特效和分类

1. 简单设计模式 观察者模式
2. 对象池模式 需要用就显示 不需要就隐藏 需要就克隆 不要就删除
3. 享元和复用
4. 脏标识模式 需要用再更新 不需要就画在那
5. 事件队列模式
6. 良好的类抽象和继承
7. 更新方法和fix更新方法
8. 单例模式
9. 测试和逻辑覆盖
10. 结对编程和代码审查
11. 软件设计流程的改变 瀑布模型到迭代开发精益开发