

**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 | 喻思成 |
| 学号 | 1170300229 |
| 班号 | 1736101 |
| 电子邮件 | [928623089@qq.com](mailto:928623089@qq.com) |
| 手机号码 | 15573071218 |

**目录**

[1 实验目标概述 1](#_Toc11079201)

[2 实验环境配置 1](#_Toc11079202)

[3 实验过程 2](#_Toc11079203)

[3.1 ADT设计方案 2](#_Toc11079204)

[3.1.1 架构 2](#_Toc11079205)

[3.1.2 Ladder 4](#_Toc11079206)

[3.1.3 Strategy 6](#_Toc11079207)

[3.1.4 Monkey & MonkeyGenerator 6](#_Toc11079208)

[3.1.5 RiverPanel 8](#_Toc11079209)

[3.1.6 Applications 11](#_Toc11079210)

[3.2 Monkey线程的run()的执行流程图 11](#_Toc11079211)

[3.3 至少两种“梯子选择”策略的设计与实现方案 13](#_Toc11079212)

[3.3.1 策略1 Wait Strategy 13](#_Toc11079213)

[3.3.2 策略2 One Side Strategy 14](#_Toc11079214)

[3.3.3 策略3 Velocity Strategy 15](#_Toc11079215)

[3.3.4 策略4 Partition Strategy 15](#_Toc11079216)

[3.3.5 策略5 Extreme Strategy 17](#_Toc11079217)

[3.3.6 策略6 Small Strategy 18](#_Toc11079218)

[3.3.7 策略7 Large Strategy 19](#_Toc11079219)

[3.4 “猴子生成器”MonkeyGenerator 20](#_Toc11079220)

[3.5 如何确保threadsafe？ 21](#_Toc11079221)

[3.6 系统吞吐率和公平性的度量方案 23](#_Toc11079222)

[3.7 输出方案设计 24](#_Toc11079223)

[3.8 猴子过河模拟器v1 26](#_Toc11079224)

[3.8.1 参数如何初始化 26](#_Toc11079225)

[3.8.2 使用Strategy模式为每只猴子选择决策策略 27](#_Toc11079226)

[3.9 猴子过河模拟器v2 28](#_Toc11079227)

[3.9.1 对比分析：固定其他参数，选择不同的决策策略 29](#_Toc11079228)

[3.9.2 对比分析：变化某个参数，固定其他参数 30](#_Toc11079229)

[3.9.3 分析：吞吐率是否与各参数/决策策略有相关性？ 31](#_Toc11079230)

[3.9.4 压力测试结果与分析 32](#_Toc11079231)

[3.10 猴子过河模拟器v3 34](#_Toc11079232)

[4 实验进度记录 35](#_Toc11079233)

[5 实验过程中遇到的困难与解决途径 37](#_Toc11079234)

[6 实验过程中收获的经验、教训、感想 37](#_Toc11079235)

[6.1 实验过程中收获的经验和教训 37](#_Toc11079236)

[6.2 针对以下方面的感受 37](#_Toc11079237)

# 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

# 实验环境配置

1.log4j的安装和使用，因为log4f 2只能使用xml文件配置不如像log4j 1.x版本的java特性文件配置来的方便，我使用了log4j 1.2.17，下载网址为<http://logging.apache.org/log4j/1.2/download.html>，同时编写日志的配置文件log4j.properties，配置appender属性和log的输入输出流

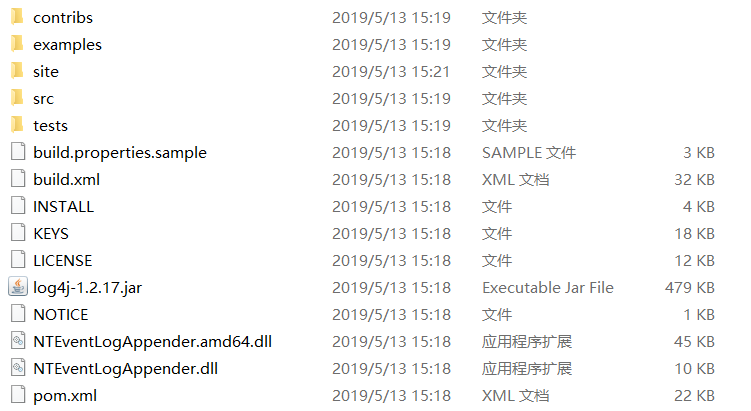


图2.1 log4j 1.2.17 的内容

最后直接将其中的log4j-1.2.17.jar加入到项目的build-path中即可



图2.2 项目中加入的log4j-1.2.17

已将log4j-1.2.17.jar放入到项目的lib文件夹中

在这里给出你的GitHub Lab6仓库的URL地址（Lab6-学号）。

Lab6-1170300229: <https://github.com/ComputerScienceHIT/Lab6-1170300229>

# 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！）。

**使用方法**

**1.如果根据外部配置文件正常启动模拟器见3.8.1节**

**2.如果根据外部配置文件启动吞吐量竞赛模拟器见3.10节**

## 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；

（可选）以类图形式给出多个类之间的关系。

### 架构

1.首先分析需求，实验总体要求如下

* + 建模猴子，作为单独的线程运行，能够决策梯子并移动
  + 建模整个背景，包括GUI，两个出生地和生成器，计算吞吐量和公平性
  + 建模梯子，能让在其上的猴子按特殊规律行动
  + 使用Strategy模式来使决策多样化。
  + 结合上述部件成为一个总体的模拟器

可以看到这些要求之间有明显的依赖关系，如猴子的行为依赖Strategy的选择。我们可以按拓扑顺序逐步构建整个项目。

将梯子设计成mutable的类，每个梯子记录了当前梯子上猴子的状态，能够模拟猴子的移动或者是加入猴子，并且提供一系列API来查询梯子的相关信息。实验中所有的梯子都存入一个List，被各ADT共享数据。而构造整个场景的ADT，因为选择的是动画展示过河过程，因此整体的模型被整合进可视化的RiverPanel。Strategy是一个接口，有很多实现的具体Strategy类，这些类是根据共享的梯子的数据来对进行决策。Monkey类实现Runable接口，拥有id，初始位置和速度三个属性，同时还有成员变量Strategy，即具体的决策，同时还有RiverPanel，每次Monkey选择并加入到梯子之后或者是移动之后都会通知RiverPanel，RiverPanel接到通知之后就会用动画展现这个过程，实现的run方法中能自主决策选择梯子并每隔一秒通知RiverPanel进行移动。从而实现了多线程。猴子生成器则根据参数随机生成猴子并启动对应的线程。而Application则整合了猴子生成器，RiverPanel和梯子的List，同时可以解析文件来配置参数，完成整个模拟。

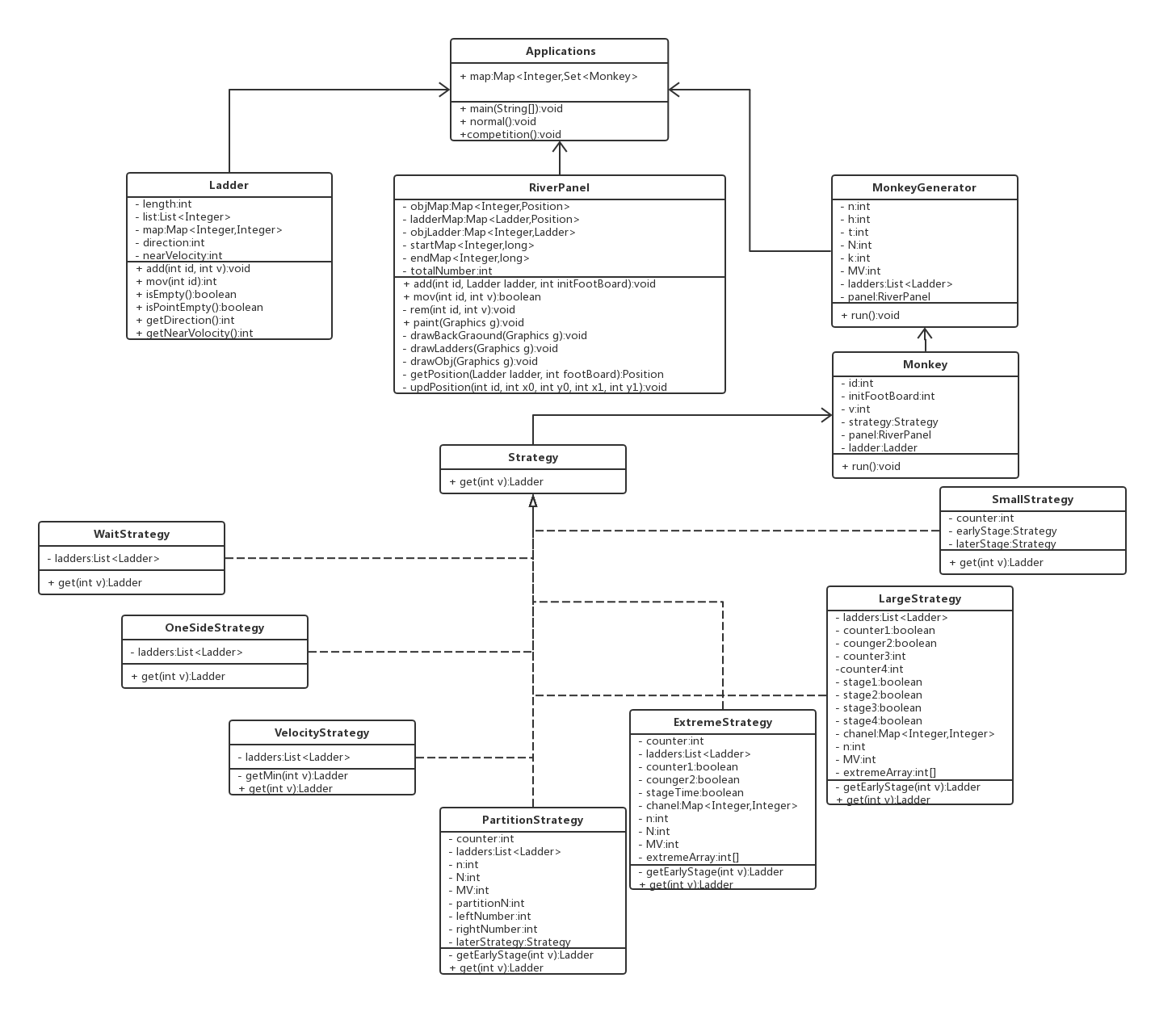


图3.1 UML图

可以看到Applications下包含三个大的分支。具体的细节将在下面描述。而这三个大的分支之间由前面描述过的由Ladder组成的List联系起来，共享数据并更新系统状态。

### Ladder

1.Ladder是对梯子的抽象，是mutable的类，他能记录当前梯子上所有猴子的位置和有关当前梯子的一些状态信息。同时能根据当前梯子的状态更新猴子的位置或是添加一只猴子到梯子上。下面介绍Ladder的方法。

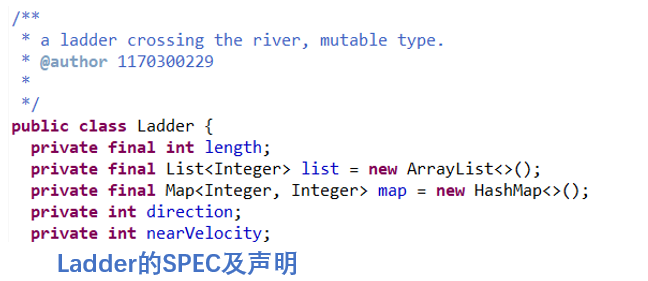


图3.2 Ladder的SPEC及声明

2.add，next方法，add方法用于将一个猴子加入到当前的梯子中，需要指定猴子的id和初始的位置和初始速度（刻在梯子上）。Ladder根据初始的位置可以判断当前梯子上的猴子的方向并记录，同时记录最近加入到梯子的猴子的速度。

next方法则是指定当前梯子上一个猴子的id，根据当前梯子的状态来更新移动之后的位置。根据实验的要求，最多移动的步数为速度或是前面一个猴子的后面的位置。最后next方法返回一个int类型的结果，表明猴子移动后的在梯子上的位置，如果位置已经大于踏板的数量或是小于1，说明猴子移动后已经到达了对岸。

需要说明的是，根据实验要求，不能采用上帝视角，即每只猴子都是根据当前的状态做决策，因此当一只猴子加入到梯子或是在梯子上移动时，其他猴子不能修改当前梯子的状态，因此add方法和next方法都是synchronized的。

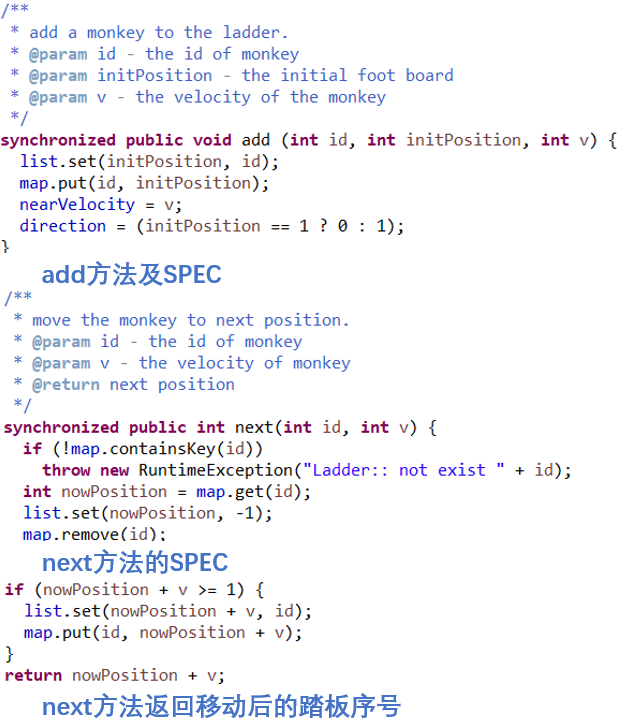


图3.3 add方法和next方法的部分代码

3.其他方法，Ladder还有判断当前梯子是否为空的isEmpty方法，判断当前梯子的两个端点是否为空的isPointEmpty方法，获得当前梯子的方向的方法getDirection，和获得最近上梯子的猴子的速度的方法getNearVelocity。需要说明的是，由于这些方法都涉及对梯子的状态的访问，因此都是synchronized的。更多具体的细节详见代码。

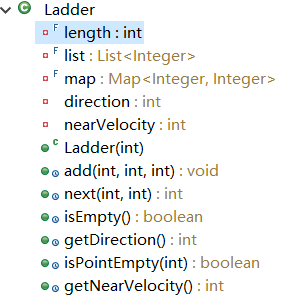


图3.4 Ladder的方法

### Strategy

1.Strategy是一个接口，由于实现Strategy设计模式的接口。用于帮助一个未选择梯子的猴子根据当前所有梯子的状态选择一个梯子。Strategy的具体的实现类则是不同的策略来选择梯子。Strategy的实现类在构造的时候应该以一个Ladder的List作为参数传入，即共享数据的梯子组，并在今后的决策过程以这个Ladder的List作为依据来决策。Strategy只有一个get方法，get方法通过输入猴子的速度来决策当前应该选择的梯子。

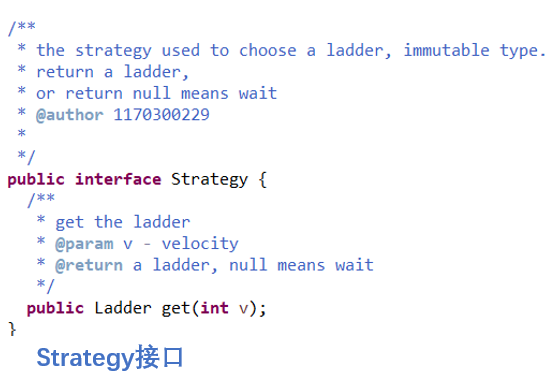


图3.5 Strategy接口的声明，SPEC及方法

可以发现一个性质，所有的猴子在决策的时候，只依赖当前所有梯子的状态和自身的速度，与是哪个猴子无关，因此实际上所有的猴子可以只共享一个Strategy类，即所有的Monkey类在构造的时候传入的是同一个Strategy对象。这样将有助于通过锁住Strategy对象来保证thread safe。

Strategy具体的实现类的介绍将在3.3节详细描述。

### Monkey & MonkeyGenerator

1.Monkey类，Monkey类实现了Runnable接口，可以作为Thread的参数并被开启。Monkey类中的成员变量有id，即猴子的ID，initFootBoard，即猴子的初始踏板（实际上这个就是猴子的方向的另一种记录的方法），v，即猴子的id。同时还有strategy，即猴子的选择Ladder时所用的策略，ladder，如果为null，说明猴子还在等待，否则就是猴子选择的梯子。panel，是一个RiverPanel类，每次猴子的位置发生了变化，猴子会通知panel来可视化展示这个过程。



图3.6 Monkey的SPEC及声明

Monkey只有一个方法，即Runnable的run方法，每sleep一秒之后，首先判断当前ladder是否为null，如果为null则调用strategy的get方法来获得ladder，如果返回的依然是null则保持等待，继续sleep一秒，如果不为null，则将自己加入到ladder中，并通知panel。如果当前的ladder不为null，说明已经在梯子上，则直接通知panel进行下一步移动，panel会返回一个boolean类型的变量，如果为true，说明猴子依然在梯子上，如果为false，说明猴子已经到达对岸，则直接break结束当前的线程。



图3.7 monkey的run方法

2.MonkeyGenerator类，这个类通过构造时传入一系列的参数来构造一个猴子生成器。传入的参数有h，踏板的数量，t，生成时间的间隔，N，总共生成猴子的数量，k，每次生成猴子的个数，MV，最大速度上限，ladders，所有Ladder组成的List，panel，用于可视化的RiverPanel。

MonkeyGenerator的run方法会根据当前的参数来选择合适的Strategy，具体的选择策略将在3.3节描述。然后每隔t秒生成k个猴子并启动线程。其中猴子的速度和方向服从均匀分布。最后一秒生成的猴子数为N%k个。生成完N个猴子之后，run方法结束。



图3.8 MonkeyGenerator的部分代码

### RiverPanel

1.RiverPanel类，负责动画展示过河过程。内部构建了整个场景的二维平面模型。同时有辅助类Position。每次由单独的Monkey线程通知后，会根据变化调用相应的方法来更新猴子的位置并动态展示出来。

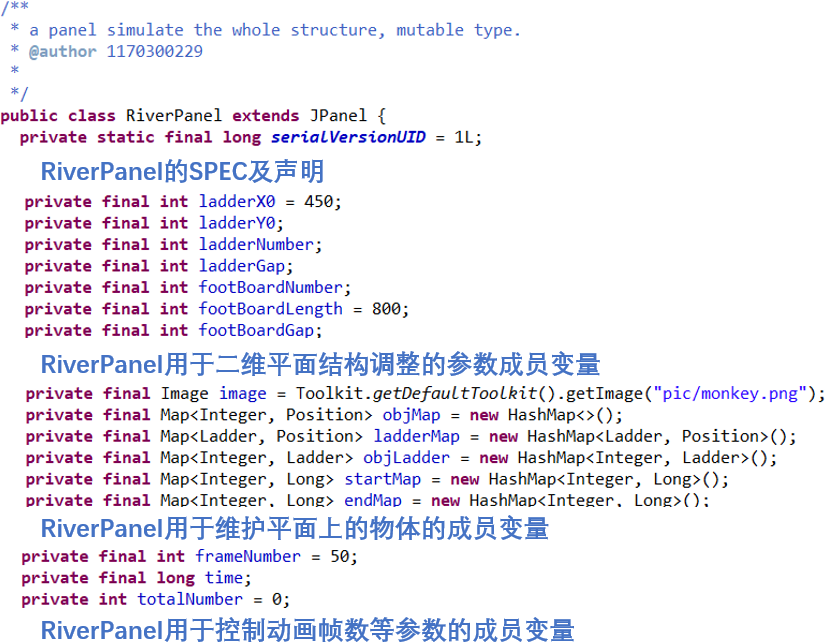


图3.9 RiverPanel的成员变量

2.piant方法，RiverPanel的paint方法分为三个部分，paintBackGround，画出背景色，两个出生点，实时更新吞吐量，以及模拟过程结束后计算并展示公平性。drawLadders用于画出所有的Ladder，根据成员变量中有关二维平面结构的参数和梯子的个数和踏板的个数来自动调整并画出梯子。drawObj，画出所有的猴子，通过记录猴子位置的成员变量objMap来画。



图3.10 RiverPanel的paint方法调用的方法的部分代码

通过上面的描述可知，整个模拟器在运行的时候，实际上我们只需用改变objMap中对应monkey的位置（Position类的对象）并调用repaint方法就可以方便地展示整个过程。

同时注意到由于存在大量的线程的并发操作，调用这些方法时会对某些成员变量产生非常激烈的竞争，这一点将在3.5节中描述。

3.add方法，mov方法和rem方法，分别是用于将某个id的猴子加入，移动或者删除的方法。其中add都是和rem方法都是直接将这个id的猴子加入当前二维平面模型中，并且根据所在的梯子更新他们在模型中的位置。mov方法通过接收猴子的id，v，调用对应id的猴子所在的ladder的next方法获得下一次移动到的踏板的序号，判断是否到达了目的地，如果到达了则调用rem方法并返回false，否则调用updPosition来更新猴子的位置并返回true。



图3.11 RiverPanel控制模型的方法的部分代码

4.updPosition方法，每次RiverPanel改变了某个monkey的位置，需要调用updPosition来更新其位置。updPosition通过在内部创建一个新的线程并启动，同时不断更新objMap并调用repain来实现动画的效果。同时控制移动步长，精确移动到目标位置上。

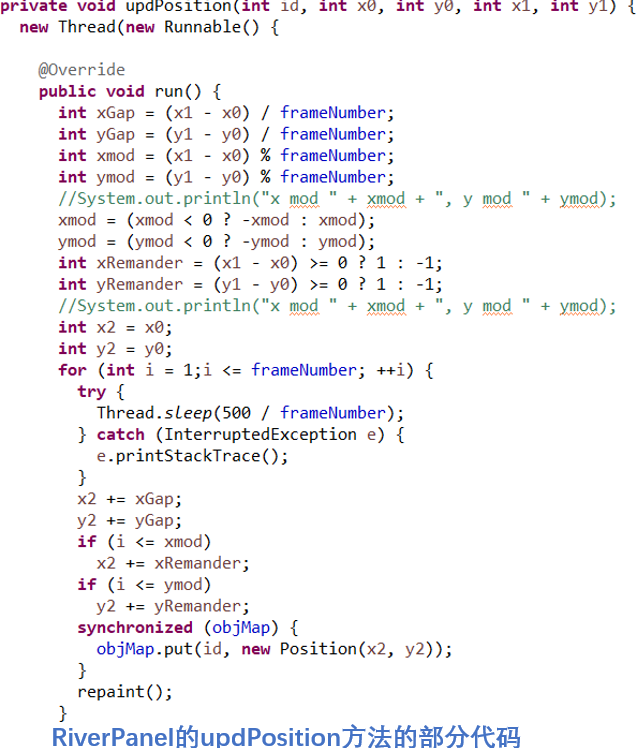


图3.12 RiverPanel实现动画效果的updPosition方法的部分代码

### Applications

1.Applications类整合了上述所有的类，构建成模拟器。运行Applications的main函数即可启动模拟器。Applications中有对两个函数的调用（master分支上），如果想根据外部配置文件（位于txt/configuration.txt）启动模拟器，则取消normal函数的注释并注释掉competition函数

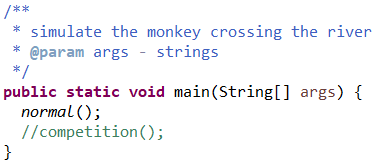


图3.13 根据txt/configuration.txt配置文件启动模拟器

如果想切换为吞吐率竞赛的模式，请注释掉normal函数并取消对competition函数的注释，再次运行main函数将启动对txt/competition.txt配置文件的吞吐量竞赛模式。



图3.14 根据txt/competition.txt配置文件启动吞吐率竞赛

2.normal函数，normal函数用于根据配置文件正常启动模拟器。通过parseFile方法解析文件，注意文件的格式有要求，具体要求见3.8.1节。首先构造梯子组ladders，然后根据ladders和踏板数量构造可视化RiverPanel，最后构造MonkeyGenerator并启动run方法开始运行模拟器。

3.competition函数将根据实验要求中的文件格式解析文件并自主选择策略运行模拟器。具体见3.10节

## Monkey线程的run()的执行流程图

1.流程图见下

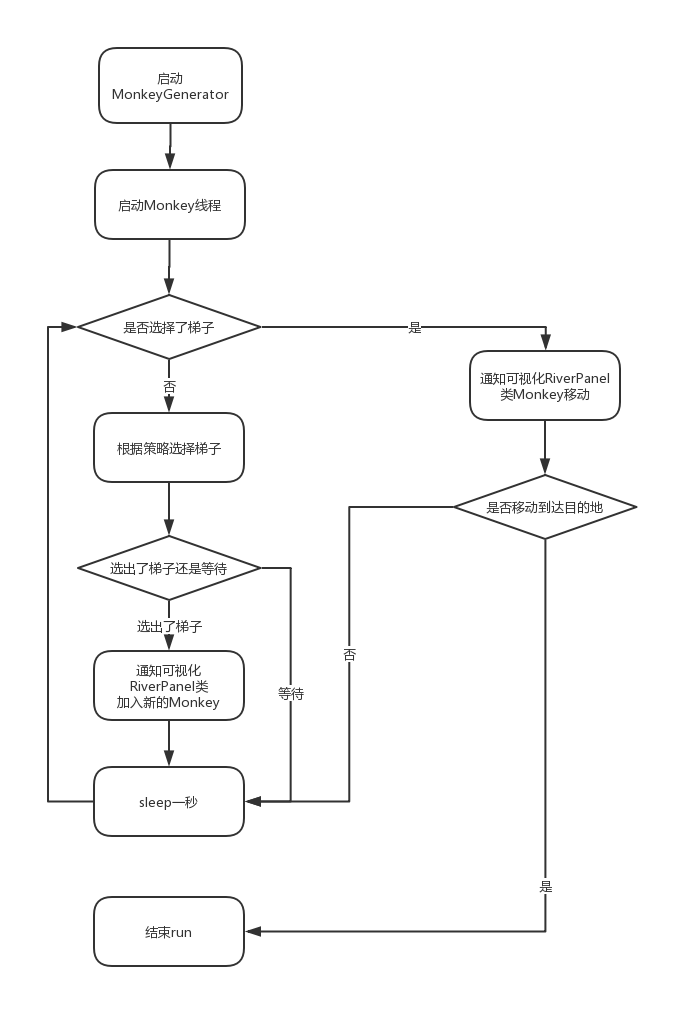


图3.15 流程图

2.一些说明

* MonkeyGenerator会启动很多Monkey线程，这里只画出了一个
* 每次决策的时候，因为不能采用上帝视角，所以会锁住strategy，这就要求所有的Monkey类使用的是用一个Strategy对象，这在之前已经描述过了，同时因为只有一个窗口并在这上面实现可视化，因此所有的Monkey共享一个RiverPanel。注意到Monkey并没有直接接触到梯子组ladders
* 吞吐率的计算是实时的，每次更新位置都会更新吞吐率。而公平性在所有的猴子都到达目的地的时候才会被计算（公平性要求的计算量比较大，不适合实时更新）

## 至少两种“梯子选择”策略的设计与实现方案

这一部分是整个实验最有趣的部分，加上针对吞吐率竞赛的两个策略，一共写了7个策略。在一些策略的分析里面，**我分析了整个游戏的影响因素并反馈到策略中**。

以下策略的性能测试都是基于如下的参数

n = 10（梯子数量）

h = 20（踏板数量）

t = 1（生成猴子的时间间隔）

N = 1000（生成的猴子的总数）

k = 50（每次生成的猴子的数量）

MV = 10（生成的猴子的最大速度）

### 策略1 Wait Strategy

1.WaitStrategy比较简单，主要是我前期为了构建并测试RiverPanel时使用的。中心思想是每次决策的时候遍历所有的梯子，如果没有空的梯子就等，否则就选择遍历到的第一个空的梯子。由于实际上每个梯子上只有一个猴子，因此梯子的利用率不高，显然吞吐率不会很高。整个模拟过程的平均吞吐率大概在1.5-1.8之间。



图3.16 WaitStrategy的get方法

### 策略2 One Side Strategy

1.OneSideStrategy的策略也比较简单，遍历所有的梯子，如果存在一个梯子满足：1.梯子上的所有的猴子的方向与当前决策的猴子的方向相同 2.初始的踏板（最左边或者最右边的踏板）是空的，那么选择这个梯子，否则等待。整个模拟过程的平均吞吐率大概在5.2到5.5之间。

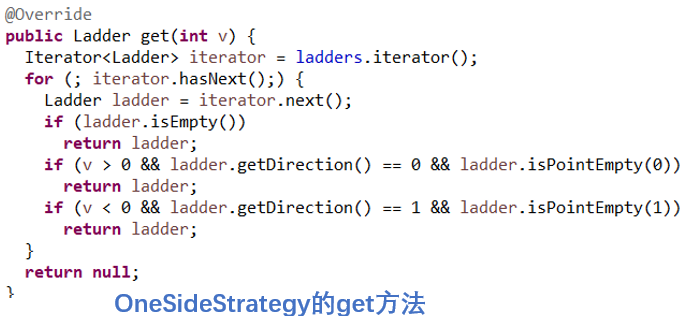


图3.17 OneSideStrategy的get方法

2.真正有趣的现象是，这种看上去随机的方法实际上在真正运行的时候的吞吐率其实比某些根据速度推断的策略的吞吐率不相上下。多次模拟取平均消除一定的随机误差后实际上吞吐率只比根据速度选择的策略低1e-1级别的吞吐率。

为了解释这种现象，我们考虑所有猴子过河所需的最少步数的总和。考虑一只猴子对总和的贡献显然为h / v取上整数。即完成整个过程，所有的猴子必须走的步数至少为 sum = Σ[h / v]。但是显然在梯子有限的情况下，由于存在生成的时间差和速度差，必定会存在速度慢的猴子阻塞速度快的猴子。假设速度快的猴子可以紧紧跟随速度慢的猴子（即每次在前面速度慢的猴子先走，在后面速度快的猴子后走，这样速度快的猴子就不会被卡在原地了），实际上这个时候，速度快的猴子的速度已经变成了速度慢的猴子的速度。根据这个可以计算出速度快的猴子实际上多走的步数为 d = [h / v2] – [h / v1]，其中v1是速度快的猴子的速度，v2是速度慢的猴子的速度。而每次猴子走一步都需要耗费一秒，所以我们要尽量减少多走的总步数。显然总步数取决每一项中两个速度的差值，但是无论是每次选择同方向起点空闲的梯子，还是选择同方向且梯子整体推进速度（或者最近速度）最大或是最小的梯子的策略，实际上都没有考虑v2。所以每当我们选择梯子的时候如果不考虑当前做决策的猴子的速度和梯子上离当前猴子最近的猴子的速度，在大数据和随机生成的先后顺序下在数学期望的差值都是相等的。由此可以解释在大数据意义下OneSideStrategy和其他根据梯子速度等信息的决策方案的差异不太大的原因。

### 策略3 Velocity Strategy

1.虽然上面已经解释过了，OneSideStrategy和其他根据推进速度进行决策的策略的差别不是太大，但是我们还需要实验来证明。这里写出了VelocityStrategy的策略是

每次猴子选择梯子的时候会率先选择同方向且梯子上离他最近的猴子的速度最大/最小（我写了两个版本，在一个Strategy里面）的梯子，如果没有同方向且起点空闲的梯子则等待。整个模拟过程的平均吞吐率大概在5.4到5.8之间。

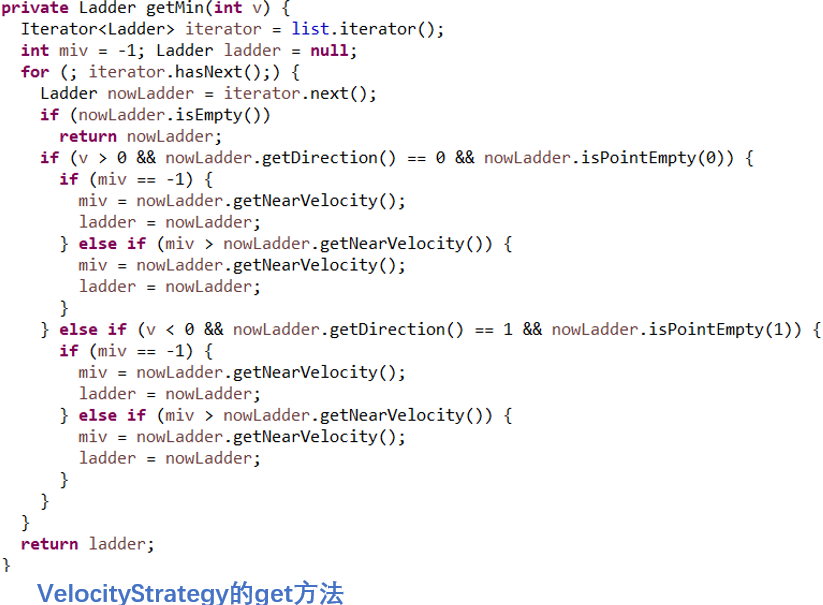


图3.18 Velocity的get方法

2.实验显示，实际上VelocityStrategy的吞吐率和OneSideStrategy的差别确实不大，已经在随机误差的范围内。具体的实验数据可以看3.9节。

### 策略4 Partition Strategy

1.前面已经说明，增加吞吐率的关键之一就是减少梯子上相邻的两个猴子之间的速度差。而在随机顺序的情况我们无法确定下一个猴子的速度。但是我们可以限定两个猴子之间的最大速度差，我们可以人为地将猴子根据速度的不同划分为不同的等价类，同一个等价类内的猴子之间的速度差的最大值是可以得到保证的，然后位于同一等价类的猴子走相同的梯子。

例如，如果在某个参数配置中，最大速度MV为10，现在有10架梯子，我们首先将梯子划分为1-5号梯子和6-10号梯子，1-5号梯子只走从左到右的猴子，6-10号梯子只走从右到左的猴子。进一步，1号梯子只走速度为1和2的猴子，2号梯子只走速度为3和4的猴子，3号梯子只走速度为5和6的猴子，4号梯子只走速度为7和8的猴子，5号梯子只走速度为9和10的猴子。6-10号梯子也类似，不过上面的猴子都是从右向左的。

值得注意的是，这种策略在模拟的过程中可以到达极高的吞吐率（8.0以上，作为对比，OneSideStrategy平均只有5.5左右），但是在快结束的时候吞吐率却极具下降，甚至比OneSideStrategy还低，分析可知这是由于由于我们限定了速度分区，因此在后期某些梯子都是空的，但是有些梯子却有很多的猴子。根据这一点，新的优化是将整个过河过程分为两个过程，前期使用根据速度的等价类划分，后期使用OneSideStrategy。

以上的整个策略就是完整的PartitionStrategy。整个模拟过程的平均吞吐率大概在6.4-7.0之间。



图3.19 PartitionStrategy的部分代码

### 策略5 Extreme Strategy

1.Extreme Strategy是针对n比较大（大于6），MV比较大（大于n-1），N比较大（大于100）的策略，是测试过的最优秀的策略，有时平均吞吐率可以到达8.1，最高观测到的平均吞吐率是8.4784（远高于OneSideStrategy和PartitionStrategy）。而n=10的参数配置下，即使每秒每个梯子上都有猴子到达目的地（实际上不可能），吞吐率为10。已经非常接近这个不可能达到的理论极限。

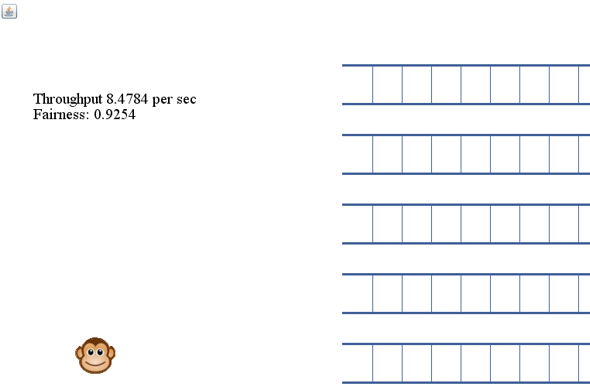


图3.20 观测到的最高吞吐率

我们换一个角度来重新看吞吐率，从吞吐率的计算公式 thoughput = N / T，其中N是猴子的数量，T是最后一只到达目的地的猴子的时间，我们可以发现，对于N，由输入的参数决定，我们无法改变。我们只能改变T，即最后一只到达目的地的猴子的时间。

在PartitionStrategy中促使我们在后期使用OneSideStrategy的真正原因是我们想降低T，但是有许多速度为1的猴子由于等待的时间较长致使T增大。这提示着一种新的策略，为了提高最后一只猴子到达目的地的时间最短，而速度为1的猴子通过梯子所花费的时间过长（是速度为2的猴子的时间的2倍），通常情况下最后一只到达目的地的猴子都是速度为1的猴子，我们可以先保证速度为1的猴子都能以最快速度到达目的地，这样我们可以提高最后一只到达目的地的猴子的期望速度，从而降低T，提高吞吐率，同时，先让速度为1的猴子过梯子也保证了所有的必要时间的最优化，剩下的阻塞时间将大大减少，也会降低T，提高吞吐率。我们专门分配出两条梯子分别让速度为1，从左向右和从左向右的猴子过梯子。将剩下的梯子按速度等价类划分，但是两个方向的相同速度的猴子将会竞争同一个梯子。一旦当所有速度为1的猴子都到达了目的地，立刻转化为OneSideStrategy让所有速度大于1的猴子尽快到达目的地。

以上就是ExtremeStrategy的策略。整个模拟过程的平均吞吐率大概在7.5-8.1之间。



图3.21 ExtremeStrategy的部分代码

### 策略6 Small Strategy

1.SmallStrategy是专门针对小型的吞吐率竞赛配置文件设计的策略，大致思想与PartitionStrategy相同，加入了精确到个的前期和后期的划分标准。具体吞吐率的数值见3.10节。

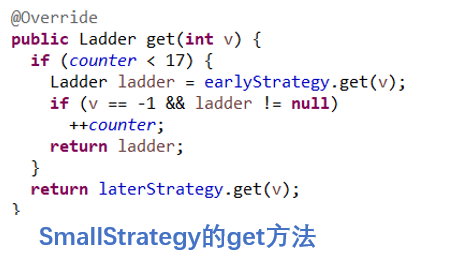


图3.22 SmallStrategy的get方法

### 策略7 Large Strategy

1.Large Strategy是专门为大型的吞吐率竞赛的配置文件设计的策略，主体思想是采用ExtremeStrategy，加入了精细的优化，建立了针对速度为1和速度为2的专用的梯子，同时提高了梯子的利用率，在一定程度上允许特定时期特定梯子的混合使用来提高吞吐率，存在精确到个的时期控制。具体吞吐率的数值见3.10节。



图3.23 LargeStrategy的get方法的部分代码

## “猴子生成器”MonkeyGenerator

1.MonkeyGenerator类，用于生成猴子，核心代码是循环，每次生成k个猴子，并且保持一个number来计数，一旦总个数大于N则跳出循环结束生成猴子的过程，否则将生成速度和方向服从均匀分布的猴子。然后调用Thread.sleep(t \* 1000)，让当前线程休眠t秒。

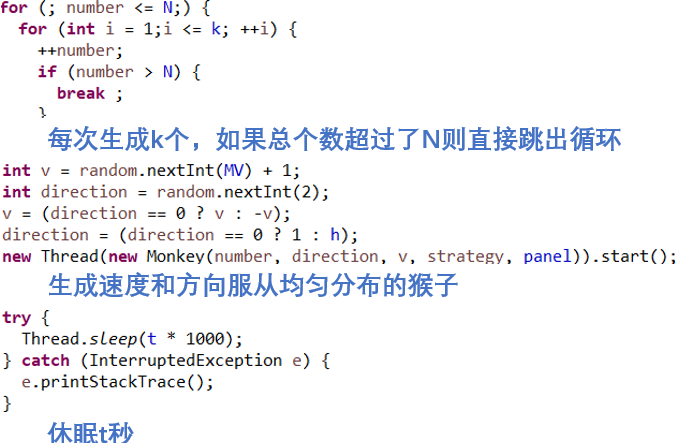


图3.24 MonkeyGenerator生成猴子的代码

2.MonkeyGenerator对梯子策略的选择，根据前面的分析，我们可以得知表现最优秀的算法ExtremeStrategy可以达到很高的吞吐率，但是前提是大数据和随机生成顺序，同理PartitionStrategy也具有这样的要求但是没有ExtremeStrategy强。根据实验结果，实际上在数据量较小的时候，由于系统的随机性，策略之间的差异将会趋于减小，而原本要求在大数据下的策略也有可能因为不均匀的猴子的方向和速度分布导致吞吐率的下降。

根据众多实验结果，最后MonkeyGenerator选择梯子的策略是

当N（生成的猴子总数）小于等于50时，选择OneSideStrategy

当n（梯子的数量）大于6并且MV（最大速度）大于等于n-1的时候，选择ExtremeStrategy

当n大于等于4时选择PartitionStrategy

其他情况选择VelocityStrategy

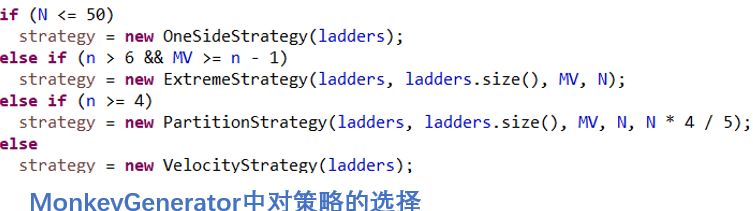


图3.25 MonkeyGenerator中对策略的选择

## 如何确保threadsafe？

1.Monkey中对threadsafe的保证，由于Monkey各线程之间独立，因此在选择梯子的时候会产生很强的竞争，一旦多个猴子同时选择了一个梯子将会导致严重的后果。根据之前介绍的内容，实际上所有的Monkey共享一个Strategy对象。因此每次Monkey在决策梯子的时候，我们只需要锁住梯子，让一个Monkey决策完之后再让下一个Monkey决策，由此解决了Monkey之间对梯子的选择之间的竞争

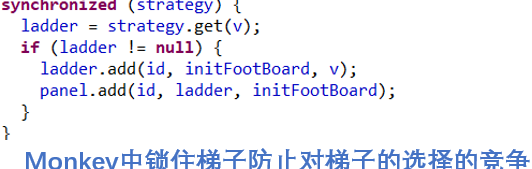


图3.26 Monkey中对threadsafe的保证

2.Ladder中对threadsafe的保证，由于Ladder是mutable并且承担了在各线程之间共享数据的功能，因此Ladder对threadsafe有很强的需求，同一个梯子在改变自身状态的时候应当只被一个线程操作。由于h比较小，最多为20，而Ladder的方法中最高的时间复杂度为O(h)，因此将Ladder的所有方法都设置为synchronized的来保证每个Ladder对象的threadsafe

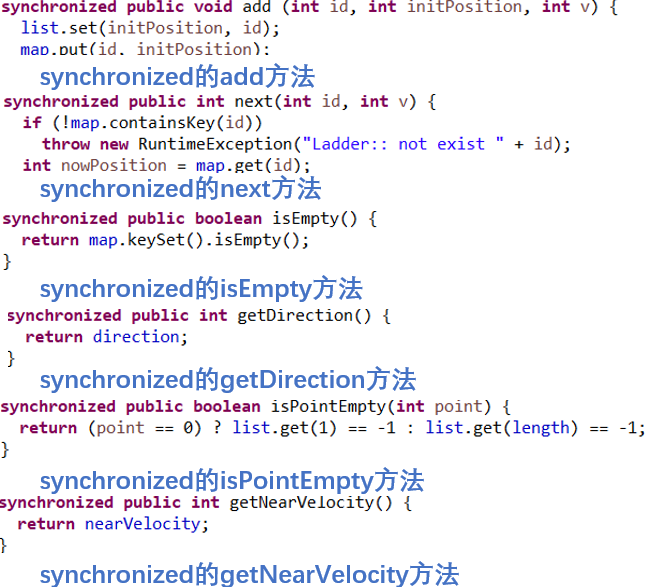


图3.27 Ladder中对threadsafe的保证

3.RiverPanel中对threadsafe的保证，RiverPanel是线程最密集的地方，除了Monkey的线程会发起对RiverPanel方法的调用来改变RiverPanel的状态外，RiverPanel内置的动画线程也会不断地调用repaint。其中竞争最激烈的是表示猴子在平面上的位置的集合类objMap，和统计猴子的出生时间的startMap，

经过测试发现Map的get方法，put方法也是不安全，因此RiverPanel中所有对objMap的调用的地方都被加了锁，同时startMap在put和判断集合大小的时候也被加了锁来保证线程安全。

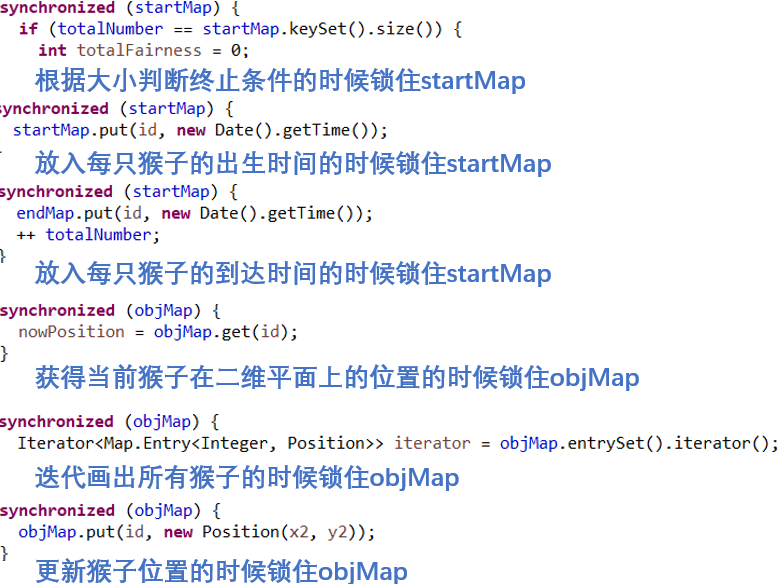


图3.28 RiverPanel中对threadsafe的保证

## 系统吞吐率和公平性的度量方案

1.吞吐率的度量比较简单，在RiverPanel中记录当前到达目的地的总猴子数，每当重画的时候就获取当前时间并利用吞吐率公式throughput = N / T来实时计算吞吐率。

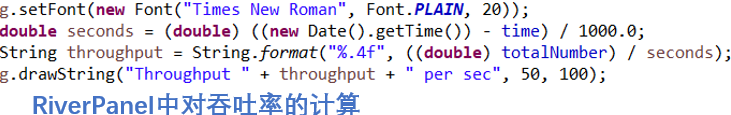


图3.29 RiverPanel中对吞吐率计算的代码

在GUI设计中，左上角会显示实时的吞吐率

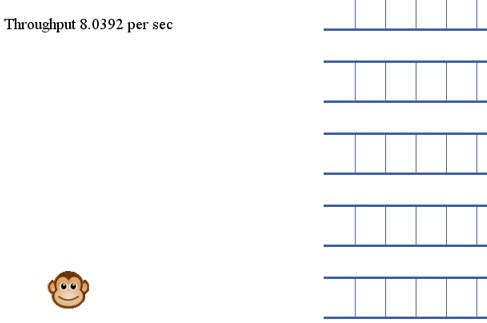


图3.30 实时显示吞吐率

2.公平性的度量，在RiverPanel中通过判断产生的猴子的数量是否等于到达的猴子的数量来判断当前模拟过程时候结束，一旦模拟过程结束，RiverPanel会根据记录的猴子的出生时间和到达目的地的时间来计算公平性。主要逻辑是双层for循环来计算每一对猴子之间的出生时间和到达时间的先后关系并记入答案。



图3.31 RiverPanel中对公平性的计算

在整个模拟过程结束后，公平性会显示在左上角

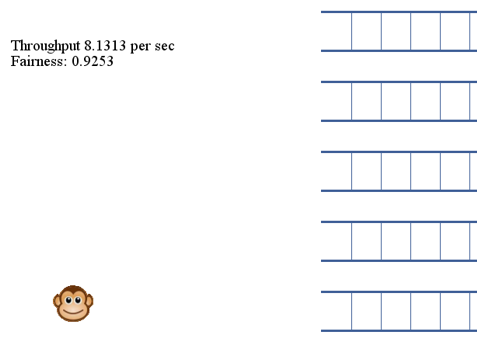


图3.32 整个模拟过程结束后公平性会显示在左上角

## 输出方案设计

**日志文件位于/src/log/info.log**

1.日志，与前面的实验类似，在本次实验中我使用的第三方日志库是log4j 1.2.17。原因有三点，第一系统自带的logging需要在jdk里面改动配置才能使其不在控制台输出，虽然在代码中也可以配置属性，但是由于logger是全局变量，每次都配置比较麻烦。第二log4j 2只支持xml格式的配置，log4j 1.x 可以使用java特性来配置，语法上比较方便。第三log4j对输出到文件的支持很完善，可以自定义格式很方便地输出到log，也方便在筛日志的时候使用正则表达式匹配。

每次猴子被生成，移动或者到达目的地后，都会通过logger的info记录到日志文件中。

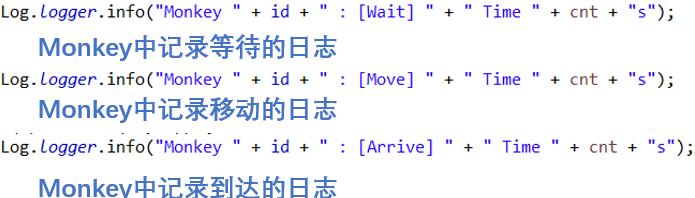


图3.33 Monkey中的日志

打开/src/log/info.log即可看到日志信息

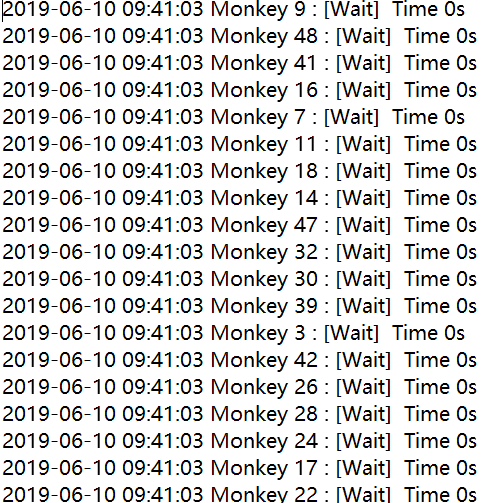


图3.34 部分日志信息

2.动画，为了直观地展示猴子过河的全过程，我实现了动画来展示。通过继承JPanel并重写其paint函数，同时不断调用repaint可以实现动画效果，这一点在之前的实验已经非常详细地介绍了，这里就不再描述，具体的实现细节见代码。

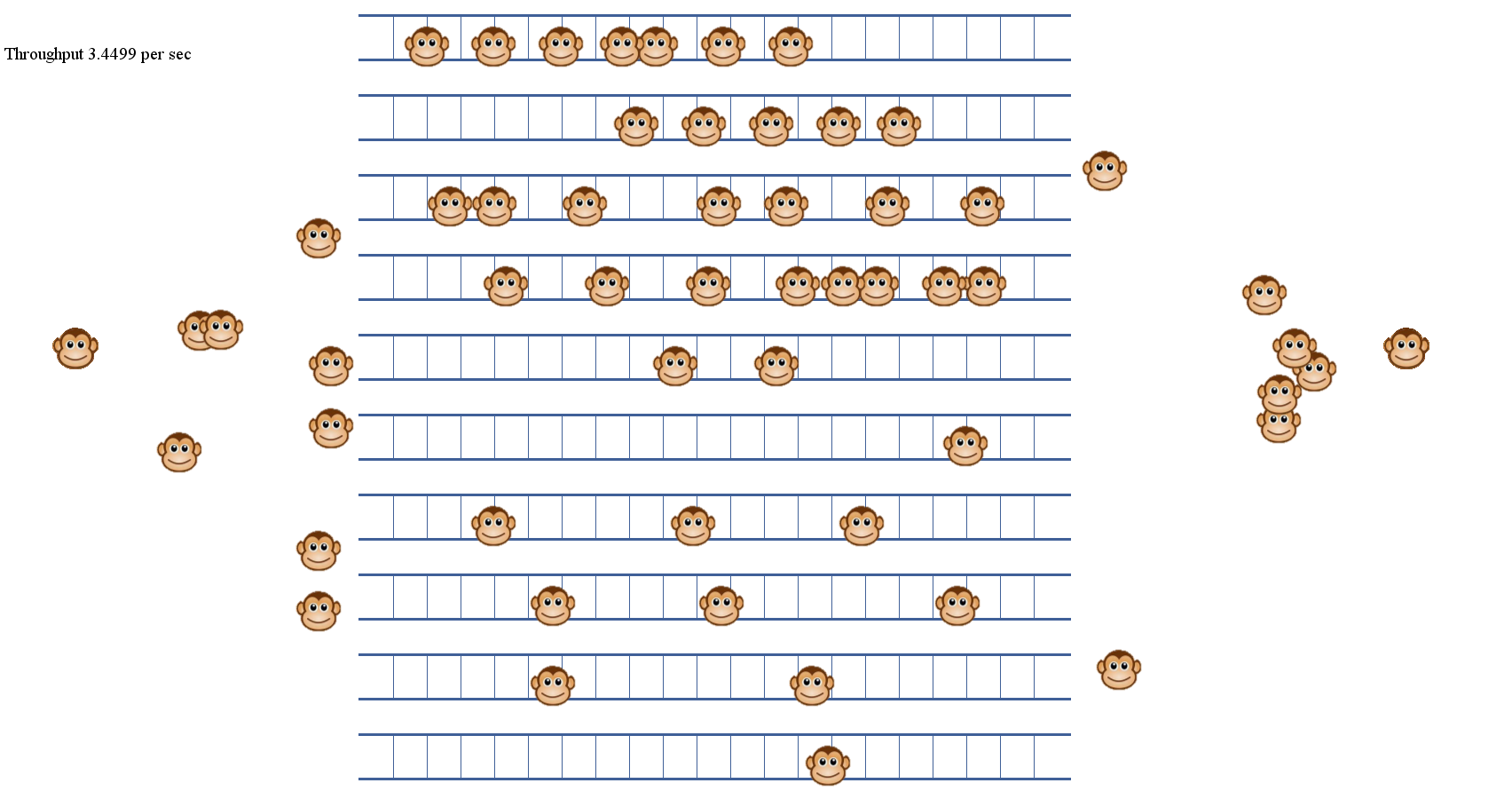


图3.35 动画效果截图

具体的使用方法见3.8节，3.9节和3.10节

## 猴子过河模拟器v1

**如果在v1分支下，配置文件位于txt/ configuration.txt，配置文件请遵守如下格式（v2分支请看 3.9节，v3分支请看3.10节）**

**n=数字**

**h=数字**

**t=数字**

**N=数字(在v1分支中，这里是100，测试ExtremeStrategy时请改为1000)**

**k=数字**

**MV=数字**

**配置完成之后，在applications包下找到Applications，直接运行主函数即可开始模拟**

**由于动画效果的渲染需要大量计算资源，建议在开启模拟器的时候重启Eclipse并关闭其他不必要的应用。**

**测试ExtremeStrategy的最高吞吐率建议在v2分支下进行**

### 参数如何初始化

1.参数从外部文件读入（txt/ configuration.txt），通过正则表达式解析来获得n，h，t，N，k，MV



图3.36 正则表达式解析外部文件

解析完成之后根据n构造有n个Ladder对象的List，构造RiverPanel，然后构造MonkeyGenerator，同时将RiverPanel放入JFrame中实现可视化。最后启动MonkeyGenerator的run方法即可开始模拟过程。

### 使用Strategy模式为每只猴子选择决策策略

1.根据之前的描述，每个猴子都有一个Strategy成员变量来帮助决策梯子，而Strategy是一个接口，我们通过创立不同的Strategy的具体实现类并作为创建Monkey时的参数即可实现Strategy设计模式。



图3.37 Strategy设计模式的应用

2.在具体实现的时候，结合3.4节所说的决策的选择方式，针对不同参数选择不同的策略来使得吞吐率尽量大

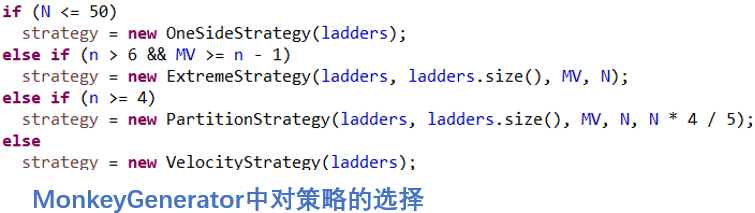


图3.38 Strategy设计模式中对实现类的选择

## 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。

**在v2分支下利用配置文件开启模拟器的方法和配置文件与v1相同，具体方法见v1**

### 对比分析：固定其他参数，选择不同的决策策略

1.固定参数为

|  |  |
| --- | --- |
| 参数 | 大小 |
| n | 10 |
| h | 20 |
| t | 1 |
| N | 1000 |
| k | 50 |
| MV | 10 |

分别对WaitStrategy，OneSideStrategy，VelocityStrategy，PartitionStrategy和ExtremeStrategy测试三次，结果如下

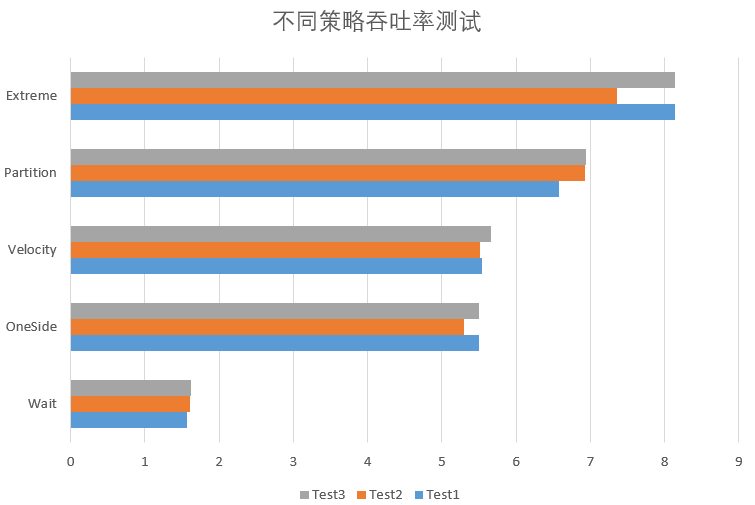


图3.39 不同策略的吞吐率测试

可以看在大型数据下，所有的策略的表现和3.3节中所说的相同。WaitStrategy由于对梯子的使用率不高所以导致吞吐率较低，而OneSideStrategy和VelocityStrategy大致相当，PartitionStrategy较高，ExtremeStrategy最高，具体的分析可以见3.3节中对各策略的描述。

2.由于公平性测试实际上对参数不是非常敏感，我们将在3.9.3节中描述不同参数和不同策略下公平性的表现和性质。

### 对比分析：变化某个参数，固定其他参数

1.变化参数n，具体的参数变化如下

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | CONFIG1 | CONFIG2 | CONFIG3 |
| n | 10 | 9 | 8 |
| h | 20 | 20 | 20 |
| t | 1 | 1 | 1 |
| N | 1000 | 1000 | 1000 |
| K | 50 | 50 | 50 |
| MV | 10 | 10 | 10 |

对所有的策略进行测试，得到结果如下

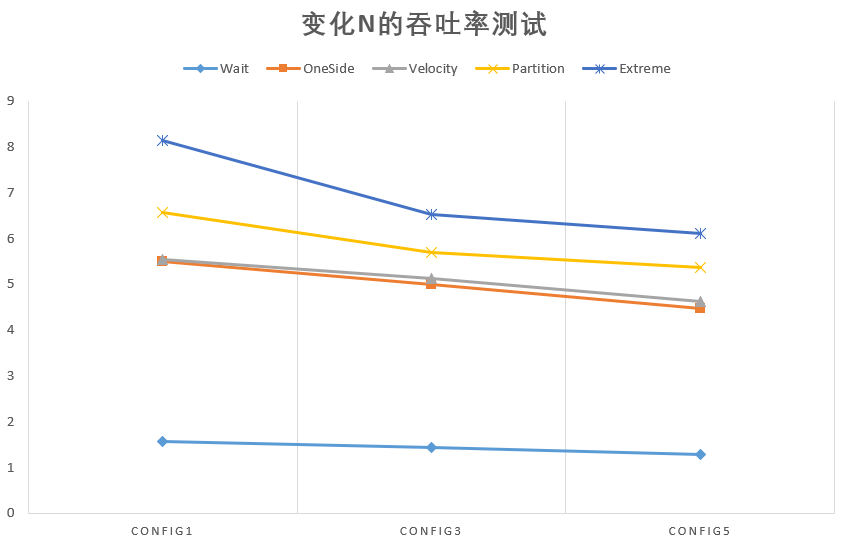


图3.40 变化n的吞吐率测试

2.可以看到随着梯子数量n的下降，所有策略的吞吐率都在下降。这是显然的，总量不变，梯子数量减少，吞吐率必然下降。可以看到下降的最明显的是ExtremeStrategy，这是因为ExtremeStrategy高度依赖梯子的数量，当梯子数量下降的时候，某些速度等价类的大小上升导致不同速度之间阻塞的增加，使得吞吐率下降，同理PartitionStrategy也采用了按速度等价类划分，因此受到的影响也较大。而VelocityStrategy和OneSideStrategy中影响决策的因素并不包含梯子的数量，因此我们可以看到吞吐率大致呈线性下降。

### 分析：吞吐率是否与各参数/决策策略有相关性？

1.根据上面和3.3节的分析，我们可以知道吞吐率既和参数有关，也和决策有关，两者共同决定了吞吐率的大小。参数决定了吞吐率的理论上限和整个过河过程的变化趋势，策略决定了有多接近理论的上限和具体的行为。

2.我们还可以知道，根据3.3节的分析和3.9.1节的实验结果，不同的决策实际上有不同的作用范围，每种策略只有在自身的范围中才能更好地发挥自身的长处。当数据量较小的时候，随机因素影响较大，有梯子就上是较好的策略，而当数据量较大的时候，整体呈现出规律性，我们便可以利用规律（3.3节中介绍的）来优化策略从而实现更高的吞吐率。

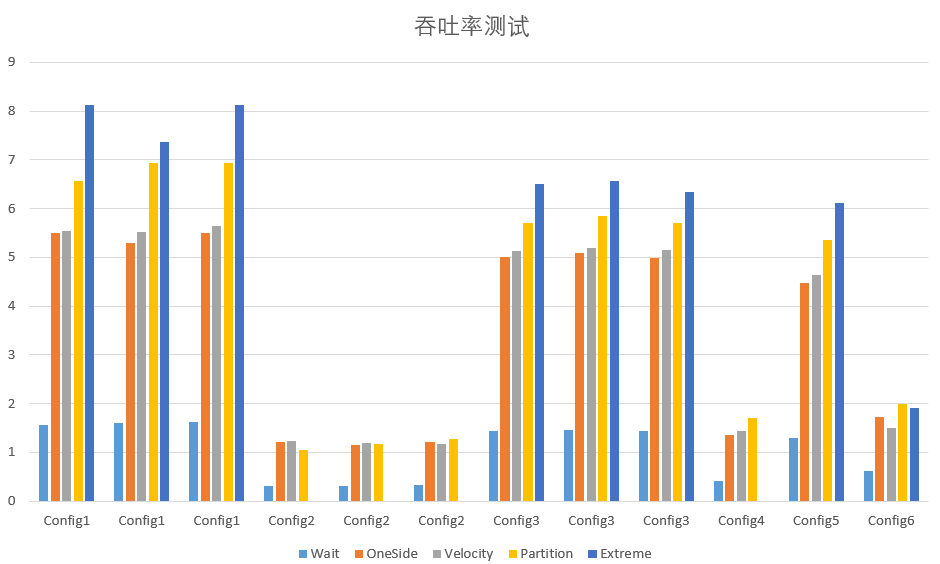


图3.41 不同配置不同策略下的吞吐率测试

从上表也可以看出，在不同的配置下，吞吐率之间的差别非常大，config1，config3和config5是大型数据，因此吞吐率较高，config2，config4，config6是小型数据，因此吞吐率较低。同时在同一配置下，不同的策略带来的吞吐率也不一样。

3.公平性测试，从实验可以看出，公平性实际上不是很依赖参数，而是与策略有着非常密切的关系，与策略的吞吐率大致成反比。关于这一点的解释，我们可以认为如果想得到更高的吞吐率，无疑我们需要更加有序的编排方案，而人为的有序将带来公平性的下降。

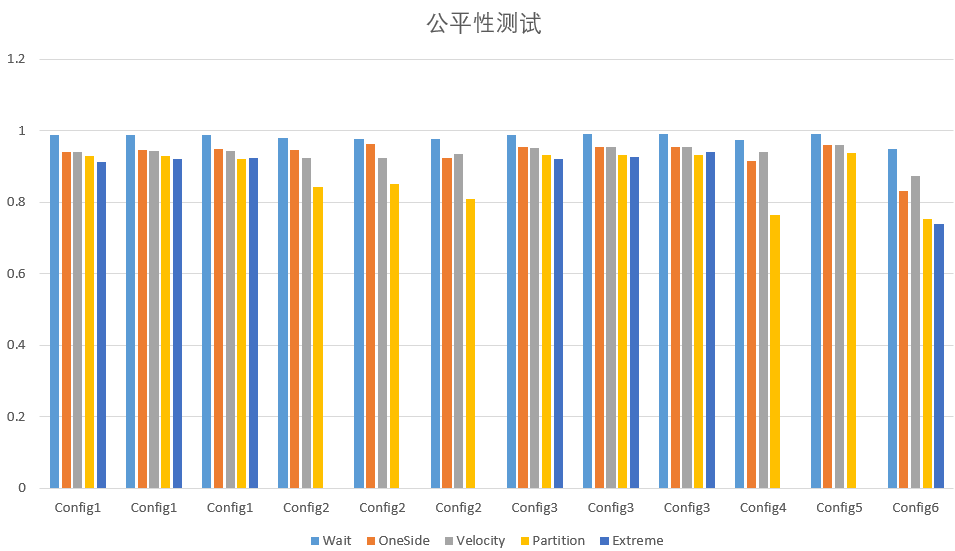


图3.42 不同策略和不同参数的情况下公平性对比

### 压力测试结果与分析

1.压力测试一，高密度猴子和有限数量的梯子，参数配置如下

|  |  |
| --- | --- |
| 参数 | 大小 |
| n | 6 |
| h | 20 |
| t | 1 |
| N | 1000 |
| K | 50 |
| MV | 10 |

测试的结果如下

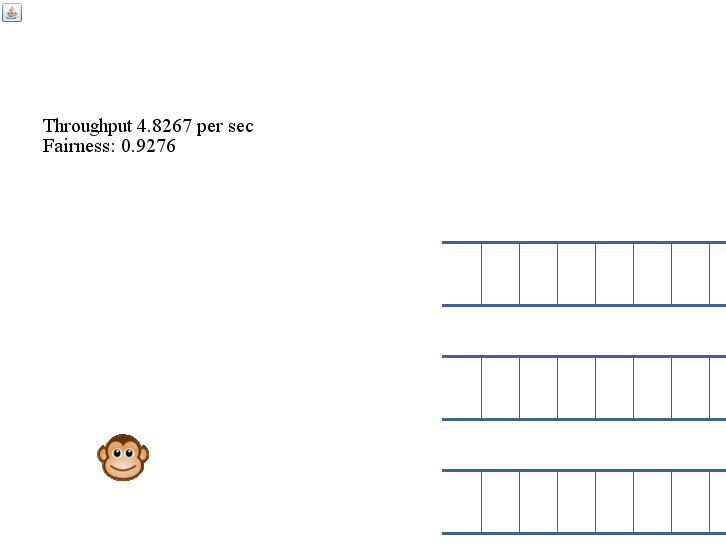


图3.43 压力测试一结果

可以看到吞吐率达到了4.8267，公平性为0.9276。根据之前的分析可知，此时采用的是PartitionStrategy。应当说是比较高的吞吐率了。

2.压力测试二，速度差异大，实际上在前面的测试中最大速度都是10，由分析可知此时在大型数据中应当采用ExtremeStrategy来获得最大的吞吐率。参数如下

|  |  |
| --- | --- |
| 参数 | 大小 |
| n | 10 |
| h | 20 |
| t | 1 |
| N | 1000 |
| K | 50 |
| MV | 10 |

测试结果

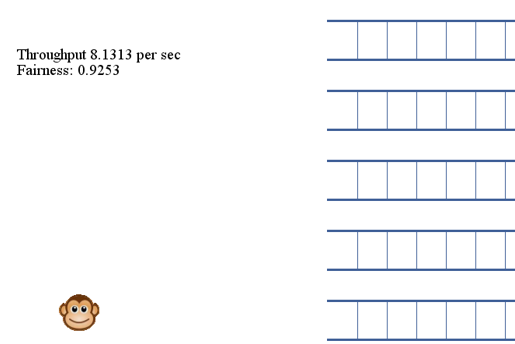


图3.44 压力测试二测试结果

可以看到在速度差异较大的情况下，ExtremeStrategy策略还是非常有效的。

## 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。

**v3的使用方法（在v3分支下）**

**吞吐率竞赛模式：在txt/competition.txt中粘贴三个吞吐率竞赛文件中的某一个的内容，然后在applications包下找到Applications类，直接运行main函数即可，吞吐率实时更新，公平性将在模拟结束后显示（运行main函数之前请确保main函数中对competition函数的调用）**

**普通模拟器模式：在main函数中取消对normal的注释，注释掉competition函数，在txt/configuration.txt中进行配置（格式与v1，v2一样），运行main函数即可。**

|  |  |  |
| --- | --- | --- |
|  | 吞吐率 | 公平性 |
| Competiton\_1.txt | n=5 | h=20 |
| 第1次模拟 | 3.005 | 0.8982 |
| 第2次模拟 | 3.0935 | 0.8935 |
| 第3次模拟 | 3.1213 | 0.8872 |
| 第4次模拟 | 3.0875 | 0.8886 |
| 第5次模拟 | 3.0981 | 0.8897 |
| 第6次模拟 | 3.0308 | 0.8818 |
| 第7次模拟 | 3.0614 | 0.8928 |
| 第8次模拟 | 3.0323 | 0.8959 |
| 第9次模拟 | 3.0915 | 0.8791 |
| 第10次模拟 | 3.1571 | 0.8815 |
| 平均值 | **3.07785** | **0.88883** |
| Competiton\_2.txt | n=10 | h=20 |
| 第1次模拟 | 6.5207 | 0.8497 |
| 第2次模拟 | 6.4115 | 0.8498 |
| 第3次模拟 | 6.3322 | 0.8526 |
| 第4次模拟 | 6.3324 | 0.8442 |
| 第5次模拟 | 6.9514 | 0.8551 |
| 第6次模拟 | 6.8584 | 0.8591 |
| 第7次模拟 | 6.7726 | 0.8521 |
| 第8次模拟 | 6.5951 | 0.8468 |
| 第9次模拟 | 6.7632 | 0.8477 |
| 第10次模拟 | 6.954 | 0.8588 |
| 平均值 | **6.64915** | **0.85159** |
| Competiton\_3.txt | n=3 | h=20 |
| 第1次模拟 | 1.6138 | 0.8384 |
| 第2次模拟 | 1.6174 | 0.8242 |
| 第3次模拟 | 1.6444 | 0.8141 |
| 第4次模拟 | 1.4497 | 0.8154 |
| 第5次模拟 | 1.5692 | 0.8162 |
| 第6次模拟 | 1.5879 | 0.8154 |
| 第7次模拟 | 1.4526 | 0.8251 |
| 第8次模拟 | 1.5664 | 0.8485 |
| 第9次模拟 | 1.5221 | 0.8533 |
| 第10次模拟 | 1.516 | 0.8267 |
| 平均值 | **1.55395** | **0.82773** |

# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2019-6-1 | 18:00-23:30 | 看java多线程编程相关的知识和注意事项 | 大致了解了多线程和GUI |
| 2019-6-2 | 14:00-15:30 | 设计架构 | 设计完了，在做实验看是不是和想象的一样 |
| 2019-6-2 | 21:00-23:30 | 在做小型实验验证动画制作相关的想法 | 和想象的一致，扫清了所有疑问，可以开始写了 |
| 2019-6-4 | 14:00-17:30 | 开始写parts包 | 写完了主要部件，包括RiverPanel |
| 2019-6-4 | 19:00-23:30 | 继续写parts包 | 写完了parts包，但是没有通过测试，依赖关系有一些问题，同时二维平面模型的定位不太精确 |
| 2019-6-5 | 15:30-17:30 | 完成parts包的测试，整合parts包，写applications包 | 写完了，初步测试通过，达到了可以使用的程度 |
| 2019-6-5 | 20:00-23:30 | 开始写strategy包 | 写出了OneSideStrategy和VelocityStrategy，分析问题的结构 |
| 2019-6-6 | 14:00-17:30 | 整理优化，开发新的strategy | 完成了PartitionStrategy，吞吐率有了很大的提升 |
| 2019-6-6 | 18:30-23:30 | 继续优化 | 完成了ExtremeStrategy，感觉吞吐率已经快到极限了 |
| 2019-6-7 | 15:40-17:30 | 完成细节，例如log功能和公平性 | 搞定 |
| 2019-6-8 | 18:00-23:30 | 完成综合测试，测试整体架构的稳定性 | 改进了竞争的问题，优化了GUI |
| 2019-6-9 | 18:00-23:30 | 编写针对吞吐率竞赛的strategy | 优化完成 |
| 2019-6-10 | 14:00-15:30 | 写完实验报告 | 搞！定！了！ |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 可视化多线程之间的竞争问题 | 关键是很难重现问题，试了好多次，找到了竞争最激烈的地方，锁住了 |
| 一步步优化吞吐率 | 根据实验要求写完的两个策略没有很大的差异，很不爽，分析问题后发现了更好的策略 |
| 测试多线程的正确性 | 没有找到特别有效的，主要是输出多线程中的信息来对比看是否符合。 |
| GUI上的猴子总是漂移 | 发现是二维平面模型更新位置的时候定位不精确，用了一点数学工具解决了 |
| GUI多个猴子上了同一个梯子 | 是在决策梯子的时候产生了强烈的竞争导致的，后面发现了决策时的性质，让所有猴子共享同一种策略并且在决策的时候锁住，解决了竞争问题 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

* 多锻炼多线程编程能力，之前这方面接触的非常少
* 多写程序，事实证明对代码的掌控能让自己的完成目标的时候更加自如
* 增强代码的调试能力
* 多使用现代化工具

## 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？

出现了竞争的问题。能，多线程控制下的GUI的刷新速度是根据自身的变化而决定的，在lab3中单线程不得不不断刷新，虽然有时候没有变化。多线程则解决了这个问题。

1. 你采用了什么设计决策来保证threadsafe？如何做到在threadsafe和性能之间很好的折中？

控制mutable的范围，控制线程之间共享数据的范围。将所有线程共享的范围缩减到Ladder和Strategy上，一旦出现竞争问题我们只需要关注这两个ADT。折中依赖具体的模型，泛泛而谈折中不如根据实际的应用模型做出决定。

1. 你在完成本实验过程中是否遇到过线程不安全的情况？你是如何改进的？

是，猴子在决策梯子时候出现了竞争。让所有猴子共享同一个Strategy并在决策时锁住Strategy。

1. 关于本实验的工作量、难度、deadline。

已经习惯了lab系列的工作量了，因为之前写了很多GUI，这次难度主要集中在多线程的掌控上。ddl在期末死亡周之前…

1. 到此为止你对《软件构造》课程的意见和建议。

还是不变的态度，可以学到很多东西，也十分有用，建议还是希望能让步伐再从容一点

1. 还有一周就要期末考试了，你准备如何复习？

从课堂ppt和雨课堂复习知识点，重点看写过的实验，再次思考实验过程使用到的上课介绍过的原理和知识，重新演练场景，看看学了一学期之后再写刚开始学的时候写的实验自己会做出怎样的设计。最后谢谢自己以前把实验报告写的很详细。