Моделирование коррупции на основе игры в развернутой форме

Сычев Роман Сергеевич, математика и компьютерные науки (кафедра математического анализа)

Научный руководитель: Глазков Д.В., кандидат ф.-м. н., доцент

Данная работа рассматривает вопросы минимизации сожалений в игровых моделях коррупции на основе игр в развернутой форме. В работе приводится описание моделей и рассматриваются некоторые возможные программные реализации с применением методов объектно ориентированного программирования.

Работа включает три главы.

В первой главе приведено описание алгоритма CFR, который является адаптацией алгоритма минимизации сожалений для игр с неполной информацией. В главе приведены все основные определения и описана процедура минимизации контрафактических сожалений на основе алгоритма Блэквелла. Алгоритм рассматривает конечные игры в развернутой форме с неполной информацией. Реализуется итеративная процедуру, на каждом шаге которой обновляется профиль стратегий игроков. После любого числа итераций можно оценить качество полученных стратегических профилей, вычислив сумму сожалений игроков.

Во второй главе приведено описание двух игровых моделей коррупции. В качестве первого примера выбрана модель раскрытия совместного преступления, и модель коррупции в иерархической структуре в качестве второго примера. Модель раскрытия совместного реализует механизм подкупа чиновника клиентом с возможной проверкой обоих инспектором. Модель коррупции в иерархической структуре использует похожие механизмы распределения информации между игроками, но уже для случая иерархии сотрудгиков. Основным отличием является возможность разоблачения руководителя, для смягчения собственного наказания. Обе модели предусматривают ассиметричные выплаты игрокам.

В третьей главе приведено описание реализации алгоритма CFR для двух игровых моделей коррупции. Для каждой модели приведено описание в соответствии с определениями из первой главы. Для иллюстрации работы алгоритма приведены некоторые результаты работы соответствующих программ, часть кода которых представлена в приложениях.

В заключении сделаны выводы о проделанной работе и подведен итог исследованию. В ходе работы были полученна программная реализация минимизации сожалений для двух задач моделирования коррупции.

Реферат

Дипломная работа «Моделирование коррупции на основе игры в развернутой форме» 57 стр., 13 илл., 8 источников, 3 прил.

Ключевые слова: теория игр, искуственный интелект, моделирование коррупции, неполная информация, игры в развернутой форме.

объектом исследования являются игровые модели коррупции на основе игр в развернутой форме.

Целью работы является программная реализация минимизации сожалений для игровых моделей коррупции.

В данной работе была рассмотрена реализация минимизации сожалений для некоторых задач моделирования коррупции. Полученные в результате работы программные классы могут использоваться для практического расчета. К приемуществам полученной иерархической модели можно отнести масштабируемость с точки зрения проверяемых лиц и свободный выбор альтернатив игроками. Полученные алгоритмы и методики могут быть использованы как для изучения старых, так и для построения новых моделей и абстракций.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«Ярославский государственный университет им. П.Г. Демидова»

Выпускная квалификационная работа

Моделирование коррупции на основе игры в развернутой форме

02.04.01. Математика и компьютерные науки

Исполнитель: Сычев Р.С.
гр. МКН-21 МО
Руководитель: к.ф.-м.н. Глазков Д.В.

Ярославль 2022

Содержание

Введение

В настоящее время под явлением коррупции понимается неправомерное использование должностных привилегий в личных целях. Например, к коррупции относят присвоение ренты и получение взяток. При этом, некоторые из этих процессов могут быть описаны с помощью моделей теории игр. Одной из таких моделей является модель совместного преступления[1]. Такой подход подразумевает эндогенную природу игровых ситуаций и ассиметричные выплаты. Похожий вопрос, связанный с механизмом повторной проверки, в достаточно большом объеме изучен для однородного набора из нескольких проверяемых лиц[2]. В то же время, проверяемые лица могут иметь свою организационную структуру, наличие которой может вносить значительные коррективы в распределение информации на различных этапах игры. Например, за счет иерархии можно реализовать механизм дополнительных проверок[3]. Имеет смысл построение и анализ более обобщенных с точки зрения организационной структуры и распределения информации моделей.

Подобные модели могут быть представлены как игры в развернутой форме. При такой постановке задачи можно близким к естественному способом отразить в игровой форме структуру последовательного принятия решений набором участников в конфликтной ситуации.

Существуют методы позволяющие достаточно эффективно сформировать приближенное равновесие для заданной игры в развернутой форме с неполной информацией. Довольно популярен итеративный алгоритм минимизации контрафактических сожалений (Conterfactual Regret Minimization)[4] и его модификация предусматривающая использование метода Монте-Карло (MCCFR)[5]. Данные алгоритмы появились не так давно, но на их основе уже получен ряд недостижимых до этого по сложности результатов.

Целью данной работы является программная реализация минимизации сожалений для игровых моделей коррупции.

В соответствии с темой работы были поставлены следующие задачи:

— выбор конкретной формы модели с использованием уже существующихи;
— выбор алгоритма для поиска решения и анализа модели;
— проведение расчетов и анализ результатов.

Данная работа может быть интересна людям желающим ознакомится с некоторыми современными техниками решения игр с неполной информацией.

1 Первая глава. Описание алгоритмов

1.1 Игры в развернутой форме и равновесие

Игра в развернутой форме представляют компактную общую модель взаимодействий между агентами и явно отражает последовательный характер этих взаимодействий. Последовательность принятия решений игроками в такой постановке представлена деревом решения. При этом, листья дерева отождествлены с терминальными состояниями, в которых игра завершается и игроки получают выплаты. Любой нетерминальный узел дерева представляет точку принятия решения. Неполнота информации выражается в том, что различные узлы игрового дерева считаются неразличимыми для игрока. Совокупность всех попарно неразличимых состояний игры называется информационными состояниями. Приведем формальное определение.

Определение 1 *Конечная игра в развернутой форме с неполной информацией содержит следующие компоненты:*

— конечное множество игроков N;

— конечное множество историй действий игроков H, такое, что $\emptyset \in H$ и любой префикс элемента из H также принадлежит H. $Z \subseteq H$ представляет множество терминальных историй (множество историй игры на являющихся префиксом). $A(h) = \{a \colon (h, a) \in H\}$ — доступные после нетерминальной истории $h \in H$ действия;

— функция $P \colon H \setminus Z \to N \cup \{c\}$, которая сопоставляет каждой нетерминальной истории $h \in H \setminus Z$ игрока, которому предстоит принять решение, либо игрока c представляющего случайное событие;

— функция f_c, которая сопоставляет всем $h \in H$, для которых $P(h) = c$, вероятностное распределение $f_c(\cdot|h)$ на $A(h)$. $f_c(a|h)$ представляет вероятность выбора a после истории h;

— для каждого игрока $i \in N$ $\mathcal{I}_i$ обозначает разбиение $\{h \in H \colon P(h) = i\}$, для которого $A(h) = A(h')$ всякий раз когда h и h' принадлежат одному элементу разбиения. Для $I_i \in \mathcal{I}_i$ определим $A(I_i) = A(h)$ и $P(I_i) = i$ для всех $h \in I_i$. $\mathcal{I}_i$ называют информационным набором игрока i, а $I_i \in \mathcal{I}_i$ информационным состоянием игрока i;

— для каждого игрока $i \in N$ определена функция выигрыша $u_i \colon Z \to \mathbb{R}$. Если для игры в развернутой форме выполняется $\forall z \in Z \sum_{i \in N} U_i(z) = 0$, то такую игру называют игрой с нулевой суммой. Определим $\Delta_{u,i} = max_{z \in Z}\ u_i(z) - min_{z \in Z}\ u_i(z)$ для диапазона выплат игрока.

Отметим, что информационные наборы могут использоваться не только для реализации правил конкретной игры, но и могут быть использованы для того, чтобы заставить игрока забыть о предыдущих действиях. Игры в которых игроки не

забывают о действиях называют играми с полной памятью. В дальнейшем мы будем рассматривать конечные игры в развернутой форме с полной памятью.

Стратегия игрока i — это функция σ_i, которая ставит в соответствие каждому информационному состоянию $I_i \in \mathcal{I}_i$ вероятностное распределение на $A(I_i)$. Обозначим за Σ_i множество всех стратегий игрока i. Стратегический профиль σ содержит стратегии для каждого игрока $i \in N$. При этом за σ_{-i} обозначим σ без σ_i.

Обозначим за $\pi^\sigma(h)$ вероятность того, что игроки достигнут h руководствуясь σ. Мы можем представить π^σ как $\pi^\sigma = \prod_{i \in N \cup \{c\}} \pi_i^\sigma(h)$, выделяя вклад каждого игрока. В таком случае, $\pi_i^\sigma(h)$ обозначает вероятность принятия совокупности решений игрока i, ведущих от $\emptyset$ к h. Иными словами

$$\pi_i^\sigma(h) = \begin{cases} \prod_{h \sqsubset h' \wedge P(h')=i \wedge h \sqsubset (h',a)} \sigma(h')(a) & \{h' | h \sqsubset h' \wedge P(h') = i\} \neq \emptyset \\ 1 & \text{иначе.} \end{cases}$$

Запись $h \sqsubset h'$ означает, что h' является префиксом h. Обозначим за $\pi_{-i}^\sigma(h)$ вероятность достижения истории h всеми игроками (включая c) за исключением i. Для $I \subseteq H$ определим $\pi^\sigma(I) = \sum_{h \in I} \pi^\sigma(h)$. Аналогично, введем $\pi_i^\sigma(I)$ и $\pi_{-i}^\sigma(I)$.

Ожидаемое значение выплаты для игрока i обозначим как $u_i(\sigma) = \sum_{h \in Z} u_i(h)\pi^\sigma(h)$.

Традиционным способом решения игр в развернутой форме является поиск равновесного профиля стратегий σ, который удовлетворяет следующему условию

$$\forall i \in N, \;\; u_i(\sigma) \geqslant \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i}). \tag{1.1}$$

Такой стратегический профиль называют равновесием по Нэшу. В случае, если стратегический профиль σ удовлетворяет условию

$$\forall i \in N, \epsilon > 0, \;\; u_i(\sigma) + \epsilon \geqslant \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i}). \tag{1.2}$$

его называют ϵ – равновесием по Нэшу.

Для рассматриваемых далее алгоритмов наиболее интересен вариант игры с нулевой семмой для двух игроков. Именно для него имеется строгое математическое обоснование сходимости к равновесию Нэша.

1.2 Контрафактические сожаления и их минимизация

Минимизация сожалений является популярным концептом, для построения итеративных алгоритмов приближенного решения игр в развернутой форме [6]. Приведем связанные с ней определения. Рассмотрим дискретный отрезок времени T включающий T раундов от 1 до T. Обозначим за σ_i^t стратегию игрока i в раунде t.

Определение 2 *Средним общим сожалением игрока i на момент времени T называют величину*

$$R_i^T = \frac{1}{T}\max_{\sigma_i^* \in \Sigma_i} \sum_{t=1}^{T} u_i(\sigma_i^*, \sigma_{-i}^t) - u_i(\sigma^t) \tag{1.3}$$

В дополнении к этому, определим $\bar{\sigma}_i^T$ как среднюю стратегию относительно всех раундов от 1 до T. Таким образом для каждого $I \in \mathcal{I}_i$ и $a \in A(I)$ определим

$$\bar{\sigma}_i^T(I) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I)(a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)}. \tag{1.4}$$

Теорема 1 *Если для игры с двумя игроками и с нулевой суммой на момент времени T средние общие сожаления игроков меньше ϵ, то σ является 2ϵ равновесием [4].*

Говорят, что алгоритм выбора σ^t реализует минимизацию сожалений, если средние общие сожаления игроков стремятся к нулю при t стремящимся к бесконечности. И как результат, алгоритм минимизации сожалений может быть использован для нахождения приближенного равновесия по Нэшу, в случае игр двух игроков с нулевой суммой. Вообще говоря, в случае ненулевой суммы или большего числа игроков алгоритм минимизации сожалений не приводит к равновесию Нэша. Однако, он приводит к другому классу равновесий. Доказано, что полученное решение сходится к грубому коррелированному равновесию и, более того, устраняет итеративно доминируемые действия в профилях стратегий[7].

Понятие контрафактического сожаления служит для декомпозиции среднего общего сожаления в набор дополнительных сожалений, которые могут быть минимизированы независимо для каждого информационного состояния.

Обозначим через $u_i(\sigma, h)$ цену игры с точки зрения истории h, при условии, что h была достигнута, и игроки спользуют в дальнейшем σ.

Определение 3 *Контрафактической ценой $u_i(\sigma, I)$ назовем ожидаемую цену, при условии, что информационное состояние I было достигнуто, когда все игроки кроме i играли в соответствии с σ. Формальноо*

$$u_i(\sigma, I) = \sum_{h \in I, h' \in Z} \pi_{-i}^{\sigma}(h)\pi^{\sigma}(h, h')u_i(h'), \tag{1.5}$$

где $\pi^{\sigma}(h, h')$ — вероятность перехода из h в h'.

Обозначим за $\sigma^t|_{I \to a}$ стратегический профиль идентичный σ за исключением того, что i всегда выбирает a в I.

Средним немедленным контрафактическим сожалением назовем

$$R_{i,imm}^T(I) = \frac{1}{T}\max_{a \in A(I)} \sum_{t=1}^{T} u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I). \tag{1.6}$$

Интуитивно это выражение можно понимать как аналог среднего общего сожаления в терминах контрафактической цены. Однако, вместо рассмотрения всевозможных максимизирующих стратегий рассматриваются локальные модификации стратегии. Положим $R^{T,+}_{i,imm}(I) = \max(R^T_{i,imm}(I), 0)$. Связь немедленных контрафактических сожалений и общих средних сожалений раскрывает следующая теорема.

Теорема 2 $R^T_i \leqslant \sum_{I \in \mathcal{I}_i} R^{T,+}_{i,imm}(I)$*[4].*

Минимизация средних немедленных контрафактических сожалений приводит к минимизации средних общих сожалений. В свою очередь, минимизация среднего немедленного контрафактического сожаления может происходить за счет минимизации выражений под функцией максимума. Таким образом, мы приходим к понятию среднего контрафактического сожаления

$$R^T_i(I, a) = \frac{1}{T} \sum_{t=1}^{T} u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I). \tag{1.7}$$

Контрафактическое сожаление рассматривает действие в информационном состоянии. В свою очередь, для минимизации средних контрафактических сожалений можно применить алгоритм приближения Блэквела[6], который приведет к следующей последовательности стратегий

$$\sigma^{T+1}_i(I)(a) = \begin{cases} \frac{R^{T,+}_i(I, a)}{\sum_{a \in A(I)} R^{T,+}_i(I, a)} & \text{если } \sum_{a \in A(I)} R^{T,+}_i(I, a) > 0, \\ \frac{1}{|A(I)|} & \text{иначе.} \end{cases} \tag{1.8}$$

Другими словами, действие выбирается в пропорции соотношения позитивных контрафактических сожалений о не выборе этого действия. Обоснование сходимости полученного решения и оценку ее скорости предоставляет следующая теорема.

Теорема 3 *Если игроки придерживаются стратегий, заданных выражением (1.8), то* $R^T_{i,imm}(I) \leqslant \Delta_{u,i}\sqrt{|A_i|}/\sqrt{T}$ *и, следовательно,* $R^T_i \leqslant \Delta_{u,i}|\mathcal{I}_i|\sqrt{|A_i|}/\sqrt{T}$*, где* $|A_i| = \max_{h:\, P(h)=i} |A(h)|$*[4].*

Таким образом, по мере увеличения числа проведенных итераций уменьшаются средние общие сожаления.

2 Вторая глава. Описание моделей

2.1 Модель раскрытия совместного преступления

Модель раскрытия совместного преступления, предложенная Спенглером [1], предполагает игру в развернутой форме для трех игроков с эндогенным характером формирования игровых историй. Игра построена следующим образом. В игре участвуют три игрока: клиент(C), чиновник(O) и инспектор(I). Игру начинает клиент. Клиент может подкупить чиновника с вероятностью γ или нет с вероятностью $1-\gamma$. Чиновник, в случае подкупа, может ответить взаимностью с вероятностью β или нет с вероятностью $1-\beta$. Взаимность определяется как акт возврата благосклонности за взятку, то есть возвращение некоторых привелегий (например, государственный контракт) клиенту. Коррупция, как взаимное взяточничество, может произойти только при совместных усилиях клиента и чиновника. Игра включает в себя четыре штрафа, один за подкуп p_L и один для получения взаимности q_L (штрафы клиента), а также один для принятия взятки и взаимности q (штраф чиновника). Это позволяет использовать асимметричное распределение штрафов. Штрафы применяются с вероятностью инспектирования, которая представлена действием инспектора. Таким образом инспектор может провести проверку с вероятностью α, либо не проводить с вероятностью $1-\alpha$. При этом награда инспетора зависит факта инспектирования и наличия преступления.

Для подкупа чиновника клиент тратит b на взятку и получает выгоду от взаимности чиновника в размере v. Чиновник в случае подкупа получает взятку в размере b , а неотвечая взяимностью получает r. Параметр r играет роль нейтральной выплаты, для подкрепления непринятия взятки и может быть расценен как моральное удавлетворение от несовершения преступления. В случаях выявленного преступления клиент и чиновник должны выплатить соответствующие штрафы. Распределение выплат инспектору показано в таблице 2.1.

История игры	Провести проверку: α	Не проводить: $1-\alpha$
Взаимная взятка: $\gamma\beta$	$x+\Delta x$	x
Невзяимная взятка: $\gamma(1-\beta)$	$y+\Delta y$	y
Не было взятки: $(\gamma-1)$	z	$z+\Delta z$

Таблица 2.1 — Схема распределения выплат инспектору

Предполагается, что проверка приводит к лучшим для инспектора результатам в случае (взаимного) взяточничества, чем в случае отсутствия взяточничества и наоборот: $0<\Delta x, \Delta y, \Delta z$.

Для клиента мы предполагаем, что взяточничество является прибыльным, если оно встречает взаимность, но не с проверкой, где b - взятка, а v - это выгода от

взаимного обращения с клиентом. Это подразумевает, что $0 < b < v$ и $0 < p_L$, p_H и $v - b - p_L - p_H < 0$. Для инспектора предполагается, что проверка является прибыльной, если по крайней мере один правонарушитель совершает правонарушения, но обходится дорого, если нет. Это подразумевает, что $x < 0 < x + \Delta x$ и $y < 0 < y + \Delta y$, но $z < 0 < z + \Delta z$. Эта настройка отражает интуицию, что успешный осмотр стоит того, поскольку он приводит к продвижению или аналогичным выгодам, в то время как безуспешная проверка просто стоит усилий. Сложность модели требует, чтобы мы сделали некоторые дополнительные предположения о выплатах инспектора. Мы предполагаем, что проверка взаимного взяточничества является более прибыльной, чем проверка простого взяточничества, и, аналогично, что не проверки взаимного взяточничество несет большую потерю, чем не проверка простого взяточничества из-за более высокой альтернативной стоимости неинспекции. Это подразумевает, что $0 < y + \Delta y < x + \Delta x$ и $x < y < 0$. Для чиновника мы предполагаем, что получение взятки является прибыльным, пока нет проверки, подразумевая, что $0 < r < b < q$.

Последовательность выборов игроков и распределение выплат данной игры могут быть представлены в развернутой форме. Развернутая форма игры представлена на рисунке 2.1.

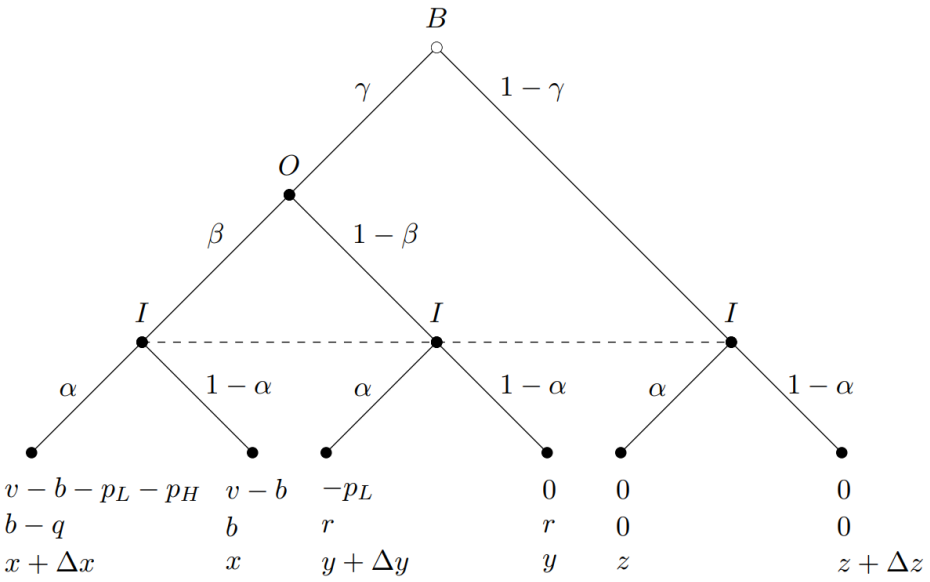

Рисунок 2.1 — Развернутая форма игры

Рассмотрим равновесие данной игры. Чтобы алгебраически выразить равновесие, для каждого игрока приравняем выплаты каждой стратегии. Используя выплаты на рисунке 2.1, мы получаем уравнения (2.1), (2.2) и (2.3), связанные с выплатами клиента, чиновника и инспектора соответственно.

$$\beta(v - b - \alpha(p_L + p_H)) + (1 - \beta)(-\alpha p_L) = 0 \tag{2.1}$$

$$\alpha(b - q) + (1 - \alpha)b = \alpha r + (1 - \alpha)r \tag{2.2}$$

$$\gamma\beta(x+\Delta x)+\gamma(1-\beta)(y+\Delta y)+(1-\gamma)z=\gamma\beta x+\gamma(1-\beta)y+(1-\gamma)(z+\Delta z) \quad (2.3)$$

Мы получаем следующие вероятности равновесия для v, β и α из предыдущих уравнений:

$$\beta=\frac{\alpha p_L}{v-b-\alpha p_H} \quad (2.4)$$

$$\alpha=\frac{b-r}{q} \quad (2.5)$$

$$\gamma=\frac{\Delta z}{\beta(\Delta x-\Delta y)+\Delta y+\Delta z} \quad (2.6)$$

Дальнейшие преобразования (2.4-2.6) позволяют получить выражения β и γ заисящие от параметров:

$$\alpha=\frac{(b-r)p_L}{(v-b)q-(b-r)p_H} \quad (2.7)$$

$$\gamma=\frac{((v-b)q-(b-r)p_H)\Delta z}{(b-r)p_L(\Delta x-\Delta y)+((v-b)q-(b-r)p_H)(\Delta y+\Delta z)} \quad (2.8)$$

Относительно полученных значений можно отметить следующие предположения:

— для клиента вероятность предложения взятки должна быть равна нулю, если вероятность принятия меньше вероятности принятия/возврата в уравнении (2.4) и равна одному в обратном случае;

— для чиновника вероятность принятия должна быть равна нулю, если вероятность проверки больше, чем вероятность проверки в уравнении (2.5) и равен одному в обратном случае;

— для инспектора вероятность проверки должна быть равна нулю, если вероятность предложения взятки меньше, чем вероятность предложения взятки в уравнении(2.6) и равна единице в обратном случае.

Помимо приведенного выше примера, также рассматривается вариант данной игры, в котором не проводится проверка после отклонения взятки, а сразу выписывается штраф [1]. Но в силу того, что проверка коррупции часто инициируется заранее, то факт проверки часто не зависит от факта предложения взятки. Таким образом ограничемся только этим более обобщенным вариантом.

2.2 Модель коррупции в иерархической структуре

Орлов[3] предложил иерархическую модель коррупции для отражения дополнительной связи между руководителями и подчиненными. В некоторых элементах этой модели прослеживается сходство с моделью разоблачения совместного преступления, но вместо одного клиента указывается их иерархическая структура. В данной работе возьмем за основу эту модель, но, как и в прошлом примере, попробуем

эндогенезировать действия игроков. Коррупция моделируется как иерархическая игра, которая состоит из двух стадий: кража и проверка. Предполагается, что игроки действуют из соображений нейтрального риска и максимизации личного выигрыша. Это предполагает фиксированный размер кражи, который выбирается оптимальным образом до начала игры. Множество игроков содержит иерархию сотрудников, чиновника и инспектора. Рассматриваются только денежные выплаты. Модель предполагает, что иерархия сотрудников представляет из себя древовидную структуру, в которой от корня к вершинам ведут связи от начальника к подчиненному. Зададим иерархию сотрудников следующими элементами:

— Конечное множество различных сотрудников C;

— $s\colon C \to \mathbb{R}$ — сумма средств, которую сотрудник может присвоить себе не прекращая производственный процесс;

— $b\colon C \to \mathbb{R}$ — размер взятки, которую сотрудник готов предложить проверяющему чиновнику;

— $d\colon C \to C$ — руководитель сотрудника. В случае, если $b(c) = c$, то c не имеет руководства.

Будем придерживаться следующих ограничений: $\forall c \in C \quad 0 < b(c) < s(c)$, $s(c) < s(d(c))$, если $b(c) \neq c$ и $d(c1) = d(c2) => s(c1) = s(c2)$. Таким образом, взятка не может превосходить украденную сумму, а подчиненный не может располагать большей суммой, чем руководитель. Данные ограничения соответствуют концепции нейтрального риска[3].

Для формирования выплат игроков введем дополнительные неотрицательные числовые параметры:

— co — стоимость проверки сотрудника;

— ci — стоимость проверки инспектора;

— fs — размер штрафа, который сотрудник должен выплатить за выявленную кражу;

— fb — размер штрафа, который сотрудник получает за предложение взятки;

— fq — размер штрафа, который получает чиновник за получение взятки;

— fns — штраф чиновника за каждую совершенную кражу;

— fe — штраф за ложный донос;

— fbc — размер штрафа инспектору за непроверенную взятку чиновника;

— rs — награда, которую чиновник получает за выявленную кражу;

— rb — награда, которую чиновник получает за отклонение взятки;

— rq — награда, которую получает инспектор за выявленную взятку чиновника.

Помимо каждого сотрудника из иерархии, в игре участвуют еще 2 игрока: чиновник (O) и инспектор (I). Чиновник может провести проверку сотрудников, а ин-

спектор может проверить факт получения взятки чиновником. Саму игру разобьем на три этапа.

На первом этапе инспектор может инициировать проверку чиновника, либо не инициировать. В случае, если начата проверка, инспектор теряет сумму ci. Факт проведения проверки чиновника гарантирует, что полученная чиновником взятка будет замечена и все участники этого преступления понесут соответствующие наказания. За невыявленную взятку, инспектору начисляется штраф в размере fq.

На втором этапе компания выделяет сумму сотрудникам, так, что каждый сотрудник $c \in C$ имеет возможность присвоить $s(c)$. Затем, каждый сотрудник c может совершить кражу в размере $s(c)$. Обозначим за S_c обьем кражи сотрудника c, которую он совершил на данном этапе. $S_c = 0$, если сотрудник c не совершил кражу. По итогу данного этапа чиновнику начисляется штраф за все совершенные кражи $fns * \big|\{c \in C | S_c > 0\}\big|$.

На третьем этапе чиновник принимает решение о проверке одного из сотрудников. В некоторых предыдущих работах [3], предполагается, что проверка проводится с вероятностью, которая зависит от объема краж всех вышестоящих сотрудников. Но в данной работе такой подход не рассматривается, и чиновник самостоятельно принимает решение. Чиновник может и вовсе не проводить проверку. Проверенного сотрудника обозначим как c_{insp}. Если проверка не проводилась или если $S_{c_{insp}} = 0$, то игра заканчивается, и игроки получают свои выплаты. В случае, если $S_{c_{insp}} > 0$, игра продолжается и c_{insp} может выбрать одну из следующих альтернатив:

— признать свое преступление и выплатить штраф fs;

— предложить чиновнику взятку в размере $b(c_{insp})$, в обмен на сокрытие своей кражи;

— попытаться разоблачить руководителя.

Разоблачение руководителя известно руководителю, но он точно не знает разоблачившего его подчиненного. Обозначим разоблаченного руководителя как c_{boss}. Если c_{boss} не совершал кражу, то c_{insp} получает дополнительный штраф за ложный донос. Если c_{boss} совершил кражу он может либо выплатить штраф, либо предложить взятку.

Если сотрудник предложил взятку, инспектор может либо принять взятку, либо отклонить. Если чиновник принимает взятку, то он получает ее сумму, если не инициирована проверка инспектором, а клиенты сохраняют украденное. Если чиновник берет взятку во время проверки инспектором, то он не получает сумму взятки и должен выплатить штраф, а сотрудник получает ту же выплату, что и в случае отклонения взятки. В случае отклонения взятки, сотрудник выплачивает штрафы, и за взятку, и за кражу. При этом, если инспектор не принимает взятку, то все проверенные сотрудники возвращают украденное.

Для наглядной иллюстрации применения всех этих правил рассмотрим один пример. Возьмем иерархию из двух сотрудников, один будет подчиняться другому.

$$C = \{C_0, C_1\},$$

$$\begin{aligned} s(C_0) &= s_0, \\ s(C_1) &= s_1, \\ b(C_0) &= b_0, \\ b(C_1) &= b_1, \\ d(C_0) &= d(C_1) = C_0. \end{aligned} \tag{2.9}$$

Тогда в игре будут участвовать 4 игрока: сотрудник C_0, сотрудник C_1, чиновник O и инспектор I. Схематичное изображение развернутой формы этой игры представлено на рисунках 2.2 и 2.3. Изображение разбито на 2 части, в зависимости от выбора инспектора.

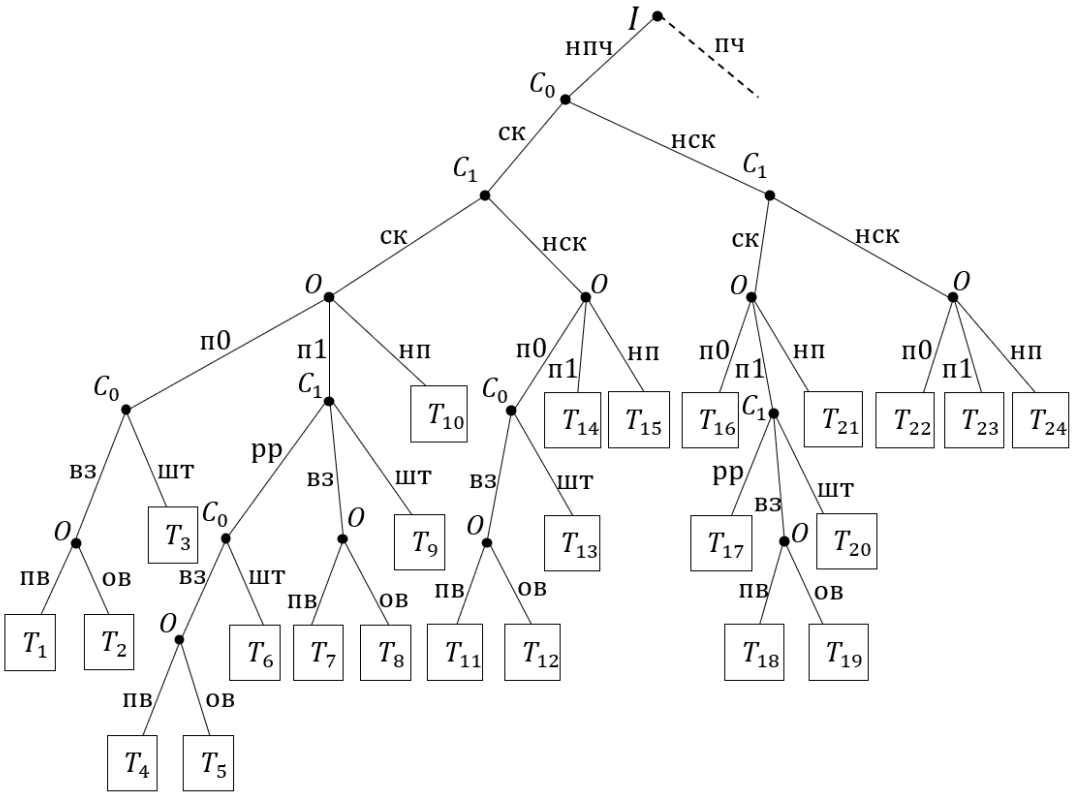

Рисунок 2.2 — Развернутая форма игры. Часть 1

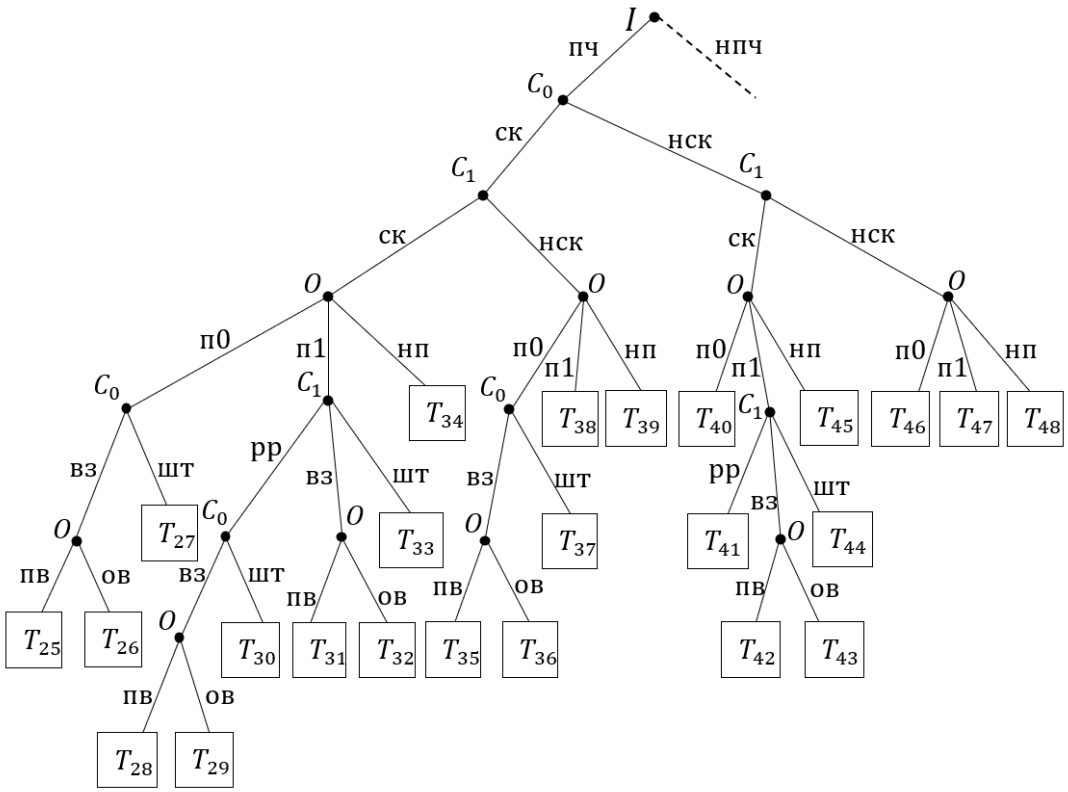

Рисунок 2.3 — Развернутая форма игры. Часть 2

На рисунках 2.2 и 2.3 возле ребер подписаны соответствующие им действия. Их можно расшифровать так:

— пч — проверять чиновника;
— нпч — не проверять чиновника;
— ск — совершить кражу;
— нск — не совершать кражу;
— п0 — проверить сотрудника C_0;
— п1 — проверить сотрудника C_1;
— нп — не проверять сотрудников;
— рр — разоблачить руководителя;
— вз — предложить взятку;
— шт — выплатить штраф;
— пв — принять взятку;
— ов — отклонить взятку.

На рисунках 2.2 и 2.3 буквами T обозначены все 48 различных выплат в терминальных узлах. Ниже представлены величины этих выплат.

$T_1 = (s_0 - b_0, s_1, b_0 - 2 * fns - co, -fbc)$

$T_2 = (-b_0 - fs - fb, s_1, rs + rb - fns - co, 0)$

$T_3 = (-b_0 - fs, s_1, rs - fns - co, 0)$

$T_4 = (s_0 - b_0, s_1, b_0 - fns * 2 - co, -fbc)$

$T_5 = (-b_0 - fs - fb, 0, rs * 2 + rb - co, 0)$ $\quad$ $T_6 = (-fs, 0, rs * 2 - co, 0)$

$T_7 = (s_0, s_1 - b_1, b_1 - fns * 2 - co, -fbc)$

$T_8 = (s_0, -b_1 - fs - fb, rs + rb - fns - co, 0)$

$T_9 = (s_0, -fs, rs - co, 0)$ $\quad$ $T_{10} = (s_0, s_1, -fns * 2, 0)$

$T_{11} = (s_0 - b_0, 0, b_0 - fns - co, -fbc)$ $\quad$ $T_{12} = (-b_0 - fs - fb, 0, rs + rb - co, 0)$

$T_{13} = (-fs, 0, rs - co, 0)$ $\quad$ $T_{14} = (s_0, 0, -fns - co, 0)$

$T_{15} = (s_0, 0, -fns, 0)$ $\quad$ $T_{16} = (0, s_1, -fns - co, 0)$

$T_{17} = (0, -fs - fe, rs - co, 0)$ $\quad$ $T_{18} = (0, s_1 - b_1, b_1 - fns - co, -fbc)$

$T_{19} = (0, -b_1 - fs - fb, rs + rb - co, 0)$ $\quad$ $T_{20} = (0, -fs, rs, 0)$

$T_{21} = (0, s_1, -fns, 0)$ $\quad$ $T_{22} = (0, 0, -co, 0)$

$T_{23} = (0, 0, -co, 0)$ $\quad$ $T_{24} = (0, 0, 0, 0)$

$T_{25} = (-b_0 - fs - fb, s_1, fq - fns * 2 - co, rq - ci)$

$T_{26} = (-b_0 - fs - fb, s_1, rs + rb - fns - co, -ci)$

$T_{27} = (-fs, s_1, rs - fns - co, -ci)$ $\quad$ $T_{28} = (-fs - fb, 0, fq - co, rq - ci)$

$T_{29} = (-fs - fb, 0, rs * 2 + rb - co, -ci)$ $\quad$ $T_{30} = (-fs, 0, rs * 2 - co, ci)$

$T_{31} = (s_0, -b_1 - fs - fb, fq - fns - co, rq - ci)$

$T_{32} = (s_0, -b_1 - fs - fb, rs + rb - fns - co, -ci)$

$T_{33} = (s_0, -fs, rs - co, -ci)$ $\quad$ $T_{34} = (s_0, s_1, -fns * 2, -ci)$

$T_{35} = (-b_0 - fs - fb, 0, fq - co, rq - ci)$

$T_{36} = (-b_0 - fs - fb, 0, rs + rb - co, -ci)$

$T_{37} = (-fs, 0, rs - co, -ci)$ $\quad$ $T_{38} = (s_0, 0, -fns - co, -ci)$

$T_{39} = (s_0, 0, -fns, -ci)$ $\quad$ $T_{40} = (0, s_1, -fns - co, -ci)$

$T_{41} = (0, -fs - fe, rs - co, -ci)$

$T_{42} = (0, -b_1 - fs - fb, fq - co, rq - ci)$

$T_{43} = (0, -b_1 - fs - fb, rs + rb - co, -ci)$ $\quad$ $T_{44} = (0, -fs, rs, -ci)$

$T_{45} = (0, s_1, -fns, -ci)$ $\quad$ $T_{46} = (0, 0, -co, -ci)$

$T_{47} = (0, 0, -co, -ci)$ $\quad$ $T_{48} = (0, 0, 0, -ci)$

3 Третья глава. Программная реализация моделей

3.1 Общая схема вычислений

В рассмотренных далее примерах рассматривалась вероятностная реализация алгоритмов CFR и MCCFR. При использовании метода MCCFR значения случайных событий генерируются перед началом каждой обучающей итерации. Данный подход позволяет сократить обьем памяти и ускорить вычисления в некоторых случаях[5]. При реализации примеров были выделены следующие компоненты:

— описание правил игры (зависит от настроек);
— модуль с реализацией алгоритма относительно определенных правил.

Настройки игры, например, по возможности могут включать число игроков, параметры выплат и т.п.

Правила игры включают структуру игрового дерева, механизм распределения событий и функцию выплат. Игровое дерево строится с применением узлов – обьектов с информацией о историии игры, о игроке и о возможных действиях.

Итерации алгоритма проходят рекурсивно, начиная с вершины дерева. В ходе одной итерации t + 1 происходит следующее:

— расчет стратегий q^{t+1} используя контрафактические сожаления t(1.8);
— расчет $t+1$ слагаемого контрафактических сожалений;
— обновление сумм контрафактических сожалений (1.7).

Сам расчет итераций CFR происходит на основе определенных для каждого конкретного случая правил игры. Работа алгоритма начинается с создания игрового дерева. Далее происходит расчет заданного числа итераций. После любой итерации можно получить стратегии игроков, которые представляют из себя приближенное коррелированное равновесие.

Программный код с реализацией алгоритма приведен в приложении А.

3.2 Моделирование раскрытия совместного преступления

Рассмотрим несколько частных случаев задачи разоблачения совместного преступления. В соответствии с определением 1, опишем игровые истории и информационные наборы игроков.

$$A = \{\text{Предложить взятку}, \text{Не предлагать взятку}, \text{Принять взятку},$$
$$\text{Отклонить взятку}, \text{Провести проверку}, \text{Не проводить проверку}\}$$

Также определим перечень всех досупных игровых историй, включая терминальные:

$$
\begin{aligned}
hB =& \{\text{Предложить взятку}\},\\
hBT =& \{\text{Предложить взятку}, \text{Принять взятку}\},\\
zBTC =& \{\text{Предложить взятку}, \text{Принять взятку}, \text{Провести проверку}\},\\
zBTnC =& \{\text{ Предложить взятку}, \text{Принять взятку}, \text{Не проводить проверку}\},\\
hBnT =& \{\text{Предложить взятку}, \text{Отклонить взятку}\},\\
zBnTC =& \{\text{ Предложить взятку}, \text{Отклонить взятку}, \text{Провести проверку}\},\\
zBnTnC =& \{\text{ Предложить взятку}, \text{Отклонить взятку}, \text{Не проводить проверку}\},\\
hnB =& \{\text{ Не предлагать взятку}\},\\
znBC =& \{\text{ Не предлагать взятку}, \text{Провести проверку}\},\\
znBnC =& \{\text{ Не предлагать взятку}, \text{Не проводить проверку}\},
\end{aligned}
$$

$$H = \{\emptyset, hB, hBT, zBTC, zBTnC, hBnT, zBnTC, zBNCnC, hnB, znBC, znBnC\},$$

$$Z = \{zBTC, zBTnC, zBnTC, zBnTnC, znBC, znBnC\}. \tag{3.1}$$

Будем рассматривать множество N, состоящее из трех игроков: клиент, чиновник и инспектор

$$N = \{0, 1, 2\}$$

Далее, определим информационные состояния игроков. В данном случае информационные наборы каждого игрока состоят из одного информационного состояния

$$
\begin{aligned}
\mathcal{I}_0 =& \{\{\emptyset\}\}\\
\mathcal{I}_1 =& \{\{hB\}\}\\
\mathcal{I}_2 =& \{\{hBT, hBnT, hnB\}\}
\end{aligned}
$$

Данное определение информационных состояний позволяет сопоставить игроков и различные игровые истории и таким образом определить функцию P.

$$P(h0) = 0$$

$$P(hB) = 1$$

$$P(hBT) = P(hBnT) = P(hnB) = 2$$

Наконец, определим следующие терминальные выплаты для игроков из N на множестве Z. В таблице 3.1 указано значение функции $u_i(z)$, для игрока $i \in N$ и терминальной истории $z \in Z$.

u	$zBTC$	$zBTnC$	$zBnTC$	$zBnTnC$	$znBC$	$znBnC$
0	$v - b - p_L - p_H$	$v - b$	$-p_L$	0	0	0
1	$b - q$	b	r	r	0	0
2	$x + \Delta x$	x	$y + \Delta y$	y	z	$z + \Delta z$

Таблица 3.1 — Значения функции выплат u

Проведем расчеты для некоторых значений параметров. Рассмотрим набор параметров из таблицы 3.2 и проведем расчет равновесия для данного случая

v	b	p_L	p_H	q	r	Δx	x	Δy	y	Δz	z
6	4	3	3	5	1	6	-3	4	-2	2	-1

Таблица 3.2 — Значения параметров для первого примера

Учитывая параметры из таблицы 3.2 построим игровое дерево со случайным профилем стратегий. Схематичное изображение игрового дерева показано на рисунке 3.1.

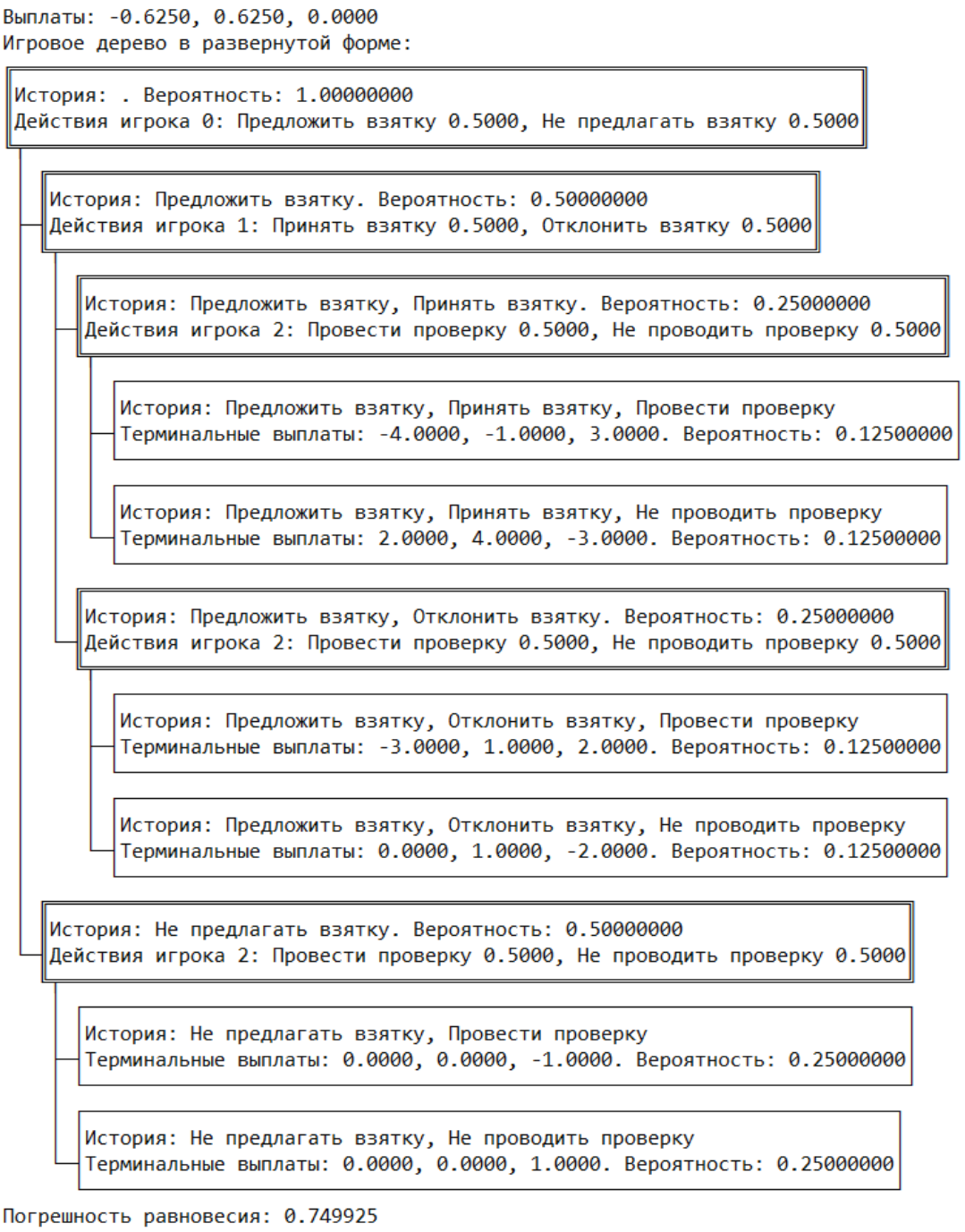

Рисунок 3.1 — Дерево игры с первой группой параметров. Случайные стратегии

На рисунке 3.1 прямоугольными элементами отмечены все игровые истории, начиная с начала игры. Ребра между элементами обозначают возможные переходы между игровыми историями. Каждой нетерминальной игровой истории соответствует перечень доступных действий и стратегия их выбора. Терминальные истории сопровождаются информацией о выплатах игрокам. Для каждой истории указывается вероятность ее реализации. Расчетная эксплуатируемость данного профиля стратегий составляет примерно 0.75. Для достижения этого значения инспектору достаточно проводить проверку с вероятностью 1.0, изменив тем самым свой ожидаемый

выигрыш с 0 до 0.75. Данный стратегический профиль достаточно далек от равновесия.

Попробуем улучшить стратегический профиль. Проведем $T = 10000$ обучающих итераций алгоритма на данном игровом дереве. Информация о обновленном стратегическом профиле представлена на рисунке 3.2.

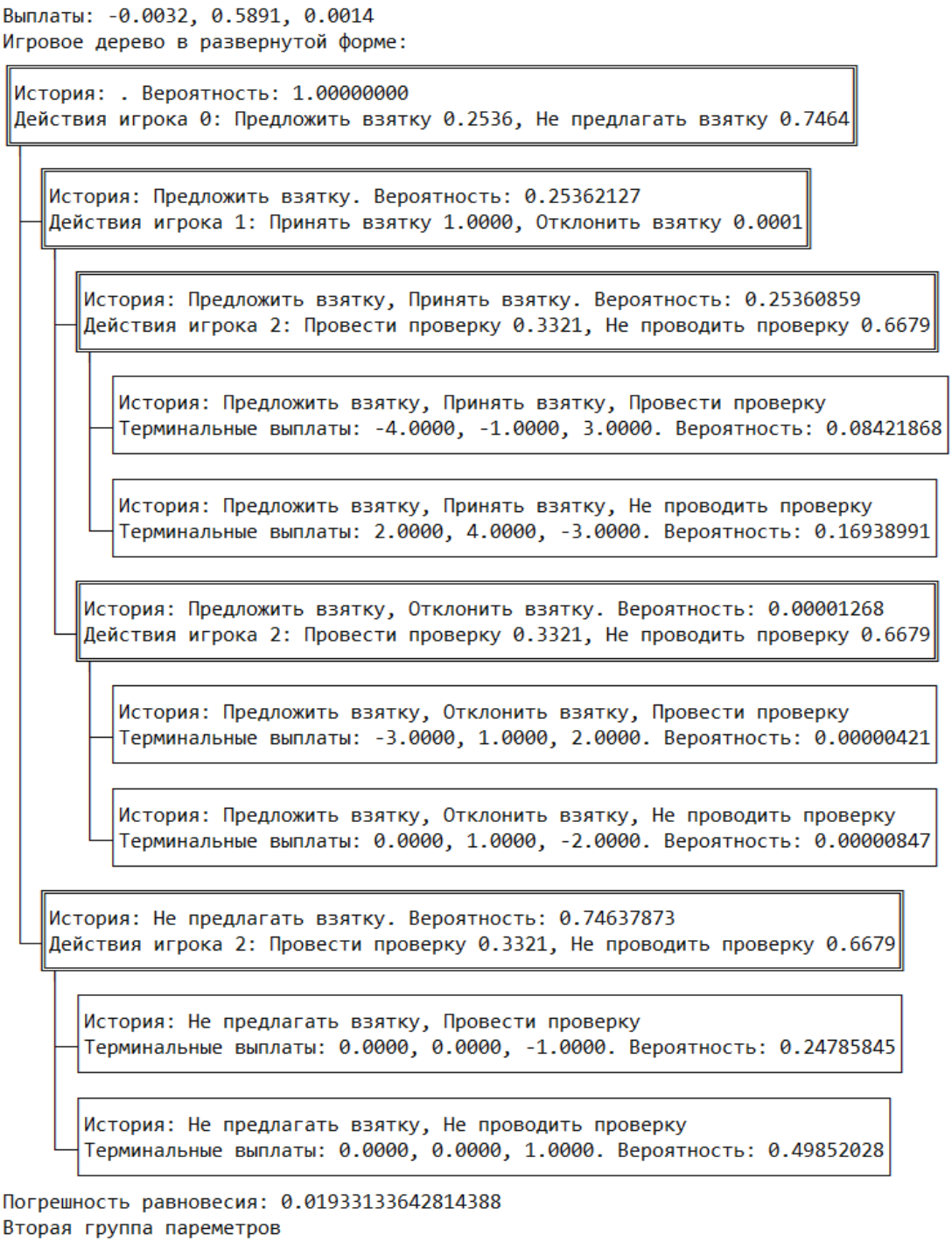

Рисунок 3.2 — Дерево игры с первой группой параметров. $T = 10000$

На рисунке 3.2 отмечены выплаты, измененные стратегии игроков и измененные вероятности достижения различных игровых историй. Как можно заметить,

погрешность равновесия снизилась до порядка 0.01, что сопоставимо с оценкой из теоремы 3.

Дальнейшее увеличение числа итераций приводит к уменьшению погрешности равновесия. График изменения расчетной погрешности равновесия для данного примера приведен на рисунке 3.3.

Рисунок 3.3 — Расчетная эксплуатируемость стратегий в зависимости от T

Рассмотрим другой набор параметров алгоритма (Таблица 3.3).

v	b	p_L	p_H	q	r	Δx	x	Δy	y	Δz	z
12	3	8	8	4	2	9	1	1	7	3	5

Таблица 3.3 — Значения параметров для второго примера

Построим игровую модель и проведем 10000 обучающих итераций. Полученный профиль стратегий представлен на рисунке 3.4.

```
Выплаты: -0.0001, 0.0001, 7.9995
Информационные состояния:

0:Выбор клиента
Действия игрока 0: Предложить взятку 0.0001, Не предлагать взятку 1.0000

1:Выбор чиновника
Действия игрока 1: Принять взятку 0.0001, Отклонить взятку 1.0000

2:Выбор инспектора
Действия игрока 2: Провести проверку 0.0002, Не проводить проверку 0.9998

Погрешность равновесия: 0.00034997666242020387
```

Рисунок 3.4 — Стратегии игроков для второго примера. $T = 10000$

В результате мы получили профиль, который состоит из чистых стратегий. Хотя данное решение и является равновесием, оно маловероятно на практике по интуитивным соображениям. Так как алгоритм предоставляет единственное решение, имеет смысл наложить дополнительные ограничения на рассматриваемую задачу. Попробуем получить дополнительную информацию о данной игре. Для этого попробуем зафиксировать стратегию одного игрока и найти равновесие для двух оставшихся. Таким образом, найдем зависимости β и γ от α, α и γ от β и зависимость α и β от γ. Графики соответствующих зависимостей представлены на рисунках .

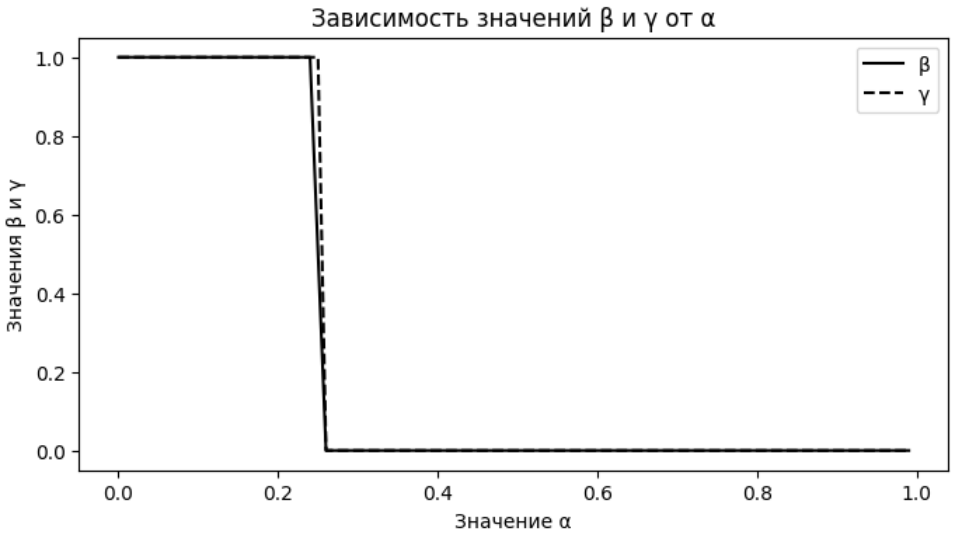

Рисунок 3.5 — График зависимости β и γ от α

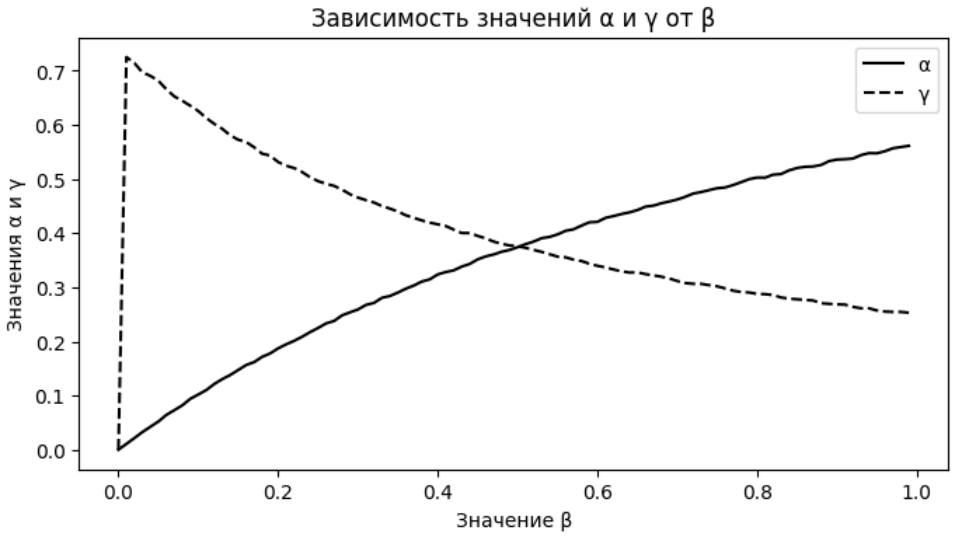

Рисунок 3.6 — График зависимости α и γ от β

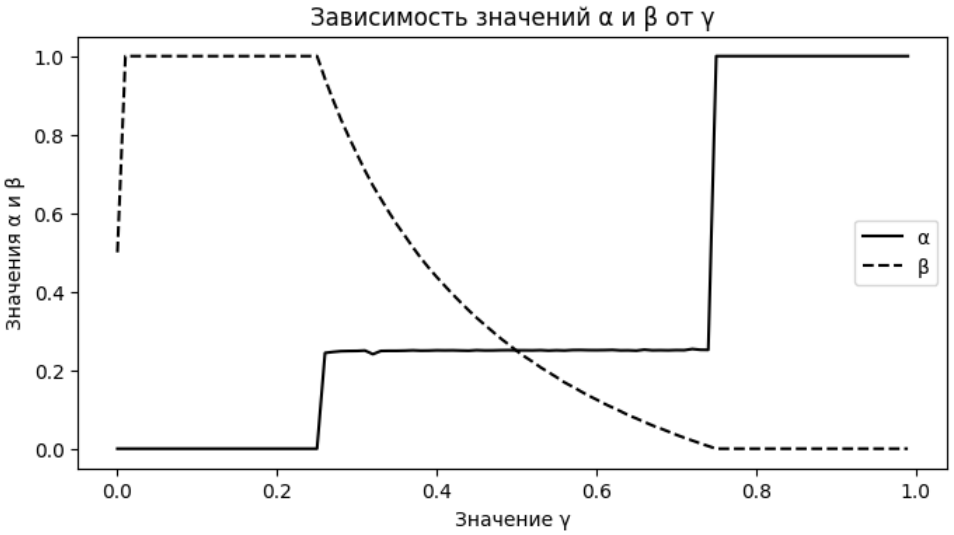

Рисунок 3.7 — График зависимости α и β от γ

В соответствии с оценкой (2.5), вероятность проверки α влияет на решение чиновника о принятии взятки. В данном случае, критическое значение равно 0.25, и на графиках 3.5 и 3.7 прослеживается изменение поведения участников при его преодолении инспектором.

Программный код с реализацией данной модели приведен в приложении Б.

3.3 Моделирование коррупции в иерархической структуре

В данной работе в качестве второго объекта исследования была выбрана модель коррупции в иерархической структуре. Рассмотрим частный случай параметров иерархии, которые описаны в (2.9). Как и в предыдущем примере, опишем игровые истории и информационные наборы игроков.

$$A = \{\text{НеСовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{ПредлВз}, \text{ВыплШт},$$
$$\text{РазРук}, \text{ПринВз}, \text{ОтклВз}, \text{ПровЧин}, \text{НеПровЧин}\}$$

Стоит отметить, что приведенный набор действий не содержит отдельные действия для проверки первого и второго сотрудника. Вместо этого будет использоваться решение о проверке каждого или переход к следующему. Далее, определим перечень всех досупных игровых историй без терминальных:

$$h_1 = \{\text{НеПровЧин}\}, \quad h_2 = \{\text{НеПровЧин}, \text{СовКр}\}, \quad h_3 = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}\},$$

$$h_4 = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}, \text{ПровСот}\},$$

$$h_5 = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}, \text{ПровСот}, \text{ПредлВз}\},$$

$$h_6 = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}\},$$

$$h_7 = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}\},$$

$$h_8 = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{РазРук}\},$$

$$h_9 = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{РазРук}, \text{ПредлВз}\},$$

$$h_{10} = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{ПредлВз}\},$$

$$h_{11} = \{\text{НеПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{НеПровСот}\},$$

$$h_{12} = \{\text{НеПровЧин}, \text{СовКр}, \text{НеСовКр}\}, \quad h_{13} = \{\text{НеПровЧин}, \text{СовКр}, \text{НеСовКр}, \text{ПровСот}\}$$

$$h_{14} = \{\text{НеПровЧин}, \text{СовКр}, \text{НеСовКр}, \text{ПровСот}, \text{ПредлВз}\},$$

$$h_{15} = \{\text{НеПровЧин}, \text{СовКр}, \text{НеСовКр}, \text{НеПровСот}\},$$

$$h_{16} = \{\text{НеПровЧин}, \text{НеСовКр}\}, \quad h_{17} = \{\text{НеПровЧин}, \text{НеСовКр}, \text{СовКр}\},$$

$$h_{18} = \{\text{НеПровЧин}, \text{НеСовКр}, \text{СовКр}, \text{НеПровСот}\},$$

$$h_{19} = \{\text{НеПровЧин}, \text{НеСовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}\},$$

$$h_{20} = \{\text{НеПровЧин}, \text{НеСовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{ПредлВз}\},$$

$$h_{21} = \{\text{НеПровЧин}, \text{НеСовКр}, \text{НеСовКр}\},$$

$$h_{22} = \{\text{НеПровЧин}, \text{НеСовКр}, \text{НеСовКр}, \text{НеПровСот}\},$$

$$h_{23} = \{\text{ПровЧин}\}, \quad h_{24} = \{\text{ПровЧин}, \text{СовКр}\}, \quad h_{25} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}\},$$

$$h_{26} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}, \text{ПровСот}\},$$

$$h_{27} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}, \text{ПровСот}, \text{ПредлВз}\},$$

$$h_{28} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}\},$$

$$h_{29} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}\},$$

$$h_{30} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{РазРук}\},$$

$$h_{31} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{РазРук}, \text{ПредлВз}\},$$

$$h_{32} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{ПредлВз}\},$$

$$h_{33} = \{\text{ПровЧин}, \text{СовКр}, \text{СовКр}, \text{НеПровСот}, \text{НеПровСот}\},$$

$$h_{34} = \{\text{ПровЧин}, \text{СовКр}, \text{НеСовКр}\}, \quad h_{35} = \{\text{ПровЧин}, \text{СовКр}, \text{НеСовКр}, \text{ПровСот}\}$$

$$h_{36} = \{\text{ПровЧин}, \text{СовКр}, \text{НеСовКр}, \text{ПровСот}, \text{ПредлВз}\},$$

$$h_{37} = \{\text{ПровЧин}, \text{СовКр}, \text{НеСовКр}, \text{НеПровСот}\},$$

$$h_{38} = \{\text{ПровЧин}, \text{НеСовКр}\}, \quad h_{39} = \{\text{ПровЧин}, \text{НеСовКр}, \text{СовКр}\},$$

$$h_{40} = \{\text{ПровЧин}, \text{НеСовКр}, \text{СовКр}, \text{НеПровСот}\},$$

$$h_{41} = \{\text{ПровЧин}, \text{НеСовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}\},$$

$$h_{42} = \{\text{ПровЧин}, \text{НеСовКр}, \text{СовКр}, \text{НеПровСот}, \text{ПровСот}, \text{ПредлВз}\},$$

$$h_{43} = \{\text{ПровЧин}, \text{НеСовКр}, \text{НеСовКр}\},$$

$$h_{44} = \{\text{ПровЧин}, \text{НеСовКр}, \text{НеСовКр}, \text{НеПровСот}\},$$

$$H \setminus Z = \{\emptyset, h_1, \ldots, h_{44}\}. \tag{3.2}$$

Терминальные истории подобного примера приведены в главе 2. Будем рассматривать множество N, состоящее из четырех игроков: сотрудник C_0, сотрудник C_1, чиновник O и инспектор I

$$N = \{0, 1, 2, 3\}$$

Далее, определим информационные состояния игроков. В данном случае информационные наборы некотрых игроков состоят из нескольких информационных состояний

$$I_1 = \{\emptyset\},\ I_2 = \{h_1,\ h_{23}\},\ I_3 = \{h_2,\ h_{16},\ h_{h24},\ h_{38}\},$$

$$I_4 = \{h_3,\ h_{12},\ h_{17},\ h_{21},\ h_{25},\ h_{34},\ h_{39},\ h_{43}\},$$

$$I_5 = \{h_6,\ h_{15},\ h_{18},\ h_{22},\ h_{28},\ h_{37},\ h_{40},\ h_{44}\},$$

$$I_6 = \{h_4,\ h_{13},\ h_{26},\ h_{35}\},\ I_7 = \{h_{h8},\ h_{30}\},$$

$$I_8 = \{h_9,\ h_{31}\},\ I_9 = \{h_5,\ h_{14},\ h_{27},\ h_{36}\},$$

$$I_{10} = \{h_7,\ h_{h_19},\ h_{29},\ h_{41}\},\ I_{11} = \{h_{10},\ h_{20},\ h_{32},\ h_{42}\},$$

$$\mathcal{I}_0 = \{I_2,\ I_6,\ I_7\},$$

$$\mathcal{I}_1 = \{I_3,\ I_{10}\},$$

$$\mathcal{I}_2 = \{I_4,\ I_5,\ I_8,\ I_9,\ I_{11}\},$$

$$\mathcal{I}_3 = \{I_1\}.$$

Данное определение информационных состояний позволяет сопоставить игроков и различные игровые истории.

Функцию $u_i(z)$, для игрока $i \in N$ и терминальной истории $z \in Z$, определим в соответствии с формой игры на рисунках 2.2 и 2.3.

Проведем расчеты для некоторых значений параметров. Рассмотрим набор параметров из таблицы 3.4 и проведем расчет равновесия для данного случая

s_0	b_0	s_1	b_1	co	ci	fs	fb	fq	fns	fe	fbc	rs	rb	rq
80	24	40	12	2	2	110	15	4	2	1	10	4	1	10

Таблица 3.4 — Значения параметров для примера

Учитывая параметры из таблицы 3.4 построим игровое дерево со случайным профилем стратегий. Схематичное изображение игрового дерева показано на рисунке 3.1.

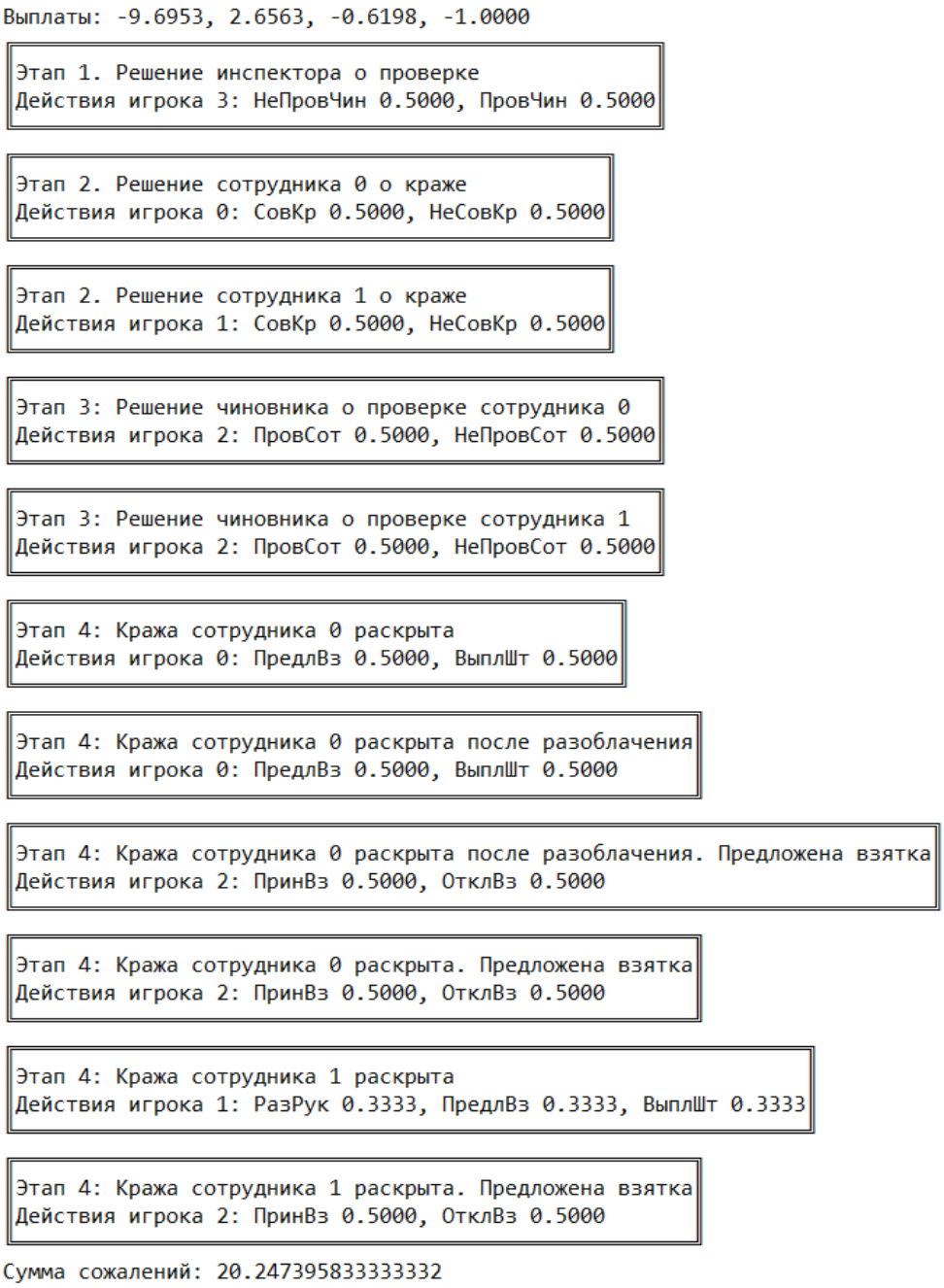

Рисунок 3.8 — Дерево игры с первой группой параметров. Случайные стратегии

На рисунке 3.8 прямоугольными элементами отмечены все информационные состояния игроков. Каждому информационному состоянию соответствует перечень доступных действий и стратегия их выбора. Расчетная расчетная сумма сожалений всех игроков превосходит 20 и данный стратегический профиль достаточно далек от равновесия и можно провести минимизацию сожалений.

Попробуем улучшить стратегический профиль. Проведем $T = 10000$ обучающих итераций алгоритма на данном игровом дереве. Информация о обновленном стратегическом профиле представлена на рисунке 3.9.

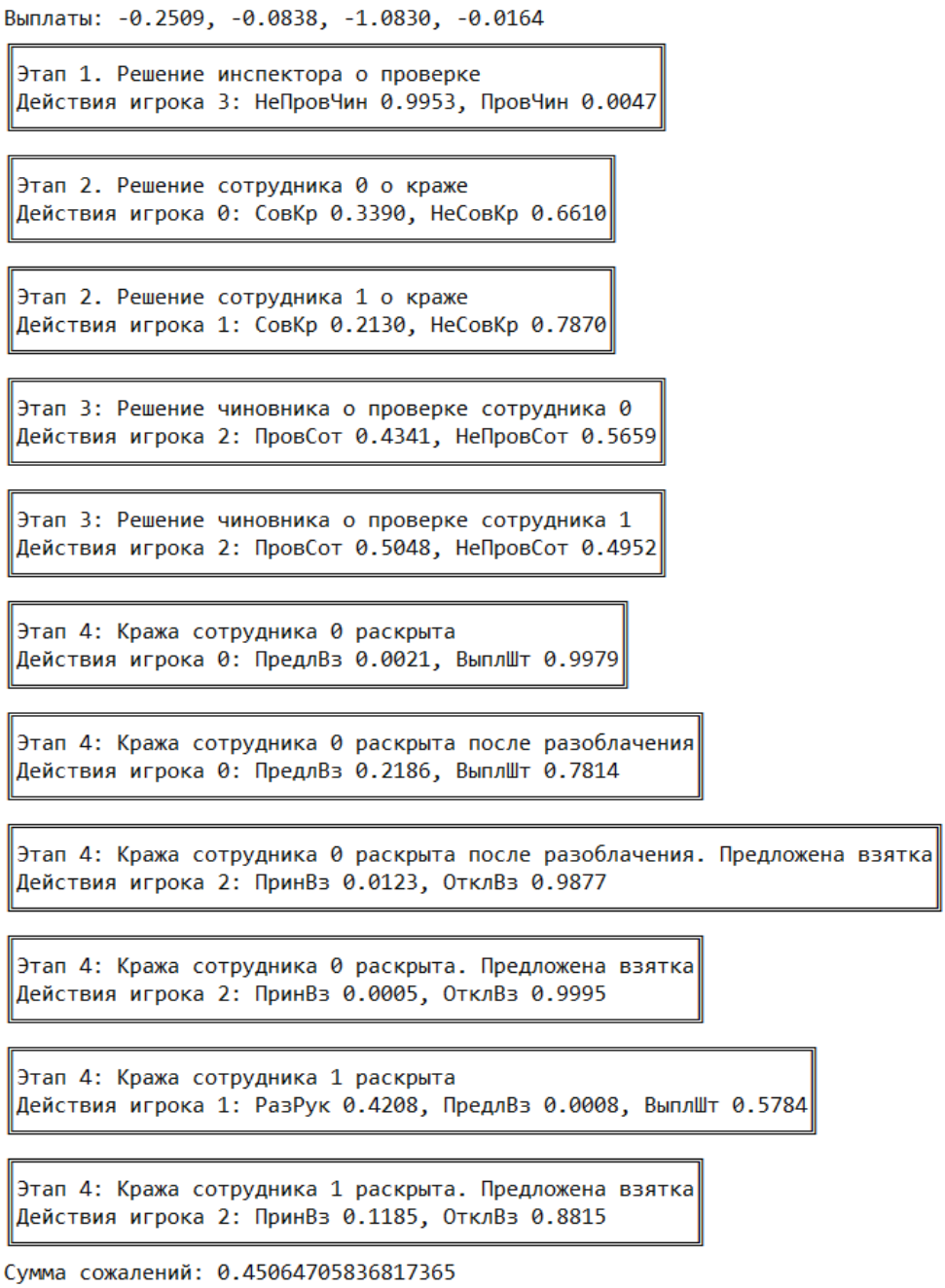

Рисунок 3.9 — Дерево игры с первой группой параметров. $T = 10000$

На рисунке 3.9 отмечены измененные стратегии игроков. Как можно заметить, сумма сожалений снизилась до 0.45. Дальнейшее увеличение числа итераций приводит к уменьшению суммы сожалений. График изменения чуммы сожалений для данного примера приведен на рисунке 3.10.

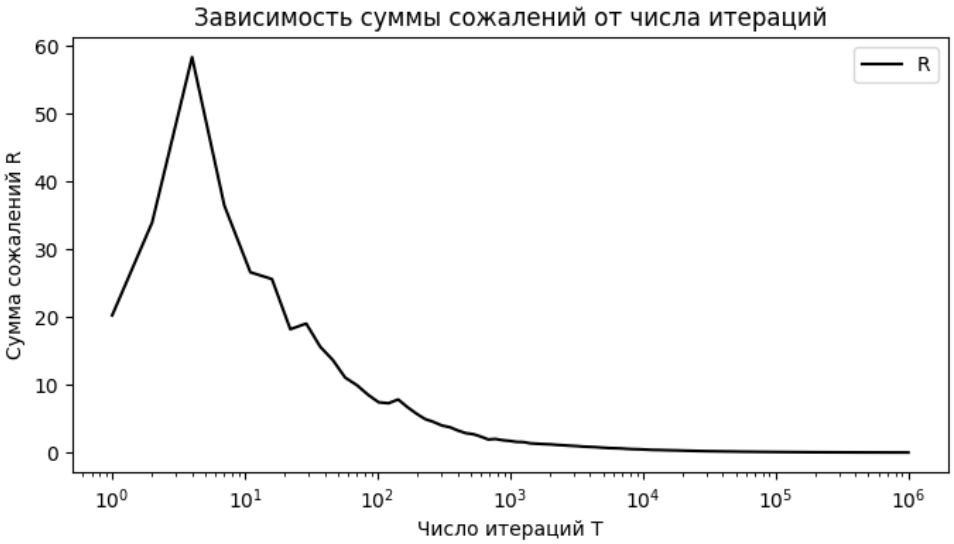

Рисунок 3.10 — Расчетная сумма сожалений в зависимости от T

Программный код с реализацией данной модели приведен в приложении В.

Заключение

В данной работе была рассмотрена реализация минимизации сожалений для некоторых задач моделирования коррупции. Отработанные в ходе практической реализации модели показывают, что подобные алгоритмы решения могут генерировать устойчивые стратегические профили, которые связаны как с теоритической оценкой, так и с достаточно оптимальным поведением участников. Используемый алгоритм позволяет строить более обобщенные абстракции и проводить анализ используя естественные предположения о информированности игроков относительно всех игровых историй. Основным приемуществом данного подхода являются развитые методики построения решения, которые позволяют в будующем масштабировать результаты для более сложных игровых абстракций. Так, можно рассматривать произвольную организационную структуру и строить расчеты непосредственно исходя из практических потребностей организации механизмов проверок.

К приемуществам полученной иерархической модели можно отнести масштабируемость с точки зрения проверяемых лиц и свободный выбор альтернатив игроками. Отсутствие случайных событий в базовой постановке позволяет естественным образом отразить целесообразность тех или иных действий.

К недостаткам можно отнести фиксированный размер штрафов и прочих параметров. В качестве альтернативы можно предложить функциональную зависимость от выплат другим участникам. Также, к возможным доработкам можно отнести и структуру проверяющего органа, который может включать более чем одного игрока, вследствии ограниченных возможностей одного проверяющего агента.

В целом, полученные алгоритмы и методики могут быть использованы как для изучения старых, так и для построения новых моделей и абстракций.

Список использованных источников

1. *Spengler, D.* Detection and Deterrence in the Economics of Corruption: a Game Theoretic Analysis and some Experimental Evidence. / D. Spengler. — University of York, 2014.

2. *Kumacheva, S. Sh.* The Strategy of Tax Control in Conditions of Possible Mistakes and Corruption of Inspectors. Contributions to Game The ory and Management (Petrosyan, L. A., Zenkevich, N. A. eds), Vol. 6 / S. Sh. Kumacheva. — St. Petersburg University, 2013, pp. 264–273.

3. *Orlov I. M., Kumacheva S. Sh.* Hierarchical Model of Corruption: Game theoretic Approach. Mathematical Control Theory and Its Applications Proceedings / Kumacheva S. Sh. Orlov I. M. — CSRI ELEKTROPRIBOR, St. Petersburg, 2020, pp. 269–271.

4. *Martin Zinkevich Michael Johanson, Michael Bowling Carmelo Piccione.* Regret minimization in games with incomplete information. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, Advances in Neural Information Processing Systems 20 / Michael Bowling Carmelo Piccione Martin Zinkevich, Michael Johanson. — MIT Press, Cambridge, 2008, pages 1729–1736.

5. *Marc Lanctot Kevin Waugh, Martin Zinkevich Michael Bowling.* Monte carlo sampling for regret minimization in extensive games. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, Advances in Neural Information Processing Systems 22 / Martin Zinkevich Michael Bowling Marc Lanctot, Kevin Waugh. — MIT Press, Cambridge, 2009, pages 1078–1086.

6. *Hart, Sergiu.* A simple adaptive procedure leading to correlated equilibrium / Sergiu Hart, Andreu Mas-Colell. — Econometrica, 68(5), September 2000, pages 1127–1150.

7. *Gibson, R.* Regret minimization in games and the development of champion multiplayer computer poker-playing agents, Ph.D. thesis / R. Gibson. — University of Alberta, 2014.

8. *Brown, Noam.* Supplementary Materials for Superhuman AI for multiplayer poker / Noam Brown, Tuomas Sandholm. — Science First Release DOI: 10.1126/science.aay2400, 11 July 2019.

Приложение А Алгоритм решения

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.IO;
namespace CorruptionModel{
    //узел игрового дерева
    public class GameTreeNode : IComparable<GameTreeNode>{
        public string Name { get; set; }//имя
        public NodeType NodeType { get; set; }//тип
        public List<GameTreeNode> Childs { get; set; }//дочерние узлы
        //возможные дочерние узлы
        public List<GameTreeNode> PossibleChilds { get; set; }
        public List<string> Actions { get; set; }//действия
        public double[] TStrategy { get; set; }//стратегии
        public double[] RegretSum { get; set; }//сожаления
        public double[] tRegretSum { get; set; }
        public double[] StrategySum { get; set; }
        public double[,] Util { get; set; }//знычение
        //контрафактические значения
        public double[] CfValues { get; set; }
        public double[] P { get; set; }
        public int Player { get; set; }
        public List<string> History { get; set; }
        public int NumOfPlayers { get; set; }
        public GameTreeNode(int numOfPlayers, NodeType nodeType, int
            player, IEnumerable<string> actions){
            NumOfPlayers = numOfPlayers;
            NodeType = nodeType;
            Player = player;
            if (nodeType != NodeType.TerminalNode)
                Actions = new List<string>(actions);
            InitValues();
        }
        public void InitValues(){
            CfValues = new double[NumOfPlayers];
            P = new double[NumOfPlayers];
            History = new List<string>();
            if (NodeType == NodeType.TerminalNode) return;
            TStrategy = new double[Actions.Count];
```

```
40              StrategySum = new double[Actions.Count];
41              RegretSum = new double[Actions.Count];
42              tRegretSum = new double[Actions.Count];
43              Util = new double[Actions.Count, NumOfPlayers];
44              Childs = new List<GameTreeNode>();
45          }
46          //нормализация стратегии
47          private void NormalizePositiveSum(double[] source, double[]
                dest){
48              double normSum = 0;
49              for (int a = 0; a < source.Length; a++)
50              {
51                  if (source[a] > 0)
52                  {
53                      dest[a] = source[a];
54                      normSum += source[a];
55                  }
56                  else
57                      dest[a] = 0;
58              }
59              for (int a = 0; a < source.Length; a++)
60              {
61                  if (normSum > 0)
62                      dest[a] /= normSum;
63                  else
64                      dest[a] = 1.0 / Actions.Count;
65              }
66          }
67          public double[] GetAvgStrategy(){
68              if (NodeType == NodeType.ChanceNode)
69                  return TStrategy;
70              double[] avgStrategy = new double[Actions.Count];
71              NormalizePositiveSum(StrategySum, avgStrategy);
72              return avgStrategy;
73          }
74          public void CalcTStrategy(double weight){
75              NormalizePositiveSum(tRegretSum, TStrategy);
76              for (int a = 0; a < Actions.Count; a++)
77                  StrategySum[a] += TStrategy[a] * weight;
78          }
79          public int CompareTo(GameTreeNode other){
```

```
80              return Name.CompareTo(other.Name);
81          }
82      }
83      public interface IGameRooles{
84          int NumOfPlayers { get; }
85          GameTreeNode GenerateRoot();
86          void UpdateChilds(GameTreeNode node);
87          void SetTerminalEquity(GameTreeNode terminalNode);
88      }
89      public enum NodeType{
90          PlayNode,//игровой узел
91          ChanceNode,//случайный выбор
92          TerminalNode//терминальый узел
93      }
94      public static class TSRandom{
95          [ThreadStatic] public static Random rand = new Random();
96      }
97      //выбор действия
98      public static class Strategy{
99          public static int GetRandomChoice(double[] strategy){
100             double ch = TSRandom.rand.NextDouble();
101             int a = 0;
102             while (a < strategy.Length - 1 && strategy[a] <= ch) a++;
103             return a;
104         }
105     }
106     //реализация алгоритма
107     public class Cfr{
108         double[] resultUtil;//выплаты
109         public int iterSum { get; private set; }
110         public GameTreeNode root;
111         public int NumOfPlayers;
112         public bool External;//внешние выплаты
113         public IGameRooles gameRooles;//правила игры
114         public Cfr(IGameRooles gameRooles, bool external = false){
115             NumOfPlayers = gameRooles.NumOfPlayers;
116             this.gameRooles = gameRooles;
117             External = external;
118         }
119         public void Init(){
120             if (root == null)
```

```
121                 root = gameRooles.GenerateRoot();
122             for (int player = 0; player < NumOfPlayers; player++)
123                 root.P[player] = 1.0;
124             resultUtil = new double[NumOfPlayers];
125             iterSum = 0;
126         }
127         int player;
128         public void Iterate(int iterations){
129             int count = iterations;
130             while (count-- > 0){
131                 root.History.Clear();
132                 SaveTRegretsRec(root);
133                 player = count % NumOfPlayers;
134                 RegRec(root);
135                 for (int player = 0; player < NumOfPlayers; player++)
136                 {
137                     resultUtil[player] += root.CfValues[player];
138                 }
139             }
140             iterSum += iterations;
141         }
142         public double[] ReturnUtil(){
143             double[] resUtil = resultUtil.Clone() as double[];
144 
145             for (int player = 0; player < NumOfPlayers; player++){
146                 resUtil[player] /= iterSum;
147             }
148 
149             return resUtil;
150         }
151         //подсчет сожалений
152         public void RegRec(GameTreeNode node){
153             if (node.NodeType == NodeType.TerminalNode){
154                 gameRooles.SetTerminalEquity(node);
155                 return;
156             }
157             if (node.NodeType != NodeType.ChanceNode){
158                 node.CalcTStrategy(node.P[node.Player]);
159             }
160             gameRooles.UpdateChilds(node);
161 
```

```
162             if (!External || node.Player == player)
163                 CalcUtil(node);
164             else
165                 CalcUtilExternal(node);
166         }
167         //подсчет сожалений
168         public void CalcUtil(GameTreeNode node){
169             for (int player = 0; player < NumOfPlayers; player++)
170                 node.CfValues[player] = 0;
171             for (int a = 0; a < node.Childs.Count; a++){
172                 GameTreeNode child = node.Childs[a];
173                 child.History.Clear();
174                 child.History.AddRange(node.History);
175                 child.History.Add(node.Actions[a]);
176                 for (int player = 0; player < NumOfPlayers; player++)
                    {
177                     if (player != node.Player)
178                         child.P[player] = node.P[player];
179                     else
180                         child.P[player] = node.TStrategy[a] * node.P[
                            player];
181                 }
182                 RegRec(child);
183                 for (int player = 0; player < NumOfPlayers; player++)
                    {
184                     node.Util[a, player] = child.CfValues[player];
185                     node.CfValues[player] += node.TStrategy[a] * node
                        .Util[a, player];
186                 }
187             }
188             //обновление сожалений
189             if (node.NodeType != NodeType.ChanceNode){
190                 for (int a = 0; a < node.Childs.Count; a++){
191                     double regret = node.Util[a, node.Player] - node.
                        CfValues[node.Player];
192                     if (!External)
193                         node.RegretSum[a] += GetPiMinusI(node.P, node
                            .Player) * regret;
194                     else
195                         node.RegretSum[a] += regret;
196                 }
```

```
197             }
198         }
199         //внешняя выборка
200         public void CalcUtilExternal(GameTreeNode node){
201             int a = Strategy.GetRandomChoice(node.TStrategy);
202             GameTreeNode child = node.Childs[a];
203             child.History.Clear();
204             child.History.AddRange(node.History);
205             child.History.Add(node.Actions[a]);
206             for (int player = 0; player < NumOfPlayers; player++)
207                 child.P[player] = node.P[player];
208             RegRec(child);
209             for (int player = 0; player < NumOfPlayers; player++)
210                 node.CfValues[player] = child.CfValues[player];
211         }
212         public double GetPiMinusI(double[] p, int i){
213             double ans = 1.0;
214             for (int player = 0; player < NumOfPlayers; player++)
215                 if (player != i)
216                     ans *= p[player];
217             return ans;
218         }
219         //сохранение сожалений
220         public void SaveTRegretsRec(GameTreeNode node){
221             if (node.NodeType == NodeType.TerminalNode)
222                 return;
223             for (int a = 0; a < node.Childs.Count; a++)
224                 node.tRegretSum[a] = node.RegretSum[a];
225             foreach (var c in node.PossibleChilds)
226                 SaveTRegretsRec(c);
227         }
228     }
229     public class StringTree{
230         public string[] strs;
231         public int type = 0;
232         public StringTree[] childs;
233     }
234     public static class Tools{
235         private static string FillStr(string str, int len){
236             while (str.Length < len) str += " ";
237             return str;
```

```
238         }
239         //отрисовка игрового дерева
240         public static void printTree(TextWriter tw, StringTree node,
                string indent = ""){
241             string leftIndent = indent != "" ? (indent.Substring(0,
                    indent.Length - 3) + " | ") : "";
242             string midleIndent = indent != "" ? indent.Substring(0,
                    indent.Length - 3) + (indent.EndsWith(" | ") ? " +—" :
                     " +—") : "";
243             int maxLen = node.strs.Select(x => x.Length).Max();
244             char line = node.type == 0 ? '—' : '=';
245             tw.WriteLine("{0}+{1}+", leftIndent, new String(line,
                    maxLen));
246             for (int i = 0; i < node.strs.Length; i++){
247                 if (i == node.strs.Length / 2 && indent != ""){
248                     tw.WriteLine("{0}+{1}|", midleIndent, FillStr(
                            node.strs[i], maxLen));
249                     leftIndent = indent;
250                 }
251                 else
252                     tw.WriteLine("{0}|{1}|", leftIndent, FillStr(node
                            .strs[i], maxLen));
253             }
254             tw.WriteLine("{0}+{1}{2}+", indent, node.childs.Length >
                    0 ? "+" : line.ToString(), new String(line, maxLen -
                    1));
255             if (node.childs != null)
256                 for (int a = 0; a < node.childs.Length; a++){
257                     var child = node.childs[a];
258                     if (a != node.childs.Length - 1)
259                         printTree(tw, child, indent + " | ");
260                     else
261                         printTree(tw, child, indent + "   ");
262                 }
263         }
264         //генерация представления дерева
265         public static StringTree getTree(GameTreeNode node,
                IGameRooles gameRooles, double p = 1.0, double minP = 0,
                int depth = 100){
266             if (node.NodeType != NodeType.TerminalNode)
267                 gameRooles.UpdateChilds(node);
```

```
268                 else
269                     gameRooles.SetTerminalEquity(node);
270                 StringTree ans = new StringTree();
271                 var strs = new List<string>();
272                 if (node.NodeType == NodeType.PlayNode){
273                     ans.type = 1;
274                     strs.Add(node.Name);
275                     strs.Add(string.Format("История: {0}. Вероятность:
                        {1:0.00000000}", HistoryToStr(node.History), p));
276                     strs.Add(string.Format("Действия игрока {0}: {1}",
                        node.Player, StrategyToStr(node.GetAvgStrategy(),
                        node.Actions)));
277                 }
278                 else if (node.NodeType == NodeType.TerminalNode){
279                     ans.type = 0;
280                     strs.Add(node.Name);
281                     strs.Add(string.Format("История: {0}", HistoryToStr(
                        node.History)));
282                     strs.Add(string.Format("Терминальные выплаты: {0}.
                        Вероятность: {1:0.00000000}", StrategyToStr(node.
                        CfValues), p));
283                 }
284                 else{
285                     ans.type = 1;
286                     strs.Add(node.Name);
287                     strs.Add(string.Format("История: {0}. Вероятность:
                        {1:0.00000000}", HistoryToStr(node.History), p));
288                     strs.Add(string.Format("Случайное распределение: {0}"
                        , StrategyToStr(node.GetAvgStrategy(), node.
                        Actions)));
289                 }
290                 ans.strs = strs.ToArray();
291                 var childs = new List<StringTree>();
292                 if (node.PossibleChilds != null){
293                     var avgStrategy = node.GetAvgStrategy();
294                     for (int a = 0; a < node.Actions.Count; a++){
295                         var child = node.Childs[a];
296                         child.History = new List<string>(node.History);
297                         child.History.Add(node.Actions[a]);
298                         if (p * avgStrategy[a] >= minP && depth > 0)
```

```
299                         childs.Add(getTree(child, gameRooles, p *
                                avgStrategy[a], minP, depth - 1));
300                 }
301             }
302             ans.childs = childs.ToArray();
303             return ans;
304         }
305         //генерация информационных состояний
306         public static StringTree[] getTree2(GameTreeNode node, int
                depth = 0){
307             return GetInfosets(node).Select(tnode =>{
308                 StringTree ans1 = new StringTree();
309                 ans1.type = 1;
310                 var strs = new List<string>();
311                 if (node.NodeType == NodeType.PlayNode){
312                     strs.Add(tnode.Name);
313                     strs.Add(string.Format("Действия игрока {0}: {1}"
                            , tnode.Player, StrategyToStr(tnode.
                            GetAvgStrategy(), tnode.Actions)));
314                 }
315                 else{
316                     strs.Add(tnode.Name);
317                     strs.Add(string.Format("Случайное распределение:
                            {0}", StrategyToStr(tnode.GetAvgStrategy(),
                            tnode.Actions)));
318                 }
319                 ans1.childs = new StringTree[0];
320                 ans1.strs = strs.ToArray();
321                 return ans1;
322             }).ToArray();
323         }
324         public static string HistoryToStr(IEnumerable<string> history
                ){
325             return string.Join(", ", history);
326         }
327         public static string StrategyToStr(double[] strategy,
                IEnumerable<string> actions = null){
328             StringBuilder sb = new StringBuilder();
329             for (int a = 0; a < strategy.Length; a++){
330                 if (actions != null)
```

```
331                     sb.Append(String.Format("{0} {1:0.0000}, ",
                            actions.ToArray()[a], strategy[a]));
332                 else
333                     sb.Append(String.Format("{0:0.0000}, ", strategy[
                            a]));
334             }
335             if (sb.Length > 1)
336                 sb.Remove(sb.Length - 2, 2);
337             return sb.ToString();
338         }
339         public static ICollection<GameTreeNode> GetInfosets(
                GameTreeNode node){
340             var ans = new SortedSet<GameTreeNode>();
341             var q = new Queue<GameTreeNode>();
342             q.Enqueue(node);
343             while (q.Count > 0){
344                 var tnode = q.Dequeue();
345                 if (!ans.Contains(tnode)){
346                     ans.Add(tnode);
347                     if (tnode.PossibleChilds != null)
348                         for (int a = 0; a < tnode.PossibleChilds.
                                Count; a++)
349                             if (tnode.PossibleChilds[a].NodeType !=
                                    NodeType.TerminalNode)
350                                 q.Enqueue(tnode.PossibleChilds[a]);
351                 }
352             }
353             return ans;
354         }
355         //вычисление суммы сожалений
356         public static double GetImmRegSum(Cfr cfr){
357             var infosets = GetInfosets(cfr.root);
358             return infosets.Select(x => x.RegretSum.Select(r => Math.
                    Max(0, r)).Max()).Sum() / cfr.iterSum;
359         }
360     }
361 }
```

Приложение Б Реализация правил игры для модели разоблачения совместного преступления

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.IO;
namespace CorruptionModel{
    //набор всех возможных действий
    public static class Actions1{
        public static string bribery = "Предложить взятку";
        public static string notBriding = "Не предлагать взятку";
        public static string reciprocating = "Принять взятку";
        public static string notReciprocating = "Отклонить взятку";
        public static string inspect = "Провести проверку";
        public static string notToInspect = "Не проводить проверку";
    }
    public class Settings1{
        public double v, b, pl, ph, q, r, delx, x, dely, y, delz, z;
    }
    //правила игры для иерархической модели
    public class GameRooles1 : IGameRooles{
        Settings1 s;
        public GameRooles1(Settings1 settings){
            this.s = settings;
        }
        public int NumOfPlayers { get; } = 3;
        //функция для создания игрового дерева
        public GameTreeNode GenerateRoot(){
            GameTreeNode terminal = new GameTreeNode(NumOfPlayers,
                NodeType.TerminalNode, -1, null);
            terminal.Name = "Терминальный узел";
            GameTreeNode inspectorCh = new GameTreeNode(NumOfPlayers,
                NodeType.PlayNode, 2,
                new[] { Actions1.inspect, Actions1.notToInspect }
            );
            inspectorCh.PossibleChilds = new List<GameTreeNode>(new[]
                { terminal });
            inspectorCh.Name = "2:Выбор инспектора";
```

```
35              GameTreeNode officialCh = new GameTreeNode(NumOfPlayers,
                    NodeType.PlayNode, 1,
36                  new[] { Actions1.reciprocating, Actions1.
                        notReciprocating }
37              );
38              officialCh.PossibleChilds = new List<GameTreeNode>(new[]
                    { inspectorCh });
39              officialCh.Name = "1:Выбор чиновника";
40              GameTreeNode clientCh = new GameTreeNode(NumOfPlayers,
                    NodeType.PlayNode, 0,
41                  new[] { Actions1.bribery, Actions1.notBriding }
42              );
43              clientCh.PossibleChilds = new List<GameTreeNode>(new[] {
                    officialCh, inspectorCh });
44              clientCh.Name = "0:Выбор клиента";
45              return clientCh;
46          }
47          //метод для обновления истори
48          public void UpdateChilds(GameTreeNode node){
49              node.Childs.Clear();
50              if (node.History.Count == 0){
51                  node.Childs.Add(node.PossibleChilds[0]);
52                  node.Childs.Add(node.PossibleChilds[1]);
53              }
54              else{
55                  node.Childs.Add(node.PossibleChilds[0]);
56                  if (node.Actions.Count > 1)
57                      node.Childs.Add(node.PossibleChilds[0]);
58              }
59          }
60          private bool HistoryEquals(List<string> a, string[] b){
61              if (a.Count != b.Length)
62                  return false;
63              for (int i = 0; i < a.Count; i++)
64                  if (a[i] != b[i]) return false;
65              return true;
66          }
67          private string[] brc = new[] { Actions1.bribery, Actions1.
                reciprocating, Actions1.inspect };
68          private string[] brnc = new[] { Actions1.bribery, Actions1.
                reciprocating, Actions1.notToInspect };
```

```
69          private string[] bnrc = new[] { Actions1.bribery, Actions1.
               notReciprocating, Actions1.inspect };
70          private string[] bnrnc = new[] { Actions1.bribery, Actions1.
               notReciprocating, Actions1.notToInspect };
71          private string[] nbc = new[] { Actions1.notBriding, Actions1.
               inspect };
72          private string[] nbnc = new[] { Actions1.notBriding, Actions1
               .notToInspect };
73          //функция для начисления терминальных выплат
74          public void SetTerminalEquity(GameTreeNode terminalNode){
75              var cfVal = terminalNode.CfValues;
76              Array.Clear(cfVal, 0, terminalNode.CfValues.Length);
77              var history = terminalNode.History;
78              if (HistoryEquals(history, brc))
79              { cfVal[0] = s.v - s.b - s.pl - s.ph; cfVal[1] = s.b - s.
                   q; cfVal[2] = s.x + s.delx; }
80              else if (HistoryEquals(history, brnc))
81              { cfVal[0] = s.v - s.b; cfVal[1] = s.b; cfVal[2] = s.x; }
82              else if (HistoryEquals(history, bnrc))
83              { cfVal[0] = -s.pl; cfVal[1] = s.r; cfVal[2] = s.y + s.
                   dely; }
84              else if (HistoryEquals(history, bnrnc))
85              { cfVal[0] = 0; cfVal[1] = s.r; cfVal[2] = s.y; }
86              else if (HistoryEquals(history, nbc))
87              { cfVal[0] = 0; cfVal[1] = 0; cfVal[2] = s.z; }
88              else if (HistoryEquals(history, nbnc))
89              { cfVal[0] = 0; cfVal[1] = 0; cfVal[2] = s.z + s.delz; }
90          }
91      }
92      public class Program{
93          static void Demo1(){
94              Console.WriteLine("Первая группа пареметров");
95              Settings1 settings1 = new Settings1() { v = 6, b = 4, pl
                   = 3, ph = 3, q = 5, r = 1, delx = 6, x = -3, dely = 4,
                    y = -2, z = -1, delz = 2 };
96              GameRooles1 r1 = new GameRooles1(settings1);
97              Cfr cfr1 = new Cfr(r1, external: false); cfr1.Init();
98              cfr1.Iterate(1);
99              Console.WriteLine("Выплаты: " + Tools.StrategyToStr(cfr1.
                   ReturnUtil()));
100             Console.WriteLine("Игровое дерево в развернутой форме:");
```

```
101             Tools.printTree(Console.Out, Tools.getTree(cfr1.root, r1)
                    );
102             Tools.getTree2(cfr1.root).ToList().ForEach(x => Tools.
                    printTree(Console.Out, x));
103             Console.WriteLine("Сумма сожалений: " + Tools.
                    GetImmRegSum(cfr1).ToString());
104             Cfr cfr2 = new Cfr(r1, external: false); cfr2.Init();
105             cfr2.Iterate(100000);
106             Console.WriteLine("Выплаты: " + Tools.StrategyToStr(cfr2.
                    ReturnUtil()));
107             Console.WriteLine("Информационные состояния:");
108             Tools.printTree(Console.Out, Tools.getTree(cfr2.root, r1)
                    );
109             Tools.getTree2(cfr2.root).ToList().ForEach(x => Tools.
                    printTree(Console.Out, x));
110             Console.WriteLine("Сумма сожалений: " + Tools.
                    GetImmRegSum(cfr2).ToString());
111             Console.WriteLine("Вторая группа пареметров");
112             Settings1 settings2 = new Settings1() { v = 12, b = 3, pl
                     = 8, ph = 8, q = 4, r = 2, delx = 9, x = 1, dely = 1,
                     y = 7, z = 5, delz = 3 };
113             GameRooles1 r2 = new GameRooles1(settings2);
114             Cfr cfr3 = new Cfr(r2, external: false);
115             cfr3.Init();
116             cfr3.Iterate(10000);
117             Console.WriteLine("Выплаты: " + Tools.StrategyToStr(cfr3.
                    ReturnUtil()));
118             Console.WriteLine("Информационные состояния:");
119             Tools.getTree2(cfr3.root).ToList().ForEach(x => Tools.
                    printTree(Console.Out, x));
120             Console.WriteLine("Сумма сожалений: " + Tools.
                    GetImmRegSum(cfr3).ToString());
121         }
122         static void Main(string[] args){
123             System.Threading.Thread.CurrentThread.CurrentCulture =
                    new System.Globalization.CultureInfo("en-US");
124             Demo1();
125         }
126     }
127 }
```

Приложение В Реализация правил игры для иерархической модели коррупции

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.IO;
namespace CorruptionModel{
    //набор всех возможных действий
    public static class A2{
        public static string DontSteal = "НеСовКр",
            Steal = "СовКр", DontCheckOfficial = "НеПровСот",
            CheckOfficial = "ПровСот", OfferBride = "ПредлВз",
            Agree = "ВыплШт", ExposeBoss = "РазРук",
            TakeBride = "ПринВз", RejectBride = "ОткклВз",
            CheckInspector = "ПровЧин", DontCheckInspector = "
               НеПровЧин";
    }
    //параметры
    public class Settings2{
        public int NumOfPlayers;
        public Employee Hierarchy;
        public int numOfOfficials { get { return NumOfPlayers - 2; }
           }
        public int inspectorNumber { get { return NumOfPlayers - 2; }
            }
        public int controllerNumber { get { return NumOfPlayers - 1;
           } }

        public double co, ci, fs, fb, fq, fns, fe, fbc, rs, rb, rq;
        public Settings2(Employee hierarchy){
            Hierarchy = hierarchy;
            NumOfPlayers = hierarchy.GetCount() + 2;
            hierarchy.UpdateBoss();
        }
    }
    //информация о сотруднике и подчиненных
    public class Employee{
        public string Name { get; set; }
        public double S, B;
```

```
35          public Employee Boss = null;
36          public Employee[] Subordinates = null;
37          public int Number = -1;
38          public Employee(string name, double s, double b, Employee[]
                subordinates = null){
39              Name = name; S = s; B = b;
40              if (subordinates != null)
41                  Subordinates = (Employee[])subordinates.Clone();
42          }
43          public int GetCount(){
44              if (Subordinates == null)
45                  return 1;
46              else
47                  return Subordinates.Sum(x => x.GetCount()) + 1;
48          }
49          public void UpdateBoss(){
50              if (Subordinates != null){
51                  foreach (var o in Subordinates){
52                      o.Boss = this;
53                      o.UpdateBoss();
54                  }
55              }
56          }
57      }
58      //реализация правил игры для иерархической модели коррупции
59      public class GameRooles2 : IGameRooles{
60          public int NumOfPlayers => s.NumOfPlayers;
61          public Settings2 s;
62          public GameRooles2(Settings2 settings){
63              s = settings;
64          }
65          bool[] steals = null;
66          //функция для создания игрового дерева
67          public GameTreeNode GenerateRoot(){
68              if (steals == null) steals = new bool[s.NumOfPlayers -
                    2];
69              Array.Clear(steals, 0, steals.Length);
70              Employee hierarchy = s.Hierarchy;
71              Queue<Employee> officials = new Queue<Employee>();
72              officials.Enqueue(hierarchy);
73              GameTreeNode offch = null;
```

```
74              GameTreeNode root = null;
75              int offNumber = 0;
76              while (officials.Count > 0){
77                  Employee tmpOfficial = officials.Dequeue();
78                  tmpOfficial.Number = offNumber;
79                  var nextOffch = new GameTreeNode(s.NumOfPlayers,
                        NodeType.PlayNode, tmpOfficial.Number, new[] { A2.
                        Steal, A2.DontSteal });
80                  nextOffch.Name = string.Format("Этап 2. Решение
                        сотрудника {0} о краже", offNumber);
81                  if (offch != null)
82                      offch.PossibleChilds = new List<GameTreeNode>(new
                            [] { nextOffch });
83                  else{
84                      GameTreeNode iich = new GameTreeNode(s.
                            NumOfPlayers, NodeType.PlayNode, s.
                            controllerNumber,
85                           new[] { A2.DontCheckInspector, A2.
                                CheckInspector });
86                      iich.Name = string.Format("Этап 1. Решение
                            инспектора о проверке");
87                      iich.PossibleChilds = new List<GameTreeNode>(new
                            [] { nextOffch });
88                      root = iich;
89                  }
90                  offch = nextOffch;
91                  if (tmpOfficial.Subordinates != null)
92                      foreach (var s in tmpOfficial.Subordinates)
93                          officials.Enqueue(s);
94                  offNumber++;
95              }
96              //терминальный узел
97              GameTreeNode terminal = new GameTreeNode(s.NumOfPlayers,
                    NodeType.TerminalNode, -1, null);
98              terminal.Name = string.Format("Этап 5: Терминальный узел"
                    );
99              var inspch = new GameTreeNode(s.NumOfPlayers, NodeType.
                    PlayNode, s.inspectorNumber,
100              new[] { A2.CheckOfficial, A2.DontCheckOfficial });
101             inspch.Name = string.Format("Этап 3: Решение чиновника о
                    проверке сотрудника {0}", 0);
```

```
102             //возможна проверка первого лица
103             offch.PossibleChilds = new List<GameTreeNode>(new[] {
                    inspch });
104             //предложить влятку либо смириться со штрафом
105             GameTreeNode off0SI = new GameTreeNode(s.NumOfPlayers,
                    NodeType.PlayNode, 0, new[] { A2.OfferBride, A2.Agree
                    });
106             off0SI.Name = string.Format("Этап 4: Кража сотрудника {0}
                     раскрыта", 0);
107             GameTreeNode inp0SIB = new GameTreeNode(s.NumOfPlayers,
                    NodeType.PlayNode, s.inspectorNumber,
108                  new[] { A2.TakeBride, A2.RejectBride });
109             inp0SIB.Name = string.Format("Этап 4: Кража сотрудника
                    {0} раскрыта. Предложена взятка", 0);
110             off0SI.PossibleChilds = new List<GameTreeNode>(new[] {
                    inp0SIB, terminal });
111             //принятие либо отклонение взятки
112             inp0SIB.PossibleChilds = new List<GameTreeNode>(new[] {
                    terminal });
113             var offISI = off0SI;
114             if (hierarchy.Subordinates != null)
115                 foreach (var s in hierarchy.Subordinates)
116                     officials.Enqueue(s);
117             while (officials.Count > 0){
118                 Employee tmpOfficial = officials.Dequeue();
119                 var nextInspch = new GameTreeNode(s.NumOfPlayers,
                        NodeType.PlayNode, s.inspectorNumber,
120                          new[] { A2.CheckOfficial, A2.DontCheckOfficial })
                             ;
121                 nextInspch.Name = string.Format("Этап 3: Решение
                        чиновника о проверке сотрудника {0}", tmpOfficial.
                        Number);
122                 //руководитель предлагает взятку
123                 GameTreeNode inpSI2EB = new GameTreeNode(s.
                        NumOfPlayers, NodeType.PlayNode, s.inspectorNumber
                        ,
124                          new[] { A2.TakeBride, A2.RejectBride });
125                 inpSI2EB.Name = string.Format("Этап 4: Кража
                        сотрудника {0} раскрыта после разоблачения.
                        Предложена взятка",
126                          tmpOfficial.Boss.Number);
```

```
127                 inpSI2EB.PossibleChilds = new List<GameTreeNode>(new
                        [] { terminal });
128                 //сотрудник доносит на руководителя, и он тоже
                        совершил кражу
129                 GameTreeNode offIBossSIE = new GameTreeNode(s.
                        NumOfPlayers, NodeType.PlayNode, tmpOfficial.Boss.
                        Number,
130                     new[] { A2.OfferBride, A2.Agree });
131                 offIBossSIE.Name = string.Format("Этап 4: Кража
                        сотрудника {0} раскрыта после разоблачения",
                        tmpOfficial.Boss.Number);
132                 offIBossSIE.PossibleChilds = new List<GameTreeNode>(
                        new[] { inpSI2EB, terminal });
133                 //взятка подчиненного
134                 GameTreeNode inpISB = new GameTreeNode(s.NumOfPlayers
                        , NodeType.PlayNode, s.inspectorNumber,
135                     new[] { A2.TakeBride, A2.RejectBride });
136                 inpISB.Name = string.Format("Этап 4: Кража сотрудника
                        {0} раскрыта. Предложена взятка", tmpOfficial.
                        Number);
137                 inpISB.PossibleChilds = new List<GameTreeNode>(new[]
                        { terminal });
138                 //совершивший кражу подчиненный проверен
139                 GameTreeNode nextOffISI = new GameTreeNode(s.
                        NumOfPlayers, NodeType.PlayNode, tmpOfficial.
                        Number,
140                     new[] { A2.ExposeBoss, A2.OfferBride, A2.Agree })
                            ;
141                 nextOffISI.Name = string.Format("Этап 4: Кража
                        сотрудника {0} раскрыта", tmpOfficial.Number);
142                 nextOffISI.PossibleChilds = new List<GameTreeNode>(
                        new[] { inpISB, offIBossSIE, terminal });
143                 //принятие инспектором решения о проверке текущего
                        подчиненного либо переход к следующему
                        подчиненному
144                 inspch.PossibleChilds = new List<GameTreeNode>(new[]
                        { nextInspch, offISI, terminal });
145                 inspch = nextInspch;
146                 offISI = nextOffISI;
147                 if (tmpOfficial.Subordinates != null)
148                     foreach (var s in tmpOfficial.Subordinates)
```

```
149                     officials.Enqueue(s);
150             }
151             inspch.PossibleChilds = new List<GameTreeNode>(new[] {
                    terminal, offISI, terminal });
152             return root;
153         }
154         //метод для обновления истори
155         public void UpdateChilds(GameTreeNode node){
156             node.Childs.Clear();
157             int offCh = 0, checkedOfficial = 0;
158             bool officialChecked = false;
159             var history = node.History;
160             int i = 0;
161             if (history.Count == 0){
162                 node.Childs.Add(node.PossibleChilds[0]);
163                 node.Childs.Add(node.PossibleChilds[0]);
164             }
165             else{
166                 i++;
167                 while (offCh < Math.Min(history.Count - 1, s.
                        NumOfPlayers - 2)){
168                     if (history[i] == A2.Steal)
169                         steals[offCh] = true;
170                     else
171                         steals[offCh] = false;
172                     i++;
173                     offCh++;
174                 }
175                 if (offCh < s.NumOfPlayers - 2){
176                     node.Childs.Add(node.PossibleChilds[0]);
177                     node.Childs.Add(node.PossibleChilds[0]);
178                 }
179                 else{
180                     while (checkedOfficial < Math.Min(history.Count -
                            1 - (s.NumOfPlayers - 2), s.NumOfPlayers - 2)
                           ){
181                         checkedOfficial++;
182                         if (history[i] == A2.CheckOfficial){
183                             officialChecked = true; i++; break;
184                         }
185                         i++;
```

```
186                     }
187                     if (!officialChecked && checkedOfficial < s.
                            NumOfPlayers - 2){
188                         //случай продолжения цикла проверок/переходов
                                к следующему
189                         if (steals[checkedOfficial])
190                             node.Childs.Add(node.PossibleChilds[1]);
191                         else
192                             node.Childs.Add(node.PossibleChilds[2]);
193                         node.Childs.Add(node.PossibleChilds[0]);
194                     }
195                     else if (officialChecked && steals[
                            checkedOfficial - 1]){
196                         //проверка выявила кражу (иначе terminal)
197                         Employee chOfficial = findByNumber(s.
                                Hierarchy, checkedOfficial - 1);
198                         while (i < history.Count && history[i] == A2.
                                ExposeBoss){
199                             chOfficial = chOfficial.Boss;
200                             i++;
201                         }
202                         if (i == history.Count){
203                             //предложить взятку, выплатить штраф,
                                    разоблачить руководителя
204                             if (chOfficial.Number == 0){
205                                 node.Childs.Add(node.PossibleChilds
                                        [0]);
206                                 node.Childs.Add(node.PossibleChilds
                                        [1]);
207                             }
208                             else{
209                                 if (steals[chOfficial.Boss.Number])
210                                     node.Childs.Add(node.
                                            PossibleChilds[1]);
211                                 else
212                                     node.Childs.Add(node.
                                            PossibleChilds[2]);
213                                 node.Childs.Add(node.PossibleChilds
                                        [0]);
214                                 node.Childs.Add(node.PossibleChilds
                                        [2]);
```

```
215                         }
216                     }
217                     else{
218                         //принять или отклонить взятку
219                         node.Childs.Add(node.PossibleChilds[0]);
220                         node.Childs.Add(node.PossibleChilds[0]);
221                     }
222                 }
223             }
224         }
225     }
226     private List<Employee> checkedOfficials = new List<Employee
        >();
227     //функция для начисления терминальных выплат
228     public void SetTerminalEquity(GameTreeNode terminalNode){
229         terminalNode.CfValues = terminalNode.CfValues == null ?
            new double[s.NumOfPlayers] : terminalNode.CfValues;
230         var cfVal = terminalNode.CfValues;
231         Array.Clear(cfVal, 0, terminalNode.CfValues.Length);
232         var history = terminalNode.History;
233         int i = 0;
234         //решение инспектора о проверке
235         bool inpCh = history[i] == A2.CheckInspector;
236         if (inpCh) cfVal[s.controllerNumber] = -s.ci;
237         i++;
238         for (int off = 0; off < s.numOfOfficials; off++){
239             if (history[i] == A2.Steal){
240                 steals[off] = true;
241                 cfVal[off] += findByNumber(s.Hierarchy, off).S;
242             }
243             else
244                 steals[off] = false;
245             i++;
246         }
247         int checkedOff = 0;
248         while (i < history.Count && history[i++] == A2.
            DontCheckOfficial)
249             checkedOff++;
250         //происходит начисление штрафа чиновнику за все
            совершенные кражи
251         cfVal[s.inspectorNumber] -= steals.Count(x => x) * s.fns;
```

```
252             if (checkedOff != s.numOfOfficials){
253                 //проверка сотрудника
254                 cfVal[s.inspectorNumber] -= s.co;
255                 if (steals[checkedOff]){
256                     //проверяемый сотрудник совершил кражу
257                     double CActionckedSteal = findByNumber(s.
                            Hierarchy, checkedOff).S;
258                     if (history[i] == A2.Agree){
259                         //сотрудник согласен со штрафом
260                         cfVal[checkedOff] += -CActionckedSteal - s.fs
                                ;
261                         cfVal[s.inspectorNumber] += s.rs + s.fns;
262                     }
263                     else if (history[i] == A2.OfferBride){
264                         //сотрудник предложил взятку
265                         i++;
266                         double offBride = findByNumber(s.Hierarchy,
                                checkedOff).B;
267                         cfVal[checkedOff] -= offBride;
268                         if (history[i] == A2.RejectBride){
269                             //чиновник отклонил взятку
270                             cfVal[checkedOff] += -CActionckedSteal -
                                    s.fs - s.fb;
271                             cfVal[s.inspectorNumber] += s.rs + s.rb +
                                    s.fns;
272                         }
273                         else if (history[i] == A2.TakeBride){
274                             //чиновник принял взятку
275                             if (!inpCh){
276                                 //инспектор не проводит проверку
277                                 cfVal[s.inspectorNumber] += offBride;
278                                 cfVal[s.controllerNumber] += -s.fbc;
279                             }
280                             else{
281                                 //инспектор проводит проверку
282                                 cfVal[checkedOff] +=
                                        -CActionckedSteal - s.fs - s.fb;
283                                 cfVal[s.inspectorNumber] += -s.fq;
284                                 cfVal[s.controllerNumber] += s.rq;
285                             }
286                         }
```

```
287                     }
288                     else if (history[i] == A2.ExposeBoss){
289                         //сотрудник донес на руководителя
290                         checkedOfficials.Clear();
291                         checkedOfficials.Add(findByNumber(s.Hierarchy
                                , checkedOff));
292                         checkedOfficials.Add(findByNumber(s.Hierarchy
                                , checkedOff));
293                         int bossNumber = findByNumber(s.Hierarchy,
                                checkedOff).Boss.Number;
294                         i++;
295                         if (!steals[bossNumber]){
296                             //руководитель не совершал кражу
297                             checkedOfficials.ForEach((x) => cfVal[x.
                                    Number] -= x.S);
298                             cfVal[checkedOfficials.Last().Number] +=
                                    -s.fs - s.fe;
299                             cfVal[s.inspectorNumber] +=
                                    checkedOfficials.Count * (s.rs + s.fns
                                    );
300                         }
301                         else{
302                             //руководитель совершил кражу
303                             double cActionckedBossSteal =
                                    findByNumber(s.Hierarchy, bossNumber).
                                    S;
304                             if (history[i] == A2.Agree){
305                                 //руководитель согласен со штрафом
306                                 checkedOfficials.ForEach((x) => cfVal
                                        [x.Number] -= x.S);
307                                 cfVal[bossNumber] +=
                                        -cActionckedBossSteal - s.fs;
308                                 cfVal[s.inspectorNumber] += (s.rs + s
                                        .fns) * (checkedOfficials.Count +
                                        1);
309                             }
310                             else if (history[i] == A2.OfferBride){
311                                 //руководитель предложил взятку
312                                 i++;
313                                 double offBossBride = findByNumber(s.
                                        Hierarchy, bossNumber).B;
```

```
314                             cfVal[bossNumber] -= offBossBride;
315
316                             if (history[i] == A2.RejectBride){
317                                 //чиновник отклонил взятку
318                                 checkedOfficials.ForEach((x) =>
                                        cfVal[x.Number] -= x.S);
319                                 cfVal[bossNumber] +=
                                        -cActionckedBossSteal - s.fs -
                                         s.fb;
320                                 cfVal[s.inspectorNumber] += (s.rs
                                         + s.fns) * (checkedOfficials.
                                        Count + 1) + s.rb;
321                             }
322                             else if (history[i] == A2.TakeBride){
323                                 //чиновник принял взятку
324                                 if (!inpCh){
325                                     //инспектор не проводит
                                            проверку
326                                     cfVal[s.inspectorNumber] +=
                                            offBossBride;
327                                     cfVal[s.controllerNumber] +=
                                            -s.fbc;
328                                 }
329                                 else{
330                                     //инспектор проводит проверку
331                                     checkedOfficials.ForEach((x)
                                            => cfVal[x.Number] -= x.S)
                                            ;
332                                     cfVal[bossNumber] +=
                                            -CActionckedSteal - s.fs -
                                             s.fb;
333                                     cfVal[s.inspectorNumber] +=
                                            -s.fq;
334                                     cfVal[s.controllerNumber] +=
                                            s.rq;
335                                 }
336                             }
337                         }
338                     }
339                 }
340             }
```

```
341             }
342         }
343         private Employee findByNumber(Employee boss, int number){
344             if (boss.Number == number)
345                 return boss;
346             if (boss.Subordinates != null)
347                 foreach (var s in boss.Subordinates){
348                     var ans = findByNumber(s, number);
349                     if (ans != null)
350                         return ans;
351                 }
352             return null;
353         }
354     }
355     public class Program{
356         static void Demo2(){
357             Employee hierarchy = new Employee("official1", 80, 24,
358                 new[] { new Employee("official1", 40, 12, null) }
359             );
360             Settings2 settings2 = new Settings2(hierarchy)
361             { co = 2, ci = 2, fs = 110, fb = 15, fq = 4, fns = 2, fe
                  = 1, fbc = 10, rs = 4, rb = 1, rq = 10 };
362             GameRooles2 gr2 = new GameRooles2(settings2);
363             Cfr cfr1 = new Cfr(gr2, false);
364             cfr1.Init();
365             cfr1.Iterate(1);
366             Console.WriteLine("Выплаты: " + Tools.StrategyToStr(cfr1.
                  ReturnUtil()));
367             Tools.printTree(Console.Out, Tools.getTree(cfr1.root, gr2
                  ));
368             Tools.getTree2(cfr1.root).ToList().ForEach(x => Tools.
                  printTree(Console.Out, x));
369             Console.WriteLine("Сумма сожалений: " + Tools.
                  GetImmRegSum(cfr1).ToString());
370             Cfr cfr2 = new Cfr(gr2, false);
371             cfr2.Init();
372             cfr2.Iterate(10000);
373             Console.WriteLine("Выплаты: " + Tools.StrategyToStr(cfr2.
                  ReturnUtil()));
374             Tools.printTree(Console.Out, Tools.getTree(cfr2.root, gr2
                  ));
```

```
375                 Tools.getTree2(cfr2.root).ToList().ForEach(x => Tools.
                        printTree(Console.Out, x));
376                 Console.WriteLine("Сумма сожалений: " + Tools.
                        GetImmRegSum(cfr2).ToString());
377             }
378             static void Main(string[] args){
379                 System.Threading.Thread.CurrentThread.CurrentCulture =
                        new System.Globalization.CultureInfo("en-US");
380                 Demo2();
381             }
382         }
383 }
```