Министерство образования и науки Российской Федерации
МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(государственный университет)
ФАКУЛЬТЕТ УПРАВЛЕНИЯ И ПРИКЛАДНОЙ МАТЕМАТИКИ
КАФЕДРА ИНФОРМАТИКИ

Исследование некооперативных конфликтных игр

Выпускная квалификационная работа
студента 275 группы
Бойко Дмитрия Константиновича

Научный руководитель
Меньшиков И.С., к.ф.-м.н.

г. Долгопрудный
2016

Оглавление

Введение 3
Постановка задачи 4
Обзор существующих моделей 4
Детерминированная модель конфликта 5
Необходимые свойства модели 5
Описание выбранной модели 5
Симметричное равновесие 6
Компьютерные исследования 8
Программа для поиска всех симметрично-равновесных исходов 8
Применение подхода динамического программирования 9
Эффективность подхода динамического программирования 11
Вероятности выигрыша 14
Симметрично-равновесные траектории 14
Коэффициент раскулачивания 15
Эксперименты с участниками 18
Проведение игр 18
Коэффициент раскулачивания 18
Коэффициент инертности выбора 19
Время хода 20
Выбор траектории при отсутствии симметричного равновесия 21
Влияние пола участников 22
Игра (программное приложение) 23
Настройка приложения 23
Работа приложения 25
Выводы (Анализ результатов) 26
Список литературы 27
Приложение 1 28
Game.java 28
GameState.java 31
MultiArray.java 33
Приложение 2 35
Игры с 3 участниками 35
Игры с 4 участниками 36
Игры с 5 участниками 37

Игры с 6 участниками.....38
Игры с 7 участниками.....39
Игры с 8 участниками.....40
Игры с 1 стартовой жизнью у каждого из участников.....41

Введение

Данная работа является исследованием в теории игр, экспериментальной экономики и социальной психологии. В ней рассматривается проблема социального взаимодействия в условиях конфликта и агрессии всех участников такового.

Математическая проблема рассмотрения конфликтов трёх участников возникла ещё в 1946 году. В соответствующей линейке работ она рассматривалась как некоторая вероятностная динамическая модель, т.е. если участник и совершал своё действие (смертельный выстрел из пистолета в другого участника), то он делал это с заведомой вероятностью успеха. Так выводились особые правила и стратегии поведения участников в определённых условиях. В данной работе рассматривается ситуация, когда «выстрел» совершается со стопроцентным успехом, но одной «пули» обычно не хватает для «полного поражения».

Также, похожие проблемы замечены в ряде салонных игр, а именно, настольных игр англо-американской школы, берущей своё начало в 50-ых годах XX века. В данных играх возникает несколько специфичных эффектов, таких как “Kingmaking” [1] и “Bashing the leader” [2]. Они вызывают немалые дискуссии среди поклонников и разработчиков хобби-игр. С точки зрения немецкой школы настольных игр такие эффекты лишают игровой процесс осмысленности и содержательности, так как “Kingmaking” и “Bashing the leader” по своему определению не позволяют честно и непредвзято выявить более опытного игрока. (Woods, 2012)

Автор данной работы наблюдал такие эффекты в следующих настольных играх: “Nightfall” [3], “Star Realms” [4].

Нередко и в политической, и в экономической сфере конкуренция приобретает деструктивный, конфликтный характер.

Эти и другие наблюдения вызвали интерес в изучении конфликтного процесса. Идея была предложена Меньшикову И.С., который одобрил её, предложил сначала провести теоретические исследования, а потом провести опыты в Лаборатории экспериментальной экономики МФТИ, так как данное исследование полностью входит в круг интересов лаборатории.

---

[1] Явление, когда игрок без шансов на победу результатом своих действий влияет на то, кто из его оппонентов станет победителем. Такого игрока называют «kingmaker», что переводится на русский язык как «влиятельное лицо, от которого зависит назначение на высокий пост».

[2] «Битьё лидера». Явление, когда все противники лидирующего игрока совершают действия, направленные больше на уменьшение возможности победы лидера, чем на увеличение возможности своей собственной победы.

[3] https://boardgamegeek.com/boardgame/88408/nightfall

[4] https://boardgamegeek.com/boardgame/147020/star-realms

Постановка задачи

Цель работы – исследование модели конфликта трёх и более участников и создание программного обеспечения для этой цели. Более конкретно – это написание программы с графическим интерфейсом для проведения экспериментов в лаборатории, создание анализатора полученных данных в целях интерпретации поведения участников и разработка методов теоретического анализа.

У данной работы чистых аналогов нет, хотя ряд эффектов, возникающих в ходе игры, имеет интуитивно-понятную подоплёку, и потому должен быть изучен.

Задача заключается в том, чтобы сначала предложить достаточно элегантную и интуитивно-понятную модель конфликта, затем получить по ней теоретические, и в конце провести ряд экспериментов с аудиторией подопытных и аналитически обработать результаты.

Обзор существующих моделей

Дуэль трёх лиц.

В данной модели конфликта трёх лиц, являющейся первой рассматриваемой в математике, три участника передают по очереди пистолет и стреляют из него в своих оппонентов. Каждый игрок имеет определённую точность выстрела, т.е. шанс попасть в выбранного противника: точности α, β и γ стрелков A, B, C соответственно ($0 < \alpha < \beta < \gamma = 1$). Игрок C считается сильнейшим, но, несмотря на это, в результате изучения этой модели было показано, что игрок C почти всегда проигрывает. (Kinnaird, 1946)

Детерминированная модель конфликта

В данной главе рассматривается игровая модель конфликта, которая была выбрана для данного исследования. Даётся её математическое описание и подход к поиску равновесных исходов.

Необходимые свойства модели

Задачей было рассмотреть поведение людей в конфликтной ситуации. Для этого следовало построить подходящую модель, обладающую следующими свойствами:

1) Динамика. Это должна быть не одноразовая игра, а игра, в которой результаты предыдущих ходов влияют на последующие стратегии участников.
2) Полная информация. Игрок основывает решение на всех предыдущих ходах.
3) Отсутствие ослабления игрока, пока он в игре. Игрок в свой ход имеет ровно один выстрел со 100% вероятность.
4) Отсутствие «игральной кости». В игре не должен присутствовать фактор случайности, чтобы не повторять эксперимент больше, чем это следовало бы делать из-за различных реализаций случайных величин.
5) Интуитивно-понятные правила, но не интуитивно-понятная стратегия, приводящая к победе.
6) Выбывание игроков. Игрок, достигнувший определённого состояния, считается проигравшим и больше не влияет на игровой процесс.

Описание выбранной модели

Была принята следующая модель игры:

1) N – число игроков, натуральное число большее либо равное двум.
2) $\{1, 2, \dots N\}$ – набор индексов всех игроков.
3) l_i – стартовое число жизней игрока с индексом i, натуральное число большее либо равное единице.
4) $l = (l_1, l_2, \dots, l_n)$ – набор жизней всех игроков.
5) W – число игроков-победителей, натуральное число, $N > W \geq 1$.
6) k – индекс игрока, который должен сделать ход.
7) Игроки ходят по очереди:
$$(k, k+1, k+2, \dots, N-1, N, 1, 2, \dots, k-1)$$
8) Если игрок i имеет ноль жизней ($l_i = 0$), то он пропускает свои ходы.
9) Если игрок i ещё жив ($l_i > 0$), и он должен ходить ($i = k$), то игрок i обязан выбрать одного из своих ещё живых противников ($\forall j \neq k,\ l_j > 0$) и произвести «выстрел» ($l_j \coloneqq l_j - 1$).
10) Игра заканчивается сразу, когда после очередного хода игрока, оказывается ровно W победителей:
$$W = \sum_1^N I\{l_i > 0\},$$
I – индикаторная функция.
11) После окончания все выжившие игроки признаются победителями, т.е. их выигрыш равен единице, а у всех других игроков выигрыш равен нулю.
$$u_i = I\{l_i > 0\},$$
u_i – выигрыш игрока i.

В теории игр обычно принято записывать динамические игры в виде дерева, но из-за специфичного характера описываемой модели формальное определение будет дано рекуррентно, через подыгры. (Aumann & Hart, 1992)

$$G_{l,k}^{N,W} = \begin{cases} \Lambda, & W = \sum_i I\{l_i > 0\}, \\ \left\{\forall i \neq k \mid S(i,l) \times G_{L(i,l),K(k,L(i,l))}^{N,W}\right\}, & \text{иначе,} \end{cases}$$

$$S(i,l) = \begin{cases} i, & l_i > 0, \\ \emptyset, & \text{иначе,} \end{cases}$$

$$L(i,l) = (l_1, l_2, \dots, l_i - 1, \dots, l_N),$$

$$K(k,l) = \left\{k + \min \left(\text{Argmax}_{p \in \{1,\dots,N\}} I\left\{l_{(k+p) mod\ N} > 0\right\}\right)\right\} mod\ N,$$

где $G_{l,k}^{N,W}$ – множество всех траекторий игры с заданными параметрами (N, W, l, k), Λ – символ конца игры, $S(i,l)$ – допустимая стратегия игрока, $L(i,l)$ – множество жизней всех N игроков, где у игрока i отнята единица жизни, $K(k,l)$ – номер игрока, который должен ходить следующим.

Примечание. Для удобства будем считать, что $p \in \mathbb{N}: pn\ mod\ n = n$, а не как обычно принято $pn\ mod\ n = 0$.

Пример. Зададим параметры игры $N = 3, W = 1, l = (1,1,1), k = 1$. Получается, что

$$G_{(1,1,1),1}^{3,1} = \left\{\left(2, G_{(1,0,1),3}^{3,1}\right), \left(3, G_{(1,1,0),2}^{3,1}\right)\right\} =$$

$$= \left\{\left(2,1, G_{(0,0,1),3}^{3,1}\right), \left(3,1, G_{(0,1,0),2}^{3,1}\right)\right\} =$$

$$= \{(2,1,\Lambda), (3,1,\Lambda)\}.$$

Объясняя простым языком, в первом случае первый игрок выстрелил во второго, затем третий выстрелил в первого и стал победителем. Во втором случае первый игрок выбрал третьего, затем второй выбрал первого и победил в игре.

Также стоит отметить, что по заданным параметрам игры и последовательностей, записанной только с помощью номеров выбираемых игроков и символа конца игры, можно однозначно и без особого труда на каждом ходу восстановить номер игрока, совершающего этот ход.

Симметричное равновесие

Одним из возникающих вопросов является вероятности выигрыша каждого из игроков в конкретной версии модели конфликта. (Меньшиков, 2010) Например, очевидно, что в игре $G_{3,1}(\{1,1,1\},1)$ отсутствует траектория, в которой первый игрок становится победителем. Если считать, что этот игрок не предвзят и не имеет отношений вне игры ни со вторым, ни с третьим игроком, то первый игрок равновероятно выберет, в кого он будет стрелять. Получается, что вероятность победы второго и третьего игрока равна 0,5.

Предположение о симметричном равновесии:

Пусть игрок на своём ходу имеет набор стратегий $S = \{s_1, \dots, s_n\}$. $p(s)$ - вероятность победы игрока при выборе на данном ходу стратегии s. S^* - непустое множество, состоящие из всех таких элементов S, которые имеют вероятность выигрыша максимальную на всё множестве S, то есть $s \in S^* \Leftrightarrow s \in S, p(s) = \max_{q \in S} p(q)$. В таком случае игрок выбирают одну из стратегий во множестве S^*. Вероятность выбора любой стратегии из этого множества полагается равной $|S^*|^{-1}$.

Исходя из аксиомы симметричного равновесия, найдём вероятности выигрыша всех игроков для выбранной модели конфликта.

$$p\left(G_{l,k}^{N,W}\right) = \begin{cases} (I\{l_1 > 0\}, I\{l_2 > 0\}, \dots, I\{l_N > 0\}), & W = \sum_i I\{l_i > 0\}, \\ \dfrac{\sum_{q \in S^*} p\left(G_{L(q,l),K(k,L(q,l))}^{N,W}\right)}{|S^*|}, & \text{иначе,} \end{cases}$$

$$S^* = \underset{i \neq k,\ l_i > 0}{\operatorname{Argmax}}\, p_k\left(G_{L(i,l),K(k,L(i,l))}^{N,W}\right),$$

где $p\left(G_{l,k}^{N,W}\right) = \left(p_1\left(G_{l,k}^{N,W}\right), \dots, p_N\left(G_{l,k}^{N,W}\right)\right)$ – вероятности выигрыша всех игроков на множестве $G_{l,k}^{N,W}$, S^*- множество стратегий с максимально возможной вероятностью выигрыша.

И теперь уже можно найти все траектории при условии, что каждый из игроков старается максимизировать свои шансы на победы и непредвзято относится к своим противникам, то есть подчиняется аксиоме симметричного равновесия.

$$\gamma\left(G_{l,k}^{N,W}\right) = \begin{cases} \{\Lambda\}, & W = \sum_i I\{l_i > 0\}, \\ \left\{ \begin{array}{c} \forall i \neq k, \quad l_i > 0, \\ p_k\left(G_{l,k}^{N,W}\right) = p_k\left(G_{L(i,l),K(k,L(i,l))}^{N,W}\right) \\ \mid s(i,l) \times \gamma\left(G_{L(i,l),K(k,L(i,l))}^{N,W}\right) \end{array} \right\}, & \text{иначе,} \end{cases}$$

где $\gamma\left(G_{l,k}^{N,W}\right)$ – множество всех симметрично-равновесных траекторий на множестве $G_{l,k}^{N,W}$.

Компьютерные исследования

Исследовать аналитически предложенную модель конфликта достаточно сложно. Поэтому было решено проделать это с помощью программных средств.

Программа для поиска всех симметрично-равновесных исходов

Сначала была написана программа, которая искала все симметрично-равновесные исходы с помощью рекурсии.

На входе программы строка из натуральных чисел через пробел: $N, W, k, l_1, l_2, \dots, l_N$

На выходе программа выдаёт:

1. Первая строчка содержит вероятности выигрыша всех игроков в формате точки с плавающей запятой через пробел.
2. Вторая и каждая последующая строчка содержит одну из найденных симметрично-равновесных траекторий. Траектории выводятся в виде последовательности номеров выбираемых игроков, записанных через пробел.

Пример:

Входные данные	Выходные данные
3 1 1 1 1 1	0 0,5 0,5 2 1 3 1

Опишем, что делает программа:

1) Создаётся *P* - многомерный массив массивов с плавающей точкой размерности N. Число измерений $P - N$, длина массива в i-ом измерении равна $l_i + 1$. Программный код такого массива содержится в листинге MultiArray.java (см. Приложение 1).
2) Вызывается функция поиска вероятностей выигрыша *FindProbability(n, w, k, l).* В качестве аргументов передаются входные данные N, W, k и массив $l = \{l_1, l_2, \dots, l_N\}$. Функция делает следующее:
   a. Если в игре ровно *w* игроков с ненулевым числом жизней, то для *i* от 1 до *n*:
      i. Если *l[i]>0*, то *P[l[1], l[2], …, l[n]][i]:=1.*
      ii. Если *l[i]=0*, то *P[l[1], l[2], …, l[n]][i]:=0.*

   Иначе происходит следующее. Для *i* от *k*+1 до *n*, а затем от *1* до *k*:
      i. Если игрок *i* ещё жив, то *l[i]:=l[i]-1*, ищется *k** - следующий по порядку хода игрок с ненулевым числом жизней и вызывается функция *FindProbability(n, w, k*, l).* Затем *l[i]:=l[i]+1.*
   b. Из всех возможных ходов ищутся все лучшие варианты хода (массив *BestMoves*), т.е. с максимальной вероятностью выигрыша, при помощи результатов из массива *P*.
   c. Для *i* от 1 до *n* и для всех *j* из массива *BestMoves* происходит вычисление вероятности победы на данном шаге игры*:*
      i. *P[l[1], l[2], …, l[n]][i]:= P[l[1], l[2], …, l[n]][i] +*
      + P[l[1], l[2], …l[j]-1,…, l[n]][i] / length(BestMoves).
   d. Возврат из функции.

3) Выводятся все значения *P[l[1], l[2], …, l[n]]* – это и есть вероятности выигрыша игроков в заданной игре.
4) Вызывается функция поиска всех симметрично-равновесных траекторий *PrintPaths(n, w, k, l).* В качестве аргументов передаются входные данные N, W, k и массив $l = \{l_1, l_2, \dots, l_N\}$. Заранее заводится специальный стек *T*, в него будут записываться ходы. Функция делает следующее:
    a. Если в игре ровно *w* игроков с ненулевым числом жизней, то выводится стек *T*, начиная с первого положенного в него элемента и заканчивая последним. Это и будет симметрично-равновесная траектория.
    Иначе происходит следующее. Для *i* от *k*+1 до *n*, а затем от *1* до *k-1*:
        i. Если игрок *i* ещё жив, и если выбор этого игрока сулит наибольшую вероятность победы в игре, т.е. *P[l[1], l[2], …, l[n]][k]= P[l[1], l[2], …l[i]-1,…, l[n]][k]*, то происходит следующее: *l[i]:=l[i]-1*, ищется *k** - следующий по порядку хода игрок с ненулевым числом жизней, *T.push(i)* вызывается функция *PrintPaths(n, w, k*, l).*
        Затем *l[i]:=l[i]+1, T.pop.*
    b. Возврат из функции.
5) Программа успешно завершает работу.

Листинг кода описанной программы не прилагается, написать её представляется вполне возможным на популярных языках программирования C, Java, Python и т.д. Дело в том, что рекурсивные решения таких задач требуют очень большого времени исполнения даже на довольно малых входных данных. Это будет показано в следующем разделе при сравнении времени исполнения такого варианта программы и улучшенного.

Применение подхода динамического программирования

Динамическое программирование - способ решения сложных задач путём разбиения их на более простые подзадачи. Очевидно, что наша задача в совершенстве подходит для такого подхода. Рассмотрим траектории $G^{3,1}_{\{2,2,2\},1}$. Приведём следующую табличку (см. Таблица 1. Все траектории игры: 3 игрока, 2 жизни у каждого, до одного победителя.). В первой колонке содержится порядковый номер траектории. В строке заголовка записаны номера ходов. В строках отображены траектории с помощью количества жизней игроков, имеющихся на определённом ходу. Для удобства полужирным начертанием отмечено число жизней игрока, совершающий ход.

	1	2	3	4	5	6	7
1	***2***,2,2	2,***1***,2	2,1,***1***	***1***,1,1	1,0,***1***	0,0,1	-
2	***2***,2,2	2,***1***,2	2,1,***1***	***1***,1,1	1,***1***,0	0,1,0	-
3	***2***,2,2	2,***1***,2	2,1,***1***	***2***,0,1	2,0,0	-	-
4	***2***,2,2	2,***1***,2	1,1,***2***	0,***1***,2	0,1,***1***	0,0,1	-
5	***2***,2,2	2,***1***,2	1,1,***2***	***1***,0,2	1,0,***1***	0,0,1	-
6	***2***,2,2	2,***2***,1	***2***,2,0	2,***1***,0	1,***1***,0	1,0,0	-
7	***2***,2,2	2,***2***,1	1,2,***1***	0,***2***,1	0,2,0	-	-
8	***2***,2,2	2,***2***,1	1,2,***1***	***1***,1,1	1,0,***1***	0,0,1	-
9	***2***,2,2	2,***2***,1	1,2,***1***	***1***,1,1	1,***1***,0	0,1,0	

Таблица 1. Все траектории игры: 3 игрока, 2 жизни у каждого, до одного победителя.

При рассматривании таблицы появляются две идеи:

1. Заметим, что игра в состояние (***1***,*1,1)* попадала по двум разным траекториям: строка 1 и 8. Это значит, что мы уже знали результат после вычисление траектории 1, и при вычислении траектории 8 могли бы им воспользоваться.
2. Также стоит отметить, что в колонке 5 в строках 1, 2, 4, 5, 6, 8, и 9 получились состояния игры подобные друг другу с той лишь разницей, что в них уже один игрок имел жизнь со значением ноль, и право хода было у разных игроков. Можно считать, что игра 3 игроков преобразовалось в игру 2 игроков $G^{2,1}_{\{1,1\},1}$, выполнить грамотную новую нумеровку игроков и взять результаты $p(G^{2,1}_{\{1,1\},1})$.

В нашей программе функцию *FindProbability(n, w, k, l)* для игры $G^{3,1}_{\{2,2,2\},1}$ придётся вызвать 30 раз. Если же применить первую идею, то число рекурсивных вызовов снизится до 25, а если ещё и вторую, то - до 11.

Потому наша программа должна претерпеть следующие изменения:

1. Создаётся не один массив *P*, а *N-2* массивов ему подобных. Выглядит следующим образом. Для *i* от *2* до *N:*
   a. Массив *P[i]* есть многомерный массив массивов с плавающей точкой размерности *i*. Число измерений *P[i]* – i, длина массива в любом измерении равна $\max_{i\in\{1,\dots,N\}} l_i + 1$.
2. Функция поиска вероятностей выигрыша *FindProbability(n, w, k, l)* теперь работает так. В качестве аргументов передаются входные данные N, W, k и массив $l = \{l_1, l_2, \dots, l_N\}$. Функция делает следующее (изменения отмечены жирным):
   a. Если в игре ровно *w* игроков с ненулевым числом жизней, то для *i* от 1 до *n*:
      i. Если *l[i]>0,* то ***P[n]****[l[1], l[2], …, l[n]][i]:=1.*
      ii. Если *l[i]=0,* то ***P[n]****[l[1], l[2], …, l[n]][i]:=0.*

   Иначе происходит следующее. Для *i* от *k*+1 до *n,* а затем от *1* до *k*:
      ii. Если игрок *i* ещё жив, то *l[i]:=l[i]-1,* ищется *k** - следующий по порядку хода игрок с ненулевым числом жизней, ***n* - число игроков с количеством жизни больше нулю.***
      Если n* стало меньше, чем n, то создаётся l* - массив l без значения игрока i. и вызывается функция FindProbability(n*, w, k*, l*).
      Если *n** осталось таким же, то вызывается функция *FindProbability(n, w, k*, l).*
      Затем *l[i]:=l[i]+1.*

   Пункты *b* и *c* меняются аналогично *a*. Теперь также приходится перенумеровывать игроков при их выбывании и восстанавливать нумеровку при получении результатов из *P[n*]*. Задаётся правило следующего вида, исходя из которого, игрок совершающий ход (ход уже после хода с выбыванием) становится первым игроком в подыгре:

   Если число всех жизней в данный момент игры делится на число её участников с остатком 0, то считается, что ходит первый игрок, если 1, то - последний, если 2, то – предпоследний и т.д. Формула порядка хода:

$$k = N + 1 - \left(\sum_{i=1}^{N} l_i\right) \bmod N$$

Примечание. Для удобства. $p \in \mathbb{N}: pn \bmod n = n$.

3. Пункты 3 и 4 меняются аналогично оговоренному выше подходу.

Листинг программы, которая реализует только что описанное, приведён в Приложении 1.

Эффективность подхода динамического программирования

Рассмотрим серию игр 3 игроков, у которых одинаковое число стартовых жизней. Играется до одного победителя.

Число жизней	Рекурсия	Динамическое программирование	Ускорение
1	5	5	1,00E+00
2	30	25	1,20E+00
3	185	74	2,50E+00
4	1184	159	7,45E+00
5	7888	296	2,66E+01
6	54055	490	1,10E+02
7	378326	759	4,98E+02
8	2690705	1106	2,43E+03
9	19379351	1551	1,25E+04
10	141000646	2095	6,73E+04
11	1034498857	2760	3,75E+05
12	7680361587	3545	2,17E+06

Таблица 2. Количество вызовов функции FindProbability

В таблице 2 в столбцах «Рекурсия» и «Динамическое программирование» записано число вызовов функции FindProbability при том или ином подходе. В столбце «Ускорение» отображено отношение, показывающее во сколько раз больше вызовов при использовании

рекурсии, чем при использовании динамического программирования.

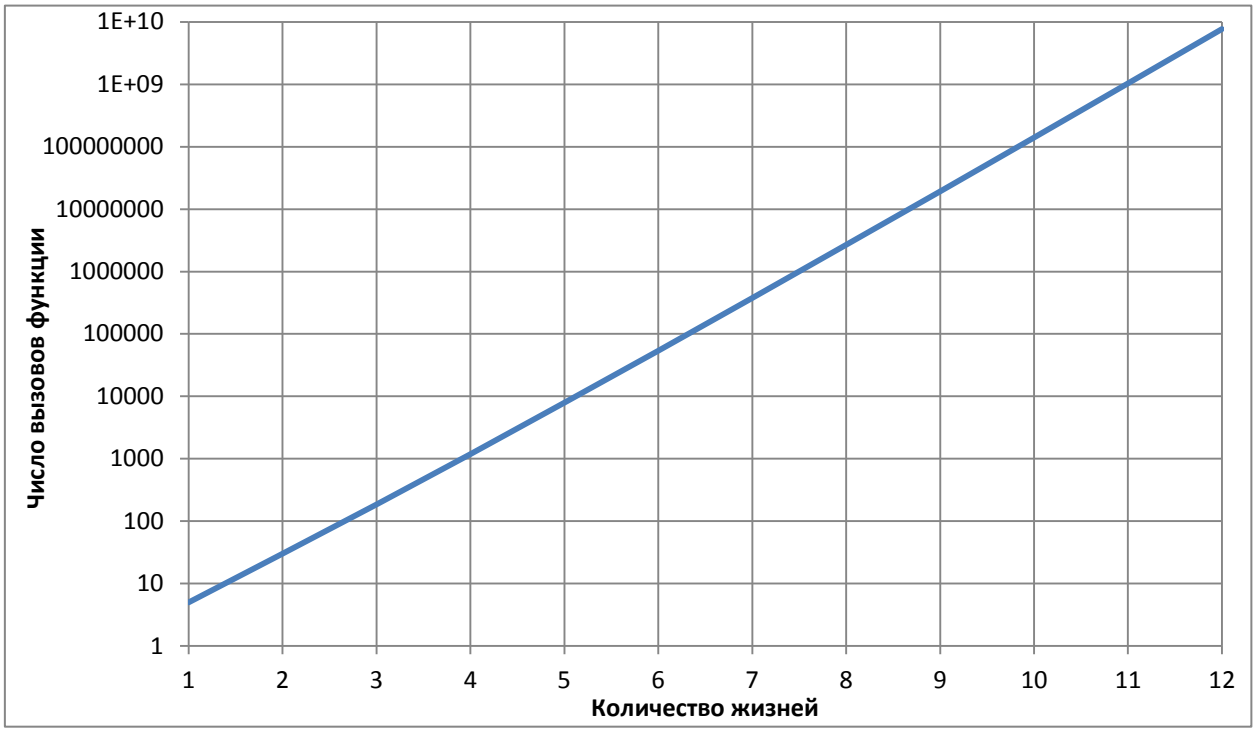


Рисунок 1. Результаты рекурсивного подхода.

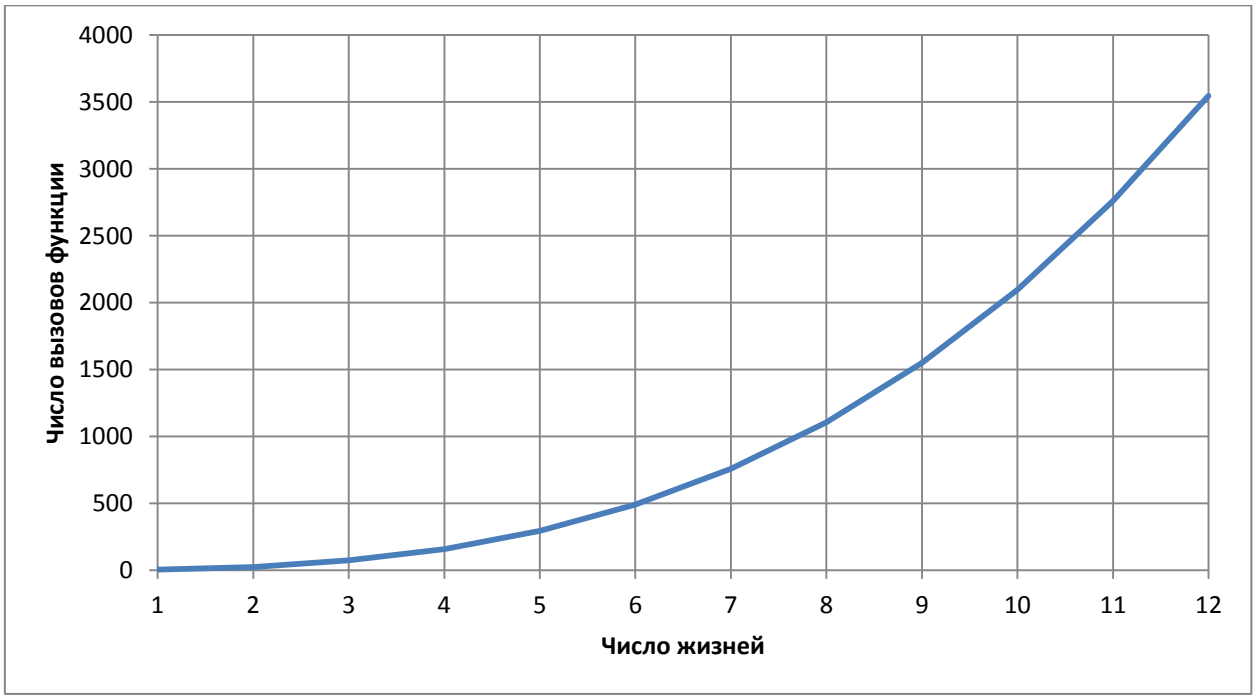


Рисунок 2. Результаты подхода динамического программирования.

Рисунок 1 показывает, как быстро возрастает число вызовов при использовании рекурсивного варианта программы, а Рисунок 2 – при использовании подхода динамического программирования. Стоит отметить, что вертикальная ось на Рисунке 1 специально сделана логарифмической.

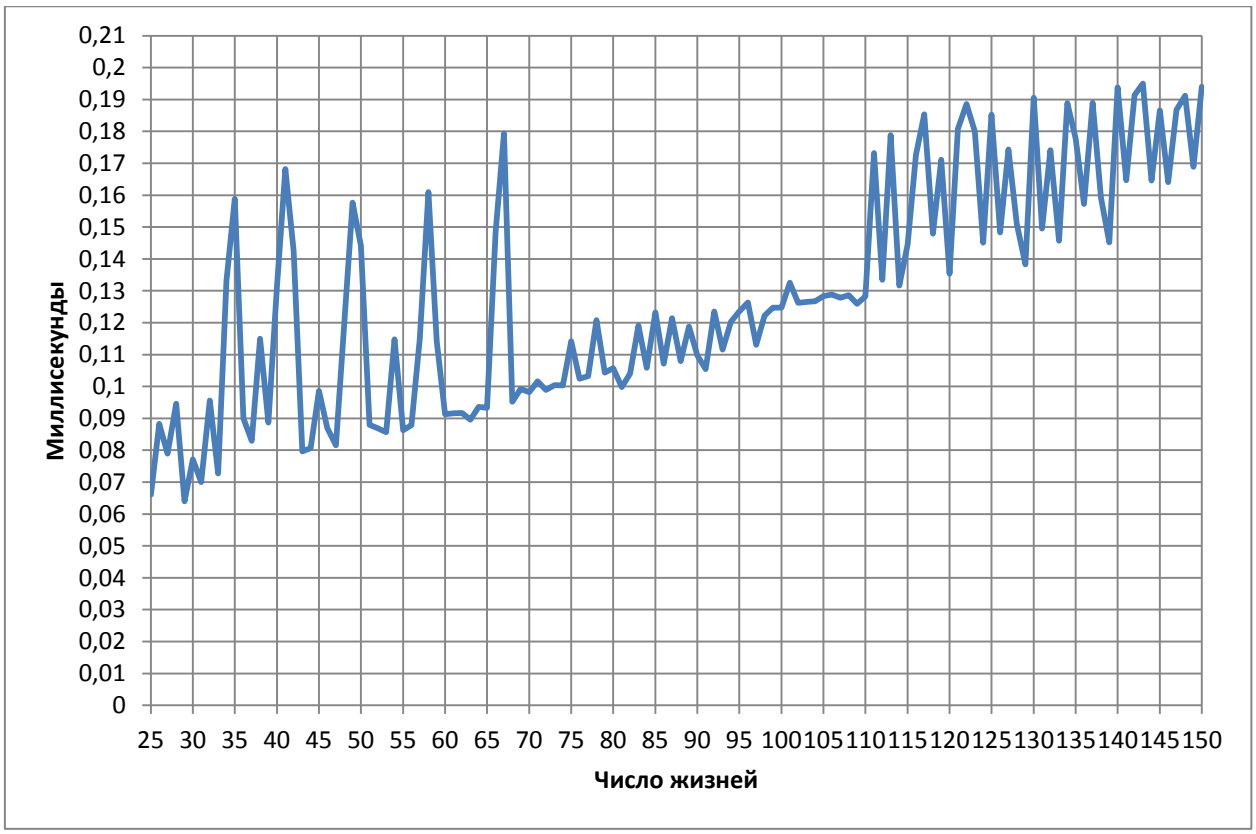


Рисунок 3. Среднее время выполнения функции (динамическое программирование).

На Рисунке 3 отображено, сколько миллисекунд занимает работа функции FindProbability, не считая рекурсивные вызовы внутри её, в зависимости от количества стартовых жизней у игроков (от 25 до 150) при использовании динамического программирования. Время работы вычисляется, как отношения время работы программы к числу вызовов функции. Средние значения равны $0{,}156 \pm 0{,}062$ миллисекунды.

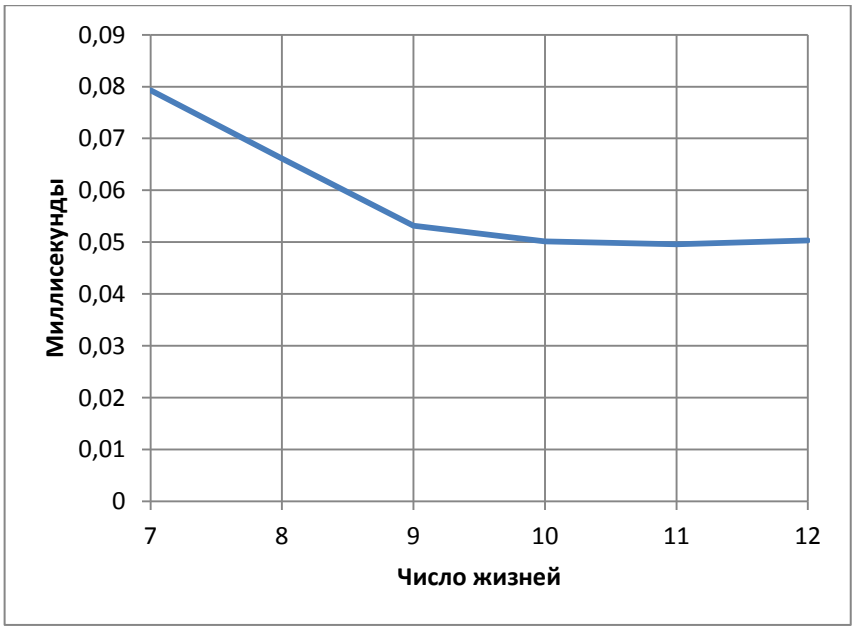


Рисунок 4. Среднее время выполнения функции (рекурсия)

На Рисунке 4 отображено, сколько миллисекунд занимает работа функции FindProbability, не считая рекурсивные вызовы внутри её, в зависимости от количества стартовых жизней у игроков (от 7 до 12) при использовании явной рекурсии. Средние значения равны $0{,}058 \pm 0{,}010$ миллисекунд.

Из-за того, что функция FindProbability делает больше операций (проверок условий и извлечений результатов из массивов) в варианте с динамическим программированием, её среднее время выполнения почти в три раза выше, чем у варианта с рекурсией. Но это несравнимо с тем, что число вызовов возрастает каждый раз в среднем в 6,86 раз (на отрезке от 1 до 12 жизней на старте).

Вероятности выигрыша

В приложении 2 даются таблицы и графики с полученными теоретическими вероятностями выигрыша игроков. Во всех случаях игра продолжается до одного победителя и у всех игроков одинаковое количество стартовой жизни.

Таблицы имеют следующий формат. Первый столбец – число жизней на старте, первая строка – номер игрока. На пересечении столбца и строки располагается вероятность победы игрока.

Графики по вертикальной оси показывают вероятность победы, по горизонтальной оси – число жизней на старте. Игроки обозначаются линиями разных цветов.

Отдельно стоит отметить такой факт, что в подавляющем большинстве игр существует игрок, у которого вероятность победы равна нулю. С одной стороны, это может говорить о несовершенстве модели, с другой, наоборот, о том, что в модели заранее существует игрок-“Kingmaker”, который определяет больше судьбу других, чем свою.

Отдельно представлены в Приложении 2 графики игр с 1 жизнью у всех игроков на старте и числом игроков от 5 до 27. Графики по вертикальной оси показывают вероятность победы, по горизонтальной оси – номер игрока. Количество игроков обозначается линиями разных цветов.

Удивительный факт заключается в том, что при росте количества игроков линия на графиках приближается к синусоиде.

Симметрично-равновесные траектории

На таблице 3 представлено количество всех симметрично-равновесных траекторий. Первый столбец – число игроков, первая строка – количество жизней на старте. На пересечении столбца и строки располагается число траекторий.

	1	2	3	4	5	6
3	2	4	7	51	121	980
4	6	8	56	260	1146	1053
5	24	84	390	1525	26158	1470976
6	120	992	1170	16150	49804	1724168

Таблица 3. Количество всех симметрично-равновесных траекторий

На рисунке 5 изображен график роста числа всех симметрично-равновесных траекторий от числа жизней в начале игры. Вертикальная ось сделана логарифмической. По данному графику видно, как экспоненциально быстро растёт количество траекторий. При том в играх 5 и 6 участников с 6 жизнями траекторий уже больше миллиона. Что само собой показывает, как сложно изучить такую задачу и понять, что делать, чтобы победить в игре. Поэтому требуется предложить какой-то особый подход для изучения поведения игроков.

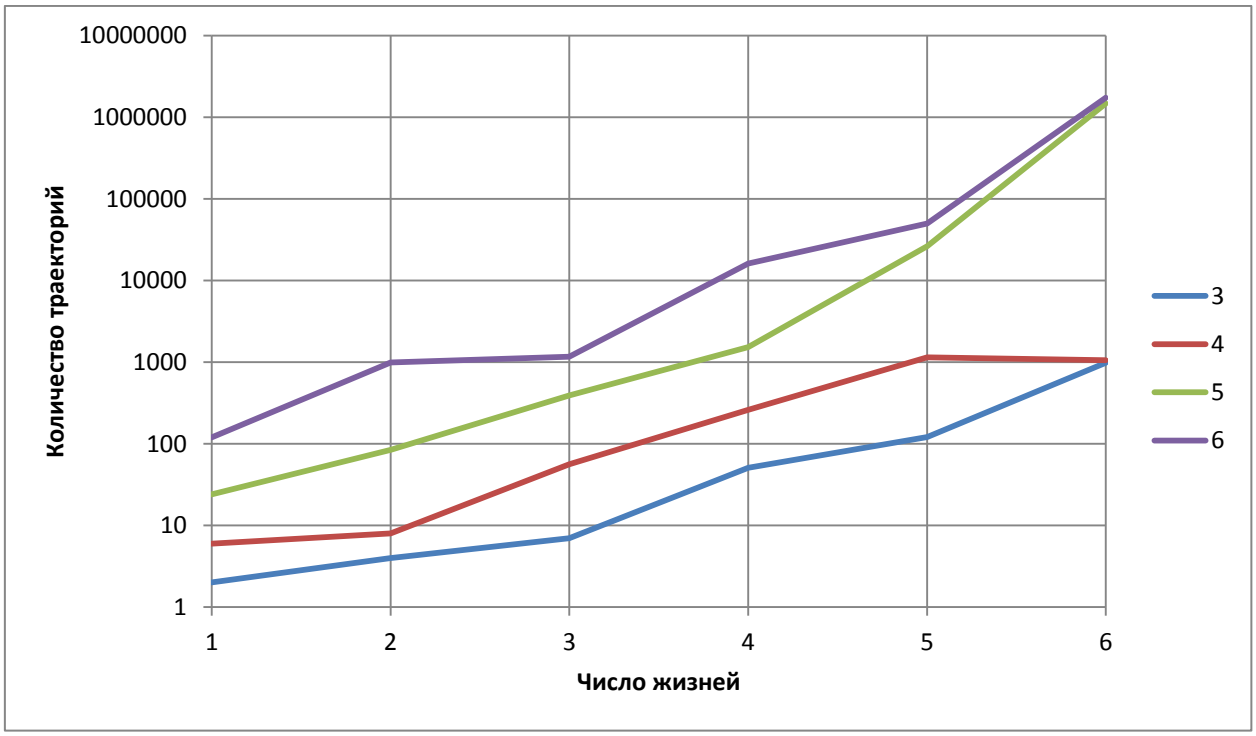


Рисунок 5. Рост числа траекторий от числа жизней.

Коэффициент раскулачивания

Раскулачивание (также “ Bashing the leader”, «битьё лидера») – явление, когда игроков с большим числом жизней на данный момент начинают активней выбирать в качестве цели, чем игроков с меньшим числом жизней. (Woods, 2012)

Коэффициент раскулачивания – число, отображающее выбор игрока на конкретном одном ходу траектории игры с точки зрения дифференциации числа жизней у его противников. Если участник k выбирает одного из оппонентов с $\min_{i \neq k} l_i$ на данном ходу, то его коэффициент равен нулю. Если участник k выбирает одного из оппонентов с $\max_{i \neq k} l_i$ на данном ходу, то коэффициент раскулачивания равен единице. Формула такая:

$$\phi_{k,s}^t = \frac{l_{s^t}^t - \min\limits_{i \neq k} l_i^t}{\max\limits_{i \neq k} l_i^t - \min\limits_{i \neq k} l_i^t},$$

где $\phi_{k,s}$ – коэффициент раскулачивания (при выборе игроком k своей целью игрока s в момент хода с порядковым номером t), l_i^t - число жизней игрока i на момент хода с номером t.

Здесь стоит учитывать следующие соображения:

1. Траектория игры представляется в виде конечного количества ходов:
$$\tau = \left((k^1,\ s^1), \dots, (k^t,\ s^t)\right),$$
где $\left(k^t,\ s^t\right)$ – пара «ходивший игрок – выбранный игрок» на ходу с порядковым номером t.
2. Считается, что при достижении одним из игроков нуля жизней, он выбывает из игры, т.е. число игроков уменьшается (N^t), а число жизней этого игрока отсутствует во множестве жизней игроков l на всех последующих ходах после выбывания данного участника.

Таким образом, можно найти коэффициент раскулачивания игрока за всю партию:

$$LBF(i) = \frac{\sum_t \phi^t_{k^t,s^t}}{T},$$

где $LBF(i)$ – коэффициент раскулачивания игрока i, $T = |\tau|$ - количество ходов в конкретной траектории игры.

Также следует учесть то, что в конкретный момент игры лидеры и отстающие могут насчитывать разное количество, да и их количество жизней может по-разному градироваться. Поэтому предлагается учитывать это:

$$\phi^t_{k,s} = \frac{(l^t_{s^t} - \min\limits_{i \neq k} l^t_i) N^t_{\leq l^t_{s^t}}}{(\max\limits_{i \neq k} l^t_i - \min\limits_{i \neq k} l^t_i) N^t}$$

где N^t - число игроков в игре на момент хода t, $N^t_{\leq l^t_s}$ - число игроков в игре на момент хода t с числом жизней меньшим, либо равным l^t_s.

Примечание. В случае если данный игрок всегда на своём ходу имел ситуацию с отсутствием дифференциации противников по числу жизней, его коэффициент раскулачивания не определён.

На графике (см. Рисунок 6) представлены результаты игры 5 участников со 7 стартовыми жизнями (до одного победителя). По горизонтальной оси отложены интервалы от 0 до 1 коэффициента раскулачивания, а по вертикальной – процент ходов с таким коэффициентом. Данный график строился на основе всех равновесных исходов для игры, 24% - это выбор самого отстающего игрока, 46% - самого живого, а 30% - оставшиеся стратегии. Не было ни одного хода с уровнем BLF от 65% до 95% процентов. Такие стратегии можно считать слабо доминируемыми. (Меньшиков, 2010)

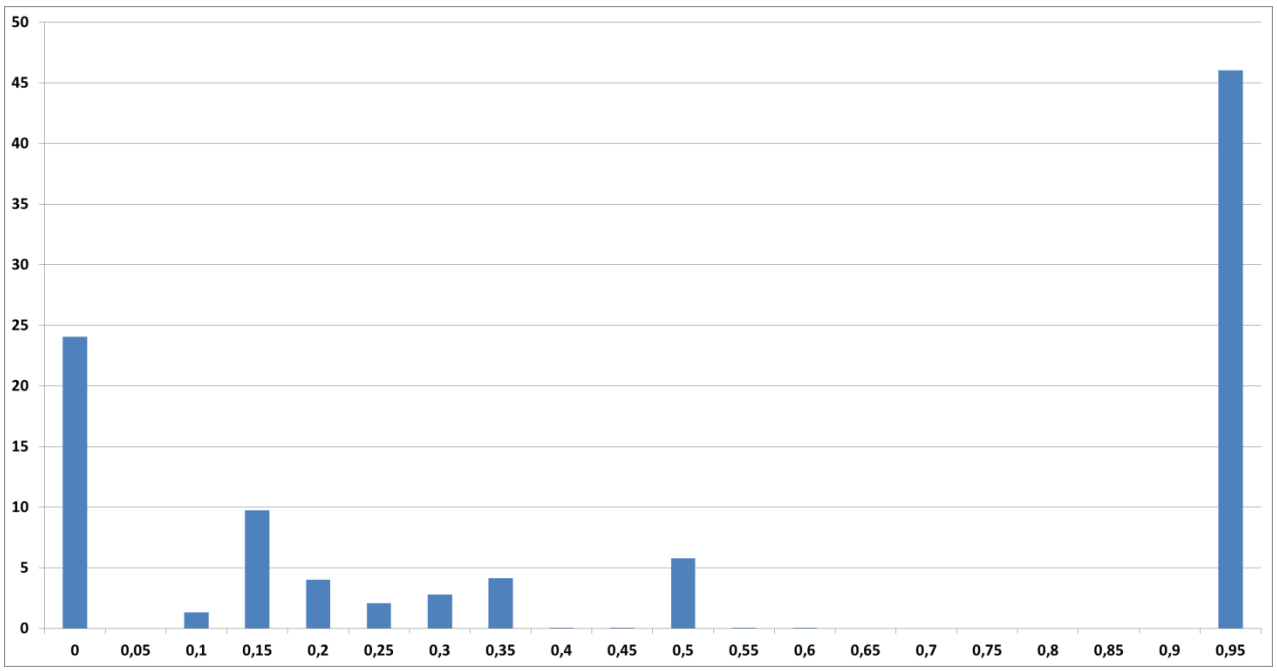


Рисунок 6. Распределение коэффициента раскулачивания ϕ.

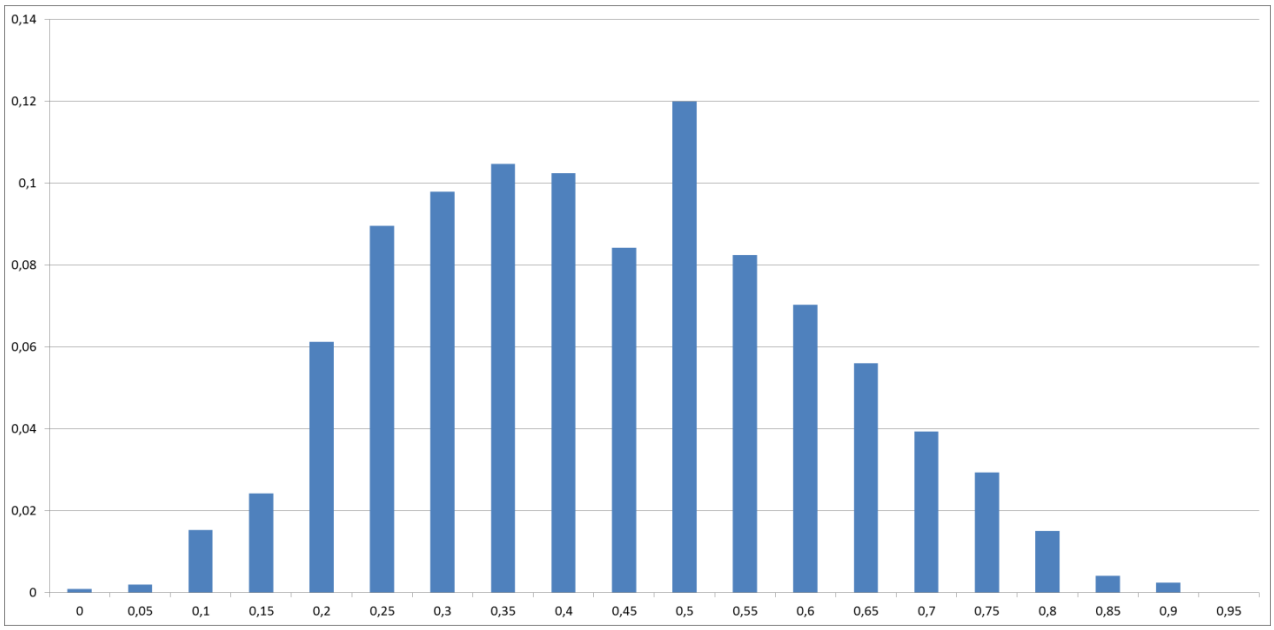


Рисунок 7. Распределение коэффициента раскулачивания игроков *LBF*.

На графике (см. Рисунок 7. Распределение) по горизонтальной оси отложены интервалы от 0 до 1 коэффициента раскулачивания игроков, а по вертикальной – процент таких коэффициентов. Данный график строился на основе всех равновесных исходов для игры. Заметим, что данный график отражает, что виртуальные игроки часто смешивают примерно 50 на 50 две основные стратегии: выбор самого сильного и выбор самого слабого противника. Среднее значение коэффициента раскулачивания игроков равно 43,6%, а стандартное отклонение 4,82%.

Эксперименты с участниками

Проведение игр

В Лаборатории Экспериментальной Экономики МФТИ было проведён ряд экспериментов с 11 студентами. Здесь следует отметить, что все участники эксперимента высказали свои мнения по эксперименту и данной модели игры: они, как и ожидалось, разделились примерно пополам: кому-то игра понравилась, кому-то – нет. Кто-то посчитал, что «таких игр быть не должно» (данный игрок принимал участие во всех играх и в целом очень быстро вылетал), что «это напоминает игру «Мафия»/компьютерные игры», «победа сильно зависит от везения». Было замечено, что некоторые игроки пребывали во фрустрации, а другие были настроены достаточно азартно. Интересно, что те игроки, которые прошли эксперимент несколько раз, и сначала не понимали «в чём смысл такой игры», отметили, что теперь для них игра стала осмысленней. Всего было проведено две игры с 11 игроками, 8 игр с 3 игроками и 4 игры с 6 игроками.

Также было создано специальное программное обеспечение, а именно игра с графическим пользовательским интерфейсом для проведения экспериментов. Само приложение будет описано в следующей главе.

Понимание правил игры и того, как пользоваться приложением, у участников полностью было и не вызывало вопросов.

Дальше полученные данные экспериментов и результатов тестов MBTI (Myers-Briggs Type Indicator) и Эннеаграмма нужно было исследовать. (Меньшикова, Меньшиков, & Седуш, 2014)

Коэффициент раскулачивания

Участник	1	2	3	4	5	6	7	8	9	Корреляция
1. Перфекционист	14	11	19	17	16	14	22	23	23	0,717783798
2. Помощник	15	11	22	19	22	19	17	9	7	-0,204726535
3. Мотиватор	19	14	16	13	8	14	13	19	22	-0,052490616
4. Романтик	14	25	24	20	20	24	23	19	10	-0,293723969
5. Мыслитель	20	20	11	7	13	11	15	19	20	0,092792408
6. Лоялист	15	20	17	19	22	26	15	22	9	-0,258840379
7. Энтузиаст	17	18	11	19	16	8	10	5	22	0,039628556
8. Лидер	20	9	10	17	9	8	14	16	19	0,17603855
9. Миротворец	10	16	14	13	18	20	15	12	12	-0,158277053
Extraversion	19	2	16	17	12	6	2	8	12	-0,053838629
Introversion	8	26	11	10	15	20	24	23	14	0,117452786
Sensing	17	7	4	23	23	21	16	28	10	0,237495607
Intuition	9	20	18	6	2	5	12	4	19	-0,115375517
Thinking	28	11	7	20	14	10	8	28	27	0,224844314
Feeling	0	8	15	6	9	11	15	2	2	-0,074171576
Judging	18	0	18	19	14	25	21	26	12	0,019762968
Perception	9	29	11	11	17	2	10	2	16	0,086042839
Коэффициент раскулачивания игрока	0,117096	0,295416	0,446132	0,516774	0,90641	0,036353	0,674129	0,85	0,81916	

Таблица 4. Корреляция коэффициента раскулачивания игроков с их психологическими параметрами

На удивление, данный параметр слабо коррелирует с психологическими параметрами, кроме как с «Перфекционист». Трактовать это можно тем, что люди данной категории бо-

ятся дисбаланса и отсутствия равновесия. Также данный тип личности считает себя меньше мира, но двигается против него, т.е. настроен агрессивно. (Wagner). Из проведённых опытов: средний коэффициент раскулачивания игроков равен 51,8%, минимум равен 11,7%, максимум равен 90,6%, стандартное отклонение = 26,2%.

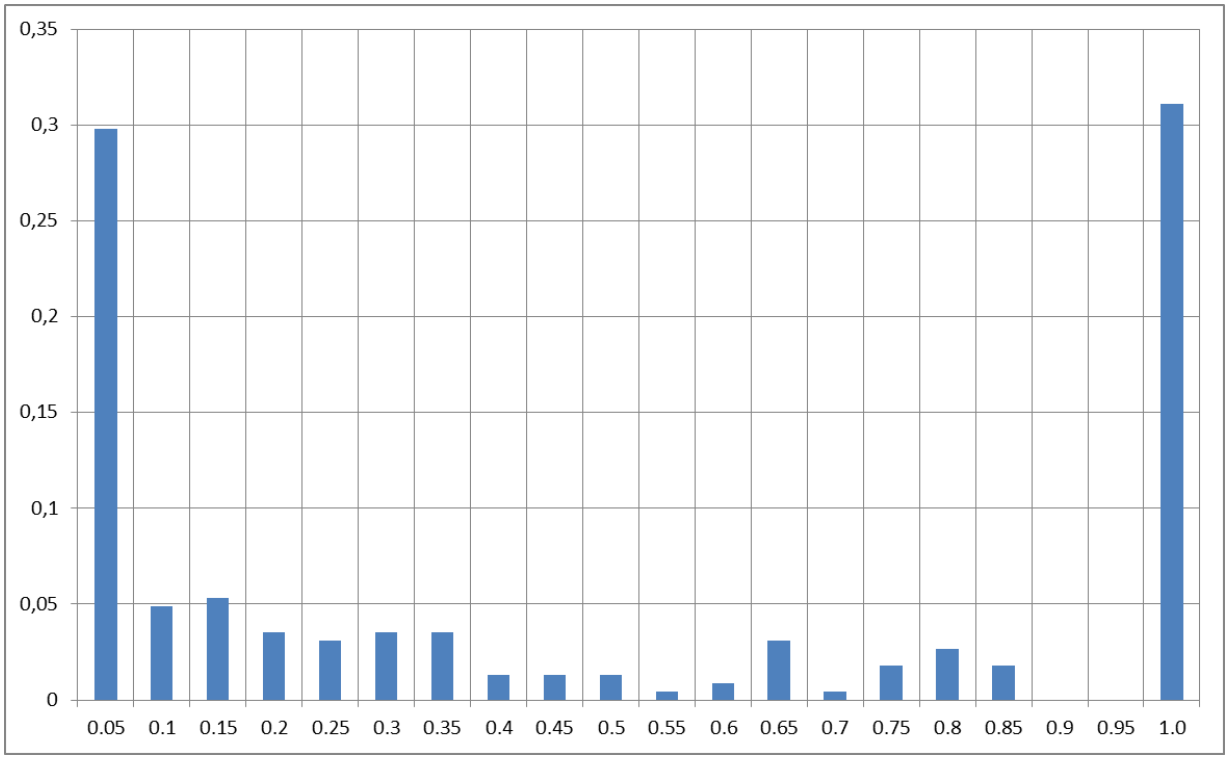


Рисунок 8. Распределение коэффициента раскулачивания ϕ на основе экспериментов.

На графике (см. Рисунок 8) представлено распределение коэффициента раскулачивания на основе экспериментов. По горизонтальной оси отложены интервалы от 0 до 1 коэффициента раскулачивания, а по вертикальной – процент ходов с таким коэффициентом. Заметим, что по сравнению с теоретическими результатами в экспериментах процент битья самых сильных и процент битья самых слабых почти одинаков. Это может возникать из-за такого социального момента как поведение толпы (агрессивной толпы). (Лебон, СПБ,1995г)

Коэффициент инертности выбора

Инертности выбора (Inertness of Choice) показывает, насколько часто игрок меняет свою стратегию, то есть выбирает других игроков, по ходу всей игры.

$$\sigma_i = \sqrt{\frac{\sum_{j=1}^{N} p_j^i (k_j^i - \langle k^i \rangle)^2}{\sum_{j=1}^{N} p_j^i}}$$

σ_i – средняя инертность выбора i-ого игрока, p_j^i – количество возможностей (число раз) в течение всей игры выбрать i-ым игроком j-ого игрока, k_j^i – количество выбора i-ым игроком j-ого игрока в течение всей игры, $\langle k^i \rangle$ – среднее число ходов на одного противника i-ого игрока.

Участник	1	2	3	4	5	6	7	8	9	Корреляция
1. Перфекционист	14	11	19	17	16	14	22	23	23	-0,60780912
2. Помощник	15	11	22	19	22	19	17	9	7	-0,03451014
3. Мотиватор	19	14	16	13	8	14	13	19	22	-0,298643434
4. Романтик	14	25	24	20	20	24	23	19	10	0,144419551
5. Мыслитель	20	20	11	7	13	11	15	19	20	0,21526817
6. Лоялист	15	20	17	19	22	26	15	22	9	-0,067921391
7. Энтузиаст	17	18	11	19	16	8	10	5	22	0,626538694
8. Лидер	20	9	10	17	9	8	14	16	19	-0,412520655
9. Миротворец	10	16	14	13	18	20	15	12	12	0,317088073
Extraversion	19	2	16	17	12	6	2	8	12	-0,203559399
Introversion	8	26	11	10	15	20	24	23	14	0,132794839
Sensing	17	7	4	23	23	21	16	28	10	-0,561099891
Intuition	9	20	18	6	2	5	12	4	19	0,499720458
Thinking	28	11	7	20	14	10	8	28	27	-0,313130533
Feeling	0	8	15	6	9	11	15	2	2	0,094227206
Judging	18	0	18	19	14	25	21	26	12	-0,948421281
Perception	9	29	11	11	17	2	10	2	16	0,937155259
Инертность выбора	1,171192	2,440157	1,248397	1,092633	1,763519	1,085401	0,976875	0,489898	1,505407	

Таблица 5. Корреляция инертности выбора с психологическими параметрами

Инерция тем выше, чем игрок чаще выбирает одного и того же игрока, и тем ниже, когда он чаще меняет цель. Согласно таблице 5, видна сильная обратная корреляция этого параметра с людьми, которые рассудительны (Judging) и принимают решения, взвешивая все против и за. И, соответственно, сильная корреляция есть с людьми, действующими по ощущению (Perception). Заметим, что перфекционисты стремятся выбирать разных оппонентов для удара. Также в небольшой степени это присуще лидерам. А люди с энтузиазмом пытаются чаще добить свою цель, довести своё дело до конца. (The 16 MBTI® Types)

Время хода

В нашей программе-игре также записывалось время хода игрока.

Участник	1	2	3	4	5	6	7	8	9	Корреляция
1. Перфекционист	14	11	19	17	16	14	22	23	23	0.523681616
2. Помощник	15	11	22	19	22	19	17	9	7	-0.774271244
3. Мотиватор	19	14	16	13	8	14	13	19	22	0.456804887
4. Романтик	14	25	24	20	20	24	23	19	10	-0.412653591
5. Мыслитель	20	20	11	7	13	11	15	19	20	0.580400627
6. Лоялист	15	20	17	19	22	26	15	22	9	-0.070101876
7. Энтузиаст	17	18	11	19	16	8	10	5	22	-0.218562729
8. Лидер	20	9	10	17	9	8	14	16	19	0.278365712
9. Миротворец	10	16	14	13	18	20	15	12	12	-0.113568113
Extraversion	19	2	16	17	12	6	2	8	12	-0.438499344
Introversion	8	26	11	10	15	20	24	23	14	0.513223535
Sensing	17	7	4	23	23	21	16	28	10	0.368784054
Intuition	9	20	18	6	2	5	12	4	19	-0.11506821
Thinking	28	11	7	20	14	10	8	28	27	0.512597275
Feeling	0	8	15	6	9	11	15	2	2	-0.459573098
Judging	18	0	18	19	14	25	21	26	12	0.169701899
Perception	9	29	11	11	17	2	10	2	16	-0.210821458
Среднее время хода	4.823529	5.5	3.3125	4.1875	5.392857	6.069444	6.044643	8.708333	7.964286	

Таблица 6. Корреляция времени хода (время хода в секундах)

Сильная обратная корреляция видна с таким типом как «Помощник». Возможно, «помощники» заботятся о том, чтобы другие игроки меньше ждали времени в эксперименте. Также наблюдается средняя корреляция в связи с интроверсией-экстраверсией и thinking-feeling. (The 16 MBTI® Types)

Выбор траектории при отсутствии симметричного равновесия

В ходе игры участник может сделать ход, который будет считаться симметрично-равновесным (из описанного выше в теоретической части работы). Но также он может и выбрать такую траекторию, на которой он имеет меньшую вероятность победы. Вполне можно подсчитать, используя теоретические результаты для конкретной игры, сколько таких «ошибок» сделал каждый игрок.

Участник	1	2	3	4	5	6	Корреляция
1. Перфекционист	16	14	22	23	23	9	-0,602193792
2. Помощник	22	19	17	9	7	22	0,518019127
3. Мотиватор	8	14	13	19	22	20	0,040128618
4. Романтик	20	24	23	19	10	12	0,292704693
5. Мыслитель	13	11	15	19	20	9	-0,723574605
6. Лоялист	22	26	15	22	9	19	0,366062595
7. Энтузиаст	16	8	10	5	22	19	-0,268124127
8. Лидер	9	8	14	16	19	16	-0,243975018
9. Миротворец	18	20	15	12	12	18	0,59404013
Extraversion	12	6	2	8	12	22	0,030729254
Introversion	15	20	24	23	14	6	-0,078048818
Sensing	23	21	16	28	10	8	-0,271562723
Intuition	2	5	12	4	19	13	0,03216199
Thinking	14	10	8	28	27	20	-0,51828159
Feeling	9	11	15	2	2	4	0,4141601
Judging	14	25	21	26	12	15	0,405384695
Perception	17	2	10	2	16	15	-0,367194037
Число выборов	2	6	5	3	2	6	

Таблица 7. Корреляция числа ошибок и психологических типов.

Из таблицы 7 видно, что у «мыслителей» (Эннеаграмма) и «Thinking» (MBTI) корреляция отрицательная и равна 72.4% и 51,8% соответственно, что видимо, говорит, о том, что эти люди умеют быстро просчитывать ход игры. «Перфекционисты» тоже совершают мало ошибок, т.к. мы уже знаем, что бить самого сильного – достаточно неплохая стратегия. Из двух таблиц с номером 4 и 7 также можно получить, что количество ошибок отрицательно коррелирует с коэффициентом раскулачивания игрока на уровне 86.6%. А вот «помощникам» и «миротворцам» ошибки вполне присущи – 51,8% и 59,4% соответственно.

Влияние пола участников

	Отношение числа ходов до выбывания к полному числу ходов	Среднее для группы
Участники мужского пола	0,622222222	0,639236111
	0,322222222	
	0,783333333	
	0,824074074	
	0,431712963	
	0,851851852	
Участники женского пола	1	0,962191358
	0,917824074	
	0,96875	

Таблица 8. Взаимосвязь пола участника с его временем нахождения в игре.

Представители женского пола почти всегда выигрывали игру, или же оставались предпоследними. Если говорить об отношении числа ходов существования участника в игре (пока его жизнь была больше нуля) к общей продолжительности игры, то процент нахождения в игре для девушек равен 96,2%, в то время как у представителей мужского пола он равен 63,9%. Это отображено на таблице 8.

Стоит отметить, что по наблюдениям д.ф.-м.н. Савватеева А.В. в постановочных дуэлях трёх и более лиц из 30 раз 29 раз победителем оказывался представитель женского пола.[5]

[5] Университетское ТВ УрФУ. Дуэли трёх лиц. Часть 2. https://youtu.be/e8K4ce07zxs?t=56m33s

Игра (программное приложение)

Приложение, реализующее игру по сети, было написано на языке Java 8 с использованием библиотеки графического интерфейса Swing. Приложение доступно для скачивания на сайте w17502.vdi.mipt.ru.

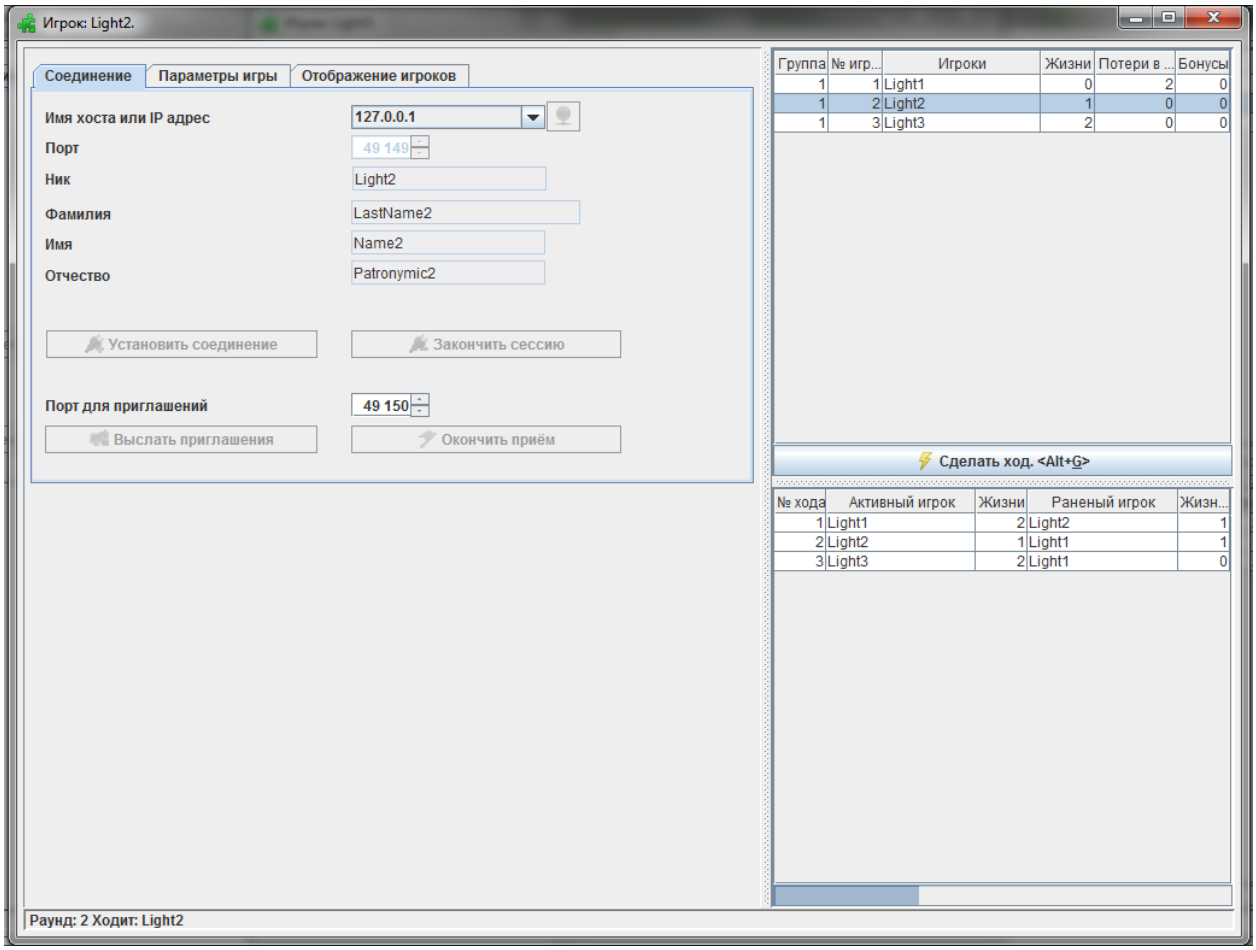


Рисунок 9. Вид приложения в процессе игры.

Приложение работает в локальной сети компьютеров по протоколу TCP/IP и имеет два варианта: серверный и клиентский. Это сделано для того, чтобы на компьютерах игроков невозможно было записать серверный вариант и мешать эксперименту. Также приложение по умолчанию заполняет собою весь экран. Это нужно, чтобы игроки не отвлекались в процессе эксперимента.

Стоит отметить, что Меньшикову И.С., другим сотрудникам Лаборатории экспериментальной экономики МФТИ и участникам опытов приложение очень понравилось. Все пожелания пользователей учтены и реализованы. Приложение вошло в архив ЛЭЭ МФТИ.

Настройка приложения

Закладка «Соединение» (см. рисунок 11) содержит следующие поля «Имя хоста или IP адрес», «Порт», «Ник», «Фамилия», «Имя», «Отчество» и «Порт для приглашений».

- Если приложение запущено как сервер, то «Фамилия», «Имя», «Отчество» могут быть не заполнены, а если запущено как клиент, то эти поля обязаны быть заполненными.

- «Ник» может быть любым.
- Рядом с полем «Имя хоста или IP адрес» есть кнопка, которая позволяет запросить клиенту хост сервера, используя UDP-протокол.
- Кнопка «Установить соединение» должна быть нажата, чтобы запустить сетевую часть программы на выставленном хосте и порте. «Закончить сессию» делает обратное, а потому потом можно будет открыть новую сессию на уже другом порте или хосте.
- Кнопка «Выслать приглашения» позволяет начать клиентам подключаться к игре и находить хост сервера через UDP-протокол.
- Кнопка «Окончить приём» завершает возможность приёма клиентов сервером и закрывает рассылку через UDP-протокол.

Вкладка «Параметры игры» содержит следующее.

- «Режим игры» принимает только два значения: «До оставшихся» или «До выбывших». В зависимости от выбора ниже вводится «Количество оставшихся/выбывших игроков».
- «Кол-во жизней» отображает, сколько жизней у игроков на старте (у всех одинаково).
- «Время на ход в сек.» позволяет задать время хода участников и служит для последующего контроля процесса игры.
- «Доп. Ход за потерю жизни» может быть выставлен от 0 до 1 десятичной дробью и служит для особой вариации игры.
- Галочка «Делить на группы» может быть выставлена, если игроков слишком много и удобней было бы разбить их на группы и проводить в каждой отдельный эксперимент. «Минимальная группа» указывает на то, какого минимальное число игроков в группе при разбивке.
- «Путь для сохранения файлов» указывает на то, куда будут сохраняться файлы (в формате .csv) хода эксперимента в файловой системе. По умолчанию, они сохраняются в том же каталоге, в котором запускается приложение сервера. Файлы сохраняются только на машине с запущенным приложением-сервером.

Также есть вкладка «Отображение игроков»:

- Кнопка «Старт игры» запускает игру с уже выбранными настройками.
- «Отображение игроков» предоставляет 4 поля на выбор: «По нику», «По номеру», «По ФИО», «По списку заготовок». Если выбран последний пункт, то придётся настроить «Кодовую страницу» и «Путь к файлу со списком заготовок».
- «Кодовая страница» имеет два варианта: “UTF-8”, “WINDOWS-1251”. Выбирать надо тот, что будет соответствовать кодировке файла со списком заготовок.
- «Путь к файлу со списком заготовок» даёт возможность указать на текстовой файл в файловой системе со списком заготовок. Список заготовок – это псевдонимы, идущие в каждой новой строчке. Это надо, чтобы повысить чистоту эксперимента. Например, список наименований овощей и фруктов использовался в одном из проведённых опытов.

Работа приложения

После того, как на сервере ведущий нажал кнопку «Старт игры», начинается игра.

- В клиентском приложении в самом низу отображается, кто сейчас ходит. Если этот самый игрок и ходит, то внизу справа начинает заполняться строка прогресса, отмеряя для игрока визуально, сколько время прошло, а сколько осталось. Если время истекло, то игрок не прекращает ходить, но это служит сигналом ведущему о том, что где-то человек задерживает ход эксперимента.
- В свой ход игрок должен выбрать другого игрока из списка вверху справа и нажать клавишу «Сделать ход. Alt+G» или комбинацию на клавиатуре “Alt+G”. После этого ход перейдёт следующему игроку с количеством жизней больше нуля. Если игрок выбрал себя или уже выбывшего игрока, то приложение выведет в диалоговом окне сообщение об этом, и игроку придётся выбрать другой вариант.
- В таблице вверху справа отображается следующая информация обо всех игроках: «Группа», «№ игрока» (порядок хода), «Игроки» (здесь отображаются их псевдонимы), «Жизни», «Потери в раунде» (сколько жизней было отнято у игрока после его предыдущего хода), «Бонусы» (накопленные ходы для особой вариации игры).
- В таблице снизу справа отображается история ходов. Каждый ход есть запись следующего вида: «№ хода», «Активный игрок», «Жизни» (жизни активного игрока), «Раненый игрок», «Жизни*» (жизни раненого игрока после хода).
- Информация в данных таблицах открыта всю игру всем участникам.

В результате окончания одной из игры её данные сохраняются в файлах формата .csv в виде таблиц. Файл со словом “Casting” содержит то, какие псевдонимы соответствовали каким ФИО, а “Moves” – все записанные ходы. Файлы можно открыть редакторами таблиц и затем изучить проведенные опыты.

Выводы (Анализ результатов)

В рамках данной дипломной работы получены следующие результаты:

1. Были описаны детерминированная модель конфликта трёх и более участников и симметричное равновесие для неё.
2. Создано программное приложение для поиска вероятностей выигрыша и поиска всех симметрично-равновесных траекторий.
3. Рассмотрен подход к анализу траекторий на основе коэффициента раскулачивания и коэффициента раскулачивания игрока.
4. Создано программное обеспечение для проведения игр в лаборатории.
5. Изучена связь психологических типов участников и их поведения в игре.

Список литературы

Aumann, R. J., & Hart, S. (1992). *Handbook of Game Theory with Economic Applications* (изд. North-Holland, T. 1).

Kinnaird, C. (1946). *Encyclopedia of Puzzles and Pastimes.* Citadel, Secaucus, NJ(USA).

The 16 MBTI® Types. (б.д.). Получено 26 Июня 2016 г. г., из The Myers & Briggs Foundation: http://www.myersbriggs.org/my-mbti-personality-type/mbti-basics/the-16-mbti-types.htm

Wagner, J. (б.д.). *Karen Horney's Three Trends (Moving Towards, Against, Away From) and the Enneagram Styles*. Получено 26 Июня 2016 г. г., из The Enneagramm Spectrum: http://www.enneagramspectrum.com/184/karen-horneys-three-trends-moving-towards-against-away-from-and-the-enneagram-styles/

Woods, S. (2012). *Eurogames: the design, culture and play of modern European board games.* Jefferson, North Carolina 28640, United States of America: McFarland & Company, Inc., Publishers.

Лебон, Г. (СПБ,1995г). *Психология народов и масс.* СПБ: "Макет".

Меньшиков, И. (2010). *Лекции по теории игр и экономическому моделированию.* Москва: ООО «Контакт Плюс».

Меньшикова, О. Р., Меньшиков, И. С., & Седуш, А. О. (2014). *Моделирование процессов обработки информации.* Москва.

Приложение 1

Программный код приложения ищущего все равновесные исходы конкретной модели игры.

Game.java

```
import java.io.PrintStream;
import java.util.ArrayList;
public class Game {
	int survivalsCount;
	MultiArray[] gameArrays;
	GameState gameState;
	int operationCount = 0;
	final double epsilon = 1e-16;
	/**
	 * Конструктор игры для 3 и более игроков
	 * @param players Массив стартовых жизней игроков
	 * @param survivalsCount Сколько должно остаться живых
	 */
	public Game(int[] players, int survivals)
	{
		survivalsCount = survivals;
		boolean liveError = false;
		int maxLive = 0;
		for(int i = 0; i < players.length; i++)
		{
			if(players[i] < 1)
			{
				liveError = true;
				break;
			}
			if(players[i] > maxLive)
				maxLive = players[i];
		}
		if(players.length == 0 || survivalsCount <= 0 || survivalsCount >= play-
ers.length || liveError)
			try {
				throw new Exception("Game can\'t be initialized with such parame-
ters!");
			} catch (Exception e) {
				e.printStackTrace();
			}
		gameArrays = new MultiArray[players.length + 1];
		for(int i = 1; i < players.length + 1; i++)
			gameArrays[i] = MultiArray.CreateMultiCube(i, maxLive + 1);
		gameState = new GameState(0, players);
		try {
			Analyze(gameState);
			System.out.println("Operations made: " + operationCount);
		} catch (Exception e) {
			e.printStackTrace();
		}

	}
	PrintStream printStream;
	public void PrintAllEquilibriumPaths(PrintStream out)
	{
		shots = new ArrayList<Integer>();
		victims = new ArrayList<Integer>();
		gameNumber = 0;
		printStream = out;
		gameState.TransposeActivePlayer();
		try {
			PrintAllEquilibriumPaths(gameState);
		} catch (Exception e) {
			e.printStackTrace();
		}
```

```
                System.out.println("All equilibrium paths are found!");
        }
        ArrayList<Integer> shots, victims;
        int gameNumber;
        void PrintAllEquilibriumPaths(GameState gameState) throws Exception
        {
                GameState nextState = new GameState(gameState);
                for(int i = 0; i < gameState.GetAliveCount() - 1; i++)
                {
                        int target = (gameState.GetActivePlayer() + i + 1) %
gameState.GetAliveCount();
                        GameState localState;
                        boolean killed = nextState.Shoot(target);
                        if(killed)
                                localState = GameState.Degenerate(gameState, target);
                        else
                                localState = nextState;
                        if(localState.IsGameFinished(survivalsCount) == true)
                        {
                                gameNumber++;
                                shots.add(gameState.InitialPlayers[gameState.GetActivePlayer()]);
                                victims.add(gameState.InitialPlayers[target]);
                                PrintShotsAndVictims();
                                shots.remove(shots.size() - 1);
                                victims.remove(victims.size() - 1);
                        }
                        else
if(Math.abs(gameArrays[gameState.GetAliveCount()].At(gameState.Lives).Profit[gameState.GetActi
vePlayer()] -
                                        gameAr-
rays[localState.GetAliveCount()].At(localState.Lives).Profit[localState.GetActivePlayer()]) <
epsilon)
                        {
                                shots.add(gameState.InitialPlayers[gameState.GetActivePlayer()]);
                                victims.add(gameState.InitialPlayers[target]);
                                localState.PassTurn();
                                PrintAllEquilibriumPaths(localState);
                                localState.PassTurnCCW();
                                shots.remove(shots.size() - 1);
                                victims.remove(victims.size() - 1);
                        }
                        nextState.Heal(target);
                }
        }
        void PrintShotsAndVictims()
        {
                int length = shots.size();
                for(int i = 0; i < length; i++)
                        printStream.println(Integer.toString(gameNumber) + ";" + shots.get(i) +
";" + victims.get(i));
        }
        public void PrintChancesToWin(PrintStream out)
        {
                try
                {
                        for(int i = 0; i < gameState.GetAliveCount(); i++)
                        {

        out.println(gameArrays[gameState.GetAliveCount()].At(gameState.Lives).Profit[(i +
gameState.PlayersDelta()) % gameState.GetAliveCount()]);
                        }
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
        }
```

```
	public double[] GetChancesToWin()
	{
		double[] chances = new double[gameState.GetAliveCount()];
		try
		{
			for(int i = 0; i < gameState.GetAliveCount(); i++)
			{
				chances[i] = gameAr-
rays[gameState.GetAliveCount()].At(gameState.Lives).Profit[(i + gameState.PlayersDelta()) %
gameState.GetAliveCount()];
			}
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
		return chances;
	}

	void FightDuel(GameState gameState) throws Exception
	{
		if(gameState.Lives[0] > gameState.Lives[1] || (gameState.Lives[0] ==
gameState.Lives[1] && gameState.GetActivePlayer() == 0))
			gameArrays[gameState.GetAliveCount()].At(gameState.Lives).Profit = new
double[]{1, 0};
		else
			gameArrays[gameState.GetAliveCount()].At(gameState.Lives).Profit = new
double[]{0, 1};
	}
	void Analyze(GameState gameState) throws Exception
	{
		if(gameArrays[gameState.GetAliveCount()].Existing(gameState.Lives)) //Checking
for ready results
			return;
		operationCount++;
		if(gameState.IsGameFinished(survivalsCount))
		{
			for(int i = 0; i < gameState.GetAliveCount(); i++)
				gameAr-
rays[gameState.GetAliveCount()].At(gameState.Lives).Profit[i] = gameState.Lives[i] == 0 ? 0 :
1;
			return;
		}
		if(gameState.GetAliveCount() == 2) //Two-player check
		{
			FightDuel(gameState);
			return;
		}
		int[] target = new int[gameState.GetAliveCount() - 1];
		double[][] profits = new double[gameState.GetAliveCount() - 1][];
		double bestResult = -1;
		ArrayList<Integer> bestChoice = new ArrayList<Integer>();
		for(int i = 0; i < gameState.GetAliveCount() - 1; i++)
		{
			target[i] = (gameState.GetActivePlayer() + i + 1) %
gameState.GetAliveCount();
			GameState localState;
			boolean killed = gameState.Shoot(target[i]);
			if(killed)
				localState = GameState.Degenerate(gameState, target[i]);
			else
				localState = gameState;
			localState.PassTurn();
			Analyze(localState);
			profits[i] = gameAr-
rays[localState.GetAliveCount()].At(localState.Lives).Profit;
			localState.PassTurnCCW();
			double result = profits[i][localState.GetActivePlayer()];
```

```
                        if(bestResult + epsilon < result)
                        {
                                bestResult = result;
                                bestChoice.clear();
                                bestChoice.add(i);
                        }
                        else if (Math.abs(result - bestResult) < epsilon)
                        {
                                bestChoice.add(i);
                        }
                        if(killed) //Have to restore profits
                        {
                                double[] correctProfit = new double[gameState.GetAliveCount()];
                                for(int j = 0; j < gameState.GetAliveCount(); j++)
                                {
                                        correctProfit[j] = 0;
                                        for(int k = 0; k < localState.GetAliveCount(); k++)
                                        {
                                                if(gameState.InitialPlayers[j] == lo-
calState.InitialPlayers[k])
                                                {
                                                        correctProfit[j] = profits[i][k];
                                                        break;
                                                }
                                        }
                                }
                                profits[i] = correctProfit;
                        }
                        gameState.Heal(target[i]);
                }
                double p = 1.0 / (double)bestChoice.size();
                for(int i = 0; i < bestChoice.size(); i++)
                {
                        for(int j = 0; j < gameState.GetAliveCount(); j++)
                                gameAr-
rays[gameState.GetAliveCount()].At(gameState.Lives).Profit[j] += p * prof-
its[bestChoice.get(i)][j];
                }
        }
```

GameState.java

```
import java.util.Arrays;

class GameState
{
        private int ActivePlayer;
        public int[] Lives, InitialPlayers;
        private int sumOfLives, legalActivePlayer, wrongActivePlayer;
        private GameState()
        {}
        public GameState(int activePlayer, int[] lives)
        {
                ActivePlayer = activePlayer;
                Lives = Arrays.copyOf(lives, lives.length);
                InitialPlayers = new int[lives.length];
                for(int i = 0; i < lives.length; i++)
                        InitialPlayers[i] = i;

                CalculateLegalActivePlayer();
                if(ActivePlayer != legalActivePlayer)
                        TransposeActivePlayer();
        }
        public GameState(GameState copyState)
        {
                ActivePlayer = copyState.ActivePlayer;
                Lives = Arrays.copyOf(copyState.Lives, copyState.Lives.length);
```

```
                InitialPlayers = Arrays.copyOf(copyState.InitialPlayers,
copyState.InitialPlayers.length);
                sumOfLives = copyState.sumOfLives;
                legalActivePlayer = copyState.legalActivePlayer;
                wrongActivePlayer = copyState.wrongActivePlayer;
        }
        public int PlayersDelta()
        {
                return legalActivePlayer - wrongActivePlayer;
        }
        public int GetActivePlayer()
        {
                return ActivePlayer;
        }
        public int GetAliveCount()
        {
                return Lives.length;
        }
        /**
         * To shoot index-player
         * @param index
         * @return If true, index-player is killed
         */
        public boolean Shoot(int index)
        {
                sumOfLives--;
                Lives[index]--;
                if(Lives[index] == 0)
                        return true;
                return false;
        }
        public void Heal(int index)
        {
                sumOfLives++;
                Lives[index]++;
        }
        void CalculateLegalActivePlayer()
        {
                sumOfLives = 0;
                for(int i = 0; i < Lives.length; i++)
                {
                        sumOfLives += Lives[i];
                }
                legalActivePlayer = sumOfLives % Lives.length;
                if(legalActivePlayer != 0)
                        legalActivePlayer = Lives.length - legalActivePlayer;
        }
        public void TransposeActivePlayer()
        {
                wrongActivePlayer = ActivePlayer;
                ActivePlayer = legalActivePlayer;
                TransposeInnerData(ActivePlayer, wrongActivePlayer);
        }
        void TransposeInnerData(int active, int passive)
        {
                int[] transposedLives = new int[Lives.length];
                int[] transposedInitialPlayers = new int[Lives.length];
                for(int i = 0; i < Lives.length; i++)
                {
                        try
                        {
                                int a = (active + i) % Lives.length, p = (passive + i) %
Lives.length;
                                transposedLives[a] = Lives[p];
                                transposedInitialPlayers[a] = InitialPlayers[p];
                        }
                        catch(Exception e)
                        {
```

```
                        System.out.println(e.getMessage());
                    }
                }
                Lives = transposedLives;
                InitialPlayers = transposedInitialPlayers;
        }
        public void RestoreActivePlayer()
        {
                ActivePlayer = wrongActivePlayer;
                TransposeInnerData(ActivePlayer, legalActivePlayer);
        }
        boolean IsGameFinished(int survivalsCount)
        {
                int survivals = 0;
                for(int i = 0; i < Lives.length; i++)
                {
                        if(Lives[i] > 0)
                                survivals++;
                }
                if(survivals == survivalsCount)
                        return true;
                else if(survivals < survivalsCount)
                        try {
                                throw new Exception("Number of survivals is less that it can
be!");
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                return false;
        }
        public static GameState Degenerate(GameState gameState, int deadPlayer)
        {
                GameState degenerated = new GameState();
                degenerated.Lives = new int[gameState.GetAliveCount() - 1];
                degenerated.InitialPlayers = new int[gameState.GetAliveCount() - 1];
                for(int i = 0; i < gameState.Lives.length; i++)
                {
                        if(deadPlayer == i)
                                continue;
                        int k = deadPlayer > i ? i : i - 1;
                        degenerated.Lives[k] = gameState.Lives[i];
                        degenerated.InitialPlayers[k] = gameState.InitialPlayers[i];
                }
                degenerated.ActivePlayer = gameState.ActivePlayer > deadPlayer ?
gameState.ActivePlayer - 1 : gameState.ActivePlayer;
                degenerated.CalculateLegalActivePlayer();
                if(degenerated.ActivePlayer != degenerated.legalActivePlayer)
                        degenerated.TransposeActivePlayer();
                return degenerated;
        }
        public void PassTurn()
        {
                ActivePlayer = (ActivePlayer + 1) % GetAliveCount();
        }
        public void PassTurnCCW()
        {
                ActivePlayer = (ActivePlayer + GetAliveCount() - 1) % GetAliveCount();
        }
}
```

MultiArray.java

```
class CellInfo {
        int dimensionsCount;
        public double[] Profit;
        public CellInfo(int dimensions)
        {
                this.dimensionsCount = dimensions;
```

```
                Profit = new double[dimensionsCount];
        }
}

public class MultiArray {
        int dimensionsCount, volume;
        int[] dimensionLengths;
        CellInfo[] cells;
        int[] shifts;
        public MultiArray(int[] dimensions) {
                for(int i = 0; i < dimensions.length; i++)
                        if(dimensions[i] < 1)
                                try {
                                        throw new Exception("A dimension can not have a length
less than one!");
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                dimensionLengths = dimensions.clone();
                dimensionsCount = dimensionLengths.length;
                shifts = new int[dimensionsCount];
                shifts[0] = 1;
                for(int i = 1; i < dimensionsCount; i++)
                        shifts[i] = shifts[i - 1] * dimensionLengths[i - 1];
                volume = shifts[dimensionsCount - 1] * dimensionLengths[dimensionsCount - 1];
                cells = new CellInfo[volume];
        }
        public static MultiArray CreateMultiCube(int dimCount, int dimLength)
        {
                int[] dimensions = new int[dimCount];
                for(int i = 0; i < dimCount; i++)
                        dimensions[i] = dimLength;
                return new MultiArray(dimensions);
        }
        public CellInfo At(int[] coordinate) throws Exception
        {
                int index = GetInternalIndex(coordinate);
                if(cells[index] == null)
                        cells[index] = new CellInfo(dimensionsCount);
                return cells[index];
        }
        public boolean Existing(int[] coordinate) throws Exception
        {
                return cells[GetInternalIndex(coordinate)] == null ? false : true;
        }
        private int GetInternalIndex(int[] coordinate) throws Exception
        {
                if(coordinate.length != dimensionsCount)
                        throw new Exception("Wrong coordinate: not equal count of dimensions!");

                int index = 0;
                for(int i = 0; i < dimensionsCount; i++)
                {
                        if(coordinate[i] < 0)
                                throw new Exception("Wrong coordinate: coordinate has negative
element!");
                        if(coordinate[i] >= dimensionLengths[i])
                                throw new Exception("Wrong coordinate: coordinate's element is
out of a dimension!");
                        index += shifts[i] * coordinate[i];
                }
                return index;
        }
}
```

Приложение 2

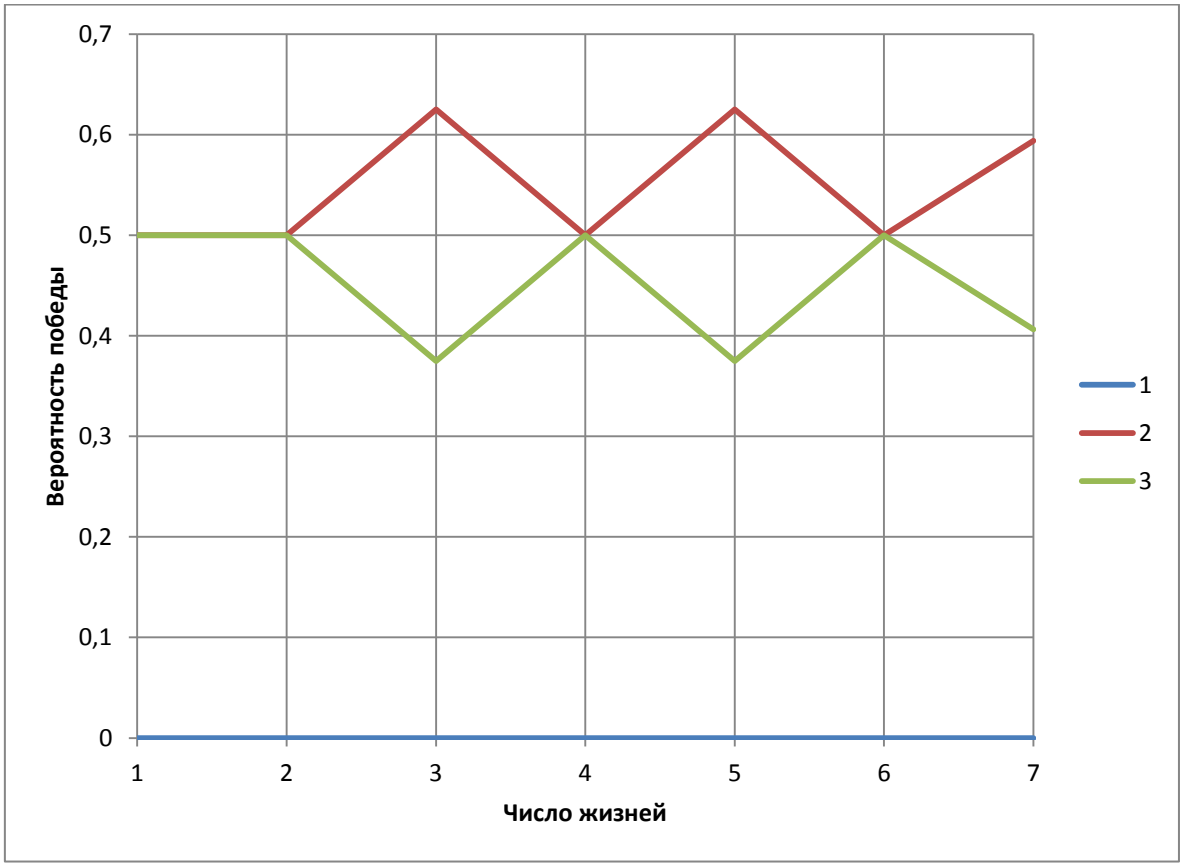
Таблицы и графики теоретических вероятностей победы для игр с 3-8 участниками.

Таблицы имеют следующий формат. Первый столбец – число жизней на старте, первая строка – номер игрока. На пересечении столбца и строки располагается вероятность победы игрока.

Графики по вертикальной оси показывают вероятность победы, по горизонтальной оси – число жизней на старте. Игроки обозначаются линиями разных цветов.

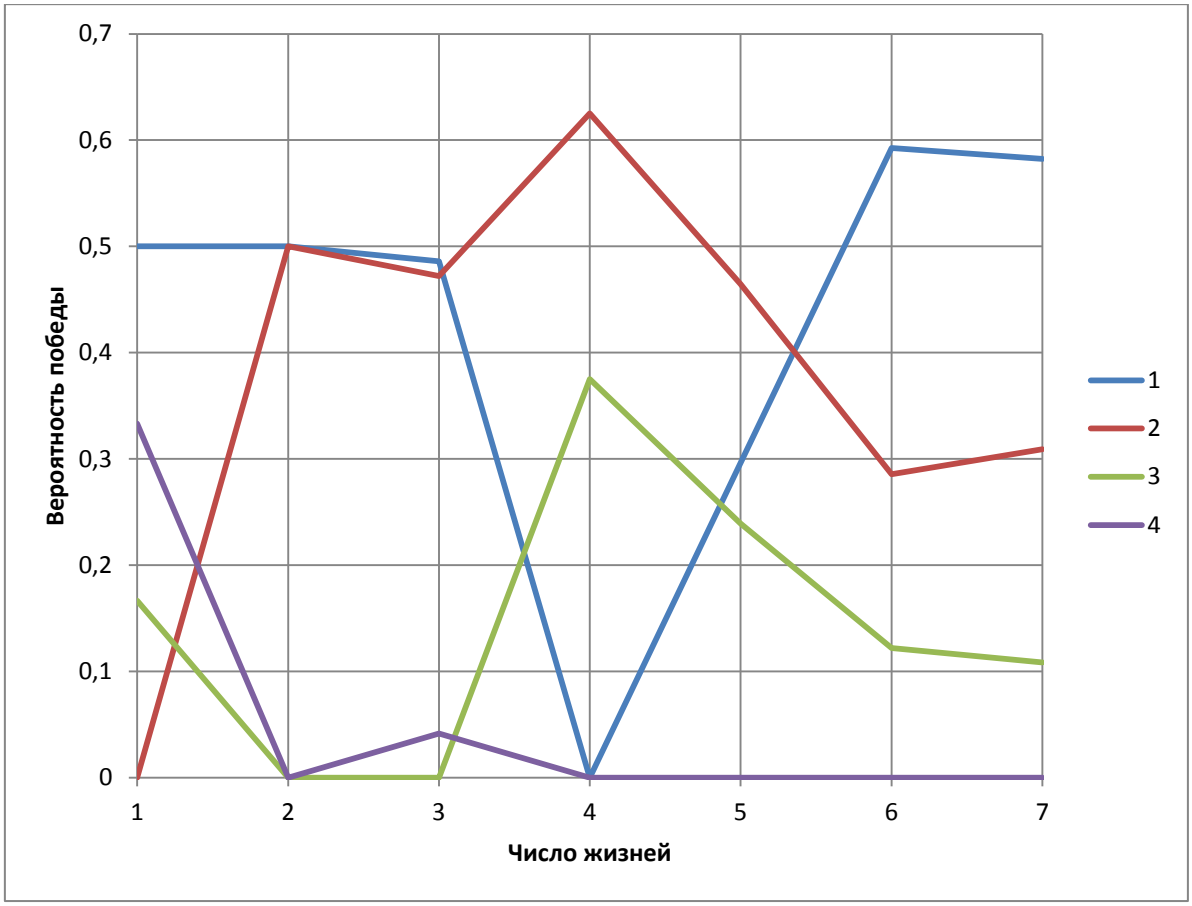
Игры с 3 участниками

	1	2	3
1	0	0,5	0,5
2	0	0,5	0,5
3	0	0,625	0,375
4	0	0,5	0,5
5	0	0,625	0,375
6	0	0,5	0,5
7	0	0,59375	0,40625



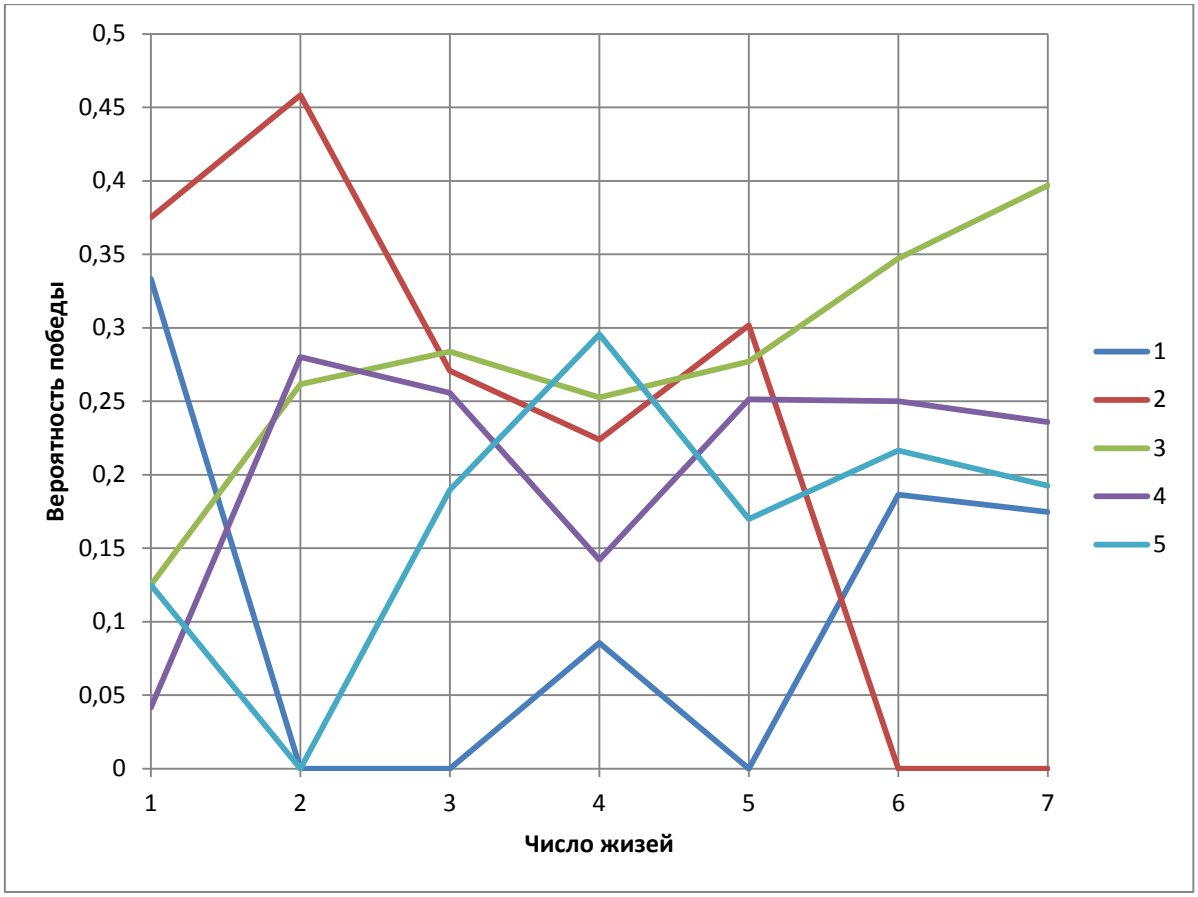
Игры с 4 участниками

	1	2	3	4
1	0,5	0	0,166667	0,333333
2	0,5	0,5	0	0
3	0,486111	0,472222	0	0,041667
4	0	0,625	0,375	0
5	0,296296	0,464506	0,239198	0
6	0,592593	0,285494	0,121914	0
7	0,582305	0,309199	0,108496	0



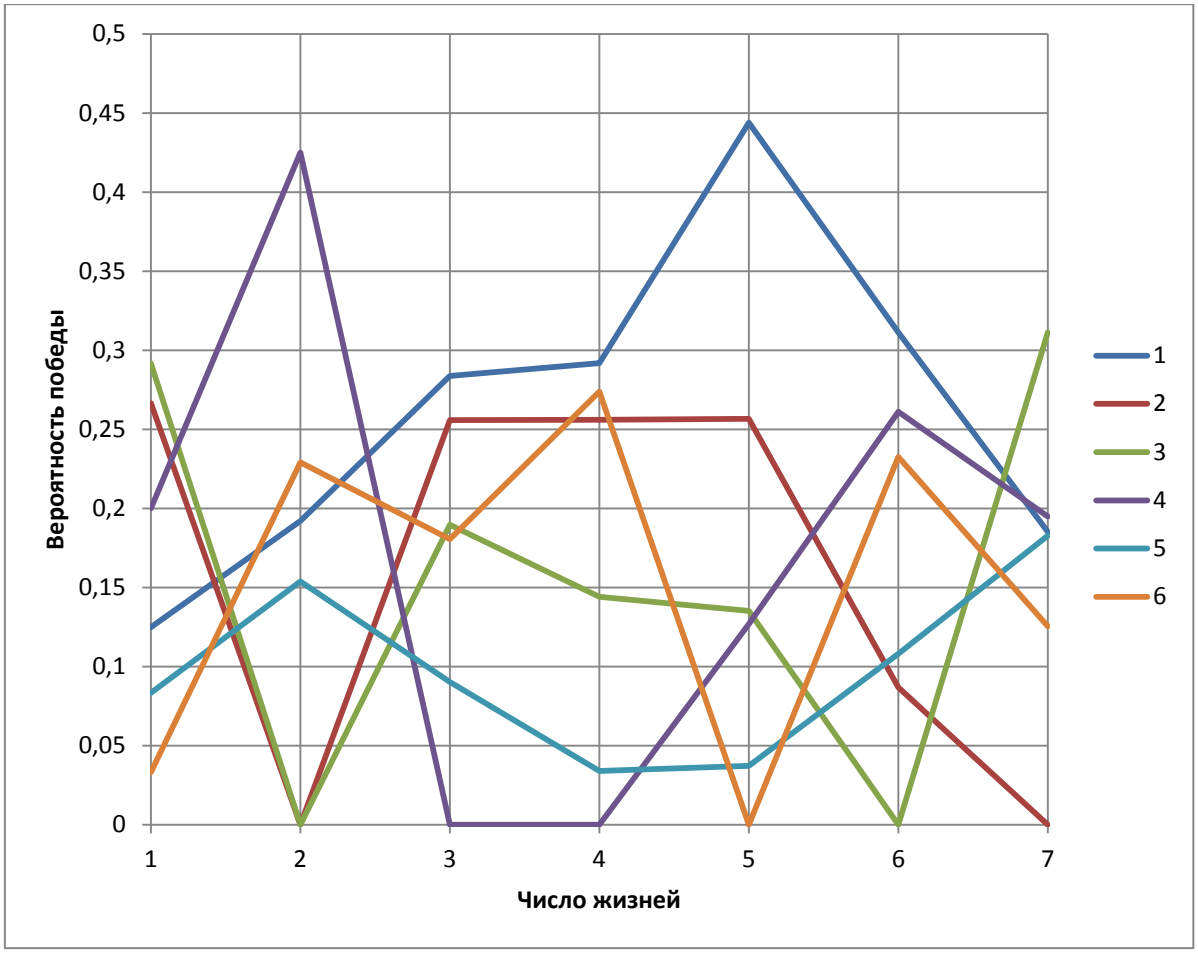
Игры с 5 участниками

	1	2	3	4	5
1	0,333333	0,375	0,125	0,041667	0,125
2	0	0,458333	0,261574	0,280093	0
3	0	0,270833	0,283709	0,255787	0,18967
4	0,085503	0,223958	0,252713	0,142253	0,295573
5	0	0,301676	0,277086	0,251424	0,169813
6	0,186343	0	0,347222	0,25	0,216435
7	0,174655	0	0,396946	0,235921	0,192478



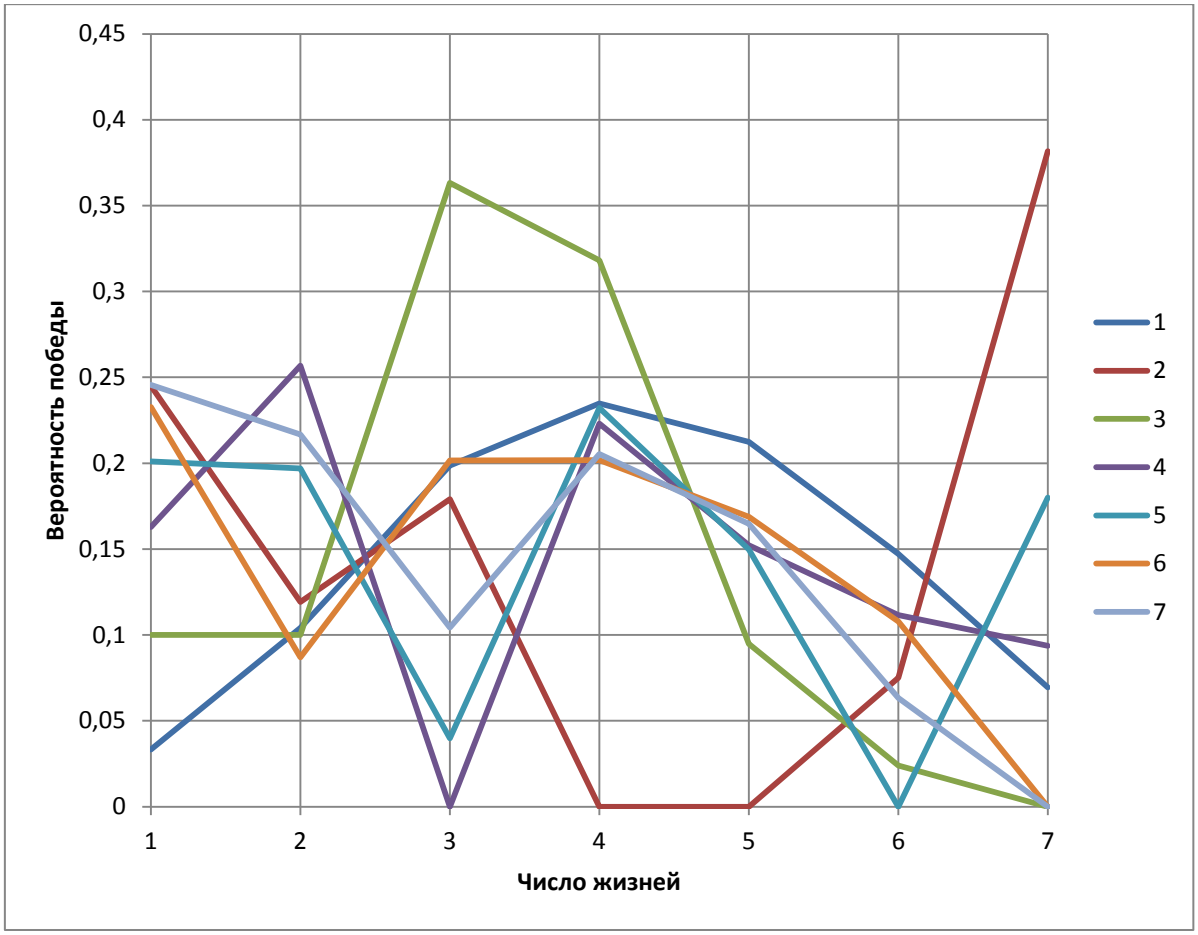
Игры с 6 участниками

	1	2	3	4	5	6
1	0,125	0,266667	0,291667	0,2	0,083333	0,033333
2	0,192148	0	0	0,425	0,153815	0,229037
3	0,283709	0,255787	0,18967	0	0,090278	0,180556
4	0,29184	0,256068	0,144045	0	0,034115	0,273932
5	0,443845	0,256675	0,135243	0,126875	0,037361	0
6	0,311302	0,086839	0	0,261044	0,108161	0,232654
7	0,185215	0	0,311379	0,194968	0,18285	0,125588



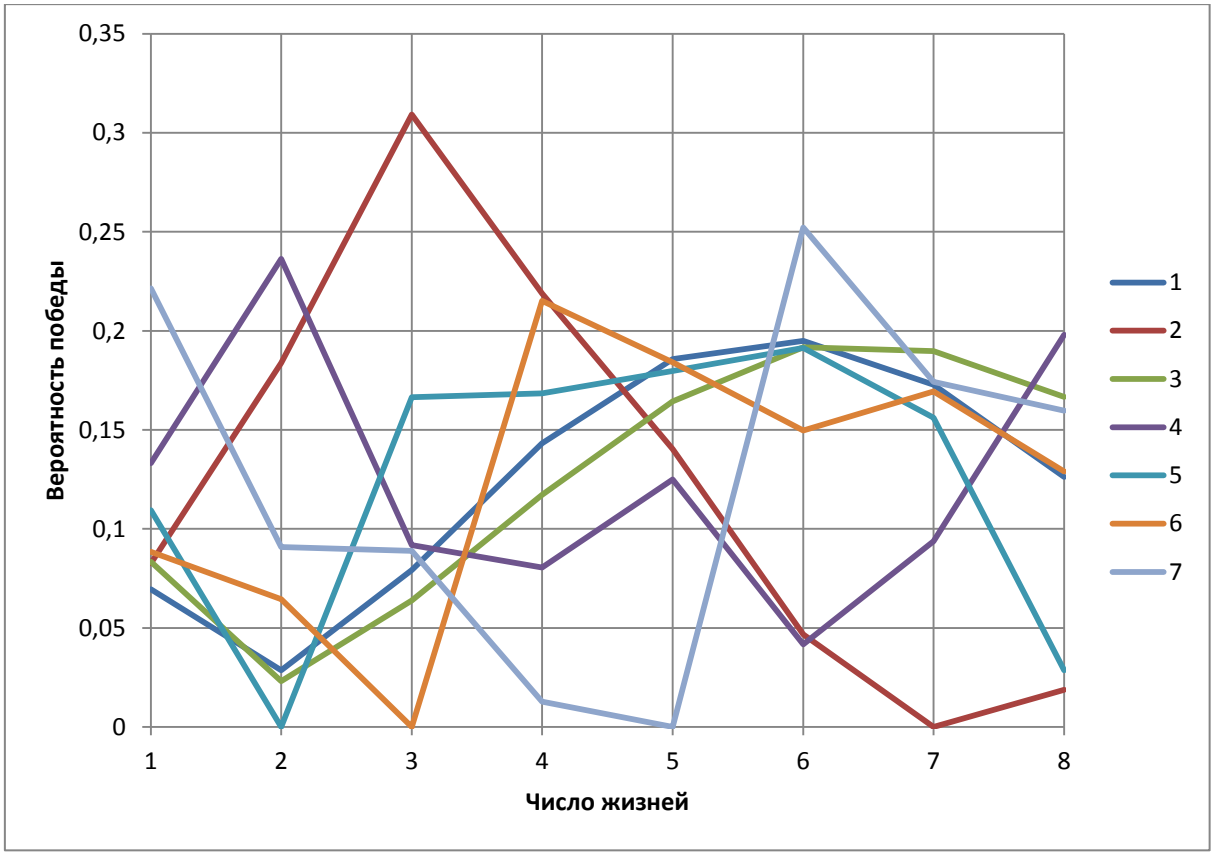
Игры с 7 участниками

	1	2	3	4	5	6	7
1	0,033333	0,104167	0,198611	0,234722	0,2125	0,147222	0,069444
2	0,245	0,119204	0,17913	0	0	0,075	0,381667
3	0,1	0,1	0,363194	0,318056	0,094792	0,023958	0
4	0,162792	0,256826	0	0,22305	0,152162	0,111521	0,093649
5	0,201098	0,196924	0,039797	0,232261	0,149873	0	0,180047
6	0,232809	0,087001	0,201678	0,201862	0,168785	0,107865	0
7	0,245697	0,216728	0,104383	0,205264	0,16464	0,063288	0



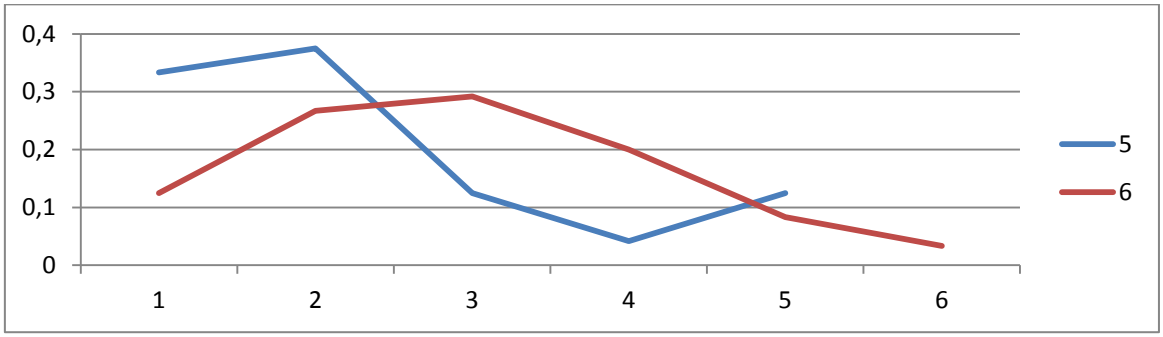
Игры с 8 участниками

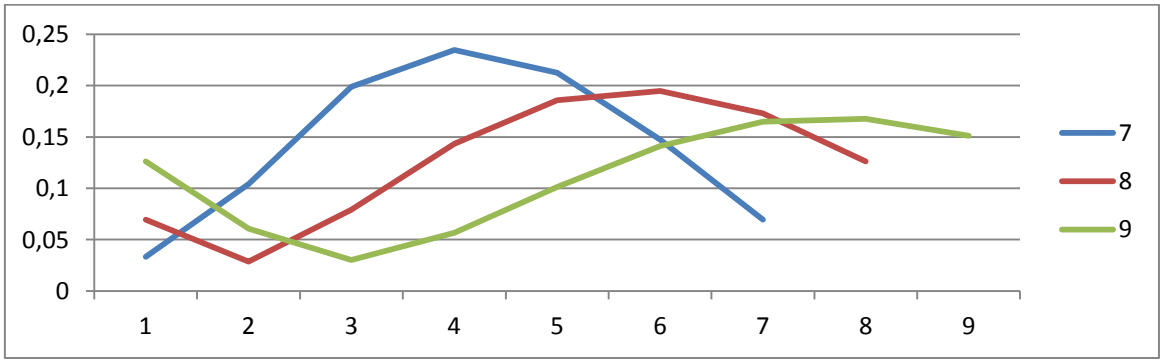
	1	2	3	4	5	6	7	8
1	0,069444	0,028571	0,079167	0,143254	0,185714	0,194841	0,172817	0,12619
2	0,0825	0,18375	0,309167	0,218889	0,140208	0,046736	0	0,01875
3	0,083333	0,023148	0,063889	0,11713	0,164352	0,191667	0,189815	0,166667
4	0,133102	0,236372	0,091725	0,080469	0,125	0,041667	0,09375	0,197917
5	0,10943	0	0,166442	0,168379	0,179757	0,191276	0,156085	0,02863
6	0,088375	0,064458	0	0,215073	0,184119	0,149679	0,169397	0,1289
7	0,22146	0,090795	0,088862	0,012814	0	0,25213	0,174294	0,159645

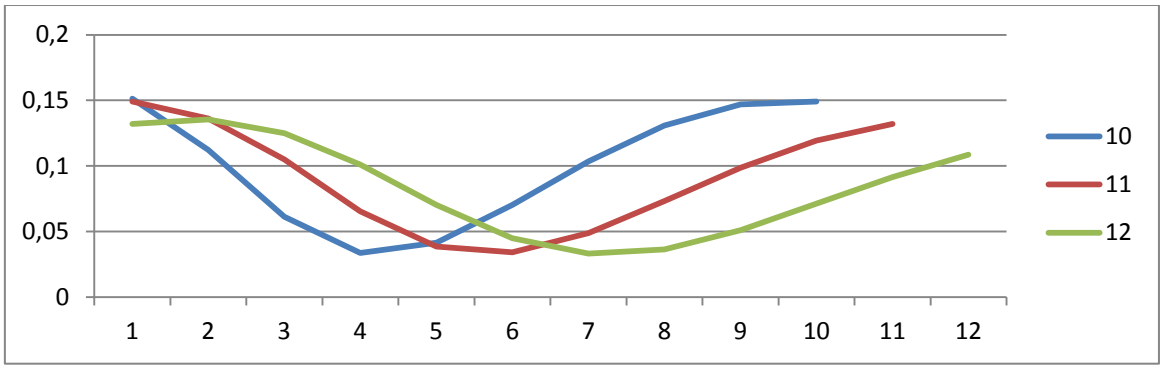


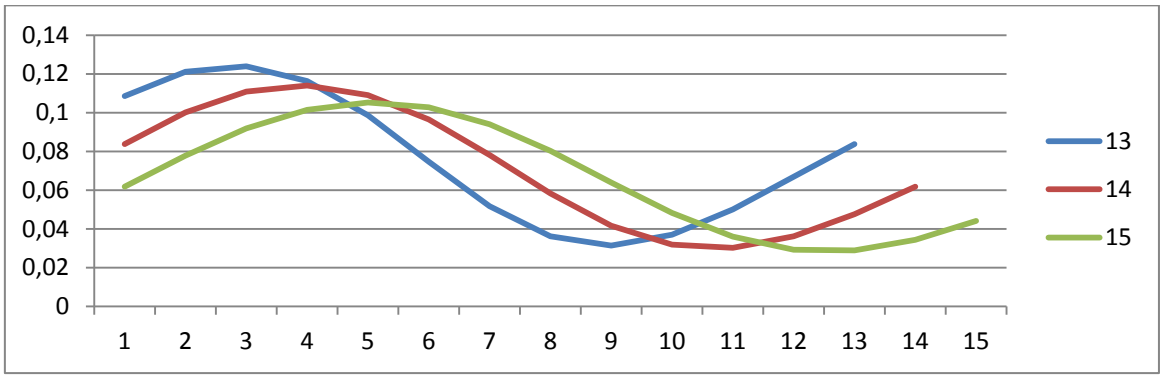
Игры с 1 стартовой жизнью у каждого из участников

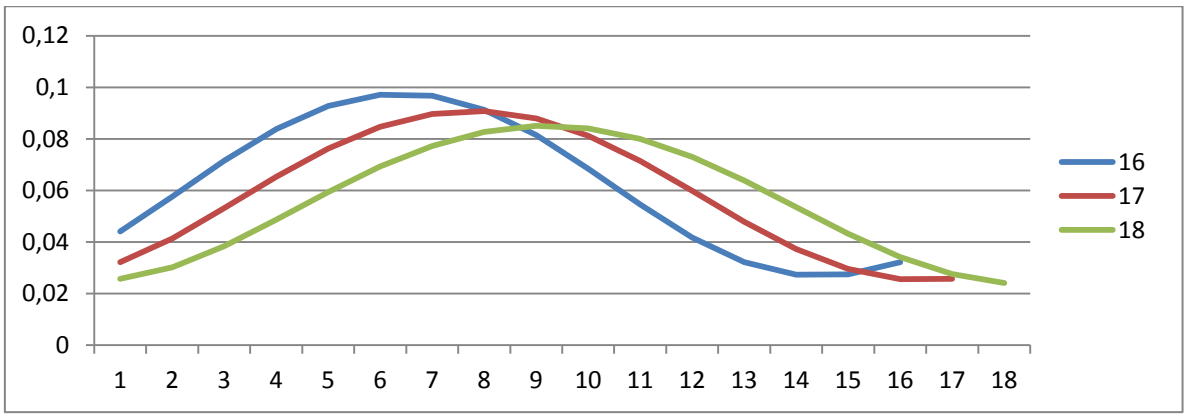
Графики по вертикальной оси показывают вероятность победы, по горизонтальной оси – номер игрока. Количество игроков обозначается линиями разных цветов.

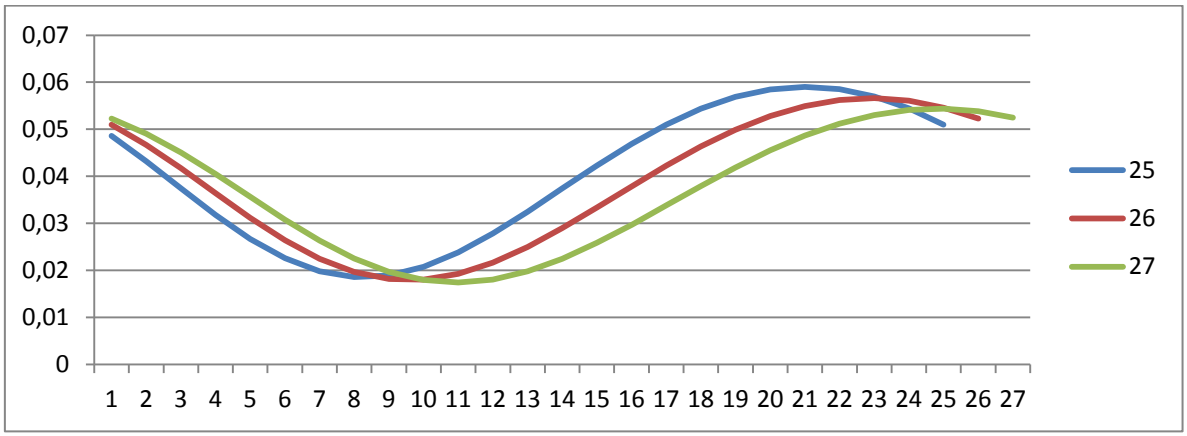


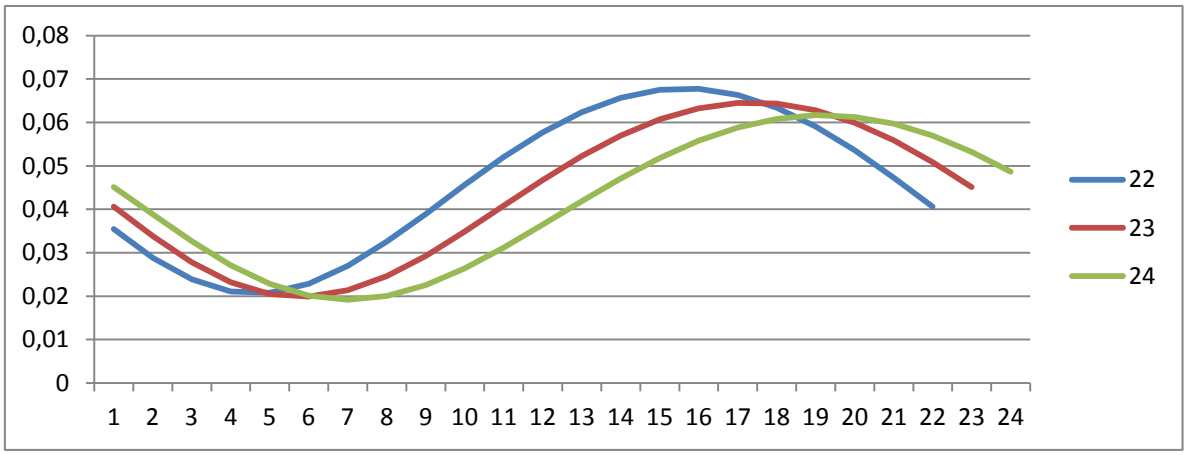


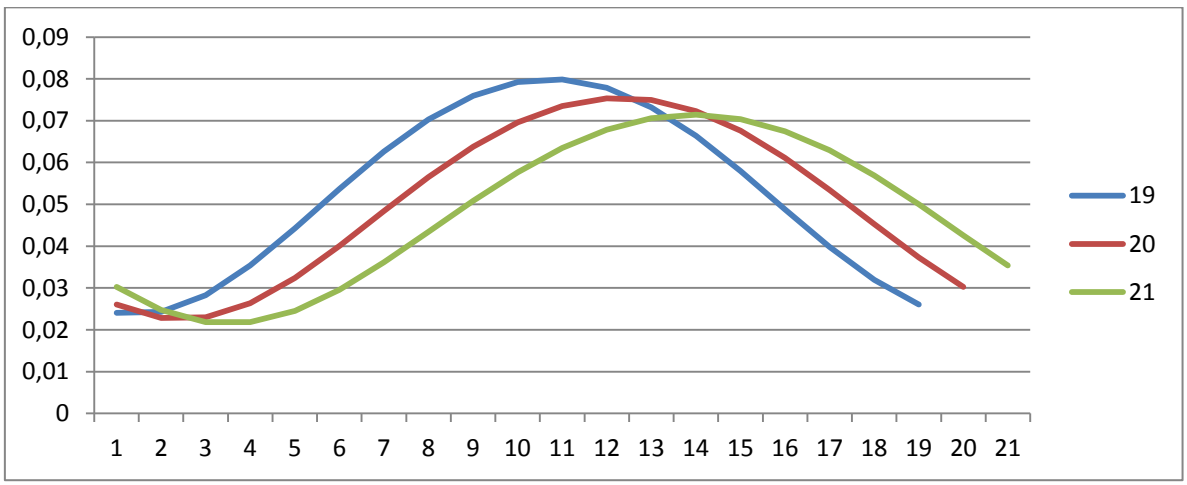





0,12
0,1
0,08
0,06
0,04
0,02
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
16
17
18


0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
25
26
27


0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
22
23
24


0,09
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
19
20
21