

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 6实验报告**

|  |  |
| --- | --- |
| 姓名 | 沈子鸣 |
| 学号 | 1170301007 |
| 班号 | 1703010 |
| 电子邮件 | [2508754153@qq.com](mailto:2508754153@qq.com) |
| 手机号码 | 18800421860 |

**目录**

[1 实验目标概述 1](#_Toc1393927)

[2 实验环境配置 1](#_Toc1393928)

[3 实验过程 1](#_Toc1393929)

[3.1 ADT设计方案 1](#_Toc1393930)

[3.2 Monkey线程的run()的执行流程图 1](#_Toc1393931)

[3.3 至少两种“梯子选择”策略的设计与实现方案 2](#_Toc1393932)

[3.3.1 策略1 2](#_Toc1393933)

[3.3.2 策略2 2](#_Toc1393934)

[3.3.3 策略3（可选） 2](#_Toc1393935)

[3.4 “猴子生成器”MonkeyGenerator 2](#_Toc1393936)

[3.5 如何确保threadsafe？ 2](#_Toc1393937)

[3.6 系统吞吐率和公平性的度量方案 2](#_Toc1393938)

[3.7 输出方案设计 2](#_Toc1393939)

[3.8 猴子过河模拟器v1 2](#_Toc1393940)

[3.8.1 参数如何初始化 2](#_Toc1393941)

[3.8.2 使用Strategy模式为每只猴子随机选择决策策略 2](#_Toc1393942)

[3.9 猴子过河模拟器v2 2](#_Toc1393943)

[3.9.1 对比分析：固定其他参数，选择不同的决策策略 3](#_Toc1393944)

[3.9.2 对比分析：变化某个参数，固定其他参数 3](#_Toc1393945)

[3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？ 3](#_Toc1393946)

[3.9.4 压力测试结果与分析 3](#_Toc1393947)

[4 实验进度记录 3](#_Toc1393948)

[5 实验过程中遇到的困难与解决途径 3](#_Toc1393949)

[6 实验过程中收获的经验、教训、感想 4](#_Toc1393950)

[6.1 实验过程中收获的经验和教训 4](#_Toc1393951)

[6.2 针对以下方面的感受 4](#_Toc1393952)

# 实验目标概述

本次实验训练学生的并行编程的基本能力，特别是Java 多线程编程的能力。

根据一个具体需求，开发两个版本的模拟器，仔细选择保证线程安全（threadsafe）的构造策略并在代码中加以实现，通过实际数据模拟，测试程序是否是线程安全的。另外，训练学生如何在threadsafe 和性能之间寻求较优的折中，为此计算吞吐率和公平性等性能指标，并做仿真实验。

* Java 多线程编程
* 面向线程安全的ADT 设计策略选择、文档化
* 模拟仿真实验与对比分析

# 实验环境配置

本次实验环境配置与之前相同。

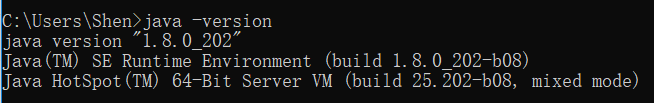
硬件环境：Intel Core i5-7200U，64位; 4G;

软件环境：Windows 10家庭中文版

IDE：Eclipse Java 2018-12

关于Java：JDK 8, JRE 8

关于Git：Git 2.20.1





GitHub仓库的URL地址：

<https://github.com/ComputerScienceHIT/Lab6-1170301007>

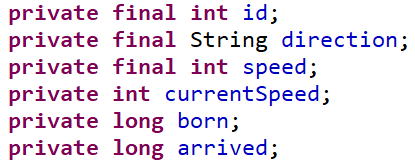
# 实验过程

## ADT设计方案

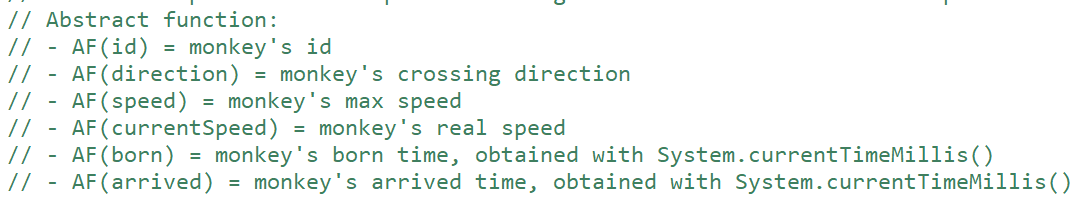
设计了哪些ADT、各自的作用、属性、方法；

### Monkey

Monkey类对象即表示为一只猴子，具有如下属性



它们的抽象函数表示如下：



Monkey类中的方法有Constructor public Monkey(int id, int speed, String direction)，以及所有属性的Observer和Mutator方法，即getter和setter，此外还有重写的equals和hashCode方法。

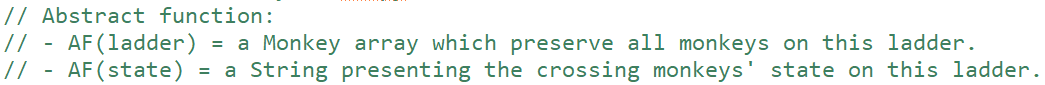
Specification: Monkey类表示一只猴子，它的唯一标识为id，它的行进方向为direction，它的（最大）速度为speed，它的真实速度为currentSpeed，它的出生时间为born，到达时间为arrived，这两个时间是由System.currentTimeMillis()方法确定的long变量。

### Ladder

Ladder对象即表示一个梯子，具有如下属性：



它们的抽象函数表示如下：

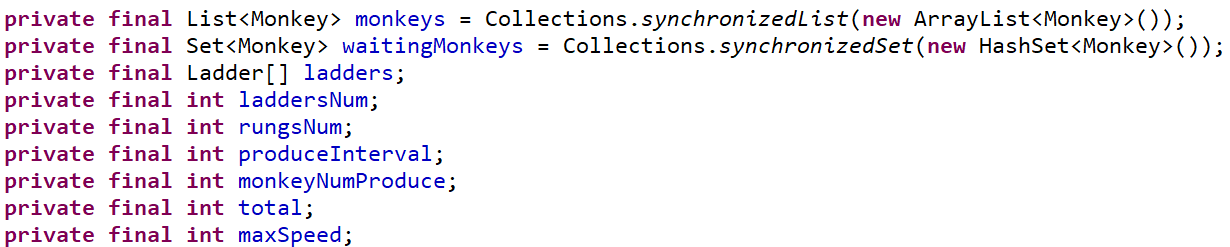


Ladder类中有的方法除了构造器public Ladder(int h)将ladder初始化，还有所有属性的observer方法和mutator方法，即getter和setter，其中，ladder的observer方法做了防御式拷贝，且其mutator方法为两个，public synchronized void setMonkey(int i, Monkey monkey)将ladder的第i个位置置为monkey，public synchronized void removeMonkey(int i)将ladder的第i个位置置为null。

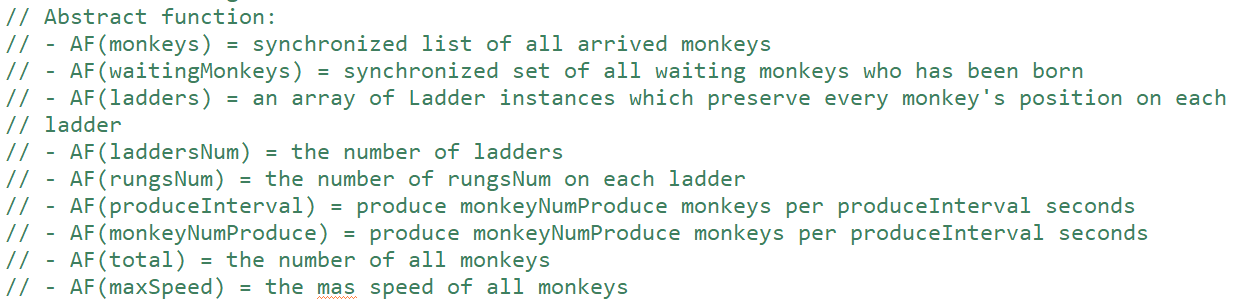
Specification: ladder是一个Monkey数组，其真实表示是从索引1到索引h（h为梯子上的横杆数，即长度）。如果索引i处有Monkey对象，则表示该处有猴子，如果i处为null，则表示这里没有猴子。并且，不管该梯子上的猴子是从左到右，还是从右到左，都是从小索引到大索引的方向移动。

### MonkeyGenerator

MonkeyGenerator对象是用来生成猴子和监视过河情况的。具有如下属性：



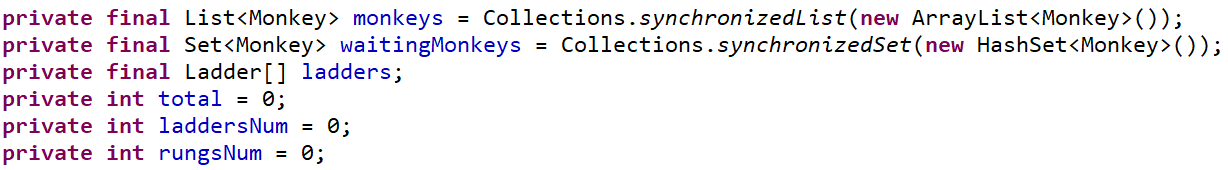
它们的抽象函数表示为：



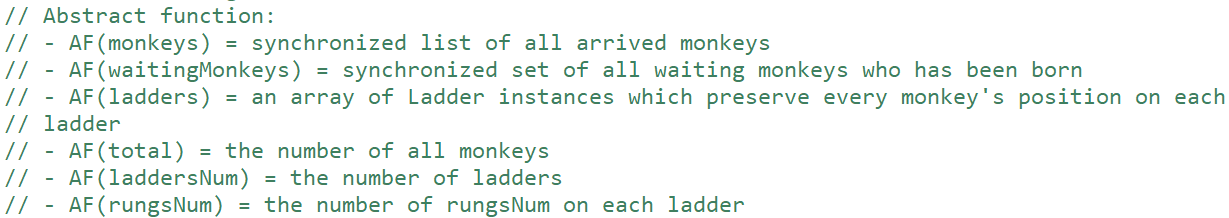
该类中只有三个方法，一个是构造器，用来初始化属性参数，一个是generate方法，根据参数来循环生成猴子，每生成一个猴子，就调用CrossExecute中的方法执行过河线程。一个是visualize方法，实时根据属性绘制梯子状态，用来监视过河过程。在主函数运行中，新建一个MonkeyGenerator对象，先调用visualize方法，再调用generate方法，即可启动猴子生成并开启过河。因为已经有了main线程和visualize中的GUI线程，所以在生成猴子开始后，只有当当前线程数再次为2时，这时猴子都已经过河完毕，generate方法才会计算吞吐率和公平性。

### Competition

Competition类和MonkeyGenerator类十分相似，是用来在v3读取文件生成猴子开启过河线程的。由于是读取文件生成猴子，只有n和h两个参数，为了判断已过河的猴子是否等于总数，还要有一个total属性，在读文件时修改total属性，使其最终为猴子总数。所以只具有如下属性：



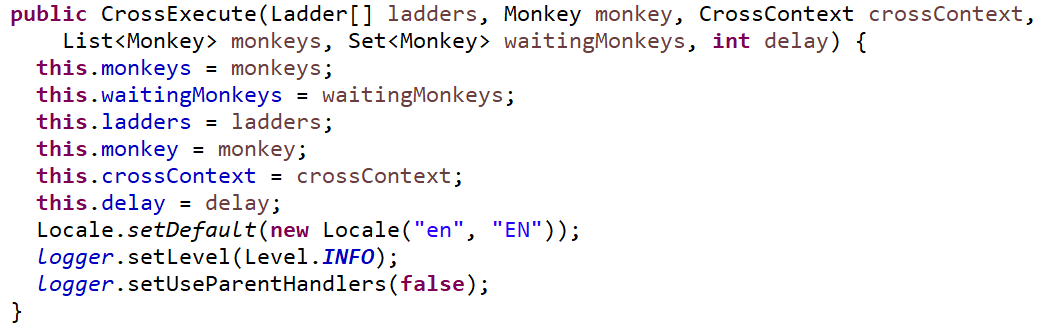
它们的抽象函数表示为



该类的构造器中读取竞赛数据文件，并为laddersNum和rungsNum赋值，之后在readFromFile方法中，重新读取文件，并根据文件生成猴子。

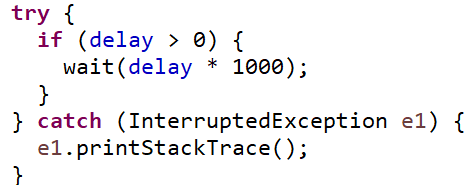
## Monkey线程的run()的执行流程图

猴子的线程类为CrossExecute。该类的构造器，通过传参来初始化共享的数据结构，如梯子数组，当前过河的猴子，等待的猴子集合，已过河的猴子列表，过河策略，线程开始延时，如下：

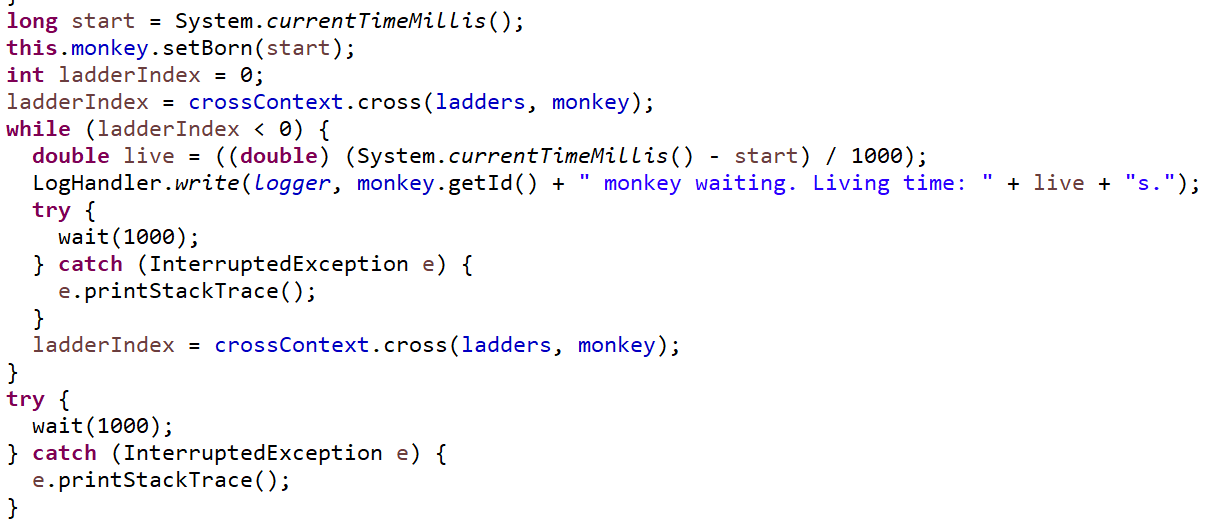


此外，为实现线程，该类继承了Thread类，并只有一个重写的run方法。

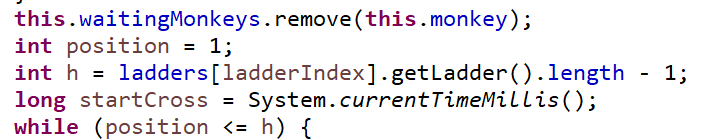
在run方法中，首先根据参数delay而延迟一定时间，这是为了在v3中读取提供的数据，而分批次执行猴子线程。如果不是读文件，那么该参数为0.



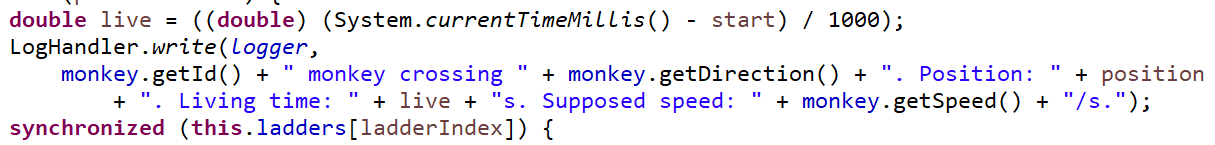
接下来开始计时，计算该猴子过河所需总时间，设置此时的时间为猴子的出生时间，并调用过和策略的cross方法，得到该猴子应该选择的梯子索引，如果返回值为-1，说明该猴子无法选择梯子，则需要等待1s后，再次选择。



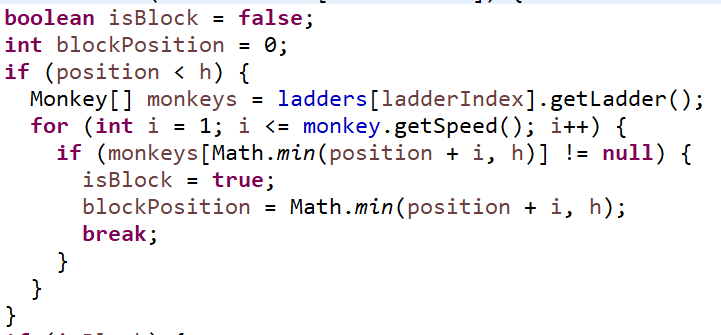
获取梯子索引后，该猴子可以开始过河，将该猴子从等待猴子集合中删除，然后初始化该猴子当前的位置为1，然后开启猴子的过河计时。进入过河循环，直到position大于h后，认为该猴子已经过河，跳出循环。



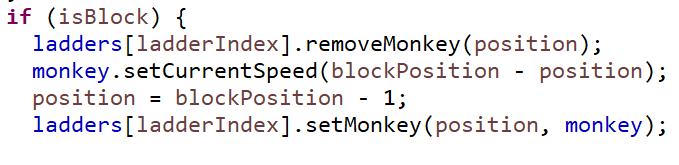
在过河循环中，每次开始就记录一次过河位置信息，然后对该猴子所在的梯子加锁，避免在该猴子决策跳跃距离时，其它线程打乱。



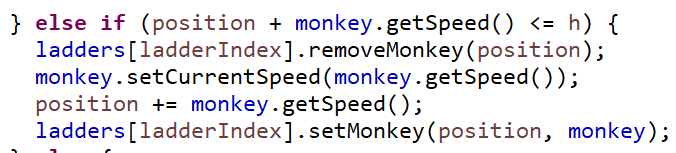
接下来，要判断该猴子在爬梯子的过程是否会被阻塞，如果阻塞的话，那么将不能按照它的最大速度前进，以该猴子当前位置为基准，向后遍历它的最大速度个横杆，判断上面是否有猴子，如果有猴子，则记录阻塞位置，并马上跳出该层循环。



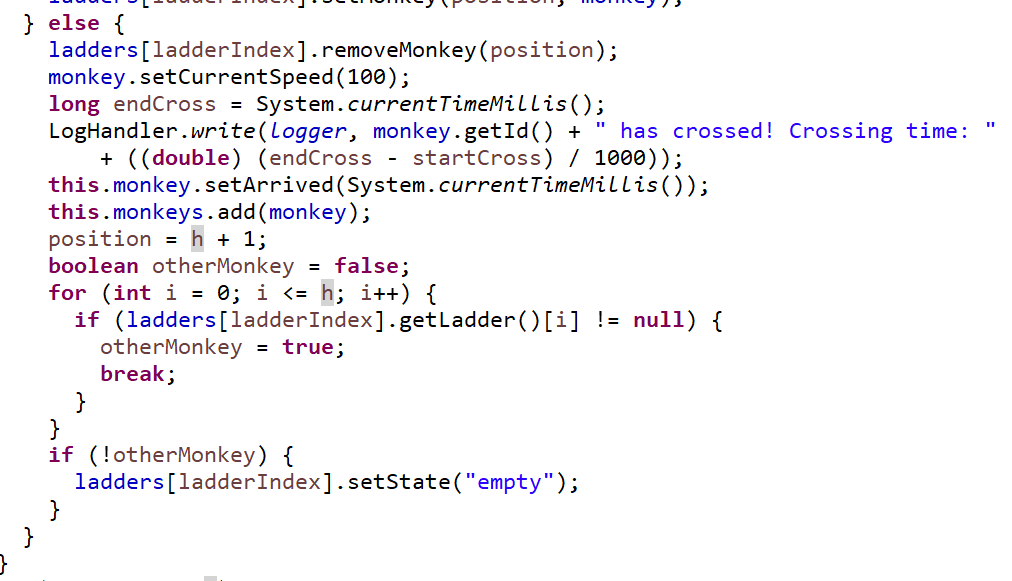
然后根据阻塞信息isBlock和blockPosition来执行跳跃操作，首先是被阻塞时，该猴子的下一步肯定不会直接过河，所以，只能跳到阻塞位置的前一个位置。



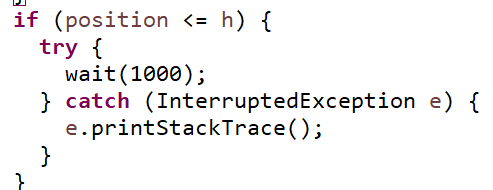
如果没有被阻塞，则要考虑是否下一步会过河，如果不会，则按照最大速度前进。



如果没有被阻塞，而下一步将会过河，则对该猴子的相关属性重新赋值，并从梯子上去掉这只猴子，结算时间和梯子状态。



最后，如果该猴子还没有过河，则执行下一次迭代之前，要间隔1s，如果已经过河了，那么就不必了。

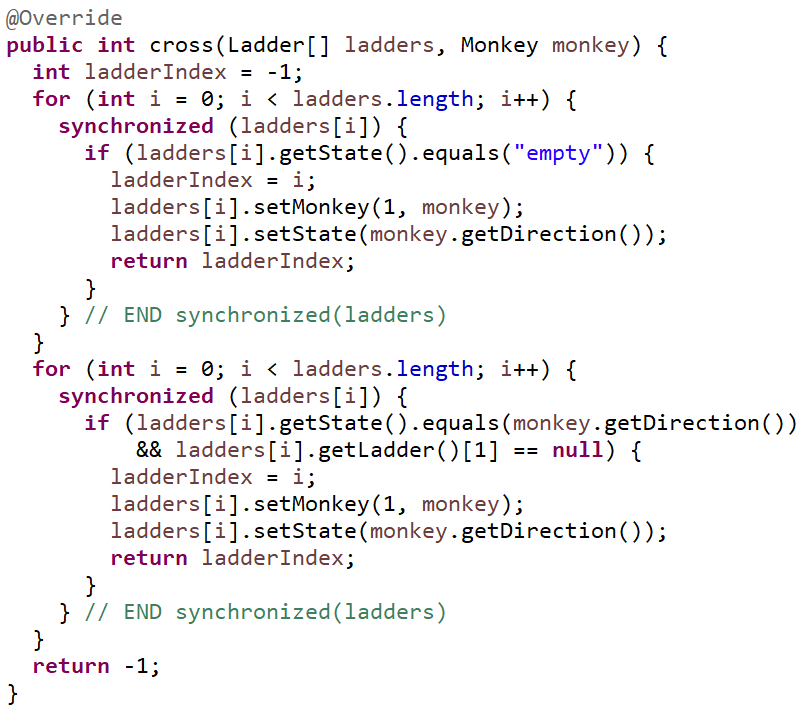


## 至少两种“梯子选择”策略的设计与实现方案

在策略实现类中的cross方法中，返回值为ladderIndex，即为符合条件的梯子索引，如果返回-1，则意味着当前该猴子没有找到梯子可以上。在以下的策略实现类中，凡是在寻找梯子的迭代中，只要有满足条件的立马返回的，都可以在迭代内部对一条梯子加锁，而不用对所有梯子加锁。而有些策略需要遍历所有梯子才能得知哪条梯子是符合条件的，这时需要在遍历循环外给整个梯子数组加锁。当找到某个满足条件的梯子并返回之前，对该梯子的方向赋给猴子的direction值，对该梯子的索引1赋给该猴子对象，意为该猴子上了该梯子的第一个横杆，并且对该梯子标识“本猴子在以该方向前进，请对面的猴子不要过来了”。

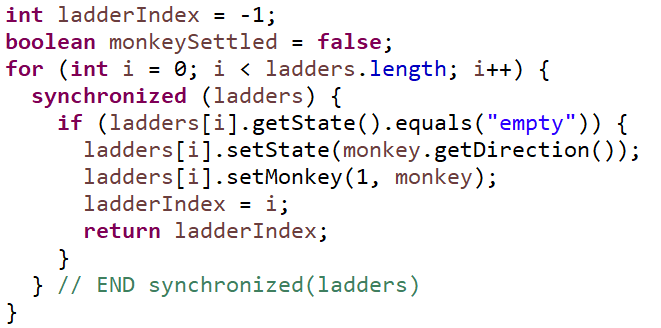
### Naïve Cross Strategy

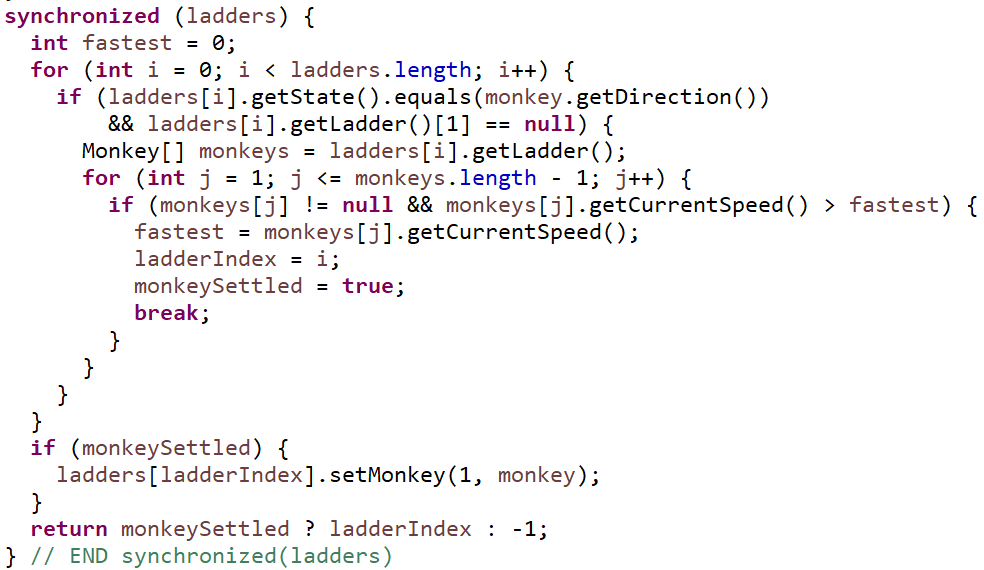
该策略优先选取空梯子，如果没有空梯子，则寻找同向的梯子，满足条件的多个梯子中不作区分。该策略即为能上就上，不能上则等，没有太复杂的安排。



### Push Fast Strategy

该策略优先选择空梯子，如果没有空梯子，则在同向的梯子中，选择距离出发点最近的猴子真实速度最大的那个梯子。该策略可以尽可能的让当前这只猴子不会被阻挡，且如果该猴子速度为1，那么它的后面跟的猴子也不会很多。

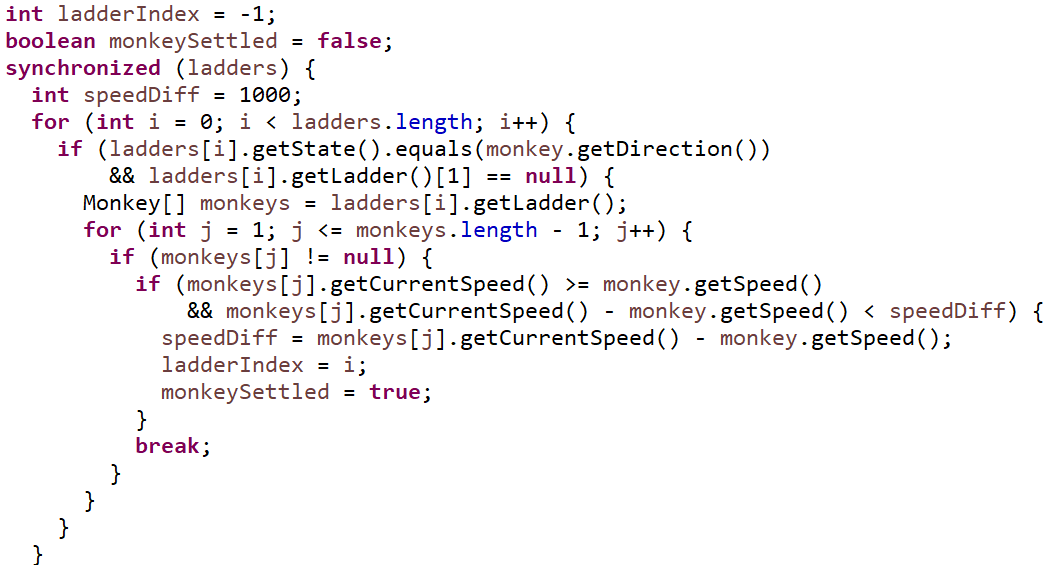


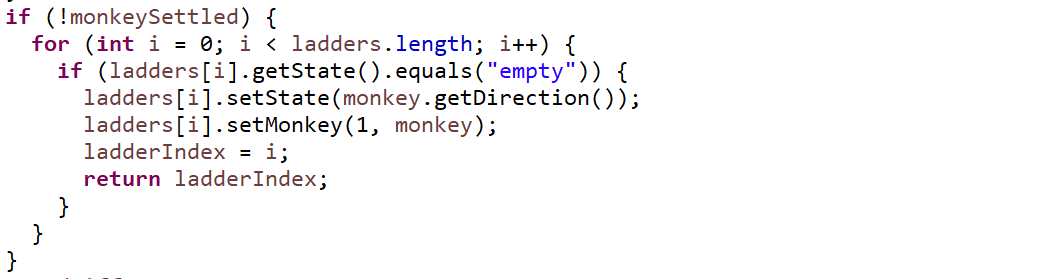


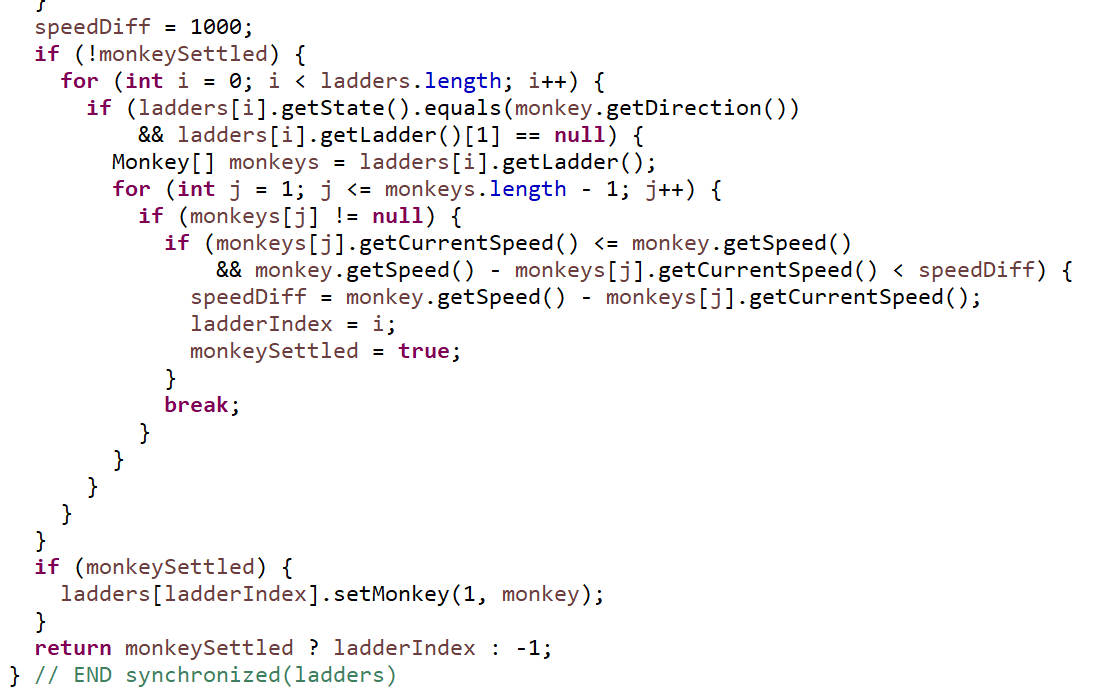
### Speed Distribution Strategy

该策略优先在同向的梯子中，且最近的那只猴子的真实速度不小于当前猴子的最大速度，选择这个速度差值最小的。如果没有这样的梯子，那么选择空梯子。如果空梯子也没有，则说明所有同向梯子上都有猴子，且所有最近的那只猴子的真实速度都要比当前这只猴子的最大速度小，那么也找一个速度差值最小的上。

该策略在实际中，对于三个竞赛文件，是表现最好的。理论上，可以使每个梯子上，从远到近的猴子的最大速度不严格递减，也就是说，大多数情况下不会存在阻塞。只有当前生成的猴子速度太大，才会发生阻塞，而且只有这一只猴子阻塞，对后面的都不影响。如果取消最后一个选择，令该猴子在不满足前两条时就等待，那么就会严格地不存在阻塞，但是，该猴子等待过程中带来了时间开销，结果并不理想。





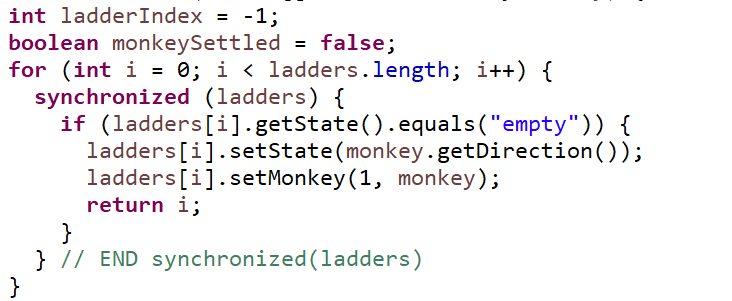


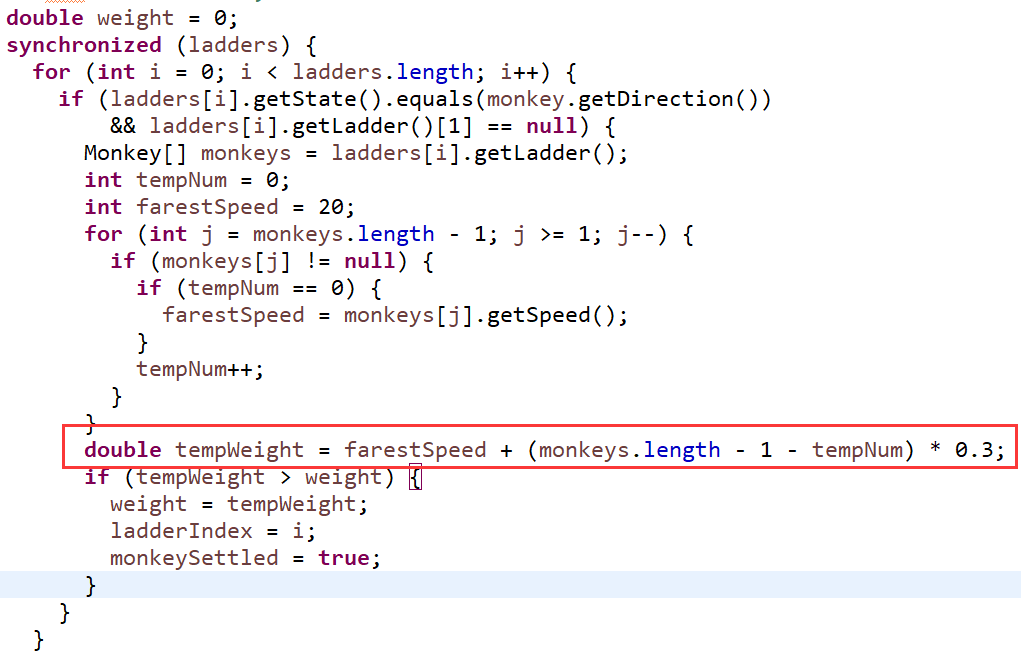
### Average Distribution Strategy(with weight)

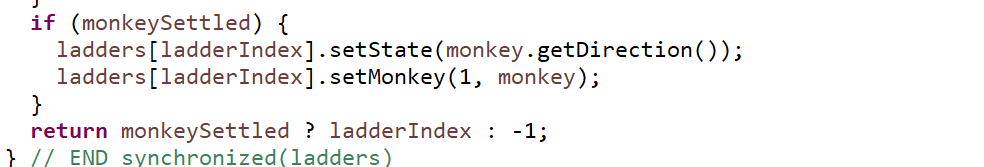
该策略优先选择空梯子，如果没有空梯子，则在同向的梯子中选择猴子数目最小的。到这里，是平均策略的思想，可以对其做少许优化，加上权重。

带权重的平均策略优先选择空梯子，如果没有空梯子，则在同向的梯子中，选取最远猴子的速度和该梯子上的猴子数目指标的加权和最大的。猴子数目指标为横杆数减去该梯子上的猴子数，也就是说，该梯子上的猴子数越小，就越倾向于选该梯子，该梯子上最远猴子的速度越大，也越倾向于选该梯子，经过实验，认为将权值定为0.2或0.3是比较好的选择。

之所以选择平均策略，是因为多次实验GUI观测发现，对于一个速度比较快的猴子，不管它是跟在一个速度为1的猴子后面，还是自己走一条空梯子，对结果来说是没有区别的。但是，如果一个速度为1的猴子后面跟了太多的不同速度的猴子，那么就有可能对结果造成额外的代价。所以尽可能地充分利用梯子的空闲位置，使梯子的利用率最大，同时也尽可能避免效率低下的阻塞发生。



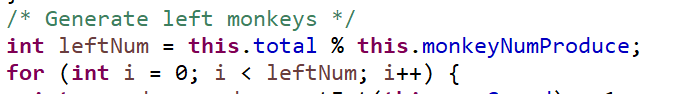




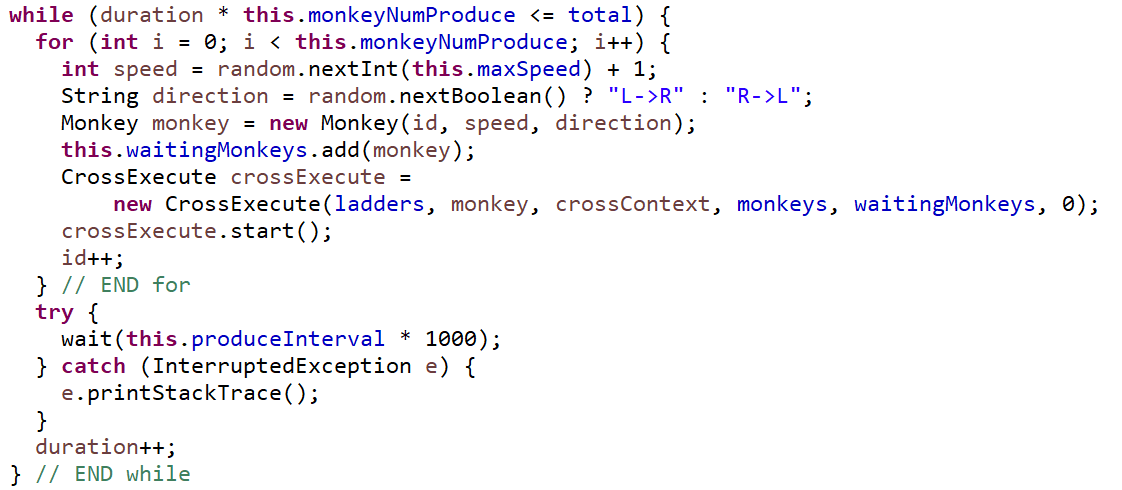
## “猴子生成器”MonkeyGenerator

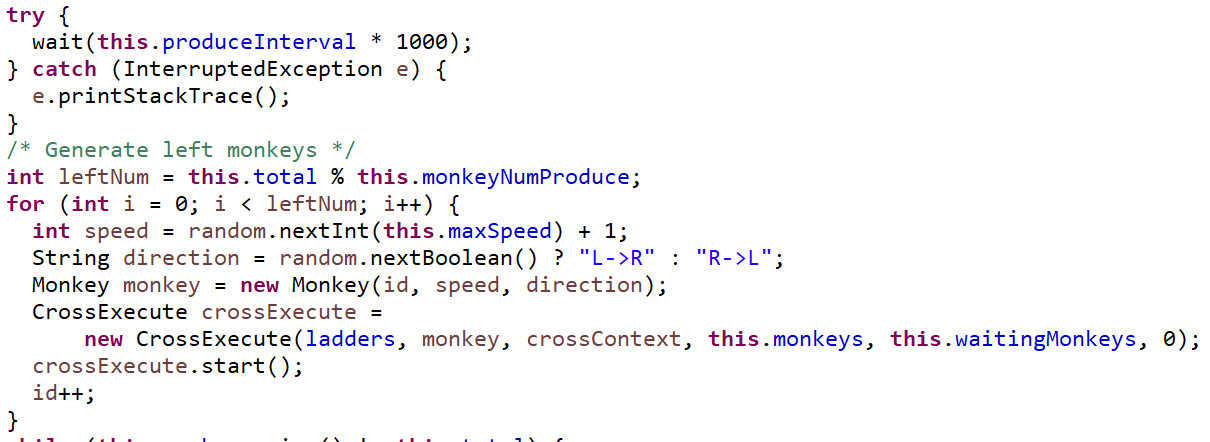
猴子生成器是每隔produceInterval秒，就生成monkeyNumProduce只猴子，直到将total只猴子生成完，如果最后一次生成不足monkeyNumProduce只猴子，则要生成total%monkeyNumProduce只猴子。所以有两段循环为：



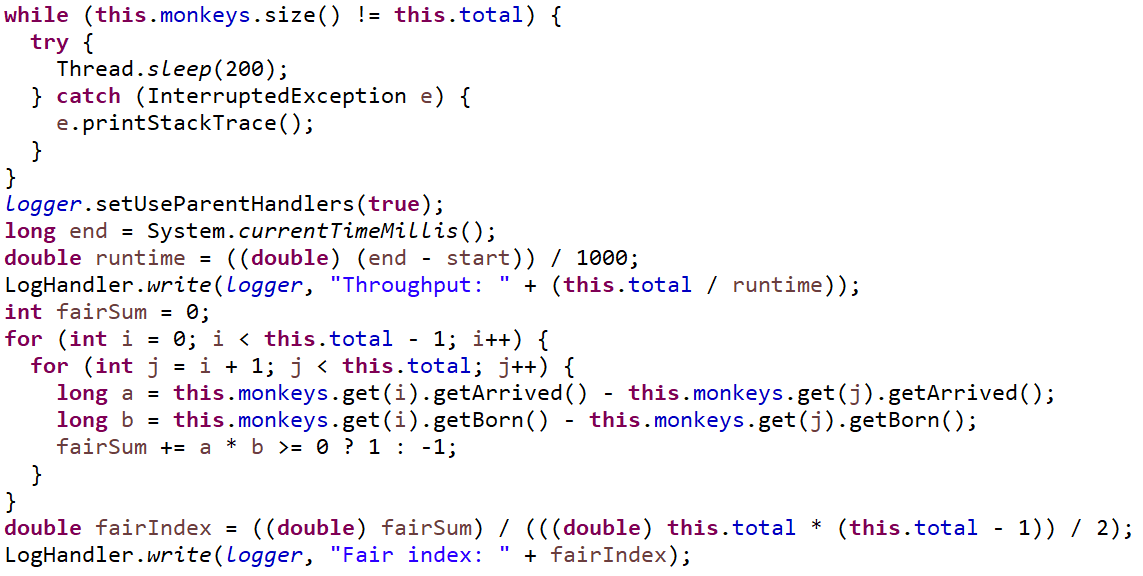


这两段循环中生成猴子的方式是一样的。首先随机生成maxSpeed和direction参数，用这些参数来构造一个猴子对象，然后将该猴子保存到猴子列表中，也将其加入到等待猴子集合中，再构造一个线程类，开启猴子过河线程即可。





然后，猴子生成完毕，线程全部开启，此时要等待猴子过河结束后计算吞吐率和公平性，要有一个暂停的设计，直到猴子过河结束再继续运行。可以认为当线程数为2时（主线程和GUI线程），猴子线程全部结束，过河完成。

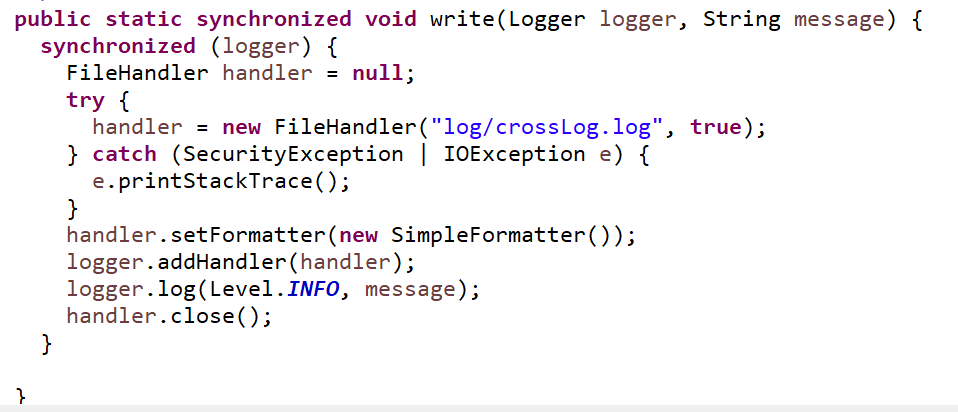


## 如何确保threadsafe？

在猴子过河的过程中，有些数据结构是共享的。比如说梯子数组ladders，等待猴子集合waitingMonkeys，日志logger，这些是共享的且在各个线程内部会被访问和改变的，为了防止在一只猴子访问和修改这些数据的过程中，被另一只猴子也访问和修改，可以采用加锁的策略，确保线程安全。

线程安全一方面可以在ADT中，将observer和mutator方法声明为synchronized，这样该方法在调用时就是原子的，不会被抢占。还有一些数据结构可以声明为final，避免它被修改。

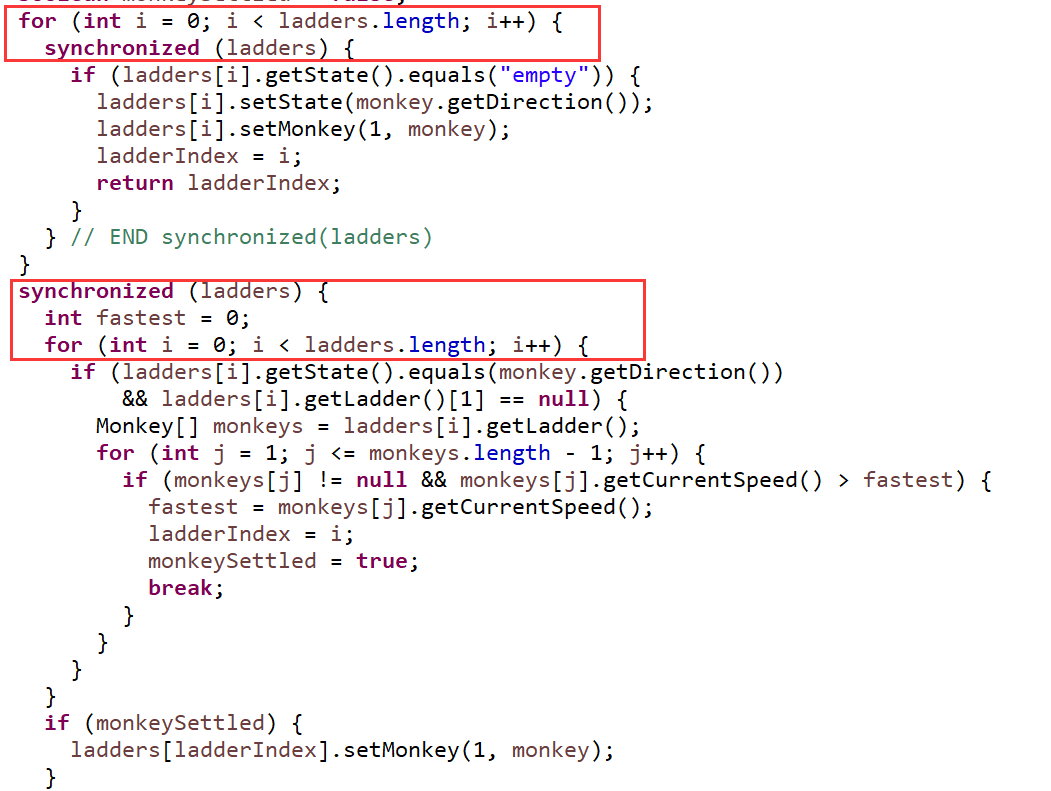
而一些一定会被修改的数据结构，则需要精密地加锁。对于logger的加锁，是把logger的写操作封装在了一个类的静态方法中，需要写日志时静态地调用该方法即可。



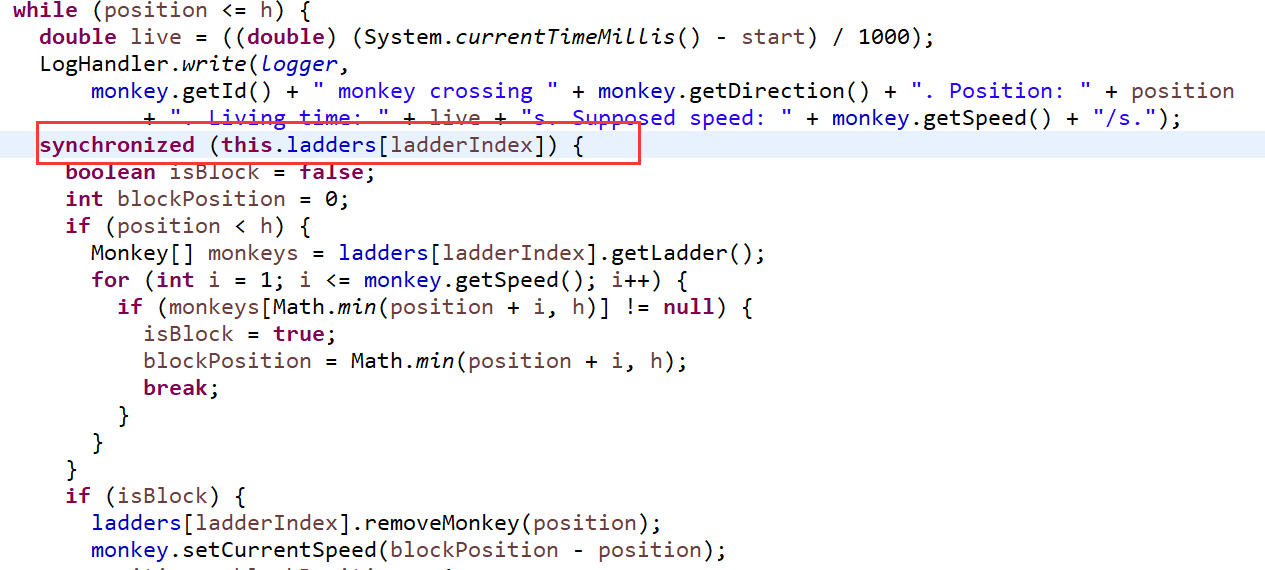
又waitingMonkeys是线程安全的集合类，对它的操作也只有add和remove，所以可以认为它也是线程安全的。

因此，精密加锁的地方主要是CrossExecute线程类和策略实现类中对ladders的访问和修改。

在执行梯子选取策略时，凡是在寻找梯子的迭代中，只要有满足条件的立马返回的，都可以在迭代内部对一条梯子加锁，而不用对所有梯子加锁。而有些策略需要遍历所有梯子才能得知哪条梯子是符合条件的，这时需要在遍历循环外给整个梯子数组加锁。比如说：

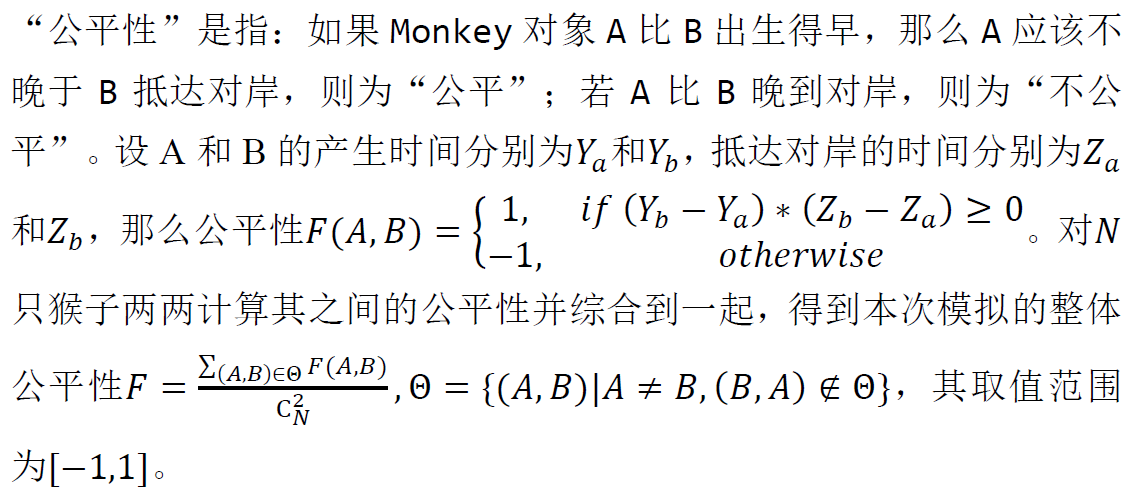


然后就是执行过河的线程中的加锁。这里加锁就比较简单，在每一次执行移动时，分为判断是否阻塞，和移动两步，这两步都是对ladders的访问，且不能被分割，要一起加锁，且只需要对该猴子所在的梯子加锁即可。



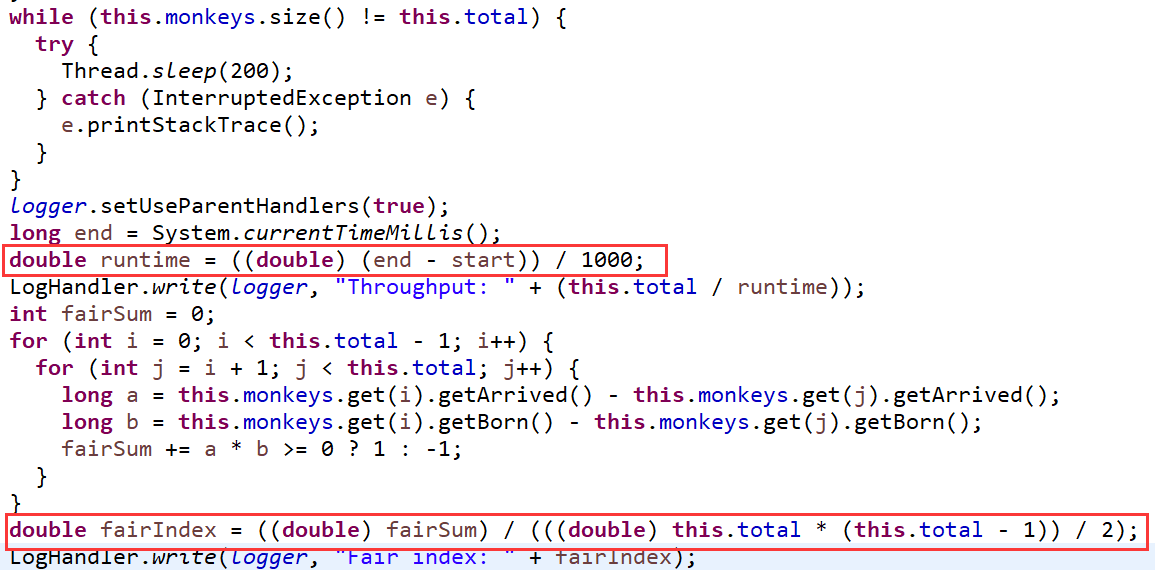
## 系统吞吐率和公平性的度量方案

吞吐率和公平性将在所有猴子过河结束后计算。吞吐率为猴子总数除以总用时，而公平性将按照如下的公式计算：



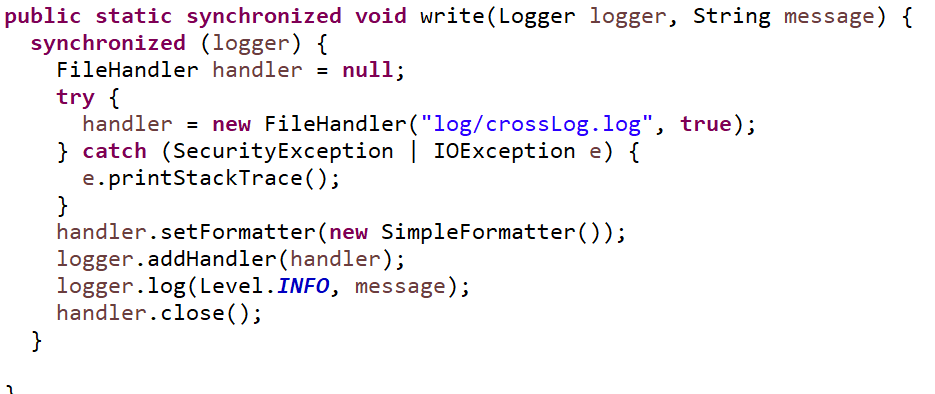
因为，为计算公平性，所有猴子的过河用时应该保存下来，这里保存在了Monkey类的born和arrived属性中，所有的Monkey对象又保存在monkeys列表中。

计算如下：



## 输出方案设计

将写日志操作封装在了一个类的静态方法中，需要写日志时静态地调用该方法即可。在该方法中，创建file handler指向日志文件，写入一条日志信息后，马上关闭file handler，可以有效避免在程序运行结束后.lck文件的出现，也避免了多个重名的.log文件。



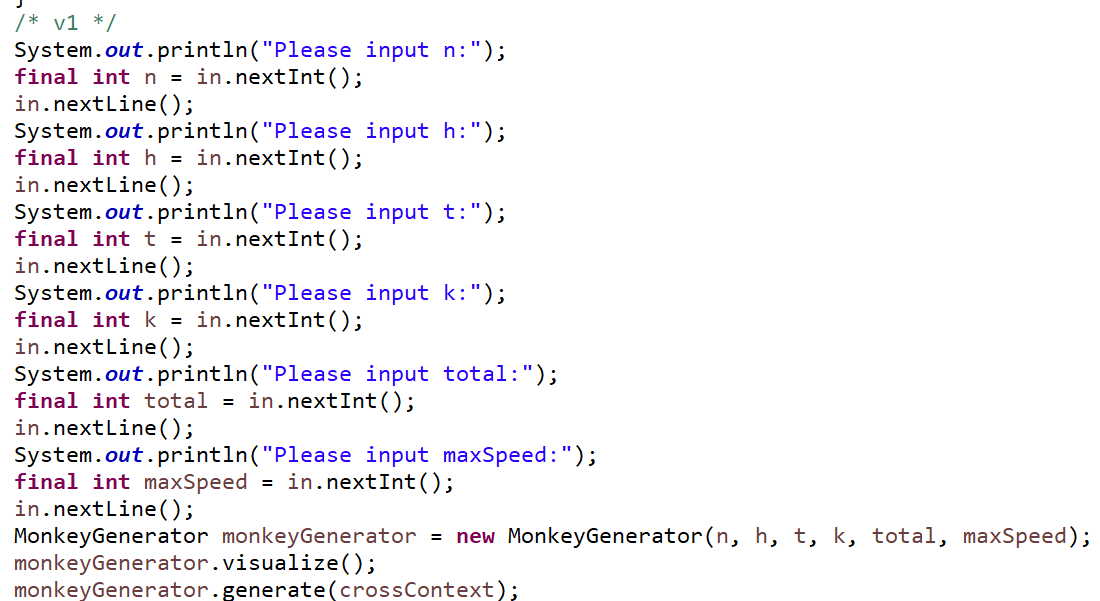
我的GUI代码完全仿照实验五中的行星运动模拟，并加以改造。该动态GUI策略借鉴了《Java轻松学》中的教学，具体实现在本人的GitHub上有总结。实现思路是在visualize方法中，初始化JFrame和JPanel，这里的JPanel是自己写的一个VisualPanel类，该类继承了JPanel，该类中还有一个内部类VisualListener,该类应用了ActionListner接口，并只有一个重写方法actionPerformed。该方法中调用方法repaint(). 在VisualPanel的构造器中，初始化定时器Timer对象，用javax.swing.Timer传入延迟参数和一个VisualListener实例参数，所以每隔延迟秒数，就会构造一次VisualListener监听器中的actionPerformed方法，并调用repaint，repaint方法会自动触发paintComponent方法，该方法是VisualPanel中的方法，该方法与JPanel中的VisualPanel方法重名，属于重写。该方法中，具体绘制了一个画面，该画面是当前梯子状态，参数等。所以最终效果，就是每隔延迟秒数，就会根据当前状态刷新一次Panel. 如果延迟秒数足够小，比如33毫秒，那么就是30帧，这个帧数在人眼看来，就是连续的动态画面了。

GUI参考的代码模板（《Java轻松学》）总结在个人的GitHub中：<https://github.com/szm981120/Learn_Java_the_Easy_Way/tree/master/BubbleDrawGUI>

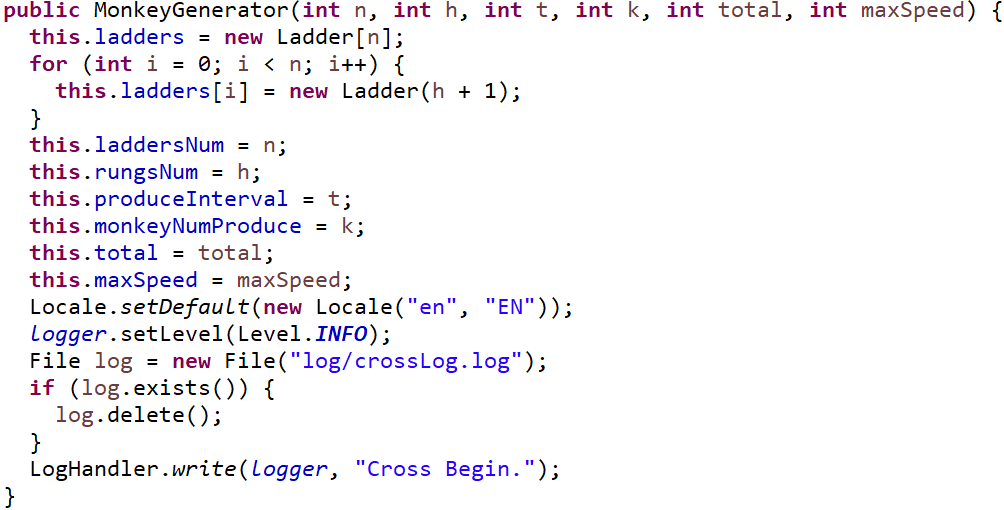
## 猴子过河模拟器v1

### 参数如何初始化

关于过河条件的6个参数，是由用户标准输入给出的。

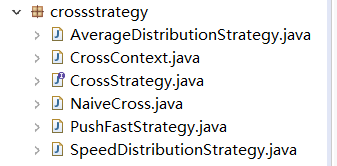


在猴子生成器构造时，向构造器中传参，初始化这些参数。



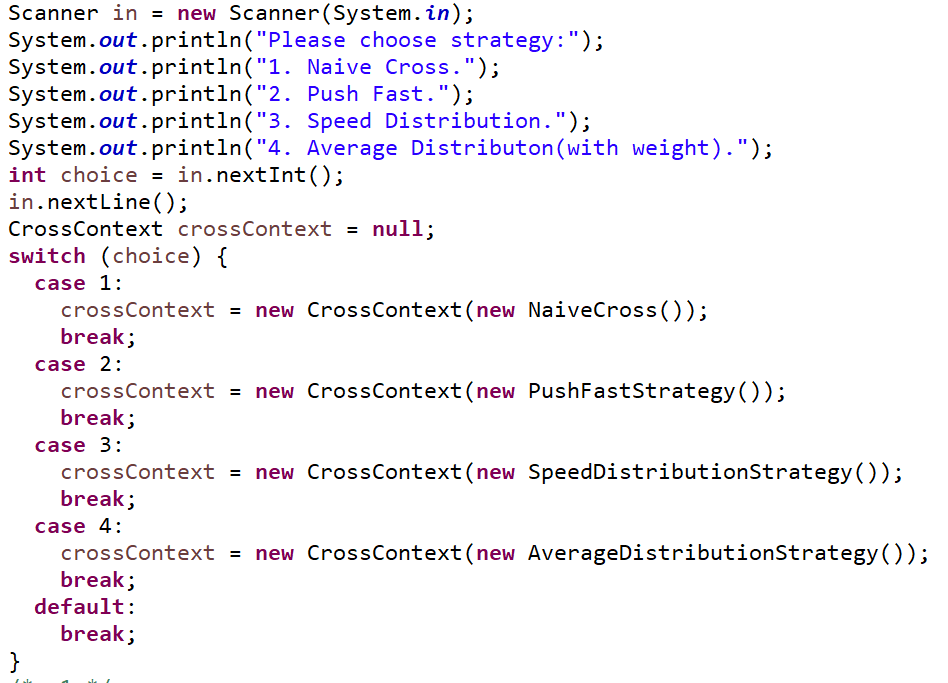
### 使用Strategy模式为每只猴子选择决策策略

Strategy模式已经连续两次实验中使用了，具体使用方法不再赘述。Strategy类的结构如下所示：



CrossStrategy是策略接口。CrossContext是策略上下文类。其余的四个是策略具体实现类，应用了策略接口，重写策略接口中的方法。

策略的选择由用户从标准输入给出。每只猴子的策略相同。

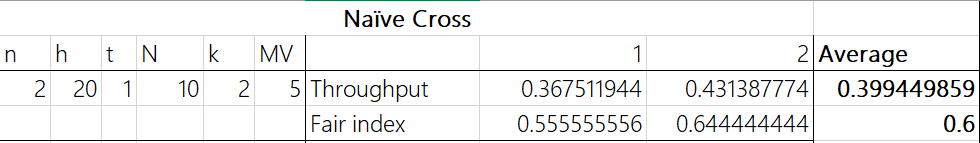


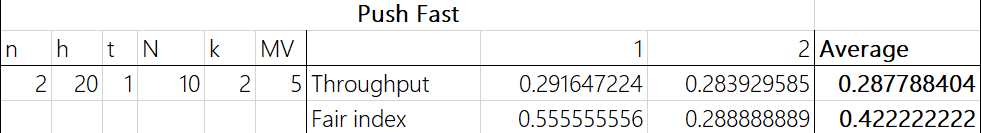
## 猴子过河模拟器v2

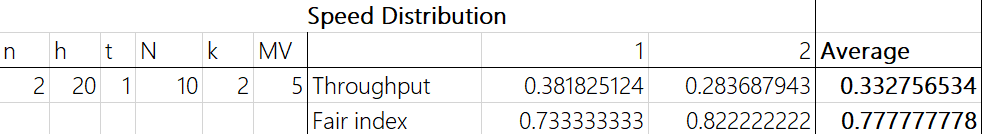
在不同参数设置和不同“梯子选择”模式下的“吞吐率”和“公平性”实验结果及其对比分析。

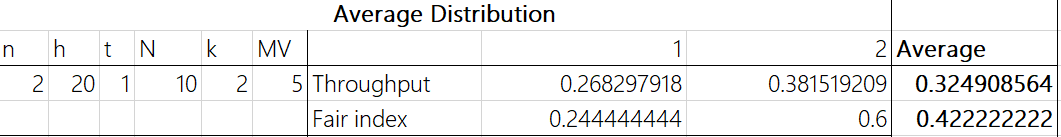
### 对比分析：固定其他参数，选择不同的决策策略

注：详细表格在3.9.2节给出。

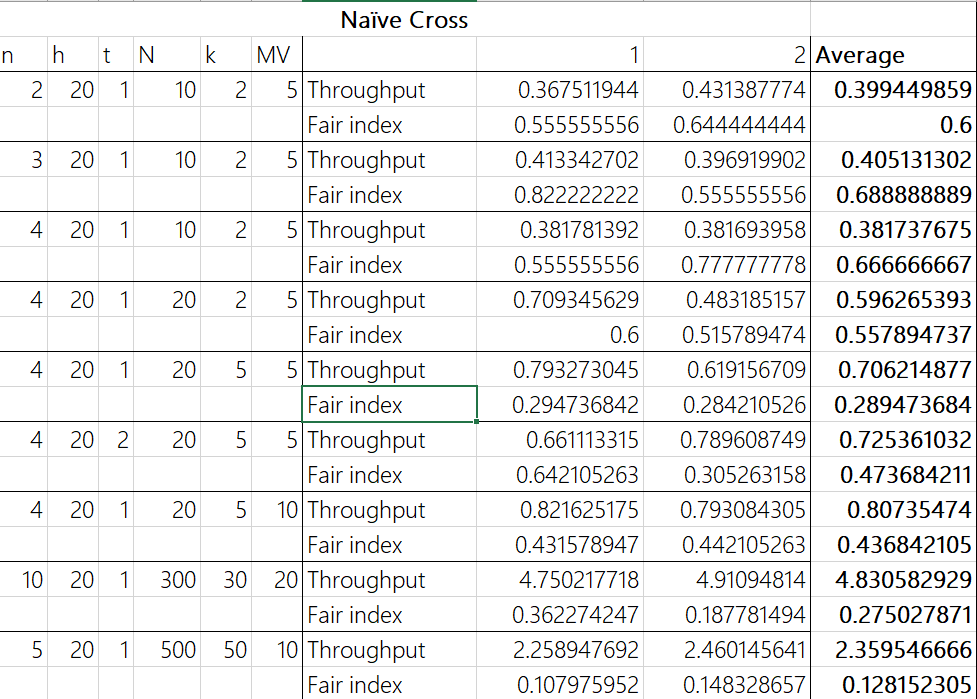


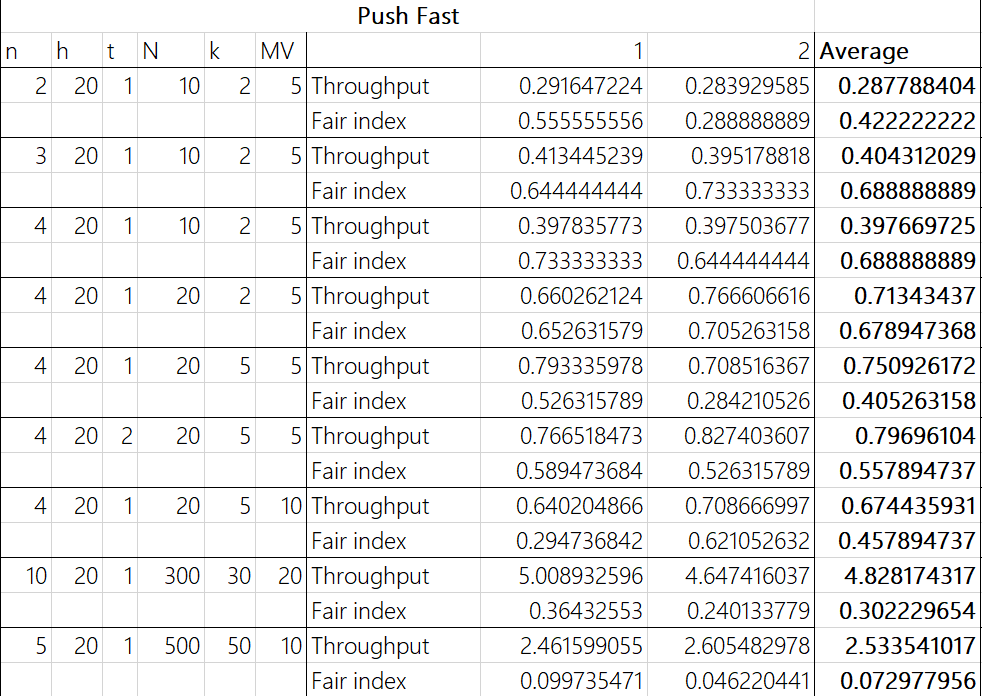


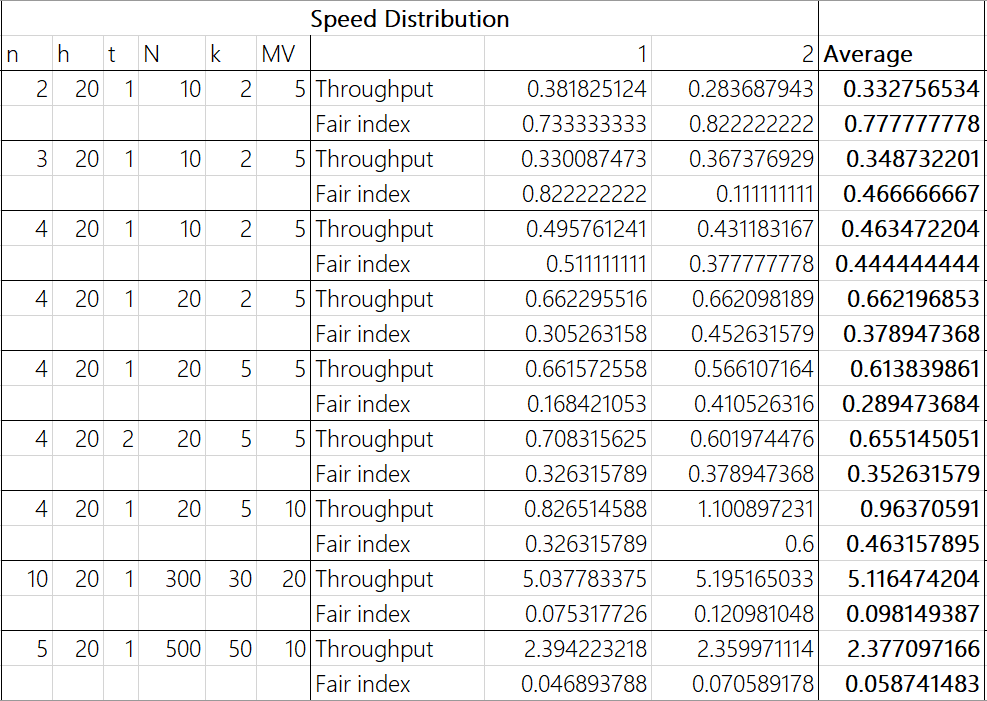


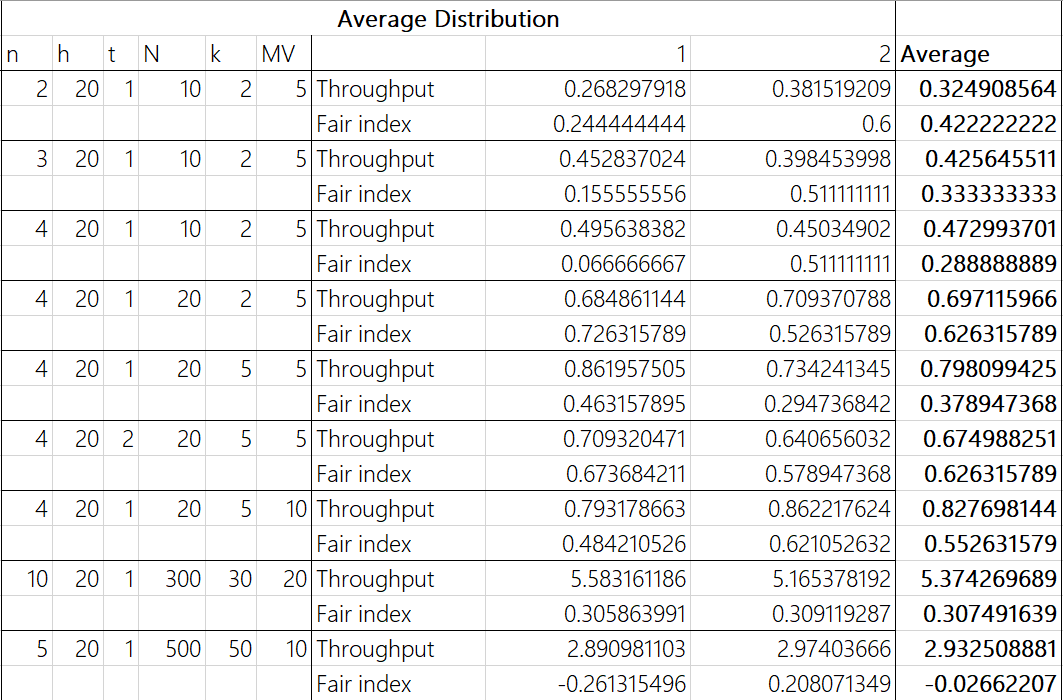


### 对比分析：变化某个参数，固定其他参数









### 分析：吞吐率是否与各参数/决策策略有相关性？

结论：吞吐率与各参数有直接关系，与决策策略有关，但关系不大。

由于v1和v2的测试中，各参数由测试者决定，且每只猴子的方向为随机生成，又每只猴子的速度在[1,MV]范围内随机生成。显然，MV越大，整体推进速度越快，但是这仍然不能绝对影响吞吐率，因为如果某次测试中，即时前面的猴子生成的速度很快，到后面的猴子生成的速度很慢，仍然会使整体运行时间变长。

能对吞吐率有直接较大影响的是，是梯子数n，横杆数h（本实验中横杆数h固定为20），显然n越大，h越小，吞吐率越大，这对吞吐率有绝对性的影响。

其余参数，比如t值，在上面的实验结果中可以发现，当其它参数不变，而t从1变为2时，前三种策略的吞吐率都得到了小幅提升，而策略4反而下降了0.12.

猴子总数N对吞吐率也有一定影响。如果实时观测吞吐率可以发现，在N比较大的时候，在运行时，吞吐率的趋势是递增的，直到一段时间后趋于稳定，所以说，N比较小的时候，不能反映该策略的吞吐率，只有N比较大的时候，才能反应该策略的吞吐率。而且，在其他参数不变，策略不变的前提下，N在一定范围内越大，吞吐率越高。

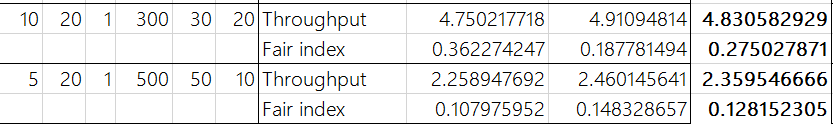
猴子的最大速度MV也对吞吐率有一定影响，但这个影响不是绝对的，上面已经分析过了。比如在Speed Distribution策略中，当其他参数不变，MV从5变为10时，吞吐率上升了0.35，但在Average Distribution策略中，其他参数不变，MV从5变为10时，吞吐率仅上升了0.02，这说明总体上MV越大，吞吐率越高，但也有例外，这取决于随机生成时的速度分布。

至于k值，理论上来说，k值的大小能否影响吞吐率，还是要取决于梯子的数目n，如果只有2条梯子，即使每秒生成10个猴子，也总还是要有8只猴子要等待的。但如果有10条梯子，那么k=10显然要比k=5吞吐率更高。

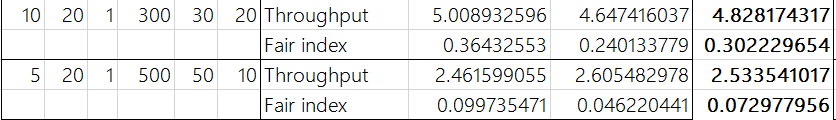
关于策略对吞吐率的影响，各个策略的优劣在3.3节已经介绍了，Speed Distribution是最不容易出现阻塞的策略，但Average Distribution是这四个中最好的策略。理论上也分析了，性能表现如何，取决于速度较慢的猴子是如何分布的，慢猴子后面是否跟了许多猴子，在过河后期，是否存在一架梯子上有一长队猴子，而其他梯子上会提前空闲出来，如果有这种现象出现，那么显然将那一长队平均分配到这些早就空闲出来的梯子上是更快的。所以有了平摊策略。

### 压力测试结果与分析

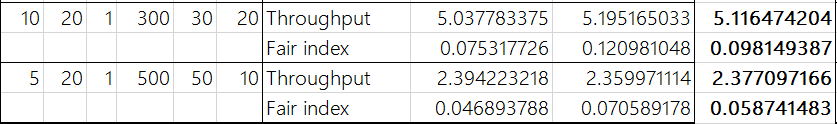
Naïve Cross



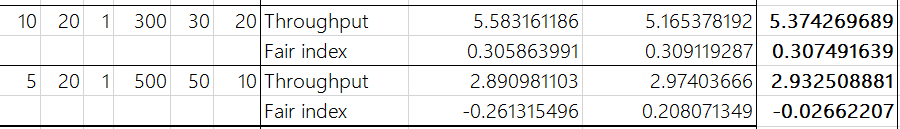
Push Fast



Speed Distribution



Average Distribution



显然，猴子越多，公平性越差，这是因为猴子数量越多，等待的猴子就会越多，很有可能连续很长一段时间里，所有梯子都是从左向右行进的，但是这期间总是会生成出一些从右向左的猴子，这些等待的猴子会降低公平性。

另外，由于数据量比较大，虽然各个参数不尽相同，但是可以看出，n=10的那一组测试的吞吐率近似是n=5这一组的2倍，验证了梯子数量对吞吐率有决定性影响的猜想。

另外大数据的测试也体现了各个策略的优劣，显然平摊策略的四个中最优的，其次是很少发生阻塞的速度分配策略。但速度分配策略在梯子数量较小时表现不太好，这可以看成是车道较少，速度分配策略一个体现就是把速度相近的分到了同一个梯子上，如果梯子选择本来就少，那么它的优点就不容易体现出来了。

## 猴子过河模拟器v3

针对教师提供的三个文本文件，分别进行多次模拟，记录模拟结果。

Speed Distribution Strategy是四个策略中，运行竞赛文件表现最好的策略。

|  |  |  |
| --- | --- | --- |
|  | 吞吐率 | 公平性 |
| Competiton\_1.txt | Speed Distribution Strategy | |
| 第1次模拟 | 2.77325839372873 | 0.592329988851727 |
| 第2次模拟 | 2.66096628555716 | 0.58515050167224 |
| 第3次模拟 | 2.72294077603812 | 0.607357859531772 |
| 第4次模拟 | 2.74810838539472 | 0.629163879598662 |
| 第5次模拟 | 2.639079489074211 | 0.4218060200668896 |
| 第6次模拟 | 2.698011565476244 | 0.39054626532887404 |
| 第7次模拟 | 2.6744165314600528 | 0.5674024526198439 |
| 第8次模拟 | 2.6385920472835696 | 0.4688963210702341 |
| 第9次模拟 | 2.6857894878199446 | 0.5802452619843924 |
| 第10次模拟 | 2.648796122162477 | 0.6060646599777034 |
| 平均值 | 2.6889959083995 | 0.5448963210702 |
| Competiton\_2.txt | Speed Distribution Strategy | |
| 第1次模拟 | 5.7217403245371115 | 0.6875030060120241 |
| 第2次模拟 | 5.619303431146675 | 0.5986693386773547 |
| 第3次模拟 | 5.334812854765055 | 0.43953507014028054 |
| 第4次模拟 | 5.335951506872706 | 0.5571302605210421 |
| 第5次模拟 | 5.309772104581271 | 0.633314629258517 |
| 第6次模拟 | 5.2782704163499705 | 0.36878557114228455 |
| 第7次模拟 | 5.395722271383248 | 0.6392464929859719 |
| 第8次模拟 | 5.634310698429154 | 0.6602004008016032 |
| 第9次模拟 | 5.6232848981060775 | 0.698565130260521 |
| 第10次模拟 | 5.494867793480889 | 0.5876553106212424 |
| 平均值 | 5.4748036299652 | 0.5870605210421 |
| Competiton\_3.txt | Speed Distribution Strategy | |
| 第1次模拟 | 1.2506878783330833 | 0.6642424242424242 |
| 第2次模拟 | 1.2664159163152364 | 0.6351515151515151 |
| 第3次模拟 | 1.2560131630179483 | 0.5842424242424242 |
| 第4次模拟 | 1.1843710397593357 | 0.25696969696969696 |
| 第5次模拟 | 1.3690190978164145 | 0.6703030303030303 |
| 第6次模拟 | 1.4483098224372157 | 0.7038383838383838 |
| 第7次模拟 | 1.3669792492549964 | 0.6901010101010101 |
| 第8次模拟 | 1.3482540110556829 | 0.6880808080808081 |
| 第9次模拟 | 1.3693002875530604 | 0.7151515151515152 |
| 第10次模拟 | 1.4075387776933252 | 0.711919191919192 |
| 平均值 | 1.3266889243236 | 0.632 |

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 如何给猴子创建线程，并在线程之间共享梯子状态 | 一开始是在猴子生成器中，每创建一个猴子就创建一个内部类启动线程，但发现这么做不太灵活，所以就设计一个线程类CrossExecute，继承Thread类，重写run方法，设计一些需要共享的rep，并用给构造器传参的形式共享这些数据。 |
| 如何判断所有猴子都已经过河？ | 可以设计一个暂停循环，当当前线程数不为2时，就进入循环，并sleep 500毫秒（或小于1000毫秒的一段时间），当猴子都已经过河后，线程数为2，为主线程和GUI线程。但是，如果当GUI没有启用的话，又要将这里的线程数改为1.所以后来又将属性monkeys定义为已经过河的猴子列表，循环判断该列表的大小是否为total，如果不是，则进入sleep，跳出循环后，就可以计算吞吐率和公平性了。 |
|  |  |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

这次实验训练的创建线程和多线程的线程安全设计，灵活运用synchronized对多个线程之间共享的数据加锁，也学到了如何合适地在多个线程中共享数据，加深了GUI动画和strategy设计模式的印象。

## 针对以下方面的感受

1. 多线程程序比单线程程序复杂在哪里？你是否能体验到多线程程序在性能方面的改善？

主要就复杂在多线程之间可能发生冲突，多线程程序可以同时执行一些代码。

1. 你采用了什么设计决策来保证threadsafe？如何做到在threadsafe和性能之间很好的折中？

尽量使用简单数据类型和线程安全的数据结构，对mutable的数据要精确地用synchronized加锁。首先要必须保证threadsafe，如果线程不安全，很有可能程序就是错误的，在此基础上，考虑能否减少synchronized的作用范围，使同一时刻更多的线程同时工作。

1. 你在完成本实验过程中是否遇到过线程不安全的情况？你是如何改进的？

用GUI可以发现最开始做的时候，会有猴子无视阻挡的情况。只要在合适的范围上对共享的梯子数组加锁即可。

1. 关于本实验的工作量、难度、deadline。

工作量最小，难度不大，deadline充裕。

1. 到此为止你对《软件构造》课程的意见和建议。

六个实验太多了，减一个吧，把第五个去掉，或者把第四个和第五个工作量再减一减，这个课从第一周第一天开始就在做实验，做了六个实验竟然做到了16周考试周，太长了。

1. 还有一周就要期末考试了，你准备如何复习？

算法课和软件构造要同时复习了，看PPT吧，从头到尾看一遍，总结一下。