МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Ярославский государственный университет имени П. Г. Демидова»

Кафедра теоретической информатики

Сдано на кафедру
«_____» _____ 2024 г.
Заведующий кафедрой,
д. ф.-м. н.

_____ Е. В. Кузьмин

Выпускная квалификационная работа

Стратегии для «Игры в подстановки»

по направлению подготовки 02.03.02 Фундаментальная информатика и информационные технологии

Научный руководитель
к. ф.-м. н., доцент

_____ А. В. Смирнов
«_____» _____ 2024 г.

Студент группы ИТ-41БО

_____ А. Н. Гладков
«_____» _____ 2024 г.

Ярославль, 2024

Реферат

Объем ?? с., ?? гл., ?? рис., ?? табл., ?? источников, ?? прил.

Стратегия, сравнительный анализ, контекстно-свободная грамматика, теория игр, чистая стратегия, смешанная стратегия.

Объектом исследования является «Игра в подстановки». Это антагонистическая игра с двумя участниками, основанная на контекстно-свободной грамматике.

Целью работы является разработка эффективных стратегий для этой игры.

В процессе работы была разработана программа для моделирования «Игры в подстановки», разработаны стратегии и графический интерфейс для запуска и просмотра партий, а также графический интерфейс для непосредственного участия в игре.

В результате исследования был проведён сравнительный анализ разработанных чистых и смешанных стратегий и были выявлены наиболее эффективные стратегии.

Содержание

Введение

В работе рассматриваются стратегии для «Игры в подстановки». В этой игре дана контекстно-свободная грамматика, продукции которой разбиты на 11 групп. Игроки кидают кубики, и по результатам броска определяется группа продукций, из которой игроку предстоит выбрать одну продукцию, чтобы применить её к имеющимся выводам. Целью игроков является получение слов как можно большей суммарной длины.

Целью исследования является нахождение эффективных стратегий для этой игры.

1. Постановка задачи

Дана контекстно-свободная грамматика, разбитая на 11 групп продукций. Все продукции в одной группе начинаются с одного нетерминала, но при этом продукции из разных групп могут начинаться одним нетерминалом. Например:

1. $S \to ABC$
2. $A \to aA \mid B$
3. $A \to bc \mid CA$
4. $A \to a$
5. $S \to AB \mid AC$
6. $B \to b$
7. $B \to BBB$
8. $C \to AA \mid c$
9. $C \to A \mid B \mid c$
10. $C \to T \mid A$
11. $B \to abacabade$

В начале игры задаётся количество ходов n, которое может сделать каждый из игроков. Имеется общий банк продукций и 2 шестигранных кубика.

Правила игры:

1. Каждый игрок делает n ходов. Ходы осуществляются по очереди.
2. Ход начинается с броска обоих кубиков. Сумма чисел на верхних гранях кубиков определяет группу продукций, из которой игрок должен будет выбрать одну продукцию.
3. Если это группа продукций, начинающаяся с нетерминала S, то можно либо создать новый вывод, либо использовать эту группу продукций для получения следующего шага вывода (если нетерминал S присутствует в каком-то текущем выводе). Количество начатых игроком выводов не ограничивается. Если это группа продукций, начинающаяся другим нетерминалом, то если есть вывод с соответствующим нетерминалом, игрок обязан применить продукцию из выпавшей группы к одному из подходящих выводов, если же вывода с соответствующим нетерминалом нет, группа продукций сохраняется в банке.
4. Если после применения какой-то продукции в выводах игрока присутствует нетерминал, для которого есть подходящая группа продукций в банке, игрок обязан выбрать продукцию из этой группы и применить эту продукцию к одному из подходящих выводов. Если в банке находится несколько подходящих групп продукций, игрок выбирает любую из них.

5. Если на очередном шаге в текущем выводе или в нескольких выводах есть несколько подходящих нетерминалов, заменить можно любой на выбор игрока.
6. Ход заканчивается, когда в банке нет ни одной продукции, подходящей для какого-либо текущего вывода игрока, либо когда выпавшая группа продукций не подходит ни для одного текущего вывода игрока.

Цель игры — получить как можно больше слов из терминальных символов как можно большей суммарной длины, то есть получить больше очков.

По окончании ходов всех игроков сентенциальные формы, содержащие нетерминалы, удаляются, а подсчёт очков идёт по правилу: 3 очка за каждое слово и по 1 очку за каждую букву.

Приведём пример хода для этой игры. Пусть имеется такое состояние банка (количество групп продукций в банке подписано справа от соответствующей группы в квадратных скобках):

1. $S \to ABC[0]$
2. $A \to aA \mid B[0]$
3. $A \to bc \mid CA[2]$
4. $A \to a[2]$
5. $S \to AB \mid AC[0]$
6. $B \to b[0]$
7. $B \to BBB[0]$
8. $C \to AA \mid c[1]$
9. $C \to A \mid B \mid c[0]$
10. $C \to T \mid A[0]$
11. $B \to abacabade[1]$

Пусть у игрока есть вывод: AB. Выпали числа на кубиках 3 и 4. Сумма равна 7. Тогда можно получить вывод ab, применив выпавшую продукцию $B \to b$ и продукцию из группы под номером 4 $A \to a$.

Также можно получить вывод $ccbb$, применив выпавшую продукцию $B \to b$, после этого продукцию $A \to CA$ из 3 группы, продукцию $C \to c$ из 8 группы, продукцию $A \to bc$ из 3 группы. Цепочка выводов: $AB \Rightarrow Ab \Rightarrow CAb \Rightarrow cAb \Rightarrow ccbb$.

Пусть на кубиках выпали числа 5 и 5. Сумма равна 10. В данном случае игрок не может совершить никакой ход, поскольку нет выводов, содержащих нетерминал C, даже при том, что в банке есть подходящие продукции, поэтому группа продукций 10 будет помещена в банк.

2. Обзор родственных исследований

Человеком придумано много различных игр, связанных с составлением слов. К примеру, можно взглянуть на игру „Scrabble“, или в русском варианте «Эрудит». Возьмём краткое изложение её правил с сайта [3]. Игровое поле состоит из 15×15, то есть 225 квадратов, на которые участники игры выкладывают буквы, составляя тем самым слова. В начале игры каждый игрок получает 7 случайных букв (всего их в игре 100). На середину игрового поля выкладывается первое слово, затем следующий игрок может добавить слово из своих букв. Слова выкладываются либо слева направо, либо сверху вниз.

Наша «Игра в подстановки» похожа на неё, с тем отличием, что в «Эрудите» скорее не общий банк, а общие выводы — общее игровое поле, и не частные выводы, а частный банк — у каждого игрока свой набор букв.

Также есть игра „Number Scrabble“, взятая с сайта [4], — игра, где из имеющихся на руках карточек надо составлять правильные математические выражения. Для неё также есть разработанное приложение, чтобы можно было сыграть (см. [5]). А математические выражение очень удобно задавать КС-грамматикой.

Есть диссертация [6], в которой рассматривается близкая тема. В этой диссертации исследуется целый класс контекстно-свободных игр. В своей наиболее простой форме контекстно-свободные игры могут быть получены с помощью небольшого смягчения проблемы выводимости слов в контекстно-свободных грамматиках: дана контекстно-свободная грамматика G и сентенциальные формы w и w', возможно ли вывести w' из w в соответствии с правилами грамматики G?

В версии с двумя игроками первому игроку доступен выбор нетерминала, который необходимо заменить, а второй игрок выбирает какую последовательность символов, соответствующую правилам грамматики, вставить. Вопрос состоит в том, имеет ли первый игрок выигрышную стратегию, если его задача получить определённую сентенциальную форму или одну из сентенциальных форм из определённого множества.

В работе рассматривается вопрос разрешимости а также проводится анализ сложности этой задачи при разных ограничениях и дополнительных условиях.

3. Основные определения

Возьмём определение грамматики из книги «Введение в теорию формальных языков» В. А. Соколова (см. [1]):

Определение 1. *Грамматикой G называется следующая упорядоченная четвёрка объектов $G = (N,T,S,P)$, где*

- *N — конечное множество нетерминальных символов, называемых переменными (или нетерминалами);*
- *T — конечное множество терминальных символов, называемых терминалами;*
- *S — начальный символ, $S \in N$;*
- *P — конечное множество правил вывода, называемых продукциями.*

Здесь и далее предполагается, что множества N и T не пусты и не пересекаются. Каждая продукция $p \in P$ имеет вид $\alpha \to \beta$, где $\alpha \in (N \cup T)^+$, $\beta \in (N \cup T)^*$.

Если $\alpha \in (N \cup T)^*$, то любая последовательность вида $S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \ldots \Rightarrow \alpha_n \Rightarrow \alpha$ называется выводом строки α в грамматике G; при этом если $\alpha \in T^*$, то α называется сентенцией, а α_i — сентенциальной формой (в грамматике G).

Возьмём определение контекстно свободной грамматики из книги «Введение в теорию формальных языков» В. А. Соколова (см. [1]):

Определение 2. *Грамматика $G = (N,T,S,P)$ называется контекстно-свободной (или КС-грамматикой), если любая ее продукция имеет вид $A \to \alpha$, где $A \in N$, $\alpha \in (N \cup T)^*$.*

Также там приводится определение бесполезных продукций: *бесполезные продукции* — те, которые никогда не участвуют в выводе никакой строки терминалов.

На основе этого определим бесполезные нетерминалы и выводы: *бесполезный нетерминал* — нетерминал, с которого начинаются только бесполезные продукции; *бесполезный вывод* — вывод, в котором есть хоть один бесполезный нетерминал.

Также определим *начальные продукции* — те, с помощью которых можно создавать новые выводы; это продукции вида $S \to \ldots$.

Напомним определение динамической игры с совершенной информацией из книги А. В. Захарова «Теория игр в общественных науках» (см. [2]).

Для того чтобы описать *динамическую игру*, нам нужно следующее. Во-первых, нам (как и в статической игре) нужно определить множество игроков. Для того, чтобы моделировать случайные события, влияющие на выигрыш игроков,

нам необходимо ввести ещё одного игрока — природу. Таким образом, мы имеем $I = \{1,...,N\} \cup \{ \text{природа} \}$.

Во-вторых, нужно определить, в каком порядке игроки ходят, и какие действия им доступны на каждом ходе.

Наконец, в-третьих, нам надо определить, как выигрыши игроков зависят от ходов, которые были сделаны. Формально в каждой конечной вершине дерева игры мы определяем выигрыши для каждого игрока.

Каждому ходу, который игрок делает в какой-то вершине, соответствует вершина, в которой игра оказывается после сделанного хода. В любом дереве игры для любой вершины, кроме начальной, однозначно задаётся *история игры* — последовательность вершин, через которые игра уже успела пройти. В частности, для любой вершины (опять же, кроме начальной) существует ровно одна вершина, непосредственно предшествующая данной.

Определение 3. *Каждое информационное множество содержит одну или несколько вершин, в которых принимает решения какой-то один игрок. Если несколько вершин находятся в одном информационном множестве, то игрок, принимающий решения в этих вершинах, не знает, в какой именно вершине он находится.*

В тех играх, в которых игроки могут наблюдать за предыдущими действиями всех других игроков, включая «природу», каждое информационное множество содержит ровно одну вершину.

Определение 4. *В игре Γ в развёрнутой форме множество чистых стратегий игрока i есть $S_i = \prod_{h_i \in H_i} A(h_i)$, где H_i содержит все информационные множества, в которых делает ход игрок i, а $A(h_i)$ обозначает все действия, доступные игроку i в информационном множестве h_i.*

Определение 5. *Множеством смешанных стратегий игрока i будет $\Delta^{|\prod_{h_i \in H_i} A(h_i)|-1}$, где Δ^n означает симплекс размерности n.*

Совокупность дерева игры и информационных множеств игроков позволяет нам определить множество стратегий для каждого игрока.

Определение 6. *Пусть каждое информационное множество в игре Γ содержит ровно одну вершину. Такая игра называется игрой с совершенной информацией.*

Нашу «Игру в подстановки» можно рассматривать как игру в развёрнутой форме с совершенной информацией с тремя игроками: игрок 1, игрок 2 и природа или случайность (то, что выпало на кубиках). Дерево игры будет выглядеть так: первым ход делает природа (бросок кубика первым игроком), далее ход переходит к игроку 1, который совершает ход, соответствующий правилам игры, далее снова следует ход природы (бросок кубика вторым игроком), далее ход переходит игроку 2, который совершает ход, соответствующий правилам игры, далее снова

ход совершает природа (бросок кубика первым игроком), и так далее, пока игрок 1 и игрок 2 оба не сделают указанное количество ходов.

Природа в каждой вершине, где ей принадлежит ход, имеет 11 чистых стратегий (сумма на кубиках от 2 до 12), но она всегда играет смешанную стратегию со следующим распределением вероятностей:

Таблица 3.1

Распределение вероятности выпадения

x_i	2	3	4	5	6	7	8	9	10	11	12
$p(x_i)$	1/36	2/36	3/36	4/36	5/36	6/36	5/36	4/36	3/36	2/36	1/36

Здесь x_i — сумма, выпавшая на кубиках, а $p(x_i)$ — шанс её выпадения.

Введём следующие обозначения: W_i — множество выводов, имеющихся у игрока i в данный момент.

Множество всех групп продукций обозначим P, группу продукций под номером i обозначим P_i, продукцию под номером j из группы продукций i обозначим P_{ij}.

Определение 7. *Также определим множество A_{ij} — множество выводов игрока i, допустимое для группы продукций P_j. Пусть группа продукций P_j имеет следующий вид: $B \rightarrow$ …, тогда*

$$A_{ij} = \{w \mid w \in W_i \wedge w = \alpha B \beta, \forall \alpha, \beta \in (N \cup T)^* \vee w = S \wedge B = S\},$$

то есть это множество выводов, содержащих нетерминал левой части продукции, и символ S, если в левой части продукции стоит S.

4. Описание стратегий

4.1. Случайная стратегия

Задумка стратегии простая — совершать случайный доступный ход. Данная стратегия делает все выборы с равными весами, но для более простого описания других стратегий опишем алгоритм получения случайного хода с произвольными весами.

Распределение весов 1. *Равные веса:*

$$\omega_1(\alpha) = 1,$$

где α — группа продукций, продукция или слово.

Алгоритм 1. *Алгоритм получения случайного хода.*

Определим вспомогательную функцию:

$$\varphi(r,(b_1,b_2,\ldots,b_n)) = \begin{cases} 1, & r \leqslant b_1, \\ n, & r > \sum_{i=1}^{n} b_i, \\ x : \sum_{i=1}^{x-1} b_i < r \leqslant \sum_{i=1}^{x} b_i, & \text{иначе}. \end{cases}$$

0. *Пронумеруем все выводы игрока.*
1. ***Совершение первого хода.*** *Пусть по результатам броска кубиков выпала группа продукций под номером d. Если множество A_{pd}, где p — текущий игрок, пусто, то ход игрока заканчивается. Иначе переходим к шагу 3.*
2. ***Выбор группы продукций.*** *Зададим множество номеров допустимых групп продукций I такое, что $j \in I \iff A_{pj} \neq \varnothing$ и группа продукций P_j содержится в банке. Если множество I пусто, то ход игрока заканчивается. Получим вектор $B = (\omega_1(P_j) \mid \forall j \in I)$. Генерируем случайное число $r_1 \in \mathbb{R}$ с равномерным распределением на отрезке $\left[0, \sum_{i=1}^{|B|} b_i\right]$. Находим число $d = \varphi(r_1,B)$.*
3. ***Выбор продукции из группы продукций.*** *Пусть группа продукций P_d имеет следующий вид: $A \to \alpha_1 | \ldots | \alpha_n$. Получим вектор $C = (\omega_1(\alpha_i) \mid i \in \overline{1,n})$. Генерируем случайное число $r_2 \in \mathbb{R}$ с равномерным распределением на отрезке $\left[0, \sum_{i=1}^{|C|} c_i\right]$. Находим число $k = \varphi(r_2,C)$.*
4. ***Выбор подходящего вывода.*** *Находим вектор $D = (\omega_1(\alpha) \mid \forall \alpha \in A_{pd})$. Генерируем случайное число $r_3 \in \mathbb{R}$ с равномерным распределением на отрезке $\left[0, \sum_{i=1}^{|D|} d_i\right]$. Получаем число $a = \varphi(r_3,D)$.*
5. ***Завершение хода.*** *Применяем продукцию P_{dk} к выводу с номером a. Переходим ко 2 шагу алгоритма.*

4.2. Глупая стратегия коротких слов

Идея стратегии в том, чтобы за как можно меньшее число шагов получить сентенцию.

Стратегия основана на анализе продукций грамматики. Для каждой продукции и группы продукций считается метрика того, насколько быстро её можно полностью привести к сентенции.

Ещё перед началом игры стратегия высчитывает метрику M_1 для каждой группы продукции.

Метрика 1 (M_1)**.**

1. *Метрика строки α равна количеству нетерминалов в этой строке.*
2. *Метрика продукции $A \to \alpha$ равна метрике строки α, то есть*

$$M_1(A \to \alpha) = M_1(\alpha).$$

3. *Метрика группы продукций $A \to \alpha_1 \mid ... \mid \alpha_k$ равна минимуму метрик продукций $A \to \alpha_i, 1 \leqslant i \leqslant k$, то есть*

$$M_1(A \to \alpha_1 \mid ... \mid \alpha_k) = \min_{1 \leqslant i \leqslant k} M_1(A \to \alpha_i).$$

Будем считать, что строка (продукция) α имеет лучшую метрику 1, чем другая строка (продукция) β, если $M_1(\alpha) < M_1(\beta)$.

Также перед началом игры для каждой группы продукций P_j находим продукцию k_j с лучшей метрикой 1.

Алгоритм 2.

1. *Пусть по результатам броска кубиков игроком p выпала группа продукций P_d. Если множество A_{pd} пусто, то ход заканчивается. Иначе находим метрику M_1 для каждого вывода из A_{pd}, и выбираем тот вывод, у которого метрика M_1 наилучшая. Если это S, то создаём новый вывод из правой части продукции k_d. Иначе применяем к выводу, соответствующему этому элементу продукцию k_d.*
2. *Выбираем группу продукций P_j с наилучшей метрикой, для которой A_{pj} не пусто и P_j содержится в банке. Если таких групп продукций нет, то заканчиваем ход.*
3. *Находим метрику M_1 для каждого вывода из A_{pj}, выбираем вывод с наилучшей метрикой 1 и применяем к нему продукцию k_j. Переходим к шагу 2.*

Приведём пример, для упрощения возьмём грамматику с четырьмя группами продукций:

1. $S \to A[0]$

2. $A \to BB \mid aA[2]$
3. $B \to b[2]$
4. $A \to a[1]$

Вычисленная метрика для этих продукций:

$$M_1(S \to A) = 1;$$

$$M_1(A \to BB \mid aA) = \min\{M_1(A \to BB), M_1(A \to aA)\} = \min\{2,1\} = 1;$$

$$M_1(B \to b) = 0;$$

$$M_1(A \to a) = 0.$$

Пусть у игрока имеется вывод A, и выпала группа продукций 2. Тогда будет совершён ход $A \Rightarrow aA \Rightarrow a$. В выпавшей группе продукций будет выбрана продукция $A \to aA$. Далее будет выбрана продукция из банка $A \to a$. Заметим, что можно построить и сентенцию большей длины, но целью алгоритма является за как можно меньшее число шагов получить сентенцию.

Также заметим, что если бы количество групп продукций $A \to a$ в банке было равно 0 или имелась бы грамматика без этой группы продукций, то был бы совершён ход $A \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA$. Таким образом, сентенция и вовсе не была бы получена, хотя это возможно. Соответствующая цепочка выводов: $A \Rightarrow BB \Rightarrow bB \Rightarrow bb$.

Именно из-за этого стратегия названа «глупой» — её можно «обмануть». К примеру, добавить продукции, образующие цикл, или бесполезные продукции.

4.3. Стратегия коротких слов

Идея этой стратегии также состоит в том, чтобы за меньшее число шагов строить сентенцию.

Была разработана метрика, сконструированная так, чтобы избежать проблем, присущих метрике 1, которые были кратко описаны выше.

Метрика 2 (M_2). *Она строится итеративно:*

1. *Изначально метрика каждой продукции равна* 0.
2. *Метрика группы продукций* $A \to \alpha_1 \mid \ldots \mid \alpha_k$ *равна максимуму из метрик продукций этой группы:*

$$M_2(A \to \alpha_1 \mid \ldots \mid \alpha_k) = \max_{1 \leqslant i \leqslant k} M_2(A \to \alpha_i).$$

3. *Метрика продукции пересчитывается следующим образом: метрика продукции* $A \to \alpha$ *равна метрике строки* α: $M_2(A \to \alpha) = M_2(\alpha)$. *Здесь*

метрика строки равна произведению метрики каждого символа, содержащегося в строке: $M_2(\alpha) = \prod_{a \in \alpha} M_2(a)$. *Метрика терминала равна 1:* $M_2(a) = 1, \forall a \in T$. *Метрика нетерминала равна сумме произведений метрики соответствующей группы продукции и её вероятности выпадения:*

$$M_2(A) = \sum M_2(q) \cdot p(q) \, \forall A \in N,$$

где q *— группа продукций, вида* $A \to$ ..., $p(q)$ *— шанс выпадения группы продукций* q.

4. *Переходим к шагу 2.*

Будем считать, что строка (продукция) α имеет лучшую метрику 2, чем другая строка (продукция) β, если $M_2(\alpha) > M_2(\beta)$.

Заметим, что в алгоритме не указан момент остановки.

Утверждение 1. *Метрика 2 каждой группы продукций при совершении итераций алгоритма сходится к некоторому числу от* 0 *до* 1.

Тогда можно задать необходимую точность δ, при достижении которой алгоритм будет останавливаться:

$$\forall A : M_2(A) \cdot \delta > |M_2^*(A) - M_2(A)|,$$

где A — продукция, M_2 — метрика, вычисленная на предыдущей итерации, M_2^* — метрика, вычисленная на текущей итерации.

Для совершения хода используется Алгоритм 2 с использованием метрики 2.

Приведём пример вычисленной метрики и пример хода, для упрощения возьмём грамматику с 3 группами продукций:

1. $S \to A[0]$
2. $A \to BB \mid aA[1]$
3. $B \to b[2]$

Распределение вероятностей: $p(1) = \frac{1}{3}, p(2) = \frac{1}{3}, p(3) = \frac{1}{3}$.

Тогда результаты вычисления метрики 2 с точностью $\delta = 0{,}05$ будут следующими:

$$M_2(S \to A) = 0{,}037037;$$

$$M_2(A \to BB) = 0{,}111111;$$

$$M_2(A \to aA) = 0{,}037037;$$

$$M_2(B \to b) = 1.$$

Пусть у игрока имеется вывод A и выпала продукция 2, тогда был бы совершён ход $A \Rightarrow BB \Rightarrow bB \Rightarrow bb$. В выпавшей группе продукций будет выбрана

продукция $A \to BB$, так как у неё больше метрика. Дальше будет применяться продукция $B \to b$. Заметим, что при использовании метрики 1 на этой грамматике и того же алгоритма невозможно получить сентенцию. Продукция $A \to aA$ всегда будет считаться лучше, чем $A \to BB$, таким образом, нетерминал A никогда не заменится на терминальный символ.

Теперь рассмотрим следующую грамматику:

1. $S \to A[0]$
2. $A \to BB \mid aA[1]$
3. $B \to b[2]$
4. $A \to a[0]$

Распределение вероятностей: $p(1) = \frac{1}{4}, p(2) = \frac{1}{4}, p(3) = \frac{1}{4}, p(4) = \frac{1}{4}$.
Значения метрики 2 будут следующими:

$$M_2(S \to A) = 0{,}333329;$$

$$M_2(A \to BB) = 0{,}0625;$$

$$M_2(A \to aA) = 0{,}333329;$$

$$M_2(B \to b) = 1;$$

$$M_2(A \to a) = 1.$$

Тогда будет совершён ход: $A \to aA \to aaA$. Заметим, что в таком случае не будет получена сентенция, хотя, как показано в предыдущем примере, её можно получить. Отсюда и возникла идея следующей стратегии — смотреть на несколько шагов вперёд.

4.4. Переборная стратегия

Идея этой стратегии в том, чтобы перебрать разные варианты использования продукций из банка, чтобы получить лучший вывод. Изначально планировалось собирать сразу терминальные строки, осуществляя полный перебор вариантов, но в процессе разработки была обнаружена проблема: программа начинает работать слишком долго. Поэтому было принято решение ограничить глубину перебора, получилась не одна стратегия, а целая группа стратегий с параметром — глубиной перебора. А вместо поиска сентенции — поиск вывода с лучшей метрикой 2. Опишем алгоритм перебора продукций из банка.

Алгоритм 3 (перебор ходов)**.**

Вход: *вывод α, относительно которого производится перебор, число k — глубина перебора.*

Выход: *число f — метрика лучшего получаемого вывода, ход m, при котором получается вывод с указанной метрикой.*

Алгоритм перебора задаётся рекурсивно:

1. *Если $k = 0$, то считаем метрику 2 вывода α и заканчиваем алгоритм, возвращая пустой ход и вычисленную метрику.*
2. *Найти множество номеров доступных групп продукций Q: $q \in Q \iff$ группа продукций P_q применима к рассматриваемому выводу α, и группа продукций P_q есть в банке. Также задаём число $b_f = 0$ и ход b_m пустым.*
3. *Если множество Q пусто, переходим к шагу 6, иначе выбираем любой элемент q из множества Q, удаляем группу продукций P_q из банка.*
4. *Перебираем $i \in \overline{1,|P_q|}$. Получаем вывод α', применяя продукцию P_{qi} к текущему выводу α, запускаем алгоритм 3 с параметрами α' и $k - 1$, получаем из него число f и ход m. Если $f > b_f$, то присваиваем $b_f = f$, а b_m получаем, дополнив ход m информацией о группе продукций P_q и продукции P_{qi}, повторяем текущий шаг алгоритма, пока не переберём все значения i.*
5. *Возвращаем P_q в банк, удаляем q из множества Q. Переходим к шагу 3.*
6. *Возвращаем число b_f и ход b_m.*

Теперь представим алгоритм, определяющий поведение стратегии.

Алгоритм 4. *Алгоритм имеет параметр k — глубину перебора.*

1. *Пусть P_d — выпавшая группа продукций по результатам броска кубиков игроком p. Если A_{pd} пусто, заканчиваем алгоритм.*
2. *Найдём среди множества A_{pd} вывод α с наилучшей метрикой 2. Инициализируем число $b_f = 0$ и пустой ход b_m.*
3. *Перебираем $i \in \overline{1,|P_q|}$. Получаем вывод α' применением продукции P_{di} к выводу α. Запускаем алгоритм 3 для вывода α' и глубины k. Из него получим число f и ход m. Если $f > b_f$, то присваиваем $b_f = f$, а b_m получаем, дополнив ход m информацией о группе продукций P_d и продукции P_{di}, повторяем текущий шаг алгоритма, пока не переберём все продукции в группе P_d.*
4. *Если $\alpha = S$, создаём новый вывод S и применяем к нему ход b_m, иначе применим ход b_m к выводу α.*
5. *Далее находим множество выводов A, содержащее выводы игрока, к которым можно применить какую-либо продукцию из банка: $a \in A \iff \forall a \in W_p$ и $\exists i$: группа продукций P_i применима к выводу a и P_i есть в банке. Если множество A пусто, то ход игрока заканчивается. Найдём в множестве A вывод α, имеющий наилучшую метрику 2. Применим алгоритм 3 для вывода α и глубины k, получим из него ход m. Если ход m не пуст, то применим ход m к выводу α и повторим этот шаг алгоритма, иначе завершаем ход.*

Отметим, что для применения алгоритма надо вычислить метрику 2 для имеющихся групп продукций.

Приведём пример на основе данной грамматики:

1. $S \to A[0]$
2. $A \to BB \mid aA[1]$
3. $B \to b[2]$
4. $A \to a[0]$

Распределение вероятностей: $p(1) = \frac{1}{4}, p(2) = \frac{1}{4}, p(3) = \frac{1}{4}, p(4) = \frac{1}{4}$.
Метрики будут следующими:

$$M_2(S \to A) = 0{,}333329;$$

$$M_2(A \to BB) = 0{,}0625;$$

$$M_2(A \to aA) = 0{,}333329;$$

$$M_2(B \to b) = 1;$$

$$M_2(A \to a) = 1.$$

Пусть выпала продукция $S \to A$. Тогда при глубине перебора 4 будет совершён ход: $S \Rightarrow A \Rightarrow BB \Rightarrow bB \Rightarrow bb$.

4.5. Умная случайная стратегия

Идея заключается в модификации случайной стратегии с помощью изменения весов выбираемых вариантов. В качестве первого способа нахождения этих весов возьмём метрику 2. Она обладает хорошими свойствами: её значение для выводов, которые нельзя преобразовать в терминальные, и бесполезных продукций равно нулю; её значения меньше для слов, которые можно за меньшее количество ходов привести к сентенции, следовательно, стратегия не будет строить излишне длинные выводы.

Распределение весов 2. *Веса в соответствии с метрикой 2:*

$$\omega_2(\alpha) = M_2(\alpha),$$

где α — группа продукций, продукция или слово.

Стратегия показывает результаты схожие со стратегией коротких слов. Часто она ведёт себя как стратегия коротких слов. Ниже объяснено, почему это происходит.

4.6. Проблемы метрики 2 и создание новой

В ходе более глубокого анализа метрики 2 был обнаружен большой недостаток. Из-за того, как она устроена, она очень быстро стремится к нулю при

увеличении количества нетерминалов в правой части продукции. Хоть и для нахождения лучшей продукции она вполне применима, то для использования в качестве весов для случайной стратегии, она плохо подходит.

Для примера возьмём грамматику меньшего масштаба, будто бы в игре используется 2 кубика с тремя гранями:

1. $S \to AA|DD$
2. $A \to BB$
3. $D \to ee$
4. $C \to DD$
5. $B \to CC$

Тогда получатся следующие значения метрики:

$$M_2(S \to AA) \approx 1{,}03988 \cdot 10^{-18};$$

$$M_2(S \to DD) \approx 0{,}11111.$$

Разница составляет много порядков, тогда при выборе между соответствующими продукциями можно с большой долей вероятности сказать, что всегда будет выбрана вторая продукция. Таким образом, умная случайная стратегия во многих случаях ведёт себя как стратегия коротких слов.

Идея новой метрики заключается в изменении правила получения метрики слова: она равна минимальной метрике нетерминала, встречающегося в слове, делённой на общее количество нетерминалов в слове. Опишем это формально:

Метрика 3 (M_3)**.** *Она строится итеративно:*

1. *Изначально метрика каждой продукции равна* 0.
2. *Метрика группы продукций* $A \to \alpha_1 \mid ... \mid \alpha_k$ *равна максимуму из метрик продукций этой группы:*

$$M_3(A \to \alpha_1 \mid ... \mid \alpha_k) = \max_{1 \leqslant i \leqslant k} M_3(A \to \alpha_i).$$

3. *Метрика продукции пересчитывается следующим образом: метрика продукции* $A \to \alpha$ *равна метрике строки* α*:* $M_3(A \to \alpha) = M_3(\alpha)$*. Здесь метрика строки равна минимуму метрики нетерминала, встречающегося в строке, делённому на количество всех нетерминалов в строке* α*:*

$$M_3(\alpha) = \min_{a \in \alpha \wedge a \in N} M_3(a) \div count(a \in \alpha \wedge a \in N).$$

Метрика терминала равна 1: $M_3(a) = 1 \; \forall a \in T$*. Метрика нетерминала равна сумме произведений метрики соответствующей группы продукции и*

вероятности выпадения этой продукции:

$$M_3(A) = \sum M_3(q) \cdot p(q) \forall A \in N,$$

где q — группа продукций вида $A \to \ldots$, $p(q)$ — шанс выпадения группы продукций q.

4. *Переходим к шагу 2.*

Тогда получатся следующие значения метрики:

$$M_3(S \to AA) \approx 0{,}00011;$$

$$M_3(S \to DD) \approx 0{,}16666.$$

Видно, что масштаб уже совершенно другой, но это всё ещё слишком большая разница. Поэтому дополнительно имеет смысл рассмотреть разные варианты преобразований этих весов. К примеру, извлечь из них квадратный корень. До преобразования разница составляла 8748 раз, то после преобразования разница составит $\approx 93{,}53$ раз.

4.7. Улучшенная умная случайная стратегия

Стратегия основана на формировании весов с использованием метрики 3 с последующим извлечением корня.

Распределение весов 3. *Веса в соответствии с метрикой 3 с последующим извлечением корня:*

$$\omega_3(\alpha) = \sqrt{M_3(\alpha)},$$

где α — группа продукций, продукция или слово.

Стратегия ведёт себя лучше умной случайной стратегии, но всё так же проигрывает переборной стратегии.

4.8. Адаптивная случайная стратегия

Перечисленные ранее стратегии всё так же стремятся создавать короткие выводы. Следующая же стратегия предназначена для построения более длинных выводов.

Поведение стратегии делится на 2 этапа. На первом этапе будут применяться продукции, не ведущие к уменьшению количества нетерминалов. На втором этапе будут совершаться ходы, ведущие к наискорейшему окончанию всех выводов.

Отметим, что адаптивная случайная стратегия также является представителем класса смешанных стратегий, но с другим правилом смешения, основанном не на вероятностном выборе, а на оценке текущего положения в игре.

На втором этапе мы рассмотрим применение переборной стратегии и стратегии коротких слов и сравним их эффективность. Для первого этапа требуется разработка новой стратегии. Определим **инвертированную случайную стратегию**, задав веса для случайного выбора.

Распределение весов 4. *Инвертированные веса:*

$$\omega_4(\alpha) = \begin{cases} 0, & \textit{если } M_3(\alpha) = 0, \\ 1{,}01 - M_3(\alpha), & \textit{иначе}, \end{cases}$$

где α — группа продукций, продукция или слово.

В формирование весов включён отдельный случай при метрике равной нулю, чтобы бесполезные продукции имели минимальный приоритет. В другом случае вес формируется вычитанием метрики из $1{,}01$, отступ на $0{,}01$ от 1 сделан для того, чтобы в случае когда на выбор имеется только продукция с метрикой равной 1 и бесполезная продукция, предпочтение отдавалось первой.

Отметим, что инвертированная случайная стратегия не является самостоятельной, так как она почти никогда не будет получать законченные выводы.

Вернёмся к адаптивной случайной стратегии. Важным вопросом является момент перехода от первого этапа ко второму. Переход будет совершаться при выполнении следующего условия:

$$\{\text{число нетерминалов во всех выводах игрока}\} \cdot \Omega > \{\text{число оставшихся ходов}\}.$$

Стратегия получается довольно сильно зависимой от параметра Ω; если его взять слишком большим, то стратегия станет очень похожей на переборную стратегию, иначе же, если его взять слишком маленьким, то у стратегии не будут получаться законченные выводы. Вопрос нахождения оптимального значения параметра Ω будет рассмотрен в главе 6.

4.9. Смешанные стратегии

Идея заключается в выборе на каждом шаге одной из чистых стратегий с какой-то вероятностью. Основываться смешанные стратегии будут не на всех чистых стратегиях, а лишь на четырёх: стратегии коротких слов, улучшенной умной случайной стратегии, переборной стратегии и адаптивной случайной стратегии.

Для формирования смешанных стратегий, как в работе [7], введём термин *очки вероятности*. *Очки вероятности* — это числа, сопоставляемые каждой чистой стратегии в соответствии с одним из вариантов распределения и характеризующие вероятность каждой чистой стратегии внутри смешанной. Вероятность чистой

стратегии внутри смешанной равняется её очкам вероятности, делённым на сумму всех очков вероятности.

Для распределения очков вероятности было выбрано несколько вариантов распределения:

1. Самый простой вариант из них — «Равные вероятности», в котором каждая чистая стратегия внутри смешанной получает по одному очку вероятности.
2. Контрастирующий с предыдущим вариантом вариант «Пики», при котором более выгодные стратегии получают намного больше очков вероятности, чем остальные.
3. Вариант «Линейный рост», при таком варианте все очки вероятности различны и отличаются на 1; чем выгоднее стратегия, тем больше очков вероятности она получает.
4. Вариант «Процент от побед», в нём каждая чистая стратегия получает число очков вероятности равное отношению суммы её побед к сумме всех побед чистых стратегий, это определяется на основе статистики вычислительных экспериментов.

5. Программная реализация

5.1. Описание реализации используемых стратегиями классов

Рассмотрим программную реализацию стратегий и использованные для этого классы. Для написания программы был выбран язык C#, выбран он был за то, что он предоставляет удобные средства для разработки оконных приложений и имеет большой спектр возможностей, а также заранее скомпилированная программа будет работать почти на любом компьютере с ОС Windows.

Первым мы рассмотрим класс `GameSettings` — этот класс хранит данные о конфигурации текущей партии. Класс является неизменяемым, его поля открыты только для чтения. Поле `NumberOfMoves` задает количество ходов, которое должен сделать каждый из игроков. Список `productions` объектов класса `ProductionGroup` хранит грамматику, задаваемую набором из 11 групп продукций, поле является закрытым, чтобы избежать его изменения. Получение доступа к полю осуществляется с помощью метода `GetProductions()`. Также есть свойство `ProductionsCount`, возвращающее количество групп продукций, в нашем случае значение всегда 11. Поле `RandomSettings`, являющееся экземпляром класса `RandomSettings`, задаёт распределение вероятностей выпадения групп продукций, в нашем случае оно всегда установлено соответственно таблице 3.1. Также имеются методы для записи конфигурации в формате .xml в файл и в поток — методы `WriteToFile()` и `WriteToStream()` соответственно. Также имеются и методы для считывания конфигурации в формате .xml из файла и из потока — методы `ReadFromFile()` и `ReadFromStream()` соответственно.

Теперь рассмотрим ранее упомянутый класс `RandomSettings`, который, как уже было сказано, задаёт распределение вероятностей выпадения продукций. Этот класс также является неизменяемым, в нашем случае он всегда задаёт распределение вероятностей, как в таблице 3.1. Вероятность хранится в виде обычной дроби: `totalPossibility` — знаменатель, доступ к которому предоставляется с помощью метода `getTotalPossibility()`, и список `possibilityList` — числитель для каждой группы продукций, доступ к которому предоставляется с помощью метода `getProductionPossibility()`.

Рассмотрим класс `ProductionGroup`. Этот класс задаёт группу продукций и является неизменяемым. Поле `Left` хранит нетерминал, находящийся в левой части продукций, а список `right` хранит строки, являющиеся правыми частями продукций, доступ к которому предоставляется с помощью методов `getRightAt()` и `getRights()`, а свойство `RightSize` возвращает количество

продукций в группе. Есть возможность записать класс в строковом формате и считать его из строки, за это отвечают методы `ToString()` и `fromString()` соответственно.

Класс `Bank` описывает банк продукций. Список `productionsBank` хранит количество каждой продукции, доступ осуществляется с помощью методов: `addProduction()` — для добавления группы продукций в банк, `removeProduction()` — для удаления группы продукций из банка, метод `getProductionCount()` возвращает количество групп продукций указанного типа в банке, метод `getProductions()` возвращает количество всех групп продукций в данке.

Рассмотрим классы `Move` и `PrimaryMove` — ход и простейший ход. `PrimaryMove` является неизменяемым классом и имеет 3 поля, доступных для чтения: `WordNumber` — номер вывода игрока, `ProductionGroupNumber` — номер группы продукции и `ProductionNumber` — номер продукции в группе. `Move` хранит список `moves` объектов класса `PrimaryMove`, предоставляет методы: `addMove()` — для добавления нового простейшего хода, `popMove()` — для удаления по правилам стека, и `getMoves()` — для чтения. Также реализует запись в строковый формат и чтение из строкового формата — методы `ToString()` и `FromString()`, с их помощью программы стратегий передают полученный ход.

Также перед рассмотрением кода стратегий рассмотрим класс `StrategyUtilitiesClass`, этот класс является статическим. Он предоставляет методы, которые являются общими для разных стратегий. Метод `isHaveLetter()` — возвращает индекс первого вхождения символа в выводе или -1, если символ не найден. Метод `findMatches()` реализует поиск множества A_{ij} слов, допускаемых продукцией, с тем лишь отличием, что тут нет элемента, соответствующего созданию нового слова. Также там присутствуют методы для применения ходов к списку имеющихся слов. В класс добавлены методы для вычисления всех трёх определённых нами метрик.

Также взглянем на классы `SimplifiedWord` и `SimplifiedProductionGroup`. Это упрощённый вывод и упрощённая группа продукций. Упрощение заключается в том, что от строкового представления мы переходим к представлению, где хранится количество терминалов и количество каждого нетерминала. Возникает это упрощение с целью оптимизации работы программы, поскольку работа со строками слишком трудоёмкая. Для определения вхождения символа в строку необходимо просмотреть её всю.

В классе `SimplifiedWord` есть поле `terminals`, соответствующее количеству терминалов, и массив `nonterminals`, соответствующий количеству нетерминалов. В нём также есть методы, предназначенные для удобного доступа к полям класса. Класс `SimplifiedProductionGroup` имеет поле `Left`, которое, как и в классе `ProductionGroup`, отвечает за нетерминал, находящийся в левой части

продукций. Также он имеет список rights объектов класса SimplifiedWord — правые части продукций в упрощённом виде. За конвертацию из обычного вида в упрощённый отвечает класс Simplifier, который по введённой грамматике строит прямое и обратное отображение из нетерминальных символов в целые числа, чтобы хранить количество нетерминалов в словах в виде массива.

5.2. Описание реализации стратегий

Каждая из стратегий является наследником абстрактного класса Strategy. Он предоставляет интерфейс для взаимодействия со стратегиями. Для оптимизации классы сделаны так, что один экземпляр класса можно использовать одновременно в нескольких партиях с одинаковыми правилами. Каждый ход делается независимо, классы не хранят никакие данные связанные с конкретной партией. Класс содержит 2 основных метода: метод makeMove, который надо реализовать, чтобы написать новую стратегию; метод setGameSettings(), который позволяет поменять правила игры. Также есть событие GameSettingsChanged, на которое можно добавить обработчик при необходимости проведения дополнительных действий при изменении правил игры.

Начнём со **случайной стратегии**. Она представлена классом RandomStrategy. В нём есть 5 интересующих нас методов: метод findFirstMove() отвечает за нахождение применения для выпавшей продукции, то есть выполнение шагов 1, 3, 4 и 5 алгоритма 1; метод findMove() отвечает за шаги 2, 3, 4 и 5 алгоритма 1, то есть за нахождение применения для продукций из банка; методы getGroupsWeights, getProductionsWeights, getWordsWeights отвечают за нахождение весов для случайного выбора. По умолчанию всему присваиваются веса равные единице. Именно эти методы будут переопределяться, чтобы получать разные вариации случайной стратегии.

Реализация **умной случайной стратегии**, её **улучшенной** и **инвертированной** версии реализовано переопределением упомянутых выше методов.

Теперь рассмотрим **глупую стратегию коротких слов**. Она представлена классом StupidShortWordsStrategy. Перед началом ходов для продукций вычисляется метрика 1, а также для каждой группы продукций находится лучшая продукция. Метод findFirstMove() отвечает за шаг 1 алгоритма. Метод findMove() отвечает за шаги 2 и 3 алгоритма.

Теперь рассмотрим **стратегию коротких слов**. Она представлена классом ShortWordsStrategy. Её реализация отличается от предыдущей лишь тем, что вычисляется метрика 2.

Теперь рассмотрим **переборную стратегию**. Она представлена классом SearchStrategy. При создании экземпляра класса можно передать параметр, определяющий глубину перебора. Метод searchMove реализует алгоритм 3. Метод findFirstMove реализует шаги 1—4 алгоритма 4, то есть нахождение лучшего

хода для выпавшей продукции. Метод `findMove` реализует шаг 5 этого алгоритма.

Смешанная стратегия представлена классом `MixedStrategy`, а **адаптивная случайная стратегия** представлена классом `AdaptiveRandom`. Реализация этих стратегий заключается в создании экземпляров других стратегий и передачи хода им.

Рассмотренный выше код представлен в приложении Б.

5.3. Описание интерфейса

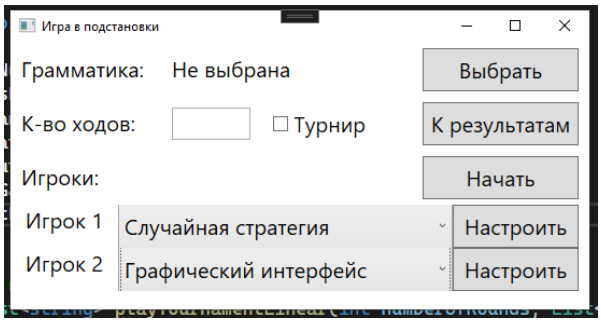

Рис. 1 — Скриншот основного окна

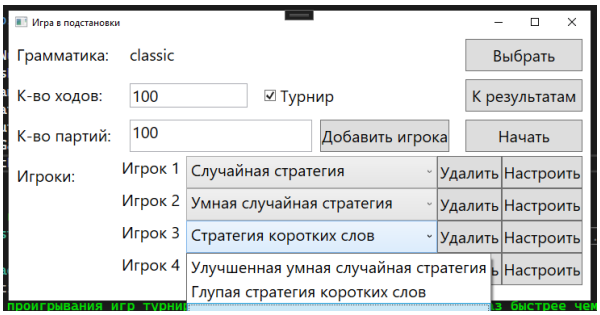

Рис. 2 — Скриншот основного окна. Выбрана опция турнир

Перед запуском игры надо выбрать грамматику, это можно сделать нажав на кнопку **«Выбрать»** в верхнем правом углу окна. После нажатия на неё появится окно, скриншот которого можно увидеть на рис. 3. В левой части окна выбора грамматики, представленном на рис. 3, можно увидеть уже имеющиеся в программе грамматики, а в правой части можно увидеть саму грамматику. Также можно создать свою грамматику, это делается нажатием на кнопку **«Создать»** в левой нижней части окна, после нажатия на которую откроется окно, представленное на рис. 4. Для выбора грамматики надо нажать на кнопку **«Выбрать»** в правой нижней части окна.

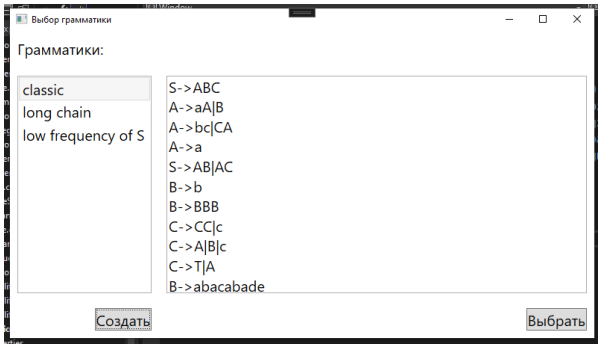

Рис. 3 — Скриншот окна выбора грамматики

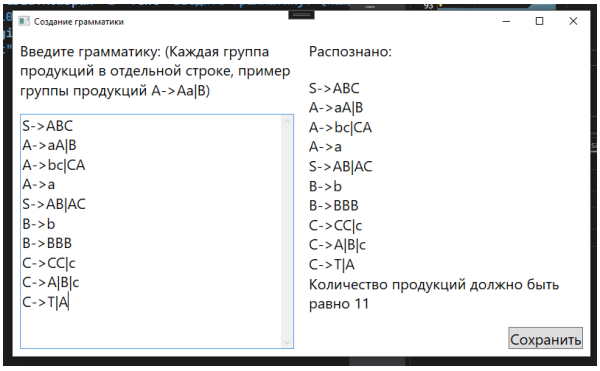

Рис. 4 — Скриншот окна создания грамматики

В окне создания грамматики, представленном на рис. 4, в левой части есть поле для ввода грамматики, вводить её следует так: каждая группа продукций

в отдельной строке, то есть разделитель — перенос строки. Каждая группа продукций записывается так: нетерминал, стрелка (->) и продукции, разделённые вертикальной чертой. Пример можно увидеть на рис. 4. После завершения ввода надо нажать кнопку **«Сохранить»** в правой нижней части экрана.

Также перед запуском игры необходимо указать то, сколько ходов делают игроки, для этого нужно в основном окне, представленном на рис. 1, ввести это число в поле напротив метки **«К-во ходов»**.

В программе есть 2 варианта запуска игры: турнир и одна партия. При запуске одной партии есть возможность в качестве игрока выбрать графический интерфейс и самим сыграть в игру против какой-либо стратегии. Режим запуска одной партии активен, когда галочка у опции **«Турнир»** не стоит, это можно увидеть на рис. 1.

Далее перед запуском партии необходимо выбрать игроков. Для этого надо кликнуть на элемент находящийся рядом с надписью **«Игрок [номер]:»**, появится выпадающее меню с вариантами на выбор, это меню можно увидеть на рис. 2.

Если же надо запустить турнир, то галочку у опции **«Турнир»** надо поставить. Для турнира необходимо указать, сколько партий будет сыграно каждой парой игроков, для этого нужно ввести это число в поле, рядом с меткой **«К-во партий»**. Далее надо выбрать игроков, для этого их сначала надо добавить, это делается нажатием на кнопку **«Добавить игрока»**, после этого надо выбрать игрока, делается это так же, как было описано выше. Перечисленное можно увидеть на рис. 2.

Для стратегий, у которых есть какой-либо параметр, есть возможность его указать, это производится в отдельном окне, которое можно увидеть на рис. 6. Открывается окно кликом по кнопке **«Настроить»**, эту кнопку можно увидеть на рис. 1 и рис. 2.

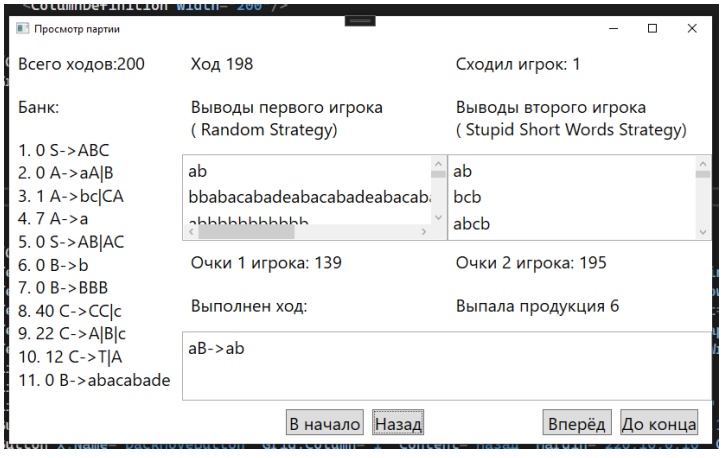

Рис. 5 — Скриншот окна просмотра партии

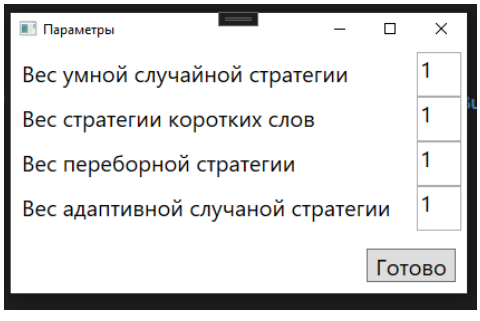

Рис. 6 — Скриншот окна параметров

Для запуска игры надо нажать на кнопку **«Начать»**. Если был выбран турнир, то начнётся разыгрывание партий между игроками, внизу окна будет отоб-

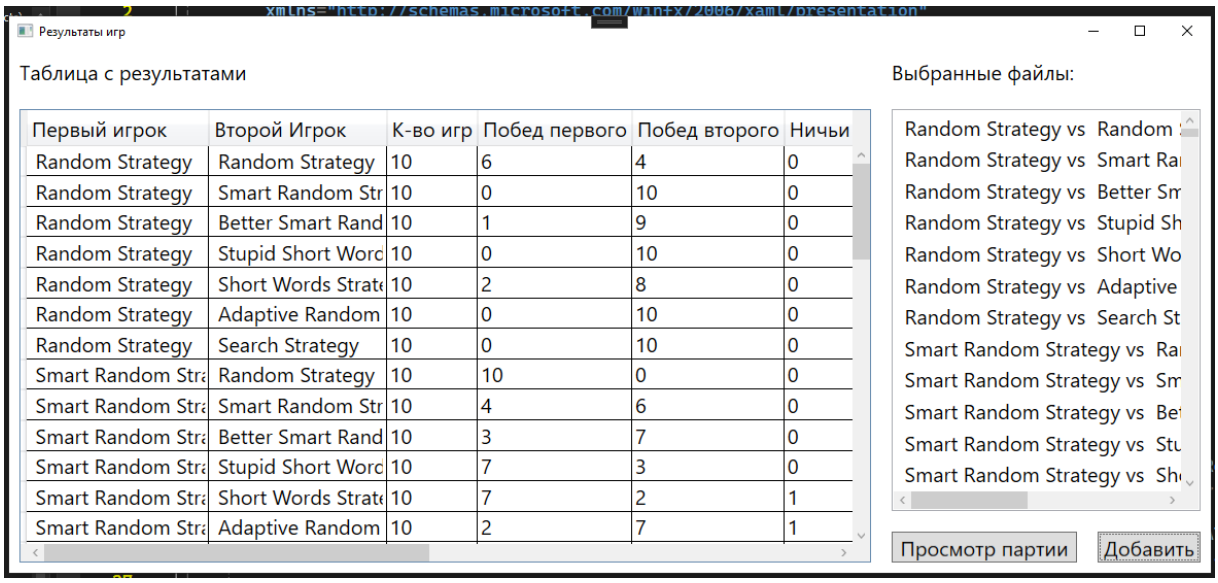

Рис. 7 — Скриншот окна результатов турнира

ражаться шкала прогресса. После завершения всех партий откроется новое окно с таблицей результатов, его можно увидеть на рис. 7. Просмотреть ход конкретной партии турнира можно, выбрав её в правой части и нажав на кнопку **«Просмотр партии»**, после нажатия на неё откроется окно которое можно увидеть на рис. 5.

Если же игра была запущена, и была выбрана одна партия, то сразу откроется окно просмотра партии. В окне просмотра партии есть кнопки **«Назад»**, **«Вперёд»**, **«В начало»** и **«До конца»** для проматывания ходов. В окне отображается вся информация об игре.

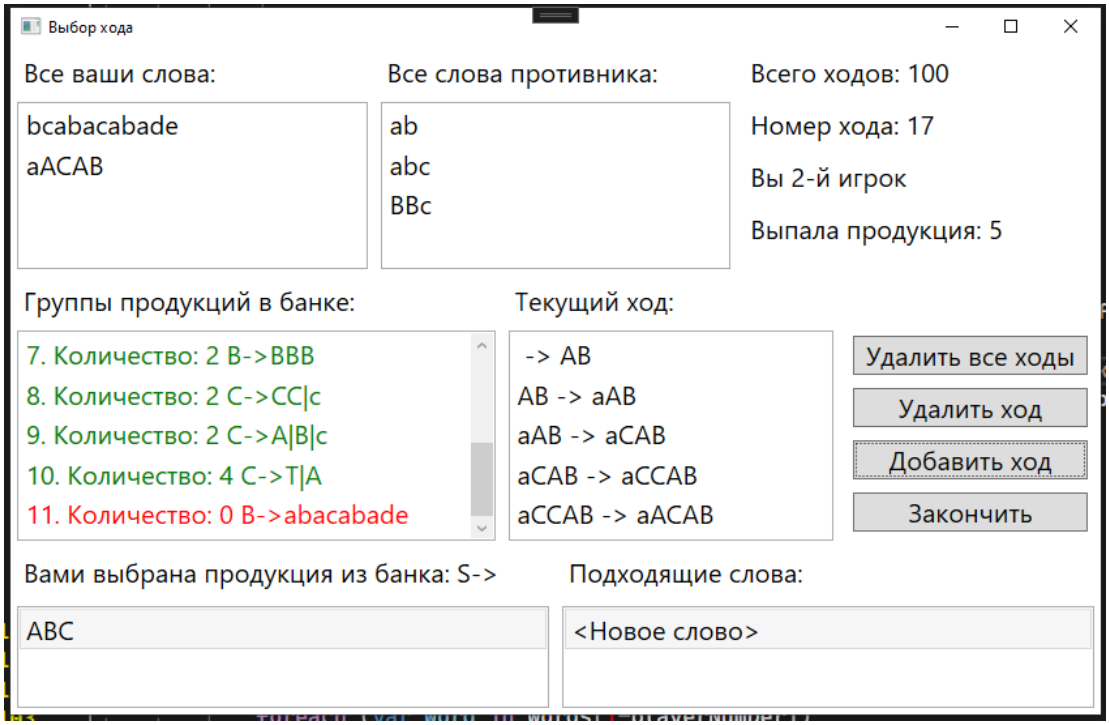

Рис. 8 — Скриншот окна ввода хода через графический интерфейс

Если выбрать в качестве игрока графический интерфейс, то для ввода хода откроется окно, которое можно увидеть на рис. 8. В правой верхней части окна можно увидеть основные параметры партии, также в верхней части окна можно увидеть имеющиеся выводы обоих игроков. В средней части окна слева находится банк продукций, в середине отображается текущий ход, а справа кнопки для управления. В нижней части окна можно увидеть элементы для добавления хода. Для добавления хода нужно в банке продукций выбрать желаемую группу продукций, далее в нижней левой части выбрать продукцию из группы продукций и в нижней правой части выбрать слов, к которому надо применить продукцию, после этого нажать на кнопку **«Добавить ход»**.

6. Сравнительный анализ стратегий

Теперь проведём сравнительный анализ стратегий. Имеет смысл рассматривать поведение стратегий на грамматиках разного вида. Для этого можно выделить некоторые классы грамматик. К примеру, можно делить по частоте выпадения начальных продукций (позволяющих начинать новые выводы). Поскольку большинство стратегий так или иначе конструировались таким образом, чтобы как можно быстрее получать сентенции, то при малой частоте выпадения начальных продукций эти стратегии будут получать малое количество очков.

Также можно выделить грамматики, имеющие бесполезные продукции, на них случайная стратегия будет работать плохо, а, при определённых условиях, и глупая стратегия коротких слов. Отдельно мы такие грамматики рассматривать не будем, так как плохая работа упомянутых стратегий на таких грамматиках очевидна, а эффективность других стратегий будет сильнее зависеть от частоты выпадения начальных продукций.

В качестве грамматики со средним шансом выпадения начальных продукций возьмём грамматику из постановки задачи. В качестве грамматики с низким шансом выпадения начальных продукций возьмём ту же грамматику но немного её изменим: заменим нетерминал в левой части пятой группы продукций с S на A.

Проведение турнира с использованием грамматики с высоким шансом выпадения начальных продукций не является интересным. Поскольку большинство стратегий строилось для построения коротких выводов, то результаты будут примерно одинаковыми, а наилучшей будет переборная стратегия. Задача построения выводов кратчайшим способом во многом уже решена и не является интересной. Намного более интересной задачей является построение более длинных выводов.

Для сравнения стратегий будет проводиться круговой турнир из 500 партий с 200 шагов в каждой. Результаты кругового турнира с использованием грамматики из постановки задачи между всеми стратегиями можно увидеть в таблице 6.1. В первом столбце указаны стратегии игрока, ходившего первым, а в первой строке — вторым. В ячейках до перовой наклонной черты указано количество побед первого игрока, после — второго, а после второй наклонной черты указано количество ничьих.

В таблице применены сокращения названий стратегий до первых букв слов в названии, а число, идущее после, соответствует значению параметра. Перечислим используемые сокращения: СС — случайная стратегия, УСС — умная случайная стратегия, УУСС — улучшенная умная случайная стратегия, ГСКС — глупая стратегия коротких слов, СКС — стратегия коротких слов, ПС — переборная

Таблица 6.1

Распределение побед в турнире с использованием грамматики с средним шансом выпадения начальных продукций

	СС	УСС	УУСС	ГСКС	СКС	ПС1	ПС4	ААС10
СС	238/256/6	36/464/0	35/461/4	64/432/4	50/449/1	28/471/1	12/488/0	14/485/1
УСС	461/37/2	258/240/2	243/250/7	313/180/7	374/125/1	167/330/3	91/407/2	105/390/5
УУСС	453/46/1	229/267/4	253/243/4	282/215/3	342/154/4	163/332/5	109/389/2	105/390/5
ГСКС	430/69/1	180/319/1	207/290/3	259/238/3	262/234/4	136/357/7	83/415/2	72/426/2
СКС	454/45/1	102/396/2	168/328/4	227/271/2	241/251/8	62/436/2	38/462/0	149/347/4
ПС1	479/21/0	343/156/1	321/176/3	362/134/4	431/68/1	251/243/6	149/349/2	160/336/4
ПС4	486/14/0	415/85/0	403/95/2	427/72/1	463/36/1	332/166/2	273/223/4	247/246/7
ААС10	485/14/1	374/125/1	390/106/4	423/75/2	358/140/2	312/185/3	252/246/2	256/241/3

стратегия, АСС — адаптивная случайная стратегия. Сейчас рассмотрим результаты всех стратегий, исключая адаптивную случайную стратегию.

Случайная стратегия показывает себя плохо в сравнении со всеми остальными стратегиями. Её улучшенные варианты показывают себя значительно лучше, но всё так же проигрывают переборной стратегии.

Глупая стратегия коротких слов неожиданно показывает себя лучше стратегии коротких слов. Это вызвано тем, что при равенстве метрик, то есть при равенстве количества нетерминалов в продукциях, она считает лучшей ту, где больше терминальных символов, так она набирает больше очков. Также, как было сказано в описании стратегии, метрика не выполняет свою изначальную задумку и не позволяет строить кратчайшие выводы. Но стоит отметить, что в грамматику легко добавить ловушку для данной стратегии, которая совершенно не повлияет на поведение других стратегий. К примеру, можно добавить в первую и пятую группы продукций бесполезные продукции. Тогда получим следующее группы продукций: $S \to ABC \mid T$ и $S \to AB \mid AC \mid T$. Таким образом, при создании нового слова глупая стратегия коротки слов всегда будет выбирать бесполезную продукцию. Результаты проведения игр с использованием новой грамматики можно увидеть в таблице 6.2. Как видно, стратегия ни разу не победила.

Таблица 6.2

Распределение побед в играх с грамматикой, сконструированной специально против ГСКС

Противник	ГСКС	СС	УСС	УУСС	СКС	ПС1	ПС4	АСС10
ГСКС первый игрок	0/0	0/20	0/20	0/20	0/20	0/20	0/20	0/20
ГСКС второй игрок	0/0	20/0	20/0	20/0	20/0	20/0	20/0	20/0

Стратегия коротких слов показывает себя плохо из-за того, что она слишком ориентирована на получение выводов с наименьшим количеством шагов.

Переборная стратегия в обеих своих вариациях показывает себя хорошо. Хотя ПС1 ведёт перебор всего на один шаг вперёд, но этого уже достаточно чтобы обойти все остальные стратегии. Тем не менее, ПС4 значительно увеличивает отрыв от других стратегий. Так, ПС4 можно назвать строго доминирующей стратегией.

Также можно отметить, что очерёдность хода игроков почти не влияет на результат. А имеющаяся разница может быть объяснена влиянием случайности.

Теперь поговорим об адаптивной случайной стратегии. Отметим, что у переборной стратегии результативность прямо пропорциональна основному параметру — глубине перебора. У адаптивной случайной стратегии связь результативности с параметром Ω устроена сложнее. При слишком низком значении параметра Ω стратегия будет слишком поздно переходить ко второму этапу и будет не успевать заканчивать выводы. При большом значении параметра стратегия будет слишком рано переходить на второй этап и будет вести себя как переборная стратегия.

Для нахождения оптимального значения параметра отвечающего за смену поведения, проведём исследование. Переберём значения параметра на отрезке от 1 до 20 и будем смотреть на количество побед в играх против переборной стратегии. Переборная стратегия выбрана в качестве оппонента как самая сильная из остальных представленных стратегий. Также сразу рассмотрим эффективность применения стратегии коротких слов на втором этапе. Результаты исследования можно увидеть на Рис. 9. При каждом значении параметра проводилось 500 партий с использованием грамматики с низким шансом выпадения начальных продукций. На графике сплошной линией обозначена доля побед в играх с использованием ПС на втором этапе, пунктирной линией — с использованием СКС на втором этапе.

По результатам можно сказать, что при увеличении параметра до 10 ре-

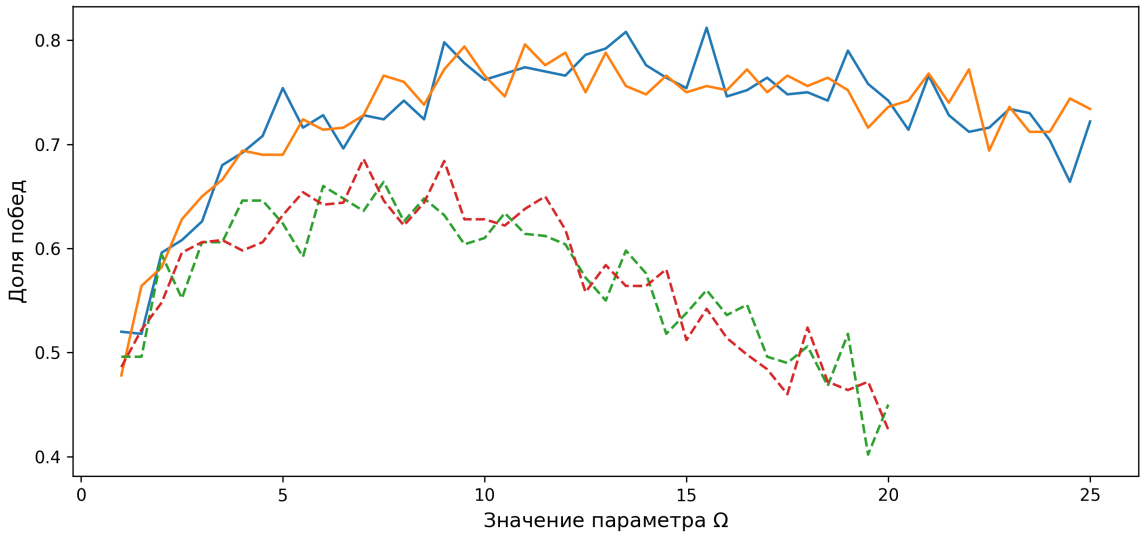

Рис. 9 — Доля побед ААС в зависимости от параметра Ω

зультативность растёт. При использовании ПС на втором этапе рост результативности останавливается при значениях параметра больше 10, а при значениях больше 20 результативность начинает падать. При использовании СКС на втором этапе результативность начинает падать уже после 10. Таким образом, 10 является оптимальным значением параметра. Также использование ПС на втором этапе является более предпочтительным.

Теперь взглянем на результаты этой стратегии, приведённые в таблице 6.1. Стратегия показывает себя на уровне ПС4. Но на данной грамматике это вполне ожидаемый и хороший результат.

Таблица 6.3

Распределение побед в турнире с использованием грамматики с низким шансом выпадения начальных продукций

	СС	УСС	УУСС	ГСКС	СКС	ПС1	ПС4	ААС10
СС	250/238/12	219/275/6	199/297/4	185/313/2	268/232/0	185/306/9	153/341/6	91/405/4
УСС	270/227/3	246/250/4	166/332/2	97/398/5	375/116/9	172/327/1	118/380/2	74/423/3
УУСС	305/192/3	305/186/9	249/244/7	181/318/1	380/117/3	259/238/3	198/300/2	100/399/1
ГСКС	349/145/6	384/115/1	321/177/2	265/233/2	435/61/4	357/140/3	314/184/2	120/378/2
СКС	228/269/3	141/349/10	90/408/2	75/423/2	233/227/40	62/436/2	30/468/2	55/443/2
ПС1	313/183/4	300/196/4	233/266/1	142/354/4	430/68/2	261/230/9	202/294/4	64/435/1
ПС4	352/141/7	378/116/6	286/209/5	200/298/2	460/39/1	333/164/3	244/244/12	89/410/1
ААС10	406/93/1	420/79/1	397/102/1	374/122/4	446/53/1	434/65/1	396/103/1	228/270/2

Теперь взглянем на результат в таблице 6.3 проведения турнира с использованием грамматики с низким шансом выпадения начальных продукций.

Выделяется явный фаворит в лице адаптивной случайной стратегии. Она даже подходит на место строго доминирующей стратегии.

Также выделяются плохие показатели стратегии коротких слов. Она даже в партиях со случайной стратегий чаще проигрывает, чем побеждает. Причиной этого является та же проблема слишком быстрого получения сентенций.

Снова довольно хорошо показывает себя глупая стратегия коротких слов, это происходит всё потому же, из-за недостатка метрики 1, который выражается в построении не самых коротких выводов. Но в такой ситуации этот недостаток скорее является преимуществом. Важно подчеркнуть что можно сконструировать грамматику специально против этой стратегии, что уже было проделано ранее.

Можно отметить уменьшение отрыва переборной стратегии от всех остальных. Это показывает ухудшение эффективности данной стратегии на грамматиках такого вида, что вполне объяснимо, ведь эта стратегия так же, как и другие стремится строить кратчайшие выводы. Тем не менее, можно подчеркнуть, что при большой (или неограниченной) глубине перебора, эта стратегия смогла бы показывать намного лучшие результаты, но из-за ограничений времени это невозможно.

Теперь исследуем смешанные стратегии. В формировании смешанных стра-

тегий будут участвовать не все чистые стратегии. Будут взяты лишь лучшие представители: улучшенная умная случайная стратегия, стратегия коротких слов, переборная стратегия и адаптивная случайная стратегия.

Для упрощения записи будем сокращать название смешанной стратегии до СмС (поскольку СС уже занято случайной стратегией) и перечисления весов. К примеру смешанная стратегия с очками вероятности равными единице будет записана как СмС 1 1 1 1.

Получим распределения очков вероятности, используя метод «Процент от побед». По результатам из таблицы 6.1: УУСС $= 1581$, СКС $= 1177$, ПС $= 2706$, ААС $= 2263$, такую смешанную стратегию сократим до ПП1. А по результатам из таблицы 6.3: УУСС $= 1890$, СКС $= 844$, ПС $= 2194$, ААС $= 2746$, такую смешанную стратегию сократим до ПП2.

В соответствии с правилом «Линейный рост» рассмотрим 4 вариации раздачи очков вероятности: Смс 2 1 3 4, СмС 1 0 2 3, СмС 2 1 4 3, СмС 1 0 3 2.

В соответствии с равилом «Пики» рассмотрим такие вариации раздачи очков вероятности: СмС 1 1 1 5, СмС 1 1 5 1, СмС 1 1 1 10, СмС 1 1 10 1.

Сначала проанализируем результаты турнира с использованием грамматики со средним шансом выпадения начальных продукций, которе приведена в таблице А.1. Все рассмотренные смешанные стратегии имеют схожие результаты. Все они проигрывают переборной стратегии или играют с ней наравне. И даже в играх смешанных стратегий со смешанными стратегиями нет победителей. С другими стратегиями они показывали результаты хуже переборной стратегии или на уровне переборной стратегии.

Далее рассмотрим результаты турнира с низким шансом выпадения начальных продукций, которые приведены в таблице А.2. Все стратегии проигрывают адаптивной случайной стратегии. В партиях среди смешанных стратегий лучше всего себя показывают варианты СмС 1 1 1 10 и СмС 1 1 1 5. И в целом можно отметить зависимость: лучше себя показывают стратегии, у которых больше очки вероятности, назначенные адаптивной случайной стратегии. Это показывает, что на данной грамматике наиболее эффективной является адаптивная случайная стратегия.

По результатам рассмотрения смешанных стратегий, основанных на вероятностном выборе, можно сказать, что они не являются эффективными. Это вполне объяснимо, поскольку у нас имеется почти строго доминирующая стратегия. При смешивании такой стратегии с другими получается лишь ослабленная вариация изначально хорошей стратегии. Также это может быть связано с тем, что все стратегии, кроме адаптивной случайной стратегии, направлены на получение коротких выводов и при их смешении не получается строить более длинные выводы.

Заключение

В ходе работы получилось разработать эффективные стратегии для игры в подстановки для разных типов грамматик.

Адаптивная случайная стратегия уверенно показывает себя в играх с грамматиками с низким шансом выпадения начальных продукций. И показывает себя не хуже других в играх с грамматиками со средним шансом выпадения продукций. И может быть рекомендована как самая выгодная стратегия.

Смешанные стратегии, основанные на вероятностном смешении, оказались неэффективными. Это объясняется наличием строго доминирующей стратегии. В данной ситуации повышение эффективности должно достигаться не смешением имеющихся стратегий, а качественным улучшением поведения стратегии, что и получилось сделать в адаптивной случайной стратегии.

В будущих исследованиях можно рассмотреть вопрос создания новых стратегий. Также можно рассмотреть варианты модернизации алгоритма переборной стратегии для оптимизации. А также рассмотреть вариации «Игры в подстановки» с другими правилами.

Список литературы

[1] Соколов В. А. Введение в теорию формальных языков. Ярославль: ЯрГУ, 2014.

[2] Захаров А. В. Теория игр в общественных науках. М.: ВШЭ, 2015.

[3] Эрудит (игра) [Электронный ресурс] URL: https://dic.academic.ru/dic.nsf/ruwiki/1213567 (дата обращения: 20.05.2023)

[4] Goldwater D. Number Scrabble — the Game (aka: Math Scrabble) [Электронный ресурс] URL: https://www.instructables.com/Number-Scrabble-The-Game-aka-Math-Scrabble/ (дата обращения: 22.05.2023)

[5] Cosse C. Tux Math Scrabble [Электронный ресурс] URL: https://sourceforge.net/projects/tuxmathscrabble/ (дата обращения: 20.05.2023)

[6] Schuster M. Context-free games on strings and nested words: Dissertation // Technischen Universität Dortmund an der Fakultät für Informatik, 2017. URl: http://dx.doi.org/10.17877/DE290R-18135.

[7] Кругликов А. М., Смирнов А. В. Разработка стратегий для игры «Укроти мустанга» // Путь в науку: прикладная математика, информатика и информационные технологии : Тезисы докладов Всероссийской молодёжной научно-практической конференции, Ярославль, 15–19 апреля 2024 года. – Ярославль: ЯРГУ, 2024. – С. 28-31.

Приложение А

Таблицы турниров смешанных стратегий

Таблица А.1

Таблица турнира смешанных стратегий с грамматикой с средним шансом выпадения начальных продукций

	ПП1	ПП2	СмС 1 1 1 1	СмС 2 1 3 4	СмС 1 0 2 3	СмС 2 1 4 3	СмС 1 0 3 2	СмС 1 1 1 5	СмС 1 1 5 1	СмС 1 1 1 10	СмС 1 1 10 1	ПС4	ACC10	УУСС	СКС
ПП1	50/49/1	43/57/0	51/49/0	49/50/1	53/47/0	42/56/2	48/52/0	43/56/1	46/54/0	51/49/0	44/56/0	41/58/1	44/55/1	73/27/0	81/19/0
ПП2	51/49/0	54/45/1	64/36/0	42/57/1	49/50/1	48/51/1	52/48/0	48/51/1	49/50/1	39/59/2	45/55/0	41/58/1	40/59/1	84/16/0	76/24/0
СмС 1 1 1 1	44/55/1	39/61/0	60/39/1	46/53/1	48/51/1	42/58/0	38/62/0	38/61/1	41/59/0	48/52/0	43/57/0	33/64/3	41/59/0	85/15/0	80/20/0
СмС 2 1 3 4	50/48/2	52/46/2	46/54/0	46/54/0	48/50/2	50/49/1	50/50/0	48/52/0	50/48/2	53/46/1	36/64/0	41/59/0	45/54/1	70/30/0	73/26/1
СмС 1 0 2 3	58/42/0	55/43/2	53/45/2	43/56/1	52/46/2	54/45/1	54/45/1	49/51/0	49/50/1	55/44/1	48/52/0	46/53/1	48/52/0	80/20/0	81/19/0
СмС 2 1 4 3	33/67/0	57/42/1	50/49/1	44/54/2	44/54/2	42/55/3	45/54/1	53/47/0	39/61/0	51/47/2	53/46/1	40/60/0	49/50/1	75/23/2	82/17/1
СмС 1 0 3 2	53/46/1	47/52/1	63/36/1	58/42/0	52/46/2	60/40/0	53/45/2	53/45/2	59/41/0	52/48/0	62/38/0	46/54/0	58/42/0	76/23/1	79/19/2
СмС 1 1 1 5	46/54/0	50/49/1	52/48/0	52/47/1	51/47/2	54/46/0	49/51/0	50/49/1	52/46/2	50/50/0	41/58/1	49/51/0	37/63/0	82/18/0	74/26/0
СмС 1 1 5 1	48/51/1	59/41/0	63/36/1	45/54/1	60/40/0	53/46/1	45/54/1	58/40/2	48/51/1	48/51/1	42/58/0	44/56/0	57/42/1	73/27/0	92/8/0
СмС 1 1 1 10	46/54/0	54/45/1	47/52/1	47/53/0	51/47/2	50/49/1	43/53/4	54/46/0	45/54/1	45/55/0	48/51/1	34/63/3	57/42/1	84/16/0	65/35/0
СмС 1 1 10 1	49/50/1	60/38/2	57/42/1	60/39/1	56/44/0	55/44/1	54/46/0	50/50/0	42/58/0	44/55/1	50/48/2	46/54/0	43/57/0	82/18/0	89/11/0
ПС4	57/42/1	55/43/2	60/39/1	53/47/0	51/49/0	52/48/0	49/50/1	51/46/3	58/42/0	56/44/0	56/44/0	X	X	X	X
ACC10	54/46/0	62/37/1	54/46/0	54/45/1	48/51/1	53/46/1	53/47/0	56/44/0	49/49/2	48/52/0	50/49/1	X	X	X	X
УУСС	19/81/0	21/79/0	29/69/2	27/72/1	26/73/1	13/86/1	14/84/2	19/81/0	19/81/0	21/79/0	19/81/0	X	X	X	X
СКС	21/78/1	23/76/1	18/81/1	14/85/1	24/75/1	21/79/0	13/87/0	21/78/1	12/88/0	24/74/2	11/87/2	X	X	X	X

Таблица A.2

Таблица турнира смешанных стратегий с грамматикой с средним шансом выпадения начальных продукций

	ПП1	ПП2	СмС 1 1 1 1	СмС 2 1 3 4	СмС 1 0 2 3	СмС 2 1 4 3	СмС 1 0 3 2	СмС 1 1 1 5	СмС 1 1 5 1	СмС 1 1 1 10	СмС 1 1 10 1	ПС4	АСС10	УУСС	СКС
ПП1	46/53/1	44/56/0	60/37/3	36/62/2	43/57/0	47/53/0	43/57/0	35/65/0	39/59/2	28/72/0	49/50/1	51/48/1	21/78/1	66/34/0	88/11/1
ПП2	55/44/1	53/46/1	72/28/0	45/53/2	45/54/1	53/47/0	44/56/0	45/55/0	61/38/1	33/67/0	47/53/0	54/45/1	26/74/0	60/39/1	90/10/0
СмС 1 1 1 1	37/62/1	39/60/1	55/45/0	38/62/0	30/69/1	37/62/1	45/55/0	45/55/0	44/55/1	31/69/0	52/47/1	42/57/1	16/83/1	61/38/1	86/14/0
СмС 2 1 3 4	64/36/0	52/48/0	60/40/0	52/48/0	38/61/1	55/45/0	50/50/0	35/65/0	58/41/1	32/68/0	62/37/1	59/41/0	29/71/0	66/33/1	91/8/1
СмС 1 0 2 3	57/42/1	54/46/0	71/28/1	55/44/1	47/53/0	63/37/0	59/40/1	52/47/1	71/29/0	35/63/2	63/37/0	57/41/2	37/63/0	68/32/0	98/2/0
СмС 2 1 4 3	53/46/1	48/51/1	60/40/0	38/61/1	35/64/1	48/52/0	43/57/0	41/59/0	51/47/2	33/67/0	58/41/1	50/48/2	20/80/0	69/30/1	92/8/0
СмС 1 0 3 2	54/45/1	55/45/0	62/38/0	53/47/0	43/55/2	49/50/1	47/53/0	54/46/0	51/47/2	36/64/0	61/39/0	60/40/0	31/69/0	66/33/1	93/7/0
СмС 1 1 1 5	62/38/0	60/40/0	70/27/3	45/54/1	46/53/1	60/40/0	53/47/0	52/48/0	62/38/0	45/54/1	58/42/0	67/33/0	35/64/1	73/26/1	87/13/0
СмС 1 1 5 1	52/48/0	41/58/1	55/45/0	41/58/1	35/64/1	33/67/0	46/53/1	34/65/1	42/57/1	27/72/1	57/42/1	42/58/0	31/68/1	62/38/0	89/11/0
СмС 1 1 1 10	66/34/0	54/45/1	73/27/0	63/37/0	50/50/0	66/34/0	63/37/0	65/35/0	75/25/0	49/51/0	74/26/0	71/29/0	43/55/2	78/22/0	86/14/0
СмС 1 1 10 1	50/49/1	37/63/0	55/45/0	45/55/0	31/67/2	41/59/0	42/56/2	47/53/0	49/51/0	30/70/0	44/53/3	48/52/0	21/79/0	63/36/1	89/11/0
ПС4	51/48/1	47/53/0	57/42/1	44/55/1	37/63/0	51/49/0	54/45/1	36/62/2	48/51/1	31/68/1	48/52/0	X	X	X	X
АСС10	75/24/1	73/27/0	74/26/0	75/23/2	73/27/0	69/30/1	75/25/0	71/28/1	76/24/0	55/44/1	74/25/1	X	X	X	X
УУСС	44/55/1	35/65/0	42/56/2	31/69/0	33/67/0	42/57/1	33/66/1	39/60/1	35/65/0	22/78/0	38/61/1	X	X	X	X
СКС	12/87/1	11/88/1	19/81/0	4/96/0	12/88/0	7/93/0	10/89/1	11/89/0	15/85/0	6/93/1	7/93/0	X	X	X	X

Приложение Б

Исходный код программы на C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Remoting;
using System.Runtime.Serialization;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ProductionsGameCore
{
   public class ProductionGroup
   {
      public char Left
      {
         get;
         private set;
      }
      private List<string> right = new List<string>();

      public ProductionGroup(char left, List<string> right)
      {
         if (!(left >= 'A' && left <= 'Z'))
            throw new ArgumentException("Letter in left part of prosuction must be capital english
      letter.");
         this.Left = left;
         if (right.Count == 0)
            throw new ArgumentException("In each production group must at least one production."
      );
         right.ForEach(x => this.right.Add(x));
      }

      public ProductionGroup(char left, params string[] right)
      {
         this.Left = left;
         foreach (string tmp in right)
            this.right.Add(tmp);
      }

      public int RightSize
      {
         get => right.Count;
      }

      public string getRightAt(int index)
      {
         if (index >= 0 && index < RightSize)
            return right[index];
         throw new IndexOutOfRangeException(
            String.Format("Index {0} was outside of [0,{1}).", index, RightSize)
            );
```

```
    }

    public IEnumerable<string> getRights()
    {
      return right.AsEnumerable();
    }

    public override string ToString()
    {
      StringBuilder sb = new StringBuilder();
      sb.Append(Left + "->");
      if (right.Count != 0)
        sb.Append(right[0].ToString());
      for (int index = 1; index < right.Count; ++index)
      {
        sb.Append("|" + right[index].ToString());
      }
      return sb.ToString();
    }

    public static ProductionGroup fromString(string s)
    {
      StringBuilder sb = new StringBuilder();
      char? left = null;
      List<string> right = new List<string>();
      for (int i = 0; i < s.Length; ++i)
      {
        if (s[i] == '|')
        {
          if (left == null)
            throw new ArgumentException("Left part of production is not specified.");
          right.Add(sb.ToString());
          sb.Clear();
          continue;
        }
        if (s[i] == '-' && i + 1 < s.Length && s[i + 1] == '>')//found ->
        {
          if (left == null)
            if (sb.Length == 1 && sb[0] >= 'A' && sb[0] <= 'Z')//check that Neterminal in left
            {
              left = sb[0];
              sb.Clear();
              i++;//skip >
              continue;
            }
            else
              throw new ArgumentException("Left part of production must be capital engliah
    letter.");
        }
        sb.Append(s[i]);
      }
      right.Add(sb.ToString());
      if (left == null)
        throw new ArgumentException("Left part of production is not specified.");
      return new ProductionGroup(left.Value, right);
    }
  }
}
```

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProductionsGameCore
{
   public class Grammatic
   {
      public string Name { get; private set; }
      private List<ProductionGroup> productions;

      public Grammatic(string name,IEnumerable<string> productions) {
         Name = name;
         this.productions = new List<ProductionGroup>();
         foreach (var prod in productions) {
            this.productions.Add(ProductionGroup.fromString(prod));
         }
      }

      public IEnumerable<ProductionGroup> getProductions() {
         return this.productions;
      }

      public override string ToString()
      {
         StringBuilder sb = new StringBuilder();
         foreach (var prod in productions)
            sb.AppendLine(prod.ToString());
         return sb.ToString();
      }
   }
}

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Xml;
using System.Xml.Linq;
using System.Xml.Serialization;

namespace ProductionsGameCore
{
   public class GameSettings
   {
      private List<ProductionGroup> productions = new List<ProductionGroup>();
      private List<SimplifiedProductionGroup> simplifiedProductions;
      public Simplifier Simplifier { get; private set; }
      public RandomSettings RandomSettings { get; private set; }
      public int NumberOfMoves { get; private set; }
      public int ProductionsCount { get { return productions.Count; } }
```

```
public GameSettings(int numberOfMoves,
   IEnumerable<ProductionGroup> productions,
   RandomSettings randomSettings)
{
   if (numberOfMoves <= 0)
      throw new ArgumentException("Number of moves must be non−nagative number.");
   NumberOfMoves = numberOfMoves;
   this.productions = productions.ToList();
   RandomSettings = randomSettings;
   Simplifier = new Simplifier(this.productions);
   simplifiedProductions = Simplifier.ConvertProductions(this.productions);

}

public GameSettings(int numberOfMoves,
   Grammatic grammatic,
   RandomSettings randomSettings)
   :this(numberOfMoves,grammatic.getProductions(),randomSettings)
{
}

public ProductionGroup getProductionGroup(int index)
{
   if (index >= 0 && index < ProductionsCount)
      return productions[index];
   throw new IndexOutOfRangeException(
      String.Format("Index {0} was outside of [0,{1}).", index, ProductionsCount)
      );
}

public SimplifiedProductionGroup getSimplifiedProductionGroup(int index)
{
   if (index >= 0 && index < ProductionsCount)
      return simplifiedProductions[index];
   throw new IndexOutOfRangeException(
      String.Format("Index {0} was outside of [0,{1}).", index, ProductionsCount)
      );
}

public IEnumerable<ProductionGroup> GetProductions()
{
   return productions.AsEnumerable();
}

public IEnumerable<SimplifiedProductionGroup> GetSimplifiedProductions()
{
   return simplifiedProductions.AsEnumerable();
}

public static GameSettings ReadFromFile(string filename)
{
   using (FileStream fs = new FileStream(filename, FileMode.Open))
   {
      return ReadFromStream(fs);
   }
}

public static GameSettings ReadFromStream(Stream stream)
```

```
{
   XElement XGameSettings = null;
   try
   {
      XGameSettings = XElement.Load(stream);
   }
   catch
   {
      throw new IOException("Input in wrong format.");
   }
   int numberOfMoves = int.Parse(XGameSettings.Attribute("NumberOfMoves").Value);

   XElement XProductions = XGameSettings.Element("Productions");
   List<ProductionGroup> productions = new List<ProductionGroup>();
   foreach (var XProduction in XProductions.Elements())
   {
      char left = XProduction.Attribute("Left").Value[0];
      List<string> rights = new List<string>();
      foreach (var XRight in XProduction.Elements())
         rights.Add(XRight.Value);
      productions.Add(new ProductionGroup(left, rights));
   }
   XElement XRandomSettings = XGameSettings.Element("RandomSettings");
   int totalPossibility = int.Parse(XRandomSettings.Attribute("TotalPossibility").Value);
   List<int> possibilities = new List<int>();
   foreach (var XPossibility in XRandomSettings.Element("Possibilities").Elements())
      possibilities.Add(int.Parse(XPossibility.Value));
   RandomSettings randomSettings;
   randomSettings = new RandomSettings(totalPossibility, possibilities);
   return new GameSettings(numberOfMoves, productions, randomSettings);
}

public void WriteToFile(string filename)
{
   var currentDirectory = Directory.GetCurrentDirectory();
   var purchaseOrderFilepath = Path.Combine(currentDirectory, filename);
   using (StreamWriter fs = new StreamWriter(purchaseOrderFilepath))
   {
      WriteToStream(fs);
   }
}

public void WriteToStream(StreamWriter stream)
{
   XElement XEGameSettings = new XElement("GameSettings");
   XEGameSettings.Add(new XAttribute("NumberOfMoves", NumberOfMoves));
   {// add productions
      XElement XProductions = new XElement("Productions");
      foreach (var production in productions)
      {
         XElement Xprod = new XElement("Production");
         Xprod.Add(new XAttribute("Left", production.Left));

         for (int i = 0; i < production.RightSize; ++i)
            Xprod.Add(new XElement("Right", production.getRightAt(i)));
         XProductions.Add(Xprod);
      }
      XEGameSettings.Add(XProductions);
```

```
            }
            {//add random settings
                XElement XRandomSettings = new XElement("RandomSettings");
                XRandomSettings.Add(new XAttribute("TotalPossibility", RandomSettings.
        getTotalPossibility()));
                XElement XPossibilities = new XElement("Possibilities");
                for (int i = 0; i < ProductionsCount; ++i)
                    XPossibilities.Add(new XElement("Possibility", RandomSettings.getProductionPossibility(
        i)));
                XRandomSettings.Add(XPossibilities);
                XEGameSettings.Add(XRandomSettings);
            }
            {//write
                var settings = new XmlWriterSettings();
                settings.OmitXmlDeclaration = true;
                settings.Indent = true;
                using (XmlWriter xmlWriter = XmlWriter.Create(stream, settings))
                {
                    XEGameSettings.WriteTo(xmlWriter);
                }
            }
        }

        /// <summary>
        /// format :
        /// [numberOfMoves]
        /// [numberOfProdustion]
        /// produstions in next [numberOFProdution] lines 1 production in one line
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();
            sb.AppendLine(NumberOfMoves.ToString());
            sb.AppendLine(ProductionsCount.ToString());
            foreach (ProductionGroup pr in productions)
            {
                sb.AppendLine(pr.ToString());
            }
            return sb.ToString();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ProductionsGameCore
{
    public class Bank
    {
        List<int> productionsBank;

        public Bank(int productionsNumber)
        {
            productionsBank = new List<int>(productionsNumber);
            for (int i = 0; i < productionsNumber; ++i)
```

```
            productionsBank.Add(0);
        }

        public Bank(IEnumerable<int> productions)
        {
            productionsBank = productions.ToList();
        }

        public void addProduction(int productionIndex)
        {
            addProduction(productionIndex, 1);
        }

        public void addProduction(int productionIndex, int count)
        {
            productionsBank[productionIndex] += count;
        }

        public void removeProduction(int productionIndex)
        {
            removeProduction(productionIndex, 1);
        }

        public void removeProduction(int productionIndex, int count)
        {
            if (productionsBank[productionIndex] < count)
                throw new ArgumentException("The number of profuctions of index " + productionIndex +
         " in bank was less than " + count + ".");
            productionsBank[productionIndex] −= count;
        }

        public IEnumerable<int> getProductions()
        {
            return productionsBank.AsEnumerable();
        }

        public int getProductionCount(int productionIndex)
        {
            return productionsBank[productionIndex];
        }

        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();
            foreach (var item in productionsBank)
            {
                sb.Append(item + " ");
            }
            return sb.ToString();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ProductionsGameCore
{
   public class Move
   {
      private List<PrimaryMove> moves;
      public Move()
      {
         moves = new List<PrimaryMove>();
      }

      public int MovesCount { get { return moves.Count; } }

      public void addMove(int wordNumber, int groupNumber, int productionNumber)
      {
         moves.Add(new PrimaryMove(wordNumber, groupNumber, productionNumber));
      }

      public void addMove(PrimaryMove move)
      {
         moves.Add(move);
      }

      public void addMove(Move move)
      {
         moves.AddRange(move.getMoves());
      }

      public void popMove() {
         moves.RemoveAt(moves.Count − 1);
      }

      public IEnumerable<PrimaryMove> getMoves()
      {
         return moves.AsEnumerable();
      }

      public static Move FromString(string move)
      {
         if(move == null || move == "")
            return new Move();
         Move moveRez = new Move();
         string[] splittedMove = move.Split(',');
         for (int i = 0; i < splittedMove.Length; i++)
         {
            string[] sss = splittedMove[i].Split(' ');
            int wordNumber, productionGroupNumber, productionNumber;
            if (!int.TryParse(sss[0], out wordNumber)
               || !int.TryParse(sss[1], out productionGroupNumber)
               || !int.TryParse(sss[2], out productionNumber))
               throw new ArgumentException("Input string in wrong format.");
            moveRez.addMove(wordNumber, productionGroupNumber, productionNumber);
         }
         return moveRez;
      }

      public override string ToString()
      {
         StringBuilder sb = new StringBuilder();
```

```
            if (moves.Count != 0)
                sb.Append(moves[0].ToString());
            for (int index = 1; index < moves.Count; ++index)
            {
                sb.Append("," + moves[index].ToString());
            }
            return sb.ToString();
        }
    }
}

using System;

namespace ProductionsGameCore
{
    public class PrimaryMove
    {
        public int WordNumber { get; }
        public int ProductionGroupNumber { get; }
        public int ProductionNumber { get; }

        public PrimaryMove(int wordNumber, int productionGroupNumber, int productionNumber)
        {
            WordNumber = wordNumber;
            ProductionGroupNumber = productionGroupNumber;
            ProductionNumber = productionNumber;
        }

        public override string ToString()
        {
            return String.Format("{0} {1} {2}",
                WordNumber.ToString(),
                ProductionGroupNumber.ToString(),
                ProductionNumber.ToString());
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Converters;
using System.Windows.Media.Media3D;

namespace ProductionsGame
{
    public class Parameter
    {
        public String Id { get; private set; }
        public string Name { get; private set; }
        public Parameter(string id, string name)
        {
            Id = id;
            Name = name;
        }
    }
```

```
    public class IntParameter:Parameter
    {
        public int Value { get; set; }
        public IntParameter(string id, string name, int value):base(id,name)
        {
            Value = value;
        }
    }

    public class DoubleParameter : Parameter
    {
        public double Value { get; set; }
        public DoubleParameter(string id, string name, double value) : base(id, name)
        {
            Value = value;
        }
    }

    public class Parameters
    {
        List<Parameter> parameters = new List<Parameter>();
        public Parameters() { }

        public void addParameter(Parameter parameter)
        {
            parameters.Add(parameter);
        }

        public Parameter getParameter(string id)
        {
            foreach (var parameter in parameters)
                if (parameter.Id == id)
                    return parameter;
            return null;
        }

        //public int getParameterValue(string id)
        //{
        //    foreach (var parameter in parameters)
        //        if (parameter.Id == id)
        //            return parameter.Value;
        //    return 0;
        //}

        public IEnumerable<Parameter> getParameters()
        {
            return parameters;
        }
    }
}

using System.CodeDom;
using System.Collections.Generic;
using System.Linq;
using System;
using System.Text;

namespace ProductionsGameCore
{
```

```
    public class SimplifiedWord
    {
        public int terminals { get; set; }
        public int this[int i]{
            get => nonterminals[i];
            set => nonterminals[i] = value;
        }
        public int NonterminalsCount { get => nonterminals.Length; }
        internal int[] nonterminals;

        internal SimplifiedWord()
        {
            terminals = 0;
        }

        public SimplifiedWord(SimplifiedWord word)
        {
            terminals = word.terminals;
            nonterminals = new int[word.NonterminalsCount];
            for (int i = 0; i < word.NonterminalsCount; i++)
                nonterminals[i] = word.nonterminals[i];
        }

        public void addNonterminal(int index,int count) {
            nonterminals[index]+=count;
        }

        public int getNonterminal(int c)
        {
            return nonterminals[c];
        }

        public int[] getNonterminals() {
            return nonterminals;
        }

        public int getScore() {
            foreach (var nonterminal in nonterminals)
                if (nonterminal > 0)
                    return 0;
            return 3 + terminals;
        }

        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();
            foreach (var item in nonterminals)
                sb.Append(string.Format("{0} ",item));
            return sb.ToString();
        }
    }
}

using System.Collections.Generic;
using System.Runtime.InteropServices;

namespace ProductionsGameCore
{
    public class SimplifiedProductionGroup
```

```
    {
        public int Left { get; private set; }
        public int RightSize { get { return rights.Length; } }
        public SimplifiedWord this[int i] { get => rights[i]; internal set => rights[i] = value; }
        private SimplifiedWord[] rights;

        internal SimplifiedProductionGroup(int left,int rightSize)
        {
            this.Left = left;
            rights = new SimplifiedWord[rightSize];
        }

        public int getRightTerminalsAt(int index)
        {
            return rights[index].terminals;
        }

        public IEnumerable<SimplifiedWord> getRights()
        {
            return rights;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProductionsGameCore
{
    public class Simplifier
    {
        Dictionary<char, int> letters = new Dictionary<char, int>();
        Dictionary<int, char> reverseLetters = new Dictionary<int,char>();


        public Simplifier(List<ProductionGroup> groups) {
            foreach(var group in groups) {
                if (!letters.ContainsKey(group.Left))
                {
                    reverseLetters.Add(letters.Count, group.Left);
                    letters.Add(group.Left, letters.Count);

                }
                foreach (var right in group.getRights())
                    foreach (var letter in right)
                        if (letter >= 'A' && letter <= 'Z')
                            if (!letters.ContainsKey(letter))
                            {
                                reverseLetters.Add(letters.Count, letter);
                                letters.Add(letter, letters.Count);
                            }
            }
        }


        public List<SimplifiedProductionGroup> ConvertProductions(List<ProductionGroup> groups)
```

```
        {
            List<SimplifiedProductionGroup> rez = new List<SimplifiedProductionGroup>();
            foreach (var group in groups)
            {
                SimplifiedProductionGroup prod = new SimplifiedProductionGroup(letters[group.Left],group.
        RightSize);
                for (int i=0;i<group.RightSize;++i)
                    prod[i] = this.ConvertWord(group.getRightAt(i));
                rez.Add(prod);
            }
            return rez;
        }

        public SimplifiedWord ConvertWord(string word) {
            SimplifiedWord sword = new SimplifiedWord();
            sword.nonterminals = new int[letters.Count];
            foreach (var letter in word) {
                if (letter >= 'A' && letter <= 'Z')
                    sword[letters[letter]] += 1;
                else
                    sword.terminals += 1;
            }
            return sword;
        }

        public int ConvertChar(char letter) => letters[letter];
        public char GetChar(int letterIndex) => reverseLetters[letterIndex];
    }
}

using ProductionsGameCore;
using StrategyUtilities;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StrategyUtilities
{
    public static class StrategyUtilitiesClass
    {
        public static bool isHaveLetter(SimplifiedWord word, int letterIndex)
        {
            return word[letterIndex] >0;
        }

        public static List<int> findMatches(IEnumerable<SimplifiedWord> words, int letterIndex)
        {
            List<int> indexes = new List<int>();
            int index = 0;
            foreach (var word in words)
            {
                if (isHaveLetter(word, letterIndex)) indexes.Add(index);
                index++;
            }
            return indexes;
        }
```

```
public static int isHaveLetter(string word, char c)
{
  return word.IndexOf(c);
}

public static List<int> findMatches(IEnumerable<string> words, char c)
{
  List<int> indexes = new List<int>();
  int index = 0;
  foreach (string word in words)
  {
    if (isHaveLetter(word, c) != −1) indexes.Add(index);
    index++;
  }
  return indexes;
}


public static void countMetric(List<SimplifiedProductionGroup> productions,
  RandomSettings rs,
  out double[] netMetric,
  out double[][] prodMetric
  )
{
  netMetric = new double[productions.Count];
  prodMetric = new double[productions.Count][];
  int productionsCount = productions.Count;

  //initialization
  for (int prodIndex = 0; prodIndex < productionsCount; ++prodIndex)
  {
    netMetric[prodIndex] = −1;
    prodMetric[prodIndex] = new double[productions[prodIndex].RightSize];
    for (int rightIndex = 0; rightIndex < productions[prodIndex].RightSize; ++rightIndex)
      if (productions[prodIndex][rightIndex].NonterminalsCount == 0)
        netMetric[prodIndex] = prodMetric[prodIndex][rightIndex] = 1;
      else
        prodMetric[prodIndex][rightIndex] = −1;//not aclculated −1
  }
  bool found = true;
  while (found)//stop when exceed needed accuracy
  {
    found = false;
    for (int prodIndex = 0; prodIndex < productionsCount; ++prodIndex)
      for (int rightIndex = 0; rightIndex < productions[prodIndex].RightSize; ++rightIndex)
      {
        var right = productions[prodIndex][rightIndex];
        if (right.NonterminalsCount != 0)
        {//if production contains neterminal − calculating
          double rightSum = countWordMetric(right, rs, netMetric, productions);
          if (Math.Abs(prodMetric[prodIndex][rightIndex] − rightSum) > prodMetric[prodIndex][
rightIndex] * 0.05)//eps
          {
            found = true;
            prodMetric[prodIndex][rightIndex] = rightSum;
            netMetric[prodIndex] = Math.Max(netMetric[prodIndex], rightSum);
          }
```

```
            }
        }
    }
}

public static double countWordMetric(SimplifiedWord word,
    RandomSettings rs,
    double[] netMetric,
    List<SimplifiedProductionGroup> productions)
{
    int productionsCount = productions.Count;
    double rightSum = 1;
    for (int j =0;j<word.NonterminalsCount;++j)
    {
        double sum = 0;
        for (int i = 0; i < productionsCount; ++i)
            if (netMetric[i] != -1 && productions[i].Left == j)
                sum += netMetric[i] * rs.getProductionPossibility(i) / rs.getTotalPossibility();
        sum = Math.Pow(sum, word[j]);
        rightSum *= sum;
    }
    return rightSum;
}

public static void countBetterMetric(List<SimplifiedProductionGroup> productions,
    RandomSettings rs,
    out double[] netMetric,
    out double[][] prodMetric
    )
{
    netMetric = new double[productions.Count];
    prodMetric = new double[productions.Count][];
    int productionsCount = productions.Count;

    //initialization
    for (int prodIndex = 0; prodIndex < productionsCount; ++prodIndex)
    {
        netMetric[prodIndex] = -1;
        prodMetric[prodIndex] = new double[productions[prodIndex].RightSize];
        for (int rightIndex = 0; rightIndex < productions[prodIndex].RightSize; ++rightIndex)
            if (productions[prodIndex][rightIndex].NonterminalsCount == 0)
                netMetric[prodIndex] = prodMetric[prodIndex][rightIndex] = 1;
            else
                prodMetric[prodIndex][rightIndex] = -1;//not aclculated -1
    }
    bool found = true;
    while (found)//stop when exceed needed accuracy
    {
        found = false;
        for (int prodIndex = 0; prodIndex < productionsCount; ++prodIndex)
            for (int rightIndex = 0; rightIndex < productions[prodIndex].RightSize; ++rightIndex)
            {
                var right = productions[prodIndex][rightIndex];
                if (right.NonterminalsCount != 0)
                {//if production contains neterminal - calculating
                    double rightSum = countBetterWordMetric(right, rs, netMetric, productions);
                    if (Math.Abs(prodMetric[prodIndex][rightIndex] - rightSum) > prodMetric[prodIndex][
rightIndex] * 0.05)//eps
```

```
            {
                found = true;
                prodMetric[prodIndex][rightIndex] = rightSum;
                netMetric[prodIndex] = Math.Max(netMetric[prodIndex], rightSum);
            }
          }
        }
    }
}

public static double countBetterWordMetric(SimplifiedWord word,
    RandomSettings rs,
    double[] netMetric,
    List<SimplifiedProductionGroup> productions)
{
    int productionsCount = productions.Count;
    double min = 1;
    int count = 0;
    for (int j = 0; j < word.NonterminalsCount; ++j)
    {
        double sum = 0;
        for (int i = 0; i < productionsCount; ++i)
            if (netMetric[i] != -1 && productions[i].Left == j && word[j]>0)
                sum += netMetric[i] * rs.getProductionPossibility(i) / rs.getTotalPossibility();
        if (word[j] > 0)
        {
            min = Math.Min(min, sum);
            count += word[j];
        }
    }
    if(count>0)
        return min/count;
    else
        return 1;
}

public static void countStupidMetric(List<SimplifiedProductionGroup> productions,
    out int[] netMetric,
    out int[][] prodMetric
    )
{
    netMetric = new int[productions.Count];
    prodMetric = new int[productions.Count][];
    int productionsCount = productions.Count;

    for (int prodIndex = 0; prodIndex < productionsCount; ++prodIndex)
        prodMetric[prodIndex] = new int[productions[prodIndex].RightSize];

    for (int prodIndex = 0; prodIndex < productionsCount; ++prodIndex)
    {
        for (int rightIndex = 0; rightIndex < productions[prodIndex].RightSize; ++rightIndex)
        {
            var right = productions[prodIndex][rightIndex];
            int rightSum = countWordStupidMetric(right);
            prodMetric[prodIndex][rightIndex] = rightSum;
            netMetric[prodIndex] = Math.Min(netMetric[prodIndex], rightSum);
        }
    }
```

```
}

public static int countWordStupidMetric(SimplifiedWord word)
{
  int rightSum = 0;
  for (int j = 0; j < word.NonterminalsCount; ++j)
    rightSum += word[j];
  return rightSum;
}

public static void applyMove(PrimaryMove move, List<string> words, List<ProductionGroup>
prods)
{
  ProductionGroup prod = prods[move.ProductionGroupNumber];
  if (move.WordNumber != −1)
  {
    string word = words[move.WordNumber];
    int letterIndex = StrategyUtilitiesClass.isHaveLetter(word, prod.Left);
    string newWord = word.Substring(0, letterIndex) +
          prod.getRightAt(move.ProductionNumber) +
          word.Substring(letterIndex + 1, word.Length − letterIndex − 1);
    words[move.WordNumber] = newWord;
  }
  else
  {
    string newWord = prod.getRightAt(move.ProductionNumber);
    words.Add(newWord);
  }
}

public static void applyMove(Move move, Bank bank, SimplifiedWord word, List<
SimplifiedProductionGroup> prods)
{
  foreach (var m in move.getMoves())
  {
    bank.removeProduction(m.ProductionGroupNumber);
    var prod = prods[m.ProductionGroupNumber];
    word.addNonterminal(prod.Left, −1);
    word.terminals += prod[m.ProductionNumber].terminals;
    for (int j=0;j< prod[m.ProductionNumber].NonterminalsCount;++j)
      word.addNonterminal(j, −prod[m.ProductionNumber][j]);
  }
}

public static void applyMove(PrimaryMove move, List<SimplifiedWord> simpleWords, List<
SimplifiedProductionGroup> prods)
{
  var prod = prods[move.ProductionGroupNumber];
  if (move.WordNumber != −1)
  {
    SimplifiedWord word = simpleWords[move.WordNumber];
    word.addNonterminal(prod.Left, −1);
    word.terminals += prod[move.ProductionNumber].terminals;
    for (int j = 0; j < prod[move.ProductionNumber].NonterminalsCount; ++j)
      word.addNonterminal(j, −prod[move.ProductionNumber][j]);
  }
  else
  {
```

```
            SimplifiedWord word = new SimplifiedWord(prod[move.ProductionNumber]);
            simpleWords.Add(word);
          }
        }
      }
    }

using ProductionsGame.Properties;
using ProductionsGameCore;
using System;
using System.Collections.Generic;
using System.Diagnostics.Eventing;
using System.Linq;
using System.Runtime;
using System.Text;
using System.Threading.Tasks;

namespace ProductionsGame
{
    public abstract class Strategy
    {
        public string Name { get; protected set; }
        public string ShortName { get; protected set; }
        //public int PlayerNumber { get; private set; }
        protected Simplifier simplifier{ get; private set; }
        protected GameSettings GameSettings { get; private set; }
        protected List<ProductionGroup> productions;
        protected List<SimplifiedProductionGroup> simplifiedProductions;
        protected RandomSettings rs;

        /// <summary>
        /// Subscribe this if it is necessary to perform some actions when game settigs change (count
        metric, etc.).
        /// </summary>
        protected event EventHandler GameSettingsChanged = delegate { };

        protected Strategy()
        {
        }

        public static Parameters getParameters()
        {
            return new Parameters();
        }

        public void setGameSettings(GameSettings gameSettings)
        {
            GameSettings = gameSettings;
            productions = this.GameSettings.GetProductions().ToList();
            simplifiedProductions = this.GameSettings.GetSimplifiedProductions().ToList();
            this.rs = this.GameSettings.RandomSettings;
            simplifier = gameSettings.Simplifier;
            GameSettingsChanged.Invoke(this, null);
        }



        /// <summary>
        /// Метод который надо переопределить чтобы сделать свою стратегию.
```

```
        /// </summary>
        /// <param name="productionNumber"></param>
        public abstract Move makeMove(int playerNumber,
            int moveNumber,
            int productionNumber,
            List<List<string>> words,
            List<List<SimplifiedWord>> simplifiedWords,
            Bank bank);

        public override string ToString()
        {
            return Name;
        }
    }
}

using ProductionsGameCore;
using StrategyUtilities;
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace Strategies
{
    public class SmartRandomStrategy:RandomStrategy
    {
        protected double[] netMetric;
        protected double[][] prodsMetric;

        public SmartRandomStrategy():base() {
            Name = "Smart Random Strategy";
            ShortName = "SRS";
            this.GameSettingsChanged += beforeStart;
        }

        protected virtual void beforeStart(object sender, EventArgs e) {
            StrategyUtilitiesClass.countMetric(simplifiedProductions, rs, out netMetric, out prodsMetric);
        }

        /// <summary>
        /// Counts weights of production groups.
        /// override this method for new strategy
        /// </summary>
        protected override List<double> getGroupsWeights(List<int> indexes)
        {
            return indexes.Select((index) => netMetric[index]).ToList();
            return indexes.Select((index) => Math.Sqrt(netMetric[index])).ToList();
        }
        /// <summary>
        /// Counts weights of productions in production groups.
        /// </summary>
        protected override List<double> getProductionsWeights(int index)
        {
            return prodsMetric[index].Select((m) => m).ToList();
```

```
        return prodsMetric[index].Select((m)=> Math.Sqrt(m)).ToList();
    }
    /// <summary>
    /// Counts weights of words.
    /// </summary>
    protected override List<double> getWordsWeights(List<SimplifiedWord> words)
    {
        return words.Select(
            (word) =>
            StrategyUtilitiesClass.countWordMetric(word, rs, netMetric, simplifiedProductions)
            //Math.Sqrt(StrategyUtilitiesClass.countWordMetric(word, rs, netMetric,
    simplifiedProductions))
            ).ToList();
    }
}

public class ImprovedRandomStrategy : RandomStrategy
{
    protected double[] netMetric;
    protected double[][] prodsMetric;
    protected double[] netStupidMetric;
    protected double[][] prodsStupidMetric;

    public ImprovedRandomStrategy() : base()
    {
        Name = "Improved Random Strategy";
        ShortName = "IRS";
        this.GameSettingsChanged += beforeStart;
    }

    protected virtual void beforeStart(object sender, EventArgs e)
    {
        StrategyUtilitiesClass.countMetric(simplifiedProductions, rs, out netMetric, out prodsMetric);
        StrategyUtilitiesClass.countMetric(simplifiedProductions, rs, out netStupidMetric, out
    prodsStupidMetric);
    }

    /// <summary>
    /// Counts weights of production groups.
    /// override this method for new strategy
    /// </summary>
    protected override List<double> getGroupsWeights(List<int> indexes)
    {
        return indexes.Select((index) =>{
            if (netMetric[index] == 0 )
                return 0;
            else if(netMetric[index] == 1)
                return 0.01;
            else
                return netStupidMetric[index];
        }).ToList();
    }
    /// <summary>
    /// Counts weights of productions in production groups.
    /// </summary>
    protected override List<double> getProductionsWeights(int index)
    {
        List<double> weights = new List<double>();
```

```
      for (int i = 0; i < simplifiedProductions[index].RightSize; i++)
      {
        if (prodsMetric[index][i] == 0)
          weights.Add(0);
        else if (prodsMetric[index][i] == 1)
          weights.Add(0.01);
        else
          weights.Add(prodsStupidMetric[index][i]);
      }
      return weights;
    }
    /// <summary>
    /// Counts weights of words.
    /// </summary>
    protected override List<double> getWordsWeights(List<SimplifiedWord> words)
    {
      return words.Select((word) => {
        double m = StrategyUtilitiesClass.countWordMetric(word, rs, netMetric,
    simplifiedProductions);
        if (m == 0)
          return 0d;
        else if (m == 1)
          return 0.01;
        else
          return StrategyUtilitiesClass.countWordStupidMetric(word);
      }).ToList();
    }
}

public class BetterSmartRandomStrategy : SmartRandomStrategy
{

    public BetterSmartRandomStrategy() : base()
    {
      Name = "Better Smart Random Strategy";
      ShortName = "BSRS";
    }

    protected override void beforeStart(object sender, EventArgs e)
    {
      StrategyUtilitiesClass.countBetterMetric(simplifiedProductions, rs, out netMetric, out
    prodsMetric);
    }

    /// <summary>
    /// Counts weights of production groups.
    /// override this method for new strategy
    /// </summary>
    protected override List<double> getGroupsWeights(List<int> indexes)
    {
      return indexes.Select((index) => Math.Sqrt(netMetric[index])).ToList();
    }
    /// <summary>
    /// Counts weights of productions in production groups.
    /// </summary>
    protected override List<double> getProductionsWeights(int index)
    {
      return prodsMetric[index].Select((m) => Math.Sqrt(m)).ToList();
```

```
    }
    /// <summary>
    /// Counts weights of words.
    /// </summary>
    protected override List<double> getWordsWeights(List<SimplifiedWord> words)
    {
        return words.Select(
            (word) =>
            Math.Sqrt(StrategyUtilitiesClass.countWordMetric(word, rs, netMetric, simplifiedProductions
    ))
            ).ToList();
    }
}

public class InversedSmartRandomStrategy : SmartRandomStrategy
{

    public InversedSmartRandomStrategy() : base()
    {
        Name = "Inversed Smart Random Strategy";
        ShortName = "ISRS";
    }

    protected override void beforeStart(object sender, EventArgs e)
    {
        StrategyUtilitiesClass.countBetterMetric(simplifiedProductions, rs, out netMetric, out
    prodsMetric);
    }

    /// <summary>
    /// Counts weights of production groups.
    /// override this method for new strategy
    /// </summary>
    protected override List<double> getGroupsWeights(List<int> indexes)
    {
        return indexes.Select((index) =>
        {
            if (netMetric[index] == 0)
                return 0d;
            else
                return 1.01d - netMetric[index];
        }).ToList();
    }
    /// <summary>
    /// Counts weights of productions in production groups.
    /// </summary>
    protected override List<double> getProductionsWeights(int index)
    {
        return prodsMetric[index].Select((m) => {
            if (m == 0 )//|| m == 1
                return 0d;
            else
                return 1.01d - netMetric[index];
        }).ToList();

    }
    /// <summary>
    /// Counts weights of words.
```

```
        /// </summary>
        protected override List<double> getWordsWeights(List<SimplifiedWord> words)
        {
            return words.Select((word) => {
                double m = StrategyUtilitiesClass.countBetterWordMetric(word, rs, netMetric,
        simplifiedProductions);
                if (m == 0)//|| m == 1
                    return 0d;
                else
                    return 1.01d - m;
            }).ToList();
        }
    }
}

using ProductionsGame;
using ProductionsGameCore;
using StrategyUtilities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Strategies
{
    public class StupidShortWordsStrategy : Strategy
    {
        int[] netMetric;
        int[][] prodsMetric;
        int[] bestProd;

        public StupidShortWordsStrategy() : base() {
            Name = "Stupid Short Words Strategy";
            ShortName ="SSWS";
            this.GameSettingsChanged += beforeStart;
        }

        protected void beforeStart(object sender,EventArgs e)
        {
            StrategyUtilitiesClass.countStupidMetric(simplifiedProductions, out netMetric, out prodsMetric
        );

            bestProd = new int[netMetric.Length];
            for (int i = 0; i < bestProd.Length; ++i)
            {
                int minMetric = int.MaxValue;
                for (int j = 0; j < prodsMetric[i].Length; ++j)
                {
                    if (prodsMetric[i][j] < minMetric ||
                        prodsMetric[i][j] == minMetric &&
                        simplifiedProductions[i][j].terminals > simplifiedProductions[i][bestProd[j]].terminals)
                    {
                        minMetric = prodsMetric[i][j];
                        bestProd[i] = j;
                    }
                }
            }
        }
```

```
public override Move makeMove(int playerNumber,
   int moveNumber,
   int productionNumber,
   List<List<string>> words,
   List<List<SimplifiedWord>> simplifiedWords,
   Bank bank)
{
   Move move = new Move();

   List<SimplifiedWord> simpleWords = simplifiedWords[playerNumber];
   //get simplified form of words

   while (true)
   {
      PrimaryMove primaryMove;
      if (move.MovesCount == 0)//make first move
         primaryMove = findFirstMove(simpleWords, productionNumber);
      else//make move from bank
         primaryMove = findMove(simpleWords, bank);

      if (primaryMove != null)
      {
         if (move.MovesCount != 0)//delete production from bank
            bank.removeProduction(primaryMove.ProductionGroupNumber);
         //make this move
         move.addMove(primaryMove);
         StrategyUtilitiesClass.applyMove(primaryMove, simpleWords, simplifiedProductions);
      }
      else
         break;
   }
   return move;
}

PrimaryMove findMove(List<SimplifiedWord> simpleWords, Bank bank)
{
   int groupNumber;
   List<List<int>> allowedWords = new List<List<int>>();
   foreach (var pr in simplifiedProductions)//find words allowed for productions
      allowedWords.Add(StrategyUtilitiesClass.findMatches(simpleWords, pr.Left));

   //select production from bank
   {//find production with better metric
      groupNumber = -1;
      int minMetric = int.MaxValue;
      for (int i = 0; i < simplifiedProductions.Count; ++i)
         if (allowedWords[i].Count > 0 && bank.getProductionCount(i) > 0)
            if (netMetric[i] < minMetric)
            {
               minMetric = netMetric[i];
               groupNumber = i;
            }
      if (groupNumber == -1)
         return null;//not found production
   }

   SimplifiedProductionGroup prod = simplifiedProductions[groupNumber];
```

```
            int productionNumber = bestProd[groupNumber];
            int wordNumber = -1;
            //select worfd with better metric
            {
                int minMetric = int.MaxValue;
                for (int i = 0; i < allowedWords[groupNumber].Count; ++i)
                {
                    SimplifiedWord word = simpleWords[allowedWords[groupNumber][i]];
                    int metric = StrategyUtilitiesClass.countWordStupidMetric(word);
                    if (metric < minMetric)
                    {
                        minMetric = metric;
                        wordNumber = i;
                    }
                }
                wordNumber = allowedWords[groupNumber][wordNumber];
            }
            return new PrimaryMove(wordNumber, groupNumber, productionNumber);
        }

        PrimaryMove findFirstMove(List<SimplifiedWord> simpleWords,int productionGroupNumber)
        {
            int groupNumber;
            var prod = simplifiedProductions[productionGroupNumber];
            List<int> allowedWords = new List<int>();
            allowedWords = StrategyUtilitiesClass.findMatches(simpleWords, prod.Left);
            if (simplifier.GetChar(prod.Left) == 'S')//if can create new word
                allowedWords.Add(-1);

            if (allowedWords.Count == 0)
                return null;
            groupNumber = productionGroupNumber;

            int productionNumber = bestProd[groupNumber];
            int wordNumber = -1;
            //select word with better metric
            {
                int minMetric = int.MaxValue;
                for (int i = 0; i < allowedWords.Count; ++i)
                {
                    SimplifiedWord word;
                    if (allowedWords[i] == -1)
                        word = simplifier.ConvertWord("S");
                    else
                        word = simpleWords[allowedWords[i]];
                    int metric = StrategyUtilitiesClass.countWordStupidMetric(word);
                    if (metric < minMetric)
                    {
                        minMetric = metric;
                        wordNumber = i;
                    }
                }
                wordNumber = allowedWords[wordNumber];
            }
            return new PrimaryMove(wordNumber, groupNumber, productionNumber);
        }
    }
}
```

```
using ProductionsGame;
using ProductionsGameCore;
using StrategyUtilities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Strategies
{
   public class ShortWordsStrategy : Strategy
   {
      double[] netMetric;
      double[][] prodsMetric;
      int[] bestProd;

      public ShortWordsStrategy():base() {
         Name = "Short Words Strategy";
         ShortName = "SWS";
         this.GameSettingsChanged += beforeStart;
      }

      protected void beforeStart(object sender, EventArgs e)
      {
         StrategyUtilitiesClass.countMetric(simplifiedProductions, rs, out netMetric, out prodsMetric);
         bestProd = new int[netMetric.Length];
         for (int i = 0; i < bestProd.Length; ++i)
         {
            double maxMetric = -1;
            for (int j = 0; j < prodsMetric[i].Length; ++j)
            {
               if (maxMetric == -1 || prodsMetric[i][j] > maxMetric ||
                  prodsMetric[i][j] == maxMetric &&
                  simplifiedProductions[i][j].terminals > simplifiedProductions[i][bestProd[j]].terminals)
               {
                  maxMetric = prodsMetric[i][j];
                  bestProd[i] = j;
               }
            }
         }
      }

      public override Move makeMove(int playerNumber,
         int moveNumber,
         int productionNumber,
         List<List<string>> words,
         List<List<SimplifiedWord>> simplifiedWords,
         Bank bank)
      {
         Move move = new Move();

         //get simplified form of words
         List<SimplifiedWord> simpleWords = simplifiedWords[playerNumber];

         while (true)
         {
            PrimaryMove primaryMove;
```

```
      if (move.MovesCount == 0)
        primaryMove = findFirstMove(simpleWords, productionNumber);
      else
        primaryMove = findMove( simpleWords,  bank);
      if (primaryMove != null)
      {
        if (move.MovesCount != 0)//delete production from bank
          bank.removeProduction(primaryMove.ProductionGroupNumber);
        //make this move
        move.addMove(primaryMove);
        StrategyUtilitiesClass.applyMove(primaryMove, simpleWords, simplifiedProductions);
      }
      else
        break;
    }
    return move;
}

PrimaryMove findMove(List<SimplifiedWord> simpleWords, Bank bank)
{
    int prosuctionGroupNumber;
    List<List<int>> allowedWords = new List<List<int>>();
    foreach (var pr in simplifiedProductions)//find words allowed for productions
      allowedWords.Add(StrategyUtilitiesClass.findMatches(simpleWords, pr.Left));


    //select production from bank
    {
      prosuctionGroupNumber = -1;
      double maxMetric = -1;
      for (int i = 0; i < simplifiedProductions.Count; ++i)
        if (allowedWords[i].Count > 0 && bank.getProductionCount(i) > 0)
          if (netMetric[i] > maxMetric)
          {
            maxMetric = netMetric[i];
            prosuctionGroupNumber = i;
          }
      if (prosuctionGroupNumber == -1)
        return null;//not found production
    }

    SimplifiedProductionGroup prod = simplifiedProductions[prosuctionGroupNumber];
    int productionNumber = bestProd[prosuctionGroupNumber];
    int wordNumber = -1;
    //select worfd with better metric
    {
      double maxMetric = -1;
      for (int i = 0; i < allowedWords[prosuctionGroupNumber].Count; ++i)
      {
        SimplifiedWord word = simpleWords[allowedWords[prosuctionGroupNumber][i]];
        double metric = StrategyUtilitiesClass.countWordMetric(word,
          rs,
          netMetric,
          simplifiedProductions);
        if (metric > maxMetric)
        {
          maxMetric = metric;
          wordNumber = i;
```

```
            }
          }
          wordNumber = allowedWords[prosuctionGroupNumber][wordNumber];
        }
        return new PrimaryMove(wordNumber, prosuctionGroupNumber, productionNumber);
      }

      PrimaryMove findFirstMove(List<SimplifiedWord> simpleWords, int productionGroupNumber)
      {
        int groupNumber;
        var prod = simplifiedProductions[productionGroupNumber];
        List<int> allowedWords = new List<int>();

        allowedWords = StrategyUtilitiesClass.findMatches(simpleWords, prod.Left);
        if (simplifier.GetChar(prod.Left) == 'S')//if can create new word
          allowedWords.Add(-1);

        if (allowedWords.Count == 0)
          return null;
        groupNumber = productionGroupNumber;

        int productionNumber = bestProd[groupNumber];
        int wordNumber = -1;
        //select word with better metric
        {
          double maxMetric = -1;
          for (int i = 0; i < allowedWords.Count; ++i)
          {
            SimplifiedWord word;
            if (allowedWords[i] == -1)
              word = simplifier.ConvertWord("S");
            else
              word = simpleWords[allowedWords[i]];
            double metric = StrategyUtilitiesClass.countWordMetric(word,
              rs,
              netMetric,
              simplifiedProductions);
            if (metric > maxMetric)
            {
              maxMetric = metric;
              wordNumber = i;
            }
          }
          wordNumber = allowedWords[wordNumber];
        }
        return new PrimaryMove(wordNumber, groupNumber, productionNumber);
      }
    }
  }

  using ProductionsGame;
  using ProductionsGameCore;
  using StrategyUtilities;
  using System;
  using System.Collections.Generic;
  using System.Linq;
  using System.Runtime.Remoting.Channels;
  using System.Runtime.Serialization;
  using System.Text;
```

```
using System.Threading.Tasks;
using System.Xml.Schema;

namespace Strategies
{
   public class SearchStrategy : Strategy
   {
      //TODO add parameters for strategies for depth of search and for wieghts of combined strategy

      //TODO change it if this strategy works too long, or you want to make strategy better
      //max depth of search
      int maxDepth = 4;
      double[] netMetric;
      double[][] prodsMetric;


      /// <summary>
      /// Gets parameter depth – septh of recursion in search.
      /// </summary>
      /// <param name="parameters"></param>
      public SearchStrategy(Parameters parameters) : base()
      {
         if (parameters != null)
         {
            var param = parameters.getParameter("depth");
            if (param != null && param is IntParameter && (param as IntParameter).Value >= 0)
               maxDepth = (param as IntParameter).Value;
         }
         this.GameSettingsChanged += beforeStart;
         Name = "Search Strategy "+maxDepth;
         ShortName = "SS"+maxDepth;
      }

      public static new Parameters getParameters() {
         Parameters searchParameters = new Parameters();
         searchParameters.addParameter(new IntParameter("depth", "Глубина перебора", 4));
         return searchParameters;
      }

      protected void beforeStart(object sender, EventArgs e)
      {
         //count metric of broductions
         StrategyUtilitiesClass.countMetric(simplifiedProductions, rs, out netMetric, out prodsMetric);
      }

      public override Move makeMove(int playerNumber,
         int moveNumber,
         int productionNumber,
         List<List<string>> words,
         List<List<SimplifiedWord>> simplifiedWords,
         Bank bank)
      {

         Move move = new Move();

         List<double> wordsMetric =new List<double>();
         //count metric of all words
         var simpleWords = simplifiedWords[playerNumber];
```

```
   wordsMetric.Clear();
   for (int i = 0; i < simpleWords.Count; ++i)
      wordsMetric.Add(StrategyUtilitiesClass.countWordMetric(simpleWords[i], GameSettings.
RandomSettings, netMetric, simplifiedProductions));

   Move mov = findFirstMove(simpleWords, wordsMetric, bank, productionNumber);
   if (mov != null && mov.MovesCount != 0)
      move.addMove(mov);
   else
   {
      return move;
   }

   while (true)
   {
      Move move1 = findMove(simpleWords, wordsMetric, bank);
      if (move1 != null && move1.MovesCount != 0)
         move.addMove(move1);
      else
         break;
   }
   return move;
}

void searchMove(SimplifiedWord oldWord,
   int wordIndex,
   Bank bank,
   out string bestMove,
   out double bestMetric,
   out double bestTerminals,
   Move currentMove = null,
   int depth = 0)
{

   bestMove = "";
   bestMetric = -1;
   bestTerminals = 0;
   bool found = false;
   if (depth < maxDepth)
   {
      if (currentMove == null)
         currentMove = new Move();
      string bMove;
      double bMetric, bTerminals;
      SimplifiedProductionGroup prod;
      for (int prodIndex = 0; prodIndex < simplifiedProductions.Count; ++prodIndex)
      {
         if (bank.getProductionCount(prodIndex) <= 0) continue;

         prod = simplifiedProductions[prodIndex];
         if (oldWord.getNonterminal(prod.Left) <= 0) continue;
         found = true;
         bank.removeProduction(prodIndex);
         oldWord.addNonterminal(prod.Left, -1);
         for (int rightIndex = 0; rightIndex < prod.RightSize; ++rightIndex)
         {
            oldWord.terminals += prod[rightIndex].terminals;
            for(int nonterminal=0;nonterminal< prod[rightIndex].NonterminalsCount;++
```

```
nonterminal)
                oldWord.addNonterminal(nonterminal, prod[rightIndex][nonterminal]);
            currentMove.addMove(wordIndex, prodIndex, rightIndex);
            searchMove(oldWord, wordIndex, bank,
                out bMove, out bMetric, out bTerminals, currentMove, depth + 1);
            if (bestMetric == −1 || bMetric > bestMetric || bMetric == bestMetric && bTerminals >
bestTerminals)
            {
                bestMetric = bMetric;
                bestMove = bMove;
                bestTerminals = bTerminals;
            }
            currentMove.popMove();

            oldWord.terminals −= prod[rightIndex].terminals;
            for (int nonterminal = 0; nonterminal < prod[rightIndex].NonterminalsCount; ++
nonterminal)
                oldWord.addNonterminal(nonterminal, −prod[rightIndex][nonterminal]);
        }
        oldWord.addNonterminal(prod.Left, 1);
        bank.addProduction(prodIndex);
    }
  }
  if (!found)
  {
    bestMetric = StrategyUtilitiesClass.countWordMetric(oldWord, rs, netMetric,
simplifiedProductions);
    bestMove = currentMove.ToString();
    bestTerminals = oldWord.terminals;
  }
}

Move findFirstMove(List<SimplifiedWord> simpleWords,
  List<double> wordsMetric, Bank bank, int productionGroupNumber
  )
{

  var prod = simplifiedProductions[productionGroupNumber];
  //fond allowed words
  List<int> allowedWords = StrategyUtilitiesClass.findMatches(simpleWords, prod.Left).ToList();

  int maxIndex;
  SimplifiedWord word = null;
  {
    double maxMetric = −1;
    maxIndex = 0;
    //find word with minimum metric
    foreach (var index in allowedWords)
    {
      if (wordsMetric[index] > maxMetric)
      {
        maxMetric = wordsMetric[index];
        maxIndex = index;
        word = simpleWords[index];
      }
    }
    //if can create new word
    if (simplifier.GetChar(prod.Left) == 'S' &&
```

```
        maxMetric < StrategyUtilitiesClass.countWordMetric(simplifier.ConvertWord("S"), rs,
netMetric, simplifiedProductions))
      {
        maxIndex = −1;
        word = simplifier.ConvertWord("S");
      }
      else if (maxMetric == −1)
        return null;
    }
    Move move = new Move();
    string bestMove = "", tmpMove;
    double bestMetric = −1, tmpMetric, bestTerminals = 0, tmpTerminals;
    //if we can create new word (production S−>)
    int newIndex = (maxIndex == −1 ? simpleWords.Count : maxIndex);
    //senumerate all productions what we can apply to the word
    word.addNonterminal(prod.Left, −1);
    for (int rightIndex = 0; rightIndex < prod.RightSize; ++rightIndex)
    {
      word.terminals += prod[rightIndex].terminals;
      for(int nonterminal = 0; nonterminal < prod[rightIndex].NonterminalsCount;++nonterminal)
        word.addNonterminal(nonterminal, prod[rightIndex][nonterminal]);

      move.addMove(maxIndex, productionGroupNumber, rightIndex);
      searchMove(word,
        newIndex,
        bank,
        out tmpMove,
        out tmpMetric,
        out tmpTerminals,
        move);
      if (bestMetric == −1 || tmpMetric > bestMetric
        || tmpMetric == bestMetric && tmpTerminals > bestTerminals)
      {
        bestMetric = tmpMetric;
        bestMove = tmpMove;
        bestTerminals = tmpTerminals;
      }
      move.popMove();

      word.terminals −= prod[rightIndex].terminals;
      for (int nonterminal = 0; nonterminal < prod[rightIndex].NonterminalsCount; ++nonterminal
)
        word.addNonterminal(nonterminal,−prod[rightIndex][nonterminal]);
    }
    word.addNonterminal(prod.Left, 1);

    move = Move.FromString(bestMove);
    bank.addProduction(productionGroupNumber);
    StrategyUtilitiesClass.applyMove(move, bank, word, simplifiedProductions);
    return move;
}

Move findMove(List<SimplifiedWord> simpleWords,
    List<double> wordsMetric,
    Bank bank
    )
{
    string bestMove;
```

```
            double bestMetric, bestTerminals;
            List<int> allowedIndexes = new List<int>();
            for (int i = 0; i < simpleWords.Count; i++)
            {
                for (int prodIndex = 0; prodIndex < simplifiedProductions.Count; ++prodIndex)
                {
                    var prod = simplifiedProductions[prodIndex];
                    if (simpleWords[i].getNonterminal(prod.Left) > 0 && bank.getProductionCount(prodIndex)
        > 0)
                        allowedIndexes.Add(i);
                }
            }
            if (allowedIndexes.Count == 0)//no words found
                return null;
            int wordNumber = -1;
            //select word with best metric
            {
                double maxMetric = -1;
                for (int i = 0; i < allowedIndexes.Count; ++i)
                {
                    var word = simpleWords[allowedIndexes[i]];
                    double metric = StrategyUtilitiesClass.countWordMetric(word,
                        rs,
                        netMetric,
                        simplifiedProductions);
                    if (metric > maxMetric && metric != 1)
                    {
                        maxMetric = metric;
                        wordNumber = i;
                    }
                }
                wordNumber = allowedIndexes[wordNumber];
            }
            //find best move
            searchMove(simpleWords[wordNumber],
                wordNumber,
                bank,
                out bestMove,
                out bestMetric,
                out bestTerminals);
            Move move1 = Move.FromString(bestMove);
            wordsMetric[wordNumber] = bestMetric;
            StrategyUtilitiesClass.applyMove(move1, bank, simpleWords[wordNumber],
        simplifiedProductions);
            return move1;
        }
    }
}

using ProductionsGame;
using ProductionsGameCore;
using StrategyUtilities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Headers;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace Strategies
{
   public class RandomStrategy : Strategy
   {
      protected Random random;


      public RandomStrategy() : base()
      {
         Name = "Random Strategy";
         ShortName = "RS";
         random = new Random((int)DateTime.Now.Ticks * Thread.CurrentThread.ManagedThreadId);
      }
      /// <summary>
      /// Counts weights of production groups.
      /// override this method for new strategy
      /// </summary>
      protected virtual List<double> getGroupsWeights(List<int> indexes) {
         return indexes.Select((index)=>1d).ToList();
      }
      /// <summary>
      /// Counts weights of productions in production groups.
      /// </summary>
      protected virtual List<double> getProductionsWeights(int index) {
         int prods = productions[index].RightSize;
         List<double> weights = new List<double>();
         for (int i = 0; i < prods; ++i)
            weights.Add(1d);
         return weights;
      }
      /// <summary>
      /// Counts weights of words.
      /// </summary>
      protected virtual List<double> getWordsWeights(List<SimplifiedWord> words) {
         return words.Select((index) => 1d).ToList();
      }

      private int getRand(List<double> weights)
      {
         double sum = 0;
         foreach (double weight in weights)
            sum += weight;
         double rand = random.NextDouble() * sum;
         int i = 0;
         foreach (double weight in weights)
         {
            rand -= weight;
            if (rand <= 0)
               return i;
            i++;
         }
         return 0;
      }

      public override Move makeMove(int playerNumber,
         int moveNumber,
         int productionNumber,
```

```
    List<List<string>> words,
    List<List<SimplifiedWord>> simplifiedWords,
    Bank bank)
{
    Move move = new Move();


    while (true)
    {//TODO use simplified forms
        PrimaryMove primaryMove;
        if (move.MovesCount == 0)//make first move
            primaryMove = findFirstMove(simplifiedWords[playerNumber], productionNumber);
        else//make move from bank
            primaryMove = findMove(simplifiedWords[playerNumber], bank);

        //make this move
        if (primaryMove != null)
        {
            if (move.MovesCount != 0)
                bank.removeProduction(primaryMove.ProductionGroupNumber);
            move.addMove(primaryMove);
            StrategyUtilitiesClass.applyMove(primaryMove, simplifiedWords[playerNumber],
simplifiedProductions);
        }
        else
            break;
    }
    return move;
}


private PrimaryMove findFirstMove(List<SimplifiedWord> words, int productionGroupNumber)
{
    List<int> allowedWordsIndexes = new List<int>();
    List<SimplifiedWord> allowedWords;


    var prod = simplifiedProductions[productionGroupNumber];
    allowedWordsIndexes = StrategyUtilitiesClass.findMatches(words, prod.Left);
    allowedWords = allowedWordsIndexes.Select((index) => words[index]).ToList();
    if (simplifier.GetChar(prod.Left) == 'S')
    {//if can create new word
        allowedWordsIndexes.Add(-1);
        allowedWords.Add(simplifier.ConvertWord("S"));
    }

    if (allowedWordsIndexes.Count == 0)
        return null;//not found production

    int productionNumber = getRand(getProductionsWeights(productionGroupNumber));
    int wordnumber = getRand(getWordsWeights(allowedWords));
    wordnumber = allowedWordsIndexes[wordnumber];//select word

    return new PrimaryMove(wordnumber, productionGroupNumber, productionNumber);
}

private PrimaryMove findMove(List<SimplifiedWord> words, Bank bank)
{
```

```
        List<List<int>> allowedWordsIndexes = new List<List<int>>();
        List<List<SimplifiedWord>> allowedWords = new List<List<SimplifiedWord>>();

        int productionGroupNumber;
        foreach (var pr in simplifiedProductions)//find words allowed for productions
        {
          allowedWordsIndexes.Add(StrategyUtilitiesClass.findMatches(words, pr.Left));
          allowedWords.Add(allowedWordsIndexes.Last().Select((index) => words[index]).ToList());
          if (simplifier.GetChar(pr.Left) == 'S')
          {//if can create new word
            allowedWordsIndexes.Last().Add(-1);
            allowedWords.Last().Add(simplifier.ConvertWord("S"));
          }
        }

        List<int> allowedGroupIndexes = new List<int>();
        for (int i = 0; i < productions.Count; ++i)
          if (allowedWordsIndexes[i].Count > 0 && bank.getProductionCount(i) > 0)
            allowedGroupIndexes.Add(i);
        if (allowedGroupIndexes.Count == 0)
          return null;//not found production
        productionGroupNumber = getRand(getGroupsWeights(allowedGroupIndexes));
        productionGroupNumber = allowedGroupIndexes[productionGroupNumber];

        int productionNumber = getRand(getProductionsWeights(productionGroupNumber));
        int wordnumber = getRand(getWordsWeights(allowedWords[productionGroupNumber]));
        wordnumber = allowedWordsIndexes[productionGroupNumber][wordnumber];//select word

        return new PrimaryMove(wordnumber, productionGroupNumber, productionNumber);
      }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ProductionsGameCore
{
  public class RandomSettings
  {
    private int totalPossibility;
    private List<int> possibilityList;

    public RandomSettings(int totalPossibility, IEnumerable<int> possibilityList)
    {
      this.totalPossibility = totalPossibility;
      this.possibilityList = possibilityList.ToList();
      int sum = 0;
      foreach (int possibility in possibilityList)
      {
        sum += possibility;
        if (possibility <= 0)
          throw new ArgumentException("Probabolity must be non-negative number.");
      }
```

```
            if (sum != totalPossibility)
                throw new ArgumentException("Sum of probabilities must be equal to totalPossibility.");
        }

        public int getProductionPossibility(int index)
        {
            return possibilityList[index];
        }

        public int productionsCount()
        {
            return possibilityList.Count;
        }

        public int getTotalPossibility()
        {
            return totalPossibility;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using ProductionsGameCore;

namespace ProductionsGame
{
    public class RandomProvider
    {
        public RandomSettings RandomSettings { get; private set; }
        private Random random;
        public int Seed;

        public RandomProvider(RandomSettings randomSettings)
        {
            RandomSettings = randomSettings;
            //создаём случайный сид для будуещей генерации случайных чисел, добавив некую защи
        ту от повторений при многопоточности
            Seed = (int)DateTime.Now.Ticks * Thread.CurrentThread.ManagedThreadId;
            random = new Random(Seed);
        }

        public int getRandom()
        {//получаем номер продукции с учётом их вероятностей
            int r = random.Next(RandomSettings.getTotalPossibility());
            for (int i = 0; i < RandomSettings.productionsCount(); ++i)
            {
                r -= RandomSettings.getProductionPossibility(i);
                if (r < 0) return i;
            }
            return RandomSettings.productionsCount() - 1;
        }
    }
}
```

```
using ProductionsGame;
using ProductionsGameCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Strategies
{
   public class MixedStrategy : Strategy
   {
      List<Strategy> strats = new List<Strategy>();
      List<int>probs = new List<int>();
      int sum;
      Random random;


      public MixedStrategy(Parameters parameters) : base()
      {
         int randomProb = 1;
         int searchProb = 1;
         int shortProb = 1;
         int adaptiveProb = 1;
         if (parameters != null)
         {
            var param = parameters.getParameter("randomProb");
            if (param != null && param is IntParameter && (param as IntParameter).Value >= 0)
               randomProb = (param as IntParameter).Value;
            param = parameters.getParameter("searchProb");
            if (param != null && param is IntParameter && (param as IntParameter).Value >= 0)
               searchProb = (param as IntParameter).Value;
            param = parameters.getParameter("shortProb");
            if (param != null && param is IntParameter && (param as IntParameter).Value >= 0)
               shortProb = (param as IntParameter).Value;
            param = parameters.getParameter("adaptiveProb");
            if (param != null && param is IntParameter && (param as IntParameter).Value >= 0)
               adaptiveProb = (param as IntParameter).Value;
         }
         probs.Add(randomProb);
         probs.Add(searchProb);
         probs.Add(shortProb);
         probs.Add(adaptiveProb);
         sum = randomProb + searchProb + shortProb + adaptiveProb;
         strats.Add(new BetterSmartRandomStrategy());
         strats.Add(new SearchStrategy(parameters));
         strats.Add(new ShortWordsStrategy());
         strats.Add(new AdaptiveRandomStrategy(parameters));
         random = new Random((int)DateTime.Now.Ticks * Thread.CurrentThread.ManagedThreadId);
         this.GameSettingsChanged += beforeStart;
         Name = string.Format("Mixed Strategy {0} {1} {2} {3}",randomProb, shortProb, searchProb,
      adaptiveProb);
         ShortName = string.Format("MS{0}{1}{2}{3}", randomProb, shortProb, searchProb,
      adaptiveProb);

      }
```

```
        public static new Parameters getParameters() {
            Parameters mixedParameters = new Parameters();
            //mixedParameters.addParameter(new IntParameter("depth", "Глубина перебора", 4));
            mixedParameters.addParameter(new IntParameter("randomProb", "Вес умной случайной с
        тратегии", 1));
            mixedParameters.addParameter(new IntParameter("shortProb", "Вес стратегии коротких с
        лов", 1));
            mixedParameters.addParameter(new IntParameter("searchProb", "Вес переборной стратег
        ии", 1));
            mixedParameters.addParameter(new IntParameter("adaptiveProb", "Вес адаптивной случа
        ной стратегии", 1));
            return mixedParameters;
        }

        protected void beforeStart(object sender,EventArgs e)
        {
            foreach(var strat in strats)
                strat.setGameSettings(GameSettings);
        }

        public override Move makeMove(int playerNumber,
            int moveNumber,
            int productionNumber,
            List<List<string>> words,
            List<List<SimplifiedWord>> simplifiedWords,
            Bank bank)
        {
            int rez = random.Next(sum);
            for (int i = 0; i < strats.Count; ++i) {
                rez -= probs[i];
                if (rez < 0)
                    return strats[i].makeMove(playerNumber,moveNumber,productionNumber,words,
        simplifiedWords,bank);
            }
            return new Move();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ProductionsGame;
using ProductionsGameCore;

namespace Strategies
{
    public class AdaptiveRandomStrategy: Strategy
    {
        //public int changeMoves = 20;
        private double changeStrategyCoef = 10d;
        Strategy startingStrategy;
        Strategy finishingStrategy;

        public AdaptiveRandomStrategy(Parameters parameters) : base()
        {
            startingStrategy = new InversedSmartRandomStrategy();
```

```
            //startingStrategy = new ImprovedRandomStrategy();
            finishingStrategy = new SearchStrategy(parameters);
            //finishingStrategy = new ShortWordsStrategy();
            if (parameters != null)
            {
                var param = parameters.getParameter("finish_coef");
                if (param != null && param is DoubleParameter && (param as DoubleParameter).Value >= 0)
                    changeStrategyCoef = (param as DoubleParameter).Value;
            }
            this.GameSettingsChanged += beforeStart;
            Name = "Adaptive Random "+changeStrategyCoef.ToString();
            ShortName = "AR" + changeStrategyCoef.ToString();
        }


        protected void beforeStart(object sender, EventArgs e) {
            startingStrategy.setGameSettings(this.GameSettings);
            finishingStrategy.setGameSettings(this.GameSettings);
        }

        public static new Parameters getParameters()
        {
            Parameters searchParameters = new Parameters();
            searchParameters.addParameter(new DoubleParameter("finish_coef", "Коэффициент смены
         стратегии", 10d));
            return searchParameters;
        }

        protected int countNonterminals(List<SimplifiedWord> words) {
            int count = 0;
            foreach (SimplifiedWord word in words)
                for(int nonterminal=0;nonterminal<word.NonterminalsCount;++nonterminal)
                    count += word[nonterminal];
            return count;
        }

        public override Move makeMove(int playerNumber, int moveNumber, int productionNumber,
        List<List<string>> words, List<List<SimplifiedWord>> simplifiedWords, Bank bank)
        {
            //TODO make smarter strategy shange
            int nonterminals = countNonterminals(simplifiedWords[playerNumber]);
            //if(MoveNumber < GameSettings.NumberOfMoves − changeMoves)+ changeMoves + 20
            if (nonterminals * changeStrategyCoef < GameSettings.NumberOfMoves − moveNumber )//||
        nonterminals >= 10
                return startingStrategy.makeMove(playerNumber, moveNumber, productionNumber, words,
        simplifiedWords, bank);
            else
                return finishingStrategy.makeMove(playerNumber, moveNumber, productionNumber, words,
         simplifiedWords, bank);

        }
    }
}

using ProductionsGameCore;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
```

```
using System.Text;
using System.Threading;

namespace ProductionsGame
{
   public class Game:IDisposable
   {
      public GameSettings GameSettings { get; private set; }
      public RandomSettings RandomSettings { get { return GameSettings.RandomSettings; } }
      public IEnumerable<ProductionGroup> Productions { get { return GameSettings.GetProductions()
      ; } }
      private List<SimplifiedProductionGroup> simplifiedProductions;
      private List<Strategy> players;
      protected RandomProvider RandomProvider { get; private set; }
      public Bank Bank { get; private set; }

      public int MoveNumber { get; private set; }
      public int ActivePlayer { get; private set; }

      public List<List<string>> words;
      public List<List<SimplifiedWord>> simplifiedWords;
      List<int> scores;
      private bool scoresOutdated;
      private Winner CurrentWinner;

      public enum State {
         Ready,
         Active,
         Finished,
         Failed
      }

      public enum Winner
      {
         First = 0,
         Second = 1,
         Draw = -1
      }

      public State state { get; private set; }
      public string lastError { get; private set; }

      private StreamWriter log;
      public string LogFilename {get; private set;}

      public Game(GameSettings gameSettings,IEnumerable<Strategy> players, string logFilename =
      null) {
         GameSettings = gameSettings;
         if (players.Count() != 2)
            throw new ArgumentException("There are must be two players in game.");
         this.players = players.ToList();
         MoveNumber = 0;
         ActivePlayer = 0;
         Bank = new Bank(GameSettings.ProductionsCount);
         RandomProvider = new RandomProvider(GameSettings.RandomSettings);
         words = new List<List<string>> { new List<string>() , new List<string>() };
         simplifiedWords = new List<List<SimplifiedWord>> { new List<SimplifiedWord>(),new List<
      SimplifiedWord>()};
```

```
    simplifiedProductions = GameSettings.GetSimplifiedProductions().ToList();
    scores = new List<int> { 0, 0 };
    state = State.Ready;

    if (logFilename != null)
      this.LogFilename = logFilename;
    else
    {
      StringBuilder sb = new StringBuilder();
      if (!Directory.Exists(@"./logs/"))//Создаём директорию для записи туда результатов
        Directory.CreateDirectory(@"./logs/");
      sb.Append(@"./logs/");
      sb.Append(DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss-"));
      sb.Append(Thread.CurrentThread.ManagedThreadId);
      sb.Append(".txt");
      this.LogFilename = sb.ToString();
    }
    log = new StreamWriter(this.LogFilename);
    log.AutoFlush = false;
    GameSettings.WriteToStream(log);
    log.WriteLine();
    for (int i = 0; i < 2; i++)
      log.WriteLine("Игрок {0}: {1}", i + 1, this.players[i].ToString());
  }

  private List<List<string>> getWordsCopies() {
    List<List<string>> nwords = new List<List<string>>();
    foreach(var playerW in words)
    {
      nwords.Add(new List<string>());
      playerW.ForEach(x => nwords.Last().Add(x));
    }
    return nwords;
  }

  private List<List<SimplifiedWord>> getSWordsCopies()
  {
    List<List<SimplifiedWord>> nwords = new List<List<SimplifiedWord>>();
    foreach (var playerW in simplifiedWords)
    {
      nwords.Add(new List<SimplifiedWord>());
      playerW.ForEach(x => nwords.Last().Add(new SimplifiedWord(x)));
    }
    return nwords;
  }

  private void logMove(Move move,int production) {
    if (state == State.Failed) {
      log.WriteLine("ERROR");
      log.WriteLine(lastError);
    }
    log.WriteLine("Индекс выпавшей продукции: {0}",production);
    log.WriteLine("Номер хода:{0}, Игрок: {1}", MoveNumber+1, ActivePlayer);
    log.WriteLine("Ход:{0}", move);
    log.WriteLine("Банк: {0}", Bank);
    log.Flush();
  }
```

```
private void logFinish()
{
   if (scoresOutdated)
      countScores();
   log.WriteLine("Результаты:");
   log.WriteLine("Счёт игрока 0: {0}", scores[0]);
   log.WriteLine("Счёт игрока 1: {0}", scores[1]);
   log.Flush();
   log.Close();
}

private bool applyMoves(Move moves,int firstProductionNumber) {
   scoresOutdated = true;
   if (moves == null)
   {
      lastError = string.Format("The {0} Player {1} finished game with error: {2}",
         ActivePlayer, players[ActivePlayer].Name, "player error");
      state = State.Failed;
      return false;
   }
   bool isfirst = true;


   foreach (var move in moves.getMoves())
   {
      int productionGroupNumber = move.ProductionGroupNumber,
         productionNumber = move.ProductionNumber,
         wordNumber = move.WordNumber;
      ProductionGroup production = GameSettings.getProductionGroup(productionGroupNumber)
;
      SimplifiedProductionGroup simplifiedProduction = simplifiedProductions[
productionGroupNumber];
      char left = production.Left;
      int leftIndex = simplifiedProduction.Left;
      string right = production.getRightAt(productionNumber);

      if (isfirst)
      {//Проверка что указана верная продукция
         if (firstProductionNumber != productionGroupNumber) {
            lastError = String.Format("Неверный ход. Номер группы продукции должен быт
ь {0}, но был {1}.", firstProductionNumber, productionNumber);
            state = State.Failed;
            return false;
         }
      }
      else
      {//Проверка что указаная продукция есть в банке
         if (Bank.getProductionCount(productionGroupNumber) == 0)
         {
            lastError = String.Format("Неверный ход. В банке не содержится продукций с но
мером {0}.", productionGroupNumber);
            state = State.Failed;
            return false;
         }
      }
      if (wordNumber >= 0 && wordNumber < words[ActivePlayer].Count)
      {
         string word = words[ActivePlayer][wordNumber];
```

```
        SimplifiedWord sword = simplifiedWords[ActivePlayer][wordNumber];
        int index = word.IndexOf(left);
        if (sword.getNonterminal(leftIndex) <= 0 || index==−1)
        {
          lastError = String.Format("Неверный ход. Вывод {0} не содержит нетерминала {1}
", word, left);
          state = State.Failed;
          return false;
        }
        else
        {//Применяем продукцию. B−>cDe : aBf−> acDef
          string newWord = word.Substring(0, index) +
            right +
            word.Substring(index + 1, word.Length − index − 1);
          words[ActivePlayer][wordNumber] = newWord;
          //Применяем к упрощённой форме
          sword.addNonterminal(leftIndex, −1);
          sword.terminals += simplifiedProduction[productionNumber].terminals;
          for(int i = 0; i < simplifiedProduction[productionNumber].NonterminalsCount;++i)
            sword.addNonterminal(i, simplifiedProduction[productionNumber][i]);
        }
      }
      else
      {//Применяем продукцию к новому слову
        if (left == 'S')
        {
          words[ActivePlayer].Add(right);
          simplifiedWords[ActivePlayer].Add(new SimplifiedWord(simplifiedProduction[
productionNumber]));
        }
      }
      if (!isfirst)  //Удаляем продукци?? из банка
        Bank.removeProduction(productionGroupNumber);
      isfirst = false;
    }
    if (isfirst)//осталось истиной −> ни одна продукция не была применена, добавить её в банк
      Bank.addProduction(firstProductionNumber);

    {// проверить что применены все возможные продукции
      if (moves.MovesCount == 0)//??сли не была применена первая продукция
      {
        char left = GameSettings.getProductionGroup(firstProductionNumber).Left;
        int leftIndex = simplifiedProductions[firstProductionNumber].Left;

        if (left == 'S')
        {
          lastError  = String.Format("Группа продукций {0} может быть применена для соз
дания нового вывода, но не была применена.", firstProductionNumber);
          state = State.Failed;
          return false;
        }
        foreach (var word in simplifiedWords[ActivePlayer])
        {
          if (word.getNonterminal(leftIndex) >0)
          {
            lastError = String.Format("Группа продукций {0} может быть применена к выв
о??у {1}.", firstProductionNumber, word);
            state = State.Failed;
```

```
                return false;
            }
        }

    }
    else
    {//проверим сто в банке нет применимых прдукций
        int productionNumber = 0;
        foreach (var production in GameSettings.GetProductions())
        {
            if (Bank.getProductionCount(productionNumber) <= 0)
                continue;
            char left = production.Left;
            productionNumber++;
            foreach (var word in simplifiedWords[ActivePlayer])
            {
                if (word.getNonterminal(left) > 0)
                {
                    lastError = String.Format("Группа продукций {0} может быть применена к вы
воду {1}.", firstProductionNumber, word);
                    state = State.Failed;
                    return false;
                }
            }
        }
    }
  }
  return true;
}

public void play()
{
  if (state == State.Ready || state == State.Active)
  {
    state = State.Active;
    for (MoveNumber = 0; MoveNumber < GameSettings.NumberOfMoves; ++MoveNumber)
    {
      for (ActivePlayer = 0; ActivePlayer < 2; ++ActivePlayer)
      {
        int production = RandomProvider.getRandom();
        Move move = null;
        try
        {
          move = players[ActivePlayer]//TODO – send copies + apply moves
            .makeMove(ActivePlayer, MoveNumber, production, getWordsCopies(),
getSWordsCopies(), new Bank(Bank.getProductions()));
        }
        catch (Exception e)
        {
          lastError = string.Format("The {0} Player {1} finished game with error: {2}",
            ActivePlayer, players[ActivePlayer].Name, e.Message);
          state = State.Failed;
          logMove(move, production);
          logFinish();
          return;
        }
        bool result = applyMoves(move, production);
        logMove(move, production);
```

```
            if (!result)
            {
               logFinish();
               return;
            }
         }
      }
      state = State.Finished;
      logFinish();
   }
}

public Move playOneMove() {
   if (state == State.Ready || state == State.Active)
   {

      //TODO chatch exception, log it
      Move move=null;
      int production= RandomProvider.getRandom();
      try
      {
         move = players[ActivePlayer]//TODO − send copies + apply moves
            .makeMove(ActivePlayer, MoveNumber, production, getWordsCopies(), getSWordsCopies
(), new Bank(Bank.getProductions()));
      }
      catch (Exception e)
      {
         lastError = string.Format("The {0} Player {1} finished game with error: {2}",
            ActivePlayer, players[ActivePlayer].Name, e.Message);
         state = State.Failed;
         logMove(move, production);
         logFinish();
         return null;
      }
      bool result = applyMoves(move, production);
      logMove(move, production);
      if (!result) {
         logFinish();
         return null;
      }
      ++ActivePlayer;
      if (ActivePlayer >= 2)
      {
         ++MoveNumber;
         ActivePlayer = 0;
      }
      if (MoveNumber >= GameSettings.NumberOfMoves)
      {
         state = State.Finished;
         logFinish();
      }
      return move;
   }
   return null;
}

public List<string> getPlayers() {
   return players.Select(x=> x.ToString()).ToList();
```

```
        }

        public Winner getWinner() {
            if(scoresOutdated)
                countScores();
            return CurrentWinner;
        }

        public List<int> getScores() {
            if (scoresOutdated)
                countScores();
            return scores.ToList();
        }

        private void countScores() {
            scoresOutdated = false;
            for(int i=0;i<2;++i) {
                scores[i]  = simplifiedWords[i].Select(word=>word.getScore()).Sum();
            }
            if (scores[0] > scores[1])
                CurrentWinner = Winner.First;
            else if (scores[0] < scores[1])
                CurrentWinner = Winner.Second;
            else
                CurrentWinner = Winner.Draw;
        }

        public void Dispose()
        {
            log.Close();
        }
    }
}
```