Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Конструирование программного обеспечения (КПО)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

«Игровой движок для русских шашек»

БГУИР КР 1-40 01 01 115 ПЗ

Студент: гр. 251001 Кривицкий Ф. Ю.

Руководитель:

асс. Шостак Е. В.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2023 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Кривицкому Федору Юрьевичу

1. Тема работы Игровой движок для русских шашек.

2. Срок сдачи студентом законченной работы 28.12.2023 г.

3. Исходные данные к работе: среда разработки Microsoft Visual Studio.

Входные данные: библиотека SFML.

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по установке и использованию;

Список используемой литературы

Заключение 5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. "Игровой движок для русских шашек", А1, схема программы, чертеж.

6. Консультант по курсовой работе

Шостак Е.В.

7. Дата выдачи задания 15.09.2023 г.––––––––––––––––––––––   –

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 01.10.2023 – 15 % готовности работы;

разделы 2, 3 к 01.11.2023 – 30 % готовности работы;

разделы 4, 5 к 01.12.2023 – 60 % готовности работы;

раздел 6 к 15.12.2023 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 2 – 100 % готовности работы.

Защита курсовой работы с 21.12.2023 по 28.12.2023

РУКОВОДИТЕЛЬ–––––– Е. В. Шостак

(подпись)

Задание принял к исполнению –––\_\_\_\_––

(дата и подпись студента)

# **СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 4](#_Toc152667987)

[ВВЕДЕНИЕ 5](#_Toc152667988)

[1 АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ 7](#_Toc152667989)

[1.1 Анализ литературных источников 7](#_Toc152667990)

[1.2 Анализ существующих аналогов 8](#_Toc152667991)

[1.3 Постановка задачи 9](#_Toc152667992)

[2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАМНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 10](#_Toc152667993)

[2.1 Описание функциональных требований к программному средству 10](#_Toc152667994)

[2.2 Спецификация функциональных требований 10](#_Toc152667995)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 12](#_Toc152667996)

[3.1 Построение дерева возможных ходов 12](#_Toc152667997)

[3.2 Разработка алгоритмов подсчёта оценки 14](#_Toc152667998)

[3.3 Добавление ходов 16](#_Toc152668002)

[3.4 Оптимизация перебора 17](#_Toc152668006)

[4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА 18](#_Toc152668008)

[4.1 Проектирование модулей программного средства 18](#_Toc152668009)

[4.2 Разработка модулей программного средства 18](#_Toc152668010)

[4.3 Проектирование основных структур данных 22](#_Toc152668011)

[5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 23](#_Toc152668013)

[5.1 Тестирование и проверка работоспособности программного средства 23](#_Toc152668014)

[5.2 Анализ полученных результатов 25](#_Toc152668015)

[6 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ 26](#_Toc152668021)

[ЗАКЛЮЧЕНИЕ 28](#_Toc152668022)

[ПРИЛОЖЕНИЕ А 30](#_Toc152668025)

[ПРИЛОЖЕНИЕ Б 35](#_Toc152668026)

# **ВВЕДЕНИЕ**

**Русские шашки** — традиционный и наиболее популярный вид [шашек](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D1%88%D0%BA%D0%B8) в России, [странах](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D1%81%D0%BE%D0%B2%D0%B5%D1%82%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE) бывшего [СССР](https://ru.wikipedia.org/wiki/%D0%A1%D0%A1%D0%A1%D0%A0) и в [Израиле](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D1%80%D0%B0%D0%B8%D0%BB%D1%8C). Цель игры — лишить противника возможности хода путём взятия или запирания всех его шашек (в [обратных русских шашках](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B5_%D1%80%D1%83%D1%81%D1%81%D0%BA%D0%B8%D0%B5_%D1%88%D0%B0%D1%88%D0%BA%D0%B8) цель противоположна — лишить себя возможности хода).

[Шашечная доска](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D1%88%D0%B5%D1%87%D0%BD%D0%B0%D1%8F_%D0%B4%D0%BE%D1%81%D0%BA%D0%B0) располагается между партнерами таким образом, чтобы слева от играющего находилось тёмное угловое поле. В начальной позиции у каждой стороны по 12 шашек, которые занимают первые три ряда с каждой стороны. Игра ведется по тёмным полям. Шашки делятся на простые и дамки. В начальном положении все шашки простые. При достижении последнего (восьмого от себя) горизонтального ряда простая шашка превращается в дамку. Дамка обычно обозначается либо двумя шашками, поставленными друг на друга, либо перевернутой шашкой.

Подробнее о правилах русских шашек:

https://shashki.ru/variations/draughts64

Цель данной курсовой работы – разработать игровой движок для русских шашек, который будет находить наилучшие ходы для текущей ситуации на доске и производить их, уметь строить дерево ходов на несколько вперёд, перебирать все возможные ходы игрока и оценивать ситуацию на доске некоторым числом, которое объективно отображало бы позицию. Также создать графическую оболочку для движка с целью удобного взаимодейтсвия программы и пользователя.

Данная пояснительная записка содержит следующие основные разделы:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.
7. **АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ**
   1. **Анализ литературных источников**

Когда создавались первые вычислительные машины, их воспринимали только как дополнение к человеческому разуму. И до недавнего времени так и было. Программисты учили компьютеры играть в шахматы с 1960-х годов. И тогда победа у игрока-новичка уже считалась большим прогрессом. О серьёзных матчах даже не задумывались.

В 1980-х программа Belle достигла рейтинга Эло в 2250 пунктов, что примерно соответствует рейтингу мастера спорта. И с того времени развитие компьютерных шахмат вышло на совершенно новый уровень. Сначала честь человечества не смог защитить Гарри Каспаров в 1996 году, а сегодня уже создана нейросеть с рейтингом около 5000 Эло, что в разы превосходит даже сильнейших игроков.

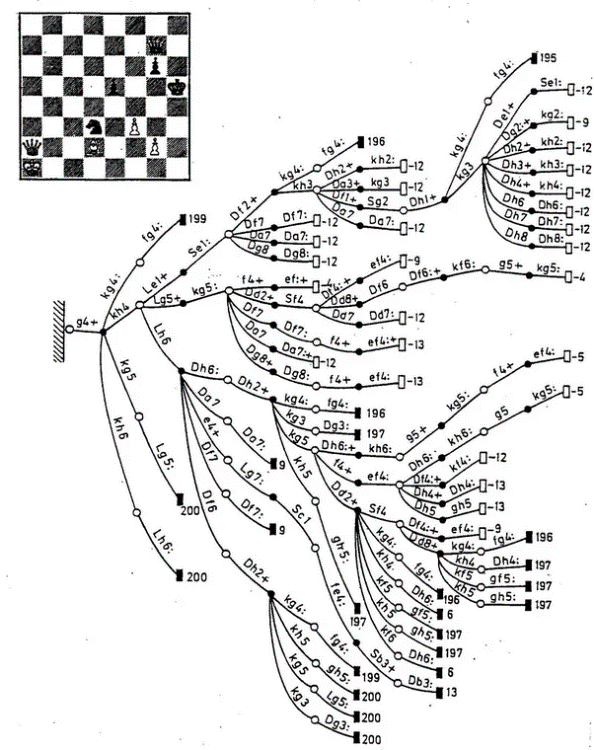
Основой движка является построение дерева ходов. Идея алгоритмов была взята из open-source движка для шахмат «Stockfish». Неофициальное название игрового движка для русских шашек «Vobla» пошло от дословного перевода на русский язык «Stockfish», поскольку «Vobla» строится на алгоритмах, адаптированных под русские шашки.

Рисунок 1.1 – Пример дерева возможных ходов Stockfish

* 1. **Анализ существующих аналогов**

**Aurora Borealis** (с лат. — «северное сияние») — компьютерная шашечная программа. Разработчик Александр Свирин. Фактически, первая широко распространенная программа для работы с базами партий в 64-клеточные шашки.

Отличительная особенность программы — все виды шашек собраны в одном модуле программы, в то время как в других проектах разные виды шашек представляют собой отдельные программы (например, серии программ Plus, WinDames, более современные Торнадо, Эдэон).

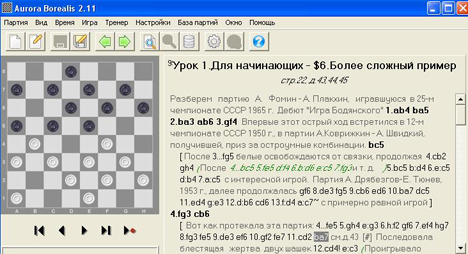
Текущая версия (3.7 — январь 2016) работает уже с 14 системами шашечной игры, наиболее популярными в мире.

Рисунок 1.2 – Интерфейс «Aurora Borealis»

I чемпионат мира по русским шашкам среди компьютерных программ проходил 20-21 января 2008 года в Москве. Турнир был организован FMJD (Всемирной федерацией шашек) и включён в официальный календарь турниров на 2008 год.

«Чемпион мира» — Торнадо и её разработчик — Михаил Глизерин. Торнадо в дополнительном матче переиграла со счётом 5:3 Kallisto (разработчик — Игорь Коршунов).

«Aurora Borealis» заняла 5 место.

* 1. **Постановка задачи**

В своём проекте, в соответствии с заданием, я буду использовать следующие алгоритмы:

* Метод построения дерева всех возможных ходов для текущей ситуации;
* Метод частичного перебора возможных ходов.
* Метод оценки текущей позиции на доске.

После построения ходов в глубину необходимо построить все возможные сбития, иначе оценка позиции может быть не объективна. Сбивания при построении считаются за единицу глубины.

Задачей курсовой работы будет игровой движок, который будет просичтывать ходы за время, зависящее от глубины просчета. При слишком быстром просчете ходов, следует увеличить глубину:

* Глубина 3: от 1 до 5 мс;
* Глубина 4: от 5 до 20 мс;
* Глубина 5: от 20 до 200 мс;
* Глубина 6: от 200 мс до 2 с;
* Глубина 7: от 2 до 15 с;
* Глубина 8: от 15 с до 2 мин.

Конечным результатом хода должно быть изменение доски, путем передвижения шашки компьютером по координатам самого лучшего хода, определенным исходя из дерева всех возможных ходов. После хода необходимо удалить все дочерние узлы хода, которые больше не понадобятся при анализе. В противном случае, оперативная память будет быстро засоряться ненужными данными.

1. **АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАМНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**
   1. **Описание функциональных требований к программному средству**

Цель разработки программного средства (ПС) – поиск лучшего хода в шашечной партии при данной позиции на доске. Для реализации любого программного средства, необходимо чётко определить функциональные требования, которые в дальнейшем послужат базисом для оформления и реализации тематики курсовой работы. Используя данные, полученые из раздела «Анализ литературных источников», опираясь на изученные аналоги и вводные данные курсовой работы, для реализации движка выделяются следующие основные требования к функционалу к программному средству:

* Выбор цвета шашек для пользователя;
* Выбор сложности движка, то есть глубины просчета ходов;
* Отрисовка шашечной доски;
* Возможность осуществления ходов;
* Построение дерева ходов и выбор наилучшего хода для движка;
* Шкала оценки позиции для игрока;
* Определение победы, ничьей, поражения для игрока;
* Высвечивание сообщения при завершении партии;
* Возможность начать новую партию, не выходя из программы.
  1. **Спецификация функциональных требований**

1. Выбор цвета шашек для пользователя:

* По кнопке на стартовом экране пользователь может выбрать цвет, за который он хочет играть;
* Если цвет – черный, то доска дожна отображаться в перевернутом виде.

1. Выбор сложности движка, то есть глубины просчета ходов:

* Должны быть следующие уровни сложности:
  + Easy (4 хода);
  + Medium (5 ходов);
  + Hard (6 ходов);
  + Impossible (7 ходов).

1. Отрисовка шашечной доски:

* Шашечная доска должна быть зелено-белой;
* Белые и черные шашки должны быть соответствующих цветов;
* Белая дамка должна иметь внутри черный круг, а черная – белый.

1. Возможность осуществления ходов:

* Возможность осуществлять все те и только те возможные ходы, которые позволяют правила при данной ситуации на доске;
* Должна быть возможность ходить при помощи мыши.

1. Построение дерева ходов и выбор наилучшего хода для движка:

* Должна происходить сортировка ходов по оценке после прохода всех элементов дерева;
* Движок должен ходить по лучшему из вариантов.

1. Шкала оценки позиции для игрока:

* Справа около доски должна быть шкала оценки, которая отображает саму оценку текстом и графически.

1. Определение победы, ничьей, поражения для игрока:

* Победа и поражение должны определяться по правилам ФШР;
* Ничья может определяться правилом 15 ходов без взятий.

1. Высвечивание сообщения при завершении партии:

* При окончании партии должно высвечиваться соответствующее сообщение;
* Текст сообщения должен исходить из итогов партии.

1. Возможность начать новую партию, не выходя из программы:

* Меню и основная программа должны взаимодействовать друг с другом в пределах одного окна;
* В любой момент партии должна быть возможность прервать ее и начать новую

1. **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**
   1. **Построение дерева возможных ходов**

Каждый узел дерева содержит координаты хода, который привел к образованию этого узла; доску, которая получится при этом ходе; и цвет узла, определяющий очередь совершения хода. Если ход белых, то узел содержит True, в противном случае – False. Также при создании узла считается оценка текущего положения на доске, которая будет играть ключевую роль при выборе наилучшего хода.

Корневым элементом дерева считается узел, содержащий начальную доску, ход белых и координаты хода (0; 0; 0; 0). Затем, если нет необходимости бить, строятся все возможные ходы для текущей ситуации (всего существует 7 вариантов первого хода для белых) и добавляются как дочерние элементы для узла, у них становится противоположным цвет. Если на каком-либо этапе построения дерева возникает необходимость сбивать, то цвет узла дерева меняется на противоположный только после завершения сбиваний.

Алгоритм продолжается, пока высота дерева не достигнет некоторой определенной глубины.

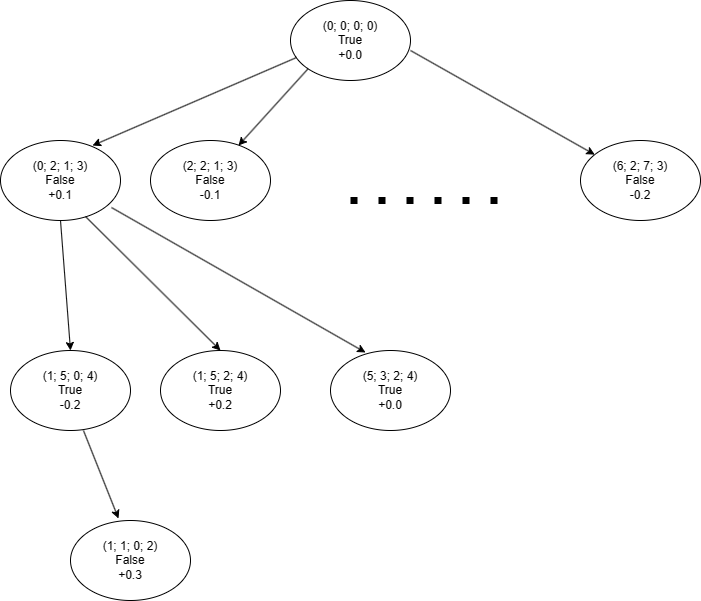


Рисунок 3.1.1 – Пример дерева возможных ходов

После построения дерева идет этап оценки каждого из дочерних элементов корневого узла путем рекурсивного концевого обхода. Алгоритм следующий:

1. Если в корне стоит True, то выбрать ход с максимальной оценкой;
2. Иначе выбрать ход с минимальной.

Для примера возьмем узел (0; 2; 1; 3). В корне стоит False, а значит

необходимо выбрать ход с минимальной оценкой. У дочерних узлов (1; 5; 2; 4) и (5; 3; 2; 4) нет потомков, а значит их оценка никак не поменяется. У узла

(1; 5; 0; 4) есть единственный потомок (1; 1; 0; 2), поэтому оценка (1; 5; 0; 4) станет равной оценке (1; 1; 0; 2).

Вернёмся к узлу (0; 2; 1; 3). Минимальная оценка из его дочерних у хода

(5; 3; 2; 4). Присваиваем её и получаем дерево, которое прошло алгоритм оценки ходов:

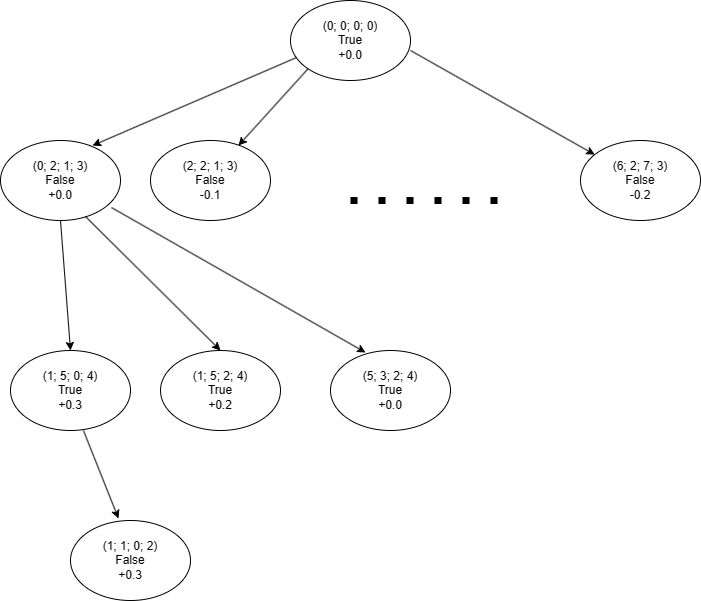


Рисунок 3.1.2 – Оценённое дерево возможных ходов

Далее происходит сортировка дочерних элементов корневого элемента по оценке и выбор наилучшего хода. Узел с наилучшей оценкой становится корневым, остальные удаляются из памяти. Алгоритм продолжается до тех пор, пока существует следующий ход. При отсутствии такового, объявляется конец игры. Кто не смог сделать ход – проиграл.

* 1. **Разработка алгоритмов подсчёта оценки**
     1. **Подсчёт оценки для обычных шашек**

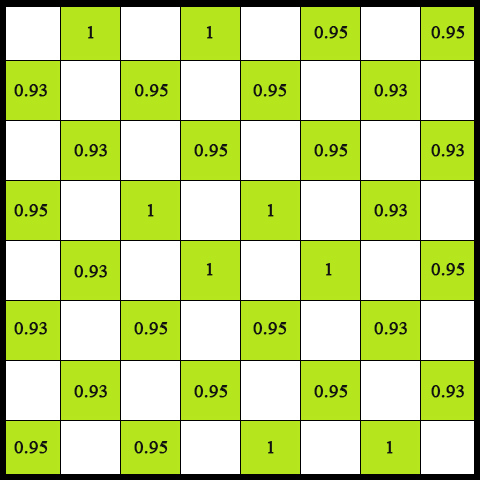
Позиция на доске характеризуется двумя важными составляющими: расположение шашек и их количество. Иногда бывает, что три шашки в центре могут держать четыре шашка по краям доски, поэтому пришлось ввести доску оценки позиции для каждой шашки:

Рисунок 3.2.1 – Оценка позиции обычной шашки

Исходя из рисунка 3.2.1 видно, что хорошо оцениваются шашки, стоящие ближе к центру. По словам известного шашиста Остапчука М.Г., важными позициями сделаны клетки e1 и g1, а также симметричные им относительно центра b8 и d8. Эти позиции являются ключевыми, открывать их стоит только в крайних ситуациях. Позиции недалеко от центра, но не на краях доски считаются плохими из-за сложности сделать «петлю». Вместо двух шашек понадобятся три.

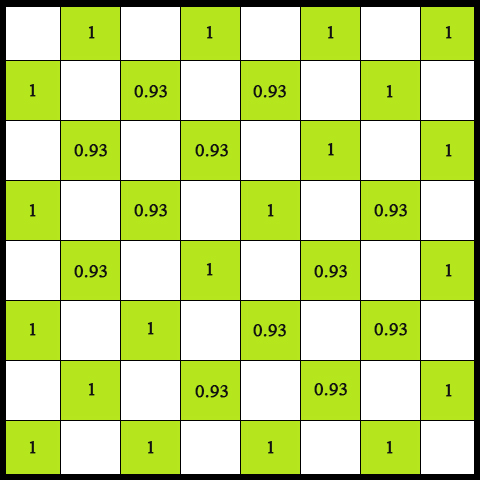
* + 1. **Подсчёт оценки для дамок**

Рисунок 3.2.2 – Оценка позиции для дамки

Хорошей позицией считается край доски и центральная диагональ, поскольку при такой позиции дамку поймать сложно или невозможно. Плохой позицией – все остальные.

* + 1. **Подсчёт итоговой оценки позиции**

Итоговая оценка считается по формуле:

Оценка = Б – Ч + 5 \* (Бд - Чд)

Б – сумма оценок белых шашек;

Ч – сумма оценок черных шашек;

Бд – сумма оценок белых дамок;

Чд – сумма оценок чёрных дамок.

Дамка считается в 5 раз ценнее обычной шашки, поэтому движок будет стараться создать максимальное количество дамок.

Если нет ни одной белой шашки или дамки на доске, то алгоритм выдает оценку = -100, что означает победу черных. Соответственно, при отсутствии черных шашек или дамок оценка будет = 100.

* 1. **Добавление ходов**

Так как при выполении хода высота дерева уменьшается, чтобы в какой-то момент дерево не стало нулевой высоты, существует алгоритм добавления ходов. Суть алгоритма:

1. Пройти в конец дерева;
2. Достроить для каждого конечного элемента ходы вглубь;
3. Построить все возможные сбивания.

Алгоритм необходимо проделать определенное конечное число раз, чтобы количество узлов в дереве соответствовало глубине просчета ходов в следующей таблице:

Таблица 3.3 – Зависимоть количества узлов от глубины

|  |  |
| --- | --- |
| Глубина просчёта | Количество узлов |
| 1 | >1 |
| 2 | >8 |
| 3 | >69 |
| 4 | >575 |
| 5 | >4787 |
| 6 | >39818 |
| 7 | >331203 |

Зависимость количества узлов от глубины определяется по формуле:

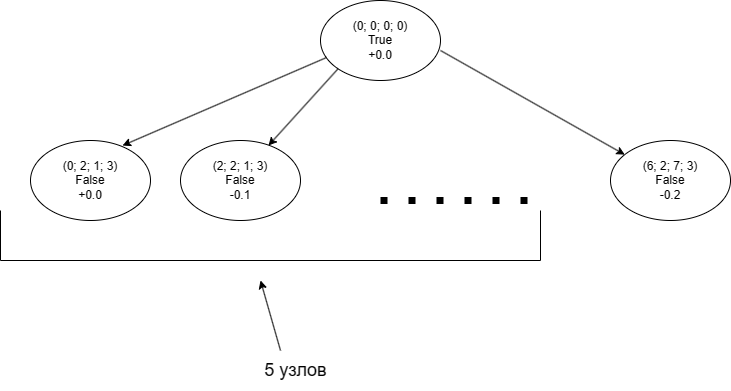
Число 8.31794 было получено на практике как среднее количество возможных ходов. Количество узлов растёт экспоненциально, поэтому в данной курсовой работе не будет рассматриваться глубина просчёта более 7 ходов.

* 1. **Оптимизация перебора**

Полный перебор всех возможных долгий по времени и вызывает большие затраты оперативной памяти. Необходимо ввести оптимизацию перебора. Был использова следующий принцип:

1. В самом начале строится полное дерево ходов
2. Сортировка по оценке

…



1. Добавление узлов только в 5 самых лучших
2. Сортировка первых 5 ходов

Это позволяет избегать полного просчёта заведомо проигрышных ходов.

1. **СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА**
   1. **Проектирование модулей программного средства**

Разработка программного средства выполнялась на основе алгоритмов, представленных в разделе 3, а также на основе спецификации функциональных требований. Полный текст подпрограмм данных модулей представлен в Прило­жении Б.

В программном средстве используется 5 модулей:

* Модуль Board – содержит класс Board, который хранит данные об игровой доске.
* Модуль PossibleMoves – сожержит класс PossibleMoves, который строит возможные ходы для игрока на основе текущей игровой доски.
* Модуль Tree – сожержит класс Tree, который строит дерево ходов исходя из ходов, сгенерированных PossibleMoves.
* Модуль Engine – содержит класс Engine, который работает с деревом ходов Tree и является внешней оболочкой движка.
* Модуль Source – содержит логику взаимодействия программы с пользова­телем.
  1. **Разработка модулей программного средства**

Основные подпрограммы, использованные в разработке форм, перечислены в таблицах 4.2.1-4.2.2:

Таблица 4.2.1 – Основные подпрограммы модуля Board

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрог-раммы** | **Описание** | **Заголовок подпрограммы** | **Имя парамет-ра** | **Назначение параметра** |
| NTB | Определение необходимости сбивать шашку на текущем ходу. | bool NTB(bool turn); | turn | Если ход белых, то turn – true, иначе – false. |
| Move | Ход обычной шашкой или дамкой. | void Move(mytype x1, mytype y1, mytype x2, mytype y2); | x1, y1  x2, y2 | Начальные координаты шашки  Конечные координаты шашки |

Продолжение таблицы 4.2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Beat | Сбивание шашкой | void Beat(mytype x1, mytype y1, mytype x2, mytype y2); | x1, y1  x2, y2 | Начальные координаты шашки  Конечные координаты шашки |
| FillAsses() | Оценка позиции игрока. Зависит от количества шашек и их расположения. | float FillAsses(); |  |  |

Таблица 4.2.2 – Основные подпрограммы модуля PossibleMoves

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FillMoves() | Заполняет массив ходов возможными ходами | void FillMoves(); |  |  |
| FillBeatsForOne() | Заполняет массив ходов всеми возможными сбиваниями для конкретной шашки или дамки | void FillBeatsForOne(mytype x, mytype y, mytype mode); | x, y  mode | Начальные координаты  Вектор предыдущего сбивания |
| FillDamkaBeatsAfterBeat() | Заполняет массив ходов всеми возможными повторными сбиваниями для конкретной дамки | void FillDamkaBeatsAfterBeat(mytype x, mytype y, mytype mode) | x, y  mode | Начальные координаты  Вектор предыдущего сбивания |

Таблица 4.2.3 – Основные подпрограммы модуля Tree

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FillMoves() | Построение дерева ходов с глубиной depth | void FillMoves(mytype depth); | depth | Глубина построения дерева. |
| addMoves() | Добавление элементов в дерево вглубь. | void addMoves(mytype amount); | amount | Глубина добавления новых элементов в дерево |
| sort() | Сортировка потомков корневого узла по оценке позиций. | void sort(); |  |  |

Продолжение таблицы 4.2.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| exists() | Проверка существования хода по координатам  ((x1; y1); (x2; y2)) | mytype exsists(mytype x1, mytype y1, mytype x2, mytype y2); | x1, y1  x2, y2 | Начальные координаты шашки  Конечные координаты шашки |

Таблица 4.2.4 – Основные подпрограммы модуля Engine

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PlayerMove() | Ход игрока по координатам  ((x1; y1); (x2; y2)) | mytype PlayerMove(mytype x1, mytype y1, mytype x2, mytype y2); | x1, y1  x2, y2 | Начальные координаты шашки  Конечные координаты шашки |
| EngineMove() | Ход компьютера | mytype EngineMove(); |  |  |

Таблица 4.2.5 – Основные подпрограммы модуля Source

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| StartScreen() | Отрисовка стартового экрана | void StartScreen(RenderWindow& win, Font& font, bool\* turn, mytype\* depth); | win  font  turn  depth | Окно для отрисовки  Шрифт для отрисовки текста  Цвет шашек игрока  Глубина просчета ходов |

Продолжение таблицы 4.2.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MainScreen() | Основной экран игры | void MainScreen(RenderWindow& win, Font& font, bool turn, mytype depth); | win  font  turn  depth | Окно для отрисовки  Шрифт для отрисовки текста  Цвет шашек игрока  Глубина просчета ходов |
| DrawAll() | Отрисовка доски и шкалы оценки | void DrawAll(RenderWindow& win, Engine& Vobla, Font& font, bool turn) | win  Vobla  font  turn | Окно для отрисовки  Игровой движок  Шрифт для отрисовки текста  Цвет шашек игрока |

* 1. **Проектирование основных структур данных**

Таблица 4.3.1 – Основные структуры данных

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Структура** |
| TField | Массив, который хранит игровую доску | typedef mytype TField[8][8]; |
| Coord | Массив, который хранит координаты хода | typedef mytype Coord[4]; |
| AllMoves | Массив, хранящий все возможные ходы для текущей доски | mytype\*\* AllMoves;  mytype lenMov;  mytype SIZE;  bool turn;  Board MainBoard; |
| Tree | Дерево, содержащее данные о ходе, который к нему привел, и о дочерних ходах, которые возможны | Board MainBoard;  bool turn;  float asses;  Tree\*\* children;  mytype childCount;  Coord moves; |

1. **ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**
   1. **Тестирование и проверка работоспособности программного средства**

Тестирование – важный этап разработки, поскольку при написании программного кода неизбежны. Тестирование – сравнение реального и ожидаемого поведения программы. Основная цель – выявление несоответствия функционирования программы спецификации и устранение их для повышения качества программного средства. Проводилось функциональное тестирование программы.

Функциональные тесты, проведенные над программой, представлены в таблице 5.1.

Таблица 5.1 – Результаты функционального тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Но-мер  теста | Тестируе-мая функцио-нальность | Последовательность действий | Ожидаемый результат | Получен-ный результат |
| 1 | Запуск приложения | 1) Нажатие на ярлык “Vobla” | Приложение запустилось | Тест пройден |
| 2 | Выбор сложности “impossible” и чёрного цвета | 1) Выбор в меню цвета “Black”  2) Выбор в меню сложности “impossible”  3) Нажатие кнопки “Start game” | Игра запущена, выбран чёрный цвет и сложность “impossible”. Доска отображается в перевёрнутом виде. | Тест пройден |
| 3 | Ход шашкой | 1) Нажатие на клетку доски с шашкой  2) Перенесение курсора мыши на желаемый ход  3) Отпускание курсора мыши | Успешный ход | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | Невозможный ход шашкой | Выполнение теста 3, но отпустить мышку на заведомо невозможной для хода клетке | Ход не произошёл | Тест пройден |
| 5 | Проверка ничьей | Сведение партии к ничьей | Завершение партии и высвечивание текста “Draw” | Тест пройден |
| 6 | Проверка победы | Победа в партии | Завершение партии и высвечивание текста “You win” | Тест пройден |
| 7 | Проверка поражения в партии | Поражение в партии | Завершение партии и высвечивание текста “You lose” | Тест пройден |
| 8 | Начало новой партии во время текущей | 1) Нажатие кнопки “Exit to main menu”  2) Выбор цвета и сложности  3) Нажатие кнопки “Start game” | Предыдущая партия завершена, новая партия начата. | Тест пройден |

* 1. **Анализ полученных результатов**

Исходя из полученных результатов в разделе 5.1 можно отметить, что движок не выдал ошибок, связанных с ошибками памяти, ее переполнением или зависанием программы. Насколько точно движок совершает ходы сказать сложно, приведу лишь мою статистику в играх против него для каждой сложности (В/Н/П):

* Easy: 2/3/6
* Medium: 1/2/8
* Hard: 0/0/1
* Impossible: 0/2/19

Ошибок, связанных с интерфейсом, отрисовкой доски или оценки не обнаружено. Единственный минус данного движка в том, что он плохо умеет побеждать. Если нет гарантированной победы за n-ное количество ходов, движок старается сделать максимальное количество дамок для себя, а не минимальное для противника. Из-за этого случается взятие центральной диагонали игроком, которое впоследствии превращается в ничью по правилу 15 ходов без взятий.

1. **РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ**

Для того, чтобы начать использовать программное средство, необходимо запустить файл Vobla.exe.

После открытия программы появляется окно, которое представлено на рисунке 6.1. В нём предстоит пользователю выбор, за какой цвет играть и против какой сложности.

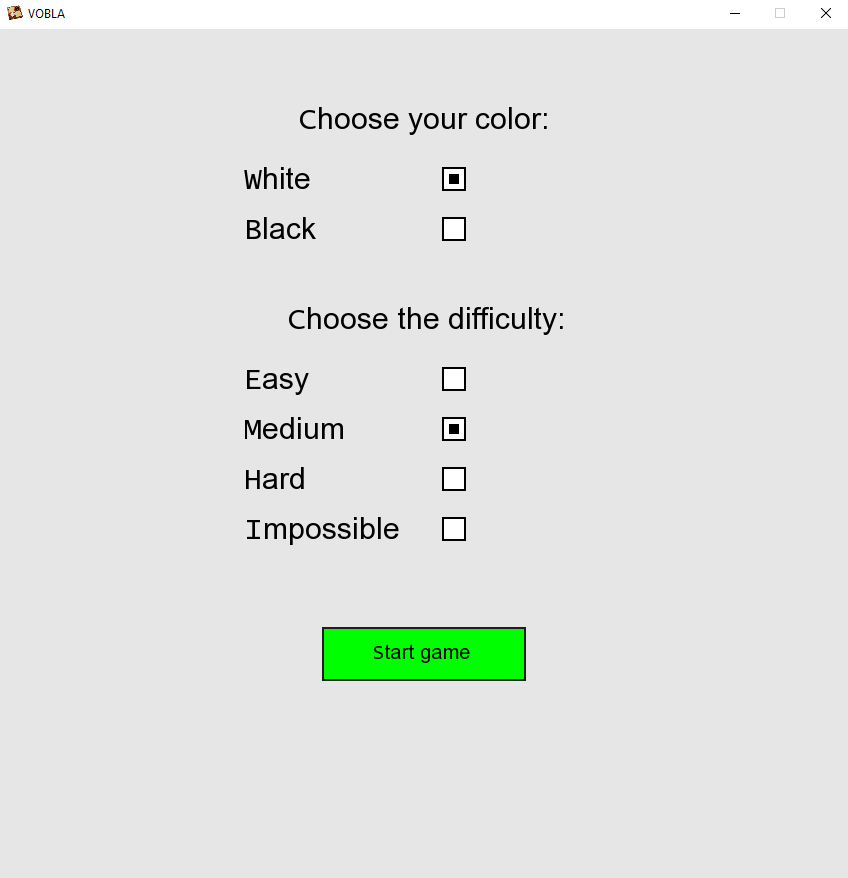
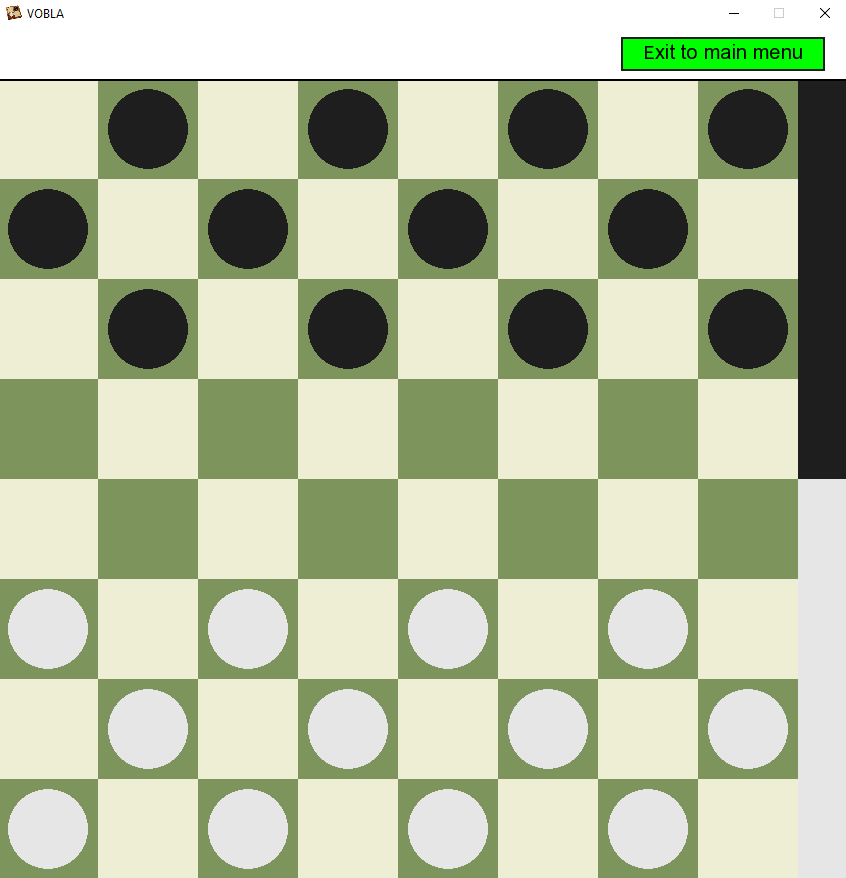


Рисунок 6.1 – Интерфейс программы

Для начала игры необходимо нажать на кнопку «Start game». После появится надпись «Wait a bit», и необходимо немного подождать.

Откроется основной экран игры:

 Рисунок 6.2 – Основной экран игры

Для совершения хода необходимо нажать на шашку, перенести курсор мыши на нужную клетку и отпустить. Справа от доски есть шкала оценки, которая показывает, насколько ваша позиция превосходит позицию соперника или наоборот уступает ей.

Для выхода в главное меню необходимо нажать на кнопку «Exit to main menu», после чего откроется меню из рисунка 6.1.

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсового проекта была проанализирована предметная область, рассмотрены существующие аналоги и выявлены их преимущества и недостатки. В качестве языка разработки использовался Visual C++ с подключением библиотеки SFML.

На этапе проектирования были разработаны блок-схемы алгоритмов. В соответствии с поставленной целью было разработано программное средство "Игровой движок для русских шашек".

В ходе курсового проекта мною было изучено дерево как структура данных, алгоритмы работы с ним. Из-за экспоненциального роста количества элементов в дереве, необходимо было использовать альфа- и бета-отсечение плохих ходов. Однако, поскольку мой опыт игры в шашки является неблоьшим, возникли сложности с оптимизацией дерева перебора. Русские шашки динамичны: отдав две шашки можно забрать три.

Был проведён матч между движком «Аврора» (разработчик Александр Свирин), который занял 5-ое место на чемпионате мира и данным программным средством, созданным в рамках курсовой работы 3-го семестра (далее – «Vobla»). Матч закончился достойным поражением «Vobla» на тридцать третьем ходу, что является весьма впечатлительным успехом. На вычисления «Vobla» потратил в два раза меньше времени и в 8 раз меньше нагружал процессор.

«Vobla» не имеет предела совершенству. Некоторые алгоритмы можно прописать на языке ассемблера, улучшить оптимизацию дерева перебора, прописать дебюты и эндшпили, прописать построение треугольника Петрова. Тогда можно будет всерьёз считать «Vobla» полноценным аналогом «Stockfish» для русских шашек.

# **Список использованной литературы**

[1] Stockfish. [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Stockfish.

[2] chess.com [Электронный ресурс] – Режим доступа:

https://www.chess.com/terms/stockfish-chess-engine.

[3] Русские шашки (википедия). [Электронный ресурс] – Режим доступа:

<https://en.wikipedia.org/wiki/Russian_draughts>

[4] Правила русских шашек. [Электронный ресурс] – Режим доступа:

<https://shashki.ru/variations/draughts64>

[5] Серебряная, Л. В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск: БГУИР, 2013. – 51 с.

[6] Документация библиотеки SFML. [Электронный ресурс] – Режим доступа:

<https://www.sfml-dev.org/learn.php>

# [7] Шахматные алгоритмы, которые думают почти так же, как человек, только лучше. [Электронный ресурс] – Режим доступа: https://habr.com/ru/companies/skillfactory/articles/544040

# **ПРИЛОЖЕНИЕ А**



Рисунок A.1 – Блок-схема программы



Рисунок А.2 – Ход компьютера



Рисунок А.3 – Ход игрока



Рисунок А.4 – Построение дерева ходов



Рисунок А.5 – Построение дерева сбиваний

# **ПРИЛОЖЕНИЕ Б**

#pragma once //Заголовочный файл “Board.h”

#include <iostream>

#include <SFML/Graphics.hpp>

#define mytype short int

typedef mytype Coord[4];

typedef mytype TField[8][8];

mytype GetMode(mytype x1, mytype y1, mytype x2, mytype y2, mytype mode);

bool CheckCoord(mytype x, mytype y);

class Board {

private:

void FillZeros(mytype x1, mytype y1, mytype x2, mytype y2);

void PossibleDamka();

bool NTBDamka(mytype x, mytype y, bool turn, mytype mode);

bool NTBSimplePiece(mytype x, mytype y, bool turn);

float CountPieces(mytype Number);

public:

TField Field;

Board();

Board(const TField Field);

Board(Board& other);

bool NTBDamkaOneMore(mytype x, mytype y, bool turn, mytype mode);

bool NTB(bool turn);

void Move(mytype x1, mytype y1, mytype x2, mytype y2);

void Beat(mytype x1, mytype y1, mytype x2, mytype y2);

void DamkaBeat(mytype x1, mytype y1, mytype x2, mytype y2, mytype mode);

mytype NumberOfPieces();

float FillAsses();

};

#include "Board.h" //Файл “Board.cpp”

mytype GetMode(mytype x1, mytype y1, mytype x2, mytype y2, mytype mode) {

mytype result = 0;

if (mode == 0) {

if (y1 > y2) {

if (x1 > x2) {

result = 4;

}

else {

result = 3;

}

}

else {

if (x1 > x2) {

result = 2;

}

else {

result = 1;

}

}

}

else if ((mode == 1) || (mode == 4)) {

if ((x1 - y1) > (x2 - y2)) {

result = 2;

}

else if ((x1 - y1) < (x2 - y2)) {

result = 3;

}

else {

result = mode;

}

}

else if ((mode == 2) || (mode == 3)) {

if ((x1 + y1) > (x2 + y2)) {

result = 4;

}

else if ((x1 + y1) < (x2 + y2)) {

result = 1;

}

else {

result = mode;

}

}

return result;

}

bool CheckCoord(mytype x, mytype y) {

return (x >= 0) && (y >= 0) && (x < 8) && (y < 8);

}

float AssesDamka[8][8] = {

{1, 0, 1, 0, 1, 0, 1, 0},

{0, 1, 0, 0.93, 0, 0.93, 0, 1},

{1, 0, 1, 0, 0.93, 0, 0.93, 0},

{0, 0.93, 0, 1, 0, 1, 0, 1},

{1, 0, 1, 0, 1, 0, 0.93, 0},

{0, 0.93, 0, 0.93, 0, 1, 0, 1},

{1, 0, 0.93, 0, 0.93, 0, 1, 0},

{0, 1, 0, 1, 0, 1, 0, 1}

};

float AssesSimple[8][8] = {

{0.95, 0, 1, 0, 1, 0, 0.95, 0},

{0, 0.93, 0, 0.95, 0, 0.95, 0, 0.93},

{0.93, 0, 0.95, 0, 0.95, 0, 0.93, 0},

{0, 0.95, 0, 1, 0, 1, 0, 0.93},

{0.93, 0, 1, 0, 1, 0, 0.95, 0},

{0, 0.93, 0, 0.95, 0, 0.95, 0, 0.93},

{0.93, 0, 0.95, 0, 0.95, 0, 0.93, 0},

{0, 0.95, 0, 1, 0, 1, 0, 0.95}

};

void Board::FillZeros(mytype x1, mytype y1, mytype x2, mytype y2) {

if (x1 > x2) {

if (y1 > y2) {

for (mytype i = x1; i > x2; i--) {

Field[i][y1 + i - x1] = 0;

}

}

else {

for (mytype i = x1; i > x2; i--) {

Field[i][y1 - i + x1] = 0;

}

}

}

else {

if (y1 > y2) {

for (mytype i = x1; i < x2; i++) {

Field[i][y1 - i + x1] = 0;

}

}

else {

for (mytype i = x1; i < x2; i++) {

Field[i][y1 + i - x1] = 0;

}

}

}

}

void Board::PossibleDamka()

{

for (mytype i = 0; i < 8; i++) {

if (Field[i][7] == 1)

{

Field[i][7] = 3;

}

if (Field[i][0] == 2)

{

Field[i][0] = 4;

}

}

}

bool Board::NTBDamka(mytype x, mytype y, bool turn, mytype mode) {

mytype temp = 1;

if (turn) {

temp = 2;

}

mytype x0 = x;

mytype y0 = y;

if (mode != 4) {

while (CheckCoord(++x0, ++y0)) {

if (Field[x0][y0] != 0) {

break;

}

}

if ((x0 < 7) && (y0 < 7)) {

if (((Field[x0][y0] == temp) || (Field[x0][y0] == temp + 2)) && (Field[x0 + 1][y0 + 1] == 0)) {

return true;

}

}

}

if (mode != 3) {

x0 = x;

y0 = y;

while (CheckCoord(--x0, ++y0)) {

if (Field[x0][y0] != 0) {

break;

}

}

if ((x0 > 0) && (y0 < 7)) {

if (((Field[x0][y0] == temp) || (Field[x0][y0] == temp + 2)) && (Field[x0 - 1][y0 + 1] == 0)) {

return true;

}

}

}

if (mode != 2) {

x0 = x;

y0 = y;

while (CheckCoord(++x0, --y0)) {

if (Field[x0][y0] != 0) {

break;

}

}

if ((x0 < 7) && (y0 > 0)) {

if (((Field[x0][y0] == temp) || (Field[x0][y0] == temp + 2)) && (Field[x0 + 1][y0 - 1] == 0)) {

return true;

}

}

}

if (mode != 1) {

x0 = x;

y0 = y;

while (CheckCoord(--x0, --y0)) {

if (Field[x0][y0] != 0) {

break;

}

}

if ((x0 > 0) && (y0 > 0)) {

if (((Field[x0][y0] == temp) || (Field[x0][y0] == temp + 2)) && (Field[x0 - 1][y0 - 1] == 0)) {

return true;

}

}

}

return false;

}

bool Board::NTBSimplePiece(mytype x, mytype y, bool turn) {

mytype temp = 1;

if (turn) {

temp = 2;

}

if ((x > 1) && (y > 1)) {

if (((Field[x - 1][y - 1] == temp) || (Field[x - 1][y - 1] == temp + 2)) && (Field[x - 2][y - 2] == 0)) {

return true;

}

}

if ((x < 6) && (y > 1)) {

if (((Field[x + 1][y - 1] == temp) || (Field[x + 1][y - 1] == temp + 2)) && (Field[x + 2][y - 2] == 0)) {

return true;

}

}

if ((x > 1) && (y < 6)) {

if (((Field[x - 1][y + 1] == temp) || (Field[x - 1][y + 1] == temp + 2)) && (Field[x - 2][y + 2] == 0)) {

return true;

}

}

if ((x < 6) && (y < 6)) {

if (((Field[x + 1][y + 1] == temp) || (Field[x + 1][y + 1] == temp + 2)) && (Field[x + 2][y + 2] == 0)) {

return true;

}

}

return false;

}

float Board::CountPieces(mytype Number)

{

float result = 0;

for (mytype i = 0; i < 8; i++)

{

for (mytype j = 0; j < 8; j++)

{

if (Field[i][j] == Number)

{

if ((Number == 1) || (Number == 2)) {

result += AssesSimple[i][j];

}

else {

result += AssesDamka[i][j];

}

}

}

}

return result;

}

TField InitialBoard = {

{1, 0, 1, 0, 0, 0, 2, 0},

{0, 1, 0, 0, 0, 2, 0, 2},

{1, 0, 1, 0, 0, 0, 2, 0},

{0, 1, 0, 0, 0, 2, 0, 2},

{1, 0, 1, 0, 0, 0, 2, 0},

{0, 1, 0, 0, 0, 2, 0, 2},

{1, 0, 1, 0, 0, 0, 2, 0},

{0, 1, 0, 0, 0, 2, 0, 2}

};

Board::Board() {

for (mytype i = 0; i < 8; i++) {

for (mytype j = 0; j < 8; j++) {

Field[i][j] = InitialBoard[i][j];

}

}

};

Board::Board(const TField Field) {

for (mytype i = 0; i < 8; i++) {

for (mytype j = 0; j < 8; j++) {

this->Field[i][j] = Field[i][j];

}

}

}

Board::Board(Board& other) {

for (mytype i = 0; i < 8; i++) {

for (mytype j = 0; j < 8; j++) {

this->Field[i][j] = other.Field[i][j];

}

}

}

bool Board::NTBDamkaOneMore(mytype x, mytype y, bool turn, mytype mode) {

if (NTBDamka(x, y, turn, mode)) {

return true;

}

if (mode == 1) {

x++;

y++;

while (CheckCoord(x, y) && (Field[x][y] == 0)) {

if (NTBDamka(x++, y++, turn, mode)) {

return true;

}

}

}

if (mode == 2) {

x--;

y++;

while (CheckCoord(x, y) && (Field[x][y] == 0)) {

if (NTBDamka(x--, y++, turn, mode)) {

return true;

}

}

}

if (mode == 3) {

x++;

y--;

while (CheckCoord(x, y) && (Field[x][y] == 0)) {

if (NTBDamka(x++, y--, turn, mode)) {

return true;

}

}

}

if (mode == 4) {

x--;

y--;

while (CheckCoord(x, y) && (Field[x][y] == 0)) {

if (NTBDamka(x--, y--, turn, mode)) {

return true;

}

}

}

return false;

}

bool Board::NTB(bool turn) {

mytype temp = 2;

if (turn) {

temp = 1;

}

for (mytype i = 0; i < 8; i++) {

for (mytype j = 0; j < 8; j++) {

if (Field[i][j] == temp) {

if (NTBSimplePiece(i, j, turn)) {

return true;

}

}

else if (Field[i][j] == temp + 2) {

if (NTBDamka(i, j, turn, 0)) {

return true;

}

}

}

}

return false;

}

void Board::Move(mytype x1, mytype y1, mytype x2, mytype y2) {

Field[x2][y2] = Field[x1][y1];

Field[x1][y1] = 0;

PossibleDamka();

}

void Board::Beat(mytype x1, mytype y1, mytype x2, mytype y2) {

Field[x2][y2] = Field[x1][y1];

Field[x1][y1] = 0;

Field[(x1 + x2) / 2][(y1 + y2) / 2] = 0;

PossibleDamka();

}

void Board::DamkaBeat(mytype x1, mytype y1, mytype x2, mytype y2, mytype mode) {

Field[x2][y2] = Field[x1][y1];

if (mode == 1) {

while ((x1 + y1) < (x2 + y2)) {

Field[x1++][y1++] = 0;

}

}

if (mode == 2) {

while ((x1 - y1) > (x2 - y2)) {

Field[x1--][y1++] = 0;

}

}

if (mode == 3) {

while ((x1 - y1) < (x2 - y2)) {

Field[x1++][y1--] = 0;

}

}

if (mode == 4) {

while ((x1 + y1) > (x2 + y2)) {

Field[x1--][y1--] = 0;

}

}

FillZeros(x1, y1, x2, y2);

}

float Board::FillAsses() {

float asses;

if ((CountPieces(1) + CountPieces(3) < 0.01)) {

asses = -100;

}

else if (CountPieces(2) + CountPieces(4) < 0.01) {

asses = 100;

}

else {

asses = CountPieces(1) - CountPieces(2) + 5 \* (CountPieces(3) - CountPieces(4));

}

return asses;

}

mytype Board::NumberOfPieces() {

mytype result = 0;

for (mytype i = 0; i < 8; i++) {

for (mytype j = 0; j < 8; j++) {

if (Field[i][j] != 0) {

result++;

}

}

}

return result;

}

#pragma once //Заголовочный файл “PossibleMoves.h”

#include "Board.h"

class PossibleMoves {

private:

mytype\*\* AllMoves;

mytype lenMov;

mytype SIZE;

bool turn;

Board MainBoard;

void FillDamkaBeatsDiag(mytype x0, mytype y0, mytype x, mytype y, mytype mode);

void FillDamkaBeatsForOne(mytype x, mytype y, mytype mode);

void FillDamkaMoves(mytype x, mytype y);

bool SimpleMoveCheck(mytype x1, mytype y1, mytype x2, mytype y2, bool turn);

bool SimpleBeatCheck(mytype x1, mytype y1, mytype x2, mytype y2, bool turn);

public:

PossibleMoves(const Board MainBoard, bool turn);

~PossibleMoves();

void Renew(const TField Field);

void Add(mytype x1, mytype y1, mytype x2, mytype y2);

mytype len();

mytype GetMoveX1(mytype index);

mytype GetMoveY1(mytype index);

mytype GetMoveX2(mytype index);

mytype GetMoveY2(mytype index);

void Reset();

void FillDamkaBeatsAfterBeat(mytype x, mytype y, mytype mode);

void FillBeatsForOne(mytype x, mytype y, mytype mode);

void FillMoves();

};

#include "PossibleMoves.h" //Файл “PossibleMoves.cpp”

#include "math.h"

void PossibleMoves::FillDamkaBeatsDiag(mytype x0, mytype y0, mytype x, mytype y, mytype mode) {

mytype temp = 1;

if (turn) {

temp = 2;

}

if (mode == 1) {

while (CheckCoord(++x, ++y)) {

if (MainBoard.Field[x][y] != 0) {

break;

}

}

if (CheckCoord(x + 1, y + 1)) {

if ((MainBoard.Field[x + 1][y + 1] == 0) && ((MainBoard.Field[x][y] == temp) || (MainBoard.Field[x][y] == temp + 2))) {

this->Add(x0, y0, ++x, ++y);

Board TempBoard = MainBoard;

TempBoard.DamkaBeat(x0, y0, x, y, 1);

if (!TempBoard.NTBDamkaOneMore(x++, y++, turn, 1)) {

while (CheckCoord(x, y) && (TempBoard.Field[x][y] == 0)) {

this->Add(x0, y0, x++, y++);

}

}

}

}

}

if (mode == 2) {

while (CheckCoord(--x, ++y)) {

if (MainBoard.Field[x][y] != 0) {

break;

}

}

if (CheckCoord(x - 1, y + 1)) {

if ((MainBoard.Field[x - 1][y + 1] == 0) && ((MainBoard.Field[x][y] == temp) || (MainBoard.Field[x][y] == temp + 2))) {

this->Add(x0, y0, --x, ++y);

Board TempBoard = MainBoard;

TempBoard.DamkaBeat(x0, y0, x, y, 2);

if (!TempBoard.NTBDamkaOneMore(x--, y++, turn, 2)) {

while (CheckCoord(x, y) && (TempBoard.Field[x][y] == 0)) {

this->Add(x0, y0, x--, y++);

}

}

}

}

}

if (mode == 3) {

while (CheckCoord(++x, --y)) {

if (MainBoard.Field[x][y] != 0) {

break;

}

}

if (CheckCoord(x + 1, y - 1)) {

if ((MainBoard.Field[x + 1][y - 1] == 0) && ((MainBoard.Field[x][y] == temp) || (MainBoard.Field[x][y] == temp + 2))) {

this->Add(x0, y0, ++x, --y);

Board TempBoard = MainBoard;

TempBoard.DamkaBeat(x0, y0, x, y, 3);

if (!TempBoard.NTBDamkaOneMore(x++, y--, turn, 3)) {

while (CheckCoord(x, y) && (TempBoard.Field[x][y] == 0)) {

this->Add(x0, y0, x++, y--);

}

}

}

}

}

if (mode == 4) {

while (CheckCoord(--x, --y)) {

if (MainBoard.Field[x][y] != 0) {

break;

}

}

if (CheckCoord(x - 1, y - 1)) {

if ((MainBoard.Field[x - 1][y - 1] == 0) && ((MainBoard.Field[x][y] == temp) || (MainBoard.Field[x][y] == temp + 2))) {

this->Add(x0, y0, --x, --y);

Board TempBoard = MainBoard;

TempBoard.DamkaBeat(x0, y0, x, y, 4);

if (!TempBoard.NTBDamkaOneMore(x--, y--, turn, 4)) {

while (CheckCoord(x, y) && (TempBoard.Field[x][y] == 0)) {

this->Add(x0, y0, x--, y--);

}

}

}

}

}

}

void PossibleMoves::FillDamkaBeatsForOne(mytype x, mytype y, mytype mode) {

if (mode != 4) {

FillDamkaBeatsDiag(x, y, x, y, 1);

}

if (mode != 3) {

FillDamkaBeatsDiag(x, y, x, y, 2);

}

if (mode != 2) {

FillDamkaBeatsDiag(x, y, x, y, 3);

}

if (mode != 1) {

FillDamkaBeatsDiag(x, y, x, y, 4);

}

}

void PossibleMoves::FillDamkaMoves(mytype x, mytype y) {

mytype x0 = x;

mytype y0 = y;

while (CheckCoord(++x, ++y) && (MainBoard.Field[x][y] == 0)) {

this->Add(x0, y0, x, y);

}

x = x0;

y = y0;

while (CheckCoord(--x, ++y) && (MainBoard.Field[x][y] == 0)) {

this->Add(x0, y0, x, y);

}

x = x0;

y = y0;

while (CheckCoord(++x, --y) && (MainBoard.Field[x][y] == 0)) {

this->Add(x0, y0, x, y);

}

x = x0;

y = y0;

while (CheckCoord(--x, --y) && (MainBoard.Field[x][y] == 0)) {

this->Add(x0, y0, x, y);

}

}

bool PossibleMoves::SimpleMoveCheck(mytype x1, mytype y1, mytype x2, mytype y2, bool turn) {

mytype temp = 2;

mytype dy = -1;

if (turn) {

temp = 1;

dy = 1;

}

if (CheckCoord(x1, y1) && CheckCoord(x2, y2)) {

if (MainBoard.Field[x1][y1] == temp) {

if (((x2 == x1 + 1) || (x2 == x1 - 1)) && (y2 == y1 + dy)) {

if (MainBoard.Field[x2][y2] == 0) {

return true;

}

}

}

}

return false;

}

bool PossibleMoves::SimpleBeatCheck(mytype x1, mytype y1, mytype x2, mytype y2, bool turn) {

mytype temp = 1;

if (turn) {

temp = 2;

}

if (CheckCoord(x1, y1) && CheckCoord(x2, y2) && (abs(x1 - x2) == abs(y1 - y2)) && (abs(x1 - x2)) == 2) {

if (MainBoard.Field[x1][y1] == 3 - temp) {

mytype x0 = (x1 + x2) / 2;

mytype y0 = (y1 + y2) / 2;

if (((MainBoard.Field[x0][y0] == temp) || (MainBoard.Field[x0][y0] == temp + 2)) && (MainBoard.Field[x2][y2] == 0)) {

return true;

}

}

}

return false;

}

PossibleMoves::PossibleMoves(const Board MainBoard, bool turn) {

this->MainBoard = MainBoard;

SIZE = 10;

lenMov = 0;

AllMoves = new mytype\* [SIZE];

for (mytype i = 0; i < SIZE; i++) {

AllMoves[i] = new mytype[4];

}

this->turn = turn;

}

PossibleMoves::~PossibleMoves() {

for (mytype i = 0; i < SIZE; i++) {

delete[] AllMoves[i];

}

delete[] AllMoves;

}

void PossibleMoves::Renew(const TField Field) {

for (mytype x = 0; x < 8; x++) {

for (mytype y = 0; y < 8; y++) {

this->MainBoard.Field[x][y] = Field[x][y];

}

}

}

void PossibleMoves::Add(mytype x1, mytype y1, mytype x2, mytype y2) {

if (lenMov == SIZE) {

SIZE = SIZE \* 3 / 2 + 1;

mytype\*\* tempArr = new mytype\* [SIZE];

for (mytype i = 0; i < SIZE; i++) {

tempArr[i] = new mytype[4];

}

for (mytype i = 0; i < lenMov; i++) {

for (mytype j = 0; j < 4; j++) {

tempArr[i][j] = AllMoves[i][j];

}

}

for (mytype i = 0; i < lenMov; i++) {

delete[] AllMoves[i];

}

delete[] AllMoves;

AllMoves = tempArr;

}

AllMoves[lenMov][0] = x1;

AllMoves[lenMov][1] = y1;

AllMoves[lenMov][2] = x2;

AllMoves[lenMov][3] = y2;

lenMov++;

}

mytype PossibleMoves::len() {

return lenMov;

}

mytype PossibleMoves::GetMoveX1(mytype index) {

return AllMoves[index][0];

}

mytype PossibleMoves::GetMoveY1(mytype index) {

return AllMoves[index][1];

}

mytype PossibleMoves::GetMoveX2(mytype index) {

return AllMoves[index][2];

}

mytype PossibleMoves::GetMoveY2(mytype index) {

return AllMoves[index][3];

}

void PossibleMoves::Reset() {

lenMov = 0;

}

void PossibleMoves::FillDamkaBeatsAfterBeat(mytype x, mytype y, mytype mode) {

mytype x0 = x;

mytype y0 = y;

if (mode != 1) {

FillDamkaBeatsDiag(x0, y0, x, y, 4);

}

if (mode != 2) {

FillDamkaBeatsDiag(x0, y0, x, y, 3);

}

if (mode != 3) {

FillDamkaBeatsDiag(x0, y0, x, y, 2);

}

if (mode != 4) {

FillDamkaBeatsDiag(x0, y0, x, y, 1);

}

if (mode == 1) {

while (CheckCoord(++x, ++y) && (MainBoard.Field[x][y] == 0)) {

FillDamkaBeatsDiag(x0, y0, x, y, 2);

FillDamkaBeatsDiag(x0, y0, x, y, 3);

}

}

if (mode == 2) {

while (CheckCoord(--x, ++y) && (MainBoard.Field[x][y] == 0)) {

FillDamkaBeatsDiag(x0, y0, x, y, 1);

FillDamkaBeatsDiag(x0, y0, x, y, 4);

}

}

if (mode == 3) {

while (CheckCoord(++x, --y) && (MainBoard.Field[x][y] == 0)) {

FillDamkaBeatsDiag(x0, y0, x, y, 1);

FillDamkaBeatsDiag(x0, y0, x, y, 4);

}

}

if (mode == 4) {

while (CheckCoord(--x, --y) && (MainBoard.Field[x][y] == 0)) {

FillDamkaBeatsDiag(x0, y0, x, y, 2);

FillDamkaBeatsDiag(x0, y0, x, y, 3);

}

}

}

void PossibleMoves::FillBeatsForOne(mytype x, mytype y, mytype mode) {

if (MainBoard.Field[x][y] <= 2) {

if (SimpleBeatCheck(x, y, x + 2, y + 2, turn)) {

this->Add(x, y, x + 2, y + 2);

}

if (SimpleBeatCheck(x, y, x - 2, y + 2, turn)) {

this->Add(x, y, x - 2, y + 2);

}

if (SimpleBeatCheck(x, y, x + 2, y - 2, turn)) {

this->Add(x, y, x + 2, y - 2);

}

if (SimpleBeatCheck(x, y, x - 2, y - 2, turn)) {

this->Add(x, y, x - 2, y - 2);

}

}

else {

FillDamkaBeatsForOne(x, y, mode);

}

}

void PossibleMoves::FillMoves() {

mytype temp = 2;

if (turn) {

temp = 1;

}

if (MainBoard.NTB(turn)) {

for (mytype i = 0; i < 8; i++) {

for (mytype j = 0; j < 8; j++) {

if ((MainBoard.Field[i][j] == temp) || (MainBoard.Field[i][j] == temp + 2)) {

FillBeatsForOne(i, j, 0);

}

}

}

}

else {

mytype dy = -1;

if (turn) {

dy = 1;

}

for (mytype x = 0; x < 8; x++) {

for (mytype y = 0; y < 8; y++) {

if (MainBoard.Field[x][y] == temp) {

if (SimpleMoveCheck(x, y, x + 1, y + dy, turn)) {

this->Add(x, y, x + 1, y + dy);

}

if (SimpleMoveCheck(x, y, x - 1, y + dy, turn)) {

this->Add(x, y, x - 1, y + dy);

}

}

else if (MainBoard.Field[x][y] == temp + 2) {

FillDamkaMoves(x, y);

}

}

}

}

}

#pragma once //Заголовочный файл “Tree.h”

#include "PossibleMoves.h"

class Tree {

private:

mytype GetMasMax();

mytype GetMasMin();

void invert();

void invert(mytype WIDTH);

public:

Board MainBoard;

bool turn;

float asses;

Tree\*\* children;

mytype childCount;

Coord moves;

Tree(Coord arr, Board MainBoard, bool turn);

~Tree();

void FillBeatsForOne(mytype x, mytype y, mytype mode, mytype depth);

void FillMoves(mytype depth);

void addMoves(mytype amount);

void deleteExcept(mytype index);

void addChild(Tree\* child);

int NumberOfNodes();

void sort();

void sort(mytype WIDTH);

float FillAsses();

mytype exsists(mytype x1, mytype y1, mytype x2, mytype y2);

};

#include "Tree.h" //Файл “Tree.cpp”

mytype Tree::GetMasMax() {

float max = children[0]->FillAsses();

float temp1 = 0;

mytype index = 0;

for (mytype i = 1; i < childCount; i++) {

temp1 = children[i]->FillAsses();

if (temp1 > max) {

max = temp1;

index = i;

}

}

return index;

}

mytype Tree::GetMasMin() {

float min = children[0]->FillAsses();

float temp1 = 0;

mytype index = 0;

for (mytype i = 1; i < childCount; i++) {

temp1 = children[i]->FillAsses();

if (temp1 < min) {

min = temp1;

index = i;

}

}

return index;

}

void Tree::invert() {

for (mytype i = 0; i < childCount / 2; i++) {

Tree\* temp = children[i];

children[i] = children[childCount - i - 1];

children[childCount - i - 1] = temp;

}

}

void Tree::invert(mytype WIDTH) {

for (mytype i = 0; i < WIDTH / 2; i++) {

Tree\* temp = children[i];

children[i] = children[WIDTH - i - 1];

children[WIDTH - i - 1] = temp;

}

}

Tree::Tree(Coord arr, Board MainBoard, bool turn) {

for (mytype i = 0; i < 4; i++) {

moves[i] = arr[i];

}

asses = MainBoard.FillAsses();

childCount = 0;

children = nullptr;

this->turn = turn;

this->MainBoard = MainBoard;

}

Tree::~Tree() {

if (children != nullptr) {

for (mytype i = 0; i < childCount; i++) {

if (children[i]) {

delete children[i];

children[i] = nullptr;

}

}

delete[] children;

children = nullptr;

}

}

void Tree::deleteExcept(mytype index) {

for (mytype i = 0; i < childCount; i++) {

if (i != index) {

delete children[i];

children[i] = nullptr;

}

}

}

void Tree::addChild(Tree\* child) {

if (children == nullptr) {

children = new Tree \* [1];

children[0] = child;

childCount = 1;

}

else {

Tree\*\* newChildren = new Tree \* [childCount + 1];

for (mytype i = 0; i < childCount; i++) {

newChildren[i] = children[i];

}

newChildren[childCount] = child;

delete[] children;

children = newChildren;

childCount++;

}

}

int Tree::NumberOfNodes() {

int result = 0;

for (mytype i = 0; i < childCount; i++) {

result += children[i]->NumberOfNodes();

}

return result + 1;

}

void Tree::FillMoves(mytype depth) {

mytype x1, y1, x2, y2;

if (MainBoard.NTB(turn)) {

PossibleMoves PM(MainBoard, turn);

PM.FillMoves();

for (mytype i = 0; i < PM.len(); i++) {

Board oldBoard = MainBoard;

x1 = PM.GetMoveX1(i);

y1 = PM.GetMoveY1(i);

x2 = PM.GetMoveX2(i);

y2 = PM.GetMoveY2(i);

if (oldBoard.Field[x1][y1] <= 2) {

oldBoard.Beat(x1, y1, x2, y2);

}

else {

oldBoard.DamkaBeat(x1, y1, x2, y2, 0);

}

mytype mode = GetMode(x1, y1, x2, y2, 0);

Coord arr = { x1, y1, x2, y2 };

Tree\* NewNode = new Tree(arr, oldBoard, turn);

addChild(NewNode);

NewNode->FillBeatsForOne(x2, y2, mode, depth);

}

}

else if (depth > 0) {

PossibleMoves PM(MainBoard, turn);

PM.FillMoves();

for (mytype i = 0; i < PM.len(); i++) {

Board oldBoard = MainBoard;

x1 = PM.GetMoveX1(i);

y1 = PM.GetMoveY1(i);

x2 = PM.GetMoveX2(i);

y2 = PM.GetMoveY2(i);

oldBoard.Move(x1, y1, x2, y2);

Coord arr = { x1, y1, x2, y2 };

Tree\* NewNode = new Tree(arr, oldBoard, !turn);

addChild(NewNode);

NewNode->FillMoves(depth - 1);

}

}

}

void Tree::FillBeatsForOne(mytype x, mytype y, mytype mode, mytype depth) {

PossibleMoves PM(MainBoard, turn);

mytype x2, y2;

if (MainBoard.Field[x][y] <= 2) {

PM.FillBeatsForOne(x, y, mode);

}

else {

PM.FillDamkaBeatsAfterBeat(x, y, mode);

}

if (PM.len() == 0) {

turn = !turn;

FillMoves(depth - 1);

}

else {

mytype tempmode = mode;

for (mytype i = 0; i < PM.len(); i++) {

Board TempBoard = MainBoard;

x2 = PM.GetMoveX2(i);

y2 = PM.GetMoveY2(i);

if (TempBoard.Field[x][y] <= 2) {

TempBoard.Beat(x, y, x2, y2);

}

else {

TempBoard.DamkaBeat(x, y, x2, y2, mode);

}

tempmode = GetMode(x, y, x2, y2, mode);

Coord arr = { x, y, x2, y2 };

Tree\* NewNode = new Tree(arr, TempBoard, turn);

addChild(NewNode);

NewNode->FillBeatsForOne(x2, y2, tempmode, depth);

}

}

}

void Tree::addMoves(mytype amount) {

if (childCount == 0) {

FillMoves(amount);

}

else {

for (mytype i = 0; i < childCount; i++) {

children[i]->addMoves(amount);

}

}

}

void Tree::sort() {

FillAsses();

bool swapped;

do {

swapped = false;

for (mytype i = 0; i < childCount - 1; i++) {

if (children[i]->asses < children[i + 1]->asses) {

Tree\* temp = children[i];

children[i] = children[i + 1];

children[i + 1] = temp;

swapped = true;

}

}

} while (swapped);

if (!turn) {

invert();

}

}

void Tree::sort(mytype WIDTH) {

for (mytype i = 0; i < WIDTH; i++) {

children[i]->FillAsses();

}

bool swapped;

do {

swapped = false;

for (mytype i = 0; i < WIDTH - 1; i++) {

if (children[i]->asses < children[i + 1]->asses) {

Tree\* temp = children[i];

children[i] = children[i + 1];

children[i + 1] = temp;

swapped = true;

}

}

} while (swapped);

if (!turn) {

invert(WIDTH);

}

}

float Tree::FillAsses() {

if ((childCount == 0)) {

return MainBoard.FillAsses();

}

mytype index;

if (this->turn) {

index = GetMasMax();

}

else {

index = GetMasMin();

}

this->asses = children[index]->asses;

return this->asses;

}

mytype Tree::exsists(mytype x1, mytype y1, mytype x2, mytype y2) {

for (mytype i = 0; i < childCount; i++) {

if ((children[i]->moves[0] == x1) && (children[i]->moves[1] == y1) && (children[i]->moves[2] == x2) && (children[i]->moves[3] == y2)) {

return i;

}

}

return -1;

}

#pragma once //Заголовочный файл “Engine.h”

#include "Tree.h"

class Engine {

private:

mytype depth;

bool turn;

mytype moves;

mytype pieces;

Tree\* firstroot;

Tree\* root;

float GetNumber(mytype depth);

public:

Board MainBoard;

float asses;

Engine(mytype depth);

~Engine();

mytype PlayerMove(mytype x1, mytype y1, mytype x2, mytype y2);

mytype EngineMove();

};

#include "Engine.h" //Файл “Engine.cpp”

#include "math.h"

const mytype WIDTH = 5;

float Engine::GetNumber(mytype depth) {

return exp((depth - 1) \* log(8.31794));

}

Engine::Engine(mytype depth) {

moves = 0;

pieces = 24;

turn = true;

asses = 0.0;

Coord arr = { 0 };

root = new Tree(arr, MainBoard, turn);

firstroot = root;

this->depth = depth;

root->FillMoves(depth);

}

Engine::~Engine() {

delete firstroot;

}

mytype Engine::PlayerMove(mytype x1, mytype y1, mytype x2, mytype y2) {

if (moves == 15) {

return 0;

}

mytype index = root->exsists(x1, y1, x2, y2);

if (index == -1) {

return -1;

}

root->deleteExcept(index);

root = root->children[index];

MainBoard = root->MainBoard;

asses = root->asses;

if ((root->childCount != 0) && (root->turn == this->turn)) {

return -1;

}

turn = !turn;

return 1;

}

mytype Engine::EngineMove() {

if (root->childCount == 0) {

return -1;

}

root->sort();

mytype temp = WIDTH;

if (temp > root->childCount) {

temp = root->childCount;

}

for (mytype i = 0; i < 2 \* depth; i++) {

if (root->NumberOfNodes() > GetNumber(depth)) {

break;

}

for (mytype j = 0; j < temp; j++) {

root->children[j]->addMoves(1);

}

}

root->sort(temp);

asses = root->asses;

while ((root->childCount != 0) && (root->turn == this->turn)) {

root->deleteExcept(0);

root = root->children[0];

MainBoard = root->MainBoard;

root->sort();

}

if (MainBoard.NumberOfPieces() != pieces) {

moves = 0;

pieces = MainBoard.NumberOfPieces();

}

else {

moves++;

}

if (root->childCount == 0) {

return 1;

}

turn = !turn;

return 0;

}

#include "Engine.h" //Файл “Source.cpp”

using namespace sf;

const int menuH = 50;

const int tileSize = 100;

class TChoise {

public:

RectangleShape field;

RectangleShape in;

bool isSelected;

TChoise(int x0, int y0) {

field.setSize(Vector2f(20, 20));

field.setFillColor(Color::White);

field.setOutlineColor(Color::Black);

field.setOutlineThickness(2);

field.setPosition(x0, y0);

isSelected = false;

in.setSize(Vector2f(10, 10));

in.setFillColor(Color::Black);

in.setPosition(x0 + 5, y0 + 5);

}

};

void drawCheckerboard(RenderWindow& win, const mytype board[8][8]) {

for (mytype i = 0; i < 8; i++) {

for (mytype j = 0; j < 8; j++) {

RectangleShape cell(Vector2f(tileSize, tileSize));

if ((i + j) % 2 == 1) {

cell.setFillColor(Color(238, 238, 213));

}

else {

cell.setFillColor(Color(125, 148, 93));

}

cell.setPosition(i \* tileSize, (7 - j) \* tileSize + menuH);

win.draw(cell);

CircleShape checker(tileSize / 2 - 10);

checker.setPosition(i \* tileSize + 10, (7 - j) \* tileSize + 10 + menuH);

if ((board[i][j] == 1) || (board[i][j] == 3)) {

checker.setFillColor(Color(230, 230, 230));

}

else if ((board[i][j] == 2) || (board[i][j] == 4)) {

checker.setFillColor(Color(30, 30, 30));

}

if (board[i][j] != 0) {

win.draw(checker);

}

if (board[i][j] == 3) {

CircleShape in(tileSize / 4 - 5);

in.setPosition(i \* tileSize + tileSize / 4 + 5, (7 - j) \* tileSize + tileSize / 4 + 5 + menuH);

in.setFillColor(Color(30, 30, 30));

win.draw(in);

}

else if (board[i][j] == 4) {

CircleShape in(tileSize / 4 - 5);

in.setPosition(i \* tileSize + tileSize / 4 + 5, (7 - j) \* tileSize + tileSize / 4 + 5 + menuH);

in.setFillColor(Color(230, 230, 230));

win.draw(in);

}

}

}

}

void drawAsses(RenderWindow& window, float asses, Font& font, bool turn) {

const int x = 800;

int height = 0;

asses = std::round(asses \* 10) / 10;

std::string result;

if ((asses > 9) || (asses < -9)) {

result = "win";

}

else {

result = std::to\_string(asses);

mytype dot = result.find('.');

result = result.substr(0, dot+2);

}

Text text(result, font, 15);

if (asses > 5) {

height = 800;

}

else if (asses < -5) {

height = 0;

}

else {

height = round(80 \* (asses + 5));

}

RectangleShape Black(Vector2f(50, 800 - height));

Black.setFillColor(Color(30, 30, 30));

RectangleShape White(Vector2f(50, height));

White.setFillColor(Color(230, 230, 230));

if (turn) {

Black.setPosition(x, menuH);

White.setPosition(x, 800 - height + menuH);

}

else {

Black.setPosition(x, height + menuH);

White.setPosition(x, 0 + menuH);

}

window.draw(Black);

window.draw(White);

if (abs(asses) > 0.01) {

if (turn) {

if (asses > 0) {

text.setPosition(x + 16, 770 + menuH);

text.setFillColor(Color(30, 30, 30));

}

else {

text.setPosition(x + 12, 10 + menuH);

text.setFillColor(Color(230, 230, 230));

}

}

else {

if (asses > 0) {

text.setPosition(x + 16, 10 + menuH);

text.setFillColor(Color(30, 30, 30));

}

else {

text.setPosition(x + 12, 770 + menuH);

text.setFillColor(Color(230, 230, 230));

}

}

window.draw(text);

}

}

void DrawAll(RenderWindow& win, Engine& Vobla, Font& font, bool turn) {

win.clear();

RectangleShape menu(Vector2f(850, menuH));

menu.setFillColor(Color::White);

menu.setOutlineColor(sf::Color::Black);

menu.setOutlineThickness(2);

if (turn) {

drawCheckerboard(win, Vobla.MainBoard.Field);

}

else {

TField ToPrint;

for (mytype i = 0; i < 8; i++) {

for (mytype j = 0; j < 8; j++) {

ToPrint[i][j] = Vobla.MainBoard.Field[7 - i][7 - j];

}

}

drawCheckerboard(win, ToPrint);

}

drawAsses(win, Vobla.asses, font, turn);

win.draw(menu);

}

void StartScreen(RenderWindow& win, Font& font, bool\* turn, mytype\* depth) {

\*depth = 5;

\*turn = true;

bool exit = false;

RectangleShape fon(Vector2f(850, 850));

fon.setPosition(0, 0);

fon.setFillColor(Color(230, 230, 230));

Text tChoise("Choose your color:", font, 30);

tChoise.setFillColor(Color::Black);

tChoise.setPosition(300, 70);

Text whiteBText("White", font, 30);

whiteBText.setPosition(245, 130);

whiteBText.setFillColor(Color::Black);

Text blackBText("Black", font, 30);

blackBText.setPosition(245, 180);

blackBText.setFillColor(Color::Black);

TChoise choise1(445, 140);

TChoise choise2(445, 190);

Text tDiff("Choose the difficulty:", font, 30);

tDiff.setFillColor(Color::Black);

tDiff.setPosition(289, 270);

Text easyBText("Easy", font, 30);

easyBText.setPosition(245, 330);

easyBText.setFillColor(Color::Black);

Text mediumBText("Medium", font, 30);

mediumBText.setPosition(245, 380);

mediumBText.setFillColor(Color::Black);

Text hardBText("Hard", font, 30);

hardBText.setPosition(245, 430);

hardBText.setFillColor(Color::Black);

Text impossibleBText("Impossible", font, 30);

impossibleBText.setPosition(245, 480);

impossibleBText.setFillColor(Color::Black);

TChoise ch1(445, 340);

TChoise ch2(445, 390);

TChoise ch3(445, 440);

TChoise ch4(445, 490);

Text startBText("Start game", font, 20);

startBText.setPosition(373, 610);

startBText.setFillColor(Color::Black);

RectangleShape startB(sf::Vector2f(200, 50));

startB.setPosition(325, 600);

startB.setFillColor(Color::Green);

startB.setOutlineColor(Color(30, 30, 30));

startB.setOutlineThickness(2);

while (win.isOpen() && !exit) {

Event event;

while (win.pollEvent(event)) {

if (event.type == sf::Event::Closed) {

win.close();

}

else if (event.type == sf::Event::MouseButtonPressed) {

Vector2f mousePosition = Vector2f(Mouse::getPosition(win));

if (choise1.field.getGlobalBounds().contains(mousePosition)) {

\*turn = true;

}

else if (choise2.field.getGlobalBounds().contains(mousePosition)) {

\*turn = false;

}

else if (ch1.field.getGlobalBounds().contains(mousePosition)) {

ch1.isSelected = true;

ch2.isSelected = false;

ch3.isSelected = false;

ch4.isSelected = false;

\*depth = 4;

}

else if (ch2.field.getGlobalBounds().contains(mousePosition)) {

ch1.isSelected = false;

ch2.isSelected = true;

ch3.isSelected = false;

ch4.isSelected = false;

\*depth = 5;

}

else if (ch3.field.getGlobalBounds().contains(mousePosition)) {

ch1.isSelected = false;

ch2.isSelected = false;

ch3.isSelected = true;

ch4.isSelected = false;

\*depth = 6;

}

else if (ch4.field.getGlobalBounds().contains(mousePosition)) {

ch1.isSelected = false;

ch2.isSelected = false;

ch3.isSelected = false;

ch4.isSelected = true;

\*depth = 7;

}

else if (startB.getGlobalBounds().contains(mousePosition)) {

startBText.setString("Wait a bit...");

exit = true;

}

}

}

win.clear();

win.draw(fon);

win.draw(tChoise);

win.draw(blackBText);

win.draw(whiteBText);

win.draw(choise1.field);

win.draw(choise2.field);

if (\*turn) {

win.draw(choise1.in);

}

else {

win.draw(choise2.in);

}

win.draw(tDiff);

win.draw(easyBText);

win.draw(mediumBText);

win.draw(hardBText);

win.draw(impossibleBText);

win.draw(ch1.field);

win.draw(ch2.field);

win.draw(ch3.field);

win.draw(ch4.field);

if (\*depth == 4) {

win.draw(ch1.in);

}

if (\*depth == 5) {

win.draw(ch2.in);

}

if (\*depth == 6) {

win.draw(ch3.in);

}

if (\*depth == 7) {

win.draw(ch4.in);

}

win.draw(startB);

win.draw(startBText);

win.display();

}

}

void MainScreen(RenderWindow& win, Font& font, bool turn, int depth) {

Engine Vobla(depth);

bool LP = false;

bool LR = false;

Text exitBText("Exit to main menu", font, 20);

exitBText.setPosition(645, 10);

exitBText.setFillColor(Color::Black);

RectangleShape exitB(sf::Vector2f(200, 30));

exitB.setPosition(625, 10);

exitB.setFillColor(Color::Green);

exitB.setOutlineColor(Color(30, 30, 30));

exitB.setOutlineThickness(2);

Text lose("You lose", font, 150);

lose.setPosition(120, 250);

lose.setFillColor(Color::White);

lose.setOutlineColor(Color(30, 30, 30));

lose.setOutlineThickness(10);

Text youwin("You win", font, 150);

youwin.setPosition(140, 250);

youwin.setFillColor(Color::White);

youwin.setOutlineColor(Color(30, 30, 30));

youwin.setOutlineThickness(10);

Text isdraw("Draw", font, 150);

isdraw.setPosition(230, 250);

isdraw.setFillColor(Color::White);

isdraw.setOutlineColor(Color(30, 30, 30));

isdraw.setOutlineThickness(10);

mytype x1, y1, x2, y2;

if (!turn) {

Vobla.EngineMove();

}

if (win.isOpen()) {

DrawAll(win, Vobla, font, turn);

win.draw(exitB);

win.draw(exitBText);

win.display();

}

while (win.isOpen())

{

Event event;

while (win.pollEvent(event))

{

if (event.type == Event::Closed)

win.close();

if (event.type == Event::MouseButtonPressed) {

Vector2f mousePressPosition = Vector2f(Mouse::getPosition(win));

if (exitB.getGlobalBounds().contains(mousePressPosition)) {

return;

}

else {

x1 = mousePressPosition.x / tileSize;

y1 = (mousePressPosition.y - menuH) / tileSize;

if (turn) {

y1 = 7 - y1;

}

else {

x1 = 7 - x1;

}

if (event.mouseButton.button == sf::Mouse::Left) {

LP = true;

}

}

}

if (event.type == Event::MouseButtonReleased) {

if (LP) {

Vector2i mouseReleasePosition = Mouse::getPosition(win);

x2 = mouseReleasePosition.x / tileSize;

y2 = (mouseReleasePosition.y - menuH) / tileSize;

if (turn) {

y2 = 7 - y2;

}

else {

x2 = 7 - x2;

}

if (event.mouseButton.button == sf::Mouse::Left) {

LR = true;

}

}

}

if (LP && LR) {

LP = false;

LR = false;

mytype result = Vobla.PlayerMove(x1, y1, x2, y2);

DrawAll(win, Vobla, font, turn);

win.draw(exitB);

win.draw(exitBText);

if (result == 0) {

win.draw(isdraw);

}

win.display();

if (result == 1) {

mytype result2 = Vobla.EngineMove();

DrawAll(win, Vobla, font, turn);

win.draw(exitB);

win.draw(exitBText);

if (result2 == -1) {

win.draw(youwin);

}

else if (result2 == 1) {

win.draw(lose);

}

win.display();

}

}

}

}

}

int main()

{

bool turn = true;

mytype depth = 5;

RenderWindow win(VideoMode(850, 850), "VOBLA", Style::Close);

Image icon;

if (!icon.loadFromFile("Image/icon.png")) {

return 1;

}

win.setIcon(512, 512, icon.getPixelsPtr());

Font font;

if (!font.loadFromFile("Fonts/arialmt.ttf")) {

return 2;

}

while (win.isOpen()) {

StartScreen(win, font, &turn, &depth);

if (win.isOpen()) {

MainScreen(win, font, turn, depth);

}

}

return 0;

}

ВЕДОМОСТЬ

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1-40 01 01 115 ПЗ | | | | Пояснительная записка | | | | 56 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 251001 115 СП | | | | "Игровой движок для русских шашек"  А1, схема программы, чертеж | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 115 ПЗ | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Игровой движок для русских шашек  Ведомость курсовой работы |  | | | | Лист | Листов |
| Разраб. | | Кривицкий Ф.Ю. |  |  | Т |  | |  | 52 | 52 |
| Пров. | | Шостак Е.В. |  |  | Кафедра ПОИТ  гр. 251001 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |