Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Основы алгоритмизации и программирования»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И. С. Чайкин |
|  | \_\_\_.\_\_\_\_.2024 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Создание приложения для игры в нарды по сети против ИИ игрока»**

БГУИР КП 6-05-0612-02 001 ПЗ

|  |  |
| --- | --- |
|  | Выполнила студентка группы 353504  АНТОНОВА Лидия Сергеевна \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_\_\_\_\_\_.2024  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2024

**Содержание**

[Введение 5](#_Toc166915667)

[1 Платформа программного обеспечения 6](#_Toc166915668)

[1.1 Структура и архитектура платформы 7](#_Toc166915669)

[1.2 История 7](#_Toc166915670)

[1.3 Linux ubuntu 10](#_Toc166915671)

[1.4 Обоснование выбора платформы 12](#_Toc166915672)

[2 Теоретическое обоснование разработки программного продукта 14](#_Toc166915673)

[2.1 Обоснование необходимости разработки 14](#_Toc166915674)

[2.2 Технологии программирования, используемые для решения поставленных задач 14](#_Toc166915675)

[3 Проектирование функциональных возможностей программы 18](#_Toc166915676)

[3.1 Алгоритмы, которые могут быть использованы 18](#_Toc166915677)

[4 Тестирование 21](#_Toc166915678)

[4.1 Этапы тестирования 21](#_Toc166915679)

[4.2 Виды тестирования 22](#_Toc166915680)

[Заключение 28](#_Toc166915681)

[Список использованных источников 29](#_Toc166915682)

[Приложение А (обязательное) исходный код программы 30](#_Toc166915683)

[Приложение Б (обязательное) функциональная схема алгоритма 41](#_Toc166915684)

[Приложение В (обязательное) блок-схема алгоритма 43](#_Toc166915685)

[Приложение Г (обязательное) блок-схема алгоритма 45](#_Toc166915686)

[Приложение Д (обязательное) блок-схема алгоритма 47](#_Toc166915687)

[Приложение Е (обязательное) блок-схема алгоритма 49](#_Toc166915688)

[Приложение Ж (обязательное) блок-схема алгоритма 51](#_Toc166915689)

[Приложение И графики сравнения 53](#_Toc166915690)

# ВВЕДЕНИЕ

**Актуальность темы курсовой работы.**

Сегодняшний день является временем научно-технического прогресса, очень сложно представить себе жизнь и быт современного общества без использования искусственного интеллекта.

Искусственный интеллект (ИИ) становится все более важным в современном обществе. Разработка ИИ для игры в нарды поможет улучшить алгоритмы машинного обучения и искусственного интеллекта, что в свою очередь может привести к прогрессу в этой области.

Игры являются отличным способом обучения и развлечения. Создание приложения для игры в нарды с ИИ может помочь людям улучшить свои навыки в нардах, а также предоставить им возможность играть в любое время и в любом месте.

Использование С++ и Linux Ubuntu для создания приложения позволяет разработчикам улучшить свои навыки в этих областях. Это также может способствовать распространению этих технологий среди разработчиков.

Сетевые игры становятся все более популярными, поскольку они позволяют людям со всего мира играть друг с другом. Создание сетевого приложения для игры в нарды может привлечь большое количество пользователей.

В целом, тема курсовой работы может способствовать развитию искусственного интеллекта, обучению и развлечению пользователей, а также распространению С++ и Linux Ubuntu среди разработчиков.

**Цель и перечни задач:**

1 Разработать описание и перечень функций, которые будут

реализованы в программном продукте.

2 Разработать функциональную схему и блок-схему программного

продукта.

3 Разработать прототип пользовательского интерфейса.

4 Реализация программного продукта, учитывая функциональную схему

и прототип пользовательского интерфейса.

5 Тестирование программного продукта.

**Детальная постановки задачи.**

В данной работе рассмотрена операционная система Linux (её архитектура, история развития, достоинства и недостатки) и реализация игры нарды под данную систему.

Исходный код проекта расположен на github [1].

# 1 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Курсовой проект выполняется с использованием операционной системы Ubuntu 22.04.

Linux **— это** совокупность Unix-подобных операционных систем с открытым исходным кодом, основанных на ядре Linux. Это бесплатная операционная система с открытым исходным кодом, которой может быть изменен и распространен любым лицом на коммерческой или некоммерческой основе по лицензии GNU General Public License.

Логотип Linux представлен на рисунке 1.1.

Изображение выглядит как пингвин, текст, Нелетающая птица, мультфильм

Автоматически созданное описание

Рисунок 1.1 — логотип Linux

Дистрибутив Linux — это операционная система, которая состоит из набора программного обеспечения, основанного на ядре Linux и других вспомогательных библиотек.

Доступно более 600 различных дистрибутивов. Вот некоторые из популярных дистрибутивов Linux: Linux Mint, elementary, Ubuntu, Debian, Deepin и т.д.

Таким образом, Linux — это очень мощная и распространенная система.

## 1.1 Структура и архитектура платформы

Архитектура ОС Linux состоит из следующих компонентов [2]:

1 Ядро: Ядро Linux — это основа операционной системы Linux. Оно абстрагирует общие аппаратные ресурсы компьютера, чтобы предоставить каждому процессу его собственную копию виртуальных ресурсов. Что заставляет этот процесс думать так, как будто он единственный процесс, запущенный на компьютере. Ядро также отвечает за предотвращение и разрешение конфликтов между различными процессами.

Ядра ОС Линукс различаются по типу:

- Монолитное ядро

- Гибридные ядра

- Экзо-ядра

- Микроядра

2 Системная библиотека: это специальные функции, которые используются для реализации функциональности операционной системы.

3 Оболочка: это интерфейс к ядру, который скрывает сложность функций ядра от пользователей. Он принимает команды от пользователя и выполняет функции ядра.

4 Аппаратный уровень: Этот уровень состоит из всех периферийных устройств, таких как оперативная память, жесткий диск, процессор и т. д.

5 Системная утилита: Она предоставляет пользователю функциональные возможности операционной системы.

## 1.2 История

Корни Linux уходят в два других проекта: Unix и Multics, которые ставили своей целью разработать многопользовательскую операционную систему.

Что такое Unix?[3]

Unix – это собрание кроссплатформенных многопользовательских и многозадачных операционных систем.

Можно сразу сказать, что в данный момент Unix-системы являются одними из самых важных операционных систем. Влияние Unix распространилось и на языки программирования: язык C был разработан во время разработки Unix-систем.

Разработкой Unix занималась корпорация Bell Laboratories – в 1969 году они показали первую систему Unix. Чем дальше, тем большую популярность обретали системы Unix – в 70-х их начали устанавливать на компьютеры в учебных заведениях.

При создании Unix разработчики поставили перед собой три основные задачи:

- Использование минимального количества функций, сохранение простоты.

- Общность: одинаковые методы и механизмы в разных случаях.

- Комбинирование программ для решения задач, а не разработка новых программ с нуля.

Что касается отличительных особенностей Unix, то это:

- Практически постоянное использование командной строки.

- Использование контейнеров.

- Настройка системы через использование простых файлов.

Unix имеет свою собственную философию. Программист Дуглас Макилрой, который разработал контейнер в Linux, определил следующие правила:

- Пишите программы, которые делают что-то одно и делают это хорошо.

- Пишите программы, которые бы работали вместе.

- Пишите программы, которые бы поддерживали текстовые потоки, поскольку это универсальный интерфейс.

Одна из проблем, коснувшаяся Unix, – наличие разных версий и множества программ, которые писали разработчики под свои нужды. Из-за низкой совместимости программы, работающие с одной версией Unix, могли не работать на машинах с другими версиями. В итоге было решено создать общий документ со стандартами, которым должны следовать разработчики.

В 1983 году было объявлено о создании GNU, Unix-подобной операционной системы. Произошло это под влиянием идеи основателя проекта Ричарда Столманна о необходимости создания свободно распространяемой операционной системы и программного обеспечения с открытым исходным кодом.

Ричард Столманн также основал движение свободного программного обеспечения и сформулировал четыре права, которыми должен обладать пользователь: он может запускать программу для любых целей, он может изучать программы и изменять их согласно своим потребностям, он может распространять программу, чтобы помочь другим, и он может публиковать улучшения программы, чтобы помочь сообществу в целом. Все это говорило о том, что исходный код программы должен быть доступен всем.

Именно эта мысль вдохновила Линуса Торвальдса, создателя Linux, начать в 1991 году работу над своей операционной системой. Linux, как и GNU, это Unix-подобная система, то есть система, появившаяся под влиянием Unix.

В дальнейшем именно система GNU/Linux станет той системой, которую сейчас называют просто Linux.

Что такое Multics?

Multics – Multiplexed Information and Computing Service («Мультиплексная информационная и вычислительная служба») – это одна из самых первых операционных систем, в которой была реализована плоская модель хранения данных и четко разделена концепция файлов. Создание Multics началось в 1964 году. Над системой работали разработчики компании Bell Laboratories – через несколько лет часть разработчиков начнет работу над созданием Unix.

Multics разрабатывали для того, чтобы, во-первых, дать возможность использовать ресурсы ЭВМ большому количеству пользователей одновременно, во-вторых, дать пользователям возможность совместно использовать данные, в-третьих, обеспечить хорошую скорость работы с данными.

Однако главные вычислительные задачи не были достигнуты при выпуске первой версии системы, и компания Bell Laboratories перевела свой интерес на другой проект, в результате которого на свет появился Unix.

Истоки Linux

История Linux начинается в 1991 году, когда финский программист Линус Торвальдс стал разрабатывать ядро операционной системы для своего компьютера. Свои наработки он выложил на сервере, и это стало ключевым событием в истории Linux. Сначала десятки, потом сотни и тысячи разработчиков поддержали его проект – общими усилиями на свет появилась полноценная операционная система.

Как уже было сказано, на Linux значительно повлияла система Unix, это заметно даже по названию. Впрочем, изначально проект назывался Freax, от слов «free» (бесплатный) и «freak» (странный), но в дальнейшем название было изменено на гибрид имени создателя (Линус) и Unix.

Эмблемой Linux стал Такс (Tux) – пингвин, нарисованный в 1996 году программистом и дизайнером Ларри Юингом. Впрочем, идею использовать именно пингвина придумал сам Линус Торвальдс. Теперь Такс является символом не только Linux, но и свободного программного обеспечения в целом.

Первая официальная версия Linux 1.0 вышла в 1994 году, вторая – в 1996 году. Товарный знак Linux был зарегистрирован на год раньше – в 1995.

С самого начала и по сей день Linux распространяется как свободное программное обеспечение с лицензией GPL. Это значит, что исходный код операционной системы может увидеть любой пользователь, и не только увидеть, но и доработать его. Единственное условие – измененный, модифицированный код должен быть доступен всем и распространяться по лицензии GPL. Это важно, так как дает возможность разработчикам использовать код и в то же время не бояться проблем из-за авторских прав.

## 1.3 Linux Ubuntu

Ubuntu — это полностью бесплатная и открытая, простая в освоении операционная система (ОС), разработанная британской компанией Canonical Ltd. на базе дистрибутива Linux и Unix-подобной ОС Debian. Выпущена в трех редакциях: Desktop для настольных ПК и ноутбуков, Server для серверов и Core для интернета вещей.

Как появилась и развивалась ОС.[4]

История Ubuntu началась в 2004 году. Изначально это была не самостоятельная ОС, а надстройка другой GNU/Linux-системы, Debian. Ее главными разработчиками являются британский предприниматель Марк Шаттлворт и основанная им компания Canonical, в которую он набрал многих из тех специалистов, которые трудились над Debian. Сам он в прошлом также разрабатывал эту ОС и программное обеспечение для GNU/Linux. Новая операционная система взяла многое от «Дебиан» — например, систему управления пакетами Advanced Packaging Tool. Общей для обеих ОС является также концепция свободного и открытого ПО. Тем не менее, хотя Ubuntu основана на дистрибутиве Debian, некоторые компоненты обеих ОС несовместимы.

Первая версия Ubuntu под кодовым названием Warty Warthog была опубликована в открытом доступе в 2004 году и имела номер 4.10 — по месяцу и числу публикации. В дальнейшем такая нумерация станет традицией, с помощью которой разработчики будут обозначать новые версии, выходящие с периодичностью раз в полгода.

На рисунке 1.2 показан логотип Ubuntu.

Изображение выглядит как логотип, Графика, графическая вставка, Шрифт

Автоматически созданное описание

Рисунок 1.2 — логотип Ubuntu

Особенности Ubuntu.

1 Установка и использование.

Ubuntu легче и производительнее, чем Windows, поэтому она часто используется на старых маломощных ПК. Установить Ubuntu так же легко, как Windows. Не нужно ставить отдельно каждый драйвер и определять компоненты ядра.

2 Сообщество.

Основной разработчик Ubuntu — компания Canonical. Но у Ubuntu огромное сообщество, которое постоянно поддерживает операционную систему. Обновления выходят раз в полгода. При необходимости можно всегда обратиться за помощью на форуме, в чате и пр.

3 Безопасность.

Ubuntu — стабильный дистрибутив, особенно версия с длительной поддержкой. ОС не требуется антивирус: для Linux практически нет вирусов благодаря защищенной архитектуре. Ubuntu внимательно следит за безопасностью и вовремя закрывает найденные уязвимости. На случай потери данных есть мощные инструменты резервного копирования.

4 Поддержка популярных приложений.

Недостаток ПО — один из минусов семейства Linux. Ubuntu выделяется среди «собратьев». Canonical и сообщество энтузиастов развивают не только систему, но и программное обеспечение для нее. Все существующие сегодня приложения бесплатно доступны в официальном и публичном репозиториях.

Несмотря на то, что для Ubuntu больше приложений, чем для менее популярных версий Linux, их все еще меньше, чем для Windows и macOS. Особенно это касается игр и профессиональных платных программ. Бесплатные аналоги не всегда удобны и производительны. Поэтому пользователи, которым требуется специальное ПО, пользуются двумя операционными системами — Linux и Windows одновременно. Тем более что у обеих ОС имеется возможность установки другой платформы в пределах основной.

## 1.4 Обоснование выбора платформы

В ходе курсового проекта выбор программной платформы играет ключевую роль. В данном случае было принято решение использовать операционную систему Linux Ubuntu в качестве платформы для проведения исследований.

Преимущества Linux

Главное преимущество ОС Linux заключается в том, что это операционная система с открытым исходным кодом. Это означает, что исходный код легко доступен для всех, и любой желающий может изменять и распространять код ОС без каких-либо разрешений. С точки зрения безопасности Linux более безопасен, чем любая другая операционная система. Но это не означает, что на ПК можно запускать любые исполняемые файлы без разбора и Linux на 100 процентов безопасен, для него также есть вредоносные программы, но он менее уязвим, чем любая другая операционная система в силу своей архитектуры безопасности.

Linux находится в свободном доступе для скачивания в Интернете. Он пользуется большой поддержкой сообщества. Что обеспечивает ему высокую стабильность. Производительность системы Linux намного выше, чем у других операционных систем. Это позволяет большому количеству людей работать одновременно и эффективно справляться с ним. Удобен для работы в сети. Гибкость Linux также высока. Нет необходимости устанавливать полный комплект программ Linux; можно устанавливать только необходимые компоненты. Linux совместим с большим количеством форматов файлов.

Преимущества Ubuntu.

1 Человек ориентированность.

Если прочие дистрибутивы разрабатывались в основном для IT-специалистов, Ubuntu создавалась как «операционная система с дружественным интерфейсом». Дистрибутив считается одним из самых понятных среди всего семейства Linux. В нем удобный графический интерфейс, частично похожий на Windows и macOS, и простая навигация.

При этом Ubuntu — полноценный Linux с его возможностью глубокой настройки, централизованным скачиванием данных, функциональной консолью, с помощью которой управлять компьютером можно даже без графического интерфейса.

2 Доступность и многообразность.

Ubuntu развивается с учетом потребностей разных категорий пользователей. Операционную систему легко установить даже на слабый ПК или на нетбук, существуют версии для серверов, IoT, других устройств.

Многие разработчики облачных решений применяют эту систему из-за внедренной в ней поддержки OpenStack. Другие поставщики, разработчики устройств IoT и робототехники используют Ubuntu для экспериментов с новыми идеями.

3 Свобода и открытость.

Это принципы всех операционных систем семейства GNU/Linux. Само ядро Linux распространяется по лицензии GNU GPL, которая передает ПО в общественное пользование и делает его полностью открытым. Компания Canonical, которая занимается разработкой Ubuntu, придерживается похожих принципов.

На официальном сайте проекта указано, что все версии Ubuntu всегда будут бесплатными, а в комплекте с ОС поставляется только свободное программное обеспечение. Это заявление от разработчиков называется Ubuntu Promise, или «обещание Ubuntu». Оно исполняется и сейчас.

4 Минимальные системные требования.

Ubuntu имеет относительно низкие системные требования, что делает ее доступной для большого числа пользователей. Это может расширить потенциальную аудиторию игры.

5 Универсальность.

Ubuntu поддерживает широкий спектр аппаратного обеспечения и может быть установлена на различные платформы, включая персональные компьютеры, серверы и встроенные системы. Это делает Ubuntu гибкой и универсальной системой, которая может быть использована в различных средах разработки.

# 2 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

## 2.1 Обоснование необходимости разработки

На данных момент на рынке программных продуктов для игры в нарды под Linux имеется не так уж и много программ. В ходе анализа существующих программных продуктов выявлено, что графический интерфейс не соответствует ожиданиям многих пользователей. Данная курсовая работа будет нацелена на улучшения качества приложения для игры в нарды.

Использование Qt для разработки приложения обеспечивает множество преимуществ, таких как кроссплатформенность, богатые возможности для создания графического интерфейса и поддержка различных языков программирования. Это позволяет создать качественное приложение, которое будет работать на разных операционных системах, что обеспечит большее удобство для многих пользователей, желающих играть по сети на различных операционных системах.

## 2.2 Технологии программирования, используемые для решения поставленных задач

1 Язык программирования C++.

Для данного проекта можно выбрать несколько языков программирования, однако был выбран язык С++.

С++ — это высокоуровневый, компилируемый, небезопасный, строго типизированный и  статически типизированный язык программирования общего назначения. C++ содержит в себе все основные черты объектно-ориентированных языков программирования: наличие объектов и инкапсуляцию данных, наследование, полиморфизм и абстрак­цию типов. Наряду с этим C++ полностью поддерживает техно­логию структурного программирования.

Ниже представлены ключевые факторы, которые подчеркивают превосходство С++ в контексте задач такого типа.

Высокая производительность: С++ обладает производительностью, сравнимой с языком ассемблера. Это позволяет создавать высокоэффективные программы, что критически важно для оперативной обработки информации.

Уровень абстракции: Несмотря на схожесть с языком ассемблера, С++ — это высокоуровневый язык, близкий к человеку. Программирование на нем более интуитивно понятно, и разработчикам предоставляются удобные средства абстракции.

Богатство библиотек: C++ имеет обширный набор библиотек и фреймворков, упрощающих процесс разработки. От стандартных библиотек до инструментов, таких как STL (Standard Template Library), C++ предоставляет множество возможностей.

Управление памятью: Разработчики на С++ имеют полный контроль над управлением памятью. Это важно при создании вычислительных алгоритмов, где эффективное использование ресурсов играет ключевую роль.

Совместимость с C: С++ — это улучшенная версия языка C, сохраняющая совместимость с уже написанным кодом на C. Это облегчает интеграцию C++ в существующие системы.

В итоге С++ — оптимальный выбор для создания программного обеспечения, где важны эффективность, производительность и простота разработки.

2 Qt Creator.

Qt Creator — это кроссплатформенная, полностью интегрированная среда разработки (IDE), которая может быть использована для создания приложений для различных платформ, включая Android и iOS. Он доступен для Linux, Windows и macOS. Проекты в Qt Creator могут быть выполнены на языках программирования C++ или Python [5].

Основная цель Qt Creator — упростить процесс создания графических приложений с использованием фреймворка Qt на различных платформах. Он предлагает множество функций для работы с интерфейсом [6], что, в сочетании с его высокой функциональностью, эффективностью и удобством настройки рабочего пространства, делает его идеальной средой разработки для данного проекта.

Кроме того, Qt Creator обладает встроенным отладчиком, который позволяет разработчикам тестировать и отлаживать свои приложения прямо в IDE. Это значительно упрощает процесс отладки и помогает быстрее находить и исправлять ошибки в коде.

В итоге за счет его высокой функциональности и эффективности, в сочетании с удобной настройкой рабочего пространства, он является идеальной средой разработки для данного курсового проекта.

Открытое приложение Qt Creator выглядит как показано на рисунке 2.1.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 2.1 — Qt Creator

3 Графический дизайн и визуализация.

Внимание к графическому дизайну приложения весьма важно. Он должен соответствовать ожиданиям пользователей и обеспечивать удобство использования. В данном проекте графический дизайн также реализуется с помощью Qt Framework.

Qt Designer [7] — кроссплатформенная свободная среда для разработки графических интерфейсов (GUI) для программ, использующих библиотеку Qt. Входит в состав Qt framework.

Qt Designer позволяет создавать графические интерфейсы пользователя при помощи ряда инструментов. Существует панель инструментов «Панель виджетов», в которой доступны для использования элементы интерфейса — виджеты, такие как, например, «выпадающий список» ComboBox, «поле ввода» LineEdit, «кнопка» PushButton и многие другие. Каждый виджет имеет свой набор свойств, определяемый соответствующим ему классом библиотеки Qt. Свойства виджета могут быть изменены при помощи «Редактора свойств». Для каждого класса свойств виджета существует свой специализированный редактор.

Характерной особенностью Qt Designer является поддержка визуального редактирования сигналов и слотов. Так, например, можно связать сигнал, генерируемый по переключению состояния виджета CheckBox со слотом отвечающим за доступность другого виджета.

Qt Designer может быть запущен как отдельное приложение, так и во встроенном в IDE Qt Creator в виде, изображенном на рисунке 2.2.

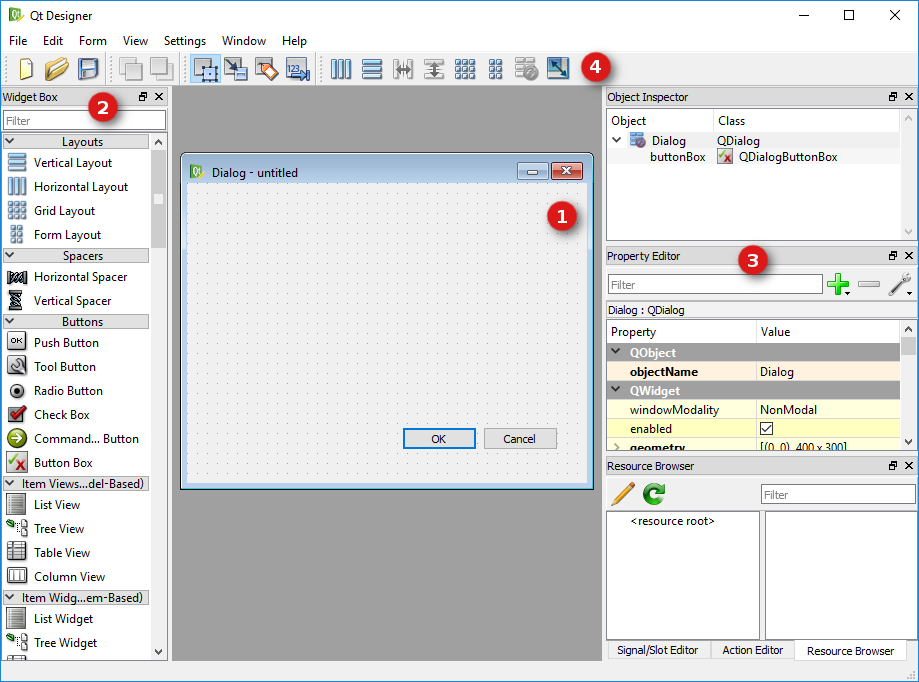


Рисунок 2.2 — Qt Desiner

Хороший графический дизайн — это не только красивый внешний вид, но и удобство использования.

4 Искусственный интеллект.

Искусственный интеллект (ИИ) — это увлекательная область, которая находится в центре современных технологий. Реализация бота для игры — это интересное и практичное применение ИИ. Он может решить сразу две проблемы:

- Самостоятельная игра: если у вас нет партнера для игры, бот может стать отличным выходом из данной ситуации.

- Прокачка навыков: если вы хотите потренировать свои навыки в игре, бот может стать отличным соперником. Вы можете настроить его уровень сложности и учиться разрабатывать стратегии, анализировать ходы и принимать решения.

5 Сетевое программирование.

Сетевое программирование позволяет вам создать возможность игры с друзьями по сети. Данное приложение сможет подключаться к другим экземплярам игры через сеть, обмениваться данными о ходах и синхронизировать состояние игрового поля между игроками.

# 3 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

В рамках данного проекта создается приложение для игры в нарды на С++ в Qt Creator. Игра в нарды — это двухигровая настольная игра с 15 фишками для каждого игрока и доской, разделенной на 24 ячейки [8]. Существуют короткие и длинные нарды.

Важные аспекты реализации игры в нарды:

- Движение фишек: в ходе игры все фишки игрока должны пройти путь по всему полю. В длинных нардах движение происходит против часовой стрелки.

- Бросок костей: Игроки бросают игральные кости, чтобы определить количество шагов. Если выпадает дубль (одинаковый бросок двух костей), общее число шагов удваивается. Например, при броске 4:4 игрок ходит не на 8 лунок вперед, а сразу на 16.

- Цель игры: Ваша цель — вернуть все свои фишки на свое домашнее поле и затем «выкинуть их» с помощью игральных костей. Победа достигается, когда вы исчерпаете все фишки на доске для игры в нарды.

- Бот: Разработка алгоритма для бота, который будет принимать решения о ходе на основе текущего состояния игры и правил. Учитываются стратегии движения фишек, броска костей и цели игры.

## 3.1 Алгоритмы, которые могут быть использованы

1 Алгоритм выполнения хода.

Сначала функция проверяет, не являются ли ячейки from и to одной и той же ячейкой, и доступна ли ячейка to для перемещения. Если оба условия выполняются, функция продолжает работу, в противном случае она прекращает выполнение и возвращает значение false.

Далее функция пытается удалить фишку из ячейки from. Если удаление прошло успешно, функция проверяет, является ли ячейка from «головой». Если это так, устанавливается флаг fromHead. Затем функция добавляет фишку в ячейку to и снимает выделение с ячейки from.

После этого функция вычисляет расстояние между ячейками from и to и удаляет это расстояние из списка доступных ходов. Наконец, функция вызывает endOfMovements() для завершения хода и возвращает true, указывая, что ход был успешно выполнен.

Таким образом, функция tryMakeMove(Cell &from, Cell &to) обеспечивает корректное выполнение ходов в данном проекте, обрабатывая различные сценарии, которые могут возникнуть во время игры.

2 Алгоритм считывания значений с кубиков.

Данный алгоритм представляет собой функцию getDicesMovements(), которая возвращает вектор целых чисел, представляющих собой значения бросков двух кубиков в игре.

Функция начинается с создания пустого вектора result для хранения результатов бросков кубиков. Затем проверяется условие равенства значений двух кубиков с помощью переменной sameDices.

Значение первого кубика добавляется в вектор result. Если значения обоих кубиков одинаковы (то есть sameDices истинно), то значение первого кубика добавляется в вектор result еще раз.

После этого значение второго кубика добавляется по тому же принципу.

В конце функция возвращает вектор result, содержащий значения бросков кубиков. Если значения обоих кубиков были одинаковыми, то в векторе будет четыре элемента, иначе - два.

3 Алгоритм проверки доступности хода.

Этот алгоритм представляет собой функцию validateMovement(), которая проверяет, возможно ли выполнить ход на заданное количество клеток в игре.

Функция начинается с вызова метода chipsRemoveAvailable(), который проверяет, доступно ли удаление фишек с доски. Результат этого вызова сохраняется в переменной removeAvailable.

Затем функция проходит по всем клеткам игровой доски в цикле. Если цвет фишек в текущей клетке совпадает с цветом фишек текущего игрока, функция определяет тип возможного хода с помощью метода getMoveType().

Если тип хода является обычным ходом или удалением фишки с доски при условии, что такое удаление доступно (то есть removeAvailable истинно), функция возвращает true, указывая на то, что ход возможен.

Если ни одна из клеток не позволяет сделать ход, функция возвращает false, указывая на то, что ход невозможен.

4 Алгоритм присвоения типа хода.

Алгоритм представляет собой функцию getMoveType(), которая определяет тип хода, основываясь на текущем состоянии игровой доски и заданном количестве клеток для перемещения.

Функция начинается с получения цвета фишек в клетке, откуда предполагается сделать ход. Если в этой клетке нет фишек, функция возвращает значение moveForbidden, указывая на то, что ход невозможен.

Затем функция определяет идентификатор клетки, в которую фишка должна переместиться после хода. Если фишка белого цвета и ее текущий идентификатор больше идентификатора клетки после хода, или если фишка черного цвета и ее текущий идентификатор меньше 12, а идентификатор клетки после хода больше 11, функция возвращает значение removeFromBoard, указывая на то, что фишка должна быть удалена с доски.

Далее функция получает цвет фишек в клетке, в которую предполагается сделать ход. Если в этой клетке есть фишки и их цвет не совпадает с цветом перемещаемой фишки, функция возвращает значение moveForbidden.

Если текущая клетка является «головой» и фишка должна быть перемещена из «головной» клетки, функция также возвращает значение moveForbidden.

Во всех остальных случаях функция возвращает значение regularMove, указывая на то, что предполагаемый ход является обычным ходом.

5 Алгоритм реализации бота.

Алгоритм представляет собой функцию aiMove(), которая реализует ход искусственного интеллекта в игре.

Функция начинается с проверки, есть ли доступные ходы. Если доступных ходов нет, функция вызывает метод endOfMovements() и завершает свою работу.

Затем функция создает вектор aiCells, который будет содержать все клетки доски, в которых есть фишки черного цвета. Если таких клеток нет, функция завершает свою работу.

После этого функция входит в цикл, который продолжается, пока есть доступные ходы. В каждом цикле функция выбирает случайную клетку из aiCells и пытается сделать ход из этой клетки.

Для каждого доступного хода функция определяет его тип с помощью метода getMoveType(). Если тип хода является обычным ходом, функция выбирает клетку, пытается сделать ход и, если ход успешен, завершает свою работу. Если ход не удался, функция отменяет выбор клетки и продолжает цикл.

Если удаление фишек с доски доступно и тип хода является удалением фишки с доски, функция выбирает клетку и удаляет фишку с доски. После этого функция завершает свою работу.

Таким образом, функция aiMove() реализует логику хода искусственного интеллекта в игре.

# 4 Тестирование

Тестирование — это проверка программного обеспечения, которая показывает, соответствует ли оно ожиданиям разработчиков и правильно ли работает.[9]

## 4.1 Этапы тестирования

Как правило, в большинстве проектов этапы тестирования схожи.

1 Проработка требований к продукту.

На этом этапе тестировщики внимательно изучают требования продукта — это могут быть документы, спецификации, описание того, как пользователь взаимодействует с продуктом (по-другому это называют пользовательскими сценариями). Четкое понимание требований помогает определить области, которые нужно протестировать.

2 Анализ требований.

Анализ требований позволяет выяснить, какие возможные риски или сложности могут возникнуть при тестировании. Также на этом этапе можно выявить возможные несоответствия или недостаточно ясные требования, которые требуют уточнения у разработчиков или заказчика.

3 Разработка стратегии и плана тестирования.

Когда все требования к продукту понятны, остается разработать план тестирования. В него входит:

- Выбор методов тестирования.

- Анализ потенциальных рисков, которые могут повлиять на качество и успешность тестирования, и планирование мер по их минимизации.

- Планирование ресурсов — кто будет тестировать продукт, каким оборудованием и инструментами можно при этом пользоваться и сколько времени займет тестирование, к какому сроку оно должно быть закончено.

4 Создание тестовой документации.

На этом этапе на основе требований и анализа тестировщики создают тестовые случаи, тест-планы, отчетность и другую документацию, которая будет использоваться во время тестирования. Тестовая документация определяет, какие тесты будут проведены, как будут собраны результаты и как будет оценено качество ПО.

5 Тестирование.

После того как команда утверждает стратегию тестирования и тестовую документацию, проводится тестирование. Тестирование программного обеспечения — это длительный и обширный процесс. По ходу составляются отчеты о выявленных недостатках, проводится набор тестовых сценариев, создается тестовая среда и выполняется тестирование согласно заранее задокументированным видам тестов, описанным в тестовой документации.

Важно понимать, что найти все ошибки в продукте невозможно. Главная цель заключается не в создании идеального продукта без ошибок, а в обнаружении максимального числа дефектов, которые могут потенциально повлиять на работу системы.

6 Эксплуатация и поддержка.

После того как разработчики устраняют дефекты и выпускают продукт, тестировщик переходит к тестированию продукта в рабочей среде. Важно отметить, что на этом этапе не только происходит релиз продукта, но и начинается пост-релизовая поддержка.

Невозможно предусмотреть все особенности использования и окружение, в котором будет работать продукт. Поэтому на данном этапе акцент делается на обратной связи пользователей. Теперь они становятся главными тестировщиками, а продукт становится частью их повседневной жизни. Устранение дефектов и поиск ошибок проводится быстро, но тщательно.

## 4.2 Виды тестирования

В своей работе тестировщики используют различные виды и методы тестирования, а также прорабатывают сценарии, в которых продукт может оказаться. Есть много способов тестирования, по разным оценкам в среднем их больше 30.

Каждый из видов тестирования направлен на проверку различных аспектов программного обеспечения. Условно их можно разделить на шесть групп.

1 По характеру сценариев.

Сценарий в тестировании — это описание того, как пользователь будет взаимодействовать с готовым продуктом. В эту группу входят два вида тестирования: позитивных сценариев и негативных.

Тестирование позитивных сценариев проверяет, как должна работать программа в нормальных условиях. Например, если это веб-приложение, тестирование позитивных сценариев проверит, что пользователь может успешно зарегистрироваться, войти в систему и без проблем использовать основные функции.

Тестирование негативных сценариев проверяет, как программа ведет себя в необычных или некорректных ситуациях. Такие сценарии показывают, что программа корректно обрабатывает ошибки и не позволяет пользователю выполнить действия, которые не предполагаются в нормальной работе приложения.

2 По критериям запуска программы или кода.

Критерии запуска программы или кода означают условия, которые необходимо выполнить для запуска тестов. То есть для тестирования необходимо, либо чтобы программа была запущена, либо, наоборот, не запущена. Под эту группу подходит статическое и динамическое тестирования.

Статическое тестирование — это вид проверки программного обеспечения, который выполняется без запуска программы. Вместо этого тестировщики анализируют исходный код программы или другие составляющие, например, документацию. Динамическое тестирование — это вид проверки программного обеспечения, который выполняется во время работы программы.

3 По степени автоматизации тестирования.

Ручное тестирование — это проверка программного обеспечения вручную, без использования автоматизированных инструментов. Тестировщик взаимодействует с программой как обычный пользователь.

В ходе ручного тестирования тестировщик выполняет различные сценарии использования и тестовые сценарии, вводит данные, наблюдает за результатами и проверяет, нет ли ошибок или неожиданного поведения. Если обнаруживаются проблемы, тестировщик документирует их, чтобы разработчики могли исправить ошибки.

Автоматизированное тестирование — это проверка программного обеспечения с использованием специальных программных инструментов, которые выполняют тесты автоматически, без участия человека. Тестировщик создает скрипты или сценарии тестирования, которые содержат инструкции для выполнения определенных действий и проверки результатов.

Эти сценарии запускаются на специальных инструментах для автоматизации тестирования, которые эмулируют действия пользователя и анализируют результаты выполнения.

4 По объектам тестирования.

Эта группа объединяет в себе виды, которые предполагают определение того, какие части программы или системы подвергаются тестированию.

Функциональное тестирование проверяет соответствие программы или системы заранее определенным функциональным требованиям и ожиданиям. Основная цель функционального тестирования — убедиться, что программа выполняет свои функции и операции согласно спецификациям, а также работает правильно и без сбоев.

Во время функционального тестирования тестируются различные сценарии использования, входные данные и выходные результаты, чтобы удостовериться в правильности работы приложения.

Функциональное тестирование делится на подвиды:

Unit-тестирование — проводится во время создания исходного кода. На этом этапе тестируются отдельные компоненты приложения. Тестировщики пишут тесты, чтобы убедиться, что каждый компонент будущей программы работоспособен и дает правильные результаты при различных входных данных.

Интеграционное тестирование. На следующем этапе тестируется то, как компоненты будущего приложения взаимодействуют между собой.

Системное тестирование. На этом этапе специалисты тестируют все компоненты программы как единое приложение. Тестировщики проверяют, что продукт корректно обрабатывает различные сценарии и ситуации.

Приемочное тестирование. На последнем этапе продукт тестирует уже клиент или заказчик. Они проверяют, соответствует ли проект их ожиданиям и требованиям. А еще убеждаются, что программа дает правильные результаты и работает без ошибок.

Нефункциональное тестирование проверяет нефункциональные аспекты программы — производительность, безопасность, надежность, масштабируемость и совместимость. Основная цель нефункционального тестирования — убедиться, что программа не только выполняет свои функции, но также соответствует требованиям к качеству, производительности и безопасности.

Нефункциональное тестирование часто охватывает атрибуты программы, которые не всегда видны конечному пользователю, но критически важны для обеспечения стабильной и надежной работы приложения.

Нефункциональное тестирование делится на подвиды:

Нагрузочное тестирование — для проверки производительности приложения под нагрузкой. Во время теста программа подвергается нагрузке: тестировщик, например, увеличивает число пользователей или операций и проверяет, как она будет работать.

Тестирование на проникновение — для проверки уровня безопасности. Этот вид тестирования проводится, чтобы узнать, насколько безопасна программа или веб-сайт от потенциальных кибератак и несанкционированного доступа. Тестировщики, как настоящие хакеры, используют различные методы, чтобы проверить защиту программы и предотвратить возможные угрозы для безопасности данных.

Тестирование совместимости. На этом этапе тестировщики проверяют работу программы на различных платформах, устройствах и браузерах, чтобы убедиться в их совместимости.

Стресс-тестирование — этот вид тестирования помогает выявить уязвимости и слабые места в системе, которые могут проявиться при больших нагрузках.

Тестирование на отказоустойчивость — помогает удостовериться, что приложение может успешно справляться с различными неполадками, такими как сбои серверов, потеря связи или другие неблагоприятные события, и продолжать функционировать нормально без значительных нарушений или потери данных.

Тестирование интерфейса пользователя — подразумевает проверку удобства, доступности и правильности работы пользовательского интерфейса программы.

Тестирование на восстановление. В ходе этого тестирования создаются различные сценарии отказов: отключение серверов или потеря связи, чтобы убедиться, что приложение может быстро и корректно восстановиться, и продолжить работу без значительных проблем.

5 По степени знания системы.

Эта группа объединяет в себе виды, которые используются в зависимости от того, насколько тестировщик знаком с тестируемым продуктом.

Тестирование «черного ящика» — это способ проверки программного обеспечения, когда тестировщик не знает внутренней структуры или деталей работы самой программы. Он смотрит на нее как на «черный ящик», и проверяет, как система взаимодействует с внешним миром и выполняет свои функции.

Такой подход позволяет сосредоточиться на тестировании того, как программа взаимодействует с пользователем и окружающей средой, не вдаваясь в детали ее внутренней реализации.

Тестирование «белого ящика», наоборот, предполагает, что тестировщик имеет доступ к внутренней структуре и коду программы. Он изучает, как работает программа «изнутри», чтобы убедиться, что все компоненты и функции написаны правильно и соответствуют требованиям.

Такой подход позволяет проверить детали реализации программы и выявить возможные ошибки, которые могли бы остаться незамеченными при тестировании «черного ящика».

Существует еще и тестирование «серого ящика» — это комбинация тестирования «черного ящика» и «белого ящика». Тестировщик знает некоторые детали внутренней структуры программы, но не обладает полной информацией о них. Он проверяет как внешнее поведение программы, так и использует некоторые знания о коде для определения эффективности и корректности работы программы.

Этот подход позволяет объединить преимущества обоих типов тестирования и обеспечить более полное и всестороннее тестирование программного обеспечения.

Под группу «по степени знания системы» также подходит еще несколько видов тестирования:

Тестирование по документации — основано на анализе документов, созданных во время разработки — спецификации, требования и дизайн. Тестировщики изучают эти документы, чтобы понять, как должно работать программное обеспечение, и затем создают тестовые сценарии и тестовые случаи для проверки соответствия программы этим требованиям и спецификациям.

Интуитивное тестирование — выполняется на основе интуиции и опыта тестировщика, а не по строгому плану тестирования.

6 По времени проведения тестирования.

В эту группу попадают виды тестирования, которое проводят в разные моменты разработки продукта. Расположим эти виды в том порядке, в каком их проводят до официального выпуска продукта.

Альфа-тестирование — это этап тестирования программного обеспечения, который происходит перед его официальным выпуском и предполагает проверку продукта внутри компании-разработчика или ограниченной группой тестировщиков. Альфа-тестирование помогает выявить возможные проблемы и ошибки перед предоставлением продукта пользователю.

Дымовое тестирование — это быстрая проверка программного обеспечения, которую выполняют после внесения значительных изменений или обновлений в код. Этот вид тестирования напоминает «пробный пуск» программы, чтобы убедиться, что основные функции работают без критических ошибок.

Если после дымового тестирования в продукт добавляют какую-то фичу или просто хотят убедиться, что все предыдущие функции работают правильно, то проводят регрессионное тестирование. Тестировщики убеждаются, что новая функция работает правильно и выполняет свои задачи так, как ожидается, а все остальное не вызывает новых ошибок.

Приемочное тестирование выполняют представители заказчика, чтобы удостовериться, что продукт вышел качественным, и что за него можно заплатить деньги. Чтобы успешно пройти приемочное тестирование, обычно нужно просто выполнить тесты, которые доказывают соответствие программы требованиям.

И последний этап — бета-тестирование. Тестировщики предоставляют готовую программу ограниченной группе реальных пользователей, которые могут сами с ней повзаимодействовать. Юзеры выявляют дополнительные проблемы, чтобы получить обратную связь от пользователей и улучшить программу перед ее окончательным выпуском для широкой аудитории.

Итог

В рамках данного проекта было создано приложение, которое было тщательно протестировано на различных компьютерах с разнообразными процессорами.

Процессоры, на которых проводились тесты, включали в себя как высокопроизводительные, так и более обычные модели. Например, одним из процессоров, на котором проводились тесты, был \*\*Intel Core i7 – 10700K\*\*. Этот процессор известен своей высокой производительностью, особенно в играх, и обычно превосходит другие процессоры по мощности.

Тем не менее, приложение также было протестировано на более обычных процессорах, таких как AMD Ryzen 5 3500 U. Эти процессоры обычно используются для повседневных задач, и они не обладают такой же мощностью, как более дорогие модели. Однако, несмотря на это, приложение работало хорошо и без каких-либо проблем даже на этих процессорах.

В результате тестирования никаких проблем с функционалом разработанного приложения не было обнаружено. Это подтверждает, что приложение было правильно разработано и оптимизировано для работы на различных процессорах.

Таким образом, можно с уверенностью заявить, что разработанное приложение является надежным и эффективным решением, которое может быть успешно использовано на различных компьютерах с разными процессорами.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта было разработано приложение, позволяющее играть в нарды против компьютерного игрока. Это приложение представляет собой уникальное сочетание классической настольной игры и современных технологий, что делает его интересным как для любителей нард, так и для энтузиастов компьютерных игр.

Основной целью проекта было создание компьютерного игрока, способного эффективно играть в нарды, и эта цель была успешно достигнута. Компьютерный игрок, разработанный в рамках проекта, демонстрирует уровень игры, который делает его достойным соперником для любого игрока.

Важным аспектом проекта было обеспечение качественного пользовательского интерфейса, который был разработан с учетом потребностей пользователей и обеспечивает удобство и простоту использования приложения.

В целом, проект является успешным примером применения современных технологий в области развлечений и может служить основой для дальнейшего развития и улучшения приложения для игры в нарды против компьютерного игрока. Это может включать в себя улучшение алгоритмов игры, добавление новых функций и опций для пользователей, а также расширение возможностей приложения для проведения онлайн-турниров и соревнований.

Таким образом, проект демонстрирует возможности и перспективы использования современных технологий в игровой индустрии, открывая новые горизонты для развития и инноваций.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Github [Электронный ресурс]. — Режим доступа: <https://github.com/Vediz-antonova/Backgammon>. – Дата доступа: 18.05.2024.

[2] MyRusakov [Электронный ресурс]. – Режим доступа: <https://myrusakov.ru/what-is-it-linux-part-2.html>. – Дата доступа: 06.04.2024.

[3] Habr [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/1cloud/articles/441554/> – Дата доступа: 10.05.2024.

[4] Skillfactory [Электронный ресурс]. – Режим доступа: <https://blog.skillfactory.ru/glossary/ubuntu/>– Дата доступа: 02.05.2024.

[5] Qt Group [Электронный ресурс]. – Режим доступа: <https://www.qt.io/product/supported-platforms-languages>. – Дата доступа: 24.04.2024.

[6] Qt Documentation [Электронный ресурс]. – Режим доступа: <https://doc.qt.io/qt-6/qmessagebox.html>. – Дата доступа: 23.04.2024.

[7] Qt Documentation [Электронный ресурс]. – Режим доступа: <https://doc.qt.io/qt-6/designer-using-a-ui-file.html>. – Дата доступа: 15.05.2024.

[8] Habr [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/timeweb/articles/691580>. – Дата доступа: 24.04.2024.

[9] Hexlet [Электронный ресурс]. – Режим доступа: <https://ru.hexlet.io/blog/posts/vidy-testirovaniya>. – Дата доступа: 28.04.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Исходный код программы

// MainWidget.cpp

#include "MainWidget.h"

#include "ui\_MainWidget.h"

MainWidget::MainWidget(QWidget \*parent)

: QWidget(parent)

, ui(new Ui::MainWidget)

{

ui->setupUi(this);

this->showMaximized();

scene = new QGraphicsScene();

QPixmap pixBoard("/home/vediz/OAiP/pictures/background.png");

pixBoard = pixBoard.scaled(this->size(), Qt::KeepAspectRatioByExpanding);

QGraphicsPixmapItem \*item = new QGraphicsPixmapItem(pixBoard);

scene->addItem(item);

ui->graphicsView->setScene(scene);

ui->graphicsView->setRenderHint(QPainter::Antialiasing);

ui->graphicsView->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

ui->graphicsView->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

game = &Game::getInstance();

Game::getInstance().setScene(scene);

Game::getInstance().startNewGame();

}

MainWidget::~MainWidget()

{

delete ui;

}

// MainWidget.h

#ifndef MAINWIDGET\_H

#define MAINWIDGET\_H

#include <QWidget>

#include <QGraphicsScene>

#include <QShortcut>

#include <QTimer>

#include <QGraphicsView>

#include <QGraphicsPixmapItem>

#include <QMouseEvent>

#include <QResizeEvent>

#include "game.h"

#include "rulesscreen.h"

QT\_BEGIN\_NAMESPACE

namespace Ui {

class MainWidget;

}

QT\_END\_NAMESPACE

class MainWidget : public QWidget

{

Q\_OBJECT

public:

explicit MainWidget(QWidget \*parent = 0);

~MainWidget();

private:

Ui::MainWidget \*ui;

QGraphicsScene \*scene;

Game \*game;

};

#endif // MAINWIDGET\_H

// cell.cpp

#include "cell.h"

Cell::Cell() {

chipsColor = none;

chipsCount = 0;

selected = false;

availableToMove = false;

}

Cell::~Cell() {}

int Cell::getId() const { return id; }

void Cell::setId(int value) { id = value; }

bool Cell::getSelected() const { return selected; }

void Cell::setSelected(bool value) {

selected = value;

update();

}

bool Cell::getAvailableToMove() {

return availableToMove;

}

void Cell::setAvailableToMove(bool value) {

if (availableToMove != value) {

availableToMove = value;

update();

}

}

bool Cell::addChip(ChipColor color) {

if (chipsColor != none && chipsColor != color) {

return false;

}

if (chipsCount >= 15) {

return false;

}

chipsColor = color;

chipsCount += 1;

update();

return true;

}

bool Cell::removeChip() {

if (getChipsCount() > 0) {

chipsCount -= 1;

if (chipsCount == 0) {

chipsColor = none;

}

update();

return true;

}

return false;

}

int Cell::getChipsCount() const {

return chipsCount;

}

ChipColor Cell::getChipsColor() const {

return chipsColor;

}

void Cell::setCallbackFunc(void (\*func) (int id)) {

callbackFunc = func;

}

void Cell::paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) {

Q\_UNUSED(option);

Q\_UNUSED(widget);

for (int i = 0; i < chipsCount; i++) {

drawChip(painter, i);

}

drawAvailableMark(painter);

}

QRectF Cell::boundingRect() const

{

return QRectF(0, 0, CELL\_WIDTH, CELL\_HEIGHT);

}

void Cell::mousePressEvent(QGraphicsSceneMouseEvent \*event) {

Q\_UNUSED(event);

if (callbackFunc) {

callbackFunc(id);

}

}

void Cell::drawChip(QPainter \*painter, int pos) {

Qt::GlobalColor color = chipsColor == white ? Qt::white : Qt::black;

Qt::GlobalColor borderColor = chipsColor == white ? Qt::black : Qt::white;

int chipYOffset = pos \* CHIP\_SHIFT;

int chipY;

if (isTopRow()) {

chipY = chipYOffset;

} else {

chipY = CELL\_HEIGHT - CHIP\_SIZE - chipYOffset;

}

if(selected && pos == getChipsCount() - 1) {

painter->setPen(QPen(Qt::red, 3));

} else {

painter->setPen(borderColor);

}

painter->setBrush(color);

painter->drawEllipse(0, chipY, CHIP\_SIZE, CHIP\_SIZE);

}

void Cell::drawAvailableMark(QPainter \*painter) {

if (availableToMove) {

int markX = (CELL\_WIDTH - AVAILABLE\_MARK\_SIZE) / 2;

int markY = getChipsCount() \* CHIP\_SHIFT + AVAILABLE\_MARK\_Y\_OFFSET;

if (getChipsCount() > 0) {

markY += CHIP\_SIZE - CHIP\_SHIFT;

}

if (!isTopRow()) {

markY = CELL\_HEIGHT - AVAILABLE\_MARK\_SIZE - markY;

}

painter->setBrush(Qt::blue);

painter->setPen(Qt::blue);

painter->drawEllipse(markX, markY, AVAILABLE\_MARK\_SIZE, AVAILABLE\_MARK\_SIZE);

}

}

bool Cell::isTopRow() {

return id < 12;

}

// cell.h

#ifndef CELL\_H

#define CELL\_H

#include <QObject>

#include <QGraphicsItem>

#include <QPainter>

#include <QGraphicsScene>

#include <QGraphicsSceneMouseEvent>

#include <utility>

#include "enums.h"

#include "sizes.h"

class Cell : public QObject, public QGraphicsItem {

Q\_OBJECT

public:

Cell();

~Cell();

int getId() const;

void setId(int value);

bool getSelected() const;

void setSelected(bool value);

bool getAvailableToMove();

void setAvailableToMove(bool value);

bool addChip(ChipColor color);

bool removeChip();

int getChipsCount() const;

ChipColor getChipsColor() const;

void setCallbackFunc(void (\*func) (int id));

void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget);

QRectF boundingRect() const;

void mousePressEvent(QGraphicsSceneMouseEvent \*event);

private:

int id;

ChipColor chipsColor;

bool selected;

bool availableToMove;

int chipsCount;

void (\*callbackFunc)(int id) = 0;

bool isTopRow();

void drawChip(QPainter \*painter, int pos);

void drawAvailableMark(QPainter \*painter);

};

#endif // CELL\_H

// currentplayer.cpp

#include "currentplayer.h"

CurrentPlayer::CurrentPlayer(QObject \*parent)

: QObject{parent}

{}

void CurrentPlayer::setColor(ChipColor color) {

chipColor = color;

update();

}

ChipColor CurrentPlayer::getColor(){

return chipColor;

}

void CurrentPlayer::paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) {

Q\_UNUSED(option);

Q\_UNUSED(widget);

Qt::GlobalColor color = chipColor == white ? Qt::white : Qt::black;

Qt::GlobalColor borderColor = chipColor == white ? Qt::black : Qt::white;

painter->setBrush(color);

painter->setPen(borderColor);

painter->drawEllipse(0, 0, CHIP\_SIZE, CHIP\_SIZE);

}

QRectF CurrentPlayer::boundingRect() const {

return QRectF(0, 0, CHIP\_SIZE, CHIP\_SIZE);

}

// currentplayer.h

#ifndef CURRENTPLAYER\_H

#define CURRENTPLAYER\_H

#include <QObject>

#include <QGraphicsItem>

#include <QPainter>

#include "enums.h"

#include "sizes.h"

class CurrentPlayer: public QObject, public QGraphicsItem

{

Q\_OBJECT

public:

explicit CurrentPlayer(QObject \*parent = nullptr);

void setColor(ChipColor color);

ChipColor getColor();

void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget);

QRectF boundingRect() const;

private:

ChipColor chipColor;

};

#endif // CURRENTPLAYER\_H

// dice.cpp

#include "dice.h"

Dice::Dice(QObject \*parent) : QObject(parent), QGraphicsItem()

{

setValue(1);

setEnabled(true);

}

Dice::~Dice(){}

bool Dice::isWhiteTurn = true;

void Dice::setValue(int new\_value) {

diceValue = new\_value;

update();

}

int Dice::getValue() { return diceValue; }

void Dice::setEnabled(bool value) {

enabled = value;

update();

}

bool Dice::getEnabled() {

return enabled;

}

/\*

void Dice::setPlayed(bool value) {

played = value;

update();

}

bool Dice::getPlayed() {

return played;

}

\*/

void Dice::setCallbackFunc(void (\*func) ()) {

callbackFunc = func;

}

void Dice::paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

Q\_UNUSED(option)

Q\_UNUSED(widget)

painter->setBrush(enabled ? Qt::white : Qt::gray);

painter->setPen(Qt::black);

painter->drawRect(0, 0, DICE\_SIZE, DICE\_SIZE);

painter->setPen(Qt::black);

painter->setBrush(Qt::black);

if (getValue() == 1 || getValue() == 3 || getValue() == 5) {

painter->drawEllipse(QPointF(DICE\_SIZE/2, DICE\_SIZE/2), DICE\_DOT\_SIZE, DICE\_DOT\_SIZE); // центральная точка

}

if (getValue() >= 2) {

painter->drawEllipse(QPointF(DICE\_DOT\_L, DICE\_DOT\_L), DICE\_DOT\_SIZE, DICE\_DOT\_SIZE); // верхний левый угол

painter->drawEllipse(QPointF(DICE\_DOT\_R, DICE\_DOT\_R), DICE\_DOT\_SIZE, DICE\_DOT\_SIZE); // нижний правый угол

}

if (getValue() >= 4) {

painter->drawEllipse(QPointF(DICE\_DOT\_R, DICE\_DOT\_L), DICE\_DOT\_SIZE, DICE\_DOT\_SIZE); // верхний правый угол

painter->drawEllipse(QPointF(DICE\_DOT\_L, DICE\_DOT\_R), DICE\_DOT\_SIZE, DICE\_DOT\_SIZE); // нижний левый угол

}

if (getValue() == 6) {

painter->drawEllipse(QPointF(DICE\_DOT\_L, DICE\_SIZE/2), DICE\_DOT\_SIZE, DICE\_DOT\_SIZE); // левая сторона

painter->drawEllipse(QPointF(DICE\_DOT\_R, DICE\_SIZE/2), DICE\_DOT\_SIZE, DICE\_DOT\_SIZE); // правая сторона

}

}

QRectF Dice::boundingRect() const

{

return QRectF(0, 0, DICE\_SIZE, DICE\_SIZE);

}

void Dice::Game()

{

setValue(QRandomGenerator::global()->bounded(1, 7));

update();

}

void Dice::mousePressEvent(QGraphicsSceneMouseEvent \*event)

{

Q\_UNUSED(event);

if (enabled && callbackFunc) {

callbackFunc();

}

}

// dice.h

#ifndef DICE\_H

#define DICE\_H

#include <QObject>

#include <QGraphicsItem>

#include <QPainter>

#include <QGraphicsScene>

#include <QPropertyAnimation>

#include <QRandomGenerator>

#include <QGraphicsSceneMouseEvent>

#include "sizes.h"

class Dice : public QObject, public QGraphicsItem

{

Q\_OBJECT

public:

explicit Dice(QObject \*parent = 0);

~Dice();

static bool isWhiteTurn;

void setValue(int new\_value);

int getValue();

void setEnabled(bool value);

bool getEnabled();

/\*

void setPlayed(bool value);

bool getPlayed();

\*/

void setCallbackFunc(void (\*func) ());

void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

QRectF boundingRect() const override;

void mousePressEvent(QGraphicsSceneMouseEvent \*event);

void Game();

protected:

int diceValue;

bool enabled;

// bool played;

void (\*callbackFunc)() = 0;

};

#endif // DICE\_H

// enums.h

#ifndef ENUMS\_H

#define ENUMS\_H

enum ChipColor {

none,

black,

white

};

enum MoveType {

moveForbidden,

regularMove,

removeFromBoard

};

#endif // ENUMS\_H

// sizes.h

#ifndef SIZES\_H

#define SIZES\_H

#define CHIP\_SIZE 70

#define CHIP\_SHIFT 25

#define CELL\_WIDTH CHIP\_SIZE

#define CELL\_HEIGHT 420

#define DICE\_SIZE 50

#define DICE\_DOT\_OFFSET 10

#define DICE\_DOT\_SIZE 5

#define DICE\_DOT\_L DICE\_DOT\_OFFSET

#define DICE\_DOT\_R DICE\_SIZE - DICE\_DOT\_OFFSET

#define LEFT\_BOARD\_OFFSET 245

#define RIGHT\_BOARD\_OFFSET 335

#define TOP\_CELL\_OFFSET 50

#define CELL\_X\_SPACE 24

#define AVAILABLE\_MARK\_SIZE 15

#define AVAILABLE\_MARK\_Y\_OFFSET 5

#define REMOVE\_BUTTON\_SIZE 100

#define REMOVE\_BUTTON\_CENTRE\_SIZE 20

#endif // SIZES\_H

# ПРИЛОЖЕНИЕ Б (обязательное) Функциональная схема алгоритма

# ПРИЛОЖЕНИЕ В (обязательное) Блок-схема алгоритма

# ПРИЛОЖЕНИЕ Г (обязательное) Блок-схема алгоритма

# ПРИЛОЖЕНИЕ Д (обязательное) Блок-схема алгоритма

# ПРИЛОЖЕНИЕ Е (обязательное) Блок-схема алгоритма

# ПРИЛОЖЕНИЕ Ж (обязательное) Блок-схема алгоритма

# ПРИЛОЖЕНИЕ И Графики сравнения

|  |
| --- |
| Ryzen 5000 Gaming Performance |
| Рисунок И.1 — Intel Core i7 – 10700K && Intel core i9 – 10980XE |
| Ryzen 7 5800X and Ryzen 5 5600X Gaming Benchmarks |
| Рисунок И.2 — AMD Ryzen 5 5600X && AMD Ryzen 5 3600X |