САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА МОДЕЛИРОВАНИЯ ЭКОНОМИЧЕСКИХ СИСТЕМ

Каримов Зуфар Рафаэлевич

Дипломная работа

Компьютерная модель дуополии Хотеллинга

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Ковшов А.М.

Санкт-Петербург
2016

Содержание

Аннотация

Объектом исследования данной работы стало влияние начальных параметров на равновесие в дискретной модели дуополии Хотеллинга: начальных цен, шаг изменения цены, стоимости проезда за единицу пути. Кроме этого опробованы три различные стратегии поведения продавцов. На каждой стратегии были рассмотрены различные начальные данные. Сделаны выводы о их влиянии на динамику цен. Реализована компьютерная программа на языке Java. В дипломную работу входит: введение, две главы, выводы, заключение и приложение.

Введение

В 1929г. Г.Хотеллинг предложил новую модель ценообразования [1]. Дополнив модели Курно и Бертрана, кроме самой цены на товар, он учитывал еще и затраты на преодоление расстояния от дома покупателя до месторасположения продавца. Ему удалось найти равновесие цен на линейном рынке. Но задача о расположении осталась нерешённой. Салоп в своей работе [2] преобразовал линейную модель Хотеллинга в круговую, расположив её вокруг города. В работе [3] исследована проблема оптимального расположения на плоскости. В [4] найдено равновесие на плоскости и решена задача о размещении, город принят за единичный круг. Квадратный город представлен в [5], решены задачи о ценообразовании и расположении.

В данной работе создана компьютерную модель дуополии Хотеллинга, с возможностью выбора как линейного, так и плоского города в виде квадрата. Реализовано три стратегии поведения продавцов при выборе цен на товары. Оценено влияние начальных параметров на динамику цен при выборе разных стратегий.

Постановка задачи

Рассмотрим дискретную модель дуополии Хотеллинга в двух вариантах: линейный город находящийся на прямой и город располагающийся на плоскости. В городах через равные промежутки проживают покупатели. Также в городе есть два продавца, которые продают один и тот же товар по ценам C_1 и C_2 соответственно. Координаты продавцов возьмем за x_1, y_1 и x_2, y_2. Покупатели стремятся приобрести товар минимизировав свои затраты. Стоимость товара S для покупателя суммируется из цены на товар C и транспортных издержек T_c, равных произведению расстояния от покупателя до продавца на цену проезда одной единицы пути α. Объем денежных средств покупателя будем считать неограниченным. Продавцы, в свою очередь, стремятся максимизировать свою прибыль, путем изменения цены на товар. Реализуем данную модель в виде компьютерной программы и рассмотрим три стратегии поведения продавцов при различных начальных параметрах.

Глава 1. Описание компьютерной модели

1.1. Задание начальных параметров.

Для города реализуем возможность задания следующих переменных: размер, вид (линейный или квадратный), метрика для расчета расстояния (Евклида или Манхеттена) и стоимость проезда единицы пути (рис. 1).

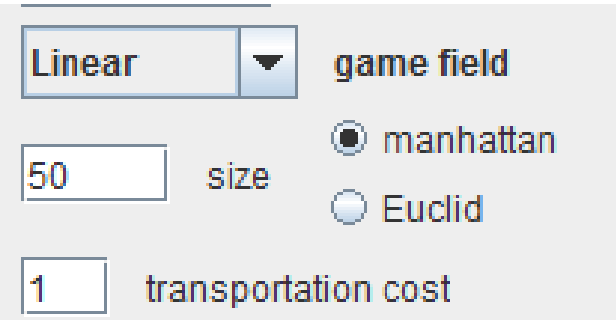


Рис. 1 Задание параметров города

Также необходимо задать параметры продавцов: их координаты x, y и начальную цену на товар C_1 и C_2 соответственно. Для более удобного расположения продавцов на диагонали используем кнопку $x_1=y_1$.

Переменная c1 min будет отвечать за минимальный шаг изменения цены в некоторых стратегиях. Прибыль H_1 и H_2 высчитывается программой после каждого изменения цены любым из продавцов (рис. 2).

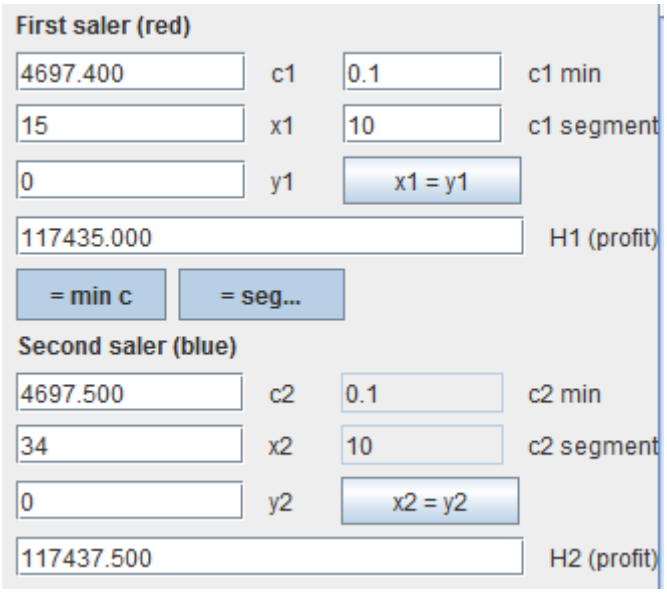


Рис. 2 Задание параметров продавца

1.2. Стратегии поведения продавцов.

Для максимизации прибыли продавец прибегает к изменению цены на свой товар. Введем три стратегии:

1) Продавец волен изменять цену на любое количество денежных единиц. Обозначим как max1.
2) Продавец изменяет цену строго на определенную величину c1 min (c2 min). Обозначим как diff1.
3) Продавцу доступно изменение цены на определённую величину c1 min (c2 min) как во втором случае. Кроме этого появляется вероятность выбора неоптимальной цены.

Первые две стратегии подразумевают поочередное изменение цен продавцами, в третьей продавцы действуют одновременно. Рассмотрим поподробнее последнюю стратегию. Каждому продавцу задается параметр риска R. С вероятностью $P = \frac{R}{10}$ он рискнет и изменит цену не в оптимальном направлении, в надежде что с учетом изменения цены вторым продавцом его прибыль увеличится. Каждый раз после изменения цены параметр R может быть изменен: если продавец рискнул и его прибыль увеличилась, то R увеличится на 1, если прибыль уменьшилась, то R уменьшится на 1. Если же продавец придерживался стратегии максимизации прибыли и не рисковал, то R остается неизменным.

1.3. Управление программой.

Для ускорения поиска оптимальной цены введем переменную c1 segment (рис. 2). Поиск оптимальной цены будет происходить на отрезке
[C – c1 segment, C + c1 segment]. Уменьшая этот параметр мы ускоряем поиск. Переключатели =min c и =seg служат для удобства задания одинаковых значений для обоих продавцов.

Чтобы начать работу зададим начальные параметры и нажмем кнопку

init.

Справа можно наблюдать схему города, размещение продавцов и долю покупателей, которая выберет конкретного продавца (рис. 3).

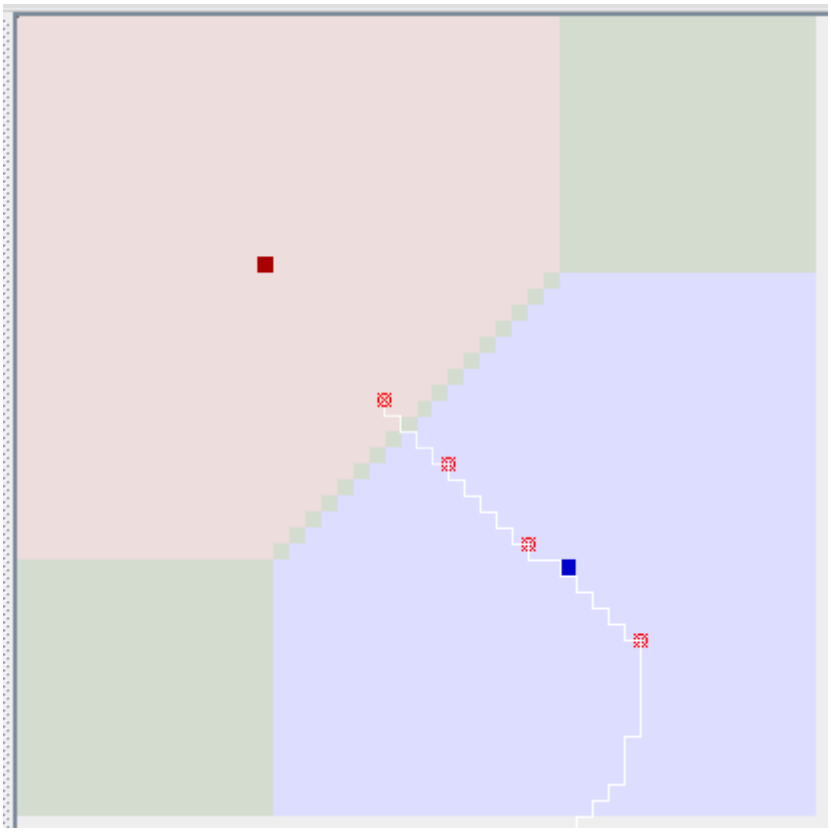


Рис. 3 Графическое представление города, координаты продавцов и траектория изменения цен.

Красная область охватывает покупателей которым выгоднее ехать к красному (первому) продавцу, и голубая для синего (второго) продавца. Покупатели из серой области нейтральны, затраты на приобретение товара одинакова при выборе любого продавца. При расчете прибыли будем считать, что одна половина «серых» покупателей поедет к одному продавцу, а другая ко второму. Белая линия представляет собой траекторию изменения цен: горизонтальная ось отвечает за красного продавца, вертикальная отвечает за синего продавца.

Поочередные ходы продавцов выполняются за счет кнопок **optim. c1** и **optim. c2**. При нажатии на эти кнопки будем производить поиск максимума

прибыли H1 на заданном отрезке, определяемом выбранной стратегией. Параметру C_1 присвоим новое значение, на которым был найден максимум, также изменим и переменную H1, записав туда найденный максимум. Для того чтобы выполнить несколько ходов за раз добавим кнопку **optim. c1c2** и форму для ввода кол-ва шагов. Для выбора стратегии 1 или 2 необходимо выставить **max 1** или **diff 1** в выпадающем меню **price policy**.

Для стратегии 3 используем кнопку **opt. both** и форму для ввода кол-ва шагов за 1 нажатие. Оптимизация прибыли по третьей стратегии возможна только этой кнопкой. Здесь же можно задать переменные R_1 и R_2 отвечающие за уровень риска продавцов. Во время работы эти переменные будут изменяться. Красные и зелёные окошки справа от R показывают рисковал ли продавец на последнем шаге (рис. 4).

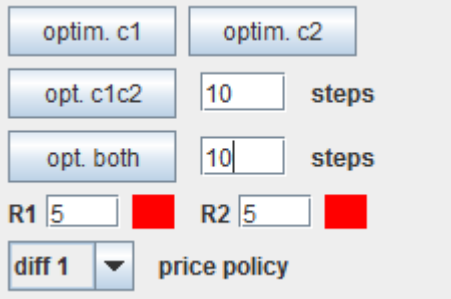
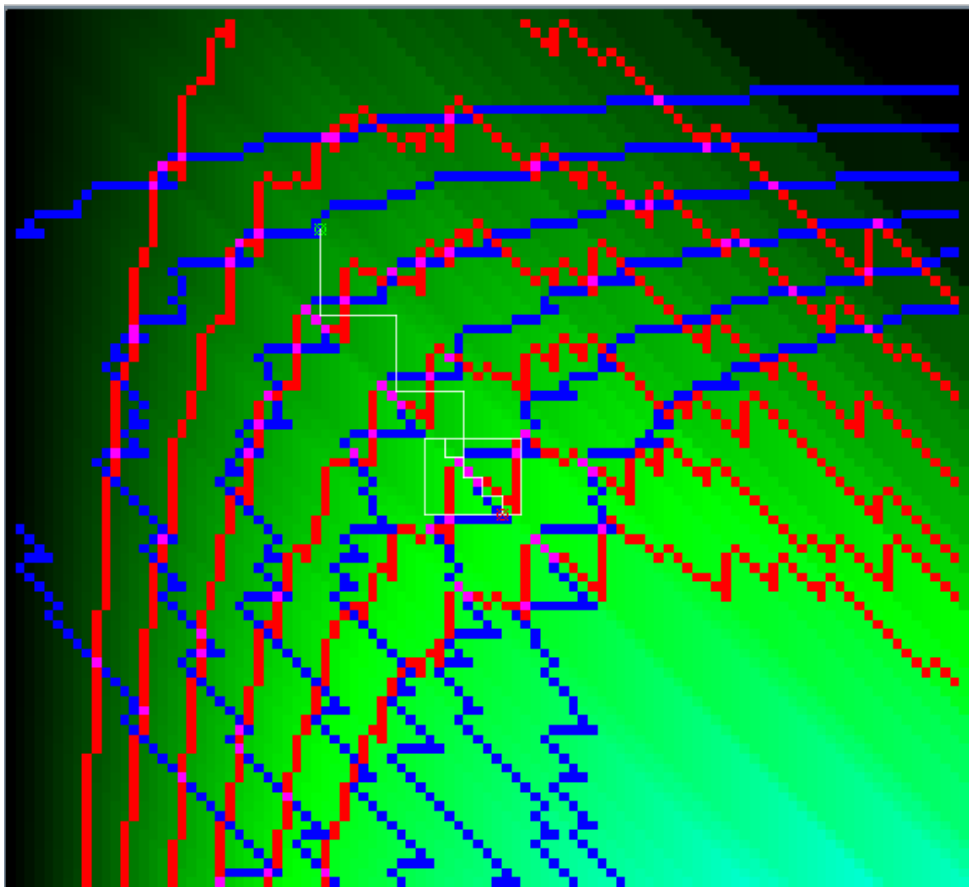


Рис. 4 Элементы управления, выбор стратегии, задание и отслеживание рисков.

Отображение траектории цен включается кнопкой **show**, удаление кнопкой **clear**.

1.4. Графическое представление функции прибыли.

Для лучшего понимания поведения продавцов реализуем графическое отображение функции прибыли. На горизонтальной оси расположим цену первого продавца, на вертикальной цену второго продавца. Красными и синими изолиниями отобразим заданную прибыль для каждого из продавцов. Интересующие величины прибыли можно установить в блоке profit function,



через запятую. Для отображения необходимо нажать кнопку **H1.**

Рис. 5 Изолинии прибыли с заданными значениями 120, 130, 140, 150, 160, 170

Взглянув на рис. 5 можно заметить, как синий продавец при выборе цены стремится попасть в вершину изолинии прибыли. Код программы можно посмотреть в Приложении.

Глава 2. Применение модели

2.1. Выбор начальных данных.

Воспользуемся программой и проследим за динамикой цен при использовании всех трех реализованных стратегий. Т.к. набор начальных параметров достаточно широк, ограничимся изменением начальных цен продавцов C_1 и C_2, минимальным шагом изменения цен c1 min (приравняем c2 min к c1 min) и ценой на проезд одной единицы пути α. А также риском R для третьей стратегии. Координаты продавцов и размеры города возьмем равными $x_1=y_1= 15$, $x_2 = y_2 = 34$, size = 50. Для расчета расстояний будем использовать метрику Манхеттена. Выбранные наборы начальных параметров:

1) C1 = C2 = 1, c1 min = 0.1, α = 0.1
2) C1 = C2 = 10, c1 min = 0.1, α = 0.1
3) C1 = 1, C2 = 10, c1 min = 0.1, α = 0.1
4) C1 = C2 = 5, c1 min = 0.5, α = 0.1
5) C1 = C2 = 5, c1 min = 0.1, α = 0.01
6) C1 = C2 = 5, c1 min = 0.1, α = 1

Обозначим их п.1 – п.6

Первые три набора помогут проанализировать динамику цен при различных начальных ценах, четвёртый при увеличенном ограничении на минимальный шаг изменения цены на товар, пятый и шестой при низких и высоких ценах на проезд.

В отличии от классической непрерывной модели Хотеллинга, данная модель дискретна, что часто приводит к невозможности нахождения равновесных цен.

2.2. Линейный город.

Запустим модель с начальным данными п.1. Выбрав стратегию max1 попытаемся получить равновесные цены. Как видно по траектории изменения цен на Рисунке 6, равновесия достичь не удается. Оба продавца постепенно

поднимают цены пока не входят в цикл: попеременного снижения и поднятия цен. В данном случае продавцы настолько снижают цену, что периодически захватывают весь город, оставляя конкурента с нулевой прибылью.

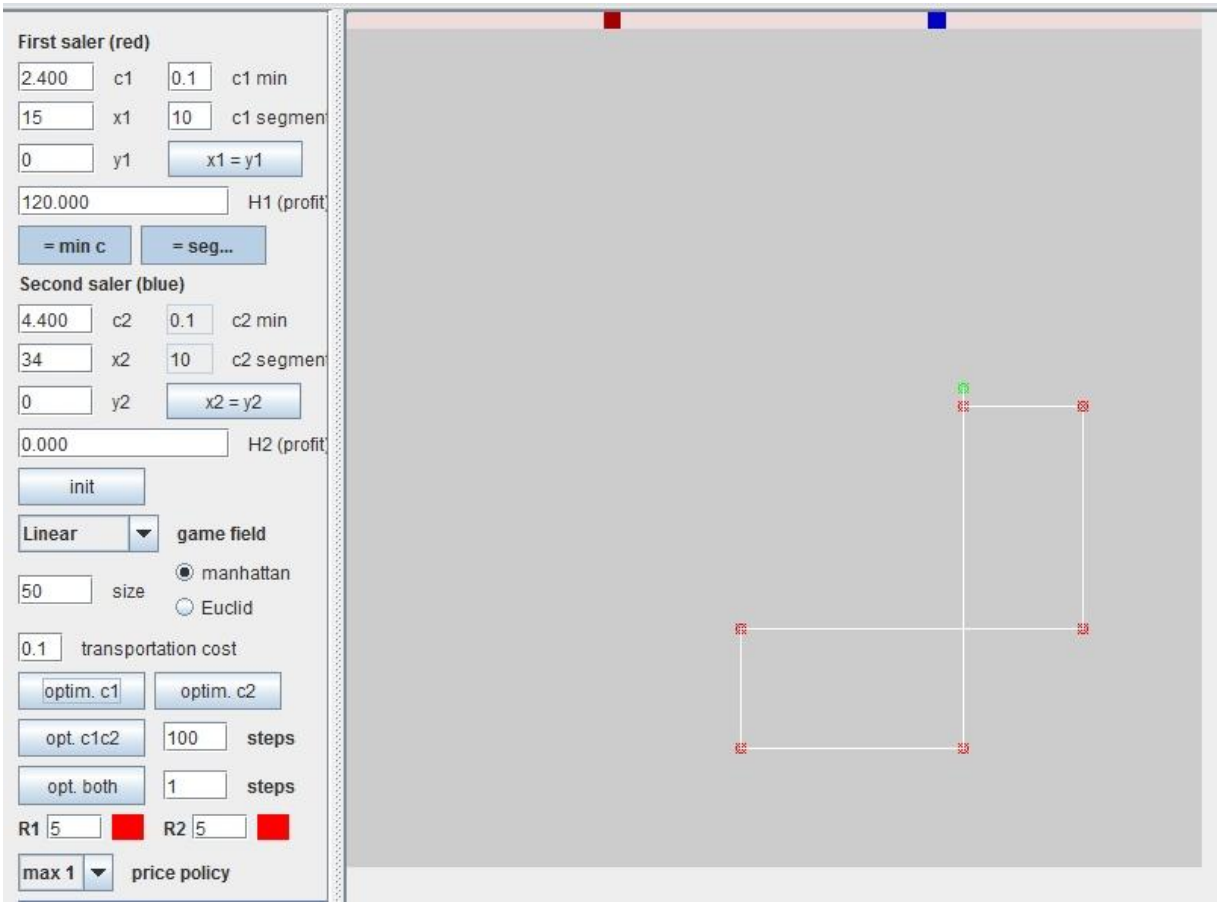


Рис. 6 Динамика цен на линейном городе при выбранной стратегии max1

Подстановка данных п.2 и п.3 результатов не изменяет, продавцы снижают и поднимают цены до тех пор пока не придут к указанному выше циклу. Из чего можно сделать вывод что начальная цена при достаточно большом количестве шагов для стратегии max1 значения не имеет.

Данные п.4 приводят к тому что продавцы вновь попадают в цикл, но уже из четырех точек: колебания цен вверх и вниз на c1 min. Также не возникает ситуаций с нулевой прибылью.

После подстановки данных п.5 и п.6 можно заметить тенденцию к сильному удешевлению и подорожанию товаров. Стоимость проезда является очень важным параметром в образовании цены.

Стратегия diff1 вынуждает продавцов либо изменять цену незначительно, либо вообще не трогать, если один из продавцов имеет слишком высокую цену и изменение на c1 min не приносит результатов, прибыль по прежнему равна нулю. Без возможности к резким скачкам продавцам удалось добиться равновесных цен (или попасть в цикл из 4 точек) используя данные п.1, п.2 и п.4. Можно заметить, что начальные цены C_1 и C_2 влияют на их конечные (оптимальные) величины. Если один из продавцов изначально выставил заметно большую цену, то и при попадании в цикл снижения увеличения на c1 min, его цена будет выше. Если же начальные цены были почти равны, то и в точке равновесия они будут одинаковы.

В пятом случае (п.5) цены упали до минимума, в шестом (п.6) они бесконечно растут. Стоит обратить внимание на результаты подстановки начальных данных п.3. Один из продавцов установил слишком высокие цены и его прибыль равна нулю. Каждый шаг он пытается оптимизировать цены, но из-за узкого радиуса поиска он не находит правильного направления и просто оставляет цену как есть. Второй продавец в это время постепенно поднимает цены и увеличивает свою прибыль. В тот момент, когда цена первого приближается к цене второго, продавец с нулевой прибылью понимает, что для увеличения дохода ему необходимо снижать цену. В итоге оба продавца приходят к равновесной цене. Если бы продавец с низкой прибылью вовремя остановил рост своих цен, то он бы так и остался без конкуренции (рис. 7).

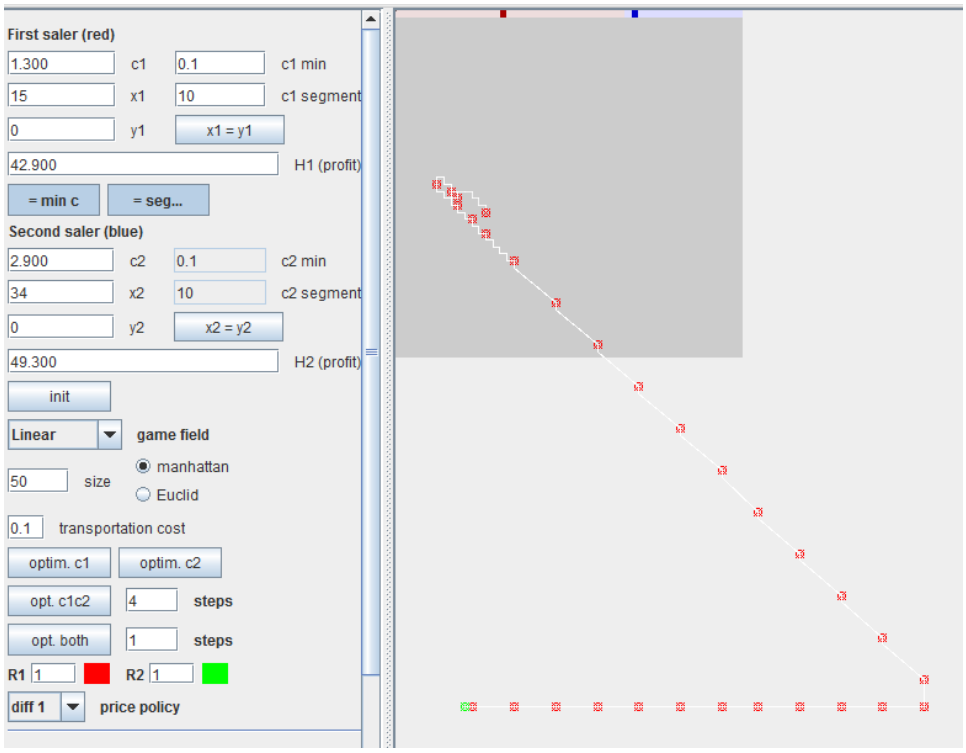


Рис. 7 Стратегия diff1 с начальными данными из п.3.

Использование стратегии с параметром риска показывает результаты похожие на данные полученные при использовании стратегии diff1, но с небольшими возмущениями. При высоких начальных R, продавцы то стремятся к равновесным ценам, то наоборот удаляются от них. Но через определённое количество шагов понимают, что риск чаще приводит к негативным последствиям и R стремится к единице. В этом случае динамика изменения цен стратегии risk максимально приближена к динамике изменения цен стратегии diff1: цены точно так же стремятся к равновесию, минимуму и бесконечности при разных начальных данных (рис. 8).

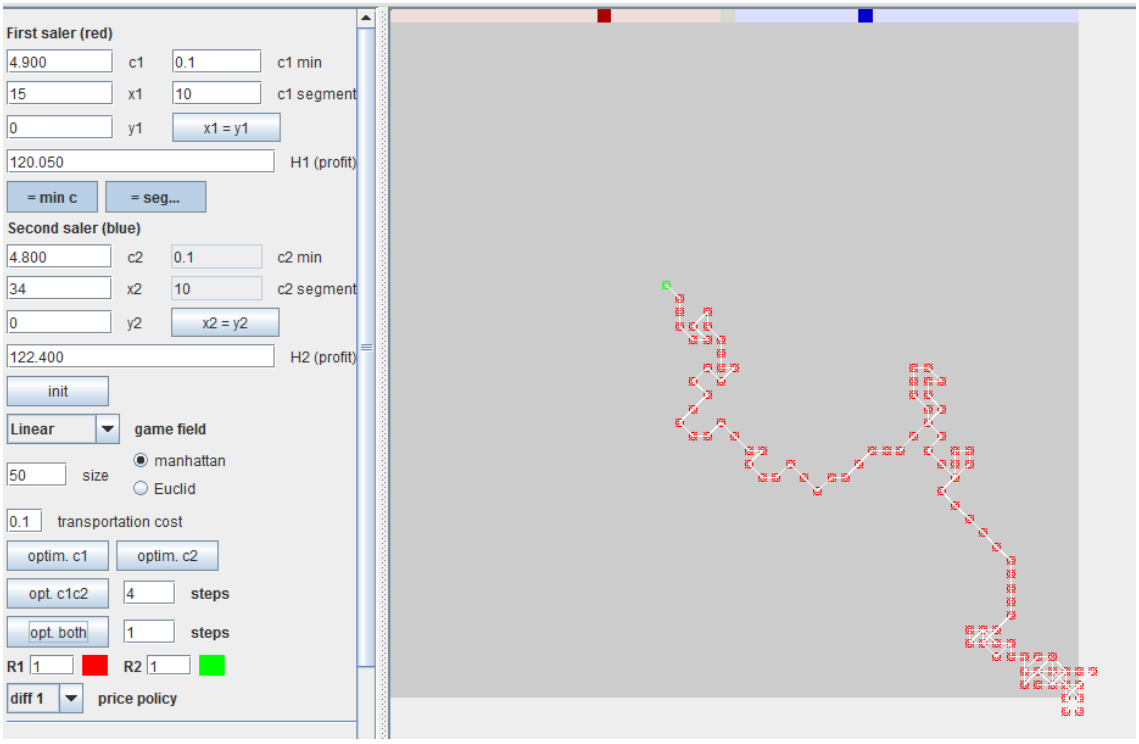


Рис. 8 Стратегия risk, колебания возле равновесных C_1=C_2=5

2.3. Город на плоскости.

Так как использовалась метрика Манхеттена, то довольно большое количество людей оказались в нейтральной зоне. Можно этого избежать расположив продавцов возле одной из граней, сократив расстояние между ними или используя метрику Евклида.

Рассмотрим результаты использования различных стратегий и начальных данных по аналогии с линейным городом. Переход от прямой к плоскости не помогает в достижении равновесных цен, кроме того появляются новые виды «равновесных циклов» (рис. 9). Аналогично линейному случаю, при использовании max1 начальные цены не влияют на конечную точку равновесия или цикл. Динамика цен для diff1 и max1 при использовании п.5 похожа на линейный вариант: цены достигают минимума при низких значениях α. Однако, при высоких значениях α (п.6), стратегия max1 приводит к цикличному изменению цен на высоком уровне, а diff1 либо сводит цены к минимуму, при большой разнице C_1 и C_2, либо бесконечно их повышает. Продавцы использующие diff1 никогда не достигали равновесных цен, при любых начальных параметрах попадали в цикл из 4 точек.

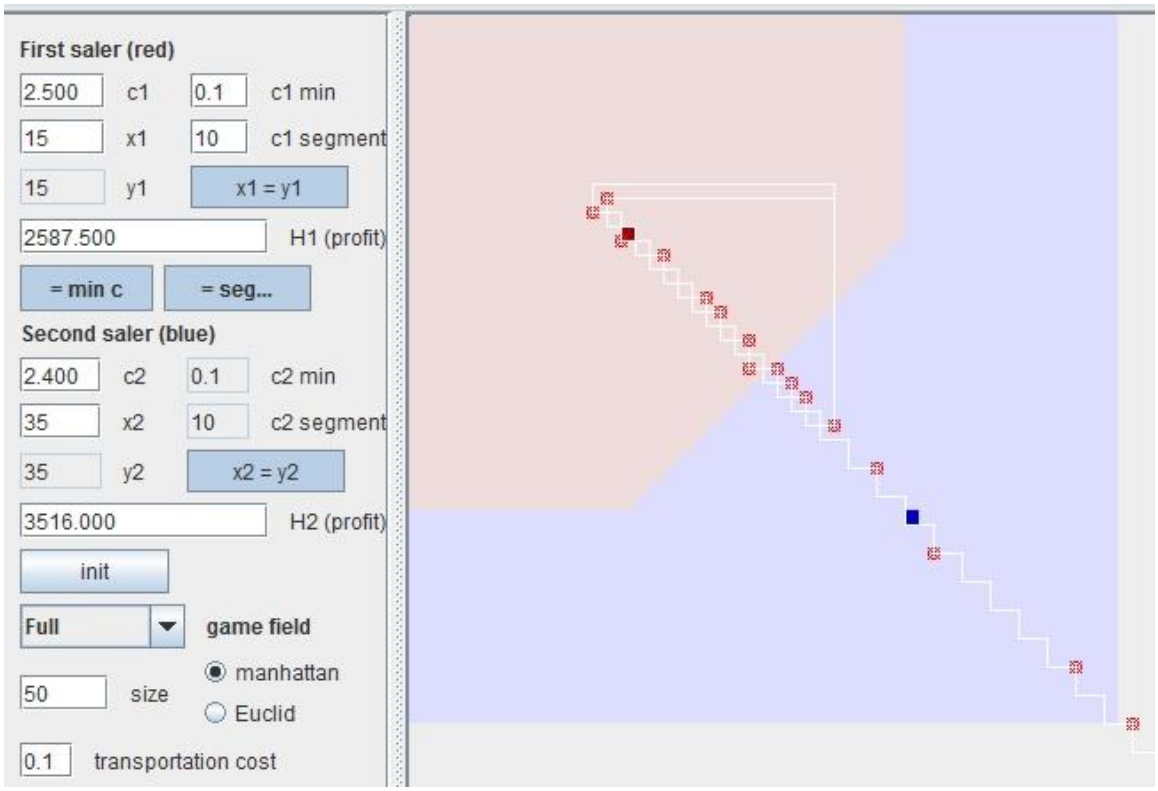


Рис. 9 «равновесный цикл» долгое попеременное снижение и два резких подъема цен.

В силу большего количества покупателей при размещении на плоскости, стратегия risk показывает более сильные отклонения от средних цен (рис. 10).

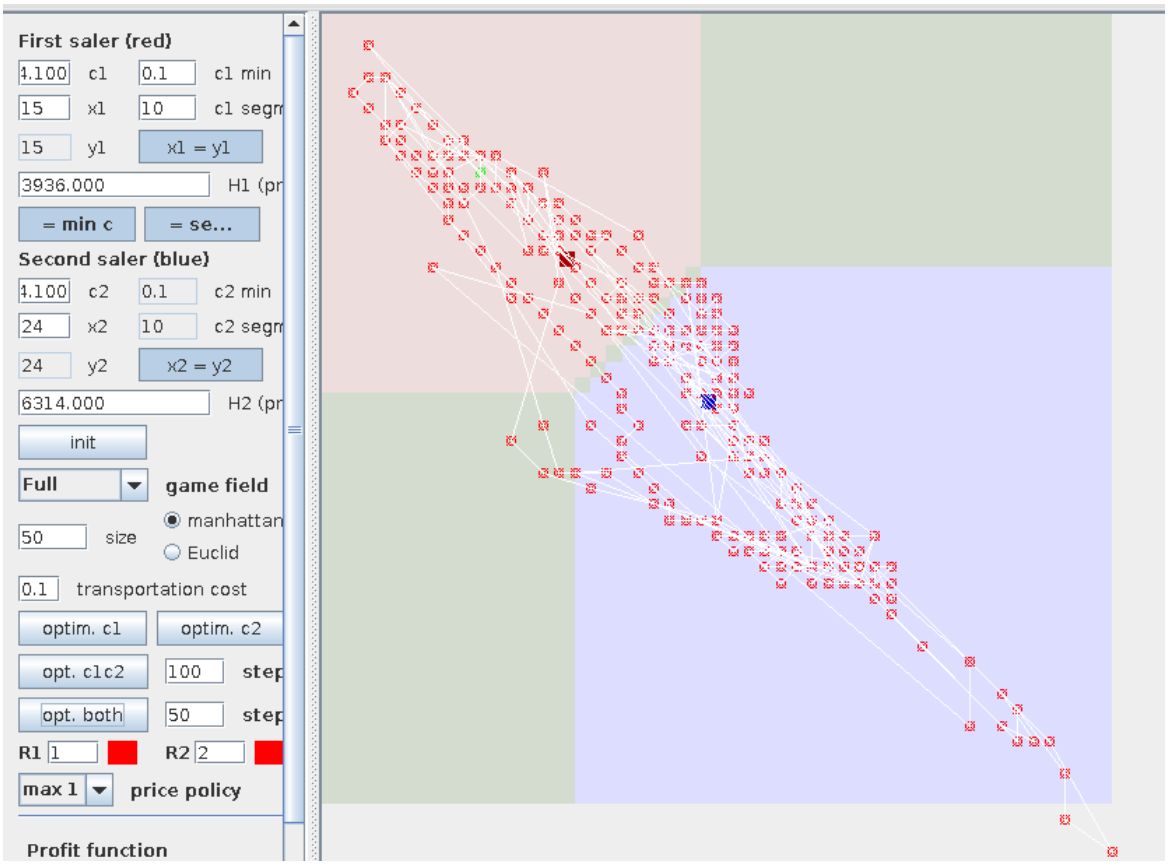


Рис. 10 Стратегия risk в городе на плоскости.

Стратегия risk хорошо показала себя при использовании данных п.6.При C1 = C2, высоких транспортных расходах и большом количестве покупателей в «серой зоне», снижение цены кажется очевидным шагом к повышению прибыли. Так решают оба продавца, одновременно снижают и оказываются в предыдущей ситуации, но с меньшей прибылью. Снижение будет продолжаться до минимума. Но если один из продавцов рискует и меняет стратегию, сразу же начинается рост цен и прибыли. Как видно на рис. 11, происходит снижение цен на диагонали, и рост вне её.

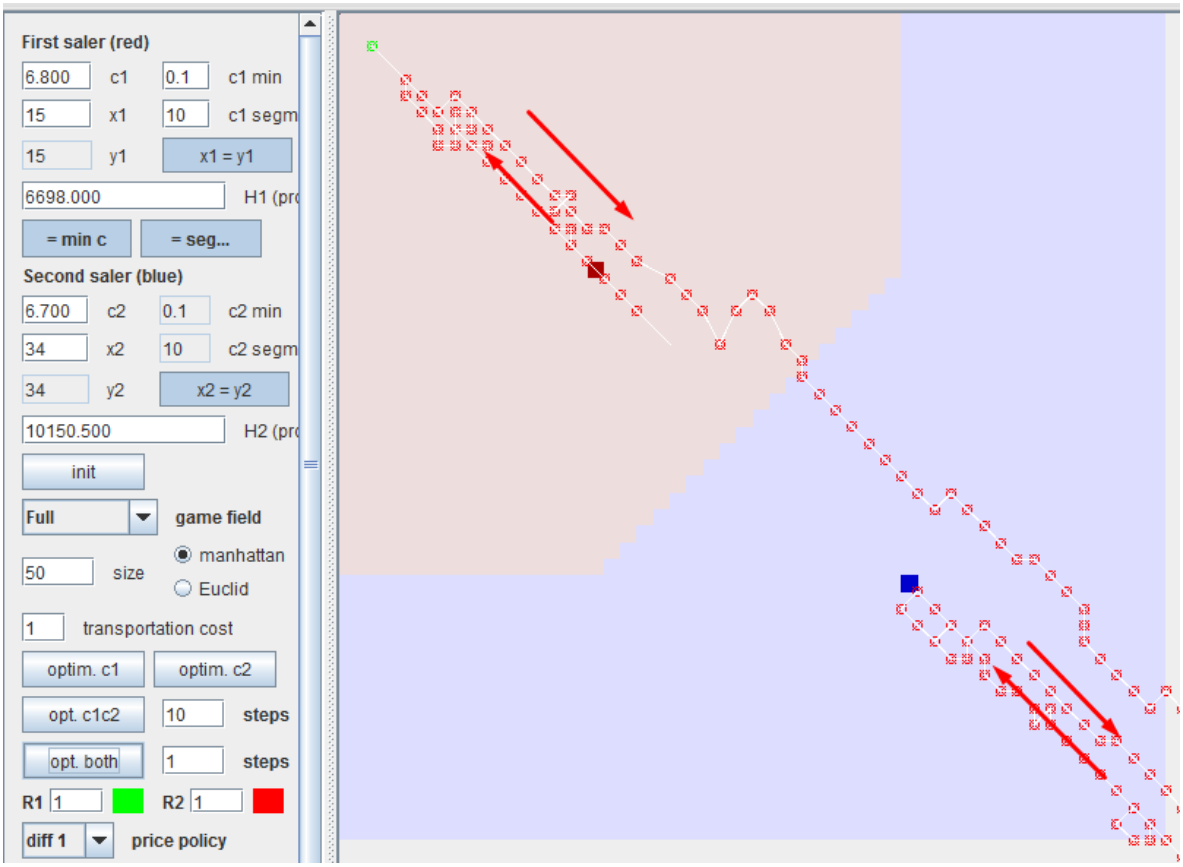


Рис. 8 Стратегия risk, данные п.6 снижение цен при равных C1 и C2, увеличение в остальных случаях.

Выводы

Практические результаты показывают, что при использовании стратегии max1 начальные цены обоих продавцов не оказывают влияния на конечные значения цен, при достижении равновесия или вхождения в цикл. C1 min влияет на детальность выбора цен и скорость схождения к равновесию. Важным оказался параметр α, низкие значения которого приводят к минимизации цен, а высокие - к бесконечному росту цен на линейном городе.

Заключение

В данной работе была описана схема работы реализованной компьютерной модели дуополии Хотеллинга. Опробованы три стратегии поведения продавцов, при различных начальных параметрах, на прямой и плоскости. Получены оценки величины влияния начальных параметров на динамику цен. Можно сделать вывод что модели поведения на прямой и плоскости различаются не сильно. Стратегия risk почти не отличается от стратегии diff1, однако элемент случайности помогает выбираться из точки равновесия с минимальными ценами.

Список литературы

1. Hotelling H. Stability in Competition // Economic Journal. 1929. Vol. 39.
2. Salop S. Monopolistic competition with outside goods // Bell journal of Economics. 1979. Vol. 10. P. 141–156.
3. Dresner, Z. Competitive location strategies for two facilities// Regional Science and Urban Economics, Vol. 12, 1982, 485-493.
4. Мазалов В. В., Щипцова А. В., Токарева Ю. С. Дуополия Хотеллинга и задача о размещении на плоскости // Экономика и математические методы. 2010. т. 46, Вып. 4.
5. Мазалова А. В. Дуополия Хотеллинга на плоскости в метрике Манхеттена // Вестн. С.-Петерб. ун-та. Сер. 10: Прикладная математика, информатика, процессы управления. 2012. Т. 2. С. 33–43.

Приложение

```
public class Game {
    public SellerGamer seller1 = new SellerGamer(new Point(), 0, 0);
    public SellerGamer seller2 = new SellerGamer(new Point(), 0, 0);
    public double unitDistanceCost = 0;
    public Metric metric;
    public Dimension gameFieldSizes = new Dimension();
    public GameField gameField;
    public int[] p = null;

    public int[] constructProfitFunction(double stPrice1, double fnPrice1, double stPrice2, double fnPrice2, float[] levels) {
        double savePrice1 = seller1.getPrice();
        double savePrice2 = seller2.getPrice();
        double d1 = seller1.getMinCoin();
        double d2 = seller2.getMinCoin();
        double r1 = 1;
        double r2 = 1;
        int width =  (int) ((fnPrice1 - stPrice1) / d1) + 1 ;
        int height = (int) ((fnPrice2 - stPrice2) / d2) + 1 ;
        int length = width * height;
        int[] ret = new int[2 * length];
        int k = 0;
        double price2 = stPrice2;
        for (int j = 0; j < height; j++, price2 += d2) {
            seller2.setPrice(price2);
            double price1 = stPrice1;
            for (int i = 0; i < width; i++, price1 += d1, k++) {
```

```
            seller1.setPrice(price1);
            calculateCurrentPosition();
            ret[k] = (int) (seller2.getProfit() / r2);
            ret[k + length] = ret[k];
            ret[k] = (int) (seller1.getProfit() / r1);
         }
      }
      for (int i = 0; i < levels.length; i++) {
         for (int j = width; j < ret.length - width; j++) {
            if (j >= ret.length / 2 - width && j < ret.length / 2 + width)
               continue;
            if (j % width == 0 || j % width == width - 1) continue;
            int n = ret[j - width] & 0x00FFFFFF;
            int s = ret[j + width] & 0x00FFFFFF;
            int w = ret[j - 1] & 0x00FFFFFF;
            int o = ret[j + 1] & 0x00FFFFFF;
            int c = ret[j] & 0x00FFFFFF;
            double l = levels[i];
            if (c <= l && s > l || c <= l && o > l || c <= l && n > l || c <= l && w >
l) {
               ret[j] |= 0xFF000000;
            }
         }
      }
      k = -1;
      while (++k < ret.length) {
         if (ret[k] < 0) {
            if (k < ret.length / 2) {
               ret[k] = 0xFFFF0000;
            } else {
```

```
                ret[k] = 0xFF0000FF;
                ret[k - length] &= 0xFFFF0000;
                    ret[k - length] |= 0xFF0000FF;
                }
                continue;
            }
            if (ret[k] > 511) {
                ret[k] -= 511;
                ret[k] <<= 16;
                ret[k] |= 0xFF00FFFF;
            } else if (ret[k] > 255) {
                ret[k] -= 0xFF;
                ret[k] |= 0xFF00FF00;
            } else {
                ret[k] <<= 8;
                ret[k] |= 0xFF000000;
            }
        }
        seller1.setPrice(savePrice1);
        seller2.setPrice(savePrice2);
        ret[ret.length - 1] = width;
        return ret;
    }

    public void calculateCurrentPosition() {
        double p1 = 0; //first  seller profit
        double p2 = 0; //second seller profit
        int ci = 0; // index of array p elements
        for (int j = 0; j < gameFieldSizes.height; j++) {
            for (int i = 0; i < gameFieldSizes.width; i++, ci++) {
```

```
            if (!gameField.belong(i, j)) {
                p[ci] = 0xFFCCCCCC;
                continue;
            }
            double s1 = metric.distance(new double[]{seller1.getPosition().x - i, seller1.getPosition().y - j}) * unitDistanceCost;
            double s2 = metric.distance(new double[]{seller2.getPosition().x - i, seller2.getPosition().y - j}) * unitDistanceCost;
            if (s1 + seller1.getPrice() > s2 + seller2.getPrice() + Math.min(seller1.getMinCoin(), seller2.getMinCoin()) * 0.001) {
                p[ci] = 0xFFDDDDFF;
                p2 += seller2.getPrice();
            } else if (s1 + seller1.getPrice() < s2 + seller2.getPrice() - Math.min(seller1.getMinCoin(), seller2.getMinCoin()) * 0.001) {
                p[ci] = 0xFFEEDDDD;
                p1 += seller1.getPrice();
            } else
            {
                p[ci] = 0xFFD4DCD0;
                p1 += seller1.getPrice() / 2;
                p2 += seller2.getPrice() / 2;
            }
        }
    }
    p[seller1.getPosition().y * gameFieldSizes.width + seller1.getPosition().x] = 0xFFA80000; //seller1 point
    p[seller2.getPosition().y * gameFieldSizes.width + seller2.getPosition().x] = 0xFF0000CC; //saller2 point
    seller1.setProfit(p1);
    seller2.setProfit(p2);
```

```
    }

    public void optimizeAllPrices(int policy, int cycles) throws NumberFormatException {
        double minCoin = Math.min(seller1.getMinCoin(), seller2.getMinCoin()) * 0.9;
        calculateCurrentPosition();
        double price10 = seller1.getPrice();
        double price20 = seller2.getPrice();
        double price11 = price10;
        double price21 = price20;
        int count = 0;
        do {
            optimizePrice(seller1, policy);
            optimizePrice(seller2, policy);
            price10 = price11;
            price20 = price21;
            price11 = seller1.getPrice();
            price21 = seller2.getPrice();
        } while ((Math.abs(price10 - price11) > minCoin || Math.abs(price20 - price21) > minCoin) && count++ < cycles);


    }

    public void optimizePrice(SellerGamer seller, int policy) {
        switch (policy) {
            case 0:
                optimizePrice(seller);
                break;
```

```
            case 1:
                optimizePriceLocal(seller);
                break;
            default:
                throw new AssertionError();
        }
    }

    public void optimizePriceBoth(SellerGamer seller1, SellerGamer seller2, int cycles) {
        double minCoin = Math.min(seller1.getMinCoin(), seller2.getMinCoin()) * 0.9;
        calculateCurrentPosition();
        double profit10 = seller1.getProfit();
        double profit20 = seller2.getProfit();
        for(int count = 0;count < cycles; count++){
            optimizePriceB(seller1, seller2);
            if (seller1.getR()) {
                if (seller1.getProfit()>profit10) {
                    seller1.incRisk(1);
                }else{
                if (seller1.getProfit()<profit10) {seller1.decRisk(1);}
                }
            }
            if (seller2.getR()) {
                if (seller2.getProfit()>profit20) {
                    seller2.incRisk(1);
                }else{
                if (seller2.getProfit()<profit20) {seller2.decRisk(1);}
                }
```

```
        }
      }
   }

   public void optimizePriceB(SellerGamer seller, SellerGamer seller2) throws NumberFormatException {
      Random rand = new Random();
      int riskNum = rand.nextInt(10)+1;
      boolean randBool = rand.nextBoolean();
      double c0 = seller.getPrice();
         double maxProfit = seller.getProfit();
         double cExtr = c0;
         double cBad = c0;
         for (double cc = c0 - seller.getMinCoin(); cc <= c0 + seller.getMinCoin(); cc += seller.getMinCoin()) {
            seller.setPrice(cc);
            calculateCurrentPosition();
            if (seller.getProfit() > maxProfit) {
               maxProfit = seller.getProfit();
               cExtr = cc;
            }
         }
         if (cExtr==c0){if(randBool){cBad=c0-seller.getMinCoin();}else{cBad=c0+seller.getMinCoin();}}
         if (cExtr==(c0-seller.getMinCoin())){if(randBool){cBad=c0;}else{cBad=c0+seller.getMinCoin();}}
         if (cExtr==(c0+seller.getMinCoin())){if(randBool){cBad=c0-seller.getMinCoin();}else{cBad=c0;}}
         seller.setPrice(c0);
```

```
            calculateCurrentPosition();
        c0 = seller2.getPrice();
        maxProfit = seller2.getProfit();
        double cExtr2 = c0;
        double cBad2 = c0;
        for (double cc = c0 - seller2.getMinCoin(); cc <= c0 +
seller2.getMinCoin(); cc += seller2.getMinCoin()) {
            seller2.setPrice(cc);
            calculateCurrentPosition();
            if (seller2.getProfit() > maxProfit) {
                maxProfit = seller2.getProfit();
                cExtr2 = cc;
            }else{cBad2 = cc;}
        }
        if                                    (cExtr2==c0){if(randBool){cBad2=c0-
seller.getMinCoin();}else{cBad2=c0+seller.getMinCoin();}}
        if                                                    (cExtr2==(c0-
seller.getMinCoin())){if(randBool){cBad2=c0;}else{cBad2=c0+seller.getMinCo
in();}}
        if              (cExtr2==(c0+seller.getMinCoin())){if(randBool){cBad2=c0-
seller.getMinCoin();}else{cBad2=c0;}}
      if (seller.getRisk()>=riskNum){
        seller.setR(true);
        seller.setPrice(cBad);
      }else{
        seller.setR(false);
        seller.setPrice(cExtr);
      }
        riskNum = rand.nextInt(10)+1;
```

```
        if (seller2.getRisk()>=riskNum){
            seller2.setR(true);
            seller2.setPrice(cBad2);
        }else{
            seller2.setR(false);
            seller2.setPrice(cExtr2);
        }
            calculateCurrentPosition();
    }

    public void optimizePriceLocal(SellerGamer seller) throws
NumberFormatException {
        double c0 = seller.getPrice();
        double cExtr = c0;
        double maxProfit = seller.getProfit();
        double profit;
        seller.setPrice(c0 - seller.getMinCoin());
        calculateCurrentPosition();
        profit = seller.getProfit();
        if(profit > maxProfit) {
            maxProfit = profit;
            cExtr = c0 - seller.getMinCoin();
        }
        seller.setPrice(c0 + seller.getMinCoin());
        calculateCurrentPosition();
        profit = seller.getProfit();
        if(profit > maxProfit) {
            cExtr = c0 + seller.getMinCoin();
        }
        seller.setPrice(cExtr);
```

```
        calculateCurrentPosition();
    }

    public void optimizePrice(SellerGamer seller) throws NumberFormatException
{
        double iniProfit;
        do {
            double c0 = seller.getPrice();
            double maxProfit = seller.getProfit();
            iniProfit = maxProfit;
            double cExtr = c0;
            for (double cc = c0 - seller.getPriceStep(); cc <= c0 + seller.getPriceStep();
cc += seller.getMinCoin()) {
                seller.setPrice(cc);
                calculateCurrentPosition();
                if (seller.getProfit() > maxProfit) {
                    maxProfit = seller.getProfit();
                    cExtr = cc;
                }
            }
            seller.setPrice(cExtr);
            calculateCurrentPosition();
        } while (iniProfit < seller.getProfit() - seller.getMinCoin() * 0.001);
    }
    public static void main(String[] args) {
    }
}
```