哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2019 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	张景润
学号	1172510217
班号	1703002
电子邮件	2584363094@qq.com
手机号码	18530272728

目录

1 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

2 实验环境配置

- 建立本地仓库并于远程仓库关联

  1，README 文件：echo “Lab6-1172510217” >> README.md

  2，初始化本地仓库：git init

  3，添加到暂存区：git add README.md

  4，提交到本地仓库 git commit -m “first commit”

  5，建立与远程仓库的联系：git remote add origin https://github.com/ComputerScienceHIT/Lab6-1172510217.git

  6，推送到远程仓库：git push -u origin master
- Lab6 的 URL 地址（我的学号是：1172510217）

  https://github.com/ComputerScienceHIT/Lab6-1172510217

3 实验过程

3.1 ADT 设计方案

3.1.1 Monkey implements Runnable

- 他的 spec 如截图

```
/**
 * AF:id,猴子的唯一标识
 * direction,0=L->R;1=R->L
 * speed:猴子在梯子上的爬行速度,正整数
 * RI: speed等均为整数
 * safe from exposure:
 * 这个类中所有的成员变量除了liveTime都是private且一经创建不会改变，不会有表示泄漏的问题
 * thread safe:
 * for这个类没有多线程公共的数据，不会有线程安全问题.
 */
```

- 属性

StringBuilder *logBuilder*	便于日志构造
Integer id	猴子唯一标识
Integer direction	猴子的方向
Integer speed	猴子的速度
Integer bornTime	猴子的出生时间
Integer liveTime	猴子的存活时间
LadderStrategy ladderStrategy	选择的过河策略

- 方法

toString()	重写 toString
run()	线程的 run 方法
getter()	所有的属性获取方法
directionString()	将 0 或 1 的方向表示为 L->R 和 R->L
Monkey()	构造器
getLogString()	获取猴子的经历信息

3.1.2 Ladder

- 他的 spec 如截图

```
/**
 * AF:id,梯子的唯一标识
 * length,梯子的长度,为正整数
 * RI: 属性均为正数
 * safe from exposure:
 * for这个类中所有的成员变量一经创建不会改变，不会有表示泄漏的问题
 * thread safe:
 * for这个类没有多线程公共的数据，不会有线程安全问题.
 */
```

- 属性

Integer length	梯子踏板数
Integer id	梯子唯一标识

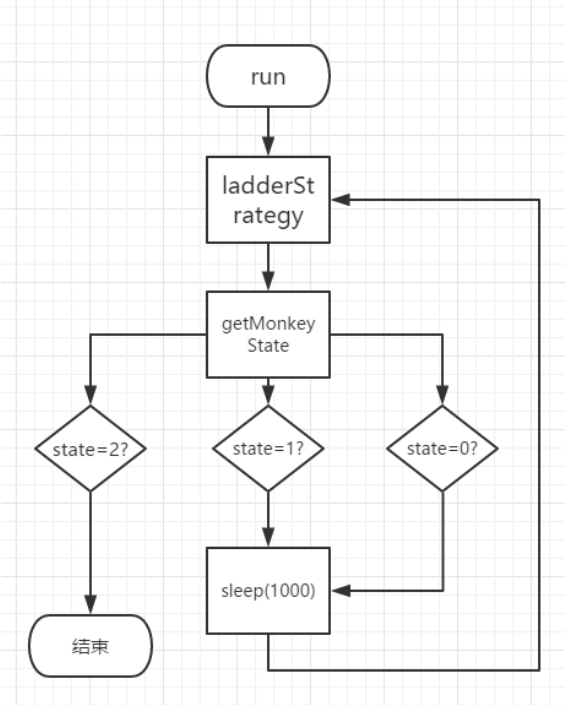
- 方法

toString()	重写 toString
ladder()	梯子构造器
getter()	所有的属性获取方法

3.2 其他类的简单介绍

- SystemConstant.java
  为简单化参数构造的一个类，其中设置了两个参数：系统加速常数 `Integer speedUp` 以及设计的策略的数目 `Integer strategyNum`
  我可以通过简单的修改这个类中的参数，从而实现加速猴子过河过程的模拟。
- LadderFactory.java：用于管理所有的梯子以及被占用的情况
- MonkeyFactory.java：用来管理所有的猴子以及其过桥状态
- MonkeyCrossRiverSystem.java：用来读取配置文件建立系统
- Gui.java：用于实现 GUI
- 四个策略类：用于实现具体的猴子过河策略

3.3 Monkey 线程的 run()的执行流程图

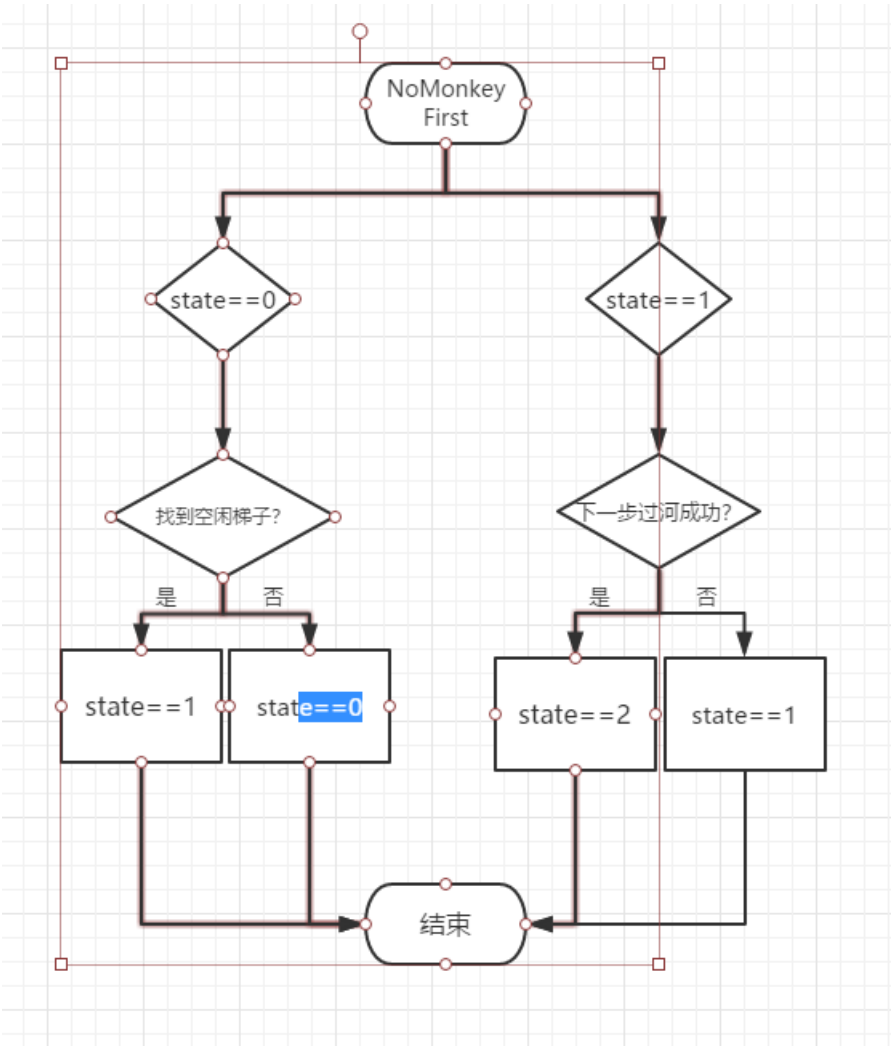


3.4 至少两种“梯子选择”策略的设计与实现方案

说明 1：我实现了三种策略：两种 pdf 上给出的策略+一种自己设计的策略
说明 2：这三种策略均实现接口 LadderStrategy，采用了 strategy 设计模式

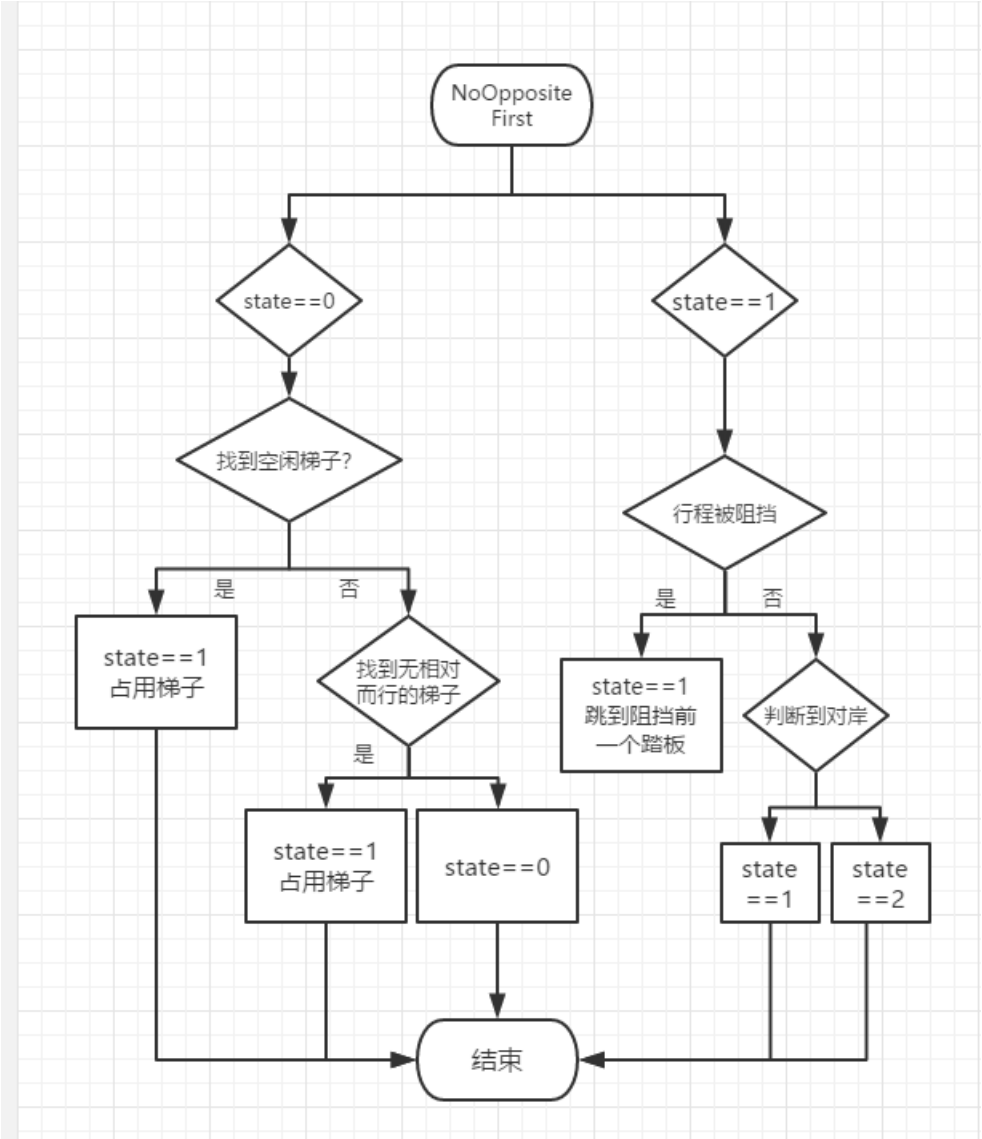
3.4.1 策略 1：NoMonkeyFirst

猴子只选择没有被占用的梯子，若均被占用，则只是简单的原地等待。



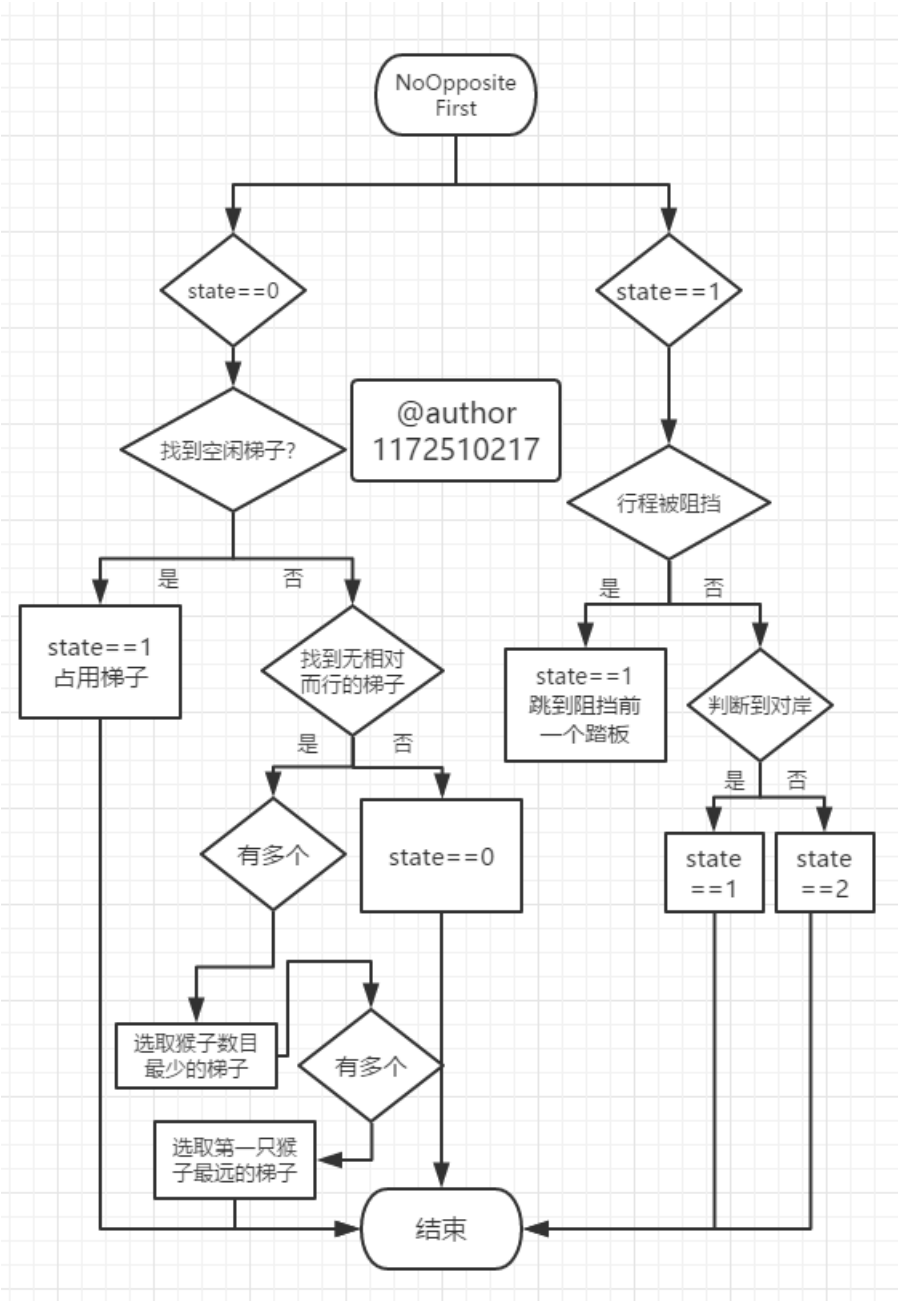
3.4.2 策略 2：NoOppositeFirst

猴子优先选择未被占用的梯子，若均被占用，则选取无相对而行猴子的梯子，若这样的选择有很多，进行随机选取。



3.4.3 策略 3（可选）：MyStrategy

猴子优先选择未被占用的梯子，若均被占用，则选取无相对而行猴子的梯子，若这样的选择有很多，则优先选择梯子上猴子数目最少的一个梯子；若数目仍然一致，则选取梯子上的第一只猴子距离自己最远的梯子。



3.5 “猴子生成器”MonkeyGenerator

- 该方法在读取配置文件并创建系统的时候被调用。目的是隔一段时间产生一定数目的猴子，并启动他们的线程。
- 该方法内部有一个 Timer 对象和一个 TimeTask 对象。目的是可以做到隔一段时间执行一次 run 方法。
- 具体的 run 方法内部代码如下。不断生成猴子，当生成的猴子数目等于要生成的猴子数目时，调用 Timer 对象的 cancel 方法结束此线程。生成猴子的同时将猴子线程启动。

```
Timer timer = new Timer();
TimerTask task = new TimerTask() {

  @Override public void run() {
    for (int i = 0; i < num; i++) {
      int size = allMonkeys.size();
      if (size < monkeyNum) {
        Monkey monkey = new Monkey(size + 1, (int) (Math.random() * 2),
            (int) (Math.random() * maxSpeed + 1), size / num * time,
            createLadderStrategy());
        MonkeyFactory.addMonkey(monkey);
        Thread monkeyThread = new Thread(monkey);
        monkeyThread.start();// 启动猴子过河进程
      } else {
        timer.cancel();
      }
    }
  }
};
timer.scheduleAtFixedRate(task, 0, time * 1000 / SystemConstant.getSpeedUp()
```

- 同时注意到该 Timer 对象调用了 scheduleAtFixedRate 方法，且参数中右 SystemConstant.getSpeedUp 方法。目的是可以做到加速此模拟猴子过河系统的进行。

3.6 如何确保 threadsafe？

- 使用了线程安全的 Map 和 List 保证线程安全。
  如截图，我的集合结构的声明确保是线程安全的。

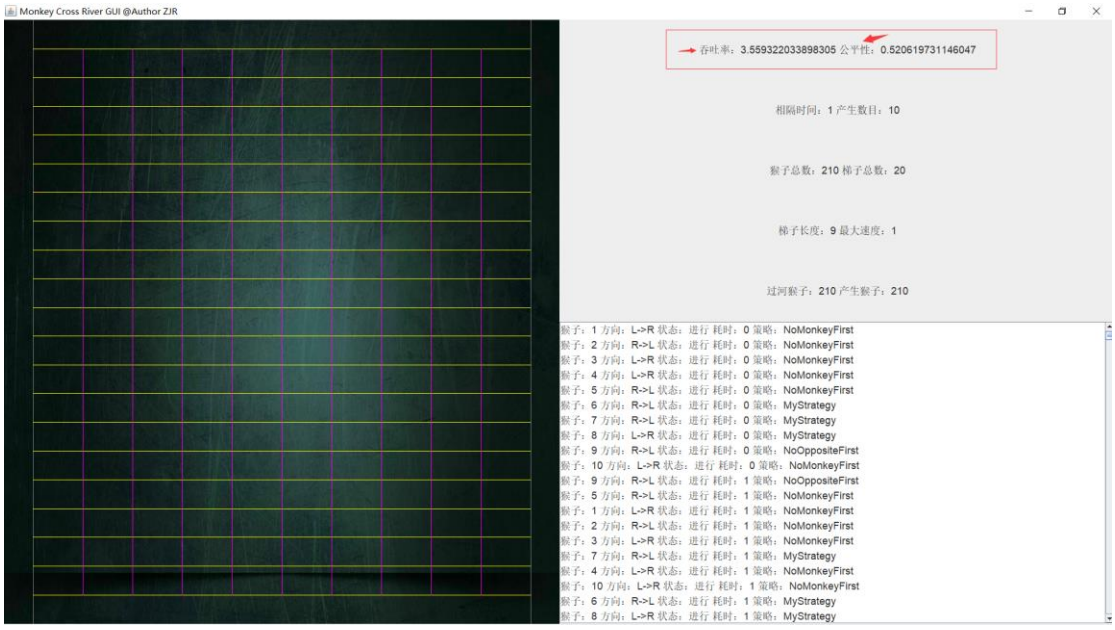
```
private static List<Monkey> allMonkeys =
    Collections.synchronizedList(new ArrayList<>());
private static Map<Integer, Integer> monkeyToState =
    Collections.synchronizedMap(new HashMap<>());// 猴子及其状态
```

- 使用线程锁 Synchronized 关键字。
  确保每一只猴子在进行自己的策略选择梯子时独占线程且不被中断。

```
@Override public void run() {
  liveTime = 0;
  Integer monkeyState = 0;
  do {
    synchronized (Monkey.class) {
      ladderStrategy.ladderStrategy(this);
      monkeyState = MonkeyFactory.getMonkeyState(id);
```

3.7 系统吞吐率和公平性的度量方案

- 注：吞吐率和公平性的度量只在所有猴子均已过河时，才会启动。
- 这些度量可以在 GUI 上显示，如截图。



- 遍历所有的猴子，得到最后一只过河猴子的存活时间和出生时间即可确定所有猴子的过河的最长时间，即可确定吞吐率
- 两层 for 循环遍历所有的猴子，判断两只猴子是否过河公平，若公平，则公平数++，否则公平数--。最终即可确定公平性度量常量。

3.8 输出方案设计（实现了动态化模拟过程）

3.8.1 日志-java 自带的 log

- 在 Monkey 类中设置了一个静态属性 Logger。确保所有的猴子在进行自己的线程时，均会在 Logger 中留下记录。

```
private static Logger logger =
    Logger.getLogger(MonkeyCrossRiverSystem.class.getSimpleName());//猴子日志
```

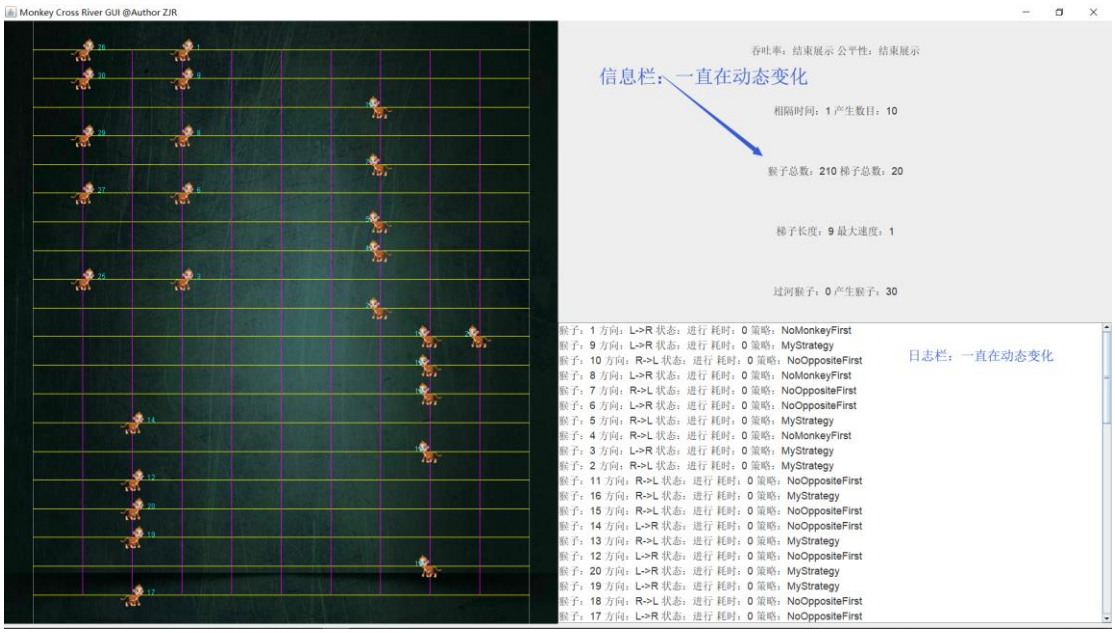
- 在 Monkey 的 run 方法中执行对 Logger 的记录。分三种情况：等待，进行和成功，分别进行单独记录。

```
if (monkeyState.equals(0)) {
  logger.info("猴子：" + id + " 方向：" + directionString() + " 状态：等待 耗时：" + liveTime
      + " 策略：" + ladderStrategy);
  logBuilder.append("猴子：").append(id).append(" 方向：").append(directionString())
      .append(" 状态：等待 耗时：").append(liveTime++).append(" 策略：")
      .append(ladderStrategy).append("\n");
} else if (monkeyState.equals(1)) {
  logger.info("猴子：" + id + " 方向：" + directionString() + " 状态：进行 耗时：" + liveTime
      + " 策略：" + ladderStrategy);
  logBuilder.append("猴子：").append(id).append(" 方向：").append(directionString())
      .append(" 状态：进行 耗时：").append(liveTime++).append(" 策略：")
      .append(ladderStrategy).append("\n");
} else {
  logger.info("猴子：" + id + " 方向：" + directionString() + " 状态：成功 耗时：" + liveTime
      + " 策略：" + ladderStrategy);
  logBuilder.append("猴子：").append(id).append(" 方向：").append(directionString())
      .append(" 状态：成功 耗时：").append(liveTime).append(" 策略：").append(ladderStrategy)
      .append("\n");
  break;
}
```

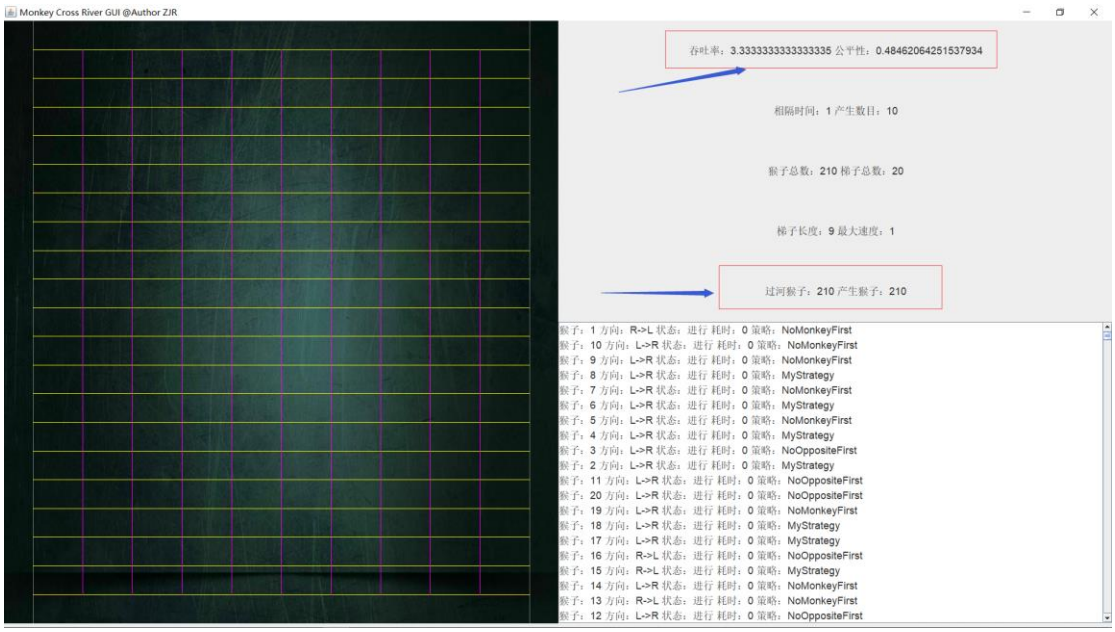
- 终端显示如截图，做到了信息清晰易读。

```
信息: 猴子：102 方向：L->R 状态：进行 耗时：43 策略：NoMonkeyFirst
六月 16, 2019 5:54:48 下午 adt.Monkey run
信息: 猴子：39 方向：R->L 状态：进行 耗时：49 策略：NoMonkeyFirst
六月 16, 2019 5:54:48 下午 adt.Monkey run
信息: 猴子：39 方向：R->L 状态：成功 耗时：50 策略：NoMonkeyFirst
六月 16, 2019 5:54:48 下午 adt.Monkey run
信息: 猴子：119 方向：L->R 状态：成功 耗时：43 策略：NoMonkeyFirst
六月 16, 2019 5:54:48 下午 adt.Monkey run
信息: 猴子：180 方向：R->L 状态：进行 耗时：38 策略：NoMonkeyFirst
六月 16, 2019 5:54:48 下午 adt.Monkey run
信息: 猴子：102 方向：L->R 状态：成功 耗时：44 策略：NoMonkeyFirst
六月 16, 2019 5:54:48 下午 adt.Monkey run
信息: 猴子：180 方向：R->L 状态：成功 耗时：39 策略：NoMonkeyFirst
```

3.8.2 GUI 界面展示



- GUI 展示划分：动态化猴子过河区；右上为配置文件信息展示区以及过河猴子数目、产生猴子数目；右下为日志栏
- 这三部分均是动态变化的：日志不断刷新、猴子数目不断刷新、猴子过河动画不断刷新。
- 过河结束 GUI 展示

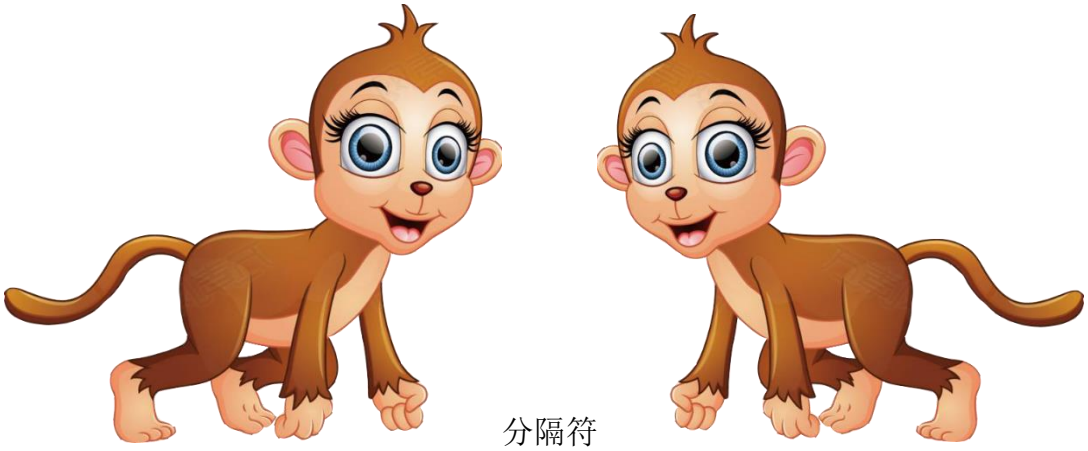


3.8.3 动态显示整个过程

一点说明

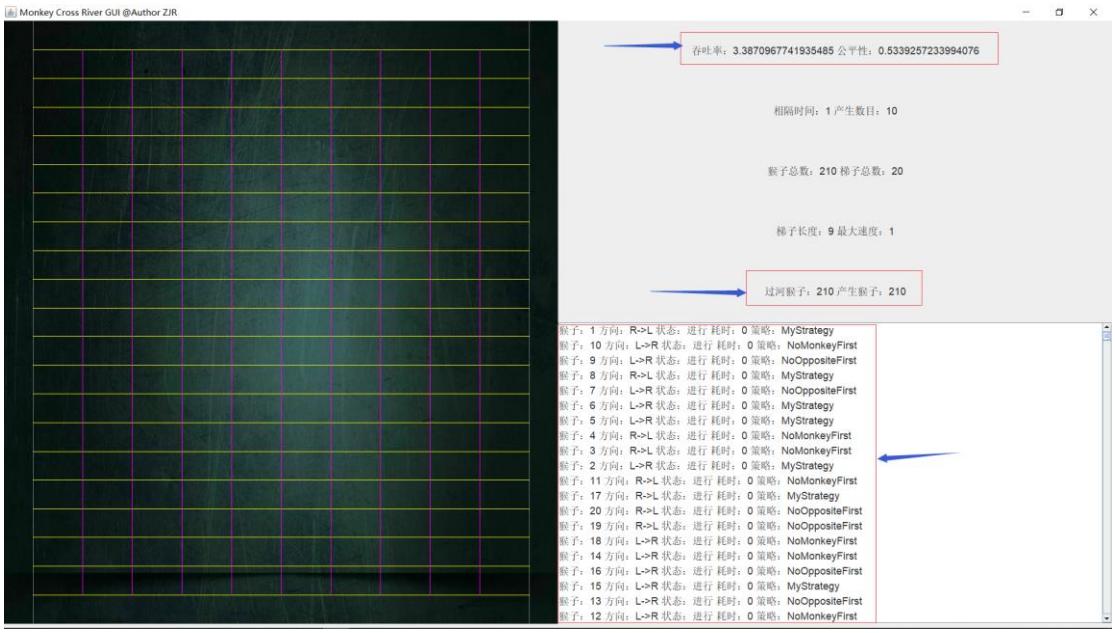
为了更好的观看理解整个过程，我特意选取了一张猴子的图片，同时将其进行对称镜像处理，从而可以很好的看到猴子运行的方向；

同时我还运用 PS 技术对选取的图片进行了抠图处理，目的是扣去图片上的白色背景板。如截图。



分隔符

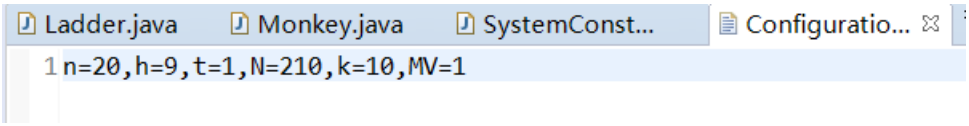
- 整个过程是动态显示的：猴子可移动、右上信息栏不断刷新、右下日志栏不断刷新、吞吐率以及公平性会自动在过河结束自动显示。
- 在截图上用箭头标识出所有的可以动态化的组件



3.9 猴子过河模拟器 v1

3.9.1 参数如何初始化

- 在 src 文件夹下新建了一个配置文件，只需要修改配置文件，即可完成对参数的初始化。文件目录 src/ConfigurationFile/Configuration1.txt



- 通过调用 MonkeyCrossRiverSystem.readFileAndCreateSystem 的方法。实现参数初始化，并建立系统。

3.9.2 使用 Strategy 模式为每只猴子随机选择决策策略

- 在 MonkeyFactory 类内部调用 createLadderStrategy 方法实现为猴子随机分配一种策略。注意需要在 SystemConstant 类中手动修改一个参数 strategyNum，从而确保随机数可以产生数字 1，2，3，从而进行随机策略分配。
- 分析 createLadderStrategy 方法。如截图

```
Integer num = SystemConstant.getStrategyNum();
LadderStrategy ladderStrategy = new NoMonkeyFirst();
int randomNum = (int) (Math.random() * num) + 1;
if (randomNum == 2) {
  ladderStrategy = new NoOppositeFirst();
}
if (randomNum == 3) {
  ladderStrategy = new MyStrategy();
}
return ladderStrategy;
```

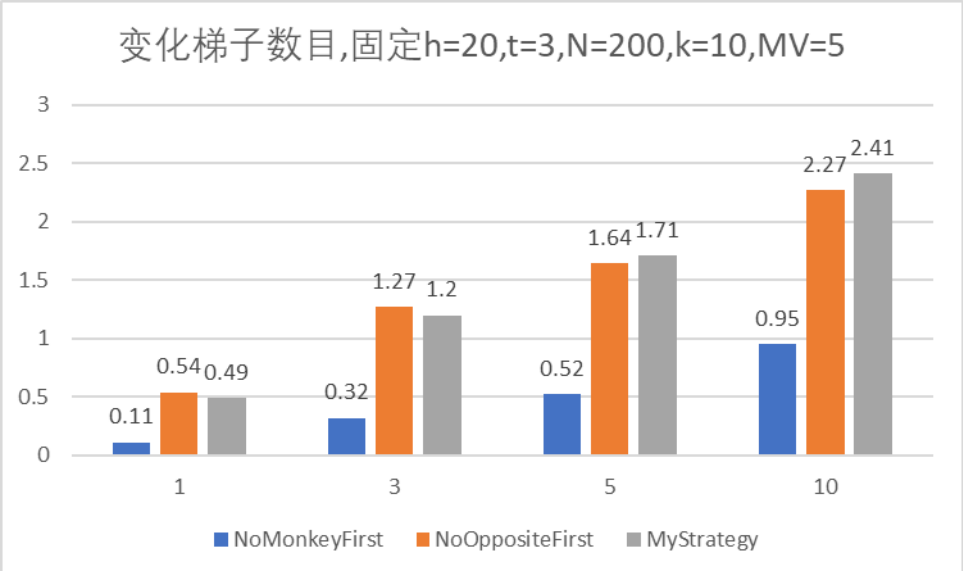
3.10 猴子过河模拟器 v2

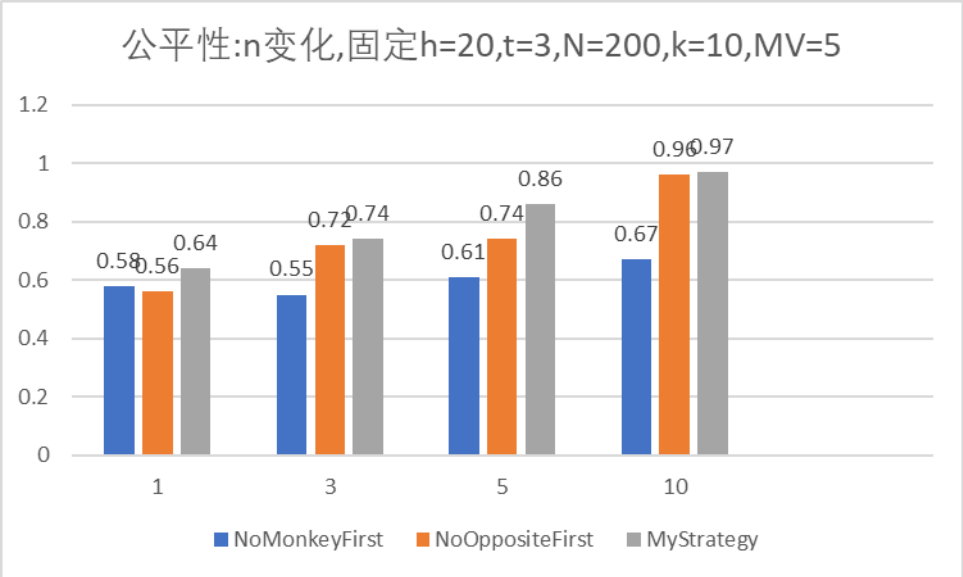
五点说明：
1，实现所有猴子选择同一种策略的方法是：
2，实现更改猴子的初始化参数的方法是：
3，由于原来实验中的 3.9.1 和 3.9.2 可以进行通过图表进行合并处理，因此我将其改为现在的 3.9.1 加以处理分析；
4，为了统计以及作图的方便，做了两个操作：吞吐率和公平性数据统一取两位有效数字，设置了过河进行速率比例常数为 100 表示加速这个过程 100 倍而不失正确性；
5，截图为 GUI 的原始数据，作图使用的是取两位小数的真实数据。

3.10.1 对比分析：固其他参数，不同的决策策略和某个参数之间的关系对比

3.10.1.1 对参数梯子数目进行变化(截图分别为 n=1, 3, 5, 10)

n	h	t	N	k	MV
1-10	20	3	200	10	5





1，对于 NoMonkeyFirst 策略

吞吐率：0.11389521640091116 公平性：0.5811557788944723

吞吐率：0.32102728731942215 公平性：0.5474874371859296

吞吐率：0.5181347150259067 公平性：0.6125125628140704

吞吐率：0.9523809523809523 公平性：0.6706030150753769

2，对于 NoOppositeFirst 策略

吞吐率：0.5434782608695652 公平性：0.5580402010050252

吞吐率：1.2658227848101267 公平性：0.7229145728643216

吞吐率：1.639344262295082 公平性：0.7412060301507538

吞吐率：2.272727272727273 公平性：0.9581909547738694

3，对于 MyStrategy 策略

吞吐率：0.4889975550122249 公平性：0.642713567839196

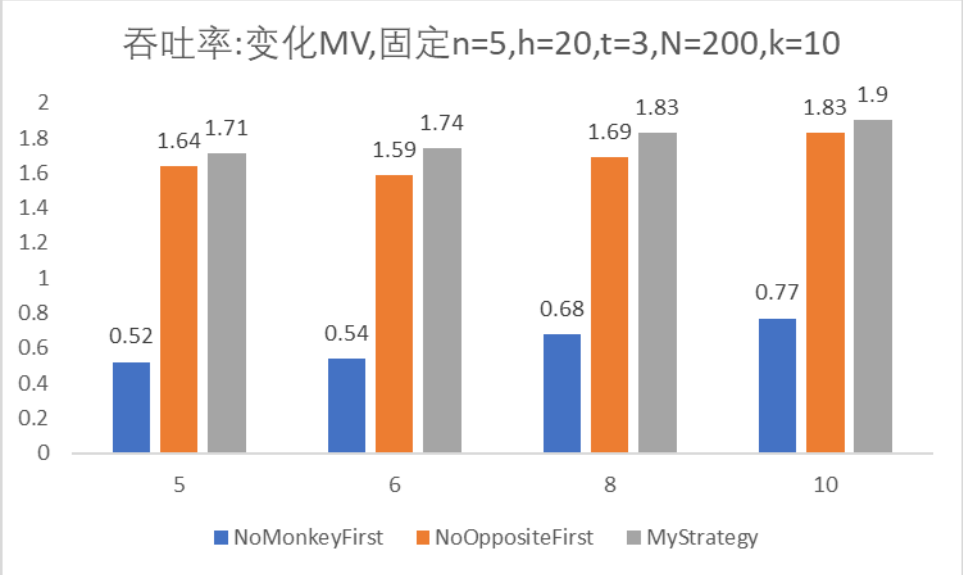
吞吐率：1.1976047904191616 公平性：0.7398492462311558

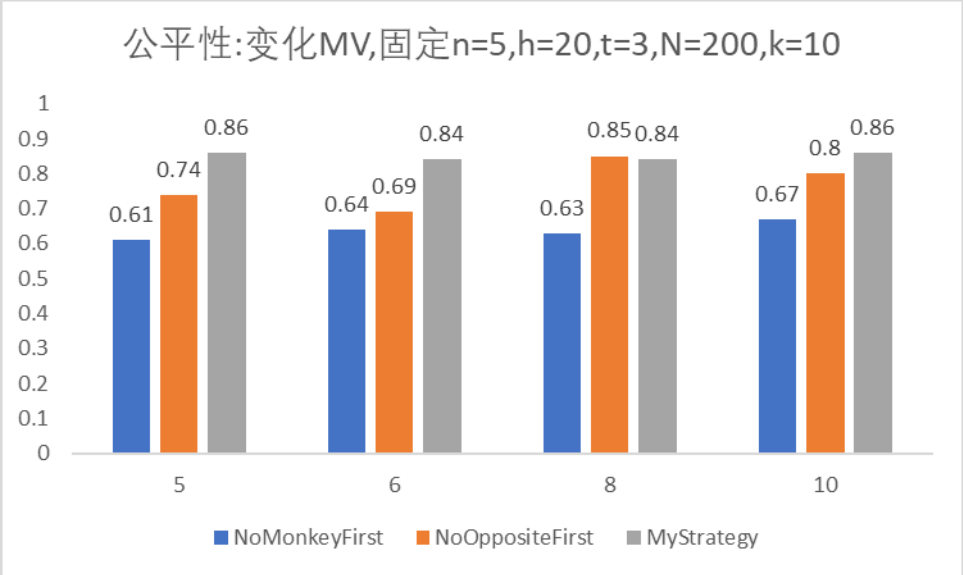
吞吐率：1.7094017094017093 公平性：0.8587939698492463

吞吐率：2.4096385542168677 公平性：0.9673366834170855

3.10.1.2 对参数最大速度进行变化(截图分别为 MV=5,6,8,10)

n	h	t	N	k	MV
5	20	3	200	10	5-10





1，对于 NoMonkeyFirst 策略

吞吐率：0.5181347150259067 公平性：0.6125125628140704

吞吐率：0.5449591280653951 公平性：0.641859296482412

吞吐率：0.6802721088435374 公平性：0.6309045226130653

吞吐率：0.7722007722007722 公平性：0.6747236180904522

2，对于 NoOppositeFirst 策略

吞吐率：1.639344262295082 公平性：0.7412060301507538

吞吐率：1.5873015873015872 公平性：0.6916582914572864

吞吐率：1.694915254237288 公平性：0.8499497487437186

吞吐率：1.834862385321101 公平性：0.8010050251256281

3，对于 MyStrategy 策略

吞吐率：1.7094017094017093 公平性：0.8587939698492463

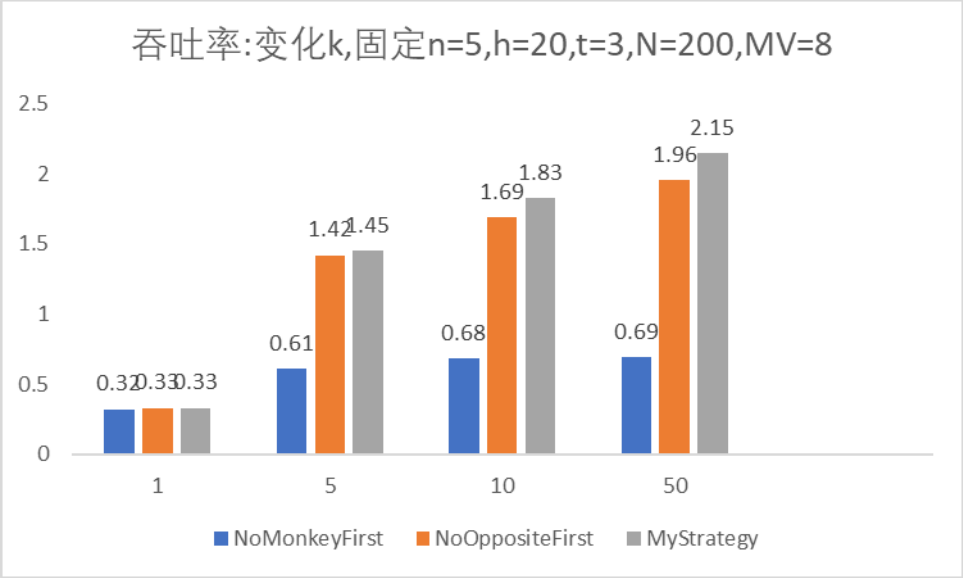
吞吐率：1.7391304347826086 公平性：0.8379899497487437

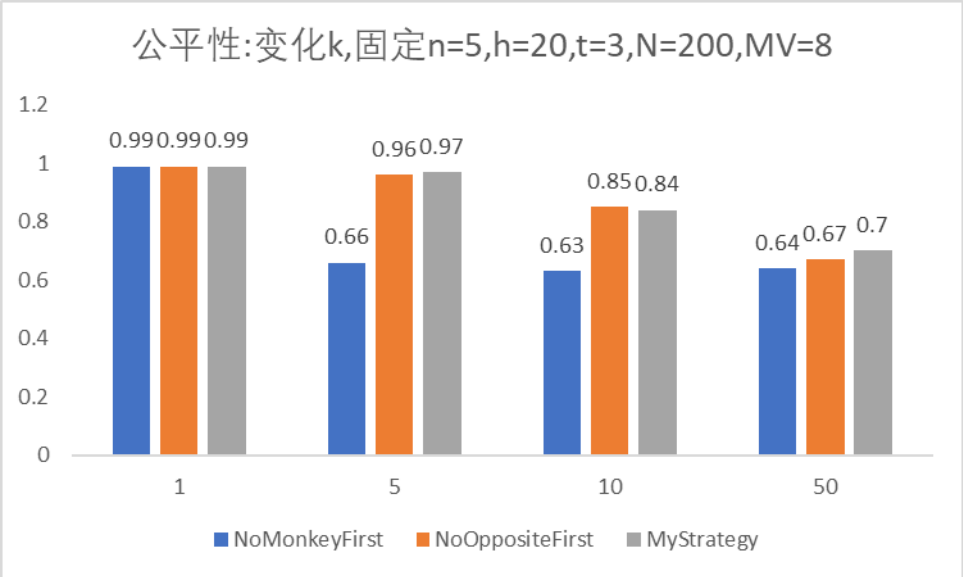
吞吐率：1.834862385321101 公平性：0.8387437185929648

吞吐率：1.9047619047619047 公平性：0.8588442211055276

3.10.1.3 对参数 k 进行变化(截图分别为 k=1, 5, 10, 50)

n	h	t	N	k	MV
5	20	3	200	1-50	8





1，对于 NoMonkeyFirst 策略

吞吐率：0.322061191626409 公平性：0.9926633165829146

吞吐率：0.6134969325153374 公平性：0.6611557788944724

吞吐率：0.6802721088435374 公平性：0.6309045226130653

吞吐率：0.6896551724137931 公平性：0.6378894472361809

2，对于 NoOppositeFirst 策略

吞吐率：0.32894736842105265 公平性：0.9930150753768844

吞吐率：1.4184397163120568 公平性：0.9604522613065327

吞吐率：1.694915254237288 公平性：0.8499497487437186

吞吐率：1.9607843137254901 公平性：0.6954773869346733

3，对于 MyStrategy 策略

吞吐率：0.33112582781456956 公平性：0.9943718592964824

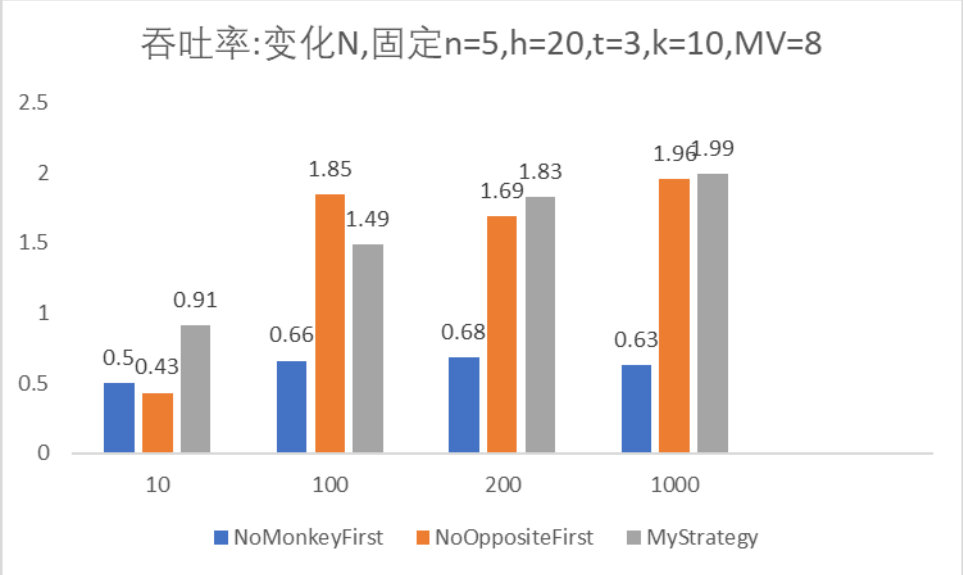
吞吐率：1.4492753623188406 公平性：0.9744723618090452

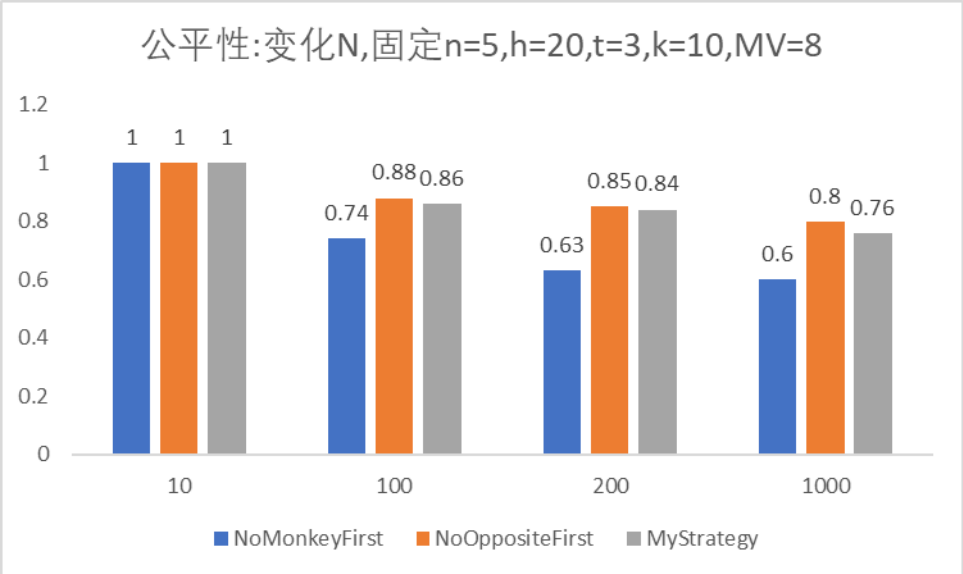
吞吐率：1.834862385321101 公平性：0.8387437185929648

吞吐率：2.150537634408602 公平性：0.6997989949748744

3.10.1.4 对参数 N 进行变化(截图分别为 N=10, 100, 200, 1000)

n	h	t	N	k	MV
5	20	3	2-1000	10	8





1，对于 NoMonkeyFirst 策略

吞吐率：0.5 公平性：1.0

吞吐率：0.6578947368421053 公平性：0.7436363636363637

吞吐率：0.6802721088435374 公平性：0.6309045226130653

吞吐率：0.6325110689437066 公平性：0.5972312312312312

2，对于 NoOppositeFirst 策略

吞吐率：0.43478260869565216 公平性：1.0

吞吐率：1.8518518518518519 公平性：0.8763636363636363

吞吐率：1.694915254237288 公平性：0.8499497487437186

吞吐率：1.9607843137254901 公平性：0.7993553553553554

3，对于 MyStrategy 策略

吞吐率：0.9090909090909091 公平性：1.0

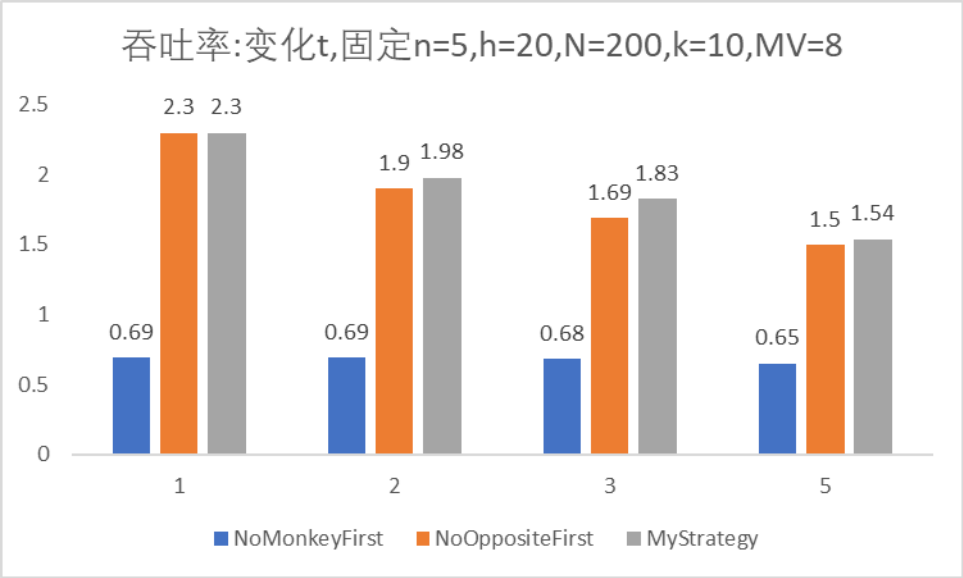
吞吐率：1.492537313432836 公平性：0.8646464646464647

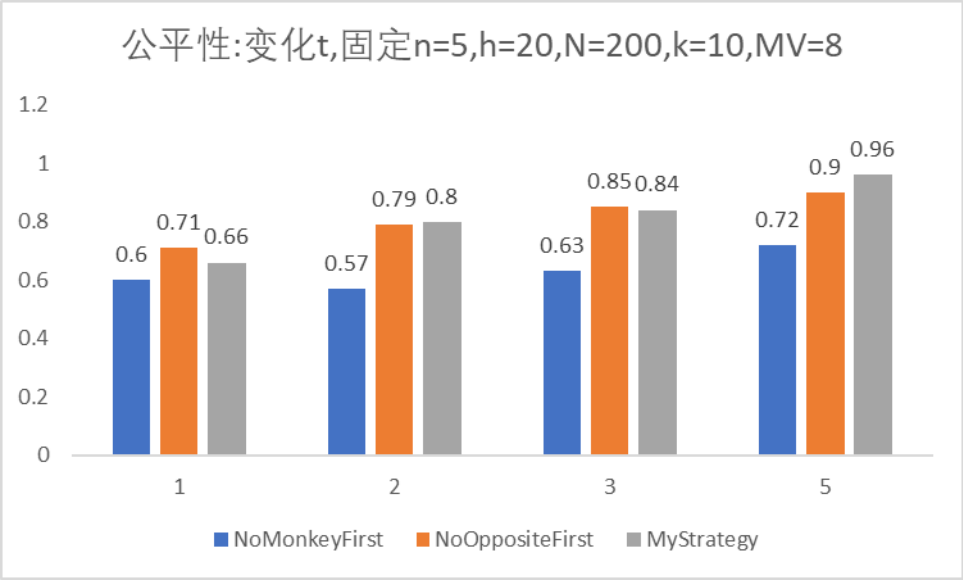
吞吐率：1.834862385321101 公平性：0.8387437185929648

吞吐率：1.9920318725099602 公平性：0.75802002002002

3.10.1.5 对参数 t 进行变化(截图分别为 t=1, 2, 3, 5)

n	h	t	N	k	MV
5	20	1-5	200	10	8





1，对于 NoMonkeyFirst 策略

吞吐率：0.6896551724137931 公平性：0.5951256281407035

吞吐率：0.6920415224913494 公平性：0.5712060301507538

吞吐率：0.6802721088435374 公平性：0.6309045226130653

吞吐率：0.6493506493506493 公平性：0.7155276381909548

2，对于 NoOppositeFirst 策略

吞吐率：2.2988505747126435 公平性：0.7111055276381909

吞吐率：1.9047619047619047 公平性：0.7892964824120603

吞吐率：1.694915254237288 公平性：0.8499497487437186

吞吐率：1.5037593984962405 公平性：0.9015577889447236

3，对于 MyStrategy 策略

吞吐率：2.2988505747126435 公平性：0.6577386934673367

吞吐率：1.9801980198019802 公平性：0.7968341708542713

吞吐率：1.834862385321101 公平性：0.8387437185929648

吞吐率：1.5384615384615385 公平性：0.9565829145728644

3.10.2 分析：吞吐率是否与各参数/决策策略有相关性？

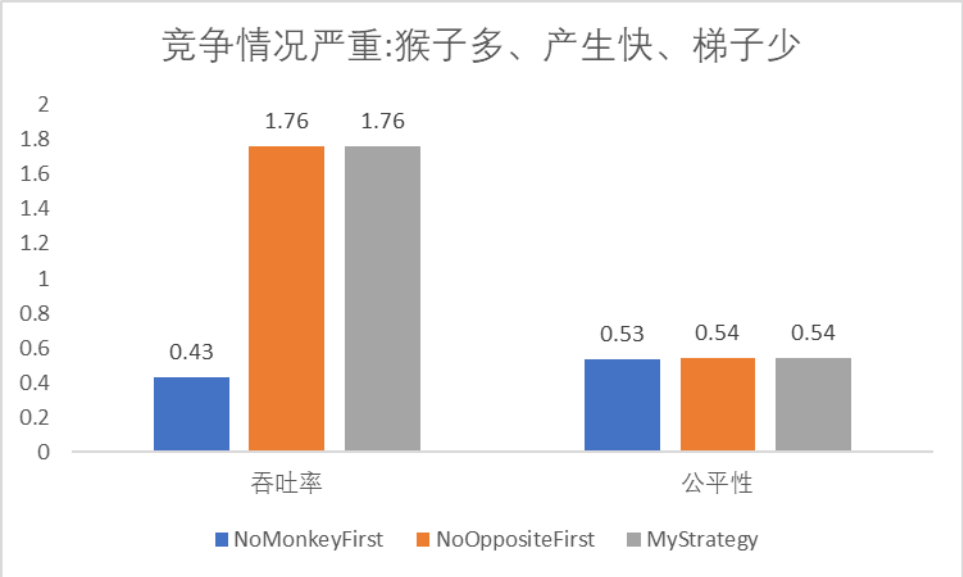
总结：与参数配置以及决策策略关系很大，下面对上一个部分的 5 组实验进行细致的分析。

- 第一组：梯子数目越多，显然吞吐率越高以及公平性越高；同时发现 MyStrategy 策略是最优的。
- 第二组：最大速度越大，说明平均一只猴子过河所需时间越短，那么岸上猴子需要等待时间越短，公平性越高，且 MyStrategy 是最优的设计。
- 第三组：每次产生的猴子数目越多，整体吞吐率和公平性均得到了提高。说明整体的性能未得到完全的挖掘，未达到阙值。但是公平性却逐渐下降，说明等待的猴子越来越多，随机性也越来越大。
- 第四组：猴子总数越多，整体的公平性越低，但是吞吐率在猴子数目较多时，变化不太明显。因为猴子总数使整个参数配置达到了阈值，在进行改变时，变化不会太大。
- 第五组：猴子产生间隔越短，整体的吞吐率和公平性越低，因为有大量的猴子进行等待，而且梯子数目不是特别多，只要第一波猴子的方向均为从左到右或者从右到左，那么就会有的大量的不同方向的猴子进行等待。而我们的策略选择是选择无对向行驶的猴子的梯子，这样会使所有的梯子被同一个方向的猴子占用，那么恶性循环，反方向的猴子不会得到梯子的使用权，直到这些一个方向的猴子全部过河。因此整体的公平性逐渐降低。而且说明系统的整体的吞吐率与第一波产生的猴子的方向的均匀性有极大的关系。

3.10.3 压力测试结果与分析

3.10.3.1 竞争情况极其严重：猴子多+产生快+梯子少

- 参数配置为：`n=3,h=20,t=1,N=1000,k=50,MV=8`
- 分析：MyStrategy 和 NoOppositeStrategy 策略模式的整体差别不大，因为梯子数目少，导致两种策略的梯子选择差别很小很小，但是公平性测度和吞吐率测度均好于 NoMonkeyStrategy 策略



1，对于 NoMonkeyFirst 策略

吞吐率：0.4291845493562232 公平性：0.5345525525525525

2，对于 NoOppositeFirst 策略

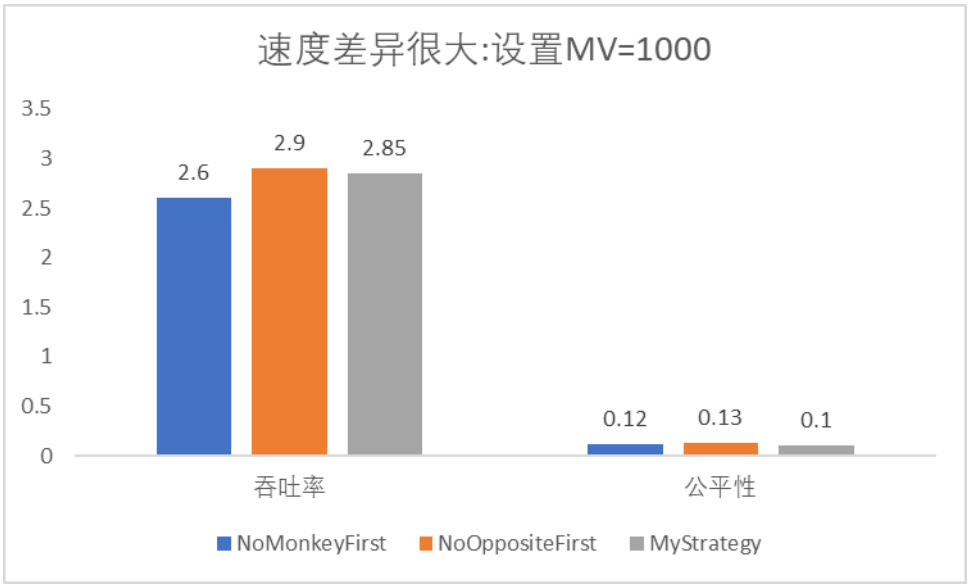
吞吐率：1.757469244288225 公平性：0.5444104104104104

3，对于 MyStrategy 策略

吞吐率：1.757469244288225 公平性：0.5443243243243243

3.10.3.2 猴子速度差异很大

- 参数配置为：`n=3,h=20,t=1,N=1000,k=50,MV=1000`
- 分析：速度很大导致不公平现象越来越多，但同时由于整体推进速度很快，使总体吞吐率接近极限 3（永远不可能超过 3）。



1，对于 NoMonkeyFirst 策略

```
吞吐率：2.6041666666666665 公平性：0.1245005005005005
```

2，对于 NoOppositeFirst 策略

```
吞吐率：2.8901734104046244 公平性：0.13442642642642644
```

3，对于 MyStrategy 策略

```
吞吐率：2.849002849002849 公平性：0.1036036036036036
```

3.11 猴子过河模拟器 v3

3.11.1 实现读取文件建立系统

- 均采用我自己设计的策略 MyStrategy
- 将原本的读取配置文件的输入文件改为下载的文件路径

```
("src/Spring2019_HITCS_SC_Lab6-master/Competition_2.txt");
```

- 从 v2->v3 思路改变

  原来思路是猴子创建的同时，便调用 Thread.start 方法；现在改为先产生所有猴子及其线程，再根据文件参数分析得到相关的时间间隔，一次产生的猴子数目参数，最终对这些线程进行统一的管理，如截图。对线程进行管理的集合结构是线程安全的。

```
private static List<Thread> monkeyThreads =
    Collections.synchronizedList(new ArrayList<>());

Timer timer = new Timer();
TimerTask task = new TimerTask() {

  @Override public void run() {
    synchronized (createMonkeyNum) {
      for (int i = 0; i < num; i++) {
        monkeyThreads.get(createMonkeyNum).start();
        createMonkeyNum++;
        if (createMonkeyNum == monkeyNum - 1) {
          timer.cancel();
        }
      }
    }
  }
};
```

3.11.2 测试第一个文件（吞吐率极限为 3）（MyStrategy 策略）

平均吞吐率：2.308405
平均公平性：0.340053

十次测试结果如下

吞吐率：2.380952380952381 公平性：0.5255295429208473

吞吐率：2.272727272727273 公平性：0.33199554069119286

吞吐率：2.2900763358778624 公平性：0.29890746934225193

吞吐率：2.3255813953488373 公平性：0.305685618729097

吞吐率：2.2900763358778624 公平性：0.3232998885172798

吞吐率：2.3076923076923075 公平性：0.2708138238573021

吞吐率：2.3076923076923075 公平性：0.3454180602006689

吞吐率：2.3255813953488373 公平性：0.33114827201783725

吞吐率：2.272727272727273 公平性：0.3415830546265329

吞吐率：2.2900763358778624 公平性：0.3161204013377926

3.11.3 测试第二个文件（吞吐率极限为 10）（MyStrategy 策略）

平均吞吐率：4.897559
平均公平性：0.639368
十次测试结果如下

吞吐率：4.9504950495049505 公平性：0.5990220440881764

吞吐率：4.9504950495049505 公平性：0.6025971943887776

吞吐率：4.854368932038835 公平性：0.6927454909819639

吞吐率：4.854368932038835 公平性：0.6603446893787576

吞吐率：4.854368932038835 公平性：0.6715671342685371

吞吐率：5.0 公平性：0.6492825651302605

吞吐率：4.901960784313726 公平性：0.6256032064128256

吞吐率：4.854368932038835 公平性：0.6391983967935871

吞吐率：4.854368932038835 公平性：0.620376753507014

吞吐率：4.901960784313726 公平性：0.6053226452905811

3.11.4 测试第三个文件（吞吐率极限为 3）（MyStrategy 策略）

平均吞吐率：1.124451
平均公平性：0.4146
十次测试结果如下

吞吐率：1.1111111111111112 公平性：0.34626262626262627

吞吐率：1.1494252873563218 公平性：0.6048484848484849

吞吐率：1.098901098901099 公平性：0.2771717171717172

吞吐率：1.1494252873563218 公平性：0.5781818181818181

吞吐率：1.1111111111111112 公平性：0.3595959595959596

吞吐率：1.1904761904761905 公平性：0.6290909090909091

吞吐率：1.0869565217391304 公平性：0.3022222222222222

吞吐率：1.1111111111111112 公平性：0.6513131313131313

吞吐率：1.1111111111111112 公平性：0.4

吞吐率：1.1111111111111112 公平性：0.38626262626262625

4 实验进度记录

日期	时间段	计划任务	实际完成情况
06-04	晚上	读懂题意	部分完成
06-06	下午实验课	彻底读懂题意	绝大部分完成
06-10	下午+晚上	adt 设计完毕	ok
06-13	下午+晚上	策略完成	完成两个
06-14	上午+晚上	完成第三个策略	ok+完成部分 gui
06-15	全天	完成 gui	完成 gui+v2
06-16	全天	完成 v3+实验报告	完成

5 实验过程中遇到的困难与解决途径

- 多线程编程基础差，运用能力差：不断尝试+网上搜索问题的产生与解决
- 多线程 debug 极其困难：对自己的代码整体进行细致线程安全检查
- adt 设计的时候比较犹豫：纠结于如何进行更好的数据结构管理，最终设计 factory 类进行统一的管理

6 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？

(2) 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？

(3) 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？

(4) 关于本实验的工作量、难度、deadline。

(5) 到此为止你对《软件构造》课程的意见和建议。

- debug 困难，不知道何处出现了 bug；保证线程安全也很困难。
- 多线程编程性能上确实有了较大的提高
- 遇到过线程不安全的情况。通过使用线程安全的集合类以及加以线程锁保证线程安全。
- 本实验工作量适度，但是赶上了考试周，就有点顾此失彼，个人需要好好的平衡时间。