statuМинистерство 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](#_Toc482021973)

[Постановка задачи 4](#_Toc482021974)

[Алгоритмы 5](#_Toc482021975)

[Алгоритм полного перебора 5](#_Toc482021976)

[Alpha-Beta с отсечениями 5](#_Toc482021977)

[Форсирование 5](#_Toc482021978)

[Оценочная функция 5](#_Toc482021979)

[Реализация 6](#_Toc482021980)

[Алгоритмическое ядро (С++) 7](#_Toc482021981)

[Описание алгоритмов 8](#_Toc482021982)

[Описание структур данных 8](#_Toc482021983)

[Графический интерфейс (C#) 9](#_Toc482021984)

[Апробация 11](#_Toc482021985)

[Заключение 12](#_Toc482021986)

[Литература 13](#_Toc482021987)

[Приложение. Фрагменты исходного кода программы 14](#_Toc482021988)

[template <class Data> class Stack 14](#_Toc482021989)

# 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.

# Постановка задачи

Цели работы: исследовать различные алгоритмы поиска лучшего хода в игровой программе, реализовать и по возможности 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 для белых, либо победа для черных, либо ничья. Мы можем обозначить это с помощью функции ***f(pos).*** Если бы у нас был бесконечно быстрый компьютер, мы могли бы вычислить ее следующим образом:

1. Присвоим всем финальным позициям значения –1, 0, 1, в зависимости от исхода игры.
2. Применим рекурсивное правило

f(pos) = max(pos→pos‘) – f(pos‘),где pos→pos‘ обозначает все допустимые ходы из позиции pos. Знак минус используется потому, что игроки делают ходы поочередно, то есть, если позиция p соответствует ходу белых, тогда позиция pos‘ соответствует ходу черных.

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.

Введем следующее понятие: *оценочная функция* – это функция определенная на множестве позиций и принимающая числовое значение, характеризующее данную позицию. Чем больше это значение, тем выгодней эта позиция.

### Алгоритм полного перебора

Пусть дано некоторое начальное состояние игры:

Сгенерируем все возможные ходы:

### Alpha-Beta с отсечениями

### Форсирование

### Оценочная функция

# 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 по схеме MVC:

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 представлена классом Game;

представление в проекте представлено классом Form1;

контроллер в проекте представлен классом Controller;

#### Структура класса Game

Класс хранит информацию необходимую для игры.

##### Игровые структуры данных:

* static LogicBoard board; — игровая доска
* List<Checker>[] checkers; — массив списков шашек;

checkers[0] — список шашек для белого игрока;

checkers[1] — список шашек для черного игрока;

* List<Checker> selectedCheckers; — выделенные на игровой доске шашки
* static Moves moves; — ходы доступные для выделенной шашки
* static BotMove botMove; — ходы, которые будет совершать бот

##### Состояние игры:

* Color color; — 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

##### Настройки поиска лучшего хода:

* Search[] statusSearch; — тип поиска лучшего хода

Принимает следующие значения:

FullSearch — тип поиска [Search](#_Search)

AlphaBetaSearch — тип поиска [Alpha-Beta](#_Alpha-Beta)

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

empty — тип поиска не установлен

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

Принимает следующие значения:

SimpleEvaluate — тип оценочной функции

SmartEvaluate — тип оценочной функции

empty — тип оценочной функции не 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)//конструктор инициализации

{

if (\_size >= 0)

top = -1;

else throw NEGATIVE\_INDEX;

size = \_size;

ar = new Data[size];

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

void push(Data num)

{

if (top + 1 < size)

ar[++top] = num;

else throw STACK\_FULL;

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

void quickPush(Data num)

{

ar[++top] = num;

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

Data pop()

{

if (top > 0)

return ar[top--];

else throw STACK\_EMPTY;

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

Data quickPop()

{

return ar[top--];

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

Data& showTop()

{

return ar[top];

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

Data getElem(int i)

{

return ar[i];

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

bool isEmty()

{

return (top == -1);

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

bool isFull()

{

return (top + 1 == size);

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

int getNumTop()

{

return top;

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

void setTop(int \_top)

{

top = \_top;

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

~Stack()//деструктор

{

if (ar != 0)

delete[] ar;

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