

**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 | 孙毓霄 |
| 学号 | 1170300403 |
| 班号 | 1736101 |
| 电子邮件 | [997948850@qq.com](mailto:997948850@qq.com) |
| 手机号码 | 18846758039 |

**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。

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

# 实验环境配置

简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

在这里给出你的GitHub Lab6仓库的URL地址（Lab6-学号）。

<https://github.com/ComputerScienceHIT/Lab6-1170300403>

# 实验过程

请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

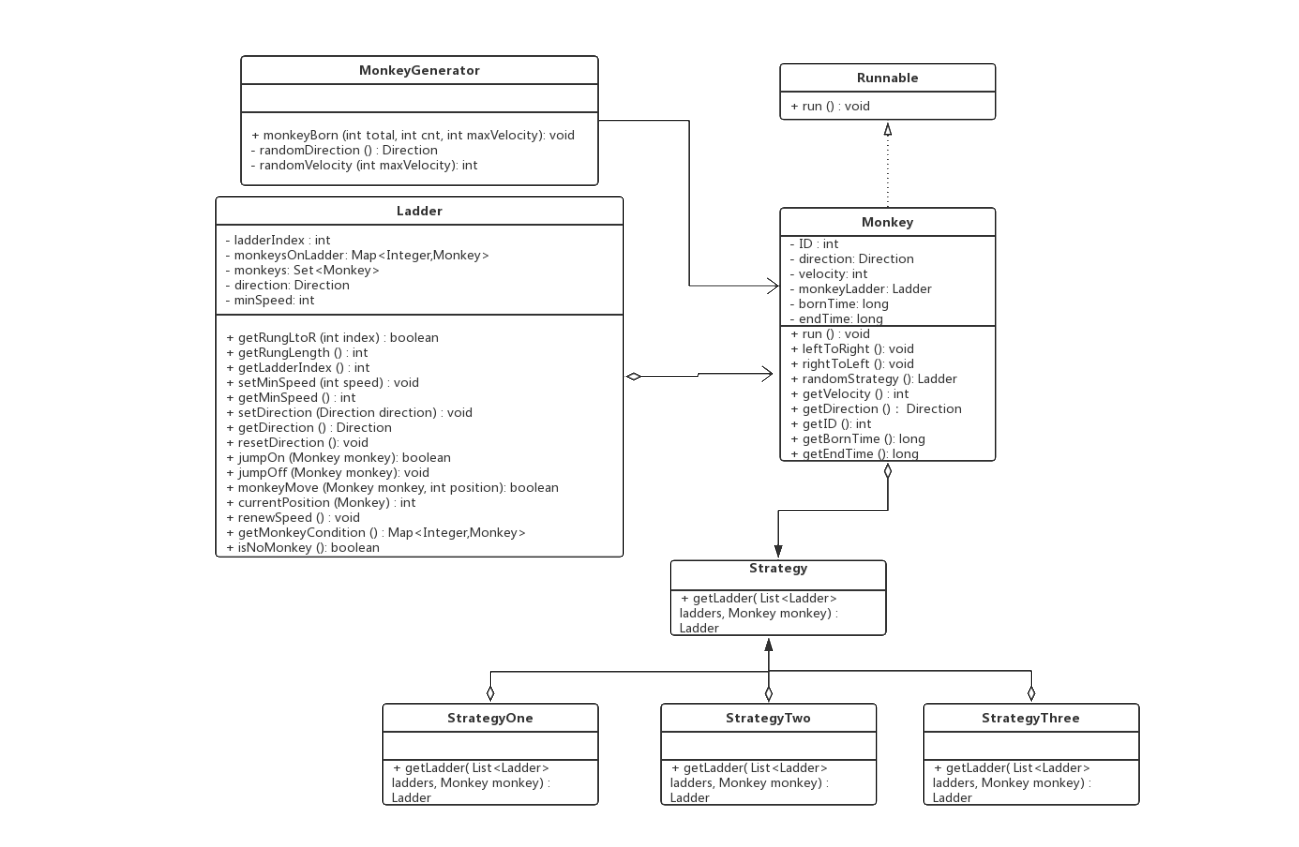
## ADT设计方案

设计了哪些ADT、各自的作用、属性、方法；

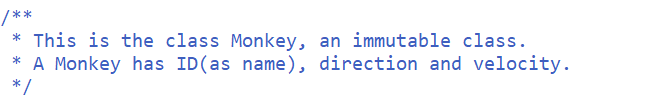
给出每个ADT的specification；

（可选）以类图形式给出多个类之间的关系。

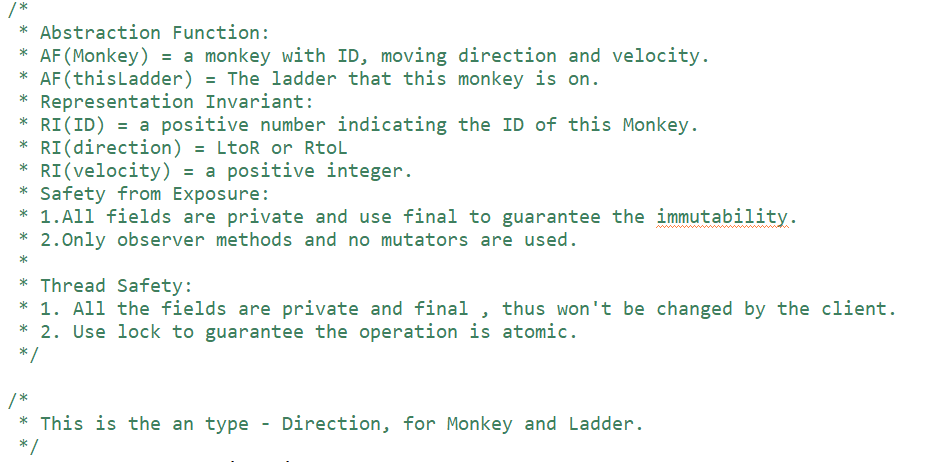
设计了梯子Ladder，猴子Monkey，猴子生成MonkeyGenerator, 接口Strategy和其三个实现类。客户端Client 和 可视化Cartoon。



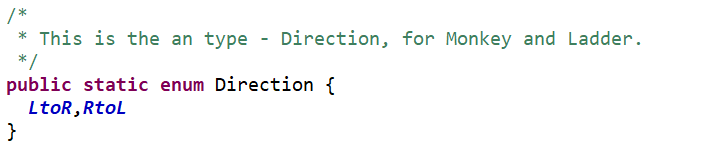
1. **Monkey**



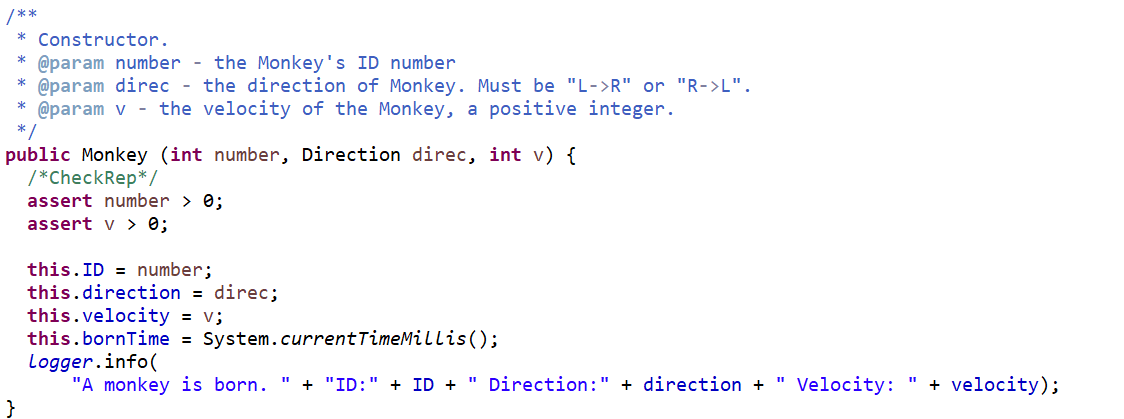
具体：



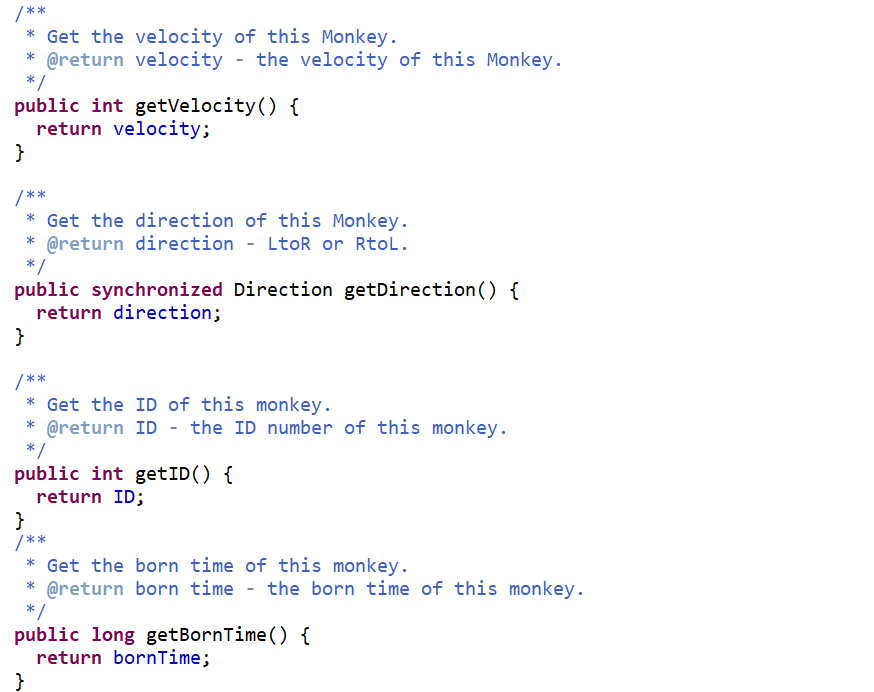
定义枚举类方向:



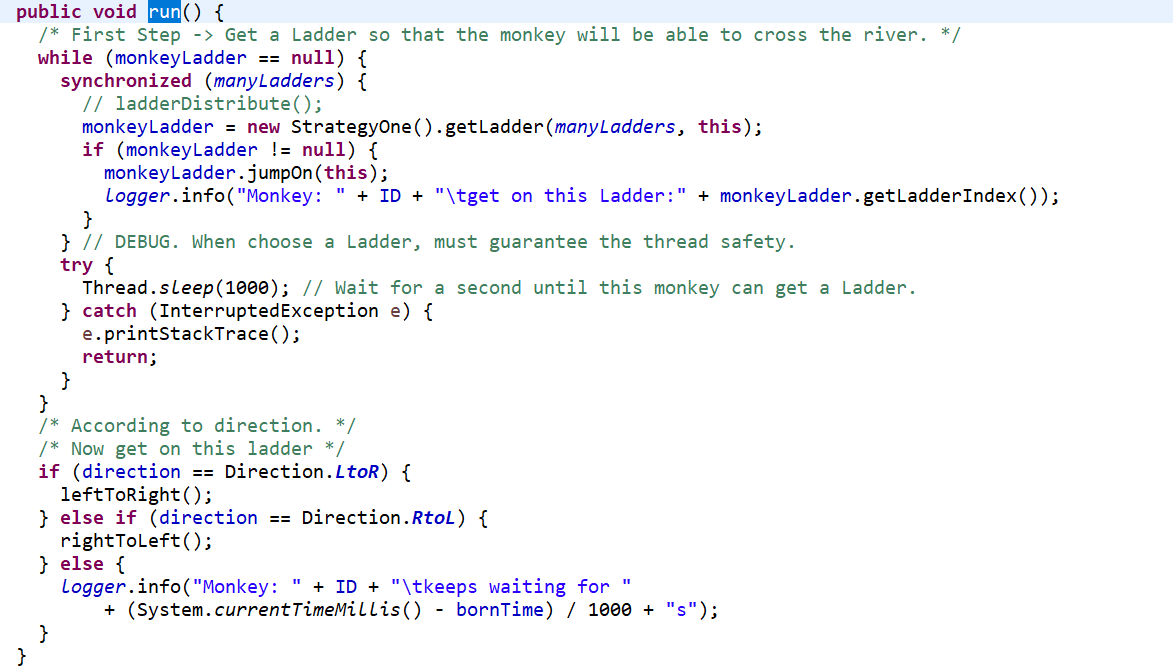
构造器：定义方向，ID和速度



多个Observer：



@override 继承了runnable，run方法：



在此，由于猴子具有两种方向，故设计两种运动: leftToRight()和rightToLeft。 以从左往右运动为例，基本思路：首先根据梯子上该猴子的位置currentPosition（this）确定猴子是否运动。如果不运动，线程结束，记载log信息。猴子跳下梯子。如果继续运动，则获取能够抵达的位置。如果能直达梯子的尽头并且（速度 + 现在位置）> 梯子尽头位置，则猴子在下一秒内跳下梯子。如果不能，则查看猴子前方一个踏板的位置是否有猴子，没有猴子则代表向前运动是可行的，则根据判断的reachPosition来使猴子顺利到达。

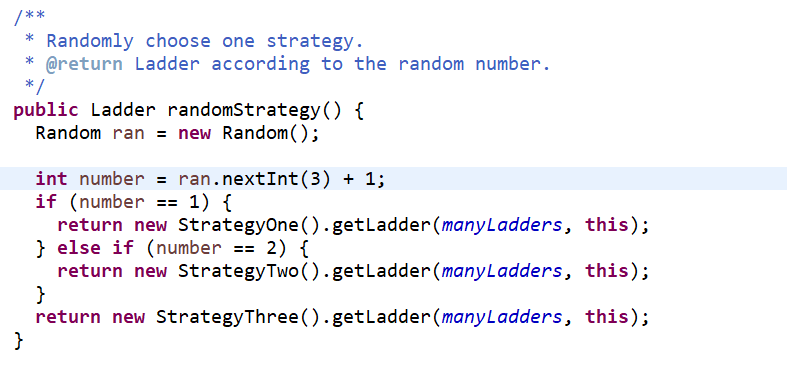
之后猴子要停一秒来完成此运动。





猴子的randomStrategy方法：

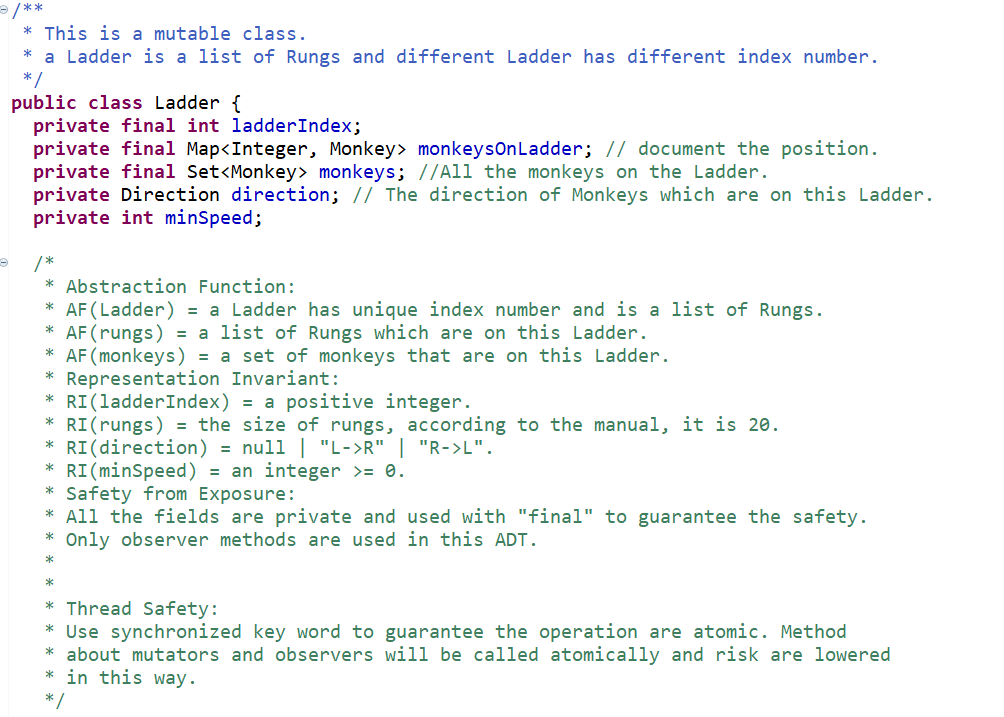
根据随机数1,2或3 来选择生成的策略。



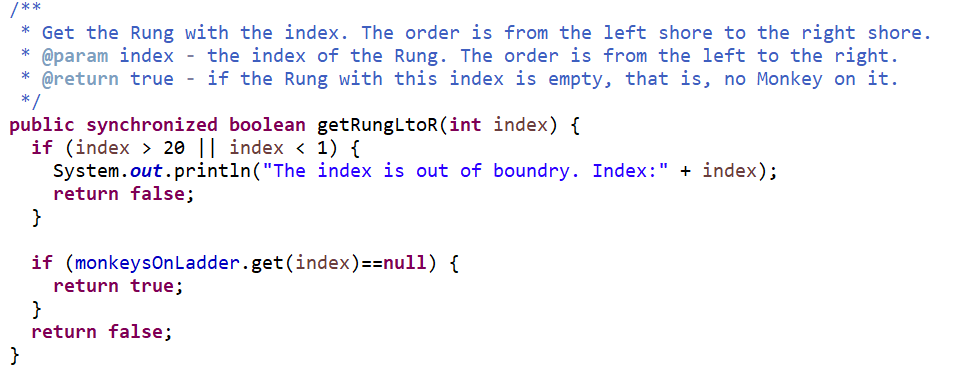
1. **Ladder**

主要方法：getRungLtoR(int index), monkeyMove(Monkey monkey, int position), jumpOn(Monkey monkey).  
Ladder的spec和其他说明：

踏板抽象为正整数序列。根据题意，脚标从1到20 。

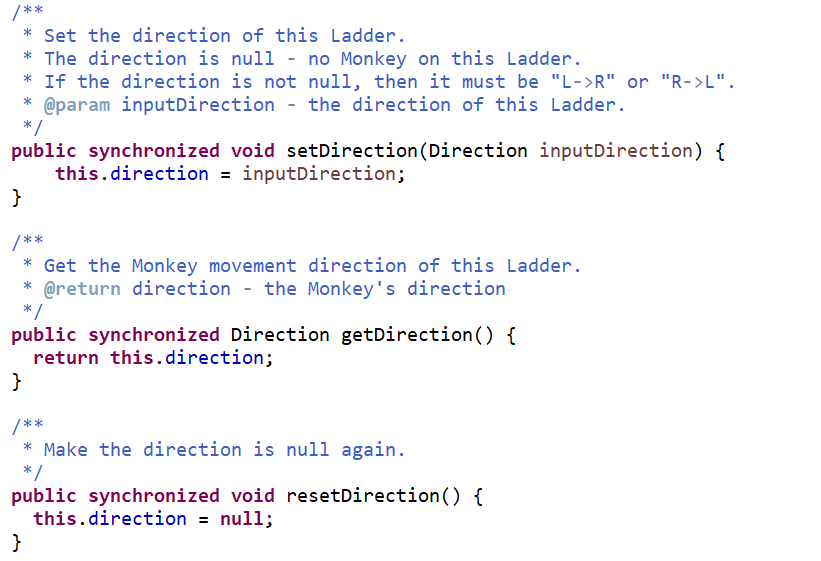


1. **public** **synchronized** **boolean** getRungLtoR(**int** index)



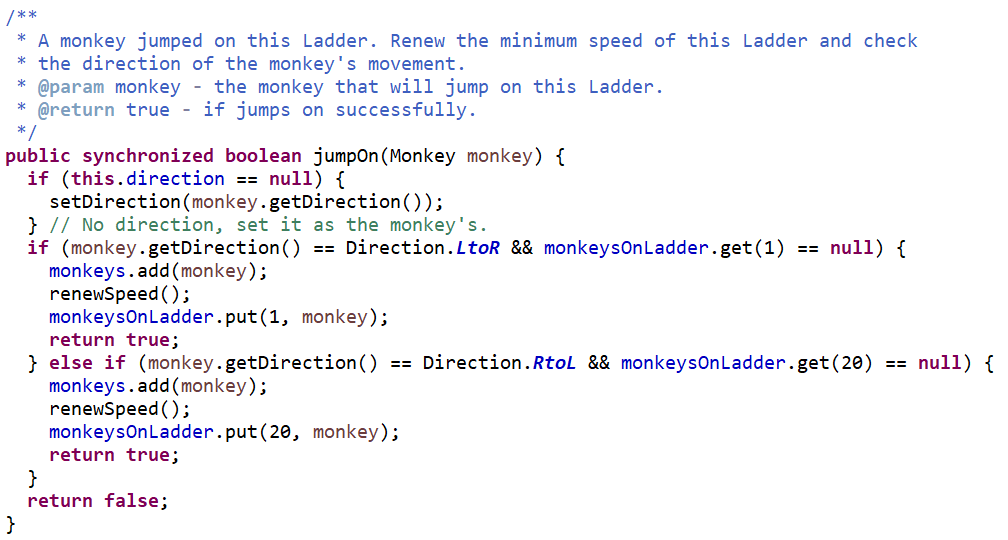
根据从左往右的顺序判断该踏板是否为空，若为空返回真，否则为false。

1. 梯子的方向

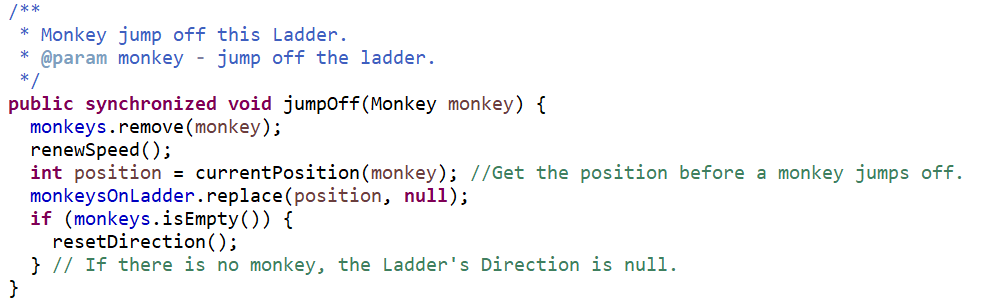


根据后面的策略设计，梯子是有方向的，梯子的方向即为上面猴子的方向。并且要访问梯子的方向。如果梯子上没有方向，则重置为null。

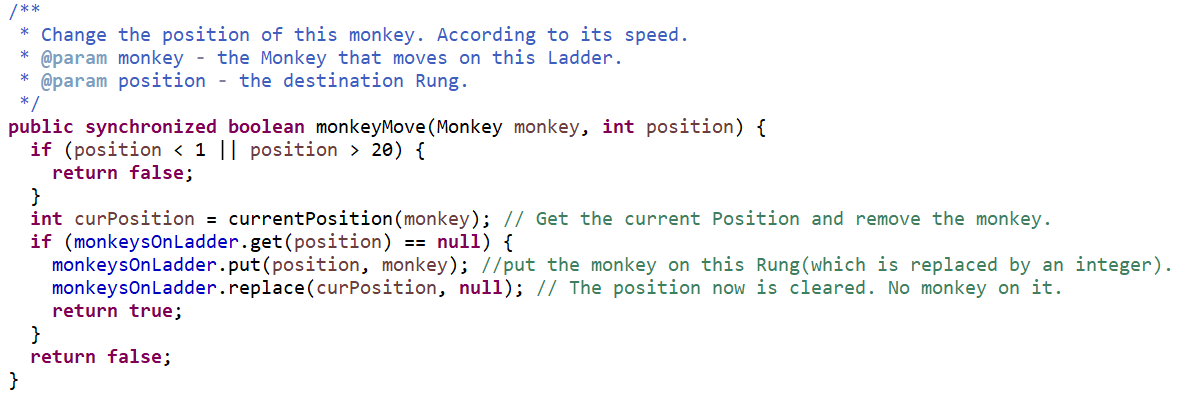
1. **public** **synchronized** **boolean** jumpOn(Monkey monkey)



猴子能否跳上梯子。判断思路：如果梯子没有方向（上面也没有猴子），则重置梯子的方向为跳上该梯子的猴子的方向，并且根据猴子的方向（左还是右）来放置其为第1个板还是第20个板。之后在map中放入key（板子的index）和猴子，返回true。否则返回false。

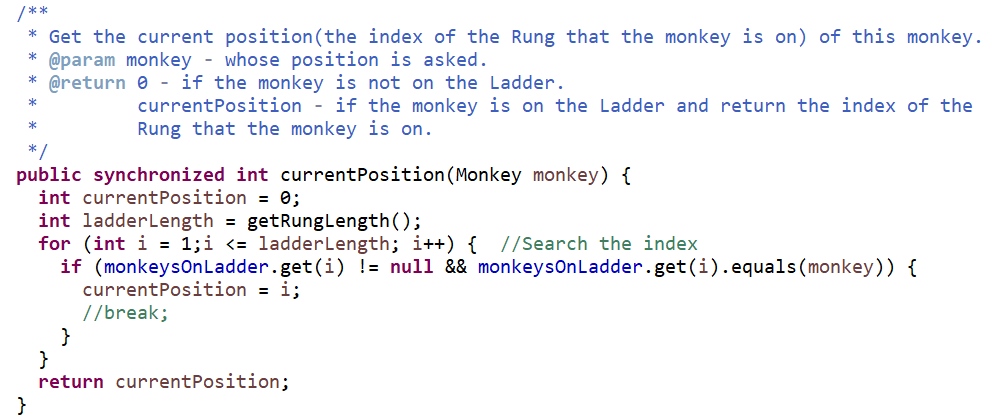


1. **public** **synchronized** **boolean** monkeyMove(Monkey monkey, **int** position)



该方法的思路是通过“猴子-目标踏板序号”来使得猴子运动到相应的位置，也就是到达某个踏板。

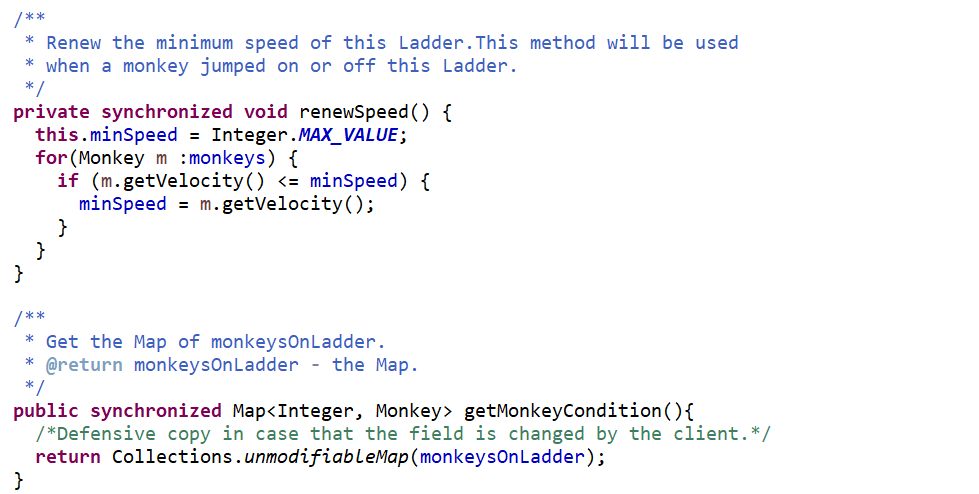
**public** **synchronized** **int** currentPosition(Monkey monkey)



获取当前猴子的位置。输入猴子，遍历这个梯子比较有没有与此猴子相同的猴子，若没有，则返回0（出界的位置），如果存在，返回该猴子的梯子的脚标。

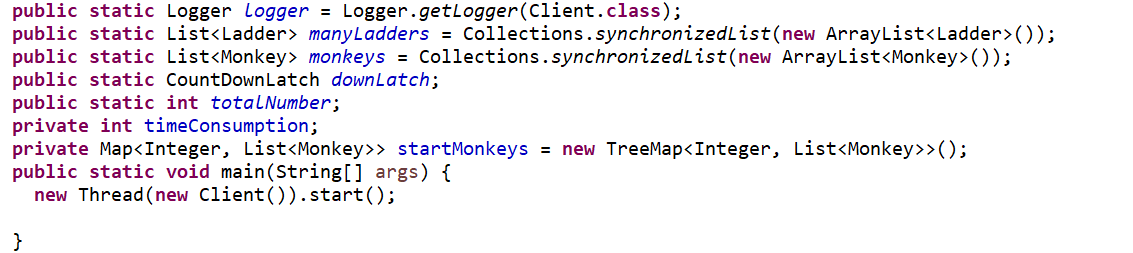
1. 更新速度和返回一个Map<Integer, Monkey>。

运用到了defensive copy的策略和unmodifiable来确保不会表示泄露。



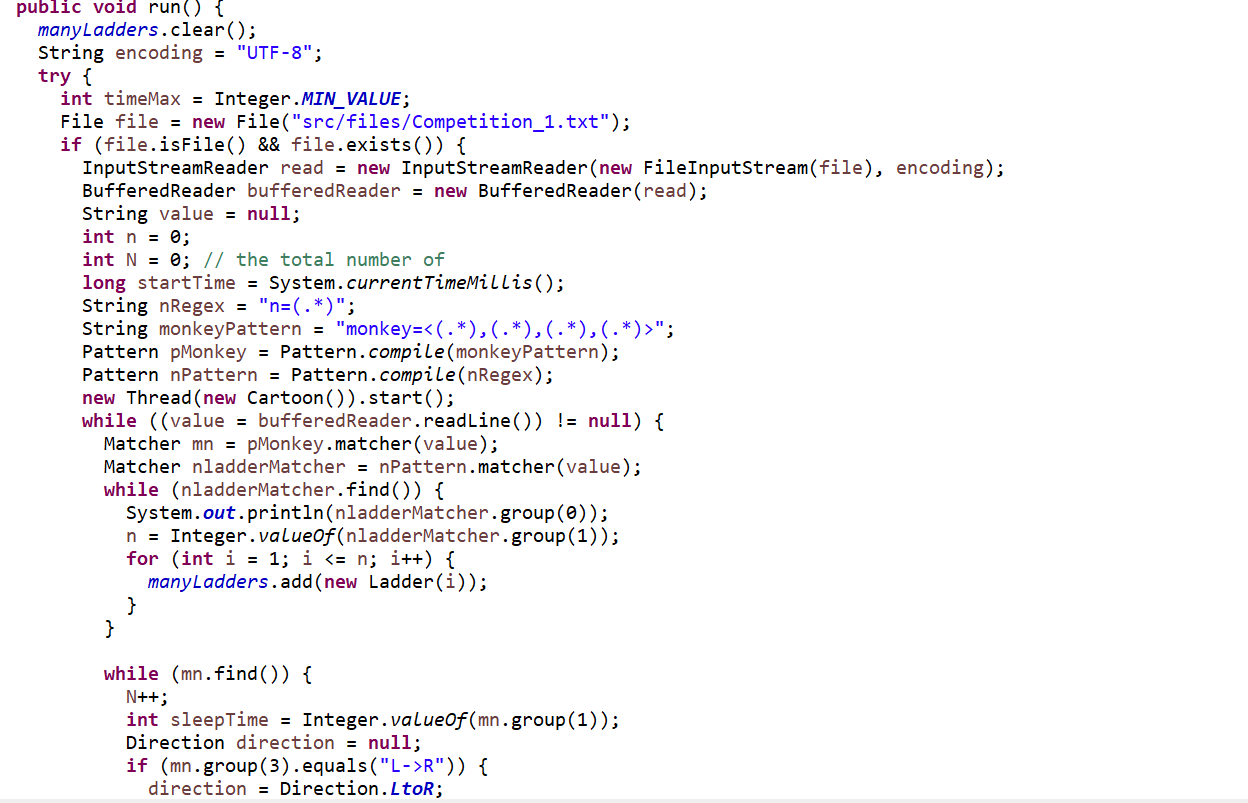
1. **Client**

该类中最多的是静态属性，希望全项目中的类都能够访问到



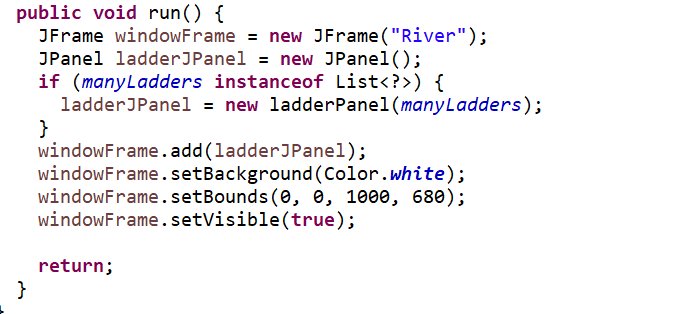
logger负责记录日志。manyLadders是记录所有的信息，并且让全部的线程都能共享该梯子的所有资源（保证一定的线程安全并且单个猴子执行操作时必须保证原子操作）。monkeys为保存所有猴子的列表，downLatch负责记录所有猴子的数目。timeConsumption是从第一只猴子诞生到最后一只猴子过河消耗的总时间。

run方法中主要是根据文本解析然后启动猴子线程

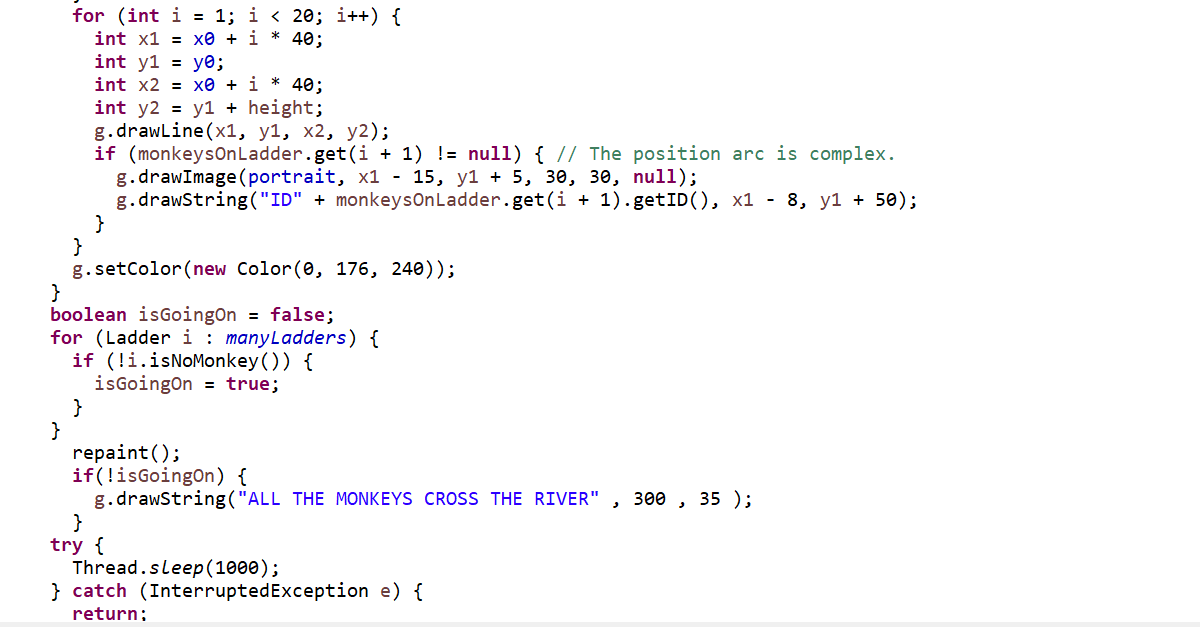


1. **Cartoon**

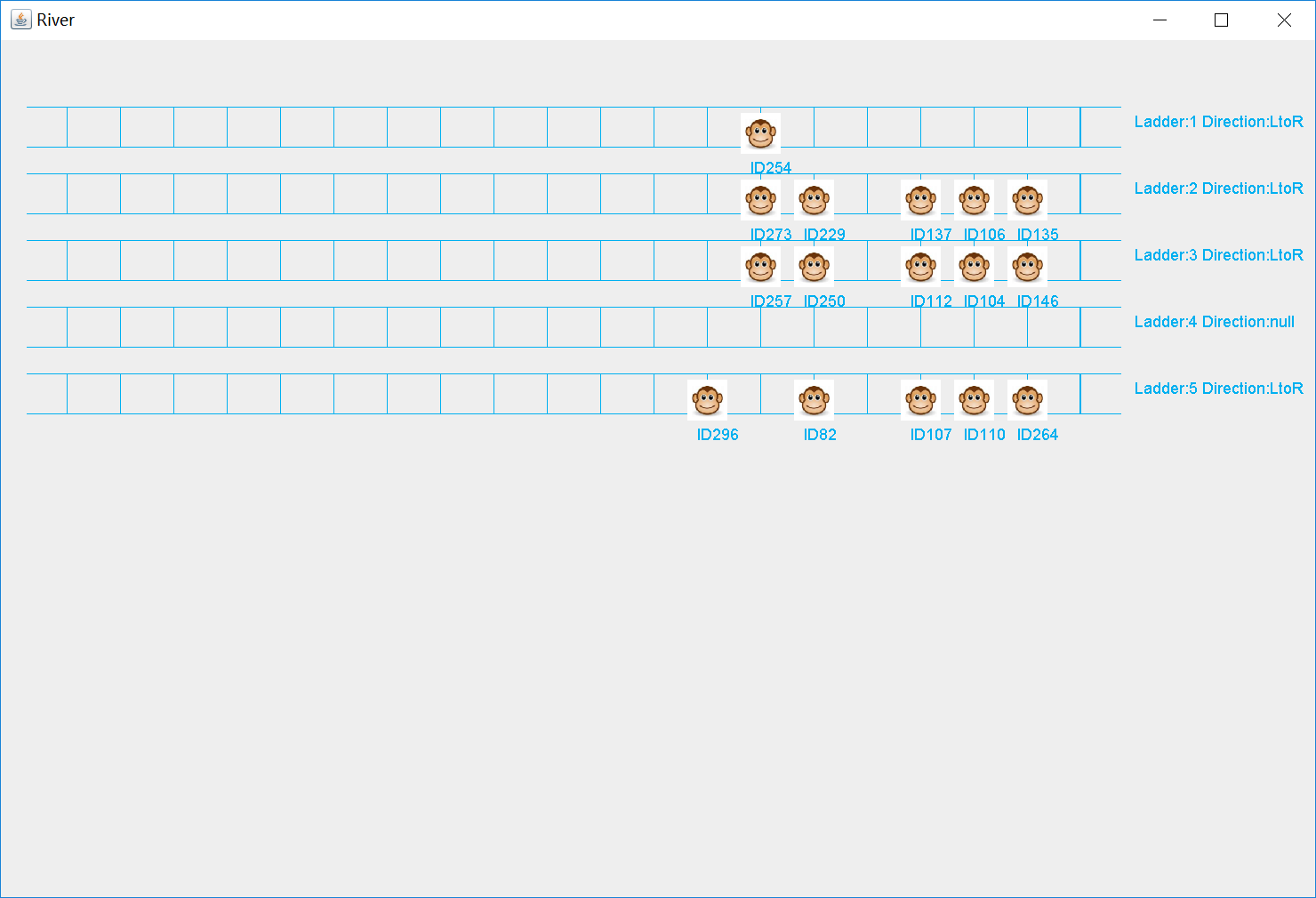
该类负责可视化，主要方法为run，画画板并且添加画笔。



根据猴子一秒一动的思路，采用每一秒重绘一次面板。



效果图：

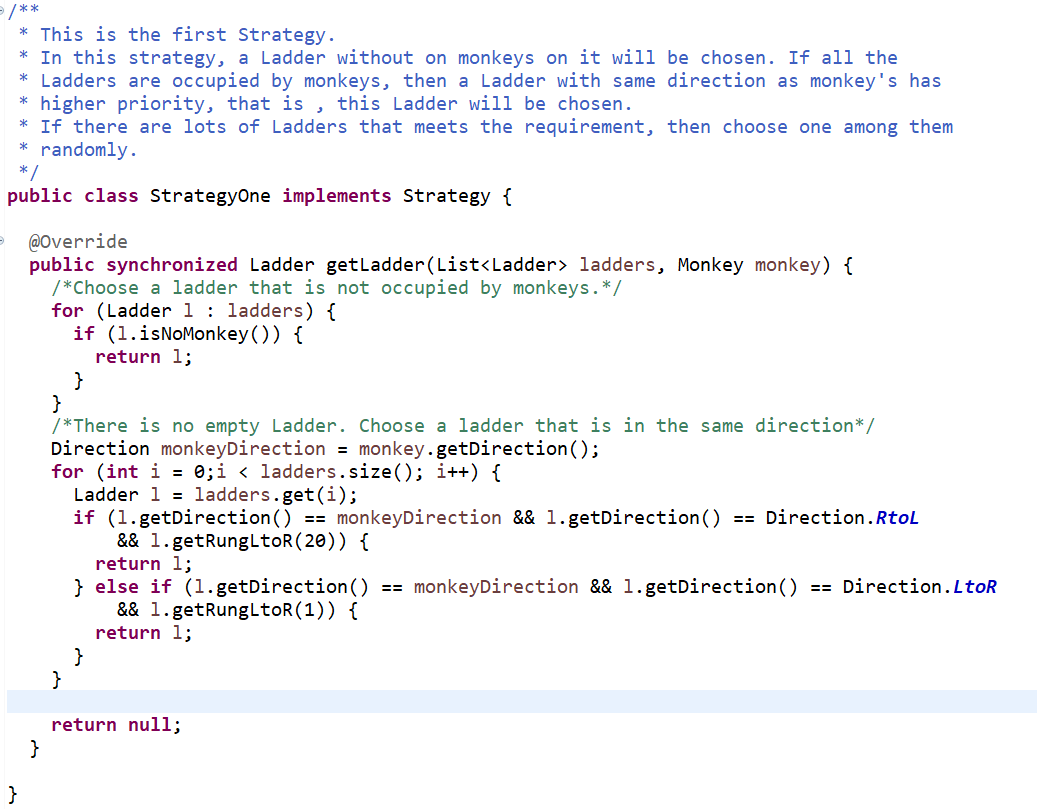


## Monkey线程的run()的执行流程图

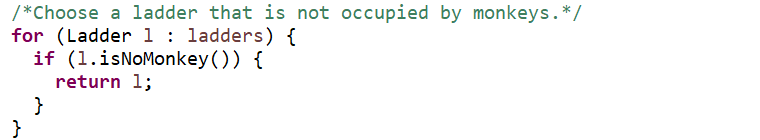
## 至少两种“梯子选择”策略的设计与实现方案

### 策略1

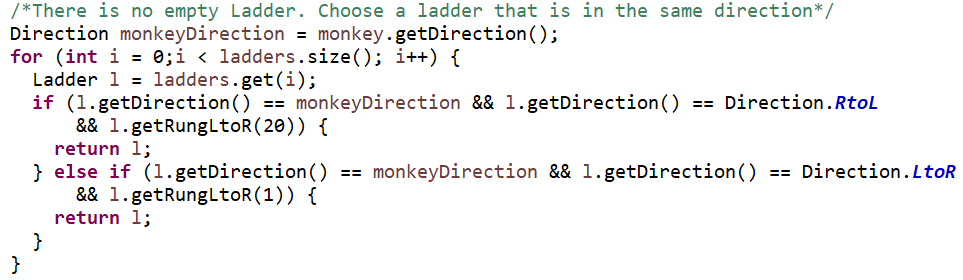
优先选择没有猴子的梯子，若所有梯子上都有猴子，则优先选择没有与我对向而行的猴子的梯子；若满足该条件的梯子有很多，则随机选择。



首选没有猴子的梯子：



如果有，则根据方向随机选择一个梯子：



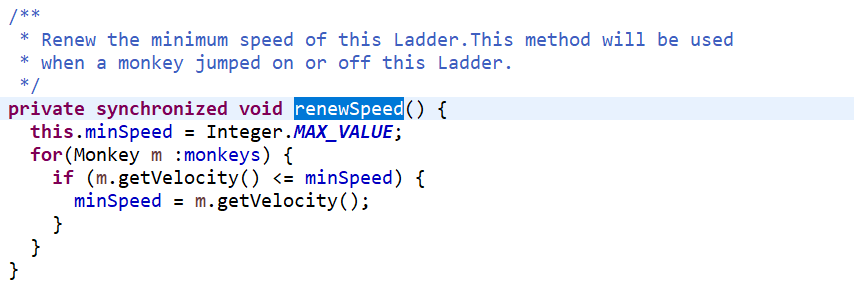
否则返回null。



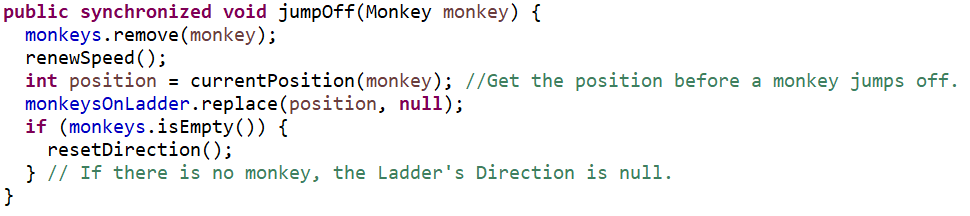
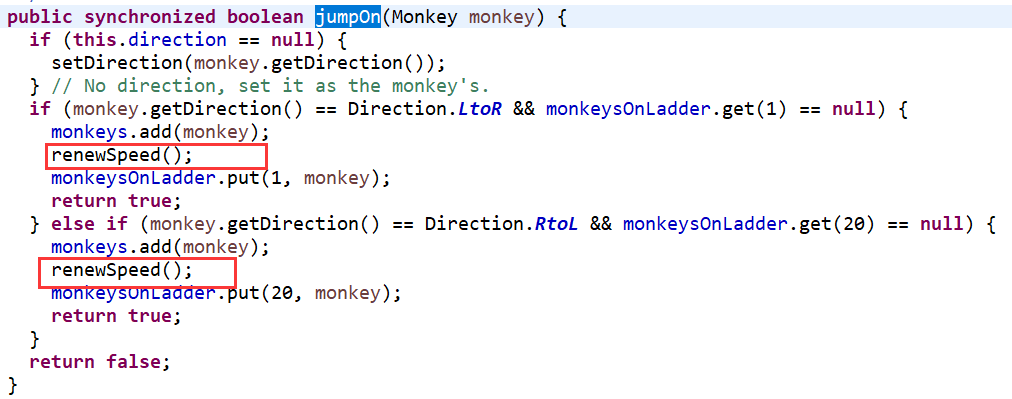
### 策略2

优先选择整体行进速度最快的梯子，前提：行进方向与本猴子的方向相同。其上猴子的最小速度最大，可以推知整体行进速度较快。

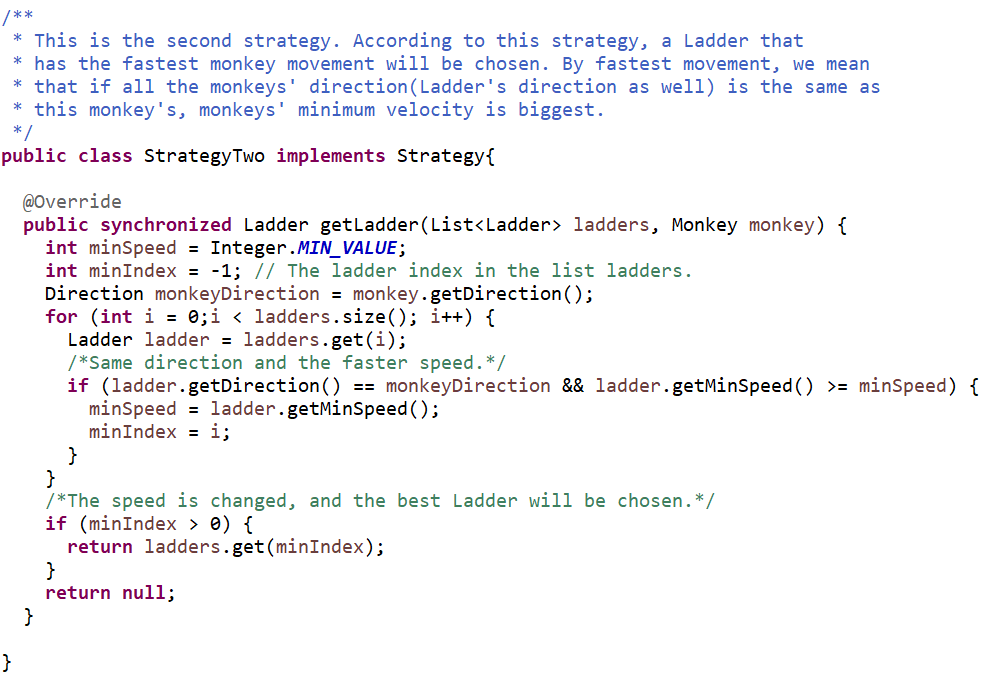
通过梯子上的minSpeed和renewSpeed在猴子跳上和跳下梯子的时候进行更新速度。



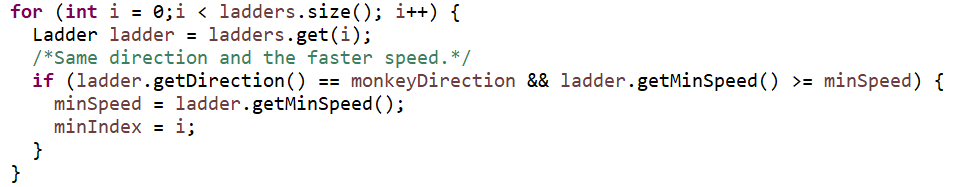
跳下和跳上梯子的时候都要更新速度。

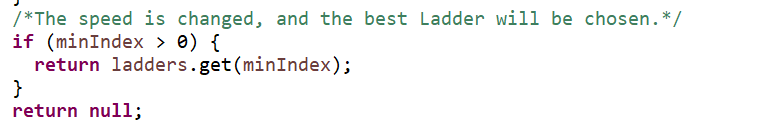
在Strategy上，利用具体类StrategyTwo来实现。



主要是通过遍历查找的算法，minIndex是具有最快速度的梯子的编号，minSpeed则是用来比较的值。



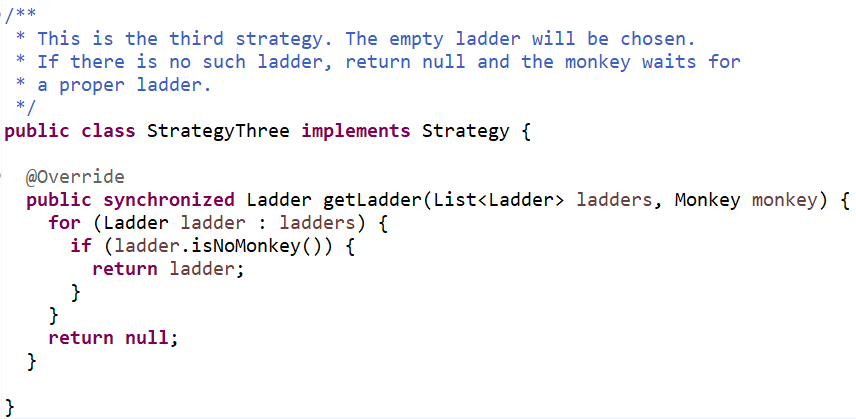
遍历来更新所选择的梯子（由于梯子数量很少，在此复杂度下算法耗时可以忽略不记）。



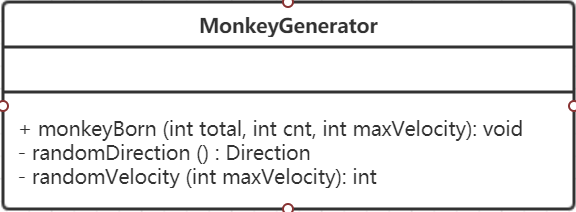
如果有满足条件的梯子，返回梯子，否则返回null。

### 策略3（可选）

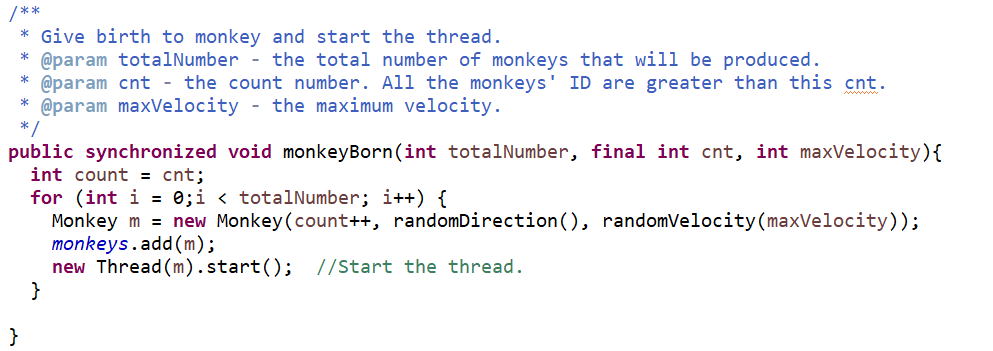
策略三则是等待性策略：优先选择没有猴子的梯子，若所有梯子上都有猴子，则在岸边等待，直到某个梯子空闲出来。



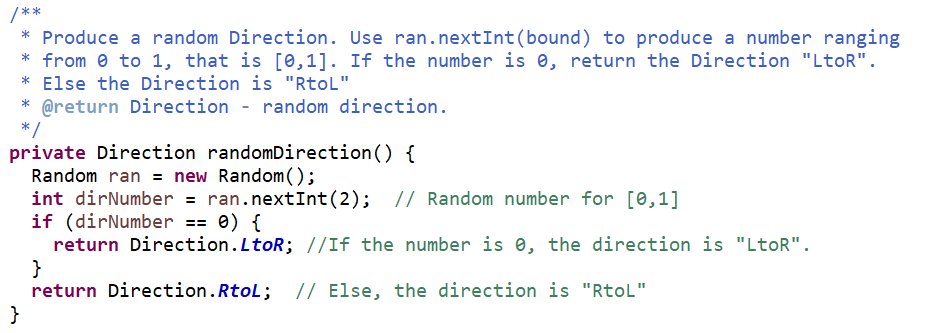
## “猴子生成器”MonkeyGenerator



通过输入的总数，当前猴子计时和猴子的最大速度来生成猴子，基本思路为构造好一个猴子就启动该猴子的线程。

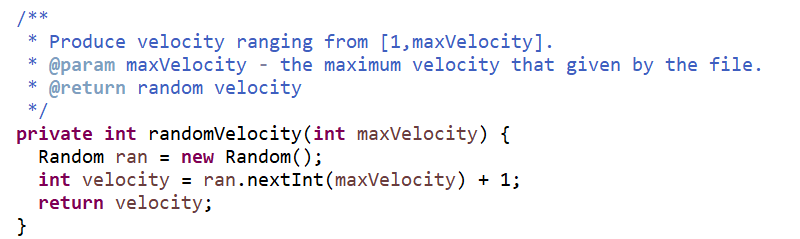


其中主要的辅助方法有随机生成速度和随机生成方向的两个。

随机生成方向：

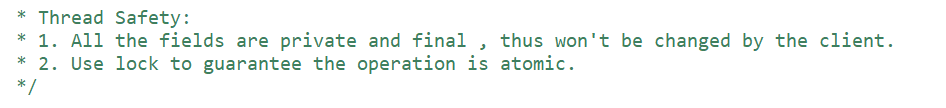
即通过随机生成数字1或者2来判断，若1则为从左到右的方向，否则为从右到左，返回。

随机生成速度：



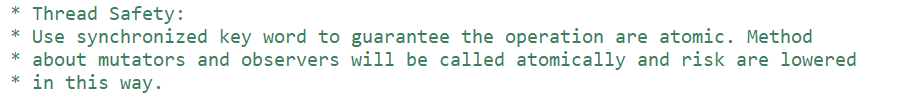
## 如何确保threadsafe？

1. 猴子类为Immutable类型



猴子的各个域均为private且重要的参数（ID，方向和速度等）都不可被修改，除了run方法，其他的方法均为observer，不会对猴子的属性产生更改。

1. Ladder方法：



使用synchronized关键字来确保各个方法均为原子操作：

**public** **synchronized** **boolean** getRungLtoR(**int** index)

**public** **synchronized** **boolean** getRungRtoL(**int** index)

**public** **synchronized** **int** getRungLength()

**public** **synchronized** **int** getLadderIndex()

**public** **synchronized** **void** setMinSpeed(**int** speed)

**public** **synchronized** **int** getMinSpeed()

**public** **synchronized** **void** setDirection(Direction inputDirection)

**public** **synchronized** Direction getDirection()

**public** **synchronized** **void** resetDirection()

**public** **synchronized** **boolean** jumpOn(Monkey monkey)

**public** **synchronized** **void** jumpOff(Monkey monkey)

**public** **synchronized** **boolean** monkeyMove(Monkey monkey, **int** position)

**public** **synchronized** **int** currentPosition(Monkey monkey)

**private** **synchronized** **void** renewSpeed()

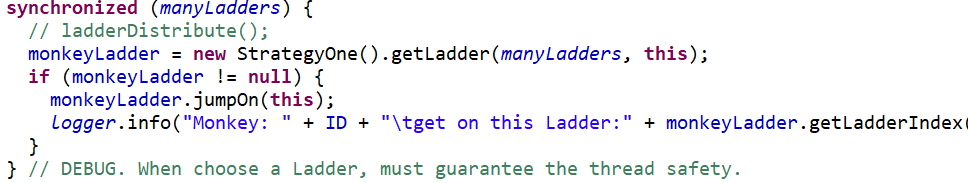
**public** **synchronized** Map<Integer, Monkey> getMonkeyCondition()

**public** **synchronized** **boolean** isNoMonkey()

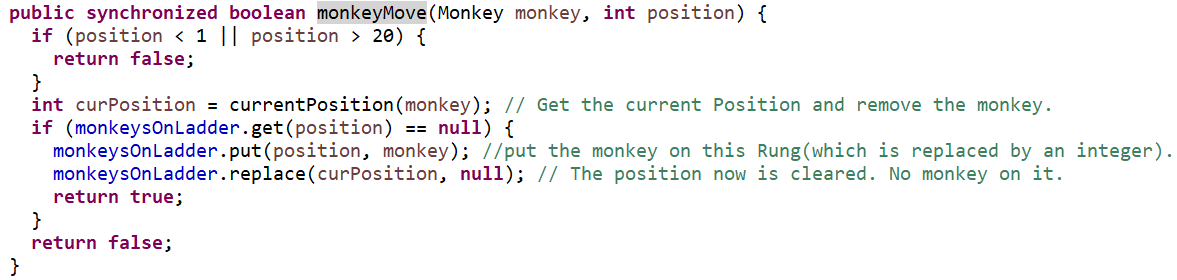
1. 加锁：

在猴子run的过程中，采用加锁的方式来确保线程安全性，加的锁为所有梯子的一个List<Ladder>

获取梯子时加锁：



1. 梯子关键方法确保正确性：



在Debug之前发现梯子上的猴子会消失，后来发现是因为在其他的猴子移动中会用replace(position,null)清空原来的猴子，导致线程不安全。采用if判断来决定能否成功执行移动的策略。

## 系统吞吐率和公平性的度量方案

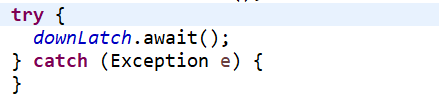
1. 吞吐量：
2. 客户端中timeConsumption记录总时长消耗：



1. 从第一个猴子开始run计时：



1. 系统等待执行完成



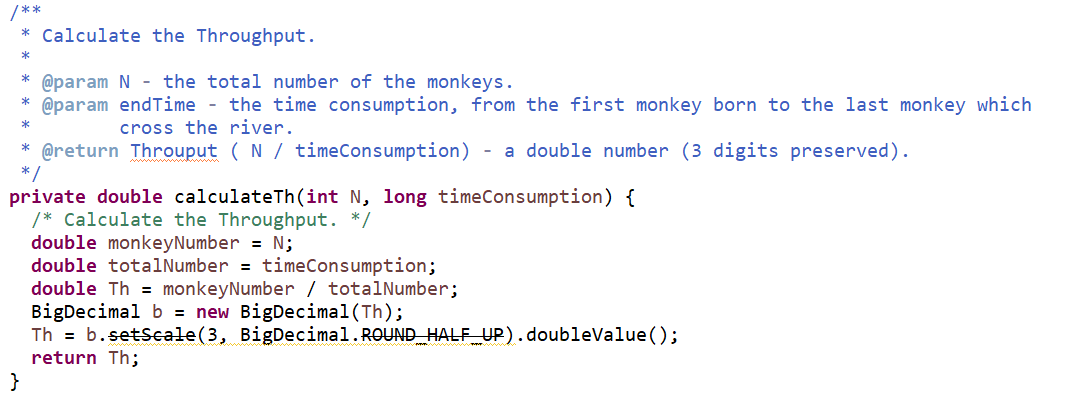
1. 所有线程终止后可以获得当前时间：



1. 计算得知时间总消耗：

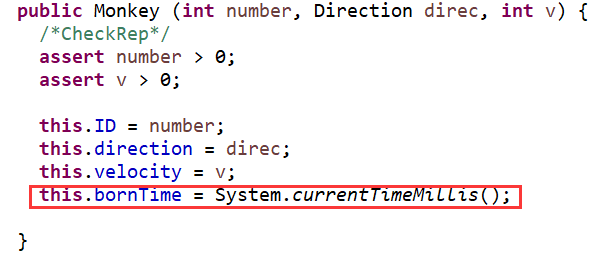


1. 根据N和timeConsumption来计算吞吐率



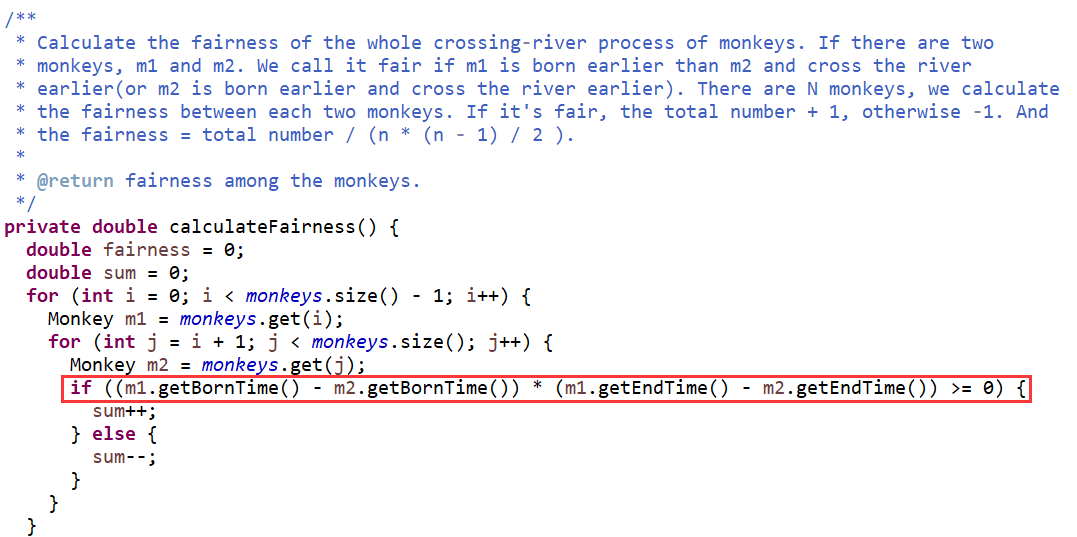
最后保留三位小数

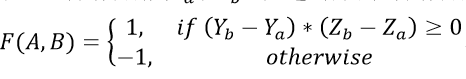
1. 计算公平性：
2. 猴子的出生时间和结束时间的测量：
3. 猴子开始时将记录出生时间，过河结束时记录过河时间



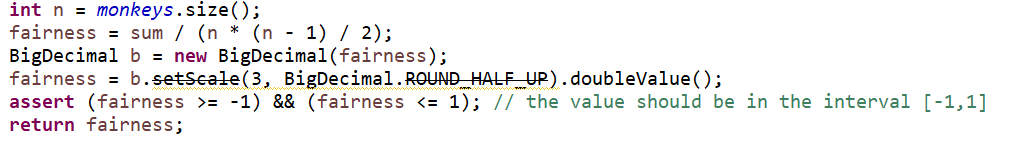


1. 每两两猴子之间遍历比较（不能重复）

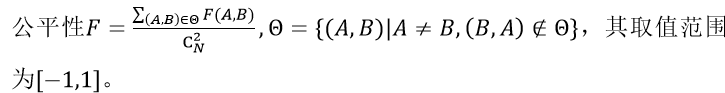


根据公式来计算。

1. 统计数据并计算

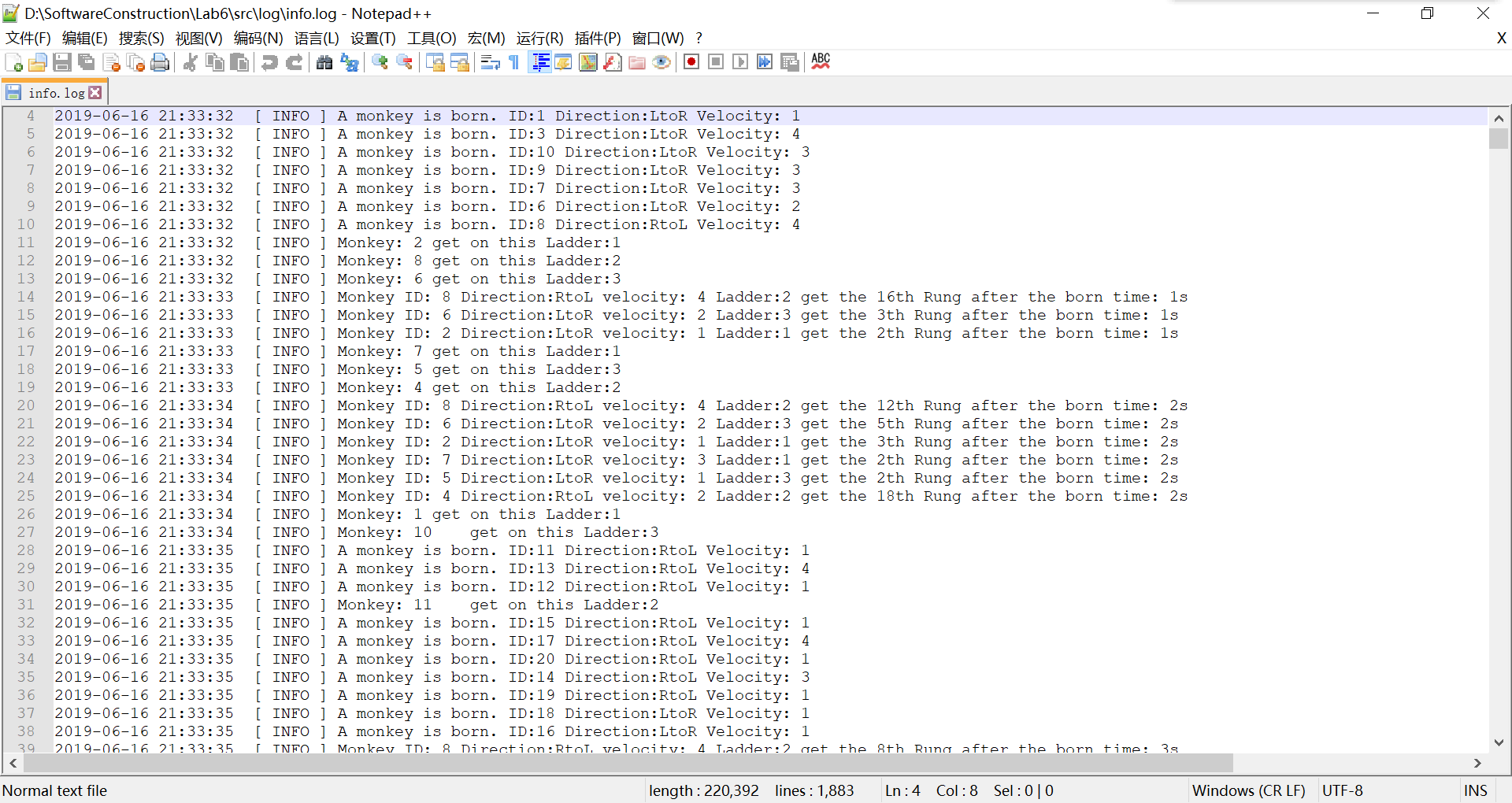


保留三位小数，根据公式来计算：



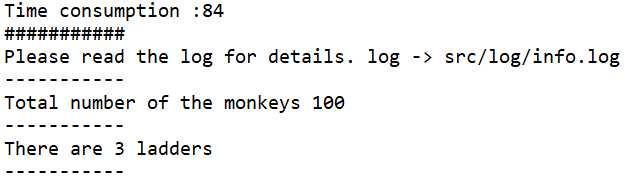
## 输出方案设计

日志 ：



GUI：

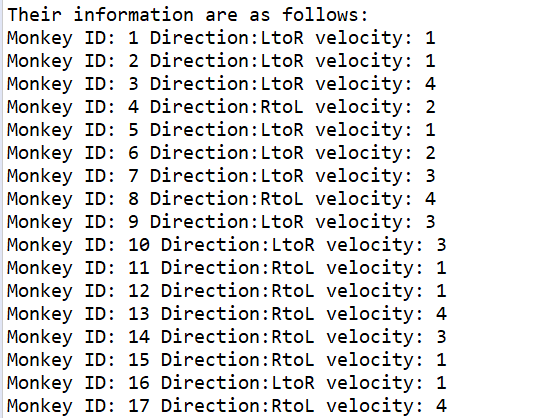
1. 记录各参数：



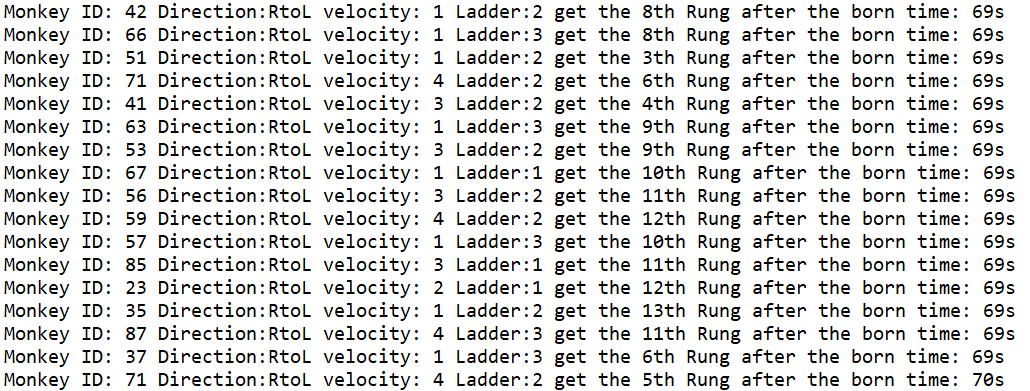
1. 记录log路径：



1. 猴子的各项信息：



1. 猴子过河的各项细节：



1. 过河成功的提示：



以及时间记录：

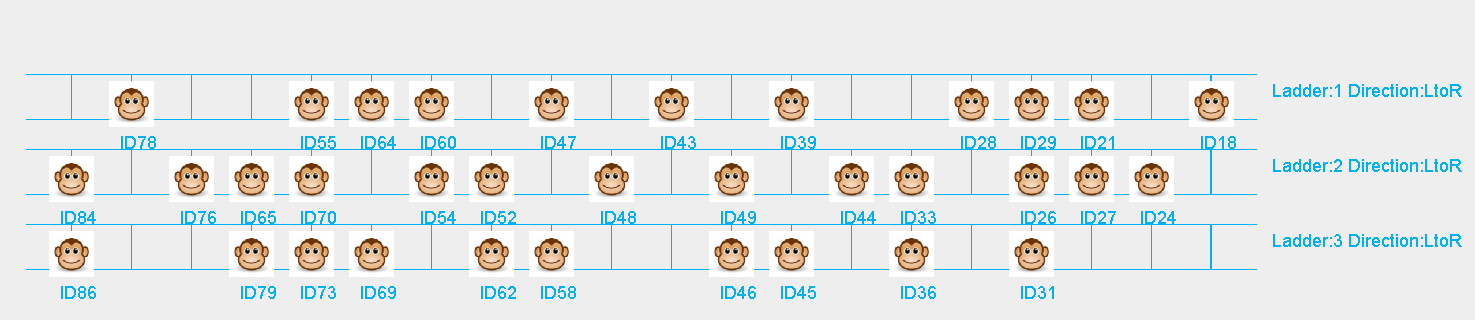
1. 所有猴子过河后得到的提示：



和时间消耗。

可视化（可选）：

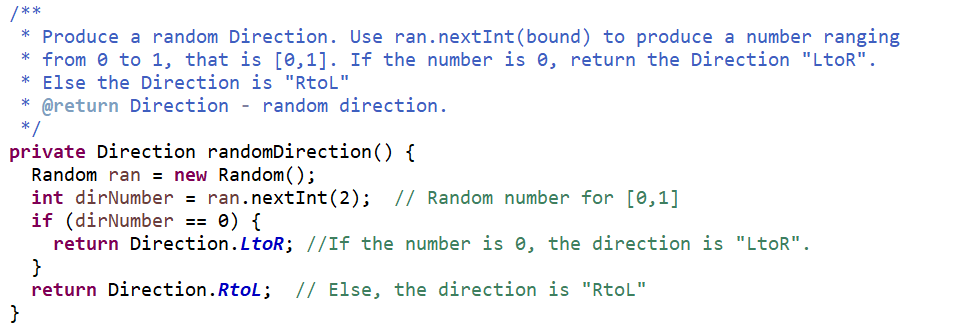
可视化有梯子，梯子的各个踏板，梯子编号和猴子动态运动过程



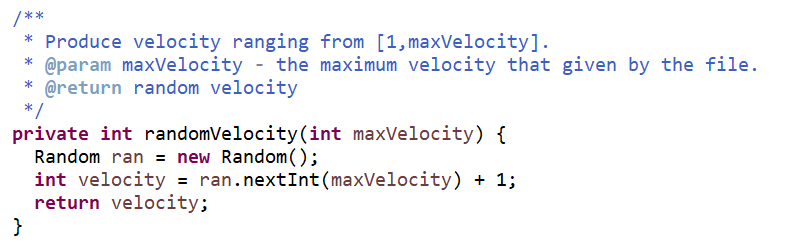
## 猴子过河模拟器v1

### 参数如何初始化

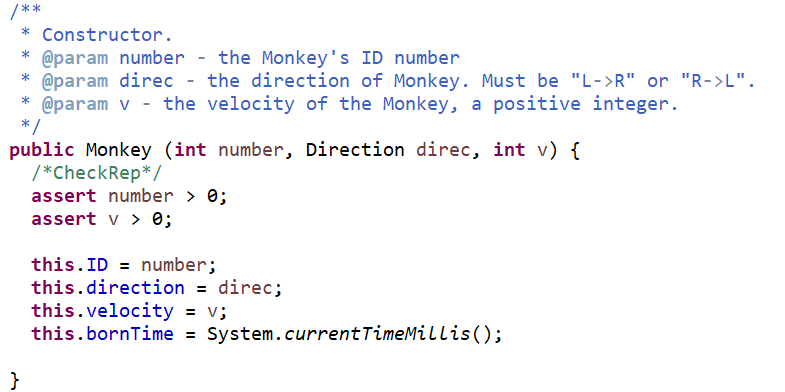
通过随机生成器：



和速度生成器：



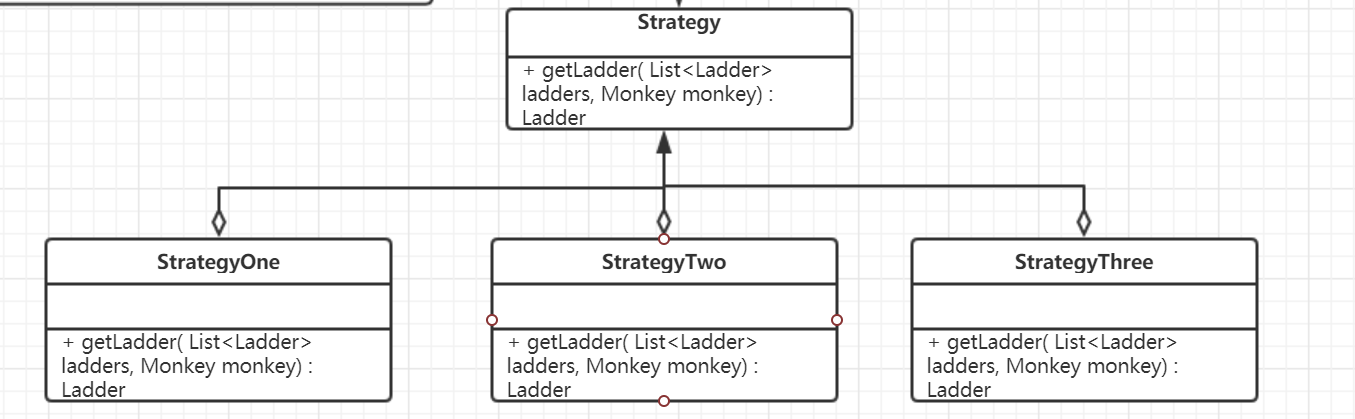
传入猴子的构造器当中：



来循环生成。

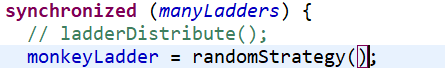
### 使用Strategy模式为每只猴子选择决策策略

（1） UML 类图

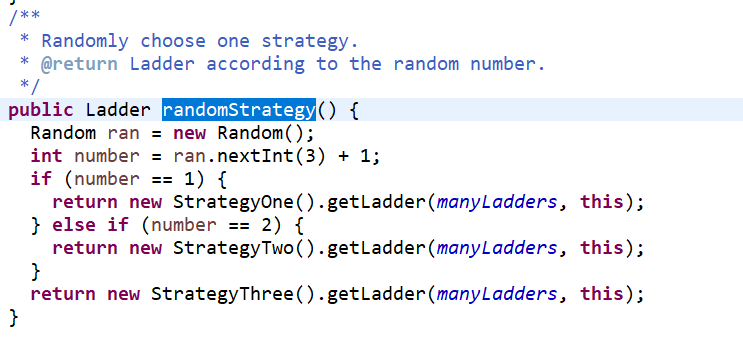


具体语境类为猴子：

（2） 猴子在run的过程中随机选择梯子：



（3） 随机梯子生成：

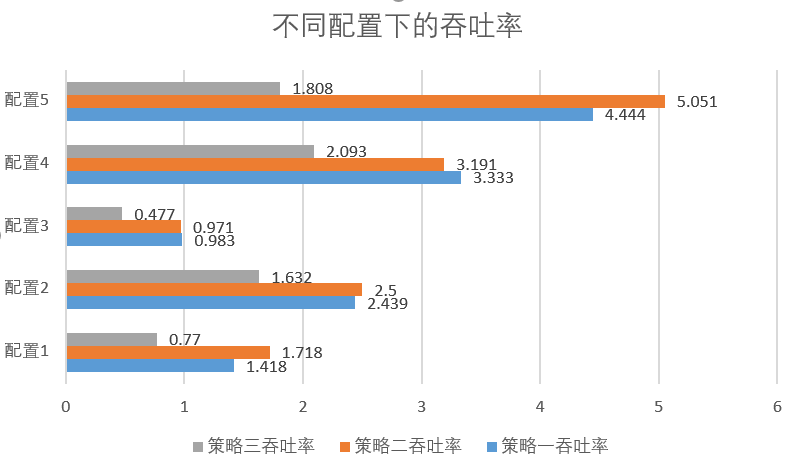


（4）该梯子是基于共享数据manyLadders的基础上来被选择的。并且加锁来确保线程的安全性。

## 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



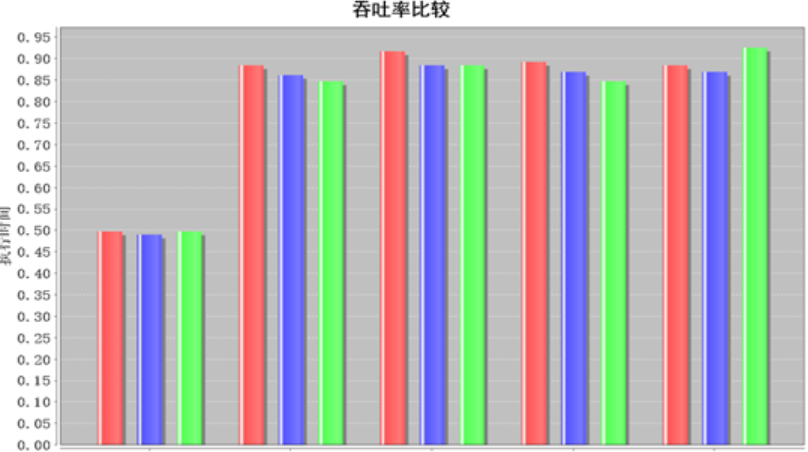
不同的配置文件：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | n | h | t | N | k | MV |
| 配置一 | 4 | 20 | 3 | 190 | 5 | 5 |
| 配置二 | 7 | 20 | 2 | 100 | 20 | 8 |
| 配置三 | 10 | 20 | 1 | 1000 | 1 | 10 |
| 配置四 | 7 | 20 | 4 | 300 | 30 | 10 |
| 配置五 | 10 | 20 | 5 | 1000 | 30 | 5 |

### 对比分析：变化某个参数，固定其他参数

1. 变化n

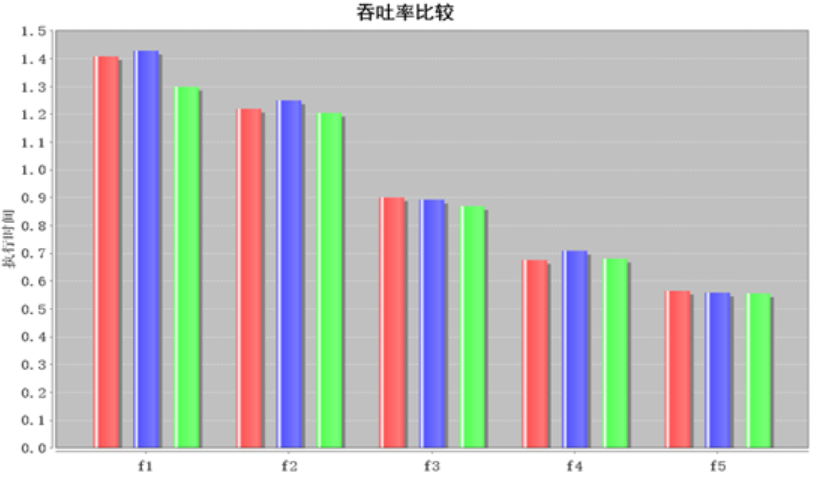
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 参数 | n | h | t | N | k | MV |
| 取值 | 1-5 | 20 | 5 | 100 | 3 | 6 |



随着梯子数目n的增加，首先从n=1变化到n=2吞吐率得到很大提高，在n=2到n=5的变化中，吞吐率变化不大。对于公平性，规律类似。说明三种策略对于增加梯子的利用效率不高。与具体算法策略关系不大。

1. 变化t

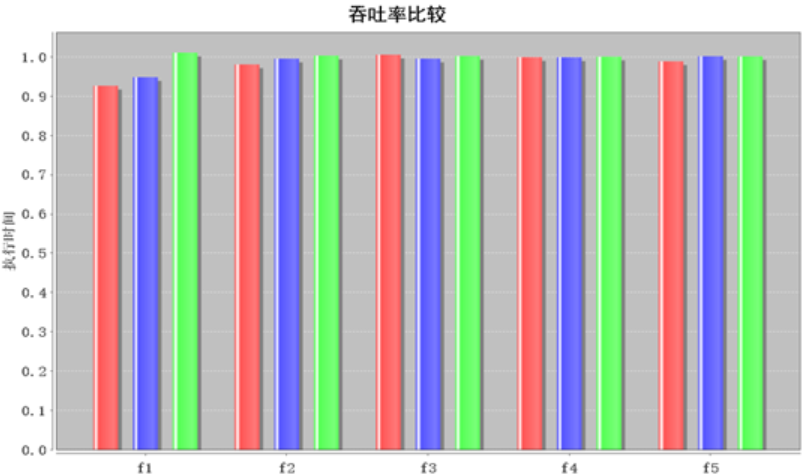
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 参数 | n | h | t | N | k | MV |
| 取值 | 4 | 20 | 1-5 | 100 | 3 | 8 |



随着生猴子间隔时间t的增加，吞吐率总体降低，公平性总体提高。生猴子频率太低密度太小，不能很好利用梯子。与具体算法策略关系不大。

1. 变化N

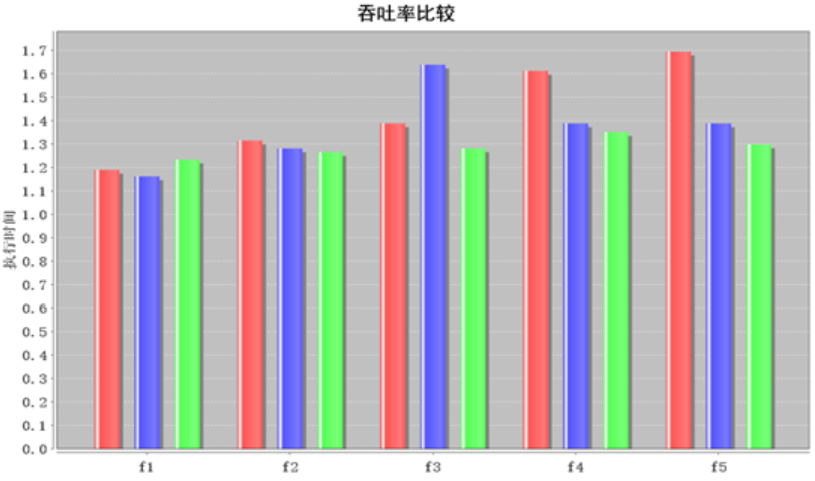
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 参数 | n | h | t | N | k | MV |
| 取值 | 4 | 20 | 5 | 200，400，600，800，  1000 | 3 | 10 |



随着生猴子总数的上升，吞吐率与公平性整体变化不大。当生猴子总数非常大的时候，主要是生猴子的频率与密度在起着影响。与具体算法策略关系不大。策略三始终较低。

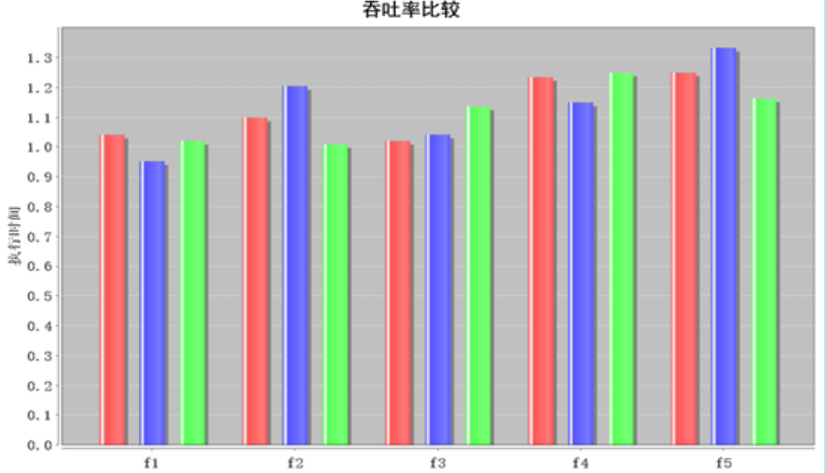
1. 变化k

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 参数 | n | h | t | N | k | MV |
| 取值 | 4 | 20 | 5 | 200 | 6，12，18…48 | 10 |

 随着每轮生的猴子的数目的增加，吞吐率整体上升，公平性整体下降。猴子密度提上去，有效利用了梯子。

1. 变化MV

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 参数 | n | h | t | N | k | MV |
| 取值 | 4 | 20 | 5 | 200 | 6 | 6，7，8，9，10 |



随着最大速度的提高，吞吐率整体上升，公平性规律不明显。速度提高了，自然总体跑得快了。算法比较总体规律不明显。

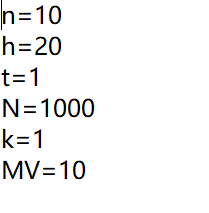
### 分析：吞吐率是否与各参数/决策策略有相关性？

具有一定的相关性。在多个猴子速度差异非常大的情况下，优先选择行进速度大的策略将大大提高性能，在其他压力测试下看出策略三，因为策略三采取等待措施——直到梯子为空才能上梯子，大大降低了吞吐率。

### 压力测试结果与分析

1. 猴子特别多：

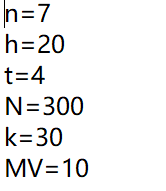
配置三文件：



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 策略一吞吐 | 策略一公平 | 策略二吞吐 | 策略二公平 | 策略三吞吐 | 策略三公平 |
| 0.983 | 0.992 | 0.971 | 0.992 | 0.477 | 0.855 |

吞吐率低时公平性较高

1. 速度差异特别大



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 策略一吞吐 | 策略一公平 | 策略二吞吐 | 策略二公平 | 策略三吞吐 | 策略三公平 |
| 2.439 | 0.467 | 2.5 | 0.609 | 1.632 | 0.734 |

吞吐率高，公平性较低。

## 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。

|  |  |  |
| --- | --- | --- |
|  | 吞吐率 | 公平性 |
| Competiton\_1.txt |  |  |
| 第1次模拟 | 2.564 | 0.414 |
| 第2次模拟 | 2.564 | 0.563 |
| 第3次模拟 | 2.521 | 0.656 |
| 第4次模拟 | 2.564 | 0.688 |
| 第5次模拟 | 2.500 | 0.644 |
| 第6次模拟 | 2.459 | 0.519 |
| 第7次模拟 | 2.586 | 0.547 |
| 第8次模拟 | 2.609 | 0.655 |
| 第9次模拟 | 2.542 | 0.611 |
| 第10次模拟 | 2.609 | 0.689 |
| 平均值 | 2.551 | 0.598 |
| Competiton\_2.txt |  |  |
| 第1次模拟 | 5.208 | 0.527 |
| 第2次模拟 | 5.051 | 0.640 |
| 第3次模拟 | 5.618 | 0.707 |
| 第4次模拟 | 5.682 | 0.701 |
| 第5次模拟 | 4.902 | 0.700 |
| 第6次模拟 | 5.051 | 0.642 |
| 第7次模拟 | 5.102 | 0.556 |
| 第8次模拟 | 4.950 | 0.594 |
| 第9次模拟 | 5.319 | 0.720 |
| 第10次模拟 | 5.682 | 0.775 |
| 平均值 | 5.256 | 0.562 |
| Competiton\_3.txt |  |  |
| 第1次模拟 | 1.087 | 0.602 |
| 第2次模拟 | 1.25 | 0.707 |
| 第3次模拟 | 1.19 | 0.664 |
| 第4次模拟 | 1.075 | 0.425 |
| 第5次模拟 | 1.19 | 0.692 |
| 第6次模拟 | 1.22 | 0.772 |
| 第7次模拟 | 1.111 | 0.518 |
| 第8次模拟 | 1.124 | 0.511 |
| 第9次模拟 | 1.087 | 0.525 |
| 第10次模拟 | 1.163 | 0.745 |
| 平均值 |  |  |

# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 6.5 | 8:30 – 9:45 | 建立仓库，了解实验基本思路 | 完成 |
| 6.6 | 9:15 - 11:15 | 完成Ladder，Monkey和Rung的设计 | 未完成 |
| 6.7 | 9:30 – 11:30 | 继续完善Ladder的测试样例 | 提前一小时完成 |
| 6.7 | 9:30 – 11:30 | 进行Monkey的ADT设计 | 未完成 |
| 6.9 | 10:30 - 11:30 | 重新完善ADT相应的域 | 完成 |
| 6.10 | 9:00 – 11:30 | 完善Ladder中的方法 | 完成 |
| 6.10 | 14:30 – 16:00 | 完善Ladder中的测试样例 | 完成 |
| 6.10 | 22:00 – 23:30 | 完成三种策略的设计 | 完成 |
| 6.11 | 8:40 – 22:00 | 完成三种策略的测试 | 完成 |
| 6.12 | 8:30 – 11:30 | 完成猴子策略修改，debug | 完成 |
| 6.12 | 14:00 – 17:30 | 加锁，debug | 未完成 |
| 6.13 | 9:00 – 11:40 | 对多线程启动debug | 完成 |
| 6.13 | 18:30 – 22:00 | 可视化 | 未完成 |
| 6.14 | 8:30 – 11:00 | 可视化 | 完成 |
| 6.15 | 9:00 – 11:40 | v1完善 | 未完成 |
| 6.16 | 9:00 – 11:40 | v1可视化和GUI完善 | 完成 |
| 6.16 | 14:00 – 16:00 | v2实验数据比较和测量 | 完成 |
| 6.16 | 18:00 – 22:30 | 实验报告撰写和其他完善 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 对Ladder设计，不知道怎么将其设置为mutable类型。 | 仔细查看手册，发现题目中设置参数为 “h = 20”, 于是在ADT设计中设置一个成员变量为List类型，存放Rung（每个踏板），并且只在构造器中对这个List进行修改，可以进行其他方法修改，使得梯子类型为mutable。 |
| 随机选择方向等，不知道如何随机生成 | 查询得知，可以利用enum枚举，然后根据随机生成的序号来访问enum中的元素。 |
| 三种策略中测试不过关，没办法随机生成，Test结果不正确。 | 发现是static 静态变量初始化的问题。 |
| 出现竞争，但不知怎么解决。 | 。  后来发现是加锁问题。并且发现：线程执行同步代码块或同步方法时，程序调用Thread.sleep(Long l)、Thread.yield()方法暂停当前线程的执行，不释放锁. |
| 选择梯子时出现竞争，没有在正确的地方加锁和解锁以至于程序串行而非并行。 | 1. 发现20只猴子过河时总有几只猴子会在最后的时间内同时获得一个梯子，然后出现竞争。 2. 用manyladders加锁 3. 发现在monkeyMove方法中会让已经爬上梯子的猴子消失，于是加上限定条件，只有该位置上没有猴子的时候才能让此猴子爬上梯子。 |
| 可视化过程中无论如何线程也不终止 | 1. 一个个线程进行打印，判断返回情况 2. 排除Monkey线程，发现是可视化的线程有问题。 3. 在客户端利用Terminate按钮可以终止进程 4. 重新优化GUI界面 |
|  |  |

# 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

1. 尽量限制mutable的数据类型范围，这样有助于线程安全。
2. 当一个线程Thread.sleep()时，并不意味着释放了所加的锁，而要通过其他的操作来释放锁。
3. 可视化可以帮助debug，在猴子过河的线程中可以利用动画信息来帮助查看猴子运动是否正常。
4. git分支一定要merge，不然出来的network图不符合要求，而且在合并前一定要提交。git的分支操作相当于提供了变相的备份操作。
5. 在merge之前如果master和分支都进行了push修改的话，会出现分叉结构。
6. 多线程可以提高程序的并行能力，但是有一定的风险，并且要确保数据共享中的原子操作性，否则将产生意外的结果。

## 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？

答：复杂在线程管理和确保线程的安全性。体会到了，提高了计算性能。

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？

答：synchronized关键字修饰和加锁。根据多组数据模拟来确定折中选择。

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？

答：遇到过，发现在某个时刻猴子会消失。通过可视化和代码审查来派出错误的。

1. 关于本实验的工作量、难度、deadline。

答：Demanding but rewarding.

1. 到此为止你对《软件构造》课程的意见和建议。

答：有时候不知道自己的方向，不知道如何设计。

1. 还有一周就要期末考试了，你准备如何复习？

答：复习材料，整理知识图谱。