

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

Курсовая работа

Поиск приближенного равновесия в играх с неполной информацией  
02.04.01. Математика и компьютерные науки

Исполнитель: Сычев Р.С.

гр. МКН-41 БО

Руководитель: к.ф.-м.н. Глазков Д.В.

Ярославль 2021

## Содержание

Введение . . . . .	1
1 Первая глава. Описание алгоритмов . . . . .	2
1.1 Игры в развернутой форме и равновесие Нэша . . . . .	2
1.2 Контрафактические сожаления и их минимизация . . . . .	3
2 Вторая глава. Программная реализация алгоритма . . . . .	6
2.1 Общая схема вычислений . . . . .	6
2.2 Первый пример. Покер Куна . . . . .	6
2.3 Второй пример. Домино . . . . .	8
Заключение . . . . .	11
Список использованных источников . . . . .	12
А Покер Куна . . . . .	13
Б Домино . . . . .	22

## Введение

В последнее время математическая теория игр с неполной информацией находит все большее применение в таких отраслях как теория операций, экономика, кибербезопасность и физическая безопасность. Не в последнюю очередь это происходит благодаря постоянному совершенствованию алгоритмов и увеличению производительности вычислительной техники. Частным случаем таких игр являются 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.

Существенным шагом в развитии данного направления является использование алгоритма подсчета сожалений (regret matching). Алгоритм предполагает итеративное вычисление последовательности стратегий в среднем сходящейся к оптимальному стратегическому профилю. Открытие этого метода привело к появлению ряда алгоритмов для поиска приближенного решения в играх с неполной информацией.

Данная работа посвящена рассмотрению одного из популярных в настоящее время итеративных алгоритмов - контрфактической минимизации сожалений (Conterfactual Regret Minimization) и его модификации предусматривающей использование метода монте карло (MCCFR). Данные алгоритмы появились не так давно, но на их основе уже получен ряд недостижимых до этого по сложности результатов.

Целью данной работы является описание и отработка на практике, приведенных выше, алгоритмов.

В соответствии с темой работы были поставлены следующие задачи:

- подготовить теоритическое описание алгоритмов;
- выделить некоторые игры в развернутой форме для последующего решения;
- реализовать алгоритм и произвести расчет стратегического профиля для приведенных игр.

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

# 1 Первая глава. Описание алгоритмов

## 1.1 Игры в развернутой форме и равновесие Нэша

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).  $A(h) = \{a: (h, a) \in H\}$  — доступные после нетерминальной истории  $h \in H$  действия;
- Функция  $P: H \setminus Z \rightarrow N \cup \{c\}$ , которая сопоставляет каждой нетерминальной истории  $h \in H \setminus Z$  игрока, которому предстоит принять решение, либо игрока с 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  $\{h \in H: P(h) = i\}$ , для которого  $A(h) = A(h')$  всякий раз когда  $h$  и  $h'$  принадлежат одному члену 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  $u_i: Z \rightarrow R$ . 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 бу-

дем рассматривать конечные игры в развернутой форме с нулевой суммой и полной памятью.

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

Обозначим за  $\pi^\sigma(h)$  вероятность того, что игроки достигнут  $h$  руководствуясь  $\sigma$ . Мы можем представить  $\pi^\sigma$  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 для двух игроков является поиск равновесного профиля стратегий  $\sigma$ , который удовлетворяет следующему условию

$$u_1(\sigma) \geq \max_{\sigma'_1 \in \Sigma_1} u_1(\sigma'_1, \sigma_2) \quad u_2(\sigma) \geq \max_{\sigma'_2 \in \Sigma_2} u_2(\sigma_1, \sigma'_2). \quad (1.1)$$

Такой стратегический профиль называют равновесием по Нэшу. В случае, если стратегический профили  $\sigma$  удовлетворяет условию

$$u_1(\sigma) + \epsilon \geq \max_{\sigma'_1 \in \Sigma_1} u_1(\sigma'_1, \sigma_2) \quad u_2(\sigma) + \epsilon \geq \max_{\sigma'_2 \in \Sigma_2} u_2(\sigma_1, \sigma'_2) \quad (1.2)$$

его называют  $\epsilon$  – равновесием.

Для рассматриваемых далее алгоритмов наиболее интересен вариант игры с нулевой суммой 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 [4]. Приведем связанные с ней определения. Рассмотрим дискретный отрезок времени  $T$  включающий  $T$  раундов от 1 до  $T$ . Обозначим за  $\sigma_i^t$  стратегию игрока  $i$  в раунде  $t$ .

**Определение 1** Средним общим сожалением игрока  $i$  на момент времени  $T$  называют величину

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

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

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

**Теорема 1** Если на момент времени  $T$  средние общие сожаления игроков меньше  $\epsilon$ , то  $\sigma$  является  $2\epsilon$  равновесием.

Говорят, что алгоритм выбора  $\sigma$  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. Однако, стратегии сформированные для игр с большим числом игроков могут также успешно применяться на практике [1].

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  $\sigma$ .

**Определение 2** 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  $\sigma$ . Формально

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

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

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

Немедленным контрафактическим сожалением назовем

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

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  $R_{i,imm}^{T,+}(I) = \max(R_{i,imm}^T(I), 0)$  Связь немедленных контрафактических сожалений и общих средних сожалений раскрывает следующая теорема.

**Теорема 2**  $R_i^T \leq \sum_{I \in \mathcal{I}_i} R_{i,imm}^{T,+}(I).$

Таким образом, минимизация немедленных контрафактических сожалений минимизирует общие сожаления. В свою очередь минимизация немедленного контрафактического сожаления может происходить за счет минимизации выражений под функцией максимума. Таким образом, мы приходим к понятию контрафактического сожаления

$$R_i^T(I, a) = \frac{1}{T} \sum_{t=1}^T \pi_{-i}^{\sigma^t}(I) (u_i(\sigma^t|_{I \rightarrow a}, I) - u_i(\sigma^t, I)). \quad (1.7)$$

Контрафактическое сожаление рассматривает действие в информационном состоянии. В свою очередь, для минимизации контрафактических сожалений можно применить алгоритм приближения Блэквела, который, применимо к рассматриваемым сожалениям, приведет к следующей последовательности стратегий

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

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

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

## 2 Вторая глава. Программная реализация алгоритма

### 2.1 Общая схема вычислений

В рассмотренных далее примерах рассматривалась вероятностная реализация алгоритма MCCFR. При использовании данного метода значения случайных событий генерируются перед началом каждой обучающей итерации. Данный подход позволяет сократить объем памяти и ускорить вычисления в некоторых случаях[2]. При реализации примеров были выделены следующие компоненты:

- настройки игры (произвольная параметризация составных частей игры);
- описание правил игры (зависит от настроек);
- модуль с реализацией алгоритма относительно определенных правил и настроек.

Настройки игры, например, по возможности могут включать число игроков, состав костяшек домино и т.п.

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

Сам расчет итераций CFR происходит в выделенном модуле, 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 средние стратегии игроков, которые представляют из себя приближенное коррелирующее равновесие.

### 2.2 Первый пример. Покер Куна

Покер Куна – это максимально упрощенная версия карточной игры покер[5].

Данная игра достаточно проста, чтобы быть решенной аналитически. Правила следующие:

- в игре участвуют 2 игрока;
- игра начинается с раздачи карт игрокам. Всего имеется 3 карты (1, 2 и 3). Каждый игрок получает одну карту. Причем каждый игрок знает свою карту и не знает карту другого игрока;
- по ходу игры игрокам доступны 2 действия «пасс» («п») и «ставка» («с»). Игру начинает первый игрок. Возможны следующие терминальные игровые истории: «пп», «сс», «сп», «псп» и «псс»;
- если терминальная игровая история заканчивается на «сс», то игрок с большей картой получает 2 очка, а игрок с меньшей их теряет;



— если терминальная игровая история заканчивается на «сп», то сделавший ставку игрок получает 1 очко, а спасовавший игрок теряет 1 очко;

— если терминальная игровая история заканчивается на «пп», то игроки получают по 0 очков.

Данная игра удобна для базовой проверки алгоритма CFR и часто служит в качестве примера той или иной реализации. Мы можем смоделировать дерево игры и информационные наборы игроков. После  $10^7$  итераций алгоритма удалось получить следующий профиль стратегий (Рисунок 2.1).

0: (0,6766, 0,3234)  
 1: (0,9999, 0,0001)  
 2: (0,0328, 0,9672)

п0: (0,6712, 0,3288)  
 п1: (1,0000, 0,0000)  
 п2: (0,0000, 1,0000)

пс0: (1,0000, 0,0000)  
 пс1: (0,3451, 0,6549)  
 пс2: (0,0000, 1,0000)

с0: (1,0000, 0,0000)  
 с1: (0,6642, 0,3358)  
 с2: (0,0000, 1,0000)

Рисунок 2.1 — Стратегия для покера Куна

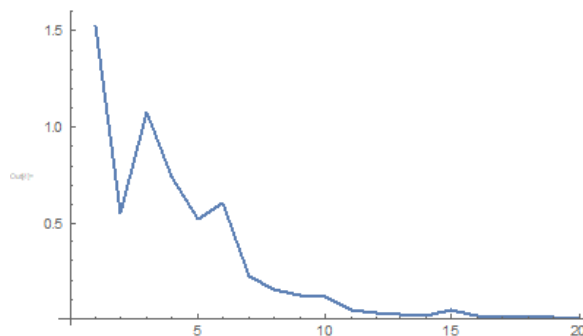


Рисунок 2.2 — График расчетной эксплуатируемости стратегий для покера Куна

На рисунке 2.2 представлен график приближенной эксплуатируемости стратегий. На приведенном графике, и в дальнейшем, горизонтальная ось содержит экспоненциальные отметки о числе итераций по основанию 2. Основная часть кода программы представлена в приложении А.

### 2.3 Второй пример. Домино

В данной работе в качестве основного объекта исследования была выбрана игра «Домино». Однако, спортивный вариант игры обладает значительной комбинаторной сложностью и было бы трудно хранить в памяти все дерево игры. В связи с этим, в данной работе рассматривались некоторые упрощенные варианты.

Из соображений вычислительной сложности целесообразно рассматривать правила игры следующего вида:

- имеется набор из не более чем 10 костяшек домино;
- игроки имеют на руках 2, 3 (размер руки) костяшки;
- игра может происходить с 2-мя, 3-мя или 4-мя игроками;
- находящиеся не на руках костяшки раздаются по мере развития игры
- игру начинает первый игрок
- все костяшки в процессе игры выкладываются в единственную линию
- если ход возможен, то он происходит по обычным правилам;
- в случае, если ход текущего игрока невозможен, то игра на этом заканчивается, и игроки получают выплаты (победитель забирает все очки, и т.к. необходима нулевая сумма, то проигравшая сторона эти очки теряет).

Приведенные выше правила игры позволяют на практике сформировать полное решение по методу MCCFR. Фрагмент кода приложения для расчета стратегий представлен в приложении Б.

Для проверки полученного приложения был проведен ряд тестовых запусков. Первый тест состоял в определении профиля стратегий для случая игры с полной информацией. Был взят набор из четырех костяшек, которые раздавались поровну двум игрокам. Это крайне простая игровая ситуация, но по ней можно судить о работоспособности в целом. Ниже приведен полученный профиль стратегий в корневом узле (Рисунок 2.3).

```
[обзор дерева решений]
текущий узел: .
тип узла: Игровой
игрок: 0
выберите переход
0) [0|1]: [0|1][1|1] (0,00002) [0|1][1|2] (0,00001) [0|1][2|2] (0,00002)
1) [1|1]: [0|1][1|1] (0,99998) [1|1][1|2] (0,99998) [1|1][2|2] (0,50000)
2) [1|2]: [0|1][1|2] (0,99999) [1|1][1|2] (0,00002) [1|2][2|2] (0,99999)
3) [2|2]: [0|1][2|2] (0,99998) [1|1][2|2] (0,50000) [1|2][2|2] (0,00001)
-1) ввести информационный набор
-2) закончить обзор
```

Рисунок 2.3 — Стратегии в корне игры для первого примера домино

```

[Настройки]
Число игроков:
2
Размер руки игрока:
2
Набор костяшек:
[0|0][0|1][0|2][1|2]

```

Рисунок 2.4 — настройки для первого примера домино

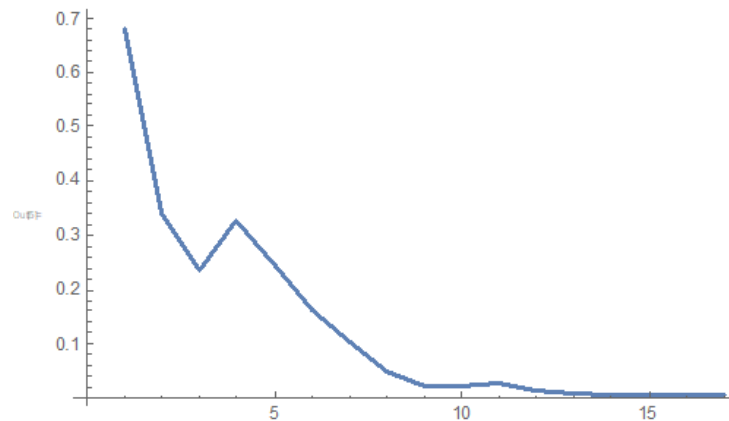


Рисунок 2.5 — График эксплуатированности стратегий для первого примера домино

Для второго теста был выбран набор из шести костяшек домино из которых каждый игрок в начале раунда получал на руки две, а остальные 2 раздавались по ходу игры. Приведем фрагмент полученной стратегии (Рисунок 2.6).

```

[обзор дерева решений]
текущий узел: .
тип узла: Игровой
игрок: 0
выберите переход
0) [0|0]: [0|0][0|1] (0,98399) [0|0][0|2] (0,98813) [0|0][1|1] (0,03686) [0|0][1|2] (0,98964) [0|0][2|2] (0,02069)
1) [0|1]: [0|0][0|1] (0,01601) [0|1][0|2] (0,92002) [0|1][1|1] (0,01486) [0|1][1|2] (0,56124) [0|1][2|2] (0,00400)
2) [0|2]: [0|0][0|2] (0,01187) [0|1][0|2] (0,07998) [0|2][1|1] (0,00652) [0|2][1|2] (0,12828) [0|2][2|2] (0,00604)
3) [1|1]: [0|0][1|1] (0,96314) [0|1][1|1] (0,98514) [0|2][1|1] (0,99348) [1|1][1|2] (0,99559) [1|1][2|2] (0,02351)
4) [1|2]: [0|0][1|2] (0,01036) [0|1][1|2] (0,43876) [0|2][1|2] (0,87172) [1|1][1|2] (0,00441) [1|2][2|2] (0,00043)
5) [2|2]: [0|0][2|2] (0,97931) [0|1][2|2] (0,99600) [0|2][2|2] (0,99396) [1|1][2|2] (0,97649) [1|2][2|2] (0,99957)
-1) ввести информационный набор
-2) закончить обзор

```

Рисунок 2.6 — Стратегии в корне игры для второго примера домино

```

Число игроков:
2
Размер руки игрока:
2
Набор костяшек:
[0|0][0|1][0|2][1|1][1|2][2|2]

```

Рисунок 2.7 — настройки для второго примера домино

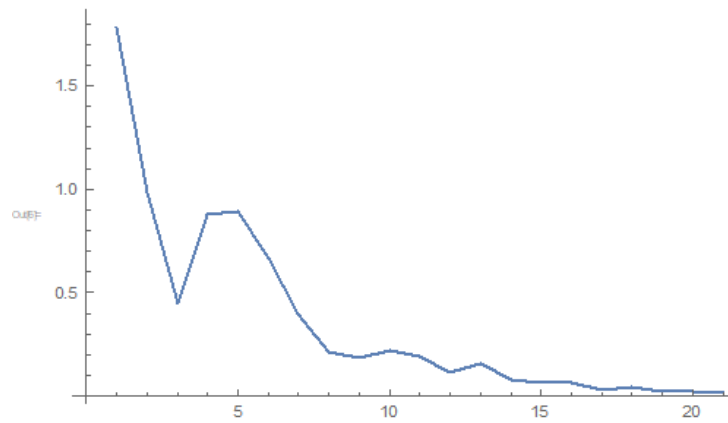


Рисунок 2.8 — График эксплуатированности стратегий для второго примера домино

Однако, данные примеры не представляют большой комбинаторной сложности. Для проверки производительности был выбран вариант игры на 10 костяшек для двух игроков. Каждый игрок получал на руки по 3 костяшки и оставшиеся 4 раздавались по ходу игры. Под эксплуатированностью стратегий подразумевается возможная выгода оппонента, если он будет менять только свою стратегию. Будем под ней понимать максимальный приближенно рассчитанный разброс выигрыша изменившего свою стратегию игрока по сравнению с оригинальным профилем. Назовем эту величину  $\tau$ . Ниже представлен график расчетной эксплуатированности стратегий в зависимости от числа обучающих итераций (Рисунок 2.10).

```
[Настройки]
Число игроков:
2
Размер руки игрока:
3
Набор костяшек:
[0|0][0|1][0|2][1|1][1|2][2|2][0|3][1|3][2|3][3|3]
```

Рисунок 2.9 — настройки для третьего примера домино

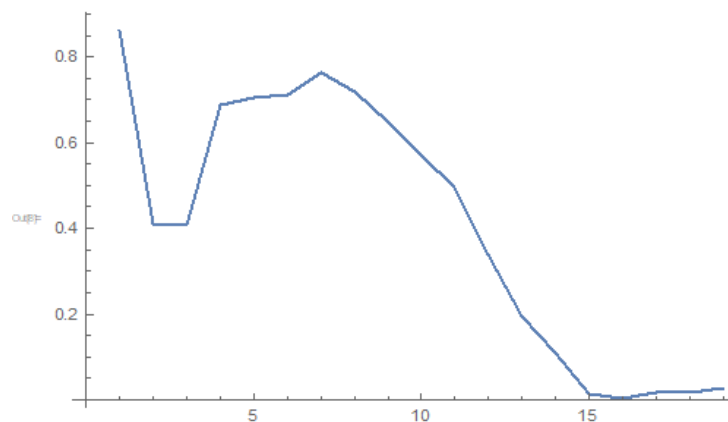


Рисунок 2.10 — График эксплуатированности стратегий для третьего примера домино

## Заключение

В данной работе был продемонстрирован один из лучших на данный момент алгоритмов для приближенного решения игр с неполной информацией. Но несмотря на наличие вполне обобщенных методов, приходится уделять большое внимание разбору частных случаев. Основной проблемой при решении подобных задач является экспоненциальный рост сложности вычислений в зависимости от увеличения числа возможных действий игроков. В связи с этим приходится идти на различные ухищрения с целью получить практически ценный аналог оригинальной задачи. К подобным приемам относят использование метода монте-карло, построение игровых абстракций и многие другие оптимизации.

В качестве объекта исследования была выбрана вполне популярная настольная игра домино. Однако, данной игре уделено довольно мало внимания в контексте рассматриваемого алгоритма. Автор данной работы постарался частично исправить данный недостаток, хотя полученные результаты пока что скромны. Был рассмотрен сильно упрощенный, по сравнению с спортивным, вариант игры с минимумом абстракций. Однако, даже подобный упрощенный вариант раскрывает широкое разнообразие смешанных стратегий, а теоретическая база позволяет говорить о строгой математической обоснованности полученных решений. Для непосредственного расчета стратегий была реализована компьютерная программа.

Дальнейшим развитием данной темы может служить построение более общих абстракций для данной игры. Решения в подобной сфере могут быть полезны как с точки зрения концепции, так и с точки зрения частных методов и оптимизаций.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *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.
2. *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.
3. *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.
4. *Hart, Sergiu*. A simple adaptive procedure leading to correlated equilibrium / Sergiu Hart, Andreu Mas-Colell. — Econometrica, 68(5), September 2000, pages 1127–1150.
5. *W., Kuhn H.* "Simplified Two-Person Poker". In Kuhn, H. W.; Tucker, A. W. (eds.). Contributions to the Theory of Games. 1. / Kuhn H. W. — Princeton University Press, 1950, pp. 97–103.

## Приложение А Покер Куна

```
1 using System;
2 using System.Linq;
3 using System.Collections.Generic;
4 using System.Text;
5
6 namespace GameSolving{
7     public class GameTreeNode{
8         public string history; //игровая история
9         public GameTreeNodeType nodeType; //тип узла
10        public List<GameTreeNode> childNodes; //дочерние узлы
11        public GameTreeNode Parent; //родительский узел
12        public GameSettings settings; //настройки игры
13        public int player; //номер игрока
14        public int[] infoSets; //информационные состояния игроков
15        public double[,] tStrategy; //итерационная стратегия
16        public double[,] regretSum; //сумма сожалений
17        public double[,] strategySum; //сумма стратегий
18        public double[,] avgStrategy; //средняя стратегия
19        public double[,] util; //вспомогательная переменная
20        public double[] p; //вероятности выбора каждого игрока
21        public double[] cfValues; //контрафактические значения
22        public double[] pays; //ставки игроков
23        public GameTreeNode(GameSettings settings){ //установка
            начальных значений
24            this.settings = settings;
25            tStrategy = new double[settings.NumOfInfoSets, settings.
                NumOfActions];
26            regretSum = new double[settings.NumOfInfoSets, settings.
                NumOfActions];
27            strategySum = new double[settings.NumOfInfoSets, settings
                .NumOfActions];
28            avgStrategy = new double[settings.NumOfInfoSets, settings
                .NumOfActions];
29            util = new double[settings.NumOfActions, settings.
                NumOfPlayers];
30            cfValues = new double[settings.NumOfPlayers];
31            pays = new double[settings.NumOfPlayers];
32            p = new double[settings.NumOfPlayers];
33            childNodes = new List<GameTreeNode>();
34            infoSets = null;
```

```

35     }
36     public void CalcTStrategy(double weight){//расчет
        итерационной стратегии
37         double normSum = 0;
38         int infoSet = infoSets[player];
39         for (int rl = 0; rl < settings.NumOfActions; rl++){
40             if (regretSum[infoSet, rl] > 0){
41                 tStrategy[infoSet, rl] = regretSum[infoSet, rl];
42                 normSum += regretSum[infoSet, rl];
43             }
44             else
45                 tStrategy[infoSet, rl] = 0;
46         }
47         for (int a = 0; a < settings.NumOfActions; a++){
48             if (normSum > 0)
49                 tStrategy[infoSet, a] /= normSum;
50             else
51                 tStrategy[infoSet, a] = 1.0 / settings.
                    NumOfActions;
52             strategySum[infoSet, a] += tStrategy[infoSet, a] *
                    weight;//подсчет суммы стратегий с учетом веса
                    реализации
53         }
54     }
55     public void CalcAvgStrategy(){//расчет средних стратегий для
        всех информационных состояний
56         for (int i = 0; i < settings.NumOfInfosets; i++){
57             double normSum = 0;
58             for (int a = 0; a < settings.NumOfActions; a++){
59                 if (strategySum[i, a] > 0){
60                     avgStrategy[i, a] = strategySum[i, a];
61                     normSum += strategySum[i, a];
62                 }
63                 else
64                     avgStrategy[i, a] = 0;
65             }
66             for (int a = 0; a < settings.NumOfActions; a++){
67                 if (normSum > 0)
68                     avgStrategy[i, a] /= normSum;
69                 else

```



```

70         avgStrategy[i, a] = 1.0 / settings.
           NumOfActions;
71     }
72 }
73 }
74 }
75 public enum GameTreeNodeType{//типы игровых узлов
76     PlayNode ,
77     ChanceNode ,
78     TerminalNode
79 }
80 public class KuhnPokerGameClass : IGameRooles{//модуль с
    правилами игры
81     Random rand;//генератор случайных чисел
82     GameSettings settings;//настройки игры
83     public KuhnPokerGameClass(){
84         rand = new Random();//инициализация генератора случайных
            чисел
85         settings = new GameSettingsKP();
86     }
87     public GameTreeNode GenerateTree(int[] infoSets){//генерация
        игрового дерева
88         GameTreeNode tmpNodeR;//корень дерева
89         tmpNodeR = new GameTreeNode(settings);
90         tmpNodeR.history = "";//присваивание игровой истории
91         tmpNodeR.nodeType = GameTreeNodeType.PlayNode;//
            присваивание типа узла
92         tmpNodeR.player = 0;//присваивание игрока
93         tmpNodeR.infoSets = infoSets;//задание
            дополнительногоинформационного набора
94
95         GameTreeNode tmpNodeP;
96         tmpNodeP = new GameTreeNode(settings);
97         tmpNodeP.history = "p";
98         tmpNodeP.nodeType = GameTreeNodeType.PlayNode;
99         tmpNodeP.player = 1;
100        tmpNodeP.infoSets = infoSets;
101        tmpNodeR.childNodes.Add(tmpNodeP);//добавление дочернего
            узла
102
103        GameTreeNode tmpNodeB;

```

```

104 tmpNodeB = new GameTreeNode( settings );
105 tmpNodeB.history = "b";
106 tmpNodeB.nodeType = GameTreeNodeType.PlayNode;
107 tmpNodeB.player = 1;
108 tmpNodeB.infoSets = infoSets;
109 tmpNodeR.childNodes.Add(tmpNodeB);
110
111 GameTreeNode tmpNodePP;
112 tmpNodePP = new GameTreeNode( settings );
113 tmpNodePP.history = "pp";
114 tmpNodePP.nodeType = GameTreeNodeType.TerminalNode;
115 tmpNodePP.pays = new double [] { 1.0, 1.0 }; // ставки
    игроков
116 tmpNodePP.player = -1;
117 tmpNodePP.infoSets = infoSets;
118 tmpNodeP.childNodes.Add(tmpNodePP);
119
120 GameTreeNode tmpNodePB;
121 tmpNodePB = new GameTreeNode( settings );
122 tmpNodePB.history = "pb";
123 tmpNodePB.nodeType = GameTreeNodeType.PlayNode;
124 tmpNodePB.player = 0;
125 tmpNodePB.infoSets = infoSets;
126 tmpNodeP.childNodes.Add(tmpNodePB);
127
128 GameTreeNode tmpNodePBP;
129 tmpNodePBP = new GameTreeNode( settings );
130 tmpNodePBP.history = "pbp";
131 tmpNodePBP.nodeType = GameTreeNodeType.TerminalNode;
132 tmpNodePBP.pays = new double [] { 1.0, 2.0 };
133 tmpNodePBP.player = -1;
134 tmpNodePBP.infoSets = infoSets;
135 tmpNodePB.childNodes.Add(tmpNodePBP);
136
137 GameTreeNode tmpNodePBB;
138 tmpNodePBB = new GameTreeNode( settings );
139 tmpNodePBB.history = "pbb";
140 tmpNodePBB.nodeType = GameTreeNodeType.TerminalNode;
141 tmpNodePBB.pays = new double [] { 2.0, 2.0 };
142 tmpNodePBB.player = -1; //признак терминального узла
143 tmpNodePBB.infoSets = infoSets;

```

```

144         tmpNodePB.childNodes.Add(tmpNodePBB);
145
146         GameTreeNode tmpNodeBP;
147         tmpNodeBP = new GameTreeNode(settings);
148         tmpNodeBP.history = "bp";
149         tmpNodeBP.nodeType = GameTreeNodeType.TerminalNode;
150         tmpNodeBP.pays = new double[] { 2.0, 1.0 };
151         tmpNodeBP.player = -1;
152         tmpNodeBP.infoSets = infoSets;
153         tmpNodeB.childNodes.Add(tmpNodeBP);
154
155         GameTreeNode tmpNodeBB;
156         tmpNodeBB = new GameTreeNode(settings);
157         tmpNodeBB.history = "bb";
158         tmpNodeBB.nodeType = GameTreeNodeType.TerminalNode;
159         tmpNodeBB.pays = new double[] { 2.0, 2.0 };
160         tmpNodeBB.player = -1;
161         tmpNodeBB.infoSets = infoSets;
162         tmpNodeB.childNodes.Add(tmpNodeBB);
163         return tmpNodeR;
164     }
165     public GameSettings GetSettings(){
166         return settings;
167     }
168     public void SetInfoSets(int[] infoSets){//установка
        информационных состояний
169         HashSet<int> otherInfoSets = new HashSet<int>();
170         int infoSet;
171         for (int player = 0; player < settings.NumOfPlayers;
            player++){
172             do{
173                 infoSet = rand.Next(0, settings.NumOfInfoSets);
174             }
175             while (otherInfoSets.Contains(infoSet));
176             infoSets[player] = infoSet;
177             otherInfoSets.Add(infoSet);
178         }
179     }
180     public void SetTerminalEquity(GameTreeNode node){//установка
        выплат в терминальных узлах
181         int winner = 0;

```

```

182         switch (node.history){
183             case "bp":
184                 winner = 0;
185                 break;
186             case "pbp":
187                 winner = 1;
188                 break;
189             default:
190                 for (int p = 1; p < settings.NumOfPlayers; p++){
191                     if (node.infoSets[p] > node.infoSets[winner])
192                         winner = p;
193                 }
194                 break;
195         }
196         for (int p = 0; p < settings.NumOfPlayers; p++){
197             node.cfValues[p] = 0;
198         }
199         for (int p = 0; p < settings.NumOfPlayers; p++){
200             node.cfValues[p] -= node.pays[p];
201             node.cfValues[winner] += node.pays[p];
202         }
203     }
204 }
205 public class GameSettingsKP : GameSettings{//настройки игры
206     public override int NumOfPlayers { get { return 2; } }
207     public override int NumOfInfosets { get { return 3; } }
208     public override int NumOfActions { get { return 2; } }
209 }
210 public class TreeCfr{//основной алгоритм
211     int[] infoSets;
212     public GameTreeNode root;//корень дерева
213     public GameSettings settings;//настройки
214     public IGameRooles gameRooles;//правила игры
215     HashSet<int> activePlayers;//обучающиеся игроки
216     public double[] util;//цена игры
217     public TreeCfr(IGameRooles gameRooles, GameTreeNode root =
218         null, HashSet<int> players = null){
219         this.root = root;
220         this.gameRooles = gameRooles;
221         settings = gameRooles.GetSettings();
222         this.infoSets = new int[settings.NumOfPlayers];

```

```

222         this.activePlayers = players;
223         if (root != null)//задание информационных состояний
224             SetInfoSetsRec(root);
225     }
226     private void SetInfoSetsRec(GameTreeNode node){//рекурсивное
        присваивание информационных состояний
227         node.infoSets = this.infoSets;
228         foreach (var c in node.childNodes)
229             SetInfoSetsRec(c);
230     }
231     public void Init(int iterations){
232         if (root == null)
233             root = gameRooles.GenerateTree(infoSets);
234         for (int player = 0; player < settings.NumOfPlayers;
            player++)
235             root.p[player] = 1.0;//изначально все вероятности
                равны 1
236         double[] resultUtil = new double[settings.NumOfPlayers];
237         int count = iterations;
238         while (count— > 0){//запуск итераций
239             gameRooles.SetInfoSets(this.infoSets);//установка
                информационных наборов
240             RegRec(root);//запуск процедуры расчета
241             for (int player = 0; player < settings.NumOfPlayers;
                player++){
242                 resultUtil[player] += root.cfValues[player];//
                    обновление суммы ожидаемых выплат
243             }
244         }
245         for (int player = 0; player < settings.NumOfPlayers;
            player++){
246             resultUtil[player] /= iterations;//подсчет цены игры
247         }
248         util = resultUtil;
249     }
250     public void RegRec(GameTreeNode node)//рекурсивный расчет
        одной итерации алгоритма
251     {
252         if (node.nodeType == GameTreeNodeType.TerminalNode){
253             gameRooles.SetTerminalEquity(node);//установка
                терминальных значений

```

```

254         return ;
255     }
256     if (activePlayers == null || activePlayers.Contains(node.
        player)){
257         node.CalcTStrategy(node.p[node.player]); //
            динамическая стратегия
258     }
259     else{//статичная стратегия
260         node.tStrategy = node.avgStrategy.Clone() as double
            [,];
261     }
262     CalcUtil(node);//расчет ожидаемых выплат
263     CalcRegrets(node);//подсчет сожалений
264 }
265 public void CalcUtil(GameTreeNode node){
266     for (int player = 0; player < settings.NumOfPlayers;
        player++){
267         node.cfValues[player] = 0;
268     }
269     for (int a = 0; a < settings.NumOfActions; a++){
270         GameTreeNode child = node.childNodes[a]; //перебор
            действий
271         for (int player = 0; player < settings.NumOfPlayers;
            player++){
272             if (player != node.player)//обновление
                вероятностей игроков
273                 child.p[player] = node.p[player];
274             else
275                 child.p[player] = node.tStrategy[node.
                    infoSets[node.player], a] * node.p[player
                        ];
276         }
277         RegRec(child);//рекурсивный вызов для связанного с
            действием дочернего узла
278         for (int player = 0; player < settings.NumOfPlayers;
            player++){
279             node.util[a, player] = child.cfValues[player]; //
                расчет ожидаемых выплат
280             node.cfValues[player] += node.tStrategy[node.
                infoSets[node.player], a] * node.util[a,
                    player];

```

```

281         }
282     }
283 }
284 public void CalcRegrets(GameTreeNode node){
285     double regret;
286     for (int a = 0; a < settings.NumOfActions; a++){
287         regret = node.util[a, node.player] - node.cfValues[
288             node.player]; //вычисление сожаления
289         node.regretSum[node.infoSets[node.player], a] +=
290             GetPiMinus1(node.p, node.player) * regret;
291     }
292 }
293 public double GetPiMinus1(double[] p, int i){//метод для
294     вычисления вероятности оппонентов
295     double ans = 1.0;
296     for (int player = 0; player < settings.NumOfPlayers;
297         player++)
298         if (player != i)
299             ans *= p[player];
300     return ans;
301 }
302 }
303 }

```

## Приложение Б Домино

```
1 using System;
2 using System.Linq;
3 using System.Collections.Generic;
4 using System.Text;
5 namespace DominoSolver{
6     public class DominoGameTreeNode{//игровой узел в данном случае
        более специфичен
7         public int iterations = 0;//число посещений узла
8         public GameTreeNodeType NodeType { get; set; }
9         public List<int> possibleInfosets { get; set; }
10        public List<List<DominoGameTreeNode>> childNodesPerInfoSets {
            get; set; }//дочерние узлы в зависимости от инф.
            состояния
11        public List<DominoGameTreeNode> childNodes { get; set; }//
            прямая ссылка на значение из предыдущего списка
12        public double[] TStrategy { get; set; }//итерационная
            стратегия
13        public List<double[]> RegretSum { get; set; }
14        public List<double[]> StrategySum { get; set; }
15        public int MaxNumOfActions { get; set; }
16        public double[,] Util { get; set; }
17        public double[] CfValues { get; set; }
18        public double[] P { get; set; }
19        public DominoGameTreeNode Parent { get; set; }
20        public DominoSettings Settings { get; set; }
21        public int Player { get; set; }
22        public int InfoSet { get; set; }
23        public DominoSettings settings;
24        public Node lastNode;//последняя выложенная костяшка
25        public List<Node> nodes;//перечень костяшек в истории
26        public int left;//число точек слева
27        public int right;//число точек справа
28        public List<DominoGameTreeNode> childs;
29        public bool pass = false;//признак пропуска хода
30        public int numOfPasses = 0;//для аппроксимации прокатов
31        public int length = 0;//длина истории
32        public int[] playersInfosets;//информационные состояния
            игроков
33        public DominoGameTreeNode(DominoSettings settings = null)
34        {
```



```

35         if (settings != null)
36             this.settings = settings;
37         else
38             this.settings = new DominoSettings();
39         Parent = null;
40         nodes = new List<Node>();
41         left = -1; //признак начала игры
42         right = -1;
43         childs = new List<DominoGameTreeNode>();
44         pass = false;
45         Counters.TREE_NODES++;
46     }
47     public DominoGameTreeNode(Node node, DominoGameTreeNode
48         parent, int left, int right, bool isPass = false){
49         this.settings = parent.settings;
50         this.Parent = parent;
51         this.nodes = new List<Node>(parent.nodes);
52         if (node != null){
53             lastNode = node;
54             if (parent == null || (left != parent.left || (left
55                 == parent.left && right == parent.right && node.b
56                 == parent.left)))
57                 //добавляем слева
58                 this.nodes.Insert(0, node);
59             else
60                 this.nodes.Add(node);
61         }
62         else{
63             if (parent != null)
64                 lastNode = parent.lastNode;
65         }
66         this.left = left;
67         this.right = right;
68         childs = new List<DominoGameTreeNode>();
69         length = parent.length + 1;
70         pass = isPass;
71         if (pass)
72             this.numOfPasses = parent.numOfPasses + 1;
73         else
74             this.numOfPasses = 0;
75         Counters.TREE_NODES++;

```

```

73     }
74     public void SetNodeType(){
75         if (childs.Count != 0)
76             NodeType = GameTreeNodeType.PlayNode;
77         else
78             NodeType = GameTreeNodeType.TerminalNode;
79     }
80     public override string ToString(){
81         StringBuilder sb = new StringBuilder();
82         foreach (var node in nodes)
83             sb.Append(node.ToString());
84         return sb.ToString();
85     }
86     public double[] GetAvgStrategy(){
87         double[] avgStrategy = new double[childNodesPerInfoSets[
88             InfoSet].Count];
89         double normSum = 0;
90         for (int a = 0; a < childNodesPerInfoSets[InfoSet].Count;
91             a++){
92             if (StrategySum[InfoSet][a] > 0){
93                 avgStrategy[a] = StrategySum[InfoSet][a];
94                 normSum += StrategySum[InfoSet][a];
95             }
96             else
97                 avgStrategy[a] = 0;
98         }
99         for (int a = 0; a < childNodesPerInfoSets[InfoSet].Count;
100             a++){
101             if (normSum > 0)
102                 avgStrategy[a] /= normSum;
103             else
104                 avgStrategy[a] = 1.0 / childNodesPerInfoSets[
105                     InfoSet].Count;
106         }
107         return avgStrategy;
108     }
109     public void CalcTStrategy(double weight){
110         double normSum = 0;
111         for (int rl = 0; rl < childNodes.Count; rl++){
112             if (RegretSum[InfoSet][rl] > 0){
113                 TStrategy[rl] = RegretSum[InfoSet][rl];

```

```

110         normSum += RegretSum[InfoSet][rl];
111     }
112     else
113         TStrategy[rl] = 0;
114 }
115 for (int a = 0; a < childNodes.Count; a++){
116     if (normSum > 0)
117         TStrategy[a] /= normSum;
118     else
119         TStrategy[a] = 1.0 / childNodes.Count;
120
121     StrategySum[InfoSet][a] += TStrategy[a] * weight;
122 }
123 }
124 }
125 public enum GameTreeNodeType{
126     PlayNode,
127     ChanceNode, //не используем
128     TerminalNode
129 }
130 public class DominoCfr{
131     double[] resultUtil;
132     public int iterSum { get; private set; }
133     public DominoGameTreeNode root;
134     public DominoSettings settings;
135     public DominoGameRooles gameRooles;
136     HashSet<int> activePlayers;
137     public DominoCfr(DominoGameRooles gameRooles,
138         DominoGameTreeNode root = null, HashSet<int> players =
139         null){
140         this.root = root;
141         this.gameRooles = gameRooles;
142         settings = gameRooles.settings;
143         this.activePlayers = players;
144     }
145     public void Init(bool exploitMode = false, HashSet<int>
146         players = null){
147         if (!exploitMode){
148             if (root == null)
149                 root = gameRooles.GenerateTree() as
150                 DominoGameTreeNode;

```

```

147     }
148     else{
149         this.activePlayers = players;
150         gameRooles.ClearActivePlayersRegretSum(activePlayers ,
            root);
151     }
152     for (int player = 0; player < settings.NumOfPlayers;
        player++)
153         root.P[player] = 1.0;
154     resultUtil = new double[settings.NumOfPlayers];
155     iterSum = 0;
156 }
157 public void Iterate(int iterations){
158     int count = iterations;
159     while (count— > 0){//можно дообучить модель
160         gameRooles.SetInfosets(root);
161         for (int p = 0; p < settings.NumOfPlayers; p++)
162             RegRec(root);
163         for (int player = 0; player < settings.NumOfPlayers;
            player++){
164             resultUtil[player] += root.CfValues[player];
165         }
166     }
167     iterSum += iterations;
168 }
169 public double[] ReturnUtil(){//расчет цены игры
170     double[] resUtil = resultUtil.Clone() as double[];
171     for (int player = 0; player < settings.NumOfPlayers;
        player++){
172         resUtil[player] /= iterSum;
173     }
174     return resUtil;
175 }
176 public void RegRec(DominoGameTreeNode node){
177     node.iterations++;
178     if (node.NodeType == GameTreeNodeType.TerminalNode){
179         gameRooles.SetTerminalEquity(node);
180         return;
181     }
182     if (activePlayers == null || activePlayers.Contains(node.
        Player)){

```

```

183         //динамическая стратегия
184         node.CalcTStrategy(node.P[node.Player]);
185     }
186     else
187     {
188         //статичная стратегия
189         double[] avgStrategy = node.GetAvgStrategy();
190         for (int i = 0; i < avgStrategy.Length; i++){
191             node.TStrategy[i] = avgStrategy[i];
192         }
193     }
194     CalcUtil(node);
195     CalcRegrets(node);
196 }
197 public void CalcUtil(DominoGameTreeNode node){
198     for (int player = 0; player < settings.NumOfPlayers;
199         player++){
200         node.CfValues[player] = 0;
201     }
202     for (int a = 0; a < node.childNodes.Count; a++){
203         DominoGameTreeNode child = node.childNodes[a] as
204             DominoGameTreeNode;
205         for (int player = 0; player < settings.NumOfPlayers;
206             player++){
207             if (player != node.Player)
208                 child.P[player] = node.P[player];
209             else
210                 child.P[player] = node.TStrategy[a] * node.P[
211                     player];
212         }
213         RegRec(child);
214         for (int player = 0; player < settings.NumOfPlayers;
215             player++){
216             node.Util[a, player] = child.CfValues[player];
217             node.CfValues[player] += node.TStrategy[a] * node
218                 .Util[a, player];
219         }
220     }
221 }
222 public void CalcRegrets(DominoGameTreeNode node){
223     double regret;

```

```

218         for (int a = 0; a < node.childNodes.Count; a++){
219             regret = node.Util[a, node.Player] - node.CfValues[
                node.Player];
220             node.RegretSum[node.InfoSet][a] += GetPiMinus1(node.P
                , node.Player) * regret;
221             node.RegretSum[node.InfoSet][a] = Math.Max(node.
                RegretSum[node.InfoSet][a], 0); // cfr+
222         }
223     }
224     public double GetPiMinus1(double[] p, int i){
225         double ans = 1.0;
226         for (int player = 0; player < settings.NumOfPlayers;
            player++)
227             if (player != i)
228                 ans *= p[player];
229         return ans > 0 ? ans : 0.000000001;
230     }
231 }
232 }

```