**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 7](#_Toc33617548)

[**1 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ** 9](#_Toc33617549)

[**СРЕДСТВУ И ПОСТАНОВКА ЗАДАЧ** 9](#_Toc33617550)

[**1.2** **Обзор существующих программных средств по теме** 10](#_Toc33617551)

[**дипломного проекта** 10](#_Toc33617552)

[**1.3 Обоснование выбора языка программирования и средств** 11](#_Toc33617553)

[**разработки** 11](#_Toc33617554)

[**1.4 Постановка задач по разработке программного средства** 19](#_Toc33617555)

[**2 РАЗРАБОТКА ИГРОВОГО АРКАДНОГО** 20](#_Toc33617556)

[**ПРОГРАММНОГО СРЕДСТВА** 20](#_Toc33617557)

[**2.1 Разработка архитектуры программного средства** 20](#_Toc33617558)

[**2.2 Разработка объектной модели программного средства** 22](#_Toc33617559)

[**2.3 Разработка алгоритмов функционирования программного** 25](#_Toc33617560)

[**средства** 25](#_Toc33617561)

[**2.4 Разработка и обоснование пользовательского интерфейса** 38](#_Toc33617562)

[**программного средства** 38](#_Toc33617563)

[**3 ИНЖЕНЕРНЫЕ РАСЧЕТЫ, ИСПОЛЬЗУЕМЫЕ** 42](#_Toc33617564)

[**В ПРОГРАММНОМ СРЕДСТВЕ** 42](#_Toc33617565)

[**3.1 Оценка файловой структуры игрового программного средства** 42](#_Toc33617566)

[**3.2 Анализ памяти, используемой программным средством** 45](#_Toc33617567)

[**4 ЭКСПЛУАТАЦИЯ ПРОГРАММНОГО СРЕДСТВА** 49](#_Toc33617568)

[**4.1 Ввод в эксплуатацию программного средства** 49](#_Toc33617569)

[**4.2 Руководство к пользованию разработанным программным** 51](#_Toc33617570)

[**средством** 51](#_Toc33617571)

[**5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ** 53](#_Toc33617572)

[**РАЗРАБОТКИ ИГРОВОГО АРКАДНОГО** 53](#_Toc33617573)

[**ПРОГРАММНОГО СРЕДСТВА «RUN OUT» ПОД** 53](#_Toc33617574)

[**ОПЕРАЦИОННУЮ СИСТЕМУ ANDROID** 53](#_Toc33617575)

[**5.1 Краткая характеристика программного продукта** 53](#_Toc33617576)

[**5.2 Расчет затрат на разработку и отпускной цены программного** 53](#_Toc33617577)

[**продукта** 53](#_Toc33617578)

[**5.3 Расчет экономической эффективности реализации на рынке** 56](#_Toc33617579)

[**игрового аркадного программного средства «Run out»** 56](#_Toc33617580)

[**под операционную систему Android** 56](#_Toc33617581)

[**ЗАКЛЮЧЕНИЕ** 59](#_Toc33617582)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 59](#_Toc33617583)

[**ПРИЛОЖЕНИЕ А** 61](#_Toc33617584)

[**ПРИЛОЖЕНИЕ** **Б (обязательное)** 62](#_Toc33617585)

[**ПРИЛОЖЕНИЕ В (обязательное)** 63](#_Toc33617586)

# **ВВЕДЕНИЕ**

На сегодняшний день такое заболевание, как сахарный диабет, приобретает все большее и большее распространение. Это обусловлено изменением стиля жизни современного человека, уменьшением подвижности его тела в связи с преобладанием умственного труда над физическим, потреблением нездоровой пищи и развитием других вредных привычек.

Диабет — это хроническая болезнь, развивающаяся в тех случаях, когда поджелудочная железа не вырабатывает достаточно инсулина или когда организм не может эффективно использовать вырабатываемый им инсулин. Инсулин — это гормон, регулирующий уровень содержания сахара в крови. Общим результатом неконтролируемого диабета является гипергликемия, или повышенный уровень содержания сахара в крови, что со временем приводит к серьезному повреждению многих систем организма, особенно нервов и кровеносных сосудов. [1] Число людей с диабетом возросло со 108 миллионов в 1980 году до 422 миллионов в 2014 году. Наблюдая такую динамику распространения болезни, невозможно легкомысленно относиться к этому явлению.

За последние 10 лет смартфоны прочно закрепились в повседневной жизни людей, так как эти устройства поддерживают огромное количество функций и заменили собой множество других гаджетов, которые ранее приходилось приобретать отдельно, такие как: калькулятор, камера, *mp3*-плеер и множество других устройств. Современные смартфоны способны заменить даже персональный компьютер. Такую универсальность смартфоны имеют благодаря наличию различных приложений и возможности расширять функционал устройства благодаря установке дополнительных программ. Если принять во внимание этот момент, то хорошим решением является использование смартфона и в оздоровительных целях, а именно использование устройства в качестве трекера здоровья, куда человек будет заносить данные о своем здоровье и, анализируя предоставляемую приложением статистику, корректировать свой образ жизни для того, чтобы избегать причин возникновения заболевания.

Объектом исследования являются существующие варианты приложений для мобильных устройств, позволяющие пользователю отслеживать свою статистику здоровья.

Предметом исследования являются способы улучшения качества наблюдения человека, болеющего сахарным диабетом, за своим состоянием здоровья.

Целью данной работы является разработка мобильного приложения для операционной системы *Android*, которое позволит пользователю вести учет потребляемых им лекарственных средств, калорий, количества времени физической активности, уровня глюкозы в крови. Также приложение должно иметь функции напоминания пользователю о времени принятия необходимых лекарственных средств, сигнализирования пользователю о превышении или занижении нормы потребляемых микроэлементов и уровня глюкозы в крови. Также для того, чтобы в процесс наблюдения здоровья было легче вовлечь детей, приложение должно иметь игровую форму с вознаграждением пользователя за постоянный ввод данных.

Для выполнения приведённых целей необходимо создать *RESTful API* серверное приложение с подключением к базе данных для обеспечения возможности получения клиентским приложением данных. Далее реализовать клиентское приложение с удобным пользовательским интерфейсом, которое бы обращалось к серверу за данными, получало их, сохраняло в памяти смартфона и предоставляло возможности для выполнения описанных выше функций.

Мобильное приложение — программное обеспечение, предназначенное для работы на смартфонах, планшетах и других мобильных устройствах. Многие мобильные приложения предустановлены на самом устройстве или могут быть загружены на него из онлайн магазинов приложений, таких как *App Store, Google Play, Samsung Store* и других [2].

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет ИКС%. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников».

Скриншот приведен в приложении А.

# **1 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ**

# **СРЕДСТВУ И ПОСТАНОВКА ЗАДАЧ**

* 1. **Анализ и описание функциональных возможностей**

**программного средства**

Основными функциями разрабатываемого в данном проекте приложения является система учета данных о состоянии здоровья пользователя, сбор статистических данных, вносимых им, осуществление напоминаний пользователю о необходимости принятия лекарственных средств, отображение среднестатистических данных на основе информации, введенной пользователем, выдача подробной детализированной информации, которая была внесена в течение определенного промежутка времени. Также важной функцией является создание учетной записи пользователя с привязкой к адресу электронной почты и номеру телефона, что позволит пользователю осуществлять доступ к приложению на более высоком уровне безопасности. В случае утери пароля пользователь сможет продолжать осуществлять доступ к приложению посредством привязанного номера телефона.

Анализируя существующие аналоги разрабатываемого приложения, можно сделать вывод о том, что вышеописанные функции присутствуют во всех похожих приложениях в том или ином виде. Все рассмотренные аналоги имеют различный интерфейс, но практически идентичны по функциям. Каждое приложение имеет функции сбора данных, напоминания, оценку состояния пользователя на основе введенных данных и т.д.

Все вышеупомянутые функции безусловно очень нужны и приносят большую пользу людям, но если немного проанализировать потенциальную аудиторию пользователей такого рода приложений, то можно обнаружить, что большую долю составляют люди подросткового и даже детского возраста. По данным Международной диабетической федерации на 2016 год, только в Европе проживает более 140 тысяч детей в возрасте до 14 лет, имеющих диабет первого типа. [3] Существующие приложения учета здоровья имеют интерфейс и логику, понятные для взрослого человека. Но далеко не каждые ребенок или подросток сможет разобраться с тем, как пользоваться такого рода приложением. Поэтому в разрабатываемом в данном проекте приложении было решено добавить игровую логику с виртуальным персонажем, взаимодействуя с которым, пользователю будет проще понять суть приложения и выполнять необходимые действия. Было решено создать функцию чат-бота для симуляции общения пользователя и виртуального персонажа, а также систему вознаграждения пользователя за ввод данных.

Приложение, которое должно получиться в результате выполнения дипломного проекта на данный момент не имеет аналогов в магазинах приложений.

## **Обзор существующих программных средств по теме**

## **дипломного проекта**

Если начать поиск похожих программных средств по теме дипломного проекта в магазинах приложений, то можно найти достаточно большое количество продуктов, которые очень схожи по своим функциям. Рассмотрим некоторые из них.

Первое из обнаруженных приложений – «*Diabetes:M*». Скриншот одного из экранов можно видеть ниже на рисунке 1.1.



Рисунок 1.1 – Скриншот приложения «*Diabetes:M*»

Данное приложение имеет интеграцию с сервисом *Google Fit*, который отслеживает данные о здоровье пользователя со смартфона и других носимых устройств. Также можно вводить в нее лабораторные исследования крови и мочи, записывать давление, задавать значения на занятия спортом и данные о болезни. Данное приложение обладает следующими функциями:

* калькулятор. Программа рассчитывает ту дозу инсулина, которую необходимо ввести, опираясь на заданные ей значения
* расчет активного инсулина в организме
* дневник питания
* наглядная визуализация данных дневника в виде графиков и диаграмм за период от недели до трех месяцев.

Рассмотрим другое приложение – «*DiaMeter*». Это не менее удобное приложение для ведения записей об уровне сахара, количестве съеденных вами хлебных единиц, инъекциях короткого и продлённого инсулина, а также о самочувствии в целом. Есть наглядная статистика и возможность синхронизации всех данных в облаке. Дополнительно DiaMeter предлагает ряд интерактивных статей, в которых можно найти немало полезной информации о правильном питании при диабете, физических нагрузках, влиянии алкоголя и другом. Скриншот данного приложения изображен на рисунке 1.2.

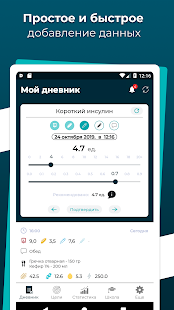


Рисунок 1.2 – Скриншот приложения «*DiaMeter*»

Оба приложения, рассмотренные выше, очень хороши, удобны и полезны для взрослого пользователя, который осознает свое заболевание и достаточно замотивирован в лечении и наблюдении за своим состоянием здоровья. Однако рассмотренные приложения будут очень сложны в восприятии более юной аудиторией.

## **1.3 Обоснование выбора языка программирования и средств**

## **разработки**

При разработке сложных программных средств под мобильные опера-ционные системы следует с особым вниманием подходить к выбору средств разработки, так как от этого зависит скорость и качество процесса создания программного средства.

Для выбора подходящей среды разработки необходимо учесть ряд опре-делённых факторов, которые могут оказать сильное влияние на разработку:

– интерфейс. Этот первый компонент, с которым сталкивается пользо-ватель после установки и который формирует первое впечатление о про-грамме, и, на основании которого, может быть осуществлен окончательный выбор. Здесь оценивается не только общий дизайн, хотя, безусловно, он тоже сказывается определенным образом, но и удобство расположения и настройки таких компонент как окно исходного кода, окно проекта;

– настройка. Соответственно после установки и первого запуска среды разработки выполняется ее настройка, т.е. указываются пути, по которым рас-полагаются установленные *SDK*, *DOCS*, *J*2*EE*. В этом компоненте, как пра-вило, существенных различий не наблюдается. Более того, современные сред-ства, как правило, самостоятельно определяют установленные компоненты;

– редактор кода. Настройка отображения исходных кодов, как правило, тоже не отличается разнообразием, в любом средстве легко можно настроить кеглю и ее размер, а также цвет. Немаловажным преимуществом является наличие помощника, когда, при «зависании» мышки на любой переменной или методе всплывает довольно подробный *ToolTip* (контекстное окно) в ко-тором развернуто, описаны все параметры объекта. Также есть масса прият-ных мелочей, вывод нумерации строк, отображение структуры класса, показ символов абзаца, проверка орфографии.

На сегодняшний день среди мобильных операционных систем получили наибольшее распространение *Android* и *iOS*. Большинство мобильных устройств работают под одной из данных систем. Согласно мировой статистике [4], 74.3% пользователей смартфонов используют операционную систему *Android* и 24.7% – *iOS*. Благодаря тому, что преобладает всего две операционные системы, широкое распространение сегодня получают различные кроссплатформенные фреймворки, позволяющие из одной и той же кодовой базы создать одновременно приложение как под одну операционную систему, так и под другую. Использование такого подхода позволяет сэкономить не только временные ресурсы, но также и финансовые. Производительность таких кроссплатформенных приложений очень зависит от выбранной технологии. Лучшие технологии для кроссплатформенной разработки на сегодняшний день мало чем уступают нативным приложениям в производительности. Исходя из этого было решено для разработки приложения данного дипломного проекта использовать кроссплатформенный фреймворк.

Одним из наиболее популярных фреймворков для разработки мобильных кроссплатформенных приложений является *React Native* [5]. *React Native* – фреймворк с открытым исходным кодом, разработанный для создания кроссплатформенных мобильных приложений для операционных систем *iOS* и *Android*. Данная платформа использует язык *JavaScript* для создания приложений. Структура платформы отображена ниже на рисунке 1.3.

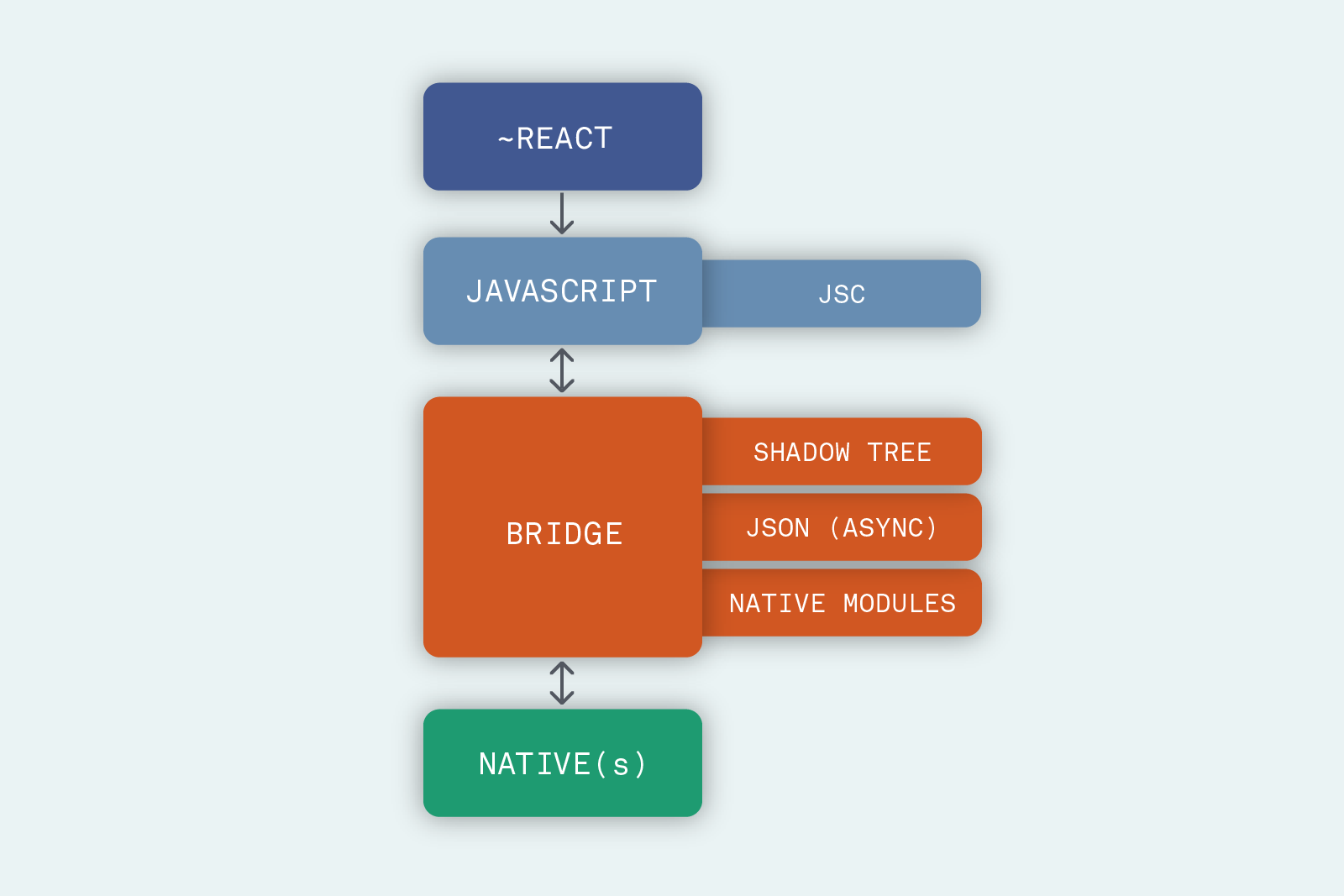


Рисунок 1.3 – Структура платформы *React Native* [5]

*React Native* использует библиотеку *React* в качестве основы для написания исходного кода приложения. Код, написанный с использованием этой библиотеки компилируется в один файл, так называемый *bundle*.

Также на рисунке можно видеть, что собранный *JavaScript*-код взаимодействует с «нативной» платформой посредством моста, который содержит в себе всю логику работы фреймворка.

Приложение, разработанное с использованием *React Native*, очень сложно отличить от разработанного с использованием родных платформе инструментов. Это обусловлено строением архитектуры данного фреймворка и его работы. Полностью готовое *React Native* приложение представляет собой «нативное» *Android* или *iOS* приложение, которое при открытии передает управление *JavaScript*-потоку, содержащему всю бизнес-логику приложения, а также пользовательский интерфейс. Данный поток может управлять работой родных для мобильной платформы компонентов. Например, открыть модальное окно, добавить *Drawer*-меню и так далее. Фреймворк также поддерживает обработку специфических для каждой платформы жестов и анимации.



Рисунок 1.4 – Пример программного кода

С точки зрения разработчика, пишущего непосредственно код приложения, *React Native* практически не отличается от библиотеки *React*. Эта библиотека является отличным примером реактивного программирования, ориентированного на реакцию к определенным событиям. Также особенностью *React* и *React Native* является *JSX*-разметка компонентов, используемая при построении пользовательского интерфейса. Данная разметка похожа на разметку *html*-документа, что облегчает освоение данных платформ разработчиками, которые имели опыт работы в *web*-разработке. На рисунке 1.4 отображен пример программного кода, написанного при использовании платформы *React Native*.

В сущности хакерской культуры Facebook, React Native начинался как проект хакатона летом 2013 года. Подобно React , React Native казался смелой нетрадиционной идеей. Не было ясно, что это действительно сработает. Как работали сенсорные переговоры между JS и собственными ScrollViews? Как насчет производительности, а как насчет отладки? Ни одна из этих проблем не помешала инженерам сосредоточиться и двигаться вперед.

Менеджер рекламы Facebook для iOS вышел в феврале 2015 года, менее чем через шесть месяцев после того, как команда разработчиков начала работать над ним. Одновременно все, кто сосредоточился на JS или iOS, работали над открытым исходным кодом реализации iOS. В январе 2015 года на React.js Conf состоялся первый публичный предварительный просмотр , а на конференции разработчиков F8 в марте 2015 года был открыт код для всех .

Сразу после этого инженеры по продукту Ads Manager начали переносить свой код JavaScript на Android, тесно сотрудничая с командой React Native Android в Лондоне. Facebook не стремился к совместному использованию кода между платформами при работе с iOS Ads Manager, но ожидалось, что это может быть хорошим побочным эффектом использования React Native. Когда Android Ads Manager был готов к поставке было понятно, что около 85 процентов кода было поделено между двумя приложениями.

В июне 2015 года, после трех месяцев разработки и месяца внутренних собеседований, была выпущена первая версия менеджера рекламы на Android . Учитывая, как взлетела React Native для iOS, Facebook сразу же переключили наше внимание на открытый исходный код React Native для Android, и ожидали большой интерес к выпуску Android. В конце концов, необходимость создавать одно и то же приложение отдельно для каждой платформы - проблема в отрасли, и это было известно из опыта работы с Менеджером рекламы, что React Native может помочь с этим. [6]

Подобно запуску iOS, Facebook хотел выпустить React Native для Android как можно раньше, чтобы получить обратную связь. Поэтому они начали только с небольшого количества представлений и модулей (Text, Image, ScrollView, Network, AsyncStorage и некоторых других). 14 сентября было выдвинуто ядро ​​среды выполнения Android и начальный набор модулей Android на GitHub и npm. И именно так React Native версии 0.11 был первым выпуском с поддержкой Android. Были добавлены следующие модули Android с тех пор, как открыли исходный код, приблизив паритет API к iOS: оповещения, AppState, CameraRoll, буфер обмена, указатели даты и времени, геолокация, намерение, модальные, NetInfo, извлечение для обновления представления, средство выбора , слайдер, просмотр пейджера и веб-просмотр.

Само собой разумеется, что принятие, которое последовало за пределами Facebook, очень вдохновило на дальнейшую работу в команде React Native.

На текущий момент последней стабильной версией React Native является 0.61.5 и с момента первого релиза изменилось многое. Но одно можно утверждать точно – React Native развивается и учитывает свои прошлые ошибки двигаясь вперед.

Платформа *React Native* обладает достаточно большим количеством достоинств. Благодаря существованию этих достоинств некоторые популярные приложения, таких как *Instagram, Airbnb, Walmart, UberEats, Tesla, Skype* и другие, уже активно используют этот фреймворк. *React Native* предоставляет все основные и передовые методы разработки гибридных мобильных приложений. Перейдем к рассмотрению некоторых причин, по которым использование *React Native* для разработки мобильных приложений является хорошим решением. [7]

**1.3.1** Достоинства *React Native*

**1.3.1.1** Единая кодовая база для *Android* и *iOS*

*React Native* позволяет писать код один раз и запускать как на *Android*, так и на *iOS*. Кодовая база будет одинаковой для обеих платформ, поэтому вам не придется тратить деньги и время на два проекта разработки.

Поскольку *React Native* имеет открытый исходный код и способствует повторному использованию, вы можете повторно использовать компоненты в любое время на любом уровне, не переписывая его и не перекомпилируя приложение.

**1.3.1.2** Возможность повторного использования

Теперь поговорим о возможности повторного использования. *React Native* приносит блоки, которые состоят из повторно используемых «нативных компонентов». Почти все компоненты *Android* и *iOS* имеют свои аналоги в *React Native*. Все эти компоненты компилируются непосредственно в нативные приложения и, таким образом, позволяют разработчикам поддерживать внешний вид платформы.

Ранее в гибридных приложениях было невозможно получить структуру, специфичную для компонента. Но с *React Native* эта структура позволяет разработчикам создавать приложения, используя гибкие веб-методологии.

С другой стороны, *React Native* также позволяет разработчикам добавлять компоненты *«native»* в код своих существующих приложений. Им не придется кодировать с нуля, чтобы преобразовать существующее мобильное приложение в приложение *React Native*.

**1.3.1.3** Производительность

*React Native* сокращает цикл разработки, позволяет разрабатывать приложения очень быстро и позволяет доставлять приложения максимально быстро. Он использует библиотеку *ReactJS UI*, разработанную *Facebook* для пользовательских интерфейсов. В то время как собственные приложения работают на центральном процессоре, *React Native* использует графический процессор.

Он поставляется с уникальной функцией под названием «живая перезагрузка», которая позволяет просматривать последние изменения кода сразу, разбивая экран на две части. Первый показывает код, а второй показывает результат кода в кадре мобильного телефона. В целом, с помощью *React Native* компании могут сократить расходы на разработку почти на 50% без ущерба для качества или производительности.

**1.3.1.4** Ориентация на пользовательский интерфейс

*React Native* – полностью ориентированный на пользовательский интерфейс. Декларативный *API* значительно упрощает понимание ваших требований и прогнозирование пользовательского интерфейса. Декларативный стиль позволяет вам контролировать поток и состояние в вашем приложении, говоря: «Это должно выглядеть так». Вам больше не придется беспокоиться о деталях реализации.

**1.3.1.5** Совместимость с сторонними плагинами

Добавление аппаратных возможностей устройства в приложение является одним из наиболее распространенных требований в наши дни. В отличие от функций *WebView, React Native* позволяет напрямую связать плагин с собственным модулем через инфраструктуру. Это приводит к более плавному запуску приложения, более быстрой загрузке и меньшим требованиям к памяти.

**1.3.2** Сравнение *React Native* с другими платформами

Для разработки приложения был выбран кросс-платформенный фреймворк *React Native*. По этому для сравнения с данной технологией, был выбран *Flutter*.

***Flutter*** — *SDK* с открытым исходным кодом для создания мобильных приложений от компании *Google*. Он используется для разработки приложений под *Android* и *IOS*, а также это пока единственный способ разработки приложений под *Google Fuchsia*. [8]

*React Native* создан для приложений на *IOS*, *Android* и *Windows*, при этом использует язык программирования *JavaScript* и библиотеку *React.js* как главное средство разработки. Фреймворк был создан и поддерживается компанией *Facebook*, которые разработали *React.js*.

*Flutter* в свою очередь создан для приложений *Android*, *IOS* и *Fuchsia*, в отличии от *React Native*, для разработки используется язык программирования *Dart*, который также служит для веб-программирования.

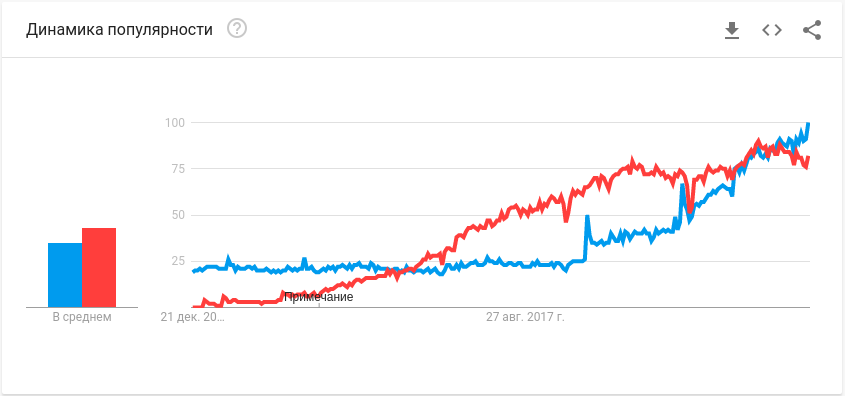


Рисунок 1.5 – График популярности *React Native* и *Flutter* [9]

На рисунке 1.5 приведен график популярности *React Native* (красный) и *Flutter* (синий), согласно статистике *Google Trends.*

Для сравнения *React Native* и *Flutter* компанией *SimbirSoft* была создана таблица в зависимости от характера приложения (рис. 1.6).

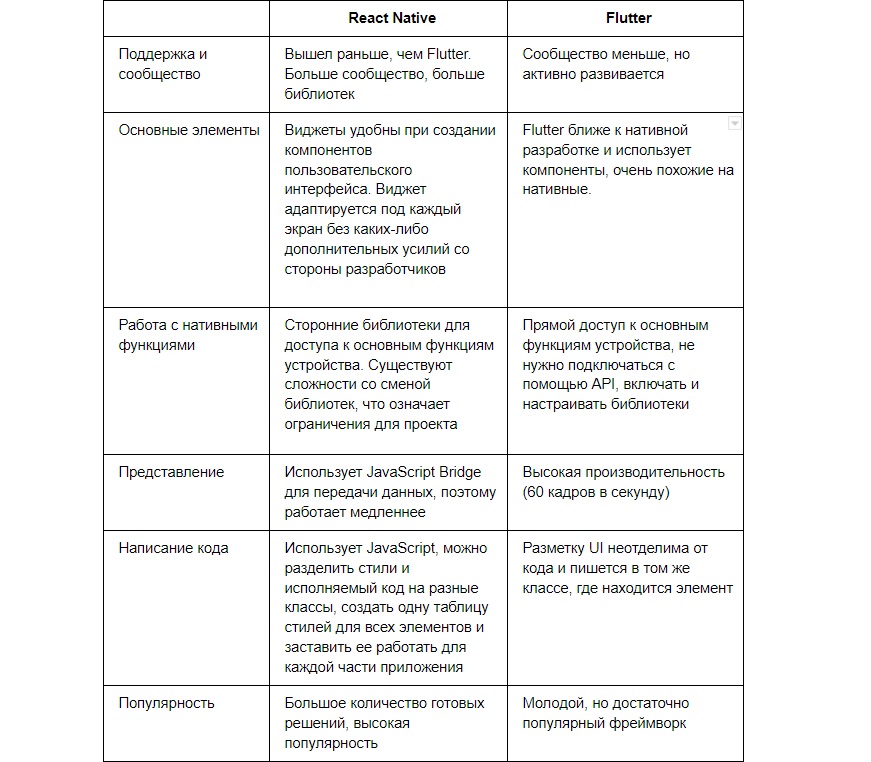


Рисунок 1.6 – Сравнение *React Native* и *Flutter* [10]

Опираясь на выше перечисленные критерии для разработки небольшого приложения с использованием сторонних библиотек хорошим выборам будет *React Native* из-за своей простоты и популярности.

В качестве основной среды разработки была выбрана *WebStorm* [11] от компании *JetBrains.* Данная интегрированная среда разработки является лучшей для написания кода на языках *JavaScript* и *TypeScript* на сегодняшний день.

Так как в теме дипломного проекта обозначена разработка под систему *Android*, то для разработки приложения также будет необходима среда *Android Studio. Android Studio* это официальная интегрированная среда разработки(*IDE*) для разработки *Android*-приложений, основанная среде разработки *IntelliJ IDEA* [12].

## **1.4 Постановка задач по разработке программного средства**

Разрабатываемое приложение должно иметь удобный, интуитивно по-нятный интерфейс, в котором самые важные функции находятся в ближайшей доступности, а более глубокие возможности не должны перегружать пользо-вательский интерфейс.

В результате анализа аналогов на рынке приложений и предметной области сформирован следующий список требований к разрабатываемой системе:

* возможность авторизации в системе;
* возможность просмотра профиля пользователя;
* возможность внесения данных о потребленных пользователем калориях, уровне глюкозы в крови, времени физической активности;
* возможность устанавливать получение уведомлений с напоминанием о принятии лекарственных средств;
* возможность получения пользователем статистики своих данных;
* реализация игровой логики и системы вознаграждения пользователя;
* реализация чат-бота.

# **2 РАЗРАБОТКА ИГРОВОГО АРКАДНОГО**

# **ПРОГРАММНОГО СРЕДСТВА**

## **2.1 Разработка архитектуры программного средства**

Когда говорится об архитектуре игры, имеется в виду, как организован код. Хорошая архитектура может помочь нам избежать таких проблем, как низкая производительность, длительное время загрузки или трудно поддерживаемый код.

Архитектура данного разрабатываемого игрового средства основывается на компонентной системе. Она играет центральную роль в разработке игр в Unity [22]. Очень важно понимать шаблон проектирования компонентов и использовать его таким образом, чтобы образовалась прочная основа для долгосрочного проекта. При более детальном рассмотрении в проекте есть игровые объекты и компоненты.

В зависимости от того, какой тип объекта требуется создать, в GameObject добавляются различные комбинации компонентов. Можно представить GameObject как пустую кастрюлю, а о компонентах как о различных ингредиентах, составляющих рецепт игры. На следующем рисунке 2.1 показана структура игровых объектов и их компонентов в разрабатываемом проекте.

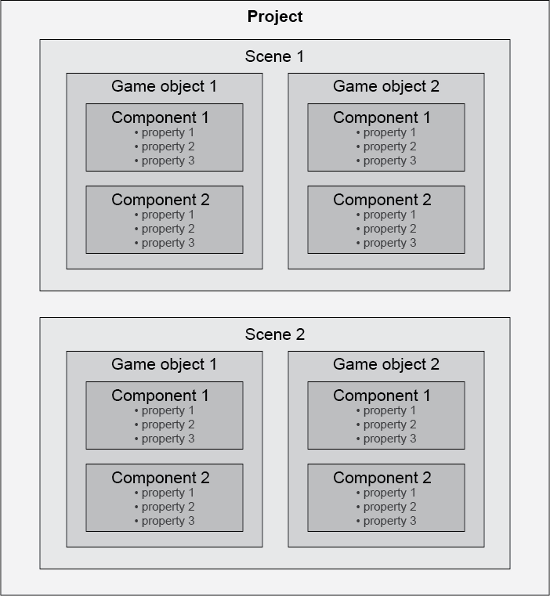


Рисунок 2.1 – Структура игровых объектов и их компонентов

В больших играх довольно сложная архитектура. Сложные сущности и сложное взаимодействие между классами. Если пытаться разрабатывать игры, используя стандартный объектно-ориентированный подход, то гарантированы постоянные переделки куча кода и сильное увеличение длительности разработки.

Проблема кроется в наследовании. Существует проблема хрупких базовых классов. Это ситуация, когда изменить реализацию типа-предка невозможно, не нарушив корректность функционирования типов-потомков.

Компонентно-ориентированный подход был использован для поиска решений в борьбе с проблемой наследования. Вкратце, суть компонентно-ориентированного подхода заключается в том, что есть некий класс-контейнер, а также класс-компонент, который можно добавить в класс-контейнер. Объект состоит из контейнера и компонентов в этом контейнере.

Компонентом называется класс, унаследованный от MonoBehaviour. Здесь само название базового класса подобрано очень хорошо, даёт понять, что это поведение, и говорит, что исполняется оно платформой mono. Весьма удобно считать, что MonoBehaviour представляет собой одно поведение. И первое, на что нужно обращать внимание при разработке компонентов: один компонент — одно поведение. Принцип единственной ответственности из SOLID и высокое сцепление методов внутри класса по GRASP в действии.

Проект Unity, по сути, является папкой, содержащей все файлы и информацию об игре [17]. Некоторые файлы называются сценами (можно считать их уровнями). Сцена содержит несколько игровых объектов (GameObjects), которые были добавлены в нее. Содержание сцен и их количество определяется разработчиком. Игру можно переключать между различными сценами, делая активными разные наборы игровых объектов.

Компоненты – это изолированные функциональные возможности, которые могут быть присоединены к объекту для придания этой функциональности этому конкретному объекту. Таким образом, когда объект требует определенного типа функциональности, разработчик просто добавляет соответствующий компонент.

Используя компоненты, можно легко собрать мощные игровые объекты, одновременно используя несколько небольших частей [18], каждая из которых отвечает за простую задачу или поведение – рендеринг игрового объекта, обработку ввода, получение урона, воспроизведение звукового эффекта. Такой подход делает игру намного проще в разработке и управлении. Вот несколько примеров компонентов, встроенных в Unity:

1Transform–единственный компонент, к которому каждый игровой объект в Unity прикреплен по умолчанию. Компонент Transform позволяет определить положение, вращение и масштаб игрового объекта. Обычно можно прикреплять, отсоединять и уничтожать компоненты в любом игровом объекте по своему желанию, но нельзя удалить Transform.

2Box Collider– коллайдер в форме куба, который можно использовать для обнаружения столкновений.

3Mesh Filter– данные сетки, которые используются для отображения 3D-модели.

4Rigidbody отвечает за физическую симуляцию.

Каждый компонент имеет ряд свойств [19], к которым можно обращаться и изменять: это могут быть целые числа или числа с плавающей запятой, строки текста, текстуры, скрипты, ссылки на игровые объекты или другие компоненты. Они используются для изменения поведения определенного компонента, влияния на его внешний вид или взаимодействие. Некоторые из свойств включают свойства положения, поворота и масштаба компонента Transform.

На следующем рисунке 2.2 показан игровой объект Wall, к которому прикреплены компоненты Transform, Mesh Filter, Box Collider, Mesh Renderer и Script, а также отображены свойства компонента Transform.

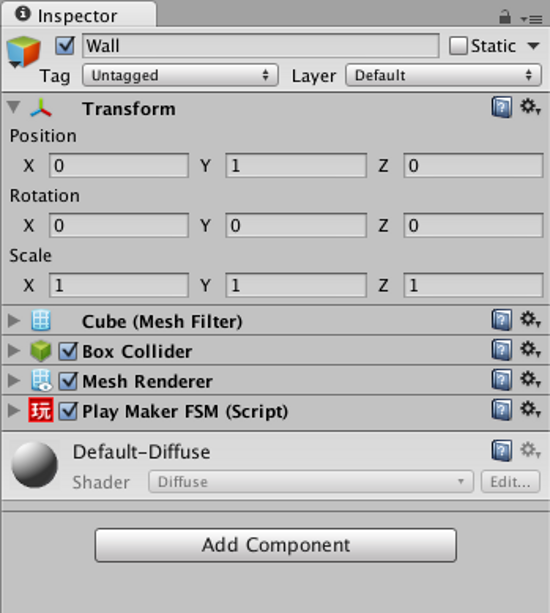


Рисунок 2.2 – Компоненты объекта Wall

Так же в Unity существуют несколько предопределенных игровых объектов, такие как камеры, источники света и примитивы, к которым уже прикреплены компоненты.

## **2.2 Разработка объектной модели программного средства**

Объектная модель описывает структуру объектов, составляющих систему, их атрибуты, операции, взаимосвязи с другими объектами. Игровые объекты(GameObject) – это фундаментальные объекты в Unity, которые представляют персонажей и декорации [23]. Каждый объект в игре является GameObject. Игровое поле содержит в себе игровые объекты, которые являются неотъемлемой частью самой игры и влияют на геймплей. В разрабатываемом игровом программном средстве были созданы следующие объекты:

– Main Camera;

– Directional Light;

– Helix;

– Wall;

– Canvas;

– Cylinder;

– Ball;

– Game Controller;

– Sound FX;

– Finish Line

– Sound FX;

– ColorBump.

Однако сам по себе игровой объект – это просто невидимый контейнер, который ничего не делает. Без добавления соответствующих компонентов он не может, например, появляться на сцене, получать входные данные от игрока или перемещаться и взаимодействовать с другими объектами. Например, объект Light создается путем присоединения компонента Light к GameObject (см. рисунок 2.3).



Рисунок 2.3 – Компоненты объекта Light

Каждый их созданных объектов содержит в себе те или иные встроенные компоненты Unity. Несмотря на то, что встроенные компоненты Unity могут быть очень универсальными, нам нужно выйти за рамки того, что они могут предоставить, для реализации своих собственных игровых функций.

Это позволит нам запускать игровые события, изменять свойства компонента с течением времени и реагировать на ввод пользователя любым удобным для нас способом. Unity позволяет нам создавать свои собственные Компоненты, используя скрипты [20].

Непосредственно программирование в Unity3D заключается, прежде всего, в разработке пользовательских классов, которые подключаются к игровым объектам в качестве компонентов.

Все такие классы должны быть унаследованы от встроенного класса MonoBehaviour. MonoBehaviour – это базовый класс, из которого происходит каждый скрипт Unity. Он предлагает некоторые функции жизненного цикла, которые облегчают вам разработку приложения и игры. Исходное содержимое класса, унаследованного от родительского класса MonoBehaviour, выглядит как показано на рисунке 2.4.

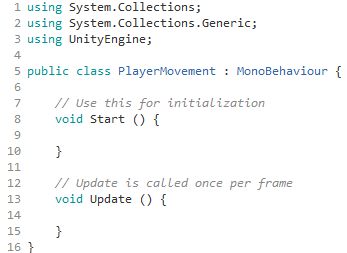


Рисунок 2.4 – Исходное содержимое класса

Имя класса берется из имени, которое было указано при создании файла скрипта. Имя класса и имя файла должны быть одинаковыми, чтобы позволить компоненту сценария быть присоединенным к GameObject. Необходимо отметить две основные функции, определенные внутри класса.

*Функция* Start будет вызываться Unity до начала игрового процесса (до первого вызова функции Update) и является идеальным местом для инициализации.

*Функция* Update – это место для размещения кода, который будет выполняться при каждом обновлении кадра. Это может включать в себя движение, запуск действий и реагирование на ввод пользователя, в основном все, что должно быть обработано во время игры [21].

Следует отметить, что в одном проекте может быть любое количество сцен, любое количество игровых объектов в одной сцене, любое количество компонентов, прикрепленных к одному игровому объекту, и, наконец, любое количество свойств в одном компоненте [24]. Unity имеет множество различных встроенных типов компонентов, а также можно создавать свои собственные компоненты, используя Unity Scripting API. Были спроектированы следующие основные классы игрового приложения:

1 Класс GameControllerотвечает за создание уровней, генерацию цветов для игровых объектов, за появление и удаление препятствий.

2 Класс Ball отвечает за цвет мяча, за начало движения и скорость мяча, за получение простого или двойного количества баллов, за воспроизведение звуковых дорожек.

3 Класс Helix отвечает за контроль вращения оси пользователем.

4 Класс Wall отвечает за создание разных по размеру препятствий, за разделение стен на две области.

5 Класс WallFragment отвечает за смену цвета части стены после того, как мяч пройдет через линию Colorbump, за смену цвета частей стен, если они окажутся одноцветными.

6 Класс Level отвечает за работу шкалы прогресса уровня, за отображение текстовых сообщений после завершения уровня.

7 Класс Score отвечает за отображение наибольшего и текущего количество набранных баллов, а также за обновление рекорда.

## **2.3 Разработка алгоритмов функционирования программного**

## **средства**

Суть игры заключается в контроле угла поворота горизонтально расположенной оси, по которой движется мяч. Также вокруг оси расположены круговые стены, каждая из которых разделена на две части с разными цветами. Мяч может проходить только через те части стен, которые окрашены в один цвет с ним. Столкнувшись же с частью стены, не того же цвета что и сам мяч, игра заканчивается. Задачей игрока является провести мяч через все стены и пересечь финишную черту.

Проведем обзор основных алгоритмов функционирования игрового приложения. В данной игре пользовательское управление заключается в контроле вращения оси со стенами. Рассмотрим алгоритм поворота оси, который представлен на рисунке 2.5.

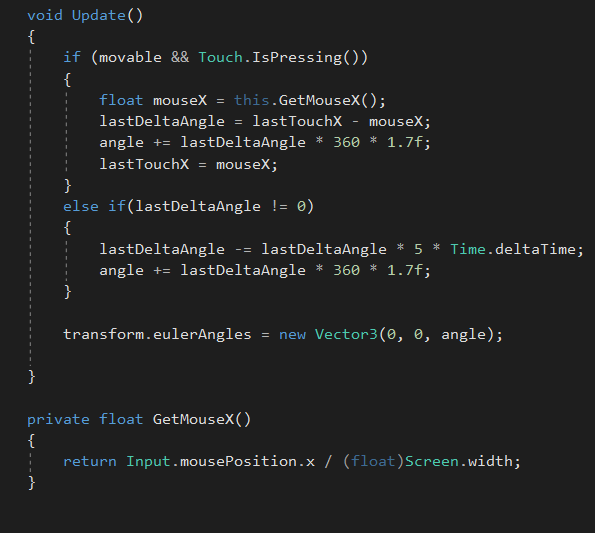


Рисунок 2.5 – Алгоритм поворота оси

Метод Update() оперирует с данными ввода от пользователя и преобразует их в угол поворота оси.

На оси вращения присутствуют препятствия в виде круговых стен (см. рисунок 2.6).

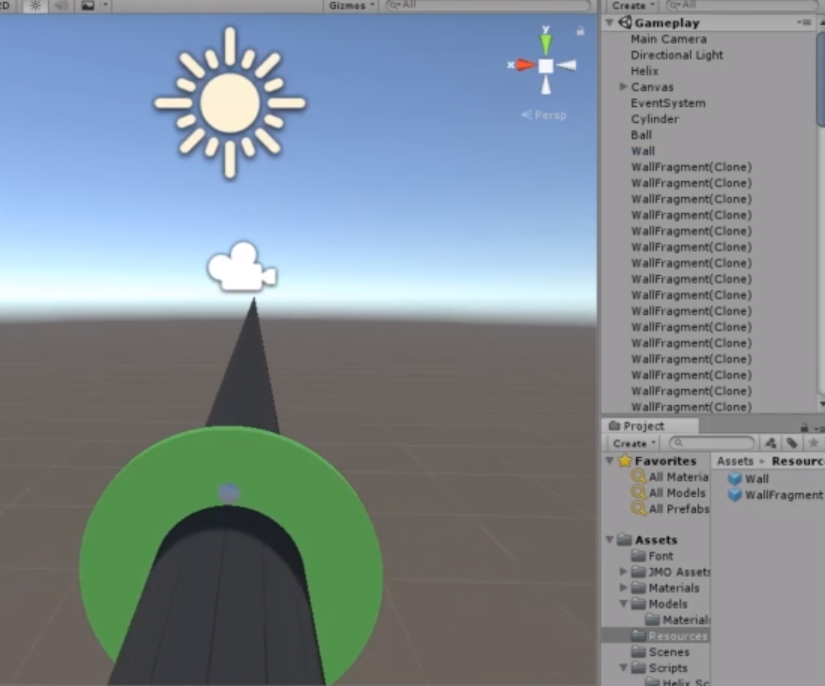


Рисунок 2.6 – Круговая стена

Однако стена обладает не цельной структурой, а состоит из ста меньших фрагментов как показано на рисунке 2.7. За создание и соединение фрагментов в стену отвечает метод SpawnWallFragments().

В методе SpawnWallFragments() используется цикл выполняемый сто раз. Каждый шаг цикла добавляет к стене новый фрагмент с увеличенным параметром rotationZ, который отвечает за расположение фрагмента относительно оси вращения.

По завершению выполнения цикла получается препятствие, которое расположено вокруг продолговатого цилиндра. Сам цилиндр являются осью, на которой размещены основные игровые объекты.

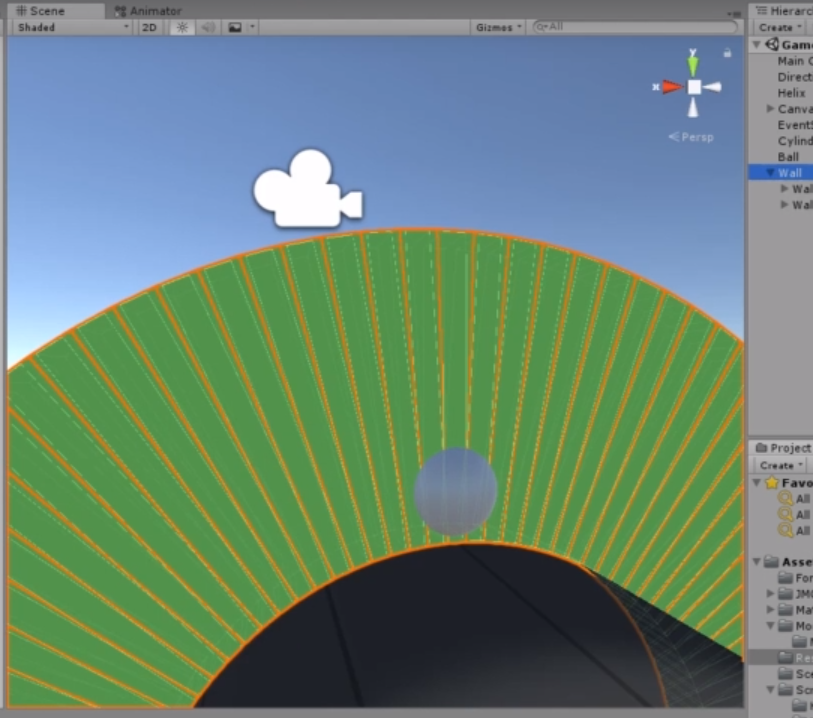


Рисунок 2.7 – Фрагменты стены

Фрагменты стены разделяются на две родительские группы, тем самым формируя две части стены. Алгоритм представлен на рисунке 2.8. Одна часть будет являть собой препятствие для мяча, а через вторую часть мяч сможет проходить.

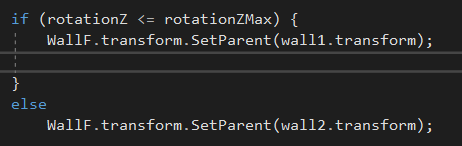


Рисунок 2.8 – Алгоритм разделения стены на две части

Для лучшей заметности, центр той части стены, через которую может пройти мяч, визуально обозначен фигурой в форме звезды белого цвета (см. рисунок 2.9). Позже будет реализовано начисление дополнительных баллов за прохождение мячом по тому участку, где расположена фигура звезды.

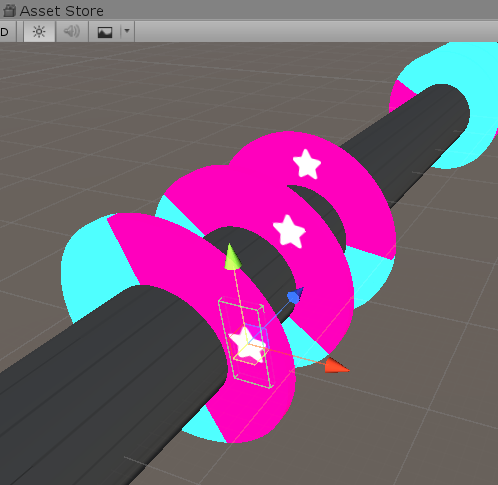


Рисунок 2.9 – Обозначение центра фигурой звезды

Для игрока вид на игру показывается через камеру, которая захватывает и отображает объекты. В данной игре движущимся объектом является мяч. Для удобного наблюдения за мячом, игроку должен показываться вид от третьего лица. Для этого стоит привязать камеру к перемещению мяча по оси. Данный механизм привязки реализован методом Update() (см. рисунок 2.10) в классе CameraFollow.

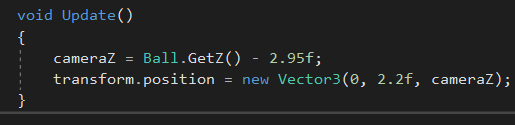


Рисунок 2.10 – Привязка камеры

Для создания вида от третьего лица, камера расположена за мячом и немного выше него (см. рисунок 2.11).

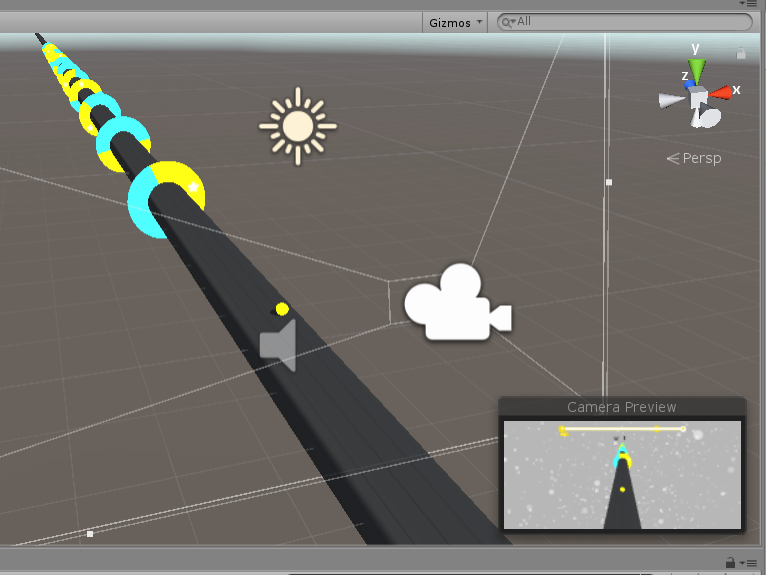


Рисунок 2.11 – Расположение камеры

Для расчета координат камеры берется координата мяча по оси z, и далее от нее отнимается некоторое значение, которое определяет удаление от мяча, в данном случае это 2.95f. Это необходимо для следования камеры за мячом.

Для окрашивания двух частей стен требуется сгенерировать два разных цвета. Необходимо исключить случаи, когда могут выбраться два одинаковых цвета. Выбор цветов и проверка на одинаковость реализуется в методе GenerateColors() класса GameController (см. рисунок 2.12).

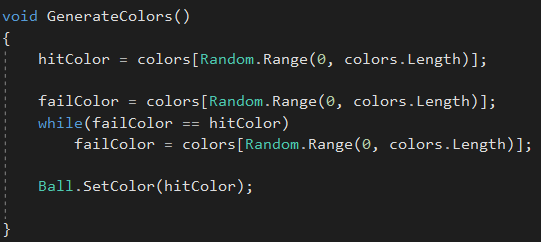


Рисунок 2.12 – Генерация цветов

Для лучшей визуальной наглядности мяч имеет тот же цвет что и стена, через которую ему можно пройти. Размещение стен вдоль оси реализует метод SpawnWalls(), код которого представлен на рисунке 2.13. Чтобы каждая стена вращалась вместе с осью, ее располагают как дочерний объект вращаемой оси Helix.

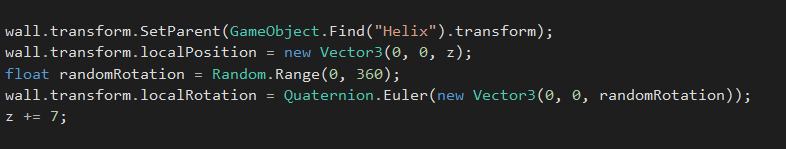


Рисунок 2.13 – Размещение стен

Каждая новая стена должна появляться на некотором отдалении от предыдущей (см. рисунок 2.14), расстояние между стенами задается параметром z.

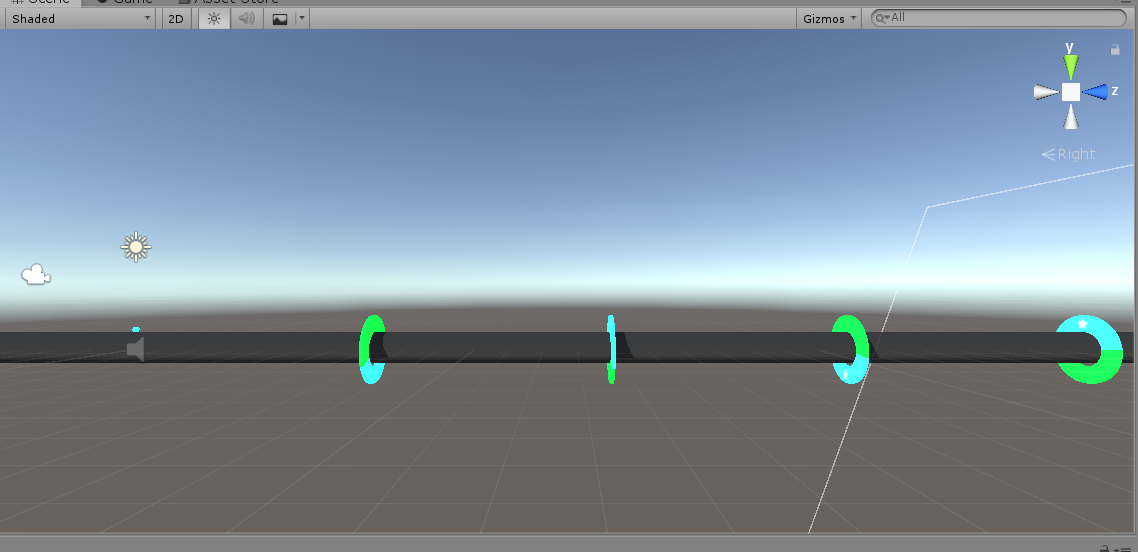


Рисунок 2.14 – Расстояние межде препятствиями

Чтобы все стены не размещались с одинаковым углом наклона, что сделало бы игру слишком простой, каждой стене стоит задать свой случайный угол наклона. Это заставит игрока постоянно маневрировать и тем самым значительно усложнит игровой процесс.

Позади всех стен должна расположиться финишная черта (см. рисунок 2.15) обозначающая конец игрового уровня.

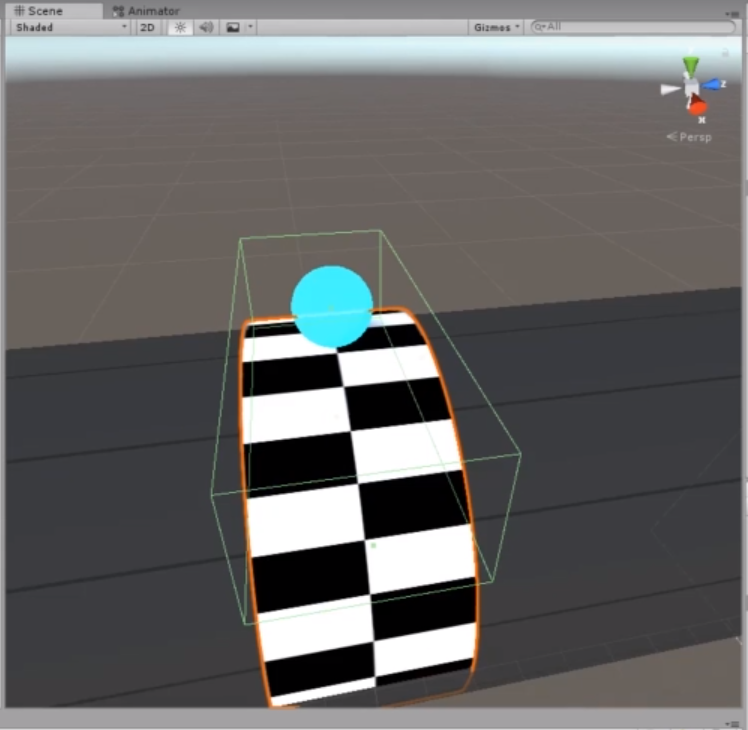


Рисунок 2.15 – Финишная черта

Размещение финишной черты прописано в том же методе SpawnWalls() (см. рисунок 2.16) класса GameController. Финишная черта разместится только после того, как на сцене появятся все стены, в количестве заданном на данном игровом уровне.

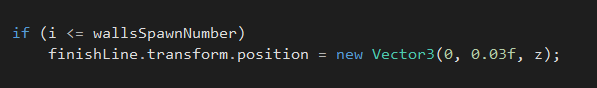


Рисунок 2.16 – Расположение финишной черты

При столкновении мяча с частью стены не того же цвета, прохождение уровня начинается заново. Перед тем как не пройденный уровень создастся заново, необходимо удалить все его объекты. За данный процесс отвечает метод DeleteWalls() (см. рисунок 2.17).

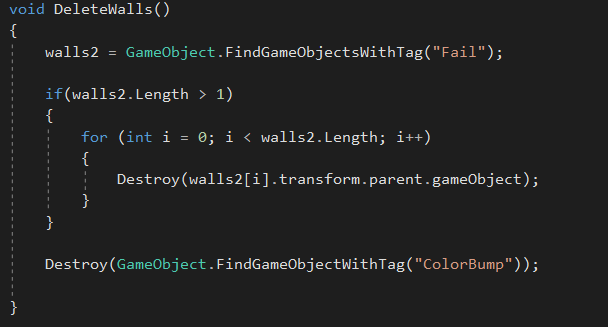


Рисунок 2.17 – Удаление старых объектов

Для большего разнообразия и усложнения прохождения игры было решено добавить случайное изменение окраски стен посреди уровня. Предупреждать о последующей смене цвета стен будет специальная линия ColorBump (см. рисунок 2.18), которая расположится на пути движения мяча.



Рисунок 2.18– Линия ColorBump

За выбор случайного цвета для линии ColorBump отвечает метод Start() класса ColorBump. Код метода представлен на рисунке 2.19.

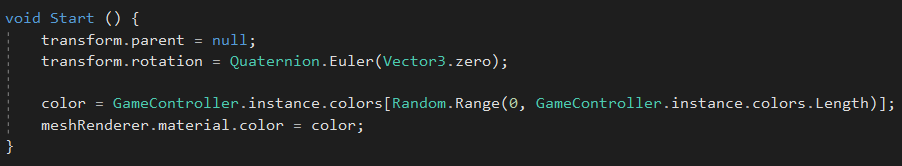


Рисунок 2.19 – Выбор случайного цвета

Вероятность появления на уровне линии ColorBump прописана в методе SpawnWalls() класса GameController. В нашей игре эта вероятность может быть задана любой. Но проведя тесты, оптимальная вероятность появления была определена как два к десяти (см. рисунок 2.20).

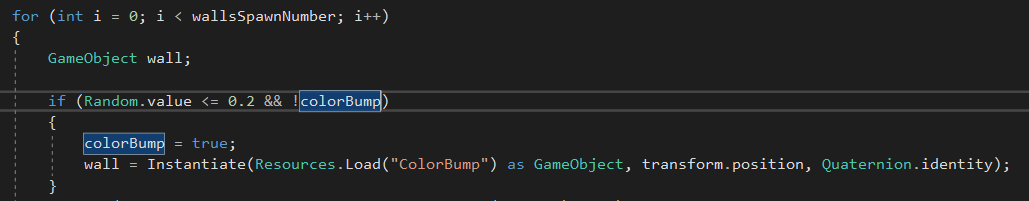


Рисунок 2.20 – Условие появления ColorBump

Алгоритм изменения цвета стен представлен на рисунке 2.21. Если, судя по координате z, стена находится за линией ColorBump, то та ее часть, через которую мяч может пройти, будет перекрашена в цвет линии ColorBump.

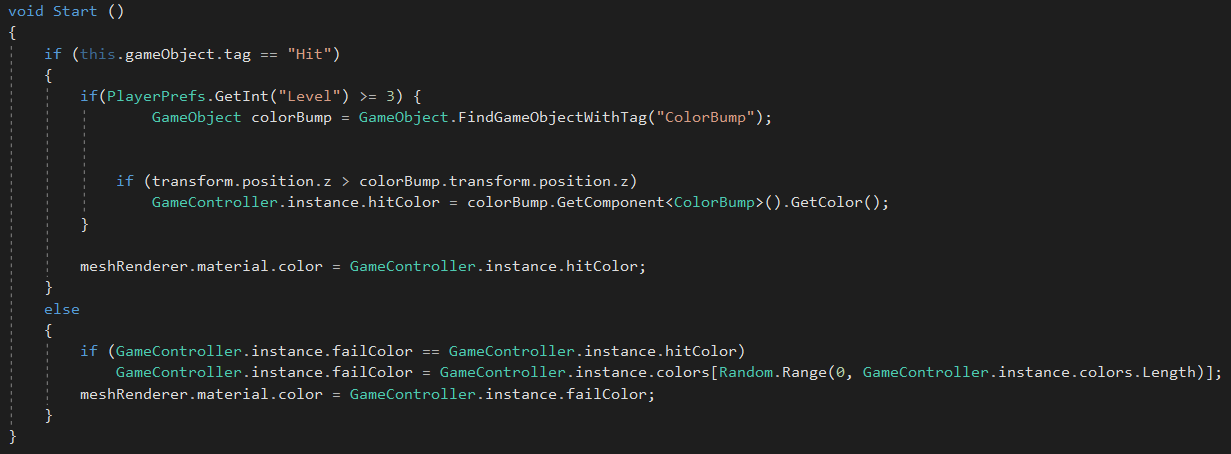


Рисунок 2.21 – Алгоритм смены цвета

Однако не исключено, что цвет линии ColorBump может оказаться таким же, как и цвет части стены через которую мяч не может пройти. Тогда вся стена станет окрашена в один цвет, игроку будет невозможно определить как пройти через стену. Для исключения таких случаев добавлена дополнительная проверка. Если цвет части стены, через которую мяч не может пройти, будет таким же, как и у остальной части, то он будет заменен на новый случайно выбранный. При пересечении мячом линии ColorBumb он также меняет цвет. Алгоритм представлен на рисунке 2.22.

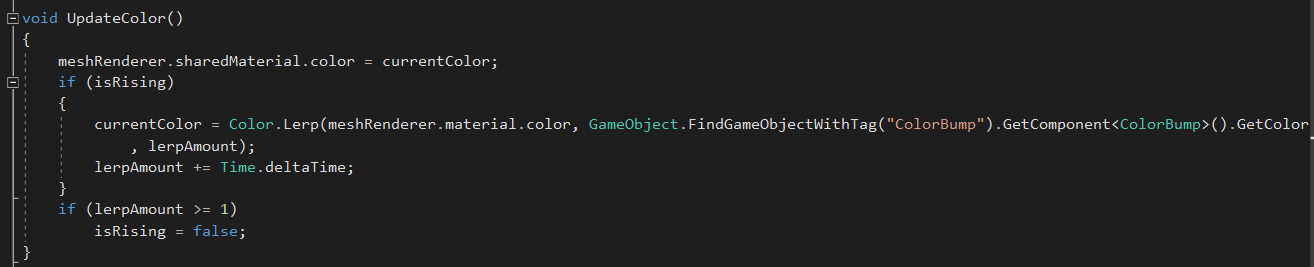


Рисунок 2.22 – Смена цвета мяча

Для создания визуального отклика на прохождение мяча через стену, в проект была добавлена функция (см. рисунок 2.23) разрушения той ее части, через которую он прошел.

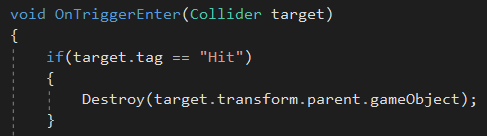


Рисунок 2.23 – Функция удаления части стены

Оставшаяся позади мяча половина стены наглядно продемонстрирована на рисунке 2.24.



Рисунок 2.24 – Удаление части стены

Помимо стен состоящих из двух одинаковых по размеру частей, в игру также добавлены стены, размеры составных частей которых имеют соотношение один к трем (см. рисунок 2.25). Все это сделано для усложнения игрового процесса.



Рисунок 2.25 – Меньшая стена

Появление таких стен задается случайным. Данная функция представлена на рисунке 2.26. Вероятность появления стены с размером прохода в девяносто градусов равна двадцати процентам.

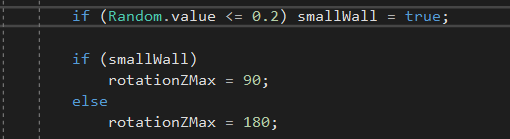


Рисунок 2.26 – Функция появления стены

В игре предусмотрено постепенное усложнение игрового процесса. Происходит это по мере увеличения количества пройденных уровней. Увеличение сложности заключается в добавлении дополнительного количества стен на уровень. Данный принцип описан в методе GenerateLevel() (см. рисунок 2.27) класса GameController.

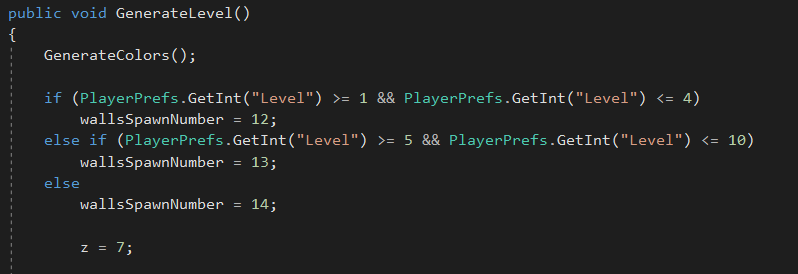


Рисунок 2.27– Увеличение количества стен

На уровнях с первого по четвертый будут появляться двенадцать стен. На уровнях с пятого по десятый количество стен увеличится на одну. Начиная с одиннадцатого уровня количество стен будет равно четырнадцати. Также чтобы не делать первые уровни слишком сложными для новичков, линия ColorBump будет появляться только после прохождения третьего уровня (см. рисунок 2.28).

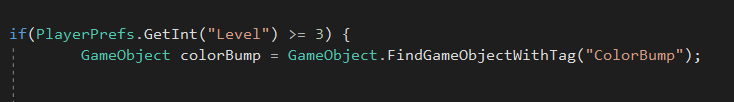


Рисунок 2.28 – Условия появления ColorBump

Стены с уменьшенным вдвое размером прохода будут появляться только после прохождения пятого уровня (см. рисунок 2.29).

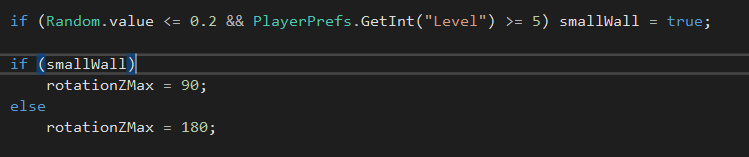


Рисунок 2.29 – Условия появления меньшей стены

Дополнительно в игру было добавлено поощрение, за прохождение мячом через участок, обозначенный фигурой звезды.За выполнение данного условия игрока ждет вознаграждение в виде удвоения баллов. Данное решение будет способствовать большей мотивации игрока.

## **2.4 Разработка и обоснование пользовательского интерфейса**

## **программного средства**

Ключевой частью создания новой мобильной игры является разработка интерфейса и обеспечение его понятности для игроков. Дизайн пользовательского интерфейса является неотъемлемой частью, которая влияет на игровой процесс и пользовательский опыт. Подавляющее большинство игр использует некоторую форму пользовательского интерфейса для отображения информации, такой как здоровье, навыки, карты, боеприпасы для оружия и многое другое [25]. Unity 3D предлагает множество элементов пользовательского интерфейса, которые можно использовать в своей игре.

В движке Unity все элементы пользовательского интерфейса размещаются внутри области называемой Canvas. Любые элементы пользовательского интерфейса, которые создаются, должны быть дочерними элементами холста Canvas. Для создания Canvas выберем его из выпадающего меню категории UI (см. рисунок 2.30).

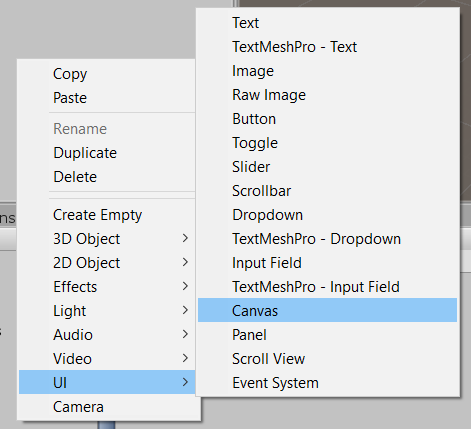


Рисунок 2.30 – Создание объекта Canvas

Для отображения счета создадим дочерний объект Score. Непосредственно в объект Score добавим текстовый элемент пользовательского интерфейса BestScoreText. Данный элемент будет использоваться для отображения лучшего игрового счета (см. рисунок 2.31).

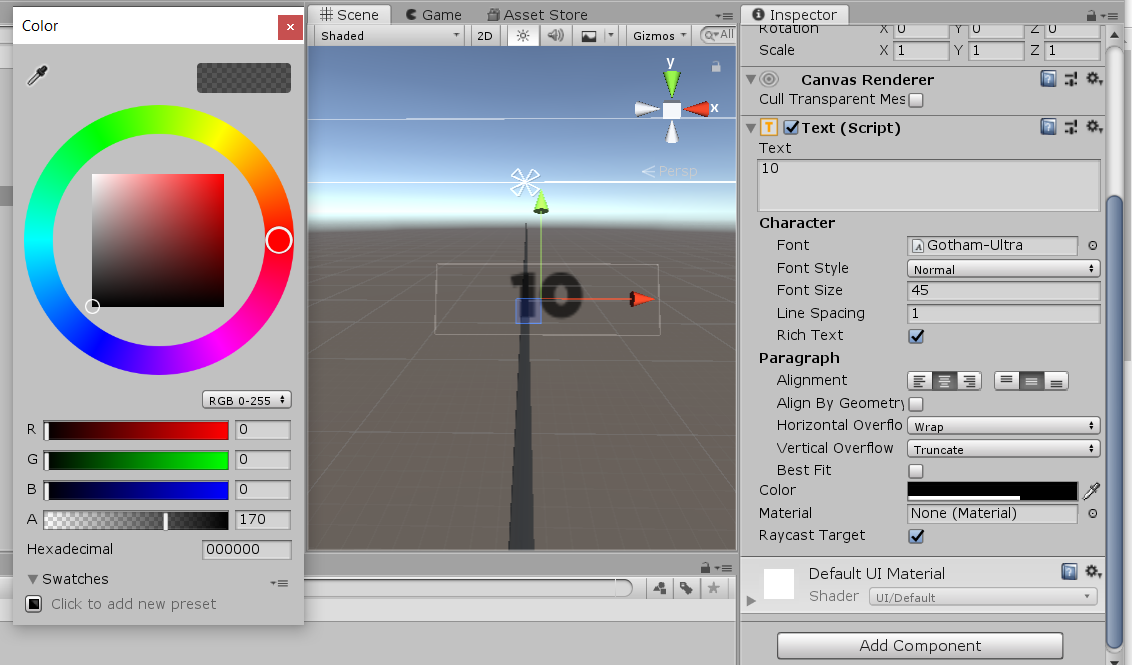


Рисунок 2.31 – Настройка элемента BestScoreText

Панель Inspector используется для того, чтобы просматривать и редактировать свойства объектов различными способами. В данной панели зададим шрифт, толщину и размер текста. В разделе Paragraph установим выравнивание по центру и черный цвет текста. Текстовый элемент разместим по центру в верхней части холста. Слева от отображаемого результата добавим изображение короны (см. рисунок 2.32). Для добавления изображения выберем его из выпадающего меню категории UI. В открывшейся панели Inspector настроим позицию слева от отображаемого счета, а также укажем желаемый размер изображения. В разделе Image выберем непосредственно саму картинку и ее цвет.

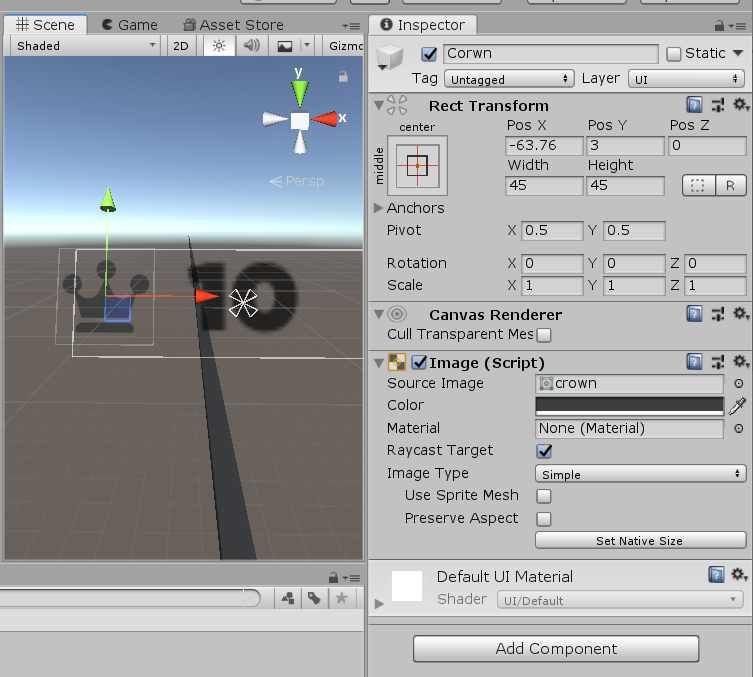


Рисунок 2.32 – Настройка изображения Crown

Помимо отображения лучшего результата необходимо показывать игроку текущий счет. По аналогии с предыдущими текстовыми элементами создадим новый с названием ScoreText. Его параметры зададим такие же как и у элемента BestScoreText.

Для того чтобы запрограммировать отображение результатов создадим файл сценария Score. Порядок отображения промежуточного и лучшего результатов отображен на рисунке 2.33.

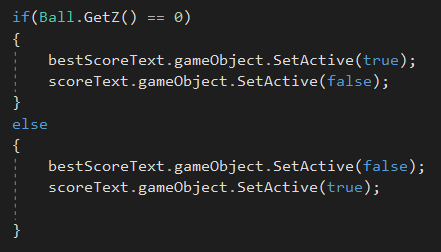


Рисунок 2.33 – Порядок отображения результатов

Перед началом игры на экране будет показываться только информация о лучшем результате. После старта отображаться будет информация только о текущем счете. Данные для отображения текущего счета берутся из переменной score класса GameController. В случае если текущий счет превысит старый рекорд, текущий счет станет новым лучшим счетом(см. рисунок 2.34).

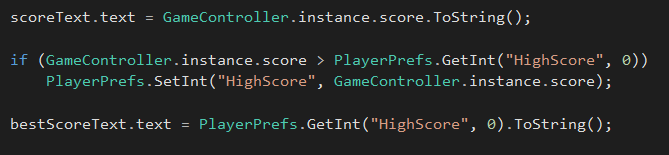


Рисунок 2.34 – Алгоритм обновления рекорда

Добавим дополнительный элемент пользовательского интерфейса чтобы показывать игроку, сколько баллов он набирает за прохождение каждого препятствия. Выберем текстовый элемент из выпадающего списка. Текстовый элемент будет называться PointDisplay (см. рисунок 2.35).

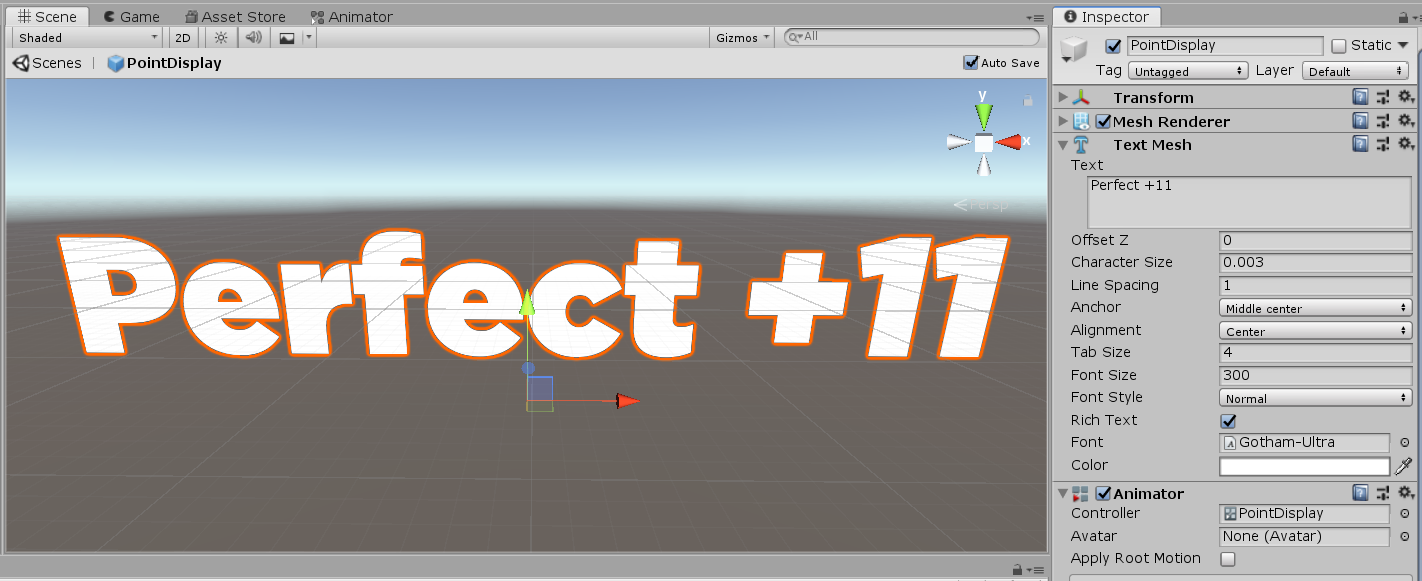


Рисунок 2.35 – Текстовый элемент PointDisplay

Данный текстовый элемент будет анимировано появляться каждый раз, как будет пройдено препятствие. Чтобы не загромождать пользовательский интерфейс, данный элемент будет самоуничтожаться через секунду после появления.

Для показа игроку, на сколько завершен текущий уровень, добавим в пользовательский интерфейс шкалу прогресса. Для этого создадим элемент пользовательского интерфейса Level. Далее добавим дочерние элементы. Общая структура и иерархия элементов представлена на 2.36.

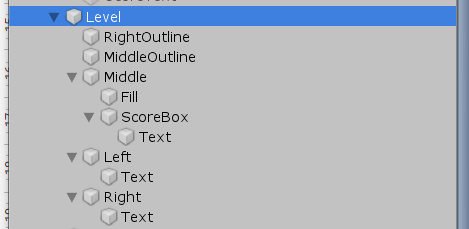


Рисунок 2.36 – Иерархия элементов шкалы прогресса

Разместим все составные элементы на холсте и зададим нужные параметры в виде размеров, шрифтов и цвета. Собранная из множества элементов шкала прогресса представлена на рисунке 2.37.

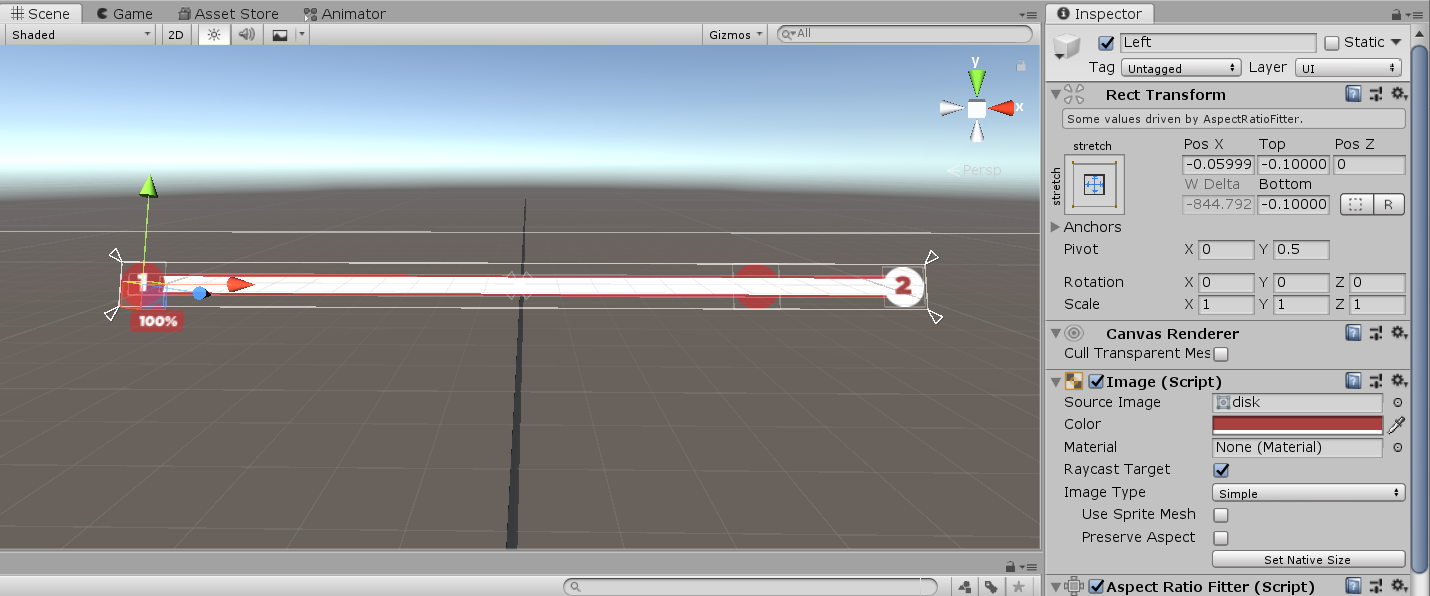


Рисунок 2.37 – Шкала прогресса

По бокам от шкалы слева направо показаны соответственно номера текущего и следующего уровней. Снизу находится панель, которая будет перемещаться по мере прохождения уровня и отображать на себе завершенность уровня в процентом соотношении.

# **3 ИНЖЕНЕРНЫЕ РАСЧЕТЫ, ИСПОЛЬЗУЕМЫЕ**

# **В ПРОГРАММНОМ СРЕДСТВЕ**

## **3.1 Оценка файловой структуры игрового программного средства**

По мере того, как проект Unity становится больше, структура папок проекта становится все сложнее, и разработчик начинает терять драгоценное время, разбираясь в своем проекте. Трудно назвать каждый файл уникальным и запоминающимся, который можно было бы легко найти позже. Если не поддерживать свой проект организованным, то позже будет труднее поддерживать темп разработки. Поэтому важно поддерживать чистоту структуры проекта [26]. При планировании файловой структуры стоит придерживаться следующих принципов:

1 Расположение ключевых файлов должно быть понятно.

2 Не стоит хранить файлы ассетов в корневом каталоге. Лучше использовать подкаталоги, когда это возможно.

3 Не создавать никаких дополнительных каталогов в корневом каталоге, если для этого нет крайней необходимости.

4 Стоит быть последовательным с именами файлов.

5 Не перемещать специальные ассеты в общие каталоги. Например, если есть материалы, сгенерированные из модели, не перемещать их в каталог материалов, потому что позже можно запутаться в том, откуда они берутся.

Папка сцены содержит только файлы сцен и ничего более. Все дочерние объекты расположены под родительскими. В результате разработки игрового приложения была создана файловая структура, содержащая ряд каталогов, которые включают в себя все файлы приложения, такие как: скрипты, графические файлы, шрифты и шаблоны объектов игрового поля. На рисунке 3.1 показана файловая структура игрового приложения.

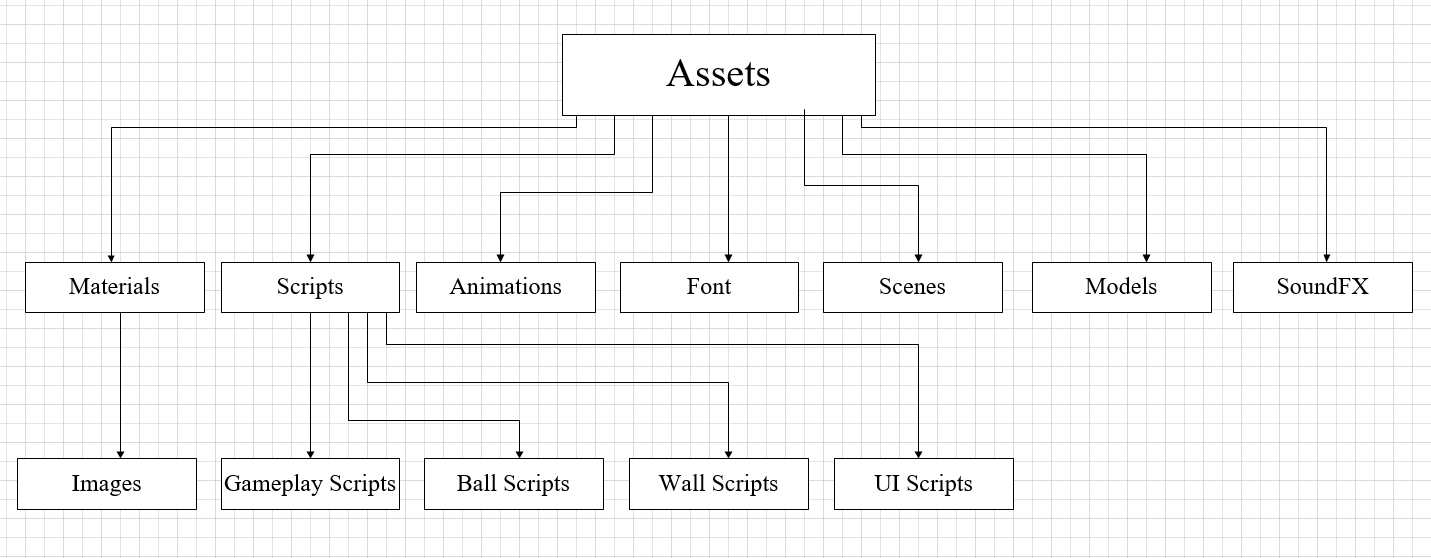


Рисунок 3.1– Файловая структура

Как наглядный пример можно взять иерархию игровых объектов на сцене. Как и структура всего проекта, структура сцены имеет тенденцию запутываться, когда сцена содержит много объектов. Сцены могут легко забиться. Часто могут возникать затруднения в нахождении нужного объекта. Поэтому стоит позаботиться о структуре в самом начале разработки. Иерархия объектов сцены в нашем проекте отражена на рисунке 3.2.

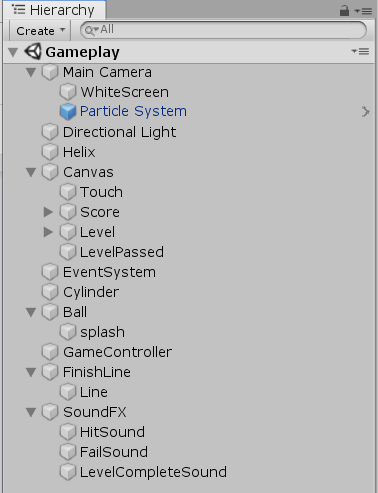


Рисунок 3.2 – Иерархия объектов сцены

Зачастую важно сохранять размер собранного приложения на минимальном уровне, особенно если речь о разработке под мобильные устройства, или для магазинов приложений, имеющих ограничения по размеру.

Первый шаг в процессе уменьшения размера – определить, какие ассеты больше всего занимают места, так как такие ассеты – первые кандидаты для оптимизации.

Эту информацию можно найти в логе редактора, сразу после совершения сборки (выбирается Open Editor Log в выпадающем меню маленькой иконки в правом верхнем углу окна Console). Данные логов представлены в таблице 3.1.

Таблица 3.1 – Количество памяти занимаемой ассетами

|  |  |
| --- | --- |
| Тип ассетов | Занимаемая память, Кбайт. |
| Textures | 150 |
| Meshes | 62 |
| Animations | 40 |
| Sounds | 87 |
| Shaders | 150 |
| Levels | 105 |
| Scripts | 90 |

Лог предоставляет сводку по ассетам, разбитым по типам и затем список отдельных ассетов, отсортированных по размеру. Как правило, текстуры, музыка и видео занимают больше всего места, в то время как скриптами, уровнями и шейдерами часто можно пренебречь.

Для того чтобы определить какую часть памяти занимают текстуры, следует рассчитать их процентное соотношение относительно общего размера ассетов. Для этого используется формула:

где – процентное соотношение;

– память, занимаемая текстурами;

– память, занимаемая мешами;

– память, занимаемая анимацией;

– память, занимаемая звуковыми дорожками;

– память, занимаемая шейдерами;

– память, занимаемая уровнями;

– память, занимаемая скриптами.

Найдем процент памяти, занимаемой текстурами:

Таким образом, можно сделать вывод о том, что в разрабатываемом игровом программном средстве текстуры занимают приемлемую часть пользовательских ассетов.

Все папки хранят типы файлов, соответствующие своему названию. Имея четкую структуру с самого начала, с проектом будет удобно работать, масштабировать, передавать другому разработчику. Правильные названия файлов, порядок позволяют экономить время. Много времени экономится на отладке, добавлении новых функций, файлов, чистке, при условии, что файловая структура понятна и удобна.

## **3.2 Анализ памяти, используемой программным средством**

Впечатление пользователя от игры, в немалой степени, зависит от ее производительности. Приятней играть в игру, которая оптимизирована должным образом, игровой процесс которой гладкий и отзывчивый. У такой игры аудитория может быть больше, так как она может работать и на менее мощных платформах [27]. Максимизация производительности может зависеть от самой игры, над которой работают, а советы по производительности, которые могут работать для одной игры, могут быть совершенно бессмысленными для другой. Хотя каждая игра может требовать своего подхода к оптимизации, существуют ключевые методы, которые можно реализовать в большинстве ситуаций.

Профайлер Unity помогает разработчикам оптимизировать игру. Он сообщает о том, как много времени тратится в различных областях вашей игры. Например, он может сообщить процент времени, потраченный на рендеринг, анимацию или внутриигровую логику. Профайлер Unity помогает разработчикам оптимизировать игру. Он сообщает о том, как много времени тратится в различных областях вашей игры. Например, он может сообщить процент времени, потраченный на рендеринг, анимацию или внутриигровую логику. Можно играть в игру в редакторе с включенным профайлером, и он будет записывать данные о производительности. Записанные данные отображаются на временной шкале в окне профайлера, позволяя обнаружить зашкаливающие (занимающие больше времени) по сравнению с другими кадры или области. Кликнув по любому кадру на временной шкале, он будет выделен и в нижней части окна Profiler будет показана подробная информация для этого кадра. Существует два режима, которые можно использовать для изучения потребления памяти в приложении. Выбрать один из них можно в выпадающем меню в верхней части нижней панели.

Простой режим (см. рисунок 3.3), в котором для выбранного кадра отображается количество используемой памяти во всем редакторе на высоком уровне (довольно обобщенно и без детализации).

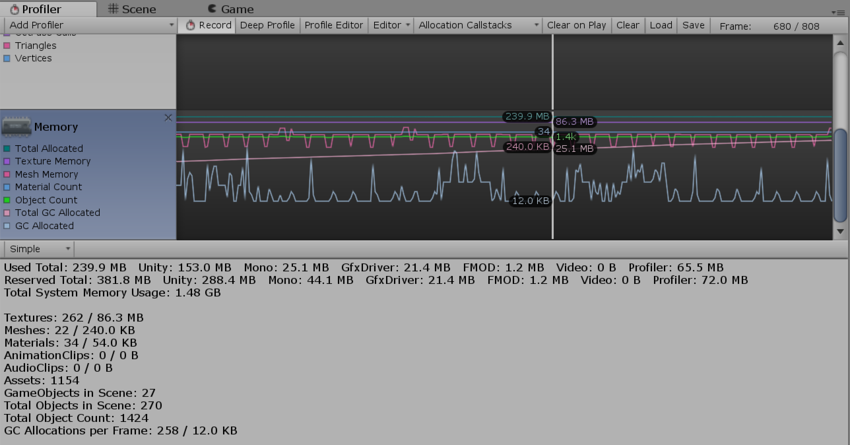


Рисунок 3.3 – Простой режим Unity Profiler

Unity резервирует пулы памяти для быстрого выделения памяти под новые объекты, то есть, чтобы избежать слишком частого запрашивания дополнительной памяти у операционной системы [28]. В нижней панели отображается как количество зарезервированной памяти (Reserved Total), так и количество используемой памяти из той, что была зарезервирована (Used Total). Выводится для следующих областей:

1Unity – это количество памяти, выделенной в нативном коде Unity.

2Mono – это общий и используемый объём т.н. “кучи”, используемой в управляемом коде – эта память обрабатывается сборщиком мусора.

3GfxDriver – это примерный размер памяти, используемый драйвером для текстур, прорисовки, шейдеров и данных меша.

4FMOD – это примерный размер памяти, потребляемый аудио драйвером.

5Profiler – это пямять, потребляемая самим профайлером.

Следует учитывать тот факт, что профайлинг должен инструментировать код программного средства, то есть, внедрять в код набор инструкций для измерения производительности. Для расчета влияния работы самого профайлера на показатели производительности в процентном соотношении используется следующая формула:

где – процентное соотношение;

– количество памяти, выделенной в нативном коде Unity;

– количество памяти, используемой в управляемом коде;

– количество памяти, используемый драйвером;

– количество памяти, потребляемой аудио драйвером;

– количество памяти, потребляемая самим профайлером;

Таблица 3.2 – Количество зарезервированной памяти

|  |  |
| --- | --- |
| Тип области | Занимаемая память, Мбайт. |
| Unity | 288,4 |
| Mono | 44,1 |
| GfxDriver | 21,4 |
| FMOD | 1,2 |
| Profiler | 72 |

Найдем процент памяти, занимаемой текстурами:

Таким образом, можно сделать вывод о том, эта инструментация немного влияет на производительность игры. Издержки недостаточно велики, чтобы повлиять на частоту кадров игры. При профайлинге обычно имеет значение только пропорциональность в процентном соотношении времени, потраченного в определенных областях.

Статистика потребления памяти отображается и для некоторых наиболее часто используемых типов объектов и включает в себя количество элементов и объем потребляемой ими памяти. Данными типами объектов являются: textures (текстуры), meshes (меши), materials (материалы), animations (анимации), audio (аудио).

ObjectCount это общее количество созданных объектов. Если это число растет со временем, значит создаются некие объекты, которые никогда не уничтожаются

В*детальном режиме* (см. рисунок 3.4) можно сделать снимок текущего состояния. Нажав кнопку “Take Sample” для совершения сбора детальных данных о потреблении памяти. Процесс получения этих данных занимает некоторое время, поэтому детальный режим не подходит для отображения данных в реальном времени. После создания снимка окно профайлера обновится и в нижней части появится подробное дерево из элементов для изучения потребления памяти [29].

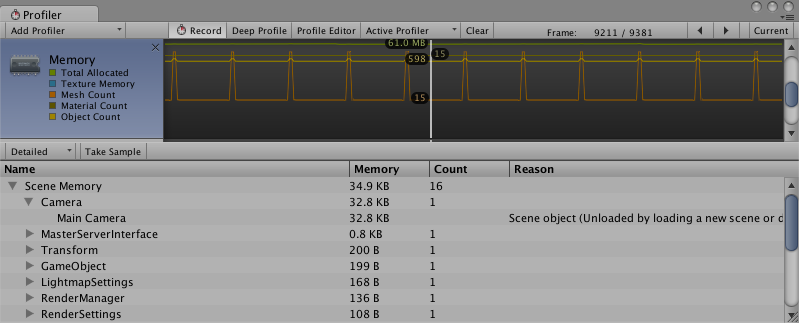


Рисунок 3.4 – Детальный режим Unity Profiler

При клике по объекту в списке, он будет выделен в сцене или в проекте, если игра запускалась в редакторе. При профилировании игры в редакторе все значения в профайлере памяти отражают потребление самим редактором. При профилировании проигрывателя значения будут меньше, т.к. не будет дополнительной нагрузки от редактора. Для более точного измерения потребления памяти приложением, стоит использовать удаленное профилирование, подключая профайлер к запущенному проигрывателю. Это позволит получить точные данные для работающего на устройстве приложения.

# **4 ЭКСПЛУАТАЦИЯ ПРОГРАММНОГО СРЕДСТВА**

## **4.1 Ввод в эксплуатацию программного средства**

Этап введения приложения в эксплуатацию является одним из самых ответственных этапов в цикле производства приложения. В конечном счете, именно от него зависит доступность приложения, сохранность исходных кодов и пользовательских данных, а также надежность приложения в целом.

На финальном этапе разработки игры требуется посмотреть на то, как она выглядит вне редактора, при сборке в качестве самостоятельного приложения. Для создания сборки надо вызвать и использовать окно Build Settings (настройки сборки). Пункт меню File Build Settings позволяет открыть окно Build Settings (см. рисунок 4.1). В нём выводится редактируемый список сцен для включения в сборку игры.



Рисунок 4.1 – Окно настройки сборки

Список будет пуст при первом открытии этого окна в проекте. В таком случае, при сборке, в игру будет включена только текущая открытая сцена. Для создания сборки с несколькими сценами можно довольно просто добавить нужные сцены в этот список. Первый способ заключается в нажатии кнопки Add Current. Текущая сцена появится в списке. Второй способ предполагает перетаскивание в список сцены из окна Project View. Следует заметить, что каждая из сцен в списке имеет свой индекс. Для публикации сборки, нужно выбрать нужную платформу в списке Platform и удостоверится, что напротив неё находится логотип Unity; если это не так, тогда нажать кнопку Switch Platform, чтобы сообщить Unity о том, под какую платформу надо совершать сборку. После этого, нажать кнопку Build. Откроется стандартное диалоговое окно сохранения файла, в котором вы можно выбрать имя и расположение для игры. После нажатия кнопки Сохранить Unity соберёт приложение.

Процесс публикации игры в магазине приложений достаточно продолжителен из-за согласования и проверок контента, содержащемся в приложении [30]. Поэтому, в рамках данного дипломного проекта будет произведен лишь процесс начала ввода его в эксплуатацию, вплоть до момента отправки заявки на его одобрение и дальнейшую публикацию в магазине мобильных приложений Google Play. Прежде чем опубликовать любое приложение в Google Play, необходимо создать учетную запись разработчика. Можно легко зарегистрироваться, используя существующую учетную запись Google (см. рисунок 4.2).

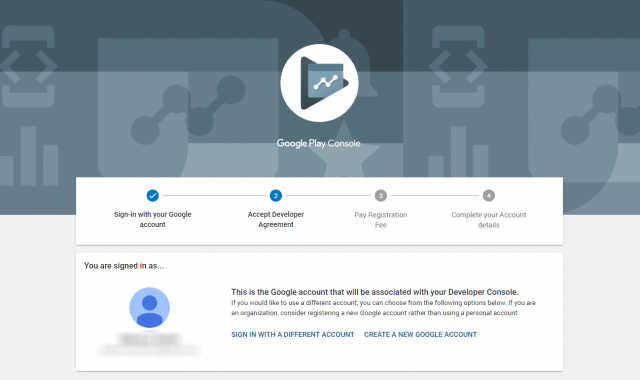


Рисунок 4.2 – Процесс создания учетной записи

Процесс регистрации довольно прост, далее нужно будет заплатить единовременный регистрационный сбор. После ознакомления с соглашением для разработчиков и его принятии, можно перейти к оплате с помощью своей кредитной или дебетовой карты.

Чтобы завершить процесс регистрации, надо заполнить все необходимые данные учетной записи, включая имя разработчика, которое будет отображаться в Google Play. Всегда можно добавить больше деталей позже. Кроме того, стоит помнить, что для полной обработки регистрации может потребоваться до 48 часов.

Далее необходимо открыть форму публикации приложения, которую нужно будет заполнить, в том числе заполнить полное и краткое описания публикуемого приложения, задать некоторые другие поля вроде ключевых слов, по которым пользователи смогут найти данное приложение, а также выбрать его название, которое было выбрано как «Run Out», т. к. оно является кратким, лаконичным и лучшим образом отображает представленный функционал и возможности использования приложения.

После проделанных действий следует отправить заявку в магазин приложений Google Play на рассмотрение возможности публикации данного приложения. Затем, после отправки заявки, остается лишь ждать результатов ее проверки и дальнейшего ответа со стороны магазина приложений Google Play.

Данный процесс ввода в эксплуатацию занимает достаточно много времени, и он достаточно сложен для подробного и детального описания без каких-либо имеющихся на данный момент подтверждающих это описание фактов. Игровое аркадное программное средство уже разработано и создано, отправлено на публикацию, и предполагается, что данный процесс будет также успешно завершен, и в итоге приложение появится в магазине Google Play, откуда его сможет загрузить на свое мобильное устройство любой желающий. На момент завершения написания данного дипломного проекта была отправлена заявка на публикацию данного приложения в магазин приложений Google Play. Ожидается что проверка пройдет успешно, приложение будет одобрено и опубликовано в магазине.

## **4.2 Руководство к пользованию разработанным программным**

## **средством**

Run Out – это трехмерная мобильная игра в аркадном жанре для Android. Run Out представляет собой бесконечную игру, в которой игрок пытается провести мяч через множество препятствий в виде стен разных цветов. Играть в Run Out просто. Управление производится пальцем, путем его перемещения по экрану слева направо, чтобы вращать спираль (см. рисунок 4.3).

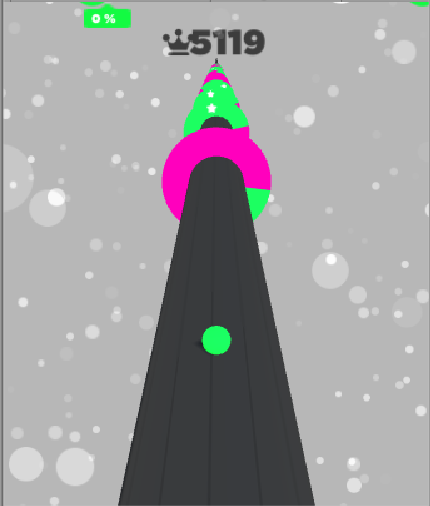


Рисунок 4.3 – Игровой процесс

Игрок не влияет на движение мяча, который находится на экране, только на ось со стенами, которые имеют разный угол наклона. Проходя через препятствие, игрок получает баллы.

Всплывающие сообщения показывают сколько баллов было получено. Однако можно получать удвоенное количество баллов, для этого требуется провести мяч через центр стены, отмеченный изображением звезды**.** Лучший результат отображается перед стартом игры. Шкала прогресса подсказывает насколько завершен текущий уровень.

Мяч может пройти через стены только того же цвета. При столкновении со стеной другого цвета игра закончится и начнется заново. Благодаря простому, но довольно увлекательному игровому процессу игра Run out подойдет для пользователей любой возрастной категории.

# **5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ**

# **РАЗРАБОТКИ ИГРОВОГО АРКАДНОГО**

# **ПРОГРАММНОГО СРЕДСТВА** **«RUN OUT» ПОД**

# **ОПЕРАЦИОННУЮ СИСТЕМУ ANDROID**

## **5.1 Краткая характеристика программного продукта**

Игровое аркадное программное средство будет предназначено для использования в развлекательных целях. Играть в мобильные игры - лучший вариант чтобы расслабиться в свободное время. Так же мобильная игра может способствовать развитию логики, внимания, рефлексов, а также помочь в преодолении депрессии и снижению уровня стресса у игрока.

В большей степени игровое приложение ориентировано на молодежь, которая относится к числу людей той возрастной категории, представители которой с наибольшей вероятностью имеют мобильный телефон. К тому же подростки обладают большим количеством свободного времени нежели люди старших возрастных категорий. Как показывают многие исследования, подростки любят играть в видеоигры, потому что они бросают вызов и этим заинтересовывают игрока. Игры предлагают умственные и визуальные стимулы, которые увеличивают вовлеченность. Многие подростки, играя в игры, любят соревноваться друг с другом и таким способом находят в играх способность к самовыражению.

Программное средство разработано в рамках дипломного проекта с целью дальнейшего размещения в сети Internet. Основной целевой аудиторией является молодежь. Ожидается экономический эффект в виде прироста прибыли от продажи копий игры. За счёт того, что пользователи могут рекомендовать игру своим друзьям, оставлять отзывы и выставлять рейтинги, игра всегда сможет привлечь новых пользователей.

## **5.2 Расчет затрат на разработку и отпускной цены программного**

## **продукта**

1 Определение затрат на основную заработную плату команды разработчиков.

Затраты на основную заработную плату команды разработчиков определяются исходя из состава и численности команды, размеров месячной заработной платы каждого из участников команды, а также общей трудоемкости разработки программного обеспечения.

Основная заработная плата исполнителей проекта определяется по формуле:

, (5.1)

где *n* − общее число функций;

− коэффициент премий (1,5-2,0);

− часовая заработная плата i-го исполнителя (руб.);

− трудоемкость работ, выполняемых i-м исполнителем (ч).

Программный продукт будет разрабатываться два месяца. В состав команды разработчиков будет входить: ведущий−разработчик, разработчик, тестировщик, дизайнер, бизнес−аналитик.

Таблица 5.1 – Расчет затрат на основную заработную плату

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Участник команды | Вид выполняемой работы | Месячная заработная плата, руб. | Часовая заработная плата, руб. | Трудоемкость работ, ч | Зарплата по тарифу, руб. |
| 1 Ведущий разработчик | Написание программного кода | 1600 | 9,5 | 336 | 3192 |
| 2 Разработчик | Написание программного кода | 1000 | 5,9 | 336 | 1982 |
| 3 Бизнес-аналитик | Разработка бизнес-модели | 1200 | 7,1 | 280 | 1988 |
| 4 Тестировщик | Тестирование приложения | 800 | 4,7 | 260 | 1222 |
| 5 Дизайнер | Разработка дизайна | 800 | 4,7 | 240 | 1128 |
| Сумма | | | | | 9512 |
| Премия, 50% | | | | | 4756 |
| Итого, руб | | | | | 14268 |

2 Определение затрат на дополнительную заработную плату команды разработчиков.

Затраты на дополнительную заработную плату команды разработчиков включает выплаты, предусмотренные законодательством о труде (оплата трудовых отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по формуле:

где – норматив дополнительной заработной платы, (10%);

– затраты на основную заработную плату, (руб.).

Найдем затраты на дополнительную заработную плату:

3 Определение отчислений на социальные нужды.

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами по формуле:

где – дополнительная заработная плата, (руб.);

– норматив отчислений в фонд социальной защиты населения и на

обязательное страхование, (34+0,6%).

4 Определение прочих затрат.

Прочие затраты включаются в себестоимость разработки ПО в процентах от затрат на основную заработную плату команды разработчиков по формуле:

где – норматив прочих затрат, (100 %).

Таблица 5.2 – Затраты на разработку программного продукта

|  |  |
| --- | --- |
| Статья затрат | Сумма, руб. |
| Основная заработная плата команды разработчиков | 14268 |
| Дополнительная заработная плата команды разработчиков | 1426,8 |
| Отчисления на социальные нужды | 5430 |
| Прочие затраты | 14268 |
| Общая сумма затрат на разработку | 35393 |

Из проведенных расчетов следует, что общая сумма затрат на разработку составила 35393 руб.

## **5.3 Расчет экономической эффективности реализации на рынке**

## **игрового аркадного программного средства «Run out»**

## **под операционную систему Android**

Программное средство планируется разместить в сети Интернет, пользователи смогут купить копию по цене, определенной на основе маркетинговых исследований рынка 2,09 руб. В среднем в год прогнозируется, что 50000 пользователей купят копию игры. Таким образом, разработчик получит выручку в размере 104500 руб. на продаже копий игры.

Налогооблагаемая прибыль, полученная разработчиком от реализации ПО на рынке, рассчитывается по формуле:

где НДС − налог на добавленную стоимость;

− Общая сумма затрат на разработку, руб.;

N − количество проданных копий игры;

– цена за копию игры.

где – норматив налога на добавленную стоимость, (20%).

Найдем налог на добавленную стоимость:

Найдем налогооблагаемую прибыль:

Чистая прибыль рассчитывается по формуле:

где – норматив налога на прибыль, (18%).

Найдем налог на прибыль:

Найдем чистую прибыль:

Для расчета рентабельности инвестиций в разработку используется формула:

где − чистая прибыль, руб.

Таким образом, все затраты на разработку программного средства окупятся менее чем за год от реализации программного средства в сети Интернет. Следовательно, реализация аркадного программного средства «Run out» под операционную систему Android на рынке является экономически эффективной и его разработку целесообразно осуществлять.

# **ЗАКЛЮЧЕНИЕ**

В рамках данной дипломной работы было разработано игровое аркадное программное средство «Run out» под операционную систему Аndroid. Была проанализирована информация о мобильных играх, жанрах, игровых движках. Было просмотрено большое количество литературы и интернет-источников. После разработки концепции игры, в ходе анализа четырех игровых движков, был выбран движок Unity3d, который удовлетворяют определённым требованиям:

– простота разработки;

– поддержка языков программирования С# и JavaScript;

– мультиплатформенность;

– бесплатная версия.

При работе со скриптами и проектной расстановке объектов на сцене был проделан тщательный анализ игр аналогов для поддержания жанра и выделении особенностей игры.

Получены большие знания в области разработке игровых приложений: работа с игровыми движками, работа с памятью мобильных устройств, оптимизация производительности. Также мной была изучена среда разработки Unity3D: возможности редактора движка, слабые и сильные стороны этого движка. Получен большой опыт разработки 3D сцен: изучены основные понятия, изучена и понята работа с текстурами как в 3D, так и в 2D. Приобретен опыт программирования на языке C#. Изучена и проанализирована библиотека Unity3D, составлены собственные классы, которые ее дополняют. Таким образом, все поставленные задачи были выполнены, цель работы достигнута.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Диабет [Электронный ресурс]. – Режим доступа: https://www.who.int/ru/news-room/fact-sheets/detail/diabetes
2. Мобильное приложение [Электронный ресурс]. – Режим доступа: https://indicator.ru/label/mobilnoe-prilozhenie
3. Мобильное приложение [Электронный ресурс]. – Режим доступа: https://www.mk.ru/social/health/2016/06/01/bolshinstvo-detey-s-sakharnym-diabetom-chuvstvuyut-sebya-neschastnymi.html
4. *Mobile OS market share worldwide* [Электронный ресурс]. – Режим доступа: https://gs.statcounter.com/os-market-share/mobile/worldwide
5. *React Native* [Электронный ресурс]. – Режим доступа: https://reactnative.dev
6. *React Native*: год спустя [Электронный ресурс]. – Режим доступа: https://engineering.fb.com/android/react-native-a-year-in-review/
7. Архитектура *React Native* приложений [Электронный ресурс]. – Режим доступа: https://2018.holyjs-piter.ru/talks/3wy7ejpjymg6cqu0igqywa/
8. *Flutter* для мобильных разработчиков [Электронный ресурс]. – Режим доступа: https://flutter.dev/docs/get-started/flutter-for/android-devs
9. *React Native, Flutter* [Электронный ресурс]. – Режим доступа: https://trends.google.ru/trends/explore?date=today%205-y&q=flutter,react%20native
10. Мобильная разработка: критерии выбора [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/simbirsoft/blog/460030/
11. *WebStorm* [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/webstorm/promo
12. *Meet Android Studio* [Электронный ресурс]. – Режим доступа: https://developer.android.com/studio/intro

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Отчёт по анализу заимствования материала**

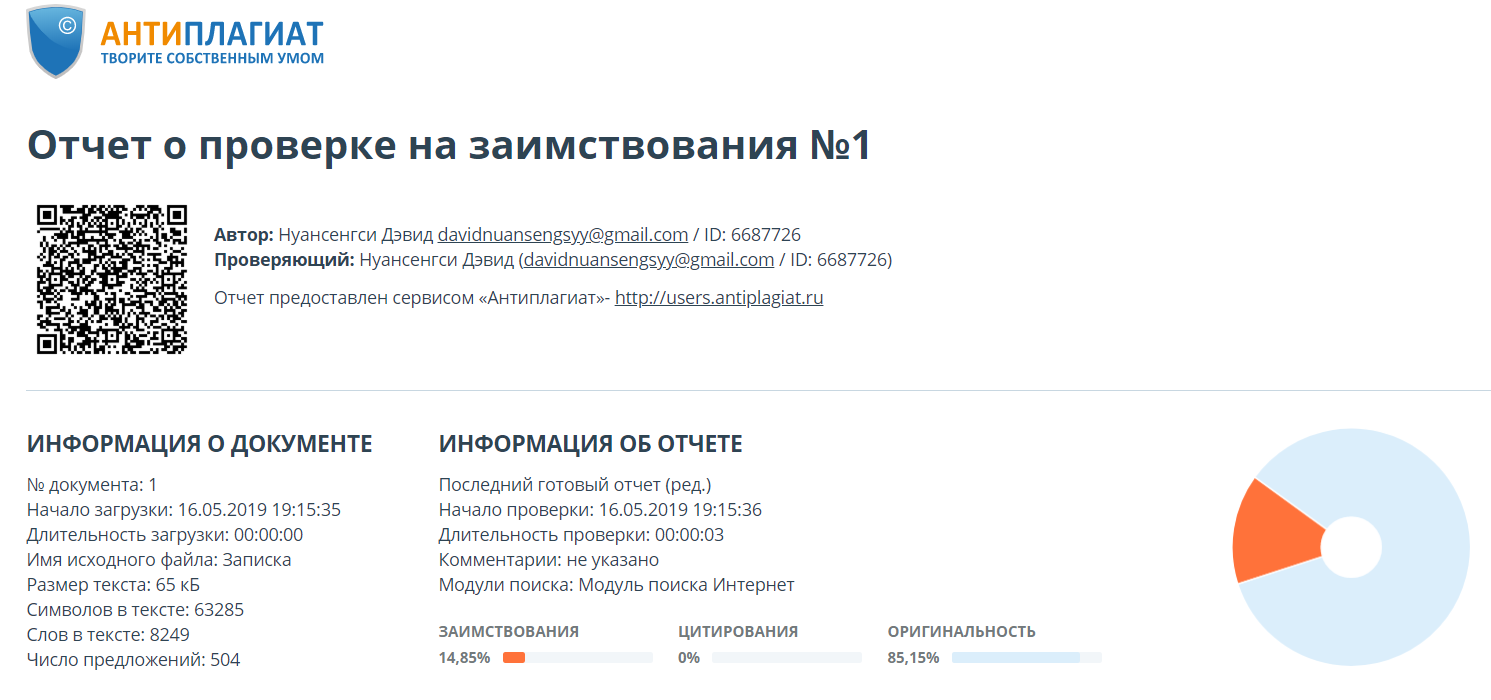


Рисунок А.1 – Скриншот отчета о проверке дипломного проекта в системе «Антиплагиат»

# **ПРИЛОЖЕНИЕ** **Б (обязательное)**

**Листинги программ**

# **ПРИЛОЖЕНИЕ В (обязательное)**

**Ведомость дипломного проекта**