哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2018 年春季学期

计算机学院大二软件构造课程

Lab 6 实验报告

姓名	朱明彦
学号	1160300314
班号	1603003
电子邮件	1160300314@stu.hit.edu.cn
手机号码	18846082306

目录

1 实验目标概述

本次实验主要针对的是并行编程的能力。根据一个具体的需求，实现两个线程安全的猴子过河模拟器。并通过数据模拟保证的线程安全。

主要训练的方面：

- Java 多线程编程
- 面向线程安全的 ADT 设计策略的选择与文档化
- 模拟仿真实验和对比实验
- 基本的 GUI 编程

2 实验环境配置

在这次的实验中，主要需要的实验环境与以往的实验环境类似，并没有新增其余的实验环境。因此没有再配置多余的实验环境。

在这里给出你的 GitHub Lab6 仓库的 URL 地址（Lab6-学号）。

https://github.com/ComputerScienceHIT/Lab6-1160300314

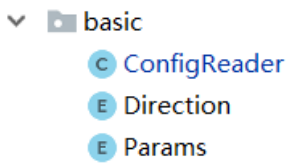
3 实验过程

3.1 ADT 设计方案

主要使用了 `basic`、`strategy` 等包，下面按照包结构进行解释设计

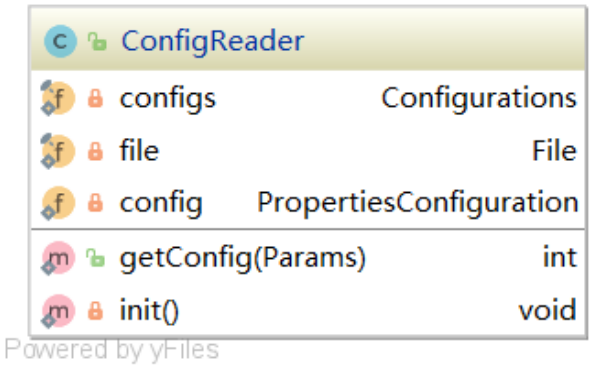
3.1.1 basic 包

包内的组织结构如下



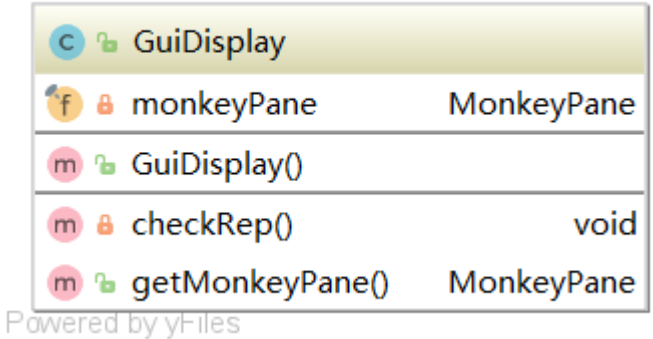
在 `basic` 包中有 `Direction`，`Params` 两个枚举类型，和 `ConfigReader` 一个类。其中 `Direction` 主要是用来规定猴子的方向只有两种 `R2L` 或者 `L2R`；对于 `Params` 枚举类型，主要是所有的参数可能的名字，如 `Params.h` 就是指

Config.properties 中的 h。
对于 ConfigReader 主要的功能就是实现在 config.properties 中读取配置信息。其详细的属性方法如下：



3.1.2 display 包

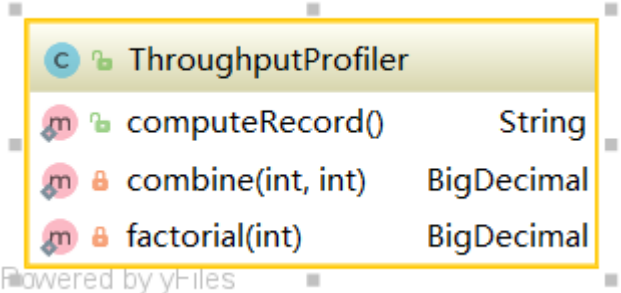
在这个包中只有一个 GuiDisplay 类，是用做使用 GUI 展示猴子过河的过程的类。其中的属性值与方法见下 UML 图



其中 MonkeyPane 是在 monkey 包中用做展示猴子位置的 Pane 类。

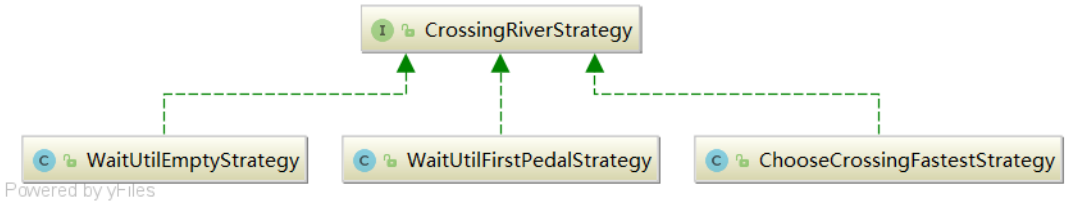
3.1.3 profiler 包

在 profiler 包中仅有 ThroughputProfiler 一个类，用做读取配置文件并分析其中的吞吐率指标和公平性指标。其中的属性与方法如下

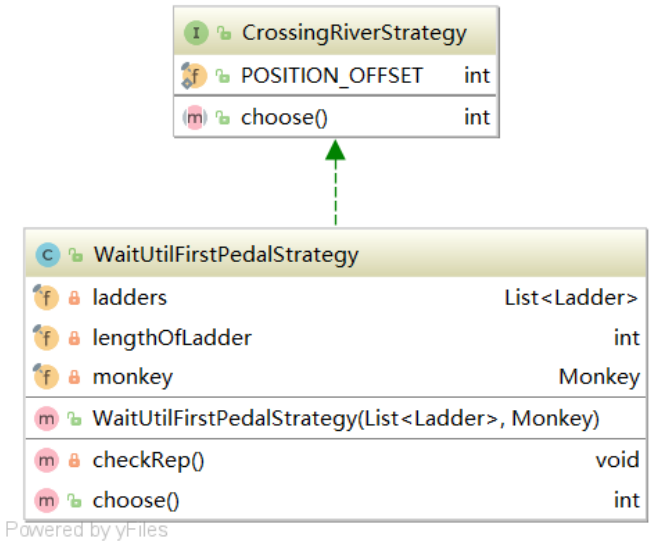


3.1.4 strategy 包

对于 strategy 包，主要是所有的策略类都在这个包中，其中的类之间的关系如下：



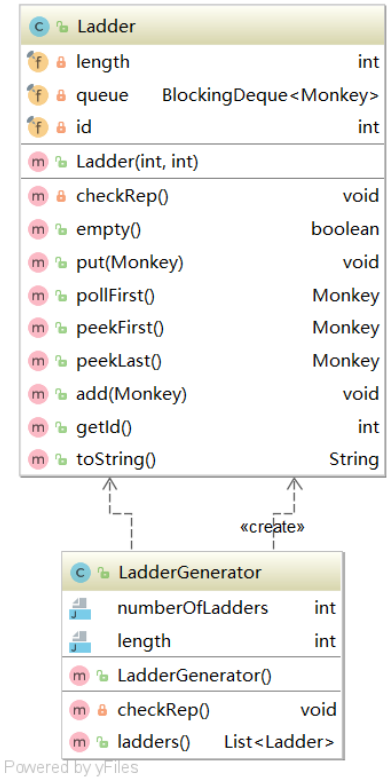
对于每一个实现 CrossingRiverStrategy 接口的类，都具有下面的结构



都必须实现 choose 方法，可以在不同的梯子之间进行选择。

3.1.5 ladder 包

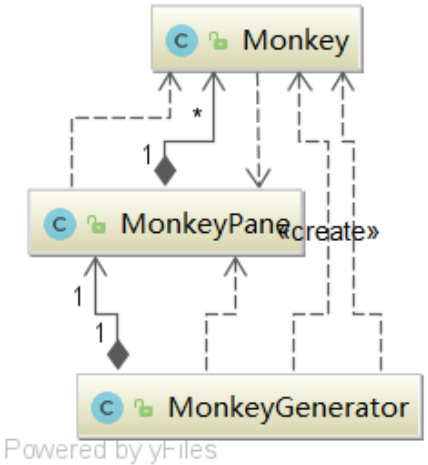
在 ladder 包中有两个类，Ladder 和 LadderGenerator 两个类。具体的方法和属性如下：



在其中对于 Ladder 类来说，由于需要保证在不同的猴子线程访问时能够保证线程安全，所以所有涉及到对其中 queue 进行修改的操作都是使用 synchronized 进行修饰，以保证线程安全。

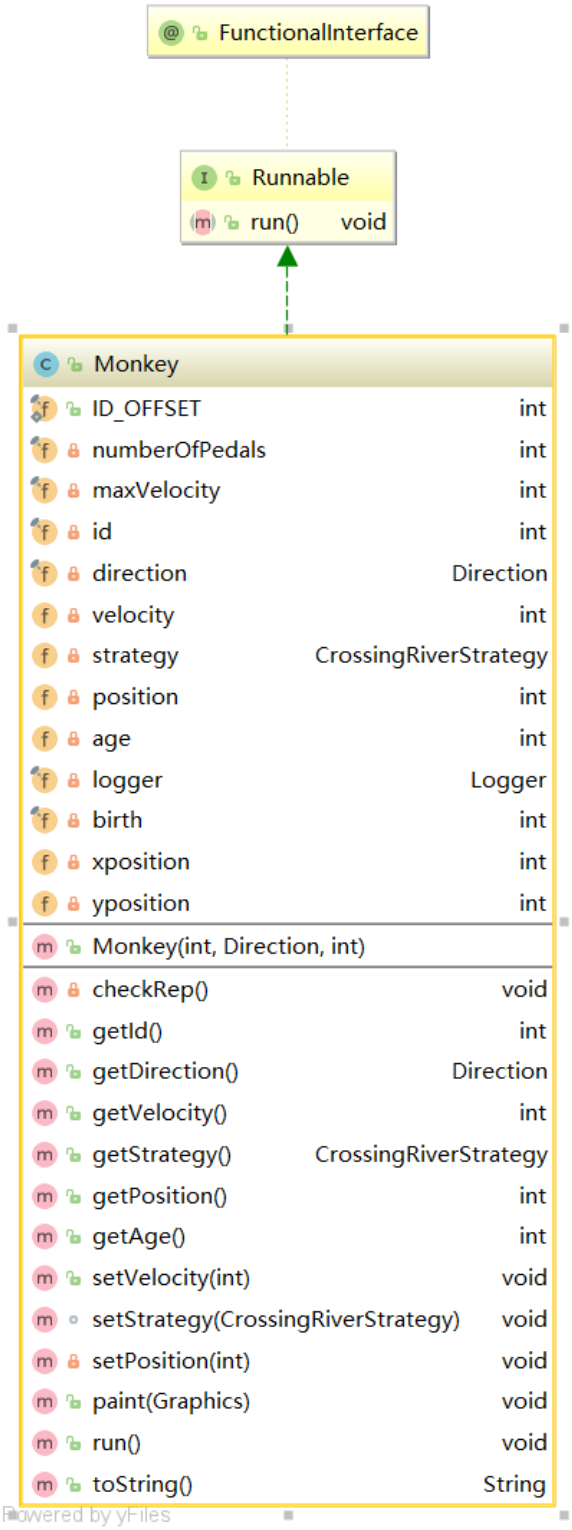
3.1.6 monkey 包

在 monkey 包中所有类的依赖关系如下：

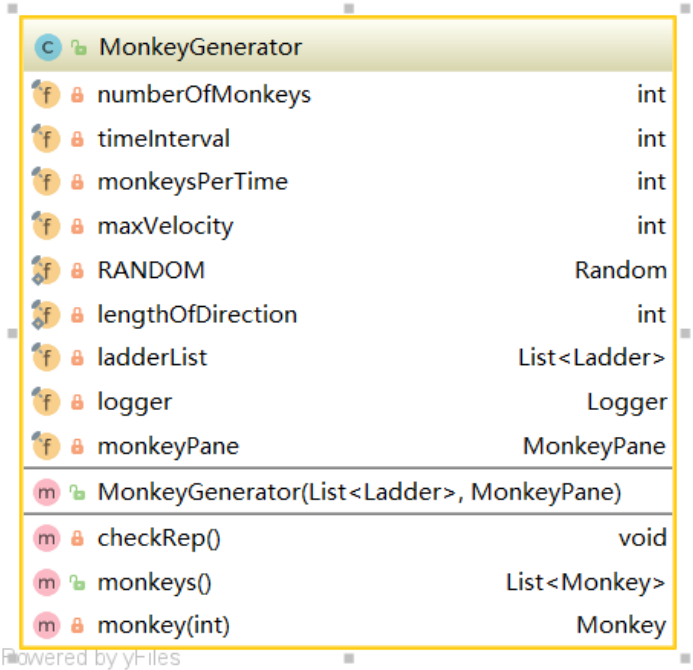


分别再看其余的类的设计如下：
Monkey 类：
其作为主要的 ADT，必须要记录应有的信息，比如 ID、direction 和速度等等。另外对于 ladder 来说其只保留在梯子上面的猴子的引用，具体对于选择

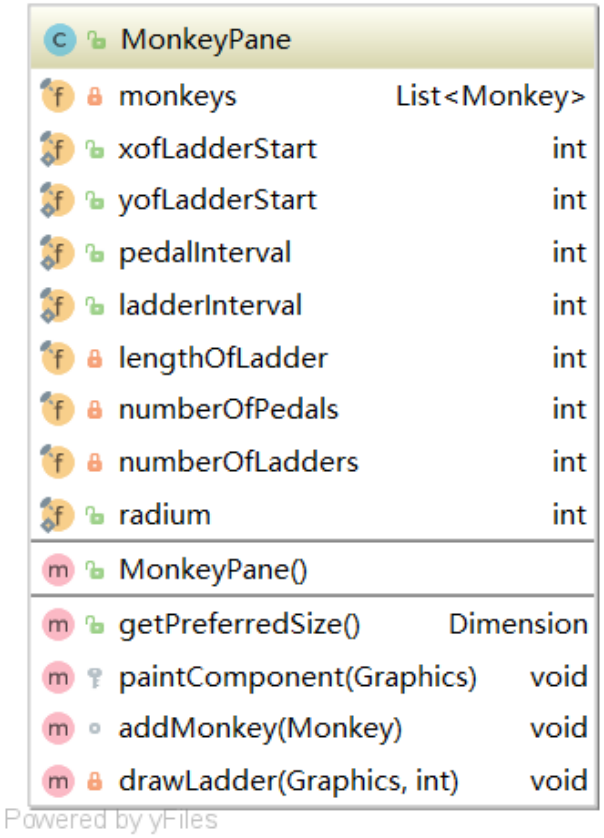
梯子的一些操作还是在 Monkey 类中实现。



MonkeyGenerator 类，类图如下：具体说明见 MonkeyGenerator



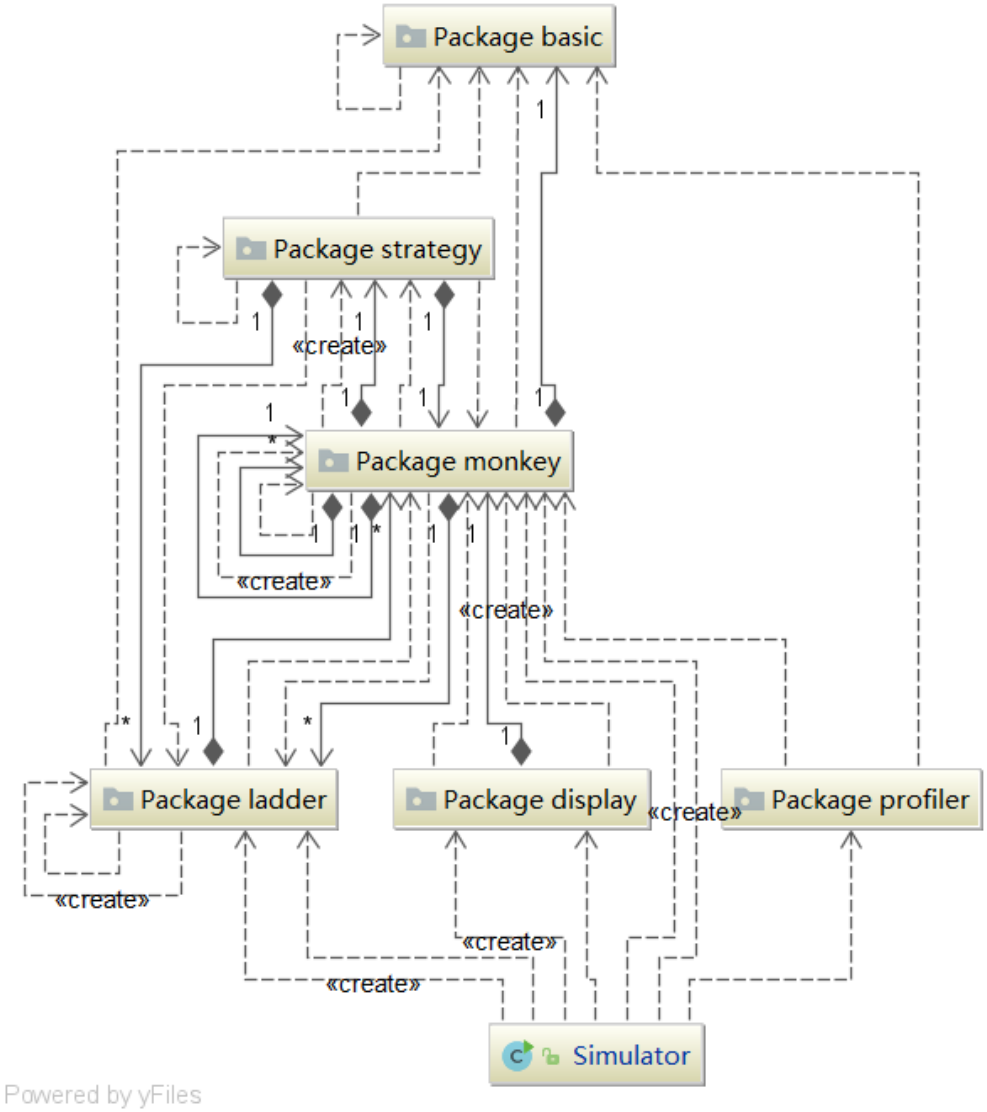
MonkeyPane 类图如下：



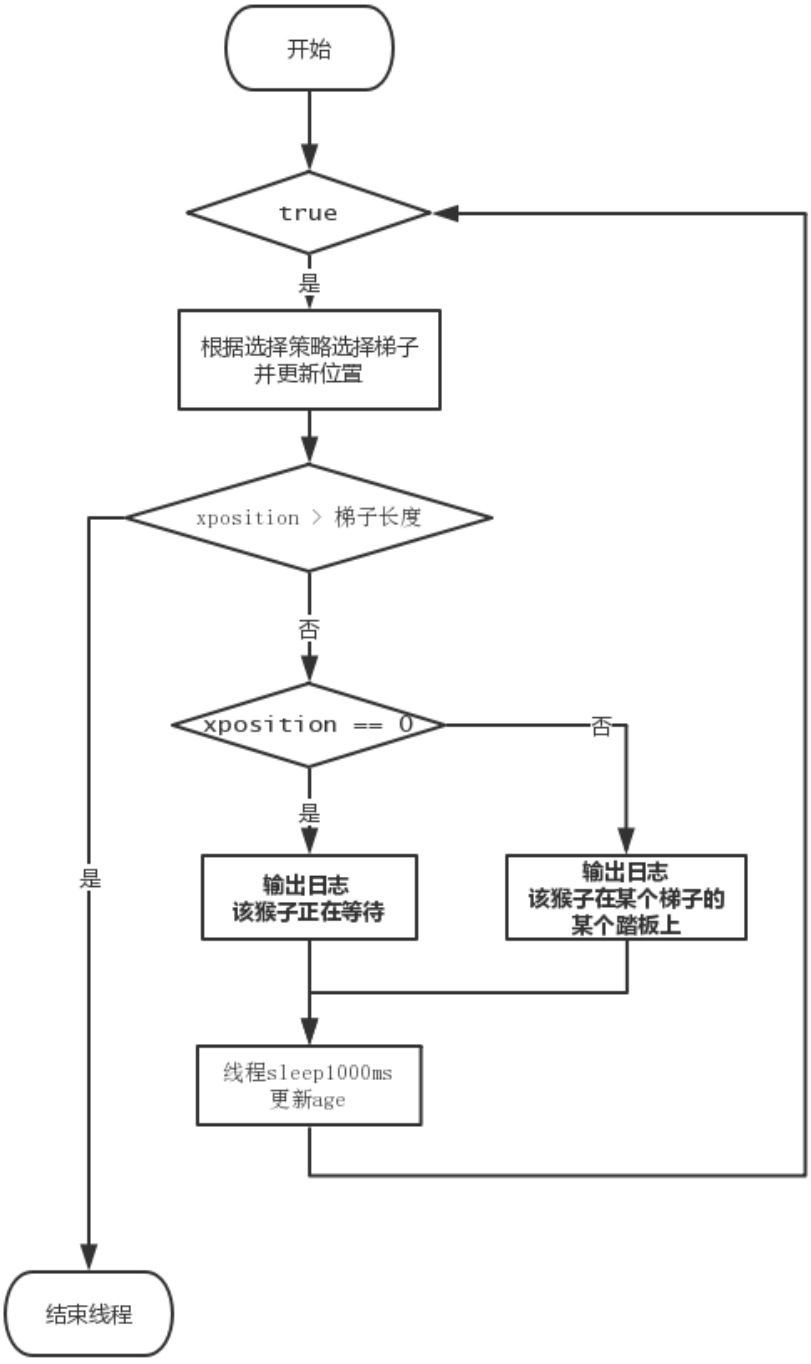
对于 MonkeyPane 来说，主要的功能就是在上面显示所有的梯子和猴子的过程，所以设计的时候需要保证实时可以向里面加入新的猴子并更新猴子的位置。基于这个原因，paintComponent 函数是 synchronized 加锁的。

3.1.7 包之间的依赖关系

由于类的数目比较多，所以直接使用类之间的图来表示关系不够清晰，在这里使用包之间的关系来进行显示。



3.2 Monkey 线程的 run()的执行流程图



此处的 xposition 表示如果猴子在梯子上，那么 xposition 为其所在的踏板编号；如果猴子不在梯子上，那么 xposition 为 0.

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

3.3.1.1 WaitUtilEmptyStrategy 的设计

`WaitUtilEmptyStrategy`，使用这种策略过河的猴子，在选择梯子的时候只会选择没有猴子的梯子。换句话说，如果梯子上面有猴子，那么使用这种策略的猴子会一直在岸边等待，直到出现有某个梯子上面没有猴子，才会选择通过梯子。

易见这种策略，会使得大量的时间在等待，不利于提高吞吐率，在后面的测试中我们也会发现这一点。

3.3.1.2 WaitUtilEmptyStrategy 的实现方案

使用这个策略的猴子，直接遍历所有的梯子，当发现有某个梯子上面没有猴子，将该梯子“占领”，并踏上第 1 个踏板。

3.3.2 策略 2

3.3.2.1 WaitFirstPedalStrategy 的设计

`WaitFirstPedalStrategy`，使用这种策略过河的猴子，在选择梯子的时候会优先选择空梯子（没有猴子在上面的梯子），但是如果这个梯子上面有猴子而且上面的猴子与选择梯子的猴子的过河方向相同并且没有猴子在第 1 个踏板上，这个猴子也认为这个梯子可以选择。

对于这种策略，我们不好直接说明白其优劣，但是对于行进速度比较快的猴子，这种选择方式会导致其速度变慢。

3.3.2.2 WaitFirstPedalStrategy 的实现方案

遍历所有的梯子，如果梯子是空的或者是梯子上面的猴子与待选择梯子的猴子的方向相同并且空出来第一个踏板，那么该猴子选择这个梯子。

3.3.3 策略 3

3.3.3.1 ChooseCrossingFastestStrategy 的设计

`ChooseCrossingFastestStrategy`，使用这种策略过河的猴子，优先选择空梯子；在没有梯子空闲的情况下，一定会遍历所有的梯子，并从中寻找到前进

方向与自己一致，并且最后一个猴子的行进速度最快的梯子。

3.3.3.2 ChooseCrossingFastestStrategy 的实现方案

遍历所有的梯子，如果可以找到空梯子，登上梯子；否则在所有的梯子中寻找在梯子上的最后一个猴子的前进方向与自己一致并且速度最大的一个梯子，登上该梯子。

3.4 “猴子生成器”MonkeyGenerator

对于“猴子生成器”主要的功能就是按照规定的时间间隔和每次产生的猴子个数，产生指定个数的猴子。所以设计的方式就是，能够从 config.properties 中读取配置信息，并且按照配置文件的要求生成猴子，在生成猴子后就将对应于该猴子的线程开启即可。

3.5 如何确保 threadsafe？

保证 Thread-Safe 的方式，由于对于每个猴子，都是单独的线程，所以主要的线程安全保证在于 Ladder 中数据结构的维护。
所以下面主要来说一下如何维护 Ladder 中的 Thread-safe
在 Ladder 中最重要的数据结构就是 BlockingDeque<Monkey>，首先 JDK 保证了其线程安全性，所以在使用的任何其原子操作都是线程安全的，进而防止 Rep exposure，所以在其外面进行了“包装”。
所有针对其的操作均是 synchronized 保证线程安全的。

```
public synchronized boolean empty() {
  return this.queue.isEmpty();
}

public synchronized void put(Monkey monkey) throws InterruptedException {
  this.queue.put(monkey);
}

public synchronized Monkey pollFirst() {
  return this.queue.pollFirst();
}

public synchronized Monkey peekFirst() {
  return this.queue.peekFirst();
}

public synchronized Monkey peekLast() {
  return this.queue.peekLast();
}

public synchronized void add(Monkey monkey) {
  this.queue.addLast(monkey);
}
```

3.6 系统吞吐率和公平性的度量方案

3.6.1 吞吐率

吞吐率的计算方式就是，将第 1 个猴子出生的时候作为起始时间，在最后一个猴子到达对岸作为终止时间，将所用的总时间 T 秒，和通过的猴子总数 N 做比。得到的 $\text{Throughput} = \frac{N}{T}$ 作为表征单位时间内通过的猴子数量。

举例说明：

如果第 1 只猴子出现，此时开始计时。在最后一只猴子（第 10 只猴子）通过梯子到达河对岸，此时的时间为 20s。

那么这个系统的吞吐率就是 $\text{Throughput} = \frac{N}{T} = \frac{10}{20} = 0.5$

3.6.2 公平性

公平性就是指，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。此处注意，我们仅仅区

分的是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，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。
基于上面的思想，我们引出两个猴子间公平性的计算公式如下：

$$\mathrm{F}(A,B)=\begin{cases}1, & if(Y_b-Y_a)\times(Z_b-Z_a)\geq 0\\ -1, & otherwise\end{cases}$$

其中 Y_a 表示 A 出生的时间，Z_a 表示 A 到达河对岸的时间。
然后再将以上的公平性进行组合得到总体的公平性：

$$F=\frac{\sum_{(A,B)\epsilon\theta}F(A,B)}{C_N^2},\theta=\{(A,B)\ |A\neq B,(B,A)\notin\theta\}$$

其中C_N^2表示的是 N 中取 2 的组合数。

3.7 输出方案设计

3.7.1 日志

对于日志的方式，将所有关于 Monkey 对象过河的全过程均记录在 monkey.log 文件中，分别记录等待、过河过程以及到达对岸；对于 GUI 的方式，主要记录的是多个猴子在河上通过梯子的动态过程。

在实验中综合使用了以上两种方式进行输出。

3.7.1.1 等待的记录

采取以下格式进行记录：

```
[INFO ] 2018-06-04 21:34:17,074 [Thread-28] 25007 Monkey90003 stay at L at 24 after 0
```

如上面这条记录，表示的是在 2018-06-04 21:34:17 的时候程序正在运行第 28 个线程，在程序开始 25007ms 后进行的记录，记录的内容是：
Monkey90003，即第 9 批产生的第 3 只猴子，待在 L（左岸）。时刻是第 1 只猴子出生后的第 24 秒，在这只猴子出生后的第 0 秒。

3.7.1.2 行进的记录

采取以下格式进行记录

```
[DEBUG] 2018-06-04 21:34:17,965 [Thread-22] 25898 Monkey70003 come to 8th pedals of 4th ladder R2L at 25 after 7
```

记录的时间是 2018-06-04 21:34:17,965，第 22 个线程，在程序开始运行的第 25898ms 进行的记录，记录的内容是：
Monkey70003，即第 7 批产生的第 3 只猴子，来到了第 4 个梯子的第 8 个踏板上，正在从右岸向左岸进发（R2L），在第一只猴子出生后的第 25 秒，在自己

出生后的第 7 秒。

3.7.1.3 到达的记录

记录的格式如下：

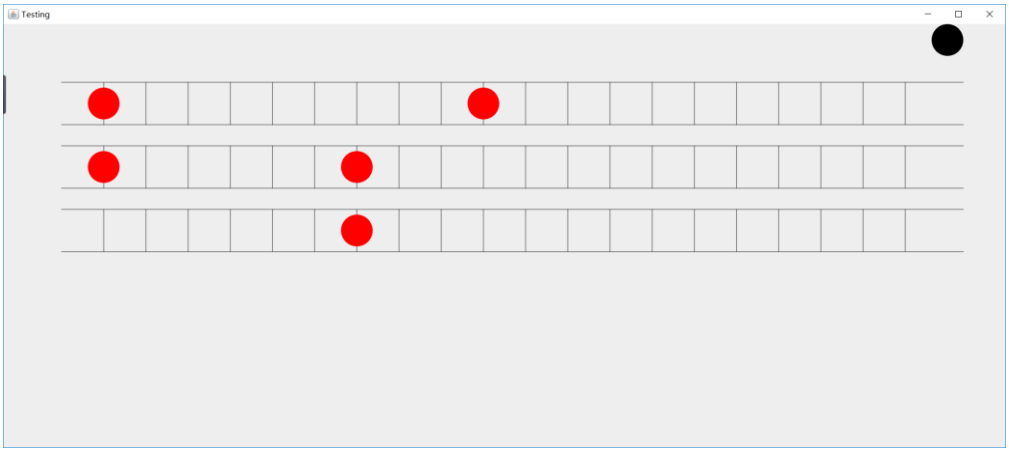
```
[INFO ] 2018-06-04 21:34:20,916 [Thread-20] 28849 Monkey70001 arrived at
3th ladder L at 28 after 10
```

记录的时间是 2018-06-04 21:34:20,916，第 20 个线程，在程序开始运行的第 28849ms 进行的记录，记录的内容是：
Monkey70001，即第 7 批产生的第 1 只猴子，通过第 3 个梯子到达河的左岸，此时的时间是第 1 只猴子出生后 28 秒，自己出生后第 10 秒。

3.7.2 GUI

GUI 的方式主要是可以形象化的表示每个时刻猴子在梯子上的位置，记录的方式如下，此处使用的测试参数为

```
h = 20, t = 3, N = 10, k = 3, MV = 5, n = 3
```



在图中红色的实心圆表示从左岸去右岸的猴子，黑色的实心圆表示从右侧去左侧的猴子。可以看到，在截图的瞬间，从左岸出发的猴子在梯子上面行进，分别到达了第 1 个踏板等等；从右岸出发的猴子，由于没有梯子可以使用（避免死锁），因此在河的右岸等待。

3.8 猴子过河模拟器 v1

3.8.1 参数如何初始化

由于各种参数的面向的类不同，此处使用 Apache 开源的配置文件读取进行参数的初始化。
在其中使用一个类 ConfigReader，当所有的类需要读取默认的参数值时，从

Config.properties 中直接读取所需要的参数。其中对于所有可能的参数，使用 enum 进行组织，Params 如下表示：

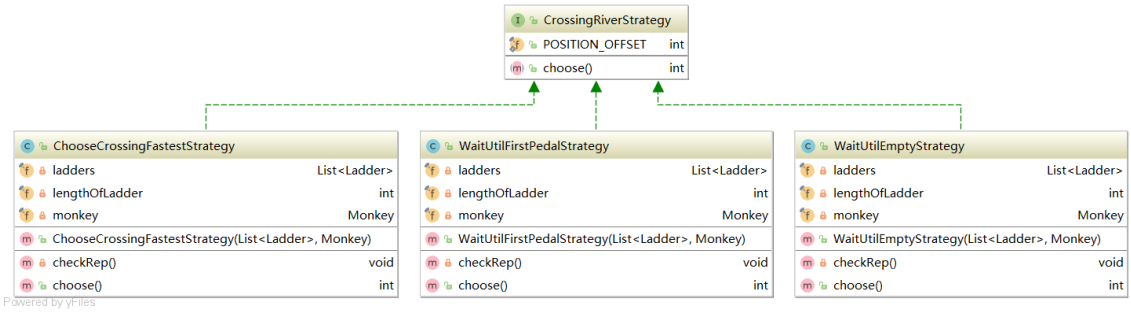
```
/**
 * Params used in Configuration.
 */
public enum Params {
  n, h, t, N, k, MV
  // n is total number of ladders
  // h is number of pedals in a ladder
  // t is the interval between every two generation of monkeys
  // N is total number of monkeys
  // k is number of monkeys in on generation
  // mv is the max value of velocity
}
```

3.8.2 使用 Strategy 模式为每只猴子随机选择决策策略

在 MonkeyGenerator 中，使用随机数为 Monkey 选择决策策略。

```
CrossingRiverStrategy strategy;
int chosen = RANDOM.nextInt(3);
switch (chosen) {
  case 0:
    strategy = new WaitUtilFirstPedalStrategy(ladderList, monkey);
    break;
  case 1:
    strategy = new WaitUtilEmptyStrategy(ladderList, monkey);
    break;
  case 2:
    strategy = new ChooseCrossingFastestStrategy(ladderList, monkey);
    break;
  default:
    assert false;
}
```

对于 Strategy 的继承关系，见下面的类图：



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。**以下所有的测试结果，都在 doc/time.xlsx 进行详细的记录。由于对于所有猴子的速度是随机产生的，为了避免随机值对于结果的影响，所以以下所有测试都进行 10 次取平均值。**

3.9.1 对比分析：固定其他参数，选择不同的决策策略

`h = 20, t = 3, N = 10, k = 3, MV = 5, n = 3`
测试的结果如下：

	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性
测试次数\	**WaitUtilFirstPedalStra**		**WaitUtilEmptyStrategy**		**ChooseCrossingFastestS**	
1	0.384615385	0.955555556	0.416666667	0.777777778	0.344827586	1
2	0.344827586	0.555555556	0.322580645	0.955555556	0.434782609	0.866666667
3	0.344827586	0.822222222	0.384615385	0.866666667	0.344827586	0.955555556
4	0.384615385	0.6	0.294117647	0.866666667	0.344827586	0.822222222
5	0.344827586	0.866666667	0.37037037	0.911111111	0.285714286	0.733333333
6	0.434782609	0.555555556	0.222222222	0.822222222	0.344827586	0.955555556
7	0.384615385	0.822222222	0.227272727	0.777777778	0.344827586	0.688888889
8	0.344827586	0.777777778	0.227272727	0.866666667	0.344827586	0.911111111
9	0.344827586	0.822222222	0.333333333	0.733333333	0.27027027	0.822222222
10	0.23255814	0.866666667	0.285714286	0.555555556	0.344827586	0.866666667
平均值	**0.354532483**	0.764444444	**0.308416601**	0.813333333	**0.340456027**	0.862222222

3.9.2 对比分析：变化某个参数，固定其他参数

1、测试 n（梯子数目）的影响
均使用 ChooseCrossingFastestStrategy
相同的参数设置如下
`h = 20, t = 3, N = 10, k = 3, MV = 5`
测试的结果如下：

n	1		2		3		4		5	
测试次数	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性
1	0.232558	0.555556	0.27027	0.955556	0.384615	0.511111	0.37037	0.688889	0.344828	0.777778
2	0.322581	0.555556	0.344828	1	0.344828	0.911111	0.625	0.955556	0.384615	0.688889
3	0.192308	0.377778	0.208333	0.511111	0.344828	1	0.384615	0.377778	0.384615	0.688889
4	0.192308	0.422222	0.30303	0.511111	0.344828	0.866667	0.37037	0.777778	0.344828	0.733333
5	0.25641	0.6	0.294118	0.911111	0.384615	0.733333	0.384615	0.911111	0.344828	0.866667
6	0.192308	0.466667	0.384615	0.288889	0.357143	0.822222	0.344828	1	0.434783	0.644444
7	0.222222	0.555556	0.344828	0.955556	0.344828	1	0.3125	0.688889	0.384615	0.733333
8	0.192308	0.422222	0.384615	0.6	0.384615	0.822222	0.344828	0.866667	0.384615	0.777778
9	0.196078	0.422222	0.263158	0.422222	0.625	1	0.5	0.688889	0.526316	1
10	0.243902	0.6	0.344828	0.733333	0.526316	0.955556	0.384615	0.511111	0.5	0.333333
平均值	**0.2243**	0.497778	**0.31426**	0.688889	**0.40416**	0.862222	**0.40217**	0.746667	**0.4034**	0.724444

2、测试 t（产生猴子的时间间隔）的影响

均使用 ChooseCrossingFastestStrategy

相同的参数设置如下

```
h = 20, n = 3, N = 10, k = 3, MV = 5
```

测试的结果如下：

t	1		2		3		4		5	
测试次数	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性
1	0.434783	0.688889	0.833333	0.688889	0.344828	0.866667	0.3125	0.955556	0.285714	0.955556
2	0.4	0.911111	0.384615	1	0.344828	0.733333	0.3125	0.688889	0.47619	0.866667
3	0.769231	0.777778	0.384615	0.911111	0.344828	0.822222	0.3125	1	0.285714	0.822222
4	0.416667	0.733333	0.625	1	0.357143	0.6	0.344828	0.644444	0.4	0.866667
5	0.454545	0.6	0.384615	0.911111	0.30303	1	0.3125	0.866667	0.333333	0.688889
6	0.416667	0.777778	0.384615	0.822222	0.434783	0.466667	0.357143	0.555556	0.333333	0.822222
7	0.714286	0.955556	0.416667	0.555556	0.344828	0.644444	0.3125	0.955556	0.232558	0.911111
8	0.434783	0.911111	0.333333	0.822222	0.344828	0.955556	0.526316	0.955556	0.285714	0.955556
9	0.454545	0.6	0.384615	0.955556	0.384615	0.688889	0.3125	0.911111	0.322581	0.911111
10	0.4	0.644444	0.416667	0.688889	0.384615	0.777778	0.3125	1	0.285714	0.955556
平均值	**0.48955**	0.76	**0.45481**	0.835556	**0.35883**	0.755556	**0.34158**	0.853333	**0.32409**	0.875556

3、测试 k（单次产生猴子的数目）的影响

均使用 ChooseCrossingFastestStrategy

相同的参数设置如下

```
h = 20, n = 3, N = 10, t = 3, MV = 5
```

k	1		2		3	
测试次数	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性
1	0.227273	0.6	0.384615	0.688889	0.384615	0.466667
2	0.212766	0.733333	0.3125	0.866667	0.526316	1
3	0.263158	0.555556	0.344828	0.511111	0.37037	0.733333
4	0.212766	0.822222	0.3125	0.6	0.625	0.955556
5	0.212766	0.777778	0.3125	1	0.30303	0.955556
6	0.212766	0.733333	0.384615	0.822222	0.344828	1
7	0.227273	0.555556	0.454545	1	0.344828	0.822222
8	0.212766	0.866667	0.30303	0.777778	0.344828	1
9	0.227273	0.911111	0.30303	0.866667	0.344828	0.822222
10	0.227273	0.733333	0.3125	0.688889	0.344828	0.688889
平均值	**0.22361**	0.728889	**0.34247**	0.782222	**0.39335**	0.844444

4、测试 N（猴子数目）的影响

均使用 ChooseCrossingFastestStrategy

相同的参数设置如下

```
h = 20, n = 3, t = 3, k = 3, MV = 5
```

测试的结果如下：

N	20		40		60		80		100	
测试次数	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性	吞吐率	公平性
1	0.408163	0.726316	0.677966	0.920513	0.779221	0.963842	0.808081	0.993671	0.840336	0.993939
2	0.526316	0.884211	0.677966	0.961538	0.769231	0.99435	0.816327	0.964557	0.840336	0.991919
3	0.526316	0.768421	0.677966	0.984615	0.779221	0.922034	0.808081	0.833544	0.840336	0.976566
4	0.512821	0.631579	0.547945	0.658974	0.769231	0.964972	0.816327	0.994304	0.840336	0.986667
5	0.526316	0.957895	0.701754	0.861538	0.779221	0.99548	0.816327	0.987342	0.840336	0.995556
6	0.526316	0.957895	0.677966	0.933333	0.759494	0.983051	0.816327	0.993038	0.840336	0.914747
7	0.425532	0.621053	0.677966	0.994872	0.769231	0.918644	0.816327	0.984177	0.840336	0.985859
8	0.416667	0.673684	0.677966	0.982051	0.769231	0.970621	0.816327	0.990506	0.840336	0.998384
9	0.526316	0.768421	0.677966	0.905128	0.779221	0.955932	0.816327	0.994304	0.840336	0.905051
10	0.555556	0.821053	0.677966	0.979487	0.769231	0.754802	0.816327	0.931646	0.840336	0.994343
平均值	**0.49503**	0.781053	**0.66734**	0.918205	**0.77225**	0.942373	**0.81468**	0.966709	**0.84034**	0.974303

3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？

3.9.3.1 吞吐率与决策策略的相关性

根据 3.9.1 节对于决策策略的测试可以看到，在固定参数的时候，`WaitUtilEmpty`（等待直到有空梯子为止）的策略吞吐率最低，只有 `0.30` 左右。而其余的两个策略相差不大，分别是 `0.34` 和 `0.35` 左右。虽然已经经过了 `10` 次的实验来减少随机速度对于最终结果的影响，但是没有办法消除。

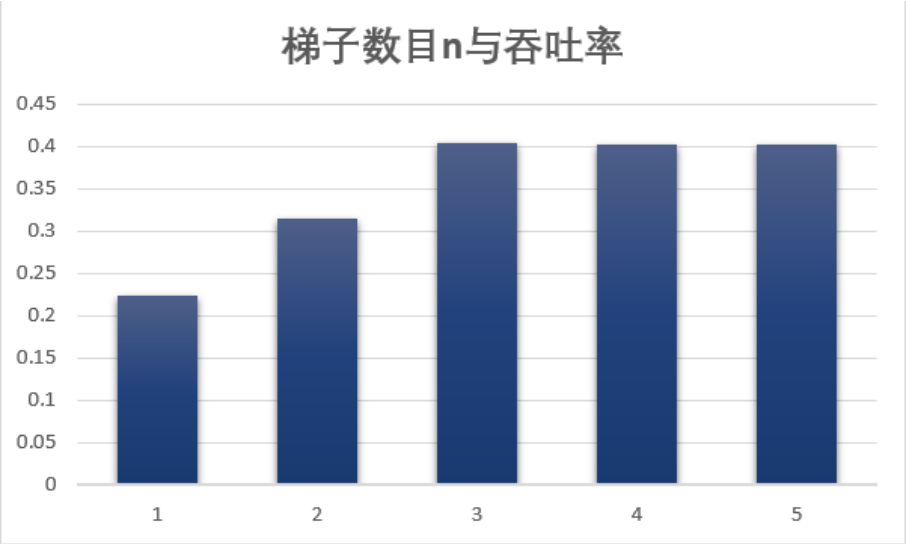
综合来看，**吞吐率与决策策略直接相关，选择 `WaitUtilEmpty` 策略的吞吐率较其余两个策略偏低。**

3.9.3.2 吞吐率与各参数之间的相关性

1、与梯子的数目 n

在通过固定其余参数来看与 n 的相关性，可以看到**随着 n 的增大，吞吐率**

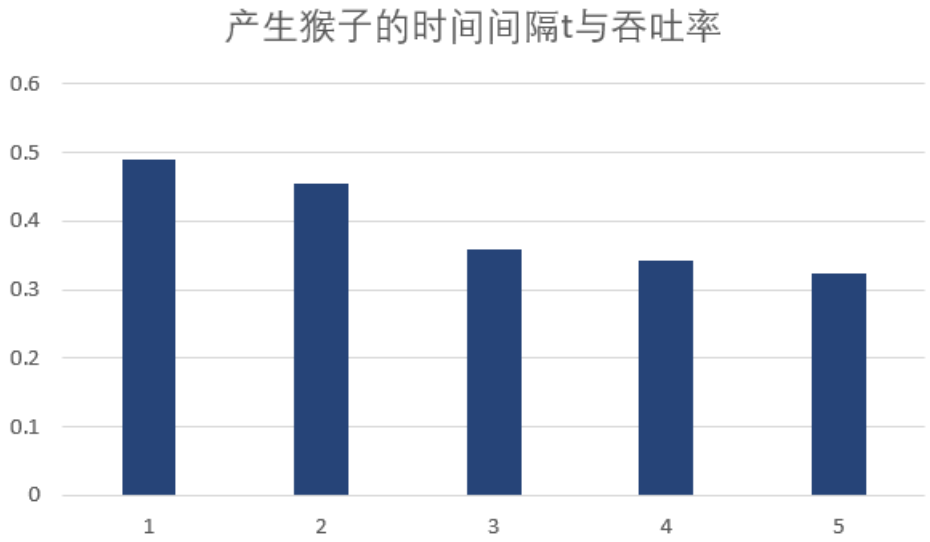
在逐渐提升，并最终趋于平稳。



可以分析一下，在猴子数目较少（N=10）的情况下，并且猴子出生的间隔也比较大的前提下，梯子数目多，因此不会有过多的猴子在同一个梯子上的情况，进而不能够无限制的提高并行的效率。

2、与产生猴子的时间间隔 t

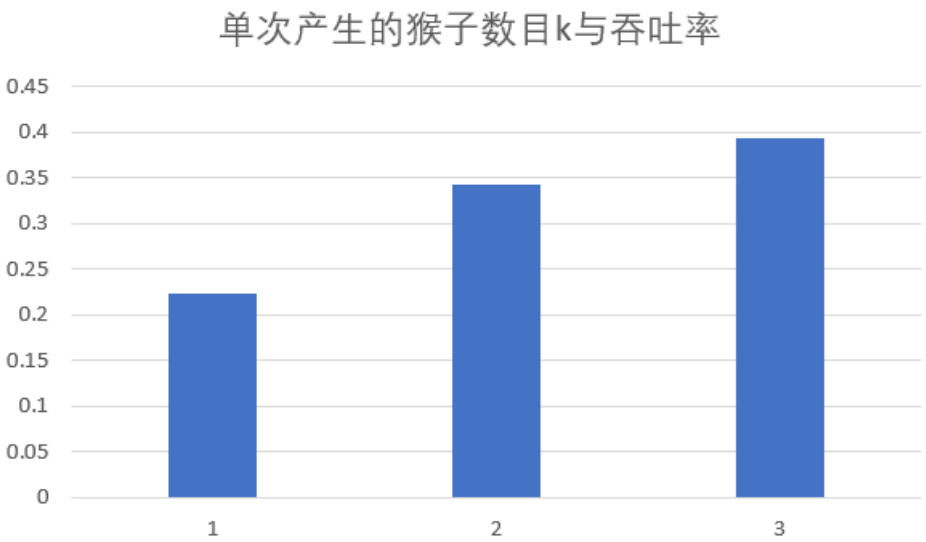
在固定其余参数，来看与 t 的相关性，可以看到**随着 t 的增大，吞吐率逐渐下降。**



可以分析一下，在猴子数目比较少（N=10）的情况下，t 越大，意味着在下一批猴子出生前，所有的猴子已经通过了梯子，因而没有并行效率提高的体现，进而导致吞吐率的下降。

3、与单次产生的猴子数目 k

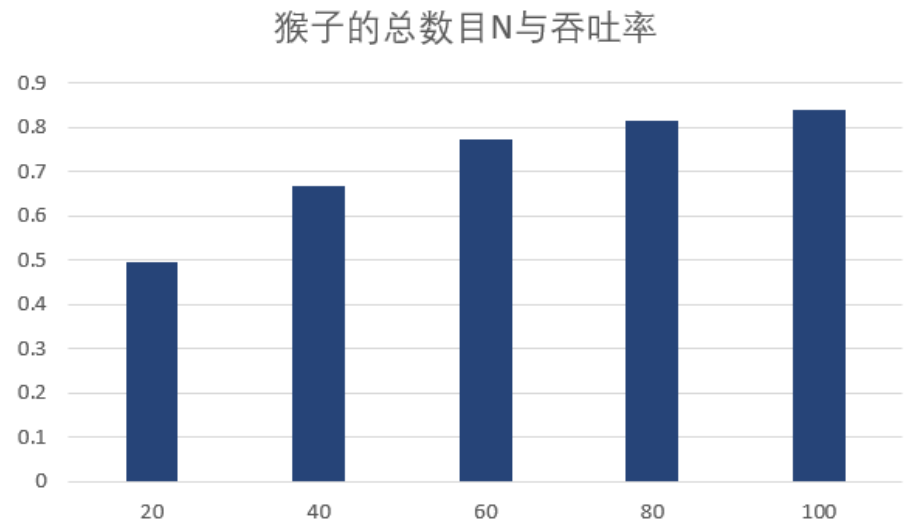
在固定其余参数，来看与 k 的相关性，可以看到**随着 k 的增大，吞吐率逐渐提升。**



可以分析一下，k 与 t 恰好是一对相反的影响因素。k 越大，意味着我有更多的猴子在同一时刻出生，那么就有可能出现在同一个梯子上有多个猴子一同过河的情况，那么理所应当的可以提高并行的效率。

4、与猴子的总数目 N

在固定其余参数的情况下，来看与 N 的相关性，可以看到**随着 N 的增大，吞吐率逐渐提升。**



可以想到，在猴子数目越多的情况下，使用策略如果不是等待直到有空梯子的策略，那么在同一时刻会有多个猴子在梯子上，并且随机的速度对于吞吐率的影响要减小，而是与第一次出现并踏上梯子的速度为 1 猴子的时间相关。

3.9.4 压力测试结果与分析

3.9.4.1 使用多猴子少梯子进行测试

测试使用的参数列表如下：

n	3

h	20
t	3
N	3000
k	3
MV	5

最终测试的 log 文件详见./log/monkey-压力测试 1.log
最终的测试结果文件详见./log/record-压力测试 1.txt
得到的最后的吞吐率和公平性如下：

```
Throughput = 0.9940357852882704
Justice = 0.9998959653
```

3.9.4.2 使用相差较大的速度进行压力测试

使用下面的参数进行测试

n	3
h	20
t	3
N	100
k	3
MV	5

其中，**随机产生的猴子速度只有两种 1 或者 5，**最终得到的完整测试日志文件见./log/monkey-压力测试 2.log，最终的完整的测试结果文件见./log/record-压力测试 2.txt。
测试得到的吞吐率和公平性见下：

```
Throughput = 0.8403361344537815
Justice = 1.0000000000
```

4 实验进度记录

请尽可能详细的记录你的进度情况。

日期	时间段	计划任务	实际完成情况
2018-5-28	13:45-17:00	写配置文件 设计类	完成
2018-5-29		完成 monkey 的初步设计	完成
2018-5-30	2018-6-3	配置 log4j 和写 GUI	延期
2018-6-3	22:00 前	GUI 写完 重构代码	完成
2018-6-4	23:00 前	报告	延期至次日 16 点完成

5 实验过程中遇到的困难与解决途径

log4j 记录日志的功能在输出到文件的时候会将部分日志文本删除，但是如果使用 Append=true 的方式输出，就不会出现类似的问题。
解决的方式：

由于在 Stack Overflow 和 log4j 官方的讨论区都没有给出相应的解决方法，所以只好通过命令行进行输出。然后通过重定向将命令行输出的文本输出到文件中。

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。

（1）单线程程序的调试是容易复现错误的，而对于多线程来说，很多错误不能复现。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，这种性能的提升更多的来自于过个任务执行时的优势。

（2）使用线程安全的数据结构和 synchronized 加锁来实现。

优先保证线程安全，然后再考虑性能问题。

（3）遇到了线程不安全的问题，在多个猴子同时访问 ladder 的时候，将占用的信息没有及时的写回。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。

修改后与预期相同。

（4）是所有实验中感觉最有趣的实验，deadline 比较合适。

（5）最后一个实验结束了，感觉做下来 6 个实验，确实对自己编程方面的能力有不少的提升。