Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системное программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

БИЗНЕС-ЛОГИКА ПРОГРАММНОГО СРЕДСТВА ДЛЯ ДЕФРАГМЕНТАЦИИ

ФАЙЛОВОЙ СИСТЕМЫ NTFS

БГУИР КП I–40 01 01 421 ПЗ

Выполнил

студент: гр. 851004 Пашкевич А.Л.

Проверил: Шульга Е.С.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и

радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

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

(подпись)

Лапицкая Н.В. 2020 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Пашкевичу Антону Леонтьевичу* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы  *Бизнес-логика программного средства для дефрагментации файловой системы NTFS.* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

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

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

2. Срок сдачи законченной работы *18.12.2020 г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Интерфейс программирования приложений Windows API, язык программирования C++. Разработать DLL-библиотеку, содержащую функции по дефрагментации файловой системы NTFS. Предоставить API данной библиотеки для использования её сторонними приложениями. Разработать функции по сбору информации о носителе данных и сбору статистики процесса дефрагментации. Предусмотреть возможность работы библиотеки в многопоточной среде.*

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

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ предметной области \_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Анализ требований к программному средству\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*3 Проектирование и разработка программного средства*

*4 Руководство по использованию программного средства \_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе  *Шульга Е.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *04.09.2020г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

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

*Раздел 1. Введение к 15.09.2020г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 01.10.2020г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 30.10.2020г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4 к 20.11.2020г. – 80% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Заключение. Приложения к 01.12.2020г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 07.12.2020г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 10.12.2020г. по 18.12.2020г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шульга Е.С*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_Пашкевич А. Л. 04.09.2020г.*

*(дата и подпись студента)*

Содержание

[Содержание 4](#_Toc58346912)

[Введение 5](#_Toc58346913)

[1 Анализ предметной области 6](#_Toc58346914)

[1.1 Достоинства файловой системы NTFS 6](#_Toc58346915)

[1.2 Структура файловой системы NTFS 7](#_Toc58346916)

[1.3 Дефрагментация диска 9](#_Toc58346917)

[1.4 Обзор существующих аналогов 10](#_Toc58346918)

[2 Анализ требований к программному средству 11](#_Toc58346919)

[3 Проектирование и разработка программного средства 12](#_Toc58346920)

[3.1 Структура программного средства 12](#_Toc58346921)

[3.2 Разработка серверной части 12](#_Toc58346922)

[3.3 Разработка клиентской части 14](#_Toc58346923)

[3.4 Разработка модуля шахматной логики 17](#_Toc58346924)

[4 Руководство по использованию программного средства 21](#_Toc58346925)

[4.1 Использование сервера 21](#_Toc58346926)

[4.2 Использование клиента 22](#_Toc58346927)

[Заключение 26](#_Toc58346928)

[Список использованной литературы 27](#_Toc58346929)

[Приложение А 28](#_Toc58346930)

[Текстовые документы 31](#_Toc58346931)

[Пояснительная записка 31](#_Toc58346932)

[Графические документы 31](#_Toc58346933)

[Формат А1 31](#_Toc58346934)

Введение

Системное программное обеспечение – это комплекс программ, которые обеспечивают управление компонентами компьютерной системы, такими как процессор, оперативная память, устройства ввода-вывода, сетевое оборудование, выступая как «межслойный интерфейс», с одной стороны которого аппаратура, а с другой — приложения пользователя. В отличие от прикладного программного обеспечения, системное не решает конкретные практические задачи, а лишь обеспечивает работу других программ, предоставляя им сервисные функции, абстрагирующие детали аппаратной и микропрограммной реализации вычислительной системы, управляет аппаратными ресурсами вычислительной системы.

На сегодняшний день системное программное обеспечение не пользуется такой большой популярностью как, например, в начале 2000-х годов. Это обуславливается тем, что, в основном, оно производится крупными корпорациями, которые являются монополистами, и больший спрос составляют различного рода веб-приложения и веб-службы. Тем не менее, важность системного ПО остаётся прежней, особенно в области разработки новых операционных систем на принципиально новых архитектурах.

Данный курсовой проект направлен на создание динамически подключаемой библиотеки (далее DLL-библиотеки), предоставляющей функции по дефрагментации файловой системы NTFS. Актуальность такого продукта подкрепляется бурным ростом размеров файлов в операционных системах, а также необходимостью рационального использования памяти при их физическом расположении на накопителе.

Помимо реализации функций по дефрагментации, в данном курсовом проекте будет разработан функционал по сбору информации о диске и предусмотрена возможность работы библиотеки в многопоточной среде.

# Анализ предметной области

## Достоинства файловой системы NTFS

NTFS (англ. new technology file system - «файловая система новой технологии») – стандартная файловая система для семейства операционных систем Windows NT фирмы Microsoft.

Для размещения файлов в NTFS используется кластеры. Размер кластера может варьироваться от 512 [байт](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82) до 64 КБ в зависимости от размера тома и версии операционной системы. Для пометки кластеров, как занятых используется по одной битовой карте для каждого тома. С целью улучшения производительности, надёжности и эффективности использования дискового пространства для хранения информации о файлах в NTFS используются специализированные структуры данных. Информация о файлах хранится в главной файловой таблице — Master File Table (MFT). Содержимое папок в NTFS представляется с помощью B± дерева, что является достаточно эффективным способом.

Отличительной особенностью данной файловой системы является то, что NTFS поддерживает хранение метаданных и разграничение доступа к данным для различных пользователей и групп пользователей, а также позволяет назначать дисковые квоты (ограничения на максимальный объём дискового пространства, занимаемый файлами тех или иных пользователей). Для повышения надёжности файловой системы в NTFS используется система журналирования USN.

NTFS обладает свойством восстанавливаемости – может возвращаться к работоспособному состоянию после возникновения сбоя. Реализуется такая возможность, во-первых, за счет поддержки атомарных транзакций, во-вторых, за счет избыточности хранения информации. Избыточность используется при хранении важнейших данных файловой системы, критически необходимых для её корректной работы;

Файловая система NTFS появилась в 1993 году заменила файловую систему FAT, использовавшуюся в ОС MS-DOS и ОС Windows и обладает рядом достоинств перед ней. Так, максимальный размер диска, который поддерживает NTFS, составляет в теории байт (16 ЭиБ), а на практике 255 тб, в то время, как FAT поддерживает приблизительно 243 байт (8 [ТиБ](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%B1%D0%B8%D0%B1%D0%B0%D0%B9%D1%82)), что значительно меньше.

Размер тома в FAT ограничивается 4 177 920 кластерами, а с учётом самих таблиц FAT и при максимальном размере кластера 32 Кбайт размер тома может быть до 127.53 ГБ. В NTFS том можно создать размером с весь диск: он будет содержать максимальное количество адресуемых кластеров - при максимальном размере кластера в 64 кб.

NTFS, в отличие от FAT, поддерживает жёсткие и символьные ссылки, соединения для каталогов. Для обеспечения безопасности, помимо атрибутов файлов, использует авторизацию с использованием DACL, шифрование с использованием EFS, аудит с использованием SACL.

Максимальный размер файла в NTFS составляет 16 терабайт, в то время как FAT32 может работать с файлами размером не более 4 Гб.

Таким образом, можно сделать вывод, что достоинства NTFS над FAT сводятся к трём пунктам:

* Способность автоматически восстанавливаться после некоторых ошибок диска (FAT не обладает такой способностью).
* Улучшенная поддержка больших объёмов данных, расширение размера диска и файлов, увеличение количества файлов.
* Более высокая степень безопасности. Возможно использование разрешений и шифрования для запрета пользовательского доступа к определенным файлам. В файловой системе FAT все файлы видны абсолютно любому пользователю.

## Структура файловой системы NTFS

В начале тома находится загрузочная запись тома (Volume Boot Record), в которой содержится код загрузки Windows, информация о томе (в частности, тип файловой системы), адреса системных файлов ($Mft и $MftMirr). Загрузочная запись занимает обычно 8 КБ (16 первых секторов).

В определенной области тома (адрес начала этой области указывается в загрузочной записи) расположена основная системная структура NTFS – главная таблица файлов (Master File Table, MFT). В записях этой таблицы содержится вся информация о расположении файлов на томе, а небольшие файлы хранятся прямо в записях MFT.

Важной особенностью NTFS является то, что вся информация, как пользовательская, так и системная, хранится в виде файлов. Имена системных файлов начинаются со знака "$". Например, загрузочная запись тома содержится в файле $Boot, а главная таблица файлов – в файле $Mft. Такая организация информации позволяет единообразно работать как с пользовательскими, так и с системными данными на томе.

Поскольку MFT является важнейшей системной структурой, к которой при операциях с томом наиболее часто происходят обращения, выгодно хранить файл $Mft в непрерывной области логического диска, чтобы избежать его фрагментации (размещения в разных областях диска), и, следовательно, повысить скорость работы с ним. С этой целью при форматировании тома выделяется непрерывная область, называемая зоной MFT (MFT Zone). По мере увеличения главной таблицы файлов, файл $Mft расширяется, занимая зарезервированное место в зоне. Остальное место на томе NTFS отводится под файлы – системные и пользовательские.

На рисунке 1.1 показаны структуры данных, связанные с NTFS.

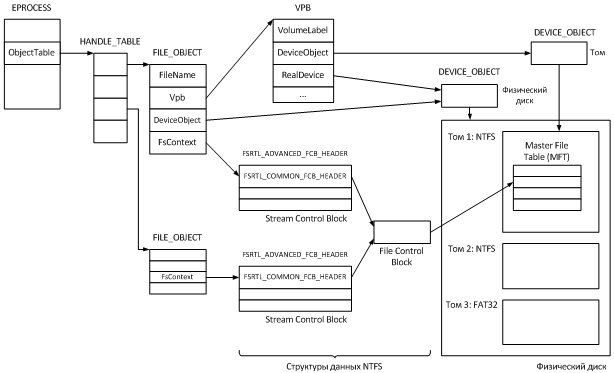


Рисунок 1.1 – Структуры данных, связанные с NTFS

Основная информация о файле содержится в файловой записи размером 1 КБ таблицы MFT, а небольшие файлы целиком хранятся в файловой записи.

Файловая запись состоит из заголовка и набора атрибутов. В заголовке содержится служебная информация о файловой записи, например, её тип и размер. Все данные, относящиеся непосредственно к файлу, хранятся в виде атрибутов. Названия атрибутов, так же как и системных файлов, начинаются с "$". Например, отдельными атрибутами являются имя файла ($FILE\_NAME), информация о его свойствах ($STANDARD\_INFORMATION), данные файла ($DATA).

На диске файловая запись всегда расположена в начале сектора, первые байты файловой записи кодируют слово "FILE". Конец записи определяется 4 байтовой последовательностью FF FF FF FF.

Физически атрибут файла хранится в виде потока байтов – простой последовательности байтов. Такое представление позволяет одинаковым образом работать с разнотипными атрибутами, а также добавлять нестандартные пользовательские атрибуты. Каждый атрибут состоит из заголовка, определяющего тип атрибута и его свойства, и тела, содержащего основную информацию атрибута. Структура файловой записи представлена на рисунке 1.2.

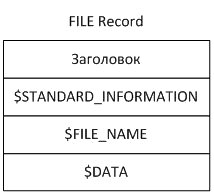


Рисунок 1.2 – Файловая запись

## Дефрагментация диска

Существует два типа фрагментации дискового пространства: внутренняя и внешняя. Внутренняя фрагментация наблюдается тогда, когда фактический размер выделенного блока памяти для файла больше чем нужный, и много памяти внутри блока не используется. Например, файл размером 512 байт занимает целый кластер в 4 Кб, при этом оставшаяся память не может использоваться файловой системой.

Внешняя фрагментация появляется тогда, когда есть достаточно большой объём свободной памяти для выделения блока, но нет непрерывного участка достаточного размера. В этом случае новый файл разбивается на фрагменты различной величины для записи в различные (не последовательные) области жёсткого диска, то есть становится фрагментированным.

Внешняя фрагментация файловой системы, по своей сути, является компромиссом между простотой и скоростью первоначальной записи данных на диск и скоростью чтения этих же самых данные в последующем. Данное явление может происходить с связи с активным использованием файловой системы. Например, при удалении файлов образуется свободный блок памяти, которого может быть недостаточно в последующем для записи большего файла, тогда он будет фрагментирован. Также при увеличении размера файла сверх размера кластера файловая система помещает оставшуюся часть файла в конце дискового пространства – файл снова оказывается фрагментированным.

К основным последствиям внешней фрагментации относятся уменьшение скорости работы накопителя с фрагментированными файлами, поскольку возникают задержки на физическое перемещение головки диска между разделёнными фрагментами, а также замедляется запись новых файлов на накопитель.

По этим причинам используется дефрагментация диска – процесс перераспределения фрагментов файлов и логических структур файловой системы для обеспечения непрерывной последовательности кластеров.

В случае использования жестких дисков, вследствие дефрагментации ускоряется чтение и запись файлов, а также работа программ и операционной системы. Это достигается за счет того, что чтение файлов после дефрагментации производится максимально линейно и непрерывно, без дополнительных передвижений головки жёсткого диска для поиска и воссоединения фрагментов.

Твердотельные накопители на основе различных видов флеш-памяти в дефрагментации файловых систем не нуждаются. Флеш-память является памятью с прямым доступом, в которой время доступа к любой ячейке сравнительно мало и одинаково вне зависимости от её адреса. Более того, дефрагментация, в некоторой степени, вредит флеш-памяти, так как она имеет намного более ограниченное количество циклов записи/перезаписи, чем накопители на магнитных дисках при должном использовании.

Существует несколько типов дефрагментации:

* Простая дефрагментация. Дефрагментируются только фрагментированные файлы, свободное же место не обрабатывается. Ускоряется работа с уже существующими файлами.
* Дефрагментация свободного пространства. Обрабатывается только свободное пространство, файлы же остаются фрагментированными. Намного ускоряет запись новых файлов.
* Полная дефрагментация. Одновременно дефрагментируется и свободное пространство, и сами файлы, которые и помещаются в начало раздела, что позволяет освободить максимально возможную свободную область диска. Ускоряет работу как с новыми файлами, так и с уже существующими.
* Оптимизация. Помимо дефрагментации перемещает физически в начало раздела мелкие и важные для работы системы файлы и перемещает в конец диска большие, пользовательские или редко используемые.

Алгоритм простой дефрагментации является самым часто используемым и наиболее простым в реализации, быстрым среди остальных алгоритмов. Проводится анализ файла и поиск всех кластеров этого файла. После составления карты кластеров проводится поиск подходящего места для расположения файла целиком и перемещение его фрагментов в этой место. Если же место для файла не найдено, то он пропускается и остается фрагментированным. Результат алгоритма простой дефрагментации схематично представлен на рисунке 1.3.

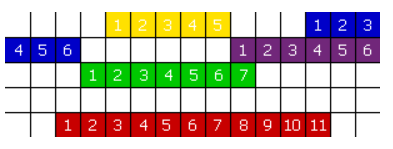


Рисунок 1.3 – Простая дефрагментация

Алгоритм полной дефрагментации в настоящее время используется крайне редко, как основной метод, так как он занимает длительное время и требует относительно много свободного дискового пространства. В отличие от простой дефрагментации, перемещение файлов осуществляется не в любое подходящее место на диске, а в конец раздела, таким образом, образуется область данных без свободных кластеров. После дефрагментации всех файлов выполняется их перемещение в свободные кластеры в начале тома. Таким образом проводится реорганизация файловой структуры, благодаря которой в начале дискового раздела создается область данных, а в конце – все доступное свободное дисковое пространство. Схематично результат алгоритма полной дефрагментации представлен на рисунке 1.4.

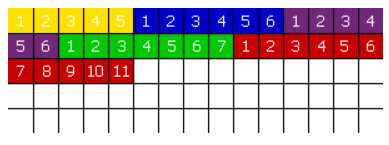


Рисунок 1.4 – Полная дефрагментация

## Обзор существующих аналогов

Почти во всех современных операционных системах существуют встроенные утилиты, выполняющие функцию дефрагментации дискового пространства. Однако, зачастую, они реализуют не самые оптимальные алгоритмы и предоставляют скудный функционал. В этой связи большой интерес представляют средние и крупные коммерческие проекты. Одним из таких приложений является IObit Smart Defrag.

Данное программное средство не только дефрагментирует сами файлы, но и распределяет их на диске по признаку частоты использования. Все операции можно настроить для их выполнения в автоматическом режиме. Разработчики утверждают, что Smart Defrag использует одни из самых эффективных алгоритмов дефрагментации. Есть возможность дефрагментировать отдельные папки или файлы, а также возможность работы по расписанию. Существует опция дефрагментации диска перед загрузкой операционной системы. Приложение имеет продуманный и удобный интерфейс и обеспечивает высокую безопасность при работе, в том числе при перебоях электропитания. Вид окна Smart Defrag, отражающего состояние диска, представлен на рисунке 1.5.

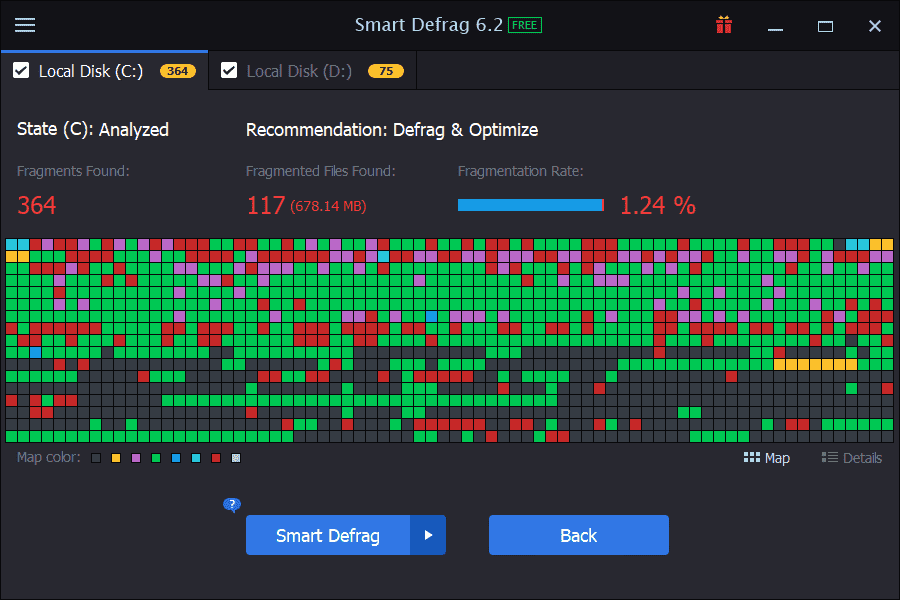


Рисунок 1.5 – Отображение состояния диска в Smart Defrag

Более простым приложением для быстрой оценки и дефрагментации жестких дисков является Defraggler. Данное приложение создано британской частной фирмой Piriform Limited и написано на языке программирования C++. Это бесплатная утилита имеет все самые необходимые функции: оценка состояния диска, анализ, дефрагментации отдельных файлов. Реализована как полный, так и простой алгоритм дефрагментации. Есть возможность автоматически определять тип носителя (HDD, SSD), что очень удобно если подключено несколько носителей. К тому же, как было сказано ранее, дефрагментировать SSD не рекомендуется. Существует портативная версия, не нуждающаяся в установке (программу можно скопировать на флеш-накопитель/внешний HDD и использовать на любом компьютере). После анализа диска можно просмотреть список фрагментированных файлов. Утилита поддерживает файловые системы NTFS и FAT32, а также все самые современные версии операционной системы Windows. Вид окна Defraggler, отражающего состояние диска, представлен на рисунке 1.6.

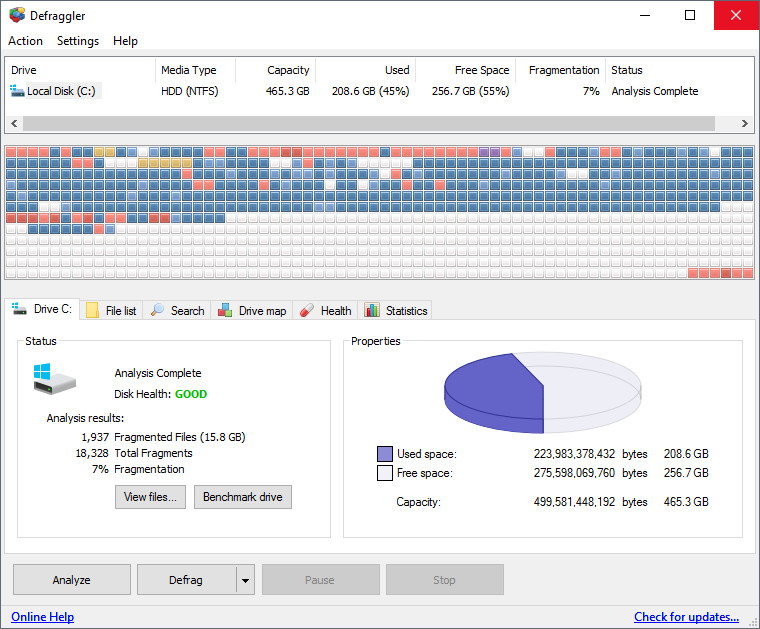


Рисунок 1.6 – Отображение состояния диска в Defraggler

В результате анализа существующих аналогов можно утверждать, что большинство крупных программных средств для дефрагментации предоставляют приблизительно одинаковый функционал с точки зрения пользователя. В него входит полное отображение пользователю информации о дисковой подсистеме, а также о ходе процесса дефрагментации, гибкие настройки управления временными и пространственными характеристиками дефрагментатора, а также возможность выполнения в автоматическом режиме. Однако, стоит заметить, что каждое программное средство реализует алгоритмы дефрагментации по-своему, лишь заявляя о их высокой эффективности. Вместе с тем, детали реализации остаются скрытыми коммерческой тайной, поэтому провести сравнительный анализ бизнес-логики дефрагментаторов не представляется возможным.

# Анализ требований к программному средству

DLL-библиотека DefragmenterCore предназначается для предоставления функций по дефрагментации файловой системы NTFS и сбору информации о носителях данных.

Интерфейсная часть (API) библиотеки должна содержать следующие элементы:

* структуру данных для хранения информации о диске, включая общий объём памяти на нём и объём свободной памяти;
* функцию получения информации о всех дисках на устройстве;
* структуру данных для хранения информации файловой системы о диске, включая количество секторов и кластеров на диске, их размерность в байтах и расположение таблицы MFT;
* функцию получения информации файловой системы о диске;
* функцию валидации пользователя библиотеки как администратора системы;
* структуры данных для хранения информации о результатах процесса дефрагментации и статистики по обработанным файлам;
* функции получения информации о результатах процесса дефрагментации и статистики по файлам;
* функцию входа в рабочий цикл дефрагментатора;
* функцию остановки процесса дефрагментации диска;
* функции по обеспечению работы библиотеки в многопоточной среде.

Часть реализации библиотеки должна соответствовать следующим требованиям:

* реализовывать алгоритм простой дефрагментации, не затрагивая свободное дисковое пространство;
* журналировать обработку файлов и собирать статистику обработки;
* обеспечивать потокобезопасность библиотеки посредством использования примитивов синхронизации.

Реализация DLL-библиотеки может использовать типы данных языка C++ стандарта ISO/IEC 14882:2011 и типы данных Windows API, а для взаимодействия с файловой системой – только функции Windows API.

# Проектирование и разработка программного средства

## Структура программного средства

Как и любая другая DLL-библиотека DefragmenterCore состоит из интерфейсной части, представленной заголовочным файлом Core.h, и части реализации. Она имеет следующую структуру файлов:

* CheckUser.cpp – реализация функции валидации пользователя библиотеки, как администратора системы;
* Core.cpp – точка входа в DLL-библиотеку;
* Defragger.cpp – реализация дефрагментации, процесса её журналирования и сбора статистики;
* DriveOperation.cpp – реализация получения информации файловой системы о диске;
* Menu.cpp – реализация получения базовой информации о диске.

Как видно, структура программного средства довольно простая, что устраняет трудности в её использовании и переработке. После сборки библиотека компилируется в выходной dll-файл, который непосредственно подключается пользователем с помощью динамического либо статического связывания.

## Разработка интерфейсной части

При разработке DLL-библиотеки с помощью директив условной компиляции необходимо объявить конструкцию, представленную на рисунке 3.1.

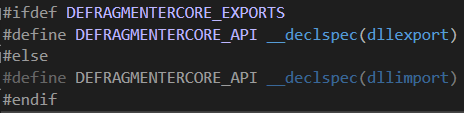


Рисунок 3.1 – Объявление константы DEFRAGMENTERCORE\_API

Впоследствии каждая интерфейсная функция будет объявляться с константой DEFRAGMENTERCORE\_API, что позволит библиотеке её экспортировать, а стороннему приложению, соответственно импортировать её. Помимо этого, функции будут иметь соглашение вызова cdecl, что обеспечит совместимость со стандартами операционной системы Windows.

Структура DriveData содержит информацию о диске. В ней хранятся буква диска, его тип (фиксированный, отсоединяемый, сетевой, CD-ROM и ОЗУ-диск), количество секторов в кластере и количество байтов в секторе, полный объём памяти на диске и объём свободного пространства. Все поля являются строками типа wchar\_t фиксированного размера. Это сделано для прямой выдачи данных пользователю в виде, удобном для непосредственного использования.

Соответственно, функция getDrives(), предназначается для получения базовой информации о дисках. Она возвращает вектор структур DriveData.

Структура DriveInfo содержит информацию файловой системы NTFS о диске. В ней находятся поля для хранения буквы диска, серийного номера тома, количество секторов и кластеров, количество свободных и зарезервированных кластеров, количество байт в секторе и кластере, а также информацию о таблицы MFT: длину мастер-таблицы, номер начального кластера таблицы, номера начального и конечного кластера зоны мастер-таблицы. Все поля представляются строками типа wchar\_t фиксированного размера.

Соответственно, функция getDriveInfo по переданной ей букве диска возвращает структуру DriveInfo с информацией файловой системы о нём.

Точкой входа в рабочий цикл дефрагментации является функция WorkIn. Она использует соглашение вызова stdcall, что необходимо для её передачи в отдельно работающий поток функцией CreateThread. В качестве аргумента она принимает указатель LPVOID на структуру StartDefragInfo, содержащую информацию о начальном месте дефрагментации. В ней содержится текущая директория, буква диска и флаг, обозначающий, является ли директория корневой.

Функции InitCS и DeleteCS вызываются сторонним приложением и обеспечивают инициализацию и удаление внутренней критической секции дефрагментатора.

Функция StopDefrager принимает handle потока, в котором работает дефрагментатор, и останавливает процесс дефрагментации.

Функция getDefragmentationStats может быть вызвана в любой момент времени и возвращает структуру DefragmentationStats. В ней хранится количество пропущенных, дефрагментированных и файлов, при дефрагментации которых произошла ошибка (например, не было найдено свободного участка памяти достаточного размера).

Структура DefragmentationLogItem является элементом журналирования и содержит полный путь к файлу и результат его обработки. Эти поля также представляются строками типа wchar\_t фиксированного размера.

Функция getDefragmentationLogs возвращает очередь таких структур. Она должна вызываться пользователем библиотеки регулярно, чтобы была возможность своевременно отображать результаты процесса дефрагментации. Для регулярного вызова возможно использование события таймера WM\_TIMER и функции SetTimer, в которую можно передать требуемый период посылки сообщения.

Таким образом, API библиотеки предоставляет пользователю необходимый контроль над процессом дефрагментации диска, а также инструменты получения оперативной информации.

## Реализация интерфейсных функций

Для валидации пользователя в качестве администратора системы разработана функция isUserAdmin. Она использует вызов AllocateAndInitializeSid, который получает значение sid группы администратора, а затем с помощью CheckTokenMembership проверяет наличие в ней значения sid текущего пользователя.

В основе реализации функции getDrives лежит вызов функции GetLogicalDriveStringsA. Она позволяет получить буфер, который содержит буквы всех логических дисков, находящихся в системе. Важную роль играет функция GetDriveTypeA, которая по букве диска может вернуть его тип. Функция GetDiskFreeSpaceA по букве диска возвращает количество байтов в секторе и секторов в кластере, а также общее число и число свободных кластеров. Далее, с помощью вычислений находятся показатели из структуры DriveData.

Для представления данных в строках используется функция std::to\_wstring, а также запись в поток std::wostringstream с последующим изъятием из него строки в кодировке Юникод.

Для реализации функции getDriveInfo необходимо открыть файл диска с помощью CreateFileA. При этом необходимо указать атрибуты GENERIC\_READ, GENERIC\_WRITE, FILE\_SHARED\_READ, FILE\_SHARED\_WRITE и OPEN\_EXISTING. Далее выделяется память для буфера с информацией файловой системы NTFS\_VOLUME\_DATA\_BUFFER и выполняется вызов DeviceIoControl. В него передаётся handle дискового файла, константа FSCTL\_GET\_NTFS\_VOLUME\_DATA и указатель на данный буфер. В результате из него можно изымать данные и помещать в структуру DriveInfo. При этом, конвертация в строковое представление осуществляется посредством записи в поток std::wostringstream.

Стоит обратить внимание на функцию DeviceIoControl, так как она будет использоваться в дальнейшем, в том числе в реализации алгоритма дефрагментации. Данная функция выполняет отправку управляющего кода на драйвер устройства, принуждая устройство выполнить данную операцию. В getDriveInfo устройство (диск) идентифицируется значением handle открытого файла, а константа FSCTL\_GET\_NTFS\_VOLUME\_DATA обеспечивает извлечение информации о томе файловой системы NTFS.

## Разработка клиентской части

Системообразующей основы работы клиента является схема переходов состояний, вид которой представлен на рисунке 3.2.

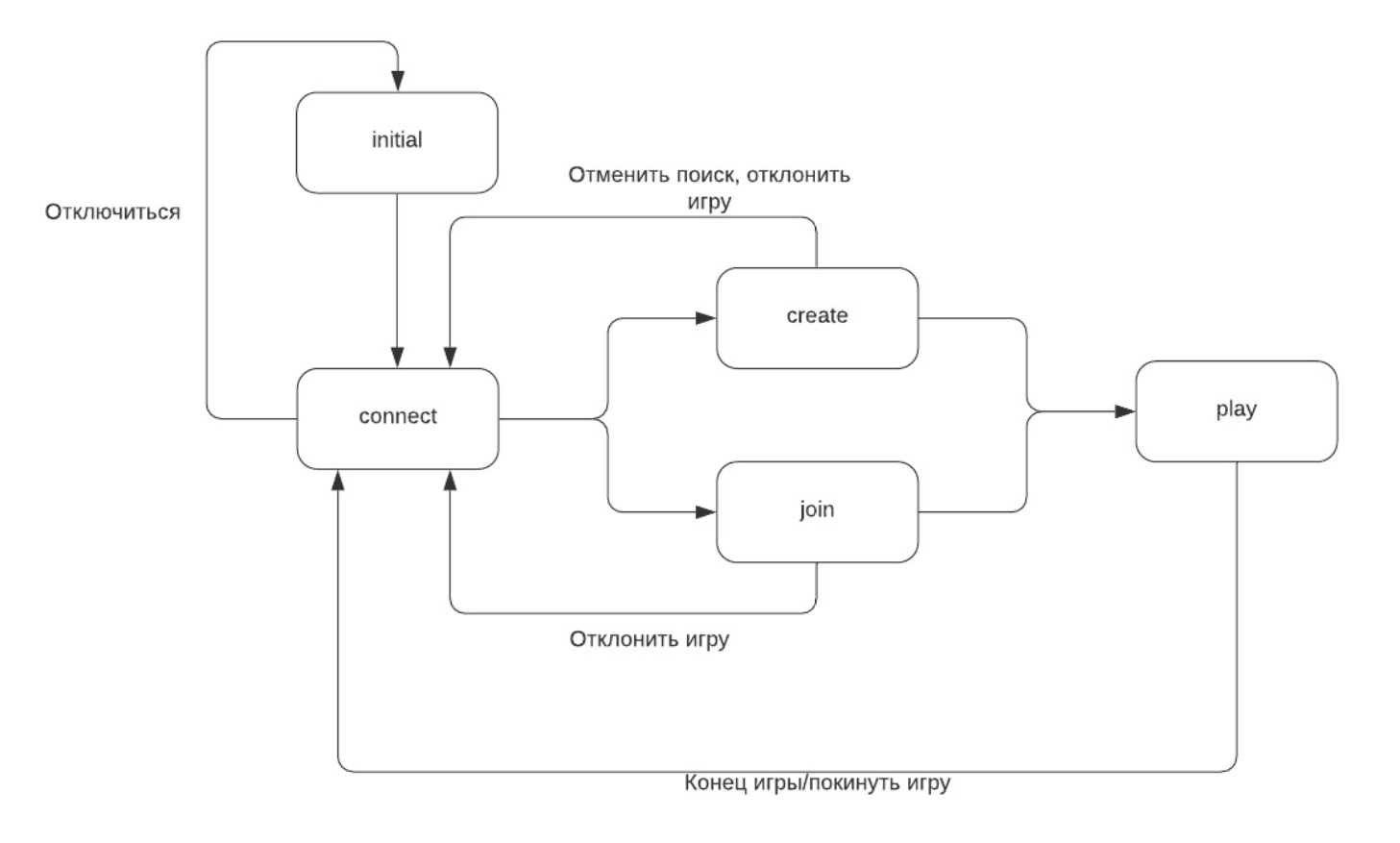


Рисунок 3.2 – Схема работы клиента

Начальное состояние initial является стартовым для клиента. Далее он двигается вперёд к connect, что означает подключение к определённому серверу. Затем клиенту предоставляется возможность либо создать игру, либо присоединиться к уже созданной. Конечное состояние play означает, что клиент находится в состоянии игры. Обратим внимание, что на схеме есть дополнительные обратные переходы, которые подписаны условиями, при которых они происходят. Таким образом обеспечивается возможность проведения нескольких игр пользователем и переподключения к разным серверам.

За счёт переходов между состояниями, на каждом этапе пользователю предоставляются соответствующие состоянию опции взаимодействия с сервером, что отражается в изменениях графического интерфейса, а также в отображении вспомогательной информации.

Для реализации клиента используется класс TcpClient, который находится в пространстве имён System.Net.Sockets. В его конструктор передаются IP-адрес сервера и номер его открытого порта. Далее клиент общается с сервером помощью методов SendMessage и GetServerResponse. Они в свою очередь используют поток объекта клиента класса NetworkStream для записи и чтения данных оттуда. При общении с сервером используются запросы, описанные ранее.

Следует заметить, что клиент устроен так, что пользователь управляет приложением (в том числе и играет) в основном потоке. При создании новой игровой сессии, стартует поток, в котором происходит ожидание игры, её подтверждение пользователем и инициализация шахматной партии. Этот подход не блокирует пользовательский интерфейс и является оптимальным для подобных приложений. Также новый поток создаётся в методе инициализации партии. В нём происходит обработка получаемых от сервера сообщений.

Следует обратить внимание на то, что изменять компоненты формы может лишь основной поток. Для отображения изменений из потока обработки ответов сервера используется метод Invoke, который применяется к компонентам формы класса Control. В данный метод передаётся лямбда-выражение, которое по сути своей является анонимным делегатом, выполняющемся в основном потоке. Пример использования метода Invoke:

menuStrip1.Invoke(new Action(() => RefreshServerControlTools()));

Для графического интерфейса шахматных фигур и доски используются компоненты класса Panel, которые могут окрашиваться в разные цвета и могут содержать в себе изображение. Шахматные фигуры были добавлены в ресурсы проекта в виде PNG изображений.

Отличительной особенностью шахматного клиента в данном курсовом проекте является подсветка для пользователя всех возможных фигур для хода, а также всех клеток, куда можно сделать ход текущей фигурой. При этом учитывается текущая ситуация на доске (находится ли король под шахом, может ли король походить под шах и т.п.). Данный аспект обеспечивается методами MarkSquaresFrom и MarkSquaresTo, использующих логику разработанную в классе Moves.cs библиотеки «Chess». Соответствующий вид шахматной доски представлен на рисунке 3.3.

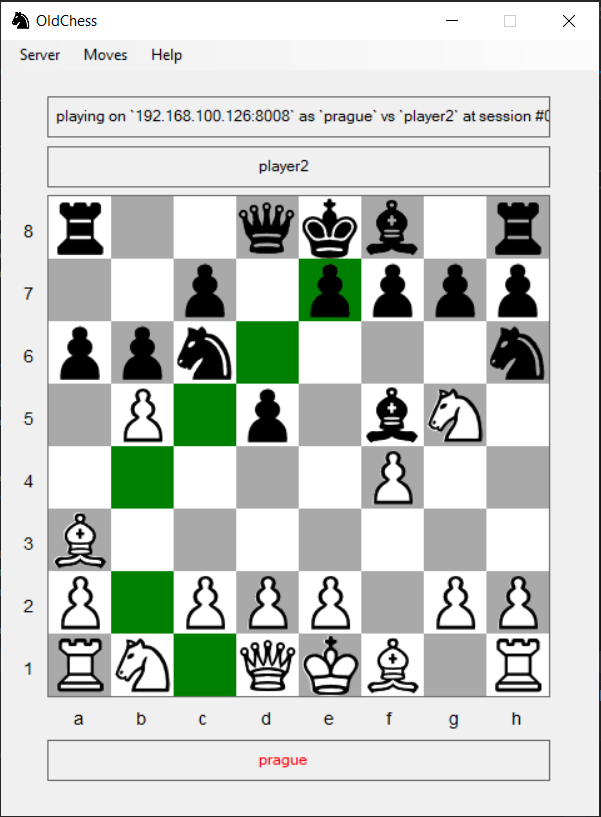


Рисунок 3.3 – Возможные позиции для хода Слоном

Отправка хода на сервер осуществляется в обработчике нажатия на клетку (то есть на компонент формы класса Panel). Если пользователь выбрал доступную фигуру и нажимает на доступную клетку для хода, то формируется строковое представление хода в форме «<фигура><начальная\_клетка><конечная\_клетка>[<превращение>]». В клиенте формируется новый объект Chess посредством метода Move, а на сервер посылается строка «MOVE <UserName> <move>». Заметим, что вместо «MOVE» могут использоваться «MOVEWIN» или «MOVEDRAW», в зависимости от ситуации на доске. Далее происходит перерисовка доски. В свою очередь другой клиент получает от сервера данную строку, создаёт новый объект Chess и выполняет перерисовку доски.

## Разработка модуля шахматной логики

Как уже было ранее, главным классом в модуле шахматной логики является Chess. В данном классе есть конструктор, в котором создаётся шахматная доска класса Board по полученному FEN. В методе Move проверяется может ли данная фигура сделать ход и не является данное перемещение ходом под шах. В случае допуска создаётся новая доска и по ней создаётся новый объект шахмат Chess.

Крайне важным является метод FindAllMoves, который ищет все допустимые ходы фигур с учётом текущего состояния доски и устранением возможности хода, который приведёт к шаху в свою сторону. Схема этого алгоритма по ГОСТ 19.701-90 приведена на рисунке 3.4.

На данном этапе стоит формализовать понятия мат и пат. Так как уже создан метод поиска всех доступных ходов, то можно использовать понятие мощности множества доступных ходов.

Пат в партии фиксируется тогда, когда мощность данного множества равна нулю, и король не находится под шахом. Мат оппоненту объявляется в том случае, когда у число доступных ходов оппонента равно нулю и король находится под шахом.

В определении шаха используется метод CanEatKing класса Board, в котором, для начала, ищется клетка короля нужного цвета. А затем ищутся все фигуры, которые могут побить короля, стоящего на найденной клетке. Если находится хоть одна такая фигура, то королю объявляется шах.

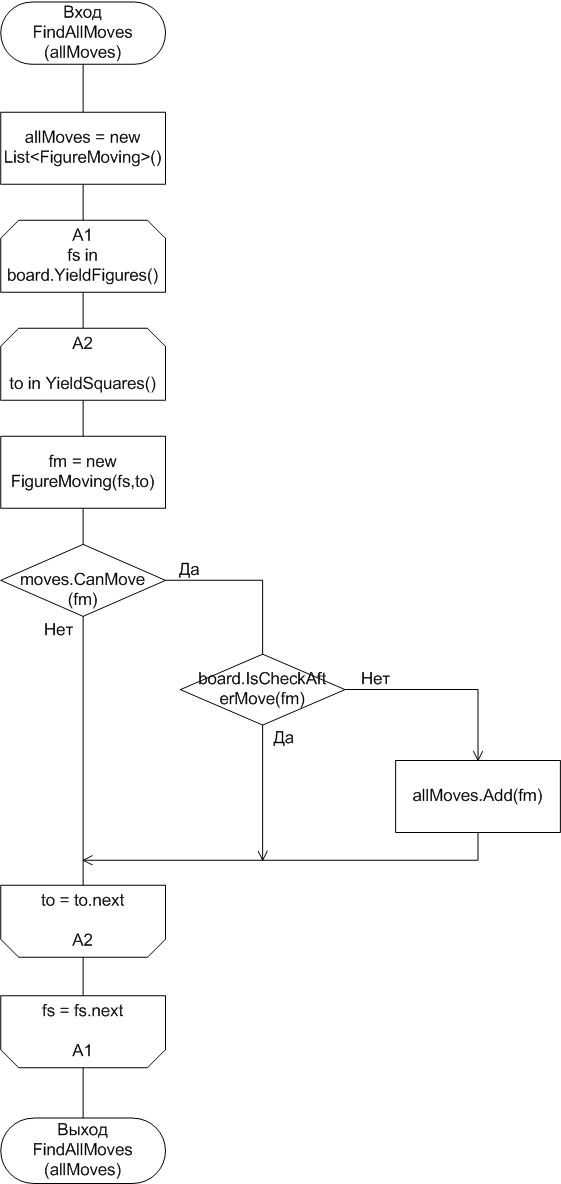


Рисунок 3.4 – Схема алгоритма FindAllMoves

Одним из самых важных методов класса Board является метод GetFen, который по текущему состоянию шахматной доски (то есть объекта Board) формирует FEN доски, являющийся основой для построения и объекта Chess, и самой доски в следующем состоянии. Схема данного алгоритма по ГОСТ 19.701-90 приведена на рисунке 3.5.

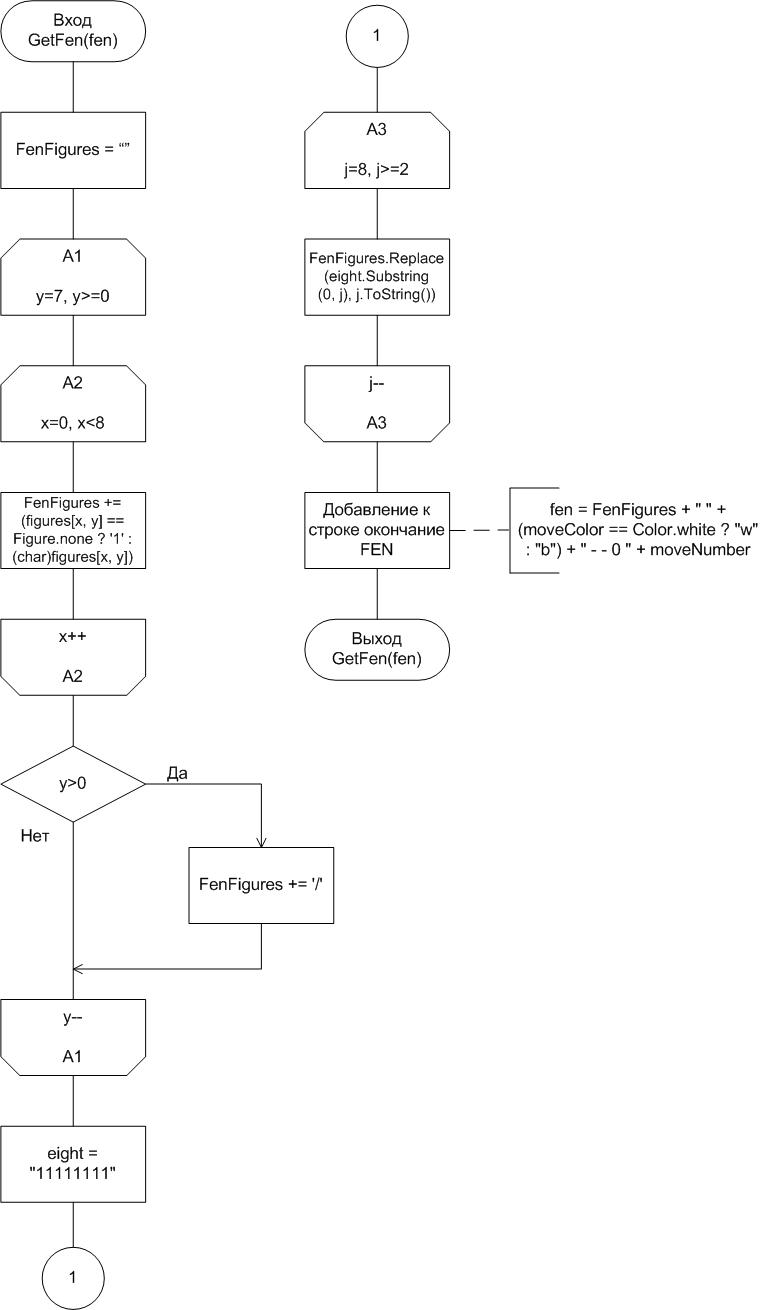


Рисунок 3.5 – Схема алгоритма GetFen

Опишем сам процесс хода (то есть метод Move класса Board). Создаётся объект доски по текущему FEN. Затем с клетки начала хода методом SetFigureAt удаляется фигура и тем же методом устанавливается на клетку конца хода. Увеличивается счётчик ходов, переключается активная сторона. В конце формируется FEN новой доски, объект которой возвращается из метода Move.

Также важную роль играет алгоритм получения матрицы шахматных фигур с помощью FEN. В метод InitFigures передаётся часть строкового представления FEN, отвечающая за позиции фигур на доске. Схема данного алгоритма представлена на рисунке 3.6.

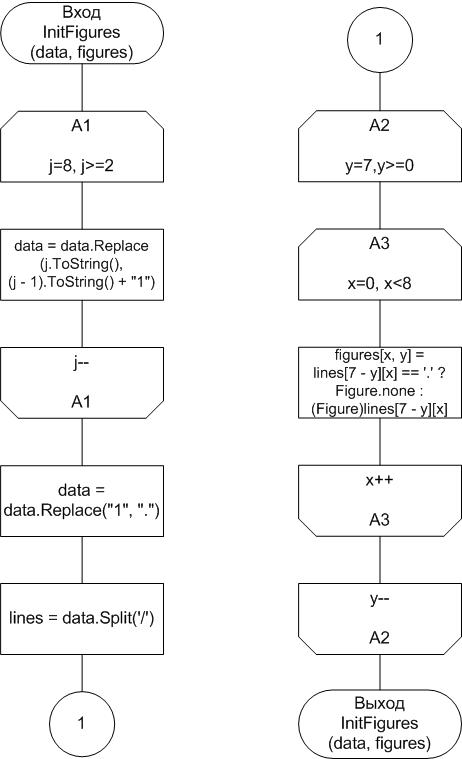


Рисунок 3.6 – Схема алгоритма InitFigures

В соответствие с выработанными требованиями были формализованы шахматные правила в отношении ходов и реализованы все возможные ходы за исключением рокировок и взятия на проходе. Также была учтена возможность превращения пешки в ферзя на первой горизонтали оппонента. Исходный код класса шахматных ходов предоставлен в приложении А.

разрушении игровой сессии

# Руководство по использованию программного средства

## Использование сервера

Серверная часть представляет собой консольное приложение, которое обеспечивает ввод данных администратора и вывод сообщений сервера.

При запуске консоли предлагается ввести IPv4 адрес и порт, на котором будет работать сервер. После ввода сервер запускается и ожидает подключений клиентов. Соответствующий вид консоли представлен на рисунке 5.1

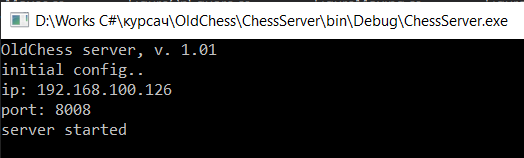


Рисунок 5.1 – Запуск сервера

При подключении пользователя, при создании им новой сессии и присоединении к существующей соответствующая информация отображается на консоль. Пример приведён на рисунке 5.2.

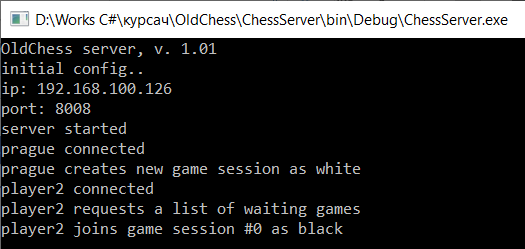


Рисунок 5.2 – Действия клиентов на сервере

Далее пользователи подтверждают сессию и начинается шахматная партия. При этом сервер отображает ходы, выполняемые игроками. Соответствующий вид приведён на рисунке 5.3.

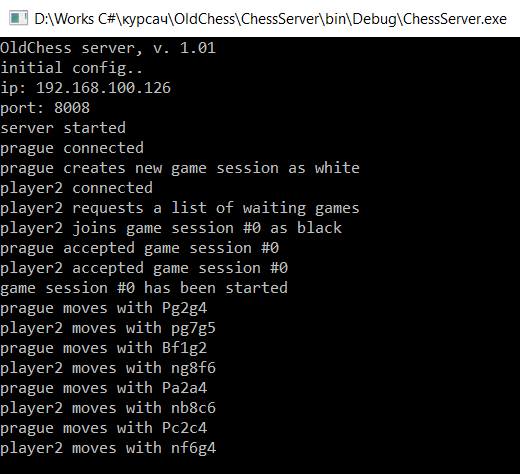


Рисунок 5.3 – Шахматная партия

В любой момент времени поддерживаются команды администратора. С помощью них можно получить информацию о пользователях, сессиях и попробовать остановить сервер. Заметим, что остановку можно выполнить лишь в том случае, когда на сервере нет подключённых пользователей. Пример приведён на рисунке 5.4.

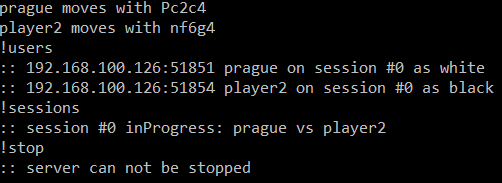


Рисунок 5.4 – Команды администратора

Пользователи могут в любой момент покинуть игру, либо отключится от сервера. В свою очередь сервер обеспечивает безопасность этого процесса. Вид окна консоли в таком случае представлен на рисунке 5.5.

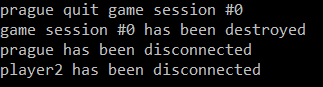


Рисунок 5.5 – Отключение клиентов

## Использование клиента

Клиент представляет собой приложение с графическим интерфейсом пользователя. Основную часть окна занимает шахматная доска и поля с информацией о статусе клиента, пользователе и оппоненте игрока. В главном меню главной является вкладка сервер, на которой расположены кнопки взаимодействия с сервером. Вид клиента представлен на рисунке 5.6.

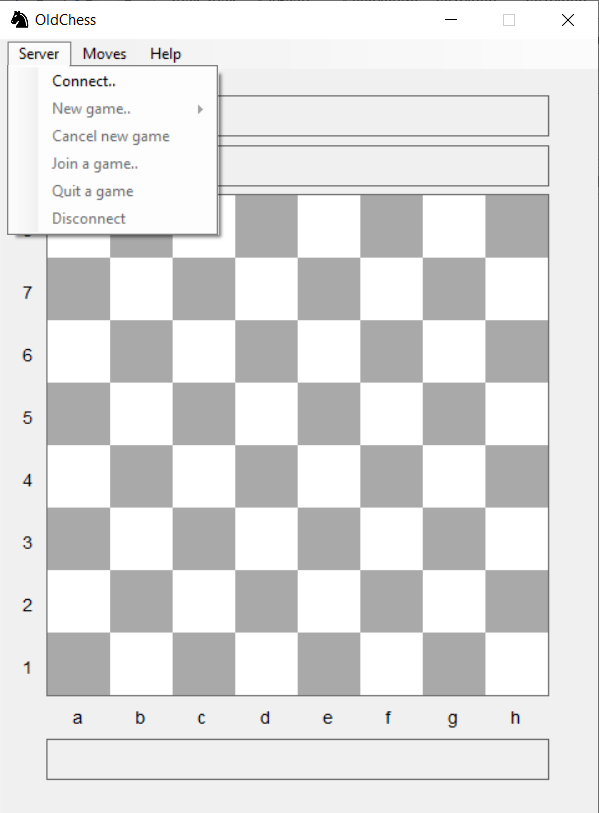


Рисунок 5.6 – Приложение клиента

При нажатии на кнопку «Connect» открывается другая форма, отвечающая за подключение к серверу. На ней будет предложено ввести желаемое имя, которое будет использоваться на сервере, а также IPv4 адрес сервера и его порт или выбрать их из списка доступных. Вид формы представлен на рисунке 5.7.

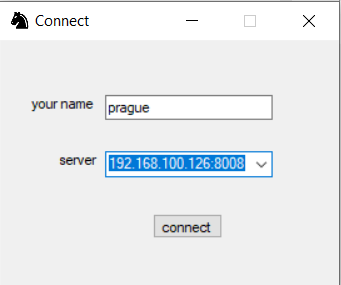


Рисунок 5.7 – Форма подключения

В случае неудачи в подключении либо неуникальности имени пользователя на сервере появится соответствующее уведомление. После подключения будет возможно создать новую сессию или присоединиться к существующей. После создания новой сессии, её возможно отменить кнопкой «Cancel new game». При нажатии кнопки «Join a game» появится форма со списком всех доступных сессий. Её вид показан на рисунке 5.8.

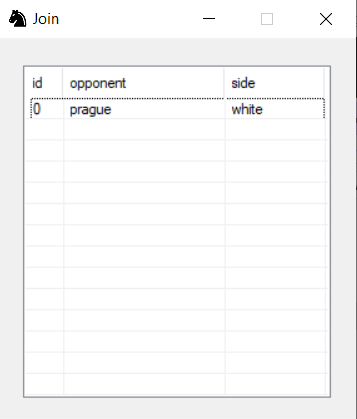


Рисунок 5.8 – Присоединение к сессии

При двойном клике по элементу списка произойдёт присоединение к сессии и обоим игрокам придёт уведомление о подтверждении. После подтверждения игры пользователями начинается шахматная партия. Игрок, который ходит, подсвечивается красным. Также подсвечиваются все доступные для хода фигуры. Соответствующий вид представлен на рисунке 5.9.

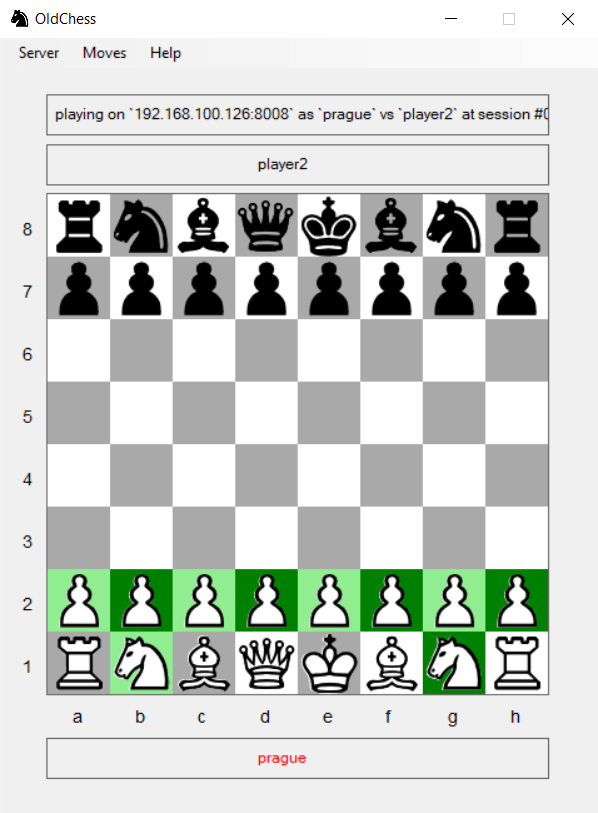


Рисунок 5.9 – Шахматная партия

После выбора фигуры отображаются все доступные для неё ходы. Нажав на вкладку «Moves» можно получить информацию о состоянии партии: её FEN и таблицу ходов. Данный аспект показан на рисунке 5.10.

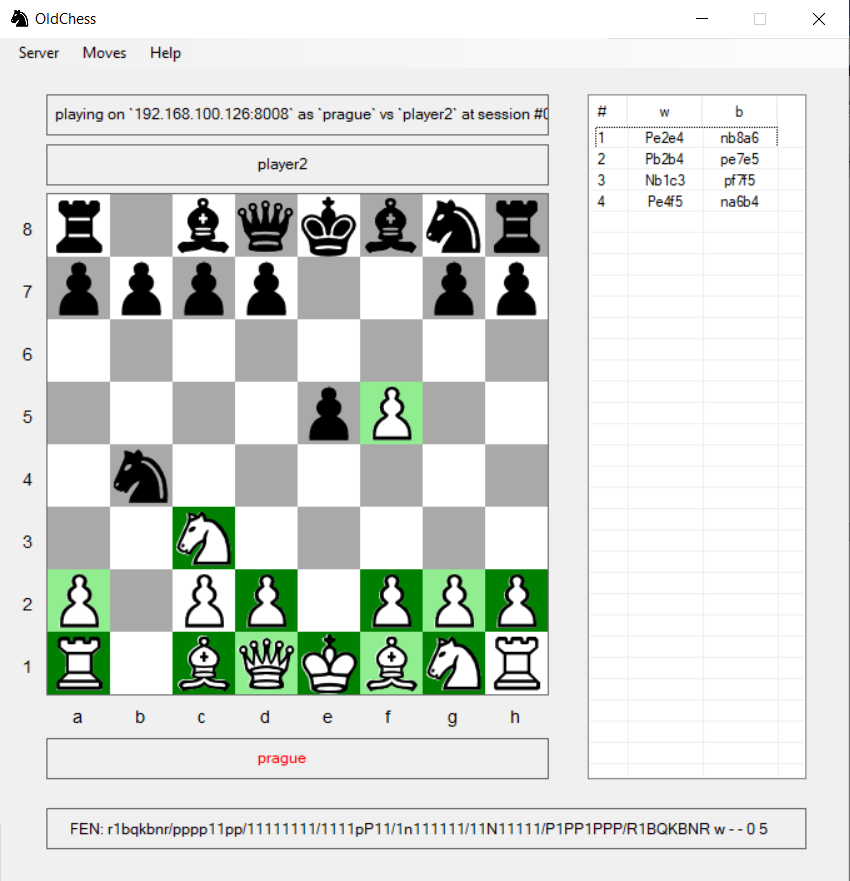


Рисунок 5.10 – Состояние шахматной партии

Игрок может покинуть игру или отключится от сервера в любой момент. Тогда его оппоненту придёт уведомление о том, что сессия разрушена. Игра заканчивается победой одного из игроков либо ничьей. Соответствующие уведомления показаны на рисунке 5.11. После этого пользователь может начать новую игру, либо переподключиться к другому серверу.

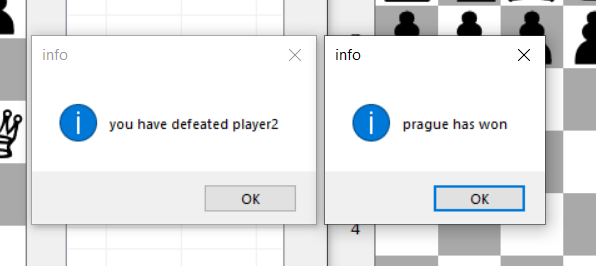


Рисунок 5.11 – Окончание партии матом

Заключение

В ходе разработки было создано многопоточное приложение с клиент-серверной архитектурой, предназначающееся для игры в шахматы по локальной сети. Помимо реализации игрового процесса в шахматной партии был разработан собственный текстовый протокол сообщений, обеспечивающий мощное взаимодействие клиента с сервером, а также предоставляющий определённую гибкость в работе клиента.

Данное приложение может быть улучшено, в первую очередь, с помощью улучшения графического интерфейса пользователя, добавления анимации перемещения фигур на шахматной доске и т.п. Также стоит реализовать специальные ходы в шахматах: рокировки и взятие на проходе. Во-вторых, в исходном коде клиент могут быть произведены существенные оптимизации, что повысит производительность приложения. Можно переработать механизм хода фигурой для повышения скорости работы. Однако следует заметить, что уже в настоящем виде приложение OldChess хорошо справляется со своей задачей.

В ходе написания данного курсового проекта в значительной ступени были улучшены знания объектно-ориентированного программирования, а также получены навыки моделирования процессов в клиент-серверной архитектуре. Приобретённые знания будут широко использоваться в дальнейшем.

Список использованной литературы

[1] Теория шахмат [Электронный ресурс] – Режим доступа:

[https://ru.wikipedia.org/wiki/Шахматы -](https://ru.wikipedia.org/wiki/Шахматы%20-) Дата доступа: 23.05.2020.

[2] Шахматный FEN [Электронный ресурс] – Режим доступа:

<https://ru.wikipedia.org/wiki/Нотация_Форсайта_—_Эдвардса> - Дата доступа: 23.05.2020.

[3] Класс TcpListener в C# [Электронный ресурс] – Режим доступа:

<https://metanit.com/sharp/net/4.2.php> - Дата доступа: 25.05.2020.

[4] Шахматный сервер Lichess [Электронный ресурс] – Режим доступа:

<https://lichess.org/about> - Дата доступа: 01.06.2020.

[5] Клиент-сервер [Электронный ресурс] – Режим доступа:

<https://ru.wikipedia.org/wiki/Клиент_—_сервер> – Дата доступа: 01.06.2020.

Приложение А

(обязательное)

Исходный код класса шахматных ходов

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Chess

{

class Moves

{

FigureMoving fm;

Board board;

public Moves(Board board)

{

this.board = board;

}

public bool CanMove(FigureMoving fm)

{

this.fm = fm;

return CanMoveFrom() && CanMoveTo() && CanFigureMove();

}

private bool CanMoveFrom()

{

return fm.from.onBoard() && fm.figure.GetColor() ==

board.moveColor;

}

private bool CanMoveTo()

{

return fm.to.onBoard() && fm.from != fm.to &&

board.GetFigureAt(fm.to).GetColor() != board.moveColor;

}

private bool CanFigureMove()

{

switch (fm.figure)

{

case Figure.whiteKing:

case Figure.blackKing:

return CanKingMove();

case Figure.whiteQueen:

case Figure.blackQueen:

return CanStraightMove();

case Figure.whiteRook:

case Figure.blackRook:

return (fm.SignX == 0 || fm.SignY == 0) &&

CanStraightMove();

case Figure.whiteBishop:

case Figure.blackBishop:

return (fm.SignX != 0 && fm.SignY != 0) &&

CanStraightMove();

case Figure.whiteKnight:

case Figure.blackKnight:

return CanKnightMove();

case Figure.whitePawn:

case Figure.blackPawn:

return CanPawnMove();

default: return false;

}

}

private bool CanPawnMove()

{

if (fm.from.y < 1 || fm.from.y > 6)

return false;

int stepY = fm.figure.GetColor() == Color.white ? 1 : -1;

return CanPawnGo(stepY) || CanPawnJump(stepY) ||

CanPawnEat(stepY);

}

private bool CanPawnGo(int stepY)

{

if (board.GetFigureAt(fm.to) == Figure.none)

if (fm.DeltaX == 0)

if (fm.DeltaY == stepY)

return true;

return false;

}

private bool CanPawnJump(int stepY)

{

if (board.GetFigureAt(fm.to) == Figure.none)

if (fm.DeltaX == 0)

if (fm.DeltaY == 2 \* stepY)

if (fm.from.y == 1 || fm.from.y == 6)

if (board.GetFigureAt(new Square(fm.from.x,

fm.from.y + stepY)) == Figure.none)

return true;

return false;

}

private bool CanPawnEat(int stepY)

{

if (board.GetFigureAt(fm.to) != Figure.none)

if (fm.AbsDeltaX == 1)

if (fm.DeltaY == stepY)

return true;

return false;

}

private bool CanStraightMove()

{

Square at = fm.from;

do

{

at = new Square(at.x + fm.SignX, at.y + fm.SignY);

if (at == fm.to)

return true;

} while (at.onBoard() && board.GetFigureAt(at) == Figure.none);

return false;

}

private bool CanKingMove()

{

if (fm.AbsDeltaX <= 1 && fm.AbsDeltaY <= 1)

return true;

else

return false;

}

private bool CanKnightMove()

{

if (fm.AbsDeltaX == 1 && fm.AbsDeltaY == 2 ||

fm.AbsDeltaX == 2 && fm.AbsDeltaY == 1)

return true;

return false;

}

}

}

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнительные сведения* | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР КП 1–40 01 01 421 ПЗ | | | | | Пояснительная записка | | | | 37 с. | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.851004-01 СА | | | | | Обработка шахматным сервером запросов клиентов | | | | Формат А1 | |
|  | | | | | Схема алгоритма | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 421 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | Игровое программное средство «Шахматы» по локальной сети  Ведомость курсового  проекта | Литера | | | Лист | Листов |
| Разраб. | | Пашкевич А.Л. |  | 05.06 | Т |  |  | 37 | 37 |
| Провер. | | Красковский П.Н. |  | 10.06 | Кафедра ПОИТ  гр. 851004 | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |