# СОДЕРЖАНИЕ

1. [ВВЕДЕНИЕ 6](#_bookmark0)
   1. [НАИМЕНОВАНИЕ ПРОГРАММЫ 6](#_bookmark1)
   2. ДОКУМЕНТ, НА ОСНОВАНИИ КОТОРОГО ВЕДЕТСЯ РАЗРАБОТКА
2. [НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ 7](#_bookmark2)
   1. [НАЗНАЧЕНИЕ ПРОГРАММЫ 7](#_bookmark3)
      1. [Функциональное назначение 7](#_bookmark4)
      2. [Эксплуатационное назначение 7](#_bookmark5)
   2. [КРАТКАЯ ХАРАКТЕРИСТИКА ОБЛАСТИ ПРИМЕНЕНИЯ 7](#_bookmark6)
3. [ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ 8](#_bookmark7)
   1. [ПОСТАНОВКА ЗАДАЧИ НА РАЗРАБОТКУ ПРОГРАММЫ 8](#_bookmark8)
   2. [ОПИСАНИЕ И ОБОСНОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММЫ 8](#_bookmark9)
      1. [Описание архитектуры программы 8](#_bookmark10)
      2. Оптимизация управления состояния на главном экране с использованием Diffable data Sourse
      3. Выделение частоиспользуемых элементов интерфейса
      4. Внедрение таймера в корректном режиме для корректной интеграции с онсовным режимом
      5. Выбор оптимального стека технологий
   3. [ОПИСАНИЕ И ОБОСНОВАНИЕ АЛГОРИТМА РАБОТЫ ПРОГРАММЫ 15](#_bookmark11)
      1. [Описание алгоритма работы программы 15](#_bookmark12)
      2. [Обоснование алгоритма работы программы 15](#_bookmark13)
   4. [ОПИСАНИЕ И ОБОСНОВАНИЕ ВЫБОРА СПОСОБА ОРГАНИЗАЦИИ](#_bookmark14) [ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ 16](#_bookmark14)
   5. [ОПИСАНИЕ И ОБОСНОВАНИЕ ВЫБОРА СОСТАВА ТЕХНИЧЕСКИХ И](#_bookmark15) [ПРОГРАММНЫХ СРЕДСТВ 16](#_bookmark15)
      1. [Состав технических и программных средств 16](#_bookmark16)
      2. [Обоснование выбора технических и программных средств. 16](#_bookmark17)
4. [ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ17](#_bookmark18)
   1. [ОРИЕНТИРОВОЧНАЯ ЭКОНОМИЧЕСКАЯ ЭФФЕКТИВНОСТЬ 17](#_bookmark19)
   2. [ПРЕДПОЛАГАЕМАЯ ПОТРЕБНОСТЬ 17](#_bookmark20)
   3. [ЭКОНОМИЧЕСКИЕ ПРЕИМУЩЕСТВА РАЗРАБОТКИ ПО СРАВНЕНИЮ С](#_bookmark21) [ОТЕЧЕСТВЕННЫМИ И ЗАРУБЕЖНЫМИ АНАЛОГАМИ 17](#_bookmark21)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18](#_bookmark22)

[ПРИЛОЖЕНИЕ 1 20](#_bookmark23)

**АННОТАЦИЯ**

Данный программный документ представляет собой пояснительную записку к программному проекту ««Мобильная игра кликер по контролю за сном»».

Раздел «Введение» включает в себя наименование программы и документ, на основании которого ведётся разработка, с указанием организации, утвердившей данный документ.

В разделе «Назначение и область применения» содержатся функциональное и эксплуатационное назначение программы и краткая характеристика области её применения.

В разделе «Технические характеристики» присутствуют следующие подразделы: постановка задачи на разработку программы, описание функционирования программы, описание и обоснование алгоритма работы программы, описание и обоснование выбора метода организации входных и выходных данных, описание работы с базой данных, описание и обоснование выбора состава технических и программных средств.

В разделе «Ожидаемые технико-экономические показатели» указана предполагаемая потребность и экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами.

Программный документ разработан в соответствии с требованиями:

1. ГОСТ 19.101-77 Виды программ и программных документов [1];
2. ГОСТ 19.102-77 Стадии разработки [2];
3. ГОСТ 19.103-77 Обозначения программ и программных документов [3];
4. ГОСТ 19.104-78 Основные надписи [4];
5. ГОСТ 19.105-78 Общие требования к программным документам [5];
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [6];
7. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к Пояснительной записке оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-78 [9].

Перед прочтением данного документа следует ознакомиться с терминологией, приведенной в Приложении 1.

# 1 ВВЕДЕНИЕ

# 1.1 Наименование программы

**Наименование темы разработки:** «Мобильная игра кликер по контролю за сном».

**Наименование темы разработки на английском языке:** ««Mobile Game Sleep Clicker»».

**Условное обозначение темы разработки: «**«Mobile Game Sleep Clicker»**».**

# 2 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

# Назначение программы

# Функциональное назначение

Программа позволяет пользователям управлять расписанием сна все большего количества персонажей, следить за различными игровыми героями, у каждого из которых свое уникальное время сна. Мониторинг сна происходит кликом. Мобильное приложение имеет развлекательное назначение

# Эксплуатационное назначение

Предположительными конечными пользователями могут быть как дети школьного возраста, так и взрослые люди, увлеченные кликерами. Расширение геймплея, в виде доступности к покупке новых персонажей, должно способствовать расширению аудитории и поддержанию ее интереса к игре.

# Краткая характеристика области применения

«Мобильная игра кликер по контролю за сном» - это игровое мобильное приложение на IOS, цель которого - обеспечить хороший сон для различных игровых персонажей. Пользователь управляет их расписанием сна, регулируя время сна и пробуждения c помощью клика. Различные игровые герои имеют разное время сна.

# 3 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

# Постановка задачи на разработку программы

«Мобильная игра кликер по контролю за сном» должно предоставлять возможность выполнения следующих задач:

* + 1. Просмотр списка действующих персонажей на главном экране;
    2. Возможность контроля состояния каждого персонажа: уровня усталости по цвету индикатор, просмотра количества жизней и состояния здоровья (жизненных сил);
    3. Возможность изменять состояние персонажа (персонаж может спать или бодрствовать);
    4. Возможность управления звуковыми эффектами в приложении: полностью отключать включать звуковое сопровождение в приложении;
    5. Возможность просматривать информацию о разработчике приложения;
    6. Возможность подключения и отключения уведомлений для захода в мобильное приложение;
    7. Возможность прохождения и пропуска экранов ознакомления, для обучения процесса игры;
    8. Возможность покупки новых персонажей;
    9. Предоставление интерфейса для просмотра накопленной валюты;
    10. Возможность покупки улучшений для временного увеличения количества, зарабатываемой в процессе игры валюты.

# Описание задач и их решения.

# Описание архитектуры программы

# Описание задачи:

# Для создания мобильной игры «Кликер по контролю за сном» и её последующего обновления и расширения возможностей необходимо определиться с выбором архитектуры, которая должна соответствовать таким основополагающим принципам, как четкая структура, модульное строение и гибкость для расширения.

# Варианты выполнения задачи:

# Были рассмотрены несколько вариантов архитектур, включая MVVM, Clean Swift (VIP), VIPER и MVC. Каждая из них имеет свои особенности и преимуществ

# Выбор решения:

Архитектура MVC (Model-View-Controller**)** является классическим паттерном проектирования, который разделяет приложение на три основных компонента:

1. Model: Компонент содержит основные данные, бизнес-логику и функции приложения. Модель отвечает за доступ к данным, их обработку, а также за правила бизнес-логики.
2. View: Компонент отображает данные пользователям и предоставляет интерфейс для взаимодействия.
3. Controller: Компонент отвечает за связь между Model и vVew. Код компонента Controller определяет, как сайт реагирует на действия пользователя

MVC, хотя и является распространенным шаблоном для разработки приложений, но может столкнуться с препятствиями по мере увеличения масштаба проекта. Это особенно заметно, когда контроллеры начинают несоразмерно нагружаться, что делает управление состоянием и добавление новых функций более сложным.

Моя задача - создать приложение с упором на легкость изменения и расширения состояния, а также добавление новых игровых элементов в будущем, мне необходима более модульная и адаптивная архитектура. По-этому эта архитектура не является оптимальной для моего проекта.

**Архитектурный подход MVVM (Model-View-ViewModel)** - один из ключевых методов, используемых при создании iOS-приложений, он включает в себя три основных элемента:

1. Model: элемент занимается всем, что связано с данными и бизнес-логикой приложения. Он отвечает за обработку, хранение данных и их трансформацию по мере необходимости.
2. View: элемент отвечает за визуальное отображение данных пользователю.
3. ViewModel: элемент функционирует как мост между моделью и представлением, не обладая прямым доступом к представлению. ViewModel реагирует на действия и запросы пользователя, обрабатывая их через бизнес-логику и при необходимости взаимодействует с моделью.

Если в проекте используются SwiftUI и Combine или аналогичные реактивные библиотеки, MVVM идеально подходит для такой среды разработки.

Однако, для разработки моего приложения был выбран фреймворк UIKit, который предлагает более широкие возможности для настройки компонентов пользовательского интерфейса, MVVM не является самым оптимальным выбором для моего проекта.

**VIPER (View, Interactor, Presenter, Entity, Router) —** это архитектурный шаблон для iOS-разработки, ориентированный на принципы чистой архитектуры. В нем взаимодействуют следующие компоненты;

1. View : отображает пользовательский интерфейс и перенаправляет действия пользователя.
2. Interactor: содержит бизнес-логику и обрабатывает запросы от Presenter.
3. Presenter: служит мостом между View и Interactor.
4. Entity: определяет модели данных.
5. Router: управляет навигацией и передачей данных между экранами.

Эта архитектура оптимальна для крупных проектов с комплексной бизнес-логикой и множеством модулей из-за её многоуровневой структуры. Она поддерживает работу как с

UIKit, так и с SwiftUI, обеспечивая гибкость в выборе технологий. Однако, учитывая её высокую структурную сложность, а также потенциальную перегрузку презентера и риск нарушения архитектурных принципов из-за переноса бизнес-логики, данная архитектура может оказаться чрезмерно сложной для моего проекта с более простым функционалом на этом этапе разработке.

**Clean Swift (VIP)** —предлагает модульный, масштабируемый и легко тестируемый подход к разработке, адаптированный под особенности iOS. В этой архитектуре:

1. Controller (ViewController): инициирует обработку пользовательских действий, передавая данные в Interactor. Он связывает View и остальную часть системы, управляя взаимодействием между пользователем и приложением.
2. Interactor: основное назначение — выполнение бизнес-логики приложения.
3. Workers: специализированные компоненты, отвечающие за операции с данными, например обращения к сети, работа с базами данных.
4. Presenter: принимает данные от Interactor, преобразует их в формат, удобный для отображения, и передает в View через ViewController.
5. Router: управляет переходами и передачей данных между экранами, в связке с ViewController.
6. Models: включают структуры, которые используются для передачи данных между компонентами архитектуры.
7. View: Представляет собой пользовательский интерфейс, через который осуществляется взаимодействие с пользователем. Она отображает данные, подготовленные Presenter и переданные через ViewController, и перенаправляет пользовательский ввод обратно в ViewController для дальнейшей обработки.

Выбор Clean Swift (VIP) для разработки мобильного приложения для контроля сна обусловлен несколькими ключевыми факторами:

1. Архитектура поддерживает развитие и расширение приложения, обеспечивая легкую интеграцию новых функций.
2. Каждый компонент архитектуры фокусируется на своей задаче, это упрощает тестирование, поддержку и развитие приложения.

# Результат:

Наиболее подходящей для моего проекта была выбрана архитектура Clean Swift (VIP). Этот выбор обоснован её способностью обеспечить высокую структурированность кода, модульность компонентов и легкость расширения приложения в будущем**.** Clean Swift (VIP) поддерживает модульную структуру, что позволяет легко добавлять новые функции и компоненты без изменения существующего кода. Это особенно важно для моего проекта «Кликер по контролю за сном», где может потребоваться добавление новых функций в будущем. Так же эта архитектура была выбрана мной из-за высокой ее востребованностью на рынке и частом использовании в крупных компаниях. Изучения и работа с ней будет очень полезна для моей дальнейшей карьеры

.

## Оптимизация управления состояниями на главном экране с

## использованием Diffable Data Sourse.

В разработке приложения ключевая задача — обеспечить эффективное управление динамическим состоянием персонажей на главном экране. Это необходимо для того, чтобы пользовательский интерфейс был отзывчивым и актуальным, отображая изменения уровней усталости персонажей в реальном времени.

Управление динамическими состояниями важно для поддержания согласованности данных приложения с его интерфейсом, что повышает пользовательский опыт и обеспечивает точность отображаемой информации.

В ходе поиска решения были рассмотрены и проанализированы стандартное обновление «UICollectionView» и новый представленный в iOS 13 подход — «Diffable Data Source», который обеспечивает упрощенное и более надежное управление данными в

«UICollectionView`»

Выбор Diffable Data Source для управления изменениями состояния персонажей в приложении позволяет автоматически и эффективно производить обновления данных, минимизируя необходимость сложной логики обработки и ручного обновления ячеек. Этот подход облегчает работу с часто меняющимися состояниями персонажей, требующими регулярных обновлений в интерфейсе, и предотвращает возможное несоответствие данных.

С использованием UICollectionViewDiffableDataSource и NSDiffableDataSourceSnapshot и применением данных в UICollectionView, это обновляет интерфейс с плавными анимациями и в синхронизации с жизненным циклом приложения. Отображение актуального индикатора усталости каждого персонажа на главном экране.

## Выделение часто используемых элементов интерфейса

**Описание задачи:** Разработать интерфейс приложения таким образом, чтобы повысить эффективность разработки и обеспечить удобство в дальнейшем добавлении функционала, при этом избегая дублирования кода.

**Варианты выполнения задачи:** Один из возможных путей – использование переиспользуемых компонентов для UI элементов, которые часто повторяются, например, кнопок в стиле облаков. Это подразумевает создание специализированных классов, которые могут быть настроены и использованы в различных частях приложения без необходимости дублирования кода.

**Выбор решения:** Решено было разработать два основных класса: CloudButtonView для индивидуальных кнопок и CloudStackView для управления группами таких кнопок. CloudButtonView позволяет настраивать внешний вид и поведение кнопок, в то время как CloudStackView облегчает их расположение и управление пространством между кнопками.

**Реализация:** В классе CloudButtonView были определены стили для кнопок, включая их визуальные аспекты и поведение при нажатии. В CloudStackView реализован механизм управления макетом, позволяющий размещать несколько CloudButtonView с заданным расстоянием и выравниванием. Это обеспечивает консистентность дизайна и упрощает модификацию интерфейса при добавлении новых функций.

**Результат:** Внедрение `CloudButtonView` и `CloudStackView` позволило стандартизировать элементы интерфейса, ускорить разработку экранов и облегчить процесс добавления нового функционала. Такой подход снизил дублирование кода и повысил масштабируемость проекта, делая его более управляемым и гибким в разработке. Это улучшило консистенцию интерфейса и облегчило расширение приложения в будущем..

## Внедрение таймера в правильном режиме для корректной интеграции с основным режимом

В моем приложении "Кликер по контролю за сном" таймер необходим для автоматического и периодического обновления индикаторов энергии персонажей, отражающих их уровень усталости и бодрствования. Это обеспечивает динамичное изменение состояния игры без постоянного вмешательства пользователя и поддерживает ощущение реального времени.

Таймер выбран как средство для циклической проверки и обновления данных, так как он позволяет планировать повторяющиеся действия с заданным интервалом. Это идеальное решение для задач, требующих регулярности и точности, как в случае с мониторингом энергии персонажей в моем приложении это один из основных элементов игрового процесса

Используется «Timer» класса «Foundation», так как он хорошо интегрируется с основным циклом. Это позволяет таймеру работать в гармонии с пользовательским интерфейсом и другими событиями приложения, обеспечивая обновление UI без сбоев и зависаний.

Для поддержания живой динамики игры и отслеживания ритма жизни каждого персонажа в "Кликере по контролю за сном" реализуется механизм таймера, синхронизированный с индивидуальными графиками сна и бодрствования персонажей. При достижении времени, достаточного для изменения статуса бодрости, таймер активирует метод, который адаптирует индикаторы энергии персонажа. Это имитирует естественный цикл активности и отдыха, внося в игру элемент реализма и повышая её интерактивность.

Таймер является критическим компонентом для поддержания актуальности игрового процесса и предоставления пользователям постоянно обновляемого и интерактивного игрового статуса персонажа.

## Выбор оптимального стека технологий

Перед началом разработки приложения выбор фреймворка являлся одной из ключевой задач. Выбор был между фреймворками: UIKit, SwiftUI, Sprite Kit.

UIKit для разработки игры "Кликер по контролю за сном" обусловлен рядом стратегических соображений. UIKit был выбран из-за своей проверенной надежности и глубоких возможностей кастомизации пользовательского интерфейса, обеспечивая четкость и последовательность в дизайне, что необходимо для интуитивно понятного геймплея. Несмотря на простоту, которую предлагает SwiftUI, необходимость детализированного управления UI определила выбор в пользу UIKit.

Дополнительно, UIKit выделяется возможностями точной настройки взаимодействия элементов интерфейса с пользовательскими жестами, что является ключевым для создания захватывающего игрового опыта. Включение расширенных компонентов, таких как StackView, значительно упрощает проектирование сложной макетной структуры, позволяя при этом сохранять четкость и последовательность в визуальном представлении игры.

Sprite Kit также рассматривался как возможный кандидат благодаря его специализации на 2D-играх и встроенным игровым циклам, что упрощает создание динамичной графики.

Однако, учитывая, что "Кликер по контролю за сном" не требует сложной двумерной графики, основной упор был сделан на UX/UI, а не на игровую механику, где Sprite Kit мог бы проявить свои наилучшие качества. Витоге, UIKit сочетает в себе необходимые характеристики для настоящего проекта: богатые возможности по созданию интерфейса и поддержку со стороны Apple и сообщества разработчиков.

# Описание и обоснование алгоритма работы программы

# Описание алгоритма работы программы

# Подробный интерфейс программы со всеми экранами располагается по этой ссылке: <https://www.figma.com/file/zt0vpJrVBPzloGKqlTO7Ha/Mobile-Game-Sleep-Clicker?type=design&node-id=0%3A1&mode=dev&t=t1umVSOC3XucNre6-1>

При первом запуске мобильного приложения пользователя встречают экраны ознакомления (OnBoardingViewController), которые предоставляют краткое введение в игровой процесс. Для навигации по этим экранам(DemoViewController) можно использовать жесты «свайпа» или просмотреть, нажимая на кнопку “Next” в правом верхнем углу экрана. Если пользователь желает не просматривать ознакомительные экраны, он может воспользоваться кнопкой “Skip” в левом верхнем углу экрана. Ознакомительный режим предусмотрен для отображения только при первом запуске приложения, после чего он больше не появляется.

После прохождения ознакомительного режима открывается главный экран приложения (MainViewController).

На игровом экране присутствует список активных персонажей, с которыми игрок может взаимодействовать. Каждый персонаж находится в своей ячейке-комнате (HeroCell) может находиться в одном из двух состояний: он может быть бодрствующим или спящим. Задача игрока — внимательно следить за уровнем усталости своих персонажей (для этого под картинкой персонажа расположена батарейка), чтобы в оптимальное время переключать их между состояниями, обеспечивая эффективное накопление валюты. Активное участие и правильное тайминг в вопросе менеджмента персонажей приводят к увеличению заработка игровой валюты, что, в свою очередь, открывает новые возможности для развития и прогресса в игре.

В правой верхней части экрана расположены кнопки перехода на другие экраны: магазин (SleepMobApp//Screens//Shop) и экран настроек (SleepMobApp//Screens//Settings) и. При нажатии одной из этих кнопок осуществляется переход на соответствующую страницу. Для кнопки с изображением шестерёнки — это страница настроек, домика —страница магазина.

Экран «Настройки» (SettingsViewController) имеет заголовок «Settings» На нем расположены кнопки-облака для изменения настроек приложения (CloudStackView) При нажатии на кнопку с надписью «sound» пользователь может включить или полностью отключить звуковые эффекты в приложении.

При нажатии на кнопку с надписью «notification» пользователь может включить или отключить локальные уведомления, которые будут напоминать ему о необходимости вернуться в игровой процесс.

При нажатии на кнопку с надписью «by dev» пользователю открывается экран с информацией и разработчике.

Так же в правом верхнем углу расположена кнопка «back», при её нажатии пользователь будет возвращён на предыдущий экран, с которого он перешёл к текущим настройкам.

Из главного экрана так же возможен переход в экраны Магазина (SleepMobApp//Screens//Shop)

В котором возможно переключение между экранами «Магазин персонажей» (ShopCharacterViewController) и «Магазин дополнительных опций» (ShopBoostViewController)

# Обоснование алгоритма работы программы

# Для мобильного приложения-кликера, цель которого – мониторинг и контроль сна персонажей с возможностью изменения их состояния (сон или бодрствование), ключевым является предоставление пользователям простого и интуитивно понятного интерфейса. Это достигается за счет минималистичного дизайна и быстрого доступа к основным функциям приложения. На главной странице находятся все необходимые элементы для навигации, что обеспечивает легкость использования и эффективность взаимодействия с приложением.

# Описание и обоснование выбора способа организации входных и выходных данных

# 3.4.1 Входные данные

Для работы интерфейса приложение принимает на вход данные, отправляемые с нажатием пользователя на экран.

# 3.4.2 Выходные данные

Выходными данными во время игрового процесса является фоновая музыка в приложении.

Выходными данными программы также являются графические элементы приложения: персонажи, валюта, комнаты-ячейки, элементы одежды и предметы дополнительных опций.

# Описание архитектуры

На изображении представлена структура проекта мобильного приложения, организованная с использованием архитектуры Clean Swift (VIP). Каждый экран приложения имеет собственную папку в разделе "Screens", что обеспечивает чёткое разделение ответственности и упрощает масштабирование проекта.

Внутри каждой папки экрана находятся компоненты VIP-цикла:

|  |  |
| --- | --- |
| ViewController.swift | Отвечает за взаимодействие с пользователем. Методы контроллера активируются при нажатии на элементы пользовательского интерфейса (UI). |
| Interactor.swift | содержит бизнес-логику приложения, обработку запросов пользователя и взаимодействует с Workers, которые выполняют работу с сетью, базами данных и модулями устройства. |
| Presenter.swift | Подготавливает полученные данные для отображения, форматируя их в соответствующем виде для ViewController. |
| Router.swift | Управляет переходами между сценами и передачей данных, обеспечивая навигацию внутри приложения |
| Models | Включает структуры для передачи данных между слоями. В нём определены Request, Response и ViewModel. |
| Request | Cодержит данные, введённые пользователем, и передаётся из ViewController в Interactor. |
| ViewModel | Cодержит данные, обработанные и форматированные Presenter для их отображения обратно в ViewController. |

Эта структура обеспечивает чистоту кода и его легкость в поддержке, а также высокую тестируемость и модульность приложения. Каждый компонент выполняет свою отдельную функцию, что позволяет разработчикам легко находить и устранять ошибки, добавлять новые функции, а также обеспечивает возможность независимой работы над разными частями проекта в команде.

# 4 ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

# Ориентировочная экономическая эффективность

Курсовой проект «Мобильное приложение кликер по контролю за сном» не предусматривал расчетов экономических показателей.

# Предполагаемая потребность

Предположительно, конечными пользователями могут быть, как дети школьного возраста, так и взрослые люди, увлеченные кликерами. Расширение геймплея, в виде доступности к покупке новых персонажей, должно способствовать расширению аудитории и поддержанию ее интереса к игре.

# Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами

# 

# Сравнение с аналогами представлено в Таблице 2 – Сравнение с аналогами

|  |  |
| --- | --- |
| Аналог | Сравнение |
| «Tap tap ants» | Минусы: нет разнообразной смены персонажейПлюсы: наличие монетизация и звуковых эффектов в приложении |
| «Pianо Tiles» | Минусы: отсутствие различных персонажей; полное отключение звуковых эффектов может сделать игру неинтереснойПлюсы: широкий выбор музыкальных треков, включая классические произведения, современные популярные песни и оригинальные композициии з |
| «Мусорный магнат: эко симулятор» | Минусы: отсутвие обучения в игре, непонятный и сложный интерфейс взаимодействияПлюсы: интересный дизайн и наличие push-уведомлений для захода в приложение |
| **«Adventure capitalist»** | **Плюсы:**  Регулярные тематические обновления, к различным праздникам  **Минусы:**  Нестабильная работа на определенных устройствах на iOS |

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов.

//Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

1. ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 15150-69 Машины, приборы и другие технические изделия. Исполнения для различных климатических районов. Категории, условия эксплуатации, хранения и транспортирования в части воздействия климатических факторов внешней среды. – М.: Изд-во стандартов, 1997.
9. ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10. Gaurav Vaish High Performance iOS Apps: Optimize your code for better apps, 2016.
11. Joe Conway, Aaron Hillegass iOS Programming: The Big Nerd Ranch Guide, 2016.
12. Clean Swift [Электронный ресурс] // Режим доступа: https://clean-

swift.com, свободный. (дата обращения: 10.02.2024).

1. Apple Developer [Электронный ресурс] // Режим доступа:

https://developer.apple.com, свободный. (дата обращения: 10.02.2024).

# ПРИЛОЖЕНИЕ 1

**ТЕРМИНОЛОГИЯ**

Таблица 1

|  |  |
| --- | --- |
| **Термин** | **Определение** |
| Свайп | Жест, который используется для выполнения команд, таких как прокрутка, переход к следующему или предыдущему элементу, или для выполнения функций, таких как разблокировка экрана или удаление уведомления. Этот жест стал основным средством навигации в современных сенсорных интерфейсах. |

# ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий  № сопроводит ельного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулирован ных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |