

**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 | 强文杰 |
| 学号 | 1171000410 |
| 班号 | 1703005 |
| 电子邮件 | [672334335@qq.com](mailto:672334335@qq.com) |
| 手机号码 | 18800421389 |

**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 9](#_Toc1393931)

[3.3 至少两种“梯子选择”策略的设计与实现方案 11](#_Toc1393932)

[3.3.1 策略1 11](#_Toc1393933)

[3.3.2 策略2 11](#_Toc1393934)

[3.3.3 策略3（可选） 12](#_Toc1393935)

[3.4 “猴子生成器”MonkeyGenerator 12](#_Toc1393936)

[3.5 如何确保threadsafe？ 13](#_Toc1393937)

[3.6 系统吞吐率和公平性的度量方案 15](#_Toc1393938)

[3.7 输出方案设计 16](#_Toc1393939)

[3.8 猴子过河模拟器v1 19](#_Toc1393940)

[3.8.1 参数如何初始化 19](#_Toc1393941)

[3.8.2 使用Strategy模式为每只猴子随机选择决策策略 20](#_Toc1393942)

[3.9 猴子过河模拟器v2 21](#_Toc1393943)

[3.9.1 对比分析：固定其他参数，选择不同的决策策略 22](#_Toc1393944)

[3.9.2 对比分析：变化某个参数，固定其他参数 24](#_Toc1393945)

[3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？ 29](#_Toc1393946)

[3.9.4 压力测试结果与分析 30](#_Toc1393947)

[4 实验进度记录 31](#_Toc1393948)

[5 实验过程中遇到的困难与解决途径 33](#_Toc1393949)

[6 实验过程中收获的经验、教训、感想 33](#_Toc1393950)

[6.1 实验过程中收获的经验和教训 33](#_Toc1393951)

[6.2 针对以下方面的感受 33](#_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。

没有特殊的实验环境需要配置。

在这里给出你的GitHub Lab6仓库的URL地址（Lab6-学号）。

https://github.com/ComputerScienceHIT/Lab6-1171000410.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；

**一. Monkey**

猴子类。继承Thread。

作用：每个猴子为一个线程，对于一只猴子在某时刻选择并爬上某个梯子，意味着它跳到第 1 个踏板上。猴子一旦上了某个梯子，就不能在中途跳到别的梯子上。当猴子跳出梯子，该线程结束。

fields：

private final int id; //num of monkey

private final String direction; //initially L or R

private int v; //speed

private int position = 0; //position in the ladder

private final int produceTimer; //timer to record produceTime

private int now; //timer to record the elapsed time

private boolean finished = false; //Whether it has passed the ladder

private Strategy ladderStratrgy; //strategy of choosing ladder

private Ladder ladder = null; //Ladder being used

方法：

// getters方法

// setters方法

主要的方法是重写run

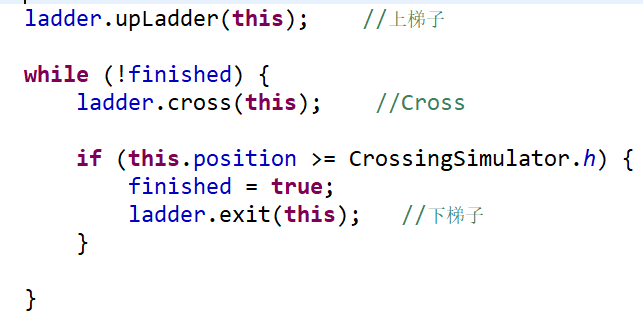
1. 方法开始要根据策略选择梯子，传入的参数是猴子的direction，返回的是选到的梯子对象或null，如果返回为null，则线程sleep 1秒，while循环中下一秒继续选择梯子。

ladder = this.ladderStratrgy.chooseLadder(this.direction); //选梯子

2. 如果选到了不为null的梯子，调用Ladder类中的方法使猴子上梯子。

ladder.upLadder(this); //上梯子

3. 接下来又是一个while循环，判断的是猴子的状态finished是否为true，while循环的内部调用Ladder.cross(this)使猴子不断在梯子上移动，如果猴子的位置超出了梯子的长度，则finished变为true，且猴子下梯子。



**二. Ladder**

梯子类。

作用：梯子提供了猴子过桥的工具，其中包含了猴子选踏板、上梯子、过梯子、下梯子等方法。

fileds：

private int leftCrossing = 0; //monkeys from the left direction

private int rightCrossing = 0; //monkeys from the right direction

private int[] pedals = new int[CrossingSimulator.h + 10]; //pedals free or occupied

private final int id; //number of ladder

方法：

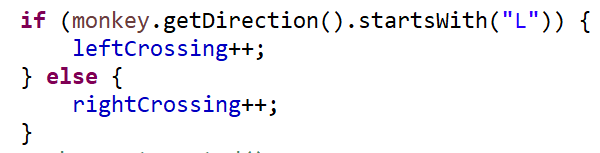
//getters

//setters

其中getters和setters都是被synchronized修饰的方法，这些方法为同步方法，synchronized作用的范围是整个方法，作用的对象是调用这个方法的Ladder对象；

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

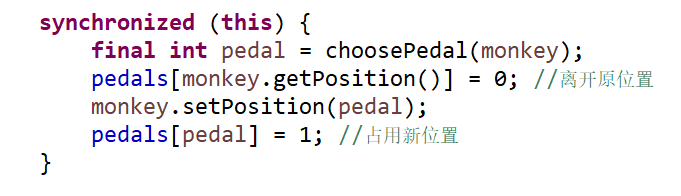
该方法为猴子上梯子，synchronized修饰的同步方法，如果猴子的方向是L->R，则leftCrossing++，否则rightCrossing++。



**public** **void** cross(Monkey monkey)

该方法为猴子过梯子，针对这个方法，为了提高线程并行的效率，未使用synchronized方法修饰方法，而是修饰一个代码块，为同步语句块，其作用的范围是大括号{}括起来的代码，加锁的对象还是ladder。

先根据梯子上的占用情况选择踏板，将猴子的原位置设置为0，将猴子的新位置设置为1，并且猴子内部的position也更新。执行这步需要1s，因此需要将猴子现在的时钟加1s。



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

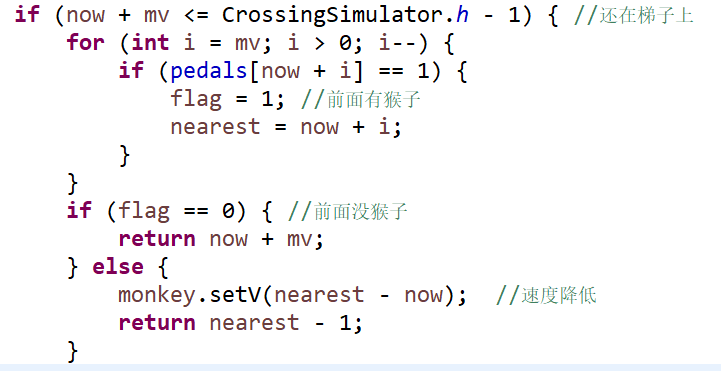
该方法为猴子下梯子，synchronized修饰的同步方法，如果猴子的方向是L->R，则leftCrossing--，否则rightCrossing--。梯子某个方向的猴子数量减一，并且猴子离开梯子前的最后一块踏板设置为0。

**private** **int** choosePedal(Monkey monkey)

猴子选择踏板。先获得猴子此时的位置，和其速度（最大可能移动距离）。

我们需要讨论猴子的最大可能移动的位置是否会超出梯子，如果不会超出梯子，再讨论移动范围内前面是否有猴子，如果没有，返回最远可能踏板，如果有，返回没有猴子的最近踏板，并且猴子速度需要降低。如果会超出梯子，再讨论梯子前面是否有猴子，如果有，则返回梯子最远距离，如果没有，返回最远可能踏板，并且猴子速度需要降低。

方法的内部，多重if else讨论的语句块中需要对pedals进行加锁。



**三. CrossingSimulator**

过河模拟器。

作用：实现与用户交互，读入用户输入的参数或v3中的文件，获得实验中monkey和ladder的参数。

fields：

public static final int h = 20; //length of ladders

public static int n; //quantity of ladders

public static ArrayList<Monkey> monkeyList = new ArrayList<Monkey>(); //list to store monkeys

public static ArrayList<Ladder> ladderList = new ArrayList<Ladder>(); //list to store ladders

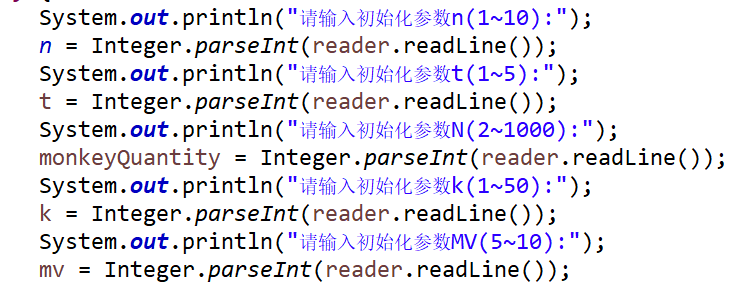
public static final Logger log = Logger.getLogger(CrossingSimulator.class.getSimpleName()); //logging

public static double extraTime = 0; //time spent on GUI and log

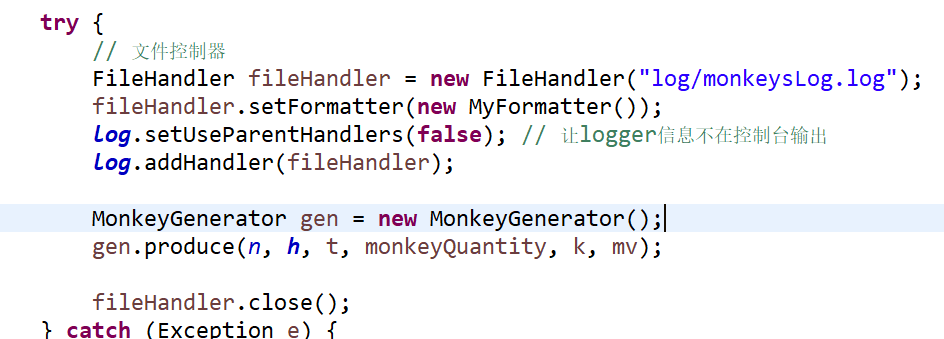
方法：

main()

读入用户输入猴子生成器需要的参数



写日志，新建一个猴子生成器，在其produce方法中传入参数。



**四. MonkeyGenerator**

猴子生成器。

1. 在猴子生成器MonkeyGenerator中只有一个方法，

public void produce(int n, int h, int t, int monkeyQuantity, int k, int mv , int s)

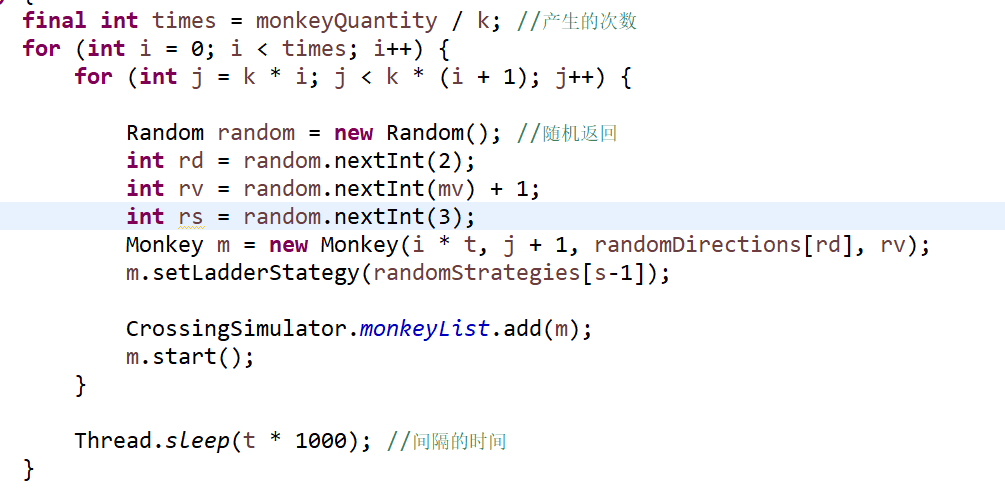
其中方法的参数由用户输入后传入，分别为梯子数n，阶梯数h（固定为20），产生猴子间隔秒数t，猴子总数，一次产生的猴子数量，猴子的最大速度，策略（猴子过河模拟器v2中使用）。

2. 得到这些参数后，我们需要随机生成猴子的信息，猴子的ID就在根据猴子对象产生的顺序编号。其他的信息，把猴子的两个方向放到一个集合中，把三种策略的对象放在一个集合中，然后以如下的形式产生随机数，即可在集合中随机获得属性。

Random random = new Random();

int rd = random.nextInt(2);

3. 至于产生猴子的时间和数量，则根据传入的参数t、k和monkeyQuantity，使用一个两重循环即可。另外，如果monkeyQuantity/𝑘不为整数，则最后一次产生的猴子个数为monkeyQuantity%𝑘。



**五. ConsoleTextArea 和** **LoopedStreams**

以上两个类是为实现GUI服务的。

1. LoopedStreams类：

作用：它用一个ByteArrayOutputStream提供和Java管道流类似的功能，但不会出现死锁和IOException异常。

这个类的内部仍旧使用管道流。构造函数很简单，它连接管道流，然后调用startByteArrayReaderThread()方法。

方法：

getOutputStream()方法返回一个OutputStream（具体地说，是一个ByteArrayOutputStream）用以替代PipedOutputStream。

写入该OutputStream的数据最终将在getInputStream()方法返回的流中作为输入出现。和使用PipedOutputStream的情形不同，向ByteArrayOutputStream写入数据的线程的激活、写数据、结束不会带来负面效果。

2. ConsoleTextArea类

作用：ConsoleTextArea类扩展Swing JTextArea捕获控制台输出。

实现：在CrossingGenerator的main()方法中创建一个JFrame，JFrame包含一个ConsoleTextArea的实例。Frame显示出来之后，main()方法启动一系列的写操作线程，写操作线程向控制台流输出大量信息。ConsoleTextArea捕获并显示这些信息。

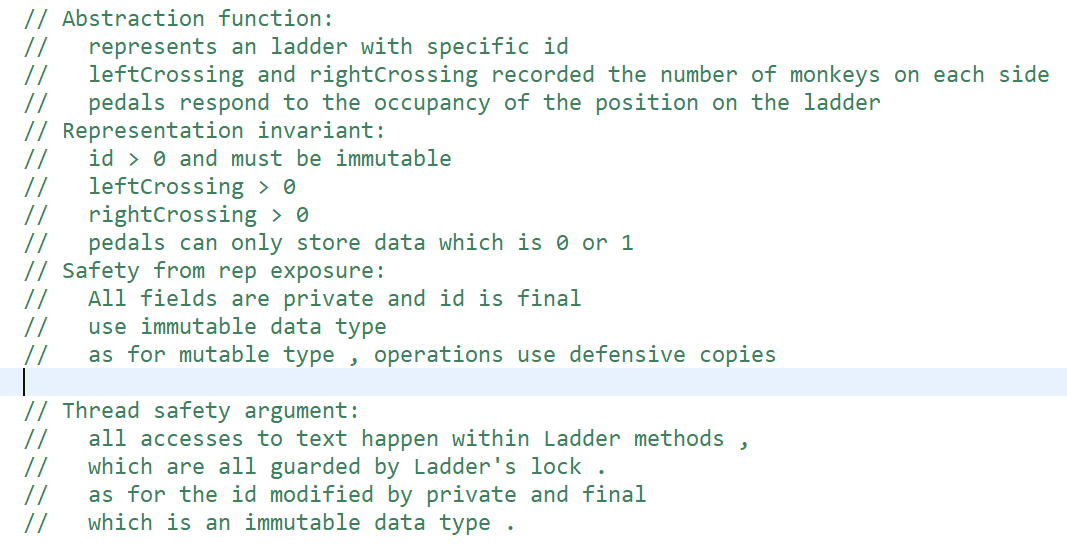
方法：

ConsoleTextArea提供了两个构造函数。没有参数的构造函数用来捕获和显示所有写入到控制台流的数据，有一个InputStream[]参数的构造函数转发所有从各个数组元素读取的数据到JTextArea。首先我们来看看没有参数的ConsoleTextArea构造函数。构造函数调用startConsoleReaderThread()，创建一个不断地把文本行追加到JTextArea的线程。

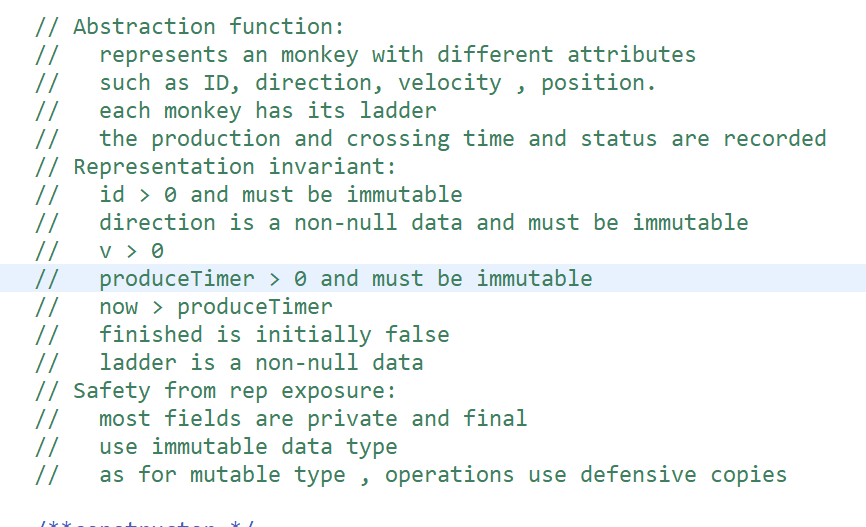
startConsoleReaderThread()方法： 在Swing中向JTextArea追加文本是一个线程安全的操作。读取一行文本之后，我们只需调用JText.append()就可以把文本追加到JTextArea的末尾。

给出每个ADT的specification；

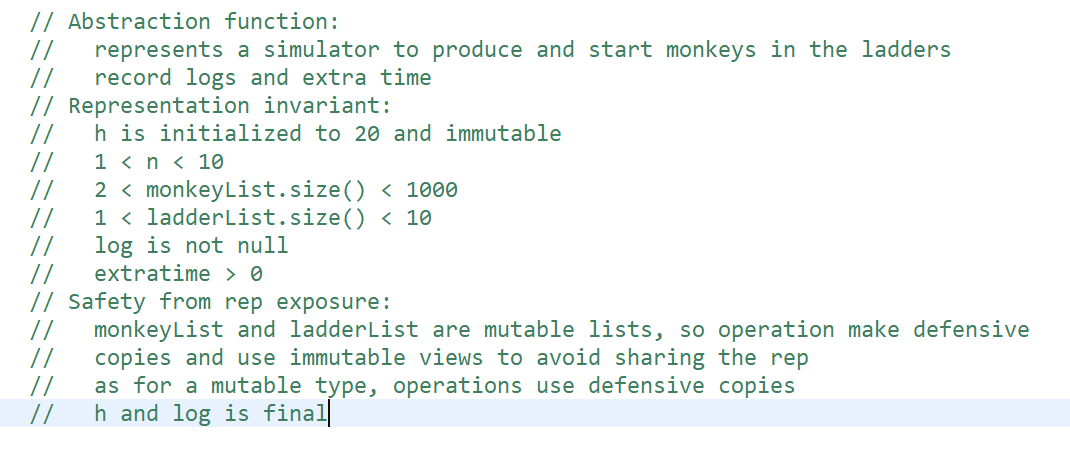
Ladder：对于梯子类，还需要保证线程安全



Monkey：



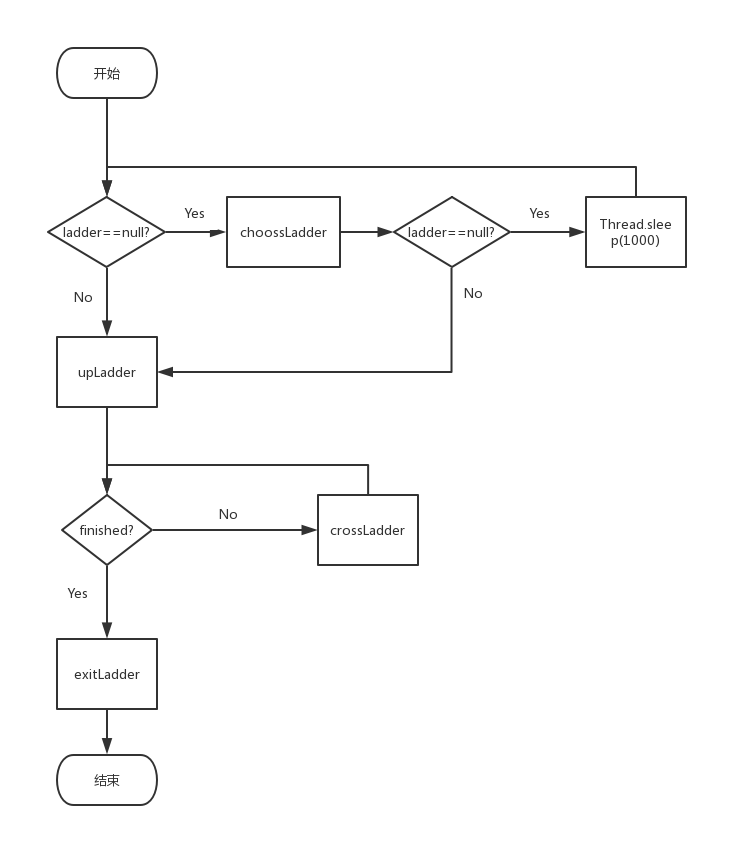
CrossingSimulator:



（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。



## 至少两种“梯子选择”策略的设计与实现方案

### 策略1

策略内容：

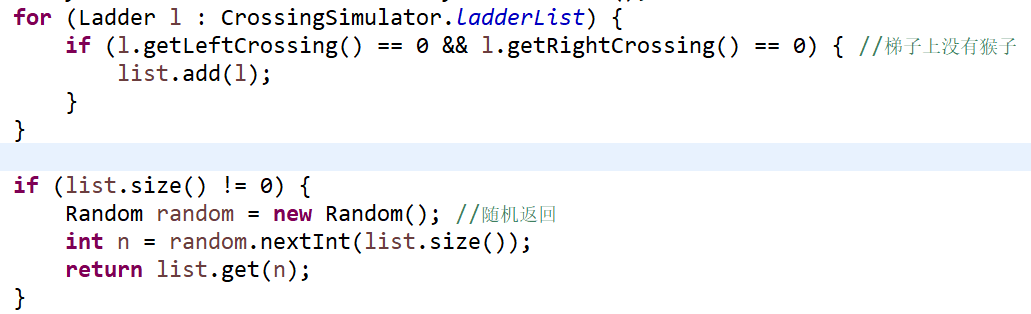
// 优先选择没有猴子的梯子

// 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

策略实现方案：

1. 遍历梯子的集合，如果梯子上左边来的猴子和右边来的猴子的数量都是0，则将该梯子加入列表中。然后从列表中随机返回一个满足条件的梯子。



2. 若找不到满足以上条件的猴子，则从没有对向而行的梯子中随机返回一个，那样的梯子满足的条件是，该猴子是L->R且梯子上没有右边来的猴子，或者该猴子是R->L且梯子上没有左边来的猴子。

3. 若找不到符合以上条件的猴子，则返回空，猴子等待下一秒进行选择梯子。

### 策略2

策略内容：

// 优先选择推进速度最快的梯子

// 综合考虑梯子上猴子数量和和我最近猴子的距离

// 将每个梯子映射到一个分数

// 选择分数最大的梯子

策略实现方案：

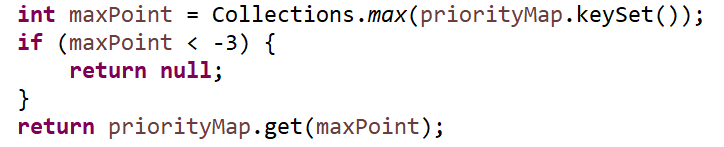
1. 刚开始找没有猴子的梯子，然后找没有相向而行猴子的梯子，判断方法和策略1一致，就是梯子满足条件，该猴子是L->R且梯子上没有右边来的猴子，或者该猴子是R->L且梯子上没有左边来的猴子。

2. 定义如下的HashMap

Map<Integer, Ladder> priorityMap = **new** HashMap<Integer, Ladder>(); //每个梯子映射一个积分

积分取决于猴子的数量和离我最近猴子的距离，数量越少，离我最近的猴子越远，积分越多，具体的权值我分别设置为-2和1.

3. 找到keyset中最大的积分，如果最大的积分仍然不理想，返回null，该猴子进入下一秒的选择；否则返回该积分对应的梯子。



### 策略3（可选）

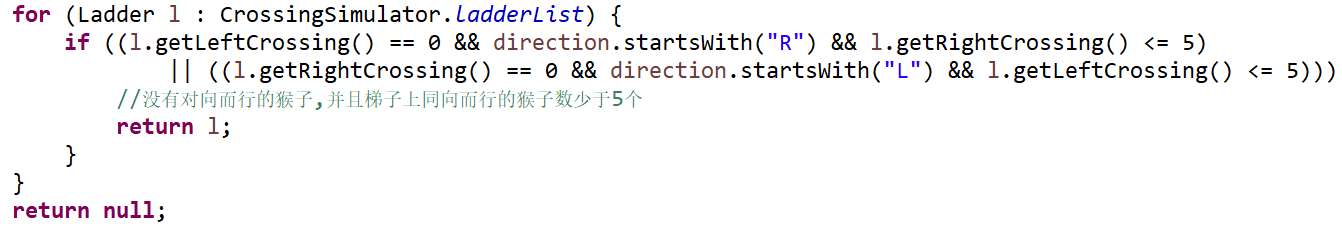
策略内容：

// 梯子上没有对向而行的猴子

// 并且梯子上同向而行的猴子数不大于5个

策略实现方案：

这个策略的实现比较简单，就是遍历梯子的集合时，在if语句中判断，如果梯子上不存在与自己相向而行的猴子，并且与自己同向而行的猴子数量不大于5，直接返回。



## “猴子生成器”MonkeyGenerator

如何设计和实现。

在上述ADT介绍部分已有说明。

1. 在猴子生成器MonkeyGenerator中只有一个方法，

public void produce(int n, int h, int t, int monkeyQuantity, int k, int mv , int s)

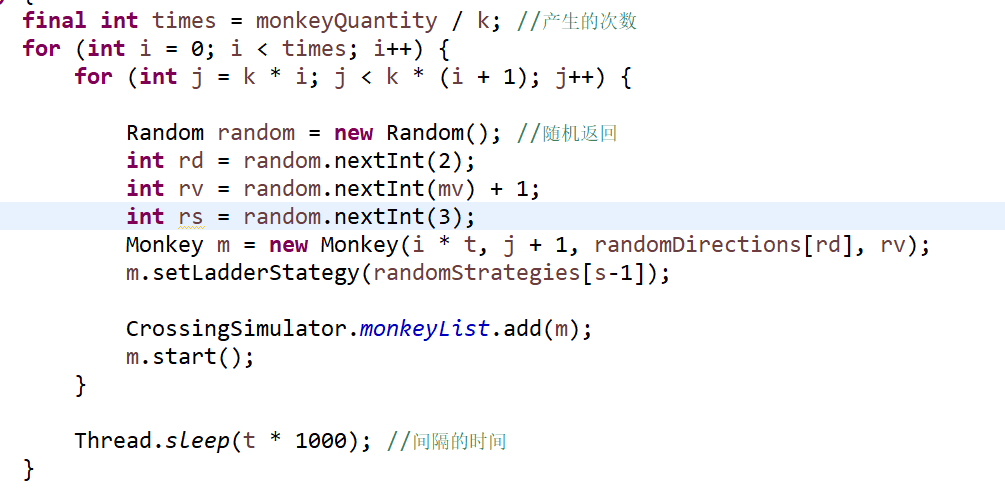
其中方法的参数由用户输入后传入，分别为梯子数n，阶梯数h（固定为20），产生猴子间隔秒数t，猴子总数，一次产生的猴子数量，猴子的最大速度，策略（猴子过河模拟器v2中使用）。

2. 得到这些参数后，我们需要随机生成猴子的信息，猴子的ID就在根据猴子对象产生的顺序编号。其他的信息，把猴子的两个方向放到一个集合中，把三种策略的对象放在一个集合中，然后以如下的形式产生随机数，即可在集合中随机获得属性。

Random random = new Random();

int rd = random.nextInt(2);

3. 至于产生猴子的时间和数量，则根据传入的参数t、k和monkeyQuantity，使用一个两重循环即可。另外，如果monkeyQuantity/𝑘不为整数，则最后一次产生的猴子个数为monkeyQuantity%𝑘。



## 如何确保threadsafe？

1. 限制数据共享

将position和v等可变数据限制在单一的Monkey线程的内部，避免竞争，猴子线程内部使用局部变量，线程之间不共享mutable数据类型。

2. 共享不可变数据

Ladder中，如id是不可变的数据类型，指向内存的引用是一直不变的，通常使用private final来修饰。

3. 共享线程安全的可变数据

因为对于monkeyList和ladderList，我们必须要用mutable的数据类型在多线程之间共享数据，但此时要使用线程安全的数据类型。即使用synchronizedList这样的线程安全集合类。

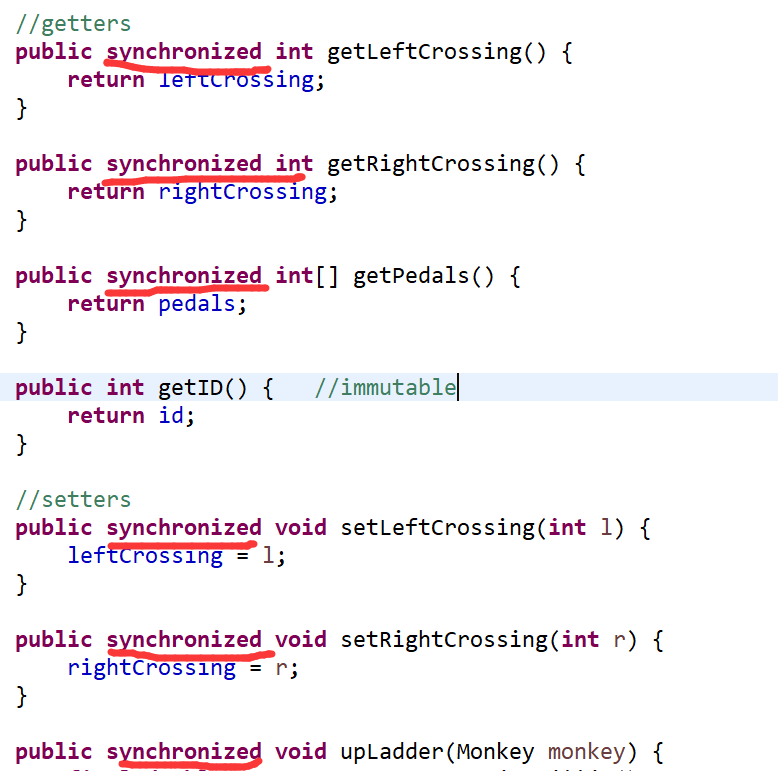
4. 同步机制（Synchronization）

对于synchronized关键字，我们在两处使用到了它：

一. synchronized 方法

在Ladder类中，所有的getters和setters，包括uoLadder()、exit()等方法都被synchronized进行修饰，Ladder实例对应一把锁，每个synchronized方法都必须获得调用该方法的Ladder实例的锁方能执行，否则所属线程阻塞，方法一旦执行，就独占该锁，直到从该方法返回时才将锁释放。

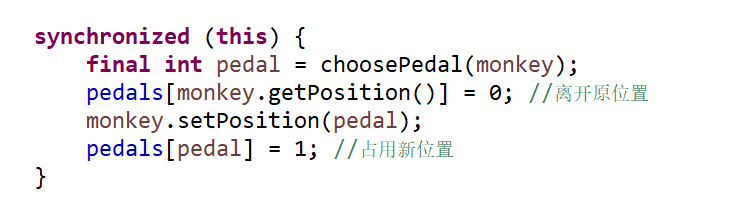
这样确保了同一时刻对于每一个Ladder实例，其所有声明为 synchronized 的成员函数中至多只有一个处于可执行状态（因为至多只有一个能够获得该Ladder实例对应的锁），从而有效避免了类成员变量的访问冲突。



二. synchronized 代码块

对于某些方法，我们为了提供更加更加精细的粒度，选择不是对方法进行synchronized修饰，而是显示给出对代码块的lock，并且尽可能减小了lock的范围。

如下：synchronized(this){/区块/}，它的作用域是当前Ladder对象



三. 还是synchronized代码块

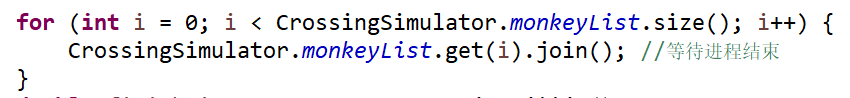
在使用策略模式选择梯子的时候，我们需要对梯子的集合进行synchronized修饰，以保证梯子选中返回后也不会被其他Monkey选中。此时，梯子集合中的Ladder对象无需再synchronized修饰，因为当我们调用Ladder类中的方法时，该Ladder的实例已经自动获得该锁。

## 系统吞吐率和公平性的度量方案

吞吐率：

过河的猴子总数很好收集，主要是获得所有猴子过河花费的总时间。

在所有猴子线程start之前，获得当前的时间，然后我们需要等待所有进程结束，获得最终的时间，这里要用Thread.join()方法。



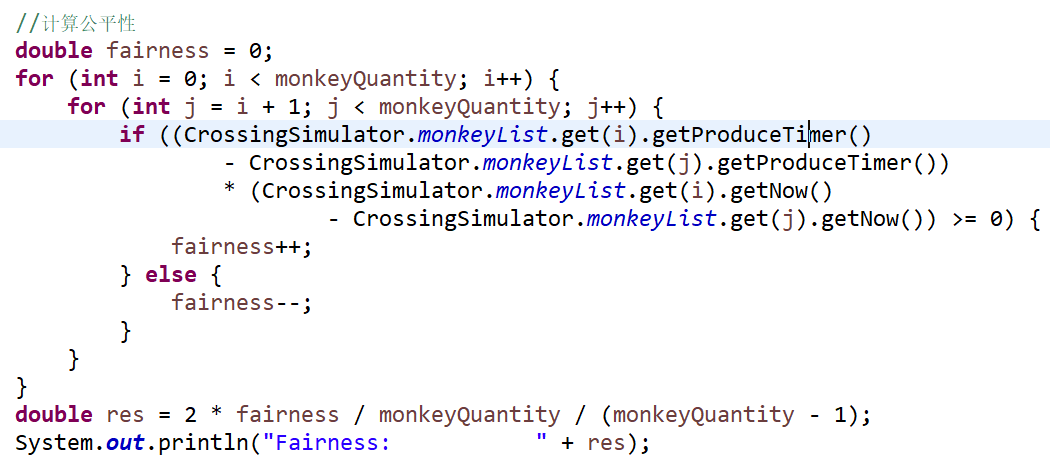
除此之外，我们需要除去在GUI打印和写入日志中花费的时间extraTime，最后使用公式，计算出总时间和吞吐率

double totalTime = finishTime - startTime - CrossingSimulator.extraTime;

System.out.println("Throughput rate: " + CrossingSimulator.monkeyList.size() \* 1000 / totalTime);

公平性：

对于系统公平性的计算，事实上是一个二重循环，用到了报告中给到的公式，由于我们在Monkey类中已经记录了猴子产生的时间produceTimer和猴子过河时的时间，只要所有猴子两两比较，如果先产生的猴子先过河则F(A.B)加一，否则减一，最后的fairness除以Cn2的组合即可。

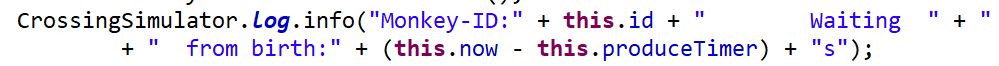


## 输出方案设计

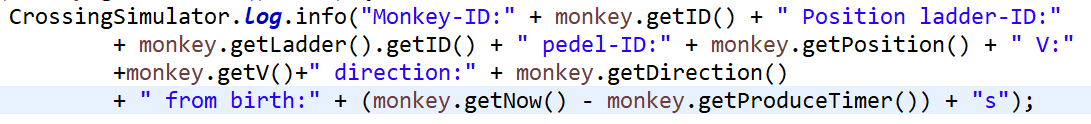
日志

猴子过河线程执行时，每隔 1 秒做一次行动决策并执行决策（原地不同、或根据梯子上的具体情况向前移动若干个横梯），并在 log 日志中记录该 Monkey 对象的当前状态，状态分为以下三种情况：

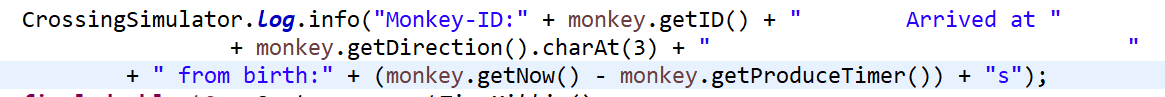
1. 正在岸边等待，在monkey每次选择梯子的时候记录日志，日志的内容包括monkey id ， monkey 离出生多长时间。



2. 在ladder的cross方法中，即猴子过河的时候，记录mokey id ， 梯子的序号 ， 所处的踏板的序号， 猴子的速度（可以反映出因为前面的猴子较慢，速度有下降） ， 猴子的方向 ， 离出生的时间。

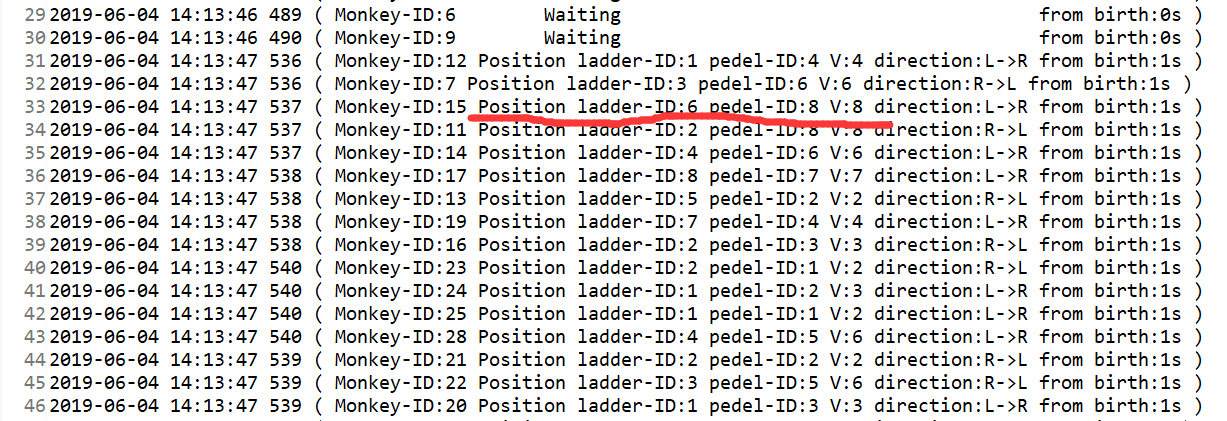


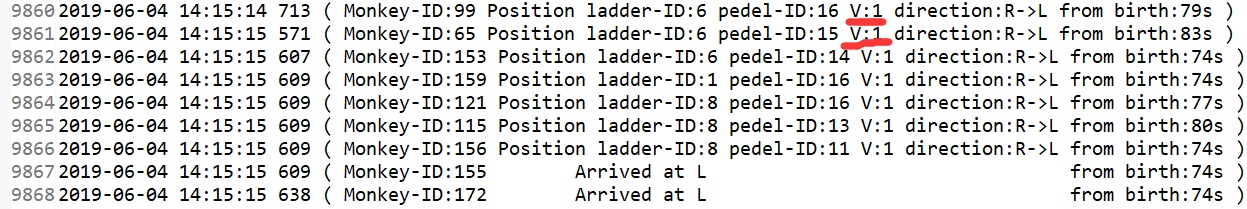
3. 在ladder的exit方法中，即猴子到达对岸的时候，记录monkey id ， 对岸所处的方向 ， 离出生的时间。



分析日志，以下分别截取了一次过河过程的前部分和后部分的日志。我们可以清楚的看到梯子的序号，和每个梯子的踏板被占用的情况，从而看出整个过河过程的正确性。

对比后面一张图，我们会发现，过河过程快结束时，梯子上monkey的速度大部分都是1，这是梯子上猴子速度不断下降导致的。





GUI

第一部分：捕获程序的控制台输出，使其在有滚轮的GUI中展示。

1. LoopedStreams类：

作用：它用一个ByteArrayOutputStream提供和Java管道流类似的功能，但不会出现死锁和IOException异常。

这个类的内部仍旧使用管道流。构造函数很简单，它连接管道流，然后调用startByteArrayReaderThread()方法。

方法：

getOutputStream()方法返回一个OutputStream（具体地说，是一个ByteArrayOutputStream）用以替代PipedOutputStream。

写入该OutputStream的数据最终将在getInputStream()方法返回的流中作为输入出现。和使用PipedOutputStream的情形不同，向ByteArrayOutputStream写入数据的线程的激活、写数据、结束不会带来负面效果。

2. ConsoleTextArea类

作用：ConsoleTextArea类扩展Swing JTextArea捕获控制台输出。

实现：在CrossingGenerator的main()方法中创建一个JFrame，JFrame包含一个ConsoleTextArea的实例。Frame显示出来之后，main()方法启动一系列的写操作线程，写操作线程向控制台流输出大量信息。ConsoleTextArea捕获并显示这些信息。

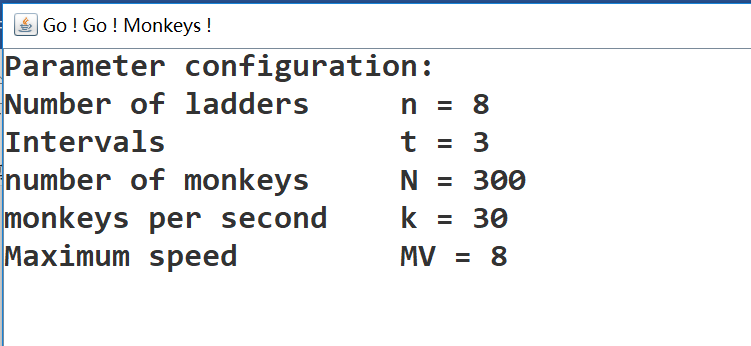
方法：

ConsoleTextArea提供了两个构造函数。没有参数的构造函数用来捕获和显示所有写入到控制台流的数据，有一个InputStream[]参数的构造函数转发所有从各个数组元素读取的数据到JTextArea。首先我们来看看没有参数的ConsoleTextArea构造函数。构造函数调用startConsoleReaderThread()，创建一个不断地把文本行追加到JTextArea的线程。

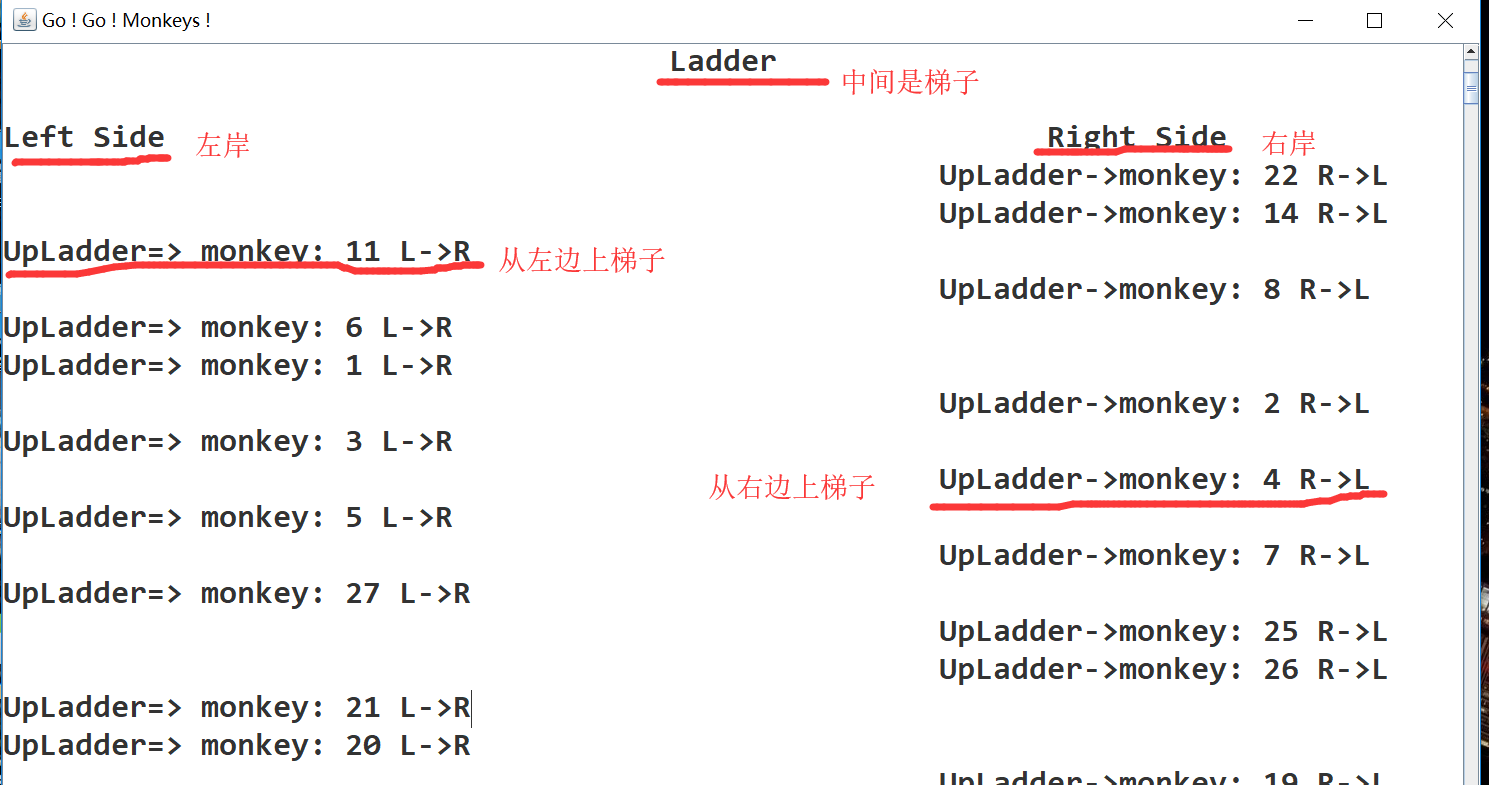
startConsoleReaderThread()方法： 在Swing中向JTextArea追加文本是一个线程安全的操作。读取一行文本之后，我们只需调用JText.append()就可以把文本追加到JTextArea的末尾。

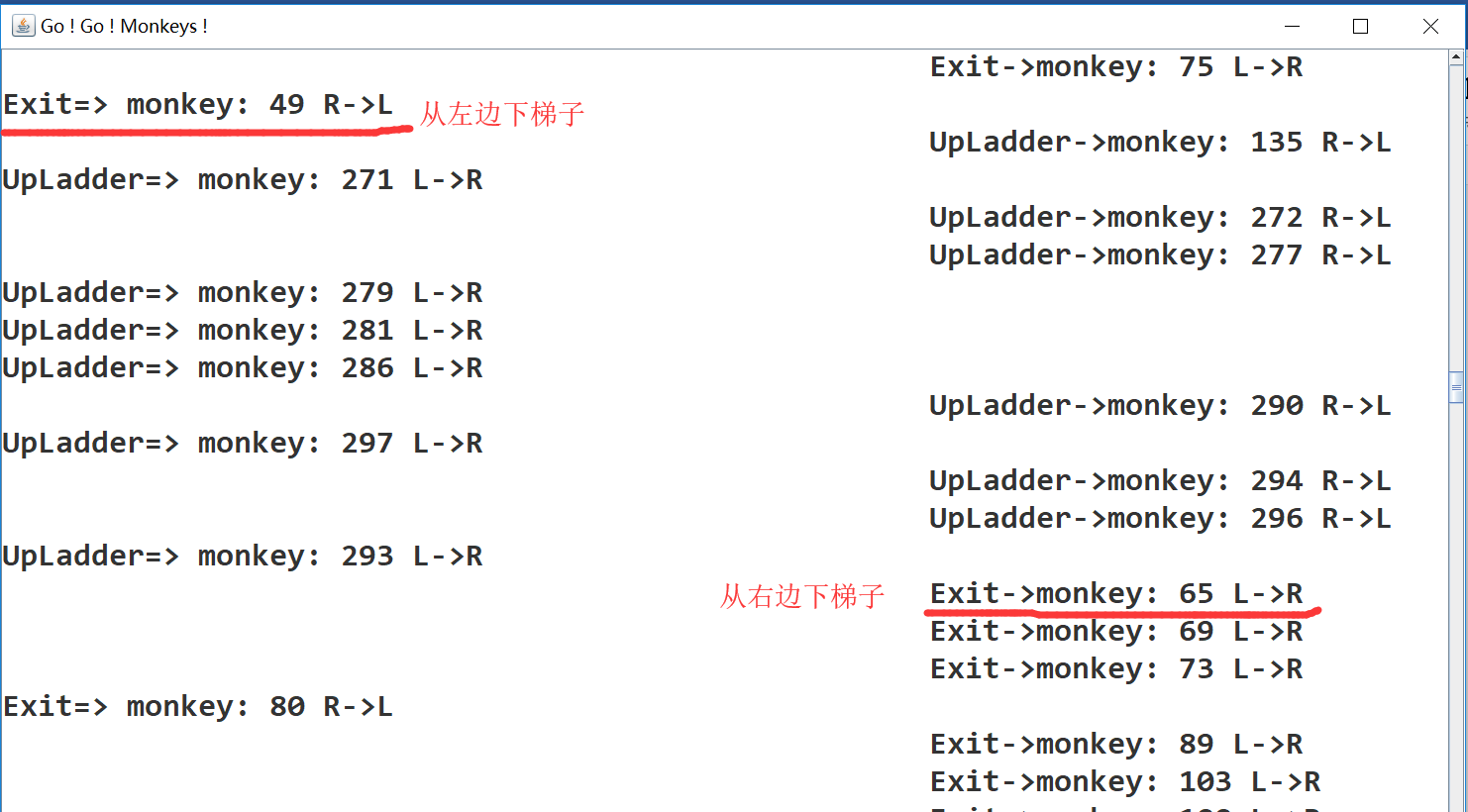
第二部分：GUI的展示效果。

1. 在GUI开头显示程序本次模拟的各个参数配置



2. GUI展示了猴子的过河过程，中间空白的地方是梯子（河），Left side为左岸，Right side为右岸，猴子信息主要反映为猴子的编号和方向，如果猴子从左岸上梯子，则它会在左岸显示UpLadder，如果从右岸下梯子，则它会在右岸显示Exit。结合下图进行说明。

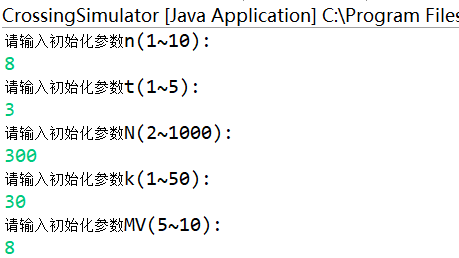


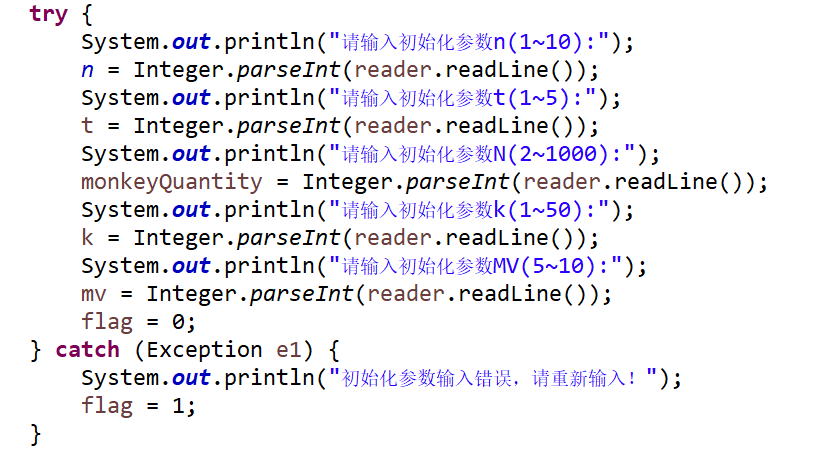


## 猴子过河模拟器v1

### 参数如何初始化

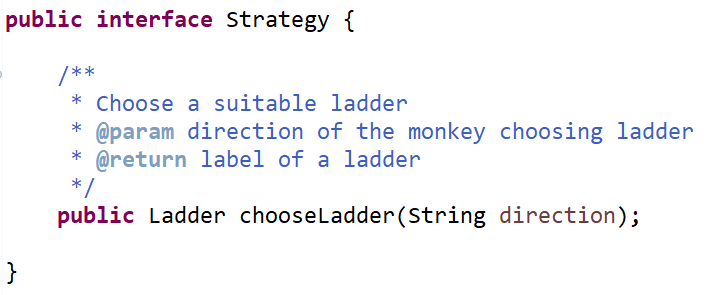
用户从控制台输入具体的参数。





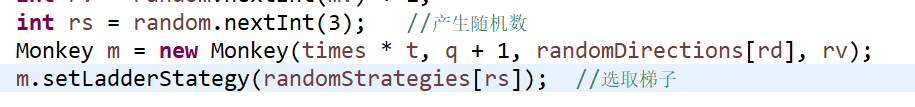
### 使用Strategy模式为每只猴子选择决策策略

**抽象策略对象(Strategy)：**它可由接口或抽象类来实现。



**具体策略对象(ConcreteStrategy)：**它封装了选取梯子的不同实现策略，所有策略都需要implements Strategy接口，每一个都根据自己这选取梯子的策略对方法进行了重写，其中具体的选梯子策略在上文已有说明。

**环境对象(Context)：**分别在用户需要不同梯子策略的时候中实现了对抽象策略中定义的接口的引用。我们在使用它的时候，直接声明一个应用场景类，通过用户的不同输入来给Context传入不同的选梯子策略即可，这样我们就能够只使用一个变量而获得多种行为模式了。

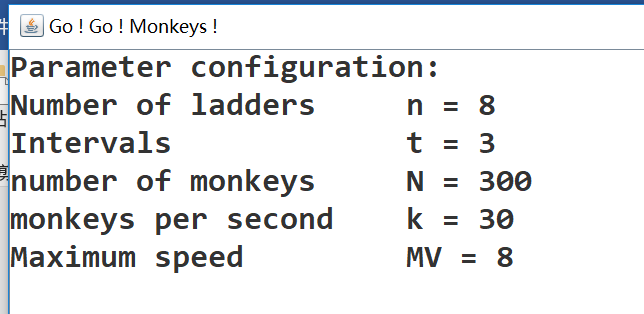


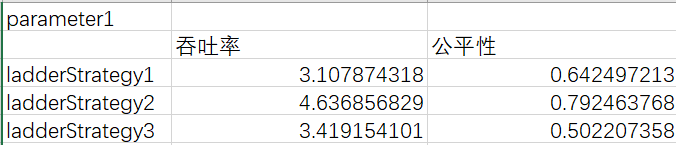
## 猴子过河模拟器v2

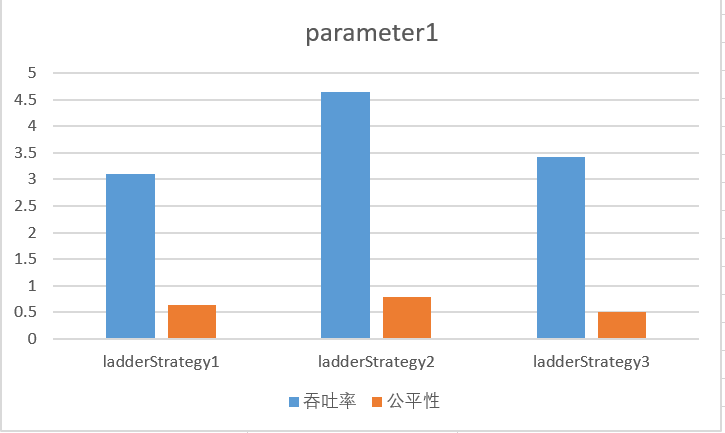
在不同参数设置和不同“梯子选择”模式下的“吞吐率”和“公平性”实验结果及其对比分析。

### 对比分析：固定其他参数，选择不同的决策策略

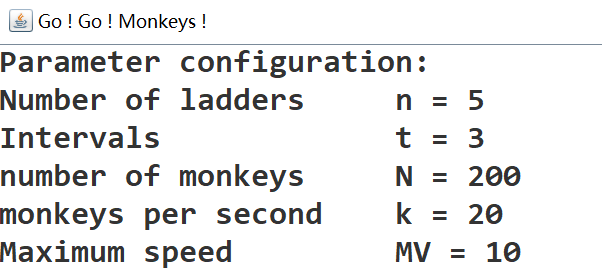
参数配置1：

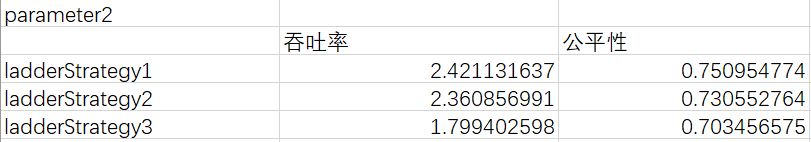


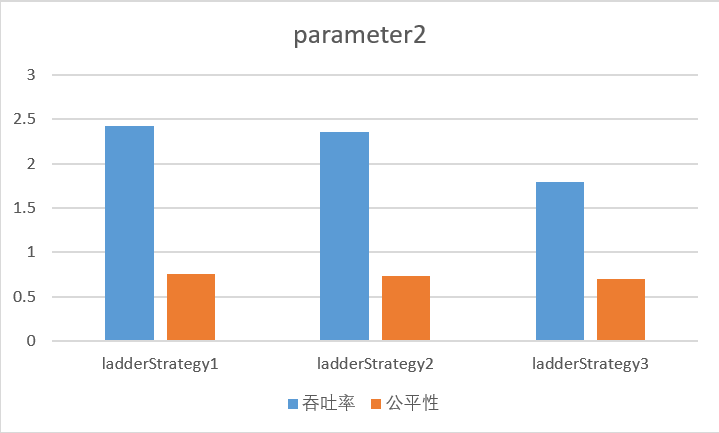




参数配置2：



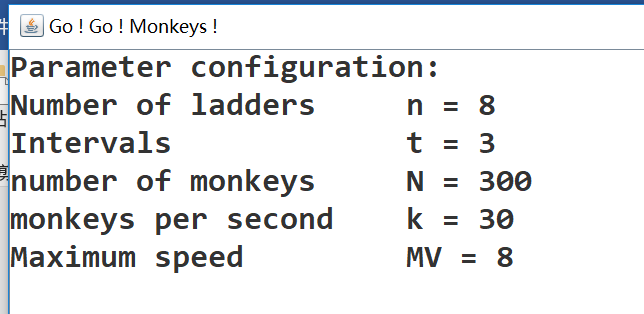




### 对比分析：变化某个参数，固定其他参数

**以下所有的实验分析的初始参数配置如下，将根据此参数进行变化。**

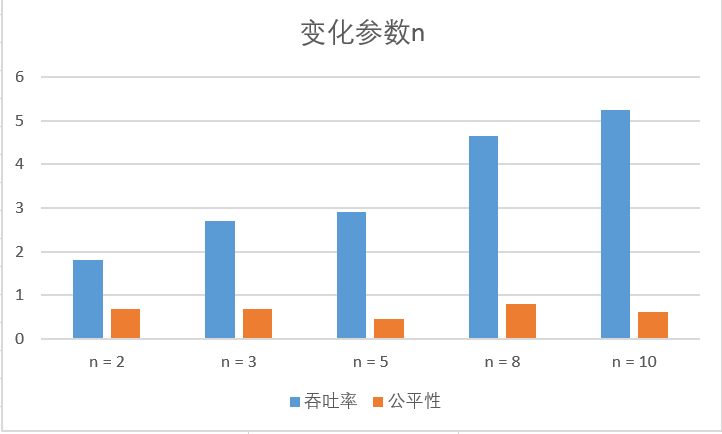
**猴子的策略固定为LadderStrategy2**



**实验一： 变化参数n**

**实验结果：当n越大，吞吐率越高。**

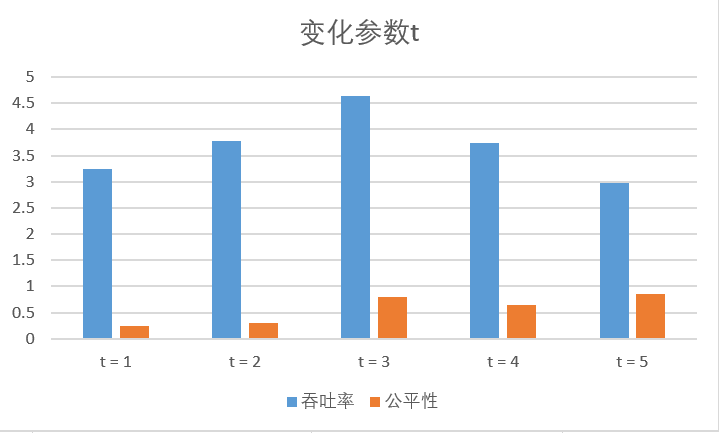




**实验二： 变化参数t**

**实验结果：实验之前估计的是吞吐率会和t成反比，但是实验结果反映的却不是这样。分析应该是同猴子选梯子的策略有关系，t过小，一次有过多的猴子在等待和拥挤到桥上，也会使吞吐率下降。最后吞吐率会呈现“中间高，两边低”**

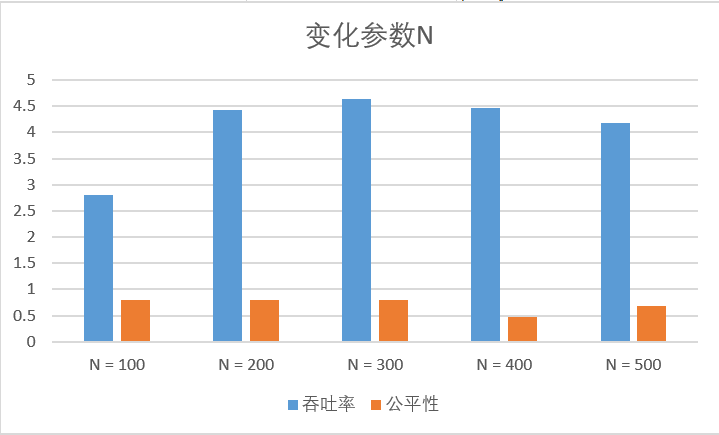
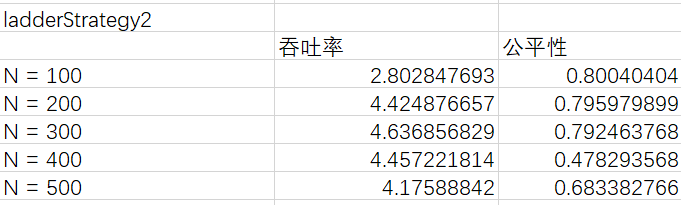




**实验三：变化参数N**

**实验结果：在LadderStrategy2下，参数N和吞吐率也没有特定的线性关系。**

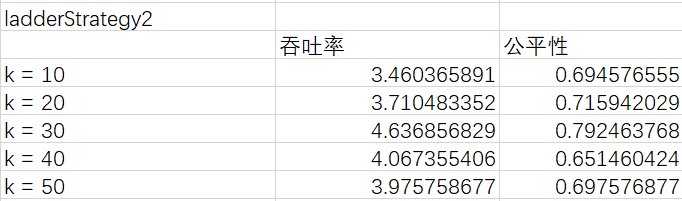
**还是“中间高，两边低”。**

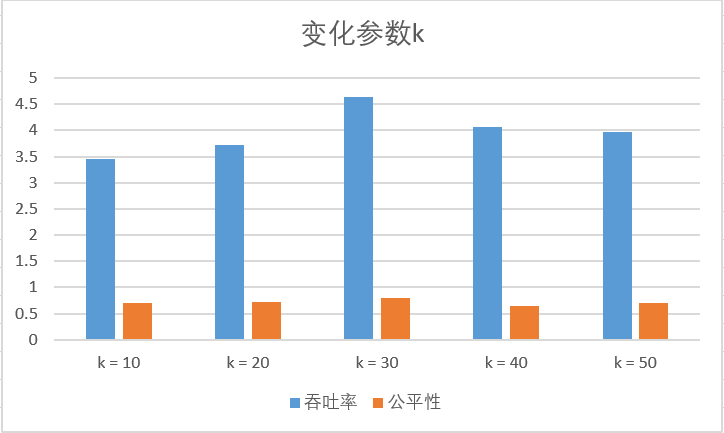


**实验四：变化参数k**

**实验结果：在LadderStrategy2下，参数k和吞吐率也没有特定的线性关系。**

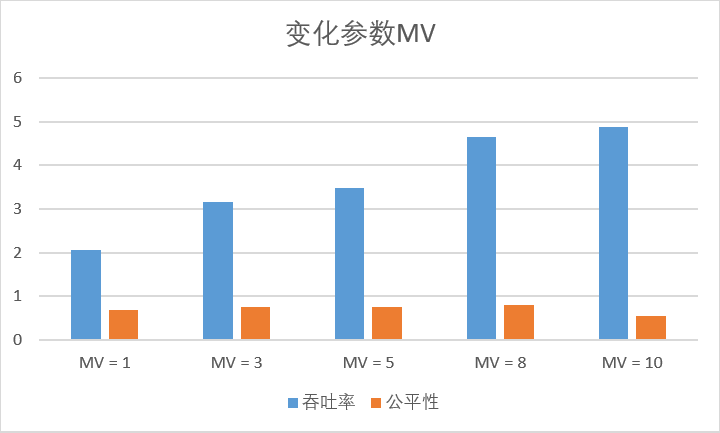
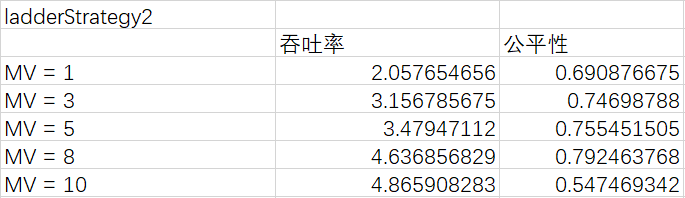
**呈现出两边低，中间高。**





**实验五：变化参数MV**

**实验结果：吞吐率和MV大小成正比，公平性无法保证。**

****

### 分析：吞吐率是否与各参数/决策策略有相关性？

根据以上的测试结果，吞吐率和各参数/决策策略关系紧密。而且有时候可能会因为参数和决策策略的选择问题，导致吞吐率在较大范围变化。

从v2的测试中，大致可以总结出以下规律：

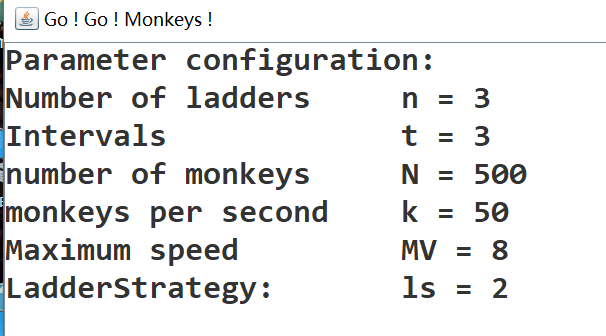
1. 对于我实现的三个选择策略，多数情况下，ladderStrategy2的吞吐率较高，在数据较多的时候，ladderStrategy3的吞吐率一般，并且公平性下降的较严重。

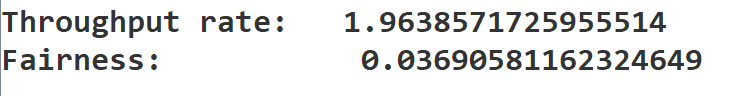
2. 各个参数和吞吐率关系的总结：n和吞吐率成正比；N、t、k和吞吐率的关系是“中间高，两边低”，即较大或较小都会对吞吐率产生坏的影响；MV和吞吐率成正比。

### 压力测试结果与分析

**压力测试一.**

参数配置：



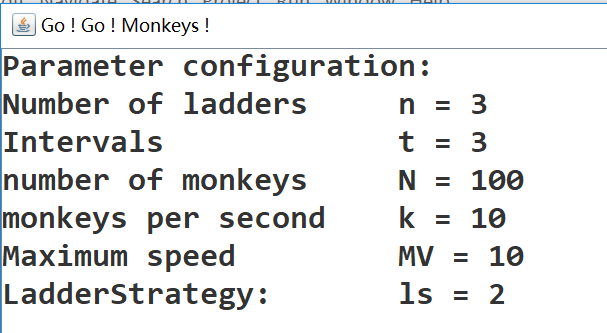


测试结果与分析：

当梯子数只有3，猴子数达到300，并且3秒就会产生50只猴子时，测试的吞吐率还很理性，能有将近2。但是由于过多猴子产生了便一起等待，上梯子的时候并没有根据产生的时间早而先上梯子，因此公平性较差。

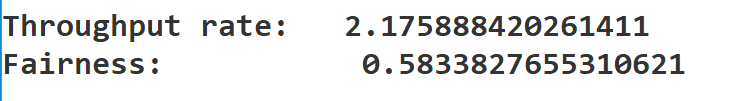
**压力测试二.**

参数配置：



测试结果与分析：

最大速度MV为10，此时产生的猴子随机出来的速度差异会很大。这样导致了可能在梯子上前面的猴子速度较慢，后面的猴子行进速度也变慢的结果。吞吐率和公平性都不高。



## 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 | 2.606633012138221 | 0.7924414715719063 |
| 第2次模拟 | 2.5019807347483423 | 0.8001114827201784 |
| 第3次模拟 | 2.765687590980986 | 0.8234732532732941 |
| 第4次模拟 | 2.8545665564433231 | 0.7667781254191272 |
| 第5次模拟 | 2.2437458989035675 | 0.7962374274292848 |
| 第6次模拟 | 2.9566748358483769 | 0.728467934124787 |
| 第7次模拟 | 2.5346874390784839 | 0.8651243172541246 |
| 第8次模拟 | 2.6843620976434234 | 0.7831546125321498 |
| 第9次模拟 | 2.6541896903700331 | 0.7914766286521876 |
| 第10次模拟 | 2.6339125014267024 | 0.7901226309921962 |
| 平均值 | 2.6896621128512946 | 0.8023652347671242 |
| Competiton\_2.txt |  |  |
| 第1次模拟 | 5.748183573990619 | 0.6430701402805611 |
| 第2次模拟 | 6.056568348373811 | 0.6847214428857715 |
| 第3次模拟 | 6.631647352432473 | 0.7612479186284124 |
| 第4次模拟 | 5.412321864681723 | 0.7812476513466871 |
| 第5次模拟 | 5.123767812649771 | 0.6421477671264214 |
| 第6次模拟 | 6.012616489721649 | 0.6514986126471926 |
| 第7次模拟 | 5.826146974871247 | 0.6921749712347012 |
| 第8次模拟 | 5.912498712548712 | 0.7679607095124197 |
| 第9次模拟 | 6.521946174912463 | 0.7177182498124875 |
| 第10次模拟 | 6.313928526329082 | 0.7721122244488978 |
| 平均值 | 6.126512349873214 | 0.7312974612980921 |
| Competiton\_3.txt |  |  |
| 第1次模拟 | 1.7273702960474397 | 0.66505050505050504 |
| 第2次模拟 | 1.8205690292814512 | 0.6272727272727273 |
| 第3次模拟 | 1.5343237462309926 | 0.6812635736461872 |
| 第4次模拟 | 1.9603603503603603 | 0.6912534791256397 |
| 第5次模拟 | 1.7821651747712646 | 0.6426148192874715 |
| 第6次模拟 | 2.0723182043726523 | 0.7003246234698762 |
| 第7次模拟 | 2.1731247247539011 | 0.6934687236477653 |
| 第8次模拟 | 1.9832645023277712 | 0.6821974635001298 |
| 第9次模拟 | 1.6312846327483273 | 0.6912353917517247 |
| 第10次模拟 | 1.7694675414283877 | 0.6898989898989899 |
| 平均值 | 1.8576325767852312 | 0.6823124771290367 |

# 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 |
| 2019.6.1 | 14:00-18:00 | 阅读实验 | 按计划完成 |
| 2019.6.1 | 18:30-23:00 | 通读实验要求，确定整体思路 | 按计划完成 |
| 2019.6.2 | 15:30-23:00 | 设计Monkey类和Ladder类 | 比预计晚2小时 |
| 2019.6.3 | 13:00-18:00 | 完成猴子生成器 | 比预计晚1小时 |
| 2019.6.3 | 19:00-23:00 | 选梯子的策略模式 | 未完成 |
| 2019.6.4 | 14:00-17:30 | 日志和GUI | 按计划完成 |
| 2019.6.4 | 18:30-22:30 | 猴子模拟器v1 | 按计划完成 |
| 2019.6.5 | 14:00-17:00 | 猴子模拟器v2和v3 | 提前半小时完成 |
| 2019.6.5 | 17:30-23:00 | 优化代码，提高吞吐率和公平性 | 按计划完成 |
| 2019.6.6 | 13:30-15:30 | 测试数据 | 未完成 |
| 2019.6.6 | 18:30-23:00 | 完成报告 | 按计划完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 刚开始对题意和具体要求理解不清 | 反复阅读指导 |
| 多线程发生Monkey过梯子的时候发生死锁 | 重新翻译上课使用的课件，结合着java多线程的相关书籍，加深理解 |
| 将控制台输出到GUI不成功 | 在网上查阅JTextArea的使用方式，不断学习 |
| 刚开始吞吐率和公平性不高 | 不断测试和优化代码，将synchronized作用的粒度变小 |

# 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

## 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：复杂在我们如果对锁机制掌握不准确，可能会导致程序出错或者发生死锁。能体验到，当我在多处使用synchronized修饰时，有时因为使用不当，导致程序变成几乎是串行执行，效率非常低。当我们减小synchronized作用粒度时，就会体验到多线程程序在性能方面的极大改善。

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：保证threadsafe的四种策略我都使用到了限制数据共享、共享不可变数据、共享线程安全的可变数据和同步机制（Synchronization）。只是实现起来难度不同。主要是使用最后的同步机制。在提升性能的前提下，我确保了threadsafe，程序的正确性。

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。程序中出现过梯子上两只相向而行的猴子，导致死锁。后来我定位到了选择梯子的时候，可能会线程不安全，于是我对梯子的集合进行synchronized修饰，问题得以解决。

1. 关于本实验的工作量、难度、deadline。

答：工作量不大，难度一般，deadline较宽裕。

1. 到此为止你对《软件构造》课程的意见和建议。

答：这是我非常喜欢的一门课，课程教会了我很多的知识，带我走进了软件构造的新世界。但是由于课程进度快，且ppt都是英文，因此课程难度较大，整个课程的设计应该还有可以完善的地方。因为课程ppt内容过多，建议多考核重点部分，不过多在意细枝末节。虽然课程已经结束了，但是对软件构造的学习不会停止，希望这门课越来越好！！