Оглавление

[Введение 2](#_Toc506423508)

[Аналитическая часть 3](#_Toc506423509)

[Конструкторская часть 5](#_Toc506423510)

[Технологическая часть 8](#_Toc506423511)

[Основные инструменты используемые для реализации проекта 8](#_Toc506423512)

[Реализация 11](#_Toc506423513)

[Результат 15](#_Toc506423514)

[Список Литературы 16](#_Toc506423515)

# Введение

Компьютерные игры достаточно давно появились на свет. Первые интерактивные электронные устройства, предназначенные специально для игр, и первые игровые программы для компьютеров были разработаны в США после Второй мировой войны. В 1948 году американские физики Томас Голдсмит-младший и Эстл Рей Манн запатентовали «развлекательное устройство на основе электронно-лучевой трубки», представлявшее собой электронный тир; более совершенным устройством был Tennis for Two (1958) — симулятор тенниса, созданный Уильямом Хигинботамом, сотрудником Брукхейвенской национальной лаборатории на базе осциллографа.

Игровые программы создавались для первых компьютеров, изначально предназначенных для других целей: так, программа OXO (1952), имитирующая игру «крестики-нолики», была составлена для компьютера EDSAC британским ученым Александером Дугласом как часть его докторской диссертации в Кембриджском Университете; претендующая на звание первой компьютерной игры программа Spacewar! (1962) была написана Стивом Расселом и двумя другими студентами Массачусетского технологического института для мини-компьютера PDP-1.

В современном мире же, уже существует огромное количество компьютерных игр самых разных жанров, и настольные игры не исключение. Если раньше для того чтобы сыграть в любимою игру надо было потратить много времени на подготовку к игре (не говоря уже о том, что все игроки должны находиться в одном месте), то сейчас это секундное дело.

# Аналитическая часть

Основной целью проекта является разработка клиент-серверного приложения для многопользовательских настольных игр, с возможностью развития.

В связи с тем, что все мы находимся на больших расстояниях и часто не хватает времени на любимое хобби, подобная платформа заинтересует тех, кто увлекается настольными играми. Её основное преимущество перед обычными настольными играми заключается в том, что на каком бы расстоянии игроки не находились, они могут сыграть друг с другом. Также плюсом, является мгновенная подготовка – не нужно расставлять фишки и фигурки перед каждой игрой, компьютер всё сделает за вас. Такое приложение намного упрощает жизни тех, кто часто играет в настольные игры.

Из уже существующих приложений для игры в настольные игры можно выделить Tabletop Simulator - симулятор популярных классических настольных занятий. Речь идёт про развлечения вроде шашек, нард, шахмат, а также широкий спектр карточных игр. Это симулятор, который позиционируется как проект, в котором можно высвободить свою агрессию, просто щёлкая по виртуальному столу. При этом, разработчиками заявляется возможность придумывать свои собственные новые развлечения и полное отсутствие каких-то предустановленных правил сервиса при этом. В такого рода занятии необходимо будет руководствоваться только физическими законами данной песочницы и замыслами самого игрока, а также его друзей. Минусом данного приложения является то, что кроме покупки самой игры, для того чтобы поиграть в какую-то игру, созданную разработчиками и не входящую в основной набор, придётся докупать дополнения к ней. Конечно, есть возможность самому создать настольную игру, но это достаточно трудоёмкая работа.

Из отечественных платформ данного жанра, можно отметить игры Вконтакте, например монополия или мафия. Но в ней полно навязчивой рекламы и влияющей на геймплей системы доната.

В отличии от приведённых примеров, моё приложение будет составлено из разных логических настольных игр, с несложными правилами на подобие шахмат. Данный жанр был выбран в связи с отсутствием элемента случайности в игре, то есть исход игры зависит только от умений игроков.

Первыми настольными играми были выбраны Коридор и Кварто, потому что еще не реализована их онлайн-версия. Сответственно я начал изучить их правила. Так как я выбрал для реализации проекта Qt Creator как среду разработки, с которой я работал впервые, также надо было научиться работать с ней. Еще парой новых открытий для меня были Git и Doxygen, которые очень сильно пригодились мне для проекта.

Для реализации программных алгоритмов (логики игр, клиент-серверная часть, графическая оболочка), мне пришлось изучить много литературы, в большинстве своём это были описания библиотек и классов Qt, но также для высокоуровневого, функционального ООП (объектно-ориентированного программирования) необходимо было изучать сам язык C++ подробнее. В этом мне помогла книга Бьёрна Страуструпа - Язык программирования C++ (специальное издание).

Так как данная платформа экономит много времени, даёт возможность играть на расстоянии и предлагает те настольные игры которые еще не были реализованы, то можно сказать, что цель проекта актуальна.

# Конструкторская часть

Для реализации поставленной задачи проект должен иметь клиент-серверную архитектуру, т.к. данная структура позволяет как иметь один отдельный сервер для всех пользователей, так и создать локальный сервер на одном из компьютеров клиента. В итоге он должен представлять из себя приложение (платформу), с помощью которого, его пользователи смогут играть в настольные игры по сети с другими людьми. Подключившись к серверу игрок может выбрать игру из списка доступных, инициировать новый сеанс игры, или же присоединиться к существующему сеансу.

Для начала работы необходимо было разделить реализацию проекта на части и спланировать их реализацию:

Создание алгоритмов логики настольных игр: Для разработки игры Коридор, нам точно понадобиться алгоритм поиска на графе состояний. Он будет необходим для того, чтобы определить существует ли путь от игрока до цели, потому что по правилам игры нельзя перекрыть его для противника полностью. Для игры в Кварто необходимы будут бинарные операции с четырёхзначными, двоичными числами. У каждой фигурки есть свой номер, который означает её свойства, то-есть при переводе данного номера в двоичную систему каждый бит отвечает за одно из четырёх свойств (размер, цвет, форма, наличие выемки). Также примерно можно представить концепцию игры: ходы игроков, сами игроки (их данные), поле игры и прочие детали алгоритма.

Создание графического интерфейса, совместно работающего с логикой игры: Для этого необходимо было опираться на реальную версию игры, чтобы смысл изображения совпадал со смыслом настоящих фигурок и поля. Кроме того необходимо связать это с алгоритмом логики игры.

Создание клиента, включающего в себя уже созданные приложения настольных игр и отправляющий команды на сервер: Так как клиент должен включать в себя не только настольные игры , но и пользовательский интерфейс для работы с сеансами игр или общения с сервером, то это значит что необходимо не просто создать независимое графическое приложение, но и реализовать способ отправки данных от клиента на сервер. В данном случае удобно будет использовать типизированные команды-объекты, которые будут нести нужную для исполнения информацию.

Создание Сервера, который будет обрабатывать полученные команды, выполнять их, а также отсылать их в ответ (клиентам): Здесь всё обстоит немного сложней, потому что он должен работать со всеми клиентами одновременно: обрабатывать полученные команды от них и отвечать на них какими-то действиями. То-есть он должен распознавать команды, считывать из них информацию и в свою очередь отправлять подобные команды клиенту, а это значит что клиенту необходимо добавить аналогичный алгоритм получения команды.

Благодаря такому делению разработки, было достаточно легко работать, т.к. с многим я столкнулся впервые, а данная схема позволила распределить изучение нового материала по мере создания платформы.

Кроме того, нельзя начать делать работу с конца, нужно начинать с фундамента. То есть в моём случае нужно начинать с той части которая работает независимо от других, как логика игры, она никак не измениться в ходе разработке, даже если изменить концепцию. А вот например клиент нельзя сделать, пока не определишься какие функции он должен выполнять. Например в данном случае это связанно с тем, что он использует логику игры с графической оболочкой, и пока они не реализованы, клиент не сможет работать так как задумывалось.

Получается вот такая схема включения одних частей в другие:

Логика игры ← пользовательский интерфейс игры ← клиент

При этом нету смысла существования сервера без клиента, так как пользователь будет использовать только клиентскую часть, а сервер нужен лишь только для обмена информацией между клиентами и для хранения информации о них.

# Технологическая часть

## Основные инструменты используемые для реализации проекта

1. Язык программирования C++

Данный язык был выбран, в первую очередь потому, что я его знаю лучше остальных. Но выбрал для изучения я его не случайно, а за его плюсы:

1. Высокая производительность: Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы. Один из базовых принципов C++ «не платишь за то, что не используешь» то есть ни одна из языковых возможностей, приводящая к дополнительным накладным расходам, не является обязательной для использования. Имеется возможность работы с памятью на низком уровне.
2. Кроссплатформенность: стандарт языка C++ накладывает минимальные требования на ЭВМ для запуска скомпилированных программ.
3. Доступность: Для C++ существует огромное количество учебной литературы, переведённой на всевозможные языки. Язык имеет высокий порог вхождения, но среди всех языков такого рода обладает наиболее широкими возможностями.
4. Поддержка различных стилей программирования: традиционное императивное программирование (структурное, объектно-ориентированное), обобщённое программирование, функциональное программирование, порождающее метапрограммирование.

Это не все достоинства C++, но основные, в связи с которыми я выбрал именно этот язык.

1. Кроссплатформенный фреймворк Qt

Для реализации моего проекта, необходима была библиотека, упрощающая работу с графикой и сетями. На примете были Qt и MFC. Чтобы сделать выбор, мне пришлось их сравнить.

Qt – кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++. Есть также «привязки» ко многим другим языкам программирования: Python — PyQt, PySide; Ruby — QtRuby; Java — Qt Jambi; PHP — PHP-Qt и другие.

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода (кроссплатформенность). Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Комплектуется визуальной средой разработки графического интерфейса Qt Designer, позволяющей создавать диалоги и формы.

Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Мета-объектная система — часть ядра фреймворка для поддержки в С++ таких возможностей, как сигналы и слоты для коммуникации между объектами в режиме реального времени и динамических свойств системы

Одним из преимуществ проекта Qt является наличие качественной документации, в отличие, например, от wxWidgets. Статьи документации снабжены большим количеством примеров. Исходный код самой библиотеки хорошо форматирован, подробно комментирован, что также упрощает изучение Qt.

Отличительная особенность — использование мета-объектного компилятора — предварительной системы обработки исходного кода. Расширение возможностей обеспечивается системой плагинов, которые возможно размещать непосредственно в панели визуального редактора. Но минусом получается то, что код написанный с помощью Qt нельзя скомпилировать на другом компьютере без установки фреймворка.

Microsoft Foundation Classes – библиотека на языке C++, разработанная Microsoft и призванная облегчить разработку GUI-приложений для Microsoft Windows путём использования богатого набора библиотечных классов.

Во-первых если сравнивать только работу с GUI, то данная библиотека работает только под Windows, то есть ни о какой кроссплатформенности речи не идёт. Но не стоит забывать о том что Qt в отличии от MFC имеет множество других полезных классов, и что важно в данном случае, упрощает работу с сетью. Во-вторых, если же в MFC создать каркас приложения без дизайнера достаточно сложно, то в Qt это зачастую даже намного удобнее и проще.

Поскольку функционал Qt намного шире, то для реализации проекта был выбран именно фреймворк Qt.

1. Среда разработки Qt creator

Qt Creator (ранее известная под кодовым названием Greenhouse) — кроссплатформенная свободная IDE для языков С, С++ и QML. Разработана Trolltech (Digia) для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

Основная задача Qt Creator — упростить разработку приложения с помощью фреймворка Qt на разных платформах. Поэтому для работы с данной библиотекой был выбран именно он.

Дополнительные инструменты:

Для хранения исходников используется система Git (на портале github.com), т.к. это крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

Для построения документации системы используется Doxygen (с помощью неё в автоматическом режиме получены диаграммы представленные в презентации). Очень полезная программа которая сильно помогает в отображении того как работает алгоритм.

## Реализация

Реализация приложения началась с создания консольной версии игры Коридор. Для этого необходимо было разработать оптимальный алгоритм удовлетворяющий правилам игры. В данной игре ходы происходят последовательно, поэтому ход игрока состоит из ввода символов по которым программа определяла что измениться на поле. Например в начале каждого хода у игрока спрашивается что он хочет сделать: поставить стену или переместить свою фишку, на что тот должен был ответить либо 'm' (**move**) или 'w' (**wall**) и так далее… Всё это войдёт в функцию хода (*turn*), которая для каждого ответа вызывает соответствующую функцию: для move это функция передвижения, а для wall это функция расположения стены, в каждой из которых свои запрашиваемые данные и свой результат воздействия на поле игры. Но каждый ответ должен проверяться на возможность исполнения, за что будет отвечать множество функций-проверок (например *search*). В конце каждого хода, соответствующая функция возвращает значение окончания игры, и в зависимости от него либо выводится поле игры и начинается следующий ход, либо пишется сообщении о победе того или иного игрока и игра заканчивается.

Когда же начались работы с графикой, я, используя уже существующий алгоритм логики игры, реализовал офлайновый вариант игры на одном компьютере, уже с визуализацией. Для этого надо было изменить концепцию того как общается игрок с игрой: вместо ввода символов, теперь он нажимал кнопки и взаимодействовал по средствам действий мышки. То-есть алгоритм должен был распознать движется игрок или ставит стену, следуя из информации о том где игрок нажал на мышку, куда передвинул и в каком месте отпустил. Поэтому к полю игры было решено добавить два вида стен: вертикальная и горизонтальная. Игрок перетаскивает одну из них на поле, и уже знаю какая стена выбрана, несложно посчитать координаты где стена была отпущена. Если же игрок просто отпускает кнопку мыши на какой-то клетке поля и при это ни одна из стен не была выбрана, то это значит, что он просто хочет переместиться в данную точку, после чего, собственно говоря, шла проверка на возможность данного действия, как и в случае со стеной.

Далее, по уже существующему шаблону, была реализована игра Кварто для одного компьютера.

Разработка носила циклический характер: написание нового кода, его отладка и последующий рефакторинг, в ходе которого код обычно сильно упрощался благодаря использованию ООП и известных паттернов проектирования. Например функция *jump* в консольной версии игры Коридор изначально состояла из четырёх функций, для каждого направления соответственно. Оказалось что все эти функции очень схожи и по сути это был повторный код, поэтому их можно соединить в одну, что намного упростило код.

Параллельно с реализацией игры Коридор создавалась серверная часть приложения. Для реализации клиент-серверного общения был выбран стандартный протокол передачи данных TCP и передача сообщений через сокеты. Сначала клиент отправлял серверу строковые сообщения, но в дальнейшем были использованы типизированные команды. Для каждой команды создавался новый класс, наследующий класс-родитель *Command*, и у каждого класса есть собственные переменные для передачи, а также собственная реализация операторов записи и вывода в/из поток(а) и метод *execute()*, который проделывал какие-то операции с полученными данными или просто оповещал об исполнении. Объект нужного класса создавался либо на клиенте, либо на сервере и отправлялся вместе с типом на противоположную сторону (сервер/клиент). Там создавалась фабрика команд, которая в первую очередь считывала тип команды и создавала пустую команду данного типа. Далее происходило чтение данных из потока в команду. И наконец в зависимости от типа, выполнялись те или иные операции, использующие переменные команды. Таким образом, сервер наполнился большим количеством команд.

Когда же дело дошло до добавления игр в клиент, то необходимо было снова изменить концепцию алгоритма игры. Теперь при нажатии на кнопку нужно было не просто исполнить ход, но еще и отправить данные об этом противнику, при этом также клиент должен уметь считывать данную информацию. По мере развития клиент вобрал в себя большую часть приложения игры и начал создавать команды, несущие данные о действиях игрока и событиях игры, которые отправлял на сервер.

В итоге игра Коридор полностью заработала в онлайн режиме. Далее, аналогичным способом была добавлена игра Кварто. То-есть, уже можно было полноценно играть по локальной сети подключившись к одному серверу в обе игры.

# Результат

В результате проведённой работы можно сделать выводы:

1. Платформа для игры в настольные игры, которые еще не реализованы востребована.
2. Мы выяснили какой подход лучше всего использовать для реализации поставленной задачи, разбив при этом разработку на отдельные части.
3. В конце концов была реализована платформа для многопользовательских настольных игры.

Таким образом, можно однозначно утверждать что, поставленная цель была достигнута.

# Список Литературы

1. Qt документация [Электронный ресурс]: http://doc.qt.io/
2. Бьёрн Страуструп Язык программирования C++ - специальное издание [Текст]
3. Правила игры Коридор [Электронный ресурс]: https://www.igroved.ru/rules/281.pdf
4. Правила игры Кварто [Электронный ресурс]: https://www.igroved.ru/rules/279.pdf
5. Паттерны проектирования [Текст]: Приёмы объектно-ориентированного проектирования / Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес