哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

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	范天祥
学号	1170300815
班号	1703008
电子邮件	2698791816@qq.com
手机号码	18846818843

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

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。
环境配置同实验五
在这里给出你的 GitHub Lab6 仓库的 URL 地址（Lab6-学号）。
https://github.com/ComputerScienceHIT/Lab6-1170300815

3 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！）。

3.1 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. **首先是 Ladder 类**
   作用：猴子过桥的组件

属性：梯子的编号 title，每个梯子上的踏板数量 n

方法：得到梯子的属性。

```
public Ladder(int n, int title) {
  // this.n = n;
  this.title = title;
  pedal = new Monkey[n];
  for (int i = 0; i < n; i++) {
    pedal[i] = no;
  }
}

public int getTitle() {
  return this.title;
}
```

Specification：

```
// Abstraction function:
//将n映射为梯子上踏板的数量
//将title映射为梯子的编号
//pedal[]数组映射为梯子上每个踏板上的猴子的集合
//静态变量no映射为一个空猴子

// Representation invariant（表示 不变性）:
//梯子的数量n要求大于0且为整数
//梯子编号的取值范围是[0,n)

// Safety from rep exposure（表示暴露的安全性）:
// 由于梯子上的踏板状态是可以改变的，也是猴子可以观察的，所以将其设置成了public
// 所有的区域块都是私有的。
```

2. 其次是 Monkey 类

作用：过桥的对象

属性：猴子编号、猴子方向、猴子速度、猴子出生时间等。

```
private final int monkeyID;
private final String direction;
private final int speed;
protected final int bornTime;
protected int reachTime;
protected String State = "UnReach";
protected int publicTime = ReadAndBuild1.time;
```

方法：　锁住猴子的类，让同一时间只能有一只猴子选择策略；让猴子跑起来。

```
if (index == 0) {
  synchronized (Monkey.class) {
    l = new ContextLadderStrategy().ChoiceStrategy(this, ReadAndBuild1.graph1,
        new Strategy1());
  }
```

```
while (true) {
  for (int i = endtemp + 1; i <= endtemp + this.speed && i <= ReadAndBuild1.h - 1; i++) {
    if (l.pedal[i] == Ladder.no) {
      temp = i;
    } else {
      break;
    }
  }
  if (temp == endtemp) {
    this.publicTime++;
    ReadAndBuild1.log1.info("第" + this.publicTime + "秒：" + this.monkeyID + "正在第" + l.getTitle()
        + "架梯子的第" + temp + "个踏板上，自" + this.direction + "前进，" + "已经出生"
        + (this.publicTime - this.bornTime) + "秒");
    Thread.sleep(1000);
```

Specification：

```
// Abstraction function:
// 将monkeyID映射为猴子的编号
// 将direction映射为猴子前进的方向
// 将speed映射为猴子前进的速度
// 将bornTime映射为猴子的出生时间
// 将reachTime映射为猴子到达对岸的时间
// 将State映射为猴子的状态
// 将publicTime映射为每个猴子的公共时间

// Representation invariant（表示 不变性）:
// monkeyID大于0
// direction只能等于"L->R"或者"R->L"
// State只能等于"UnReach"或者"Reach"

// Safety from rep exposure（表示暴露的安全性）:
// 所有的区域块都是私有的。
```

3. Relation 类构建 monkey 和 ladder 的关系：

作用：搭建过桥系统

属性：猴子、桥、关系和设置系统参数

方法：读取配置文件构建过桥系统

```
public Set<Monkey> monkeys() {
  Set<Monkey> monkeys = Collections.synchronizedSet(new HashSet<>());
  monkeys.addAll(this.monkeys);
  return monkeys;
}

public Set<Ladder> ladders() {
  Set<Ladder> ladders = Collections.synchronizedSet(new HashSet<>());
  ladders.addAll(this.ladders);
  return ladders;
}
```

```
public MLGraph(String filePath) {
  File file = new File(filePath);
  try {
    Scanner sc = new Scanner(file);
    while (sc.hasNext()) {
      String s[] = sc.nextLine().split(" ");
      if (s[0].equals("ladderNum")) {
        this.ladderNum = Integer.valueOf(s[1]);
```

4. Strategy 类

作用：提供策略

方法：具体选择方式（在此只列举一个）

```
synchronized (s) {
  for (Ladder l : s) {
    synchronized (l) {
      Monkey[] listpedal = l.pedal;
      for (int i = 0; i < listpedal.length; i++) {
        if (listpedal[i] != Ladder.no) {
          flag = 2;
          if (!listpedal[i].getDirection().equals(monkey.getDirection())) {
            flag = 0;
            break;
          }
        }
```

Specification：

```
// Abstraction function:
//静态变量no映射为一个假梯子（不存在的梯子相当于null）
```

5.factory

作用：记录操作及猴子过桥的过程

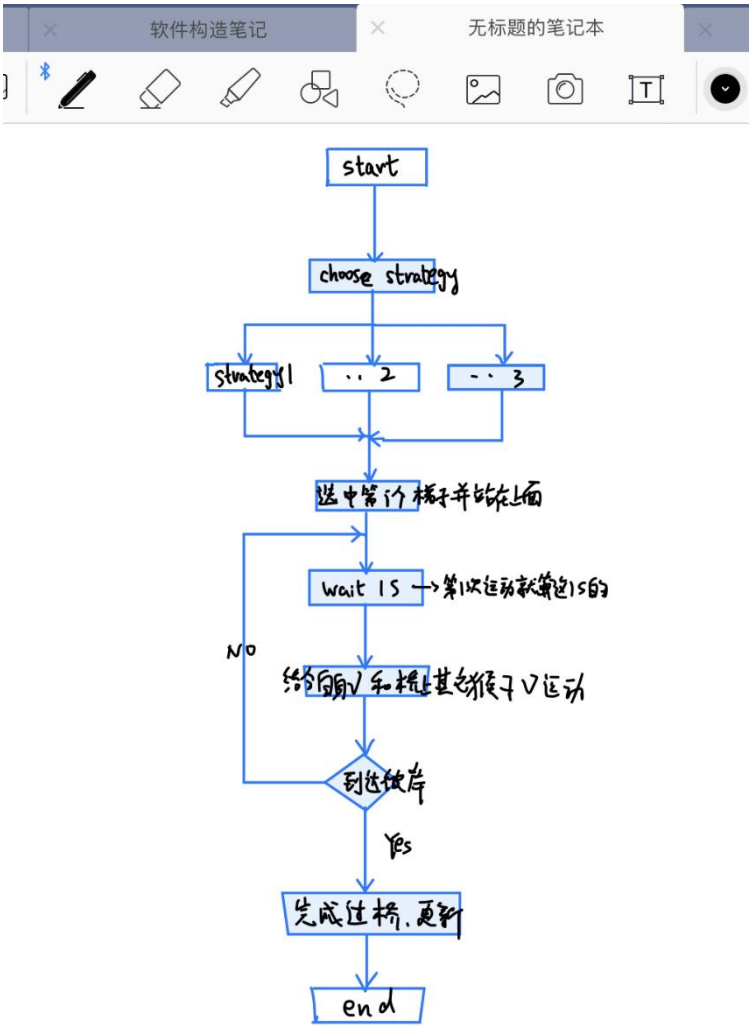
方法：就是搭建 log 的基本格式

```
try {
  fileHandler = new FileHandler(filepath, true);
  fileHandler.setLevel(level);
  if (type == 1) { //在两岸等待的猴子日志
    fileHandler.setFormatter(new Formatter() {
      @Override
      public String format(LogRecord record) {
        return record.getMessage()+"\r\n";
      }

    });
```

3.2 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。



3.3 至少两种“梯子选择”策略的设计与实现方案

3.3.1 策略 1

优先选择没有猴子的梯子，
A. 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；
B. 若满足该条件的梯子有很多，则随机选择

```
@Override
public Ladder Strategy(Monkey monkey, MLGraph graph) {
  List<Ladder> success1 = new ArrayList<>();
  List<Ladder> success2 = new ArrayList<>();
  int flag = 1;
  Set<Ladder> s = graph.ladders();
  synchronized (s) {
    for (Ladder l : s) {
      synchronized (l) {
        Monkey[] listpedal = l.pedal;
        for (int i = 0; i < listpedal.length; i++) {
          if (listpedal[i] != Ladder.no) {
            flag = 2;
            if (!listpedal[i].getDirection().equals(monkey.getDirection())) {
              flag = 0;
              break;
            }
          }
        }
```

3.3.2 策略 2

优先选择整体推进速度最快的梯子
（没有与我对向而行的猴子、其上的猴子数量最少）；

```
  synchronized (s) {
    for (Ladder l : s) {
      int num = 0;
      synchronized (l) {
        map.put(l, num);
        Monkey[] listpedal = l.pedal;
        for (int i = 0; i < listpedal.length; i++) {
          if (listpedal[i] != Ladder.no) {
            if (!listpedal[i].getDirection().equals(monkey.getDirection())) {
              map.remove(l);
              break;
            }
          }
          if (listpedal[i] != Ladder.no) {
            num++;
            map.put(l, num);
          }
        }
      }
```

3.3.3 策略 3（可选）

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；

```
synchronized (s) {
  for(Ladder l : s) {
    synchronized (l) {
      Monkey[] listpedal = l.pedal;
      for(int i=0;i<listpedal.length;i++) {
        if(listpedal[i]!=Ladder.no) {
          flag =1;
          break;
        }
      }
      if(flag ==1 ) {
        flag =0;
        continue;
      }else {
        success.add(l);
      }
    }
  }
```

3.4 “猴子生成器”MonkeyGenerator

“猴子生成器”主要是在规定的时间片段内产生设定数量的猴子。

设计方式为：按照指导书给的参数范围

初始化参数：$n = 1\sim10$，$h = 20$，$t = 1\sim5$，$N = 2\sim1000$，$k = 1\sim50$，$MV = 5\sim10$，各参数详细说明见后续描述。可由模拟器的用户在命令

1. 先写一个规范文件，然后遵守这个规范文件生成猴子，生成的猴子直接会开启线程。

2.调用配置文件的参数来初始化过桥系统，具体的生成猴子的逻辑在MonkeyGenerator()方法中。；

```
Monkey m = new Monkey((i / t) * 10 + j, map.get(index1), index2, time);
```

3.猴子生成器，猴子生成之后立即启动线程：

```
graph.addmonkey(m);
(new Thread(m)).start();
```

4.等待猴子全部过桥后，立即计算吞吐率和公平性：

```
if (ttrate != 0 && gprate != 0) {
  log1.info("吞吐率为：" + String.valueOf(ttrate));
  log1.info("公平性为：" + String.valueOf(gprate));
```

3.5 如何确保 threadsafe？

1. **将线程不安全的集合转化为线程安全的集合。**

```
public Set<Monkey> monkeys() {
  Set<Monkey> monkeys = Collections.synchronizedSet(new HashSet<>());
```

```
Set<Ladder> ladders = Collections.synchronizedSet(new HashSet<>());
```

理由如下：

A.在多线程程序中，应当**尽量使用线程安全的集合**。

B.在集合的修改和查询过程中往往涉及到很多复杂的操作。

C.比如在本实验中，**monkeys 和 ladders 是使用 set 数据结构的，**在 set 集合中，在添加或删除元素时，需要对其中的树结构进行调整，一般需要在 log(n)时间内才能完成，这样如果两个线程同时对同一个集合进行修改，就很可能造成这个集合的崩溃。可以使用读写锁来对集合的修改加以控制，但是这种控制往往是复杂的，并且低效。因此 java 提供了一些线程安全的集合类，在多线程程序中可以使用这些线程安全的集合以避免可能的不一致和崩溃现象。

D. HashSet 不同步，为了同步这些不同步的集合，使用

Set s=Collections.synchronizedSet(new HashSet());即可。

2. **在选择策略时**

锁住猴子的类，让同一时间只能有一只猴子选择策略

```
synchronized (Monkey.class) {
  l = new ContextLadderStrategy().ChoiceStrategy(this, MonkeyGenerator.graph,
      new Strategy1());
}
```

3. **在猴子移动的时候锁住梯子（l 即为梯子）**

```
synchronized (l) {
  l.pedal[temp] = this;
  l.pedal[endtemp] = Ladder.no;
  endtemp = temp;
}
```

4. **最后在生成猴子时，立即启动线程**

```
graph.addmonkey(m);
(new Thread(m)).start();
```

3.6 系统吞吐率和公平性的度量方案

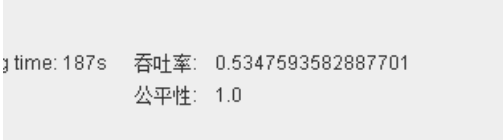
首先看指导书给的方案：

(8) 计算吞吐率和公平性：计算并输出本次仿真的吞吐率和公平性。

- “吞吐率”是指：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$X=\frac{T}{N}$秒，则吞吐率$Th=\frac{N}{T}$表征每秒钟可过河的猴子数目。
- “公平性”是指：如果 Monkey 对象 A 比 B 出生得早，那么 A 应该不晚于 B 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Y_a和Y_b，抵达对岸的时间分别为Z_a和Z_b，那么公平性$F(A,B)=\begin{cases}1, & if\ (Y_b-Y_a)*(Z_b-Z_a)\geq 0\\ -1, & otherwise\end{cases}$。对$N$只猴子两两计算其之间的公平性并综合到一起，得到本次模拟的整体公平性$F=\frac{\sum_{(A,B)\in\Theta}F(A,B)}{C_N^2}$，$\Theta=\{(A,B)|A\neq B,(B,A)\notin\Theta\}$，其取值范围为$[-1,1]$。

你的程序应追求吞吐率尽可能大。公平性并非程序追求的目标，每次模拟时只需计算出公平性的值即可（但如果你的程序能在最大化吞吐率的情况

GUI 效果：



(1) 计算吞吐率:

"吞吐率"是指:

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，

吞吐率 Th=N/T 表示每秒钟可以过河的猴子的数目。

根据线程池中的线程全部执行结束时距离第一只猴子开始过河的时间

以及猴子的总数目可以求得。

```
g.drawString("吞吐率:", 520, 50);
if (Thread.activeCount() == 3) {
  ReadAndBuild1.ttrate = (double) ReadAndBuild1.N / (double) (ReadAndBuild1.time);
  g.drawString(String.valueOf((double) ReadAndBuild1.N / (double) (ReadAndBuild1.time)), 570, 50);
}
```

(2)计算公平性:

公平性是指:

A. 如果 A 猴子比 B 猴子出生的早，但是到达河对岸的时间比 B 晚，那么对于 A 和 B 而言就不公平，相反就是公平。

B. 这只适用于不同批次产生的猴子之间的差别，比如第 1 批产生的猴子比第 3 批产生的猴子出生时间要早；但是如果两只猴子属于同一个批次，即为在相同的秒数出生，那么我们认为无论谁先到达对岸都是公平的。

```
g.drawString("公平性:", 520, 70);
double sum1 = 0;
double sum2 = (ReadAndBuild1.N * (ReadAndBuild1.N - 1)) / 2.0;
for (Monkey m1 : graph.monkeys()) {
  for (Monkey m2 : graph.monkeys()) {
    if (!m1.equals(m2)) {
      if ((m1.bornTime - m2.bornTime) * (m1.reachTime - m2.reachTime) >= 0) {
        sum1 += 1;
      } else {
        sum1 -= 1;
      }
    }
  }
}
if (Thread.activeCount() == 3) {
  ReadAndBuild1.gprate = (sum1 / sum2) / 2.0;
  g.drawString(String.valueOf((sum1 / sum2) / 2.0), 570, 70);
}
```

3.7 输出方案设计

日志：

写了一个日志文件的配置方案，内部是设置日志的等级和读写文件等基本操作，设置好时间格式，记录猴子出生、等待过河的时间、过河花费的时间和运行总时间等。

将所有关于 Monkey 对象过河的全过程均记录在 monkey.generator 的 txt 文件中。

```java
public static void addFileHandle(int type, Logger log, Level level, String filepath) {

  try {
    fileHandler = new FileHandler(filepath, true);
    fileHandler.setLevel(level);
    if (type == 1) {
      fileHandler.setFormatter(new Formatter() {
        @Override
        public String format(LogRecord record) {
          return record.getMessage() + "\r\n";
        }
```

日志效果（记录猴子出生、等待过河的时间、过河花费的时间和运行总时间等）：

MonkeyGener... Monkey.java MonkeyCrossR... LogFactory.java

```
第0秒：0正在第1架梯子的第0个踏板上，自R->L前进；已经出生0秒
第0秒：2正在右岸等待；已经出生0秒
第0秒：1正在左岸等待；已经出生0秒
第0秒：4正在第0架梯子的第0个踏板上，自L->R前进；已经出生0秒
第0秒：3正在第2架梯子的第0个踏板上，自R->L前进；已经出生0秒
第1秒：11正在右岸等待；已经出生0秒
第1秒：12正在左岸等待；已经出生0秒
第1秒：13正在左岸等待；已经出生0秒
第1秒：14正在左岸等待；已经出生0秒
第1秒：0正在第1架梯子的第2个踏板上，自R->L前进；已经出生1秒
第1秒：3正在第2架梯子的第2个踏板上，自R->L前进；已经出生1秒
第1秒：2正在第2架梯子的第0个踏板上，自R->L前进；已经出生1秒
第1秒：1正在第0架梯子的第0个踏板上，自L->R前进；已经出生1秒
第1秒：4正在第0架梯子的第1个踏板上，自L->R前进；已经出生1秒
第1秒：10正在左岸等待；已经出生0秒
第2秒：0正在第1架梯子的第4个踏板上，自R->L前进；已经出生2秒
第2秒：11正在右岸等待；已经出生1秒
第2秒：14正在左岸等待；已经出生1秒
第2秒：13正在左岸等待；已经出生1秒
第2秒：12正在左岸等待；已经出生1秒
第2秒：4正在第0架梯子的第2个踏板上，自L->R前进；已经出生2秒
第2秒：3正在第2架梯子的第4个踏板上，自R->L前进；已经出生2秒
第2秒：1正在第0架梯子的第0个踏板上，自L->R前进；已经出生2秒
第2秒：2正在第2架梯子的第1个踏板上，自R->L前进；已经出生2秒
第2秒：10正在左岸等待；已经出生1秒
第3秒：0正在第1架梯子的第6个踏板上，自R->L前进；已经出生3秒
第3秒：11正在右岸等待；已经出生2秒
第3秒：12正在左岸等待；已经出生2秒
第3秒：13正在左岸等待；已经出生2秒
第3秒：14正在左岸等待；已经出生2秒
```

GUI：

展示参数：

```java
g.drawString("参数设置为：", 10, 50);
g.drawString("n: " + String.valueOf(ReadAndBuild1.n), 100, 50);
g.drawString("h: " + String.valueOf(ReadAndBuild1.h), 150, 50);
g.drawString("t: " + String.valueOf(ReadAndBuild1.t), 200, 50);
g.drawString("N: " + String.valueOf(ReadAndBuild1.N), 250, 50);
g.drawString("k: " + String.valueOf(ReadAndBuild1.k), 300, 50);
g.drawString("MV: " + String.valueOf(ReadAndBuild1.MV), 350, 50);
g.drawString("running time: " + String.valueOf(ReadAndBuild1.time) + "s", 400, 50);
g.drawString("吞吐率:", 520, 50);
```

效果：

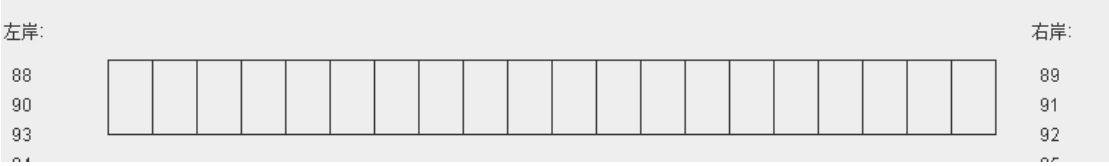
参数设置为：　n: 3　h: 20　t: 3　N: 100　k: 10　MV: 4　running time: 66s　吞吐率:
公平性:

画梯子：

```
int yPerLadder = 150;
int xPerPedal = 30;
for (int i = 1; i <= ReadAndBuild1.n; i++) {
  g.drawLine(80, (yPerLadder * i - 25), 80 + 30 * ReadAndBuild1.h, (yPerLadder * i - 25));
  g.drawLine(80, (yPerLadder * i + 25), 80 + 30 * ReadAndBuild1.h, (yPerLadder * i + 25));

  for (int k = 0; k <= ReadAndBuild1.h; k++) {
    g.drawLine(80 + xPerPedal * k, (yPerLadder * i - 25), 80 + xPerPedal * k,
        (yPerLadder * i + 25));
  }
```

效果：



画猴子：

我这里是用 BufferedImage 和 Graphics2D，实现把字符串转化为图片代替猴子，并添加到梯子上：

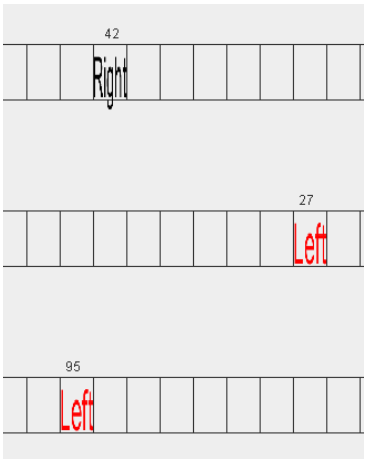
```
String text = "Left";

BufferedImage img1 = new BufferedImage(1, 1, BufferedImage.TYPE_INT_ARGB);
Graphics2D g2d = img1.createGraphics();
Font font = new Font("Arial", Font.PLAIN, 48);
g2d.setFont(font);
FontMetrics fm = g2d.getFontMetrics();
int width = fm.stringWidth(text);
int height = fm.getHeight();
g2d.dispose();

img1 = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
g2d = img1.createGraphics();
g2d.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
    RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);
g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
g2d.setRenderingHint(RenderingHints.KEY_COLOR_RENDERING,
    RenderingHints.VALUE_COLOR_RENDER_QUALITY);
g2d.setRenderingHint(RenderingHints.KEY_DITHERING, RenderingHints.VALUE_DITHER_ENABLE);
```

```
g.drawImage(img1, 80 + i * xPerPedal, yPerLadder * (l.getTitle() + 1) - 25, 30, 60,
    this);
```

效果：（Right 为右边产生的猴子，Left 为左边产生的猴子）

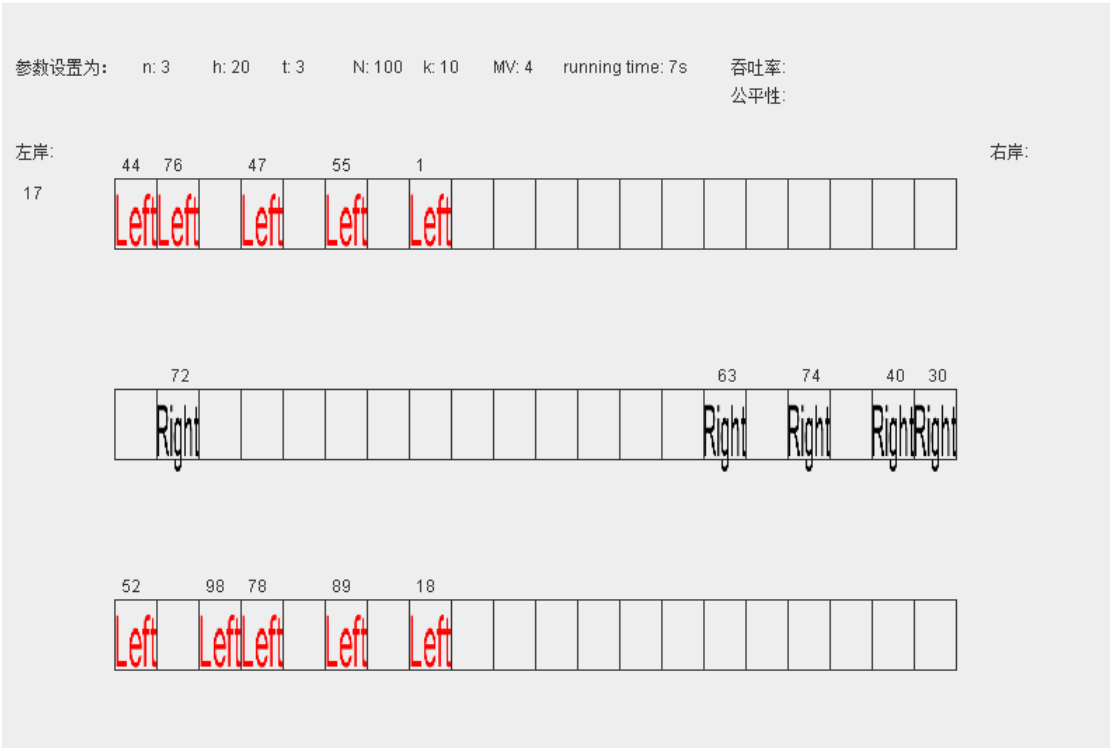


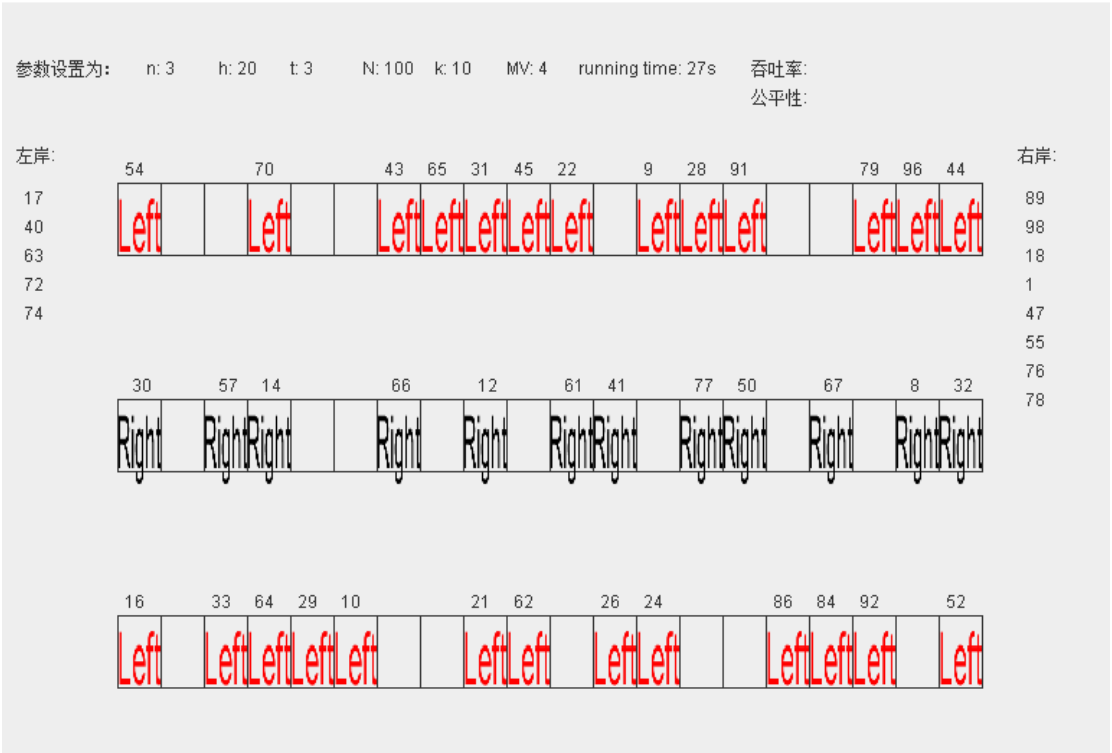
可视化（可选）：

模拟猴子运动:

```
// 模拟猴子的移动
for (Ladder l : graph.ladders()) {
  for (int i = 0; i < l.pedal.length; i++) {
    if (l.pedal[i].getID() != Integer.MAX_VALUE) {
      if ((l.pedal[i].getDirection().equals("L->R"))) {
```

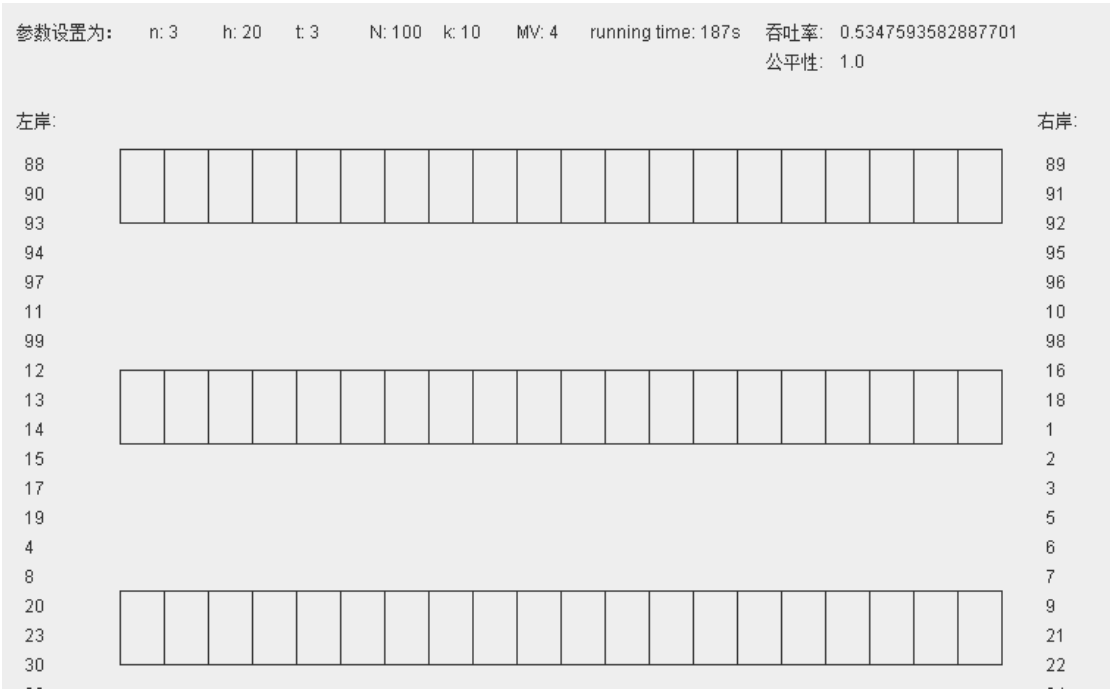
效果展示：






参数设置为： n: 3 h: 20 t: 3 N: 100 k: 10 MV: 4 running time: 47s 吞吐率:
公平性:
左岸:
右岸:
7 48 39
32 53 99 87 94 37 23 20 88 68 83
69 25 16

过河完毕：


参数设置为： n: 3 h: 20 t: 3 N: 100 k: 10 MV: 4 running time: 187s 吞吐率: 0.5347593582887701
公平性: 1.0
左岸:
右岸:

3.8 猴子过河模拟器 v1

3.8.1 参数如何初始化

写一个参数文件:

```
ladderNum 3
pedalNum 20
timeSpan 3
monkeyNum 100
monkeySpan 10
mostV 4
```

读文件的方式获取参数;

```
static MLGraph graph = new MLGraph("src/settings/data.txt");
static int time = 0;

public static int n = graph.getLadderNum();
public static int h = graph.getpedalNum();
public static int t = graph.gettimeSpan();
public static int N = graph.getmonkeyNum();
public static int k = graph.getmonkeySpan();
public static int MV = graph.getmostV();
public static double ttrate = 0;
public static double gprate = 0;
```

然后在指定的参数范围内随机产生猴子

```
Map<Integer, String> map = new HashMap<>();
map.put(0, "L->R");
map.put(1, "R->L");
int remainder = N % k;
for (int i = 0; i <= ((N / k) - 1) * t; i++) {
  if (i % t != 0 && i != 0) {
    Thread.sleep(1000);
    time++;
  } else {
    for (int j = 0; j < k; j++) {
      Random rd1 = new Random();
      int index1 = rd1.nextInt(2);
      int index2 = rd1.nextInt(MV - 1) + 1;

      Monkey m = new Monkey((i / t) * 10 + j, map.get(index1), index2, time)
      graph.addmonkey(m);
      (new Thread(m)).start();
```

3.8.2 使用 Strategy 模式为每只猴子选择决策策略

先随机生成数字作为选择的策略序号:

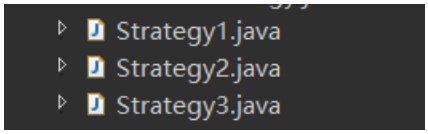
```
Random rd1 = new Random();
int index = rd1.nextInt(3);
Ladder l = null;
try {
  if (index == 0) {
    synchronized (Monkey.class) {
      l = new ContextLadderStrate
          new Strategy1());
    }
    while (l == Strategy1.no) {
      ReadAndBuild1.log1
```

再用 strategy 设计模式返回选择的方案：

```
public class ContextLadderStrategy {

  public Ladder ChoiceStrategy(Monkey monkey, MLGraph graph, LadderStrategy choice) {
    return choice.Strategy(monkey, graph);
  }
}
```

所有的三个策略类，打包在一个 strategy 的包内：

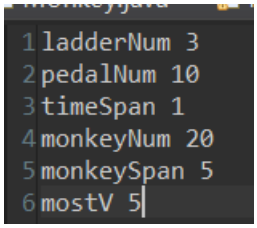


3.9 猴子过河模拟器 v2

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。

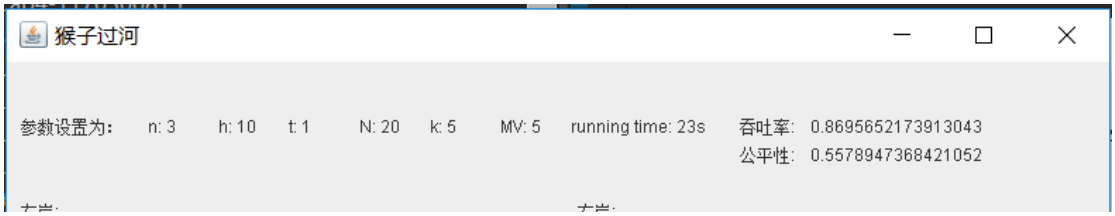
3.9.1 对比分析：固定其他参数，选择不同的决策策略

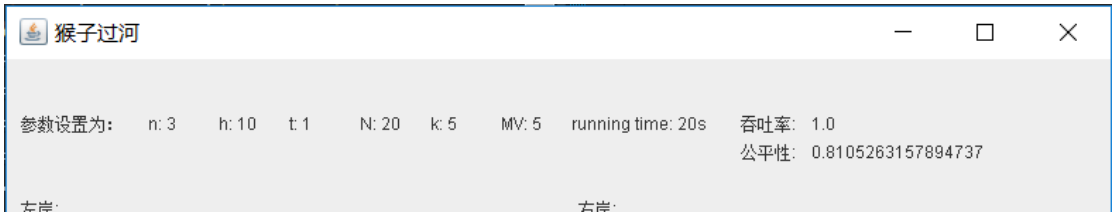
固定参数如下：



所有的猴子选择策略一：

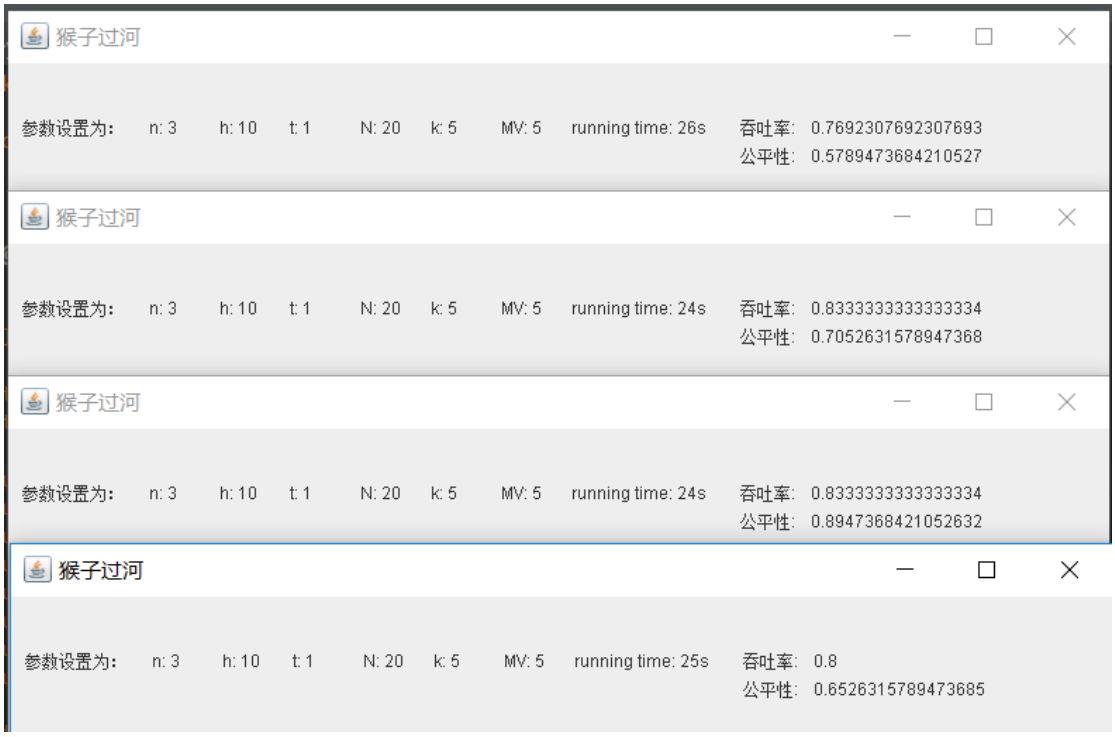
实验结果：（测量 4 次）





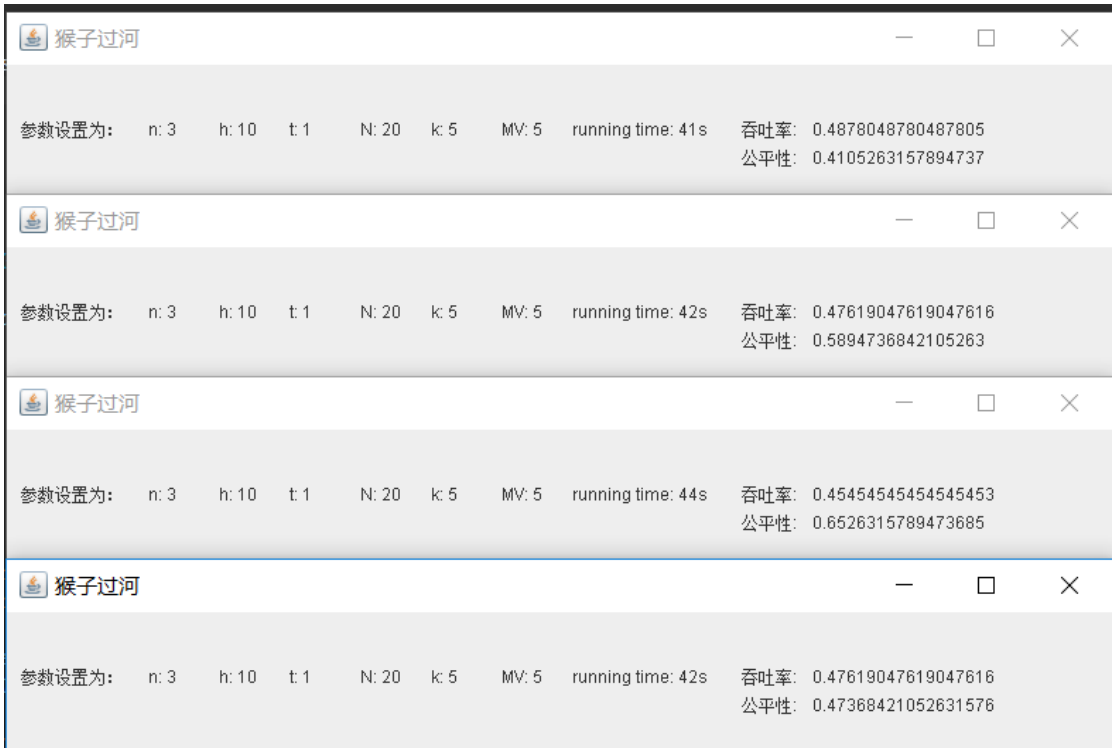
所有的猴子选择策略二：

实验结果：



所有的猴子选择策略三：

实验结果：



具体数据在 log 文件里面有详细的记录:

```
658 第37秒：23正在第1架梯子的第2个踏板上，自R->L前进；已经出生35秒
659 第38秒：23正在第1架梯子的第4个踏板上，自R->L前进；已经出生36秒
660 第39秒：23正在第1架梯子的第6个踏板上，自R->L前进；已经出生37秒
661 第40秒：23正在第1架梯子的第8个踏板上，自R->L前进；已经出生38秒
662 第41秒：23正在第1架梯子的第9个踏板上，自R->L前进；已经出生39秒
663 第42秒：23以从右岸抵达左岸，共耗时40秒
664 吞吐率为：0.47619047619047616
665 公平性为：0.47368421052631576
666
```

对比分析：

每次测试中每只猴子采用的策略相同。对比，很直观地可以看见，
策略一和策略二的吞吐率差不多，但是都比策略三强很多；
策略二公平性最高，策略一和策略三差不多；
最后是策略二最优，吞吐率和公平性都是最高的，策略三表现最差。

事实证明：

- **3.3.2 策略 2**

优先选择整体推进速度最快的梯子
（没有与我对向而行的猴子、其上的猴子数量最少）；

```
synchronized (s) {
```

这个策略最好。

3.9.2 对比分析：变化某个参数，固定其他参数

所有的测试都是基于策略二这个最优策略下进行测试的：

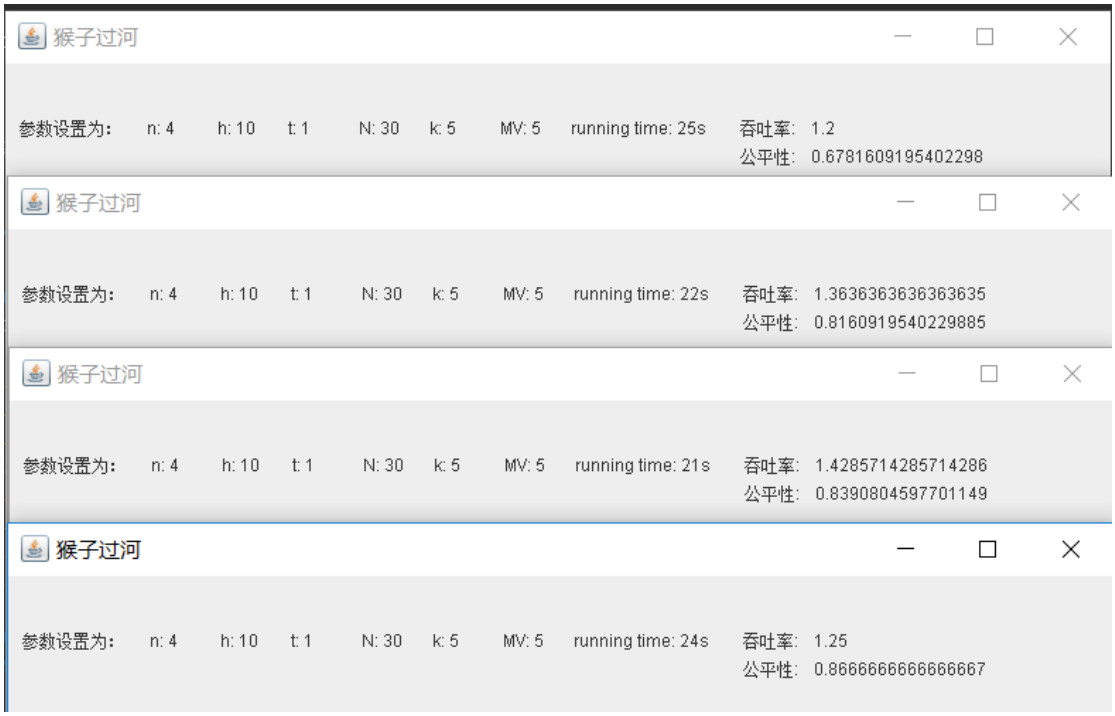
1. 参数如下：（变化梯子数量：LadderNum）

```
ladderNum 3
pedalNum 10
timeSpan 1
monkeyNum 30
monkeySpan 5
mostV 5
```

LadderNum = 2

LadderNum = 3

LadderNum = 4



LadderNum = 5

对比分析：

梯子数量 LadderNum 增加，显著增加的是吞吐率，当然公平性也稍微增加。

2. 参数如下：（变化每秒产生的猴子数量：monkeySpan）

```
1 ladderNum 3
2 pedalNum 10
3 timeSpan 1
4 monkeyNum 80
5 monkeySpan 5
6 mostV 5
```

monkeySpan = 3

monkeySpan = 5

monkeySpan = 10

对比分析：

可以看到随着 monkeySpan 的增大，吞吐率逐渐提升，公平性逐渐下降。

3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？

由实验结果可以得到：

吞吐率确实与一些参数和策略是有关的。

由 3.9.1 内容可以知道：**策略二最优**，吞吐率和公平性都是最高的，策略三表现最差。

由 3.9.2 内容可以知道：**梯子的数目 LadderNum 增加**，那么吞吐率也在上升，很明显，因为单位时间可以通过的猴子更多了。

单次产生的猴子数目 monkeySpan：可以看到随着 monkeySpan 的增大，吞吐率逐渐提升，公平性逐渐下降。

猴子的总数目 N，可以看到随着 N 的增大，吞吐率逐渐提升。

产生猴子的时间间隔 t，可以看到随着 t 的增大，吞吐率逐渐下降。

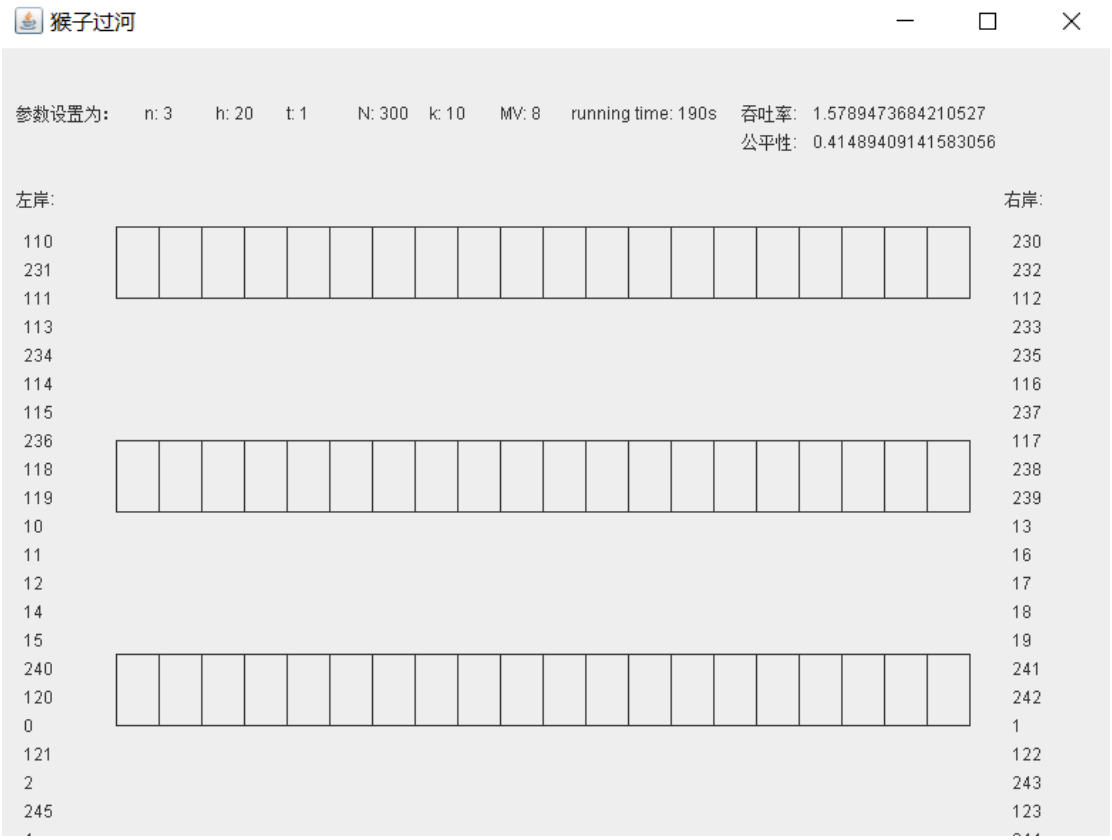
3.9.4 压力测试结果与分析

压力测试 1：

全部为策略一

参数配置：生成 300 个猴子

```
ladderNum 3
pedalNum 20
timeSpan 1
monkeyNum 300
monkeySpan 10
mostV 8
```



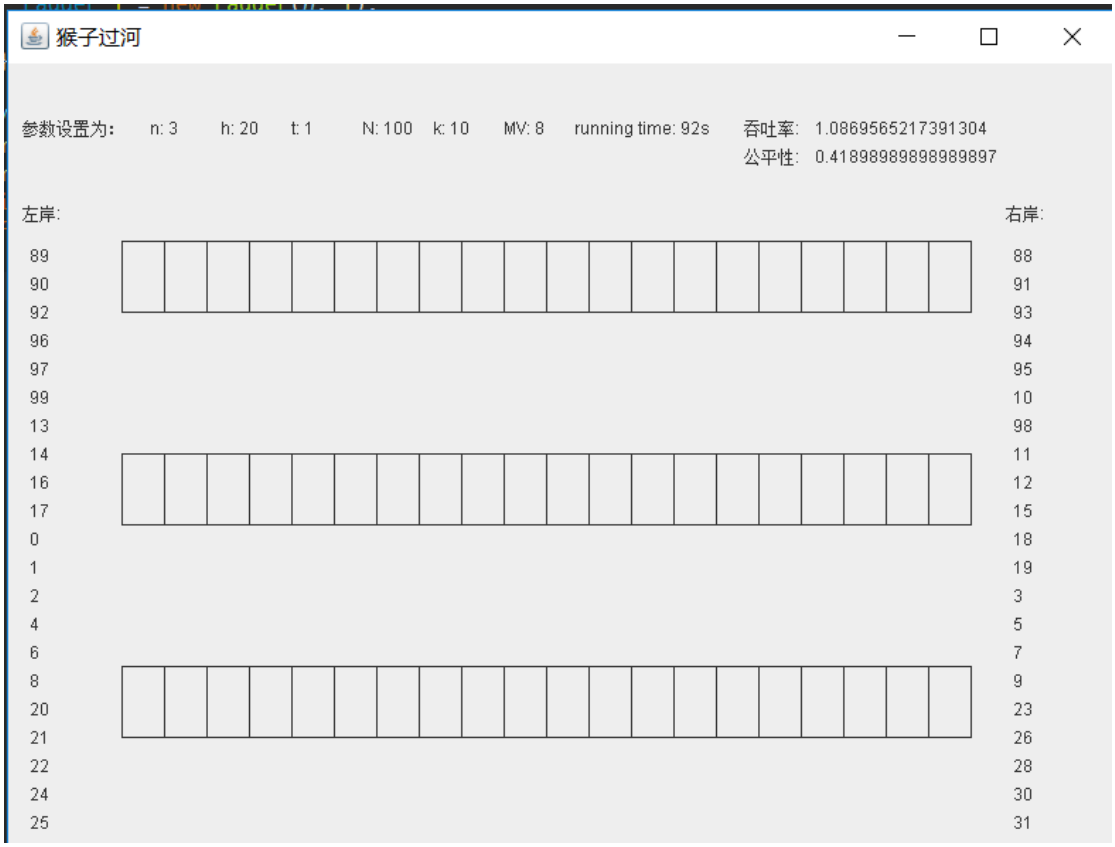
吞吐率为：1.579
公平性为：0.415
表现还行。

压力测试 2：修改源码，只生成速度为 1 和 5 的猴子

```
Random rd1 = new Random();
int index1 = rd1.nextInt(2);
int index3 = rd1.nextInt(2);
// int index2 = rd1.nextInt(MV - 1) + 1;
int index2 = 0;
if (index3 == 0) {
  index2 = 1;
} else {
  index2 = 5;
}
Monkey m = new Monkey((i / t) * 10 + j, map.get(index1), i
graph.addmonkey(m);
```

全部为策略一
参数配置：生成 100 个猴子

```
ladderNum 3
pedalNum 20
timeSpan 1
monkeyNum 100
monkeySpan 10
mostV 8
```



吞吐率为：1.087

公平性为：0.419

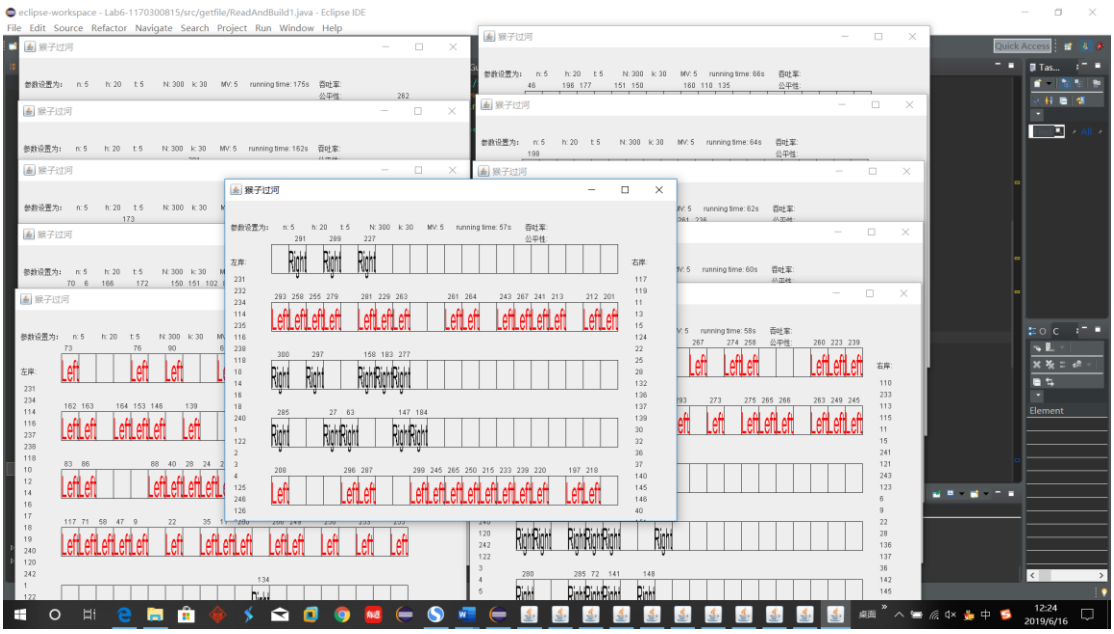
表现不如随机产生速度的方案，很多速度快的猴子被速度慢的猴子拦截了下来，由于速度差异非常大，导致速度快的猴子严重停滞，最终表现为吞吐率显著下降。

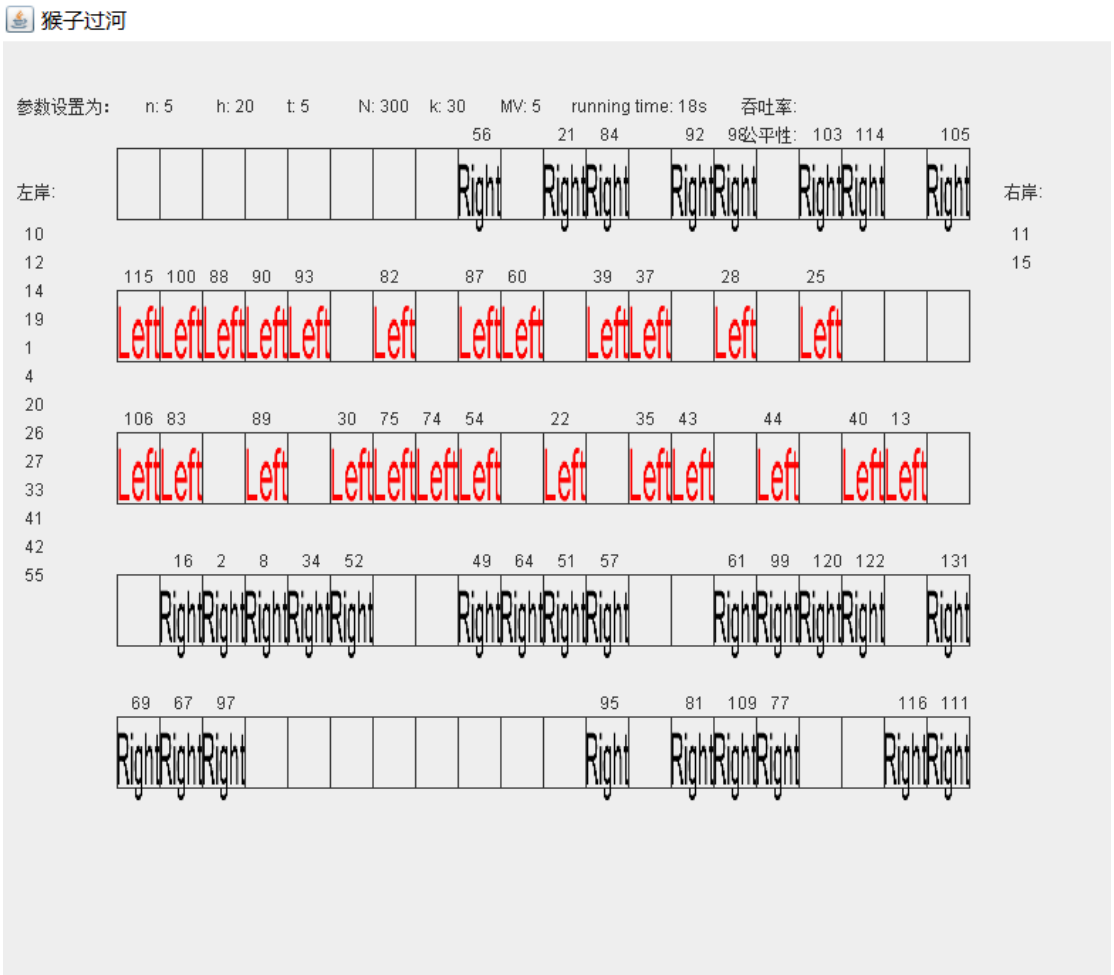
3.10 猴子过河模拟器 v3

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。

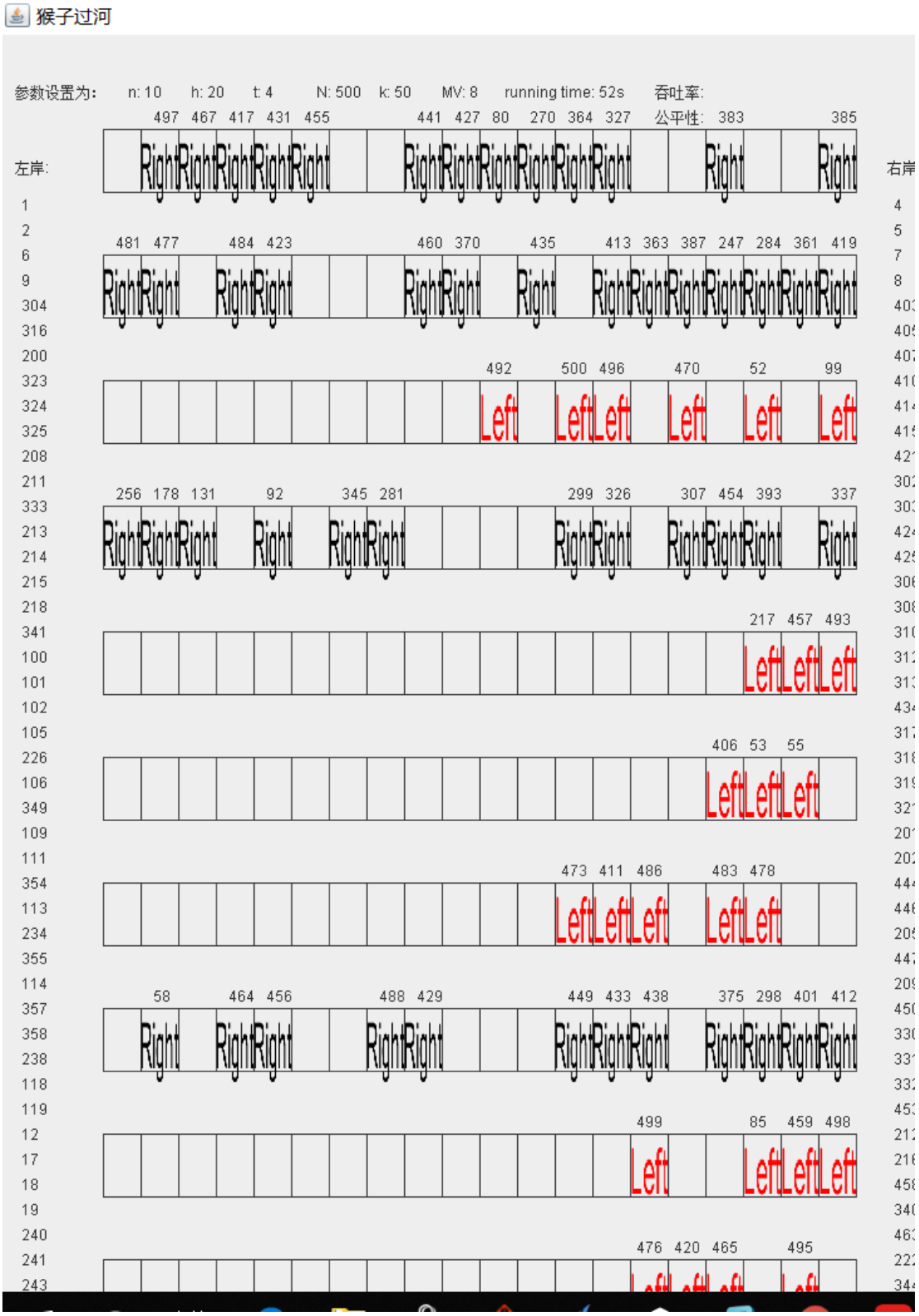
Competiton_1.txt

疯狂测试：

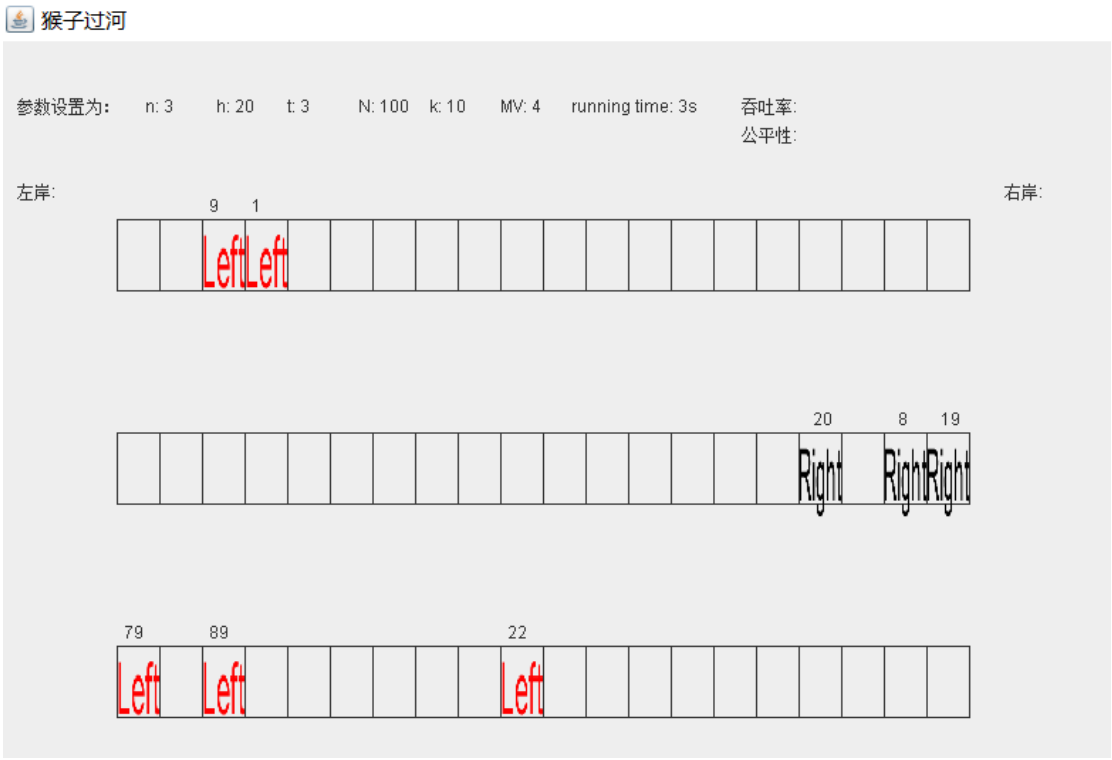




Competiton_2.txt



Competiton_3.txt



	吞吐率	公平性
Competiton_1.txt		
第 1 次模拟	0.92879	0.7863
第 2 次模拟	0.95238	0.6952
第 3 次模拟	1.16732	0.7051
第 4 次模拟	1.05634	0.7249
第 5 次模拟	1.09890	0.6745
第 6 次模拟	1.10294	0.6739
第 7 次模拟	1.04530	0.7148
第 8 次模拟	1.09890	0.6892
第 9 次模拟	1.07527	0.7371
第 10 次模拟	0.95541	0.7192
平均值	1.04816	0.7120
Competiton_2.txt		
第 1 次模拟	2.45098	0.4125
第 2 次模拟	2.41546	0.3956
第 3 次模拟	2.55102	0.4072
第 4 次模拟	2.34742	0.3568
第 5 次模拟	2.63158	0.3729
第 6 次模拟	2.56410	0.4237

第 7 次模拟	2.38095	0.3951
第 8 次模拟	2.39234	0.4215
第 9 次模拟	2.65957	0.4038
第 10 次模拟	2.55102	0.3967
平均值	2.49444	0.3986
Competiton_3.txt		
第 1 次模拟	0.45045	0.4596
第 2 次模拟	0.51020	0.5621
第 3 次模拟	0.49261	0.5347
第 4 次模拟	0.51813	0.5482
第 5 次模拟	0.51020	0.5173
第 6 次模拟	0.49261	0.4981
第 7 次模拟	0.50761	0.5217
第 8 次模拟	0.50251	0.5009
第 9 次模拟	0.50252	0.4625
第 10 次模拟	0.48309	0.5014
平均值	0.49699	0.5107

4 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
6.3	15:30-17:30	了解实验要求	完成
6.7	All Day	写 V1	未完成
6.8	All Day	写 V1	未完成
6.9	All Day	写 V1	完成
6.14	All Day	写 V2	完成
6.15	All Day	写 V3，报告	完成
6.16	10:00-13:00	撰写实验报告	完成

5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
想把字符串转化为图片	百度
线程的安全性	看 PPT 和查看 CSDN 等平台的博客
Log 的配置	百度

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训

单线程程序的调试是容易出现复现错误的，而对于多线程来说，很多错误不会复现。甚至在其中增加一条涉及 IO 操作的语句就能够彻底改变执行的顺序。

多线程性能方面的改善主要还是来自于并行，这种性能的提升主要的来自于多个任务并行执行时的优势。

6.2 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？

1. 多线程程序需要比单线程程序多考虑竞争条件的发生，需要保证数据线程安全。
2. 多线程程序在性能方面要比单线程好，并行比串行效率更高。

(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？

1.A.使用线程安全的数据结构和 synchronized 加锁来实现。

B.优先保证线程安全，然后再考虑性能问题。

2.尽量避免了不必要的同步，使同步块在保证安全的底线下尽可能小。

(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？

A. 在多个猴子同时访问 ladder 的时候，要求每节梯子同一时间只能有一只猴子；

B. 刚开始一只猴子上了梯子之后还没来得及设置状态，就是占用的信息没有及时的写回，就因为线程调度被挂起；

C. 下一只猴子开始运行，同样的上了同一架梯子，导致冲突。

D. 考虑原因是在数据结构选择上使用了线程安全的，但是没有考虑多个原子操作的组合情况。

(4) 关于本实验的工作量、难度、deadline。

工作量不算很大，难度适中，deadline 还行。

(5) 到此为止你对《软件构造》课程的意见和建议。

课程设置很有挑战性，内容丰富，实验量不小，但是经历了这一学期的锻炼，我的编程能力有了不小的提升。

我从对 java 一无所知到能够用 java 解决一些基本的问题，初步了解并应用设计模式，学习了面向对象的一些基本概念和软件构造过程的各个部分，收获非常大。

最后感谢徐老师和助教们一学期来的工作和努力！