Содержание

[Введение 5](#_Toc73967530)

[1 Анализ существующих аналогов 6](#_Toc73967531)

[2 Постановка задачи 8](#_Toc73967532)

[3 Моделирование предметной области 9](#_Toc73967533)

[3.1 Правила игры “Русские шашки” 9](#_Toc73967534)

[3.2 Протокол управления передачей данных TCP 12](#_Toc73967535)

[4 Разработка программного средства 17](#_Toc73967536)

[4.1 Описание алгоритмов решения задачи 17](#_Toc73967537)

[4.2 Интерфейс программного средства 21](#_Toc73967538)

[5 Тестирование и проверка работоспособности программного средства 25](#_Toc73967539)

[5.1 Тестирование функционала программы 25](#_Toc73967540)

[6 Руководство по использованию программного средства 33](#_Toc73967541)

[6.1 Работа с приложением 33](#_Toc73967542)

[Заключение 37](#_Toc73967543)

[Список использованных источников 38](#_Toc73967544)

[Приложение А 39](#_Toc73967545)

[Приложение Б 41](#_Toc73967546)

[Приложение В 42](#_Toc73967547)

Введение

Шашки входят в категорию так называемых игр шашечного типа, включающую множество разнообразных настольных игр для двух-четырёх игроков, в которые играют на расчерченной доске равноценными фишками. Кроме шашек, в неё также входят такие игры, как го, нарды и другие.

Достоверных сведений об изобретении шашек история не сохранила. Вполне вероятно, что различные игры шашечного типа изобретались на протяжении истории человечества многократно и независимо, чему способствовала простота инвентаря и правил. Считается, что известные сейчас игры шашечного типа представляют собой различные этапы эволюции игр на доске, так что все они в определённом смысле родственны друг другу, что подтверждается близостью инвентаря и названий игр.

Шашки - это не просто настольная логическая игра, а целая наука, в которой важна каждая деталь в ходе игры. Правила зависят от разновидности и могут кардинально отличаться от классических. Среди наиболее распространенных видов можно выделить Русские шашки, Английские, Поддавки, Уголки и т.д.

В работе предусмотрены следующие моменты:

- выбор стороны;

- создание удобного интерфейса;

- отображение счета и информации в консоли;

- игра человек-человек.

Целью данного проекта стало создание шашек по сети, способных работать по локальной сети.

При реализации шашек по сети затрагиваются следующие темы:

* программирование клиент-серверного приложения;
* многопоточное программирование;
* протокол TCP;

# Анализ существующих аналогов

Существует огромное количество программ, реализующих шашки. Среди них есть как обычные приложения, где возможна лишь игра с компьютером, так и веб-приложения.

Например веб приложение на сайте <https://logic-games.spb.ru>. Интерфейс шашек представлен на рисунке 1.1.



Рисунок 1.1 – Интерфейс шашек

На данном сайте имеется возможность как игры с компьютером, так и игры по сети. Можно создавать свои турниры, лобби, присутствует рейтинг, статистика количества матчей. Предлагаются разные вариации шашек.

Из обычных предложений можно рассмотреть EasyChekers. Можно играть только с компьютером. Из вариаций также есть большой набор видов шашек. Есть возможность сохранения и загрузки игры. Есть описание, можно посмотреть правила игры.

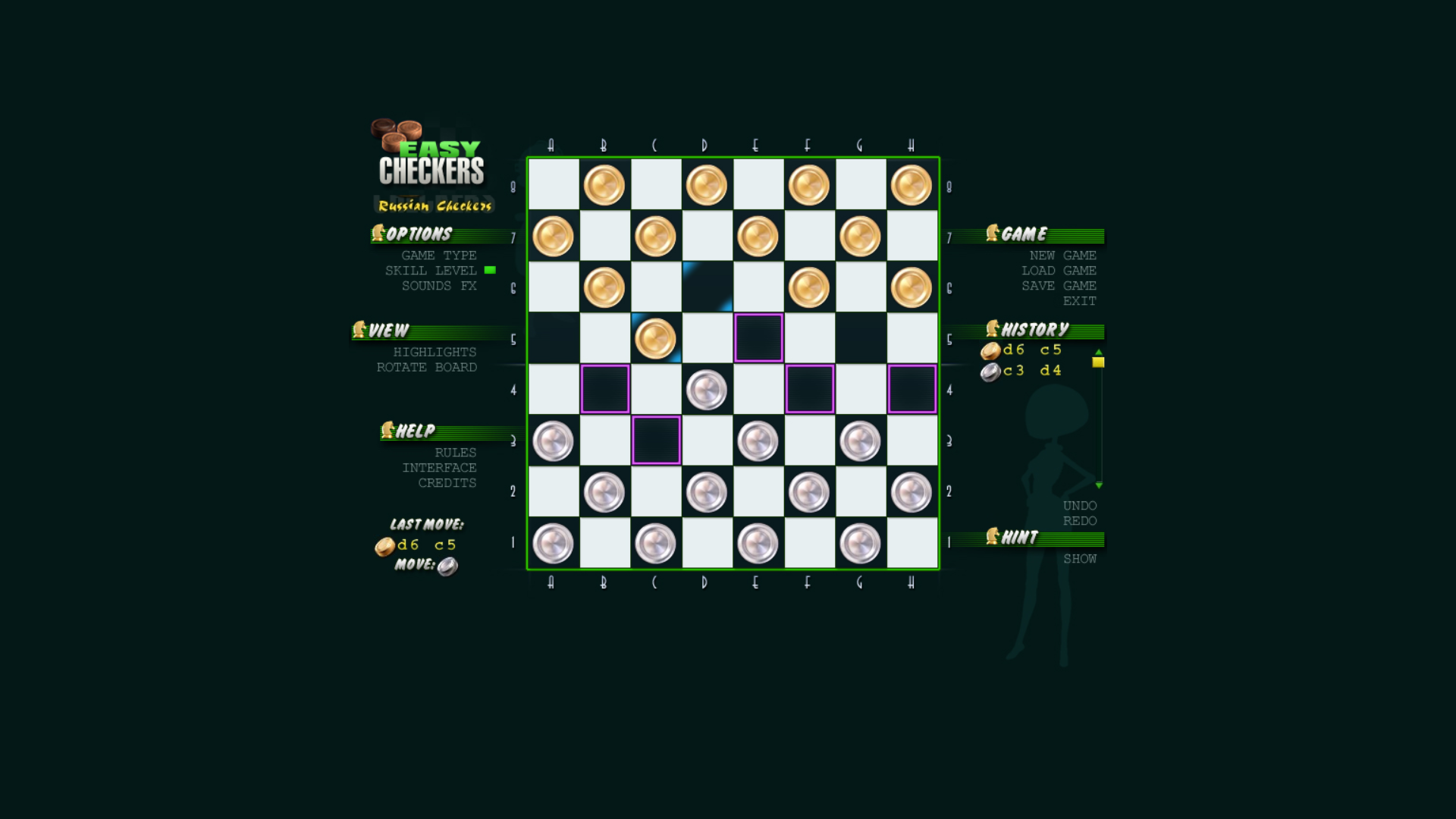


Рисунок 1.2 – Интерфейс программы EasyCheckers

В EasyCheckers также есть возможность регулировать уровень игры искусственного интеллекта.

# Постановка задачи

Целью курсового проекта является создание программного средства, представляющее простейший аналог игры в шашки, которое может выступать в качестве сервера и клиента в локальной сети. Передачу данных должна быть реализована посредством протокола TCP.

Требования к пользовательскому интерфейсу:

* игровое поле;
* выбор стороны;
* возможность создания сервера для игры;
* возможность подключения к существующей игре;
* консоль с выводом игровых сообщений

Любой пользователь может подключиться к другому пользователю, зная его IP-адрес. В этом случае, пользователям выведется сообщение о установке соединения и их стороне.

Разработка программного средства должна производиться на языке программирования С# (программная платформа ­­– Microsoft .NET). Пользовательский интерфейс должен разрабатываться с использованием Windows Representation Foundation (WPF).

# Моделирование предметной области

## Правила игры “Русские шашки”

Игровое поле – обычно квадрат 8×8, шашки занимают первые три ряда с каждой стороны. Вертикали (столбцы) обозначаются латинскими буквами от a до h, а горизонтали (строки) - цифрами от 1 до 8. В отличие от шахмат игровыми считаются не все, а только темные поля

Каждая из сторон в начале игры имеет по 12 шашек. Шашки расставляются на трех, ближайших к игрокам, горизонталях. В частности, белые шашки располагаются на полях a1, c1, e1, g1, b2, d2, f2, h2, a3, c3, e3 и g3, а черные, соответственно, на полях h8, f8, d8, b8, g7, e7, c7, a7, h6, f6, d6 и b6. Два центральных ряда полей остаются свободными. Здесь, на этих полях, происходит сближение и первое соприкосновение противостоящих сил, здесь с первых же ходов разворачивается борьба за обладание центром, что дает весомое преимущество в шашках. Расстановка начальной позиции показана на диаграмме слева.

Противники ходят поочередно, перемещая шашки своего цвета по игровым полям.

Первыми начинают белые. Возникающие на доске ситуации называют позициями, или положениями. Проведенная от начала и до завершения игра называется партией, а передвижения шашек - ходами.

Выигрывает та сторона, которой удалось уничтожить или заблокировать движение всех шашек противника.

Простые шашки показаны на рисунке 3.2

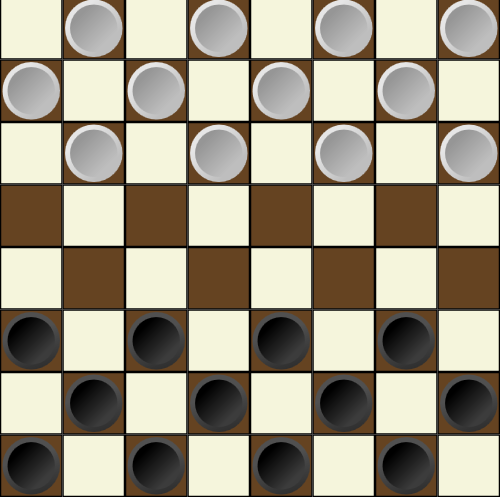


Рисунок 3.2 – простые шашки

Простая шашка может совершать, в зависимости от позиции, ходы типов: тихие и ударные.

Тихий ход - перемещение на одно поле вперед по диагонали. Тихий ход возможен, если соответствующее поле свободно и нигде на доске не требуется выполнить взятие шашки противника. Например, правильным ходом из начальной позиции будет движение шашки с поля c3 например на d4 (в нотации партии это обозначается как c3-d4), или на b4 (c3-b4).

Ударный ход (бой шашки противника) - перемещение на два поля вперед или назад по диагонали, через шашку противника. Шашка при своем ходе обязана побить (взять) шашку противника, если та находится на соседнем (по диагонали) поле и если следующее за ней поле свободно. Берущая шашка становится на это свободное поле, перескакивая через вражескую шашку, которая снимается с доски. Взятие может совершаться не только вперед, но и назад.

Если после взятия одной шашки оказывается возможным побить еще одну шашку противника, взятие продолжается, то есть в один прием (за один ход) шашка должна побить столько шашек соперника, сколько их стоит на ее пути. Взятия в русских шашках обязательны.

При возможности взятия в разных направлениях выбор, вне зависимости от количества или качества снимаемых шашек, предоставляется берущему игроку. При взятии нескольких шашек противника снимать их с доски можно только после завершения своего ударного хода.

Запрещается при взятии перескакивать более одного раза бьющей шашкой (дамкой) через одну и ту же шашку соперника (а вот на свободное поле наступать несколько раз разрешается).

Дамки показаны на рисунке 3.3

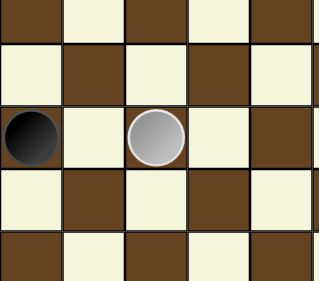


Рисунок 3.3 - Дамки

Если простая шашка достигает последней горизонтали, она становится дамкой. Дамки, в отличие от обычных шашек, способны перемещаться на произвольное количество полей по выбранной диагонали, причем как вперед, так и назад.

Дамка при своем ходе обязана взять шашку соперника (как вперед, так и назад) независимо от количества свободных полей до нее, если только эта шашка находится на одной диагонали с дамкой и за этой шашкой имеется одно или несколько свободных полей. Причем в последнем случае дамка, совершив взятие шашки противника, может остановиться на любом из них.

Если же при взятии на любой из пересекающихся диагоналей также находятся шашки соперника, за которыми имеются свободные поля, то дамка обязана продолжать взятие и этих шашек, сколько бы их ни находилось на ее пути.

Так же, как и в случае с простыми шашками, при наличии нескольких способов выполнения ударного хода (одной и той же или разными дамками), выбор последнего остается за берущим игроком.

Если обычная шашка достигает последней горизонтали в результате боя шашки противника (где ей полагается превращаться в дамку), и если ей предоставляется возможность дальнейшего взятия вражеских шашек, то она обязана тем же ходом продолжать бой, но уже на правах дамки.

Обратите внимание: это правило действует лишь в том случае, если шашка достигает последней горизонтали в результате ударного хода. В случае, если простая шашка достигнет последнего ряда без взятия и ей после этого представится возможность боя, то она должна бить (если такая возможность сохранится) лишь следующим ходом.

## Протокол управления передачей данных TCP

TCP (Transmission Control Protocol) – один из основных [протоколов передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) интернета. Предназначен для управления [передачей данных интернета](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). TCP используется как надежный протокол, обеспечивающий взаимодействие через взаимосвязанную сеть компьютеров. TCP проверяет, что данные доставляются по назначению и правильно. Структура TCP-пакета представлена на рисунке 3.4.

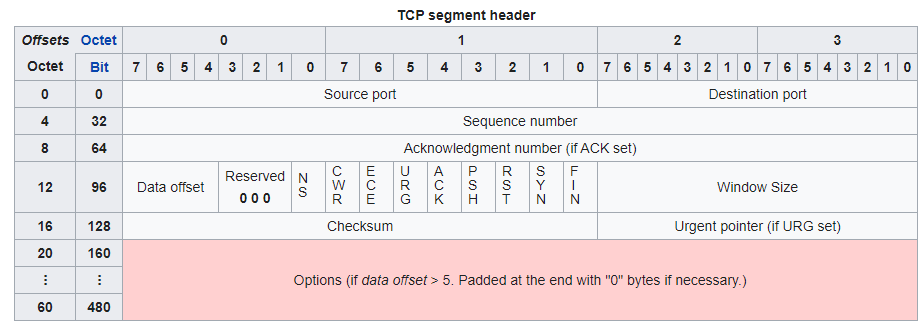


Рисунок 3.4 – Структура TCP-пакета

### Заголовок TCP

В заголовке TCP содержатся следующие поля:

* Порт отправителя (Source port, 16 бит).
* Порт получателя (Destination port, 16 бит).
* Порядковый номер (Sequence number, 32 бита). Позволяет контролировать порядок сообщений. Каждая конечная точка (как порт источника, так и порт назначения) будут поддерживать свой уникальный порядковый номер для отправляемых сообщений. При установлении соединения TCP (используется сообщение с установленным флагом SYN) в качестве изначального порядкового номера будет сгенерировано случайное число. Вернее, не совсем случайно сгенерировано, а будет содержать конкретное 32-битное число, то есть в пределах от 0 до 4294967295, которое будет соответствовать времени, прошедшему после перегрузки системы отправителя (из расчета +1 за каждые прошедшие 4 микросекунды), а также увеличенное на 64000 каждый раз при установлении нового соединения. Так как сгенерированное число будет уникальным для периода времени почти в пять часов (если при этом никакие соединения не устанавливались), то такой подход к выбору порядкового номера позволяет избежать случайных коллизий при передаче данных, когда для нескольких пакетов из разных соединений будет совпадать порядковый номер. В дальнейшем, при отправке следующих пакетов, значение порядкового номера будет увеличиваться на +1 для всех пакетов с флагом SYN, пакетов с флагом FIN и для каждого байта отправленных данных. Это позволяет принимающей системе обрабатывать пакеты в правильной последовательности, как они были сформированы при отправлении, а не в том порядке, как они были получены.
* Номер подтверждения (Acknowledgement number, 32 бита). Когда сообщение содержит флаг ACK, то значение в номере подтверждения должно соответствовать следующему порядковому номеру (SYN), которое отправитель сообщения с флагом ACK ожидает получить от передающей системы. Таким образом, отправка одного номера подтверждения способна подтвердить получение всех байтов с информацией, полученных до этого.
* Длина заголовка (Data offset, 4 бита). Содержит размер заголовка TCP, измеряемый в 32-битных сегментах. Минимальный размер заголовка TCP составляет пять 32-битных сегментов (всего 20 байт), а максимальный –пятнадцать 32-битных сегмента (или 60 байт).
* Зарезервировано (Reserved, 3 бита). Зарезервировано для будущего использования, пока просто забивается нулями. На данный момент осталось три незадействованных бита, в то время как еще три ранее зарезервированных бита уже используются как флаги.
* Флаги (Flags, 9 бит):
* NS, одноразовая сумма (Nonce Sum, 1 бит). Используется для улучшения работы механизма явного уведомления о перегрузке (Explicit Congestion Notification, ECN).
* CWR, окно перегрузки уменьшено (Congestion Window Reduced, 1 бит). Данный флаг устанавливается отправителем, чтобы показать, что TCP-фрагмент был получен с установленным полем ECE. Таким образом, это является подтверждением получения пакета данных с флажком ECE от хоста получателя и включением отправителем механизма уменьшения перегрузки (Congestion Control), позволяющим оптимизировать отправку пакетов с данными в перегруженных сетях, избежав серьезных задержек из-за отбрасывания пакетов.
* ECN-Эхо (ECN-Echo, 1 бит). Выполняет двойственную роль, в зависимости от значения флага SYN. При установленном флаге SYN это указывает на то, что отправитель пакета поддерживает ECN. Если флаг SYN сброшен (SYN=0), а ECE установлен, то это означает, что пакет с установленным флагом CE (Congestion Experienced, Подтвержденная перегрузка) был получен в заголовке IP во время обычной передачи. Таким образом, это служит индикатором перегрузки сети (или предстоящей перегрузки) для TCP-отправителя.
* URG (1 бит). Устанавливается, если необходимо передать ссылку на поле указателя срочности (Urgent pointer).
* ACK (1 бит). Устанавливается, когда пакет содержит значение номера подтверждения в поле подтверждения. Все пакеты после стартового пакета SYN будут иметь установленный флаг ACK.
* PSH (1 бит). Делает этот пакет пакетом PUSH (проталкивания). При нормальном потоке передачи данных система получателя не будет подтверждать получение каждого пакета сразу же после его получения. Вместо этого система получателя в течении некоторого времени будет собирать и хранить полученные данные в буфере, пока не передаст их приложению пользователя. Пакет PUSH инструктирует систему получателя немедленно передать все полученные ранее данные из буфера в приложение пользователя и сразу же отправить сообщение с подтверждением.
* RST (1 бит): сброс данного соединения. Отправкой пакета RST одна из сторон сообщает о немедленном разрыве соединения. При этом соединение обрывается, а буфер очищается.
* SYN (1 бит). Начинает соединение и синхронизирует порядковые номера. Первый пакет, отправленный с каждой стороны, должен в обязательном порядке иметь установленным этот флаг.
* FIN (1 бит). Одна из конечных точек отправляет пакет с установленным флагом FIN для другой конечной точки, чтобы сообщить, что все пакеты были отправлены, и соединение пора завершить.
* Размер окна приема (Window size, 16 бит). В нем указывается количество байт данных, считая от последнего номера подтверждения, которые готов принять отправитель данного пакета. Другими словами, отправитель данного пакета в этом поле сообщает другой стороне, каким доступным на данный момент размером буфера приема данных он располагает.
* Контрольная сумма (Checksum, 16 бит). Используется для проверки на наличие ошибок при передаче и/или приеме отправленного пакета. Рассчитывается с учетом заголовка (все поля заголовка, кроме самой контрольной суммы), полезной нагрузки, а также псевдо-заголовка (IP-адрес источника, IP-адрес назначения, номер протокола и длина TCP-сегмента, в которой учитывается как длина полей заголовка, так и длина данных полезной нагрузки).
* Указатель срочности (Urgent pointer, 16 бит). Если установлен флаг URG, то это означает, что поле указателя срочности содержит численное значение положительного смещения от порядкового номера в сообщении, указывающее на последний байт срочных данных.

### Механизм передачи сообщений TCP

Перед тем, как данные могут быть переданы между двумя узлами, в TCP, в отличие от UDP, предусмотрена стадия установки соединения. После того, как все данные были переданы, наступает стадия завершения соединения. Таким образом, осуществление каждого TCP-соединения можно условно разделить на три фазы:

1. Инициализация соединения. Установка соединения осуществляется с помощью, так называемого трехстороннего рукопожатия TCP (рисунок 3.5). Инициатором соединения может выступать любая сторона.

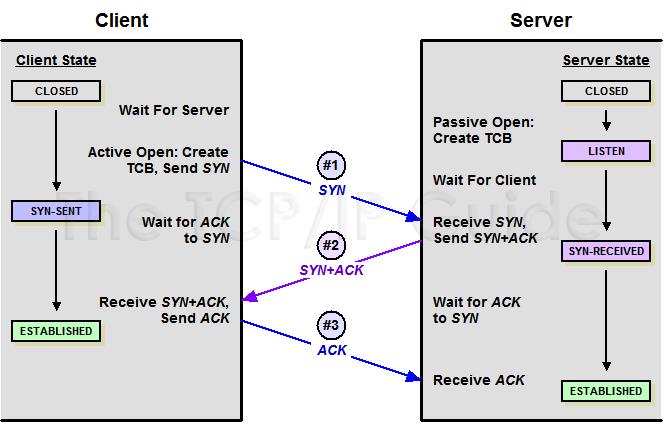


Рисунок 3.5 – Трехстороннее рукопожатие TCP

2. Загрузка данных. После инициализации соединения полезная нагрузка будет перемещаться в обоих направлениях TCP-соединения. Все пакеты в обязательном порядке будут содержать установленный флаг ACK. Другие флаги, такие как, например, PSH или URG, могут быть, а могут и не быть установленными.

3. Завершения соединения. Клиент посылает флаг FIN на завершение соединения. Сервер отправляет клиенту флаги ответа ACK, FIN, что соединение закрыто. После получения этих флагов клиент закрывает соединение и в подтверждение отправляет серверу ACK, что соединение закрыто.

TCP не посылает один пакет, ожидая прихода подтверждения, чтобы послать следующий. Вместо этого он использует принцип «скользящего окна». Этот принцип позволяет послать несколько сообщений и только потом ожидать подтверждения. «Окно» – это объем данных, который может принять получатель. Между прикладной программой и потоком данных в сети располагается буфер данных. Размер «окна» фактически представляет собой разность между размером буфера и объемом сохраненных в нем данных. Это число отправляется в заголовке, чтобы информировать удаленный хост о текущем размере окна. Такой прием называется «скользящим окном», принцип его работы схематично изображен на рисунке 3.6.

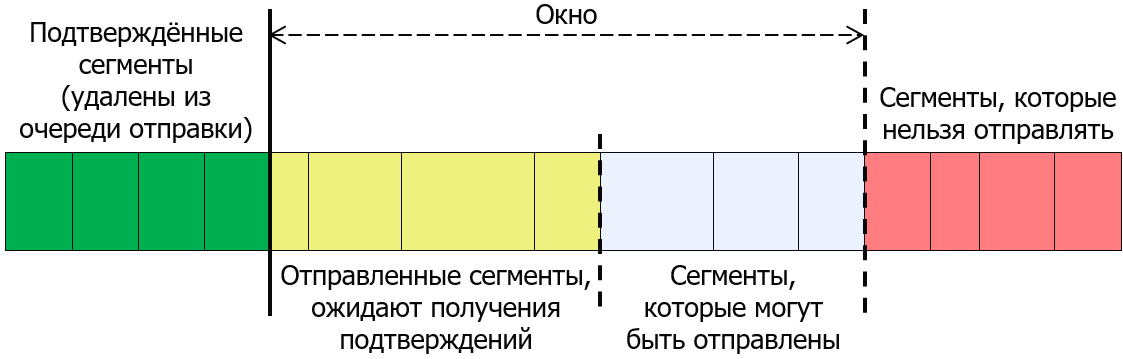


Рисунок 3.6 – Механизм скользящего окна

# Разработка программного средства

## Описание алгоритмов решения задачи

Исходя из поставленной задачи разработки программного средства, представляющее сетевую игру «Русские шашки» был четко определен функционал программного средства. Ниже рассмотрены ключевые моменты разработки клиентской и серверной частей программного средства, а также алгоритмов обработки кликов игрока по игровому полю и шашкам.

### Приложение-клиент

Основной задачей клиентской части приложения является установка соединения с сервером, принятие и отправка данных и сообщений. Также клиентская часть должна корректно обрабатывать данные, полученные от сервера. Схема основного алгоритма подключения клиента к серверу и получение данных от сервера представлена на рисунке 4.1.



Рисунок 4.1 – Схема подключения к серверу

Как видно из схемы, приложение-клиент обеспечивает получение данных: в цикле производится чтение из сетевого потока данных между клиентом и сервером. Приложение-клиент также обеспечивает отправку данных и сообщений, которая заключается в преобразовании объекта, в поток байтов (сериализация), а также записи полученного потока байтов в сетевой поток между клиентом и сервером.

### Приложение-сервер

Теперь рассмотрим работу серверного приложения. Схема алгоритма работы серверной части представлена на рисунке 4.2.



Рисунок 4.2 – Схема работы сервера

### Обработчик клика по шашке

Рассмотрим взаимодействие игрока с шашками на игровом поле. Схема алгоритма представлена на рисунке 4.3



Рисунок 4.3 – Алгоритм обработки клика на шашку

Алгоритм представленный на рисунке 4.3 нужен для взаимодействия игрока с игровым полем и его составляющим. Благодаря данной механике пользователь может смотреть возможные варианты ударным или тихих ходов.

### Обработчик клика по игровому полю

Рассмотрим взаимодействие игрока с игровым полем. Схема алгоритма представлена на рисунке 4.4



Рисунок 4.4 – Алгоритм обработки клика на шашку

Алгоритм представленный на рисунке 4.4 нужен для взаимодействия игрока с игровым полем. Данный алгоритм служит вспомогательным элементом для обработчика кликов по шашкам. Данный алгоритм реализует передвижение шашек по игровому полю и передачу данных о ходе сопернику.

## Интерфейс программного средства

Интерфейс обеспечивает комфортное взаимодействие между пользователем и приложением. Окно приложения логически разделено на разные зоны по предоставляемому функционалу. Присутствует корректная обработка ошибок, в случае неправильного ввода информации. Согласно поставленным задачам, интерфейс программного средства написан с помощью технологии Windows Presentation Foundation (WPF).

Технология WPF является частью экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов. Если при создании традиционных приложений на основе WinForms за отрисовку элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на DirectX [7]. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML: есть возможность создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C# и VB.NET, либо совмещать и то, и другое.

Схематически архитектуру WPF представлена на рисунке 4.5.

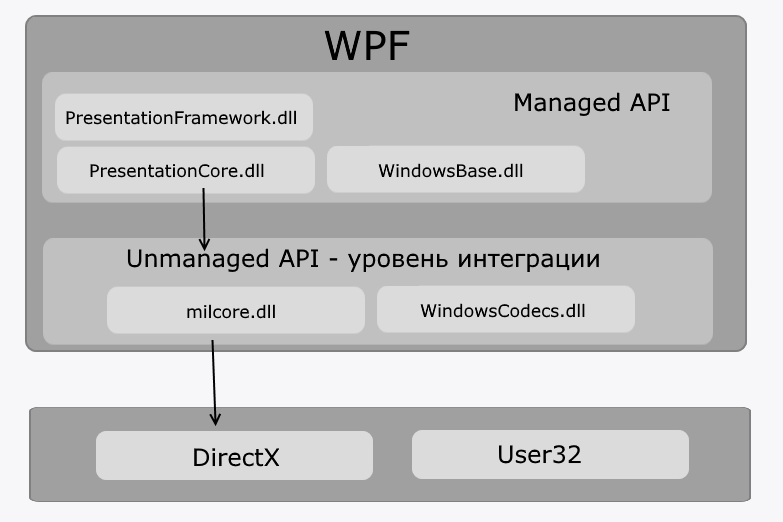


Рисунок 4.5 – Архитектура WPF

Как видно на схеме, WPF разбивается на два уровня: managed API и unmanaged API (уровень интеграции с DirectX). Managed API (управляемый API-интерфейс) содержит код, исполняемый под управлением общеязыковой среды выполнения .NET - Common Language Runtime. Этот API описывает основной функционал платформы WPF и состоит из следующих компонентов:

* PresentationFramework.dll: содержит все основные реализации компонентов и элементов управления, которые можно использовать при построении графического интерфейса;
* PresentationCore.dll: содержит все базовые типы для большинства классов из PresentationFramework.dll;
* WindowsBase.dll: содержит ряд вспомогательных классов, которые применяются в WPF, но могут также использоваться и вне данной платформы.

Unmanaged API используется для интеграции вышележащего уровня с DirectX:

* milcore.dll: обеспечивает интеграцию компонентов WPF с DirectX. Данный компонент написан на неуправляемом коде (С/С++) для взаимодействия с DirectX;
* WindowsCodecs.dll: библиотека, которая предоставляет низкоуровневую поддержку для изображений в WPF.

Еще ниже находятся компоненты операционной системы и DirectX, которые производят визуализацию компонентов приложения, либо выполняют прочую низкоуровневую обработку. В частности, с помощью низкоуровневого интерфейса Direct3D, который входит в состав DirectX, происходит трансляция.

Здесь также на одном уровне находится библиотека user32.dll. И хотя WPF не использует эту библиотеку для рендеринга и визуализации, однако для ряда вычислительных задач (не включающих визуализацию) данная библиотека продолжает использоваться.

Главное окно приложения изображено на рисунке 4.6. На главном окне можно выделить 3 логических зоны: игровое поле, меню подключения и создания сервера с выбором стороны, меню с консолью.

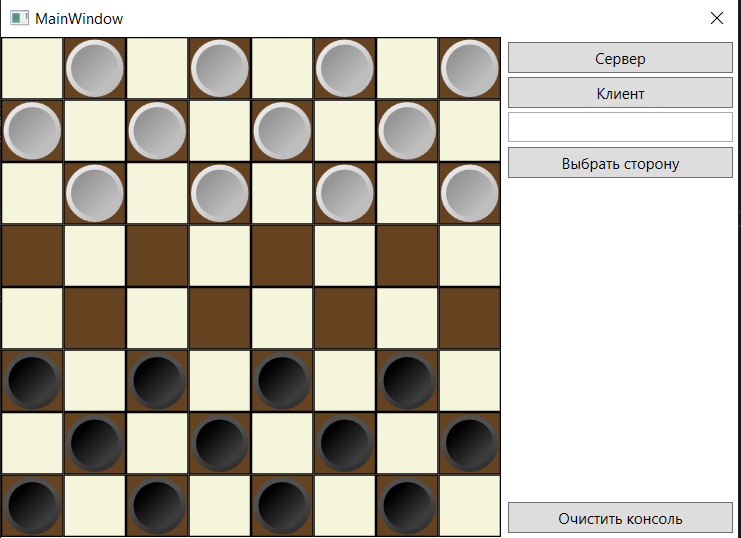


Рисунок 4.6 – Главное окно приложения

При нажатии на кнопку “Выбрать сторону” появляется окно изображенное на рисунке 4.7

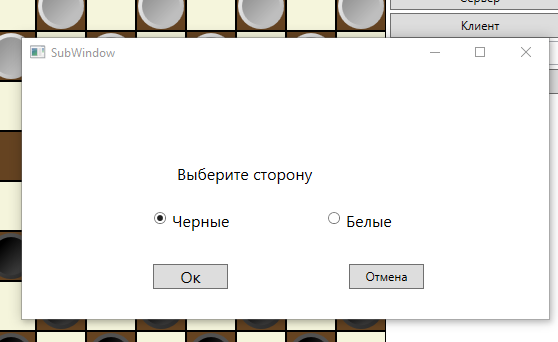


Рисунок 4.7 – Окно для выбора стороны

Данное окно позволяет игроку создающему сервер выбрать сторону за которую он хочет играть. Для применения изменений нужно нажать на кнопку “Ок”.

# Тестирование и проверка работоспособности программного средства

## Тестирование функционала программы

Таблица 5.1 – Тестирование функционала программы

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Тестируемая**  **функциональность** | **Действия** | **Ожидаемый и полученный**  **результаты** |
| 1 | Запуск серверного приложения. | Запуск серверного приложения на компьютере-сервере и ввод действительного IP адреса. | Приложение запущено, сервер доступен для подключения. |
| 2 | Запуск клиентского  приложения и  подключение к  серверу. | Запуск клиентского приложения на компьютере клиента и ввод действительного IP адреса для подключения. | Приложение запущено, клиент подключен к серверу. |
| 3 | Обработка исключения при попытке создать сервер с некорректным IP. | Ввод некорректного IP адреса. | Вывод сообщения о недопустимом IP адресе. |
| 4 | Обработка исключения при попытке подключиться к серверу с некорректным IP. | Ввод некорректного IP адреса. | Вывод сообщения о недопустимом IP адресе. |
| 5 | Обработка исключения при попытке создать сервер с IP адресом не из локальной сети. | Ввод IP адреса не из локальной сети. | Вывод сообщения о невозможности подключиться к данному адресу |
| 6 | Обработка исключения при попытке подключиться к серверу с IP адресом не из локальной сети. | Ввод IP адреса не из локальной сети. | Вывод сообщения о невозможности подключиться к данному адресу |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер теста** | **Тестируемая**  **функциональность** | **Последовательность действий** | **Ожидаемый и полученный**  **результаты** |
| 7 | Обработка исключения при попытке подключиться к IP адресу в локальной сети когда сервер не создан. | Ввод действительного IP адреса для подключения. | Вывод сообщение о не установке соединения. |

На скриншотах 5.1-5.7 представлены результаты выполнения тестов, указанных в таблице 5.1

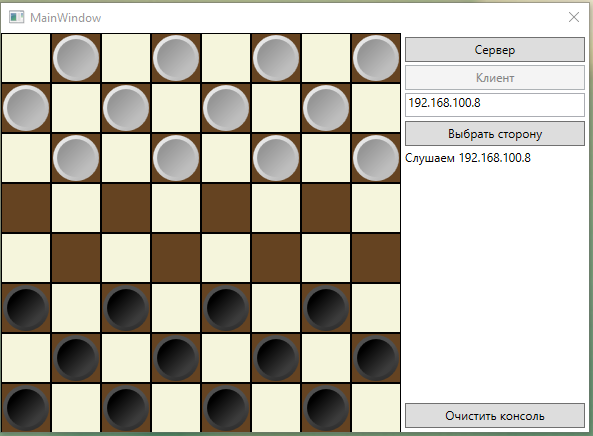


Рисунок 5.1 – Запуск серверного приложения

Из рисунка 5.1 видно, что при вводе действительного IP адреса начинается его прослушивание.



Рисунок 5.2 – Подключение к серверу

Из рисунка 5.2 видно, что соединение с сервером установлена и игра начата.

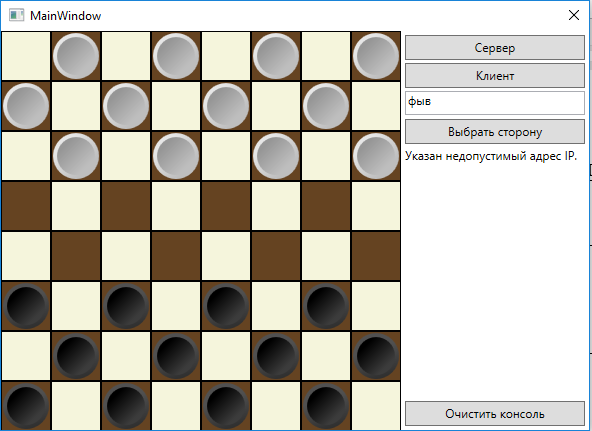


Рисунок 5.3 – Ввод некорректного IP адреса(сервер)

Из рисунка 5.3 видно, что исключение при вводе некорректного IP адреса при создании сервера обработано.

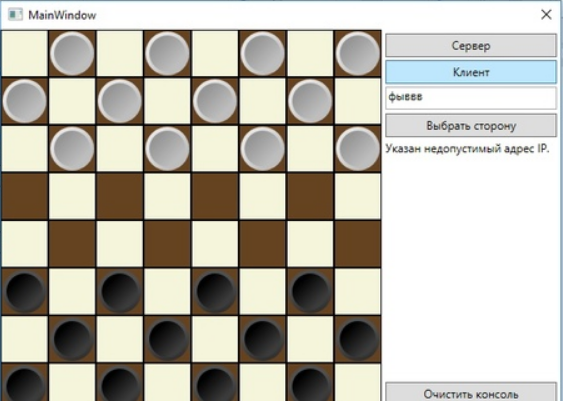


Рисунок 5.4 – Ввод некорректного IP адреса(клиент)

Из рисунка 5.4 видно, что исключение при вводе некорректного IP адреса при попытке подключения обработано.

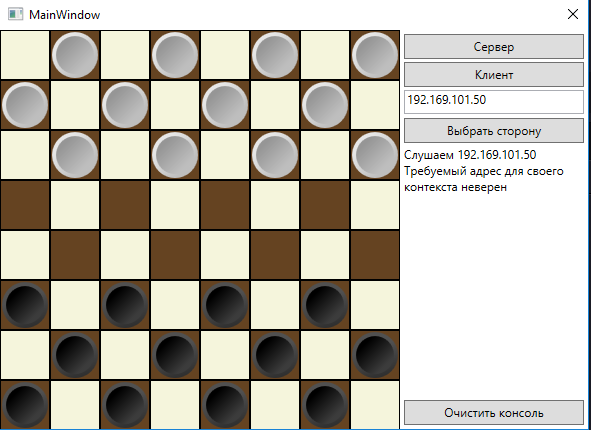


Рисунок 5.5 – Ввод IP адреса не из локальной сети(сервер)

Из рисунка 5.5 видно, что исключение при вводе IP адреса находящегося вне локальной сети обработано.

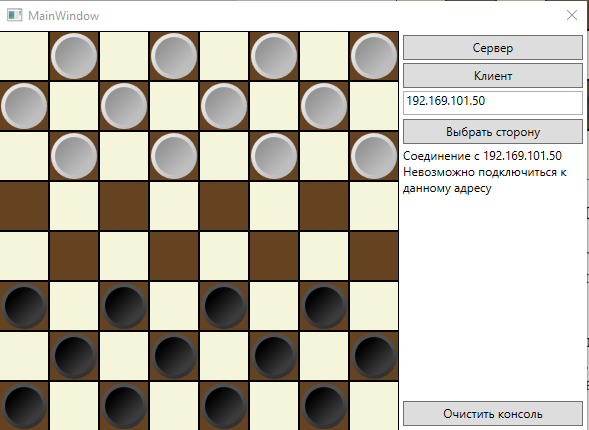


Рисунок 5.6 – Ввод IP адреса не из локальной сети(клиент)

Из рисунка 5.6 видно, что исключение при вводе IP адреса находящегося вне локальной сети и попытке подключения к нему обработано.



Рисунок 5.7 – Ввод действительного IP адреса когда сервер не создан

Из рисунка 5.7 видно, что обработано исключение при вводе действительного IP адреса когда сервер не создан.

# Руководство по использованию программного средства

## Работа с приложением

Для начала использования приложения необходимо распаковать архив и запустить файл Checkers.exe. Появится главное окно программы(рисунок 6.1).

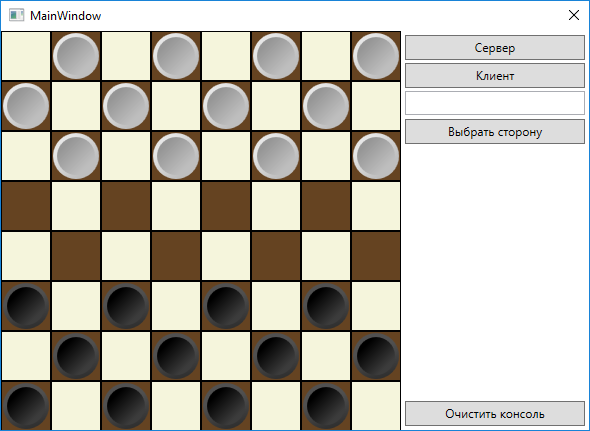


Рисунок 6.1 – Главное окно программы

Для начала игры пользователь должен найти соперника. В Правой верхней части интерфейса находятся компоненты для подключения к сети. Для создания своей игры нужно ввести свой IP адрес и нажать на кнопку “Сервер”. Тот, кто создает игру, может выбрать сторону, за которую хочет играть. По стандарту создатель игры играет за белых. Окно для выбора стороны представлено на рисунке 6.2.

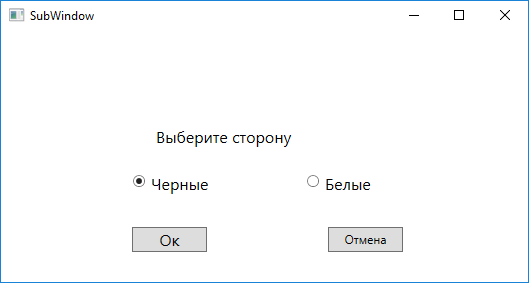


Рисунок 6.2 – Окно для выбора стороны

После создания игры пользователю остается ожидать, когда к нему подсоединится второй игрок.

Для подключения к существующей игре пользователю нужно ввести IP адрес создавшего игру и нажать на кнопку “Клиент”.

В конечном итоге, в независимости от того, являетесь вы клиентом или сервером, при успешном соединении в консоли должно вывестись сообщение показанное на рисунке 6.3.

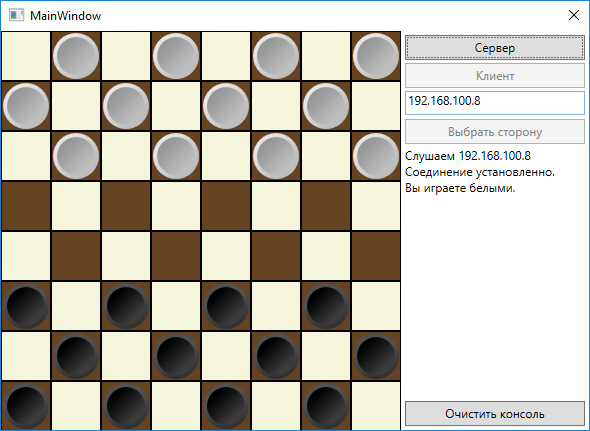


Рисунок 6.3 – Сообщение о старте игры

Для управления шашками используется компьютерная мышь. При нажатии на вашу шашку вы будете видеть доступные для нее ходы.

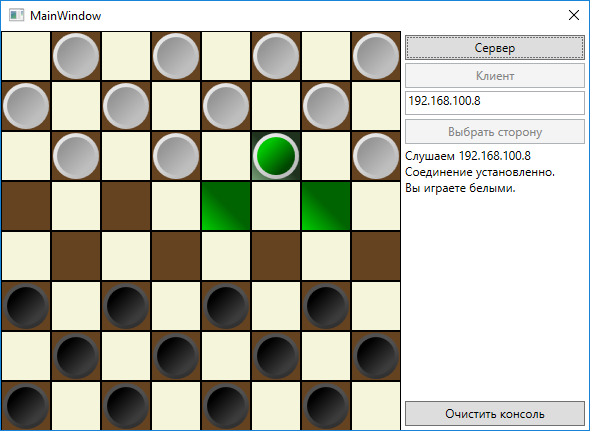


Рисунок 6.4 – Возможные варианты ходов

После клика на один из подсвеченных квадратиков шашка поменяет своей местоположение.

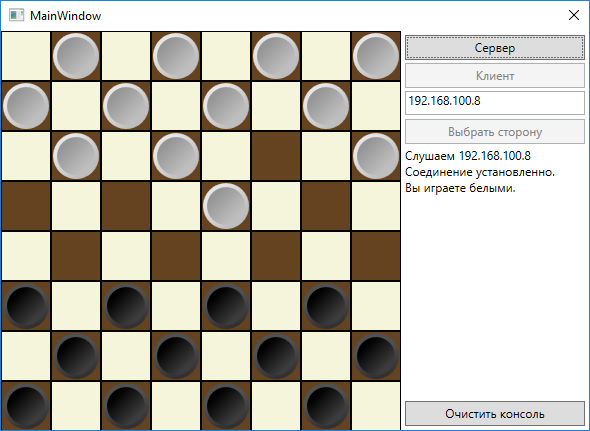


Рисунок 6.5 – Ход шашки

При достижении границ вражеской стороны ваша шашка станет дамкой.

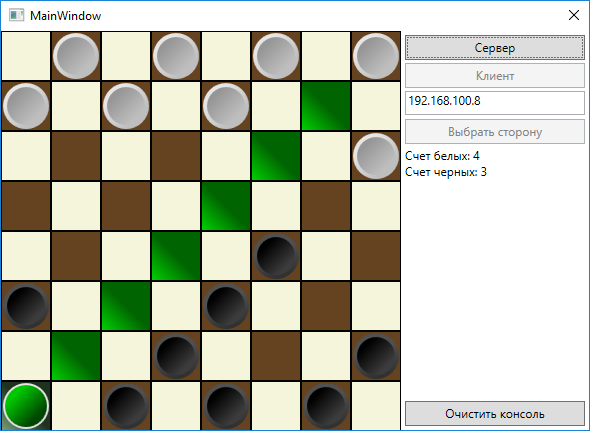


Рисунок 6.6 – Дамка

При победе одной из сторон выведется сообщение с информацией о победителе

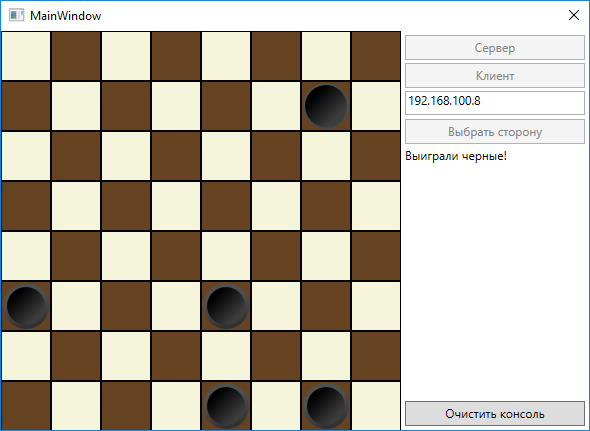


Рисунок 6.7 – Конец игры

Заключение

По итогу работы над курсовым проектом было разработано программное средство, представляющее сетевую игру “Шашки” с клиент-серверной архитектурой. Клиентское и серверное приложение обладают графическим пользовательским интерфейсом.

Разработанная игра имеет минимальный необходимый набор функций для осуществления процесса игры по правилам русских шашек. Среди преимуществ программы можно отметить интуитивно понятный интерфейс и простоту реализации.

Полезным результатом курсового проекта является полученный опыт работы с многопоточным программированием и архитектурой «клиент-сервер». Также был получен опыт работы с технологией WPF для создания графического пользовательского интерфейса, обобщены и применены все ранее полученные знания.

Приложение продемонстрировало корректную работу, обработав исключительные ситуации.

В перспективе данное программное средство может быть усовершенствовано (оптимизация кода, добавление новых функций и режимов, системы рейтинга игроков, ускорение работы алгоритмов).

Подводя итог, можно сказать, что создание данного программного средства является очень эффективным в образовательном плане. Таким образом, цели и задачи, поставленные перед данной работой, успешно достигнуты и выполнены.

Список использованных источников

[1] Бройдо, В.Л. Вычислительные системы, сети и телекоммуникации: учебник для вузов / Бройдо В.Л. – Питер, 2011. – 560 с.

[2] В. Олифер, Н. Олифер, Компьютерные сети. Принципы, технологии, протоколы (4-е издание) – СПб.: Питер, 2010. – 944 с.

[3] Таненбаум Э., Уэзеролл Д. Компьютерные сети. 5-е издание / Э. Таненбаум. – СПб.: Питер, 2012. – 960 с.

[4] WPF [Электронный ресурс]. – Электронные данные. – Режим доступа: https://professorweb.ru/my/WPF/documents-WPF/level36/36-5.php – Дата доступа: 20.04.2021.

[5] Абрамов С. А. Лекции о сложности алгоритмов. – Москва: МЦНМО, 2009. – 256 с.

[6] Клиент-серверная архитектура [Электронный ресурс]. Электронные данные – Режим доступа: <http://www.mstu.edu.ru/study/materials/zelenkov/> – Дата доступа: 19.05.2021.

[7] WPF и С# [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://metanit.com/sharp/wpf> – Дата доступа: 25.03.2021

Приложение А

(обязательное)

Исходный код создания шашек

static class CreateBlack

{

static public List<Ellipse> Create()

{

double width = 46, height = 46;

List<Ellipse> ellipsesB = new List<Ellipse>(12);

for (int i = 0; i < 12; i++)

{

ellipsesB.Add(new Ellipse());

ellipsesB[i].HorizontalAlignment = HorizontalAlignment.Left;

ellipsesB[i].VerticalAlignment = VerticalAlignment.Top;

ellipsesB[i].Width = width;

ellipsesB[i].Height = height;

ellipsesB[i].Margin = new Thickness(-50, -50, 0, 0);

ellipsesB[i].Stroke = new LinearGradientBrush(Color.FromRgb(80, 80, 80), Color.FromRgb(0, 0, 0), new Point(0, 0.5), new Point(0.5, 2));

ellipsesB[i].Fill = new LinearGradientBrush(Color.FromRgb(0, 0, 0), Color.FromRgb(60, 60, 60), new Point(0, 0.5), new Point(0.5, 1));

ellipsesB[i].Name = "c\_B\_" + i;

ellipsesB[i].StrokeThickness = 4;

}

ellipsesB[0].Margin = new Thickness(2, 252, 0, 0);

ellipsesB[1].Margin = new Thickness(102, 252, 0, 0);

ellipsesB[2].Margin = new Thickness(202, 252, 0, 0);

ellipsesB[3].Margin = new Thickness(302, 252, 0, 0);

ellipsesB[4].Margin = new Thickness(52, 302, 0, 0);

ellipsesB[5].Margin = new Thickness(152, 302, 0, 0);

ellipsesB[6].Margin = new Thickness(252, 302, 0, 0);

ellipsesB[7].Margin = new Thickness(352, 302, 0, 0);

ellipsesB[8].Margin = new Thickness(2, 352, 0, 0);

ellipsesB[9].Margin = new Thickness(102, 352, 0, 0);

ellipsesB[10].Margin = new Thickness(202, 352, 0, 0);

ellipsesB[11].Margin = new Thickness(302, 352, 0, 0);

return ellipsesB;

}

}

}

static class CreateWhite

{

static public List<Ellipse> Create()

{

double width = 46, height = 46;

List<Ellipse> ellipsesW = new List<Ellipse>(12);

for (int i = 0; i < 12; i++)

{

ellipsesW.Add(new Ellipse());

ellipsesW[i].HorizontalAlignment = HorizontalAlignment.Left;

ellipsesW[i].VerticalAlignment = VerticalAlignment.Top;

ellipsesW[i].Width = width;

ellipsesW[i].Height = height;

ellipsesW[i].Margin = new Thickness(-50, -50, 0, 0);

ellipsesW[i].Stroke = new LinearGradientBrush(Color.FromRgb(230, 230, 230), Color.FromRgb(190, 190, 190), new Point(0, 0.5), new Point(0.5, 1));

ellipsesW[i].Fill = new LinearGradientBrush(Color.FromRgb(150, 150, 150), Color.FromRgb(190, 190, 190), new Point(0, 0.5), new Point(0.5, 1));

ellipsesW[i].Name = "c\_W\_" + i;

ellipsesW[i].StrokeThickness = 4;

}

ellipsesW[0].Margin = new Thickness(52, 2, 0, 0);

ellipsesW[1].Margin = new Thickness(152, 2, 0, 0);

ellipsesW[2].Margin = new Thickness(252, 2, 0, 0);

ellipsesW[3].Margin = new Thickness(352, 2, 0, 0);

ellipsesW[4].Margin = new Thickness(2, 52, 0, 0);

ellipsesW[5].Margin = new Thickness(102, 52, 0, 0);

ellipsesW[6].Margin = new Thickness(202, 52, 0, 0);

ellipsesW[7].Margin = new Thickness(302, 52, 0, 0);

ellipsesW[8].Margin = new Thickness(52, 102, 0, 0);

ellipsesW[9].Margin = new Thickness(152, 102, 0, 0);

ellipsesW[10].Margin = new Thickness(252, 102, 0, 0);

ellipsesW[11].Margin = new Thickness(352, 102, 0, 0);

return ellipsesW;

}

}

}

Приложение Б

(обязательное)

Исходный код генерации игровой доски

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Shapes;

namespace CheckersWPF.Game

{

class Board

{

Rectangle[] rectangles = new Rectangle[64];

List<Ellipse> ellipsesWhite = new List<Ellipse>(12);

List<Ellipse> ellipsesBlack = new List<Ellipse>(12);

public Rectangle[] CreateBoard()

{

double x = 0, y = 0, step = 0;

double width = 50, height = 50;

bool color = true;

for (int i = 0; i<rectangles.Length;i++)

{

if (step == 8)

{

y += height;

step = 0;

x = 0;

color = !(color);

}

x = width \* step;

step++;

rectangles[i] = new Rectangle();

rectangles[i].Width = width;

rectangles[i].Height = height;

rectangles[i].Margin = new Thickness(x, y, 0, 0);

rectangles[i].Stroke = new SolidColorBrush(Color.FromRgb(0, 0, 0));

rectangles[i].HorizontalAlignment = HorizontalAlignment.Left;

rectangles[i].VerticalAlignment = VerticalAlignment.Top;

if (color)

{

rectangles[i].Fill = new SolidColorBrush(Color.FromRgb(245, 245, 220));

color = !(color);

}

else

{

rectangles[i].Fill = new SolidColorBrush(Color.FromRgb(101, 67, 33));

color = !(color);

}

}

return rectangles;

}

}

}

Приложение В

(обязательное)

Исходный код алгоритмов для игры

public partial class MainWindow : Window

{

public bool white { get; set; } = true;

int CountKillW = 0;

int CountKillB = 0;

int animIter = 0;

int animIter2 = 0;

volatile bool work = true;

Socket handler;

Socket sender;

Rectangle[] rectangles = new Rectangle[64];

Grid grid1 = new Grid();

List<Ellipse> ellipsesW = new List<Ellipse>(12);

List<Ellipse> ellipsesB = new List<Ellipse>(12);

List<Rectangle> rectanglesColor = new List<Rectangle>();

TextBlock label = new TextBlock();

TextBox textBox = new TextBox();

volatile string ip;

Button btnS = new Button();

Button btnC = new Button();

Button ClearConsole = new Button();

Button side = new Button();

bool mustKill = false;

Ellipse select;

Brush selectColor;

bool ServerBool = false;

bool CourseWhite = true;

Ellipse ellipseAnim;

bool \_white = true;

List<string> PointWayStandart = new List<string>();

List<string> PointWay = new List<string>();

List<List<string>> Ways = new List<List<string>>();

List<string> PointOfCourse = new List<string>();

List<string> waySelect = new List<string>();

Thread thread;

string nowPos = "";

public MainWindow()

{

InitializeComponent();

Init();

var abc = 1;

}

public void Init()

{

this.Width = 604;

this.Height = 438;

this.Closed += new EventHandler(CloseWindow);

this.ResizeMode = ResizeMode.NoResize;

Board board = new Board();

rectangles = board.CreateBoard();

for (int i = 0; i < rectangles.Length; i++)

{

rectangles[i].MouseLeftButtonDown += new MouseButtonEventHandler(RectangleClick);

//rectangles[i].MouseEnter += new MouseEventHandler(RectangleMouseEnter);

MainGrid.Children.Add(rectangles[i]);

}

ellipsesW = CreateWhite.Create();

for (int i = 0; i < ellipsesW.Count; i++)

{

ellipsesW[i].MouseLeftButtonDown += new MouseButtonEventHandler(ChechersClick);

MainGrid.Children.Add(ellipsesW[i]);

}

ellipsesB = CreateBlack.Create();

for (int i = 0; i < ellipsesB.Count; i++)

{

ellipsesB[i].MouseLeftButtonDown += new MouseButtonEventHandler(ChechersClick);

MainGrid.Children.Add(ellipsesB[i]);

}

btnS.Width = 180;

btnS.Height = 25;

btnS.Content = "Сервер";

btnS.HorizontalAlignment = HorizontalAlignment.Right;

btnS.VerticalAlignment = VerticalAlignment.Top;

btnS.Margin = new Thickness(0, 4, 4, 0);

btnS.Click += new RoutedEventHandler(btnSClick);

btnC.Width = 180;

btnC.Height = 25;

btnC.Content = "Клиент";

btnC.HorizontalAlignment = HorizontalAlignment.Right;

btnC.VerticalAlignment = VerticalAlignment.Top;

btnC.Margin = new Thickness(0, 32, 4, 0);

btnC.Click += new RoutedEventHandler(btnCClick);

textBox.Width = 180;

textBox.Height = 24;

textBox.HorizontalAlignment = HorizontalAlignment.Right;

textBox.VerticalAlignment = VerticalAlignment.Top;

textBox.Margin = new Thickness(0, 60, 4, 0);

textBox.Text = "";

side.Width = 180;

side.Height = 25;

side.Content = "Выбрать сторону";

side.HorizontalAlignment = HorizontalAlignment.Right;

side.VerticalAlignment = VerticalAlignment.Top;

side.Margin = new Thickness(0, 88, 4, 0);

side.Click += new RoutedEventHandler(sideClick);

label.Width = 180;

label.Height = 600;

label.HorizontalAlignment = HorizontalAlignment.Right;

label.VerticalAlignment = VerticalAlignment.Top;

label.Margin = new Thickness(0, 100, 4, 0);

label.Text = "";

label.TextWrapping = TextWrapping.WrapWithOverflow;

ClearConsole.Width = 180;

ClearConsole.Height = 25;

ClearConsole.Content = "Очистить консоль";

ClearConsole.HorizontalAlignment = HorizontalAlignment.Right;

ClearConsole.VerticalAlignment = VerticalAlignment.Bottom;

ClearConsole.Margin = new Thickness(0, 0, 4, 4);

ClearConsole.Click += new RoutedEventHandler(btnClearClick);

MainGrid.Children.Add(textBox);

MainGrid.Children.Add(label);

MainGrid.Children.Add(btnC);

MainGrid.Children.Add(btnS);

MainGrid.Children.Add(ClearConsole);

MainGrid.Children.Add(side);

}

void sideClick(object s, RoutedEventArgs e)

{

var frm = new SubWindow();

frm.Owner = this;

frm.ShowDialog();

if (frm.DialogResult == false)

{

return;

}

else

{

//ServerBool = true;

\_white = white;

ip = textBox.Text;

if (\_white)

{

CourseWhite = true;

}

else

{

CourseWhite = false;

}

}

}

void btnClearClick(object s, RoutedEventArgs e)

{

label.Text = "";

} // обработчик нажатия по кнопки "отчистить консоль"

void CloseWindow(object s, EventArgs e)

{

work = false;

} // через глобальную переменную work сообщеает потокам что работа закончена

void btnCClick(object s, RoutedEventArgs e)

{

ip = textBox.Text;

try

{

Client("connect");

btnS.IsEnabled = false;

btnC.IsEnabled = false;

}

catch (Exception ex)

{

btnS.IsEnabled = true;

btnC.IsEnabled = true;

}

if (sender == null)

{

btnS.IsEnabled = true;

btnC.IsEnabled = true;

}

} // обработчик нажатия кнопки "клиент"

void btnSClick(object s, RoutedEventArgs e)

{

ServerBool = true;

ip = textBox.Text;

if (\_white)

{

CourseWhite = true;

}

else

{

CourseWhite = false;

}

if (thread != null)

{

}

thread = new Thread(Server);

thread.IsBackground = true;

thread.Start();

} // обработчик нажатия кнопки "сервер"

void ChechersClick(object s, MouseButtonEventArgs e)

{

mustKill = false;

string[] sp = ((Ellipse)s).Name.Split('\_');

if (sp[1] == "W" && CourseWhite && ServerBool && \_white || sp[1] == "B" && CourseWhite && ServerBool && !\_white || sp[1] == "B" && !CourseWhite && !ServerBool && !\_white || sp[1] == "W" && !CourseWhite && !ServerBool && \_white)

{

if (select != null && selectColor != null)

select.Fill = selectColor;

select = (Ellipse)s;

selectColor = select.Fill;

select.Fill = new LinearGradientBrush(Color.FromRgb(0, 200, 0), Color.FromRgb(0, 70, 0), new Point(0, 0.5), new Point(0.5, 1));

bool isKill = ThereIsKill(select);

mustKill = false;

GetPointForCourse(select);

if (isKill != mustKill)

{

Write("Надо бить!");

Ways.Clear();

PointWay.Clear();

waySelect.Clear();

ClearColorRectangles();

if (selectColor != null)

select.Fill = selectColor;

select = null;

return;

}

GenerateColorRectangle();

}

} // обработчик клика по шашке

bool ThereIsKill(Ellipse ellipse)

{

mustKill = false;

if (ellipse.Name.Split('\_')[1] == "W")

{

foreach (var el in ellipsesW)

{

GetPointForCourse(el);

}

}

else if (ellipse.Name.Split('\_')[1] == "B")

{

foreach (var el in ellipsesB)

{

GetPointForCourse(el);

}

}

return mustKill;

}

void RectangleClick(object s, MouseButtonEventArgs e)

{

if (select != null && select.Name.Split('\_')[1] == "W" && CourseWhite && \_white || select != null && select.Name.Split('\_')[1] == "B" && CourseWhite && !\_white||

select != null && select.Name.Split('\_')[1] == "W" && !CourseWhite && \_white|| select != null && select.Name.Split('\_')[1] == "B" && !CourseWhite && !\_white)

{

double xRect = 0, yRect = 0, xPoint = 0, yPoint = 0;

GetPositionRectangle((Rectangle)s, ref xRect, ref yRect);

for (int i = 0; i < PointOfCourse.Count; i++)

{

GetPositionPoint(PointOfCourse[i], ref xPoint, ref yPoint);

if (xRect == xPoint && yRect == yPoint)

{

SetPosition(select.Name + ";" + xRect + "," + yRect);

SendMessage(select.Name + ";" + xRect + "," + yRect);

ClearColorRectangles();

select.Fill = selectColor;

//CourseWhite = !CourseWhite;

if (selectColor != null)

select.Fill = selectColor;

select = null;

}

}

}

} // обработчик клика по квадратику

void GenerateColorRectangle()

{

ClearColorRectangles();

rectanglesColor.Clear();

foreach (var rectangle in rectangles)

{

double x = 0, y = 0;

GetPositionRectangle(rectangle, ref x, ref y);

foreach (var point in PointWay)

{

if (double.Parse(point.Split(',')[0]) == x && double.Parse(point.Split(',')[1]) == y)

{

if (point.Split(',')[2] == "end")

{

rectanglesColor.Add(rectangle);

rectangle.Fill = new LinearGradientBrush(Color.FromRgb(0, 100, 0), Color.FromRgb(0, 210, 0), new Point(0.5, 0.5), new Point(0, 1));

}

else if (point.Split(',')[2] == "start")

{

rectanglesColor.Add(rectangle);

rectangle.Fill = new LinearGradientBrush(Color.FromRgb(30, 50, 30), Color.FromRgb(120, 160, 120), new Point(0.5, 0.5), new Point(0, 1));

}

else if (point.Split(',')[2] == "fork" || point.Split(',')[2] == "way")

{

rectanglesColor.Add(rectangle);

rectangle.Fill = new LinearGradientBrush(Color.FromRgb(120, 0, 0), Color.FromRgb(240, 0, 0), new Point(0.5, 0.5), new Point(0, 1));

}

}

}

foreach (var point in PointWayStandart)

{

if (double.Parse(point.Split(',')[0]) == x && double.Parse(point.Split(',')[1]) == y && PointWay.Count <= 1)

{

rectanglesColor.Add(rectangle);

rectangle.Fill = new LinearGradientBrush(Color.FromRgb(0, 100, 0), Color.FromRgb(0, 210, 0), new Point(0.5, 0.5), new Point(0, 1));

}

}

}

} // подсветка квадратиков

void ClearColorRectangles()

{

for (int i = 0; i < rectanglesColor.Count; i++)

{

rectanglesColor[i].Fill = new SolidColorBrush(Color.FromRgb(101, 67, 33));

}

rectanglesColor.Clear();

} // очищает подсвеченые квадратики

void GetPositionEllipse(Ellipse el, ref double x, ref double y)

{

x = (el.Margin.Left - 2) / 50;

y = (el.Margin.Top - 2) / 50;

} // получает координаты шашки

void GetPositionRectangle(Rectangle re, ref double x, ref double y)

{

x = re.Margin.Left / 50;

y = re.Margin.Top / 50;

} // получает координаты квадратика

void GetPositionPoint(string point, ref double x, ref double y)

{

x = double.Parse(point.Split(',')[0]);

y = double.Parse(point.Split(',')[1]);

} // получает координаты x y в типе double из сторокго представляния координаты

Ellipse GetEllipseCoordinate(double x, double y)

{

double xEl = 0, yEl = 0;

foreach (var el in ellipsesW)

{

GetPositionEllipse(el, ref xEl, ref yEl);

if (x == xEl && y == yEl)

return el;

}

foreach (var el in ellipsesB)

{

GetPositionEllipse(el, ref xEl, ref yEl);

if (x == xEl && y == yEl)

return el;

}

return new Ellipse();

} // возвращает шашку по координатам

void GetPointForCourse(Ellipse ellipse)

{

PointWayStandart.Clear();

PointWay.Clear();

Ways.Clear();

PointOfCourse.Clear();

if (ellipse.StrokeThickness != 2)

{

GeneratePointWayStandart(ellipse, PointWayStandart);

GeneratePointWay(ellipse);

GenerateWays(ellipse);

}

else

{

GeneratePointWayStandartD(ellipse, PointWayStandart);

int[,] desk = new int[8, 8];

double x = 0, y = 0;

GetPositionEllipse(ellipse, ref x, ref y);

List<string> MemoryPoints = new List<string>();

List<int[,]> MemoryMatrix = new List<int[,]>();

MemoryPoints.Add(x + "," + y);

desk = GenerateMatrix();

MemoryMatrix.Add(desk);

Ways.Clear();

desk[(int)y, (int)x] = 0;

if (ellipse.Name.Split('\_')[1] == "W")

generateWays(MemoryMatrix.Last(), (int)x, (int)y, MemoryPoints, MemoryMatrix, 1);

if (ellipse.Name.Split('\_')[1] == "B")

generateWays(MemoryMatrix.Last(), (int)x, (int)y, MemoryPoints, MemoryMatrix, 2);

foreach (var way in Ways)

{

PointWay.Add(way.First() + ",start");

if (way.Count >= 2)

{

PointWay.Add(way.Last() + ",end");

}

}

foreach (var way in Ways)

{

foreach (var point in way)

{

if (!PointWay.Exists((p) => point.Split(',')[0] + "," + point.Split(',')[1] == p.Split(',')[0] + "," + p.Split(',')[1]))

PointWay.Add(point);

}

}

for (int i = 0; i < PointWay.Count; i++)

{

if (PointWay[i].Length == 3)

PointWay[i] += ",no";

}

for (int i = 0; i < Ways.Count; i++)

{

for (int j = 0; j < Ways[i].Count; j++)

{

if (j == 0)

Ways[i][j] += ",start";

else if (j >= 1 && j < Ways[i].Count)

Ways[i][j] += ",no";

else if (j == Ways[i].Count)

Ways[i][j] += ",end";

}

}

for (int i = 0; i < Ways.Count; i++)

{

if (Ways[i].Count == 1)

Ways.RemoveAt(i);

}

}

if (PointWay.Count <= 1)

{

foreach (var point in PointWayStandart)

{

PointOfCourse.Add(point);

}

}

else

{

foreach (var point in PointWay)

{

if (point.Split(',')[2] == "end")

PointOfCourse.Add(point);

}

}

} // получаем точки на которые можно походить/ отсюда запускаются остальные алгоритмы нужные для просчета возможных ходов

public void GeneratePointWayStandartD(Ellipse ellipse, List<string> list)

{

double x = 0, y = 0, x1 = 0, y1 = 0;

GetPositionEllipse(ellipse, ref x, ref y);

for (int i = 1; i < 7; i++)

{

x1 = x - i; y1 = y - i;

if (IsRectangleFree(x1, y1) && IsPositionInLimit(x1, y1))

list.Add(x1 + "," + y1);

else

break;

}

for (int i = 1; i < 7; i++)

{

x1 = x + i; y1 = y - i;

if (IsRectangleFree(x1, y1) && IsPositionInLimit(x1, y1))

list.Add(x1 + "," + y1);

else

break;

}

for (int i = 1; i < 7; i++)

{

x1 = x + i; y1 = y + i;

if (IsRectangleFree(x1, y1) && IsPositionInLimit(x1, y1))

list.Add(x1 + "," + y1);

else

break;

}

for (int i = 1; i < 7; i++)

{

x1 = x - i; y1 = y + i;

if (IsRectangleFree(x1, y1) && IsPositionInLimit(x1, y1))

list.Add(x1 + "," + y1);

else

break;

}

} // генерируем точки на который может походить дамка если некого побить

public void generateWays(int[,] desk, int xCurr, int yCurr, List<string> list, List<int[,]> listMatrix, int color)

{

int xStart = xCurr, yStart = yCurr;

string pointStart = xStart + "," + yStart;

if (desk[yCurr, xCurr] == 0)

{

desk[yCurr, xCurr] = 10;

goOnDirWay(color, pointStart, list, listMatrix, ref desk, xCurr, yCurr, (ref int x, ref int y) => x-- != 0 && y-- != 0);

}

else if (desk[yCurr, xCurr] == 10)

{

desk[yCurr, xCurr]++;

goOnDirWay(color, pointStart, list, listMatrix, ref desk, xCurr, yCurr, (ref int x, ref int y) => x++ != 7 && y-- != 0);

}

else if (desk[yCurr, xCurr] == 11)

{

desk[yCurr, xCurr]++;

goOnDirWay(color, pointStart, list, listMatrix, ref desk, xCurr, yCurr, (ref int x, ref int y) => x++ != 7 && y++ != 7);

}

else if (desk[yCurr, xCurr] == 12)

{

desk[yCurr, xCurr]++;

goOnDirWay(color, pointStart, list, listMatrix, ref desk, xCurr, yCurr, (ref int x, ref int y) => x-- != 0 && y++ != 7);

}

else if (desk[yCurr, xCurr] == 13)

{

if (desk[int.Parse(pointStart.Split(',')[1]), int.Parse(pointStart.Split(',')[0])] == 13)

{

int[,] matrix = new int[listMatrix.Last().GetLength(0), listMatrix.Last().GetLength(1)];

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

if (listMatrix.Last()[i, j] == 10 || listMatrix.Last()[i, j] == 11 || listMatrix.Last()[i, j] == 12 || listMatrix.Last()[i, j] == 13)

matrix[i, j] = 0;

else

matrix[i, j] = listMatrix.Last()[i, j];

}

}

if (IsEnd(pointStart, matrix, xCurr, yCurr, color))

Ways.Add(new List<string>(list));

list.Remove(list.Last());

listMatrix.Remove(listMatrix.Last());

if (list.Count != 0)

generateWays(listMatrix.Last(), int.Parse(list.Last().Split(',')[0]), int.Parse(list.Last().Split(',')[1]), list, listMatrix, color);

}

}

} // алгоритм просчитывающий цепочку возможных ходов (для дамок)

void GetInfoMessage(string message, ref double xRect, ref double yRect, ref Ellipse ellipse)

{

xRect = double.Parse(message.Split(';')[1].Split(',')[0]);

yRect = double.Parse(message.Split(';')[1].Split(',')[1]);

foreach (var el in ellipsesW)

{

if (message.Split(';')[0] == el.Name)

ellipse = el;

}

foreach (var el in ellipsesB)

{

if (message.Split(';')[0] == el.Name)

ellipse = el;

}

} // получаем информацию о ходе (присланную через сеть) / координаты на которые был произведен ход и шашку которой походили

void SendMessage(string msg)

{

if (ServerBool)

{

try

{

if ( handler != null)

{

handler.Send(Encoding.UTF8.GetBytes(msg));

CourseWhite = false;

}

}

catch (Exception ex)

{

Write(ex.Message);

}

}

else

{

try

{

sender.Send(Encoding.UTF8.GetBytes(msg));

CourseWhite = true;

}

catch (Exception ex)

{

Write(ex.Message);

}

}

} // отправляет сообщение противнику

void Client(string message)

{

// Буфер для входящих данных

byte[] bytes = new byte[1024];

IPAddress ipAddr;

//IPAddress ipAddr = IPAddress.Parse("192.168.1.2");

try

{

ipAddr = IPAddress.Parse(ip);

}

catch (Exception ex)

{

Write(ex.Message);

return;

}

Write("Соединение с " + ip);

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 33377);

sender = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

try

{

sender.Connect(ipEndPoint); //подключение

}

catch (SocketException ex)

{

if (ex.ErrorCode == 10060)

{

Write("Невозможно подключиться к данному адресу");

}

else

{

Write(ex.Message);

}

sender.Shutdown(SocketShutdown.Both);

sender.Close();

return;

}

int bytesSent = sender.Send(Encoding.UTF8.GetBytes(message));

// получает сообщение от сервера

int bytesRec = sender.Receive(bytes); // получает сообщение от сервера

string serverMsg = Encoding.UTF8.GetString(bytes, 0, bytesRec); // получает сообщение от сервера

side.IsEnabled = false;

if (serverMsg != "connectW" && serverMsg != "connectB")

{

MessageBox.Show(serverMsg);

}

else if (serverMsg == "connectW")

{

\_white = false;

Write("Соединение установленно.\nВы играете черными.");

Thread clientListen = new Thread(ClientListenConnect);

clientListen.IsBackground = true;

clientListen.Start();

}

else if (serverMsg == "connectB")

{

\_white = true;

CourseWhite = false;

Write("Соединение установленно.\nВы играете белыми.");

Thread clientListen = new Thread(ClientListenConnect);

clientListen.IsBackground = true;

clientListen.Start();

}

// Write(Encoding.UTF8.GetString(bytes, 0, bytesRec));

//sender.Shutdown(SocketShutdown.Both);

//sender.Close();

} // созадем все необходимое для работы клиента/ высылаем ему сообщение о том, что мы присоединились

void ClientListenConnect()

{

while (work)

{

try

{

byte[] buffer = new byte[1024];

int countBytes = sender.Receive(buffer);

string msg = Encoding.UTF8.GetString(buffer, 0, countBytes);

Action act = () =>

{

SetPosition(msg);

//SwitchCourse(msg);

CourseWhite = false;

ServerBool = false;

};

Dispatcher.Invoke(act);

}

catch (Exception ex)

{

CourseWhite = !CourseWhite;

work = false;

Write(ex.Message);

}

}

sender.Shutdown(SocketShutdown.Both);

sender.Close();

} // принимает полученые сообщения от сервера

void Server()

{

IPAddress ipAddr;

try

{

ipAddr = IPAddress.Parse(ip);

//ipAddr = IPAddress.Parse("0.0.0.0");

}

catch (Exception ex)

{

Write(ex.Message);

Action act = () => {

btnC.IsEnabled = true;

btnS.IsEnabled = true; };

Dispatcher.Invoke(act);

return;

}

Write("Слушаем " + ip);

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 33377);

Socket sListener = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

try

{

sListener.Bind(ipEndPoint);

sListener.Listen(10);

Action act1 = () => {

btnC.IsEnabled = false;

};

Dispatcher.Invoke(act1);

handler = sListener.Accept();

ServerBool = true;

\_white = white;

if (\_white)

{

CourseWhite = true;

}

else

{

CourseWhite = false;

}

while (work)

{

try

{

byte[] bytes = new byte[1024];

int bytesRec = handler.Receive(bytes); //замирает, принимает сообщение

string data = Encoding.UTF8.GetString(bytes, 0, bytesRec);

if (data == "connect")

{

if (\_white)

{

Action act = () => { side.IsEnabled = false; Write("Соединение установленно.\nВы играете белыми."); };

Dispatcher.Invoke(act);

}

else

{

Action act = () => { side.IsEnabled = false; Write("Соединение установленно.\nВы играете черными."); };

Dispatcher.Invoke(act);

}

if (\_white)

{

handler.Send(Encoding.UTF8.GetBytes("connectW"));

}

else

handler.Send(Encoding.UTF8.GetBytes("connectB"));

}

else

{

Action act = () => {

CourseWhite = true;

SetPosition(data);

//SwitchCourse(data);

};

Dispatcher.Invoke(act);

}

}

catch (Exception ex)

{

CourseWhite = !CourseWhite;

work = false;

Write(ex.Message);

}

}

handler.Shutdown(SocketShutdown.Both);

handler.Close();

}

catch (Exception ex)

{

Action act = () => { btnC.IsEnabled = true; };

ServerBool = false;

CourseWhite = true;

Dispatcher.Invoke(act);

Write(ex.Message);

}

} // создаем все необходимое для работы сервера/ тут же принимает полученные сообщения от клиента