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	江建东
学号	1163450201
班号	1737102
电子邮件	1820985520@qq.com
手机号码	18846196989

目录

1 实验目标概述 1
2 实验环境配置 1
3 实验过程 1
3.1 ADT 设计方案 1
3.2 Monkey 线程的 run()的执行流程图 6
3.3 至少两种“梯子选择”策略的设计与实现方案 6
3.3.1 策略 1 6
3.3.2 策略 2 9
3.3.3 策略 3 10
3.3.4 策略 4 10
3.3.5 策略 5 11
3.3.6 策略 6 13
3.3.7 策略 7 13
3.4 “猴子生成器”MonkeyGenerator 14
3.5 如何确保 threadsafe？ 15
3.6 系统吞吐率和公平性的度量方案 15
3.7 输出方案设计 16
3.8 猴子过河模拟器 v1 21
3.8.1 参数如何初始化 21
3.8.2 使用 Strategy 模式为每只猴子选择决策策略 22
3.9 猴子过河模拟器 v2 22
3.9.1 对比分析：固定其他参数，选择不同的决策策略 23
3.9.2 对比分析：变化某个参数，固定其他参数 24
3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？ 28
3.9.4 压力测试结果与分析 28
3.10 猴子过河模拟器 v3 30
4 实验进度记录 31
5 实验过程中遇到的困难与解决途径 31
6 实验过程中收获的经验、教训、感想 32

6.1 实验过程中收获的经验和教训……32
6.2 针对以下方面的感受……33

1 实验目标概述

本次实验训练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。训练如何在 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

点击手册链接生成 lab6 实验仓库，拷贝地址，在本地的 git bash 中将远程仓库 clone 到本地

仓库地址：https://github.com/ComputerScienceHIT/Lab6-1163450201

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

Monkey：

猴子 ADT，用来表示一只过河的猴子，继承 Thread 类，是一个独立的线程。猴子具有以下属性

id （int）：表示猴子自己的 ID，ID 唯一
direction（String）：表示猴子的方向，方向只能为 “R→L”或”L→R”
speed（int）：猴子的速度，速度大于等于 1
ladders（Ladders）：猴子所要爬的一组梯子，用来传递给决策器决策

selector（Select）：决策策略，每个猴子有不同的决策策略

logger（Logger）：猴子的日志

starttime（long）：猴子出生时间戳

countdownLatch（CountDownLatch）：用于记录线程结束通知主线程

猴子具有以下方法：

@Override run()：

线程所要执行的方法，每隔一秒调用决策决定行动，如果到岸则记录死亡时间戳结束运行

public Logger getLogger()：

获得日志对象

public int getID()

获得猴子 ID

public String getDirection()

获得猴子方向

public int getSpeed()

获得猴子速度

public int getTime()

获得猴子当前已经存活的时间，单位为秒

public boolean equals()

比较相等

public int hashCode()

散列

Rung：

踏板 ADT，表示梯子上的踏板

属性：

position：踏板在梯子中的坐标，从左向右从 0 开始

方法：

public int getPosition()：

获得位置

Ladder：

梯子 ADT，表示过河用的梯子，梯子上有踏板，梯子也可以承载多个猴子

属性：

rungs（List<Rungs>）：一组踏板

monkeys（Set<Monkey>）：梯子当前的猴子集合

mstate（Map<Monkey, Rung>）：一组猴子和踏板的映射关系，记录猴子所在的踏板

rstate（Map<Rung, Monkey>）：一组踏板和猴子的映射关系，记录踏板对应的猴子

index（int）：梯子在梯子组中的编号

方法：

public int move(Monkey m, int step)：

将猴子按照移动方向移动指定距离，返回移动后所处踏板位置

public synchronized int occupy(Monkey m, String direction)

判断当前猴子的指定方向上是否存在猴子，如果存在返回与这个猴子之间的距离，这个距离包括那个猴子本身

public synchronized boolean contain(Monkey m)

查找猴子是否在梯子上

public synchronized boolean contain(int position)

查找某个位置是否有猴子

public synchronized boolean isEmpty()

返回梯子是否为空

public synchronized Monkey leftmost()

返回梯子最左边的猴子

public synchronized Monkey rightmost()

返回梯子最右边的猴子

public int indexOf(Monkey m)

返回猴子的位置

public Monkey get(int i)

返回某位置的猴子

public synchronized void add(Monkey m)

将猴子加到踏板起始处

public int size()

返回梯子的长度

public int sizeOfMonkeys()

返回当前梯子上猴子数量

public String toString()

Ladders：

一组梯子 ADT，表示过河用的所有梯子的集合

属性：

ladders（List<Ladder>）：一组梯子

方法：

public Ladder get(int i)

返回指定位置的梯子对象

public int size()

返回梯子数量

public int length()

返回梯子长度

public int indexOf(Ladder l)

返回指定梯子位置，从上到下从 0 开始

public Iterator<Ladder> iterator()

迭代器

Selector:

决策接口，需要实现 select 方法

方法：

boolean select(Monkey monkey, Ladders ladders)

对 monkey 执行决策策略，返回是否到达对岸状态，等待或前进返回 true，到达返回 false

每个 ADT 的 specification；

Monkey：

Monkey 是一个继承自 Thread 的类型，这是一个 immutable 的类型。表示一只想要过河的猴子，它可能从左岸到右岸也可能从右岸到左岸，取决于传入的方向。Monkey 具有 ID，速度，方向，行动策略属性，这些在建立对象时决定。

Rung：

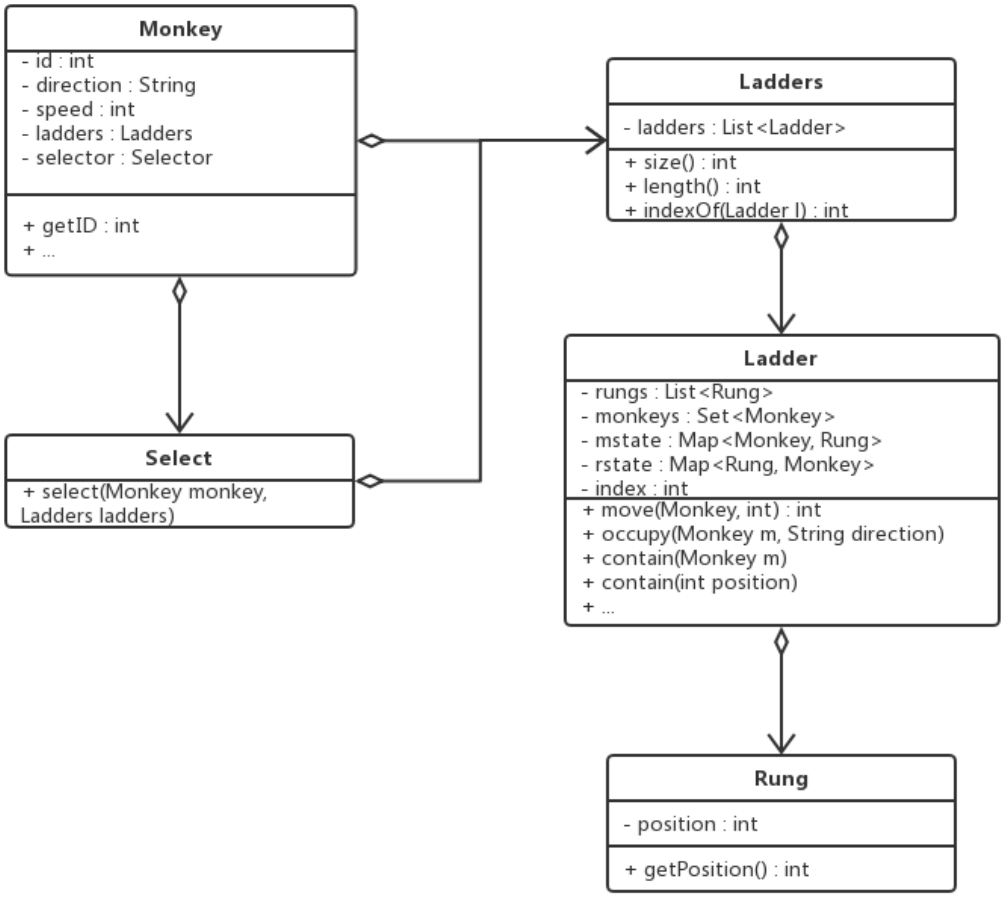
梯子上的一个横挡

Ladder：

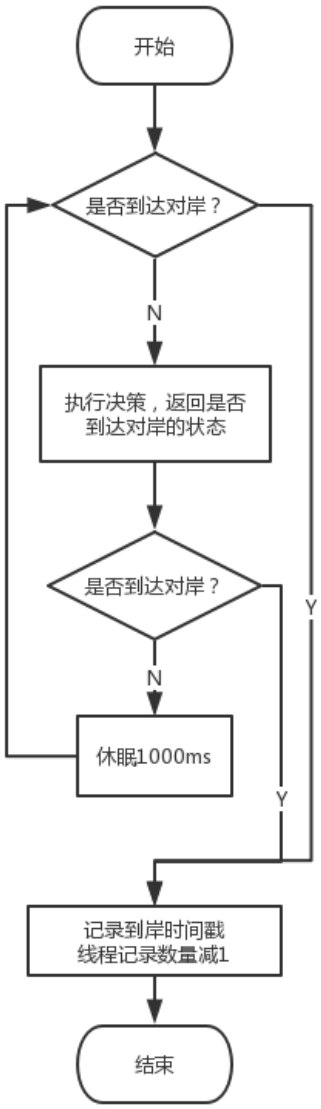
表示猴子过河使用的梯子，mutable 类型，梯子上含有至少一个踏板，每个踏板最多站一只猴子。该类型的每个方法的操作是原子操作，意味着每个方法是线程安全的，但是方法与方法之间的操作是非原子的。梯子维护猴子在梯子上的分布状态，提供方法用于对猴子的操作。

Ladders：

表示横跨河上的一组梯子，mutable 类型，梯子总数和长度不变，梯子的状态会发生变化，可以将 Ladders 对象视为 List 操作



3.2 Monkey 线程的 run()的执行流程图



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

这个策略是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

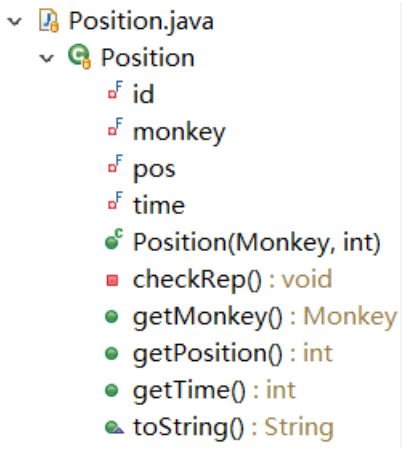
首先是不能以上帝视角观察所有猴子的行动策略，也就是不能知道这个猴子是要向左或者向右走，猴子只能通过观察变化来获取各个猴子的行动信息。

因此，在程序运行中，需要为猴子维护一组“快照”，快照保存着上一秒各个猴子自己的观察记忆。通过比较上一秒快照和当前梯子状态的变化，得知各个梯子上离上一秒自己最近的**同一只猴子**（不同猴子没有可比较性）位置变化，推测

出各个梯子真实的行进速度和各个梯子的行进方向，从而做出自己的决策。在该决策中，优先选择空梯子，再选择同向梯子。

“快照”的实现：

设计 Position 类保存一个猴子和其对应的位置



设计 Map<Ladder, Position>保存一个梯子上最左或最右的猴子的位置，在此基础上嵌套一层 Monkey 映射，保存各个猴子自身不同的观察结果。

```
Map<Monkey, Map<Ladder, Position>>
```

为了保证能够正确观察，设置两个 map，记录一秒前各个梯子最左和最右两个记录

```
Map<Monkey, Map<Ladder, Position>> preleft

Map<Monkey, Map<Ladder, Position>> preright
```

这样猴子就有了一秒钟的“记忆”

决策流程：

判断是否已经在梯子上

在梯子上：

向前移动，先查看是否有猴子挡路，挡路就移动到那猴子后面

然后判断是否到达对岸

没到：

返回 true

到了：

返回 false

不在梯子上：

判断梯子是否已经满了

没满：

没满找空梯子上去，返回 true。如果梯子都被抢了，等待返回 true

满了：

根据上一秒快照判断猴子位移，判断梯子行进方向，筛选出适合的梯子，随机选择第一个踏板没有猴子的梯子上去，返

回 true。如果都有猴子，则原地等待，生成新快照，返回 true

决策的实现：

拍摄快照

```
//锁定上次快照，进行对比
synchronized (SnapShot.preleft) {
  synchronized (SnapShot.preright) {
    //上次快照为空或者上次快照时间不为1秒前（时效性已过），新建快照
    if ((SnapShot.preleft.get(monkey) == null && SnapShot.preright.get(monkey) == null)) {
      Map<Ladder, Position> lmap = new HashMap<Ladder, Position>();
      Map<Ladder, Position> rmap = new HashMap<Ladder, Position>();
      for (Ladder l : ladders) {
        synchronized (l) {
          Position leftpos = new Position(l.leftmost(), l.indexOf(l.leftmost()));
          Position rightpos = new Position(l.rightmost(), l.indexOf(l.rightmost()));
          lmap.put(l, leftpos);
          rmap.put(l, rightpos);
        }
      }
      SnapShot.preleft.put(monkey, lmap);
      SnapShot.preright.put(monkey, rmap);
      printWaitAndShot(monkey);
      LoggerTool.waiting(monkey);
      VisualTool.updateConsole(LoggerTool.waitingString(monkey));
      return true;
    }
```

计算各个梯子的偏移量

```
List<Integer> bias = new ArrayList<Integer>();
```

R→L 观察最右边猴子的位置变化，如果猴子不见说明到岸了

```
int newposr =
    l.indexOf(SnapShot.preright.get(monkey).get(l).getMonkey());//上一次快照最右猴子新位置

int biasr = newposr - SnapShot.preright.get(monkey).get(l).getPosition();
```

L→R 观察最左边猴子的位置变化

```
int newposl =
    l.indexOf(SnapShot.preleft.get(monkey).get(l).getMonkey()); //上一次快照最左猴子新位置

int biasl = newposl - SnapShot.preleft.get(monkey).get(l).getPosition();
```

在这里，如果是向右前进，则偏移量是正的，反之偏移量是负的。根据做决策猴子的方向选择适当的梯子加入 tmp 列表中，随机选择合适梯子上去

上梯子实现：

```
synchronized ((r = tmp.get(new Random().nextInt(tmp.size())))) {
  if ((monkey.getDirection().equals("L->R") && !r.contain(0))
      || (monkey.getDirection().equals("R->L") && !r.contain(r.size() - 1))) {
    r.add(monkey);
    LoggerTool.moving(monkey, ladders, r);
    VisualTool.updateConsole(LoggerTool.movingString(monkey, ladders, r));
    printObserveAndClimb(monkey, ladders, r);
    return true;
  }
}
```

在梯子上行进的实现：

首先使用 Ladder 中 occupy 方法判断有没有猴子挡路，返回最近猴子的距离，如果超出速度大小，则前进猴子速度的距离。如果小于速度，则前进到最近的猴子后面。最后，判断返回位置是否为-1，若是，则说明已经到达对岸，记录到岸状态，返回 false。

```
synchronized (ladder) {
  int distance = ladder.occupy(monkey, monkey.getDirection());
  int pos;
  if (distance > monkey.getSpeed() || distance == -1) {
    pos = ladder.move(monkey, monkey.getSpeed());
    LoggerTool.moving(monkey, ladders, ladder);
    VisualTool.updateConsole(LoggerTool.movingString(monkey, ladders, ladder));
    printMove(monkey, monkey.getSpeed(), ladders, ladder);
  } else {
    pos = ladder.move(monkey, distance - 1);
    LoggerTool.moving(monkey, ladders, ladder);
    VisualTool.updateConsole(LoggerTool.movingString(monkey, ladders, ladder));
    printMove(monkey, distance - 1, ladders, ladder);
  }
  if (pos == -1) {
    LoggerTool.landing(monkey);
    VisualTool.updateConsole(LoggerTool.landingString(monkey));
    printLand(monkey);
    return false;
  }
}
return true;
```

3.3.2 策略 2

这个策略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、梯子上离我距离最近的猴子的真实行进速 度最快），如果没有符合条件的梯子其次选择空梯子，如果空梯子也没有就原地等待

决策流程：

判断是否已在梯子上：
- 是：
  - 在梯子上前进
- 否：
  - 之前是否有快照：
    - 没有：生成快照，原地等待
    - 有：
      - 计算偏移量，将移动速度由高到低排序，接着比较猴子数量，将相同速度猴子较少的梯子排在前面。从前往后选择梯子上去。
        - 是否有符合条件的梯子：
          - 有：选择上梯
          - 没有：选择空梯子，如果没有空梯子，原地等待

梯子的比较实现：

实现比较器 Comparator 比较速度

```
class MapValueComparator1<Ladder> implements Comparator<Map.Entry<Ladder, Integer>> {
  @Override
  public int compare(Entry<Ladder, Integer> o1, Entry<Ladder, Integer> o2) {
    int a1 = o1.getValue();
    int a2 = o2.getValue();
    return a2 - a1;
  }
}
```

冒泡排序比较猴子数量（只有 10 个梯子，速度不会很大影响）

```
// 比较猴子数量
for (int i = 0; i < entryList.size() - 1; i++) {
  if (entryList.get(i).getValue() == entryList.get(i + 1).getValue()) {
    if (entryList.get(i).getKey().sizeOfMonkeys()
          > entryList.get(i + 1).getKey().sizeOfMonkeys()) {
      Collections.swap(entryList, i, i + 1);
    }
  }
}
```

其余实现相同。

3.3.3 策略 3

这个策略是等到梯子完全空闲才登上梯子

决策流程：

是否在梯子上：
- 是：
  - 向前进
- 否：
  - 是否有空梯子：
    - 是：选一个上去
    - 否：原地等待

此实现不需要快照，直接判断即可

3.3.4 策略 4

这个策略优先选择离自己最近的猴子真实行进速度与自己最大速度最接近的梯子，其次选择空梯子

决策流程：

判断是否已在梯子上：
- 是：
  - 在梯子上前进
- 否：

之前是否有快照：
没有：生成快照，原地等待
有：
计算偏移量，选择速度与自己速度最接近的梯子，优先相同，其次偏移量在±2 以内。
是否有符合条件的梯子：
有：选择上梯
没有：选择空梯子，如果没有空梯子，原地等待

选择速度相近梯子的实现：

将方向相同的梯子中速度匹配的筛选出来，使用 List 和 add(index, object)方法实现将不同速度梯子插入列表。这里需要注意，向左方向的梯子偏移量是负的，因此在比较前需要对猴子速度进行处理，向左猴子速度变为负的，这样计算出的差为正。

```
//找到同方向上偏移量最接近本身速度的梯子，优先相同，其次偏差量在 +-2 以内
int zerosize = 0;
int onesize = 0;
int twosize = 0;
int speed = monkey.getSpeed();
if (monkey.getDirection().equals("R->L")) {
  speed = -speed;
}
List<Ladder> candidate = new LinkedList<Ladder>();
for (Ladder l : tmp) {
  if (bias.get(l) == speed) {
    candidate.add(0, l);
    zerosize++;
  } else if (Math.abs(bias.get(l) - speed) == 1) {
    candidate.add(zerosize, l);
    onesize++;
  } else if (Math.abs(bias.get(l) - speed) == 2) {
    candidate.add(onesize + zerosize, l);
    twosize++;
  }
}
```

3.3.5 策略 5

这个策略优先选择离自己最近的猴子真实行进速度与自己最大速度最接近的梯子（误差不超过 2 且方向相同），如果某只猴子最大速度为 2 及以上，则猴子不能选择速度为 1 的梯子.
若没有符合条件的梯子则选择空梯子，如果猴子行进速度为 1、2 或 3，则在选择空梯子时至少保留总梯子数量的 40%的空梯子，如果不满 1 条则补足为 1 条。剩余梯子不足时，该猴子原地等待。（如果只有一条梯子，则该猴子可以上去）
对于速度超过 3 的猴子，梯子没有限制

设计思路：

速度为 1 的猴子在 20 个踏板上行进需要 20 秒，而 2 只要 10 秒，3 只要 7 秒，可见速度为 1 的猴子需要严格的与其他猴子分开。因此一个梯子一旦上了 1 的猴子，就只允许上速度为 1 的猴子，防止速度下降。同时，为了给速度比较好的猴子腾出快速通道，每次选择时低速猴子都必须留出冗余梯子给高速猴，防止出现全面堵塞的情况。

决策流程：

判断是否已在梯子上：

是：

在梯子上前进

否：

之前是否有快照：

没有：生成快照，原地等待

有：

计算偏移量，选择速度与自己速度最接近的梯子，优先相同，其次偏移量在±2 以内。特别的，如果猴子速度为 2 或以上，剔除行进速度为 1 或 0 的梯子。

是否有符合条件的梯子：

有：选择上梯

没有：

选择空梯子，特别的，如果猴子速度为 1，2，3，首先计算出最少保留空梯数量。然后筛选空梯，知道空梯数量达到要求后，如果遇到多余的空梯，则登上梯子。特别的，如果梯子总数为 1，直接登梯。

没有空梯，原地等待。

筛选过程实现：

添加额外的条件筛选

```
if (bias.get(l) == speed) {
  candidate.add(0, l);
  zerosize++;
} else if (Math.abs(bias.get(l) - speed) == 1) {
  if (monkey.getSpeed() >= 2 && Math.abs(bias.get(l)) < 2) {
    continue;
  }
  candidate.add(zerosize, l);
  onesize++;
} else if (Math.abs(bias.get(l) - speed) == 2) {
  if (monkey.getSpeed() >= 2 && Math.abs(bias.get(l)) < 2) {
    continue;
  }
  candidate.add(onesize + zerosize, l);
  twosize++;
} else if (monkey.getSpeed() >= 9 && Math.abs(bias.get(l) - speed) == 3) {
  candidate.add(twosize + onesize + zerosize, l);
  threesize++;
}
```

空梯子选择实现：

```
int keep = (int) (ladders.size() * 0.4);
if (keep < 1) {
  keep = 1;
}
int count = 0;
for (Ladder l : ladders) {
  synchronized (l) {
    if (l.isEmpty()) {
      if (monkey.getSpeed() <= 3 && count >= keep) {
```

3.3.6 策略 6

这个策略是见到梯子就上，只要是同方向就可以

决策流程：

判断是否已在梯子上：
是：
在梯子上前进
否：
看到空梯子就上
之前是否有快照：
没有：生成快照，原地等待
有：
计算偏移量，选择方向相同的梯子
是否有符合条件的梯子：
有：选择上梯
没有：
选择空梯子，没有空梯，原地等待。

3.3.7 策略 7

这个策略是先选择空梯子, 再计算同方向上梯子猴子速度, 如果在梯子上最后一个猴子离开梯子之前当前猴子不会和它相遇, 则登上梯子, 之后选择速度与自己相差 40%以内的梯子.

决策流程：

判断是否已在梯子上：
是：
在梯子上前进
否：
之前是否有快照：
没有：生成快照，原地等待, 如果此时有空梯子, 登上梯子
有：

计算偏移量，看有没有同方向梯子，在梯子上最后一个猴子离开梯子之前当前猴子不会和它相遇，登梯。其次选择速度与自己速度最接近的梯子，优先相同，其次偏移量在 40%以内。

是否有符合条件的梯子：

有：选择上梯

没有：

原地等待。

计算相遇实现：

```
double suppose = ((double)l.size() / monkey.getSpeed()) + 1;
pos = l.indexOf(l.leftmost());
if (((double)(l.size() - pos) / bias.get(l)) + 2.7 <= suppose) {
  candidate.add(l);
}
```

3.4 “猴子生成器”MonkeyGenerator

猴子生成器中需要提供一个方法来产生猴子并启动猴子，该方法所需要输入的参数为间隔 t，每秒产生的猴子数量 k，猴子总数 N，最大速度 mv。同时，生成器还需要设置所要登陆的梯子对象，用来作为参数传入 Monkey 线程中。

实现过程：

生成器的**属性**

ladders：保存梯子

selectors（List<Select>）：保存所有的策略

direction（List<String>）：保存所有的方向

throughput（double）：记录最近一次模拟的吞吐率

fairness（double）：记录最近一次模拟的公平性

isOver（boolean）：标志该模拟是否结束

方法：

public void start(int t, int k , int K, int mv)

这个方法定时定量产生 Monkey 线程然后启动他们，在生成时随机分配选择策略和速度以及方向。在启动完所有线程后，使用 CountDownLatch 的 await 方法等待所有子线程执行完毕。随后计算吞吐率和公平性，将 isOver 标志设为 true，结束运行。

随机策略的设置：

```
selectors.get(random.nextInt(selectors.size())),
```

等待线程终止：

```
countDownLatch.await(); //等待所有猴子线程结束
```

public void setLadders()

设置所要模拟用的一组梯子

3.5 如何确保 threadsafe？

在猴子过河程序中，可能出现线程竞争的情景有：

1. 猴子选择梯子时，在选择梯子后未登上梯子前，被其他猴子抢占梯子
2. 猴子在梯子上前进时，有猴子挡路，找到前进距离后，在没有移动之前，其他猴子已经移走。

因此，为了确保线程安全，就需要确保：在这只猴子进行观察到移动或上梯这一系列动作时，保证其他的猴子线程不能拿到这个梯子的资源。

同时，这个猴子的线程只能锁当前的这个梯子，而不能将所有梯子锁住，不然就会影响到其他不相关猴子做决策的速度。

所以，在每只猴子针对某个梯子做决策时，需要将当前梯子用 synchronized 锁住，虽然 ladder 的每一个操作都是原子化的，但是为了防止操作之前出现竞争，需要将多个操作原子化，确保 threadsafe

```
for (Ladder l : ladders) {
  synchronized (l) {
    if (l.isEmpty()) {
      l.add(monkey);
      LoggerTool.moving(monkey, ladders, l);
      VisualTool.updateConsole(LoggerTool.movingString(monkey, ladders, l));
      printClimb(monkey, ladders, l);
      return true;
    }
  }
}
```

3.6 系统吞吐率和公平性的度量方案

1. 吞吐率的计算

吞吐率 = 总猴子数 / 总时间

在猴子生成器 start 方法开始执行时，保存一个 final 的系统时间作为开始时间

```
long starttime = System.currentTimeMillis();
```

在等待所有线程结束后，保存结束时间，传入时间差和猴子数，计算吞吐率

```
countDownLatch.await(); //等待所有猴子线程结束
throughput = (double)N / ((System.currentTimeMillis() - starttime) / 1000);
```

2. 公平性的计算

设置一个全局静态 Map birth 记录每个猴子的 ID 对应的出生时间戳，在线程启动时记录

```
SnapShot.birth.put(m.getID(), System.currentTimeMillis());
```

在每个 Monkey 线程内部，结束时记录死亡时间，保存在 death 中

```
SnapShot.death.put(id, System.currentTimeMillis());  //将死亡时间存入map
```

调用猴子生成器中的 getFairness 方法，传入猴子数量，计算公平性，公式由手册中提供

```
public double getFairness(int N) {
  int faircount = 0;
  for (int i = 0; i < N - 1; i++) {
    for (int j = i + 1; j < N; j++) {
      if ((SnapShot.birth.get(j) - SnapShot.birth.get(i))
            * (SnapShot.death.get(j) - SnapShot.death.get(i))
            >= 0) {
        faircount += 1;
      } else {
        faircount += -1;
      }
    }
  }
  double c = factorial(N) / (factorial(2) * factorial(N - 2));
  return ((double)faircount / c) == Double.NaN ? 0 : ((double)faircount / c);
}
```

3.7 输出方案设计

1. 日志

格式如下

等待：

```
"Monkey " + monkey.getID() + " is waiting on the LEFT bank, "
+ "time: " + monkey.getTime()
```

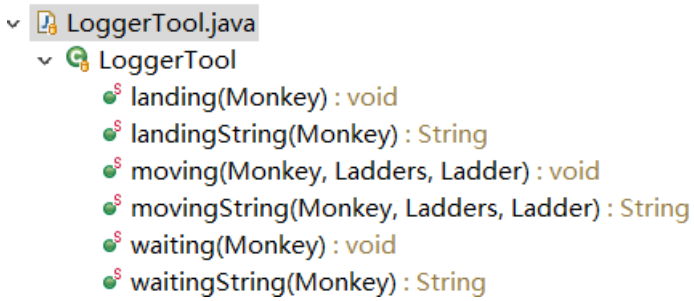
移动:

```
"Monkey " + monkey.getID() + " is on ladder " +
ladders.indexOf(ladder) + " rung " + ladder.indexOf(monkey) +
" ,direction: " + monkey.getDirection() + ", time: " +
monkey.getTime()
```

到岸：

```
"Monkey " + monkey.getID() + " is landing on the RIGHT bank from
LEFT bank, " + "time: " + monkey.getTime()
```

对于每个猴子，单独生成一个logger，在决策器做决策后，调用LoggerTool中的方法记录日志。LoggerTool是一个工具类，用于生成描述字符串和提供日志输出。



最终效果:

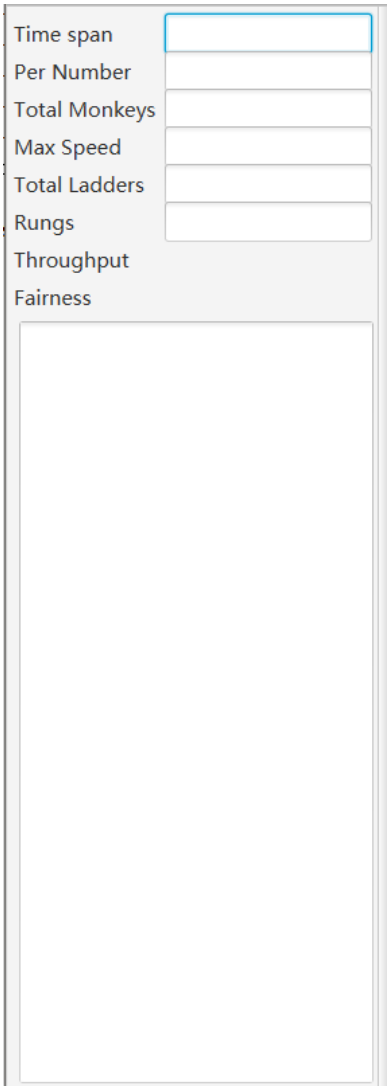
```
2019-06-08 00:36:14 [INFO] [MonkeyGenerator] Start to simulate, Time span: 5s, Per number: 30, Total: 300
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 0 is on ladder 0 rung 0 ,direction: L->R, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 10 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 6 is on ladder 3 rung 19 ,direction: R->L, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 12 is waiting on the RIGHT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 4 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 11 is on ladder 1 rung 19 ,direction: R->L, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 22 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 9 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 15 is on ladder 4 rung 0 ,direction: L->R, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 13 is waiting on the RIGHT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 2 is on ladder 2 rung 19 ,direction: R->L, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 29 is waiting on the RIGHT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 21 is waiting on the RIGHT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 27 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 14 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 28 is waiting on the RIGHT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 25 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 16 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 19 is waiting on the LEFT bank, time: 0
2019-06-08 00:36:14 [INFO] [LoggerTool] Monkey 17 is waiting on the LEFT bank, time: 0
```

2. GUI

使用 Javafx 构建 GUI，GUI 中输出信息有以下内容：

- 间隔时间
- 每次生成猴子数
- 总共猴子数
- 最大速度
- 梯子总数
- 踏板数
- 吞吐率
- 公平性
- 控制台，用于输出猴子的状态

整体界面如下



输出实现：

使用 Controller 控制除了控制台外的输出

由于控制台输出是动态过程，为了实现 UI 的动态更新，设计了可视化工具类 ViusalTool，提供对于 GUI 界面的一些绘图方法。

updateConsole 用于更新控制台输出

updateInfo 用于更新吞吐率和公平性

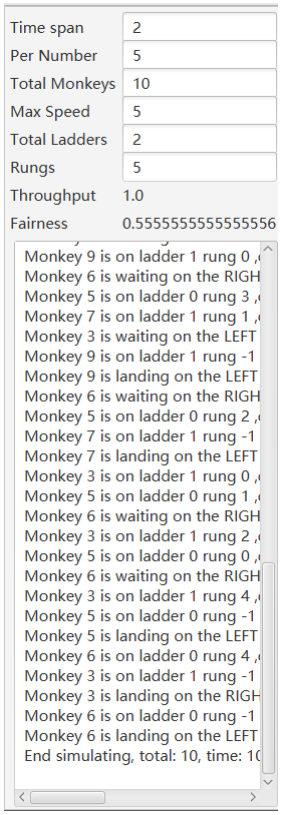
```
/**
 * .添加菜单中的控制台输出，输出语句会换行
 * @param text 待输出语句
 */
public static void updateConsole(String text) {
  if (console != null) {
    Platform.runLater(new Runnable() {
      @Override
      public void run() {
        console.appendText(text + "\n");
      }
    });
  }
}
```

```
/**
 * .更新参数信息
 */
private void updateInfo() {
  Platform.runLater(new Runnable() {
    @Override
    public void run() {
      fairness.setText(String.valueOf(mg.getFairness()));
      throughput.setText(String.valueOf(mg.getThroughput()));
    }
  });
}
```

在决策器决策行动后，会调用相关方法更新 UI

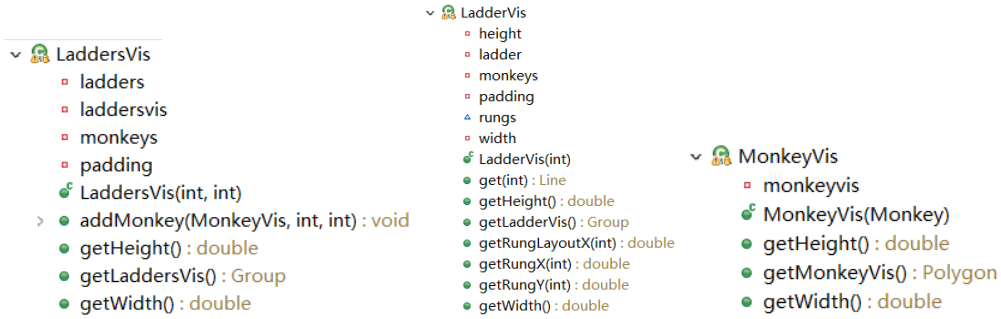
```
VisualTool.updateConsole(LoggerTool.movingString(monkey, ladders, ladder));
```

最终效果：



3. 可视化

可视化使用 Javafx 实现，为了便于对各个类型的图像操作，设计了 Ladder，Ladders 和 Monkey 的图像包装类，类中提供方法获取相关图像，图像生成，和一些方法比如将猴子图像加到指定梯子的指定踏板上。



根据猴子速度的不同，猴子有 5 种颜色

速度为 1： 

速度 2-3： 

速度 4-5： 

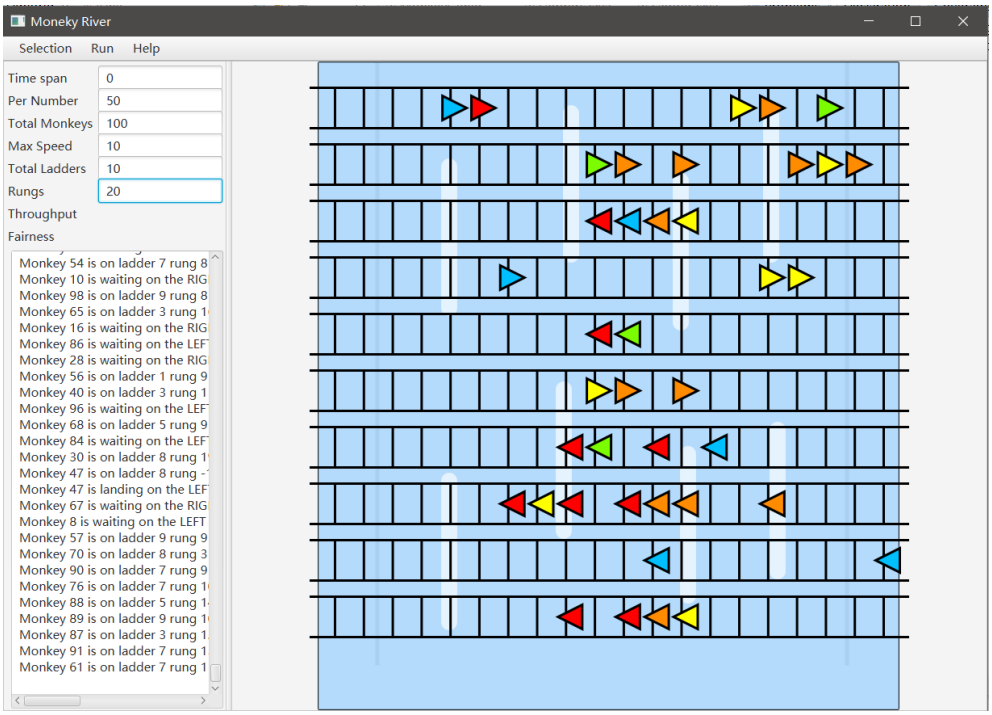
速度 6-8： 

速度 9-10： 

为了实现猴子过河的动态更新，需要实现将更新 UI 用多线程操作，为此将上面的 VIsualTool 继承 Thread 类，在开始仿真时同时启动 VisualTool 线程计算每秒的图像，使用 Runlater 方法更新 JavaFXApplicationThread 这个 UI 线程。

```
/**
 * .将Ladders图像添加到GUI
 * @param lvis ladders图像
 */
private void updateLadder(Group lvis) {
  Platform.runLater(new Runnable() {
    @Override
    public void run() {
      visualpane.getChildren().clear();
      visualpane.getChildren().add(lvis);
    }
  });
}
```

最终结果：



3.8 猴子过河模拟器 v1

3.8.1 参数如何初始化

v1 版本不支持配置文件输入。

参数的输入通过 GUI 的交互实现，用户通过 GUI 的文本框输入参数，然后点击 Run 选项，GUI 后台控制器 Controller 会解析出参数，并生成新的 Ladders 对象，传入猴子生成器，启动生成器的 start 方法

```
if (!isFile) {
  t = Integer.parseInt(timespan.getText());
  N = Integer.parseInt(totalnum.getText());
  k = Integer.parseInt(pernum.getText());
  MV = Integer.parseInt(mv.getText());
  ladders = new Ladders(Integer.parseInt(totalladders.getText()),
                Integer.parseInt(rungnum.getText()));
}
```

这里由于 Javafx 是单线程 UI，所以为了防止猴子生成器会占用 UI 线程，需要单开一个新线程

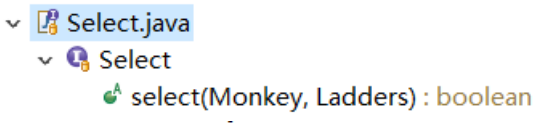
```
//start
Thread mgthread = new Thread() {
  @Override
  public void run() {
    mg.start(t, k, N, MV);
  }
};
this.mgthread = mgthread;
mgthread.setName("mgthread");
mgthread.start();
```

这样就实现了参数的初始化，而参数随机的实现是在生成器中

```
directions.get(random.nextInt(2)),
random.nextInt(mv) + 1,
selectors.get(random.nextInt(selectors.size())),
ladders,
countDownLatch);
```

3.8.2 使用 Strategy 模式为每只猴子选择决策策略

策略接口：




```
/**
 * .执行选择策略，如果抵达对岸返回false
 * @param monkey 待执行猴子
 * @param ladders 梯子集合
 * @return 如果处于等待或移动状态返回true，到岸 返回false
 */
boolean select(Monkey monkey, Ladders ladders);
```

每个策略都实现了 select 方法

```
public class Selector1 implements Select {

  @Override
  public boolean select(Monkey monkey, Ladders ladders) {
```

在猴子生成器内部保存了所有策略的 List

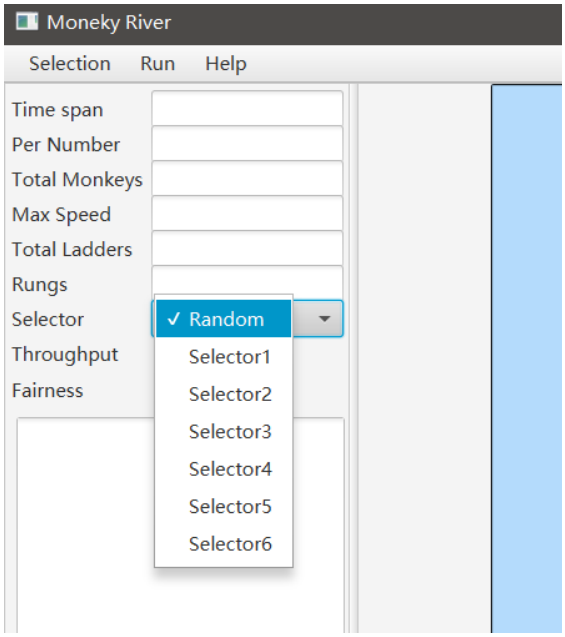
```
private List<Select> selectors = Arrays.asList(
    new Selector1(),
    new Selector2(),
    new Selector3(),
    new Selector4(),
    new Selector5(),
    new Selector6()
    );
```

猴子自身有个参数 Select 接收策略，在生成猴子时，List 中随机选择策略传给猴子。

```
selectors.get(random.nextInt(selectors.size())),
```

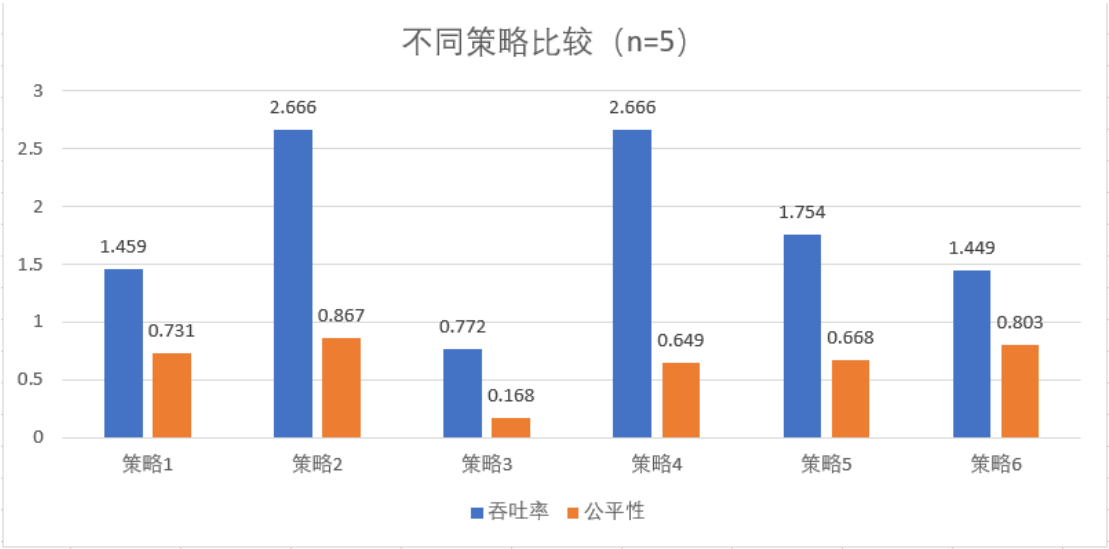
3.9 猴子过河模拟器 v2

策略选择方法：

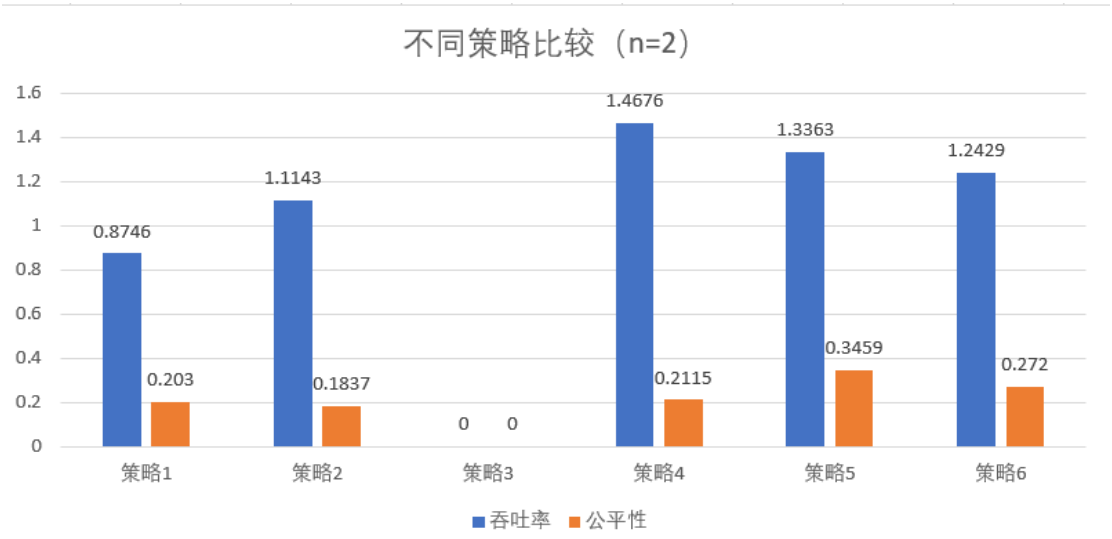


3.9.1 对比分析：固定其他参数，选择不同的决策策略

在 n=5,h=20,t=5,k=20,N=200,mv=10 条件下，测试不同策略的结果

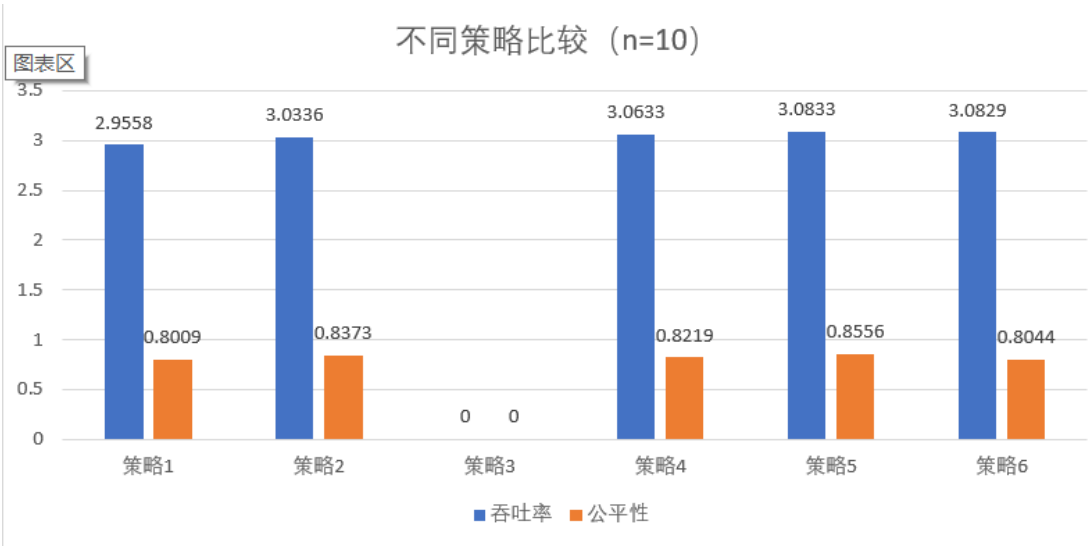


在 n=2,h=20,t=5,k=20,N=200,mv=10 的条件下测试结果（梯子变为 2）（每个策略测试 3 次，取平均值）



在 n=10,h=20,t=5,k=20,N=200,mv=10 条件下测试（梯子数为 10）

（测试 3 次，取平均值）



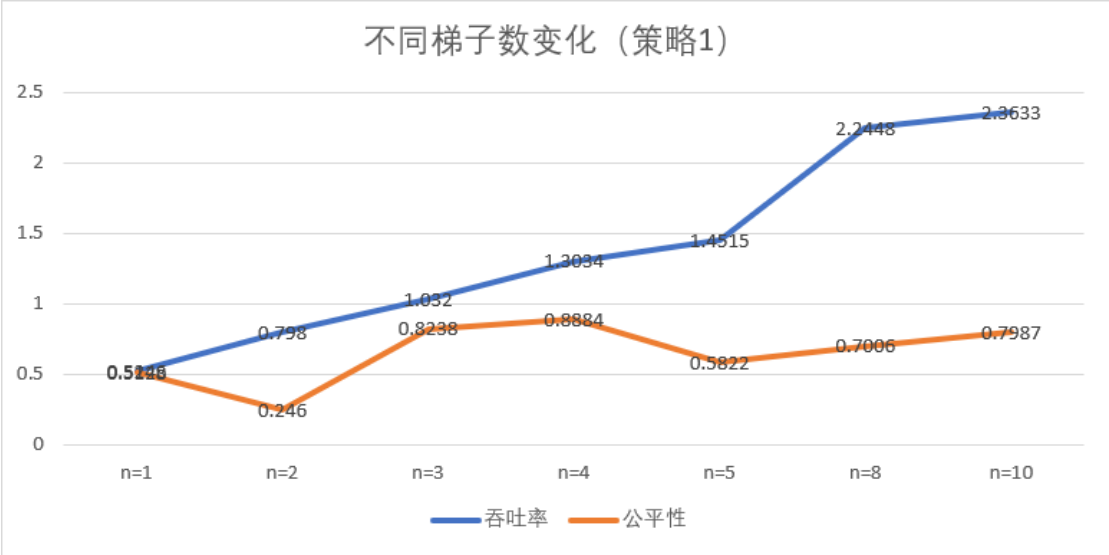
3.9.2 对比分析：变化某个参数，固定其他参数

1. 梯子数量

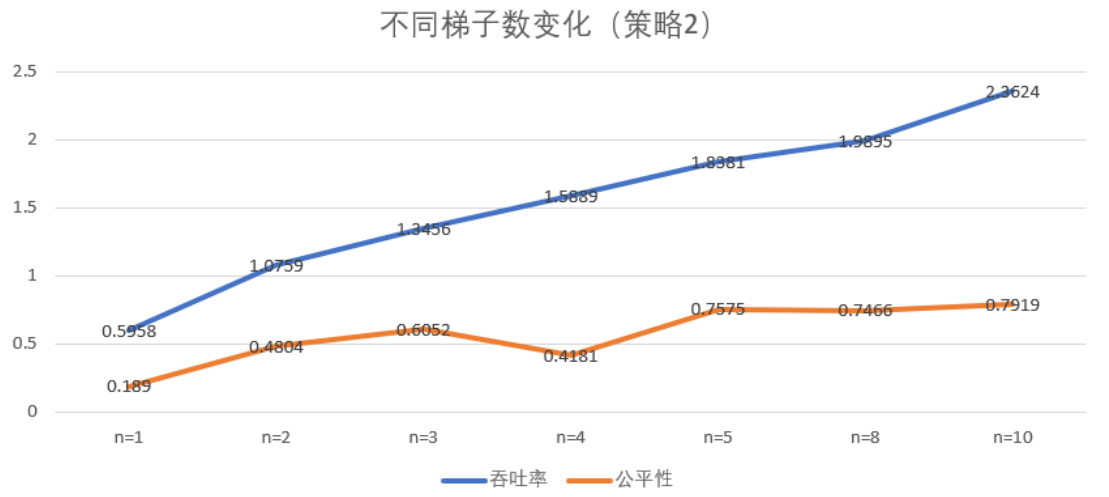
固定其他参数为 k = 20, N = 100 , mv = 5, h = 20， t=5

梯子数量从 1 变化到 5,8,10

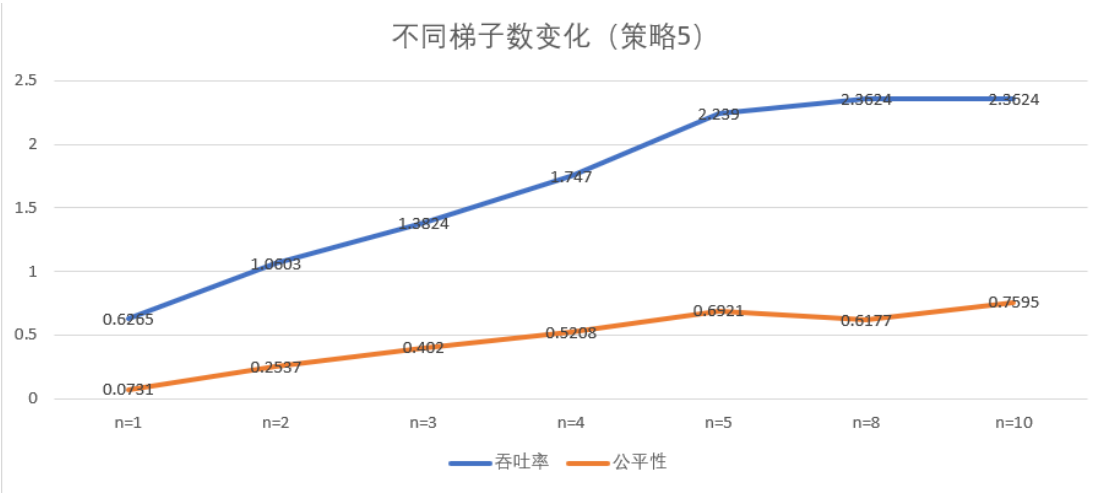
策略 1：


不同梯子数变化（策略1）
2.5
2
1.5
1
0.5
0
0.798
1.032
1.3034
1.4515
2.2448
2.3633
0.8238
0.8884
0.5822
0.7006
0.7987
0.246
n=1
n=2
n=3
n=4
n=5
n=8
n=10
吞吐率
公平性

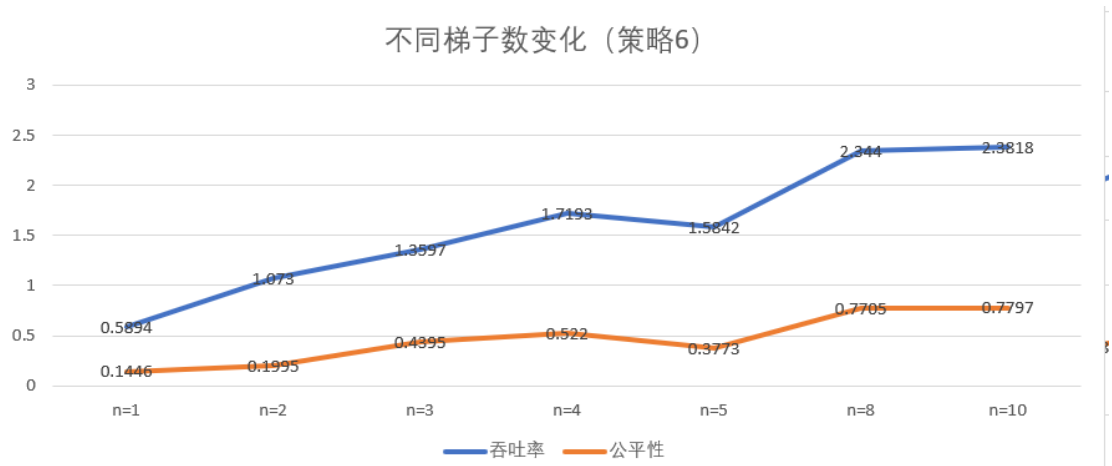
策略 2


不同梯子数变化（策略2）
2.5
2
1.5
1
0.5
0
0.5958
1.0759
1.3456
1.5889
1.8381
1.9895
2.3624
0.189
0.4804
0.6052
0.4181
0.7575
0.7466
0.7919
n=1
n=2
n=3
n=4
n=5
n=8
n=10
吞吐率
公平性

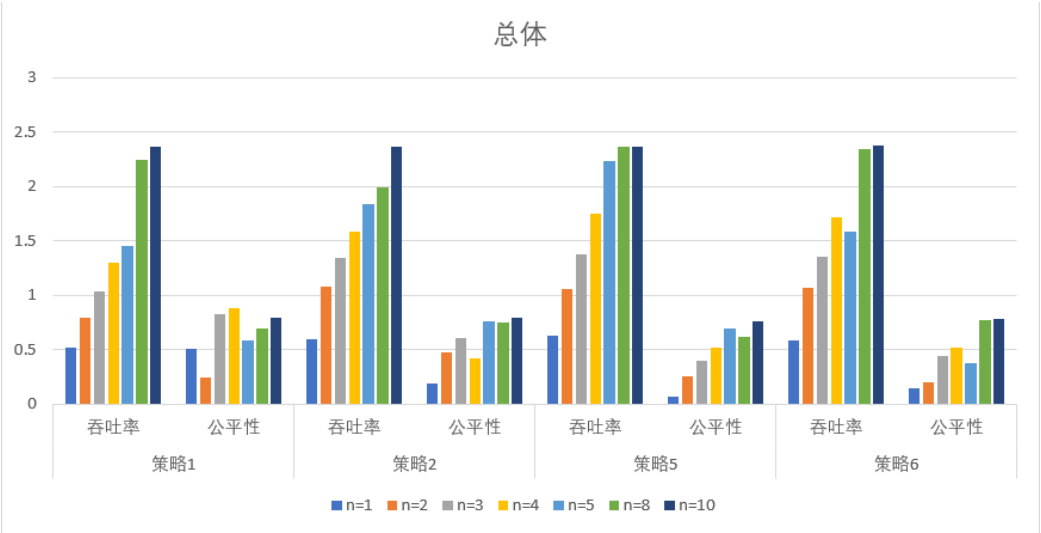
策略 5


不同梯子数变化（策略5）
2.5
2
1.5
1
0.5
0
0.6265
1.0603
1.3824
1.747
2.239
2.3624
2.3624
0.0731
0.2537
0.402
0.5208
0.6921
0.6177
0.7595
n=1
n=2
n=3
n=4
n=5
n=8
n=10
吞吐率
公平性

策略 6



总体比较：

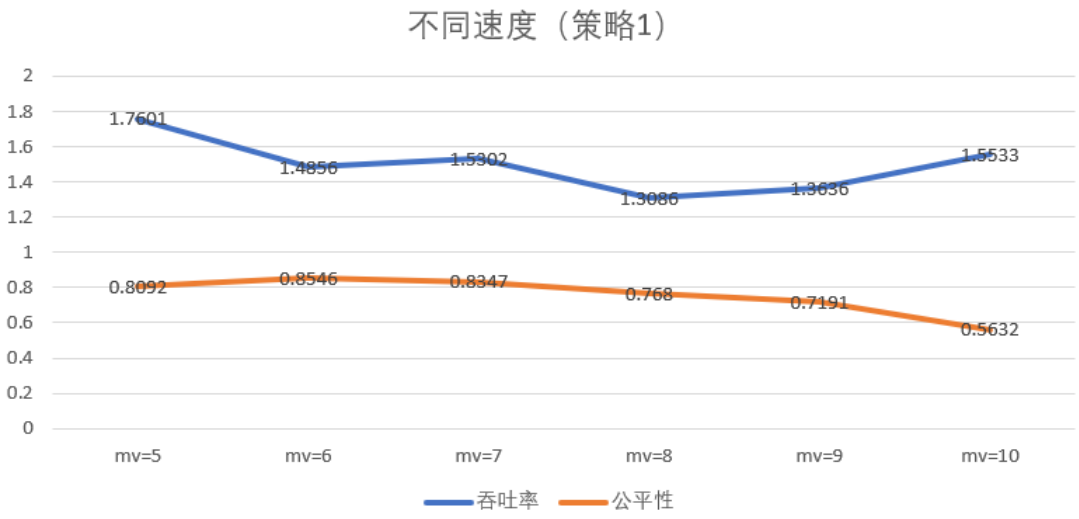


2.猴子最大速度

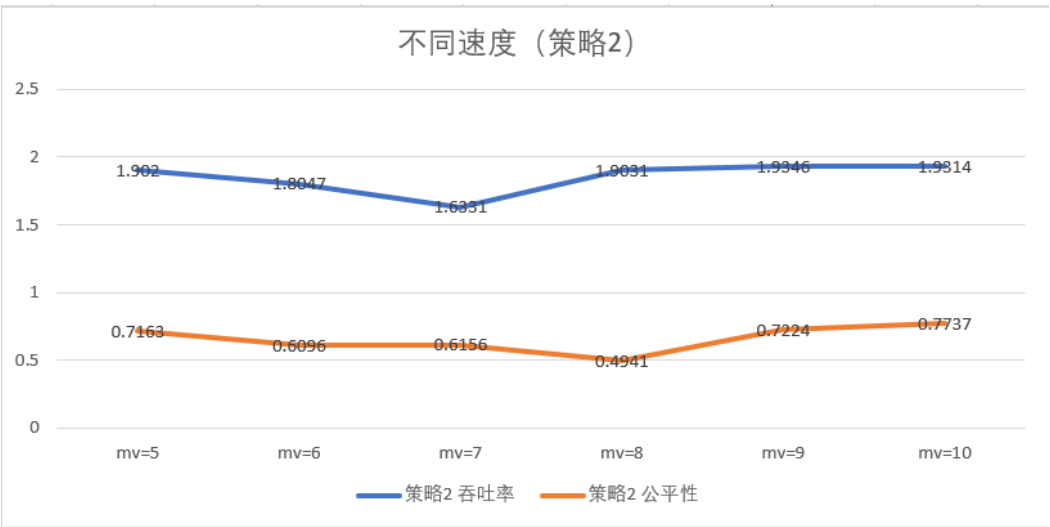
固定其他参数 k = 20, N = 100 , n = 5, h = 20，t=5

猴子最大速度从 5 变化到 10

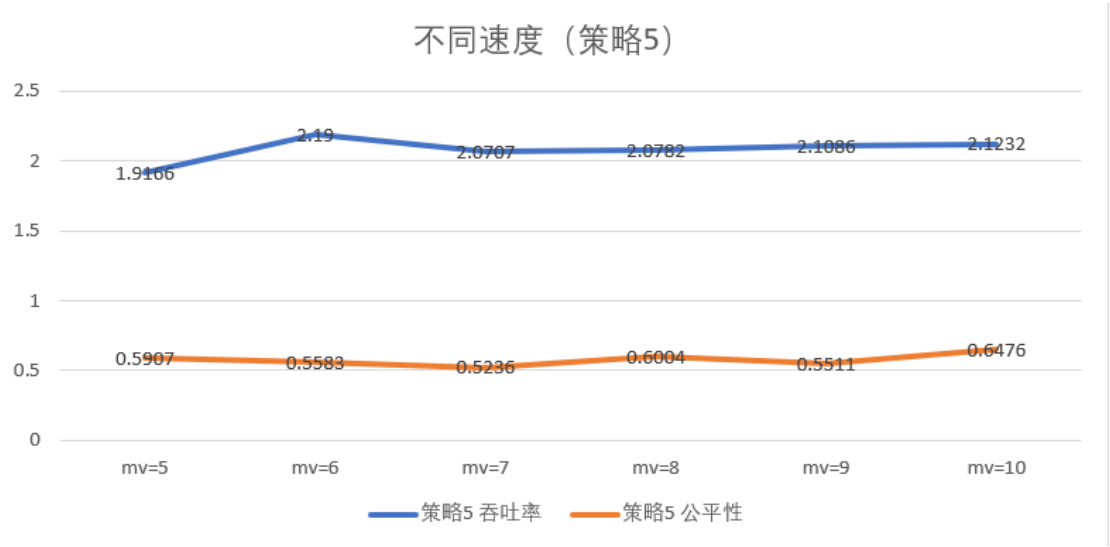
策略 1：



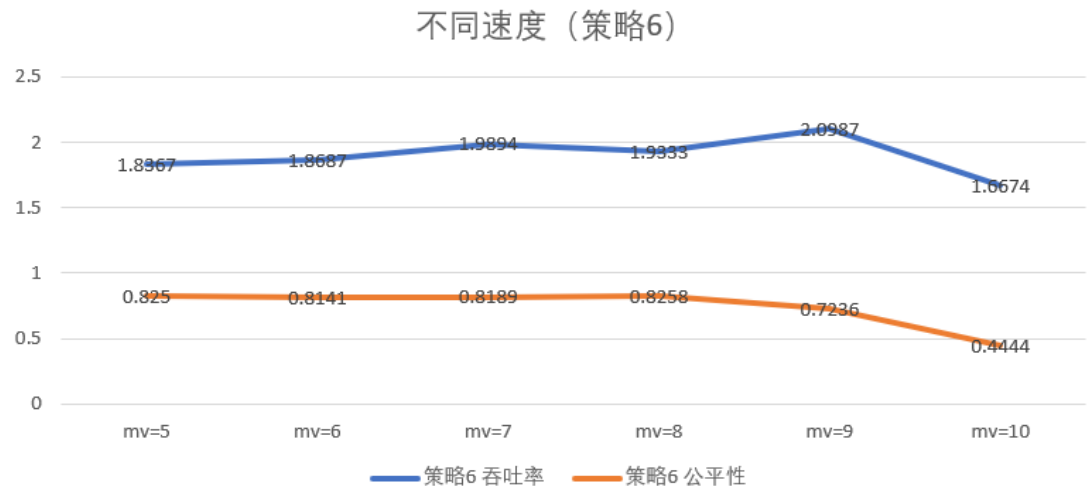
策略 2：


不同速度（策略2）
2.5
2
1.5
1
0.5
0
1.902
1.8047
1.6331
1.9031
1.9346
1.9814
0.7163
0.6096
0.6156
0.4941
0.7224
0.7737
mv=5
mv=6
mv=7
mv=8
mv=9
mv=10
策略2 吞吐率
策略2 公平性

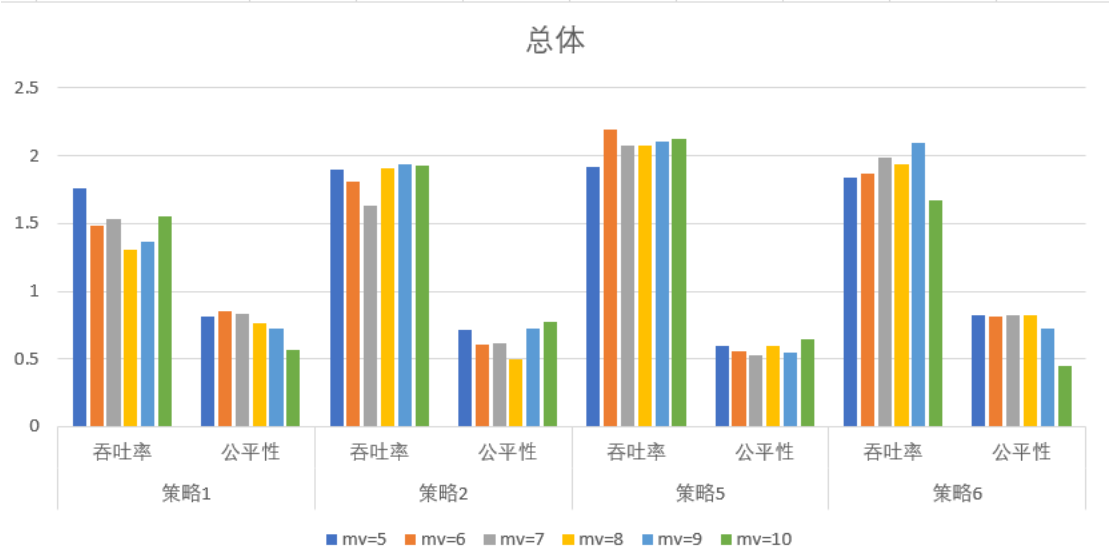
策略 5：


不同速度（策略5）
2.5
2
1.5
1
0.5
0
1.9166
2.19
2.0707
2.0782
2.1086
2.1232
0.5907
0.5583
0.5236
0.6004
0.5511
0.6476
mv=5
mv=6
mv=7
mv=8
mv=9
mv=10
策略5 吞吐率
策略5 公平性

策略 6：


不同速度（策略6）
2.5
2
1.5
1
0.5
0
1.8367
1.8687
1.9894
1.9333
2.0987
1.6674
0.825
0.8141
0.8189
0.8258
0.7236
0.4444
mv=5
mv=6
mv=7
mv=8
mv=9
mv=10
策略6 吞吐率
策略6 公平性

总体比较：



3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？

分析上述测试数据,得知

1. 吞吐率与决策策略有一定相关性

将不同速度猴子分层的策略的吞吐率要高于速度不分层的策略,将梯子空间尽量占满的策略的吞吐率要高于梯子占用不太满的策略.例如策略 5,策略 4 的吞吐率要高于速度不分层的策略 1,而占用率也是重要指标,比如相同选择策略,策略 5 要保留空梯子,因而吞吐率低于策略 4 不保留空梯子.

2. 梯子数量与吞吐率有正相关性,无论是何种策略

在所有策略中,梯子数量与吞吐率是正相关,随着梯子的数量提高,吞吐率有明显上升.

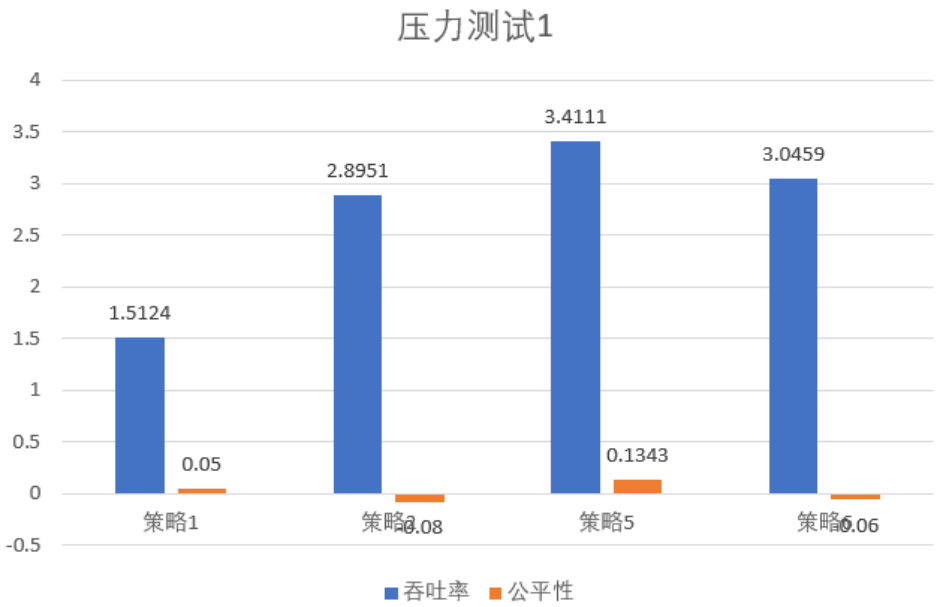
3. 猴子速度与吞吐率的相关性在梯子数量较少时相关性不明显,梯子数量上升时相关性开始显现

由图标可知,梯子数量为 5 时,速度从 5 增加到 10 的过程,吞吐率并没有很明显的变化,说明梯子数量限制了猴子速度的相关性.只有当梯子数量不再成为限制后,速度相关性才出现.但是相关测试没做,仅为理论推测

3.9.4 压力测试结果与分析

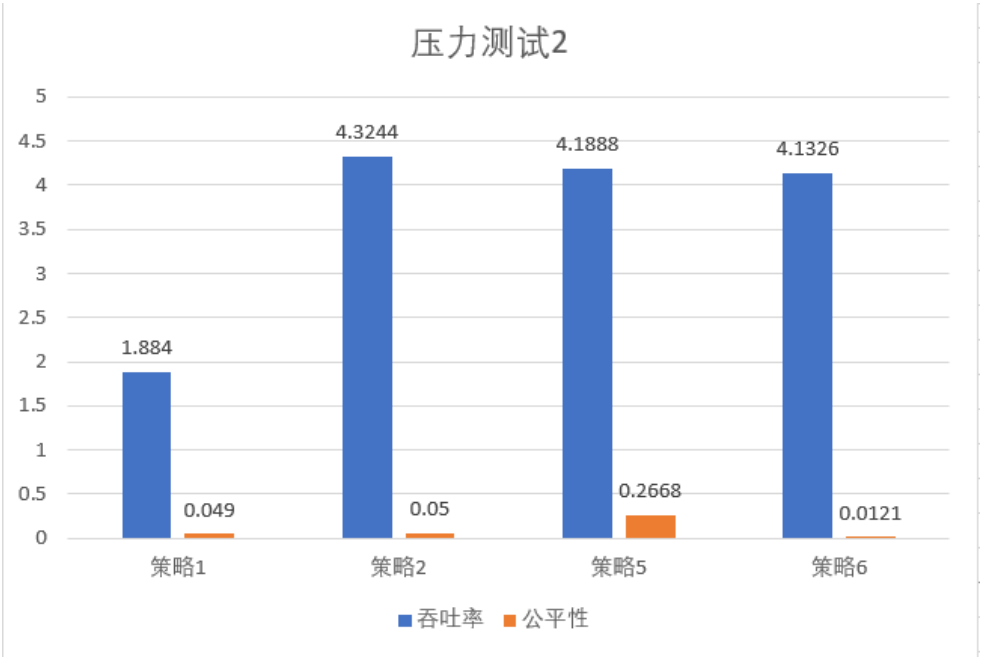
压力测试 1：

800 只猴子，80 只每秒，间隔 4 秒生成，4 个梯子，20 个踏板，最大速度为 7



在猴子数量很密集的情况下,策略 5 的吞吐率要明显高于其他吞吐率,说明将速度分层对于提高吞吐率有提高效果.

压力测试 2：
800 只猴子，100 只每秒，间隔 4 秒生成，6 个梯子，20 个踏板，速度为 1、2、3 和 8、9、10 这 6 种。



在猴子速度差异很大的情况下,后 3 个策略的吞吐率差异不大,但是策略 5 的公平性比较高,这是由于策略 5 为高速猴子提供了单独的空梯子,使得猴子不必等待其他慢速猴子走完才走.

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。

Competition1 使用策略 7

Competition2 使用策略 7

Competition3 使用策略 7

	吞吐率	公平性
Competiton_1.txt		
第 1 次模拟	3.191	0.5332
第 2 次模拟	3.0927	0.5443
第 3 次模拟	3.0	0.60664
第 4 次模拟	3.1578	0.3727
第 5 次模拟	2.654	0.180
第 6 次模拟	3.092	0.360
第 7 次模拟	2.884	0.384
第 8 次模拟	3.03	0.521
第 9 次模拟	3.125	0.313
第 10 次模拟	3.125	0.299
平均值	3.035	0.411
Competiton_2.txt		
第 1 次模拟	6.493	0.498
第 2 次模拟	6.8493	0.58186
第 3 次模拟	7.14285	0.6253
第 4 次模拟	6.8493	0.5744
第 5 次模拟	6.666	0.5978
第 6 次模拟	6.4102	0.5510
第 7 次模拟	6.493	0.5518
第 8 次模拟	6.578	0.545
第 9 次模拟	6.578	0.606
第 10 次模拟	6.849	0.541
平均值	6.714	0.564
Competiton_3.txt		
第 1 次模拟	1.369	0.319
第 2 次模拟	1.25	0.2460

第 3 次模拟	1.190	0.3115
第 4 次模拟	1.369	0.437
第 5 次模拟	1.538	0.514
第 6 次模拟	1.25	0.2117
第 7 次模拟	1.351	0.489
第 8 次模拟	1.428	0.461
第 9 次模拟	1.388	0.507
第 10 次模拟	1.176	0.296
平均值	1.33	0.379

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。

日期	时间段	计划任务	实际完成情况
6/3	20：00-22：00	写猴子线程和梯子以及一组梯子 ADT	猴子线程完成
6/4	18：00-22：00	继续写梯子 ADT 和相关方法	完成
6/5	18：00-20：00	编写策略 1 2 3 4	完成
6/6	8：00-9：45	写 GUI 输出	未完成
6/6	13：00-16：00	写 GUI 输出和梯子过河动画	完成输出未完成动画
6/6	18：00-20：00	进行猴子梯子相关绘图	完成
6/6	20：00-23：00	解决 GUI 绘图的多线程问题	完成
6/7	8：00-12：00	GUI 相关测试	完成
6/7	13：00-18：00	编写策略 5	完成
6/7	19：00-20：00	编写策略 6	完成
6/9	9：00-12：00	补全相关单元测试代码	完成
6/9	15：00-18：00	写注释和报告	未完成
6/11	19：00-22：00	写 v2 的策略选择	完成
6/12	19：00-23：00	写报告，跑相关测试策略代码	完成
6/14	18:00-20:00	写 v3，跑测试代码	未完成
6/14	20:00-22:00	写策略 7	完成
6/15	20:00-22:00	跑测试文件	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
如何设计猴子的 ADT 使得猴子能够单独做决策	上网搜索了猴子分桃子的例子，理解了将猴子看作线程共享梯子资源的实现方法

如何使得猴子在非上帝视角判断梯子上的行进方向	每只猴子在观察梯子后将梯子左右两只猴子的状态保存，下一秒做决策后将那两只猴子进行比较，做出位移判断方向
如何将更新 UI 的动作从 JavaFx Application Thread 中抽离，不抽离会导致 UI 线程卡死	在参考了大量博客后,仿照博客示例设计一个更新 UI 的新线程，绘图工作交给此线程，每次绘制完毕后，调用 runlater 通知 UI 线程渲染
绘图调用 Ladders 出现死锁 Java stack information for the threads listed above: "49": at items.Ladder.contain(Ladder.java:125) - waiting to lock <0x000000070292a0c0> (a items.Ladder) at tool.Selector3.select(Selector3.java:22) at items.Monkey.run(Monkey.java:80) "vtthread": at items.Ladders.get(Ladders.java:49) - waiting to lock <0x000000070291c688> (a items.Ladders) at tool.VisualTool.update(VisualTool.java:36) - locked <0x000000070292a0c0> (a items.Ladder) at tool.VisualTool.call(VisualTool.java:83) at tool.VisualTool.call(VisualTool.java:1) at javafx.concurrent.Task$TaskCallable.call(Task.java:1423) at java.util.concurrent.FutureTask.run(Unknown Source) at java.lang.Thread.run(Unknown Source) "35": at tool.Selector3.select(Selector3.java:32) - waiting to lock <0x000000070292a0c0> (a items.Ladder) - locked <0x000000070291c688> (a items.Ladders) at items.Monkey.run(Monkey.java:80) Found 1 deadlock.	在绘图时同样将梯子上锁，同时绘图工作与猴子做决策时间延迟 0.5 秒，避免冲突
如何提高吞吐率	经过分析后发现,尽量增大梯子的占有率对提高吞吐率有明显帮助,所以如果让速度为 2 和 3 \ 4 以上的猴子混在一起上梯要比全部分开要快.如果能够实现猴子不间断的上梯,即猴子能够在其他速度猴子跑完后及时登梯,保持占用率,也能够提高吞吐率

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

提高吞吐率不一定要将速度划分过于详细,将速度大概分区的同时提高梯子的占有率才能将吞吐率提高.

多线程编程一定要做好共享资源的检查工作,做好边界条件的检查,这里很容易出现问题,而且问题不易复现,不易处理.

如果要新建线程,最好能做好线程的管理工作,最好不要将匿名线程的建立和运行嵌入到代码中,这对后期调试会带来很大麻烦

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？

多线程可能涉及到对同一资源的使用和修改，由此可能会引发线程竞争的 bug，这种 bug 不定时出现，很难复现，也很难分析，这一点比单线程要复杂。

多线程能够在并行处理子任务，提高效率，比如在更新 GUI 时，就必须开一个子线程处理数据，否则就会占用 UI 线程。

(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？

在猴子对某一个梯子进行决策观察行动，特别是登梯时，将这个梯子锁住，来保证 threadsafe，为了平衡性能和线程安全，尽量将上锁压缩到必要范围，也就是不得不进行同步时上锁，在进行一些观察行为时不必对梯子上锁。

(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？

在绘制 GUI 时，绘图线程和猴子线程发生了死锁，两个线程互相争抢同一个梯子。最后将绘图线程中对于整体梯子对象调用时进行上锁，同时将绘图行为延迟了 0.5 秒，在猴子休眠时进行绘图，这样确保了性能不受损失。

(4) 关于本实验的工作量、难度、deadline。

本次实验的工作量不太大，但是难度，策略的设计难度较高，ddl 时间较为宽裕。

(5) 到此为止你对《软件构造》课程的意见和建议。

没啥太好的建议了，建议之前都提完了。。

(6) 还有一周就要期末考试了，你准备如何复习？

看 PPT，看期末卷，看之前写过的报告