Министерство образования Республики Беларусь

Учреждения образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ

Зав. каф. ЭВМ

\_\_\_\_\_\_\_\_В.В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проекту на тему

ГРАФИЧЕСКАЯ ЧАСТЬ ИГРОВОГО ПРИЛОЖЕНИЯ "ХОД"

БГУИР ДП 1-40 02 01 01 030 ПЗ

Студент Д.С. Кузьмин

Руководитель А.А. Калютчик

Минск 2024

Министерство образования Республики Беларусь

Учреждения образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФКСиС,Кафедра;ЭВМ.

Специальность: ”Вычислительные машины,системы и сети”.

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

————————В.В. Никульшин

“——————”————————2024г. Задание

По курсовому проекту студента

Кузьмина Дмитрия Сергеевича

1 Тема проекта: ”Графическая часть игрового приложения «Ход»” – утверждена приказом по университету от 1 февраля 2024 г. №123-с

2 Срок сдачи студентом законченного проекта 1 июня 2024 г.

3 Исходные данные к проекту:

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

1. Введение.

1. Обзор литературы.
   1. Обзор методов и алгоритмов решения поставленной задачи.
2. Функциональное проектирование.
   1. Структура входных и выходных данных.
   2. Разработка диаграммы классов.
   3. Описание классов.
3. Разработка программных модулей.
   1. Разработка схем алгоритмов (два наиболее важных метода).
   2. Разработка алгоритмов (описание алгоритмов по шагам для двух

методов).

1. Результаты работы.
2. Заключение
3. Литература
4. Приложение

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

1. Диаграмма классов.

1. Схема алгоритма.
2. Схема алгоритма .
3. Консультант по проекту (с обозначением разделов проекта) А.А.

Калютчик

1. Дата выдачи задания февраль 2024 г.
2. Календарный график работы над проектом на весь период проектирования

(с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки.

Перечень графического материала – 15 %; разделы 2, 3 – 10 %; разделы 4 к –20 %; разделы 5 к – 35 %; раздел 6,7,8 – 5 %; раздел 9 к – 5%;

оформление пояснительной записки и графического материала к

15.12.23

10 %

Защита курсового проекта с по г.

РУКОВОДИТЕЛЬ

А.А. Калютчик

Задание принял к исполнению

Д.С. Кузьмин

(дата и подпись студента)

## Содержание

ВВЕДЕНИЕ...................................................................................................5

1. ОБЗОР ЛИТЕРАТУРЫ............................................................................6

1.1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ

1.2 ПОСТАНОВКА ЗАДАЧИ

2. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ....................................................18

2.1 МОДУЛЬ ИГРОВОЙ МЕХАНИКИ

2.2 МОДУЛЬ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА(UI)

2.3 МОДУЛЬ СЕТЕВОЙ ИГРЫ

2.4 МОДУЛЬ ХРАНЕНИЯ ДАННЫХ

2.5МОДУЛЬ ТЕСТИРОВАНИЯ

2.6 МОДУЛЬ УПРАВЛЕНИЯ ИГРОВЫМИ СОСТОЯНИЯМИ

3. ФУНКЦИОНАЛЬНЕ ПРОЕКТИРОВАНИЕ.........................................21

4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ....................................37

5. РЕЗУЛЬТАТ РАБОТЫ........................................................................ ...40

# ВВЕДЕНИЕ

В современном мире компьютерные игры занимают все более значимое место в развлекательной и развивающей сферах. Благодаря технологическому прогрессу и доступности компьютеров и мобильных устройств, разработка игровых приложений стала одной из наиболее динамично развивающихся отраслей информационных технологий. Игровая индустрия предлагает широкий спектр разнообразных жанров и концепций, увлекая миллионы игроков по всему миру.

Целью данной курсовой работы является разработка игрового приложения под названием "Ход". "Ход" - это стратегическая компьютерная игра, в которой игроки соревнуются в логическом мышлении и тактическом планировании. Она предлагает разнообразные игровые ситуации, в которых игроки должны анализировать текущее положение, принимать решения и предугадывать ходы соперников.

Разработка игровых приложений требует комплексного подхода, включающего в себя проектирование игрового интерфейса, программирование игровой логики, создание графических и звуковых эффектов, а также тестирование и оптимизацию приложения. Она также требует глубокого понимания потребностей и предпочтений игроков, чтобы создать интересный, захватывающий и увлекательный геймплей.

Данная часть работы посвящена исследованию и разработке графической части, которая может быть полезна в контексте игрового приложения "Ход". Успешная разработка игрового приложения требует учета разнообразных факторов, включая создание дополнительных инструментов и функциональностей, способных улучшить пользовательский опыт и обеспечить более гибкое взаимодействие с приложением. Эта работа направлена на повышение функциональности и привлекательности игрового приложения "Ход" путем добавления дополнительных возможностей и инструментов, которые могут улучшить взаимодействие пользователей с приложением.

Задачи на курсовой проект

* Познакомится с операционной системой Linux;
* Изучить работу файловой системой Linux;
* Изучить взаимодействие устройств вывода с Linux;
* Оточить навыки написания программ на языке С/С++;
* Написать утилиту для защиты персональных данных.

# ОБЗОР ЛИТЕРАТУРЫ

**1.1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ**

# ПОСТАВЛЕННОЙ ЗАДАЧИ

Игровое приложение «Ход» было вдохновлено одной из самых старых игр в мире – шахматы.

Шахматы, одна из самых старых и увлекательных игр, привлекают внимание людей по всему миру своей глубиной, интеллектуальными вызовами и культурным наследием. Эта игра не только развлекает, но и развивает у игроков навыки анализа, стратегического мышления и психологической устойчивости. В данном эссе мы рассмотрим более подробно историю шахмат, их влияние на развитие мышления и общества, а также более глубоко погрузимся в стратегические и психологические аспекты этой удивительной игры.

История шахмат уходит в далекое прошлое, в древнюю Индию. Именно здесь, примерно в VI веке н.э., появились прародители современных шахмат. Игра называлась "чатуранга" и имела схожие, но несколько отличающиеся правила. Затем шахматы перебрались в Персию, где стали известны как "шахматы" (шах - король, мат - убийство), и приобрели многие из основных правил, которые мы знаем сегодня.

С течением времени шахматы распространились по всей Европе и Азии, а затем и по всему миру. В средние века шахматы стали частью европейской культуры и обрели широкое распространение среди аристократии. Они стали не только развлечением, но и средством обучения стратегии и тактике.

С развитием технологий и доступности информации, шахматы стали ещё более популярными. Они стали предметом изучения исследователей в области искусственного интеллекта, психологии и образования.

Шахматы известны своим способом развивать различные аспекты умственной деятельности. Это игра, которая требует от игроков анализа ситуации, планирования и принятия стратегических решений. Шахматы способствуют развитию логического мышления, внимания, памяти, умения принимать решения в условиях ограниченной информации и умения адаптироваться к изменяющимся обстоятельствам.

Многочисленные исследования подтверждают, что занятия шахматами могут улучшить когнитивные способности у людей всех возрастов. Особенно заметен этот эффект у детей, участвующих в шахматных тренировках, они чаще демонстрируют повышенные показатели в учебе и развитии мыслительных навыков.

Стратегия в шахматах играет ключевую роль в достижении успеха. Она включает в себя широкий спектр тактических и стратегических приемов, которые помогают игроку добиться преимущества над противником. Контроль центра, развитие фигур, обеспечение безопасности короля и создание атакующих возможностей - это лишь некоторые из базовых стратегических принципов, которые помогают игроку в достижении победы.

Игра в шахматы требует от игрока глубокого анализа, понимания позиции на доске и умения выявлять слабые места противника. Успешный шахматист должен уметь формировать долгосрочные планы и адаптироваться к изменениям в игровой ситуации.

Психологические аспекты игры в шахматы играют ключевую роль в определении исхода партии. Успешный шахматист должен обладать хорошим контролем над своими эмоциями, способностью оставаться спокойным и сосредоточенным в сложных ситуациях.

Умение управлять временем - один из важных аспектов психологии игры в шахматы. Каждый игрок имеет ограниченное количество времени на партию, поэтому важно эффективно использовать его и не допускать излишней траты времени на раздумья.

Шахматы - это не просто игра, это искусство, наука и спорт. Они имеют древние корни и оказывают значительное влияние на развитие человеческого мышления. Игра в шахматы требует от игрока умения анализировать сложные ситуации, принимать решения на основе обдуманного анализа и управлять своими эмоциями. Шахматы продолжают оставаться одной из самых увлекательных и популярных игр в мире, привлекая внимание миллионов людей всех возрастов и профессий.

Однако прогресс не стоит на месте, технологии развиваются и вместе со всем этим появились сетевые шахматы.

С развитием информационных технологий и интернета в нашей жизни появилась возможность играть в шахматы не только в реальном мире, но и в виртуальном пространстве. Сетевые шахматы открыли новые горизонты для шахматистов, предоставив им возможность сразиться с соперниками из любой точки мира в любое время суток. В данном эссе мы рассмотрим историю развития сетевых шахмат, их влияние на традиционную игру, технологические инновации, психологию игроков, их влияние на общество, перспективы дальнейшего развития этой формы шахматной активности, а также некоторые примеры популярных сетевых шахматных платформ.

Идея сетевых шахмат впервые появилась с появлением компьютеров и компьютерных сетей. В начале 1970-х годов были созданы первые программы для игры в шахматы на компьютерах, которые позволяли игрокам сражаться друг с другом по сети. Однако, в связи с ограниченными возможностями техники того времени, эти программы были далеки от современных стандартов.

С развитием интернета в конце XX века и появлением более мощных компьютеров стали появляться современные платформы для игры в сетевые шахматы. Первые онлайн-шахматные серверы появились в конце 1990-х годов и с тех пор продолжают развиваться и привлекать новых игроков.

С появлением сетевых шахмат традиционная игра в шахматы также изменилась. Игроки теперь могут тренироваться и соревноваться с оппонентами из любой точки мира, не выходя из дома. Это расширяет возможности для игроков, позволяя им подбирать соперников соответствующего уровня и проводить больше партий за меньшее время.

Однако, сетевые шахматы также представляют вызов для традиционной игры. Возможность использования компьютерных программ и баз данных о партиях позволяет игрокам легче подготавливаться к партиям и находить оптимальные ходы. Это может привести к снижению значимости человеческого фактора в игре и изменению традиционных стратегий и тактик.

С появлением сетевых шахмат также возникла возможность использования различных технологических инноваций, которые значительно обогащают игровой процесс. Например, современные онлайн-платформы предоставляют игрокам возможность просмотра и анализа прошлых партий, общения с другими шахматистами в чате, участия в турнирах и обучающих курсах. Также существуют специализированные программы для анализа партий и обучения, которые помогают игрокам улучшить свои навыки.

Психологические аспекты игры в сетевые шахматы также играют важную роль. Игроки должны уметь управлять своими эмоциями, оставаться спокойными в сложных ситуациях и принимать решения на основе расчета, а не эмоциональных реакций.

Один из ключевых аспектов психологии в сетевых шахматах - это управление временем. В большинстве онлайн-игр у игроков есть ограниченное количество времени на ход, поэтому важно эффективно использовать его и не тратить слишком много времени на раздумья.

Кроме того, игра в сетевые шахматы требует от игрока способности адаптироваться к стилю игры противника и предсказывать его ходы. Это требует высокой степени внимания, концентрации и гибкости мышления.

Сетевые шахматы оказывают также влияние на общество в целом. Они стимулируют развитие информационных технологий и компьютерных наук, поскольку создание и поддержка онлайн-платформ требует передовых технологических решений. Кроме того, они способствуют международному общению и культурному обмену, поскольку игроки из разных стран могут взаимодействовать и обмениваться опытом.

Примеры популярных сетевых шахматных платформ:

* Chess.com: Одна из самых известных и популярных платформ для игры в шахматы. Chess.com предлагает широкий выбор игровых режимов, обучающие материалы, аналитику партий и возможность соревноваться в турнирах.
* Lichess.org: Это бесплатная и открытая платформа для игры в шахматы. Lichess.org предлагает высококачественный интерфейс, аналитику партий, турниры и обучающие материалы.
* Chess24.com: Еще одна популярная платформа с различными игровыми режимами, обучающими материалами и трансляциями профессиональных турниров.

С развитием технологий и интернета можно ожидать дальнейшего роста популярности сетевых шахмат. Улучшение качества графики и интерфейсов, развитие искусственного интеллекта для анализа партий и обучения игроков, а также внедрение виртуальной реальности в игровой процесс - все это может сделать игру в сетевые шахматы еще более захватывающей и доступной для широкой аудитории.

Сетевые шахматы представляют собой уникальную возможность для шахматистов со всего мира встретиться и сразиться в интеллектуальном поединке. Они расширяют возможности для тренировок, соревнований и общения с другими игроками, но также представляют новые вызовы для традиционной игры и психологии игроков. Игра в сетевые шахматы требует от игрока не только хороших технических навыков, но и умения управлять своими эмоциями и адаптироваться к изменяющимся обстоятельствам. С развитием технологий и интернета можно ожидать дальнейшего роста популярности этой формы шахматной активности и ее влияния на общество.

Рассмотрим подробнее примеры самых популярных шахматных платформ:

1. Chess.com:

Chess.com является одной из самых популярных и широко используемых онлайн-платформ для игры в шахматы. На рисунке 1.1.1 представлено меню сайта:



Рисунок 1.1.1 – Главное меню сайта Chess.com

Рассмотрим преимущества и недостатки.

Преимущества:

* + - Обширное сообщество: Chess.com имеет одно из крупнейших сообществ шахматистов в мире. Это означает, что всегда есть много соперников разного уровня для игры, а также много возможностей для общения, обмена опытом и участия в турнирах.
    - Разнообразие игровых режимов: Платформа предлагает широкий выбор игровых режимов, включая классические партии, быстрые и очень быстрые игры (блиц и бюллетени), а также партии с различными временными добавками. Это позволяет игрокам выбирать формат, который соответствует их предпочтениям и доступному времени.
    - Обучающие материалы и тренировки: Chess.com предлагает обширный набор обучающих материалов для игроков всех уровней. Это включает в себя видеоуроки, интерактивные уроки, практические задания и возможность тренироваться с профессиональными тренерами.
    - Организация турниров и соревнований: Платформа регулярно организует турниры и соревнования на различных уровнях. Это позволяет игрокам проверить свои силы в реальных условиях и получить новый опыт.

Недостатки:

* + - Платные возможности: Некоторые продвинутые функции и материалы доступны только с подпиской на платную версию. Это может быть недоступно для некоторых пользователей, которые не могут или не хотят платить за дополнительные функции.
    - Нежелательная реклама: Бесплатная версия Chess.com может содержать рекламу, которая может отвлекать и раздражать пользователей во время игры. Некоторые игроки могут считать это недостатком и предпочтут альтернативные платформы с меньшим количеством рекламы.

1. Lichess.org:

Lichess.org отличается простым и минималистичным интерфейсом, а также бесплатностью и открытостью. На рисунке 1.1.2 представлено меню сайта:



Рисунок 1.1.2 – главное меню сайта Lichess.org

Преимущества:

* + - Полная бесплатность и отсутствие рекламы: Lichess.org предоставляет все свои функции бесплатно и без раздражающей рекламы. Это делает его привлекательным для многих пользователей, особенно тех, кто предпочитает не платить за игровой опыт.
    - Простой интерфейс: Платформа имеет простой и интуитивно понятный интерфейс, что делает игру в шахматы доступной и удобной для всех. На Lichess.org легко найти нужные функции и начать игру в несколько кликов.
    - Обширные обучающие материалы: Lichess.org предоставляет обширный выбор обучающих материалов и инструментов для анализа партий, что помогает игрокам улучшать свои навыки. Это включает в себя уроки, видеоуроки, практические задания и возможность обучения с использованием компьютерного анализа.

Недостатки:

* + - Ограниченные функциональные возможности: По сравнению с некоторыми другими платформами, Lichess.org может предлагать меньше продвинутых функций и инструментов для игры и обучения. Некоторые игроки могут пожелать больше возможностей для анализа партий или общения с другими игроками.
    - Ограниченные социальные возможности: Lichess.org фокусируется в основном на игровом опыте, и у него может быть меньше возможностей для социального взаимодействия и общения с другими игроками по сравнению с другими платформами.

1. Chess24.com:

Chess24.com известен своими профессиональными трансляциями, высококачественными обучающими материалами и интерактивными функциями.

Преимущества:

* + - Профессиональные трансляции и обзоры партий: Chess24.com предлагает качественные трансляции шахматных турниров и обзоры партий от опытных шахматистов, что делает его привлекательным для любителей и профессионалов. Это позволяет пользователям быть в курсе последних шахматных новостей и узнавать новые стратегии и идеи от опытных игроков.
    - Обширные обучающие материалы: Платформа предоставляет широкий спектр обучающих материалов и курсов, которые помогают игрокам улучшать свои навыки и расширять свои знания в шахматах. Это включает в себя видеоуроки, учебники, практические задания и многое другое.
    - Интерактивные функции и анализ: Chess24.com предоставляет инструменты для анализа партий, компьютерную оценку позиций и возможность общения с другими игроками и тренерами. Это способствует более глубокому и качественному обучению шахмат и помогает игрокам развивать свои навыки на новом уровне.

Недостатки:

* + - Платные возможности и подписка: Некоторые продвинутые функции и материалы могут быть доступны только с платной подпиской, что может быть недоступно для некоторых пользователей или создавать дополнительные расходы. Это может быть недостатком для тех, кто хочет получить доступ ко всем функциям платформы, но не готов платить за это.
    - Сложность интерфейса: Некоторые пользователи могут считать интерфейс Chess24.com менее интуитивно понятным по сравнению с другими платформами, что может потребовать времени на освоение и привыкание. Это может быть недостатком для новичков или тех, кто предпочитает более простой и понятный интерфейс.

Далее, для полного понимания функционала игрового приложения «Ход», необходимо ознакомиться с режимами игры, которые были представлены в сетевых шахматах и из-за которых поднялся интерес к такому типу игр. Эти различные режимы обогащают опыт шахматистов, предлагая разнообразные вызовы и стратегии для исследования.

1. Классические шахматы:

В классических шахматах каждая фигура имеет свою уникальную роль и ценность на доске. Это стратегическая игра, которая требует от игроков умения планировать ходы вперед и адаптироваться к изменяющейся ситуации на доске. Хотя правила классических шахмат остаются неизменными, но каждая партия представляет собой новый вызов для умственного развития и творчества.

Функционал: В классических шахматах каждая фигура имеет свои уникальные ходы и ценность. Игроки стремятся контролировать центральные позиции, развивать свои фигуры, защищать своего короля и нападать на короля противника. Победа достигается путем матирования короля противника.

Особенности: Этот режим шахмат является классическим и наиболее распространенным. Он базируется на стандартных правилах, которые были разработаны в течение веков и поддерживаются международной шахматной федерацией (FIDE).

1. Atomic chess:

Вариант Atomic Chess добавляет взрывной элемент к шахматам. Каждый раз, когда фигура атакует или бьет фигуру противника, происходит взрыв, который может изменить динамику игры и требовать нового уровня внимания к позиции на доске. Этот режим предлагает альтернативный способ игры, где тактика и осторожность имеют решающее значение. На рисунке 1.1.3 представлен пример “взрыва” фигуры.

Функционал: В Atomic Chess каждый ход, который приводит к уничтожению фигуры противника, вызывает взрыв, который уничтожает все фигуры вокруг битой фигуры, кроме королей. Цель игры остается та же - достигнуть матирования короля противника.

Особенности: Этот режим добавляет элемент неожиданности и стратегической глубины, так как игроки должны быть осторожны при создании атакующих комбинаций, чтобы избежать взрыва, который может нанести ущерб им самим.

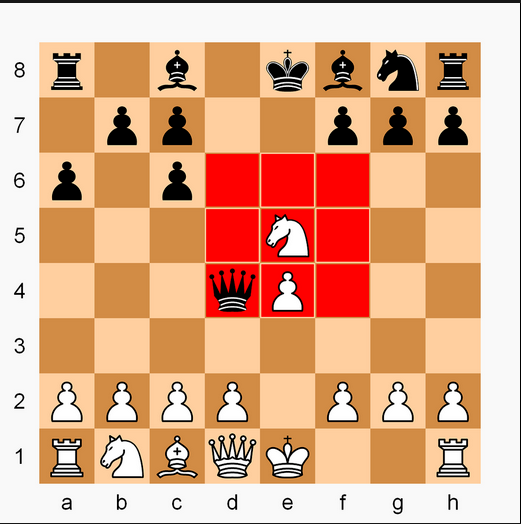


Рисунок 1.1.3 - пример “взрыва” фигуры

1. Crazyhouse:

Crazyhouse открывает двери для более динамичных и фантастических сражений. Возможность использовать захваченные фигуры противника позволяет игрокам расширить свои возможности на доске и создать уникальные комбинации. Этот режим представляет собой настоящее испытание для тех, кто ищет новые способы улучшить свою игру. На рисунке 1.1.4 представлен пример игры в crazyhouse.

Функционал: В Crazyhouse игроки могут использовать захваченные фигуры противника, возвращая их на доску в качестве своих собственных фигур. Это создает дополнительные возможности для тактических комбинаций и обмена материалом.

Особенности: Этот режим требует от игроков умения оценивать ценность фигур и прогнозировать ходы своего противника. Передача захваченных фигур может привести к неожиданным поворотам событий и динамичным позициям.

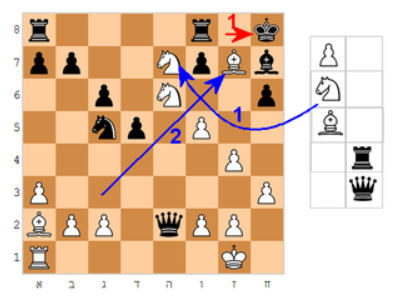


Рисунок 1.1.4 - пример игры в crazyhouse

1. Fischer Random Chess(960):

Fischer Random Chess, известный также как Chess960, обещает уникальный опыт каждый раз. Случайно сгенерированные начальные позиции фигур добавляют элемент случайности и стимулируют креативное мышление. Игроки должны быть готовы к адаптации к новым ситуациям и развивать умение быстро анализировать и оценивать позиции на доске. На рисунке 1.1.5 представлен пример расстановки фигур в режиме 960 chess.

Функционал: В Fischer Random Chess начальная позиция фигур случайным образом генерируется перед началом игры. Правила движения фигур остаются такими же, как и в классических шахматах, но начальные позиции фигур могут значительно отличаться от стандартных.

Особенности: Этот режим шахмат позволяет избежать изучения открытых начальных позиций и подчеркивает креативность и стратегическое мышление игроков. Каждая игра начинается с уникальной позиции, что создает новые вызовы и возможности для игроков.



Рисунок 1.1.5 - пример расстановки фигур в режиме 960 chess

1. Three-Check

Режимы King of the Hill и Three-Check вносят свои уникальные правила и цели, подчеркивая активное участие игроков и новые аспекты стратегии. Победа в этих режимах требует не только умения развивать свои фигуры, но и стратегического планирования, направленного на достижение особых целей на доске. На рисунке 1.1.6 представлен пример игры в режиме Three-Check chess.

Функционал: В режиме King of the Hill целью является достижение королем центральной клетки на доске. Победа достигается, когда король игрока достигает этой клетки и удерживает ее в течение определенного количества ходов. В режиме Three-Check игроки стремятся нанести три проверки королю противника, вместо того чтобы достигнуть матирования. Проверка короля происходит, когда король оказывается под атакой, но не находится под угрозой мата.

Особенности: Режим King of the Hill стимулирует игроков к активной игре и борьбе за контроль центра доски. Это также может привести к напряженным ситуациям, когда оба игрока борются за одну центральную клетку. Режим Three-Check придает игре более динамичный характер, поскольку игроки стремятся к быстрым проверкам, вместо того чтобы фокусироваться исключительно на защите своего короля. Это также может привести к быстрым завершениям партий и неожиданным результатам.



Рисунок 1.1.6 - пример игры в режиме Three-Check chess.

Каждый из этих режимов представляет собой увлекательное путешествие в мир шахмат, где игроки могут постоянно улучшать свои навыки и наслаждаться бесконечным разнообразием возможностей. Вместе они образуют богатый пейзаж шахматных вариаций, который вдохновляет и захватывает умы игроков по всему миру.

# 1.2 Постановка задачи

После рассмотрения аналогов и учитывая специфику нашего игрового приложения Ход мы определили несколько ключевых задач, которые будут выполнены в рамках данного проекта:

1.Разработка интуитивно понятного пользовательского интерфейса с необходимыми пунктами меню. Главная цель состоит в том, чтобы обеспечить удобство и простоту использования игрового приложения для пользователей.

2.Реализация логики игры Ход. Это включает в себя создание и управление игровым полем, установку фигур и определение правил движения и взаимодействия фигур в соответствии с концепцией atomic chess.

3.Обеспечение поддержки различных режимов игры, таких как классический режим, режим с измененными правилами или режим с разными вариантами стартовой позиции фигур. Это позволит пользователям настраивать игру под свои предпочтения и экспериментировать с различными вариантами игрового процесса.

Для реализации этих задач мы выбрали язык программирования C++, который обеспечивает высокую производительность и позволяет эффективно управлять объектами игры. Мы также используем фреймворк Qt для разработки графического интерфейса, так как он предоставляет удобные инструменты для создания пользовательских элементов управления и обработки событий.

Таким образом, наш курсовой проект посвящен разработке вспомогательных утилит для игрового приложения ХОД, которые сфокусированы на создании удобного пользовательского интерфейса, реализации игровой логики и поддержке различных режимов игры.

# СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Для разработки игрового приложения Ход был выбран язык программирования C++.

C++ - это мощный и гибкий язык программирования, который широко используется в разработке игровых приложений благодаря своей эффективности и возможностям. Вот несколько преимуществ использования C++ для создания игр:

1. **Производительность**: C++ является компилируемым языком, что означает, что код компилируется в машинный код, что позволяет достичь высокой производительности. Это особенно важно для игровых приложений, требующих обработки большого количества данных и выполнения сложных вычислений в реальном времени.
2. **Близость к аппаратному обеспечению**: C++ предоставляет доступ к низкоуровневым аппаратным ресурсам, что позволяет разработчикам оптимизировать игровые приложения под конкретные платформы и аппаратные устройства.
3. **Широкие возможности**: C++ предоставляет широкий набор инструментов и библиотек для разработки игр, таких как DirectX и OpenGL для работы с графикой, а также множество фреймворков и движков, например, Unreal Engine и Unity, которые частично написаны на C++.
4. **Контроль памятью**: В C++ разработчики имеют полный контроль над управлением памятью, что позволяет им оптимизировать использование ресурсов и избегать утечек памяти, особенно важно для игровых приложений с высокими требованиями к производительности.
5. **Переносимость**: Хотя C++ не является полностью переносимым языком, его код может быть относительно легко портирован на различные платформы, что позволяет создавать мультиплатформенные игры.
6. **Сообщество и ресурсы**: C++ имеет огромное сообщество разработчиков, которые активно обмениваются опытом, создают библиотеки и инструменты, а также предоставляют обширные ресурсы для изучения языка и его применения в разработке игр.
7. **Многопоточность и параллелизм**: C++ предоставляет возможности для эффективной работы с многопоточностью и параллелизмом, что особенно важно для игровых приложений, требующих обработки большого количества одновременных задач, таких как управление графикой, искусственный интеллект и физические расчеты.
8. **Настройка и оптимизация**: В C++ разработчики имеют полный контроль над каждой частью своего приложения, что позволяет им настраивать и оптимизировать его для достижения максимальной производительности и качества геймплея.
9. **Низкий уровень абстракции**: C++ предоставляет доступ к низкоуровневым конструкциям языка, таким как указатели и ссылки, что позволяет разработчикам реализовывать сложные алгоритмы и структуры данных, а также оптимизировать использование памяти и процессора.
10. **Возможность использования других языков**: В C++ можно интегрировать код, написанный на других языках программирования, таких как C, Assembly, и даже Python, что позволяет разработчикам использовать лучшие инструменты и библиотеки для своих проектов.
11. **Наследование от C**: C++ является расширением языка программирования C, что означает, что большинство существующего кода на C может быть использовано в проектах на C++, а также C++ сохраняет совместимость с библиотеками и инструментами, созданными для C.
12. **Большой выбор инструментов и библиотек**: Вокруг C++ существует огромное количество инструментов и библиотек, предназначенных специально для разработки игр, таких как библиотеки для работы с графикой, звуком, физикой и искусственным интеллектом, а также интегрированные среды разработки и фреймворки для создания игровых приложений.

В процессе разработки игрового приложения «Ход», определение функциональных блоков поможет структурировать проект, обеспечив удобное понимание его архитектуры и облегчив будущие расширения и изменения. Каждый из этих модулей играет важную роль в разработке игрового приложения. Вот как можно разделить приложение на функциональные блоки:

* 1. **Модуль Игровой Механики:**

Этот модуль содержит ключевые алгоритмы и логику, определяющую основные правила и механику игры Atomic Chess. Он включает в себя функции для проверки возможности хода каждой фигуры, обработки взрывов при уничтожении фигур и определения условий победы или ничьи. Важно, чтобы этот модуль был написан таким образом, чтобы обеспечить эффективную и точную реализацию правил Atomic Chess, включая особенности, такие как взрывы и перемещение королей.

**2.2 Модуль Пользовательского Интерфейса (UI):**

Этот модуль отвечает за визуальное представление игры и взаимодействие с пользователем. Он включает в себя элементы управления для перемещения фигур, отображения информации о текущей позиции на доске и другие пользовательские интерфейсные элементы. Хороший UI является ключевым фактором для создания приятного и удобного игрового опыта.

* 1. **Модуль Сетевой Игры:**

Так как наше игровое приложение позволяет играть по сети, этот модуль отвечает за управление сетевым взаимодействием между игроками. Это включает в себя установление соединения между клиентами, передачу игровых данных и синхронизацию состояния игры. Этот модуль также должен обеспечивать безопасность и защиту от мошенничества.

* 1. **Модуль Хранения Данных:**

Этот модуль отвечает за сохранение и загрузку игровых данных, таких как история матчей, настройки пользователя и другие параметры. Он может использовать базу данных или файловую систему для хранения данных и обеспечивать доступ к ним из других частей приложения. Реализация этого модуля должна обеспечить надежное и эффективное сохранение игровой информации.

* 1. **Модуль Тестирования:**

Этот модуль содержит тесты, которые проверяют правильность работы других модулей приложения. Тестирование игровой логики, пользовательского интерфейса и других компонентов приложения помогает обнаружить и исправить ошибки до их попадания в релизную версию. Это важный этап в разработке качественного игрового приложения.

* 1. **Модуль Управления Игровыми Состояниями:**

Этот модуль отвечает за управление различными состояниями игры и переходами между ними. Он определяет правила и логику для начала игры, хода игроков, проверки на победу и других аспектов игрового процесса. Это позволяет эффективно координировать и управлять игровым процессом и обеспечить плавный и интуитивно понятный игровой опыт для пользователей.

# ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого приложения.

**Описание классов приложения**

**3.1 Board**

Класс наследуется от классов Square, Player, PieceColor. Данный класс предоставляет комплексный интерфейс для управления шахматной доской в рамках программы, включая функциональность для игровой механики и управления состоянием.

Поля:

**-** Player \*bottomPlayer - Указатель на игрока, находящегося внизу доски. Используется для отслеживания и хранения информации о нижнем игроке, который обычно управляет фигурами белого цвета.

- Player \*topPlayer - Указатель на игрока, находящегося вверху доски. Используется для отслеживания и хранения информации о верхнем игроке, который обычно управляет фигурами черного цвета.

- Player \*currentPlayer - Указатель на игрока, который должен сделать следующий ход. Используется для отслеживания текущего игрока и определения очередности ходов.

- Square \*squares[8][8] - Двумерный массив указателей на объекты класса Square, представляющие клетки шахматной доски. Каждый элемент массива представляет одну клетку доски.

- vector<Move \*> allMoves - Вектор указателей на объекты класса Move, представляющие все сделанные ходы на доске. Используется для сохранения и отслеживания ходов в игре.

- vector<string> FENHistory - Вектор строк, содержащий историю досок в формате FEN (Forsyth-Edwards Notation). Используется для сохранения истории состояний досок, что позволяет восстанавливать прошлые позиции и повторять партии.

Методы:

- startGame(Player \*bottomPlayer, Player \*topPlayer, Player \*currentPlayer) - Начинает игру, принимая указатели на игроков и определяя, чей ход первый.

- doMove(Move \*nextMove) - Выполняет ход, передавая объект хода.

- undoMove() - Отменяет последний ход.

- checkGameStatus() - Проверяет текущее состояние игры, включая наличие шаха и мат.

- tooMuchRepetition() - Проверяет наличие слишком частого повторения позиций на доске.

- getPiecesByColor(PieceColor color) - Возвращает вектор фигур определенного цвета.

- isInCheck(PieceColor color) - Проверяет, находится ли король определенного цвета под шахом.

- checkMate() - Проверяет, находится ли какой-либо из королей в мате.

- getSquares() - Возвращает вектор всех клеток на доске.

- getSquare(short, short) - Возвращает определенную клетку на доске по ее координатам.

- getAllMoves() - Возвращает ссылку на вектор всех сделанных ходов.

- getFEN() - Возвращает строку в формате FEN, описывающую текущее состояние доски.

- getFENHistory() - Возвращает вектор строк, содержащих историю досок в формате FEN.

**3.2 ActionButton**

Класс предназначен для реагирования на события мыши, такие как нажатие или наведение курсора.

Методы:

- ActionButton(QString title) - Этот метод инициализирует объект класса ActionButton. Он может устанавливать начальные значения полей, например, заголовка, цвета или других свойств кнопки. Также конструктор может указывать, что кнопка может принимать события наведения и нажатия мыши.

- mousePressEvent(QGraphicsSceneMouseEvent \*event) - Этот метод обрабатывает событие нажатия мыши. Возможно, при нажатии кнопки будет вызван сигнал buttonPressed(). Можно предположить, что данный метод также может изменить внешний вид кнопки, чтобы показать, что она нажата.

- hoverEnterEvent(QGraphicsSceneHoverEvent \*event) - Этот метод вызывается, когда курсор мыши наводится на кнопку. Вероятно, в нём может быть вызван метод setBackgroundColor(), чтобы изменить цвет кнопки или произвести другую визуальную индикацию.

- hoverLeaveEvent(QGraphicsSceneHoverEvent \*event) - Этот метод вызывается, когда курсор покидает область кнопки. Он может восстанавливать первоначальный вид кнопки, например, меняя цвет обратно.

- setBackgroundColor(Qt::GlobalColor color) - Этот метод устанавливает цвет фона кнопки. Он может использоваться в других методах для изменения цвета при различных событиях (например, при наведении или нажатии).

**3.3 BasePawnModel**

Класс BasePawnModel описывает базовую структуру для представления шахматной фигуры. Это абстрактный класс, который должен служить основой для более конкретных типов шахматных фигур, таких как пешки, ладьи, ферзи и другие. Разберём его компоненты.

Поля:

- BoardPosition position - Хранит текущее положение фигуры на шахматной доске. Это объект, который, вероятно, представляет координаты на доске.

- PlayerType owner - Указывает, принадлежит ли фигура чёрному или белому игроку.

- PawnType type - Тип шахматной фигуры, который может быть одним из возможных значений: король, ферзь, ладья, слон, конь или пешка.

- QString imagePath - Путь к изображению, которое представляет данную фигуру на доске. Это может использоваться в графическом интерфейсе для отображения фигуры.

- bool didTakeFirstMove - Флаг, указывающий, делала ли фигура первый ход. Это особенно полезно для пешек, которые могут сделать двойной ход на первом перемещении.

Методы:

- BasePawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath) - Инициализирует объект BasePawnModel, устанавливая начальные параметры для положения, владельца, типа фигуры и пути к изображению.

- validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) - Это виртуальный метод, который должен быть реализован в классах-наследниках. Он определяет, допустимо ли перемещение фигуры на указанную позицию positionToMove. Он также учитывает фигуру, которая может находиться на этой позиции (pawnOnPositionToMove), и может возвращать изменённое requestedActivePawnPosition.

- pawnWantsToMoveByOneField(BoardPosition positionToMove) - Проверяет, перемещается ли фигура на одно поле от текущего положения. Этот метод может быть полезен для реализации перемещений, таких как ход короля или пешки при одном шаге.

- validateDiagonalMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove) - Проверяет, допустимо ли перемещение по диагонали, и может учитывать наличие другой фигуры на конечной позиции.

- validateVerticalOrHorizontalMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove) - Проверяет, допустимо ли перемещение по вертикали или горизонтали. Также может учитывать препятствия или захват других фигур.

- ~BasePawnModel() - Виртуальный деструктор, который обеспечивает корректное удаление объектов, наследуемых от BasePawnModel.

**3.4 BishopPawnModel**

Класс представляет шахматную фигуру "слон".

Методы:

- BishopPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath) - Конструктор инициализирует объект BishopPawnModel, вызывая конструктор базового класса с заданными параметрами. В этой точке объекту присваивается начальное положение, владелец (белый или чёрный), тип фигуры (слон), а также путь к изображению для отображения.

- validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) - Этот метод отвечает за проверку допустимости перемещения слона. Слон движется по диагонали, поэтому этот метод должен подтверждать, что перемещение соответствует диагональному пути, и нет препятствий на его пути.

**3.5 BoardField**

Класс BoardField представляет собой графический элемент, который является частью шахматной доски. Он наследует от QGraphicsRectItem, что позволяет отображать его как прямоугольник в графической сцене Qt.

Поля:

- BoardPosition position - Хранит позицию этого элемента на доске. Это объект, который содержит координаты или другие данные, позволяющие идентифицировать местоположение на шахматной доске.

- static int defaultWidthHeight - Это статическое поле, представляющее значение по умолчанию для ширины и высоты элемента. Оно может использоваться при создании объектов BoardField для задания размера квадрата, представляющего клетку шахматной доски.

Методы:

- BoardField(QColor backgroundColor, BoardPosition position, QGraphicsItem \*parent = nullptr) - Этот конструктор создаёт объект BoardField с заданным цветом фона (backgroundColor) и позицией (position). parent используется, если требуется вложение в другой графический элемент.

- BoardPosition getPosition() - Этот метод возвращает позицию BoardField на шахматной доске. Это позволяет узнать координаты клетки, которые могут быть использованы для различных целей, например, определения доступных ходов или установки шахматной фигуры на доске.

**3.6 BoardFrameField**

Класс BoardFrameField представляет собой графический элемент, используемый для отображения рамки или заголовка в графической сцене Qt.

Методы:

- BoardFrameField(QGraphicsItem \*parent = nullptr) - Этот конструктор создаёт объект BoardFrameField и может принимать родительский элемент (parent), который указывает, в каком контексте этот объект должен быть размещён. В конструкторе можно определить начальные свойства графического элемента, например, размер и цвет.

- setTitle(QString title) - Этот метод предназначен для установки заголовка или текста в объекте BoardFrameField. Он может быть использован для отображения текста на рамке или в другой области графической сцены. Заголовок может быть полезен для обозначения секции или предоставления контекста для пользователя.

**3.7 BoardView**

Класс BoardView представляет собой графическое представление шахматной доски, используя объекты Qt, такие как QGraphicsRectItem и QList. Класс содержит методы и поля, необходимые для отображения шахматной доски, управления фигурами, а также для визуализации некоторых элементов интерфейса. Рассмотрим компоненты этого класса и их применение.

Поля:

- static int numberOfRowsColumns - Статическое поле, указывающее количество строк и столбцов на шахматной доске. Обычно в шахматах это 8 строк и 8 столбцов.

- static int startXPosition -Статические поля, указывающие начальные координаты (по X), с которых начинается рисование шахматной доски.

- static int startYPosition -Статические поля, указывающие начальные координаты (по Y), с которых начинается рисование шахматной доски.

- QList<BoardField\*> fields - Список всех полей (клеток) на шахматной доске. Эти поля представляют собой объекты BoardField, которые могут быть частью доски.

- QList<PawnField\*> pawns - Список всех шахматных фигур на доске. PawnField вероятно представляет отдельную шахматную фигуру.

- QGraphicsTextItem \*checkWarningTitleTextItem, \*checkWarningDescriptionTextItem - Эти поля использованы для отображения текстовых предупреждений или заголовков, например, когда король находится под шахом.

Методы:

- BoardView() - Инициализирует объект BoardView и, вероятно, вызывает методы для создания шахматной доски и её элементов. Возможно, он также вызывает методы для начальной настройки, такие как размещение полей или отрисовка рамок.

- QList<BoardField\*> getFields() - Возвращает список всех полей на доске. Это может быть полезно, чтобы получить доступ ко всем элементам доски для последующих операций.

- void draw() - метод рисует шахматную доску, вызывая вспомогательные методы для размещения полей, фигур и других элементов интерфейса.

- void initializePawnFields(QList<BasePawnModel\*> pawns) - Принимает список шахматных фигур и инициализирует поля для этих фигур на доске. Это может быть вызвано при создании новой игры или при размещении фигур.

- PawnField\* getPawnAtBoardPosition(BoardPosition boardPosition) - Возвращает PawnField, представляющий шахматную фигуру, находящуюся в указанной позиции на доске. Полезно для поиска фигур по их координатам.

- PawnField\* getPawnAtMousePosition(QPoint point) - Возвращает PawnField, находящийся в указанной позиции мыши. Это может использоваться для определения, какую фигуру выбрал пользователь.

- void moveActivePawnToMousePosition(QPoint point, BasePawnModel \*pawn) - Перемещает активную шахматную фигуру (pawn) в позицию, соответствующую координатам мыши. Может быть вызвано, когда пользователь перемещает фигуру перетаскиванием или кликом.

- void placeActivePawnAtBoardPosition(BasePawnModel \*pawn, BoardPosition boardPosition) - Размещает шахматную фигуру в указанной позиции на доске. Используется при перемещении или расстановке фигур.

- void removePawnAtBoardPosition(BoardPosition boardPosition) - Удаляет фигуру из указанной позиции на доске. Полезно для удаления фигур при захвате или перемещении.

- void setPawnMoveCheckWarning(bool visible) - Устанавливает видимость предупреждения о шахе. Если король находится под шахом, это может быть визуально отображено.

- void promotePawnAtBoardPosition(BoardPosition boardPosition) - Выполняет преобразование (промоцию) шахматной фигуры, например, когда пешка достигает противоположной стороны доски. Метод должен изменить фигуру на доске соответствующим образом.

- void placeBoardFields() - Размещает поля на шахматной доске. Может использоваться в методе draw() для установки всех полей.

- void createFieldsColumn(int xPosition, int columnNumber) - Создаёт колонку полей (строк) на доске. Может быть полезно для организации доски по столбцам.

- void drawBoardFrame() - Отрисовывает рамку вокруг шахматной доски или другие элементы графического интерфейса.

- void drawBoardFrameAtPosition(QPoint point, QRectF rect, QString title) - Отрисовывает рамку или границы в заданной позиции.

- void drawCheckWarningTextItems() - Создаёт текстовые элементы для предупреждений о шахе или других важных событиях.

- QPointF getCoordinatesForBoardPosition(BoardPosition position) - Возвращает координаты, соответствующие указанной позиции на доске. Может использоваться для определения положения графических элементов.

**3.8 BoardViewModel**

Класс BoardViewModel представляет собой модель для представления состояния шахматной игры. Этот класс может использоваться для отслеживания положения шахматных фигур, управления ходами, проверки правильности движений и определения, чья очередь ходить, а также для выявления победителя. Он содержит методы для управления основными аспектами шахматной игры.

Поля:

- BasePawnModel \*activePawn - Указывает на активную шахматную фигуру, которая в данный момент находится в движении или выбрана для перемещения.

- PlayerType whosTurn - Определяет, чья очередь ходить (черный или белый).

- QList<BasePawnModel\*> blackPawns - Список всех шахматных фигур, принадлежащих черному игроку.

- QList<BasePawnModel\*> whitePawns - Список всех шахматных фигур, принадлежащих белому игроку.

- PawnViewModel pawnViewModel - Вспомогательная модель, вероятно, используемая для управления визуальным представлением фигур или доски.

- PlayerType \*winner - Указывает, кто победил в игре. Это поле может быть nullptr, если игра еще не завершена.

- bool isEnPassantAvailable - Указывает, доступен ли ход "взятие на проходе". Это особое правило в шахматах, применимое к пешкам.

Методы:

- BoardViewModel() - Инициализирует объект BoardViewModel, вызывая необходимые методы для установки начальных состояний, таких как список фигур, чья очередь ходить, и установка победителя на nullptr.

- QList<BasePawnModel\*> getBlackPawns()**,** QList<BasePawnModel\*> getWhitePawns() - Возвращают списки черных и белых шахматных фигур, соответственно.

- BasePawnModel\* getActivePawn() - Возвращает текущую активную шахматную фигуру.

- PlayerType getWhosTurn() - Возвращает игрока, чей ход в данный момент.

- PlayerType\* getWinner() - Возвращает победителя игры. Если игра еще не завершена, может вернуть nullptr.

- void setActivePawnForField(PawnField \*pawn) - Устанавливает активную фигуру, которая находится на указанном поле (PawnField).

- void setNewPositionForActivePawn(BoardPosition position) - Устанавливает новую позицию для активной фигуры. Может использоваться при перемещении или завершении хода.

- void discardActivePawn() - Сбрасывает активную фигуру, удаляя ссылку на нее. Обычно это происходит после завершения хода или отмены выбора фигуры.

- BoardPosition getBoardPositionForMousePosition(QPoint position) - Возвращает позицию на доске, соответствующую заданному местоположению мыши. Это может использоваться для определения, на какое поле указывает мышь.

- bool validatePawnPalcementForMousePosition(QPoint position) - Проверяет, допустимо ли размещение шахматной фигуры в позиции, соответствующей указанному местоположению мыши.

- bool validatePawnMove(BoardPosition positionToMove, BasePawnModel \*pawnToValidate = nullptr, BoardPosition \*requestedActivePawnPosition = nullptr) - Проверяет, допустимо ли перемещение фигуры на заданную позицию. Этот метод может использоваться для проверки правильности хода перед его выполнением.

- bool didRemoveEnemyOnBoardPosition(BoardPosition boardPosition) - Проверяет, была ли удалена вражеская фигура с указанной позиции на доске. Это может быть вызвано, когда фигура захватывает другую.

- bool isKingInCheck(PlayerType owner, bool isCheckingActivePlayer, BoardPosition positionToMoveActivePlayer) - Проверяет, находится ли король в шаху. Может использоваться для проверки, нарушает ли ход правила шахмат, где король не должен быть под шахом.

- bool didPromoteActivePawn() - Проверяет, была ли активная фигура (обычно пешка) промотирована, то есть преобразована в другую фигуру.

- void switchRound() - Меняет текущий ход на противоположного игрока. Это происходит после завершения хода.

- void initializePawns() - Инициализирует шахматные фигуры, вероятно, создавая начальное расположение для игры.

- void initializePawnsForRow(int rowNumber, PlayerType owner) - Создает фигуры для указанного ряда и определяет их принадлежность игроку (черный или белый).

- BasePawnModel\* getPawnOnBoardPosition(BoardPosition boardPosition) - Возвращает фигуру, находящуюся в указанной позиции на доске.

- bool validateAnotherPawnIntersection(BoardPosition positionToMove, BasePawnModel \*pawnToValidate, BoardPosition \*requestedActivePawnPosition = nullptr) - Проверяет, пересекается ли перемещение с другой фигурой, что может сделать ход недопустимым.

- bool validateKingsCheckForPawns(QList<BasePawnModel\*> pawns, bool isCheckingActivePlayer, BasePawnModel \*king, BoardPosition positionToMoveActivePlayer) - Проверяет, находится ли король под шахом при заданном перемещении, и учитывает список всех фигур, активного игрока, а также короля.

**3.9 CongratulationsView**

Класс CongratulationsView предназначен для отображения поздравления победителя в графической сцене Qt. Этот класс наследует от QObject для поддержки сигналов и слотов и от QGraphicsRectItem для графического представления в сцене. Он может быть использован для визуализации сообщений о победе в шахматной игре или других игровых контекстах.

Методы:

- CongratulationsView(PlayerType winner) - Инициализирует объект CongratulationsView с указанием победителя (PlayerType winner). Это позволяет классу отображать соответствующее поздравление или сообщение для победителя.

**3.10 Constants**

Класс Constants предоставляет статические константы, которые могут быть использованы в приложении для установки стандартных значений, таких как отступы, цвета и другие глобальные настройки. Этот класс может служить местом, где определяются значения по умолчанию, чтобы обеспечить согласованность в разных частях программы.

Поля:

- static int defaultMargin - Определяет значение по умолчанию для отступа. Это использовано для установки пространства между элементами в графическом интерфейсе или для определения внешних границ.

- static QColor defaultTextColor - Определяет цвет текста по умолчанию. Это может быть использовано для настройки цвета шрифтов или других графических элементов.

**3.11 GameView**

Класс GameView представляет собой графический компонент, который наследует от QGraphicsView. Этот класс предназначен для управления общим интерфейсом шахматной игры. Он включает в себя сцену, на которой расположена шахматная доска и другие элементы, а также методы и слоты для управления событиями, такими как нажатия мыши или начало игры.

Поля:

- QGraphicsScene \*scene - Основная графическая сцена, на которой размещаются все элементы интерфейса. Сцена содержит шахматную доску, фигуры и другие компоненты.

- BoardViewModel boardViewModel - Объект, который содержит логику игры, включая фигуры, чья очередь ходить, и прочее.

- bool gameStarted - Флаг, указывающий, началась ли игра.

- BoardView \*board - Представляет шахматную доску, на которой размещаются фигуры.

- PlayerView \*blackPlayerView, \*whitePlayerView - Представляют графические элементы для отображения информации о черном и белом игроках.

Методы:

- GameView() - Инициализирует объект GameView, устанавливая начальное состояние, создавая сцену и вызывая необходимые методы для начальной отрисовки.

- void displayMainMenu() - Отображает главное меню, вероятно, с вариантами для начала игры, настройки или выхода.

- void startGame() - Слот, который запускает игру, устанавливая необходимые флаги и вызывая методы для подготовки доски.

- void quitGame() - Слот, который завершает игру, возможно, возвращая в главное меню или закрывая приложение.

- void resetGame() - Слот, который сбрасывает игру, возвращая к начальному состоянию.

- void drawBoard() - Отрисовывает шахматную доску и размещает её на сцене. Возможно, вызывает вспомогательные методы для размещения фигур и других элементов.

- void drawSettingsPanel() - Отрисовывает панель настроек или другой интерфейс, связанный с управлением игрой.

- void drawUserPanel() - Отрисовывает панель для пользователя, возможно, для отображения информации о текущем игроке или состоянии игры.

- PlayerView\* drawViewForUser(PlayerType player) - Создаёт PlayerView для указанного игрока (черного или белого). Используется для отображения информации об игроках.

- void drawTitle(double yPosition, int fontSize) - Отрисовывает заголовок или текст на указанной позиции с определенным размером шрифта.

- void mousePressEvent(QMouseEvent \*event) - Обрабатывает события нажатия мыши. Этот метод может быть использован для выбора фигур или выполнения других действий при клике мыши.

- void mouseMoveEvent(QMouseEvent \*event) - Обрабатывает события движения мыши. Может использоваться для перемещения фигур или управления интерфейсом.

- void selectPawn(PawnField \*pawn) - Выбирает шахматную фигуру (PawnField) для перемещения или взаимодействия.

- void handleSelectingPointForActivePawnByMouse(QPoint point) - Обрабатывает выбор точки на доске с помощью мыши, возможно, для перемещения фигуры или другого действия.

- void setCheckStateOnPlayerView(PlayerType player, bool isInCheck) - Устанавливает состояние "шах" для указанного игрока, возможно, для отображения предупреждения.

- void moveActivePawnToSelectedPoint(QPoint point) - Перемещает выбранную фигуру к заданной точке на доске.

- void releaseActivePawn() - Освобождает активную фигуру после завершения её перемещения или отмены выбора.

- void showCongratulationsScreen(PlayerType winner) - Отображает экран поздравлений с победой, указывая на победителя.

**3.12 KingPawnModel**

Класс KingPawnModel представляет модель шахматной фигуры "король", наследуя от базового класса BasePawnModel. Этот класс реализует особые правила движения короля в шахматах, такие как однопольное перемещение в любом направлении и особый ход "рокировка".

Методы:

- KingPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath) - Инициализирует объект KingPawnModel, устанавливая начальную позицию короля, владельца (белый или черный), тип фигуры (король), а также путь к изображению для графического представления. Конструктор вызывает конструктор базового класса для инициализации общих свойств.

- bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) - Этот метод проверяет, допустимо ли перемещение короля в указанную позицию. В случае короля это означает перемещение на одну клетку в любом направлении — вертикально, горизонтально или по диагонали.

**3.13 KnightPawnModel**

Класс KnightPawnModel представляет собой модель шахматной фигуры "конь". Он наследует от базового класса BasePawnModel, предоставляя основные свойства шахматной фигуры, а также реализует правила перемещения коня в шахматах. Конь движется особым образом, что делает его уникальным среди других фигур.

Методы:

- KnightPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath) - Конструктор инициализирует объект KnightPawnModel, устанавливая начальное положение коня, владельца (белый или черный), тип фигуры (конь), а также путь к изображению для графического отображения. В конструкторе также вызывается конструктор базового класса, чтобы обеспечить начальную настройку общих свойств.

- bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) - Этот метод проверяет допустимость перемещения коня на заданное место. Перемещение коня в шахматах уникально, поскольку конь движется в форме буквы "Г": два поля в одном направлении и одно поле перпендикулярно к нему (или наоборот).

**3.14 PawnField**

Класс PawnField представляет собой графический элемент, который наследуется от QGraphicsRectItem. Он предназначен для представления шахматной фигуры на доске. Класс содержит методы для управления позицией, изображением фигуры, а также обеспечивает основную графическую функциональность. Рассмотрим компоненты этого класса.

Поля:

- BoardPosition position - Указывает на текущее положение фигуры на шахматной доске. BoardPosition представляет собой структуру с координатами x и y.

- QLabel \*imageLabel - Графический элемент для отображения изображения фигуры. QLabel обычно используется для отображения текста или изображений.

- QPixmap image - Содержит изображение, которое представляет фигуру. QPixmap используется для работы с графическими изображениями в Qt.

Методы:

- PawnField(BoardPosition position, QString imagePath, QGraphicsItem \*parent = nullptr) - Инициализирует объект PawnField, устанавливая его начальное положение (position), путь к изображению (imagePath), а также, если требуется, родительский элемент (parent).

- BoardPosition getPosition() - Возвращает текущее положение фигуры на доске. Этот метод полезен для определения местоположения фигуры в игре.

- void setPosition(BoardPosition position) - Устанавливает новое положение для фигуры на доске. Этот метод может использоваться при перемещении фигуры или изменении её положения.

- void setImage(QString imagePath) - Устанавливает новое изображение для фигуры. Этот метод может использоваться, если требуется изменить визуальное представление фигуры (например, при промоции пешки).

**3.15 PawnPawnModel**

Класс PawnPawnModel представляет модель шахматной фигуры "пешка". Этот класс наследует от базового класса BasePawnModel и реализует особенности перемещения пешки в шахматах. Пешка отличается от других фигур тем, что имеет уникальные правила движения и захвата.

Методы:

- PawnPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath) - Конструктор инициализирует объект PawnPawnModel, устанавливая начальную позицию, владельца (белый или черный), тип фигуры (пешка), и путь к изображению. Он также вызывает конструктор базового класса для выполнения общей инициализации.

- bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) - Этот метод проверяет, допустимо ли перемещение пешки на указанную позицию. Он учитывает уникальные правила движения пешки.

**3.16 PawnViewModel**

Класс PawnViewModel может использоваться для управления визуальными аспектами шахматных фигур в графическом контексте, а также для определения их начальных типов на основе позиции на шахматной доске. Этот класс содержит методы для получения пути к изображению, а также определения типа фигуры на основе её позиции.

Методы:

- PawnViewModel() - Инициализирует объект PawnViewModel. Возможно, он готовит класс для работы с шахматными фигурами, загружая ресурсы или выполняя начальные настройки.

- QString getImagePath(PawnType type, PlayerType owner) - Этот метод возвращает путь к изображению, которое соответствует указанному типу шахматной фигуры (PawnType) и её владельцу (PlayerType). Например, для белой ладьи это может быть путь к изображению белой ладьи. Этот метод может быть использован для загрузки правильных изображений при создании или обновлении фигур в графической сцене.

- PawnType getTypeForInitialPosition(BoardPosition position) - Возвращает тип шахматной фигуры (PawnType), основываясь на начальной позиции на шахматной доске. Например, на первой строке шахматной доски можно ожидать ладьи, коня, слона и так далее. Этот метод может быть полезен при инициализации шахматных фигур в начале игры или при перезапуске.

**3.17 PlayerView**

Класс PlayerView представляет собой графический компонент, который наследуется от QGraphicsRectItem. Этот класс может быть использован для отображения информации о шахматном игроке в графической сцене Qt. Он может включать информацию о текущем игроке, состоянии шаха, а также указывать на активного игрока.

Поля:

- static int defaultWidthHeight - Определяет стандартный размер графического элемента. Это может быть полезно при создании компонента для сохранения согласованного размера.

Методы:

- PlayerView(QGraphicsItem \*parent = nullptr) - Инициализирует объект PlayerView. Параметр parent указывает родительский графический элемент, если компонент является частью другой структуры. Конструктор может установить начальные значения, такие как размер, цвет, или создать внутренние графические элементы.

- void setPlayer(PlayerType owner) - Устанавливает, какому игроку принадлежит данный компонент (owner), будь то черный или белый игрок. Этот метод может изменить внешний вид или текст, чтобы отразить принадлежность игрока.

- void setActive(bool active) - Указывает, активен ли данный игрок. В контексте шахматной игры это может означать, чей сейчас ход. Если active равно true, компонент может изменить внешний вид, чтобы показать, что это активный игрок.

- void setIsInCheck(bool isCheck) - Устанавливает, находится ли король этого игрока под шахом. Если isCheck равно true, компонент может отображать предупреждение или менять цвет, чтобы указать на состояние шаха.

**3.18 QueenPawnModel**

Класс QueenPawnModel представляет шахматную фигуру "ферзь". Он наследует от BasePawnModel и реализует логику движения ферзя в шахматах. Ферзь может перемещаться на любое количество клеток по вертикали, горизонтали или диагонали, что делает его одной из самых сильных фигур в шахматах.

Методы:

- QueenPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath) - Инициализирует объект QueenPawnModel, устанавливая начальную позицию, владельца (черный или белый), тип фигуры (ферзь), а также путь к изображению. Конструктор вызывает конструктор базового класса BasePawnModel, чтобы выполнить базовую инициализацию.

- bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) - Этот метод проверяет, допустимо ли перемещение ферзя на указанное место. Ферзь может перемещаться по вертикали, горизонтали или диагонали на любое количество клеток, что объединяет характеристики движения ладьи и слона.

**3.19 RookPawnModel**

Класс RookPawnModel представляет шахматную фигуру "ладья". Он наследует от базового класса BasePawnModel и реализует особенности перемещения ладьи в шахматах. Ладья может перемещаться по вертикали или горизонтали на любое количество клеток, но не может перепрыгивать другие фигуры.

Методы:

- RookPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath) - Конструктор инициализирует объект RookPawnModel, устанавливая начальную позицию, владельца (черный или белый), тип фигуры (ладья), а также путь к изображению. Он вызывает конструктор базового класса для инициализации общих свойств.

- bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) - Этот метод проверяет, допустимо ли перемещение ладьи на заданную позицию. Ладья может двигаться по вертикали или горизонтали на любое количество клеток.

**3.20 Utils**

Класс Utils предоставляет вспомогательные функции для работы с графическими элементами в Qt. Этот класс содержит статические методы для выполнения общих задач, связанных с графическим интерфейсом, таких как установка фона, изображения или создание текстовых элементов. Использование утилитарных методов помогает избежать дублирования кода и обеспечивает удобство при работе с графическими элементами.

Методы:

- static void setBackgroundColor(QColor color, QAbstractGraphicsShapeItem \*item) - Устанавливает цвет фона для указанного графического элемента. Параметр color определяет цвет, который нужно установить, а item — графический элемент, цвет которого изменяется. Этот метод полезен при работе с элементами, у которых можно изменять цвет фона, такими как QGraphicsRectItem или QAbstractGraphicsShapeItem.

- static void setImage(QString imagePath, QGraphicsRectItem \*item) - Устанавливает изображение для графического элемента QGraphicsRectItem. Параметр imagePath определяет путь к изображению, которое нужно загрузить и установить, а item — элемент, в котором будет отображаться это изображение. Этот метод может быть использован для изменения графического представления элементов, таких как шахматные фигуры или иконки.

- static QGraphicsTextItem\* createTextItem(QString title, int fontSize, QColor textColor, QGraphicsItem \*parent = nullptr) - Создаёт текстовый элемент QGraphicsTextItem с заданными параметрами. title определяет текст, который будет отображаться, fontSize — размер шрифта, textColor — цвет текста, а parent — родительский элемент, к которому привязан созданный текстовый элемент. Этот метод полезен для добавления текста в графическую сцену.

1. **РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

**4.1 Разработка схем алгоритмов**

Метод CongratulationsView создает графический элемент, который отображает поздравления победителю в игре. Схема метода CongratulationsView показана в приложении Б.

Метод validateMove предназначен для проверки допустимости перемещения шахматной фигуры "пешка". Он учитывает множество факторов, таких как разница в координатах, направление движения, захват фигур, и особое правило первого хода для пешек. Схема метода validateMove показана в приложении В.

**4.2 Разработка алгоритмов**

Метод CongratulationsView класса CongratulationsView:

### Шаг 1: Установка заголовка

* **Создание текстового элемента**: Создается текстовый элемент titleItem с текстом "Congratulations!", размером шрифта 50 и цветом, заданным в Constants::defaultTextColor.
* **Установка позиции заголовка**: Чтобы заголовок был по центру, определяется его горизонтальная позиция titleXPosition как 600 минус половина ширины заголовка. Вертикальная позиция titleYPosition устанавливается на 100.
* **Установка позиции заголовка**: Заголовок titleItem размещается на определенных позициях с помощью setPos.

### Шаг 2: Установка изображения

* **Загрузка изображения**: Создается объект QPixmap из файла ":Images/confetti.svg".
* **Создание метки изображения**: Создается объект QLabel, который будет содержать изображение.
* **Создание прокси-виджета**: QGraphicsProxyWidget используется для включения стандартных виджетов Qt в графическую сцену. pMyProxy связывается с текущим объектом (this).
* **Настройка изображения**: Изображение масштабируется до 200x200 с сохранением пропорций и устанавливается на imageLabel.
* **Прозрачность фона**: Для imageLabel устанавливается атрибут Qt::WA\_TranslucentBackground, чтобы обеспечить прозрачность фона.
* **Добавление изображения в прокси-виджет**: Прокси-виджет pMyProxy связывается с imageLabel и устанавливается на позицию (500, 180).

### Шаг 3: Установка текста о победителе

* **Определение победителя**: Определяется, кто выиграл, используя условие winner == PlayerType::black ? "Player black" : "Player white". На основе этого создается строка winnerName.
* **Создание текстового элемента**: Создается descriptionItem с текстом, указывающим, кто победил, размером шрифта 25 и цветом из Constants::defaultTextColor.
* **Установка позиции текста**: Вычисляется горизонтальная позиция descriptionXPosition, чтобы центрировать текст. Вертикальная позиция descriptionYPosition устанавливается на 400.
* **Размещение текста**: Текст descriptionItem размещается на соответствующих позициях.

### Шаг 4: Добавление кнопки действия

* **Создание кнопки**: Создается объект ActionButton с текстом "Quit game".
* **Определение позиции кнопки**: Кнопка центрируется по горизонтали (600 минус половина ширины), а вертикальная позиция устанавливается на 500.
* **Установка позиции кнопки**: Кнопка actionButton размещается в заданных координатах.
* **Подключение сигнала к слоту**: Связывается сигнал buttonPressed() с методом quitGame() из объекта game. Это обеспечивает действие при нажатии кнопки.
* **Добавление элементов в сцену**: Кнопка actionButton и текущий объект this добавляются в сцену, что позволяет им отображаться в графическом контексте.

Метод validateMove класса PawnPawnModel:

### Шаг 1: Проверка на занятую позицию

* Если конечная позиция positionToMove занята фигурой того же владельца (pawnOnPositionToMove->owner == this->owner), то ход недопустим, возвращается false.

### Шаг 2: Вычисление разницы в координатах

* Разница в горизонтальной и вертикальной координатах (xDiference и yDiference) рассчитывается. Для перемещения пешки разница в горизонтали должна быть не более одной клетки, а в вертикали — не более двух клеток.

### Шаг 3: Проверка предельных значений

* Если горизонтальная или вертикальная разница больше допустимого (больше 1 или 2, соответственно), ход недопустим, возвращается false.

### Шаг 4: Проверка направления движения

* Пешки могут двигаться только в одном направлении: белые вверх, черные вниз. Проверяется, соответствует ли направление перемещения допустимому (wantsToMoveInGoodDirection).

### Шаг 5: Обработка разных сценариев перемещения

* **Одноклеточное перемещение**: Если пешка хочет двигаться на одну клетку (wantsToMoveByOneField), рассматриваются три сценария:
  + **Прямое движение**: Если xDiference равно 0, пешка движется вперед, и конечная позиция должна быть свободной.
  + **Диагональный захват**: Если конечная позиция содержит фигуру противника, и перемещение диагональное, захват допустим.
  + **Перемещение через препятствие**: Если requestedActivePawnPosition указывает на перекрытие позиций, или пешка хочет захватить фигуру по диагонали, это считается допустимым.
* **Двуклеточное перемещение**: Если пешка хочет двигаться на две клетки вперед (при первом ходе), проверяется, что ход прямой и нет препятствий.

1. **РЕЗУЛЬТАТ РАБОТЫ**

На рисунке 5.1 изображена начало работы программы. При старте программы показывается шахматная доска с фигурами, предоставлена возможность выбора цвета, а также сброс партии и выход. Позже игроки смогут начать играть.

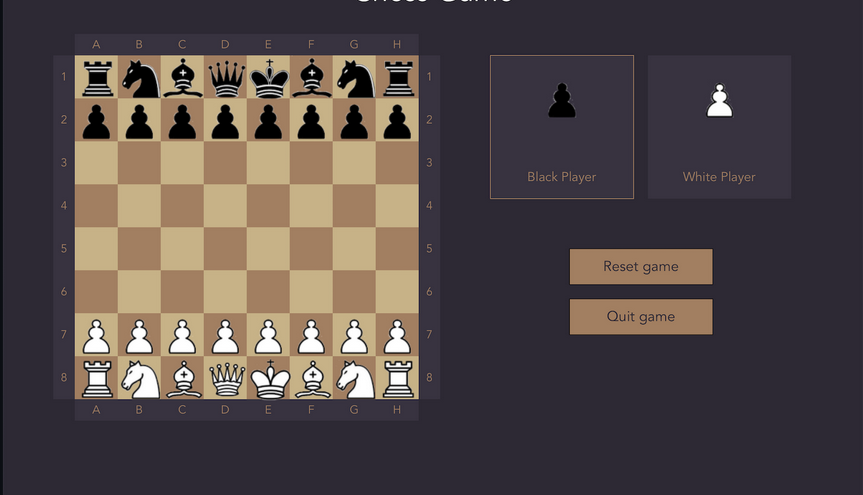


Рисунок 5.1 – начало работы игрового приложения “Ход”

На рисунке 5.2 можно детальнее рассмотреть игровую доску на которой будут происходить дальнейшие действия, которые будут влиять на ход игры.



Рисунок 5.2 – шахматное поле игрового приложения “Ход”

На рисунке 5.3 можно детальнее рассмотреть кнопки выбора сторон для взаимодействия с шахматными фигурами в игровом приложении “Ход”.

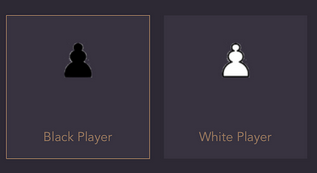


Рисунок 5.3 – кнопки выбора сторон игрового приложения “Ход”

На рисунке 5.4 можно детальнее рассмотреть кнопки “Сброс партии” и “Выйти из игры” для повторного запуска игрового приложения “Ход” или выхода из игры.



Рисунок 5.4 – кнопки “Сброс партии” и “Выйти из игры” в игровом приложении “Ход”

# ЗАКЛЮЧЕНИЕ

В процессе разработки игрового приложения "Ход" были успешно достигнуты первоначально поставленные цели. Реализована полноценная многопользовательская игра с удобным пользовательским интерфейсом, позволяющим игрокам участвовать в игре и принимать решения. Также в проекте были реализованы следующие ключевые элементы:

Логика игры: В игре "Ход" правильно реализована игровая механика, включая повзаимствованые особенности разновидностей шахмат, таймер и определение победы.

Взаимодействие с пользователем: Игра предоставляет удобный интерфейс для игроков, позволяющий им кастомизировать, выбирать сложность и многое другое.

Онлайн:

Игровое приложение “Ход” является многопользовательским, поэтому в игру может играть несколько человек.

Опции игры: Разработаны дополнительные функции, такие как возможность сброс партии, что делает игру более интересной и разнообразной.

Графика: для улучшения визуального восприятия игры были созданы графические элементы, включая изображения кнопок, фон доски, текстуры, анимации и многое другое.

Алгоритмы: Реализованы алгоритмы определения выигрыша, ходов и так далее.

В результате разработки игрового приложения “Ход” я понял, что данная игра сохраняет свою актуальность в современном мире развлечений и разработки игрового контента. “Ход” представляет собой не только классическую головоломку, но и важный элемент интеллектуальной культуры, который может быть успешно адаптирован и в современных виртуальных пространствах. Вот несколько аспектов, делающих игровое приложение “Ход” актуальным:

Универсальность:

Игровое приложение “Ход” подходит для всех возрастов. Игра предоставляют вызов как детям, так и взрослым, причём для каждого возраста существует своя сложность.

Развивающий аспект:

“Ход” способствует развитию когнитивных навыков, таких как память, внимание к деталям, концентрация внимания, логическое мышление, улучшение моторики рук и многое другое.

Онлайн:

Онлайн игры являются самым популярным жанром в мире, что доказывает актуальность игрового приложения “Ход”.

Вариативность:

Огромное разнообразие стратегий. Каждый найдёт стиль игры на свой вкус.

Социальность:

“Ход” помогает наладить сотрудничество между коллегами.

Игровое приложение "Ход" — это хороший пример того, как разработка игрового приложения может достичь успешных результатов, предоставляя пользователю увлекательный и интерактивный опыт. Приложение было тестировано на операционной системе Fedora Linux и разработано в Microsoft Visual Studio Code с помощью графики QT.

В процессе написания данной курсовой работы я глубоко погрузился в мир операционных систем и системного программирования через взаимодействие с видеокартой и рассмотрел разнообразные его аспекты. Изучение данных тем выдало для меня не только новые знания, но и значимый опыт, который открыл широкие перспективы в области разработки программного обеспечения.

Полученные знания стали ценным ресурсом для моей дальнейшей карьеры в сфере IT. Я уверен, что системное программирование в реальных проектах будет способствовать созданию более качественного и кроссплатформенного программного обеспечения. Изучение данных тем стало не только частью моего обучения, но и важным этапом в развитии моих навыков и понимания важности правильной организации кода для достижения поставленных целей в мире современной разработки ПО.

# СПИСОК ЛИТЕРАТУРЫ

1. "Таненбаум, Э. Современные операционные системы" Э. Таненбаум, Х. Бос
2. "UNIX. Профессиональное программирование" Стивенс, У. Р.
3. "UNIX для программистов и пользователей" Г. Гласс, К. Эйблс
4. "Программирование для UNIX" М. Д. Рочкинд
5. "Введение в алгоритмы" Кормен, Лейзерсон, Ривест, Штайн
6. "Искусство программирования для Unix" Э. С. Реймонд
7. "Программирование: введение в профессию" А. В. Столяров
8. "Операционная система UNIX : учебное пособие" А. М. Робачевский, С. А. Немнюгин, О. Стестик
9. "UNIX : разработка сетевых приложений" У. Р. Стивенс, Б. Феннер, Э. М. Рудофф
10. "Multiplayer Game Programming: Architecting Networked Games (Game Design)" Джошуа Глейзер, Санджай Мадхав

# ПРИЛОЖЕНИЕ А

*(обязательное)*

Диаграмма классов

# ПРИЛОЖЕНИЕ Б

##### *(обязательное)*

Схема метода CongratulationsView

# ПРИЛОЖЕНИЕ В

##### *(обязательное)*

Схема метода validateMove

ПРИЛОЖЕНИЕ Г

*(обязательное)*

Код программы

# ПРИЛОЖЕНИЕ Д

*(обязательное)*

Ведомость документов