Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, механики и математики

**Отчет по лабораторной работе**

**ИССЛЕДОВАНИЕ РАЗЛИЧНЫХ ИГРОВЫХ АЛГОРИТМОВ ПОИСКА НАИЛУЧШЕГО ХОДА НА ПРИМЕРЕ ИГРЫ «РУССКИЕ ШАШКИ»**

**Выполнили**:студент группы 0823-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Грачева Е. А.

студент группы 0826-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панов А. А.

Подпись

**Проверил**:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

Нижний Новгород

2017

**Содержание**

Оглавление

[Введение 3](#_Toc482100429)

[Постановказадачи 4](#_Toc482100430)

[Алгоритмы 5](#_Toc482100431)

[Алгоритм полного перебора 5](#_Toc482100432)

[Оценочная функция 6](#_Toc482100433)

[Alpha-Beta с отсечениями 6](#_Toc482100434)

[Форсированные варианты 7](#_Toc482100435)

[Реализация 8](#_Toc482100436)

[Алгоритмическое ядро (С++) 9](#_Toc482100437)

[Описание алгоритмов 10](#_Toc482100438)

[Описание структур данных 10](#_Toc482100439)

[Графический интерфейс (C#) 11](#_Toc482100440)

[Апробация 13](#_Toc482100441)

[Заключение 14](#_Toc482100442)

[Литература 15](#_Toc482100443)

[Приложение. Фрагменты исходного кода программы 16](#_Toc482100444)

[template<class Data> class Stack 16](#_Toc482100445)

# Введение

В настоящее время в программировании популярно направление, связанное с компьютеризацией различных логических игр. Мотивируется это тем, что, во-первых, компьютерные игры достаточно востребованы; во-вторых, логические игры представляют собой хороший материал для исследования различных алгоритмов поиска наилучшего хода.

# Постановказадачи

Цели работы: исследовать различные алгоритмы поиска лучшего хода в игровой программе, реализовать и повозможности оптимизировать их, а также выбрать наиболее эффективные. В качестве разрабатываемой игровой программы были выбраны “русские шашки”, т.к. игра имеет сравнительно простую логику, но при этом является хорошим примером, иллюстрирующим различные алгоритмы поиска оптимального хода. Сравнение с известными программами, такими как Тундра или Аврора, позволит оценить эффективность наших алгоритмов.

# Алгоритмы

Шашки – игра с конечным числом состояний. Это означает, что, обладая бесконечными вычислительными ресурсами, мы смогли бы найти решение этой игры – то есть такую стратегию, следуя которой всегда можно было бы выиграть или, по крайней мере, сыграть вничью. Каждая позиция pos в шашках - это либо победа белых, либо победа черных, либо ничья. Определим функцию ***f:P→Z***, которая ставит в соответствие позиции ***pos*** из множества всевозможных позиций ***P*** целое число, отражающее «выгодность» этой позиции для текущего игрока***.*** Такая функция называется *оценочной функцией.*Если бы у нас был бесконечно быстрый компьютер, мы могли бы вычислить ее следующим образом:

* присвоим всем финальным позициям значения –1, 0, 1, в зависимости от исхода игры;
* применим рекурсивное правило где **pos** соответствует ходу текущего игрока, **pos‘** соответствует ходу противника, а  **pos→pos‘** обозначает все допустимые ходы из позиции **pos**.

Чем больше значение функции, тем выгодней позиция. Таким образом, ставя каждому ходу в соответствие его оценку, мы бы могли однозначно определить лучший ход.

Количество [возможных позиций](http://en.wikipedia.org/wiki/Shannon_number) в шашках составляет около 5\*1020. Вычисления такого масштаба выполнить практически невозможно (канадским ученым потребовалось 200 компьютеров и 20 лет вычислений). Следовательно, находить наилучший ход нужно другим способом.

### Алгоритм полного перебора

Самым простым способом получить если не лучший, то хотя бы осмысленный ход, является метод полного перебора с возвратом в соединении с рекурсией. Если сгенерировать все возможные ходы из данной позиции для белых, затем для каждого такого хода повторить те же действия для черных, потом снова для белых и т.д., получится простой и понятный рекурсивный алгоритм, основанный на переборе позиций. Пример дерева игры, возникающего таким образом, представлен на рис.х,х1.

Однако, как уже было сказано, рассмотреть все возможные варианты до конца невозможно, поэтому приходится ограничивать глубину рекурсии. Пусть на каждом шаге известна текущая глубина рекурсии – параметр ***depth***, первоначально равный максимальной заданной глубине. Тогда если ***depth***≠0, то рекурсия будет вызываться с параметром ***depth-1***, в противном случае вычисления следует прекратить и вызвать *оценочную функцию[[1]](#footnote-2)*.

Блок-схема алгоритма полного перебора представлена на рис.х.

### Оценочная функция

Пусть P - множество всевозможных позиций на доске. Функция f:P→Z, ставящая в соответствие некоторой позиции из множества P целое число, отражающее «выгодность» этой позиции для текущего игрока, называется оценочной функцией.

Простейшая оценочная функция считает шашки на доске, т.е. суммирует вес всех белых шашек и вычитает из полученного результата сумму всех черных шашек. Если шашек поровну, функция вернет 0.

Установим вес шашек следующим образом:

простая шашка – 50;

дамка – 350.

Таким образом, простая оценочная функция при расстановке шашек как на рис. для белого игрока вернет число 250, для черного игрока -250.

Более сложная оценочная функция учитывает ценность полей доски, которая определяется следующими принципами:

надо стараться по возможности подвигаться вперед, т.к. чем ближе к последней горизонтали, тем сила полей больше;

поскольку ценность шашек a1 и h2 является наименьшей, то их надо по возможности быстро ввести в игру;

учитывая, что с полей c5 и f4 удобно препятствовать развитию сил противника, следует бороться за овладением центральными полями c5 и f4, стремиться к захвату центра;

следует придерживаться принципа пропорционального, равномерного распределения шашек по обоим флангам;

шашки с полей c1, e1, g1 лучше без особой надобности не сдвигать, чтобы не открывать поля последнего ряда;

дамке по возможности стоит контролировать главную диагональ, т.к. она будет препятствовать продвижению противника в дамки.

Согласно этим правилам, ценность полей доски была обозначена следующим образом (рис.х):

На практике оказалось, что бот, использующий простую оценочную функцию, проигрывает боту, использующему более сложную оценочную функцию.

### Alpha-Beta с отсечениями

Рассмотрим основной алгоритм оптимизации перебора - alpha-beta c отсечениями. Суть его в том, что для получения оценки такой же точности, как и при полном переборе, совершенно не обязательно просматривать все варианты. Для определения отсекаемых вариантов не требуется знать особенностей данной игры.

В рекурсивной функции мы должны ввести две новые переменные — максимум для белых (alpha) и максимум для черных (beta). При начальном вызове обе эти величины равны минимально возможному значению. Если в какой-то позиции, например, для черных, мы получили результат, превышающий максимум для черных, достигнутый до этого, мы увеличиваем этот максимум (beta). Теперь рассмотрим некоторую строку игры. Представим, что где-то в глубине дерева перебора мы обнаружили, что максимумы белых и черных сравнялись. Допустим, мы просчитываем позицию для белых. Если мы продолжим перебирать в данной позиции, то максимум для белых может еще увеличиться, а может остаться прежним, но он уже сравнялся с максимумом черных. Это значит, что когда программа поднимется на уровень выше (рекурсивно), результат не будет записан, т. к. он не превышает максимума для черных в этой позиции. Также это значит, что в данной позиции для белых мы можем прекратить перебор и вернуть полученный результат досрочно. Дальше считать нет смысла.

Проиллюстрируем вышесказанное на примере (рис.х). Пусть на каком-либо шаге рекурсии максимум для белых (alpha) равен 4. Сделаем какой-либо ход и просчитаем возможные ходы для черных (опустимся на уровень ниже). Пусть при дальнейших вычислениях для одного из ходов результат равен 1 (beta=1). Если впоследствии найдется ход, оценка которого будет меньше 1, то он будет записан в beta, но не будет записан в alpha, и наоборот, если найдется ход, оценка которого будет больше 1, то он не будет записан даже в beta, следовательно, и не будет записан в alpha. Получается, дальнейшие вычисления бессмысленны.

Блок-схема алгоритма поиска с отсечениями представлена на рис..

На практике результаты показали, что…

### Форсированные варианты

В некоторых ситуациях, например, в случае размена, прекращение вычислений по достижении максимальной глубины рекурсии может привести к крайне неверной оценке позиции (*эффект горизонта*). Рассмотрим пример (рис.х). Пусть максимальная глубина рекурсии равна 2. Ход белых. Даже не очень опытному игроку очевидно, что лучший ход – *d8-h8*, поскольку он приводит к полному уничтожению противника. Однако бот не увидит преимуществ такого хода и сходит *h2-b8*. Чтобы избежать подобных ситуаций, отдельные ветки стоит просчитывать на большую глубину.

В шашках форсированными вариантами являются взятия. Поэтому в конце каждой ветки рекурсии вызывается упрощенная функция поиска, рассматривающая только взятия. По завершению размена вызывается оценочная функция. Если же взятий не было, оценочная функция вызывается сразу.

# Реализация

Проект состоит из двух частей — вычислительного ядра и графического интерфейса. В вычислительной части написаны различные функции поиска лучшего хода, то есть искусственный интеллект для бота. Графическая оболочка предоставляет визуализацию игры, предоставляет настройки бота и дает возможность ходить реальному игроку.

Вычислительное ядро

Графическая оболочка

find best move

best move

Нахождение наилучшего хода требует большого объема вычислений и для обеспечения наилучшей производительности вычислительное ядро написано в виде динамической библиотеки на C++

Графическая оболочка, написана на C#, так как данный язык предоставляет удобный графический инструментарий.

Итоговое приложение отображает игру с помощью графической оболочки и находит лучший ход с помощью вычислительного ядра.

## Алгоритмическое ядро (С++)

### Описание алгоритмов

### Описание структур данных

## Графический интерфейс (C#)

Для удобства графическая часть выполнена посхемеMVC:

View

Controller

Model

I’m update!

user’s change

change date

В ней данные разделяются на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо. Также изменение и отображение данных происходит строго по заданной схеме:

* ***Модель*** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;
* ***Представление*** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели;
* ***Контроллер*** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений;

Модель в проекте представленаклассом Game;

представление в проекте представлено классом Form1;

контроллер в проекте представлен классом Controller;

#### Структура класса Game

Класс хранит информациюнеобходимую для игры.

##### Структуры данныххранящих информацию об игре:

* staticLogicBoardboard;— игроваядоска
* List<Checker>[] checkers; —спискишашек;

checkers[0]—список шашек для белого игрока;

checkers[1]—список шашек для черного игрока;

* List<Checker>selectedCheckers; — выделенные на игровой доске шашки
* staticMovesmoves;— ходы доступные для выделенной шашки
* staticBotMovebotMove;— ходы, которые будет совершать бот

##### Состояние игры:

* Colorcolor; —цвет текущего игрока
* StatusPlayer[] statusPlayer;—статус белого и черного игрока

Статус принимает следующие значения:

bot— игрок это компьютер

human—игрок это человек

empty—игрок не установлен

* StatusApplicationstatusApplication; —статусприложения

Статус принимает следующие значения:

game—идет процесс игры

constructor—выбран конструктор и дет конструирование игры

menu—выбрано меню

##### Настройки поиска лучшего хода:

* Search[] statusSearch; —тип поиска лучшего хода

Принимает следующие значения:

FullSearch— типпоиска[Search](#_Search)

AlphaBetaSearch—типпоиска[Alpha-Beta](#_Alpha-Beta)

ForcedSearch—типпоиска[Alpha-Beta forcing](#_Форсирование)

empty—типпоиска не установлен

* int[] statusDepth;—глубина рекурсии для поиска лучшего хода
* Evaluate[] statusEvaluate;—тип оценочной функции

Принимает следующие значения:

SimpleEvaluate— типоценочной функции

SmartEvaluate—типоценочной функции

empty—типоценочной функциине установлен

***Диаграмма классов:***

# Апробация

# Заключение

В данной лабораторной работе вычислялись значения арифметических выражений. Для этого выражения разбивались на *лексемы* и проверялись на *корректность* ввода с помощью простейшего конечного автомата. Затем выполнялась необходимая сортировка лексем и итоговое вычисление.

# Литература

* Гергель В. П., Лабутина А. А. Учебно-образовательный комплекс по методам программирования //Нижний Новгород: ННГУ им. НИ Лобачевского. – 2007.

# Приложение. Фрагменты исходного кода программы

## template<class Data> class Stack

Stack(int\_size)//конструкторинициализации

{

if (\_size>= 0)

top = -1;

elsethrowNEGATIVE\_INDEX;

size = \_size;

ar = newData[size];

}/\*-------------------------------------------------------------------------\*/

void push(Datanum)

{

if (top + 1 < size)

ar[++top] = num;

elsethrowSTACK\_FULL;

}/\*-------------------------------------------------------------------------\*/

voidquickPush(Datanum)

{

ar[++top] = num;

}/\*-------------------------------------------------------------------------\*/

Datapop()

{

if (top > 0)

return ar[top--];

elsethrowSTACK\_EMPTY;

}/\*-------------------------------------------------------------------------\*/

DataquickPop()

{

return ar[top--];

}/\*-------------------------------------------------------------------------\*/

Data&showTop()

{

return ar[top];

}/\*-------------------------------------------------------------------------\*/

DatagetElem(inti)

{

return ar[i];

}/\*-------------------------------------------------------------------------\*/

boolisEmty()

{

return (top == -1);

}/\*-------------------------------------------------------------------------\*/

boolisFull()

{

return (top + 1 == size);

}/\*-------------------------------------------------------------------------\*/

intgetNumTop()

{

return top;

}/\*-------------------------------------------------------------------------\*/

voidsetTop(int\_top)

{

top = \_top;

}/\*-------------------------------------------------------------------------\*/

~Stack()//деструктор

{

if (ar != 0)

delete[] ar;

}/\*-------------------------------------------------------------------------\*/

1. См … [↑](#footnote-ref-2)