

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 6实验报告**

|  |  |
| --- | --- |
| 姓名 | 肖锐 |
| 学号 | 1162100430 |
| 班号 | 1603004 |
| 电子邮件 | [hardworkrui@163.com](mailto:hardworkrui@163.com) |
| 手机号码 | 18846185997 |

**目录**

[1 实验目标概述 1](#_Toc514162176)

[2 实验环境配置 1](#_Toc514162177)

[3 实验过程 1](#_Toc514162178)

[3.1 ADT设计方案 1](#_Toc514162179)

[3.2 Monkey线程的run()的执行流程图 1](#_Toc514162180)

[3.3 至少两种“梯子选择”策略的设计与实现方案 2](#_Toc514162181)

[3.3.1 策略1 2](#_Toc514162182)

[3.3.2 策略2 2](#_Toc514162183)

[3.3.3 策略3（可选） 2](#_Toc514162184)

[3.4 “猴子生成器”MonkeyGenerator 2](#_Toc514162185)

[3.5 如何确保threadsafe？ 2](#_Toc514162186)

[3.6 系统吞吐率和公平性的度量方案 2](#_Toc514162187)

[3.7 输出方案设计 2](#_Toc514162188)

[3.8 猴子过河模拟器v1 2](#_Toc514162189)

[3.8.1 参数如何初始化 2](#_Toc514162190)

[3.8.2 使用Strategy模式为每只猴子随机选择决策策略 2](#_Toc514162191)

[3.9 猴子过河模拟器v2 2](#_Toc514162192)

[3.9.1 对比分析：固定其他参数，选择不同的决策策略 3](#_Toc514162193)

[3.9.2 对比分析：变化某个参数，固定其他参数 3](#_Toc514162194)

[3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？ 3](#_Toc514162195)

[3.9.4 压力测试结果与分析 3](#_Toc514162196)

[4 实验进度记录 3](#_Toc514162197)

[5 实验过程中遇到的困难与解决途径 3](#_Toc514162198)

[6 实验过程中收获的经验、教训、感想 3](#_Toc514162199)

# 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 和运行性能之间寻求较优的折中，为此计

算吞吐率等性能指标，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

 基本的 GUI 编程

# 实验环境配置

在这里给出你的GitHub Lab6仓库的URL地址（Lab6-学号）。

<https://github.com/ComputerScienceHIT/Lab6-1162100430.git>

# 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；

1. 梯子的ADT：Ladder

作用：为了描述过河程序中的梯子这个对象

属性：

1判断20个阶梯上是否有猴子的一个boolean类型的数组boolean[] b；

2桥上猴子的方向 String dire;

3桥上的阶梯对象ArrayList<Stage> stage；

方法：

/\*\*

\* 某个猴子第一次踏上梯子的第一个踏板

\* @param 某个猴子m

\*/

1 public void firstlocation(Monkey m)

此方法是让猴子踏上第一个踏板，相应的，将梯子的踏板1的数组b[1]设置成true，将踏板对象1也设置成true。然后在猴子对象里面将阶梯号1设置进去。

dire=m.getdirection();

b[1]=true;

stage.get(1).setmonkey(true);

m.setposition(1);

/\*\*

\* 设置梯子的阶梯i上有猴子

\* @param 阶梯号i

\*/

2 public void setmonkeyi(int i)

此方法是将阶梯号为i的阶梯设置为有猴子的状态

b[i] = true;

stage.get(i).setmonkey(true);

/\*\*

\* 清除阶梯i上面的猴子

\* @param 阶梯号i

\*/

3 public void clearmonkeyi(int i)

此方法是让阶梯号为i上的有猴子状态转换成没有猴子的状态

b[i] = false;

stage.get(i).setmonkey(false);

/\*\*

\* 返回阶梯号为i的阶梯对象

\* @param 阶梯号i

\* @return 阶梯号为i的对象

\*/

4 public Stage getstagei(int i)

此方法是获取阶梯号为i的阶梯对象

/\*\*

\* 判断梯子上的第i个阶梯是否有猴子

\* @param 当前梯子的阶梯号i

\* @return 阶梯号i上是否存在猴子

\*/

5 public boolean getexistmonkey(int i)

此方法是获取阶梯i上是否有猴子

/\*\*

\* 获取梯子上猴子的方向

\* @return 梯子上猴子的方向

\*/

6 public String getdirection()

此方法是获取梯子的方向

/\*\*

\* 判断梯子上是否存在猴子

\* @return 存在的话返回true，否则返回false

\*/

7 public boolean existmonkey()

此方法是获取桥上是否存在猴子，如果存在，返回true，否则，返回false

/\*\*

\* 判断桥上猴子m的速度以内的范围是否存在猴子

\* @param 猴子的速度范围x

\* @param 猴子对象m

\* @return 如果猴子m前面的速度x范围内有另外的猴子对象，返回true，否则返回false

\*/

8 public boolean overx(int x,Monkey m)

此方法是判断桥上的猴子m前面是否有猴子

1. 猴子的ADT：Monkey

属性：  
猴子的编号int ID，猴子的速度int v，猴子的方向String direction，

猴子在桥的编号int position;猴子在桥的踏板编号int ladder

方法：

/\*\*

\* 返回猴子的id值

\* @return 猴子的id值

\*/

public int getid()

获取猴子的编号ID

/\*\*

\* 返回猴子的速度

\* @return 猴子的速度v

\*/

public int getv()

获取猴子的速度v

/\*\*

\* 返回猴子的方向

\* @return 猴子的方向dir

\*/

public String getdirection()

获取猴子的方向

/\*\*

\* 设置猴子在桥上的位置

\* @param 桥上的位置i 1=<i<=20

\*/

public void setposition(int i)

设置猴子在桥上的i踏板上

/\*\*

\* 返回猴子在桥上的位置

\* @return 猴子的位置position

\*/

public int getposition()

获取猴子在桥上的踏板位置i

/\*\*

\* 设置猴子在哪一个桥上

\* @param 桥的标号i 1=<i<=5

\*/

public void setladderi(int i)

设置猴子在哪一个桥上

/\*\*

\* 返回猴子所在桥的标号

\* @return 桥的标号ladderi

\*/

public int getladderi()

获取猴子所在桥的编号

1. 阶梯的ADT：Stage

属性：

踏板上是否有猴子的标志boolean existmonkey；

方法

/\*\*

\* 设置踏板上是否有无猴子

\* @param 如果有猴子，a为true，否则，a为false

\*/

public void setmonkey(boolean a)

设置踏板上的猴子的存在状态，若有猴子为true，否则为false

/\*\*

\* 获取踏板上的状态

\* @return 是否存在猴子，若存在，返回true，否则，返回false

\*/

public boolean getexistmonkey()

获取当前踏板上是否存在猴子，若存在，返回true，否则，返回false

/\*\*

\* 将踏板的状态设置成无猴子的状态

\*/

public void clearmonkey()

此方法是将踏板的状态设置成无猴子的状态

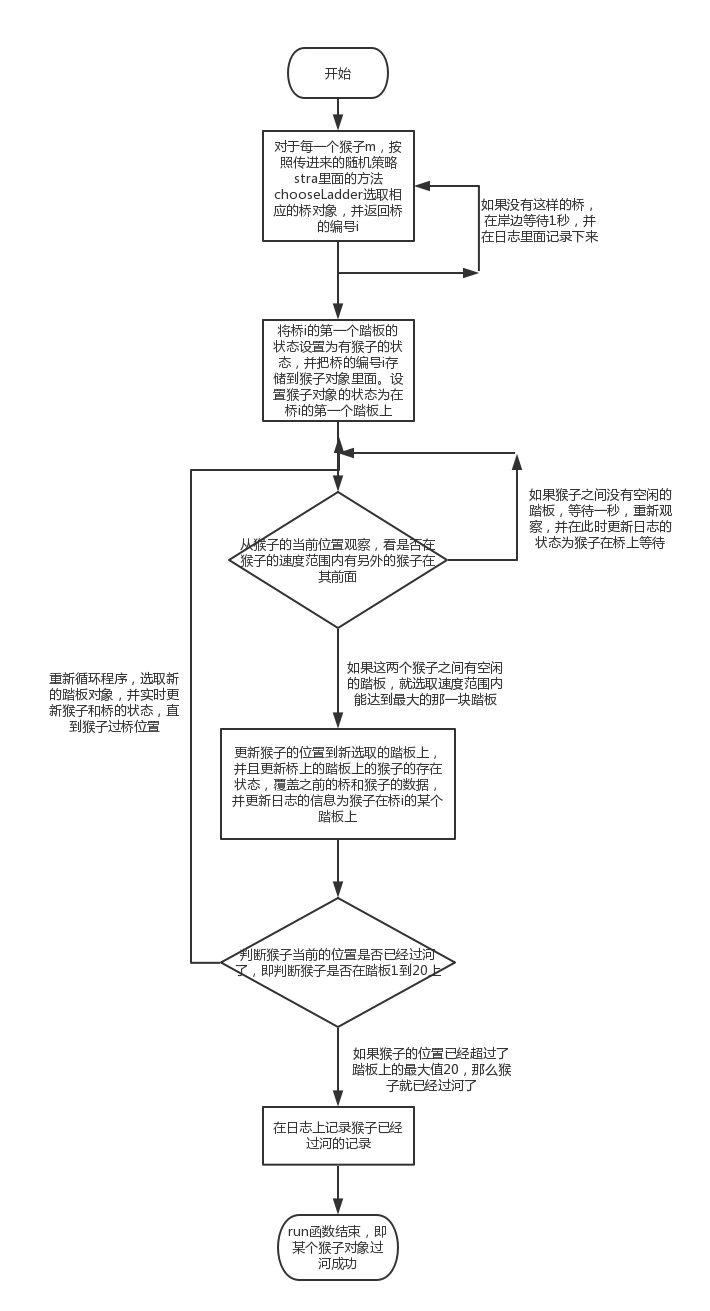
2．

（可选）以类图形式给出多个类之间的关系。

## Monkey线程的run()的执行流程图

这里无需考虑具体采用的梯子选择策略。

如下图所示：



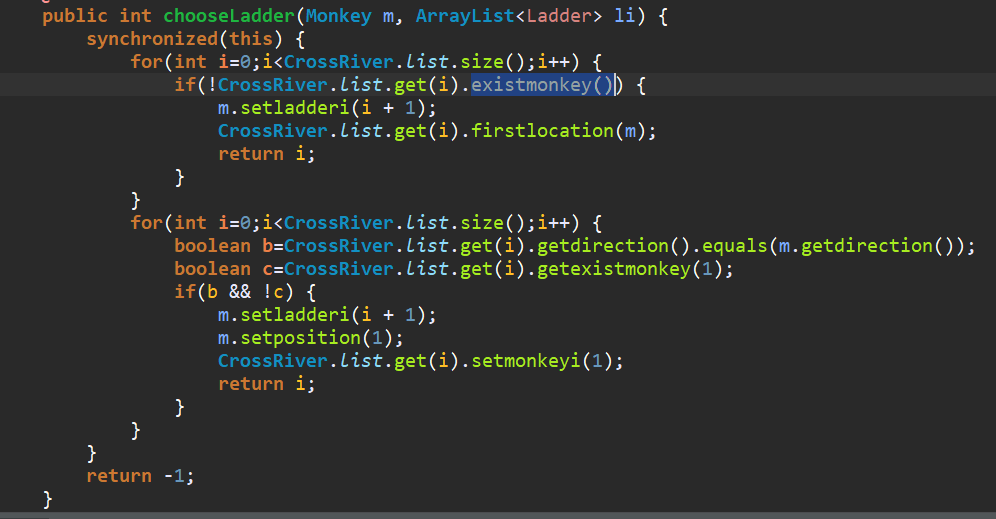
## 至少两种“梯子选择”策略的设计与实现方案

### 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；实现方案：

如图所示：



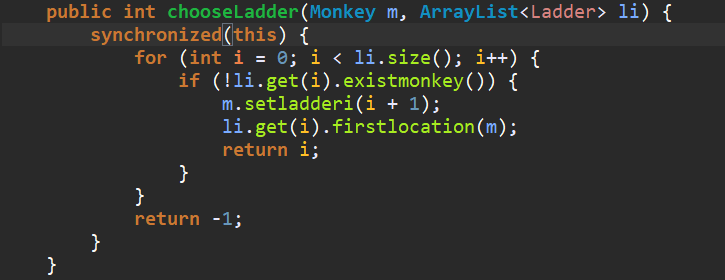
首先将创建的所有的桥对象存储到一个static变量的list集合里面。然后遍历所有的桥对象，利用上述的Ladder对象里面的existmonkey()方法判断每一个桥上面是否存在猴子，如果存在猴子的话，就返回此桥的编号。当遍历完所有的Ladder对象后，发现没有这样的桥的话，就继续判断是否有桥上的猴子的方向和自身猴子对象的方向一致，并且这座桥上的第一个踏板上没有猴子，如果存在的话，就返回桥的编号i。否则，就不存在上述的两种情况的桥，那么猴子就只有在岸边等待，直到有上述两种情况的桥出现为止。此时就返回-1，然后在调用程序里面，利用如下的方法，就可以实现等待的的目的



### 策略2

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

如图所示：



遍历所有的梯子对象，判断梯子上是否存在猴子，如果没有猴子的话就选择这个梯子，并将阶梯和猴子初始化在第一个踏板的位置。否则，就在岸边等待，直到有空闲的梯子出来为止。如图所示：

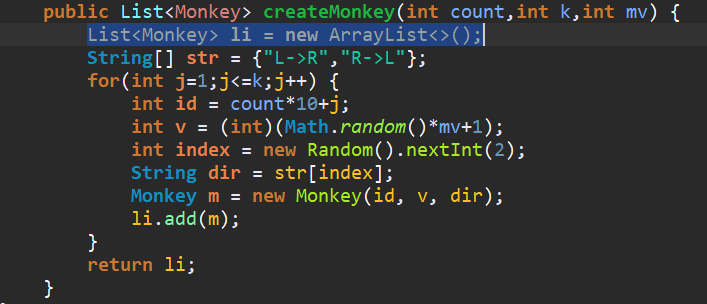


### 策略3（可选）

## “猴子生成器”MonkeyGenerator

如何设计和实现。

如图所示：



在MonkeyGenerator 类里面使用createMonkey(int count,int k,int mv)方法来创建猴子对象，其中count参数是第几次产生的猴子，k是每次需要创建的猴子的数量，mv是速度的最大值5.

首先设计一个集合用来存储每次产生的所有猴子，

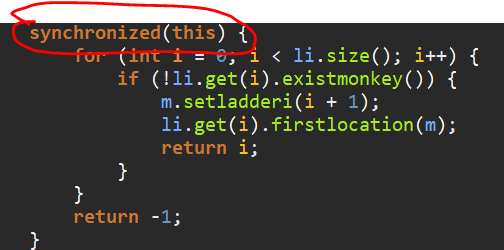
List<Monkey> li = new ArrayList<>();

然后用一个数组str来存储两个方向，以便于后面随机的选择一个方向，然后调用Math.random()就可以产生一个0.0 ~ 1.0之间的浮点数，我们用他来乘以最大速度mv再加一然后强制转换成int类型的一个数就可以把v随机赋值为1到5之间的一个数了。然后，利用new Random().nextInt(2)就可以随机产生一个0或者1的数index。我们就可以调用前面的数组str，利用str[index]就可以将方向随机的设置为由左到右或则是由右到左。然后利用id = count\*10+j;就可以产生同一个时间编号的前缀的所有猴子编号。然后创建猴子对象Monkey m = new Monkey(id, v, dir);将猴子对象加入到前面创建的list集合里面，返回即可。

## 如何确保threadsafe？

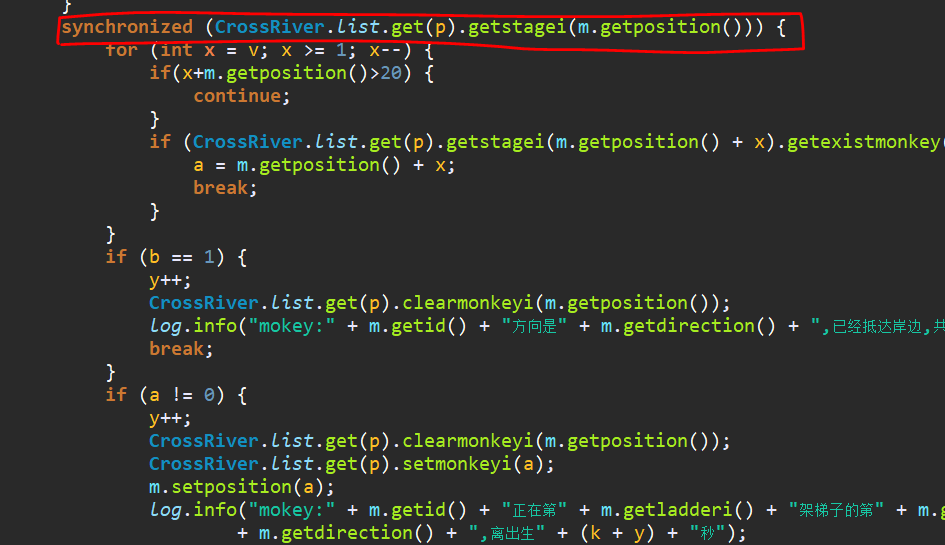
对于猴子在选取梯子的时候，由于所有梯子都是共享的数据对象，在处理的时候，我们需要设置里面的踏板的状态和猴子的状态，所以我们使用了关键字synchronized来锁住这个策略对象，以保证其他的猴子线程访问这个策略对象的时候，不会发生线程冲突的情况。如下：





而对于梯子里面的阶梯对象，它是被所有桥上的猴子共享的数据对象。因此在对它进行状态更新的时候，也要注意线程安全，避免多个线程发生数据的冲突。所以在进行过河线程时，也使用了关键字synchronized来锁住踏板这个对象，从而保证其他的猴子线程在对同一个踏板产生竞争的时候不会发生数据的冲突。

如下所示：



## 系统吞吐率和公平性的度量方案

吞吐率: 假如𝑁只猴子过河的总耗时为𝑇秒，那么每只猴子的平

均耗时为𝑋 =𝑇/𝑁秒，则吞吐率𝑇ℎ =𝑁/𝑇表征每秒钟可过河的猴子数目.

在这个程序中，以最后一个猴子过河的总时间加上此猴子的诞生时间离第一个猴子的诞生时间作为N个猴子的总时间。

例如，最后一个猴子的过河总时间为t1，然后再根据它的产生编号id就可以得到它离第一个猴子产生的时间t2，根据我的程序，我是将产生的第几次作为count再乘以10再编号得到的，因此，利用id/10就可以得到它离第一批猴子产生所化的时间了。所以，t2=id/10+t1.

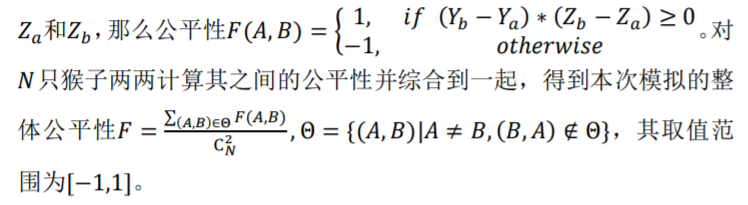
就可以用N/t2来求吞吐率了

公平性：度量方法如下：

如果 Monkey 对象 A 比 Monkey B 出生得更早，那么

A 应该比 B 更早抵达对岸，则为“公平”；若 A 比 B 晚到对岸，则为“不

公平”。设 A 和 B 的产生时间分别为𝑌𝑎和𝑌𝑏，抵达对岸的时间分别为

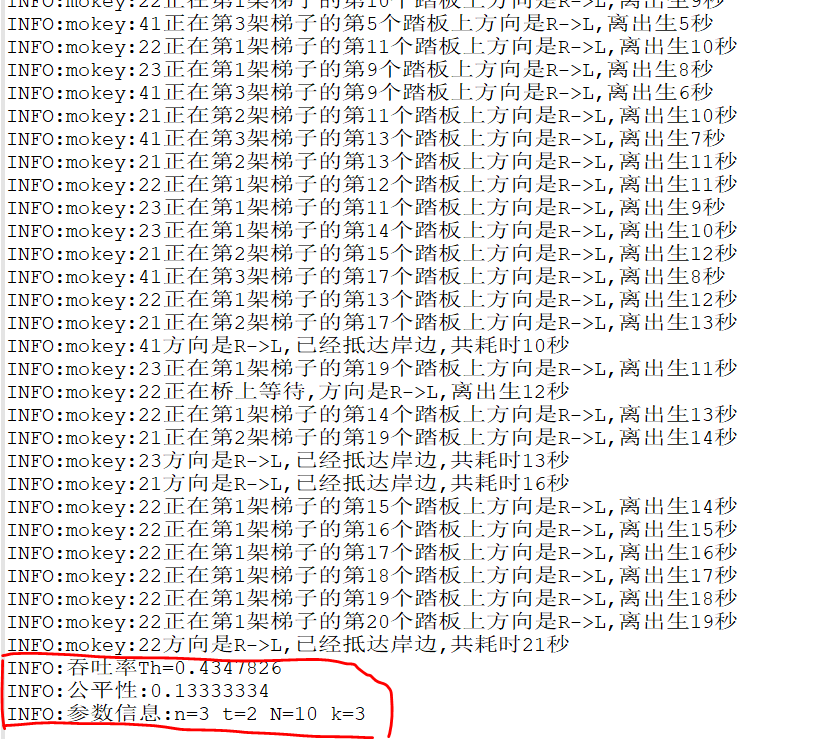


如图所示：

输入数据如下：



输出的日志结果：



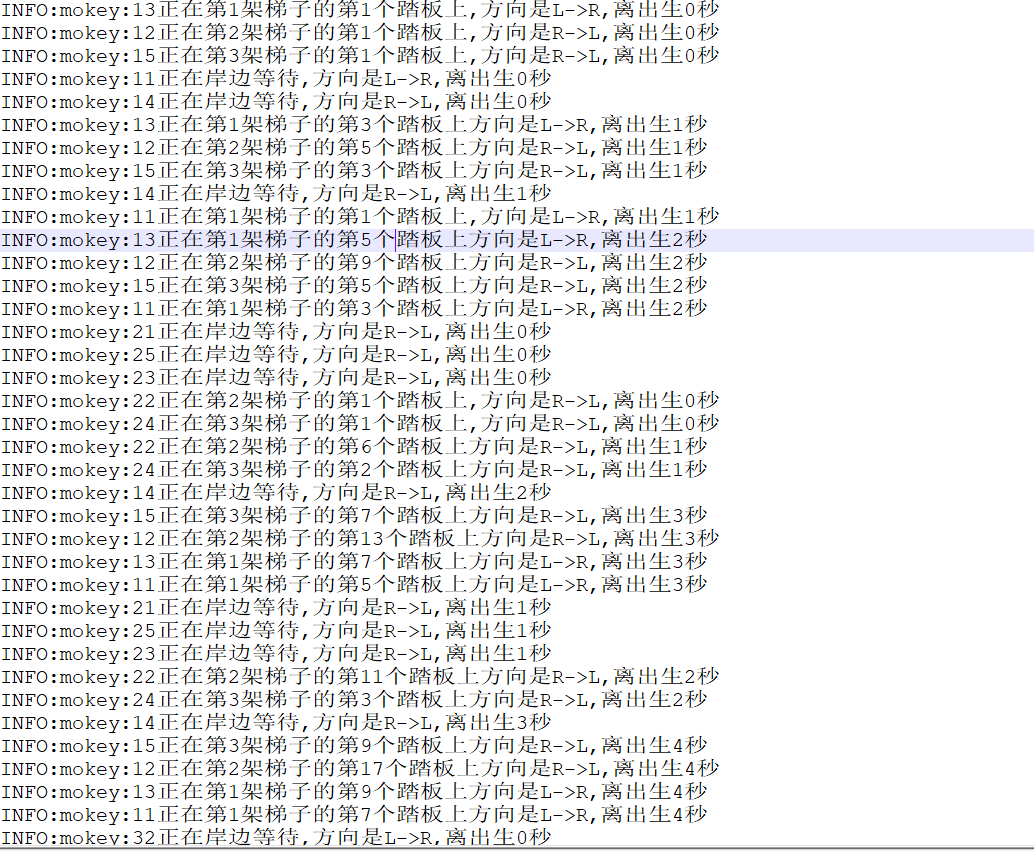
## 输出方案设计

日志

利用的日志输出，如下图所示：

在src下的logger文件夹下：

有输出的日志信息logger1.txt文本文件

## 猴子过河模拟器v1

### 参数如何初始化

利用客户端的用户输入参数n,t,N,k来初始化所有的猴子参数和桥的参数

例如：



### 使用Strategy模式为每只猴子随机选择决策策略

首先设计一个接口类Strategy

里面设计一个方法public int chooseLadder(Monkey m,ArrayList<Ladder> li)来选择梯子对象



然后设计两个策略strategy1，strategy2来实现这个接口。

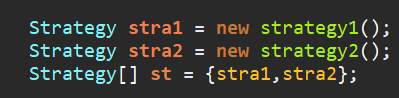
策略strategy1里面采用的是之前叙述的方法，即优先选择没有猴子的梯子，若所有梯子上都有猴子，则优先选择没有与我对向而行的猴子的梯子；若满足该条件的梯子有很多，则随机选择；



而策略strategy2采用的是之前叙述的方法，即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；

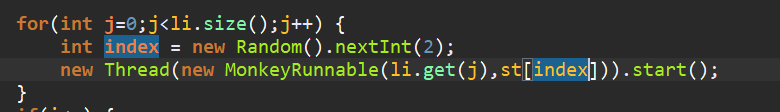


在随机选用这两个策略的时候，跟前面采取的策略相同，将两个策略放入到一个数组中，通过随机的产生数组的索引0或者1，从而达到随机采取这两种策略中的一种的目的。如下：



通过前面一样的方法，通过调用new Random().nextInt(2);就可以产生一个0或者1的随机数，从而我们就可以用这个随机数来访问数组里面的策略。

然后就可以调用线程，让猴子对象和选择的策略作为参数传入到启动线程类的构造函数里面，从而就可以进行过河程序。



## 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。

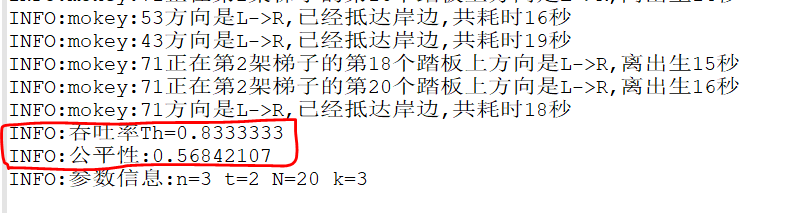
### 对比分析：固定其他参数，选择不同的决策策略

由于参数h和MV是一个固定的量，故直接使用，不再让用户输入。

固定参数n=3，t=2，N=20，k=3；观察不同的策略：

策略1：

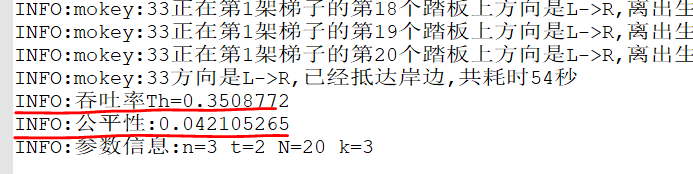
优先选择没有猴子的梯子，若所有梯子上都有猴子，则优先选择没有与我对向而行的猴子的梯子；若满足该条件的梯子有很多，则随机选择；

结果如图所示：  


其吞吐率为约为0.833，公平性约为0.5684

策略2：  
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

结果如图所示：



发现其吞吐率为0.3508，公平性为0.04.

对比下来，使用策略1的吞吐率是使用策略2的两倍多，相信增大N的值，会使这种差距越来越大。同时也发现，策略1的公平性远远的大于策略2.由此可以判断，策略1这种方法远远优于策略2这种方法。

### 对比分析：变化某个参数，固定其他参数

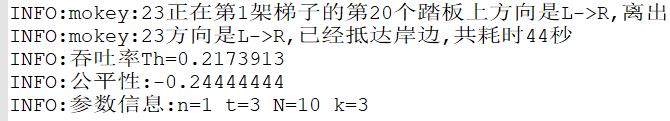
这个实验中，我采用的是性能比较优越的策略1来进行实验

1.首先，变化桥的数量n，固定其他的参数：

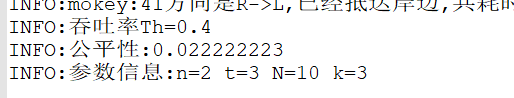
取桥的数量n由1变化到5，而其他参数设置为𝑡 = 3，𝑁 = 10，𝑘 = 3。

结果如下：

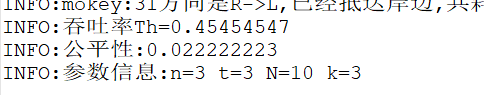
n=1:



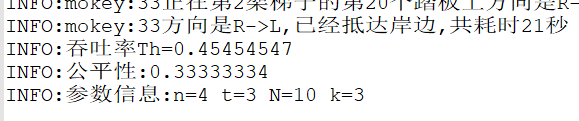
n=2：



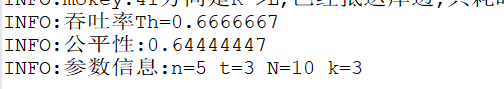
n=3：



n=4：



n=5：

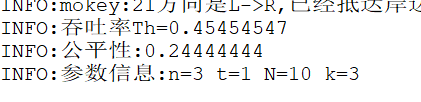


|  |  |  |
| --- | --- | --- |
| n | 吞吐率 | 公平性 |
| 1 | 0.217 | -0.244 |
| 2 | 0.4 | 0.022 |
| 3 | 0.454 | 0.022 |
| 4 | 0.454 | 0.333 |
| 5 | 0.666 | 0.644 |

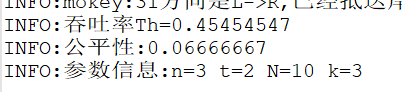
2.变化间隔时间t，其他参数保持不变：

让t从1变化到5，n=3，𝑁 = 10，𝑘 = 3，

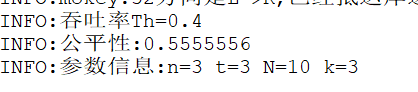
t=1



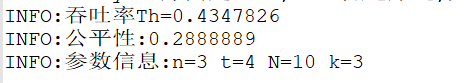
t=2



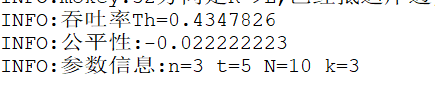
t=3



t=4



t=5

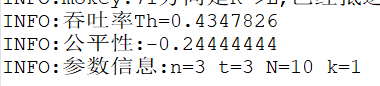


|  |  |  |
| --- | --- | --- |
| t | 吞吐率 | 公平性 |
| 1 | 0.454 | 0.244 |
| 2 | 0.454 | 0.066 |
| 3 | 0.4 | 0.555 |
| 4 | 0.434 | 0.288 |
| 5 | 0.434 | -0.022 |

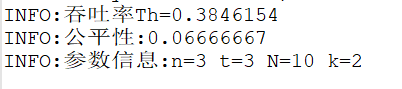
3.变化每次产生的猴子的数量k，保持其他参数不变

让k从1变化到3，n=3，𝑡 = 3，𝑁 = 10。

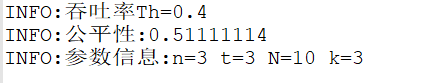
k=1



k=2



k=3



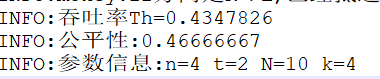
|  |  |  |
| --- | --- | --- |
| k | 吞吐率 | 公平性 |
| 1 | 0.4347 | -0.2444 |
| 2 | 0.3846 | 0.066 |
| 3 | 0.4 | 0.5111 |

4.变化猴子的数量N

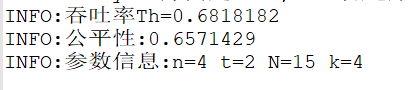
让猴子的数量N从10，15，20，25，30变化

设置n=4，t=2，k=4

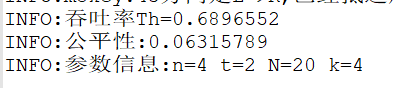
N=10



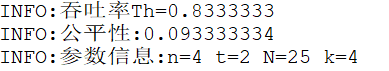
N=15



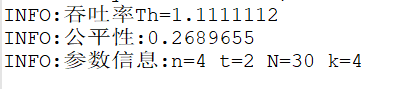
N=20



N=25



N=30



|  |  |  |
| --- | --- | --- |
| N | 吞吐率 | 公平性 |
| 10 | 0.4347 | 0.4666 |
| 15 | 0.6818 | 0.6571 |
| 20 | 0.6896 | 0.0631 |
| 25 | 0.8333 | 0.0933 |
| 30 | 1.1111 | 0.2686 |

### 分析：吞吐率是否与各参数/决策策略有相关性？

对比上述的实验数据发现，吞吐率与选择的策略有很大的关联。

我们可以从3.9.1发现，选取策略1时的吞吐率比选取策略2时的吞吐率大两倍之多。

然后从3.9.2观察，

当其他参数不变，桥的数量n从1变化到5时，n的值越大，过河程序的吞吐率也就越大，而且公平性也相应的在增大。

当其他参数不变，时间间隔t从1变化到5时，t的值越大，过河程序的吞吐率基本上还是比较稳定在一个水平，说明t值对吞吐率的影响不是太大，只是公平性变化较大

当其他参数不变，每次产生猴子的数量k从1变化到3时，吞吐率的变化也不是太大，说明k值对吞吐率的影响比其他的略小。其公平性变化较大。

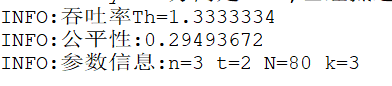
当其他参数不变，猴子总数N发生变化的时候，吞吐率的变化比较明显，N值越大，吞吐率也就越大，这是因为猴子的数量变多，使每秒过河的猴子数也相应的增多了，使吞吐率变大了。同样，其公平性变化也比较大，且无规律。

### 压力测试结果与分析

设置参数如下：

N=80，n=3，t=2，k=3

运行程序，得到如下的结果：



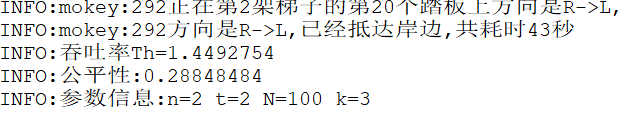
总时间大约用了40秒

继续增大猴子的数量N和减小桥的数量n

n=2，N=100，t=2，k=3

如下的文档：





总时间用了大约1分钟

# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2018.6.5-6.12 | 18：00-22：00 | 完成猴子生成器和ADT | 完成 |
| 2018.6.12-16 | 18：00-22：00 | 完成过河程序 | 完成 |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

在求吞吐率和公平性的时候，需要所有的猴子都过河完成，因此在主线程中，需要让所有子线程执行完后再来求程序的吞吐率和公平性。

我采用了java里面的CountDownLatch类来解决这个问题，它首先设定子线程的个数，然后在每一个子线程结束的时候，利用它里面的方法. CountDownLatch.countDown()来记录下此线程已经结束了。最后利用CountDownLatch.await()来等待CountDownLatch里面的参数变成0，等到所有的子线程都结束以后，就可以开始主线程里面的的求吞吐率和公平性的操作了。

# 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

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？

多线程执行的顺序太随机，无法确切知道程序具体的执行和debug。它确实可以改善很多的性能

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？

使用synchronized锁来锁住对象。以确保threadsafe为主，再辅以性能

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？

无，重头开始就考虑了

1. 关于本实验的工作量、难度、deadline。

工作量适中

1. 到此为止你对《软件构造》课程的意见和建议。

内容太多