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  старший преподаватель кафедры информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Д.Владымцев |
|  | \_\_\_.\_\_\_\_.2024 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«РАЗРАБОТКА ИГРЫ TETRIS НА QT»**

БГУИР КП 6-05 0612 02 009 ПЗ

|  |  |
| --- | --- |
|  | Выполнила студентка группы 353504  ГУСЕНЦОВА Екатерина Андреевна \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2024  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2024

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc168300896)

[1 Архитектура вычислительной системы 7](#_Toc168300897)

[1.1 Режимы работы 8](#_Toc168300898)

[2 Платформа программного обеспечения 10](#_Toc168300899)

[2.1 Преимущества использования Linux в качестве среды разработки системы Android 10](#_Toc168300900)

[2.2 Архитектура ядра Linux 11](#_Toc168300901)

[2.3 Дистрибутивы Linux 13](#_Toc168300902)

[3 Теоретическое обоснование разработки программного продукта 15](#_Toc168300903)

[3.1 Правила игры 16](#_Toc168300904)

[3.2 Технологии программирования, используемые для решения поставленных задач 17](#_Toc168300905)

[4 Проектирование функциональных возможностей программы 19](#_Toc168300906)

[4.1 Алгоритм поиска в ширину 19](#_Toc168300907)

[4.2 Алгоритм поиска в глубину 19](#_Toc168300908)

[4.3 Сортировка подсчетом 19](#_Toc168300909)

[4.4 Алгоритм поворота фигуры 20](#_Toc168300910)

[4.5 Алгоритм ближайшего соседа 20](#_Toc168300911)

[5 Архитектура разрабатываемой программы 21](#_Toc168300912)

[5.1 Структура проекта в программной среде 21](#_Toc168300913)

[5.2 Точка входа в приложение 22](#_Toc168300914)

[5.3 Проверка работоспособности программного продукта 23](#_Toc168300915)

[Заключение 25](#_Toc168300916)

[Список использованных источников 26](#_Toc168300917)

[Приложение А (Обязательное) Исходный код программы 27](#_Toc168300918)

[Приложение Б (Обязательное) Функциональная схема программного продукта 36](#_Toc168300919)

[Приложение В (Обязательное) Блок-Схема алгоритма поиска в ширину 37](#_Toc168300920)

[Приложение Г (Обязательное) Блок-Схема алгоритма поиска в глубину 38](#_Toc168300921)

[Приложение Д (Обязательное) Блок-Схема сортировки подсчетом 39](#_Toc168300922)

[Приложение Е (Обязательное) Блок-Схема алгоритма поворота фигуры 40](#_Toc168300923)

[Приложение Ж (Обязательное) Блок-Схема алгоритма ближайшего соседа 41](#_Toc168300924)

[Приложение И (Обязательное) Графический интерфейс пользователя 42](#_Toc168300925)

# ВВЕДЕНИЕ

**Актуальность темы курсовой работы.**

В современном мире игровая индустрия является одной из самых динамично развивающихся и перспективных отраслей. Ежегодно миллионы игроков со всего мира увлечены созданием и прохождением новых игр, и это создает огромный спрос на качественный и разнообразный игровой контент. "Тетрис", несмотря на свою долгую историю, остается востребованным как среди ветеранов игровой индустрии, так и среди новых поколений игроков. Ее простой, но захватывающий игровой процесс привлекает как опытных геймеров, так и новичков. Разработка собственной версии этой культовой игры на современных технологиях предоставляет уникальную возможность внести свой вклад в мир игр и получить ценный опыт в разработке программного обеспечения.

Выбор фреймворка Qt для разработки игры "Тетрис" также обусловлен многофункциональностью, гибкостью и возможностью кроссплатформенной разработки и значимостью на рынке программной разработки. Qt обеспечивает разработчиков удобными инструментами для создания графического интерфейса, управления вводом-выводом, обработки событий и многого другого, что существенно упрощает и ускоряет процесс разработки. Qt является мощным инструментом для создания кроссплатформенных приложений, что позволяет разработчикам достичь широкой аудитории и обеспечить совместимость с различными операционными системами, такими как Windows, macOS и Linux. Богатый набор инструментов и библиотек Qt обеспечивает высокую производительность и эффективность в разработке, что делает его предпочтительным выбором для создания игровых приложений любой сложности.

Процесс разработки игры "Тетрис" на Qt также представляет собой отличную возможность для изучения и практического применения принципов объектно-ориентированного программирования (ООП), дизайна пользовательского интерфейса (UI) и алгоритмического мышления. Этот проект позволяет ознакомиться с основными этапами создания игрового приложения, применить полученные знания и навыки на практике, а также расширить свой опыт в программировании на языке C++ и работы с графическими интерфейсами. Разработка игры требует не только технических навыков, но и креативного подхода к решению задач, что способствует развитию у студентов и специалистов в области информационных технологий.

Разработка игры "Тетрис" на Qt также способствует пониманию важности коллективной работы и эффективного управления проектом. Ведь создание игрового приложения требует координации усилий различных специалистов, таких как программисты, дизайнеры и тестировщики. В процессе работы над проектом студенты учатся общаться, делиться идеями, принимать обратную связь и решать возникающие проблемы в команде.

Дополнительно, применение ООП позволяет легко поддерживать и тестировать игру. Каждый компонент игры может быть протестирован независимо от других, что упрощает обнаружение и устранение ошибок.

Таким образом, курсовая работа по разработке игры "Тетрис" на Qt представляет собой актуальный и перспективный проект, который позволяет развить навыки программирования, углубить знания в области разработки игровых приложений и получить ценный опыт в создании качественного программного обеспечения.

**Цель и перечни задач:**

1 Разработать детальное описание функциональности игры "Тетрис", определить основные игровые механики и возможности для игрока.

2 Создать функциональную схему, описывающую взаимодействие различных компонентов программного продукта, таких как игровое поле, фигуры тетриса, логика игры и пользовательский ввод.

3 Подготовить блок-схему, демонстрирующую последовательность действий и алгоритмы, необходимые для корректной работы игры "Тетрис".

4 Разработать прототип пользовательского интерфейса, который включает в себя основные элементы игры, такие как игровое поле, фигуры тетриса, информационные панели и кнопки управления.

5 Произвести реализацию программного продукта, основываясь на разработанных функциональной схеме и прототипе пользовательского интерфейса. Тестирование программного продукта.

**Детальная постановки задачи.**  
Целью курсового проекта является создание полнофункциональной игры "Тетрис" на языке программирования C++ с использованием принципов объектно-ориентированного программирования (ООП). Это означает, что мы стремимся не только к реализации игрового процесса, но и к созданию модульной, легко расширяемой и поддерживаемой системы, основанной на объектах и классах. Применение ООП позволяет абстрагировать различные элементы игры в виде объектов, каждый из которых имеет свои собственные свойства и методы, что способствует улучшению читаемости кода, упрощению его модификации и обеспечению повторного использования кода.

# 1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Архитектура x86-64, или AMD64/Intel 64, представляет собой значительное расширение архитектуры x86, используемой в большинстве современных персональных компьютеров и серверов. Это расширение было разработано компаниями AMD и Intel в конце 1990-х годов и с тех пор стало стандартом в индустрии.

Одним из наиболее важных изменений, внесённых в архитектуру x86-64, является переход от 32-битного адресного пространства к 64-битному. Это позволяет процессорам обрабатывать намного большие объемы данных и адресовать значительно большие объемы оперативной памяти. Вместо предыдущего ограничения в 4 гигабайта адресуемой памяти архитектура x86-64 может адресовать до 16 эксабайт оперативной памяти.

Важно отметить, что архитектура x86-64 сохраняет обратную совместимость с предыдущими версиями x86. Это означает, что современные процессоры, поддерживающие x86-64, могут выполнять как 64-битные, так и 32-битные приложения. Это обеспечивает плавный переход к новой архитектуре без необходимости полной переработки существующего программного обеспечения.

Помимо увеличения адресного пространства, архитектура x86-64 также вводит улучшения в работе с регистрами и командами. Например, внедрены новые команды и инструкции, оптимизированные для работы с 64-битными данными и адресами, что улучшает производительность и эффективность процессоров.

Архитектура x86-64 играет ключевую роль в современных вычислительных системах, обеспечивая мощные инструменты для разработки программного обеспечения и обеспечивая высокую производительность при обработке больших объемов данных. Её обратная совместимость с предыдущими версиями x86 делает её важным звеном в эволюции компьютерных технологий.

Дополнительно, архитектура x86-64 обеспечивает улучшенную безопасность и защиту данных. Некоторые из новых функций включают в себя поддержку аппаратного исполнения NX (No-eXecute), который предотвращает выполнение кода из областей памяти, предназначенных для хранения данных. Это существенно повышает защиту от атак, связанных с выполнением кода, таких как атаки на переполнение буфера.

Важным аспектом архитектуры x86-64 является её применимость в различных областях, от десктопных компьютеров и ноутбуков до высокопроизводительных серверов и крупных вычислительных систем. Она обеспечивает высокую производительность как для типичных пользовательских приложений, так и для сложных вычислительных задач, таких как машинное обучение, обработка графики, научные и инженерные расчёты, анализ данных и другие.

Благодаря широкой поддержке со стороны индустрии и разработчиков программного обеспечения, архитектура x86-64 стала основой для множества операционных систем и приложений, что обеспечивает её распространенность и популярность среди пользователей и разработчиков.

В совокупности, архитектура x86-64 представляет собой мощный и гибкий инструмент для создания современных вычислительных систем, обеспечивая высокую производительность, безопасность и универсальность для различных потребностей и задач.

Дополнительно, архитектура x86-64 вводит ряд других важных усовершенствований и возможностей. Одним из них является расширение набора регистров, что способствует улучшению производительности за счёт распределения более широкого диапазона данных и инструкций. Это позволяет более эффективно использовать ресурсы процессора и обеспечивает более гибкую обработку данных.

Ещё одним важным аспектом является поддержка расширенных возможностей векторизации, таких как SIMD (Single Instruction, Multiple Data), что позволяет процессору выполнять одну инструкцию над несколькими данными параллельно. Это особенно полезно для обработки мультимедийных данных, научных вычислений и других вычислительно интенсивных задач, где требуется параллельная обработка больших объемов данных.

Важным достижением архитектуры x86-64 является также внедрение аппаратной виртуализации, которая обеспечивает возможность запуска нескольких виртуальных машин на одном физическом сервере. Это повышает эффективность использования вычислительных ресурсов и обеспечивает изоляцию и безопасность между виртуальными средами.

Кроме того, архитектура x86-64 поддерживает расширенные средства шифрования и защиты данных, такие как аппаратное шифрование памяти и улучшенные алгоритмы шифрования. Это помогает защитить конфиденциальность и целостность данных в системе, особенно в условиях угроз кибербезопасности.

В совокупности, эти дополнительные возможности и усовершенствования делают архитектуру x86-64 мощным и гибким инструментом для создания современных вычислительных систем, способных эффективно обрабатывать различные типы задач и обеспечивать высокую производительность, безопасность и надежность.

## Режимы работы

 На данный момент х86 процессоры могут работать в четырёх режимах:

Real-Mode - реальный, системная поддержка в виде BIOS, Protected-Mode - защищённый, поддержка Win и никсы, Virtual-Mode - для эмуляции реального режима, из защищённого, System-Management-Mode – реальный из защищённого, без эмуляции.

В основном режиме работы, защищенном режиме, микропроцессор обладает рядом ключевых особенностей, среди которых важными являются виртуальное адресное пространство, защита памяти и многозадачность. В этом режиме программа оперирует виртуальными адресами, что позволяет создавать абстракцию между адресами в программе и физической памятью, что существенно упрощает управление памятью.

Виртуальное адресное пространство программы в защищенном режиме может быть значительно больше, чем фактически доступная физическая память и может достигать 64 терабайт. Для адресации виртуального адресного пространства используется сегментированная модель, где адрес состоит из селектора сегмента и смещения внутри сегмента. Каждый сегмент связан с дескриптором, который хранит информацию о нем.

Кроме того, в защищенном режиме используется страничная трансляция, что позволяет управлять памятью на уровне страниц. Это предоставляет удобные средства для реализации функций подкачки памяти и обеспечивает 36-битную физическую адресацию памяти (до 64 гигабайт) в процессорах P6+.

Встроенные средства переключения задач обеспечивают многозадачность в защищенном режиме, что позволяет микропроцессору быстро переключаться между различными средами выполнения, имитируя параллельную работу нескольких задач.

Для обеспечения защиты задач в защищенном режиме используются средства контроля пределов и типов сегментов, контроль привилегий, привилегированные инструкции и защита на уровне страниц. Это обеспечивает целостность сегментов кода и данных, а также ограничивает доступ программы к памяти в зависимости от ее текущих привилегий.[1]

В защищенном режиме микропроцессор поддерживает различные уровни привилегий (rings), которые разделяют операции с высоким и низким уровнем доступа. Наиболее часто используется четырёхуровневая система привилегий, где уровень 0 обладает наивысшими привилегиями и используется для выполнения ядра операционной системы, а уровни 1, 2 и 3 предназначены для менее привилегированных задач, таких как драйверы и пользовательские приложения. Это разграничение позволяет защитить ядро системы от некорректного доступа со стороны пользовательских программ, предотвращая потенциальные ошибки и угрозы безопасности.

Механизмы обработки исключений и прерываний в защищенном режиме также играют важную роль в обеспечении стабильной работы системы. Исключения используются для обработки ошибок, таких как деление на ноль или нарушение доступа к памяти, тогда как прерывания позволяют внешним устройствам (например, клавиатуре или сетевой карте) взаимодействовать с процессором.

# 2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 2.1 Преимущества использования Linux в качестве среды разработки системы Android

Использование Linux в качестве платформы разработки для создания игр открывает перед разработчиками множество преимуществ и возможностей. Одним из ключевых аспектов является философия открытого исходного кода, на которой основана Linux. Это обеспечивает разработчикам свободу выбора и гибкость в работе с различными компонентами операционной системы и инструментами разработки.

Стабильность и надежность Linux также играют важную роль. Операционная система Linux известна своей стабильностью даже при высокой нагрузке и долгом времени работы. Это делает ее привлекательным выбором для разработки крупных и сложных игровых проектов, где надежность работы и минимизация возможных сбоев критически важны.

Кроме того, Linux обладает эффективным управлением ресурсами. Это особенно важно при создании игр, требующих высокой производительности и оптимизации работы с графикой, звуком и другими ресурсами. Linux способен эффективно распределять ресурсы системы, обеспечивая стабильное функционирование игрового приложения даже на компьютерах с ограниченными ресурсами.

Важным аспектом является также широкий набор инструментов разработки программного обеспечения, доступных на Linux. Это включает в себя различные редакторы кода, компиляторы, отладчики и системы контроля версий. Редакторы кода, такие как Vim, Emacs, Sublime Text и другие, предоставляют разработчикам удобное и мощное средство для написания и редактирования кода на различных языках программирования. Компиляторы, такие как GCC, Clang, Intel C++ Compiler, обеспечивают возможность компиляции и сборки кода на различных языках программирования, что важно для создания высокопроизводительных игровых приложений.

Для отладки кода на Linux также доступны мощные инструменты, такие как GDB (GNU Debugger), Valgrind, LLDB и другие. Они помогают разработчикам искать и устранять ошибки, отлаживать код и профилировать производительность приложений. Системы контроля версий, такие как Git, SVN, Mercurial, позволяют эффективно управлять версиями источников, отслеживать изменения и участвовать в коллективной разработке проектов.

Наконец, важным аспектом являются библиотеки и фреймворки для игровой разработки, которые доступны на Linux. SDL (Simple DirectMedia Layer), OpenGL, Vulkan, Unity, Unreal Engine и другие библиотеки предоставляют разработчикам мощные инструменты для создания игр с поддержкой графики, звука, физики и других аспектов разработки игр. Используя Linux в качестве платформы разработки, разработчики могут воспользоваться всеми этими инструментами и возможностями для создания высококачественных, производительных и инновационных игр.

## 2.2 Архитектура ядра Linux

Ядро операционной системы (ОС) — это ключевая часть, которая связывает программы и аппаратные ресурсы компьютера. Оно переводит команды программ на понятный для железа язык и управляет доступом программ к ресурсам. Ядро также распределяет ресурсы между программами, обеспечивает их параллельную работу и управляет драйверами устройств, обеспечивая взаимодействие с аппаратурой.

Ядро может быть монолитным или микроядерным. Монолитное работает быстрее, так как весь его функционал встроен в один блок кода. Микроядерное более гибкое, устойчивое к сбоям, но менее производительное из-за переключения между модулями.

Ядро Linux играет важную роль в операционной системе, предоставляя основные функции, необходимые для работы с аппаратным обеспечением и управления ресурсами компьютера. Оно обеспечивает управление ресурсами, такими как процессорное время и память, распределение ввода/вывода, обеспечение доступа к системным вызовам для программ, и управление устройствами, такими как жесткие диски и сетевые интерфейсы.

Одной из особенностей ядра Linux является его модульность. Модульное ядро позволяет динамически загружать и выгружать драйверы устройств, что улучшает гибкость и производительность системы. Это также позволяет конфигурировать ядро под конкретные нужды, исключая ненужные драйверы или функции, что важно для создания легких и оптимизированных систем.

Кроме основных функций, современные версии ядра Linux поддерживают различные специализированные возможности. Например, Kernel Virtual Machine (KVM) обеспечивает виртуализацию, позволяя запускать виртуальные машины на основе Linux. Технология LXC позволяет работать с контейнерами для изоляции приложений, а расширенный фильтр пакетов Berkeley (eBPF) обеспечивает возможность запуска изолированных программ в ядре. На рисунке 2.1 показано обзор виртуальная машина Кернела.

Ядро Linux также активно развивается и поддерживает множество архитектур процессоров, что делает его универсальным решением для самых разных типов устройств — от серверов и рабочих станций до встраиваемых систем и мобильных устройств. Это достигается благодаря открытому исходному коду и широкой поддержке сообщества разработчиков. Постоянные обновления и патчи обеспечивают высокую безопасность и производительность ядра, а также внедрение новых технологий и улучшение существующих функций. Например, поддержка новейших стандартов сетевых протоколов и файловых систем обеспечивает Linux конкурентоспособность на рынке серверных решений.

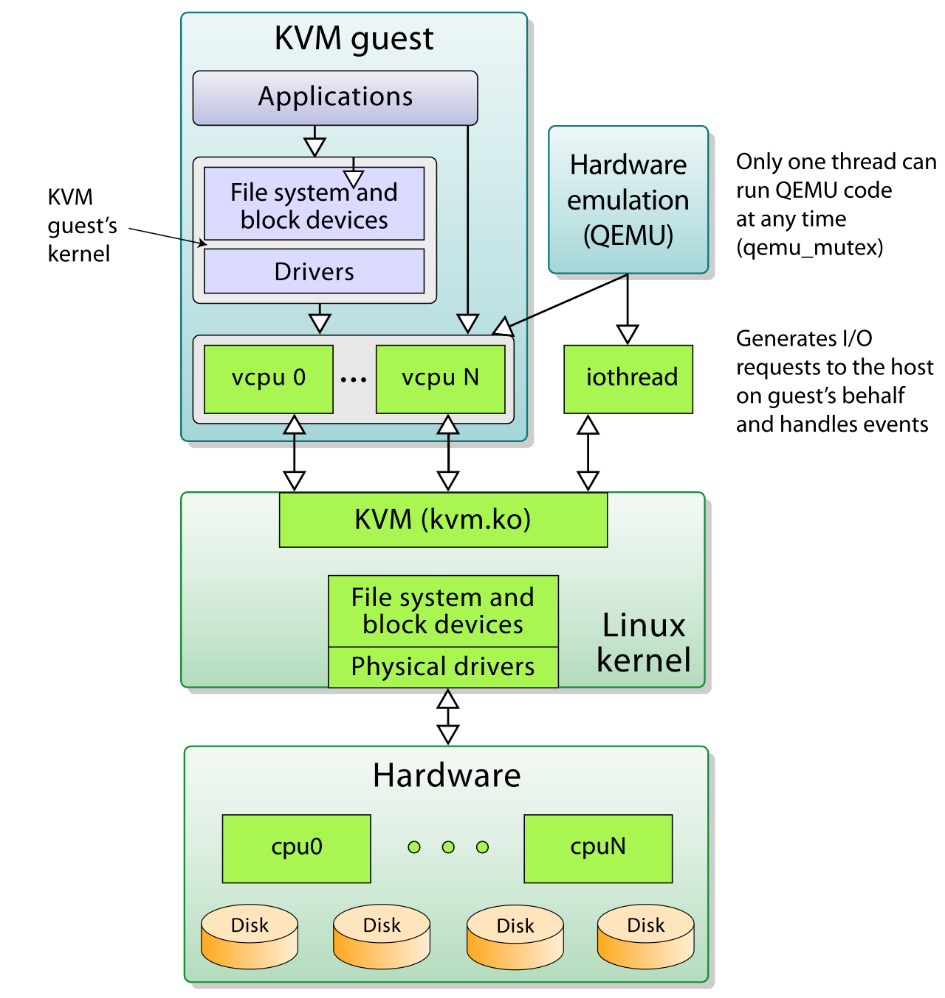


Рисунок 2.1 – Высокоуровневый обзор виртуализационной среды KVM

Ядро Linux также известно своей открытостью и поддержкой сообщества разработчиков. Исходный код ядра доступен для всех, что позволяет специалистам и энтузиастам изучать и улучшать систему. Благодаря активному сообществу разработчиков и пользователей, ядро Linux постоянно обновляется и улучшается. Это позволяет операционным системам, основанным на Linux, оставаться актуальными и безопасными в долгосрочной перспективе Эта открытость способствует постоянному развитию и совершенствованию ядра Linux, делая его одной из наиболее надежных и гибких платформ для создания современных операционных систем. На рисунке 2.2 показано применение ядра Linux.

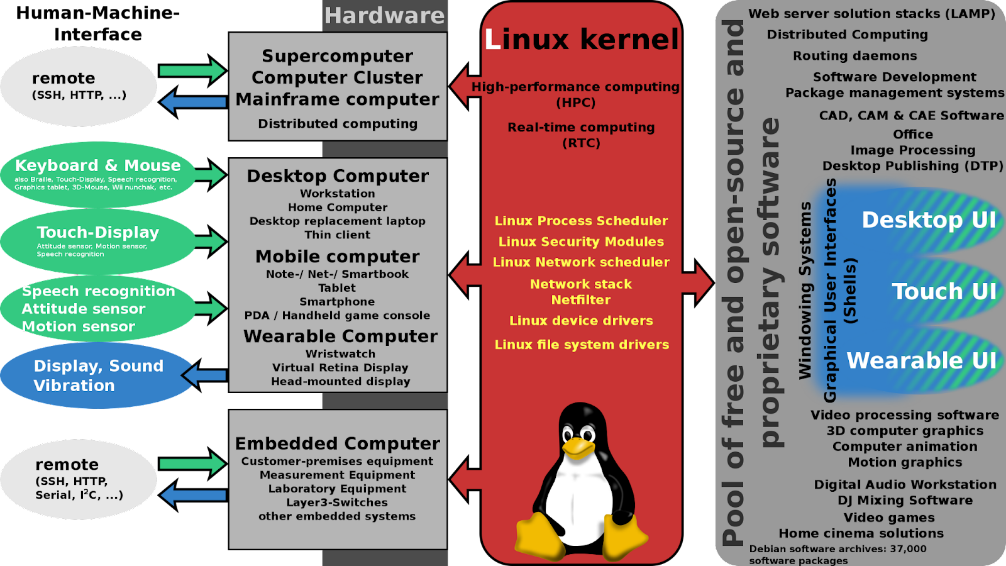


Рисунок 2.2 – Применение ядра Linux [2]

## 2.3 [Дистрибутивы Linux](https://docs.google.com/document/d/13imeuvY72PLe9hOHt5kNmMKn1-jsgvmCCshCM7Ao17c/edit#heading=h.t0a64nnzd4u3)

Существует множество дистрибутивов Linux, каждый из которых имеет свои особенности и призван удовлетворить разнообразные потребности пользователей.

Один из наиболее популярных дистрибутивов - Ubuntu, который славится своей простотой использования и удобством для новичков. Debian, с другой стороны, известен своей стабильностью и широким выбором пакетов. Fedora, поддерживаемый Red Hat, активно внедряет новые технологии и ориентирован на тех, кто следит за передовыми разработками.

openSUSE обеспечивает стабильность с превосходной поддержкой инструментов управления пакетами, таких как YaST. Arch Linux, напротив, уделяет внимание принципам минимализма и дает пользователям полный контроль над настройкой системы.

CentOS, хотя ранее был почти идентичен Red Hat Enterprise Linux (RHEL), теперь стал CentOS Stream, предлагающим более непрерывное обновление пакетов. Linux Mint, основанный на Ubuntu, ориентирован на привлекательный интерфейс и удобство работы для новичков.

Kali Linux - это специализированный дистрибутив для тестирования на проникновение и безопасности. Manjaro, основанный на Arch Linux, сочетает легкость использования и установки с преимуществами Arch.

Наконец, Gentoo известен своей высокой степенью настройки, поскольку пользователи компилируют большинство программ из исходных кодов.

Это лишь малая часть доступных дистрибутивов Linux, каждый из которых предназначен для различных целей и подходит разным категориям пользователей в зависимости от их потребностей и опыта. На рисунке 2.3 можно увидеть логотипы некоторых дистрибутивов Linux.



Рисунок 2.3 – [Дистрибутивы Linux](https://docs.google.com/document/d/13imeuvY72PLe9hOHt5kNmMKn1-jsgvmCCshCM7Ao17c/edit#heading=h.t0a64nnzd4u3)

Кроме того, существуют дистрибутивы, ориентированные на научные исследования и образование. Например, Scientific Linux разработан для научных учреждений и содержит множество предустановленных программ для научных вычислений и анализа данных. Edubuntu предназначен для образовательных учреждений и предлагает набор программного обеспечения для обучения и управления учебным процессом. Также стоит отметить дистрибутивы для мультимедийной работы, такие как Ubuntu Studio, который включает инструменты для обработки аудио, видео и графики. Таким образом, разнообразие дистрибутивов Linux позволяет пользователям выбрать именно тот вариант, который наилучшим образом соответствует их специфическим потребностям, будь то профессиональная деятельность, образование или хобби.

# 3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

Те́трис (производное от «[тетрамино](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%82%D1%80%D0%B0%D0%BC%D0%B8%D0%BD%D0%BE" \o "Тетрамино)» и «теннис»[[2]](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%82%D1%80%D0%B8%D1%81#cite_note-story-2)) — [компьютерная игра](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), первоначально изобретённая и разработанная советским программистом [Алексеем Пажитновым](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%B6%D0%B8%D1%82%D0%BD%D0%BE%D0%B2,_%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B5%D0%B9_%D0%9B%D0%B5%D0%BE%D0%BD%D0%B8%D0%B4%D0%BE%D0%B2%D0%B8%D1%87). Игра была выпущена 6 июня 1984 года — в это время Пажитнов работал в [Вычислительном центре](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%86%D0%B5%D0%BD%D1%82%D1%80_%D0%B8%D0%BC%D0%B5%D0%BD%D0%B8_%D0%90._%D0%90._%D0%94%D0%BE%D1%80%D0%BE%D0%B4%D0%BD%D0%B8%D1%86%D1%8B%D0%BD%D0%B0_%D0%A0%D0%90%D0%9D) [Академии наук СССР](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D0%B0%D0%B4%D0%B5%D0%BC%D0%B8%D1%8F_%D0%BD%D0%B0%D1%83%D0%BA_%D0%A1%D0%A1%D0%A1%D0%A0).

Тetris является классической и популярной игрой среди игроков всех возрастов и уровней опыта, что гарантирует интерес и востребованность такой версии игры на Linux. Технологические возможности Linux, такие как его открытая структура, разнообразие инструментов разработки и графических библиотек, облегчают создание качественной и интересной игры.

Поддержка открытых исходных кодов и свободных лицензий в Linux также способствует разработке Tetris, делая доступными необходимые ресурсы и инструменты без ограничений. Графические библиотеки и средства разработки в Linux, такие как SDL и OpenGL, позволяют создать привлекательную и функциональную графику для игры Tetris. Более того, Linux поддерживает множество аппаратных платформ, что расширяет аудиторию игроков и обеспечивает возможность наслаждаться Tetris на различных устройствах.

Таким образом, разработка Tetris для Linux обоснована популярностью игры, технологическими возможностями и поддержкой разнообразных инструментов и платформ, что делает ее привлекательной и перспективной для пользователей операционной системы Linux.

Те́трис (производное от «[тетрамино](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%82%D1%80%D0%B0%D0%BC%D0%B8%D0%BD%D0%BE" \o "Тетрамино)» и «теннис»[[2]](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%82%D1%80%D0%B8%D1%81#cite_note-story-2)) — [компьютерная игра](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), первоначально изобретённая и разработанная советским программистом [Алексеем Пажитновым](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%B6%D0%B8%D1%82%D0%BD%D0%BE%D0%B2,_%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B5%D0%B9_%D0%9B%D0%B5%D0%BE%D0%BD%D0%B8%D0%B4%D0%BE%D0%B2%D0%B8%D1%87). Игра была выпущена 6 июня 1984 года — в это время Пажитнов работал в [Вычислительном центре](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%86%D0%B5%D0%BD%D1%82%D1%80_%D0%B8%D0%BC%D0%B5%D0%BD%D0%B8_%D0%90._%D0%90._%D0%94%D0%BE%D1%80%D0%BE%D0%B4%D0%BD%D0%B8%D1%86%D1%8B%D0%BD%D0%B0_%D0%A0%D0%90%D0%9D) [Академии наук СССР](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D0%B0%D0%B4%D0%B5%D0%BC%D0%B8%D1%8F_%D0%BD%D0%B0%D1%83%D0%BA_%D0%A1%D0%A1%D0%A1%D0%A0).

Тetris является классической и популярной игрой среди игроков всех возрастов и уровней опыта, что гарантирует интерес и востребованность такой версии игры на Linux. Технологические возможности Linux, такие как его открытая структура, разнообразие инструментов разработки и графических библиотек, облегчают создание качественной и интересной игры.

Поддержка открытых исходных кодов и свободных лицензий в Linux также способствует разработке Tetris, делая доступными необходимые ресурсы и инструменты без ограничений. Графические библиотеки и средства разработки в Linux, такие как SDL и OpenGL, позволяют создать привлекательную и функциональную графику для игры Tetris. Более того, Linux поддерживает множество аппаратных платформ, что расширяет аудиторию игроков и обеспечивает возможность наслаждаться Tetris на различных устройствах.

Таким образом, разработка Tetris для Linux обоснована популярностью игры, технологическими возможностями и поддержкой разнообразных инструментов и платформ, что делает ее привлекательной и перспективной для пользователей операционной системы Linux.

## 3.1 Правила игры

Игра "Тетрис" — это классическая аркадная игра, в которой игроку нужно управлять падающими тетромино (геометрическими фигурами, состоящими из четырех квадратных блоков), чтобы создавать горизонтальные линии без пробелов. При создании линии она исчезает, а игрок получает очки. Цель игры — набрать как можно больше очков и не допустить заполнения игрового поля.

В "Тетрисе" есть семь различных тетромино: I-образное, J-образное, L-образное, O-образное, S-образное, T-образное и Z-образное. Каждая фигура состоит из четырех квадратных блоков и может быть повернута вокруг своей оси. Все вариации тетрамино можно рассмотреть на рисунке 3.1.

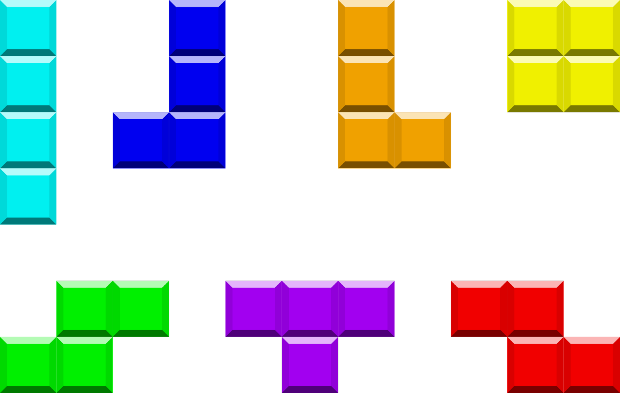


Рисунок 3.1 – Виды фигурок

Игровое поле "Тетриса" представляет собой вертикальный прямоугольник, обычно размером 10 клеток в ширину и 20 клеток в высоту. Тетромино появляются сверху игрового поля и начинают падать вниз.При появлении нового тетромино оно начинает падать сверху вниз по одной клетке за один шаг. Фигура падает до тех пор, пока не достигнет нижней границы игрового поля или другой фигуры.

Игрок управляет падением тетромино с помощью стрелок на клавиатуре или альтернативных клавиш (например, WASD). Стрелка вниз позволяет ускорить падение тетромино, а клавиша "пробел" — повернуть фигуру.

Когда вертикальная стопка тетромино достигает верхней границы игрового поля или новое тетромино приземляется, игрок может собрать горизонтальную линию, заполнив все клетки этой строки. Собранная линия исчезает, а все блоки выше опускаются вниз. Игрок зарабатывает очки за каждую собранную линию. Чем больше линий собрано одновременно, тем больше очков получает игрок. Если же новое тетромино не может появиться в игровом поле из-за заполненных клеток, игра завершается, и игрок проигрывает.

## 3.2 Технологии программирования, используемые для решения поставленных задач

 Проект был выполнен с использованием фреймворка Qt, который предназначен для разработки кроссплатформенного программного обеспечения на базе языка программирования C++.

   C++ позволяет эффективно управлять памятью, использовать объектно-ориентированный подход, создавать высокопроизводительные приложения и обеспечивать кроссплатформенную совместимость. C++ позволяет создавать многопоточные приложения, что способствует более эффективному использованию ресурсов и улучшает производительность. C++ имеет обширную стандартную библиотеку, включающую контейнеры, алгоритмы, ввод-вывод и многое другое. Это упрощает разработку и ускоряет процесс создания приложений. C++ позволяет программистам управлять памятью напрямую, что полезно для оптимизации и создания специализированных решений. Существует огромное сообщество разработчиков, которые активно обмениваются знаниями и опытом. Это обеспечивает поддержку и помощь при разработке на C++.C++ остается мощным и гибким языком программирования, который подходит для разнообразных задач и позволяет создавать эффективные и надежные приложения. Более того программы на C++ быстро обрабатываются и запускаются. Также C++ — компилируемый язык программирования. Это значит, что компилятор транслирует исходный код на C++ в исполняемый файл, который содержит набор машинных инструкций, что тоже повышает скорость.

Фреймворк Qt является мощным инструментом для разработки кроссплатформенных приложений с графическим интерфейсом. Он предоставляет обширный набор инструментов и библиотек для создания различных приложений, включая игры, утилиты, мультимедийные приложения и другие программы. Qt состоит из множества модулей, которые можно использовать по отдельности или вместе. Это позволяет разработчикам выбирать только те компоненты, которые им нужны. Кроме того, вы можете создавать собственные модули и расширять функциональность Qt.

Так как Qt кроссплатформенный, то он существует для всех популярных операционных систем: Windows, Linux, iOS и Android. Фреймворк используют при разработке под любые устройства: от микроконтроллеров до суперкомпьютеров. На рисунке  3.2 показаны некоторые преимущества QT.



Рисунок 3.2 – Положительные стороны Qt

Qt Creator позволяет писать код на языках C++, QML, JavaScript, Python и других. [В нем есть подсветка синтаксиса, автодополнение, рефакторинг и встроенная документация1](https://www.qt.io/product/development-tools). [Он интегрируется с популярными системами контроля версий, такими как Git, Subversion, Perforce и Mercurial1](https://www.qt.io/product/development-tools).Можно импортировать существующий проект или создать новый с нуля. Qt Creator генерирует все необходимые файлы. [Поддерживается использование инструментов сборки CMake и cross-compiling с помощью инструментов сборки CMake или qmake1](https://www.qt.io/product/development-tools). Более того можно собирать и запускать свое программное обеспечение на настольных, мобильных и встроенных операционных системах. [Настройки сборки позволяют легко переключаться между целями](https://www.qt.io/product/development-tools)[1](https://www.qt.io/product/development-tools).

 Также Qt Creator — среда, в которой легко разобраться. В ней есть все необходимое, важные компоненты находятся под рукой, а сам инструмент интуитивно понятен. Благодаря Qt Creator и его возможностям фреймворк подходит для создания приложений с упором на графический интерфейс.

# 4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

В программировании, алгоритм представляет собой последовательность шагов, определяющих способ решения определенной задачи. Это набор инструкций, которые выполняются в определенном порядке для достижения определенной цели. Алгоритмы используются для выполнения различных задач, таких как сортировка данных, поиск информации, решение математических задач и многое другое.

Хороший алгоритм должен быть эффективным, то есть выполнять задачу за разумное время и с минимальным использованием ресурсов, таких как память или процессорное время. Оценка эффективности алгоритма обычно осуществляется по его временной и пространственной сложности.

## 4.1 Алгоритм поиска в ширину

Поиск в ширину — это метод обхода или поиска в ширину в графе или дереве. Он начинается с заданного узла и исследует все соседние узлы на текущем уровне перед переходом к узлам следующего уровня. Это позволяет находить кратчайшие пути от начального узла до всех остальных узлов в графе.

Применение в Тетрисе: В игре Тетрис данный алгоритм может использоваться для определения, есть ли свободное место под падающим блоком. Алгоритм начинает исследование с верхней части поля, проверяя каждую клетку на наличие препятствий. Если путь до нижней части поля свободен, блок может падать. В данном проекте поиск в ширину используется для подсчета незаполненных рядов.

## 4.2 [Алгоритм поиска в глубину](https://www.baeldung.com/cs/knuth-morris-pratt" \t "_blank)

Алгоритм поиска (или обхода) в глубину (англ. depth-first search, DFS) позволяет построить обход ориентированного или неориентированного графа, при котором посещаются все вершины, доступные из начальной вершины. [9]

Применение в Тетрисе: В игре Тетрис DFS может использоваться для анализа возможных комбинаций фигур. Когда новая фигура появляется в игровом поле, вы можете использовать DFS для определения доступных мест, куда эта фигура может переместиться без столкновений с другими фигурами или краями поля. Это поможет компьютеру выбирать оптимальное местоположение для фигуры.

## 4.3 [Сортировка подсчетом](https://www.baeldung.com/cs/knuth-morris-pratt" \t "_blank)

Сортировка подсчётом — это алгоритм сортировки, предназначенный для сортировки чисел в ограниченном диапазоне. Он работает за линейное время, если диапазон чисел значительно меньше общего количества элементов. Основная идея алгоритма заключается в подсчёте числа вхождений каждого элемента в массив, а затем использовании этой информации для размещения элементов в отсортированном порядке.

Применение в Тетрисе: В игре Тетрис подсчёт используется для определения удаления заполненных строк. Когда игрок завершает строку, все её ячейки становятся заполненными. Чтобы удалить эту строку и сдвинуть верхние строки вниз, можно использовать подход, аналогичный сортировке подсчётом.

## 4.4 Алгоритм поворота фигуры

В Тетрисе алгоритм поворота фигуры отвечает за изменение ориентации тетромино (игровой фигуры) при поворотах на 90 градусов. Алгоритмы поворота могут различаться в зависимости от реализации, но основные методы включают использование матриц или предопределённых таблиц поворотов.

Каждое тетромино можно представить в виде матрицы, где 1 обозначает часть фигуры, а 0 — пустую клетку. Для поворота матрицы на 90 градусов по часовой стрелке используются транспонирование матрицы (замена строк на столбцы) и обращение каждой строки матрицы.

## 4.5 Алгоритм ближайшего соседа

Алгоритм ближайшего соседа в Тетрисе играет роль в определении оптимальной позиции для размещения следующего падающего блока. Он анализирует текущее состояние игрового поля и пытается выбрать наилучшее местоположение для блока, учитывая различные факторы, такие как высота столбцов, количество пустых пространств и возможность создания или заполнения линий.

В начале каждого хода алгоритм оценивает все доступные позиции для следующего блока. Он может рассматривать разные варианты расположения блока и вычислять оценку для каждой позиции.

После оценки всех возможных позиций алгоритм выбирает позицию с наилучшей оценкой и использует ее для размещения следующего блока на игровом поле. Это позволяет игре Тетрис принимать более информированные решения о том, как размещать блоки, что может повысить уровень сложности и интерес игры для игрока.

# 5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

Разработка программного продукта производилась в среде разработки Qt Creator с использованием [платформо-независим](http://doc.crossplatform.ru/qt/4.7.x/resources.html)ых библиотек и классов. Язык программирования – C++.

## 5.1 Структура проекта в программной среде

На рисунке 5.1 изображена структура проекта в программной среде Qt Creator.

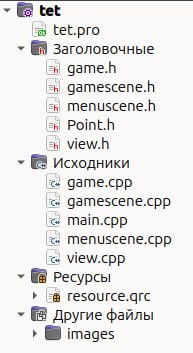


Рисунок 5.1 – Структура проекта в программной среде Qt Creator

Ресурсы, связанные с приложением, указываются в файле .qrc. Формат файла основан на XML, в котором перечисляются файлы на диске и опционально присваивает им имя ресурса, которое приложение должно использовать для доступа к ресурсу. [7]

Использования данного механизма, в том числе хранение текстур в формате .png, позволяет конечному пользователю изменить внешний вид текстур не повлияв на корректность работы проекта. Это возможно сделать путем замены исходного файла новым, который решил использовать пользователь, причем с таким же разрешением, как у исходного.

В game.qrc находятся все необходимые текстуры для корректной работы игрового проекта.

В game.cpp определена реализация методов класса Game, который управляет игровой логикой. В частности, здесь есть методы для управления фигурами, проверки их положения, подсчета очков и другие методы, относящиеся к игровому процессу.

Gamescene.cpp содержит реализацию методов класса GameScene, который управляет графическим представлением игры. Здесь реализованы методы отрисовки игрового поля, фигур, подсчета очков, а также методы для обработки событий клавиатуры и таймера, отвечающего за обновление игрового состояния.

В menuscene.cpp реализован класс MenuScene, который отвечает за отображение и управление главным меню игры. Здесь реализованы методы для отображения графических элементов меню, обработки событий мыши и перехода к другим сценам игры.

В view.cpp реализован класс View, который является виджетом для отображения сцен и управления переходами между ними. Здесь создаются сцены игры и меню, устанавливаются соединения сигналов и слотов для перехода между сценами.

Каждый из этих cpp-файлов выполняет определенную роль в структуре проекта и взаимодействует с другими компонентами (например, сценами, элементами интерфейса и игровой логикой), чтобы обеспечить полноценную работу игры.

## 5.2 Точка входа в приложение

Функция main является входной точкой в программном продукте и находится в файле main.cpp.

В данной функции определяется точка входа в графическую программу Qt с помощью создания объекта класса QApplication. После создания объекта QApplication, вызывается метод exec(), который запускает основной цикл событий Qt. Этот цикл событий ожидает действия пользователя и системные сообщения, а затем обрабатывает их. [8] После определяется объект класса MainWindow описанного выше.

## 5.3 Проверка работоспособности программного программного продукта

Для проверки корректности работоспособности программного продукта, он был запущена несколько раз. На рисунках 5.2, 5.3, 5.4 изображены запуски программного продукта.

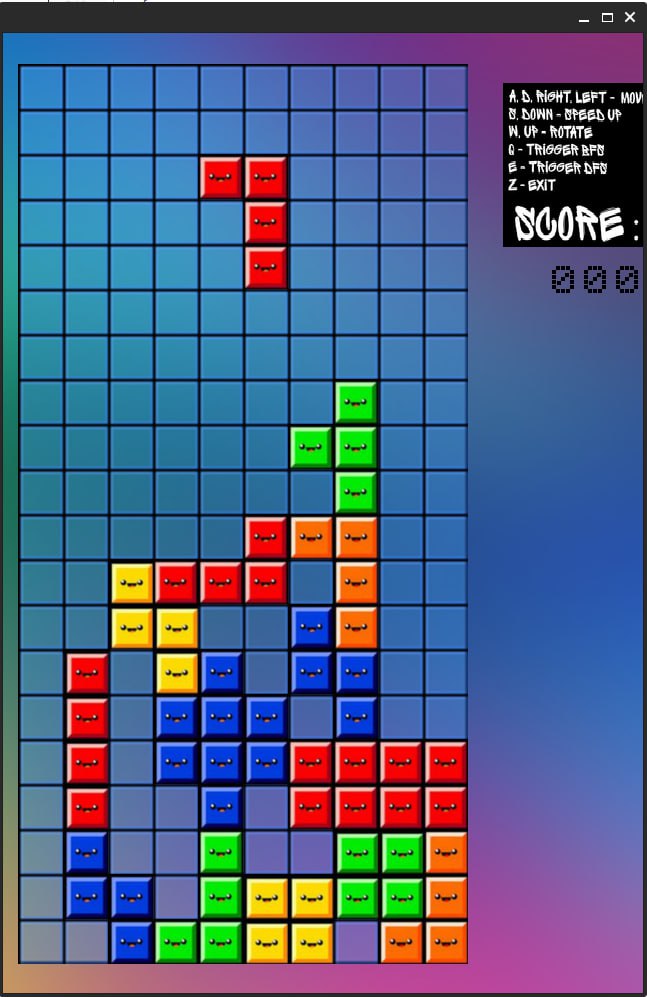


Рисунок 5.2 – Результат запуска программного продукта. Номер запуска: 1

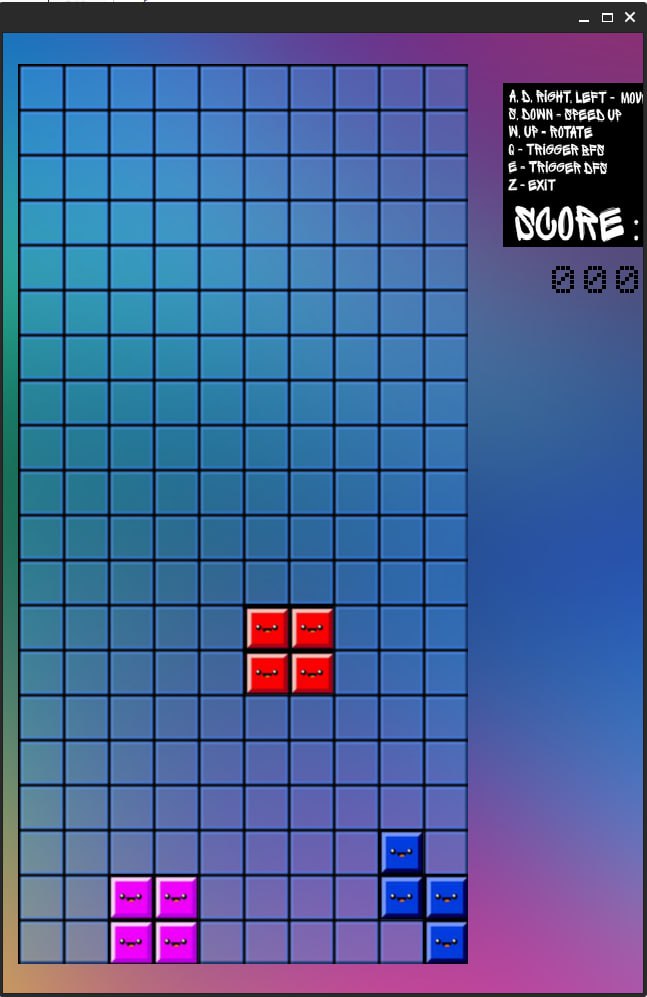


Рисунок 5.3 – Результат запуска программного продукта. Номер запуска: 2

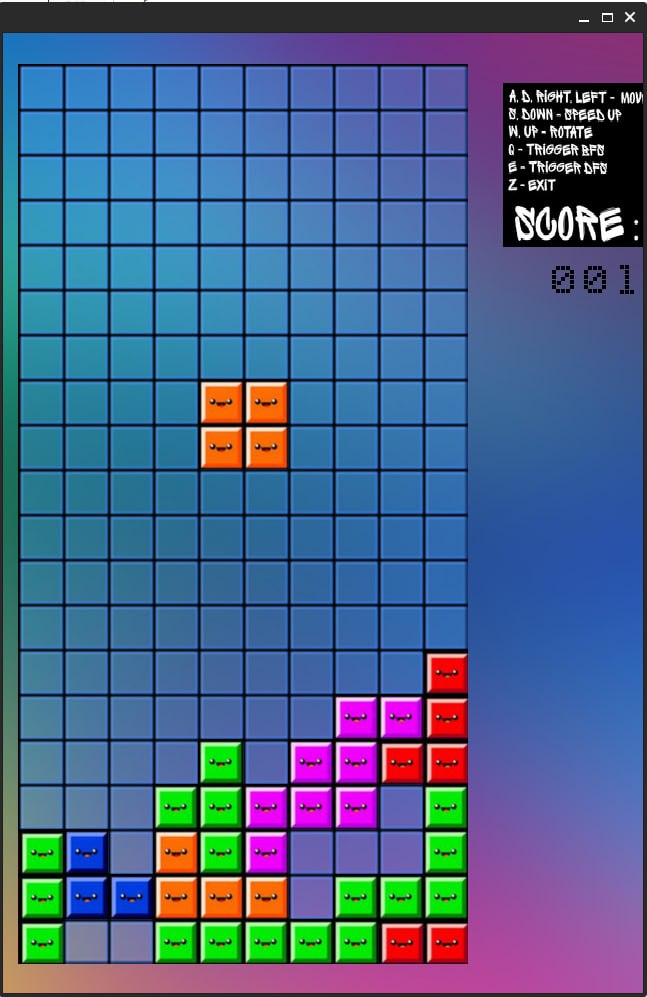


Рисунок 5.4 – Результат запуска программного продукта. Номер запуска: 3

Для проверки работоспособности программного продукта запустим его на физическом устройстве (ноутбук Lenovo Idea Pad 5, 16.0 GB RAM, AMD Ryzen 7 6800HS Creator Edition ).

В ходе нескольких запусков программного продукта, включающего игру Tetris, было проведено тщательное тестирование отображения всех графических элементов. Каждый графический элемент, будь то блок тетромино, фон или интерфейсные элементы, отображался корректно и без искажений. В процессе тестирования не было выявлено ни одного случая зависания или снижения производительности, что свидетельствует о стабильности и надёжности работы программы.

В результате проведённых тестов, ни в одном из запусков не было обнаружено ошибок или зависаний. Все команды игрока обрабатывались мгновенно и корректно, а тетромино перемещались и вращались в соответствии с ожиданиями. Визуальная составляющая игры также оставалась стабильной на протяжении всех тестов, без каких-либо артефактов или задержек в отображении.

Исходя из полученных результатов, можно сделать вывод о том, что программный продукт работает корректно. Программа демонстрирует устойчивую работу без сбоев, что обеспечивает положительный пользовательский опыт и позволяет рекомендовать её для дальнейшего использования и распространения.

# ЗАКЛЮЧЕНИЕ

В данной курсовой работе была разработана игра Тетрис с использованием различных алгоритмов: поиска в ширину (BFS), поиска в глубину (DFS), сортировки подсчётом и алгоритмов поворота фигур. Этот проект показал, как классические алгоритмы могут быть эффективно применены для решения задач, возникающих при создании интерактивных компьютерных игр.

Во-первых, применение поиска в ширину (BFS) для анализа и генерации возможных ходов и проверки состояний игрового поля позволило оптимизировать выбор наиболее выгодных действий на каждом этапе игры, что способствовало повышению уровня интеллектуальности алгоритма. Поиск в глубину (DFS) использовался для проверки достижимости различных состояний и анализа стратегий игры. Это обеспечило более глубокий анализ игровых ситуаций, что улучшило общий игровой процесс.

Сортировка подсчётом была успешно использована для управления игровым полем, особенно для определения и удаления заполненных строк. Благодаря своей линейной временной сложности, этот алгоритм позволил обрабатывать игровое поле с минимальными затратами ресурсов, обеспечивая высокую производительность. Алгоритмы поворота фигур на основе матричных преобразований также продемонстрировали свою эффективность. Они обеспечили корректное изменение ориентации тетромино при каждом повороте, что является ключевым аспектом геймплея Тетриса.

Результаты данной работы подтвердили, что эффективное использование классических алгоритмов может значительно улучшить игровую механику и оптимизировать производительность. Поиск в ширину и глубину, обычно используемые в задачах поиска и графах, показали свою применимость и полезность в контексте анализа игровых состояний. Сортировка подсчётом, хотя и редко используемая в реальном времени, оказалась полезной для специфических задач, таких как управление заполненными строками в Тетрисе.

Дальнейшее развитие проекта может включать улучшение алгоритмов искусственного интеллекта, например, с использованием машинного обучения для адаптации стратегий в реальном времени, расширение функциональности, введение многопользовательских режимов и оптимизацию производительности для обеспечения более плавного игрового процесса. Полученные результаты подтверждают эффективность выбранных методов и открывают возможности для дальнейших улучшений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] StudFiles [Электронный ресурс]. – Режим доступа: https://studfile.net/preview/7515844/page:32/.  –  Дата доступа: 23.03.2024.

[2] Stack Overflow [Электронный ресурс]. – Режим доступа: https://stackoverflow.com/questions/2886628/advantages-of-using-linux-as-primary-developer-desktop. – Дата доступа: 23.03.2024.

[3] Akamai [Электронный ресурс]. – Режим доступа: https://www.linode.com/docs/guides/why-use-linux-for-development/. – Дата доступа: 23.03.2024.

[4] TechTarget [Электронный ресурс]. – Режим доступа: https://www.techtarget.com/whatis/definition/x86-64. – Дата доступа: 23.03.2024.

[5] Metanit [Электронный ресурс]. – Режим доступа: https://metanit.com/assembler/tutorial/1.1.php. – Дата доступа: 23.03.2024.

[6] Хекслет [Электронный ресурс]. – Режим доступа: https://ru.hexlet.io/blog/posts/vidy-testirovaniya. – Дата доступа: 23.03.2024.

[7] Qt Documentation [Электронный ресурс]. – Режим доступа:  https://doc.qt.io/qt-5/resources.html. – Дата доступа: 29.05.2024.

[8] CSDN [Электронный ресурс]. – Режим доступа:  https://blog.csdn.net/weixin\_44084447/article/details/123205361. – Дата доступа: 30.05.2024.

[9] Foxford [Электронный ресурс]. – Режим доступа:  [Электронный ресурс]. – Режим доступа:  https://blog.csdn.net/weixin\_44084447/article/details/123205361. – Дата доступа: 30.05.2024. – Дата доступа: 30.05.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Исходный код программы

void GameScene::breadthFirstSearch()

{

std::vector<std::vector<bool>> visited(Game::BOARD\_HEIGHT, std::vector<bool>(Game::BOARD\_WIDTH, false));

std::queue<Point> bfsQueue;

Point startVertex(Game::BOARD\_WIDTH / 2, 0);

visited[startVertex.y][startVertex.x] = true;

bfsQueue.push(startVertex);

while (!bfsQueue.empty()) {

Point currentVertex = bfsQueue.front();

bfsQueue.pop();

for (int dy = -1; dy <= 1; ++dy) {

for (int dx = -1; dx <= 1; ++dx) {

if (dx == 0 && dy == 0) continue;

int nx = currentVertex.x + dx;

int ny = currentVertex.y + dy;

if (nx >= 0 && nx < Game::BOARD\_WIDTH && ny >= 0 && ny < Game::BOARD\_HEIGHT && !visited[ny][nx]) {

visited[ny][nx] = true;

bfsQueue.push(Point(nx, ny));

}

}

}

}

int emptyRowsCount = 0;

for (int y = 0; y < Game::BOARD\_HEIGHT; ++y) {

bool isEmpty = true;

for (int x = 0; x < Game::BOARD\_WIDTH; ++x) {

if (game.m\_field[y][x] != 0) {

isEmpty = false;

break;

}

}

if (isEmpty) {

++emptyRowsCount;

}

}

qDebug() << "Number of empty rows that can be filled: " << emptyRowsCount;

}

std::vector<Point> GameScene::depthFirstSearch(Point start, Point end)

{

std::vector<std::vector<bool>> visited(Game::BOARD\_HEIGHT, std::vector<bool>(Game::BOARD\_WIDTH, false));

std::vector<std::vector<Point>> parent(Game::BOARD\_HEIGHT, std::vector<Point>(Game::BOARD\_WIDTH, Point(-1, -1)));

std::stack<Point> dfsStack;

visited[start.y][start.x] = true;

dfsStack.push(start);

while (!dfsStack.empty()) {

Point currentVertex = dfsStack.top();

dfsStack.pop();

if (currentVertex == end) {

std::vector<Point> path;

for (Point at = end; at.x != -1 && at.y != -1; at = parent[at.y][at.x]) {

path.push\_back(at);

}

std::reverse(path.begin(), path.end());

return path;

}

std::vector<Point> directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

for (const auto& dir : directions) {

int nx = currentVertex.x + dir.x;

int ny = currentVertex.y + dir.y;

if (nx >= 0 && nx < Game::BOARD\_WIDTH && ny >= 0 && ny < Game::BOARD\_HEIGHT &&

!visited[ny][nx] && game.m\_field[ny][nx] == 0) {

visited[ny][nx] = true;

parent[ny][nx] = currentVertex;

dfsStack.push(Point(nx, ny));

}

}

}

return {};

}

GameScene::GameScene(QObject \*parent) : QGraphicsScene(parent), game(), timePerFrame(1000.0f/60.0f)

{

setSceneRect(0,0, Game::RESOLUTION.width(), Game::RESOLUTION.height());

srand(time(0));

m\_frame = new QGraphicsPixmapItem(game.m\_frame);

m\_background = new QGraphicsPixmapItem(game.m\_background);

m\_tiles = new QGraphicsPixmapItem(game.m\_tile);

timer = new QTimer(this);

connect(timer, &QTimer::timeout, this, &GameScene::update);

}

void GameScene::start()

{

game.reset();

timer->start(timePerFrame);

}

void GameScene::stop()

{

timer->stop();

}

void GameScene::renderScene()

{

QString fileName = QDir::currentPath() + QDir::separator() + "game\_scene.png";

QRect rect = sceneRect().toAlignedRect();

QImage image(rect.size(), QImage::Format\_ARGB32);

image.fill(Qt::transparent);

QPainter painter(&image);

render(&painter);

image.save(fileName);

qDebug() << "saved " << fileName;

}

void GameScene::keyPressEvent(QKeyEvent \*event)

{

if (!event->isAutoRepeat())

{

switch(event->key())

{

case Qt::Key\_A:

case Qt::Key\_Left:

if (game.m\_state == Game::State::Active)

{

game.m\_dx = -1;

}

break;

case Qt::Key\_D:

case Qt::Key\_Right:

if (game.m\_state == Game::State::Active)

{

game.m\_dx = 1;

}

break;

case Qt::Key\_W:

case Qt::Key\_Up:

if (game.m\_state == Game::State::Active)

{

game.m\_rotate = true;

}

break;

case Qt::Key\_S:

case Qt::Key\_Down:

if (game.m\_state == Game::State::Active)

{

game.m\_delay = Game::SPEED\_UP;

}

break;

case Qt::Key\_R:

if (game.m\_gameOver)

{

game.reset();

}

break;

case Qt::Key\_Q:// Trigger BFS

if (game.m\_state == Game::State::Active)

{

breadthFirstSearch();

}

break;

case Qt::Key\_E:

{

Point startVertex(Game::BOARD\_WIDTH / 2, 0);

Point endVertex(Game::BOARD\_WIDTH / 2, Game::BOARD\_HEIGHT - 1);

std::vector<Point> path = depthFirstSearch(startVertex, endVertex);

if (!path.empty()) {

qDebug() << "Path found:";

for (const Point& p : path) {

qDebug() << "(" << p.x << "," << p.y << ")";

}

} else {

qDebug() << "No path found";

}

break;

}

default:

break;

}

}

QGraphicsScene::keyPressEvent(event);

}

void GameScene::keyReleaseEvent(QKeyEvent \*event)

{

if (!event->isAutoRepeat())

{

switch(event->key())

{

case Qt::Key\_A:

case Qt::Key\_D:

case Qt::Key\_Right:

case Qt::Key\_Left:

game.m\_dx = 0;

break;

case Qt::Key\_W:

case Qt::Key\_Up:

game.m\_rotate = false;

break;

case Qt::Key\_S:

case Qt::Key\_Down:

game.m\_delay = Game::SPEED;

break;

default:

break;

}

}

QGraphicsScene::keyReleaseEvent(event);

}

void GameScene::drawScore()

{

QGraphicsPixmapItem\* scorePixmapItem = new QGraphicsPixmapItem(game.m\_scorePixmap.scaled(150,164));

addItem(scorePixmapItem);

scorePixmapItem->moveBy(500,50);

QString scoreText = QString::number(game.m\_score);

int unityPartVal = 0;

int decimalPartValue = 0;

int hendredthPartValue = 0;

if(scoreText.length() == 1) // 0 - 9

{

unityPartVal = scoreText.toInt();

decimalPartValue = 0;

hendredthPartValue = 0;

}

else if(scoreText.length() == 2) // 10 - 99

{

unityPartVal = scoreText.last(1).toInt();

decimalPartValue = scoreText.first(1).toInt();

hendredthPartValue = 0;

}

else if(scoreText.length() == 3) // 100 - 999

{

unityPartVal = scoreText.last(1).toInt();

hendredthPartValue = scoreText.first(1).toInt();

QString copyVal = scoreText;

copyVal.chop(1);

decimalPartValue = copyVal.last(1).toInt();

}

QGraphicsPixmapItem\* unityPartScoreItem = new QGraphicsPixmapItem(game.m\_numbersPixmap.copy(unityPartVal\*32, 0, 32, 32));

unityPartScoreItem->moveBy(Game::RESOLUTION.width()-32, scorePixmapItem->y()+180);

addItem(unityPartScoreItem);

QGraphicsPixmapItem\* decimalPartScoreItem = new QGraphicsPixmapItem(game.m\_numbersPixmap.copy(decimalPartValue\*32, 0, 32, 32));

decimalPartScoreItem->moveBy(Game::RESOLUTION.width()-2\*32, scorePixmapItem->y()+180);

addItem(decimalPartScoreItem);

QGraphicsPixmapItem\* hundrethPartScoreItem = new QGraphicsPixmapItem(game.m\_numbersPixmap.copy(hendredthPartValue\*32, 0, 32, 32));

hundrethPartScoreItem->moveBy(Game::RESOLUTION.width()-3\*32, scorePixmapItem->y()+180);

addItem(hundrethPartScoreItem);

}

void GameScene::drawGameState()

{

if (game.m\_gameOver)

{

game.m\_state = Game::State::Game\_Over;

QGraphicsPixmapItem\* item = new QGraphicsPixmapItem(game.m\_gameOverBackground.scaled(640, 960));

addItem(item);

QGraphicsPixmapItem\* restartTextItem = new QGraphicsPixmapItem(game.m\_restartTextPixmap);

addItem(restartTextItem);

restartTextItem->setPos(180, 570);

}

}

void GameScene::drawActiveFigure()

{

for (int i = 0; i < Game::COUNT\_OF\_BLOCKS; i++)

{

QGraphicsPixmapItem\* pixmapItem = new QGraphicsPixmapItem(game.m\_tile.copy(game.m\_colorNum \* Game::BLOCK\_SIZE.width(), 0,

Game::BLOCK\_SIZE.width(), Game::BLOCK\_SIZE.height()));

addItem(pixmapItem);

pixmapItem->setPos(game.m\_a[i].x \* Game::BLOCK\_SIZE.width(), game.m\_a[i].y \* Game::BLOCK\_SIZE.height());

pixmapItem->moveBy(m\_frame->pos().x(), m\_frame->pos().y());

}

}

void GameScene::moveFigure()

{

for (int i = 0; i < Game::COUNT\_OF\_BLOCKS ;i++)

{

game.m\_b[i]= game.m\_a[i];

game.m\_a[i].x += game.m\_dx;

}

if (!game.check())

{

for (int i = 0; i < Game::COUNT\_OF\_BLOCKS; i++)

{

game.m\_a[i] = game.m\_b[i];

}

}

}

void GameScene::rotateFigure()

{

if (game.m\_rotate)

{

Point p = game.m\_a[1];

for (int i = 0; i < Game::COUNT\_OF\_BLOCKS; i++)

{

int x = game.m\_a[i].y-p.y;

int y = game.m\_a[i].x-p.x;

game.m\_a[i].x = p.x - x;

game.m\_a[i].y = p.y + y;

}

if (!game.check())

{

for (int i = 0; i < Game::COUNT\_OF\_BLOCKS; i++)

{

game.m\_a[i] = game.m\_b[i];

}

}

game.m\_rotate = false;

}

}

void GameScene::update()

{

if(game.m\_state == Game::State::Active)

{

game.m\_timer += (timePerFrame);

}

moveFigure();

rotateFigure();

if ( game.m\_timer > game.m\_delay)

{

for (int i = 0; i < Game::COUNT\_OF\_BLOCKS; i++)

{

game.m\_b[i] = game.m\_a[i];

game.m\_a[i].y+=1;

}

if (!game.check())

{

for (int i = 0; i < Game::COUNT\_OF\_BLOCKS; i++)

{

game.m\_field[game.m\_b[i].y][game.m\_b[i].x] = game.m\_colorNum;

}

game.m\_colorNum = (rand() % (Game::COUNT\_OF\_COLORS - 1)) + 1;

int n= rand() % Game::COUNT\_OF\_FIGURES;

for (int i = 0; i < Game::COUNT\_OF\_BLOCKS; i++)

{

game.m\_a[i].x = (game.m\_figures[n][i] % 2) + game.BOARD\_WIDTH/2-1;

game.m\_a[i].y = game.m\_figures[n][i] / 2;

if(game.m\_field[game.m\_a[i].y][game.m\_a[i].x])

{

qDebug() << "Game Over";

game.m\_gameOver = true;

}

}

}

game.m\_timer=0;

}

int k = game.BOARD\_HEIGHT-1;

for (int i = game.BOARD\_HEIGHT-1; i > 0; i--)

{

int count = 0;

for (int j = 0; j < game.BOARD\_WIDTH;j++)

{

if (game.m\_field[i][j])

{

count++;

}

game.m\_field[k][j]= game.m\_field[i][j];

}

if (count < game.BOARD\_WIDTH)

{

k--;

}

else

{

game.addScore(1);

}

}

game.m\_dx=0; game.m\_rotate=false;

clear();

drawBackground();

for (int i=0;i<game.BOARD\_HEIGHT;i++)

{

for (int j=0;j<game.BOARD\_WIDTH;j++)

{

if (game.m\_field[i][j]==0) continue;

QGraphicsPixmapItem\* pixmapItem = new QGraphicsPixmapItem(game.m\_tile.copy(game.m\_field[i][j]\*45, 0, 45, 45));

addItem(pixmapItem);

pixmapItem->setPos(j \* Game::BLOCK\_SIZE.width(), i \* Game::BLOCK\_SIZE.height());

pixmapItem->moveBy(m\_frame->pos().x(), m\_frame->pos().y());

}

}

drawActiveFigure();

drawScore();

drawGameState();

}

void GameScene::countingSortByEmptyCells(std::vector<int>& arr)

{

int maxElement = \*std::max\_element(arr.begin(), arr.end());

std::vector<int> count(maxElement + 1, 0);

std::vector<int> output(arr.size());

for (size\_t i = 0; i < arr.size(); i++) {

count[arr[i]]++;

}

for (int i = 1; i <= maxElement; i++) {

count[i] += count[i - 1];

}

for (int i = arr.size() - 1; i >= 0; i--) {

output[count[arr[i]] - 1] = arr[i];

count[arr[i]]--;

}

for (size\_t i = 0; i < arr.size(); i++) {

arr[i] = output[i];

}

}

# ПРИЛОЖЕНИЕ Б (обязательное) Функциональная схема программного продукта

# ПРИЛОЖЕНИЕ В (обязательное) Блок-схема алгоритма поиска в ширину

# ПРИЛОЖЕНИЕ Г (обязательное) Блок-схема алгоритма поиска в глубину

# ПРИЛОЖЕНИЕ Д (обязательное) Блок-схема сортировки подсчетом

# ПРИЛОЖЕНИЕ Е (обязательное) Блок-схема алгоритма поворота фигуры

# ПРИЛОЖЕНИЕ Ж (обязательное) Блок-схема алгоритма ближайшего соседа

# ПРИЛОЖЕНИЕ И (обязательное) Графический интерфейс пользователя

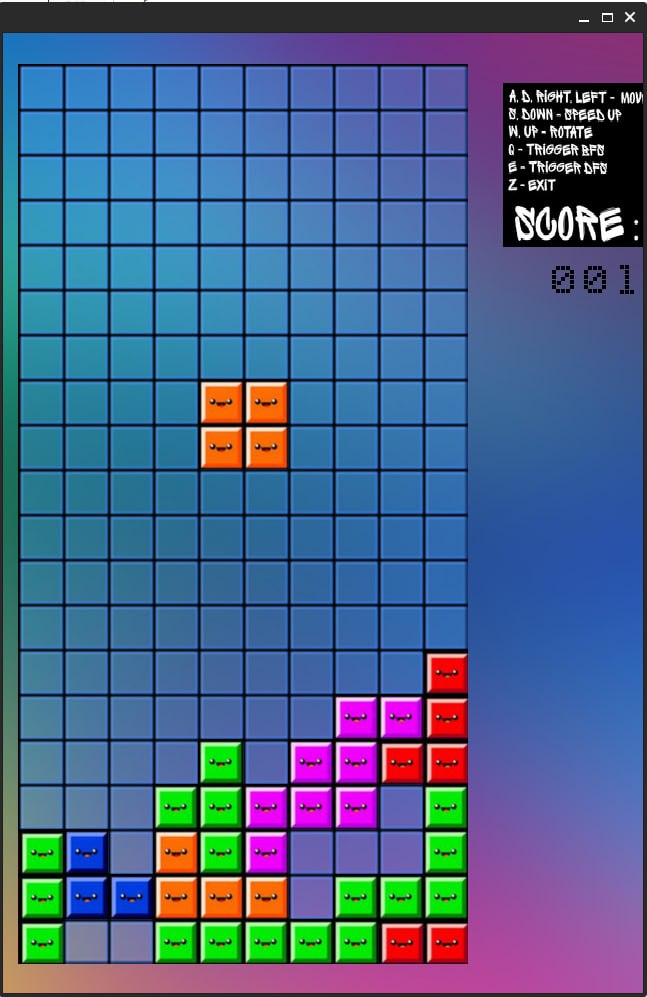


Рисунок И.1



Рисунок И.2