МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ ПО ЛАБО**РАТОРНОЙ РАБО**ТЕ № 8**

**по дисциплине**  
 **«МНОГОАГЕНТНОЕ МОДЕЛИРОВАНИЕ»**

Выполнил студент группы 45/2                                  Т. Э. Айрапетов

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Курс    4

Отчет принял доктор физико-математических наук, профессор                                                                                       А.И. Миков

Краснодар

2024 г.

**Задание**:

2150 год, Марс. Есть несколько колоний, управляемых агентами администраторами. Каждая колония имеет следующие характеристики: Уровень (максимум 10), Баланс, Затраты в цикл, Доходы в цикл, Опыт. Максимальное время моделирования T. Цикл состоит из t\_iter итераций. Баланс обновляется в начале цикла (добавляется значение текущего дохода и вычитается значение текущих расходов). Изначально у каждой колонии одинаковый баланс B, но разное соотношение затрат и доходов (доход > затраты). Когда обновляется баланс, также обновляется опыт колонии на разницу между предыдущим балансом и текущим (e=e+b\_current-b\_previous), опыт может уменьшаться, но не может стать отрицательным. Когда значение опыта достигает константного значения L, опыт обнуляется и уровень увеличивается (уровень не может уменьшаться). Условием победы колонии считается достижение максимального уровня. Условием поражения и уничтожения колонии считается уход баланса в минус. Раз в несколько циклов t\_a Земля проводит аукцион (первой или второй цены) артефактов, улучшающих состояние колонии (выигравшие колонии в аукционах не участвуют). Артефакт может, например: увеличить уровень колонии, уменьшить затраты на обслуживание, увеличить доход, спасти колонию от разрушения (не более одного артефакта у каждой колонии одновременно). Изменение может иметь как разовый эффект, так и постоянный, например для увеличения дохода может быть как единовременная выплата, так и выплаты на протяжении нескольких циклов. Также раз в несколько циклов t\_e происходят события среды: пылевая буря - уменьшает доход колонии на g пунктов и увеличивает расход на j пунктов, "ренессанс" - эффект прямо противоположный пылевой буре. Эффект события одноразовый.

**Задача.**

Каждому разработчику выдается набор из 5 артефактов с различными эффектами. Эффекты являются параметризированными, то есть конкретные значения выбираются разработчиком самостоятельно, равно как и стоимость каждого артефакта. Разработать алгоритм участия агента в аукционе. В данном случае понятно, что все артефакты только улучшают текущее состояние колонии, поэтому есть смысл участвовать в каждом из аукционов. Однако, агент должен понимать, что на следующую ставку ему может не хватить баланса, или что покупка может лишь усугубить состояние (например, снижение расхода на h пунктов не выровняет соотношение доход/расход, но баланс станет меньше, что может привести к поражению раньше). Построить график распределения вероятностей времени жизни колонии (до поражения) для различных сочетаний входных параметров. Построить графики зависимостей вероятности побед и поражений колоний в зависимости от входных параметров (провести n экспериментов, к концу моделирования рассчитать количество выигравших и уничтоженных колоний).

Артефакты:

Артефакт 11 :

[

{Эффект: Текущий баланс + {n}% от расходов,

Продолжительность: До следующего аукциона },

{ Эффект: + {n}% к опыту от текущего опыта,

Продолжительность: На протяжении {n} итераций }

],

Артефакт 30 :

[

{ Эффект: Текущий расход - {n} единиц условной валюты,

Продолжительность: На протяжении {n} итераций }

],

Артефакт 32 :

[

{ Эффект: Текущий доход + {n}% от баланса,

Продолжительность: Единоразово },

{ Эффект: Текущий баланс + {n}% от дохода,

Продолжительность: Единоразово },

{ Эффект: Текущий расход - {n} единиц условной валюты,

Продолжительность: На протяжении {n} циклов }

],

Артефакт 35 :

[

{ Эффект: + {n}% к опыту от текущего опыта,

Продолжительность: На протяжении {n} циклов }

],

Артефакт 49 :

[

{ Эффект: Текущий баланс + {n} единиц условной валюты,

Продолжительность: Единоразово },

{ Эффект: + {n}% к опыту от максимального опыта уровня,

Продолжительность: Единоразово },

{ Эффект: Текущий доход + {n}% от расходов,

Продолжительность: На протяжении {n} итераций }

]

Для всех артефактов параметр n был взят равным 10.

**Решение.**

Для моделирования был реализован класс Agent, содержащий поля:

* *ind*  - индекс агента;
* *level* - уровень колонии;
* *xp* - опыт колонии;
* *balance* - баланс колонии;
* *income* - доход колонии;
* *expense* - расход колонии;
* *effects* - текущие эффекты полученные от артефактов.

В классе агента были реализованы вспомогательные методы для пересчёта баланса, опыта и уровня на каждой итерации, а также метод для применения артефактов. Также в классе был реализован метод *bet*, который для заданного артефакта возвращает ставку агента на аукционе. Алгоритм подсчета ставки следующий:

Создается «клон» колонии и предполагается, что артефакт был куплен за цену, равную начальной стоимости артефакта + половина остатка баланса агента. Далее моделируется состояние колонии до тех пор, пока эффекты артефакта не исчерпали себя. По итогам моделирования сравнивается состояние агента до покупки и после по всем показателям (доход, расход, баланс, опыт).

Максимальная ставка делается в случае, когда агент достиг 10 уровня по итогам моделирования, иначе вычисляются отношения между показателями и коэффициент ставки будет равен полезности артефакта по ключевым для него показателям. Например, артефакт 32 увеличивает доходы и уменьшает расходы, следовательно для него коэффициент ставки рассчитывается как max(1 - prev\_income/new\_income, 1-new\_expense/prev\_expense).

Также в коде реализованы методы для генерации событий и проведения аукционов (закрытые 1 или 2 цены). Пример моделирования можно увидеть на рисунке 1.

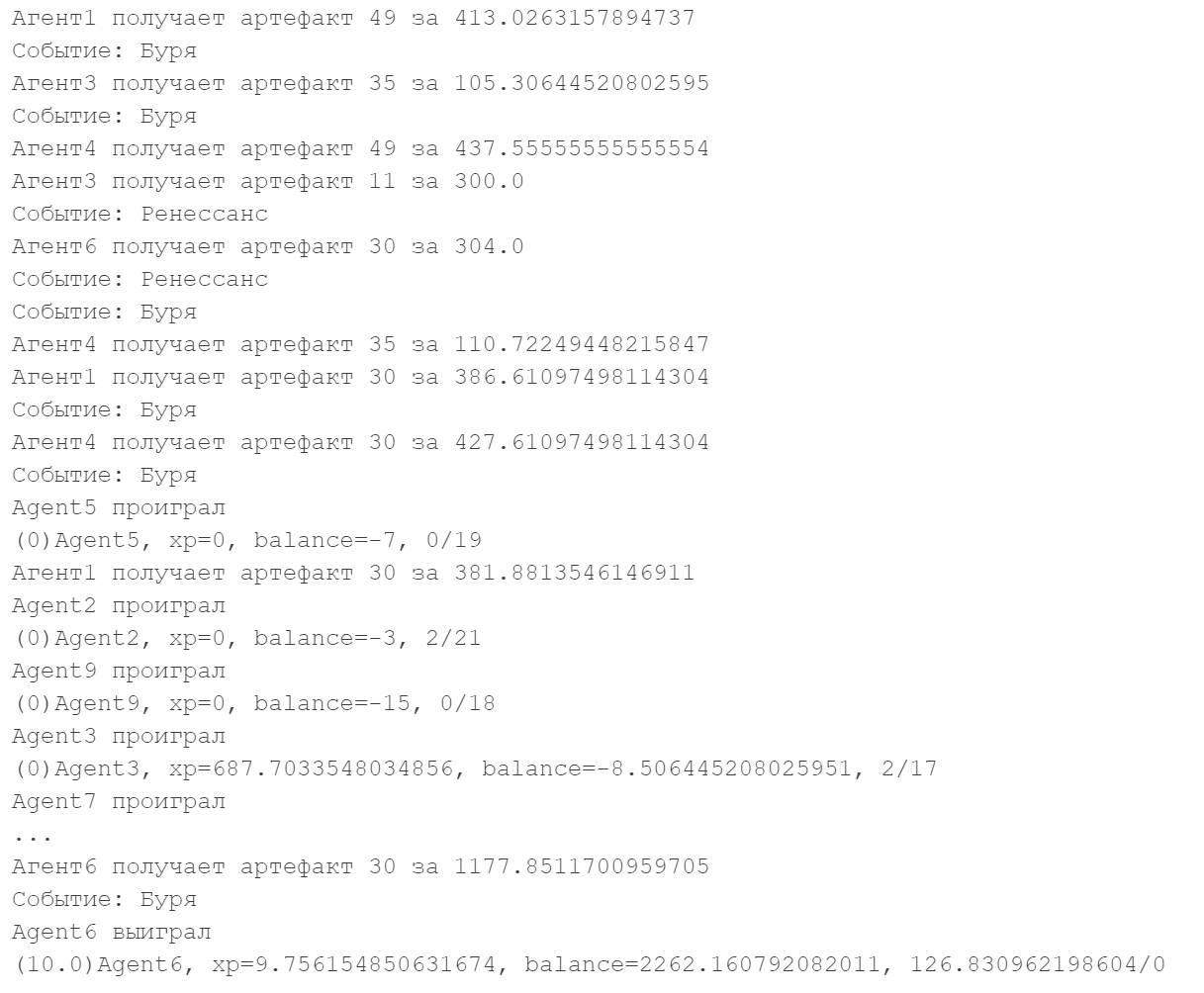


Рисунок 1 - Пример моделирования

На рисунках 2 и 3 можно увидеть графики распределения вероятности времени жизни колонии до проигрыша в зависимости от входных параметров. Для каждого параметра были рассмотрены по 6 значений, а для каждого из значений были рассмотрены средние значения при всех вариациях других параметров (6^5 комбинаций для каждого значения и 6^6 моделирований всего).

На основании построенных графиков можно увидеть, что чем выше параметры n (число агентов) и j (изменение расхода при событии) , тем выше вероятность не дожить до конца моделирования. Обратная ситуация наблюдается с параметрами B (начальный баланс колоний), t\_e (период возникновения события) и g (изменение дохода при событии).

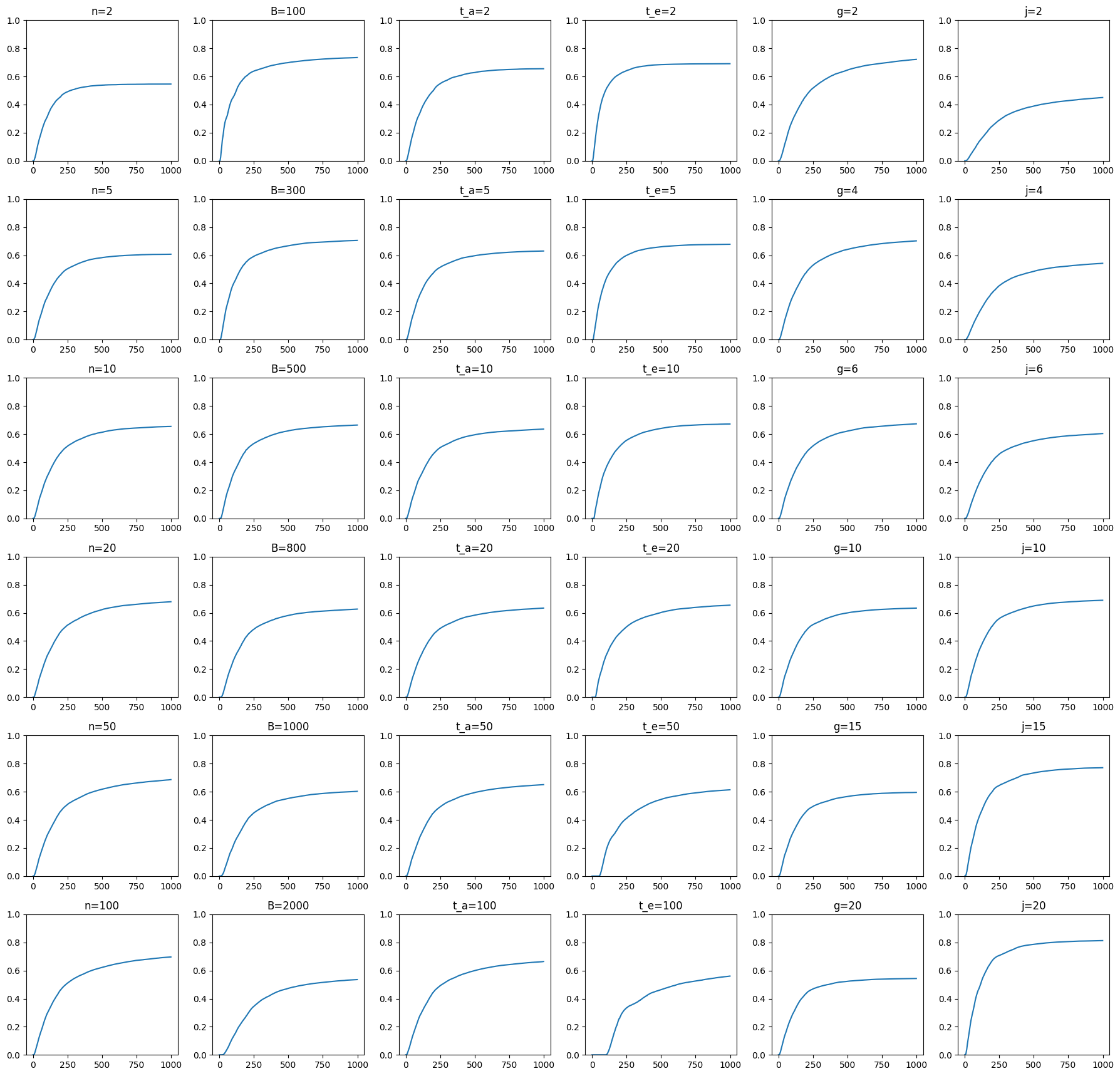


Рисунок 2 - Графики распределения вероятностей времени жизни колонии до проигрыша для параметров n, B, t\_a

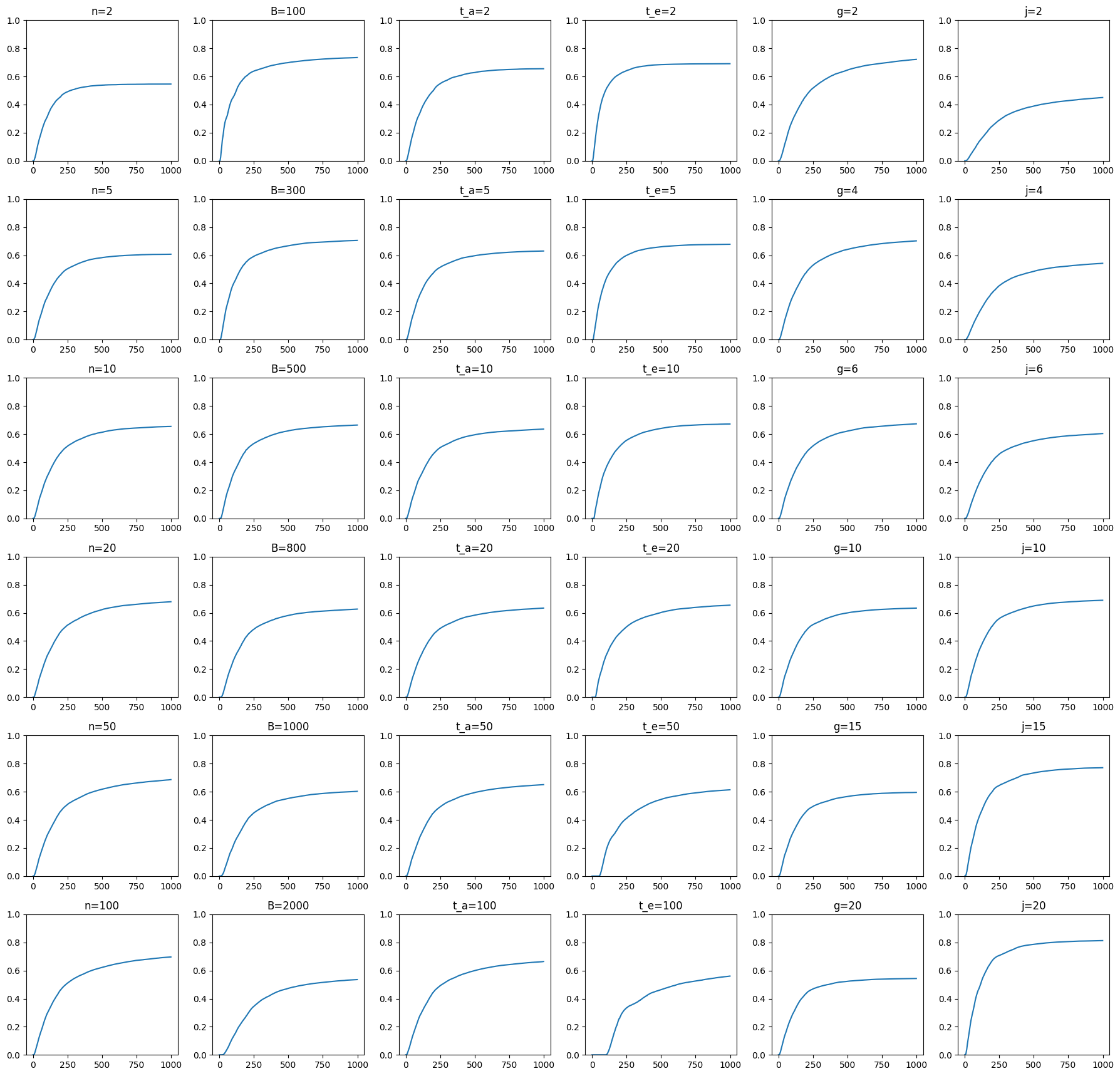


Рисунок 3 - Графики распределения вероятностей времени жизни колонии до проигрыша для параметров t\_e, g, j

На рисунках 4 можно увидеть графики зависимостей вероятности побед и поражений колоний в зависимости от входных параметров. Для всех вариаций входных параметров (по 4 значения для каждого) были проведены по 100 экспериментов и усреднены вероятности выигрыша колонии. Можно заметить, что изменения параметров влияют на вероятность выигрыша также как и на предыдущих графиках.

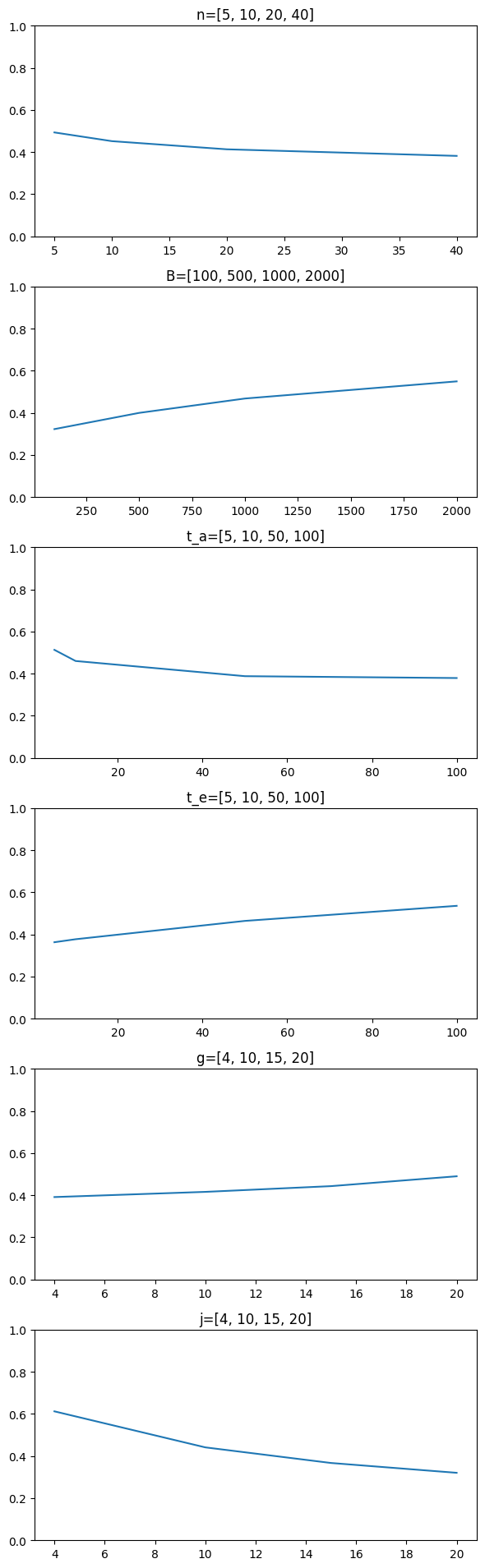
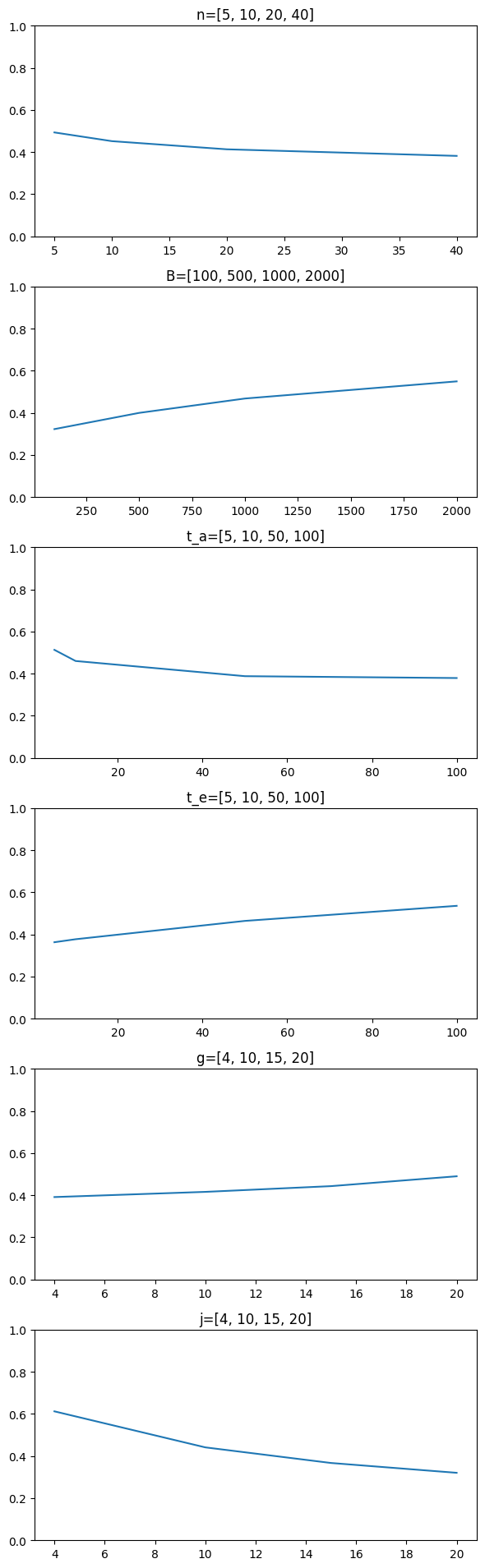


Рисунок 4 - Графики зависимостей вероятности побед и поражений колоний в зависимости от входных параметров

**Выводы.** Из проведенных экспериментов понятно, что для участия в аукционах необходима правильная оценка ценности «лота», а также найдены закономерности влияния параметров на вероятность победы.

Текст программы на языке Python:

*import* numpy *as* np

*import* copy

L = 1000 *# максимальный опыт уровня*

n\_param = 10

*# Текущий баланс + {n}% от расходов*

def effect\_11\_1(*ag*):

*ag*.balance += *ag*.expense\*n\_param/100

*# + {n}% к опыту от текущего опыта*

def effect\_11\_2(*ag*):

*ag*.update\_xp(*ag*.xp\*n\_param/100)

*# Текущий расход - {n} единиц условной валюты*

def effect\_30(*ag*):

*ag*.update\_expense(-n\_param)

*# Текущий доход + {n}% от баланса*

def effect\_32\_1(*ag*):

*ag*.income += *ag*.balance\*n\_param/100

*# Текущий баланс + {n}% от дохода*

def effect\_32\_2(*ag*):

*ag*.balance += *ag*.income\*n\_param/100

*# Текущий баланс + {n} единиц условной валюты*

def effect\_49\_1(*ag*):

*ag*.balance += n\_param

*# + {n}% к опыту от максимального опыта уровня*

def effect\_49\_2(*ag*):

*ag*.update\_xp(L\*n\_param/100)

*# Текущий доход + {n}% от расходов*

def effect\_49\_3(*ag*):

*ag*.income += *ag*.expense\*n\_param/100

artefacts = {

*# увеличение баланса и опыта*

    11: [ 300,

        [-1, effect\_11\_1], *# до следующего аукциона*

        [n\_param, effect\_11\_2]

    ],

*# уменьшение расходов*

    30: [ 200,

        [n\_param, effect\_30]

    ],

*# увеличение баланса и доходов, уменьшение расходов*

    32: [ 500,

        [1, effect\_32\_1],

        [1, effect\_32\_2],

        [n\_param, effect\_30]

    ],

*# увеличение опыта*

    35:[ 100,

        [n\_param, effect\_11\_2]

    ],

*# увеличение баланса, опыта и дохода*

    49:[ 400,

        [1, effect\_49\_1],

        [1, effect\_49\_2],

        [n\_param, effect\_49\_3]

    ]

}

class Agent:

    def \_\_init\_\_(*self*, *ind*, *balance*, *income*, *expense*, *level*=0, *xp*=0):

*self*.ind = *ind*

*self*.level = *level*

*self*.xp = *xp*

*self*.balance = *balance*

*self*.income = *income*

*self*.expense = *expense*

*self*.effects = []

    def update\_income(*self*, *new*):

*self*.income = max(0, *self*.income+*new*)

    def update\_expense(*self*, *new*):

*self*.expense = max(0, *self*.expense+*new*)

    def update\_effects(*self*):

        i = 0

*while* i < len(*self*.effects):

*self*.effects[i][0] -= 1

*# print(self.effects[i])*

            t, f = *self*.effects[i]

            f(*self*)

*if* not t:

*self*.effects.pop(i)

*else*:

                i += 1

*return* (*self*.level >= 10)

*# сделать ставку*

    def bet(*self*, *art\_id*, *price*, *effects*, *verbose*=0):

*if* *price* > *self*.balance:

*return* -1

*# стоимость минус половина остатка*

        new\_balance = (*self*.balance-*price*)/2

        temp\_agent = Agent(0, new\_balance, *self*.income, *self*.expense, *self*.level, *self*.xp)

        temp\_agent.effects = copy.deepcopy(*effects*)

*while* temp\_agent.effects:

            temp\_agent.update\_effects()

            flag = temp\_agent.update\_balance()

*if* flag:

*break*

*if* *verbose*==2:

            print('tempBalance:', new\_balance)

            print(temp\_agent)

            print('Flag:', flag)

        f1 = new\_balance / (temp\_agent.balance *if* temp\_agent.balance *else* 1)

        f2 = *self*.income / (temp\_agent.income *if* temp\_agent.income *else* 1)

        f3 = temp\_agent.expense / (*self*.expense *if* *self*.expense *else* 1)

        f4 = temp\_agent.level - *self*.level

        f5 = (f4\*L + temp\_agent.xp - *self*.xp)/(L\*10)

*if* *verbose*==2:

            print('Balance:', f1)

            print(1/f1)

            print('income:', f2)

            print('expense:', f3)

            print('level:', f4)

            print('xp:', f5)

        coef = -1

*if* flag == 1:

            coef = 1

*elif* *art\_id* in [11, 35]:

            coef = max(f5, 0)

*elif* *art\_id* == 30:

*if* not *self*.expense:

                coef = 1 - f1

*else*:

                coef = 1 - f3

*elif* *art\_id* in [32, 49]:

            coef = max(1-f2, 1-f3)

        bet = -1

*if* coef != -1:

            bet = *price* + new\_balance\*coef

*return* bet

    def update\_balance(*self*):

        prev = *self*.balance

*self*.balance += *self*.income - *self*.expense

*if* *self*.balance < 0:

*return* -1 *# проигрыш*

*return* *self*.update\_xp(*self*.balance-prev)

    def update\_xp(*self*, *new*):

*self*.xp = max(0, *self*.xp + *new*)

*if* *self*.xp >= L:

*self*.level += *self*.xp // L

*self*.xp %= L

*return* (*self*.level >= 10) *# 1 - выигрыш, 0 - продолжаем*

    def \_\_repr\_\_(*self*):

*return* f"({*self*.level})Agent{*self*.ind}, xp={*self*.xp}, balance={*self*.balance}, {*self*.income}/{*self*.expense}"

T = 1000

def gen\_event(*agents*, *g*, *j*, *verbose*=0):

    flag = np.random.randint(2) *# 1 - буря, 0 - ренессанс*

*if* flag:

*g* \*= -1

*j* \*= -1

*if* *verbose*:

        print(f"Событие: {['Ренессанс', 'Буря'][flag]}")

*for* a *in* *agents*:

        a.update\_income(*g*)

        a.update\_expense(-*j*)

def gen\_auction(*agents*, *t\_a*, *verbose*=0):

    art\_id = np.random.choice(list(artefacts.keys()), 1)[0]

    price, \*effects = copy.deepcopy(artefacts[art\_id])

*if* art\_id == 11:

        effects[0][0] = *t\_a*

*# однораундовый аукцион*

    flag = np.random.choice(2) *# первой или второй цены*

    bets = []

*for* i *in* range(len(*agents*)):

        t = *agents*[i].bet(art\_id, price, effects, *verbose*)

*if* t != -1:

            bets.append((i, t))

*if* not bets:

*return*

    bets.sort(*key*=lambda *x*: *x*[1], *reverse*=True)

*if* flag and len(bets)>1: *# аукцион второй цены*

        winner\_id = bets[0][0]

        winner\_price = bets[1][1]

*else*: *# аукцион первой цены*

        winner\_id, winner\_price = bets[0]

    winner = *agents*[winner\_id]

    winner.balance -= winner\_price

*if* *verbose*:

        print(f'Агент{winner.ind} получает артефакт {art\_id} за {winner\_price}')

    winner.effects.extend(effects)

def sim(*n*, *B*, *t\_a*, *t\_e*, *g*, *j*, *verbose*=0):

    agents = []

    times = []

*for* i *in* range(*n*):

        income = np.random.randint(10, 15)

        expense = np.random.randint(5, 10)

        agents.append(Agent(i, *B*, income, expense))

    auction = *t\_a*

    event = *t\_e*

*for* t *in* range(T):

*# применение эффектов и обновление баланса*

        i = 0

*while* i < len(agents):

            agents[i].update\_effects()

            flag = agents[i].update\_balance()

*if* flag:

                a = agents.pop(i)

*if* *verbose*:

*if* flag == -1:

                        print(f"Agent{a.ind} проиграл")

*else*:

                        print(f"Agent{a.ind} выиграл")

                    print(a)

*if* flag == -1:

                    times.append(t)

*else*:

                i += 1

*if* not agents:

*break*

*if* not event:

            event = *t\_e*

            gen\_event(agents, *g*, *j*, *verbose*)

*if* not auction:

            auction = *t\_a*

            gen\_auction(agents, *t\_a*, *verbose*)

        event -= 1

        auction -= 1

*return* t, times

def get\_disrtib(*n*, *B*, *t\_a*, *t\_e*, *g*, *j*, *iters*):

    P = [0]\*T

    k = 0

*for* \_ *in* range(*iters*):

        t, times = sim(*n*, *B*, *t\_a*, *t\_e*, *g*, *j*, *verbose*=0)

        k += len(times)

*for* i *in* times:

            P[i] += 1

*for* i *in* range(1, T):

*if* k:

            P[i] /= k

        P[i] += P[i-1]

*return* P