|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ИВАНОВСКОЙ ОБЛАСТИ  Областное государственное бюджетное профессиональное образовательное учреждение  «Ивановский промышленно-экономический колледж» | | | | | | |
|  | | | | | | |
| **КУРСОВОЙ ПРОЕКТ** | | | | | | |
|  | | | | | | |
| **Разработка мобильной игры «Переливашка»** | | | | | | |
|  | | | | | | |
| **ИВПЭК. 09.02.07. 22** | | | | | | |
|  | | | | | | |
|  | | **Специальность:** | | | **09.02.07 Информационные системы и программирование,**  **базовая подготовка** | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
| Руководитель курсового проекта | | |  | | | А.С. Горелов |
| Выполнил обучающийся группы 407а | | | |  | | Д.Д. Груздев |
|  | | | |  | |  |
|  | | | | | | |
|  | | | | | | |
| Курсовой проект выполнен и защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | |
|  | | | | | | |
| «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | | | | |
|  |  | | | | | |
|  | Иваново 2024 | | | | |  |

Содержание

[Введение 3](#_Toc181874433)

[1 Концептуальное проектирование 4](#_Toc181874434)

[1.1 Теоретический вопрос 4](#_Toc181874435)

[1.2 Спецификация требований программного обеспечения 7](#_Toc181874436)

[2 Техно-рабочий проект 13](#_Toc181874437)

[2.1Обоснование выбора средств разработки 13](#_Toc181874438)

[2.2 Разработка макета приложения 14](#_Toc181874439)

[3 Рабочая документация 17](#_Toc181874440)

[3.1 Описание разработки приложения 17](#_Toc181874441)

[3.2 Тестирование приложения 19](#_Toc181874442)

[Заключение 25](#_Toc181874443)

[Список использованной литературы 26](#_Toc181874444)

[Приложение А 27](#_Toc181874445)

# Введение

Программирование занимает ключевую позицию в современном мире, предоставляя разнообразные инструменты и технологии для создания интерактивных приложений и игр. Одним из наиболее увлекательных направлений программирования является разработка игр, которая позволяет не только развлекать пользователей, но и развивать их навыки логического мышления, реакции и креативности.

Создание игр, таких как "переливайка", становится все более популярным среди разработчиков. Эта игра, основанная на решении головоломок, предлагает пользователям уникальный опыт, требующий стратегического мышления и планирования. Разработка игры требует глубокого понимания игровых механик, дизайна уровней и пользовательского интерфейса, чтобы обеспечить интуитивное и увлекательное взаимодействие с игроками.

Целью моего курсового проекта является разработка игры "переливайка", которая будет привлекать пользователей своей простотой и интересными задачами. Основные задачи проекта включают:

* Изучение принципов разработки игр и использование игровых движков для создания интерактивного опыта.
* Проектирование структуры игры с учетом удобства и интереса пользователя.
* Реализация игрового процесса, включая механики переливания жидкостей и решения головоломок.
* Создание уровней с различной сложностью, чтобы удерживать интерес игроков и стимулировать их к дальнейшему прогрессу. Тестирование игры для обеспечения стабильной работы и комфортного игрового опыта.

# 1 Концептуальное проектирование

1.1 Теоретический вопрос

Концептуальное проектирование является основополагающим этапом в разработке программного обеспечения, направленным на определение базовых аспектов архитектуры и функциональности будущего приложения. Оно позволяет разработчику структурировать цели и задачи проекта, а также уточнить основные требования, что закладывает прочную основу для всех дальнейших этапов разработки.

Концептуальное проектирование позволяет сформулировать общее представление о том, каким должно быть приложение, как оно будет работать, какие проблемы пользователей оно будет решать и какие ключевые функции будут его отличать. Этот процесс помогает избежать излишней сложности и неопределенности на этапе разработки, а также минимизировать риски, связанные с возможными изменениями в структуре проекта на более поздних этапах.

Концептуальное проектирование играет важную роль на ранних этапах создания приложения, так как оно позволяет:

1. Выявить и формализовать цели проекта: Этот этап позволяет четко определить, зачем создается игра, какую основную задачу она должна выполнять и какие потребности игроков она будет удовлетворять. В нашем случае, цель разработки – создание увлекательной и интеллектуальной игры, которая развивает логическое мышление и внимательность, а также приносит удовольствие от процесса решения задач.
2. Определить необходимую функциональность: Концептуальное проектирование помогает выявить основные функции, которые необходимы для реализации цели проекта. В случае с игрой "Переливайка" это такие функции, как создание уровней с различной сложностью, система подсказок, управление игровыми ресурсами (например, количеством доступных переливаний) и возможность отслеживания прогресса игрока. На этом этапе продумывается, каким образом игрок будет взаимодействовать с игрой, чтобы эти функции были максимально удобны и интуитивно понятны.
3. Описать архитектурные подходы: Концептуальное проектирование позволяет продумать подход к организации данных и логике игры. Это включает выбор подходящих моделей данных для хранения информации об уровнях, состоянии игры и прогрессе игрока, определение методов доступа к этим данным и проектирование базовых взаимодействий между элементами интерфейса. Например, в данной игре хранение данных о уровнях будет организовано с помощью базы данных, а доступ к игровым данным будет осуществляться через специальные классы, обеспечивающие выполнение таких операций, как добавление, изменение и получение информации об уровнях и состоянии игры.
4. Закрепить базовые принципы интерфейса: Концептуальное проектирование также позволяет обдумать ключевые элементы интерфейса, с которыми будет взаимодействовать игрок. В данном проекте это означает разработку интуитивно понятного и привлекательного интерфейса, где игроку будет удобно начинать новые уровни, получать подсказки и отслеживать свой прогресс. На этом этапе определяются основные принципы расположения элементов интерфейса, что помогает минимизировать когнитивную нагрузку на игрока и обеспечивает плавность навигации по игровому процессу.

В контексте данного проекта концептуальное проектирование служит основой для создания четкой структуры мобильного приложения для планирования задач с функцией уведомлений. Оно позволяет выделить основные функции, которые будут реализованы в приложении, чтобы оно стало полезным и удобным инструментом для организации задач. Эти функции формируются на основе потребностей пользователей, которые заинтересованы в простом способе управления личными делами и получении напоминаний.

Основные задачи для данного приложения включают:

1. Создание интуитивно понятного интерфейса для игры: Пользователю должно быть легко понять правила и начать играть. Интерфейс должен включать простые и понятные элементы управления, такие как кнопки для начала игры, сброса уровня и управления звуками. Важно продумать визуальные компоненты, которые будут отображать текущие уровни, количество доступных действий и результаты, чтобы игрок мог легко отслеживать свой прогресс.
2. Реализация системы уровней и задач: Ключевой задачей является создание разнообразных уровней и задач, которые будут постепенно увеличивать сложность. Каждое задание должно быть уникальным и требовать от игрока применения логики и стратегии для достижения цели. Это может включать в себя различные механики, такие как ограниченное количество переливаний или использование специальных объектов, которые добавляют элементы неожиданности и интереса.
3. Обеспечение сохранения прогресса и достижений: Данные о достигнутых уровнях и успехах игрока должны сохраняться, чтобы пользователь мог продолжать игру с того места, где остановился. Это включает в себя сохранение данных в базе данных или локальном хранилище, а также возможность отображения достижений игрока в виде статистики или наград. Гибкая система сохранения позволит игрокам не терять прогресс и мотивировать их к дальнейшему прохождению игры.

Таким образом, концептуальное проектирование игры "Переливайка" позволяет создать четкое видение структуры и функциональности приложения, ориентированное на пользовательские потребности. Этот этап является основой для более детального проектирования архитектуры, интерфейса и логики игры, обеспечивая цельность и последовательность в процессе разработки.

1.1.2 Анализ предметной области и постановка задач

В основе разработки игры "Переливайка" лежит предметная область развлечений и интерактивного взаимодействия. В современном обществе существует высокий спрос на игры, которые развлекают и одновременно тренируют логическое мышление игроков. Поэтому задача приложения заключается в создании увлекательного и интуитивно понятного игрового процесса, который позволит пользователям решать головоломки, используя различные уровни сложности и уникальные механики.

Основные задачи проектирования включают:

1. Определение интерфейса пользователя, который будет простым и доступным для игроков всех возрастов.
2. Разработка логики игры, позволяющей игрокам настраивать уровень сложности и выбирать различные режимы игры.
3. Создание функционала для прохождения уровней, получения бонусов и достижения высоких результатов.
4. Определение способов хранения данных о прогрессе игроков и обеспечении их сохранности.

Цель данного проекта состоит в создании качественной и увлекательной игры "Переливайка", которая будет развлекать пользователей и предоставлять им возможность развивать свои логические навыки через интересные и разнообразные задачи.

1.1.3 Анализ аналогов и существующих решений

На рынке существует множество игр, основанных на механике сортировки и переливания жидкостей, таких как "Water Sort Puzzle" и "Color Sort". Эти игры предлагают игрокам увлекательные головоломки, где необходимо переливать жидкости между бутылками, чтобы достичь определенной цели. Однако у этих игр есть свои недостатки, такие как сложные уровни, перегруженность интерфейса и отсутствие возможности настройки сложности.

В отличие от существующих игр, разрабатываемая игра "Переливайка" будет ориентирована на минимализм, удобство и простоту использования, а также на возможность точной настройки уровней и механики игры. Особое внимание будет уделено интуитивно понятному интерфейсу и адаптации игрового процесса под конкретные предпочтения пользователей.

1.2 Спецификация требований программного обеспечения

Спецификация требований программного обеспечения определяет ключевые функциональные и нефункциональные требования к создаваемой игре "Переливайка". Эти требования представляют собой набор условий, которые игра должна выполнить, чтобы обеспечить удобство, производительность и стабильность работы для конечного пользователя.

Функциональные требования

1. Основной игровой процесс:
   * Игрок должен иметь возможность переливать жидкости между бутылками.
   * Каждая бутылка должна иметь ограниченную вместимость и содержать жидкости различных цветов.
   * Игрок должен достигать цели, переливая жидкости так, чтобы каждая бутылка содержала жидкости одного цвета.
2. Уровни и сложности:
   * Игра должна включать несколько уровней с различной сложностью, включая количество бутылок, цветов и ограничений по времени или количеству ходов.
   * Игрок должен иметь возможность выбирать уровень перед началом игры.
3. Настройки уведомлений:
   * Игрок должен иметь возможность настраивать уведомления о завершении уровня, достижении цели или истечении времени.
   * Уведомления должны быть настраиваемыми по времени и типу (звуковые, визуальные).
4. Система подсказок:
   * Игра должна предоставлять подсказки для игроков, которые испытывают трудности с прохождением уровня.
   * Подсказки должны быть активируемыми по желанию игрока.
5. Сохранение прогресса:
   * Игра должна автоматически сохранять прогресс игрока, чтобы он мог продолжить с того места, где остановился.
   * Игрок должен иметь возможность вручную сохранять и загружать игру.

Нефункциональные требования

1. Интерфейс пользователя:
   * Интерфейс должен быть интуитивно понятным и минималистичным, чтобы не перегружать игрока.
   * Дизайн должен быть адаптивным для различных устройств (мобильные телефоны, планшеты).
2. Производительность:
   * Игра должна обеспечивать плавный игровой процесс без задержек и зависаний.
   * Время загрузки уровней не должно превышать 3 секунд.
3. Совместимость:
   * Игра должна быть совместима с основными операционными системами (iOS, Android).
   * Игра должна поддерживать различные разрешения экранов.
4. Безопасность:
   * Игра должна обеспечивать защиту данных пользователей и не собирать личную информацию без согласия.
5. Обратная связь:

Игра должна предоставлять игрокам возможность оставлять отзывы и предложения по улучшению.Таким образом, спецификация требований формирует основу для реализации приложения, ориентированного на удобство и стабильность. Учет как функциональных, так и нефункциональных требований позволяет разработать приложение, которое будет полезным, надежным и удобным инструментом для планирования и напоминаний о задачах.

1.2.3 Взаимодействие с концептуальным проектированием

Концептуальное проектирование играет важную роль в разработке технического задания для игры "Переливайка", так как именно оно задает общее направление и видение будущего игрового продукта. На этапе концептуального проектирования разрабатываются базовые принципы и цели, которые помогают определить функциональные и нефункциональные требования игры.

Функциональные требования, такие как механика переливания жидкостей, система уровней с различной сложностью и возможность настройки уведомлений, основываются на анализе пользовательских потребностей, выявленных в концептуальном проектировании. Эти требования обеспечивают функциональность, заложенную в начальной концепции, и служат для достижения основной цели игры – предоставления увлекательного и интуитивно понятного игрового процесса, который позволяет игрокам решать головоломки и достигать поставленных целей.

Нефункциональные требования, такие как простота интерфейса, производительность и стабильность, также формируются на основе концептуального проектирования. Они определяют качество взаимодействия с игрой и отвечают за то, чтобы пользовательский опыт соответствовал изначальной концепции — игра должна быть удобной в использовании, быстро реагировать на действия игрока и стабильно работать на различных устройствах.

Таким образом, концептуальное проектирование служит основой для создания игры "Переливайка", обеспечивая четкое понимание функциональных и нефункциональных требований, которые в дальнейшем будут реализованы в процессе разработки.

# 2 Техно-рабочий проект

## 2.1 Обоснование выбора средств разработки

Для разработки приложения для планирования задач с функцией уведомления я решил использовать Android Studio и язык программирования Kotlin. Это современный и широко используемый стек для создания приложений под платформу Android, обладающий рядом преимуществ:

* Официальная поддержка Google:

Android Studio является официальной средой разработки для Android-приложений, что гарантирует оптимизацию под платформу, регулярные обновления, поддержку современных библиотек и инструментов. Это обеспечивает актуальность проекта и доступ к современным возможностям.

* Стабильность и производительность:

Android Studwio предлагает мощные инструменты для разработки, такие как встроенный эмулятор, инспектор памяти и профайлер, которые упрощают тестирование и отладку. Среда предоставляет стабильную производительность и помогает оптимизировать приложение для плавной работы на различных устройствах.

* Kotlin – современный и безопасный язык:

Kotlin стал официальным языком для Android-разработки и предлагает удобный, лаконичный синтаксис, повышая скорость написания кода и снижая вероятность ошибок. Благодаря поддержке null-безопасности и расширенной совместимости с Java, Kotlin обеспечивает высокую стабильность кода и облегчает поддержку.

* Широкое сообщество и обширная документация:

Вокруг Android Studio и Kotlin сложилось большое сообщество разработчиков, что облегчает доступ к руководствам, примерам кода и форумам. Это помогает решать проблемы и позволяет находить решения для нестандартных задач.

* Интеграция с инструментами разработки интерфейса:

Android Studio поддерживает инструменты визуального дизайна интерфейсов, такие как Layout Editor, что помогает создавать и настраивать интерфейс с минимальным количеством ручного кода. Эта функция особенно полезна для обеспечения удобства использования приложения.

* Расширяемость и интеграция библиотек:

Android Studio поддерживает широкий спектр библиотек и фреймворков для разработки мобильных приложений, таких как Jetpack Compose для создания пользовательских интерфейсов и WorkManager для управления задачами. Это делает разработку приложения гибкой и позволяет добавлять дополнительный функционал без значительных затрат времени.

Исходя из этих аргументов, выбор Android Studio и Kotlin в качестве инструментов разработки для проекта является обоснованным и позволяет эффективно реализовать поставленные цели, обеспечивая гибкость, надежность и высокую производительность приложения.

## 2.2 Разработка макета приложения

Для проектирования макета и интерфейса приложения используется Word.

Всего в приложении 3 экрана:

* Главное меню
* Экран выбора уровня
* Экран игры

Главное меню (Рисунок 1) содержит в себе 1 - название игры, 2 - кнопка начало игры, 3 - кнопка настроек, 4 - кнопка выхода.

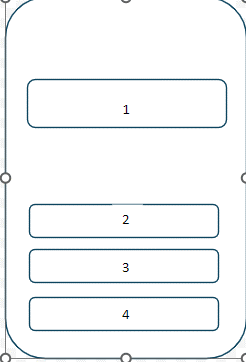


Рисунок 1 - главный экран

Экран выбора уровня (Рисунок 2) содержит 1 - кнопки выбора уровня, 2 - имя экрана "Select Level"

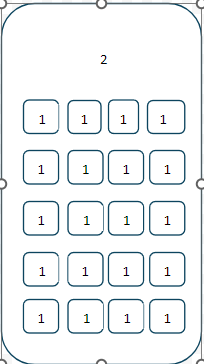


Рисунок 2 - экран выбора уровня

Экран игры (Рисунок 3) содержит в себе 1 - сколько раз выполнено движений, 2 - название игры, 3 - бутылки, 4 - кнопка перезапустить, 5 - выход.

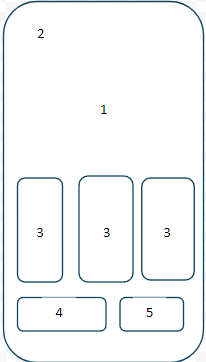


Рисунок 3 - экран игры

# 3 Рабочая документация

3.1 Описание разработки приложения

Для создания приложения необходимо реализовать несколько вещей:

1. Главное меню.
2. Экран уровней.
3. Механика игры.

Начнем с того что, создаем новый интерфейс screens.kt. В этом классе мы создаем функцию MainMenu которая отвечает за создание главного меню. Она принимает два параметра: функции обратного вызова onStartGame и onSettings, которые будут вызваны при нажатии соответствующих кнопок и ещё используется LaunchedEffect для установки видимости меню, что позволяет анимировать его появление. Меню состоит из Box, который заполняет весь экран и имеет градиентный фон. Внутри Box находится AnimatedVisibility, который управляет анимацией появления и исчезновения меню. Мы используем градиентный фон который использует вертикальный градиент для создания привлекательного фона и текст заголовка с помощь которого заголовок игры отображается в центре экрана.

Кнопки создаются с помощью функции MenuButton, которая принимает текст и действие при нажатии. Кнопки создаются с помощью функции MenuButton, которая принимает текст и действие при нажатии. Кнопка сжимается при нажатии, что создает эффект взаимодейств ия. Используется animateFloatAsState для анимации изменения масштаба кнопки. Кнопка имеет закругленные углы, тень и зеленый цвет фона, что делает её визуально привлекательной. Кнопка "Выход" завершает работу приложения, вызывая метод finish() у текущей активности. С помощью LaunchedEffect реализована задержка, которая сбрасывает состояние нажатия кнопки после 100 миллисекунд.

Функция SettingsScreen отвечает за создание экрана настроек. Она принимает функцию обратного вызова onBack, которая будет вызвана при нажатии кнопки "Назад". Используется LaunchedEffect для установки видимости экрана настроек, что позволяет анимировать его появление. Экран настроек состоит из Box, который заполняет весь экран и имеет градиентный фон. Внутри Box находится AnimatedVisibility, который управляет анимацией появления и исчезновения экрана. Используется вертикальный градиент для создания привлекательного фона экрана настроек. Заголовок "Settings" отображается в центре экрана, что делает его заметным для пользователя. С помощью кнопки назад позволяет пользователю вернуться на главное меню, вызывая функцию onBack.

Функция LevelSelectScreen отвечает за создание экрана выбора уровня. Она принимает два параметра: onLevelSelected, который вызывается при выборе уровня, и onBack, который возвращает игрока на предыдущий экран.  Используется LaunchedEffect для установки видимости экрана выбора уровня, что позволяет анимировать его появление. Экран выбора уровня состоит из Box, который заполняет весь экран и имеет градиентный фон. Внутри Box находится AnimatedVisibility, который управляет анимацией появления и исчезновения экрана. Уровни отображаются в виде сетки с использованием LazyVerticalGrid, что позволяет удобно размещать карточки уровней. Функция AnimatedLevelCard создает карточку для каждого уровня. Она принимает уровень и функцию обратного вызова onLevelSelected. Каждая карточка уровня появляется с задержкой, основанной на индексе уровня, что создает эффект последовательного появления. Карточка уровня создается с использованием компонента Card, который имеет стильный дизайн и интерактивность. Каждая карточка уровня реагирует на нажатие, вызывая функцию onLevelSelected, что позволяет игроку выбрать уровень.

Функция PaintSortGame отвечает за создание игрового экрана. Она принимает состояние игры и функцию для возврата в меню.  Используется LaunchedEffect для установки видимости игрового экрана, что позволяет анимировать его появление. Игровой экран состоит из Box, который заполняет весь экран и имеет градиентный фон. Внутри Box находится AnimatedVisibility, который управляет анимацией появления и исчезновения экрана.Игровой экран отображает количество ходов, лимит ходов и оставшееся время, что позволяет игроку отслеживать прогресс. Используется LazyVerticalGrid для отображения бутылок, что позволяет удобно организовать элементы игры. Кнопки "Reset", "Undo" и "Menu" предоставляют игроку возможность управлять игрой и возвращаться в меню.

После этого создаем класс Components.kt в нем я разробатываю ключевые компоненты интерфейса для игры. Функция TimeProgressBar отвечает за отображение прогресс-бара, который показывает оставшееся время в игре. Функция принимает два параметра: timeLeft (оставшееся время) и totalTime (общее время).

Используется LinearProgressIndicator для визуализации прогресса. Прогресс рассчитывается как отношение оставшегося времени к общему времени.Функция BottlesGrid создает сетку бутылок, отображая текущее состояние игры. Используется LazyVerticalGrid для отображения бутылок в виде сетки, что позволяет удобно организовать элементы интерфейса. Каждая бутылка отображается с помощью функции Bottle, которая принимает состояние бутылки и индекс. Функция Bottle отвечает за отображение отдельной бутылки, включая анимацию наливания. Используется LaunchedEffect для запуска анимации наливания, если бутылка находится в состоянии наливания. Внутри функции Bottle создается визуальное представление бутылки с использованием Canvas. Используется Canvas для рисования бутылки, ее содержимого и анимации наливания. Бутылка имеет градиентный цвет и эффект блеска, что делает ее более реалистичной. Функция LevelCompletedAlert создает диалоговое окно, которое уведомляет игрока о завершении уровня. : В диалоговом окне используется анимация Lottie для создания визуально привлекательного эффекта, который подчеркивает завершение уровня. Отображается сообщение о завершении уровня, количество использованных ходов и кнопка для перехода к следующему уровню.

Создаем класс GameState.kt. В этом классе я разробатываю основная механику игры.Класс Bottle представляет собой бутылку, содержащую цветные жидкости. В классе есть параметтры capacity - максимальная вместимость бутылки и contents - список цветов, находящихся в бутылке. Перечисление Level определяет различные уровни игры, каждый из которых имеет свои параметры:

* bottleCount: количество бутылок на уровне.
* colorCount: количество различных цветов.
* fullBottles: количество полных бутылок.
* emptyBottles: количество пустых бутылок.
* moveLimit: ограничение по количеству ходов (опционально).
* timeLimit: ограничение по времени (опционально).
* specialColors: список специальных цветов (опционально).
* stars: количество звезд, присуждаемых за завершение уровня.

Класс GameState управляет состоянием игры, включая бутылки, выбранные бутылки, количество ходов и другие параметры:

* bottles: список бутылок в игре.
* selectedBottle: индекс выбранной бутылки.
* moves: количество сделанных ходов.
* undoStack: стек для отмены действий.
* moveLimit и timeLeft: ограничения по ходам и времени.
* isGameOver и isLevelCompleted: флаги для проверки состояния игры.

Метод initializeLevel создает бутылки и заполняет их цветами в зависимости от параметров уровня. Метод generateColors создает список цветов, включая специальные цвета. Метод generateContents заполняет бутылки цветами, перемешивая их для случайности. Метод selectBottle обрабатывает выбор бутылки и наливание между бутылками. Метод pourBetweenBottles выполняет наливание из одной бутылки в другую, проверяя условия, такие как наличие содержимого и вместимость. Метод checkLevelComplete проверяет, завершен ли уровень.  Уровень считается завершенным, если все бутылки пустые или заполнены одним цветом. Метод updateTimeLeft обновляет оставшееся время и проверяет, не истекло ли оно. Если время истекло, вызывается метод checkGameOver. Метод navigateToNextLevel позволяет игроку перейти к следующему уровню после завершения текущего. ровни циклически переключаются, что позволяет игроку продолжать игру, пока не достигнет последнего уровня.

3.2 Тестирование приложения

Как только пользователь запустит приложение, его встретит главное меню. Главное меню показано на рисунке 3.

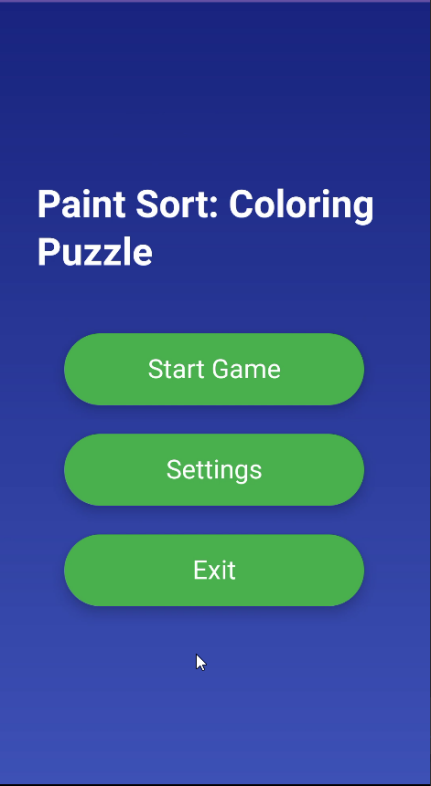


Рисунок 4 – Главное меню

После того как пользователь нажимает кнопку старт игры появляется выборка уровня. Выбор уровня показано на рисунке 5.

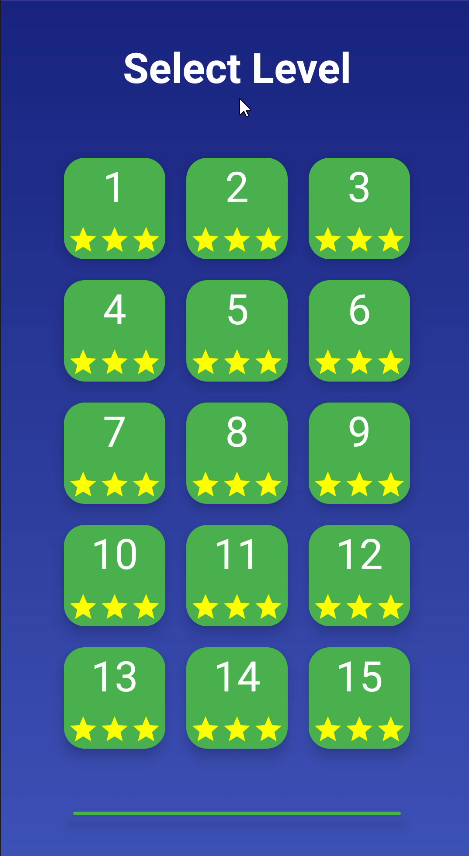


Рисунок 5 - экран выбора уровня

После того как пользователь выбрал уровень появляется игра в котором пользователю нужно решить головоломку. Экран головоломки показано на рисунке 6.

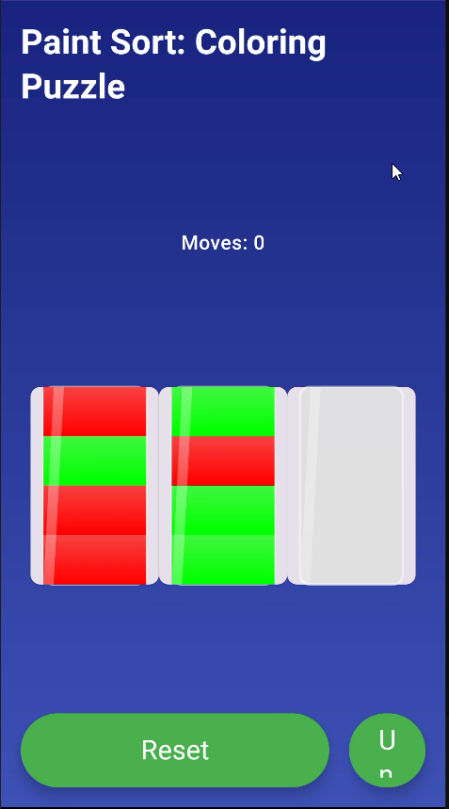


Рисунок 6 - экран головоломки

После решении головоломки появляется надпись что он выполнил уровень. Надпись выполнение уровня показано на рисунке 7.

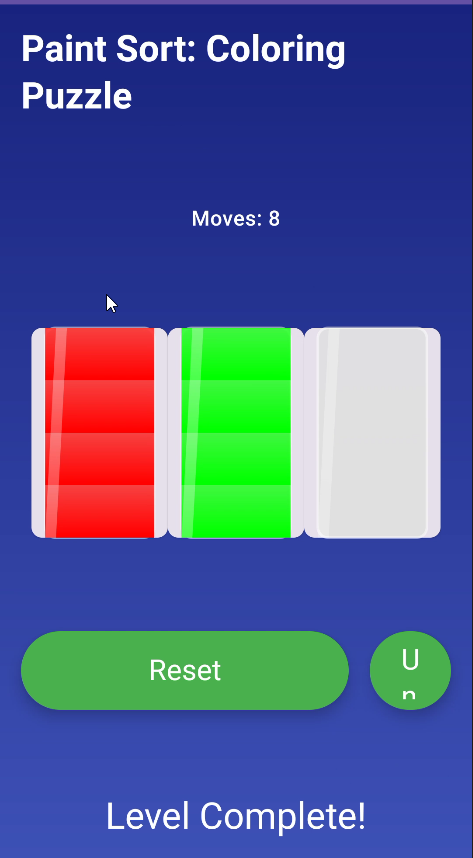


рисунок 7 - надпись выполнение уровня

# Заключение

В рамках данного курсового проекта было разработано мобильное приложение для планирования задач с функцией уведомления. Целью проекта было создание удобного и эффективного инструмента для организации задач пользователя с возможностью напоминания о предстоящих делах.

В процессе разработки были выполнены следующие задачи:

* Определена концепция и архитектура приложения, включающая функциональные и нефункциональные требования.
* Разработан интерфейс пользователя, обеспечивающий удобный доступ к функциям добавления, редактирования и удаления задач.
* Реализована система уведомлений, позволяющая пользователям получать напоминания о задачах в заданное время.
* Выполнено тестирование приложения для обеспечения стабильной работы и удобного пользовательского опыта.

В результате проделанной работы было создано мобильное приложение, которое упрощает процесс управления личными задачами и помогает пользователям более эффективно организовать своё время. Разработанный проект демонстрирует возможности Android Studio и языка программирования Kotlin для создания удобных и функциональных мобильных приложений.

Проект предоставляет основу для дальнейшего развития и усовершенствования, включая добавление новых функций и оптимизацию интерфейса для лучшего взаимодействия с пользователем.

# Список использованной литературы

1. Официальная документация Android [Электронный ресурс] - https://developer.android.com/?hl=ru, 2025
2. Сообщество разработчиков Android на Reddit [Электронный ресурс] - https://www.reddit.com/r/androiddev/, 2025
3. Форум Stack Overflow для разработчиков [Электронный ресурс] - https://stackoverflow.com/, 2025
4. Ray Wenderlich, руководства и статьи по Android Studio [Электронный ресурс] - https://www.kodeco.com/library, 2025
5. Официальная документация Kotlin [Электронный ресурс] - https://kotlinlang.org/docs/home.html, 2025
6. Руководства и гайды по Android Studio [Электронный ресурс] - https://www.linode.com/docs/guides, 2021
7. «Android Programming: The Big Nerd Ranch Guide» Билл Филипс, Крис Стюарт и Кристин Марсикано, 2016
8. «Kotlin в действии» Дмитрий Жемеров, Светлана Исакова, 2017
9. «Head First. Программирование для Android на Kotlin. 3-е изд.» Дон Гриффитс, Дэвид Гриффитс, 2023

# Приложение А

package com.example.perelevaica

import android.app.Activity

import androidx.compose.animation.AnimatedVisibility

import androidx.compose.animation.core.EaseOutBack  
import androidx.compose.animation.core.Spring  
import androidx.compose.animation.core.animateFloatAsState  
import androidx.compose.animation.core.spring  
import androidx.compose.animation.core.tween  
import androidx.compose.animation.expandVertically  
import androidx.compose.animation.fadeIn  
import androidx.compose.animation.fadeOut  
import androidx.compose.animation.scaleIn  
import androidx.compose.animation.scaleOut  
import androidx.compose.animation.shrinkVertically  
import androidx.compose.foundation.background  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.Arrangement  
import androidx.compose.foundation.layout.Box  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.PaddingValues  
import androidx.compose.foundation.layout.Row  
import androidx.compose.foundation.layout.Spacer  
import androidx.compose.foundation.layout.aspectRatio  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.foundation.layout.height  
import androidx.compose.foundation.layout.padding  
import androidx.compose.foundation.layout.width  
import androidx.compose.foundation.lazy.grid.GridCells  
import androidx.compose.foundation.lazy.grid.LazyVerticalGrid  
import androidx.compose.foundation.lazy.grid.items  
import androidx.compose.foundation.shape.RoundedCornerShape  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.*Star*import androidx.compose.material3.Button  
import androidx.compose.material3.ButtonDefaults  
import androidx.compose.material3.Card  
import androidx.compose.material3.CardDefaults  
import androidx.compose.material3.Icon  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.LaunchedEffect  
import androidx.compose.runtime.getValue  
import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.remember  
import androidx.compose.runtime.setValue  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.draw.clip  
import androidx.compose.ui.draw.scale  
import androidx.compose.ui.draw.shadow  
import androidx.compose.ui.graphics.Brush  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.text.font.FontWeight  
import androidx.compose.ui.unit.dp  
import kotlinx.coroutines.delay  
  
@Composable  
fun MainMenu(onStartGame: () -> Unit, onSettings: () -> Unit) {  
 val context = *LocalContext*.current  
 var visible by remember **{** *mutableStateOf*(false) **}** LaunchedEffect(Unit) **{** visible = true  
 **}** Box(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(  
 Brush.verticalGradient(  
 colors = *listOf*(*Color*(0xFF1A237E), *Color*(0xFF3F51B5))  
 )  
 )  
 ) **{** AnimatedVisibility(  
 visible = visible,  
 enter = *fadeIn*(animationSpec = *tween*(1000)) + *expandVertically*(  
 animationSpec = *tween*(1000, easing = *EaseOutBack*)  
 ),  
 exit = *fadeOut*() + *shrinkVertically*()  
 ) **{** Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(32.*dp*),  
 horizontalAlignment = Alignment.CenterHorizontally,  
 verticalArrangement = Arrangement.Center  
 ) **{** Text(  
 "Paint Sort: Coloring Puzzle",  
 style = MaterialTheme.typography.headlineLarge,  
 fontWeight = FontWeight.Bold,  
 color = Color.White  
 )  
 Spacer(modifier = Modifier.*height*(48.*dp*))  
 MenuButton(  
 text = "Start Game",  
 onClick = onStartGame  
 )  
 Spacer(modifier = Modifier.*height*(24.*dp*))  
 MenuButton(  
 text = "Settings",  
 onClick = onSettings  
 )  
 Spacer(modifier = Modifier.*height*(24.*dp*))  
 MenuButton(  
 text = "Exit",  
 onClick = **{** (context as? Activity)?.finish() **}** )  
 **}  
 }  
 }**}  
  
@Composable  
fun MenuButton(text: String, onClick: () -> Unit) {  
 var isPressed by remember **{** *mutableStateOf*(false) **}** val scale by animateFloatAsState(  
 targetValue = if (isPressed) 0.95f else 1f,  
 animationSpec = *spring*(dampingRatio = Spring.DampingRatioMediumBouncy, stiffness = Spring.StiffnessLow)  
 )  
  
 Button(  
 onClick = **{** isPressed = true  
 onClick()  
 **}**,  
 modifier = Modifier  
 .*width*(250.*dp*)  
 .*height*(60.*dp*)  
 .*shadow*(8.*dp*, *RoundedCornerShape*(30.*dp*))  
 .*clip*(*RoundedCornerShape*(30.*dp*))  
 .*scale*(scale),  
 colors = ButtonDefaults.buttonColors(containerColor = *Color*(0xFF4CAF50))  
 ) **{** Text(text, style = MaterialTheme.typography.titleLarge, color = Color.White)  
 **}** LaunchedEffect(isPressed) **{** if (isPressed) {  
 delay(100)  
 isPressed = false  
 }  
 **}**}  
  
@Composable  
fun SettingsScreen(onBack: () -> Unit) {  
 var visible by remember **{** *mutableStateOf*(false) **}** LaunchedEffect(Unit) **{** visible = true  
 **}** Box(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(  
 Brush.verticalGradient(  
 colors = *listOf*(*Color*(0xFF1A237E), *Color*(0xFF3F51B5))  
 )  
 )  
 ) **{** AnimatedVisibility(  
 visible = visible,  
 enter = *fadeIn*(animationSpec = *tween*(500)) + *expandVertically*(  
 animationSpec = *tween*(500, easing = *EaseOutBack*)  
 ),  
 exit = *fadeOut*() + *shrinkVertically*()  
 ) **{** Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(32.*dp*),  
 horizontalAlignment = Alignment.CenterHorizontally,  
 verticalArrangement = Arrangement.Center  
 ) **{** Text(  
 "Settings",  
 style = MaterialTheme.typography.headlineLarge,  
 fontWeight = FontWeight.Bold,  
 color = Color.White  
 )  
 Spacer(modifier = Modifier.*height*(48.*dp*))  
 Text("Settings options will be added here", color = Color.White)  
 Spacer(modifier = Modifier.*height*(48.*dp*))  
 MenuButton(  
 text = "Back",  
 onClick = onBack  
 )  
 **}  
 }  
 }**}  
  
@Composable  
fun LevelSelectScreen(onLevelSelected: (Int) -> Unit, onBack: () -> Unit) {  
 var visible by remember **{** *mutableStateOf*(false) **}** LaunchedEffect(Unit) **{** visible = true  
 **}** Box(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(  
 Brush.verticalGradient(  
 colors = *listOf*(*Color*(0xFF1A237E), *Color*(0xFF3F51B5))  
 )  
 )  
 ) **{** AnimatedVisibility(  
 visible = visible,  
 enter = *fadeIn*(animationSpec = *tween*(500)) + *expandVertically*(  
 animationSpec = *tween*(500, easing = *EaseOutBack*)  
 ),  
 exit = *fadeOut*() + *shrinkVertically*()  
 ) **{** Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(32.*dp*),  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{**