СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерные игры являются неотъемлемой частью культуры современности. Одни обучают логике и пространственному мышлению, другие развивают скорость реакции и моторику, но даже создание видеоигр позволяет развивать критическое мышление, углублять знания математики и совершенствовать навыки программирования.

Целью данной курсовой работы является создание игры «Тетрис» с использованием современных технологий. В рамках проекта будет изучена работа таких инструментов и технологий, как Qt Creator, язык программирования C++ и его особенностей, принципы объектно-ориентированного программирования, а также соответствующий фреймворк для C++ – Qt.

Выбор данной темы обусловлен тем, что, несмотря на время, пройденное с момента создания оригинального «Тетриса» в 1984 году, он до сих остается одной из самых продаваемых и популярных игр в истории, а также позволяет разработчику разобраться в основах создания игр и является отличным вариантом для первого проекта на Qt.

Задачи работы:

1. Изучение существующих игр, созданных на подобие «Тетриса»
2. Составление требований к игре
3. Реализация игры в соответствии с требованиями
4. Проведение тестирования игры
5. Составление отчета по работе
6. Презентация отчета и защита его.

Объектом данного исследования является Разработка игр на Qt.

Предметом исследования в данной работе является Разработка «Тетриса» для ПК на языке C++.

В качестве основных методов исследования применены анализ, синтез, сравнение и моделирование. Практическая реализация поставленной задачи соответствует основным парадигмам ООП, применяемым для разработки игр.

Информационной базой исследования являются открытые источники, а также материалы курса «Технологии индустриального программирования», доступные через систему дистанционного обучения РТУ МИРЭА.

В данном отчете будет представлен процесс разработки программного продукта, технологическое проектирование и описание системы. а также непосредственно результаты разработки.

1. ТЕОРИТИЧЕСКАЯ ЧАСТЬ

1.1 Описание программируемой системы

Разрабатываемая система представляет собой настольное приложение — игру, основанную на оригинальном «Тетрисе» 1984 года. Приложение реализуется на C++ с использованием фреймворка Qt. Программа разделена на два основных модуля, каждый из которых является объектом QGraphicsScene. Модули связаны между собой с помощью QStackedWidget.

Первый модуль — это начальное меню. Оно представляет собой простую «сцену» со статическим фоном и кнопкой QPushButton для перехода непосредственно к игровому процессу.

После нажатия кнопки происходит смена сцены через QStackedWidget, и создаётся объект класса QGraphicsScene, в котором размещаются элементы игрового интерфейса: QLabel для отображения очков и уровня, а также создаются первая активная фигура и следующая по очереди фигура.

В этом же классе реализована основная логика игры: проверка на наличие заполненных линий, их удаление и сдвиг, начисление очков, повышение уровня, а также проверка условия завершения игры. После приземления фигуры она разделяется на отдельные блоки для упрощения последующей обработки заполненных линий.

Игровые фигуры создаются в отдельном классе Tetrimino. С помощью QGraphicsItemGroup несколько блоков объединяются в одну фигуру, что упрощает её перемещение и вращение. В этом же классе обрабатывается ввод с клавиатуры: перемещение фигуры, её вращение и проверка столкновений с другими блоками или границами игрового поля.

Отдельные блоки реализованы в классе Block. Это сделано для облегчения расширения функциональности блока в будущем.

1.2 Обзор существующих решений

Термином компьютерная игра обозначается компьютерная программа, которая служит для организации игрового процесса, связи с партнёром по игре, или сама выступающая в качестве партнёра [1].

Компьютерные игры классифицируют по нескольким основным признакам:

- жанр;
- количество игроков;
- визуальное представление;
- платформа.

Жанр игры определяется целью и основной механикой игры.

По количеству игроков игры разделяются на два вида:

- однопользовательские;
- многопользовательские.

По визуальному представлению компьютерные игры можно разделить на следующие виды:

- текстовые – минимальное графическое представление, общение с игроком проходит с помощью текста;
- 2D – все элементы отрисованы в виде двумерной графики (спрайтов);
- 3D – все элементы отрисованы в виде трехмерной графики (3D-модели).

По типу платформы:

- персональные компьютеры;
- игровые приставки/консоли;
- мобильные телефоны.

Общий алгоритм разработки компьютерной игры включает в себя 3 больших этапах [2]:

1. Проектирование.
2. Разработка.
3. Издание и поддержка.

На этапе проектирования определяются цель игры и средства ее разработки. При определении цели выделяются идея, жанр и сеттинг игры. Идея – это то, что будет побуждать игрока играть в создаваемую игру, и она очень тесно связана с жанром. Определив основные идеи игры, жанр будет подобран практически сразу. Определившись с жанром и идеей игры, следующим шагом будет выбор сеттинга. Сеттинг – это среда, в которой будет происходить основное действие игры. Он определяет место, время и условия действия. К средствам разработки в первую очередь относят программный код и игровой движок. От их грамотного выбора зависит как скорость самой разработки, так и работоспособность самого продукта в дальнейшем. Программный код в первую очередь зависит от платформы, для которой будет создаваться компьютерная игра.

Игровой движок отвечает за низкоуровневое описание физики объектов, правил рендеринга графики и другие. При выборе игрового движка первым делом смотрят на его доступность и уже сделанный выбор языка программирования.

После выбора цели игры и средств разработки, начинается второй этап реализации проекта – разработка. Разработка самый крупный и долгий этап реализации проекта, он включает в себя большое количество шагов, без которых невозможно создать рабочий продукт.

Игровая механика основывается на цели игры, она определяет все объекты и правила, по которым игрок будет взаимодействовать с ними. Обычно параллельно с разработкой игровой механики идет написание сюжета игры. Сюжет играет немаловажную роль, он определяет то, насколько будет игроку интересно играть в вашу игру. Сюжет представляют в двух вариантах: литературный и режиссерский сценарии. Литературный сценарий описывает основные события и персонажей игры, которые участвуют в игре. Режиссерский же представляет собой подробное описание уровней игры, событий, которые на этих уровнях происходят. Также на данном этапе начинается ранняя проработка графической составляющей и дизайна игры. На основе сюжета и заранее оговоренного дизайна, создаются ранние концепт-арты, на основе которых впоследствии будет проработан основной вид игры и персонажей. После

разработки сюжета и игровой механики начинается самая важная часть – разработка самой игры.

При разработке игровых уровней изначально создается его упрощенный план, на котором схематично изображен сам уровень, а также изображены предметы, с которыми будет впоследствии взаимодействовать игрок. Следом после этого создается первая версия уровня.

Вскоре после создания первых уровней идет сборка первого прототипа игры, который называют альфа-версией игры. Она необходима для того, чтобы разработчик мог провести тестирование (альфа-тестирование) основной механики игры, и проверить насколько она соответствует заявленным требованиям. Если альфа-версия игры успешно проходит тестирование, наступает следующий этап разработки – проработка механики и объектов игры. На данном этапе идет доработка уровней и механики игры, и начинают добавлять первые сюжетные события в игру, такие как видеоролики, сюжетные диалоги и кат-сцены. Так же исправляются первые ошибки и неисправности в коде игры, которые были обнаружены при тестировании альфа-версии игры. После этого наступает этап создания второго прототипа игры, или, как принято говорить, бета-версии. Бета-версия служит для того, что протестировать игру на неисправности, фактически бета-версия представляет собой практически готовую игру. В ней могут отсутствовать какие-нибудь незначительные элементы, которые не влияют на геймплей игры.

Если игра проходит бета-тестирование, она отправляется на окончательную доработку и исправление критических ошибок, после чего идет сборка финальной версии игры и следом наступает релиз игры. После релиза игры последующая её поддержка. Поддержка заключается в выпуске патчей (файлов исправлений ошибок в готовом продукте) [3].

Тетрис — культовая игра-головоломка, породившая множество вариаций с уникальными механиками и особенностями. Ниже представлены восемь разнообразных версий «Тетриса», каждая из которых предлагает уникальные особенности, отличающие их от классической игры:

1. **Setris[4]**: В этой версии блоки, касаясь земли, превращаются в песок, формируя кучи, которые скользят с холмов и заполняют трещины. Это позволяет игроку манипулировать ландшафтом, даже если не удалось сразу собрать нижние ряды. Наблюдение за тем, как игровое поле постепенно заполняется пиксельным песком, добавляет особое удовольствие (Рисунок 1.2.1).

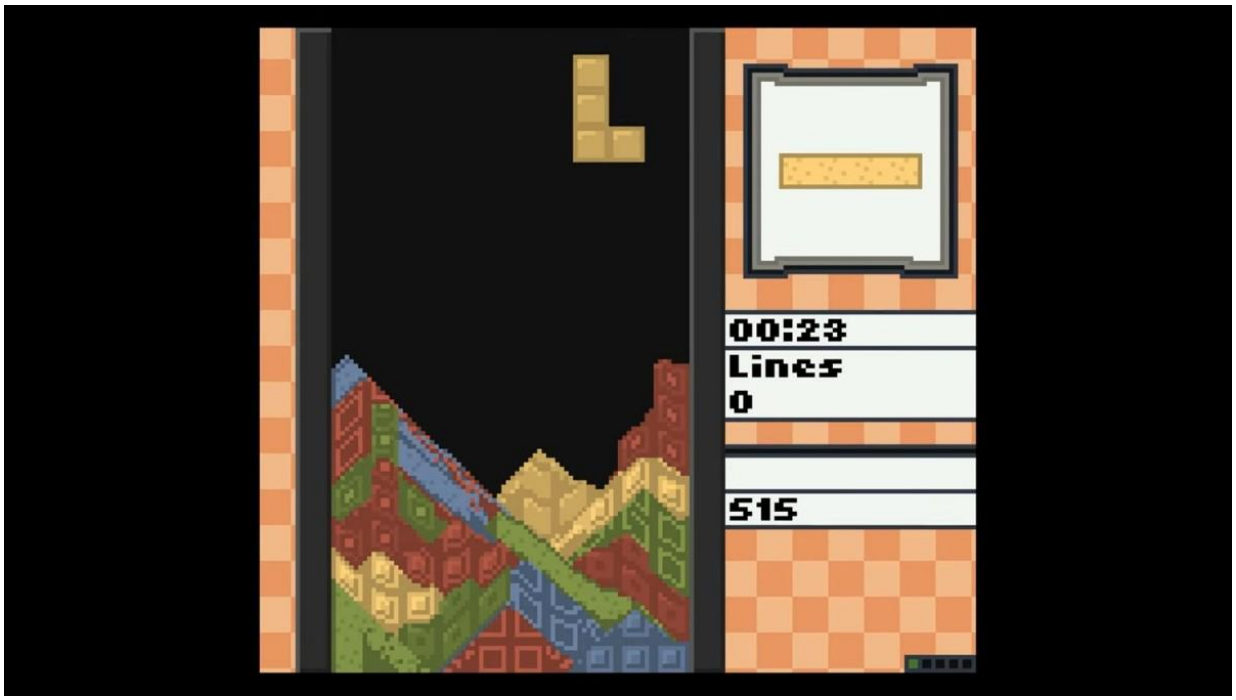

Рисунок 1.2.1 - Setris

2. **Tricky Towers[5]**: Эта игра-головоломка основана на физике и предлагает соревновательный мультиплеерный режим. Игроки управляют магами, строящими башни из фигурных блоков. Задача — построить самую высокую и устойчивую башню, учитывая физические свойства блоков ииспользуя заклинания для помощи себе или помехи соперникам (Рисунок1.2.2).

Рисунок 1.2.2 - Tricky Towers

3. **Tetris Time Warp[6]**: Эта версия представляет собой классический «Тетрис» с уникальной механикой: некоторые фигурки при стирании линий переносят игрока в другую эпоху. Это позволяет на несколько секунд оказаться в оригинальном «Тетрисе» 1984 года, в чёрно-белой портативке 1989 года или в вариации Bombliss, добавляя элемент неожиданности и разнообразия в геймплей (Рисунок 1.2.3).

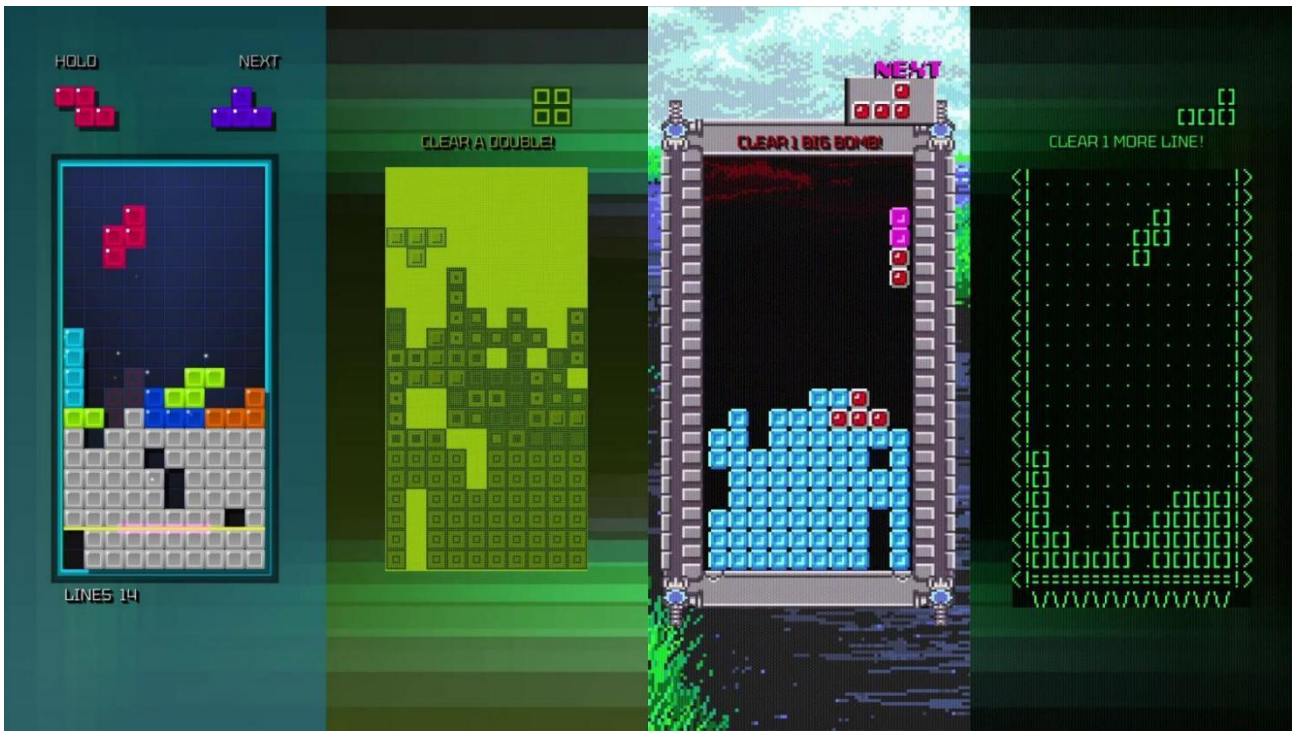

Рисунок 1.2.3 - Tetris Time Warp

4. **Cozy Blocks[7]: Undersea Adventure**: Расслабляющая трёхмерная головоломка с механикой размещения блоков. Игроки выбирают нужные части, помещают их в башню и решают головоломки, помогая персонажу Кейт Немо и морским существам выполнить сложные миссии в подводном мире. Игра сочетает знакомые элементы «Тетриса» с новыми трёхмерными вызовами (Рисунок 1.2.4).

Рисунок 1.2.4 - Cozy Blocks: Undersea Adventure

5. Tetris Effect: Connected[8]: Современная версия «Тетриса», включающая новый геймплейный элемент Zone, позволяющий игрокам останавливать время и падение фигур, чтобы выйти из сложных ситуаций или очистить дополнительные линии для получения бонусных наград. Игра сочетает классический геймплей с впечатляющими визуальными и звуковыми эффектами (Рисунок 1.2.5).

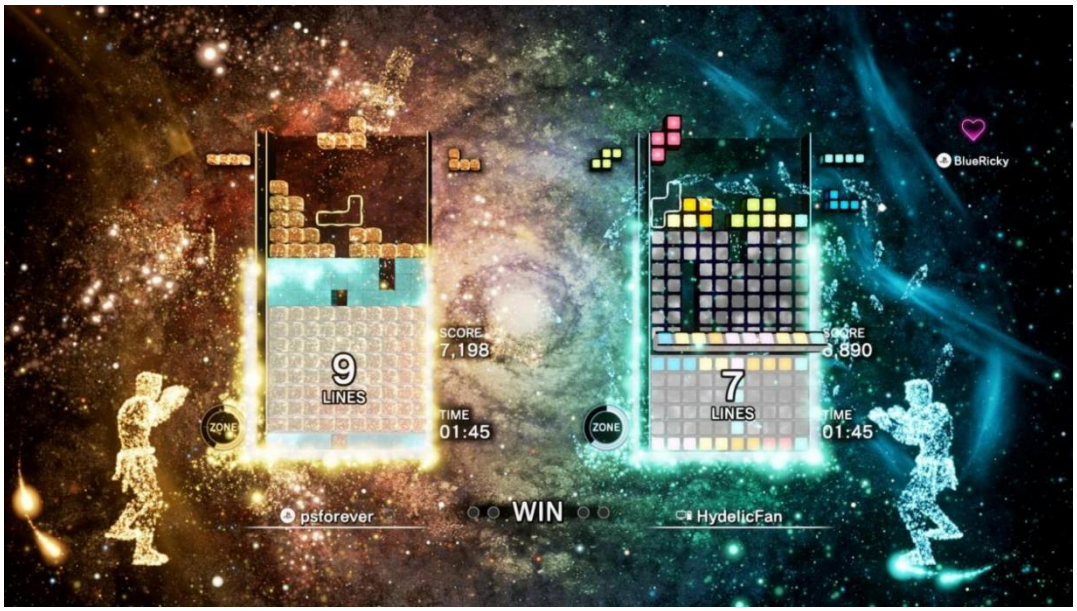

Рисунок 1.2.5 - Tetris Effect: Connected

6. Tetris 3D[9]: В этой вариации игроки манипулируют трёхмерными тетромино, заполняя объёмное пространство. Это добавляет новый уровень сложности и требует пространственного мышления, отличаясь от традиционного двумерного геймплея (Рисунок 1.2.6).

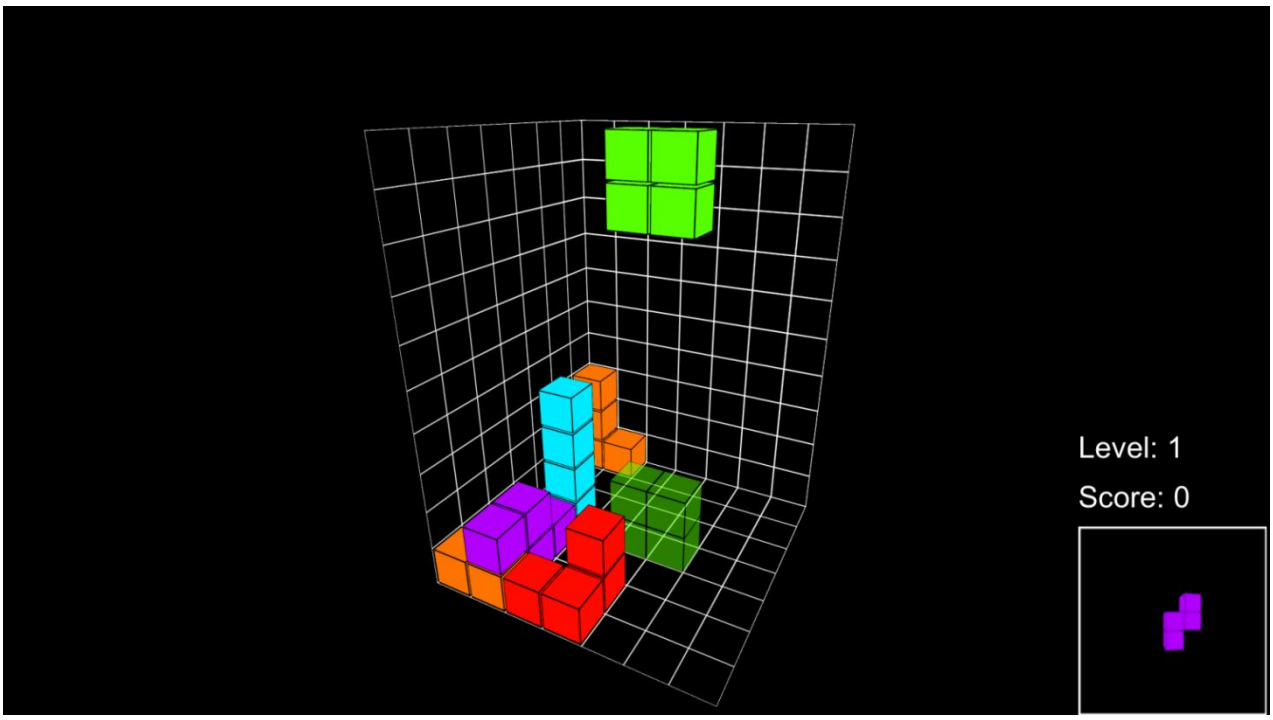

Рисунок 1.2.6 - Tetris 3D

7. Tetris Blitz[10]: Мобильная версия, предлагающая быстрые двухминутные раунды с акцентом на набор максимального количества очков за ограниченное время. Включает различные бонусы и усиления, добавляя динамичности и соревновательности в классический геймплей (Рисунок 1.2.7).

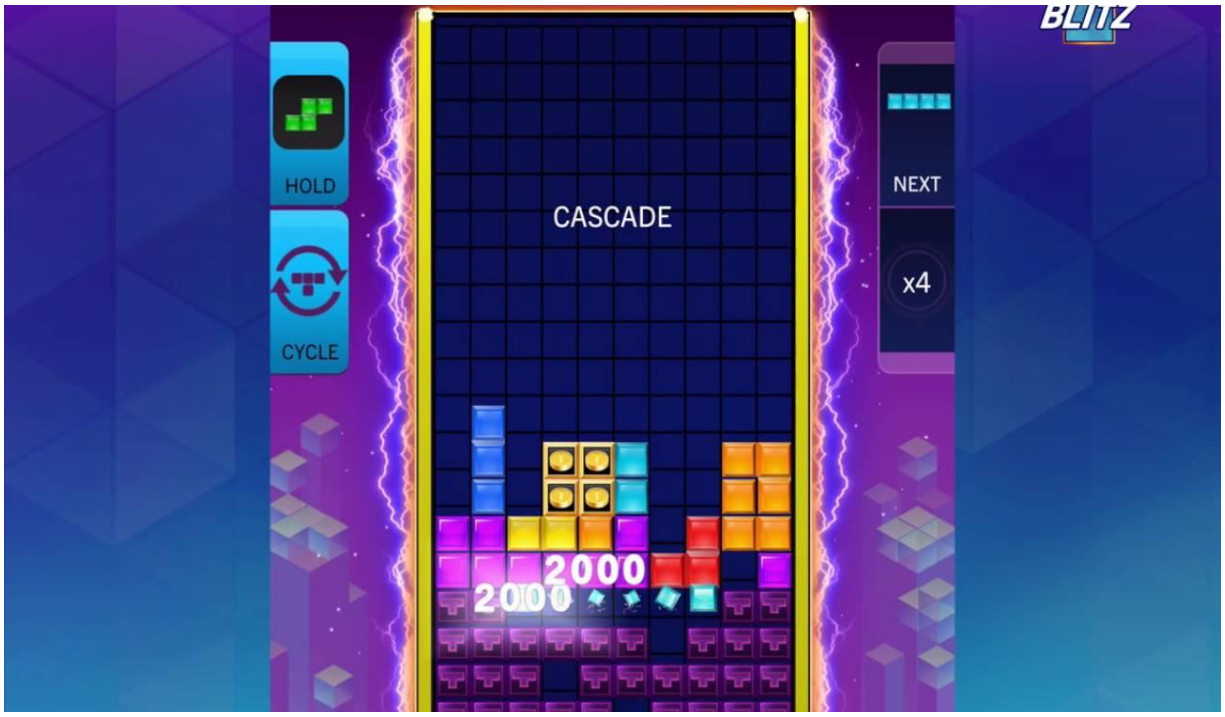

Рисунок 1.2.7 - Tetris Blitz

1.3 Требования к программируемой системе

В таблице 2.1 представлены требования к программируемой системе.

Таблица 2.1 – Требование к продукту

№	Требование	Значение
1	Язык программирования	C++
2	Корректность работы	Приложение запускается и поддерживает стабильный цикл работы от момента старта до завершения
3	Применение принципов объектно-ориентированного программирования	При написании приложения, как минимум, были использованы классы в C++, объектный подход к проектированию системы, а также инкапсуляция
4	Интерфейс пользователя	Создан интерфейс пользователя, поддерживающий корректный пользовательский опыт и содержащий все необходимые пояснения к работе и эксплуатации
5	Инструкция по эксплуатации	Написана инструкция по эксплуатации, содержащая, в том числе, основные рекомендации по использованию и пояснения к возможным ошибкам в программе
6	Графическая библиотека	Qt
7	Управление вводом	Клавиатура ПК
8	Реализация логики игры	Создано движение фигур, их вращение, удаление заполненных линий и смещение вниз после удаления
9	Физический движок	Самописный
10	Система управления версиями	Использование системы управления версиями Git для отслеживания изменений в коде и ресурсах, обеспечивающее безопасность и возможность отката к предыдущим версиям

2. ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

Для описания работы системы были использованы диаграммы, описывающие работу системы в различных ее аспектах. Были спроектированы диаграмма состояний системы (Рисунок 2.1) и диаграмма классов (Рисунок 2.2).

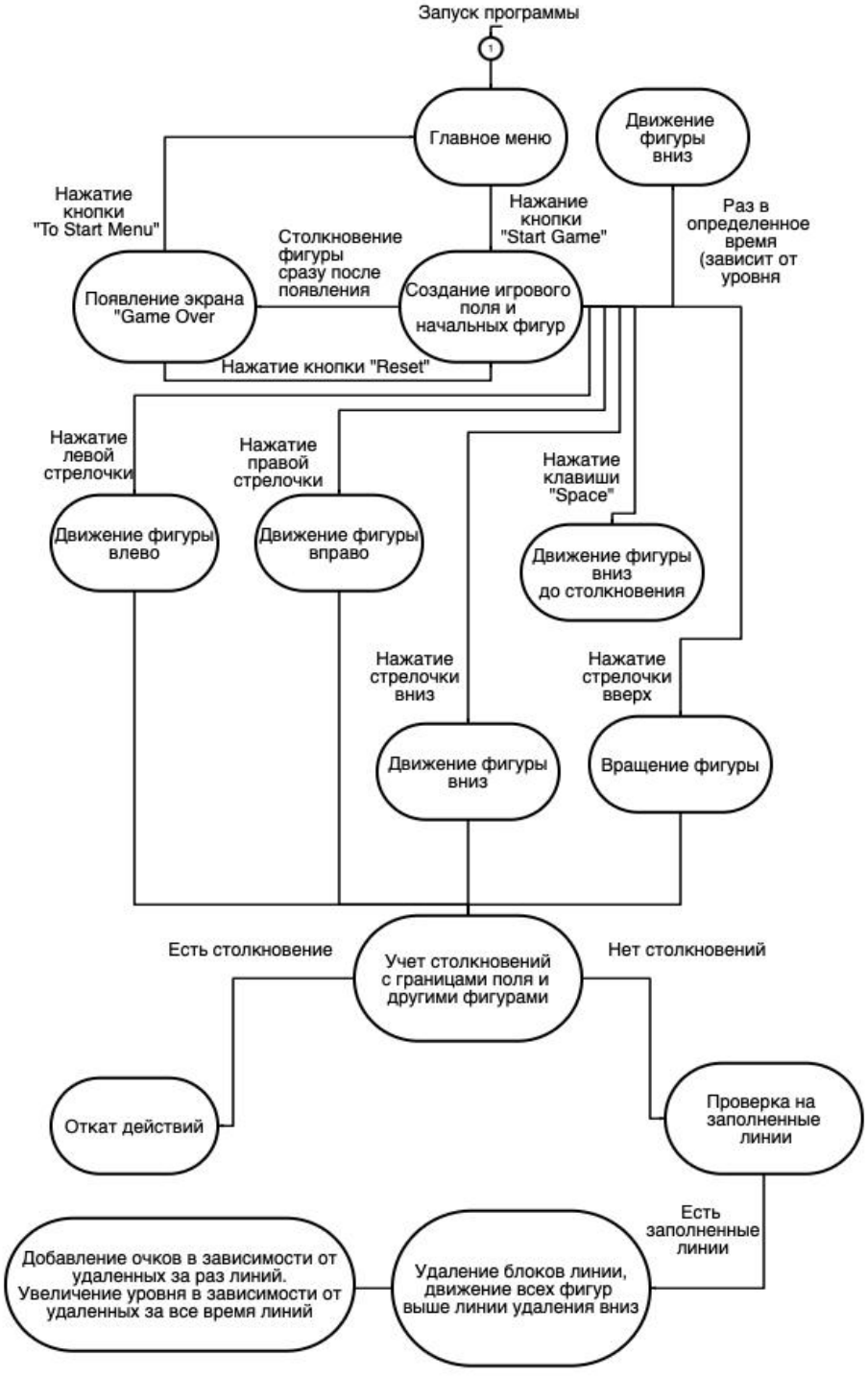

Рисунок 2.1 - Диаграмма состояний системы

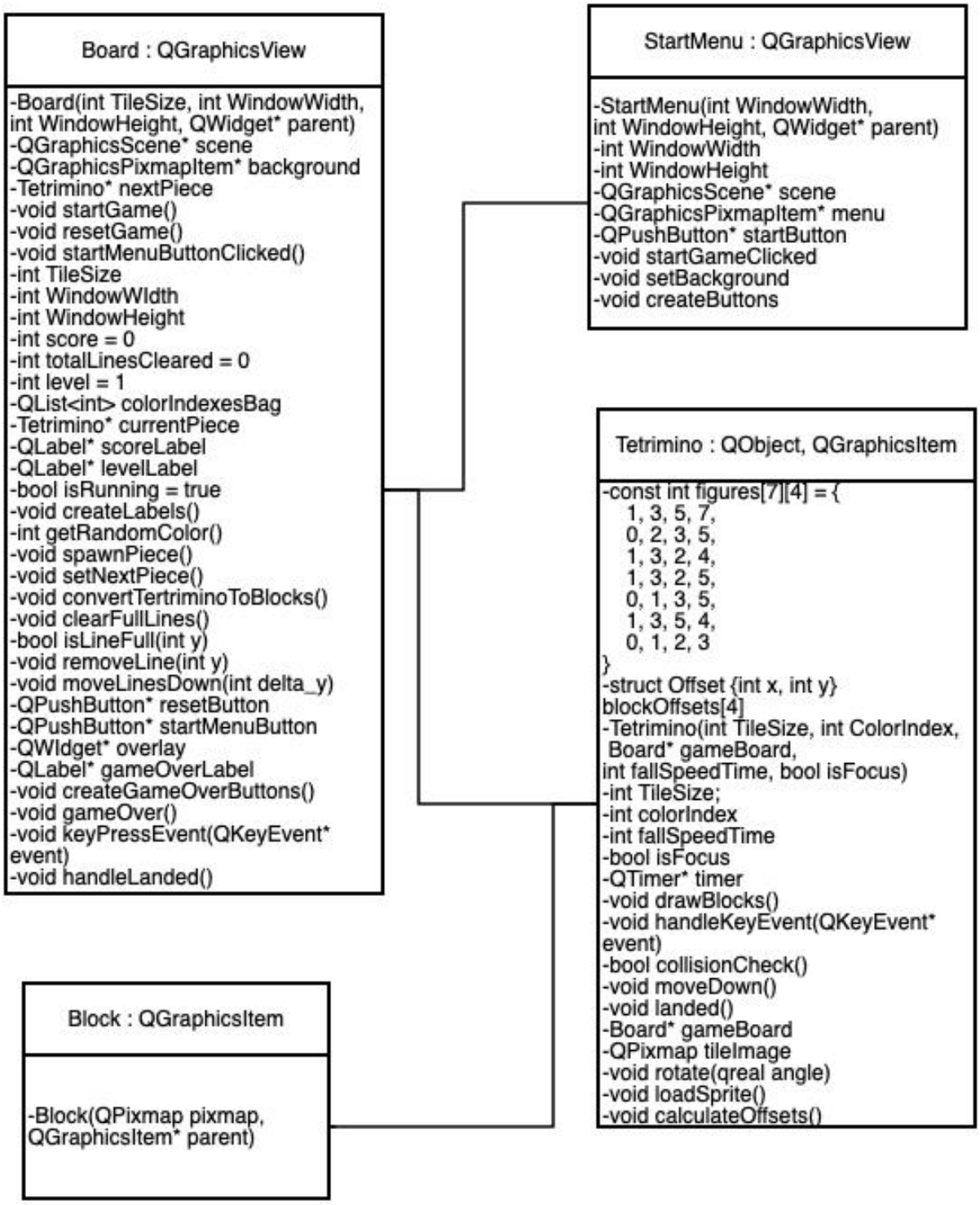

Рисунок 2.2 - Диаграмма классов

На рисунке 2.2 изображена диаграмма классов. Верхнаяя часть каждого блока написана в формате: “Название класса” : “От какого класса происходит наследование”. Далее в нижней части происходит перечисление объектов класса.

3. ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Реализация требований к системе

1. Язык программирования

В рамках курсовой работы была разработана игра-головоломка “Тетрис”, полностью написанная на языке программирования C++ и фреймворке Qt. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

2. Корректность работы

Обеспечена корректность функционирования приложения, выражающаяся в стабильности его запуска и поддержании непрерывного рабочего цикла вплоть до момента завершения сессии пользователя. Увидеть реализацию данного требования можно обратившись к скриншотам работы программы (Часть 3.2, Рисунки 3.1-3.8).

3. Применение принципов объектно-ориентированного программирования

В ходе разработки игры-платформера были применены ключевые принципы объектно-ориентированного программирования (ООП), включая инкапсуляцию, наследование и полиморфизм, что способствовало повышению модульности, расширяемости и поддерживаемости кодовой базы. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

4. Интерфейс пользователя

Интерфейс пользователя был разработан с учетом удобства взаимодействия и интуитивной понятности, обеспечивая легкий доступ ко всем функциональным возможностям игры и комфортное управление игровым процессом. Увидеть реализацию данного требования можно обратившись к скриншотам работы программы (Часть 3.2, Рисунки 3.1-3.8).

5. Инструкция по эксплуатации

Для обеспечения понимания пользователем всех аспектов работы с игрой-головоломкой “Тетрис” была подготовлена подробная инструкция по эксплуатации. Она содержит информацию о начале работы с игрой, описания интерфейса пользователя, правила и цели игры. Инструкция разработана с учетом принципов доступности и понятности для пользователей с различным уровнем опыта. Увидеть реализацию данного требования можно обратившись к специально отведенной части (3.3 Инструкция по эксплуатации) .

6. Графическая библиотека

Для реализации графического интерфейса и анимаций в игре использовался фреймворк Qt, что позволяет обеспечить полную кроссплатформенность. А встроенные системы сигналов и слотов дают возможность реализовать удобный ввод и соответственную логику, а также плавную отрисовку игровых объектов. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

7. Управление вводом

Управление вводом в приложении реализовано с использованием клавиатуры ПК, что обеспечивает точный и отзывчивый прием команд пользователя и способствует эффективному взаимодействию с игровым интерфейсом. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А, А4 – tetrmino.cpp).

8. Реализация логики игры

Реализация игры учитывает основные принципы оригинального “Тетриса”. Выполнена обработка движений фигуры и проверка на из валидность. После каждой поставленной фигуры проверяется факт наличия заполненных линий и их последующее удаление с учетом добавления очков в зависимости от количества таких линий за раз и уровня сложности игры. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А, А3 – board.cpp, А4 – tetrmino.cpp) и скриншотам игры (Часть 3.2, Рисунки 3.1-3.8).

9. Физический движок

Для реализации физики игры “Тетрис” был реализован физический движок управляющий всей логикой движения, столкновений и вращения фигур. Это позволило полностью управлять логикой игры и изменять ее под свои нужды (Приложение А, А4 – tetrmino.cpp). Увидеть реализацию данного требования можно обратившись к проекту, размещенного на github.

10. Система управления версиями

Использование системы управления версиями GIT в процессе разработки игры позволило обеспечить эффективное и удобное управление изменениям и внедрить сохранность кода, а также упростить процесс отладки и внедрения новых функций. Увидеть реализацию данного требования можно обратившись к проекту, расположенного на github.

3.2 Функциональное тестирование программного продукта

Работу игры на всех ее этапах координирует класс “StartMenu” и “Board” (Приложение А, А2 – startmenu.cpp, А3 – board.cpp), объект которых создается в главной функции (Приложение А, А1 – main.cpp).

Работа с программой начинается с вывода меню и запуска игры через кнопку “Start Game” (Рисунок 3.1). За вывод меню отвечает класс “StartMenu” (Приложение А, А2 – startmenu.cpp).

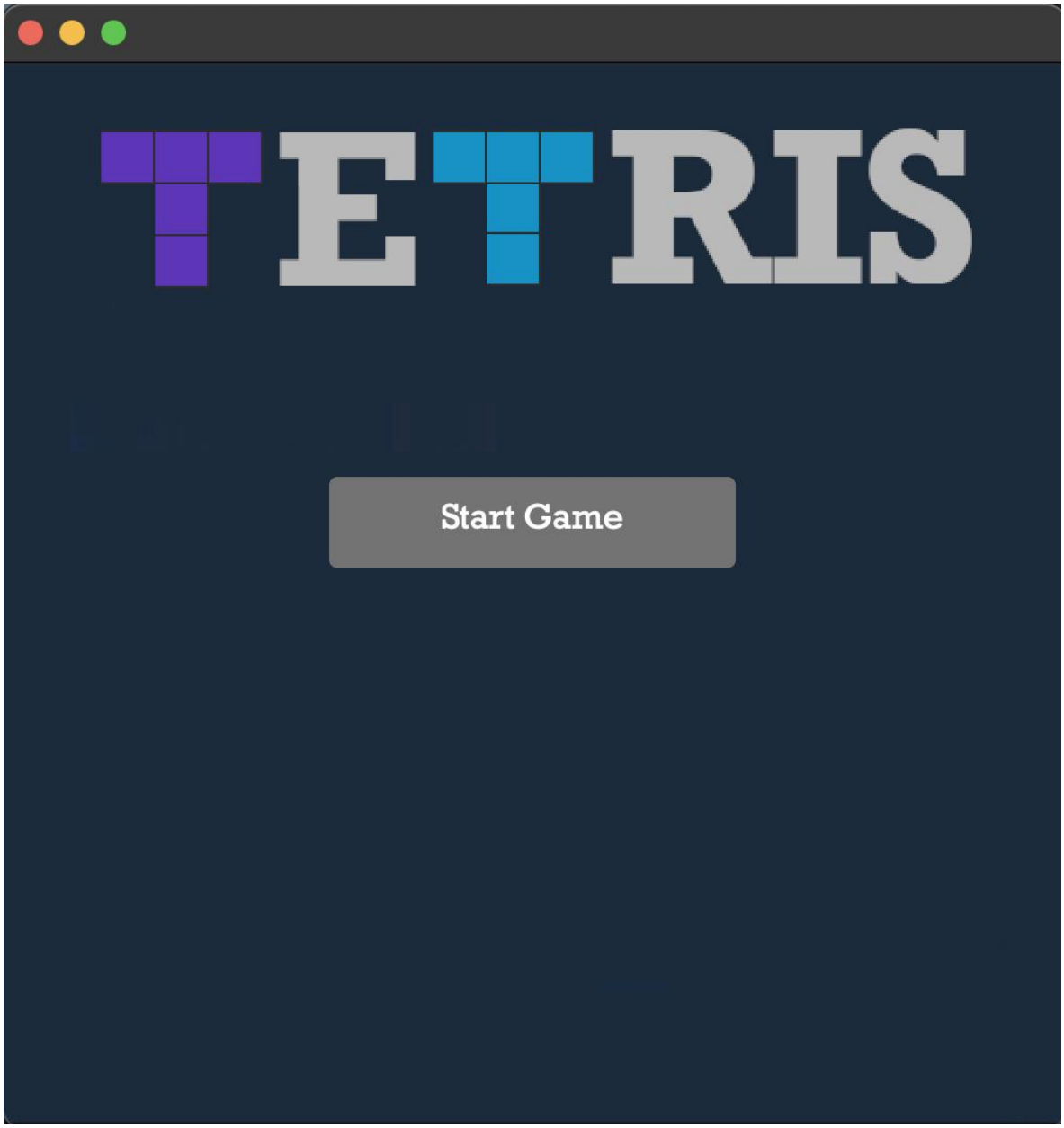

Рисунок 3.1 - Начальное меню

В случае нажатия кнопки “Start Game” приложение корректно переходит к игровому процессу (Рисунок 3.1, Рисунок 3.2). Для выхода из игры надо на любом этапе нажать на соответсвующей системе кнопку закрытия программы. Приложение будет закрыто без ошибок.

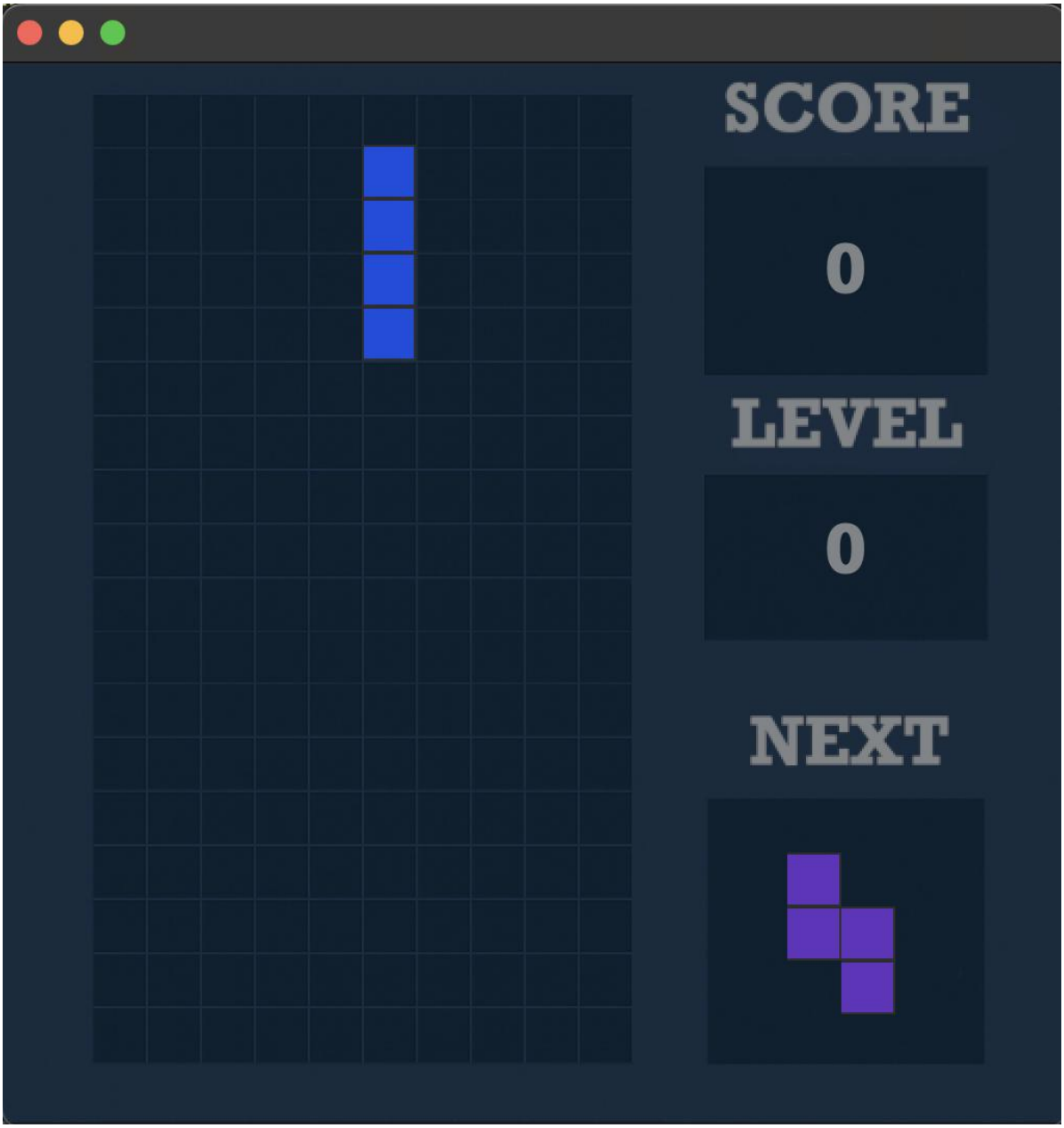

Рисунок 3.2 - начало игры

Фигуры появляются в верхней центральной части и начинают свое движение вниз с скоростью 1 блок в секунду и увеличивается скорость в зависимости от уровня игры (Рисунок 3.2). Фигуры продолжают свое движение до

столкновения с нижней границей игрового поля или до столкновения с другими поставленными фигурами (Рисунок 3.3, Рисунок 3.4)

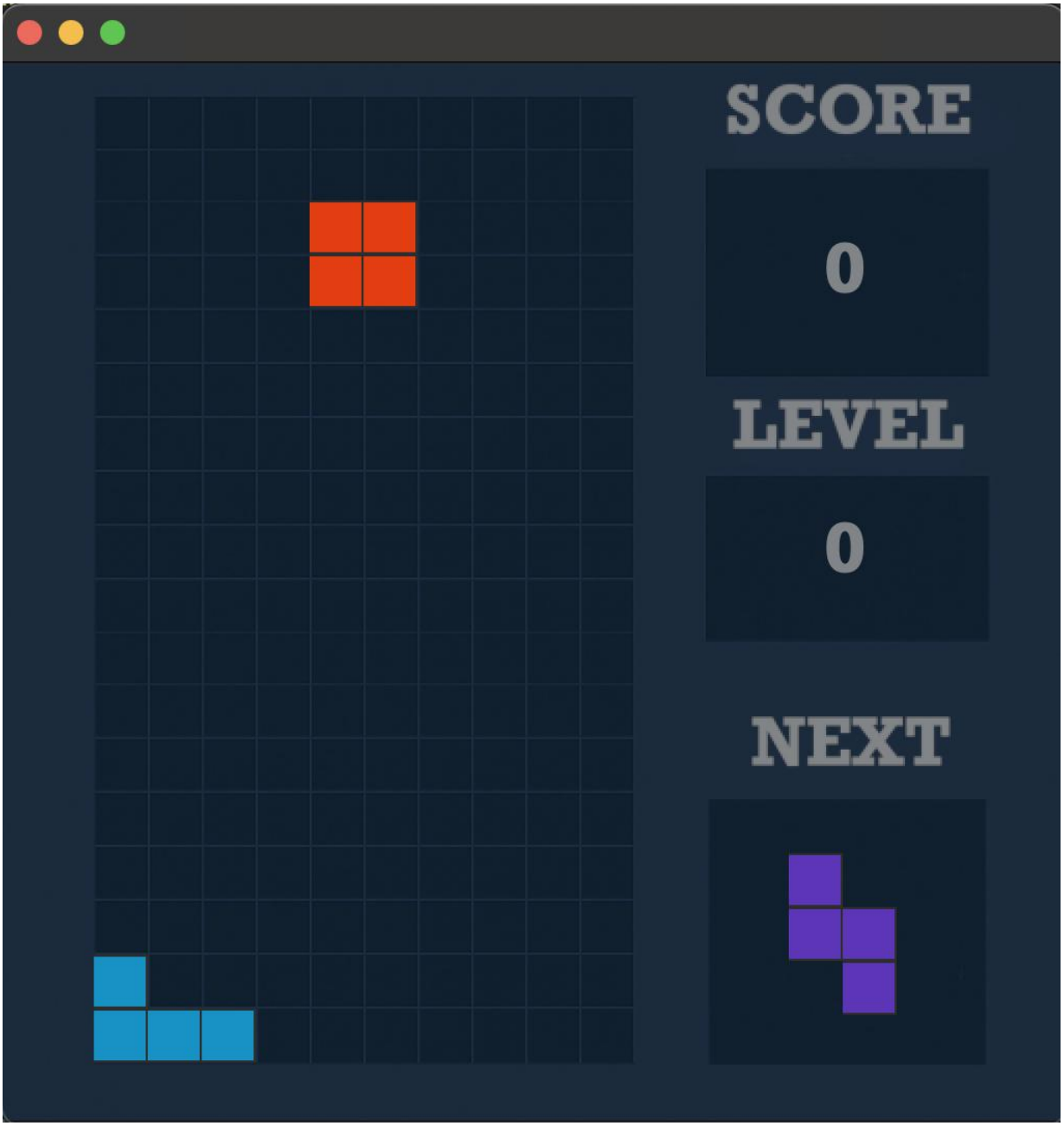

Рисунок 3.3 - падение фигуры на границу игрового поля

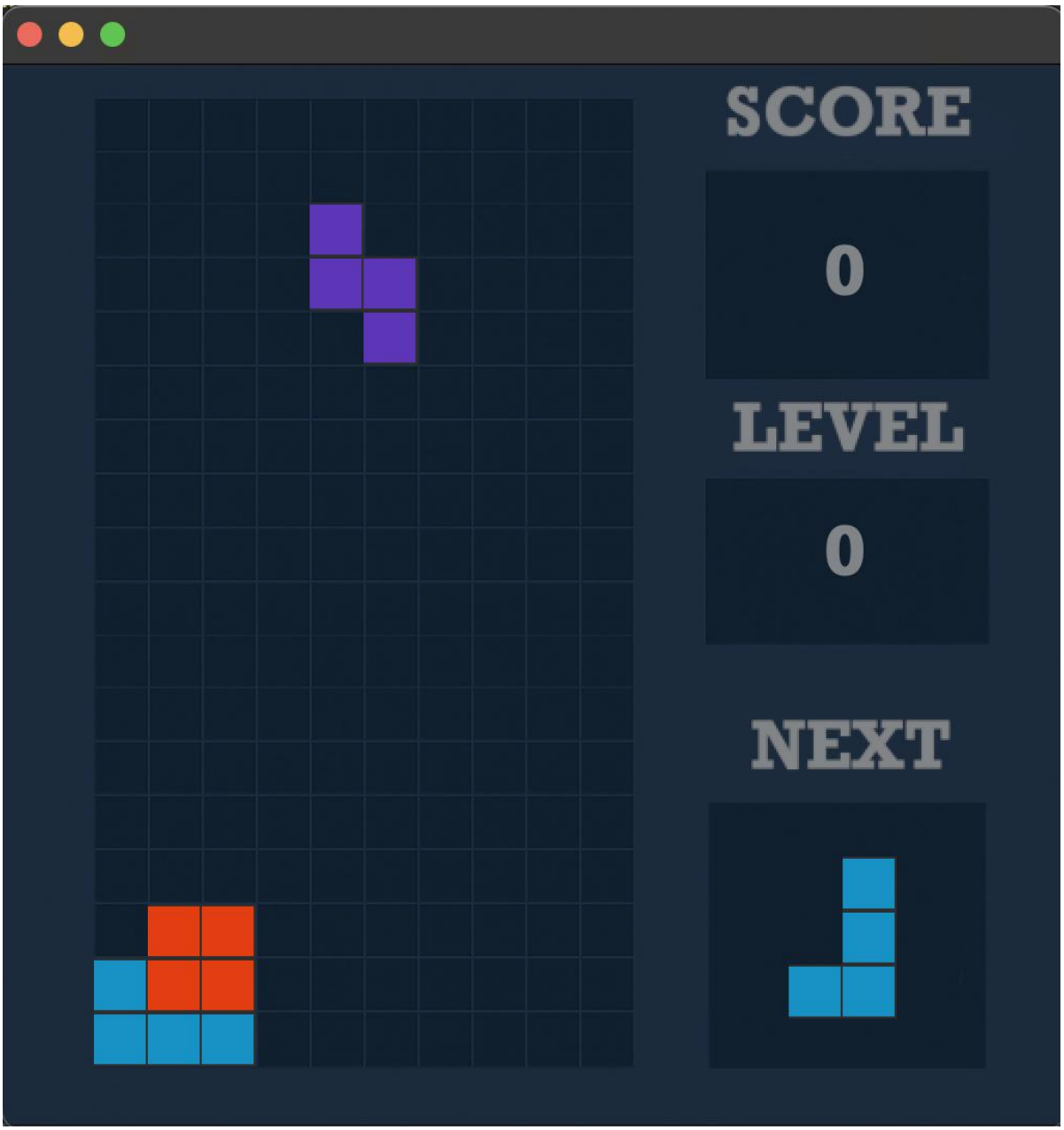

Рисунок 3.4 - падение фигуры на другую фигуру

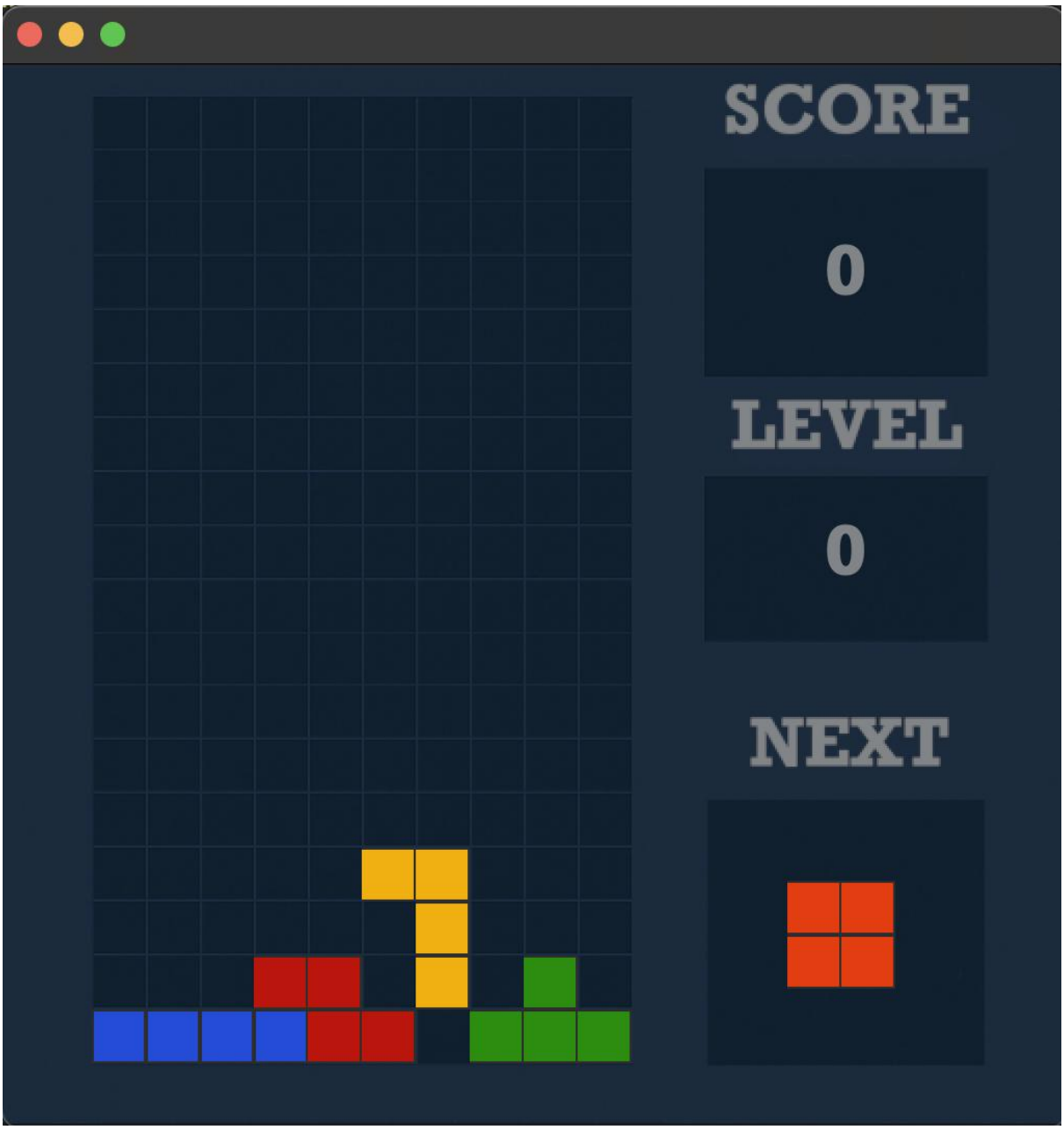

Рисунок 3.5 - желтая фигура в процесе падения

Если появляются заполненные линии, происходит их удаление и смещение всех линий выше вниз на количество удаленных линий, а также начисления очков в зависимости от количества удаленных линий (Рисунок 3.5, Рисунок 3.6). В случае, когда за одну игру было заполнено 10 линий, то увеличивается уровень на один, до десяти, что повышает скорость падения фигур и увеличивает количество получаемых очков (Рисунок 3.7).

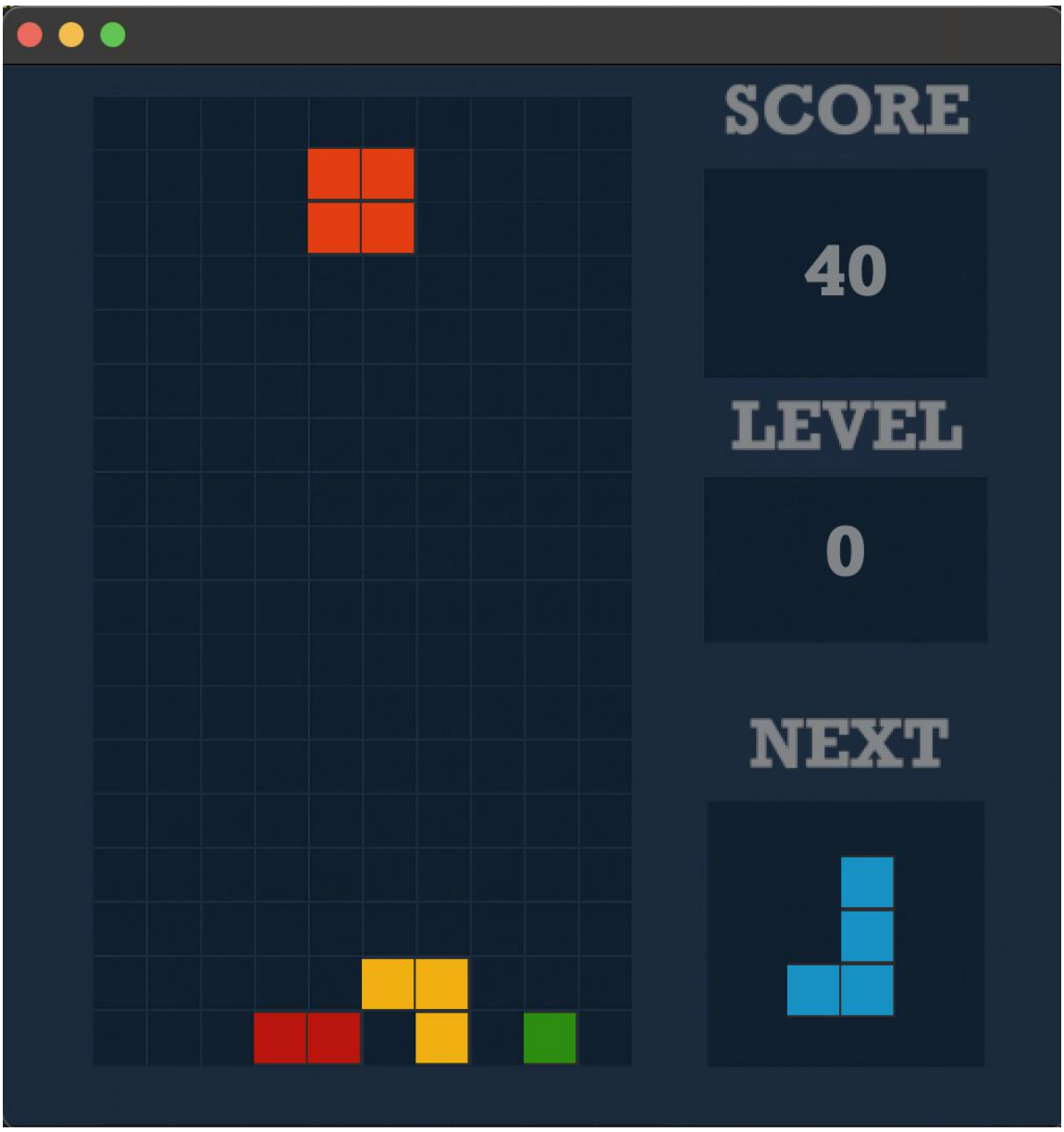

Рисунок 3.6 - удаление заполненных линий

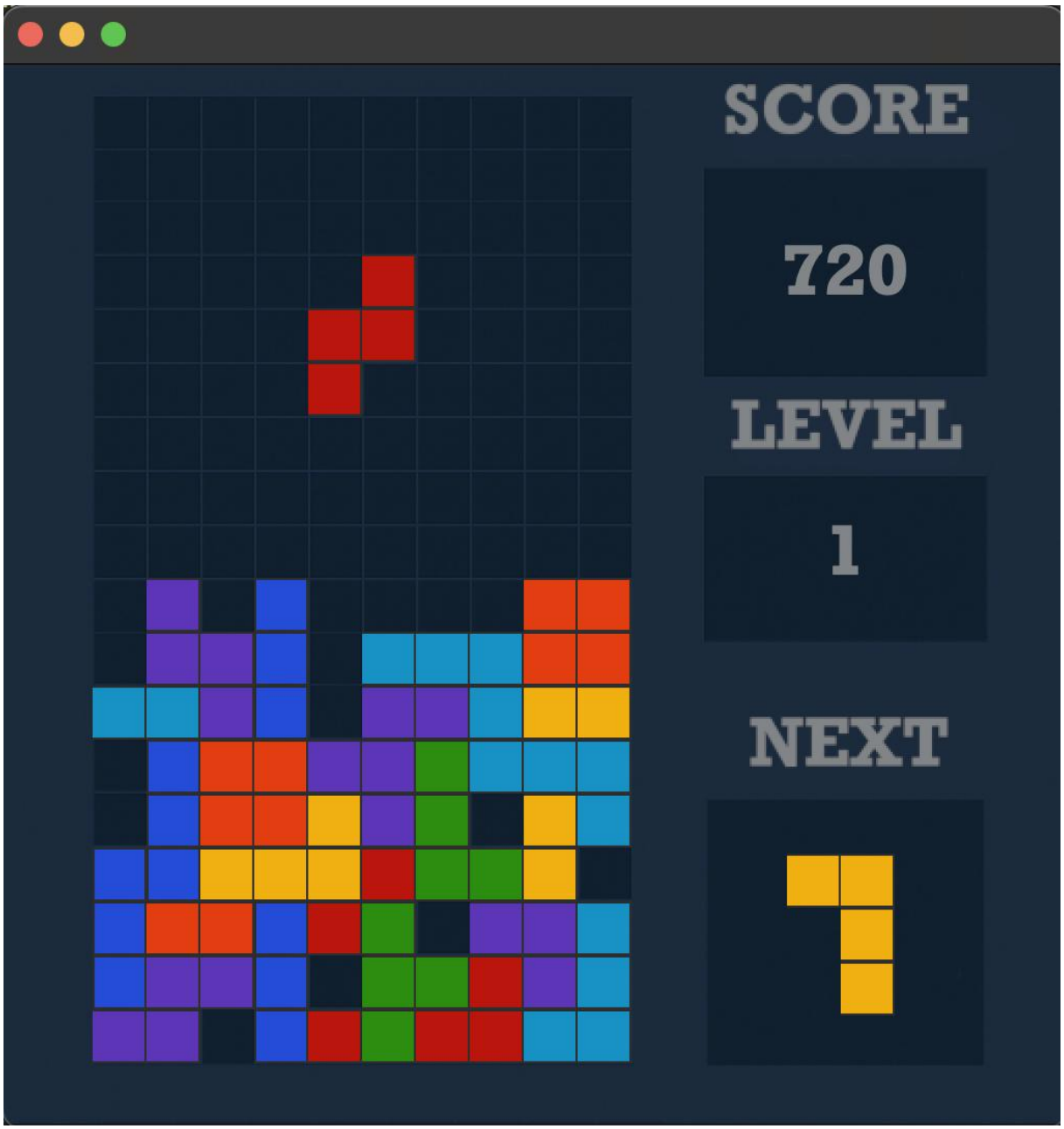

Рисунок 3.7 - увеличение уровня

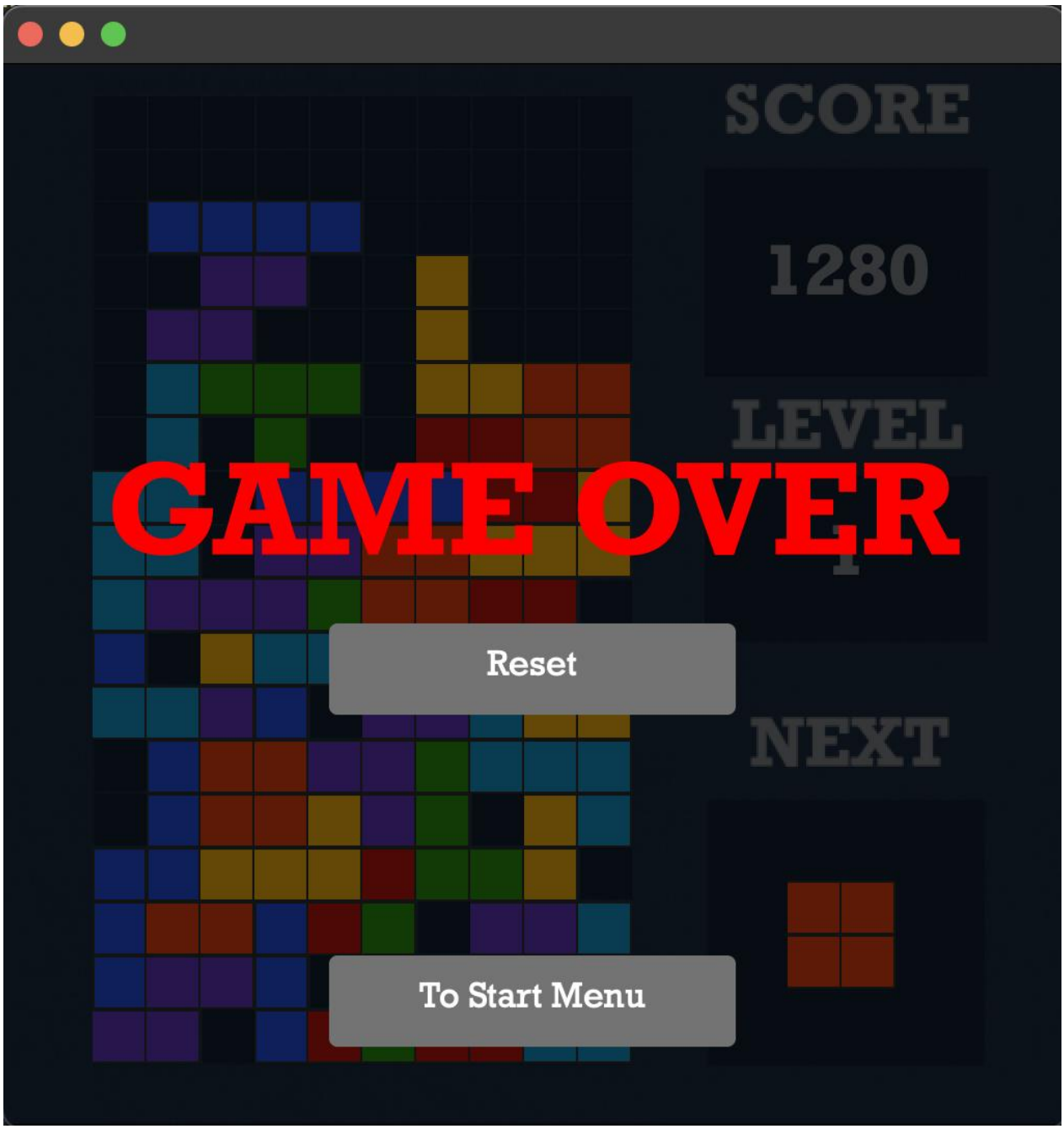

Рисунок 3.8 - конец игры

Когда сразу после появления фигруы она сталкивается с другой фигурой это считается за конец игры. Появляется экран “Game Over”, кнопка для перезапуска игры с сбросом очков и уровня “Reset”и кнопка для перехода в стартовое меню “To Start Menu” (Рисунок 3.8).

3.3 Инструкция по эксплуатации

После запуска программы пользователь попадает в меню, где ему будет доступна кнопка запуска игры “Start Game”. Для выхода из программы необходимо использовать метод закрытия приложений соответствующий операционной системы с которой был произведен запуск.

При нажатии кнопки “Start Game” запускается непосредственно игровой процесс. Пользователю доступны следующие клавиши:

1. Левая стрелочка - при нажатии на левую стрелочку фигура сместиться влево на один блок.
2. Правая стрелочка - при нажатии на правую стрелочку фигура сместиться вправо на один блок.
3. Стрелочка вверх - при нажатии на стрелочку вверх фигура совершит вращение на 90 градусов по часовой стрелке в рамках игрового поля.
4. Стрелочка вниз - при нажатии на стрелочку вниз фигура сместиться вниз на один блок.
5. “Space” - при нажатии на клавишу “Space” происходит движение фигуры вниз до момента первого столкновения.

Конечной целью игры является набор максимального количества очков, которые можно получить заполняя полностью линии. При этом важно учитывать, что если заполнять несколько линий за раз можно получить в разы больше очков, а также при заполнении 10 линий за игру будет увеличиваться уровень на один, что увеличивает скорость падения фигур, но увеличивает кол-во получаемых очков.

В случае когда после появления фигура сталкивается с другой фигурой, то происходит конец игры и появление соответствующего экрана с вариантами перейти в начальное меню и начать игру заново, сбросив очки и уровень.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была описана программируемая система, рассмотрены существующие решения-аналоги по теме, сформированы требования к системе, спроектированы диаграммы состояний, классов и последовательности самой системы. Программный продукт был разработан в соответствии с требованиями, протестирован, а также была написана инструкция по его эксплуатации.

Проект демонстрирует глубокое понимание объектно-ориентированного программирования, а также навыки работы с графической библиотекой Qt и системой управления версиями GIT. Разработанный продукт характеризуется стабильностью работы, качественной визуализацией и простотой использования.

В процессе разработки были решены сложные задачи, связанные с реализацией физического движка, столкновений фигур и корректной работой всех ситуаций в приложении. Использование GIT способствовало удобному управлению изменениями и доступности распространения игры.

Проект показал значительный потенциал для дальнейшего развития и улучшения, включая добавление новых функций, разных режимов, что может стать основой для будущих исследований и разработок в области создания игр. Работа над курсовой работой также способствовала повышению профессиональных компетенций в области программирования и дизайна игр.

Полученный опыт и знания будут служить твердой основой для дальнейшего обучения и профессионального роста в области разработки программного обеспечения и игровой индустрии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Компьютерная игра / [Электронный ресурс] // Википедия : [сайт]. — URL: 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 (дата обращения: 09.03.2024).
2. Как создаются видеоигры: процесс разработки игры / [Электронный ресурс] // : [сайт]. — URL: https://itanddigital.ru/videogame#:~:text=%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0%20%D0%B2%D0%B8%D0%B4%D0%B5%D0%BE%D0%B8%D0%B3%D1%80%20%D0%BE%D0%B1%D1%8B%D1%87%D0%BD%D0%BE%20%D0%B4%D0%B5%D0%BB%D0%B8%D1%82%D1%81%D1%8F%20%D0%BD%D0%B0,%2C%20%D0%BF%D1%80%D0%BE%D0%B4%D0%B0%D0%BA%D1%88%D0%BD%20%D0%B8%20%D0%BF%D0%BE%D1%81%D1%82%2D%D0%BF%D1%80%D0%BE%D0%B4%D0%B0%D0%BA%D1%88%D0%BD.&text=%D0%97%D0%B4%D0%B5%D1%81%D1%8C%20%D0%BD%D0%B0%D1%87%D0%B8%D0%BD%D0%B0%D0%B5%D1%82%D1%81%D1%8F%20%D0%BA%D0%B0%D0%B6%D0%B4%D1%8B%D0%B9%20%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82. (дата обращения: 09.03.2024).
3. Семь этапов создания игры: от концепта до релиза / [Электронный ресурс] // Habr : [сайт]. — URL: https://habr.com/ru/companies/miip/articles/308286/ (дата обращения: 09.03.2024).
4. Setris / [Электронный ресурс] // itch.io : [сайт]. — URL: https://lightpotato.itch.io/setris (дата обращения: 03.04.2025).
5. Tricky Towers / [Электронный ресурс] // Tricky Towers : [сайт]. — URL: https://www.trickytowers.com/ (дата обращения: 03.04.2025).

6. Tetris Time Warp / [Электронный ресурс] // Steam : [сайт]. — URL: https://store.steampowered.com/app/3180240/Tetris_Forever/ (дата обращения: 03.04.2025).
7. Cozy Blocks: Undersea Adventure / [Электронный ресурс] // Steam : [сайт]. — URL: https://store.steampowered.com/app/1522300/Cozy_Blocks_Undersea_Adventure/?l=russian (дата обращения: 03.04.2025).
8. Tetris Effect: Connected / [Электронный ресурс] // Steam : [сайт]. — URL: https://store.steampowered.com/app/1003590/Tetris_Effect_Connected/ (дата обращения: 03.04.2025).
9. Tetris 3D / [Электронный ресурс] // Tetris 3D : [сайт]. — URL: https://www.3dtetris.com/ (дата обращения: 03.04.2025).
10. Tetris Blitz / [Электронный ресурс] // Tetris : [сайт]. — URL: https://tetris.com/product/14/tetris-blitz-ios (дата обращения: 03.04.2025).

ПРИЛОЖЕНИЯ

Приложение А – Исходный код программы

Приложение А – Исходный код программы

Листинг А1 - main.cpp

```
#include <QApplication>
#include <QStackedWidget>
#include <functional>

#include "startmenu.h"
#include "board.h"

const int WindowWidth = 512;
const int WindowHeight = 512;
const int TileSize = 26;

Board* gameBoard = nullptr;

void startGame(QStackedWidget* stackedWidget, StartMenu* menu) {
    if (gameBoard) {
        stackedWidget->removeWidget(gameBoard);
        delete gameBoard;
    }

    // Create new board
    gameBoard = new Board(TileSize, WindowWidth, WindowHeight);
    stackedWidget->addWidget(gameBoard);
    stackedWidget->setCurrentWidget(gameBoard);

    QObject::connect(gameBoard, &Board::startMenuButtonClicked,
                     [=]() {
                         gameBoard->resetGame();
                         stackedWidget->setCurrentWidget(menu);
                     });

    gameBoard->startGame();
}

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
```

Продолжение Листинга А1 - main.cpp

```
        QStackedWidget* stackedWidget = new QStackedWidget;

        StartMenu* menu = new StartMenu(WindowWidth, WindowHeight);
        stackedWidget->addWidget(menu);  // Index 0

        QObject::connect(menu, &StartMenu::startGameClicked,
                         [=]() { startGame(stackedWidget, menu); });

        stackedWidget->setCurrentWidget(menu);
        stackedWidget->show();

        return a.exec();
    }
```

Листинг А1 - startmenu.cpp

```
    #include <QPixmap>
    #include <QGraphicsView>
    #include <QGraphicsScene>
    #include <QGraphicsPixmapItem>
    #include <QDebug>


    #include "startmenu.h"

    StartMenu::StartMenu(int WindowWidth, int WindowHeight, QWidget* parent) :
        WindowWidth(WindowWidth),                    WindowHeight(WindowHeight),
QGraphicsView(parent), scene(new QGraphicsScene(this))
    {
        if (menu){
            return;
        }

        setBackground();
        createButtons();

        this->setScene(scene);
        this->setFixedSize(WindowWidth, WindowHeight);
        this->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
        this->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
```

Продолжение Листинга А2 - startmenu.cpp

```
    scene->setSceneRect(0, 0, WindowWidth, WindowHeight);

}

void StartMenu::setBackground()
{
    QPixmap image(":/startmenu.png");

    menu = new QGraphicsPixmapItem();
    menu->setPixmap(image);
    scene->addItem(menu);
}

void StartMenu::createButtons()
{
    startButton = new QPushButton();
    startButton->setText("Start Game");
    startButton->setParent(this);

    startButton->setStyleSheet(
        "QPushButton {"
        "   background-color: #737373;"
        "   border: none;"
        "   color: white;"
        "   padding: 2px 8px;"
        "   text-align: center;"
        "   text-decoration: none;"
        "   font-family: 'Rockwell';"
        "   font-size: 16px;"
        "   margin: 4px 2px;"
        "   border-radius: 4px;"
        "}"
        "QPushButton:hover {"
        "   background-color: #404040;"
        "}"
        "QPushButton:pressed {"
        "   background-color: #272727;"
        "}"
        );
```

Продолжение Листинга А2 - startmenu.cpp

```
        startButton->setGeometry(WindowWidth/2 - 100, WindowHeight/2 - 60,200,
52);

        connect(startButton,            &QPushButton::clicked,            this,
&StartMenu::startGameClicked);
    }

    // void StartMenu::mousePressEvent(QMouseEvent* event)
    // {
    //     QGraphicsView::mousePressEvent(event);
    //     qDebug() << 1;  // Emit the signal when clicked
    // }
```

Листинг А3 - board.cpp

```
    #include <QTimer>
    #include <QPixmap>
    #include <QGraphicsPixmapItem>
    #include <QLabel>
    #include <QString>
    #include <QMutableListIterator>
    #include <QFontDatabase>
    #include <QPushButton>
    #include <random>



    #include "board.h"

    Board::Board(int TileSize, int WindowWidth, int WindowHeight, QWidget*
parent)
        :             TileSize(TileSize),             WindowWidth(WindowWidth),
WindowHeight(WindowHeight), QGraphicsView(parent), scene(new QGraphicsScene(this))
    {
        // устанавливаем задний фон
        QPixmap image(":/background.png");
        background = new QGraphicsPixmapItem();
        background->setPixmap(image);
        scene->addItem(background);

        setScene(scene);
        setFixedSize(WindowWidth, WindowHeight);
```

Продолжение Листинга А3 - board.cpp

```
        setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
        setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

        setFocusPolicy(Qt::StrongFocus);
        setFocus();

        scene->setSceneRect(0, 0, WindowWidth, WindowHeight);

        createLabels();
    }

    void Board::startGame()
    {
        int nextPieceColorIndex = getRandomColor();
        nextPiece = new Tetrimino(TileSize, nextPieceColorIndex, this, 30000,
false);

        spawnPiece();    // создаём первую фигуру
        currentPiece->setFocus();
    }

    void Board::createLabels()
    {
        scoreLabel = new QLabel("0", this);
        scoreLabel->setStyleSheet("QLabel { color: #828588; font-size: 32px;
font-family: 'Rockwell'; font-weight: bold; }");
        scoreLabel->setGeometry(345, 75, 125, 62);
        scoreLabel->setAlignment(Qt::AlignCenter);

        levelLabel = new QLabel("0", this);
        levelLabel->setStyleSheet("QLabel { color: #828588; font-size: 32px;
font-family: 'Rockwell'; font-weight: bold; }");
        levelLabel->setGeometry(345, 210, 125, 62);
        levelLabel->setAlignment(Qt::AlignCenter);

        //   QGraphicsTextItem   *textItem   =   scene->addText("Game   Over",
QFont("Arial", 30));
        // textItem->setDefaultTextColor(Qt::red);
        //         textItem->setPos(scene->width()/2         -         textItem-
>boundingRect().width()/2,
        //                  scene->height()/2);
```

Продолжение Листинга А3 - board.cpp

```
    }

    int Board::getRandomColor()
    {
        std::random_device rd;
        std::mt19937 rng = std::mt19937(rd());
        if (colorIndexesBag.size() == 0) {
            colorIndexesBag = {0, 1, 2, 3, 4, 5, 6};
            std::shuffle(colorIndexesBag.begin(), colorIndexesBag.end(), rng);
        }

        int colorIndex = colorIndexesBag.back();
        colorIndexesBag.pop_back();

        return colorIndex;
    }

    void Board::spawnPiece()
    {
        if (!isRunning) {
            return;
        }

        int colorIndex = nextPiece->colorIndex;

        int fallSpeedTime = 1000 - level * 100;
        currentPiece = new Tetrimino(TileSize, colorIndex, this, fallSpeedTime,
true);
        if (currentPiece->collisionCheck()) {
            gameOver();
            return;
        }
        scene->addItem(currentPiece);
        connect(currentPiece, &Tetrimino::landed, this, &Board::handleLanded);


        setNextPiece();
    }

    void Board::setNextPiece()
    {
```

Продолжение Листинга А3 - board.cpp

```
        int colorIndex = getRandomColor();
        if (nextPiece->scene()) {
            scene->removeItem(nextPiece);
        }
        nextPiece = new Tetrimino(TileSize, colorIndex, this, 30000, false);

        QPointF position = QPointF(405, 420) - QPointF(nextPiece-
>boundingRect().width() / 2, nextPiece->boundingRect().height() / 2);
        if (colorIndex == 0) {
            position -= QPointF(TileSize, 0);
        }
        nextPiece->setPos(position);
        scene->addItem(nextPiece);
    }

    void Board::handleLanded()
    {
        convertTetriminoToBlocks();

        clearFullLines();
        spawnPiece();
    }

    void Board::convertTetriminoToBlocks()
    {
        // сохраняем позиции блоков на сцене
        QList<QPointF> blocksPos;
        QList<QGraphicsItem*> blocks = currentPiece->childItems();
        qreal rotationCurrentPiece = currentPiece->rotation();
        for (QGraphicsItem* block : blocks){
            // работает на гениальности и божьей помощи, никогда не трогать
            block->setTransformOriginPoint(TileSize / 2, TileSize / 2);
            block->setRotation(360 - rotationCurrentPiece);
            blocksPos.append(block->mapToScene(QPointF(0, 0)));
        }

        // удаляем фигуру с сцены
        scene->removeItem(currentPiece);

        // добавляем отдельные блоки фигуры на сцену
        for (int i = 0; i < blocks.size(); i++) {
```

Продолжение Листинга А3 - board.cpp

```
            QGraphicsItem* block = blocks[i];

            currentPiece->removeFromGroup(block);

            block->setPos(blocksPos[i]);
            scene->addItem(block);
        }
        delete currentPiece;
        scene->update();
    }

    void Board::keyPressEvent(QKeyEvent* event)
    {
        if (currentPiece) {
            currentPiece->handleKeyEvent(event); // You'll need to add this
        }
    }

    void Board::clearFullLines()
    {
        int countFullLines = 0;
        for (int y = 14; y < 18 * TileSize + 14 ; y += TileSize) {
            if (isLineFull(y)) {
                removeLine(y);
                moveLinesDown(y);

                countFullLines++;
            }
        }

        totalLinesCleared += countFullLines;
        level = totalLinesCleared / 10;
        if (level > 10) {
            level = 10;
        }
        levelLabel->setText(QString("%1").arg(level));
        scene->update();

        if (countFullLines > 0) {
            switch(countFullLines) {
            case 1: score += 40 * (level + 1); break;
```

Продолжение Листинга А3 - board.cpp

```
            case 2: score += 100 * (level + 1); break;
            case 3: score += 300 * (level + 1); break;
            case 4: score += 1200 * (level + 1); break;
            }
            scoreLabel->setText(QString("%1").arg(score));
            scene->update();
        }
    }

    bool Board::isLineFull(int y)
    {
        int countBlocksY = 0;
        QList<QGraphicsItem*> sceneItems = scene->items();
        for (int i = 0; i < sceneItems.size(); i++) {
            QGraphicsItem* sceneItem = sceneItems[i];
            if (sceneItem == background) {
                continue;
            }

            if (sceneItem->scenePos().y() == y) {
                countBlocksY++;
            }
        }

        if (countBlocksY == 10) {
            return true;
        }
        return false;
    }

    void Board::removeLine(int y)
    {
        QListIterator<QGraphicsItem*> it(scene->items());
        while (it.hasNext()) {
            QGraphicsItem* sceneItem = it.next();

            if (sceneItem == background) {
                continue;
            }

            if (sceneItem->scenePos().y() == y) {
```

Продолжение Листинга А3 - board.cpp

```
                scene->removeItem(sceneItem);
                delete sceneItem;
            }
        }
        scene->update();
    }

    void Board::moveLinesDown(int y)
    {
        QList<QGraphicsItem*> sceneItems = scene->items();
        for (QGraphicsItem* sceneItem : sceneItems){
            if (sceneItem == background) {
                continue;
            }

            if (sceneItem->scenePos().y() < y){
                sceneItem->setPos(sceneItem->scenePos().x(),          sceneItem-
>scenePos().y() + TileSize);
            }
        }
    }

    void Board::createGameOverButtons()
    {
        resetButton = new QPushButton();
        resetButton->setText("Reset");
        resetButton->setParent(this);

        resetButton->setStyleSheet(
            "QPushButton {"
            "   background-color: #737373;"
            "   border: none;"
            "   color: white;"
            "   padding: 2px 8px;"
            "   text-align: center;"
            "   text-decoration: none;"
            "   font-family: 'Rockwell';"
            "   font-size: 16px;"
            "   margin: 4px 2px;"
            "   border-radius: 4px;"
            "}"
```

Продолжение Листинга А3 - board.cpp

```
            "QPushButton:hover {"
            "   background-color: #404040;"
            "}"
            "QPushButton:pressed {"
            "   background-color: #272727;"
            "}"
            );

        resetButton->setGeometry(WindowWidth / 2 - 100, WindowHeight / 2 +
10,200, 52);
        resetButton->show();

        connect(resetButton, &QPushButton::clicked, this, &Board::resetGame);


        startMenuButton = new QPushButton();
        startMenuButton->setText("To Start Menu");
        startMenuButton->setParent(this);

        startMenuButton->setStyleSheet(
            "QPushButton {"
            "   background-color: #737373;"
            "   border: none;"
            "   color: white;"
            "   padding: 2px 8px;"
            "   text-align: center;"
            "   text-decoration: none;"
            "   font-family: 'Rockwell';"
            "   font-size: 16px;"
            "   margin: 4px 2px;"
            "   border-radius: 4px;"
            "}"
            "QPushButton:hover {"
            "   background-color: #404040;"
            "}"
            "QPushButton:pressed {"
            "   background-color: #272727;"
            "}"
            );
```

Продолжение Листинга А3 - board.cpp

```
        startMenuButton->setGeometry(WindowWidth / 2 - 100, WindowHeight / 2 +
170,200, 52);
        startMenuButton->show();
        connect(startMenuButton,          &QPushButton::clicked,          this,
&Board::startMenuButtonClicked);
    }

    void Board::gameOver()
    {
        isRunning = false;
        overlay = new QWidget(this);
        overlay->setGeometry(0, 0, width(), height());
        overlay->setStyleSheet("background-color: rgba(0, 0, 0, 150);");
        overlay->show();

        gameOverLabel = new QLabel("GAME OVER", this);
        gameOverLabel->setStyleSheet("QLabel  {  color:  red;  font-size:  60px;
font-family: 'Rockwell'; font-weight: bold; }");
        gameOverLabel->setGeometry(0, 0, scene->width(), scene->height() - 60);
        gameOverLabel->setAlignment(Qt::AlignCenter);
        gameOverLabel->raise();
        gameOverLabel->show();

        createGameOverButtons();

        currentPiece->timer->stop();
        nextPiece->timer->stop();
    }
    void Board::resetGame()
    {
        if (overlay) {
            overlay->hide();
            overlay->deleteLater();
            overlay = nullptr;
        }

        if (gameOverLabel) {
            gameOverLabel->hide();
            gameOverLabel->deleteLater();
            gameOverLabel = nullptr;
        }
```

Продолжение Листинга А3 - board.cpp

```
    if (resetButton) {
        resetButton->hide();
        resetButton->deleteLater();
        resetButton = nullptr;
    }

    if (startMenuButton) {
        startMenuButton->hide();
        startMenuButton->deleteLater();
        startMenuButton = nullptr;
    }

    if (scene) {
        delete scene;
    }

    level = 0;
    score = 0;
    scoreLabel->setText(QString("%1").arg(score));

    scene = new QGraphicsScene(this);
    setScene(scene);

    // 3. Re-add background and reset state
    background = new QGraphicsPixmapItem(QPixmap(":/background.png"));
    scene->addItem(background);
    scene->setSceneRect(0, 0, WindowWidth, WindowHeight);

    isRunning = true;
    startGame();
}
```

Листинг А4 - tetrimino.cpp

```
#include "tetrimino.h"
#include <QPixmap>
#include <QGraphicsPixmapItem>
#include <QRect>
#include <QDebug>
#include <QList>
```

Продолжение Листинга А4 - tetrimino.cpp

```
#include <QPointF>
// #include <QRound>

#include "board.h"
#include "block.h"


Tetrimino::Tetrimino(int TileSize, int colorIndex, Board * gameBoard, int
fallSpeedTime, bool isFocus):
    TileSize(TileSize),    colorIndex(colorIndex),    gameBoard(gameBoard),
fallSpeedTime(fallSpeedTime), isFocus(isFocus)
{
    drawBlocks();

    timer->setInterval(fallSpeedTime);
    connect(timer, SIGNAL(timeout()), this, SLOT(moveDown()));

    timer->start(fallSpeedTime);
}

void Tetrimino::drawBlocks()
{
    loadSprite(); // загружаем спрайт блока

    calculateOffsets(); // вычисляем offsets для каждого блока из фигуры

    for (int i = 0; i < 4; i++)
    {
        // создание отдельных блоков, добавление offsets и соединение их в
одну Tetrimino
        QGraphicsPixmapItem * block = new Block(tileImage);

        block->setPos(blockOffsets[i].x  *  TileSize,  blockOffsets[i].y  *
TileSize);

        this->addToGroup(block);


        // устанавливаем точку вращения фигуры
        if (i == 1)
        {
```

Продолжение Листинга А4 - tetrimino.cpp

```
                this->setTransformOriginPoint(blockOffsets[i].x    *    TileSize,
blockOffsets[i].y * TileSize);
            }
        }

        // устанавливаем начальную позицию падения
        this->setPos((5 * TileSize + 18), (1 * TileSize + 14));



    }

    void Tetrimino::rotate(qreal angle) {
        setRotation(rotation() + angle);
    }

    void Tetrimino::handleKeyEvent(QKeyEvent *event)
    {
        if (event->key() == Qt::Key_Left){
            setX(x() - TileSize);

            if (collisionCheck())
            {
                this->setX(x() + TileSize);
                return;
            }
        }

        else if (event->key() == Qt::Key_Right){
            setX(x() + TileSize);

            if (collisionCheck())
            {
                this->setX(x() - TileSize);
                return;
            }
        }

        else if (event->key() == Qt::Key_Up){
            rotate(90);

            if (collisionCheck())
```

Продолжение Листинга А4 - tetrimino.cpp

```
            {
                rotate(-90);
                return;
            }
        }

        else if (event->key() == Qt::Key_Down){
            setY(y() + TileSize);

            if (collisionCheck())
            {
                this->setY(y() - TileSize);
                return;
            }
        }

        else if (event->key() == Qt::Key_Space) {
            while (true){

                setY(y() + TileSize);

                if (collisionCheck())
                {
                    this->setY(y() - TileSize);

                    // логика "приземления" тетримино
                    timer->stop();
                    emit landed();
                    return;
                }
            }
        }
    }

    bool Tetrimino::collisionCheck()
    {

        // проверки на границу поля
        if((sceneBoundingRect().left() < 44) || (sceneBoundingRect().right() >
44 + TileSize * 10) || (sceneBoundingRect().bottom() > TileSize * 19))
        {
```

Продолжение Листинга А4 - tetrimino.cpp

```
            return true;
        }

        // проверяем на столкновение с другими обьектами
        QList<QGraphicsItem*> sceneItems = gameBoard->scene->items();
        for (int i = 0; i < sceneItems.size(); i++) {
            QGraphicsItem* sceneItem = sceneItems[i];
            // пропускаем столкновение с задним фоном и само собой
            if (sceneItem == gameBoard->background) {
                continue;
            }

            if (sceneItem == this) {
                continue;
            }

            if (sceneItem == gameBoard->nextPiece) {
                continue;
            }


            for (QGraphicsItem* block : gameBoard->nextPiece->childItems()) {
                if (sceneItem == block) {
                    continue;
                }
            }


            QList<QGraphicsItem*> tetriminoChildItems = this->childItems();
            for (QGraphicsItem* tetriminoChildItem : tetriminoChildItems) {
                // пропускаем столкновение с блоками внутри самой фигуры
                if (sceneItem == tetriminoChildItem) {
                    continue;
                }

                if (tetriminoChildItem->collidesWithItem(sceneItem)) {
                    return true;
                }
            }
        }
        return false;
```

Продолжение Листинга А4 - tetrimino.cpp

```
}

// логика движения фигуры вниз спустя время
void Tetrimino::moveDown()
{
    setY(y() + TileSize);

    if (collisionCheck())
    {
        this->setY(y() - TileSize);

        // логика "приземления" тетримино
        timer->stop();
        emit landed();
        return;
    }
}

void Tetrimino::loadSprite()
{
    QPixmap spriteSheet(":/tiles.png");

    QRect tileRect(TileSize * colorIndex, 0, TileSize, TileSize);
    tileImage = spriteSheet.copy(tileRect);
}

void Tetrimino::calculateOffsets()
{
    for (int i = 0; i < 4; i++)
    {
        blockOffsets[i].x = figures[colorIndex][i] % 2;
        blockOffsets[i].y = figures[colorIndex][i] / 2;
    }
}
```

Листинг А5 - block.cpp

```
#include "block.h"

Block::Block(QPixmap pixmap, QGraphicsItem* parent):
QGraphicsPixmapItem(pixmap, parent)
{
```

Продолжение Листинга А5 - block.cpp

```
}
```