Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление подготовки: 09.03.01 “Информатика и вычислительная техника”
Профиль: “Вычислительные машины, комплексы, системы и сети”

Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

К защите допустить:
Заведующий кафедрой
д. т. н., профессор _____ М. С. Куприянов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: “Мобильная 3D игра для Android на Unity”

Студент _____ Г. А. Петров

Руководитель
д. т. н., профессор _____ А. И. Водяхо

Консультант по экономическому обоснованию
д. э. н., профессор _____ Т. Д. Маслова

Консультант от кафедры
к. т. н., доцент, с. н. с. _____ И. С. Зуев

Санкт-Петербург
2022 г.

Санкт-Петербургский государственный электротехнический университет "ЛЭТИ" им. В. И. Ульянова (Ленина)
(СПбГЭТУ "ЛЭТИ")

Направление 09.03.01 "Информатика и вычислительная техника"
Профиль "Вычислительные машины, комплексы, системы и сети"
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
"___" _____ 2022 г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студент Петров Григорий Алексеевич Группа № 8308

1. Тема Мобильная 3D игра для Android на Unity

утверждена приказом № _____ от _____
Место выполнения ВКР: Санкт-Петербургский государственный электротехнический университет им. В. И. Ульянова (Ленина).

2. Объект и предмет исследования. Объект исследования – мобильная 3D игра для операционной системы Android на Unity. Предмет исследования – разработка мобильной игры под операционную систему Android с использованием межплатформенной среды разработки игр Unity и алгоритмов трассировки лучей.

3. Цель: разработка мобильной игры с использованием алгоритмов трассировки лучей и среды разработки Unity, запускаемой на мобильных устройствах под управлением операционной системы Android.

4. Исходные данные. Межплатформенная среда разработки игр Unity, алгоритмы трассировки лучей, а также документация средств разработки, статьи и видео с сети Интернет.

5. Содержание. Анализ задачи создания мобильной игры для Android, определение путей и методов разработки игры, разработка игры с использованием среды Unity, разработка пользовательского интерфейса, тестирование мобильной игры, экономическое обоснование разработки игры, заключение, приложение.

6. Технические требования. Разрабатываемая игра должна запускаться на мобильных устройствах использующих операционную систему Android. Мобильная игра должна быть разработана в межплатформенной среде разра-

ботки игр Unity на языке программирования C#. В игре должны использоваться алгоритмы трассировки лучей, а также реализована возможность создания пользовательских уровней игры.

7. **Дополнительный раздел:** экономическое обоснование выпускной квалификационной работы.
8. **Результаты.** Готовая к использованию трехмерная мобильная игра, которая может быть использована людьми на мобильных устройствах под управлением ОС Android, пояснительная записка, реферат, аннотация, презентация, исходные коды и исполняемый файл разработанной мобильной игры.

Дата выдачи задания
«_____» _____ 2022 г.

Дата представления ВКР к защите
«17» июня 2022 г.

Руководитель
д. т. н., профессор _____ А. И. Водяхо

Студент _____ Г. А. Петров

Санкт-Петербургский государственный электротехнический университет "ЛЭТИ" им. В. И. Ульянова (Ленина)
(СПбГЭТУ "ЛЭТИ")

Направление 09.03.01 "Информатика и вычислительная техника"
Профиль "Вычислительные машины, комплексы, системы и сети"
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
"___" _____ 2022 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

Тема Мобильная 3D игра для Android на Unity

Студент Петров Григорий Алексеевич Группа № 8308

№ этапа	Наименование работ	Срок выполнения
1	Обзор литературы по теме трассировки лучей и разработке мобильных игр	10.03.2022 – 15.03.2022
2	Анализ задачи создания мобильной игры для Android	15.03.2022 – 18.03.2022
3	Определение путей и методов разработки игры	18.03.2022 – 01.04.2022
4	Разработка мобильной игры с использованием среды Unity	01.04.2022 – 01.05.2022
5	Разработка пользовательского интерфейса игры	01.05.2022 – 13.05.2022
6	Выполнение экономического обоснования разработки игры	13.05.2022 – 15.05.2022
7	Оформление пояснительной записки	15.05.2022 – 25.05.2022
8	Предварительное рассмотрение работы	01.06.2022 – 02.06.2022
9	Представление работы к защите	05.06.2022

Руководитель
д. т. н., профессор _____ А. И. Водяхо
Студент _____ Г. А. Петров

РЕФЕРАТ

Пояснительная записка содержит: 86 страниц, 14 рисунков, 24 источника, 1 приложение и 6 таблиц.

Тема работы: Мобильная 3D игра на Unity.

Цель работы: Разработка мобильной игры с использованием алгоритмов трассировки лучей и среды разработки Unity, запускаемой на мобильных устройствах под управлением операционной системы Android.

Разрабатываемая мобильная игра в отличие от довольно известной мобильной игры «Shadowmatic» (продукта деятельности армянской студии Triada Studio Games) раскрывает явление преломления света с получением тени определенного цвета, а также дает возможность создавать интересные уровни для того, чтобы делиться ими с другими пользователями.

Результатом работы является готовая к использованию мобильная игра, которая может быть использована людьми на мобильных устройствах под управлением операционной системы Android.

ABSTRACT

In the final qualifying work, a mobile game is being developed using ray tracing algorithms and the Unity development environment for the Android operating system.

The mobile game being developed reveals the phenomenon of light refraction with the production of a shadow of a certain color, and also makes it possible to create interesting levels in order to share them with other users.

The result of the work is a ready-to-use mobile game that can be launched by users on mobile devices running the Android operating system.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

IT – Information technology, информационные технологии.

ОС (OS) – операционная система (operating system).

IOS (iPhone OS) – мобильная операционная система для смартфонов, электронных планшетов, носимых проигрывателей, разрабатываемая и выпускаемая американской компанией Apple.

Android (Андроид) – мобильная операционная система для смартфонов, интернет - планшетов, электронных книг, наручных часов, игровых приставок, нетбуков, телевизоров и других устройств.

Трассировка лучей (рейтрейсинг) – технология построения изображения трёхмерных моделей в компьютерных программах, при которых отслеживается обратная траектория распространения луча (от экрана к источнику).

Интегрированная среда разработки (ИСР, IDE, среда разработки) – комплекс программных средств, используемых для разработки программного обеспечения.

Unity – межплатформенная среда разработки игр, разработанная американской компанией Unity Technologies.

Игровой процесс (геймплей) – компонент игры, отвечающий за взаимодействие игры и игрока. Геймплей описывает, как игрок взаимодействует с игровым миром, как игровой мир реагирует на действия игрока и как определяется набор действий, который предлагает игроку игра.

Сборка – подготовленный для использования информационный продукт, который чаще всего является исполняемым файлом – двоичный файл, содержащий исполняемый код (машинные инструкции) программы или библиотеки.

Игровая механика – набор правил и способов, реализующий определённым образом некоторую часть интерактивного взаимодействия игрока и игры.

Головоломка – название жанра компьютерных игр, целью которых является решение логических задач, требующих от игрока задействования логики, стратегии и интуиции.

Игровой мир – среда, в которой происходит действие игры.

FPS – Frame per second или количество кадров в секунду.

Фреймворк – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Игровой движок – базовое программное обеспечение компьютерной игры.

Рендеринг – процесс получения изображения по модели с помощью компьютерной программы.

ВВЕДЕНИЕ

Одним из направлений IT, активно развивающихся на территории России, является игровая индустрия. Особенно выделяется российский рынок мобильных игр, [1] объем которого ежегодно растет.

Первой мобильной игрой в истории стала [2] в 1994 году предустановленная на GSM-телефон Hagenuk MT-2000 мобильная игра «Тетрис». Именно с этого момента началась невероятная популярность мобильных игр, на волне которой каждая компания, выпускающая мобильные телефоны, старалась снабдить свой продукт мобильными играми. Однако на данном этапе у индустрии мобильных игр, несмотря на ее популярность, все еще оставалось много недостатков.

Решением стало появление [2] в 2008 году мобильных магазинов приложений «App Store» и «Android Market» (впоследствии переименованного в «Google Play»), которые позволяли публиковать и устанавливать различные мобильные приложения для операционных систем IOS и Android соответственно. С этого момента начался серьезный рост рынка мобильных игр, поскольку наличие общих магазинов мобильных приложений, давало практически неограниченные возможности для их распространения почти любому разработчику, как бесплатно, так и за деньги.

Алгоритм трассировки лучей, как метод получения трехмерного изображения, уже давно известен и применяется для улучшения реалистичности изображений в различных компьютерных приложениях, в том числе и в играх. Однако у этого метода есть один существенный недостаток – низкая производительность. Именно этот недостаток долгое время не давал возможность использовать рейтрейсинг на мобильных устройствах. Однако, [3] в ноябре 2019 года появилась первая новость от Huawei и NetEase о мобильной игре «Love is Justice», использующей рейтрейсинг. Стремительное развитие аппаратных со-

ставляющих мобильных устройств приводит к все более частому использованию рейтрейсинга на мобильных устройствах. Таким образом идея использования рейтрейсинга в мобильных играх является как никогда актуальной.

Разработка мобильной игры зачастую не требует серьезных денежных вложений и, в подтверждение этому, многие, ставшие популярными, мобильные игры изначально создавались маленькими компаниями или и вовсе одним разработчиком. При этом мобильные игры способны приносить серьезные денежный доход за счет встроенной рекламы, внутриигровых покупок или и вовсе платного приобретения игры, что делает разработку мобильной игры невероятно выгодным делом. На данный момент Android является наиболее популярной в России операционной системой и доля мобильных устройств, использующих OS Android, [4] в России превышает 70%. Очевидно, что начинающему разработчику мобильных игр стоит в первую очередь ориентироваться именно на эту ОС и ее мобильный магазин приложений «Google Play».

Таким образом можно с уверенностью сказать, что разработка мобильных игры с рейтрейсингом является выгодным и актуальным занятием. Именно поэтому данная работа посвящена разработке мобильной 3D игры для ОС Android с использованием рейтрейсинга.

Целью данной работы является разработка мобильной игры с использованием алгоритмов трассировки лучей и среды разработки Unity, запускаемой на мобильных устройствах под управлением ОС Android.

Объектом исследования является мобильная 3D игра для ОС Android на Unity. Предметом исследования является разработка мобильной игры под ОС Android с использованием межплатформенной среды разработки игр Unity и алгоритмов трассировки лучей.

В данной работе преследуются задачи, заключающиеся в изучении межплатформенной среды разработки игр Unity и алгоритмов трассировки лучей, с последующим их применением для разработки мобильной 3D игры под ОС Android.

Создаваемая видеоигра позволит выйти на рынки мобильных игр, за счет отсутствия аналогов в плане игрового процесса, основанного на трассировке лучей, и разработке под наиболее популярную мобильную операционную систему Android.

В первой главе описываются цели и требования, выдвигаемые к разрабатываемой игре, современные среды разработки игр и функциональные возможности среды Unity. Во второй главе определяются пути и методы разработки игры, удовлетворяющие выдвигаемым требованиям, а именно, производится выбор операционной системы, среды разработки игр и определяются методы реализации трассировки лучей. Разработка мобильной игры с использованием Unity и пользовательского интерфейса для нее рассматриваются в третьей и четвертой главе соответственно. В пятой главе проводится тестирование разработанной мобильной игры. В шестой главе производится экономическое обоснование разработки мобильной игры, с подробным расчетом всех затрат и определением возможных доходов. Заключение содержит основные вывод по проделанной работе.

1 Анализ задачи создания мобильной игры для Android

Игры на мобильных устройствах, несмотря на и так невероятно высокие показатели, продолжают все больше набирать свою популярность с каждым годом [1], появляющаяся популярность использования алгоритмов рейтрейсинга на мобильных устройствах начинает набирать обороты. При этом, зачастую, начинающие разработчики мобильных игр ориентируются на ОС Android, но в последствии создают сборку приложения и под IOS.

Разработка мобильного приложения под ОС Android с использованием рейтрейсинга выгодна не только для производителя, но и для пользователей тоже. Поскольку более 70% пользователей мобильных устройств России используют именно ОС Android [4], то именно приложение под эту операционную систему сможет удовлетворить наибольшую часть пользователей. Использование же рейтрейсинга в приложении на мобильном устройстве дает пользователю массу плюсов, начиная от улучшения реалистичности изображений в приложении, и заканчивая использованием новых функций приложения, которые дает рейтрейсинг.

В свою очередь для начинающих разработчиков выгоднее всего начать со сборок под ОС Android, поскольку именно так они смогут покрыть наибольшую аудиторию при наименьших затратах. Использование рейтрейсинга на мобильных устройствах становится все более популярным занятием, в первую очередь по причине улучшения реалистичности графики, а во вторую очередь из-за возможности демонстрации вычислительных возможностей аппаратной составляющей мобильного устройства. Так, например, производитель мобильных устройств Huawei и разработчик приложений NetEase объединились, с целью использовать рейтрейсинг в приложении на мобильном устройстве [3] для демонстрации вычислительных возможностей продукции Huawei, а также для поднятия популярности самого приложения. При всем этом, использование межплатформенной среды разработки, дает возможность при незначительных изменениях исходного кода, создавать сборки приложения под различные ОС.

1.1 Преимущества написания мобильной игры для Android в межплатформенной среде разработки с рейтрейсингом

Основные преимущества написания мобильной игры для ОС Android в межплатформенной среде разработки с использованием рейтрейсинга:

- Рынок мобильных игр является наиболее бурно развивающими на данный момент рынком игр в России [1].
- Более 70% пользователей России имеют мобильное устройство под управлением ОС Android [4], что делает эту ОС наиболее распространенной, поэтому начинать распространение приложения с нее наиболее выгодно.
- Использование алгоритмов рейтрейсинга в приложение способно серьезно поднять уровень графики, сделав ее более реалистичной и/или интересной.
- Помимо поднятия уровня графики, идеи рейтрейсинга могут использоваться для создания новых интересных механик в игре и расширения функционала.
- Учитывая невероятную популярность самой идеи рейтрейсинга, ее использование в мобильной игре положительно отразится на популярности самой игры.
- Написание кода приложения в среде разработки игр обеспечивает программиста легким доступом к различным плагинам, модулям и функциям среды разработки и упрощает их внедрение в итоговый продукт.
- Также использование среды разработки игр серьезно ускоряет процесс создания и оптимизации игры.
- Межплатформенная среда разработки в свою очередь дает возможность практически без изменений в исходном коде приложения создавать сборки под самые различные платформы, в том числе под Android и IOS.
- В перспективе межплатформенная среда разработки дает возможность выхода не только на рынок Android приложений, но и на IOS рынок.

На основании всего вышеперечисленного можно сделать вывод, что для начинающего разработчика игр наиболее выгодным будет ориентироваться на разработку под ОС Android, как наиболее популярную и распространенную, при этом по возможности выгодным решением стало бы использование наиболее популярной на данный момент технологии рейтрейсинга. Создавать такую мобильную игру удобнее всего в одной из сред разработки игр, которая, желательно, должна быть межплатформенной, для перспективы сборки приложения под все популярные платформы рынка мобильных игр.

1.2 Основные цели и требования, выдвигаемые к разрабатываемой мобильной игре

Исходя из технического задания к данной выпускной квалификационной работе, нужно разработать в межплатформенной среде разработки игр Unity на языке программирования C# мобильную игру, которая должна запускаться на мобильных устройствах использующих операционную систему Android, при этом необходимо использовать в мобильной игре алгоритмы трассировки лучей, а также реализовать возможность создания пользовательских уровней игры. Мобильная игра должна носить название «PuzzleVitrage» и являться трехмерной однопользовательской мобильной игрой в жанре головоломки. Целью мобильной игры будет прохождение уровней, для чего необходимо расположить вращением трехмерный полупрозрачный объект, таким образом, чтобы свет, проходящий через грани этого объекта образовывал цветную тень, удовлетворяющую поставленной на уровне задаче.

Игровой мир включает в себя направленный источник света, комнату со стенами и полый объект, состоящий из разноцветных полупрозрачных граней. Все эти объект игрового мира должны быть расположены в нем таким образом, чтобы свет от источника, проходя через объект, окрашивался в цвет пересеченной грани и, попадая на непрозрачную стену комнаты, образовывал цветную тень объекта. Из-за того, что перемещение и вращение камеры внутри игрового мира невозможно, необходимо учитывать, чтобы объект и его цветная

тень на стене комнаты одновременно располагались на экране мобильного устройства, при этом не пересекаясь друг с другом. Форма фигуры, цвета ее граней, текстуры стен и потолка комнаты могут меняться от уровня к уровню.

Пользователю данной мобильной игры дана возможность взаимодействовать с игровым миром, посредствам вращения полупрозрачного объекта по трем главным координатам x, y и z. Для прохождения уровня игры, пользователю необходимо при помощи этого вращения, получить на стене комнаты тень полупрозрачного объекта, изображение которой должно быть идентичным целевому изображению для данного уровня. Целевое изображение все время игры доступно в качестве подсказки на экране мобильного устройства. По мимо описанного режима прохождения уровней, в мобильной игре для пользователя будет доступен режим создания уровней. В данном режиме пользователю будет предлагать возможность выбрать текстуру комнаты, форму полупрозрачной фигуры и цвета граней данной фигуры, после чего задать вращением целевую тень объекта и ввести название создаваемого уровня.

1.2.1 Требования к функциональным характеристикам

- Исходный код необходимо написать на языках программирования C# и HLSL под ОС Android.
- Управление в мобильной игре необходимо осуществлять посредствам сенсорной системы ввода (сенсорного экрана).
- Игровая механика быть легкой в понимании для пользователя.
- Графическая составляющая игрового процесса, не должна вызывать неприятные ощущения в глазах пользователя яркими и контрастными цветами, при этом необходимо расположить все объекта на экране таким образом, чтобы они не перекрывали друг друга и не мешали игровому процессу, для чего стоит ограничиться минимальным количеством единовременно находящихся на экране объектов.
- Все экранные объекты пользовательского интерфейса и управления игровым процессом необходимо выполнить в общем, выдержанном стиле,

управление пользовательским интерфейсом должно быть интуитивно понятным.

• В мобильной игре необходимо разработать два режима: прохождение уровней и создание уровней.

• Необходимо задействовать сохранение прогресса пользователя в прохождении уровней мобильной игры, с возможностью выбора, какой уровень загрузить из уже пройденных или созданных самостоятельно, и возможностью полного сброса общего прогресса.

• Игровой процесс необходимо реализовать так, чтобы игровой мир в реальном времени реагировал на воздействие пользователя.

• В режиме прохождения уровней пользователь может воздействовать на игровой мир только по средствам вращения полупрозрачного объекта по трем главным координатам x, y и z. Любой другой способ взаимодействия с игровым миром не предусмотрен.

• В режиме создания уровней пользователь может через интерфейс и вращение по координатам x, y и z настраивать параметры уровня. После чего сохранять уровень и открывать его в режиме прохождения.

1.2.2 Требования к организации и форме представления выходных данных

Выходными данными мобильной игры должны являться графические отображения на экране мобильного устройства объектов игрового мира в формате 3D реального времени и пользовательского интерфейса, в легком для восприятия пользователем виде.

1.2.3 Требования к организации и форме представления входных данных

Входными данными для управления мобильной игрой должны являются положения прикосновения пользователя к сенсорному экрану устройства, и,

как следствие, определение задействованных интерактивных составляющих интерфейса (кнопок, ползунков и т. д.) или джойстиков.

1.2.4 Требования к надежности игры

Мобильная игра должна устойчиво (непрерывно без сбоев) работать на мобильном устройстве под управлением ОС Android, при условии устойчивой работы аппаратной составляющей устройства и операционной системы.

1.2.5 Условия эксплуатации игры

Условия эксплуатации должны соответствовать условия эксплуатации мобильного устройства, которое используется для запуска данной игры, при этом мобильная игра должна быть однопользовательской.

1.2.6 Требования к составу и параметрам технических средств

Данная мобильная игра должна запускаться на сенсорных смартфонах под управлением ОС Android версии не ниже 9.0, с процессором ARMv7 с поддержкой Neon (32-бит) или ARM64, поддержкой OpenGL ES 3.0+ или Vulkan [5] минимальной тактовой частотой GPU 650MHz и вышедших после 2018 года.

1.2.7 Требования к программной платформе и средствам разработки

- Операционная система – Android.
- Языки программирования – C#, HLSL.
- Средства разработки – Unity, Visual Studio, Blender.

1.2.8 Требования к информационной и программной совместимости

Информация, поступающая на вход и выход мобильной игры, должна легко восприниматься глазами пользователем. Необходимо использовать языки программирования C# и HLSL для написания исходного кода мобильной игры, при этом она должна успешно запускаться в ОС Android. Также в

исходном коде игры необходимо использовать такие конструкции языка программирования, как закрытые и открытые члены классов, статические классы, наследование, конструкторы, базовые классы, пространства имен.

1.2.9 Пользовательские требования

Реализуемая мобильная игра должна в однопользовательском режиме отображать трехмерный игровой мир с достаточной степенью реалистичности и качества графики. Мобильная игра должна быть выполнена в жанре головоломки и иметь удобное управление и понятную механику игры. Так же должна иметься возможность создания пользовательских уровней и возможность запуска как базовых уровней, поступающих вместе с игрой, так и сторонних уровней из файлов, созданных внутри данной игры.

1.2.10 Описание системной архитектуры

Мобильная игра содержит два модуля: модуль рейтрейсинга – реализующий отображение трехмерного игрового мира и логический модуль – реализующий всю механику игры. Логический модуль помимо механики игры, также содержит управление пользовательским интерфейсом, который содержит: главное меню, меню паузы, меню настроек, меню выбора уровня, главное меню игры, меню создания уровня и меня финиша.

1.2.11 Предполагаемые направления дальнейшего развития игры

Дальнейшее развитие игры возможно в сторону расширения количества текстур для комнат уровня и фигур для объекта вращения. Так же возможно дальнейшие усовершенствование визуализации игрового мира посредствам рейтрейсинга с сохранением или даже увеличением FPS. Учитывая использование межплатформенной среды разработки Unity, в дальнейшем предполагается выход приложения под другие платформы, например, IOS.

1.2.12 Требования к нефункциональным характеристикам игры

• Использование алгоритмов рейтрейсинга для визуализации трехмерного игрового мира.

• Хорошие показатели производительности (количество кадров в секунду). В среднем во время работы игры на мобильных устройствах, удовлетворяющих требованиям к составу и параметрам технических средств, на минимальных настройках качества графики и при наличии в кадре фигуры с двадцатью полигонами не ниже 10 fps.

• Исходный код мобильной игры должен быть написана с учетом возможности дальнейшего расширения игровой логики.

• Собранная мобильная игра не должна вызывать сложности у пользователя при ее установке и запуске.

• Созданные уровни должны храниться в универсальном формате так, чтобы уровень созданный в данной игре на одном мобильном устройстве, мог легко быть загружен внутри игры на другом мобильном устройстве.

1.2.13 Оценка реализуемости игры

В первую очередь воздаваемая мобильная игра позволит выйти на рынок мобильных Android игр, за счет отсутствия аналогов в плане игрового процесса, поскольку, не смотря на тот факт, что игровой процесс основан на известной мобильной игре «Shadowmatic» (продукт деятельности армянской студии Triada Studio Games), в данной игре он сильно расширяется за счет введения цветных теней, которые дают массу новых возможностей. Использование алгоритмов рейтрейсинга в игре на мобильном устройстве повышает актуальность и заинтересованность пользователей в продукте, на волне популярности рейтрейсинга, а также дает возможность реализовать механику цветных теней. Использование межплатформенной среды разработки игр Unity и распространение созданных в ней игр является абсолютно бесплатным, пока объем привлеченных инвестиций не превышает 100 тыс. $ за последние 12 месяцев. При этом на данном мобильной игре можно хорошо заработать, за счет внедрения в игру рекламных баннеров. Написание игры в межплатформенной среде

также даст возможность в дальнейшем практически без усилий создать сборки приложения под другие платформы. Если видеоигра не будет создана в текущем году, то повышается вероятность того, что кем-то будет выпущена видеоигра, аналогичная данной, что в свою очередь помешает стать лидером данной ниши.

Из всего вышеперечисленного можно сделать вывод, что текущий уровень технологий позволит относительно быстро и без существенных затрат, разработать описанную мобильную игру и выйти на рынок мобильных Android игр. Поэтому создание подобной мобильной игры, удовлетворяющей всем вышеописанным требованиям является вполне реализуемой задачей.

1.2.14 Требования к выполняемой работе

Для выполнения поставленной задачи, необходимо:

- изучить межплатформенную среду разработки игр Unity;
- выполнить обзор алгоритма рейтрейсинга и существующих игровых сред разработки;
- разработать и написать исходный код мобильной игры для платформы Andriod.

На этапе проектирования производится развернутое описание функциональных требований к приложению. Для более четкого описания требований, нужно определить, какие именно процессы происходят в приложении. Для более наглядного их описания используют UML диаграмму прецедентов (use case). Согласно книге [6] А. В. Леоненкова «Самоучитель UML», такая диаграмма описывает то, что будет делать приложение в процессе своей работы, как оно будет функционировать, другими словами сценарии и варианты использования игры пользователем. Такого пользователя в рамках UML принято называть актором. Более общее и формальное определение описано в «Самоучителе UML» [6]: Актор представляет собой любую внешнюю по отношению

к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Зная все это и следуя инструкциям «Самоучителя UML», определены прецеденты, акторы и взаимосвязи между ними, на основе чего построена use case диаграмма, представленная на рисунке 1.1.

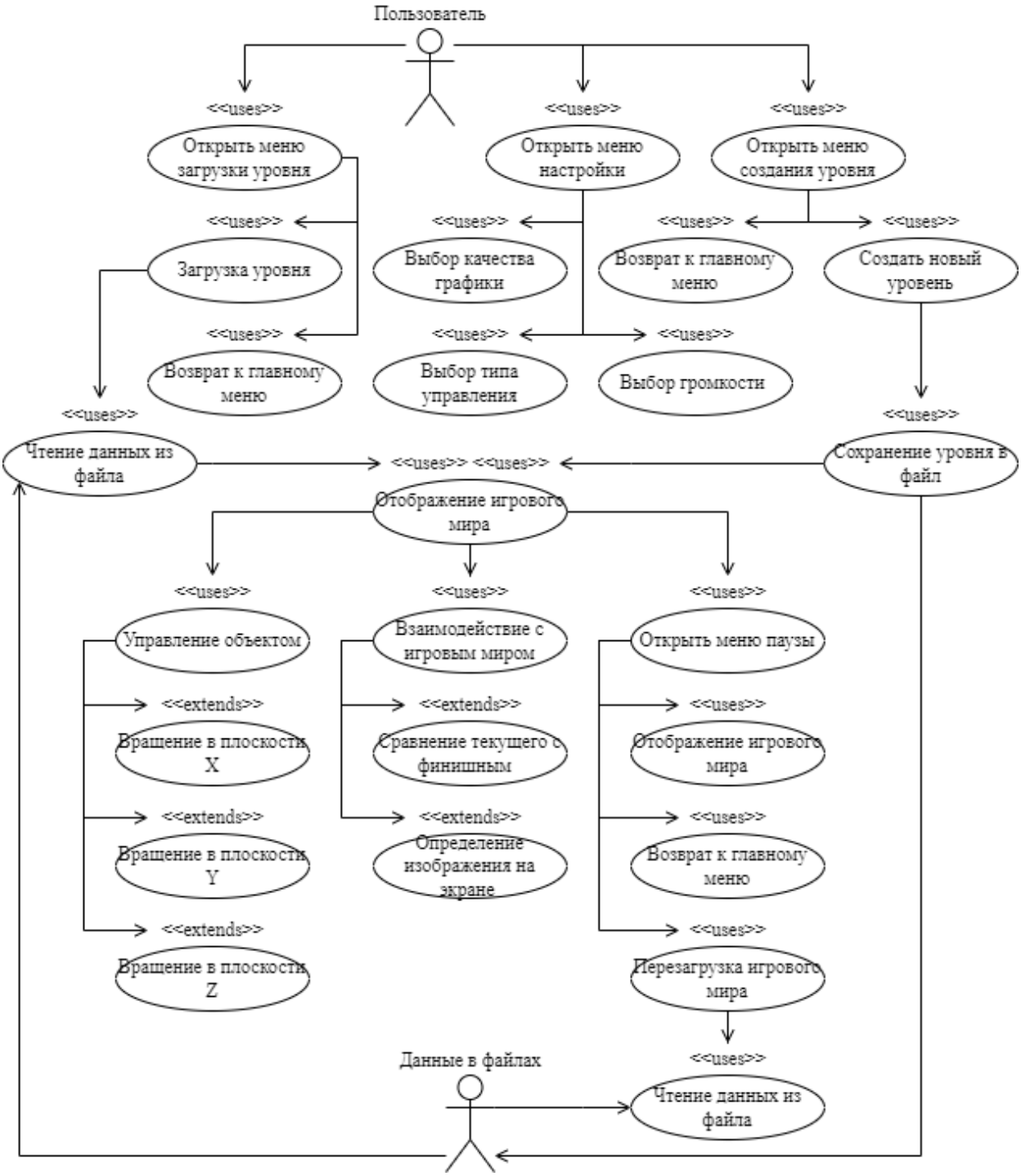

Рисунок 1.1 – Диаграмма прецедентов мобильной игры (use-case)

1.3 Функциональная возможности Unity

Безусловно в условиях современной реальности необходимо уделять особое внимание выбору программного обеспечения для разработки. В первую очередь необходимо обращать внимание на сочетание функциональных возможностей и стоимости, но также важным является факт активной поддержки программного обеспечения со стороны производителя, ведь без постоянных обновлений и развития, программное обеспечение устареет и перестанет удовлетворять реальным потребностям, а вместе с ним и приложение, созданное на основе него.

Далее в краткой форме приведено описание основных функциональных возможностей межплатформенной среды разработки Unity версии 2020.3 и выше, на основании информации предоставляемой на официальном сайте Unity [7] и сайте их официальной документации Unity Documentation [5].

1.3.1 Графика

- Конвейеры рендеринга.
- Камеры.
- Постобработка.
- Освещение.
- Модели. Основные форматы моделей, которые поддерживает Unity: «.fbx», «.dae» (Collada), «.dxf», «.obj».
- Меши.
- Материалы.
- Текстуры.
- Шейдеры.
- Системы частиц.
- Создание сред. Инструменты, которые позволяют создавать объекты окружающей среды, такие как рельеф и растительность.
- Небо, как тип фона для камеры.

• Визуальные эффекты, такие как: полноэкранные эффекты, системы частиц, проекторы, вспышки и ореолы объектива и т. д.

• Оптимизация производительности графики.

• Цветовые пространства: линейное и гамма.

• Поддержка графического API в зависимости от платформы: DirectX, Metal, OpenGL и Vulkan.

1.3.2 Аудио

• Полный пространственный 3D-звук.

• Микширование и мастеринг в реальном времени.

• Иерархии микшеров.

• Моментальные снимки.

• Предопределенные системы.

• Источники звука и слушатели.

• Поддерживаемые форматы: «.mp3», «.ogg», «.wav», «.aif», «.mod», «.it», «.s3m», «.xm».

1.3.4 Ввод

• Клавиатуры и мыши.

• Джойстики.

• Контроллеры.

• Сенсорные экраны.

• Датчики движения мобильных устройств, например, акселерометры или гироскопы.

• Контроллеры VR и AR.

1.3.5 Физика

• Встроенная 2D и 3D физика.

• Управление персонажем.

• Физика твердых тел.

- Столкновения.
- Физика суставов.
- Физика ткани.

1.3.6 2D функционал

- 2D геймплей.
- 2D физика.
- Спрайты.
- Tilemap для создания 2D уровней.

1.3.7 Анимации

- Встроенный редактор анимации.
- Перенастраиваемые анимации.
- Контроль весов анимации в реальном времени.
- События в анимации.
- Сложные конечные автоматы иерархии и переходов.
- Анимации лица.

1.3.8 Поддержка разработки под разные платформы

- Windows, macOS и Linux.
- tvOS.
- iOS.
- Android.
- WebGL.

1.3.9 Скрипты

- Поддержка интегрированной среды разработки.
- Отладка кода средствами Unity.
- Модульное тестирование.
- Собственная архитектура Unity.

• Важные и общие классы для наследования: GameObject, MonoBehavior, Object, Transform и другие.

1.3.10 Пользовательский интерфейс

• Система пользовательского интерфейса с сохраненным режимом, содержащая основные функции и функциональные возможности, необходимые для создания пользовательских интерфейсов.

• Использование HTML, XML и CSS для структурирования и стиля пользовательского интерфейса.

• Холст.

• Базовые компоновки.

• Визуальные компоненты.

• Компоненты взаимодействия.

• Интеграция анимации.

• Система событий.

• Raycasters.

1.4 Современные среды разработки игр

На сегодняшний день невозможно представить себе разработку приложения, которое бы не основывалось на фреймворке или среде разработки. Основной причиной тому является нежелание расходовать время и деньги на создание сырых версий базовых и необходимых каждому решений, которые уже давно созданы и отточены до идеала в различных фреймворках. К тому же помимо прочего, среды разработки и фреймворки дают множество дополнительного функционала, который может быть добавлен в приложение или использован для упрощения процесса разработки. Помимо прочего, на данный момент большинство сред разработки дают возможность быстро и без серьезных затрат переориентировать приложение с одной платформы на другую, тем самым помогая выходить на новые рынки приложений. При разработке игр использование сред разработки, которые так же называют движками, становится

еще более полезным, поскольку зачастую для обычной игры не требуется какой-то особенный функционал движка, и становится выгоднее использовать готовое решение, которое, например, может включать в себя реализацию физики, столкновений, графики и визуализации игры.

Согласно книге Грегори Джейсона [8], сам термин «игровой движок» появился в 1990 году в отношении шутеров от первого лица, таких как популярная в то время игра Doom компании «id Software». Именно тогда разработчики начали брать за основу движок известной и успешной игры и с минимальными изменениями создавать новые игры. Уже к концу 1990 годов, большинство компаний стали разрабатывать игры таким образом, чтобы оставлять возможность повторного использования движка этих игр, такие как Quake III Arena и Unreal. Некоторые компании даже лицензировали свои игровые движки и языки сценариев для них, так и началась эпоха многократного использования игровых движков.

Такие фреймворки, среды разработки и движки пользуются большой популярностью как у одиночных разработчиков, так и у серьезных компаний. К тому же у большинства сред разработки есть свои поклонники, как в России так за рубежом, что порождает множество обучающих видео, статей, книг и курсов. Благодаря этому начинающему разработчику, незнакомому с тем или иным решением, легко изучить его и начать заниматься разработкой приложений и игр.

К тому же большинство компаний в России используют популярные фреймворки и среды разработки для своих программных продуктов, что, наряду с большим количество обучающего материала, позволяет без затруднений стать востребованным специалистом на рынке труда.

1.4.1 Классификация сред разработки игр

Ниже приставлены основные параметры сред разработки игр.

• Основной язык программирования. Безусловно этот параметр является одним из главный при выборе среды разработки игры. Так наиболее распространёнными вариантами использующихся языков программирования являются c++, c#, java и python. Отдельно стоит выделить среды разработки, которые предназначены не для программистов, такие как Game Salad Creator и Scratch, и в которых для разработки игры достаточно использовать инструменты интерфейса среды разработки [8]. Однако такие среды сильно ограничивают получаемый результат.

• Размерность поддерживаемых измерений. Обычно выделают 2D и 3D среды разработки. Среды разработки, основанные на двумерном игровом движке ограничены созданием простых двумерных игр, однако они являются менее требовательными к вычислительным мощностям и подойдут, например, для простых мобильных игр. Трехмерные же движки имеют большие аппаратные требования, но при этом дают огромное пространство возможностей для создания самых разных игр всех жанров.

• Поддерживаемые платформы. С учетом современных реалий, остро стоит вопрос выбора целевых платформ разработки, исходя из выбора, определяется, какую среду разработки стоит использовать. Большинство сред разработки тем не менее, стараются давать возможность создавать решения под самые разные платформы, однако некоторые являются более узкоспециализированными, например, позволяющие создавать игры исключительно для ОС Android и IOS. Среды разработки, специализирующиеся на нескольких платформах принято называть межплатформенными или кроссплатформенными.

• Доступность. Для любого разработчика важным фактором выбора является доступность среды разработки. Существуют как движки с открытым кодом, являющиеся бесплатными и дающие свободный доступ к своему исходному коду, так и движки, находящиеся в частной собственности. Однако большинство движков все-таки являются лицензированным продуктом компаний,

которые выставляют свой ценник за их использование. Необходимо внимательно изучать тарифы на использование сред разработки, так некоторые среды являются бесплатными до определенного количества дохода, полученного на созданном их средствами продукте.

1.4.2 Список игровых движков

Список некоторых популярных игровых движков: Quake, Source, Unreal Engine, Glacier 2, DICE Frostbite, RAGE, Cryengine, Sony PhyreEngine, Esenthel Engine, Microsoft XNA Game Studio, Unity, Shiva3D, Tombstone, LeadWerks, HeroEngine, OGRE, Neoaxis, Panda3D, Yake, Serious Engine, Crystal Space, Game Guru, Lumberyard, Irrlicht, Torque, Multimedia Fusion 2, Game Salad Creator, Scratch.

1.4.3 Описание популярных сред разработки игр

Приведем описание наиболее популярных на данный момент игровых движков и сред разработки игр.

• Unreal – мощный движок и широкая среда разработки игр, которая начала свое развитие в 1998 году. Unreal содержит множество функций и универсальных инструментов для разработки, а так же удобный интерфейс. Так же выделятся наличием графического интерфейса создания игровой логики – Blueprints [8]. Для использования требуется небольшая ежемесячная плата, либо использование урезанной версии, а также взимается процент от прибыли за разработанную игру.

- Разработчик: Epic Games, Inc.
- Последняя версия: 5.0 (5 апреля 2022 года).
- Поддерживаемые платформы: MacOS, Microsoft Windows, Linux, iOS, Android.
- Основной язык разработки: C++.

• Cryengine – это игровой движок, впервые созданный в 2002 году. Отличается серьезным упором на реалистичность графики, для чего в Cryengine

присутствует множество инструментов и функций. К сожалению, за использование данной среды разработки тоже необходимо вносить ежемесячную плату.

- Разработчик: Crytek.
- Последняя версия: 5.7 (3 апреля 2022 года)
- Поддерживаемые платформы: Microsoft Windows, Linux, Oculus Rift, PlayStation 4, Xbox One, iOS и Android.
- Основной язык разработки: C++, C#.

• Sony PhyreEngine – выпущенный в 2008 году игровой движок, изначально нацеленный на платформы PlayStation. Отличается мощным редактором мира, оптимизацией графики и тонкой настройкой освещения. Хотя данная среда разработки и не является свободной, однако она распространяется бесплатно и с открытыми исходными кодами для всех разработчиков, которые желают разрабатывать игры под платформу PlayStation 3.

- Разработчик: Sony Interactive Entertainment.
- Последняя версия: 3.5 (27 марта 2013 года).
- Поддерживаемые платформы: PlayStation 5 PlayStation 4, PlayStation 3, PlayStation VR, PlayStation Vita, PlayStation Portable, Microsoft Windows, Google Android, iOS, Xbox One, Nintendo Switch.
- Основной язык разработки: C++, C.

• Unity – это игровой движок и мощная межплатформенная среда разработки игр. Основной особенностью Unity является простота в использовании, наряду с невероятно широким спектром возможностей и функционала. При всем этом Unity поддерживает разработку игр почти под все существующие платформы и написание исходного кода на нескольких распространенных языках. Однако за его использование придет вносить небольшую ежемесячную плату, либо использовать бесплатную версию, до момента превышения лимита количества привлеченных инвестиций в игру.

- Разработчик: Unity Technologies.
- Последняя версия: 2021.3.2f1 (6 мая 2022 года).

- Поддерживаемые платформы: Windows, macOS, Linux, tvOS, iOS, Android, WebGL, PlayStation, Xbox, Nintendo Switch, Stadia, Oculus, PlayStation VR, Google's ARCore, Apple's ARKit, Windows Mixed Reality, Magic Leap, Steam VR, Google Cardboard.
- Основной язык разработки: C#, Java-Script, HLSL.

• Game Salad Creator – это среда разработки игр, которая предназначена для людей, не имеющих опыта в программировании, потому что вся разработка игры сводится к работе в визуальным редактором среды. При всем этом данная среда разработки является кроссплатформенной и позволяет быстро, хотя и с ограниченной функциональностью, создавать простые игры. Отлично подойдет для создания простых игр или обучения созданию игр для начинающих геймдизайнеров.

- Разработчик: GameSalad.
- Последняя версия: 1.25.103.
- Поддерживаемые платформы: MacOS, Windows, iPhone, Android.
- Основной язык разработки: графический интерфейс среды.

• Panda3D – игровой движок для 3D игр, который содержит все необходимые базовые составляющие игрового движка. Хотя по функционалу Panda3D и уступает таким гигантам как Unity и Unreal, зато Panda3D является свободным программным обеспечением с открытым исходным кодом. Выделает этот игровой движок, по мимо всего прочего, возможность написания логики игры на языке скриптов Python, хотя ядро движка и написано на C++.

- Разработчик: Disney Interactive, CMU.
- Последняя версия: 1.10.11 (7 январь 2022).
- Поддерживаемые платформы: Windows, Linux, MacOS, Android, WebGL.
- Основной язык разработки: Python, C++.

1.4.4 Вывод о средах разработки игр

Безусловно каждая компания, разрабатывающая игры, имеет свои возможности и бюджет, так, например, самые крупные компании могут позволить себе создавать собственный движок для игры с целью его дальнейшего повторного использования или и вовсе продажи для использования другими компаниями. Компании среднего размера готовы платить крупные суммы денег за использование движков с самыми передовыми технологиями в области графики и игровых механик. Однако начинающим разработчикам созданы все условия для старта их проектов, так большинство сред разработки игр предоставляют разработчикам бесплатные версии, до достижения определенного уровня стабильного дохода с созданной игры, или и вовсе имеют открытый исходный код. Основными критериями выбора для начинающего разработчика игры могут быть следующие параметры среды разработки игры:

- Удобство использования. Зачастую для начинающего разработчика очень важен графический интерфейс и наглядность среды разработки.
- Межплатформенность. Большинство разработчиков не рассчитывают ограничиться одной платформой разработки, хотя и начинают с какой-то одной, поэтому им важна возможность сборки игры под другие платформы с минимальными затратами.
- Стоимость. Стоимость использования среды разработки не должна быть слишком высокой, поскольку начинающий разработчик сильно ограничен в средствах.
- Средства разработки, используемые в движке должны быть распространенными и не вызывать сложности в использовании.
- Функциональность. Важно, чтобы среда разработки удовлетворяла всем востребованным на рынке и в игре функциям и имела конкурентоспособную графику.

2 Определение путей и методов разработки игры

Важным этапом работы является выбор нужного программного обеспечения, с чем необходимо определиться до начала разработки во избежание давнейших затруднений. Программное обеспечение, с которым нужно определиться на данный момент, можно разделить на следующие пункты:

- Операционная система. Нужно определиться, под какую операционную систему будет разрабатываться игра. Также необходимо решить, на какой операционной системе будет производиться сама разработка и отладка мобильной игры.

- Движок и среда разработки игры. Необходимо подойти к выбору среды разработки игры с должной серьезностью. Выбранная среда разработки должна иметь не только высокие показатели графики и производительности, но и давать возможность осуществить весь задуманный функционал разрабатываемой игры. При этом не стоит забывать о возможном дальнейшем развитие игры, с выходом на новые платформы и расширением функционала.

- Среда разработки кода. Данная среда разработки необходима для написания исходного кода игры, поскольку зачастую среда разработки не дает возможности писать код внутри нее, либо же имеет сильно ограниченный функционал. Необходимо чтобы данная среда написания кода поддерживала язык программирования движка и могла хорошо с ним состыковываться.

- Дополнительное программное обеспечение. В этот пункт входит все программное обеспечение, которое может потребоваться для решения второстепенных задач или расширения функционала выбранных сред разработки. Например, может понадобиться программное обеспечение для создания файлов с трехмерными моделями.

Выбор программного обеспечения не всегда ведется в именно таком порядке, а зачастую и вовсе пункты взаимосвязаны и выбор одного влияет на выбор другого, так что местами выбор программного обеспечения будет подаваться без строгого обоснования.

Помимо программного обеспечения, необходимо определиться с методами реализации алгоритма трассировки лучей, что в свою очередь зависит от выбранного программного обеспечения и целевой платформы разработки.

2.1 Выбор операционных систем

Как уже упоминалось в пункте выше, необходимо выделять две, необязательно разные, операционные системы: одна для разработки и отладки продукта, а другая должна являться целевой ОС, на которую рассчитана сборка и работа приложения. Для начала нужно сказать, что большинство сред разработок игр предназначены для работы на ОС Windows и рассчитывают, что именно на ней будет производится разработки игр, поэтому очевидно, что именно ее и стоит выбрать в качестве ОС разработки и отладки. Поскольку изначальная задача заключается в создании именно мобильной игры, то выбор для целевой ОС остается небольшим: Android и IOS. Однако в России, согласно сведениями интернет ресурса Яндекс Радар [4], более 70% пользователей имеют мобильное устройство под управлением Android, поэтому именно на эту ОС и будет первоначально ориентироваться мобильная игра. Выбранные ОС имеют высокий уровень надежности и удобны в использовании.

ОС Windows 10-ой версии является уже не самое новой версией, но тем не менее все еще самая устойчивая система и поддерживается наибольшим количество движков и сред разработки игр. При этом Windows не тратит на поддержание собственной работы больших вычислительных ресурсов, что дает больше возможностей для использования требовательных игровых движков.

Как уже ранее упоминалось, Android на данный момент является наиболее распространенной мобильной ОС в России и вероятно будет еще долгое время, за счет бесплатности, свободного распространения и хорошей поддержки со стороны разработчиков. Данная ОС является хорошо оптимизированной системой, которая зажата в рамки маломощных аппаратных составля-

ющих мобильных устройств. При всем этом, Android имеет огромный ряд различных версий и важно поддерживать версии наиболее актуальные в данный момент.

Подведем черту. Выбранными версиями ОС являются Windows 10, для разработки мобильной игры, и Android актуальных и поддерживаемых средой разработки версий, как целевая платформа мобильной игры на стартовый момент.

2.2 Выбор игрового движка и среды разработки игр

Невыбранная среда разработки должна поддерживать выбранные в предыдущем пункте операционные система, и желательно, как уже упоминалось ранее, она должна давать возможность без серьезных изменений создавать сборки и под другие мобильные ОС, для дальнейшего развития.

Однако не стоит останавливаться только на данном факторе при выборе среды разработки игр. Нужно так же учитывать такие параметры, как высокая производительность на всех поддерживаемых платформах, хорошая графика, доступность и актуальность необходимых дополнительных средств и знаний. Учитывая все это, стоит выделить самые главные, основные факторы, по которым необходимо будет сделать выбор в пользу той или иной среды разработки.

- Платформы, под которые можно сделать сборку приложения.
- Актуальность и сложность в изучении и использовании языка программирования среды разработки.
- Трудоемкость разработки.
- Актуальность и распространенность используемых технологий.
- Стоимость использования среды разработки игр.
- Хорошие показатели производительности на мобильных устройствах.
- Актуальный уровень графики игривого движка.

Учитывая описанные в пункте 1.4.3 наиболее популярные движки и среды разработки, а также приведенное к ним описание, беря во внимание все

вышеперечисленные факторы, можно сделать выбор в пользу среды разработки и движка Unity. Так как он, наряду с огромным функционалом и высокой производительностью, славится своими низкими требованиями к аппаратной составляющей устройства, что делает его, совместно с поддержкой невероятно большого количества мобильных ОС, идеальной средой разработки игр для мобильных телефонов. При этом Unity можно использовать по тарифу, который является полностью бесплатным до превышения привлеченных к игре инвестиций в 100 000$, что делает его по сути бесплатным для начинающих разработчиков.

2.2.1 Достоинства и недостатки среды разработки игр Unity

Приведем основные достоинства и недостатки среды разработки Unity, а также оценим популярность данной среды разработки игры.

Достоинств у Unity много, поэтому приведем только основные:

- Высокие показатели производительности на слабых устройствах.
- Поддержка огромного количества платформ, в том числе и мобильных.
- Актуальные и простые в изучении и использовании языки программирования.
- Наличие актуальных версий среды разработки и поддержка среды со стороны разработчиков.
- Наличие официальной документации и огромного количества обучающего материала от различных сообществ.
- Возможность добавления встроенной рекламы в игру.
- Простой и удобный графический интерфейс, позволяющий часть разработки производить в нем.
- Огромный магазин среды с платными и бесплатными ассетами и плагинами.

Недостатки:

- Хотя использование и является бесплатным, но в перспективе серьезного роста доходов от игры, придется отдавать процент от заработка.

• Медлительность загруженных сцен, из-за основного используемого языка C#. Хотя это и не затрагивает разрабатываемую в данной работе игру.

• Чуть больший вес собранной версии игры, чем у аналогичных сред разработки.

Не смотря на все эти недостатки, данная среда разработки игр очень популярна. С наиболее полной статистикой использования Unity в разработке мобильных игр можно ознакомиться на сайте AppBrain [9]. Вот несколько наиболее популярных примеров мобильных игр, разработанных в среде Unity:

• Monument Valley от компании Ustwo

• Hearthstone от компании Blizzard Entertainment

• Hitman Sniper от компании Square Enix

• ShadowGun: DeadZone от компании MADFINGER Games

• Alto’s Adventure от разработчика маленькой команды разработчиков Snowman

• Mobius Final Fantasy от компании Square Enix

• Reigns от компании DevolverDigital

• Ravensword: Shadowlands от компании Crescent Moon Games, первоначально созданная индивидуальным разработчиком

• PAKO – Car Chase Simulator от компании Tree Men Games

2.3 Выбор среды написания исходного кода мобильной игры

Поскольку предполагается использовать среду разработку игр Unity, то большая часть кода будет написана на языке программирования C#, в соответствии с особенностями среды разработки и знаний разработчика. Для выбранного языка программирования существует несколько различный сред разработки, например, Visual Studio, Project Rider, Eclipse и другие. При этом наиболее популярной и известной среди разработчиков, использующих C# является среда разработки Visual Studio, это делает работу с ней проще и менее затратной, ввиду имеющихся знаний о среде. При этом данная среда интегрируется

со средой разработки игр Unity, и эта интеграция активно поддерживается с обоих сторон разработчиков. Такая интеграция дает множество возможностей, среди которых интеллектуальное авто дополнение, автоматизированное изменение исходных файлов, умная подсветка синтаксиса и многое другое.

В результате выбора, предполагается использовать среду разработки Visual Studio для написания исходного кода мобильной игры.

2.4 Выбор дополнительного программного обеспечения

В качестве дополнительного программного обеспечения потребуется среда для создания трехмерных моделей, которые будут использоваться в мобильной игре. В качестве такого программного обеспечения отлично подойдет Blender – профессиональная свободная и отрытая программа, для создания трехмерных моделей. Использование является полностью бесплатным, а широкий набор функций сильно упрощает создание моделей для начинающих разработчиков. Blender является довольно популярной программой с простым для изучения интерфейсом и массовой поддержкой. В сети Интернет существует множество статей, курсов, уроков и книг, созданный для изучения и быстрого старта использования данного программного обеспечения.

2.6 Определение методов реализации рейтрейсинга

Основной алгоритм рейтрейсинга уже давно является общеизвестным и прочитать его описание можно во множестве различных источников. Так к примеру, можно ознакомиться с идеями данного алгоритма в книге Роджерса Д. Алгоритмические основы машинной графики [10].

Кратко изложим идею данного алгоритма получения изображения трехмерного пространства. Идея данного алгоритма заключается в том, что для получения изображения необходимо моделировать поведение лучей, которые выходят из камеры в определенном направлении и, в зависимости от того, с чем столкнулся луч, определяется результирующий цвет в точке изображения. Стоит отметить, что некоторые реализации подразумевают так же генерацию

лучей, испускаемых источником света и другими объектами, однако основная суть алгоритма остается прежней: моделирование поведения лучей с расчетом их траектории.

Рассмотрим основные достоинства и недостатки алгоритма. Достоинства:

- Высокая алгоритмическая параллельность, ввиду независимого моделирования каждого луча.
- Наличие перспективы и корректного отображения поля зрения.
- Потенциально высокая реалистичность изображений, за счет близости к реальным физическим явлениям.

Недостатки:

- Низкая производительность, из-за того, что необходимо рассчитывать каждый луч отдельно, каждый раз начиная с начала.

Учитывая все перечисленное выше, для эффективного использования алгоритма рейтрейсинга, необходимо задействовать высокую степень распараллеливания вычислений. Лучшим вариантом для этого станет написание шейдера – программы, предназначенной для исполнения на видеокарте или графическом ускорителе, которые специализируются на быстрых параллельных математических вычислениях и которые создавались специально для рендеринга изображений. К счастью, Unity дает отличные возможности для написания шейдеров на языке программирования HLSL, в том числе давая возможность писать глубоко параллельный код для шейдеров, чем и стоит воспользоваться.

Однако одного распараллеливания при помощи шейдеров будет недостаточно, для получения хороших показателей FPS на мобильных устройствах. Безусловно этого хватило бы для мощной аппаратной составляющей стационарных компьютеров, но не для мобильных устройств. Поэтому необходимо оптимизировать используемые алгоритмы под конкретную задачу, мирясь с потерей универсальности кода. К примеру, за счет особенностей игры, таких

как отсутствие передвижений камеры или объектов (не считая вращения), можно ограничить проверку столкновения луча с объектами в определенных областях, где их точно не может быть согласно концепции игры. Также стоит ограничиться минимальным функционалом алгоритма, достаточным для нормального восприятия игры, тем самым уменьшив затраты на дополнительные вычисления.

Подведем итог. При использовании алгоритмов рейтрейсинга в мобильной игре, необходимо писать исходный код алгоритма в программах под графические процессоры – шейдеры, на языке HLSL, который поддерживает Unity. Так же необходимо проводить оптимизацию алгоритма и урезать функционал, без потери смысловой нагрузки игры, под конкретную задачу, сводя вычислительные затраты к минимуму.

2.7 Разработка структуры мобильной игры

При разработке мобильной игры, формируется сложная и многомодульная структура приложения. К главным составляющим структуры мобильной игры можно отнести следующие:

- Изображения, текстуры и материалы.
- Управление входящими данными.
- Основная логика игры, реализованная алгоритмами.
- Управление звуком игры.
- Система рендеринга изображений.
- Пользовательский интерфейс.

К структуре мобильных игр также относят взаимосвязь и переходы между экранами игры. Так, например, большинство мобильных игр имеют 2 главных экрана: главное меню и главный игровой экран. На первом отображаются основные возможности действий с игрой, а на втором отображается игровой мир со всеми игровыми механиками. Помимо данных главных экраном могут содержаться такие как: экран настроек игры, экран выбора уровня, экран

паузы, экран пройденного уровня и другие экраны, количество которых ограничено только концепцией игры и ее функционалом. Особенно стоит выделить экран рекламы, который может быть представлен как отдельный экран на весь дисплей устройства, так и не мешающий игровому процессу маленький баннер, или и вовсе может отсутствовать. Основная роль экрана рекламы – получение дохода от показа рекламы, чем часто и пользуются разработчики, чтобы окупить свое приложение. Стоит так же отметить, что все вышеописанное может сильно разниться и зависит только от разработчика и самой игры.

3 Разработка игры с использованием среды Unity

Разработку игры можно условно поделить на два этапа. Сначала необходимо разработать основную механику игры и игровой мир, с возможностью проходить уровни. На этом этапе можно условно говорить, что основная часть игры разработана и ей уже можно пользоваться. Второй этап призван довести разрабатываемую игру до состояния удовлетворения всем изначальным требованиям заказчика. В игру на этом этапе необходимо добавить весь недостающий функционал, описанный в требованиях, а также тот функционал, который понадобится заказчику, хотя и не был указан им. Стоит отметить, что при разработке будет уделено особое внимание разработке рендеринга рейтрейсингом и качеству получаемого изображения. Процесс получения изображения получаемого на выходе такого рендеринга, не должен снижать fps ниже затребованного минимума. Далее приведено описание разработанных классов и структуры исходного кода. С кодом основных приведенных ниже классов можно ознакомиться в приложении А.

3.1 Разработка основных механик игры

В данной мобильной игре под разработкой основных механик подразумевается следующее:

- Рендеринг игрового мира при помощи рейтрейсинга. Это основная составляющая игры, на которой все основывается и именно поэтому это необходимо разработать в первую очередь.
- Механика уровней. Безусловно это не менее важная составляющая, без которой невозможно будет пользоваться данной мобильной игрой, так как сюда входят: игровой мир уровня, управление объектом и возможность пройти уровень.

3.1.1 Разработка рендеринга с использованием рейтрейсинга

На данном этапе необходимо реализовать рендеринг изображения, которой должен визуализировать трехмерный мир игры. Игровой мир состоит в

данном случае из объектов, которые имеют свой меш и различные параметры отображения: поворот и расположение, текстура, прозрачность, цвета граней и т. д. Таким образом реализуется рендеринг описанных объектов, по средствам рейтрейсинга, который исходя из выбранного метода реализации, должен быть описан в шейдере для параллельности и ускорения вычислений.

Диаграмма классов реализованного рендеринга представлена на рисунке 3.1.

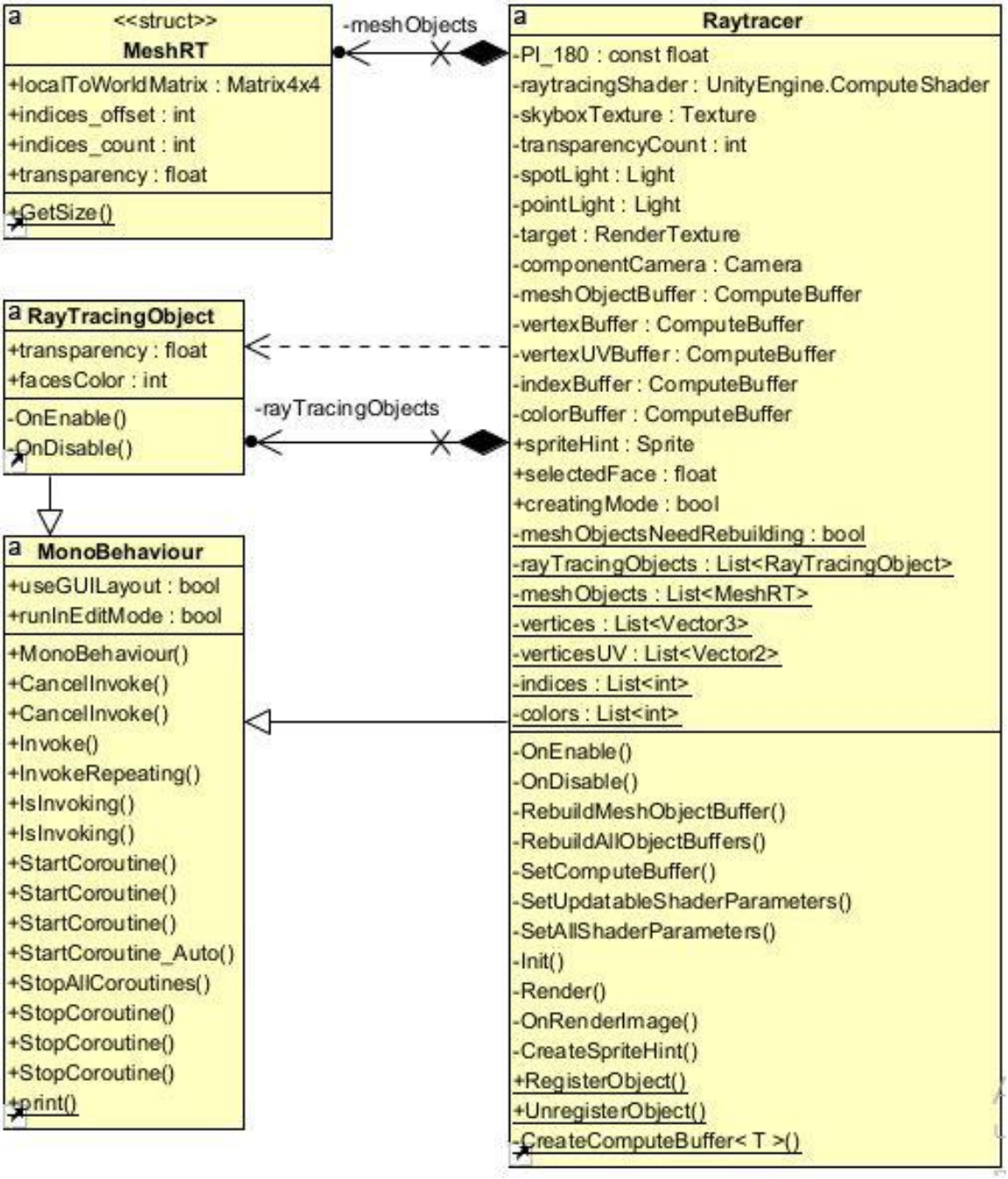

Рисунок 3.1 – Диаграмма классов рендеринга

Основным классом рендеринга является класс Raytracer. Данный класс занимается сбором информации об объектах сцены, которые необходимо отображать и всех параметрах этих объектов. Помимо этого, данный класс предназначен для создания текстуры рендеринга, обработкой которой занимается написанный шейдер с рейтрейсингом принадлежащий к базовому классу ComputeShader, назначением параметров шейдера и последующим обновлением этих параметров. Помимо основной составляющей класса Raytracer содержит метод, реализующий создание изображения тени объекта, которое строится на основании текстуры рендеринга.

Для облегчения работы с параметрами объектов была создана структура MeshRT, которая хранит дополнительную информацию о меше объекта, тогда как основная составляющая информации о меше (координаты вершин и индексы вершин для треугольников), хранятся в виде общих буферов. Такой подход с общими буферами и дополнительными структурами данных предназначен для оптимизации работы с шейдером, потому что узким местом рендеринга является передача информации между кодом, исполняющимся на CPU (Raytracer) и кодом, исполняющимся на GPU (шейдером). Помимо прочего, данная структура является сериализуемой для правильной организации передачи данных.

Для получения непосредственной и актуальной информации от игровых объектов, был создан класс RayTracingObject. Данный класс содержит дополнительные параметры, которые отсутствуют у игрового объекта, такие как прозрачность и цвета граней. Помимо этого, данный класс содержит два метода, первый из которых, при создании объекта передает информацию о себе в класс Raytracer и просит зарегистрировать себя, внести информацию о себе в буферы для шейдера.

Для написания шейдера RayTracerShader использовался C-подобный язык HLSL. В написанном шейдере RayTracerShader не использовались

классы, но для удобства написания и изучения кода, использовались структуры, диаграмма классов которых представлена рисунке 3.2.

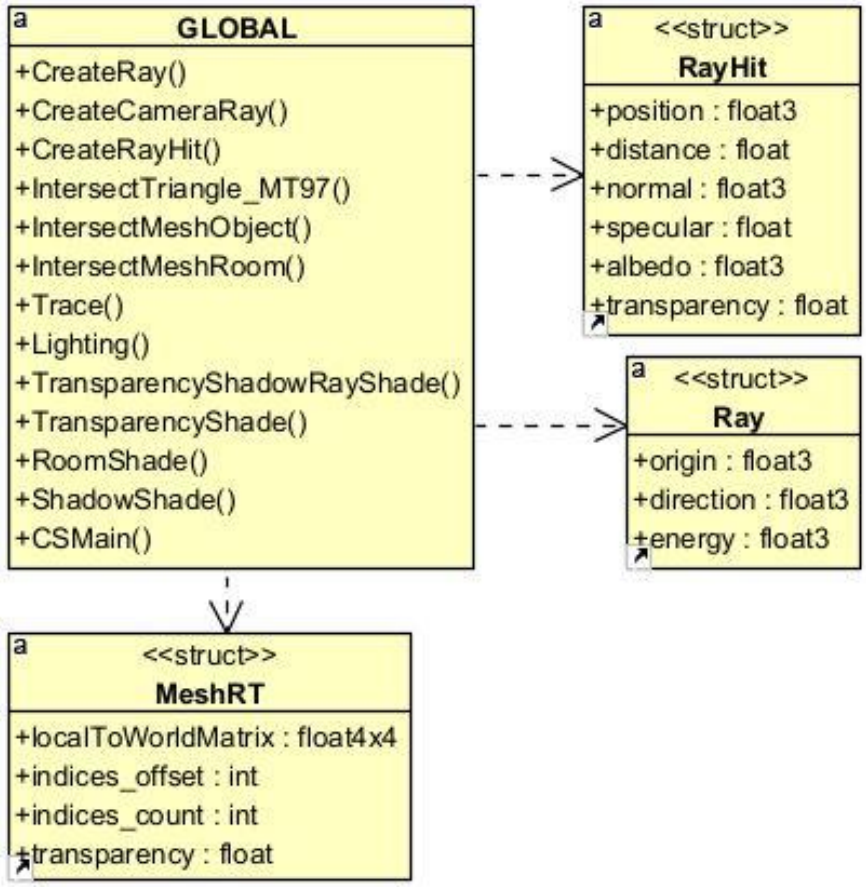

Рисунок 3.2 – Диаграмма классов шейдера

Исходный код данной программы шейдера состоит из нескольких частей. Во-первых, это общие поля параметры, которые назначаются классом Raytracer. Во-вторых, это структуры, использующиеся как для принятия и чтения переданных параметров, так и для упрощения написания кода. В-третьих, это функции, которые упрощают сокращают код. В-третьих, главный метод, который определяется директивой «#pragma kernel», этот метод и будет исполняться, при вызове данного шейдера в классе Raytracer. При помощи всех данных составляющих шейдера, бал описан алгоритм рейтрейсинга. Структура Ray используется для задания луча и содержит направление, энергию по каждому цвету и источник. Структура RayHit используется для задания параметров столкновения луча с объектом. Структура MeshRT полностью идентична одноименной структуре, описанной выше и предназначена только содействия

десериализации данных, чтобы правильно прочесть и понять данные переданные через буфер и другие параметры. Функции, реализованные в шейдере (подробнее с ними можно ознакомиться в приложении А, где приведен исходный код):

- Ray CreateRay (float3 origin, float3 direction). Конструктор структуры луча на основании двух точек.
- Ray CreateCameraRay (float2 uv). Конструктор луча, исходящего из камеры и проходящего через заданный пиксель экрана.
- RayHit CreateRayHit (). Конструктор структуры столкновения с заполнением полей значениями по умолчанию.
- bool IntersectTriangle_MT97 (Ray ray, float3 vert0, float3 vert1, float3 vert2, inout float t, inout float u, inout float v). Определение пересечения луча и треугольника по вариации алгоритма Моллера-Трумбора. Подробнее с данным алгоритмом можно ознакомиться в оригинальной статье Томаса Моллера и Бена Трумбора [11].
- void IntersectMeshObject (Ray ray, inout RayHit bestHit, MeshRT meshObject). Проверка всех треугольников меша прозрачного объекта на пересечение с лучом и определение ближайшего к источнику луча пересечения.
- void IntersectMeshRoom (Ray ray, inout RayHit bestHit, MeshRT meshObject). Проверка всех треугольников меша комнаты на пересечение с лучом и определение ближайшего к источнику луча пересечения. Дополнительно определяется цвет и нормаль в точке столкновения из соответствующих текстур комнаты.
- RayHit Trace (Ray ray). Определение столкновения путем поочередного вызова функций IntersectMeshObject, а затем IntersectMeshRoom.
- float3 Lighting(Ray ray, RayHit hit). Определение освещенности места столкновения, где учитываются два источника света: точечный и направленный. Зависимость освещенности от расстояния, угла (для направленного света)

и интенсивности источника реализована на основании Закона обратных квадратов.

- float3 TransparencyShadowRayShade (inout Ray ray, RayHit hit). Определение цвета столкновения и обновление энергии и направления луча на некоторой итерации расчета теневого луча для прозрачного объекта.
- float3 TransparencyShade (inout Ray ray, RayHit hit). Определение цвета столкновения и обновление энергии и направления луча на некоторой итерации расчета луча для прозрачного и непрозрачного объекта.
- float3 RoomShade (inout Ray ray, RayHit hit). Определение цвета столкновения для непрозрачного объекта, если столкновения не было, тогда цвет текстуры фона.
- float3 ShadowShade(inout Ray ray, RayHit hit). Определение цвета столкновения для непрозрачного объекта, а также расчет цветной тени. Для расчета тени циклически вызывается функция TransparencyShadowRayShade, а результаты особым образом суммируются и получается цвет тени.
- void CSMain (uint3 id : SV_DispatchThreadID). Главная функция шейдера, которая исполняется при назначении шейдера текстуре рендеринга. Метод выполняется параллельно и отдельно для каждого пикселя текстуры. Здесь расчет результирующего цвета пикселя с помощью методов TransparencyShade, ShadowShade и RoomShade.

В шейдере RayTracerShader реализован упрощенный и оптимизированный алгоритм трассировки лучей. В данном случае, пускается луч из камеры через пиксель экрана, для которого был вызван шейдер. Луч на данном этапе либо сталкивается с объектом, либо не сталкивается и возвращает цвет текстуры фона. Если же пускаемых из камеры луч сталкивается с прозрачным объектом, тогда необходимо учесть в результирующем цвете, цвет объекта столкновения, уменьшить энергию луча и продолжить движение по тому же направлению до тех пор, пока не будет столкновения с непрозрачным объектом или столкновение не будет отсутствовать. Если луч пускаемых из камеры

сталкивается с непрозрачным объектом, то учитывая текстуру объекта, текстуру нормалей, угол между нормалью к точке падения и направлением света, расстояние до источников света и другое, определяется цвет. Однако далее необходимо построить луч от точки столкновения в сторону источника света, если луч ни с чем не столкнулся, значит результирующий цвет не меняется и тени нет, если луч столкнулся с непрозрачным объектом, тогда необходимо вернуть черный цвет, потому что это тень, если луч столкнулся с прозрачным объектом, тогда необходимо изменить энергию луча, учесть цвет прозрачного объекта и продолжить движение в сторону источника света. Таким образом цвет столкновения с непрозрачным объектом получается исходя из его собственного цвета и тени, которая может быть от прозрачного объекта или от непрозрачного. Сам код шейдера выполняется параллельно на GPU для каждого пикселя экрана, поскольку их вычисления никак не связаны между собой.

Главным оптимизирующим решением в данном алгоритме стало разбиение текстуры рендеринга (экрана), на 3 части. В первой части содержится только непрозрачный меш комнаты, а значит нет смысла проверять луч на столкновение с мешем объекта вращения. Эта часть занимает большую часть экрана и является наиболее простой в вычислении. Второй частью является область, в которой точно находится цветная тень, но при этом явно нет самого прозрачного объекта вращения. И третьей область является область, в которой есть как прозрачный объект вращения, так и непрозрачная комната. Все эти области представляют из себя квадратные границы, определяющие способ расчета и его сложность. Границы областей необходимо настраивать в соответствии с расположением объектов в игровом мире.

Стоит заметить, что классы RayTracingObject и Raytracer унаследованы от базового класса движка Unity MonoBehaviour. Согласно официальной документации Unity [5], MonoBehaviour это класс, который дает возможность прикрепить сценарий к игровому объекту (базовый класс GameObject) в редакторе

Unity. Класс MonoBehaviour позволяет переопределять некоторый список методов, которые срабатывают в определенный момент игрового цикла., так, например, переопределенный метод Start сработает тогда, когда GameObject начнет существовать, а метод Update будет срабатывать каждый кадр игры. Наследование от MonoBehaviour созданных при разработке игры классов будет встречаться еще очень часто, поскольку использование его методов по событиям очень удобно.

Упомянутый выше и активно использующий далее класс GameObject также является базовым классом Unity, который используется для представления всех объектов на сцене. Согласно все той же документации [5], GameObject являются строительным блоком для сцен в Unity и действует как контейнер для функциональных компонентов, которые определяют, как игровой объект и что он делает.

3.1.2 Разработка механик уровней

В данном разделе необходимо реализовать основную механику игры, основанную на прохождении уровней. Механика прохождения уровней будет заключаться в необходимости расположить прозрачный объект вращения в таком положении, чтобы его цветная тень, получаемая усилиями реализованного рендеринга, полностью совпадала с изображением-подсказкой, которое расположено в левой нижней части экрана. Результатом прохождения уровня, помимо самого факта достижения цели, будет является время, затраченное на прохождение. На сцене необходимо расположить объекты игрового мира в таком порядке, чтобы подсказка была в левом нижнем углу экрана, тень располагалась в середине левой части экрана, в объект вращения в середине правой части экрана, а таймер размещается в нижнем правом углу экрана. Так же необходимо добавить в верхнюю часть экрана слайдер, который будет отображать близость к победе на данном уровне. Вращение объектов необходимо реализовать дискретным образом с определенным шагом угла, для того, чтобы упростить пользователю понимание и ориентирование в игровом мире.

Для этого в первую очередь необходимо в визуальном редакторе среды разработки Unity расположить объекты игрового мира на сцене и настроить все параметры сцены и объектов, будь то источники света, расположение в пространстве и повороты объектов, текстуры и многое другое. Далее необходимо написать класс LevelControllerScript, который будет контролировать всю логику в пределах одного уровня игры и следить за его прохождением. Затем необходимо реализовать управление вращением главного объекта игрового мира посредствам класса RotationScript использующего данные встроенного в среду разработки класса джойстика. И последним пунктом реализации в данном разделе является реализация класса TimeCounterScript, моделирующего работы таймера, для того, чтобы засекать время, затраченное на прохождение уровня. Диаграмма классов реализованной игровой механики уровней представлена на рисунке 3.3.

Класс LevelControllerScript хранит указатели на каждый объект игрового мира, являющийся важным с точки зрения логики уровня, а также финишное положение прозрачного объекта вращения. С начала запуска сцены уровня, данный класс в первую очередь активирует панель загрузки уровня, которая не дает пользователю наблюдать за предварительными настройками уровня. Во время предварительной настройки уровня происходит формирование изображения-подсказки финишной тени для данного уровня. Для этого объект вращается в финишное положения, после чего при помощи метода класса Raytracer формирующего подсказку получается изображение и помещается в левый угол экрана. Затем объект вращает в стартовое положение и экран загрузки деактивируется. Каждый кадр игры происходит сравнение финишного положения и текущего положения вращения прозрачного объекта. Если текущее положение отличается от финишного на некоторое малое значение, то объект автоматически доводится до финишного положения линейной интерполяцией. Помимо этого, каждый кадр обновляется значение слайдера, равно разнице между финишным и текущим положением прозрачного объекта.

Следующим на очереди является класс RotationScript. Данный класс каждый кадр интерпретирует значения, полученные от класса FloatingJoystick в значения вращения, которые необходимо применить к объекту вращения. Однако перед непосредственным применением значения нового положения в игровом мире, данные вращения берутся по модулю определённого шага, для дискретизации положений объекта. При отклонении джойстика на дельту от изначального положения, объект так же отклоняется на дельту, переведенную в угловые значения, в соответствии с установленным для вращения шагом и диапазоном поворота. При отпускании джойстика фиксируется текущее положение фигуры и при повторном задействовании джойстика оно будет использовано как изначальное.

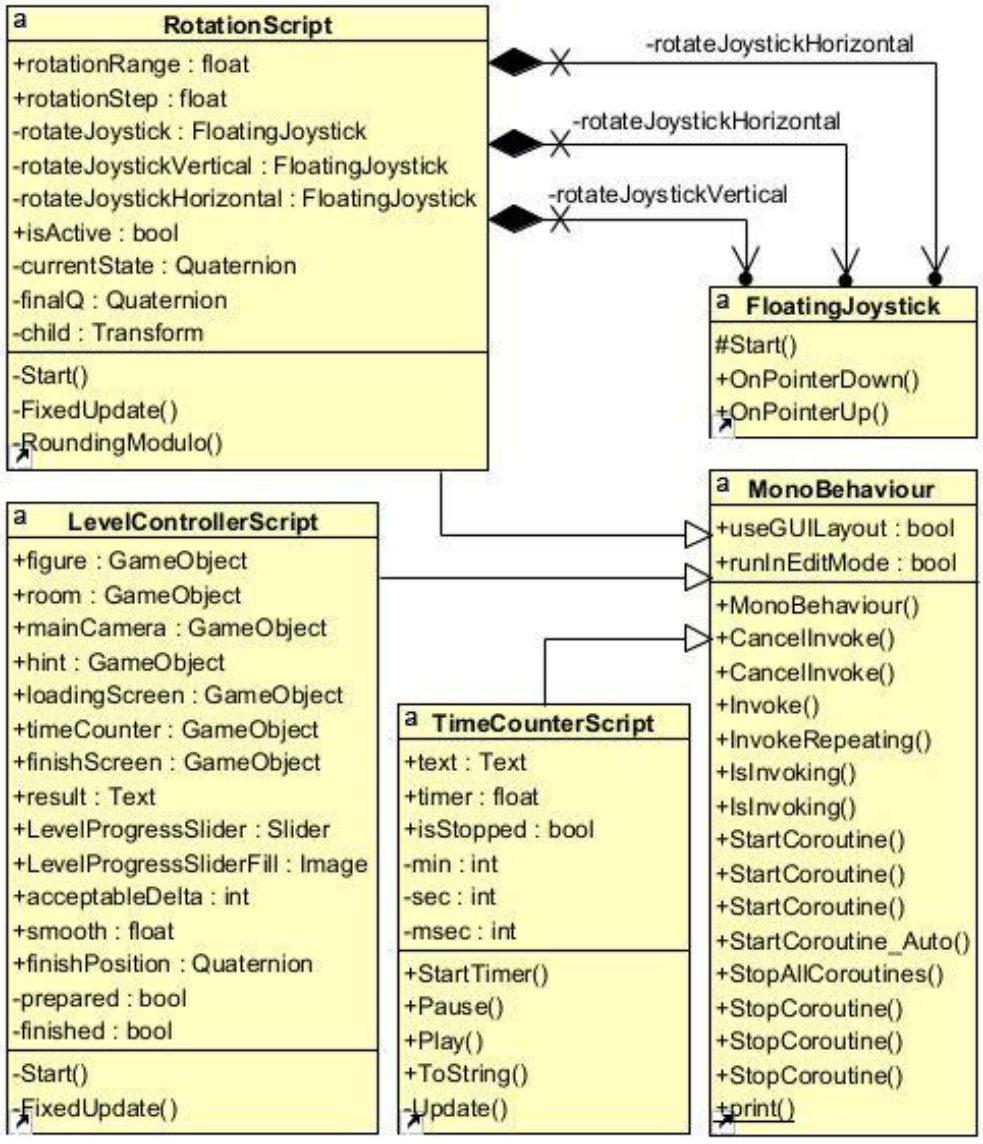

Рисунок 3.3 – Диаграмма классов игровой механики уровней

Класс TimeCounterScript играет роль секундомера, который можно запустить, поставить на паузу и сбросить. После запуска, секундомер хранит время, пройденное с момента запуска. Также секундомер имеет метод, который преобразует хранящиеся значения в строковый формат времени.

3.2 Разработка дополнительного функционала игры

После организации основного функционала игры, уже можно использовать мобильную игру, однако в ней все еще недостает функционала, описанного в требованиях, и она не представляет никакого интереса. Поэтому необходимо давить следующий функционал:

- Система сохранений. Реализует хранение уровней в файлах и сохранение прогресса пользователя в прохождении уровней.
- Режим создания уровней. Реализовать режим, дающий возможность настраивать параметры уровня и сохранять его в отдельном файле, после чего запускать этот файл для его прохождения.
- Структура игровых меню. Разработать игровые меня и весь функционал их составляющих.
- Звуковое сопровождение. Реализовать фоновое проигрывание мелодии во время прохождения и создания уровней и звуковую реакцию на различные события.

3.2.1 Разработка системы сохранений

На данном этапе необходимо реализовать систему сохранений, которая заключается в возможности сохранять и загружать данные уровней из файлов общего формата, а также сохранять и загружать информацию о прогрессе пользователя в прохождении уровней, хранящуюся в реестре использующейся платформы. Данные о прогрессе пользователя не должны самопроизвольно изменяться между сессиями использования приложения. Для этого необходимо реализовать класс CurrentData, который будет отвечать за хранение текущей информации для данной сессии использования игры и предоставлять методы

для обновления этой информации в соответствии с данными файлов и реестра. Классы Level, Progress и Settings отвечают соответственно за представление уровня, прогресса и настроек игры. А статичный класс SaveManager предоставляет методы для непосредственно сохранения и чтения информации в файлы и реестры независимо от типа данных самой информации. Диаграмма классов реализованной системы сохранений представлена на рисунке 3.4.

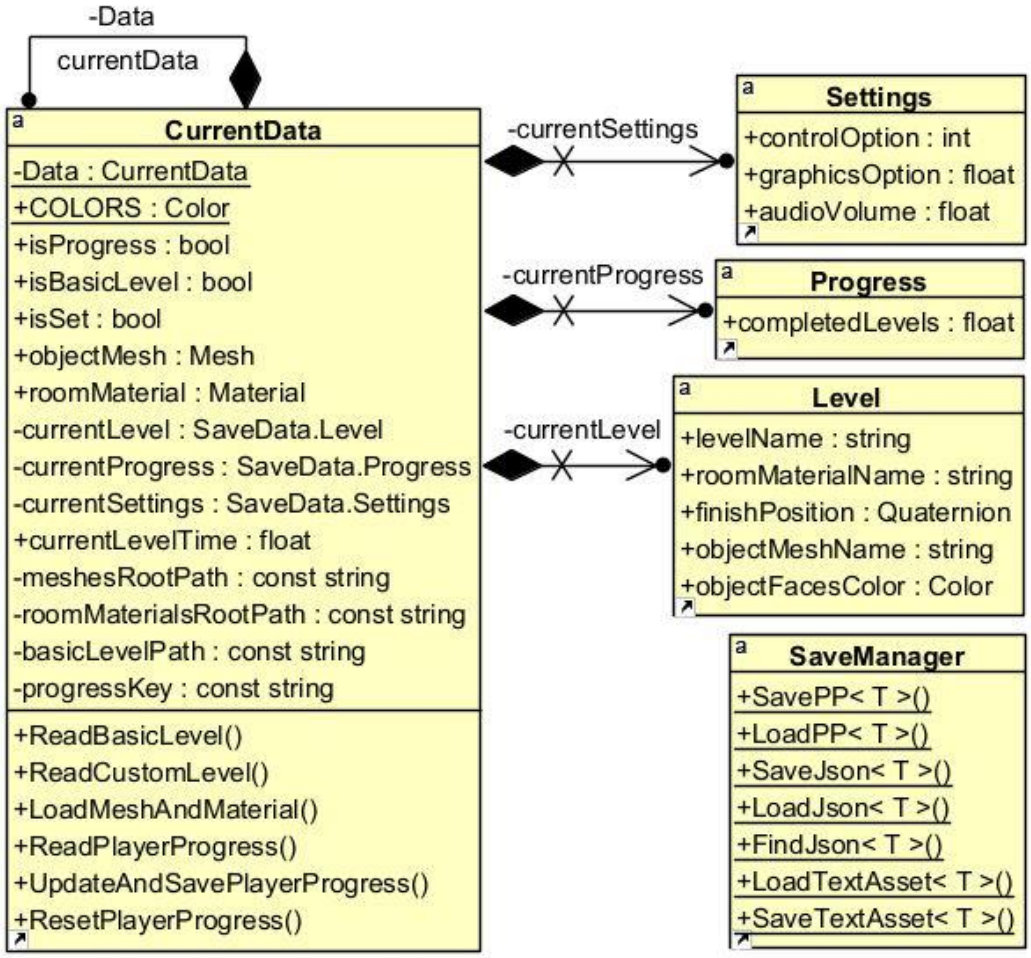

Рисунок 3.4 – Диаграмма классов системы сохранений

Класс CurrentData хранит текущую информацию: прогресс, настройки, выбранный уровень, и общую информацию, такую как расположение различных ресурсов. Этот класс содержит статичное поле типа данных CurrentData, через которое можно иметь доступ ко всем текущим данным из любого момента, сцены и класса игры. Помимо текущих данных, этот класс предоставляет методы, позволяющие загружать ресурсы игры, такие как материалы и

меши, по их названиям, а также обновлять текущие данные считыванием данных из файлов и реестров.

Следующими к рассмотрению классами являются классы: Level, Settings и Progress. Данные классы являются нереализуемыми и предназначены для описания уровня, настроек и прогресса соответственно, а также для сохранения и чтения данной информации из файлов и реестров.

И наконец класс SaveManager. Данный класс содержит методы, для сохранения и чтения данных в независимости от из типа. Для класса SaveManager реализованы возможности: работа с файлами типа «.json» для сохранения и загрузки уровней из файлов, работа с TextAsset, для загрузки уровней, добавленных в игру на этапе разработки и хранящихся после сборки в особом формате текстовых ресурсов игры, и, наконец, работа с реестром, где предполагается хранение текущего прогресса пользователя под определенным ключом. При этом сохранение уровня в формате «.json» происходит по разному пути в зависимости от платформы запуска приложения. Так для Android работа будет производится в папке: «/storage/emulated/0/Android/data/<packagename>/files», а при запуске в среде разработки для отладки или на любой другой платформе, работа будет производится с корневой папкой расположения игры.

3.2.2 Разработка режима создания уровней

На данном этапе необходимо реализовать режим, в котором можно было бы настраивать параметры уровня и сохранять его в файл «.json» для дальнейшей отправки другим пользователям и запуска данного уровня в игре для его прохождения. На экране необходимо разместить различные элементы пользовательского интерфейса, при помощи которых должно осуществляться регулирование параметров уровня. Все изменения параметров, должны в реальном времени отражаться на игровом мире. Финишное положение для уровня задается последним положением фигуры перед нажатием кнопки сохранения. Для успешного сохранения необходимо задать имя уровня, которое и станет названием файла для сохранения. Организация вращения и рендеринга изображения

не должна отличаться от таковой в режиме прохождения уровня, за исключением шага вращения, который для более точной настройки должен быть довольно маленьким. Для этого был создан класс CustomLevelCreator, который организует логику всего режима создания уровня. Диаграмма классов режима создания уровней представлена на рисунке 3.5.

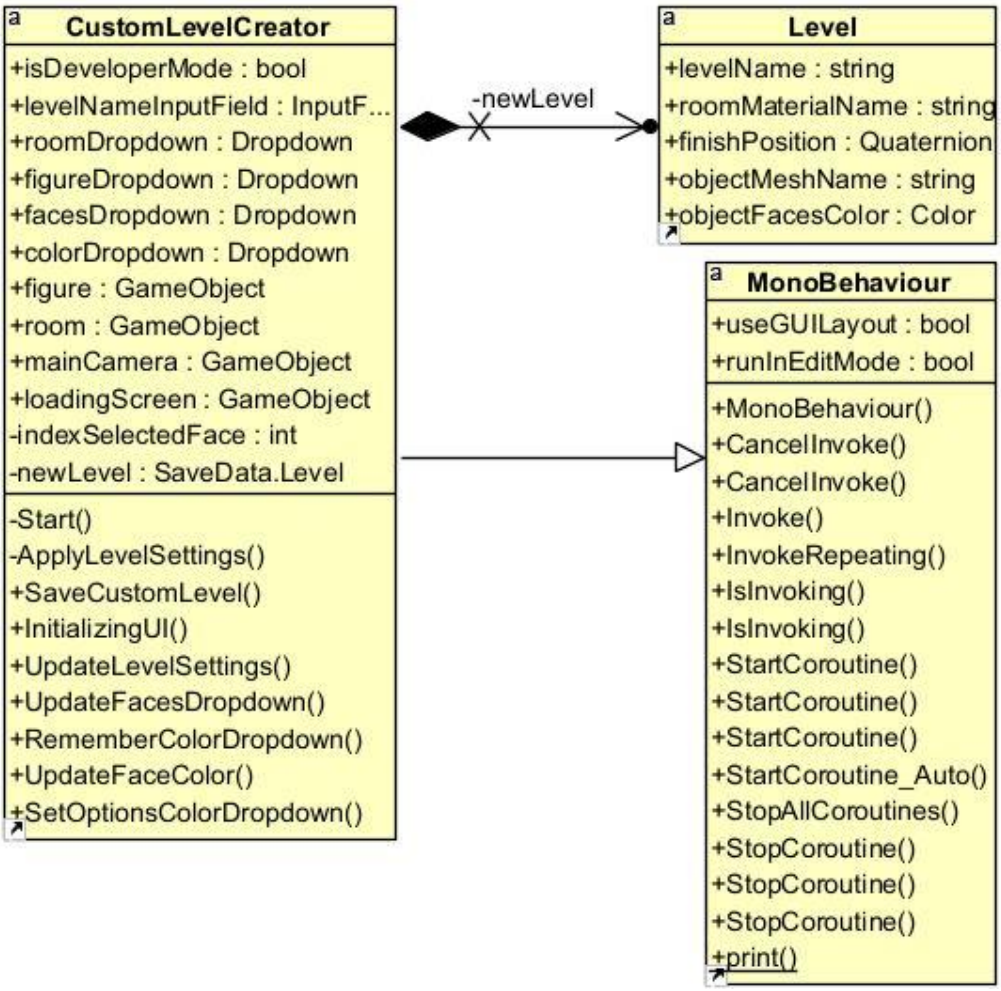

Рисунок 3.5 – Диаграмма классов режима создания уровней

Реализованный класс CustomLevelCreator хранит ссылки на объекты интерфейса и изменяемые объекты игрового мира. Данный класс следит за всеми объектами интерфейса и организует реакцию игрового мира на изменения в параметрах настройки уровня. Таким образом он дает возможность выбрать форму прозрачного объекта вращения, текстуру для стен и потолка комнаты, а также выбрать грань прозрачного объекта и задать ей один из предлагаемых цветов.

3.2.3 Разработка структуры игровых меню

При разработке структуры игровых меню необходимо создать, при помощи графического редактора и базовых объектов пользовательского интерфейса, интерфейс игры. Под этим подразумевается создание игровых меню, добавление кнопок и других интерактивных объектов пользовательского интерфейса, а также обработку всех событий, таких как нажатие кнопки и переход между экранами меню. Классы GameSettingsVisualisator и ProgressLevelVisualisator предназначены для визуализации текущих данных о настройках и прогрессе на элементах пользовательского интерфейса. А класс ButtonsScript предоставляет различные методы, которые предполагается вызывать по мере необходимости при взаимодействии с интерактивными элементами интерфейса, такими как кнопки. Диаграмма классов структуры игровых меню представлена на рисунке 3.6.

Класс GameSettingsVisualisator реализует отображение текущих настроек игры на интерактивных элементах меню настроек. Это необходимо для того, чтобы текущие данные настроек совпадали с отображаемыми пользователю.

Класс ProgressLevelVisualizator реализует отображение текущего прогресса пользователя в прохождении уровней игры. Отображение происходит посредствам затемнения кнопок и делания кнопок не интерактивными, для тех уровней, которые еще не были разблокированы. Помимо прочего сюда добавлен режим игры без прогресса, со всеми доступными уровнями, однако такой режим может включить только разработчик при отладке.

Класс ButtonsScript реализует различные сценарии использования интерактивных составляющие интерфейса игры. Так, например, в нем реализован метод, срабатывающий при нажатии кнопки для загрузки другого экранного меню, или метод реализующий изменение текущих настроек игры, в соответствии с интерактивным элементом за него отвечающим.

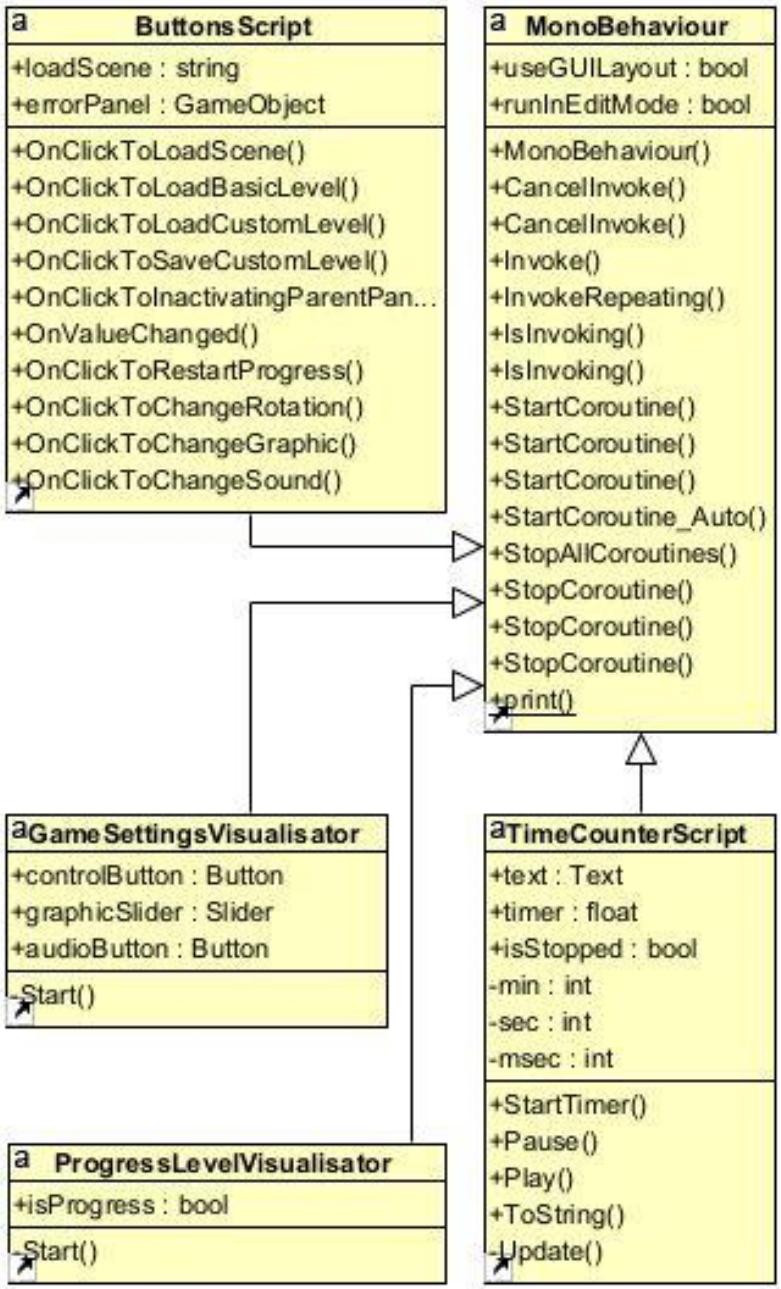

Рисунок 3.6 – Диаграмма классов структуры игровых меню

3.2.4 Разработка звукового сопровождения

Помимо прочего, необходимо разработать звуковое сопровождение для игры, в виде фоновой музыки в режимах создания и прохождения уровней. Также необходимо создать звуковую реакцию на взаимодействие с интерактивной составляющей интерфейса игры и на решение задачи уровня.

Все было реализовано средствами графического редактора среды разработки Unity. Звуковое сопровождение в движке Unity имеет две составляющие,

это слушатель и источник звука. Слушатель принимает звук от всех источников звука и воспроизводит на устройстве запуска игры. А источники звука воспроизводят звук для слушателей. Учитывая все это, были добавлены компоненты слушателей на игровые камеры сцен, которые отвечают за отображение игрового мира посредствам рендеринга. Источники звуков были расположены эти же камеры и другие объекты. По изменению значения интерактивного объекта интерфейса единожды воспроизводился звук нажатия кнопки, а при достижении финиша в уровне единожды воспроизводился звук победы. При запуске сцен прохождения уровней и создания уровней, запускался зацикленный фоновый звук, свой для каждого режима.

4 Разработка пользовательского интерфейса

Необходимо разработать простой и удобный пользовательский интерфейс, через который пользователь будет взаимодействовать с игрой. Помимо прочего такой интерфейс должен рассчитываться для использования на мобильных устройствах с сенсорным экраном, через который и будет происходить взаимодействие. Количество элементов пользовательского интерфейса должно стремиться к необходимому минимуму, чтобы не перегружать экран лишней информацией и не перекрывать основной контент игры. Все интерактивные элементы должны иметь не только логический отклик при взаимодействии с ними, но и визуальный, графический отклик, чтобы пользователь понимал с чем он взаимодействует. Так, например, при нажатии наведении пальца на кнопку, она должна затемняться или подсвечиваться.

4.1 Главное меню мобильной игры

Главным игровым меню, является меню, которое определяет основные намерения пользователя, что именно он хочет делать, посредствам интерактивных составляющих пользовательского интерфейса. Главное игровое меню необходимо активировать самым первым сразу после запуска игры. Скриншот экрана, содержащего главное игровое меню, представлен на рисунке 4.1.

Рисунок 4.1 – Скриншот главного игрового меню

Главное игровое меню содержит следующие интерактивные элементы:

• Кнопка «Play». При нажатии происходит переход к меню выбора уровня.

• Кнопка «Create». При нажатии происходит переход к экрану режима создания уровней.

• Кнопка «Setting». При нажатии происходит переход к меню настроек мобильной игры.

4.2 Меню выбора уровня

Меню выбора уровня запускается при соответствующем выборе, сделанном в главном меню игры. Меню выбора уровня предназначено для выбора пользователем уровня мобильной игры, который он хочет пройти. Есть два возможных сценария развития событий на данном меню: запуск пользовательского уровня по названию файла, если он был найден, и запуск базового уровня, встроенного в мобильную игру. Для этого меню выбора уровня разделено на две части: верхнюю и нижнюю. В верхней части находится все необходимое для запуска пользовательского уровня, а в нижней – все для выбора базового уровня. При это нижняя часть представляет из себя макет, способный прокручивать свое содержимое, при движении пальцем по нему, что позволяет представлять больше количество элементов, чем может поместиться на экране. Скриншот экрана, содержащего меню выбора уровня, представлен на рисунке 4.2.

Меню выбора уровня содержит следующие интерактивные элементы:

• Кнопка «Back». При нажатии происходит переход к главному меню.

• Поле ввода текста. Поле для ввода имени файла пользовательского уровня.

• Кнопка «Play». При нажатии происходит поиск файла пользовательского уровня с именем из поля ввода текста. При неуспешном поиске, пользо-

ватель получается текстовое сообщение о неуспехе поиска. При успешном поиске, происходит переход к главному игровому экрану, где запускается уровень из найденного файла.

• Кнопки уровней. При нажатии одной из кнопок уровней, происходит переход к главному игровому меню, где запускается базовый уровень, соответствующий нажатой кнопке.

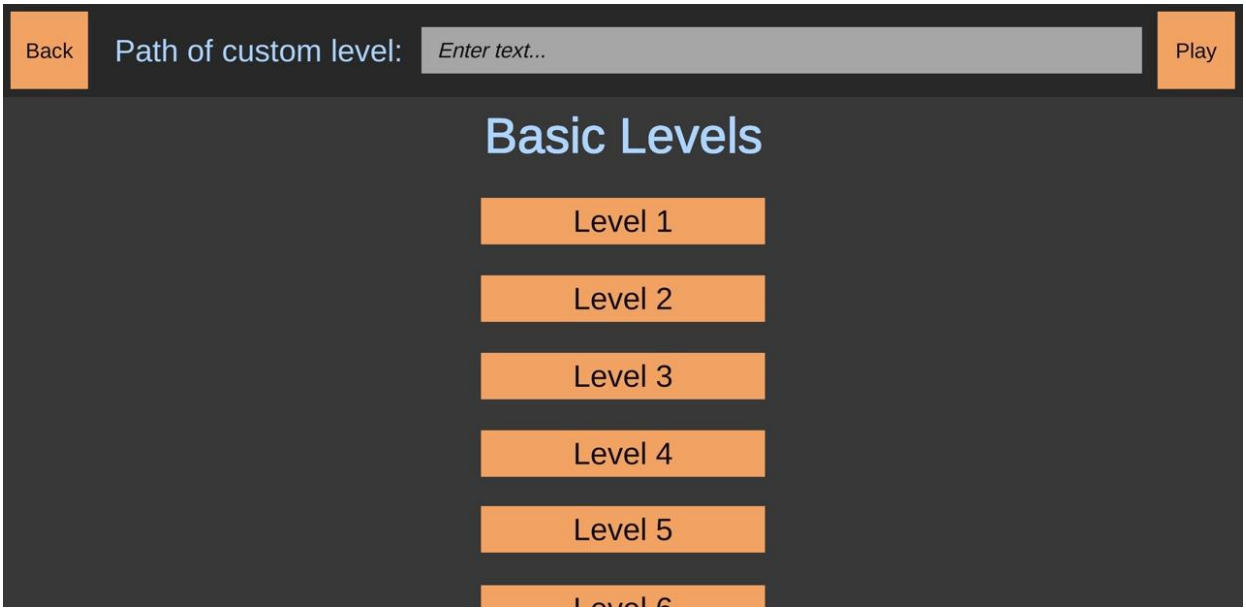

Рисунок 4.2 – Меню выбора уровня

Стоит отметить, что в игре реализован прогресс прохождения уровней, в рамках которого, пользователю доступны только те базовые уровни, которые он уже прошел и один следующий уровень, который требуется пройти. Кнопки уровней, недоступных для прохождения, являются неактивными и затемненными. Изначально пользователю доступен для прохождения только первый базовый уровень.

4.3 Главный игровой экран

Главный игровой экран запускается после выбора уровня в меню выбора уровней. Главный игровой экран предназначен для отображения игрового мира и организации механики прохождения уровня. Для этого на главном игровом экране поверх отображения трехмерного игрового мира, расположены

элементы механики уровня и пользовательского интерфейса, так чтобы не перекрывать ключевые объекты игривого мира. Пользователю предоставляется возможность взаимодействия с игровым миром, посредствам вращения объекта с помощью джойстика. Из данного экрана можно перейти по нажатию кнопки в меню паузы, а также при достижении финишного положения фигуры, автоматически происходит переход к меню пройденного уровня. Скриншот главного игрового экрана представлен на рисунке 4.3.

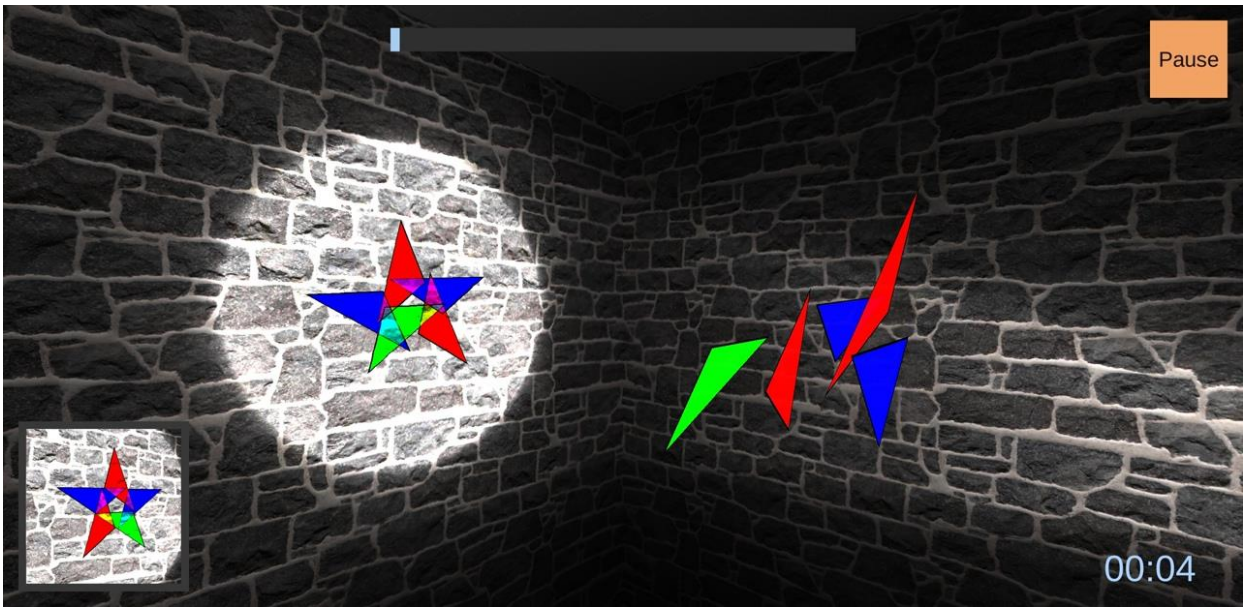

Рисунок 4.3 – Главный игровой экран

Главный игровой экран содержит следующие интерактивные элементы:

• Джойстики. Представлены 3 джойстика для организации двух типов управления, при этом одновременно все три не могут быть активны.

• Изображение финиша. Изображение, формирующееся в начале уровня и дающее представление пользователю о том, какую тень необходимо получить для прохождения уровня.

• Кнопка «Pause». При нажатии происходит переход в меню паузы, которое приостанавливает игровой процесс, с возможностью после возвращения продолжить с того же момента.

• Слайдер финиша. Слайдер, который показывает близость игрока к прохождению уровня: чем ближе слайдер к правому краю, тем ближе игрок к финишному положению фигуры.

• Таймер. Отображает время, прошедшее с момента запуска уровня. Переход к меню паузы приостанавливает таймер.

4.4 Экран режима создания уровней

Данный экран может быть запущен, путем нажатия соответствующей кнопки в главном меню игры. Экран создания уровней предоставляет такие же возможности взаимодействия с фигурой игрового мира, как и в главном игровом экране. Однако, помимо этого, данный экран дает возможность влиять в реальном времени на параметры уровня, посредствам интерактивных элементов пользовательского интерфейса, результат чего сразу же отображается в игровом мире. Этот же экран создания уровня дает возможность сохранять настроенный и созданный уровень. Стоит отметить, что финишным положением для уровня, станет последнее положение фигуры и соответствующая этому положению тень. Скриншот экрана режима создания уровней представлен на рисунке 4.4.

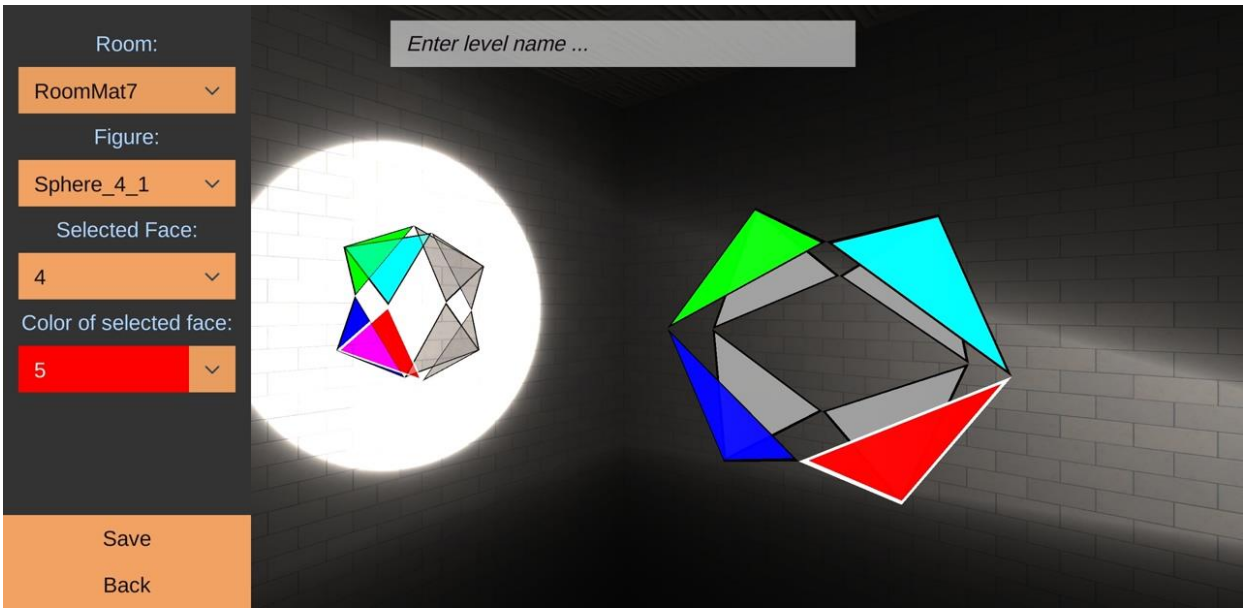

Рисунок 4.4 – Экран режима создания уровней

Экран режима создания уровней содержит следующие интерактивные элементы:

• Поле ввода текста. Поле для ввода названия уровня, и как следствие названия файла уровня.

• Выпадающий список текстур комнаты. Список доступных для выбора текстур комнаты.

• Выпадающий список форм фигуры. Список доступных для выбора форм фигуры.

• Выпадающий список граней фигуры. Список, основывающийся на выбранной фигуре и количестве ее гране. Дает возможность выбора грани фигуры. Выбранная грань фигуры обозначается в игровом мире белыми гранями.

• Выпадающий список цветов грани. Список основывающийся на выбранной грани фигуры. Дает возможность выбрать цвет грани фигуры из предложенных.

• Кнопка «Save». При нажатии происходит проверка введенного имени уровня. Если поле ввода имени уровня пустое, то пользователь оповещается текстовым сообщением о необходимости ввода имени уровня. При непустом имени уровня, уровень сохраняется в файловой систему устройства в виде файла с таким же названием, а пользователь оповещается текстовым сообщением о успешном сохранении уровня.

• Кнопка «Back». При нажатии происходит переход к главному игровому меню. Переход осуществляется без сохранения совершенных действий на этом экране.

• Джойстики. Представлены 3 джойстика для организации двух типов управления, при этом одновременно все три не могут быть активны.

4.5 Меню настроек мобильной игры

В меню настроек мобильной игры можно попасть посредствам нажатия соответствующей кнопки на главном меню игры. Меню настроек игры дает возможность взаимодействуя с интерактивными элементами пользовательского интерфейса настраивать некоторые параметры игрового процесса, такие как звук, способ управления, качество графики, а также возможность сброса прогресса пользователя в прохождении уровня. Скриншот экрана, содержащего меню настроек, представлен на рисунке 4.5.

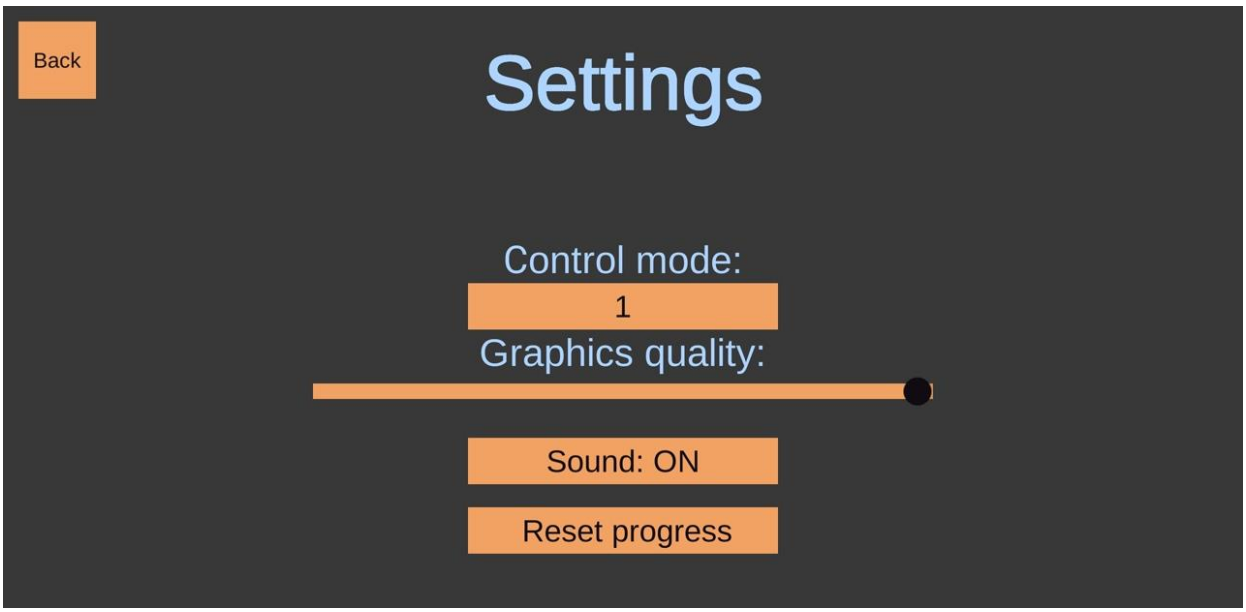

Рисунок 4.5 – Меню настроек мобильной игры

Меню настроек содержит следующие интерактивные элементы:

• Кнопка «Back». При нажатии происходит переход к главному меню.

• Кнопка «Control mode». При нажатии меняется способ управления: 2 джойстика или 1 джойстик. Надпись кнопки всегда указывает текущий номер способа управления.

• Слайдер настройки графики. Слайдер, позволяющий задать коэффициент, уменьшающий разрешение экрана, при отображение игрового мира. Правое положение слайдера: коэффициент 1, исходное разрешение экрана. Левое положение слайдера: коэффициент 0,1, 10% от исходного разрешение экрана.

• Кнопка «Sound». При нажатии факт наличия звукового сопровождения в игре меняется на противоположный. Надпись на кнопке всегда указывает на наличие звука в игре.

• Кнопка «Reset progress». При нажатии, весь прогресс пройденных уровней сбрасывается до стартового.

4.6 Меню паузы мобильной игры

Меню паузы может быть открыто, при нажатии соответствующей кнопки на главном игровом экране. Меню открывается поверх главного игрового экрана, делая его неактивным и приостанавливая игривой процесс. Скриншот экрана, содержащего меню паузы, представлен на рисунке 4.6.

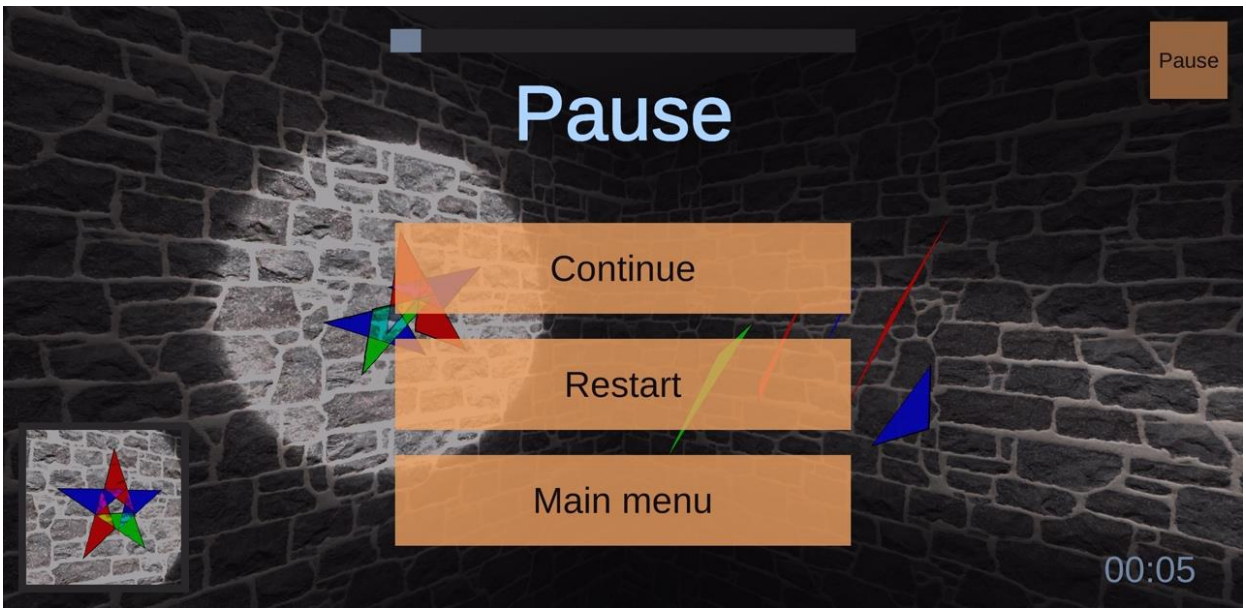

Рисунок 4.6 – Меню паузы

Меню паузы содержит следующие интерактивные элементы:

• Кнопка «Continue». При нажатии происходит закрытие меню паузы и возврат к главному игровому экрану, с сохранением игрового процесса уровня.

• Кнопка «Restart». При нажатии происходит переход к главному игровому экрану с полным сбросом игрового процесса уровня до стартового.

• Кнопка «Main menu». При нажатии происходит переход к главному меню игры.

4.7 Меню пройденного уровня

В меню пройденного уровня возможно попасть, только в том случае, если при прохождении уровня на главном игровом экране, уровень был пройден и было достигнуто финишное положение фигуры и ее тени. Меню пройденного уровня открывается поверх главного игривого экрана, делая его неактивным и приостанавливая игривой процесс. На экране меню пройденного уровня так же отображается время, затраченное на его прохождение. Скриншот экрана, содержащего меню пройденного уровня, представлен на рисунке 4.7.

Меню пройденного уровня содержит следующие интерактивные элементы:

• Кнопка «Next». При нажатии происходит переход к меню выбора уровня.

• Кнопка «Restart». При нажатии происходит переход к главному игровому экрану с полным сбросом игрового процесса уровня до стартового, но с сохранением факта прохождения уровня.

• Кнопка «Main menu». При нажатии происходит переход к главному меню игры.

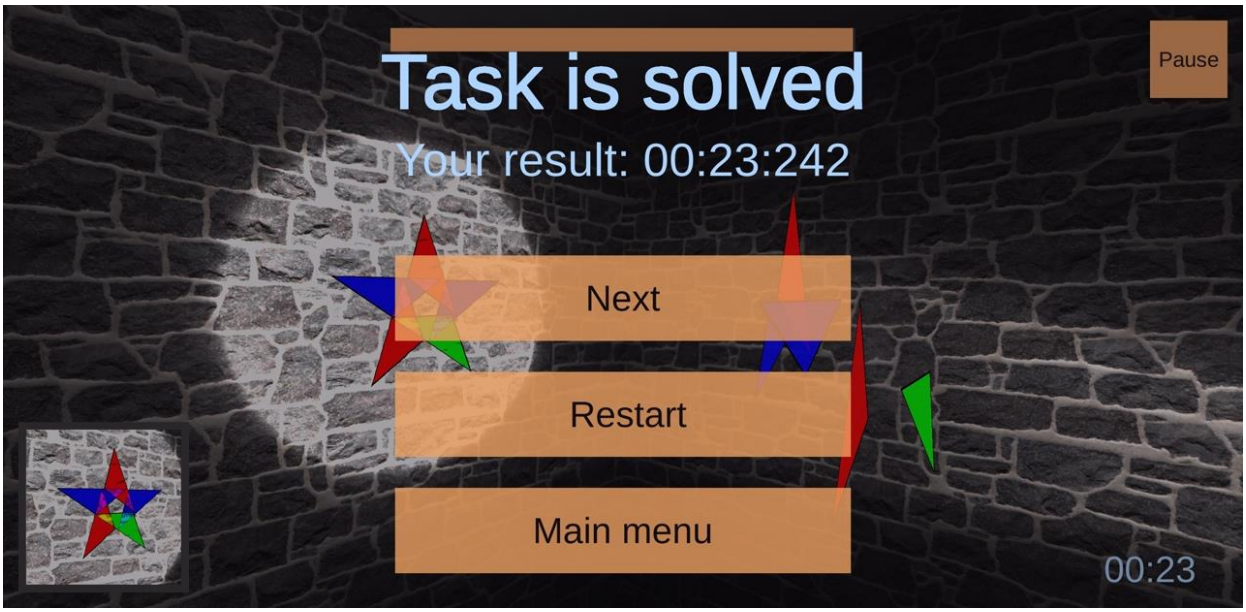

Рисунок 4.7 – Меню пройденного уровня

4.8 Вывод

Разработанный пользовательский интерфейс удовлетворяет всем поставленным требованиям, понятен и прост в использовании без дополнительных знаний и умений.

5 Тестирование мобильной игры

Необходимо провести тестирование разработанной мобильной игры «Puzzle Vitrage». А именно, определить удовлетворение разработанной игры требованиям, описанным в первой главе.

- Функциональные тесты в соответствии с требованиями к функциональным возможностям.
- Нагрузочные тесты, показывающие производительность.
- Тестирование интерфейса.

5.1 Функциональны тесты мобильной игры

Проведено тестирование мобильной игры на предмет наличия всех описанных в требованиях функций. Тестирование описанных ниже функций мобильной игры завершилось успехом, что доказывает удовлетворение требуемым функциональным возможностям.

- Управление посредствам сенсорной системы ввода.
- Взаимодействие с игрой в режиме прохождения уровней.
- Взаимодействие с игрой в режиме создания уровней.
- Сохранение общего игрового прогресса.
- Возможность запуска базовых и пользовательский уровней.
- Взаимодействие в реальном времени с игровым миром.
- Возможность вращения фигуры во всех плоскостях.
- Возможность настройки параметров пользовательского уровня в режиме создания уровней
- Сохранение пользовательских уровней в файле общего для уровней формате.
- Возможность сброса прогресса в прохождении уровней.
- Возможность настройки качества графики.
- Возможность регулирование наличия звукового сопровождения.

5.2 Нагрузочные тесты мобильной игры

Для проведения нагрузочных тестов, использовано 3 мобильных устройства со следующими важными характеристиками.

- Samsung Galaxy A72. Дисплей – 2400x1080. GPU – Adreno 618, 2019 года выпуска с тактовой частотой 700 MHz.
- Huawei P50 Pro. Дисплей – 2700×1228. GPU – Adreno 660, 2020 года выпуска с тактовой частотой 792 MHz.
- Xiaomi Mi 9. Дисплей – 2340×1080. GPU – Adreno 640, 2019 года выпуска с тактовой частотой 768 MHz.

Данные мобильные устройства полностью удовлетворяют минимальным требованиям к составу и параметрам технических средств разрабатываемой игры. Для оценки производительности и проведения нагрузочных тестов, на каждом из устройств была запущенна игра «Puzzle Vitrage». Тестирование показало устойчивую работу мобильной игры на устройствах при любых сценариях ее использования.

Для тестирования производительности на всех мобильных устройствах был запущен уровень, с фигурой из двадцати полигонов и различными настройками графики. Ниже представлены средние значения количества кадров в секунду полученных на описанных выше устройствах для данного сценария при минимальной, средней и максимальной настройках графики.

- Samsung Galaxy A72. Min – 43 fps. Middle – 15 fps. Max – 7 fps.
- Huawei P50 Pro. Min – 119 fps. Middle – 81 fps. Max – 37 fps.
- Xiaomi Mi 9. Min – 59 fps. Middle – 37 fps. Max – 16 fps.

По полученным результатам тестов, видно, что количество кадров в секунду не отпускается ниже 10 fps ни на одном устройстве при минимальном качестве графики. Это означает, что разработанная игра успешно прошла нагрузочные тесты и удовлетворяет требованиям в производительности и надежности.

5.3 Тестирование интерфейса мобильной игры

Проведено тестирование интерфейса мобильной игры на предмет удовлетворения выдвигаемым требованиям и разработанной структуре. По результатам тестов получены следующие результаты.

- Пользовательский интерфейс мобильной игры полностью совпадает со спроектированным в 4 главе интерфейсом.
- Интерфейс мобильной игр интуитивно понятен, легок в использовании и не требует специальных навыков.
- Все интерактивные объекты пользовательского интерфейса работают верно и активны для использования.

5.4 Вывод

Мобильная игра успешно прошла все описанные тесты и полностью удовлетворяет требованиям, предъявляемым к продукту. Все требуемые функции были успешно реализованы в мобильной игре, производительность не падает ниже требуемой и пользовательский интерфейс удобен и практичен в использовании. Помимо прочего можно заметить, сильный рост производительности на мобильных устройствах с более новыми и мощными GPU. Однако часто, чем новее мобильное устройство, тем больше разрешение экрана. Введенное же ограничение на максимальное разрешение изображения в игре способствует более яркому проявлению возможностей новых GPU и сильно поднимает производительность мобильной игры при их использовании.

6 Экономическое обоснование разработки игры

В данной выпускной квалификационной работе (ВКР) преследуется следующая цель: разработка мобильной игры с использованием алгоритмов трассировки лучей и среды разработки Unity, запускаемой на мобильных устройствах под управлением операционной системы Android. В работе реализуются алгоритмы игровой логики и алгоритмы визуализации, основанные на трассировке лучей. Так же разработана система интерфейсов реализующая взаимодействие пользователя с игрой.

Данная игра создана в межплатформенной среде разработки игр Unity, что позволяет успешно выложить мобильную версию данной игры на всемирно известный рынок Android приложений Google Play Market. Поскольку при использовании Unity был использован тарифный план «Personal», то разработка и распространение созданного продукта становится абсолютно бесплатным, хотя это и накладывает определенные ограничения, но для данной работы они несущественны. Помимо Unity для разработки игры использовалась бесплатная интегрированная среда разработки Visual Studio 2019. При всем этом можно получать некоторый доход с мобильной игры от рекламодателей, за счет размещения их рекламы внутри игрового процесса.

6.1 Расчет затрат на оплату труда

На первом этапе определяется предполагаемое время, затраченное на каждый этап проектирования. Продолжительность работ определяется расчетным путем с помощью экспертных оценок по формуле 6.1:

$$t^{o}_{j} = (3 \times t_{min} + 2 \times t_{max}) / 5, \quad (6.1)$$

где t_j^o – ожидаемая длительность j-й работы; t_{min} и t_{max} – наименьшая и наибольшая по мнению эксперта длительность работы [12]. Полученные данные об ожидаемой продолжительности каждого этапа представлены в таблице 6.1.

Таблица 6.1 – Длительность этапов разработки

№	Наименование работы	Исполнитель	Длительность работы, чел-день		
			tmin	tmax	to
1	Обзор литературы по теме трассировки лучей и разработке мобильных игр.	Г. А. Петров	4	6	4,8
2	Анализ задачи создания мобильной игры для Android.	Г. А. Петров	2	4	2,8
		А. И. Водяхо	1	2	1,4
3	Определение путей и методов разработки игры.	Г. А. Петров	12	17	14
		А. И. Водяхо	1	1	1
4	Разработка мобильной игры с использованием среды Unity.	Г. А. Петров	27	34	29,8
		А. И. Водяхо	1	2	1,4
5	Разработка пользовательского интерфейса игры.	Г. А. Петров	10	15	12
		А. И. Водяхо	1	1	1
6	Выполнение экономического обоснования.	Г. А. Петров	1	3	1,8
7	Оформление пояснительной записки к ВКР.	Г. А. Петров	8	13	10
8	Предварительное рассмотрение работы руководителем.	А. И. Водяхо	1	2	1,4
ИТОГО		Г. А. Петров	64	92	75,2
		А. И. Водяхо	5	8	6,2

Далее определяется ставка заработной платы в день для исполнителя каждого из этапов. При этом учитывается, что в месяце содержится 21 рабочий день [12], и, основываясь на этом факте можно определить ежедневную ставку исполнителя исходя из его ежемесячной ставки. В качестве месячной заработной платы студента взята месячная заработная плата инженера. Согласно сайту sankt-peterburg.jobfilter.ru [13] средняя зарплата для профессии «Инженер» в Санкт-Петербурге за последние 12 месяцев составила примерно 55 000 рублей. Согласно сайту sankt-peterburg.jobfilter.ru [14] за последние 12 месяцев средняя заработная плата по Санкт-Петербургу по профессии профессор составила 50 000 рублей, однако в СПбГЭТУ «ЛЭТИ», на базе которого выполняется ВКР, заработная плата профессора превышает среднюю заработную плату по региону примерно в два раза. По полученным данным рассчитаны дневные ставки для обеих профессий:

$$С_{прог} = 55000 \text{ руб.} / 21 \text{ день} = 2619{,}05 \text{ (руб./день)},$$

$$С_{проф} = 2 \times 50000 \text{ руб.} / 21 \text{ день} = 4761{,}90 \text{ (руб./день)}.$$

Полученные после расчетов результаты представлены в таблице 6.2.

Таблица 6.2 – Исполнители и трудоемкость этапов разработки

№	Этапы выполняемых работ	Исполнитель	Трудоемкость, to, чел-день	Ставка, руб./чел-день
1	Обзор литературы по теме трассировки лучей и разработке мобильных игр.	Г. А. Петров	4,8	2619,05
2	Анализ задачи создания мобильной игры для Android.	Г. А. Петров	2,8	2619,05
		А. И. Водяхо	1,4	4761,9
3	Определение путей и методов разработки игры.	Г. А. Петров	14	2619,05
		А. И. Водяхо	1	4761,9
4	Разработка мобильной игры с использованием среды Unity.	Г. А. Петров	29,8	2619,05
		А. И. Водяхо	1,4	4761,9
5	Разработка пользовательского интерфейса игры.	Г. А. Петров	12	2619,05
		А. И. Водяхо	1	4761,9
6	Выполнение экономического обоснования разработки мобильной игры.	Г. А. Петров	1,8	2619,05
7	Оформление пояснительной записки к ВКР.	Г. А. Петров	10	2619,05
8	Проверка ВКР руководителем.	А. И. Водяхо	1,4	4761,9

Теперь, зная дневную ставку каждого исполнителя и продолжительность каждого этапа в днях, рассчитаем затраты на основную и дополнительную заработные платы, а также затраты на социальные выплаты. Расходы на основную заработную плату исполнителей определяются по формуле 6.2:

$$З_{осн.\ з/пл} = \sum_{i=1}^{k} T_i \times C_i, \quad (6.2)$$

где $З_{осн.з/пл}$ – расходы на основную заработную плату исполнителей (руб.); k – количество исполнителей; T_i – время, затраченное i-м исполнителем на проведение исследования (дни); C_i – ставка i-го исполнителя (руб./день) [12].

После подстановки в формулу 6.2 данных из таблицы 6.2 получился следующий результат:

$$З_{осн.\ з/пл} = 4{,}8 \times 2619{,}05 + 2{,}8 \times 2619{,}05 + 1{,}4 \times 4761{,}9 + 14 \times 2619{,}05$$
$$+ 1 \times 4761{,}9 + 29{,}8 \times 2619{,}05 + 1{,}4 \times 4\,761{,}9 + 12 \times 2619{,}05$$
$$+ 1 \times 4761{,}9 + 1{,}8 \times 2619{,}05 + 10 \times 2619{,}05 + 1{,}4 \times 4761{,}9$$
$$= 226476{,}34 \text{ (руб.)}.$$

Расходы на дополнительную заработную плату исполнителей определяются по формуле 6.3:

$$З_{доп.\ з/пл} = З_{осн.\ з/пл} \times Н_{доп.} / 100, \quad (6.3)$$

где $З_{доп.\ з/пл}$ – расходы на дополнительную заработную плату исполнителей (руб.); $З_{осн.\ з/пл}$ – расходы на основную заработную плату исполнителей (руб.); $Н_{доп.}$ – норматив дополнительной заработной платы (%) [12].

Значение норматива на дополнительную заработную плату составляет 14% в соответствии с методическими указаниями [12]. После подстановки, полученной ранее основной заработной платы в формулу 6.3, получаем следующее значение:

$$З_{доп.\ з/пл} = 22647{,}3 \times 14 / 100 = 31706{,}69 \text{ (руб.)}.$$

Отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование с основной и дополнительной заработной платы исполнителей определяются по формуле 6.4:

$$З_{соц.} = (З_{осн.\ з/пл} + З_{доп.\ з/пл}) \times Н_{соц.} / 100, \quad (6.4)$$

где $З_{соц}$ – отчисления на социальные нужды с заработной платы (руб.); $З_{осн.з/пл}$ – расходы на основную заработную плату исполнителей (руб.); $З_{доп.з/пл}$ – расходы на дополнительную заработную плату исполнителей (руб.); $Н_{соц}$ – норматив отчислений на страховые взносы на обязательное социальное, пенсионное и медицинское страхование [12].

При выполнении расчетов в ВКР на текущий момент времени норматив отчислений на страховые взносы равен 30%. Результат вычислений, полученных путем подстановки значений основный и дополнительной заработной платы в формулу 6.4 приведен ниже:

$$З_{соц} = (226476{,}34 + 31706{,}69) \times 0{,}3 = 77454{,}91 \text{ (руб.)}.$$

6.2 Расчет накладных расходов

Поскольку работа выполняется на базе университета «ЛЭТИ», то принимаем норматив накладных расходов для данной работы, как долю в 20% от суммы основной и дополнительной заработных плат. По результатам расчетов получаем:

$$С_{нр} = (226476{,}34 + 31706{,}69) \times 0{,}2 = 51636{,}61 \text{ (руб.)}.$$

6.3 Расчет затрат на сырье и материалы

Затраты на сырье и материалы рассчитываются по формуле 6.5:

$$З_М = \sum_{l=1}^{L} G_l \times Ц_l \times (1 + Н_{т.з.} / 100), \quad (6.5)$$

где $З_м$ – затраты на сырье и материалы (руб.); l – индекс вида сырья или материала; G_l – норма расхода l-го материала на единицу продукции (ед.); $Ц_l$ – цена приобретения единицы l-го материала (руб./ед.); $Н_{т.з}$ – норма транспортно-заготовительных расходов (%) [12].

В этой работе принимаем нормой транспортно-заготовительных расходов ($Н_{т.з}$) значение в 10% от суммы средств затраченных на сырье и материалы [12]. Расчет затрат на сырье и материалы сведен в таблицу 6.3.

Таблица 6.3 – Затраты на сырье и материалы

Материал	Тип (профиль, сорт, марка, размер)	Норма расхода на изделие, ед.	Цена, руб./ед.	Сумма на изделие, руб.
Бумага офисная	"Снегурочка" А4, 500 листов [15]	1	500	500
Картридж для принтера	Картридж лазерный Sonnen (SC-725) для Canon LBP6000/LBP6020/LBP6020B [16]	1	619	619
Флэш-накопитель	USB Флэш-накопитель Kingston DataTraveler 100 G3 DT100G3/64GB 64 ГБ, черный [17]	1	724	724

По значениям, указанным в таблице 6.3 рассчитаны общие затраты на сырье и материалы:

$$З_М = 500 \times 1{,}1 + 619 \times 1{,}1 + 724 \times 1{,}1 = 2027{,}3 \text{ (руб.)}.$$

6.4 Расчет затрат на услуги сторонних организаций

Ставка НДС на текущий момент времени составляет 20 %. Затраты на услуги сторонних организаций определяются по сумме реальных затрат без учета НДС. Полученные затраты на услуги сторонних организаций сведены в таблицу 6.4.

Таблица 6.4 – Затраты на услуги сторонних организаций

Наименование	Норма расхода на единицу продукции, ед.	Цена, руб./ед.	Сумма, руб.	Сумма без НДС, руб.
Интернет, "Well Telecom" [18]	3 месяца	400	1200	1000
Печать документов в копировальном центре "Копирка" [19]	90 страниц	16	1440	1200

По результатам, указанным в таблице 6.4 рассчитаны общие затраты на услуги сторонних организаций за вычетом НДС:

$$ЗП = 1000 + 1200 = 2200 \text{ (руб.)}.$$

6.5 Расчет затрат на амортизацию ПК

Амортизационные отчисления по основному средству i за год определяются по формуле 6.6 следующим образом:

$$Аi = Цп.н.i \times Наi / 100, \quad (6.6)$$

где Аi – амортизационные отчисления за год по i-му основному средству (руб.); Цп.н.i – первоначальная стоимость i-го основного средства (руб.); Наi – годовая норма амортизации i-го основного средства (%) [12].

Поскольку при выполнении работы в качестве основного средства использовался ноутбук HP EliteBook 835 стоимостью на данный момент времени 136 000 рублей [20], то необходимо учесть и включить в затраты на амортизационные отчисления по нему. Согласно постановлению правительства РФ от 01.01.2002 №1 (ред. от 28.04.2018) «О Классификации основных средств, включаемых в амортизационные группы» персональные компьютеры и печатающие устройства к ним, относятся ко второй группе и имеют срок эксплуатации три года [21]. Тогда, выбрав линейный способ начисления амортизации, получена норма амортизации:

$$На = 100 / 3 = 33{,}3\%$$

Учитывая первоначальную стоимость ноутбука и норму амортизации в 33%, рассчитаны амортизационные отчисления за него по формуле 6.6:

$$Аi = 136000 \times 0{,}33 = 44880 \text{ (руб.)}.$$

Далее определим затраты на амортизационные отчисления по ноутбуку за время выполнения ВКР. Величина амортизационных отчислений по ноутбуку, при работе над ВКР, определяется по формуле 6.7:

$$A_{iВКР} = A_i \times T_{iВКР} / 12, \quad (6.7)$$

где $A_{iВКР}$ – амортизационные отчисления по i-му основному средству, используемому студентом в работе над ВКР (руб.); A_i – амортизационные отчисления за год по i-му основному средству (руб.); $T_{iВКР}$ – время, в течение которого студент использует i-ое основное средство (мес.) [12].

По формуле 6.7, учитывая трудоемкость этапов, в которых задействован ноутбук, из таблицы 6.2, 21 рабочий день в месяце и амортизационные отчисления по ноутбуку за год, получаем следующее значение суммарных амортизационных отчислений по ноутбуку за время выполнения ВКР:

$$A_{iВКР} = 44880 \times 75{,}2 / 21 \times 12 = 13392{,}76 \text{ (руб.)}.$$

6.6 Расчет затрат на содержание и эксплуатацию ноутбука

Затраты на содержание и эксплуатацию ноутбука определяются из расчета на 1 час работы компьютера (машино-часа, м-ч) с учетом стоимости и производительности компьютера по формуле 6.7:

$$З_{ЭО} = \sum_{i=1}^{m} C_{мч\ i} \times t_{м\ i}, \quad (6.7)$$

где $З_{эо}$ – затраты на содержание и эксплуатацию оборудования (руб.); $C_{мч\ i}$ – расчетная себестоимость одного машино-часа работы компьютера на i-й технологической операции (руб./м-ч); $t_{м\ i}$ – количество машино-часов, затрачиваемых на выполнение i-й технологической операции (м-ч) [12].

Себестоимость одного машино-часа работы компьютера определяется по формуле 6.8:

$$C_{мч} = Ц_{эл} + A_ч + Р_ч + З_{инт.ч}, \quad (6.8)$$

где $Ц_{эл}$ – затраты на электроэнергию (руб. /м-ч), $A_ч$ – часовая амортизация компьютера (руб. /м-ч), $Р_ч$ – часовые затраты на техническое обслуживание и ремонт компьютера (руб./м-ч), $З_{инт.ч}$ – часовые затраты на использование Интернета (руб. /м-ч).

Затраты на электроэнергию в час рассчитываются исходя мощности потребления электричества блоком питания для ноутбука в час, для используемой модели блока питания 19 В – напряжение, и 4,74 А – сила тока [22], и текущему тарифу стоимости электроэнергии 4,98 руб./кВт-ч [23]:

$$Ц_{эл} = 19 \times 4{,}74 \times 4{,}98 / 1000 = 0{,}45 \text{ (руб./м-ч)}.$$

Затраты на амортизацию и затраты на Интернет уже учитываются в пунктах расчета амортизационных отчислений и расчета затрат на услуги сторонних организаций соответственно, поэтому в данном случае их учитывать не будем. Часовые затраты на тех. обслуживание и ремонт определяются исходя из годовых отчислений на эти нужды и того факта, что рабочий день длится 8 часов и в месяце 21 рабочий день, при этом принято решение для данной ВКР взять за годовые отчисления значение в 5% от первоначальной стоимости ноутбука. Количество машино-часов работы с ноутбуком определяется по таблице 6.1. Учитывая то, что рабочий день длится 8 часов, а все этапы работы, кроме последнего, выполнялись на ноутбуке, получаем:

$$t_{мч} = 75{,}2 \times 8 = 601{,}6 \text{ (м-ч)}.$$

Промежуточные и итоговые результаты расчетов по формуле 6.8 сведены в таблицу 6.5.

Таблица 6.5 – Затраты на содержание и эксплуатацию ноутбука

Наименование пункта расчетов	Значение
Затраты на электроэнергию, руб./м-ч	0,45
Часовые затраты на тех. обслуживание и ремонт, руб./м-ч	3,37
Расчетная себестоимость, руб./м-ч	3,82
Количество машино-часов, м-ч	601,6

По полученным данным, используя формулу 6.7, определим общие затраты на содержание и эксплуатацию ноутбука:

$$З_{эо} = 601{,}6 \times 3{,}82 = 2299{,}02 \text{ (руб.)}.$$

6.7 Расчет себестоимости ВКР

Для расчета совокупной величины затрат, связанных с написанием программы, все проведенные расчеты сводятся в таблицу 6.6.

Таблица 6.6 – Калькуляция затрат на ВКР

№	Наименование статьи	Сумма, руб.
1	Затраты на оплату труда	258183,03
2	Отчисления на социальные нужды	77454,91
3	Затраты на сырье и материалы	2027,3
4	Услуги сторонних организаций	2200
5	Затраты на содержание и эксплуатацию оборудования	2299,02
6	Амортизационные отчисления	13392,76
7	Накладные расходы	51636,61
ИТОГО затрат		407193,63

Себестоимость проекта составила 407193 рубля и 63 копейки.

6.8 Выводы

По результатам расчетов можно выделить два основных показателя: общее время работ над игрой (601,6 часов) и сумма затрат на ее производство (407193 рубля и 63 копейки). Ожидаемый экономический эффект от разработки данной игры представляется в виде получения определенного стабильного ежемесячного дохода от внедренной в игру рекламы, при этом предполагается, что затраты на разработку игры покроются этими доходами в течение довольно короткого времени. А именно, предположив среднемесячный доход от игры в 20000 рублей [24], игра должна окупиться в течение ближайших двадцати месяцев. Таким образом, можно считать реализацию проекта экономически целесообразной.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе представлена информация о наиболее популярных и эффективных средах разработки игры и их движках. Были сформулированы требования выдвигаемые при выборе среды разработки игр и на основании проведенного анализа их достоинств и недостатков, была выбрана межплатформенная среда разработки игры Unity.

В процессе выполнения выпускной квалификационной работы была разработана готовая к использованию трехмерная мобильная игра, которая может быть использована людьми на мобильных устройствах под управлением ОС Android. Разработка игры велась с использованием межплатформенной среды разработки игр Unity и алгоритмов рейтрейсинга на языках C# и HLSL.

Мобильная игра разрабатывалась в межплатформенной среде разработки Unity, что позволит в дальнейшем без существенных затрат собрать игру под другие платформы и дает хорошие показатели графики и производительности. Это, наряду с использованием невероятно популярных на данный момент алгоритмов трассировки лучей, позволит поднять актуальность игры «Puzzle Vitrage», выйти на рынок мобильных игр и начать получать прибыль от продаж и размещения внутриигровой рекламы.

В отличие от известной мобильной игры «Shadowmatic», разработанная мобильная игра «Puzzle Vitrage», расширяет функционал и игровой процесс. За счет использования рейтрейсинга, в игре используются цветные тени от полупрозрачных объектов, цвета которых могут смешиваться при пересечении, что серьезно расширяет возможности организации игры и увеличивает сложность. Помимо этого, в игру добавлена возможность создания пользовательских уровней с сохранением их в файле общего формата, которым затем можно обмениваться с другими пользователями и запускать их для прохождения.

Путь дальнейшего развития игры представляется в виде расширения количества базовых уровней, фигур и текстур. Не исключается возможность

улучшения алгоритма рендеринга в плане графики и оптимизации производительности. Помимо прочего, предполагается выход на другие мобильные платформы и интеграция с рекламными сервисами для получения прибыли.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт ApIndex [Электронный ресурс] URL: https://adindex.ru/news/researches/2021/06/15/295311.phtml

2. Официальный сайт Медиа нетологии [Электронный ресурс] URL: https://netology.ru/blog/12-2021-mobile-games-evolution

3. Официальный сайт DNS клуб [Электронный ресурс] URL: https://club.dns-shop.ru/digest/23036-reitreising-v-mobilnyih-igrah/

4. Официальный сайт Яндекс Радар [Электронный ресурс] URL: https://radar.yandex.ru/mobile?period=all&device-category=2

5. Официальный сайт Unity Documentation [Электронный ресурс] URL: https://docs.unity3d.com/Manual/index.html

6. Леоненков А., Самоучитель UML2 : БВХ - Петербург, 2007. – 568 с.

7. Официальный сайт Unity [Электронный ресурс] URL: https://unity.com/

8. Грегори Джейсон, Игровой движок. Программирование и внутреннее устройство. Третье издание. – СПб.: Питер, 2021. – 1136 с.: ил. – (Серия «Для профессионалов»).

9. Официальный сайт AppBrain [Электронный ресурс] URL: https://www.appbrain.com/stats/libraries/details/unity/unity-3d

10. Роджерс Д., Алгоритмические основы машинной графики: Пер. с англ. – М.: Мир, 1989. – 512 с, ил.

11. Möller Tomas, Trumbore Ben. Fast, Minimum Storage Ray/Triangle Intersection // Journal of Graphics Tools – 1997.

12. Алексеева О. Г., Выполнение дополнительного раздела ВКР бакалаврами технических направлений: учеб.-метод. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 16 с.

13. Сайт вакансий Фильтр работ [Электронный ресурс] URL: https://sankt-peterburg.jobfilter.ru/career/инженер

14. Сайт вакансий Фильтр работ [Электронный ресурс] URL: https://sankt-peterburg.jobfilter.ru/career/профессор

15. Официальный сайт Ozon [Электронный ресурс] URL: https://www.ozon.ru/product/bumaga-ofisnaya-snegurochka-500-listov-a4-8234836/?asb=wQ405SScr5HbQpoERWCMOfZScJhVncWiv6mqtHGmHJg%253D&asb2=HiZMcorV0TXkfUBxIg-kUzCUHvpQkc2zImj7tY6CGxbO_ZYc2wlkks0a0q_cZNBO&keywords=бу-мага+а4&sh=VsonjFwVuQ

16. Официальный сайт Ozon [Электронный ресурс] URL: https://www.ozon.ru/product/kartridzh-lazernyy-sonnen-sc-725-dlya-canon-lbp6000-lbp6020-lbp6020b-resurs-1600-stranits-159127587/?asb=Ufq6mvvi2R22HbLd1EiIsFBsy1xNBtI5JSP1oLfBT%252Bg%253D&asb2=hOnvpox4aPpRNr9KSxFki-mCCa17u7HgNYcgkIDuh8uYQNbWnHToMQ0AY7MFyfLi&keywords=карт-ридж+для+принтера+canon&sh=VsonjPGytQ

17. Официальный сайт Ozon [Электронный ресурс] URL: https://www.ozon.ru/product/flesh-nakopitel-usb-3-0-64gb-kingston-data-traveler-dt100-g3-21892939/?asb=uQWaO4%252BG7XUNzwMaCqWTtsw4tqMaUxb7IcbOIPW%252BNgA%253D&asb2=cD5td5ak43-awzudVOeqI_gPRUDBEtzEDrgTvbXG2Z5XfclJsn1mRmhKnh_OHFOu&keywords=флешка+64пи&sh=VsonjMQ0Xw

18. Официальный сайт Well Telecom [Электронный ресурс] URL: https://billing.well-telecom.ru/?act=services

19. Официальный сайт Копирка [Электронный ресурс] URL: https://kopirka.ru/uslugi/kopirovanie-dokumentov/

20. Официальный сайт Citilink [Электронный ресурс] URL: https://www.citilink.ru/product/noutbuk-hp-elitebook-835-g8-ryzen-5-pro-5650u-16gb-ssd512gb-13-3-uwva-

1541026/?yclid=21850473480756192 10&utm_source=yandex&utm_medium=cpc&utm_campaign=cities-srch-cat-dsa_fid_test_tov&utm_term=&utm_content=%7Cc%3A67513111%7Cg%3A4748001623%7Cb%3A11401696985%7Ck%3A2014971%7Cst%3Asearch%7Ca%3Ano%7Cs%3Anone%7Ct%3Apremium%7Cp%3A1%7Cr%3AНоут-буки%7Cdev%3Adesktop%7Ccgci:9327951&utm_param1=all

21. О классификации основных средств, включаемых в амортизационные группы: постановление Правительства Рос. Федерации от 01.01.2002 №1 (ред. от 28.04.2018) // Собрание законодательства Российской Федерации

22. Официальный сайт Teslocom [Электронный ресурс] URL: https://teslocom.ru/product/ppp014l-sa-blok-pitaniya-dlya-noutbukov-hp-19v-474a-74-50mm/

23. Официальный сайт Комитет по тарифам Санкт-Петербурга [Электронный ресурс] URL: http://www.tarifspb.ru/documents/acts/3098/

24. Официальный сайт Unity 3D School [Электронный ресурс] URL: https://unity3dschool.com/skolko-mozhno-zarabotat-na-mobilnoj-igre.html

ПРИЛОЖЕНИЕ А

Исходный код мобильной игры

MeshRT.cs

```
using UnityEngine;
//Структура предназначена для хранения дополнительной
//информации о мешах объектов, поскольку основная информация,
//вершины и полигоны, передаются через буферы
struct MeshRT
{
    public Matrix4x4 localToWorldMatrix; //матрица преобрзования
    public int indices_offset; //место записи индексов в буфере для обхекта
    public int indices_count; //количество индексов в буфере объекта
    public float transparency; //прозрачность объекта

    //метод который возвращает размер структуры
    public static int GetSize()
    {
        return sizeof(float)*4*4
            + sizeof(int)
            + sizeof(int)
            + sizeof(float);
    }
}
```

Raytracer.cs

```
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

//Класс Raytracer занимается организацией рендеринга изображения
//А именно: регистрацией данных о объектах рендеринга
//Назначением текстуры рендеринга
//и управление параметрами шейдера для обраотки текстуры
//Класс предпологается использовать как компонент камеры сцены,
//для перехвата и обработки ее изображений
public class Raytracer : MonoBehaviour
{
    private const float PI_180 = 0.01745329f;//константа пи
    public Sprite spriteHint = null; //спрайт изображение подсказки для уровня
    public float selectedFace = -10; //выделенная грань
    public bool creatingMode = false; //режим создания уровня

    //ДЛя передачи данных шейдеры, параметры дожны быть сериализуемы
    [SerializeField]
    private ComputeShader raytracingShader;
    //сам вычислительный шейдер с рейстреиснгом
    [SerializeField]
    private Texture skyboxTexture; //текстура фона (неба)
    [SerializeField, Range(1f, 20f)]
    private int transparencyCount = 4; //количество итерации для прозрачности
    [SerializeField]
    private Light spotLight; //направленный источник света
    [SerializeField]
    private Light pointLight; //точечный источник света
```

```
private RenderTexture target = null;
//текстура ренедеринга, вычислением которой займется шейдер
private Camera componentCamera;
//камера сцены, которая символизирует взгляд на сцену

//RAYTRACING OBJECTS
private static bool meshObjectsNeedRebuilding = false;
//надо ли пересобрать все буферы, если появился новый объект
private static List<RayTracingObject> rayTracingObjects =
new List<RayTracingObject>(); //список объектов рейтресинга

//Списки для работы с информацией о мешах объектов
private static List<MeshRT> meshObjects = new List<MeshRT>();
//список структур мешей объектов
private static List<Vector3> vertices = new List<Vector3>();
//список вершин объектов
private static List<Vector2> verticesUV = new List<Vector2>();
//список текстурын хкоординат вершин
private static List<int> indices = new List<int>();
//список индексов объектов
private static List<int> colors = new List<int>(); //список цветов граней
//Буферы для передачи шейдеру
private ComputeBuffer meshObjectBuffer; //буфер структуры мешей
private ComputeBuffer vertexBuffer; //буфер вершин
private ComputeBuffer vertexUVBuffer; //уфер координат текстур
private ComputeBuffer indexBuffer; //буфер индексов
private ComputeBuffer colorBuffer; //буфер цветов граней

//Функция вызывается после включения обхекта на сцене
private void OnEnable()
{
    //получить компонент камеры
    componentCamera = GetComponent<Camera>();
}

//Функция срабатывает когда объект становится неактивынм
private void OnDisable()
{
    //очистка всех буферов
    meshObjectBuffer?.Release();
    vertexBuffer?.Release();
    vertexUVBuffer?.Release();
    indexBuffer?.Release();
    colorBuffer?.Release();
}

//Метод которы вызывается обхектом, для его регистрации
public static void RegisterObject(RayTracingObject obj)
{
    rayTracingObjects.Add(obj);//добавить обхект в список обхектов
    meshObjectsNeedRebuilding = true;//запросить перестройку буферов
}


//Метод для очистки памяти об объекте
public static void UnregisterObject(RayTracingObject obj)
{
    rayTracingObjects.Remove(obj);//удалить обхект из списка
    meshObjectsNeedRebuilding = true;//запросить перестройку буферов
}

//Метод который перестраивает только самый необходимый буфер
//с матрицей преобразования
```

```
private void RebuildMeshObjectBuffer()
{
    //пройтись по всем объектам
    for (int i = 0; i < rayTracingObjects.Count; i++)
    {
        //обновить значения матриц преобразования в списке для объектов
        MeshRT _meshObject = meshObjects[i];
        _meshObject.localToWorldMatrix =
        rayTracingObjects[i].transform.localToWorldMatrix;
        meshObjects[i] = _meshObject;
    }
    //обновление передаваемого буфера
    CreateComputeBuffer(ref meshObjectBuffer,
              meshObjects, MeshRT.GetSize());
}

//Метод который перестраивает все буферы общения с шейдером
private void RebuildAllObjectBuffers()
{
    //Очистка всех списков
    meshObjects.Clear();
    vertices.Clear();
    verticesUV.Clear();
    indices.Clear();
    colors.Clear();
    //важно чтобы объекты комнаты и фигуры были в одном и том же
    //порядке всегда
    if (rayTracingObjects[0].gameObject.name == "Object")
        rayTracingObjects.Reverse();
    //Проходимся по всем зарегестрированным обхектам в списке
    foreach (RayTracingObject obj in rayTracingObjects)
    {
        Mesh mesh = obj.GetComponent<MeshFilter>().sharedMesh;
        // Добавляем в списки данные о координатах вершин и текстур
        int firstVertex = vertices.Count;
        vertices.AddRange(mesh.vertices);
        verticesUV.AddRange(mesh.uv);
        //Запоминаем сдвиги и добавляем инфу об индексах в список
        int firstIndex = Raytracer.indices.Count;
        var indices = mesh.GetIndices(0);
        Raytracer.indices.AddRange(indices.Select(index => index
              + firstVertex));
        //добавляем инфу о цветах граней в список
        var colors = obj.facesColor;
        Raytracer.colors.AddRange(colors);
        //инициализируем структуру meshRT и добавляем в список
        meshObjects.Add(new MeshRT()
        {
            localToWorldMatrix = obj.transform.localToWorldMatrix,
            indices_offset = firstIndex,
            indices_count = indices.Length,
            transparency = obj.transparency,
        });
    }

    //По полученным новым списка создаем буферы
    CreateComputeBuffer(ref meshObjectBuffer, meshObjects,
        MeshRT.GetSize()); //sizeof(float*4*4 + int + int + float + float)
    CreateComputeBuffer(ref vertexBuffer, vertices,
        sizeof(float) * 3); //sizeof(float*3)
    CreateComputeBuffer(ref vertexUVBuffer, verticesUV,
        sizeof(float) * 2); //sizeof(float*3)
    CreateComputeBuffer(ref indexBuffer, indices,
```

```
            sizeof(int)); //sizeof(float)
        CreateComputeBuffer(ref colorBuffer, colors,
            sizeof(int)); //sizeof(float)
    }

    //Метод для создания буфера на основе его ссылки,
    //списка с данными и размера данных
    private static void CreateComputeBuffer<T>(ref ComputeBuffer buffer,
List<T> data, int stride)
    where T : struct
    {
        //если буфер уже имеется
        if (buffer != null)
        {
            //Если буфер не имеет нужную структуру, то сбросить буфер
            if (data.Count == 0 || buffer.count != data.Count
                  || buffer.stride != stride)
            {
                buffer.Release();
                buffer = null;
            }
        }

        //если список не пустой
        if (data.Count != 0)
        {
            //если буфер пустой, тогда создать его
            if (buffer == null)
            {
                buffer = new ComputeBuffer(data.Count, stride);
            }
            //Занести данные в буфер
            buffer.SetData(data);
        }
    }

    //Метод передает буфер в шейдер по имени поля в шейдере и буферу
    private void SetComputeBuffer(string name, ComputeBuffer buffer)
    {
        //если буфер не пустой
        if (buffer != null)
        {
            //установить поле шейдера такого же типа в значение данного буфера
            raytracingShader.SetBuffer(0, name, buffer);
        }
    }

    //Метод устанавливает только важные параметры шейдера
    private void SetUpdatableShaderParameters()
    {
        //Установка значения буфера с MeshRT,
        //где есть важная матрица преобразований
        SetComputeBuffer("_MeshObjects", meshObjectBuffer);
    }

    //Уставнока всех параметров шейдера
    private void SetAllShaderParameters()
    {
        //Установка парамтрой камеры
        raytracingShader.SetMatrix("_CameraToWorld",
            componentCamera.cameraToWorldMatrix);
        raytracingShader.SetMatrix("_CameraInverseProjection",
            componentCamera.projectionMatrix.inverse);
```

```
    raytracingShader.SetTexture(0, "_SkyboxTexture",
        skyboxTexture);
    raytracingShader.SetInt("_TransparencyCount",
        transparencyCount);
    //Установка параметров света
    Vector3 dir = spotLight.transform.forward;
    Vector3 posS = spotLight.transform.position;
    Vector3 posP = pointLight.transform.position;
    raytracingShader.SetVector("_SpotLightDirection",
        new Vector4(dir.x, dir.y, dir.z, spotLight.intensity));
    raytracingShader.SetVector("_SpotLightPosition",
        new Vector4(posS.x, posS.y, posS.z, PI_180 * spotLight.spotAngle));
    raytracingShader.SetFloat("_LightRange", pointLight.range);
    raytracingShader.SetVector("_PointLightPosition",
        new Vector4(posP.x, posP.y, posP.z, pointLight.intensity));
    //Уставнока параметров мешей объектов
    SetUpdatableShaderParameters();
    raytracingShader.SetFloat("_selectedFace", selectedFace);
    SetComputeBuffer("_Vertices", vertexBuffer);
    SetComputeBuffer("_VerticesUV", vertexUVBuffer);
    SetComputeBuffer("_Indices", indexBuffer);
    SetComputeBuffer("_Colors", colorBuffer);
    //Установка текстур и их свойств комнаты
    GameObject room = GameObject.Find("Room");
    raytracingShader.SetTexture(0, "_RoomMaskMap",
      room.GetComponent<MeshRenderer>().material.GetTexture("_MaskMap"));
    raytracingShader.SetTexture(0, "_RoomColorTexture1",
      room.GetComponent<MeshRenderer>().material.GetTexture("_MainTex1"));
    raytracingShader.SetTexture(0, "_RoomColorTexture2",
      room.GetComponent<MeshRenderer>().material.GetTexture("_MainTex2"));
    raytracingShader.SetTexture(0, "_RoomBumpMap1",
      room.GetComponent<MeshRenderer>().material.GetTexture("_BumpMap1"));
    raytracingShader.SetTexture(0, "_RoomBumpMap2",
      room.GetComponent<MeshRenderer>().material.GetTexture("_BumpMap2"));
    raytracingShader.SetFloat("_BumpScale1",
      room.GetComponent<MeshRenderer>().material.GetFloat("_BumpScale1"));
    raytracingShader.SetFloat("_BumpScale2",
      room.GetComponent<MeshRenderer>().material.GetFloat("_BumpScale2"));
    raytracingShader.SetVector("_TexTiling1",
 room.GetComponent<MeshRenderer>().material.GetTextureScale("_MainTex1"));
    raytracingShader.SetVector("_TexTiling2",
 room.GetComponent<MeshRenderer>().material.GetTextureScale("_MainTex2"));
}

//Метод инициализирует текстуру рейндеринга, которую будет считать шейдер
private void Init()
{
    //Есл текстура еще не была проинициализирвоана или
    //ее параметры не соответствую текущему экрану*натсройка качества
    if (target == null || target.width != (int)(Screen.width
        * CurrentData.Data.currentSettings.graphicsOption)
        || target.height != (int)(Screen.width
        * CurrentData.Data.currentSettings.graphicsOption))
    {
        //если текстура не пуста, очистить ее
        if (target != null)
        {
            target.Release();
        }
       //инициализируем текстуру рендеринга размерам и
       //иансттройками цветапередачи
        target = new RenderTexture((int)(Screen.width
              * CurrentData.Data.currentSettings.graphicsOption),
```

```
            (int)(Screen.height
            * CurrentData.Data.currentSettings.graphicsOption),
            0,
            RenderTextureFormat.ARGBFloat,
            RenderTextureReadWrite.Linear);
        target.enableRandomWrite = true;
        target.Create();
    }
}

//Метод занимается рендером текстуры
private void Render(RenderTexture destination)
{
    //инициализируем текстуру о необходимости
    Init();

    //Устанавливаем текстуру рендеринга шейдеру
    //как оле для записи результата
    raytracingShader.SetTexture(0, "Result", target);
    //делим экран на части для улучшения многопоточности
    int threadGroupsX = Mathf.CeilToInt((int)(Screen.width
        * CurrentData.Data.currentSettings.graphicsOption) / 8.0f);
    int threadGroupsY = Mathf.CeilToInt((int)(Screen.height
        * CurrentData.Data.currentSettings.graphicsOption) / 8.0f);
    //запускаем расчет шейдера
    raytracingShader.Dispatch(0, threadGroupsX, threadGroupsY, 1);
    //копируем полученную текстуру в текстуру, которую
    //отображает камера на экран
    Graphics.Blit(target, destination);
}

//Базовый метод Unity который вызывается после прорисовки
//для постобработки изображения на экране.
//параметры: исходное ихображение и изображение для изменений
private void OnRenderImage(RenderTexture source, RenderTexture destination)
{
    //если надо перестроить буферы или это режим создания
    if (meshObjectsNeedRebuilding || creatingMode)
    {
        meshObjectsNeedRebuilding = false; //без повторной перестройки
        RebuildAllObjectBuffers(); //перестроить все буферы
        SetAllShaderParameters(); //уставноить все параметры
    }
    else
    {
        RebuildMeshObjectBuffer();
        //перстроить только матрицу преобразований
        SetUpdatableShaderParameters();
        //обновить данные о матрице в шейдере
    }
    //получаем изображение используя  шейдер
    Render(destination);
    //Если еще не построили подсказку, но строим ее
    if (spriteHint == null)
    {
        CreateSpriteHint();
    }
}

//Метод который вырезает из результирующей текстуры
//кусок с тенью для подсказки
private void CreateSpriteHint()
{
```

```
        //задаем размеры текстуры
        int width = Screen.width, height = Screen.height;
        int right = (int)(width * 0.425), left = (int)(width * 0.225);
        int down = (int)(height * 0.3), up = (int)(height * 0.7);
        width = right - left;
        height = up - down;
        //создаем текстуру
        Texture2D tex = new Texture2D(width, height,
            TextureFormat.RGB24, false);
        //читаем пискели с экрана в указанной области
        tex.ReadPixels(new Rect(left, down, width, height), 0, 0);
        //применяем
        tex.Apply();
        //создаем спрайт
        spriteHint = Sprite.Create(tex, new Rect(0.0f, 0.0f, tex.width,
            tex.height), new Vector2(0.5f, 0.5f));
    }
}
```

RayTracingObject.cs

```
using UnityEngine;
using System.IO;

//Класс который сопутствует всем объектам, которые участуют в рейтрейсинге
//При добавлени такого обхекта на сцену, заставляет зарегестрировать себя
//и передает все свои параметры классу RayTracer
//Расширяет базовые параметры обхекта сцены прозрачностью и цветами полигонов
[RequireComponent(typeof(MeshRenderer))]
[RequireComponent(typeof(MeshFilter))]
public class RayTracingObject : MonoBehaviour
{
    [Range(0.0f, 1.0f)]
    public float transparency = 0.0f;//прозрачность объекта
    public int[] facesColor;//цвета граней объекта

    //Функция вызывается после включения обхекта на сцене
    private void OnEnable()
    {
        //определение размера массива цветов
        uint indexCount =
        GetComponent<MeshFilter>().sharedMesh.GetIndexCount(0);
        //предварительная инициализация всех цветов белым цветом
        facesColor = new int[indexCount];
        for (uint i = 0; i < indexCount; i++)
        {
            facesColor[i] = 255;
        }
        //регистрация обхекта среди обхектов рейтрейсинга
        Raytracer.RegisterObject(this);
    }

    //Функция срабатывает когда объект становится неактивынм
    private void OnDisable()
    {
        //освобождение памяти выделенной под обхект рейстрейсинга
        Raytracer.UnregisterObject(this);
    }
}
```

RayTracerShader.compute

```
/*This shader implements the calculation of the color of the screen pixel
for which it was called using a simplified and optimized ray tracing algorithm.
In the shader code, it is not possible to encode the Russian layout,
so the comments will be in English.
*/

// Each #kernel tells which function to compile; you can have many kernels
#pragma kernel CSMain

//Create a RenderTexture and set it with cs.SetTexture
RWTexture2D<float4> Result;

//Almost all the fields described below are initialized
//from the main code when the shader is called

//CAMERA PARAMETERS
float4x4 _CameraToWorld;
float4x4 _CameraInverseProjection;

//SKYBOX TEXTURE
Texture2D<float4> _SkyboxTexture;
SamplerState sampler_SkyboxTexture;

//ROOM TEXTURE
Texture2D<float4> _RoomMaskMap;
SamplerState sampler_RoomMaskMap;
Texture2D<float4> _RoomColorTexture1;
SamplerState sampler_RoomColorTexture1;
Texture2D<float4> _RoomBumpMap1;
SamplerState sampler_RoomBumpMap1;
float _BumpScale1 = 1.0f;
float2 _TexTiling1 = 1.0f;
Texture2D<float4> _RoomColorTexture2;
SamplerState sampler_RoomColorTexture2;
Texture2D<float4> _RoomBumpMap2;
SamplerState sampler_RoomBumpMap2;
float _BumpScale2 = 1.0f;
float2 _TexTiling2 = 1.0f;

//CONSTANTS
static const float PI = 3.14159265f;
static const float EPSILON = 1e-8;

//NUMBER OF ITERATION
int _TransparencyCount;

//LIGHTs PARAMETERS
float4 _SpotLightDirection;
float4 _SpotLightPosition;
float _LightRange;
float4 _PointLightPosition;


//STRUCT OF MESHRT
//(identical structure in the main code)
struct MeshRT
{
    float4x4 localToWorldMatrix;
    int indices_offset;
    int indices_count;
    float transparency;
};
```

```
//OBJECTS MESHES PARAMETERS
StructuredBuffer<MeshRT> _MeshObjects;
StructuredBuffer<float3> _Vertices;
StructuredBuffer<float2> _VerticesUV;
StructuredBuffer<int> _Indices;
StructuredBuffer<uint> _Colors;
float _selectedFace = -10;


//RAY STRUCTURER
//(with direction and energy)
struct Ray
{
    float3 origin;
    float3 direction;
    float3 energy;
};

//Constructor of the ray structure based on two points.
Ray CreateRay (float3 origin, float3 direction)
{
    Ray ray;
    ray.origin = origin;
    ray.direction = direction;
    ray.energy = 1.0f;
    return ray;
}

//Constructor of the ray coming from the camera
//and passing through the specified pixel of the screen.
Ray CreateCameraRay (float2 uv)
{
    float3 origin = mul(_CameraToWorld,
        float4(0.0f, 0.0f, 0.0f, 1.0f)).xyz;
    float3 direction = mul(_CameraInverseProjection,
        float4(uv, 0.0f, 1.0f)).xyz;
    direction=mul(_CameraToWorld,float4(direction,0.0f)).xyz;
    direction=normalize(direction);
    return CreateRay(origin,direction);
}

//HIT OF RAY
//(used to set the parameters of the hit of the ray with the object)
struct RayHit
{
    float3 position;
    float distance;
    float3 normal;
    float specular;
    float3 albedo;
    float transparency;
};

//RayHit structure constructor with filling in fields with default values
RayHit CreateRayHit()
{
    RayHit hit;
    hit.position = 0.0f;
    hit.distance = 1.#INF;
    hit.normal = 0.0f;
    hit.specular = 0.0f;
    hit.albedo = 0.0f;
    hit.transparency = 0.0f;
```

```
    return hit;
}

//INTERSECTS
//Determination of the intersection of
//a ray and a triangle by variation of the Moller-Trumbore algorithm.
bool IntersectTriangle_MT97(Ray ray, float3 vert0, float3 vert1, float3 vert2,
      inout float t, inout float u, inout float v)
{
      //find vectors for two edges sharing vert0
      float3 edge1 = vert1 - vert0;
      float3 edge2 = vert2 - vert0;
      //begin calculating determinant
      float3 pvec = cross(ray.direction, edge2);
      //if determinant is near zero, ray lies in plane of triangle
      float det = dot(edge1, pvec);
      // ray is parallel to the plane
      if (det < EPSILON && det > -EPSILON)
    {
        //no intersection
        return false;
    }
    else
    {
        float inv_det = 1.0f / det;
        // calculate distance from vert0 to ray origin
        float3 tvec = ray.origin - vert0;
        // calculate U parameter and test bounds
        u = dot(tvec, pvec) * inv_det;
        if (u < 0.0f || u > 1.0f)
        {
            //no intersection
            return false;
        }
        else
        {
              // prepare to test V parameter
              float3 qvec = cross(tvec, edge1);
              // calculate V parameter and test bounds
              v = dot(ray.direction, qvec) * inv_det;
              if (v < 0.0 || u + v > 1.0f)
            {
                //no intersection
                    return false;
            }
            else
            {
                  // calculate t, ray intersects triangle
                  t = dot(edge2, qvec) * inv_det;
                  return true;
            }
        }
    }
}

//Checking all triangles of the mesh of a transparent object
//for an intersection with a ray and determining the nearest hit.
void IntersectMeshObject(Ray ray, inout RayHit bestHit, MeshRT meshObject)
{
    //We determine the reading locations in the buffers
    float offset = meshObject.indices_offset;
    float count = offset + meshObject.indices_count;
    //we go through all the triangles
```

```
    for (float i = offset; i < count; i += 3)
    {//coordinates with account for the rotation and displacement of the object
        float3 v0 = (mul(meshObject.localToWorldMatrix,
            float4(_Vertices[_Indices[i]], 1))).xyz;
        float3 v1 = (mul(meshObject.localToWorldMatrix,
            float4(_Vertices[_Indices[i + 1]], 1))).xyz;
        float3 v2 = (mul(meshObject.localToWorldMatrix,
            float4(_Vertices[_Indices[i + 2]], 1))).xyz;
        //the distance to the intersection
        //and the barycentric coordinates of the intersection
        float t=0.0f, u=0.0f, v=0.0f;
        if (IntersectTriangle_MT97(ray, v0, v1, v2, t, u, v))
        {
            //if the intersection is closer than before
            if (t > 0 && t < bestHit.distance)
            {
                //setting the intersection parameters
                bestHit.normal = normalize(cross(v1 - v0, v2 - v0));
                bestHit.distance = t;
                bestHit.position = ray.origin + t * ray.direction;
                bestHit.transparency = meshObject.transparency;
                bestHit.albedo = float3(_Colors[i] * 0.0035f,
                  _Colors[i+1] * 0.0035f, _Colors[i+2] * 0.0035f);
                //The color of the edges is black or white if the
                //face is highlighted
                if (i - offset == _selectedFace)
                {
                    if (u < 0.03f || v < 0.03f || (1 - u - v) < 0.03f)
                    {
                        bestHit.albedo = float3(1.0f, 1.0f, 1.0f);
                    }
                }
                else
                {
                    if (u < 0.02f || v < 0.02f || (1 - u - v) < 0.02f)
                    {
                        bestHit.albedo = 0.0f;
                        bestHit.transparency = 0.0f;
                    }
                }
            }
        }
    }
}

//Checking all triangles of the mesh of the room for an intersection
//with the beam and determining the intersection closest to the
//source of the beam.
//Additionally, the color and normal at the collision point are determined
//from the corresponding textures of the room.
void IntersectMeshRoom(Ray ray, inout RayHit bestHit, MeshRT meshObject)
{
    //We determine the reading locations in the buffers
    float offset = meshObject.indices_offset;
    float count = offset + meshObject.indices_count;
    uint j = 0;
    //we go through all the triangles
    for (float i = offset; i < count; i += 3, ++j)
    {//coordinates with account for the rotation and displacement of the object
        float3 v0 = (mul(meshObject.localToWorldMatrix,
            float4(_Vertices[_Indices[i]], 1))).xyz;
        float3 v1 = (mul(meshObject.localToWorldMatrix,
            float4(_Vertices[_Indices[i + 1]], 1))).xyz;
```

```
            float3 v2 = (mul(meshObject.localToWorldMatrix,
                float4(_Vertices[_Indices[i + 2]], 1))).xyz;
            //the distance to the intersection
            //and the barycentric coordinates of the intersection
            float t=0.0f, u=0.0f, v=0.0f;
            if (IntersectTriangle_MT97(ray, v0, v1, v2, t, u, v))
            {
                //if the intersection is closer than before
                if (t > 0 && t < bestHit.distance)
                {
                    //getting the texture coordinates
                    float2 uv = _VerticesUV[_Indices[i+1]] * u
                      + _VerticesUV[_Indices[i+2]] * v
                      + _VerticesUV[_Indices[i]]*(1-u-v);
                    //mask pixel color
                    float3 maskColor = _RoomMaskMap.SampleLevel(
                      sampler_RoomMaskMap, uv, 0).xyz;
                    //given the mask, textures and texture coordinates,
                    //we find the color and normal at this point
                    float3 textureColor = _RoomColorTexture1.SampleLevel(
                      sampler_RoomColorTexture1,
                      uv * _TexTiling1, 0).xyz * maskColor.x
                      + _RoomColorTexture2.SampleLevel(sampler_RoomColorTexture2,
                      uv * _TexTiling2, 0).xyz * maskColor.y;
                    float3 normal = float3(_RoomBumpMap1.SampleLevel(
                      sampler_RoomBumpMap1, uv * _TexTiling1, 0).wy * maskColor.x
                      + _RoomBumpMap2.SampleLevel(sampler_RoomBumpMap2,
                      uv * _TexTiling2, 0).wy * maskColor.y, 0.0f);
                    normal.xy = (normal.xy * 2.0f - 1.0f) * (_BumpScale1
                      * maskColor.x + _BumpScale2 * maskColor.y);
                    normal.z = sqrt(1 - saturate(dot(normal.xy, normal.xy)));
                    normal = normal.xzy;
                    //initialize all rayhit fields
                    bestHit.normal = normalize((mul(meshObject.localToWorldMatrix,
                        float4(normal, 1.0f))).xyz);
                    bestHit.albedo = textureColor * 255;
                    bestHit.distance = t;
                    bestHit.position = ray.origin + t * ray.direction;
                    bestHit.specular = 0.65f;
                    bestHit.transparency = meshObject.transparency;
                }
            }
        }
    }


    //TRACE
    //Collision detection by alternately calling the
    //IntersectMeshObject functions, and then IntersectMeshRoom.
    RayHit Trace(Ray ray)
    {
        RayHit bestHit = CreateRayHit();
        IntersectMeshObject(ray, bestHit, _MeshObjects[1]);
        IntersectMeshRoom(ray, bestHit, _MeshObjects[0]);

        return bestHit;
    }


    //Determination of the illumination of the collision site,
    //where two light sources are taken into account: point and directional.
    //The dependence of illumination on distance, angle (for directional light)
```

```
//and intensity of the source is analyzed on the basis of the Inverse Square
Law.
float3 Lighting(Ray ray, RayHit hit)
{
    //SPOT LIGHT
    //vector from the light source to the hit
    float lightDist = distance(hit.position, _SpotLightPosition.xyz);
    float3 lightVector = (hit.position + hit.normal * 0.001f)
      - _SpotLightPosition.xyz;
    float light = _SpotLightDirection.w / (4 * PI * lightDist * lightDist);
    //projection of vectors onto the camera plane
    float2 lightVectorXY = lightVector.xy;
    float2 spotLightVectorXY = _SpotLightDirection.xy;
    //projection of vectors onto the camera plane
    float beta = acos(dot(normalize(lightVectorXY),
      normalize(spotLightVectorXY)));
    float alfa = acos(dot(normalize(lightVector),
      normalize(_SpotLightDirection.xyz)));
    //the law of inverse squares
    light *= _SpotLightDirection.w / (4 * PI * max(0.001f, (alfa * alfa
      - _SpotLightPosition.w * _SpotLightPosition.w)));
    //POINT LIGHT
    lightDist = distance(hit.position, _PointLightPosition.xyz);
    //the law of inverse squares
    light += _PointLightPosition.w / (4 * PI * max(1, lightDist * lightDist
      - _LightRange * _LightRange));
    //cone trace of a directional source
    light *= _SpotLightDirection.w / (4 * PI * max(0.1f, (beta
      - _SpotLightPosition.w)));
    //forming a reuslatt
    return saturate(dot(hit.normal, _SpotLightDirection.xyz) * -1)
        * light
        * hit.albedo;
}

//Determining the collision color and updating the energy
//and direction of the beam at some iteration of the shadow
//ray calculation for a transparent object.
float3 TransparencyShadowRayShade(inout Ray ray, RayHit hit)
{
    //Refresh ray to transparency ray
    ray.origin = hit.position + ray.direction * 0.001f;
    //the contribution coefficient of the new color
    float kefTE = hit.transparency + 0.5f;
    ray.energy = ray.energy * kefTE;

    return _SpotLightDirection.w
        * (hit.albedo);
}

//Determining the collision color and updating the energy
//and direction of the beam at some iteration of the beam
//calculation for a transparent and opaque  object.
float3 TransparencyShade(inout Ray ray, RayHit hit)
{
    //Refresh ray to transparency ray
    ray.origin = hit.position + ray.direction * 0.001f;
    float cos_a = dot(ray.direction, _SpotLightDirection.xyz);
    float kefTE = hit.transparency * saturate(cos_a);
    ray.energy = ray.energy * kefTE;

    //if object not transparency
    if (hit.transparency == 0.0f)
```

```
    {
        //simple onlu lighting
        return Lighting(ray, hit);
    }
    else
    {
        return _SpotLightDirection.w
            * hit.albedo;
    }
}

//Determining the collision color for a opaque  object,
//if there was no collision, then the color of the background texture.
float3 RoomShade(inout Ray ray, RayHit hit)
{
    if(hit.distance<1.#INF)
    //IF HIT WERE
    {
        //Refresh energy ray
        ray.energy = ray.energy * hit.transparency;

        return Lighting(ray, hit);
    }
    else
    //IF HIT WERE NOT
    {
        //we determine the color from the texture of the sky
        ray.energy = 0.0f;
        float theta = acos(ray.direction.y)/-PI;
        float phi = atan2(ray.direction.x,-ray.direction.z)/-PI*0.5f;

        return _SkyboxTexture.SampleLevel(sampler_SkyboxTexture,
            float2(phi,theta),0).xyz;
    }
}

//Determining the collision color for an opaque object,
//as well as calculating the color shadow.
//To calculate the shadow, the Transparency Shadow RayShade function
//is cyclically called, the results are summed up in a special way
//and the shadow color is obtained.
float3 ShadowShade(inout Ray ray, RayHit hit)
{
    float3 result = 0.0f;
    //Refresh energy ray
    ray.energy = ray.energy * hit.transparency;
    //Create first shadow ray
    Ray shadowRay = CreateRay(hit.position + hit.normal * 0.1f,
      _SpotLightPosition.xyz);
    RayHit shadowHit = CreateRayHit();
    IntersectMeshObject(shadowRay, shadowHit, _MeshObjects[1]);
    //if there is a shadow
    if (shadowHit.distance != 1.#INF)
    {
        //changing the result
        result += Lighting(ray, hit) * shadowRay.energy
            * TransparencyShadowRayShade(shadowRay, shadowHit);
        //iterate until the energy of the beam runs
        //out or the allowable number of iterations runs
        //out or the beam flies into the void
        for (float i = 0; i < _TransparencyCount; ++i)
        {
            shadowHit = CreateRayHit();
```

```
            IntersectMeshObject(shadowRay, shadowHit, _MeshObjects[1]);

            //if there was hit, calculate shadow of hit object
            if (shadowHit.distance == 1.#INF)
            {
                break;
            }
            //changing the result
            result += Lighting(ray, hit) * shadowRay.energy
                  * TransparencyShadowRayShade(shadowRay, shadowHit);
            result *= shadowHit.transparency + 0.2f;
            //total dimming at each step

            //IF ANY ENERGY == 0
            if (!any(shadowRay.energy))
            {
                break;
            }
        }
    }
    else
    {
        //original color of object
        result += Lighting(ray, hit);
    }
    return result;
}

//KERNEL
//The main shader function that is executed when assigning a
//shader to a rendering texture.
//The method is executed in parallel and separately for each
//pixel of the texture.
[numthreads(8,8,1)]
void CSMain (uint3 id : SV_DispatchThreadID)
{
    //determining the direction of the ray
    uint width,height;
    Result.GetDimensions(width,height);
    float2 uv = float2((id.xy + float2(0.5f, 0.5f)) / float2(width, height)
      * 2.0f - 1.0f);
    //initializing the ray
    Ray ray = CreateCameraRay(uv);
    float3 result = 0.0f;
    //screen areas
    if (uv.y > -0.9f && uv.y < 0.6f && uv.x > 0.0f && uv.x < 0.85)
    {
        //Main transparency object and room
        for (float i = 0; i < _TransparencyCount; ++i)
        {
            RayHit hit = Trace(ray);
            result += ray.energy * TransparencyShade(ray, hit);
            //IF ANY ENERGY == 0
            if (!any(ray.energy))
            {
                break;
            }
        }
    }
    else
    {
        //Only room and shadow
        RayHit hit = CreateRayHit();
```

```
        IntersectMeshRoom(ray, hit, _MeshObjects[0]);

        if (uv.y > -0.5f && uv.y < 0.5f && uv.x > -0.6f && uv.x < -0.1)
        {
            //With colore shadow
            result = ShadowShade(ray, hit);
        }
        else
        {
            //Only room and basic shadow
            result = ray.energy * RoomShade(ray, hit);
        }
    }
    Result[id.xy] = float4(result,1.0f);
}
```

CustomLevelCreator.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

//Класс предназначен для управения игровым миром и
//элементами пользовательского интерфеса на экране режимо создания уровня
public class CustomLevelCreator : MonoBehaviour
{
    //Режим разработчика (создание и сохранение базовых уровней в папку
    //ресурсов игры)
    public bool isDeveloperMode = false;
    public InputField levelNameInputField;//поле вода названия уровня
    public Dropdown roomDropdown;//выпадаюий список материалов комнаты
    public Dropdown figureDropdown;//выпадающиъ список форм фигуры
    public Dropdown facesDropdown; //выпаающий список граней фигуры
    public Dropdown colorDropdown; //выпадающий список цветов грани
    public GameObject figure; //Фигура
    public GameObject room; //Комната
    public GameObject mainCamera; //камера, которая занимается рендерингом
    public GameObject loadingScreen; //Окно загрузки уровня
    private int indexSelectedFace = -1; //индекс выбранной грани
    private SaveData.Level newLevel = new SaveData.Level(); //новый уровень

    //Метод сарта, срабатывает непосредственно перед первым обновлением фрейма
    void Start()
    {
        loadingScreen.SetActive(true); //вкл экран загрузки
        InitializingUI(); //установить данные нового уровня как стартовые
        loadingScreen.SetActive(false); //выкл экран загрузки
    }

    //Метод сохраняет уровень с текущими настройками в файл
    public bool SaveCustomLevel()
    {
        UpdateLevelSettings(); //Обновить параметры уровня
        //Если им уровня не пустое
        if (newLevel.levelName.Length != 0)
        {
            //если это режим разраотчика
            if (isDeveloperMode)
            {
#if UNITY_EDITOR
                //сохранить уровенб в ввиде ресурса базового уровня в папку
```

```
                //с ресурсами игры
                SaveManager.SaveTextAsset<SaveData.Level>(newLevel,
                  "/Levels/Resources/BasicLevels/", newLevel.levelName);
#else
                return false; //если это не среда разработки, тогда ошибка
#endif
            }
            //если это обычный режим создания
            else
            {
                //сохранить уровень в виде файла расширения json в
                //файловой системе платформы
                SaveManager.SaveJson<SaveData.Level>(newLevel,
                  newLevel.levelName);
            }
            return true; //успех
        }
        else
        {
            return false; //неуспех
        }
    }

    //Метод который проиводит стартовую инициализацию настроек уровня
    //и применяет их к игровому миру
    public void InitializingUI()
    {
        newLevel.levelName = levelNameInputField.text;
        //название уровня из поля ввода текста
        newLevel.roomMaterialName = roomDropdown.captionText.text;
        //материал комнаты
        newLevel.finishPosition = figure.transform.rotation; //финишна позиция
        newLevel.objectMeshName = figureDropdown.captionText.text;
        //меш главной фигуры
        SetOptionsColorDropdown();
        //инициализируем значения выпадающего списка цветов

        ApplyLevelSettings(); //применить текущие данные уровня к сцене
    }

    //Метод обновляет параметры уровня и применяет их к игровому миру
    public void UpdateLevelSettings()
    {
        newLevel.levelName = levelNameInputField.text; //название уровня
        newLevel.roomMaterialName = roomDropdown.captionText.text;
        //материал комнаты
        newLevel.finishPosition = figure.transform.rotation; //финишна позиция
        //если была выбрана друга форма фигуры
        if (newLevel.objectMeshName != figureDropdown.captionText.text)
        {
            newLevel.objectMeshName = figureDropdown.captionText.text;
            //применить к настройкам уровня
            facesDropdown.value = 0;
            //установить значение выбранной грани фигуры на 0
        }
        //если была выбрана другая грань фигуры
        if (facesDropdown.value != indexSelectedFace)
        {
            indexSelectedFace = facesDropdown.value;
            //применить к настройкам уровня
            //вспомнить какой цвет у данной грани для обновления
            //выпадающего списка цветов
            RememberColorDropdown();
```

```
    }
    UpdateFaceColor();
    //обновить цвет грани в соответствии с выпадающим списком цветов
    ApplyLevelSettings(); //применить текущие данные уровня к сцене
}

//Метод применяет текущие параетры уровня к игровому миру
private void ApplyLevelSettings()
{
    CurrentData.Data.currentLevel = newLevel; //Теущий уровень равен новому
    CurrentData.Data.LoadMeshAndMaterial();
    //Прочитать данные указанных названий материалов и мешей

    room.GetComponent<MeshRenderer>().material =
        CurrentData.Data.roomMaterial; //Применить материал комнаты

    //инициализируем массив цветов граней фигуры
    figure.GetComponent<RayTracingObject>().facesColor =
        new int[CurrentData.Data.currentLevel.objectFacesColor.Length
        * 3];
    //применяем цвета параметров уровня к фигуре
    for (int i = 0;
        i < CurrentData.Data.currentLevel.objectFacesColor.Length; i++)
    {
        figure.GetComponent<RayTracingObject>().facesColor[3 * i] =
              (int)(255
              * CurrentData.Data.currentLevel.objectFacesColor[i].r);
        figure.GetComponent<RayTracingObject>().facesColor[3 * i + 1] =
              (int)(255
              * CurrentData.Data.currentLevel.objectFacesColor[i].g);
        figure.GetComponent<RayTracingObject>().facesColor[3 * i + 2] =
              (int)(255
              * CurrentData.Data.currentLevel.objectFacesColor[i].b);
    }
    //если меш фигуры не соответствует мешу уровня
    if (figure.GetComponent<MeshFilter>().sharedMesh !=
        CurrentData.Data.objectMesh)
    {
        //обновить меш фигуры как в уровне
        figure.GetComponent<MeshFilter>().sharedMesh =
              CurrentData.Data.objectMesh;
        UpdateFacesDropdown();
        //обновить выпадающий список граней под новый меш
    }
}

//Метод обновляет выпадающий список граней в соответствии выбранному
//мешу фигуры
public void UpdateFacesDropdown()
{
    uint sizeList =
        figure.GetComponent<MeshFilter>().sharedMesh.GetIndexCount(0) / 3;
    //количество граней меша
    List<Dropdown.OptionData> newFacesList =
        new List<Dropdown.OptionData>((int)sizeList);
    //Список опций выпадающего списка
    for (int i = 1; i <= sizeList; i++) //проходимся по всему списку
    {
        newFacesList.Add(new Dropdown.OptionData(i.ToString()));
        //добавляем инфу о грани фигуры в список
    }
    facesDropdown.options = newFacesList;
    //Применяем новые опции к выпадающему списку
```

```
        indexSelectedFace = facesDropdown.value;
        //текущее значение списка - выбранная грань
        RememberColorDropdown();
        //обновляем выбранное значение выпадающего списка цветов к новой грани
    }

    //Метод определяет какой цвет необходимо установить для выпадающего
    //списка цветов граней
    public void RememberColorDropdown()
    {
        //Указываем ренедерингу какую грань фигуры подсветить
        mainCamera.GetComponent<Raytracer>().selectedFace =
            indexSelectedFace * 3;

        int i = 0; //инжекс выбранной грани в пределах их количества
        if (indexSelectedFace < newLevel.objectFacesColor.Length)
        {
            //ищем совпадение цвета выбранной грани и цвета в выпадающем списке
            for (i = CurrentData.COLORS.Length - 1; i >= 0
                  && !CurrentData.COLORS[i].Equals(
                  newLevel.objectFacesColor[indexSelectedFace]); i--)
            { }
        }
        colorDropdown.value = i; //выставляем найденное значение цвета
    }

    //Метод обновляет цвет грани фигуры в соответствии с выборов
    //выпадающего списка цветов
    public void UpdateFaceColor()
    {
        newLevel.objectFacesColor[indexSelectedFace] =
            CurrentData.COLORS[colorDropdown.value];
    }

    //Метод устанавливает значения выпадающего списка цветов
    public void SetOptionsColorDropdown()
    {
        int sizeList = CurrentData.COLORS.Length;
        //количество дискретных цветов
        //Инициализируем список опций
        List<Dropdown.OptionData> ColorList =
            new List<Dropdown.OptionData>(sizeList);
        for (int i = 1; i < sizeList; i++)
        {
            //Создаем картинку соответствующего цвета и назначаем
            Texture2D texture = new Texture2D(1, 1);
            texture.SetPixel(0, 0, CurrentData.COLORS[i - 1]);
            texture.Apply();
            Sprite image = Sprite.Create(texture, new Rect(0.0f, 0.0f,
                  texture.width, texture.height), new Vector2(0.0f, 0.0f));
            ColorList.Add(new Dropdown.OptionData(i.ToString(), image));
        }
        //применяем опции к выпадающему списку
        colorDropdown.options = ColorList;
    }
}
```

LevelControllerScript.cs

```
using UnityEngine;
using System.IO;
using UnityEngine.UI;
```

```
using UnityEngine.SceneManagement;
using System;

//Класс призван управлять всей логикой и обхектами игрового мира
//во время прохождения уровня
public class LevelControllerScript : MonoBehaviour
{
    public GameObject figure; //Фигуры
    public GameObject room; //комната
    public GameObject mainCamera; //главная камера
    public GameObject hint; //Подсказка
    public GameObject loadingScreen; //окно загружки уровня
    public GameObject timeCounter; //Счетчик
    public GameObject finishScreen; //Окно победы в уровне
    public Text result; //Поле результата прохождения уровня
    public Slider LevelProgressSlider; //Слайдер близости к финишу
    public Image LevelProgressSliderFill;
    //Изображение сладера блихости к финишу
    [Range(5, 30)]
    public int acceptableDelta = 10;
    //Расстояние до финиша в углах, когда включается автоматическая доводка
    [Range(0.0f, 1.0f)]
    public float smooth = 1.0f; //плавность автоматической доводки
    public Quaternion finishPosition; //финишная позиция для данного уровня
    private bool prepared = false; //Подготовитьельный этап уровня окончен
    private bool finished = false; //Уровень пройден

    //Метод сарта, срабатывает непосредственно перед первым обновлением фрейма
    private void Start()
    {
        loadingScreen.SetActive(true); //Активириуем окно загрузки
        finishScreen.SetActive(false); //Деактивируем окно финиша
        finishPosition = CurrentData.Data.currentLevel.finishPosition;
        //считываем финишную позицию

        //Настраиваем фигуры в соответствии параметрами уровня
        figure.GetComponent<MeshFilter>().sharedMesh =
            CurrentData.Data.objectMesh;
        figure.GetComponent<RayTracingObject>().facesColor =
           new int[CurrentData.Data.currentLevel.objectFacesColor.Length * 3];
        for (int i = 0;
            i < CurrentData.Data.currentLevel.objectFacesColor.Length; i++)
        {
            figure.GetComponent<RayTracingObject>().facesColor[3 * i] =
                  (int)(255
                  * CurrentData.Data.currentLevel.objectFacesColor[i].r);
            figure.GetComponent<RayTracingObject>().facesColor[3 * i + 1] =
                  (int)(255
                  * CurrentData.Data.currentLevel.objectFacesColor[i].g);
            figure.GetComponent<RayTracingObject>().facesColor[3 * i + 2] =
                  (int)(255
                  * CurrentData.Data.currentLevel.objectFacesColor[i].b);
        }
        //Настраиваем комнату в соответствии с параметрами уровня
        room.GetComponent<MeshRenderer>().material =
            CurrentData.Data.roomMaterial;
        //Крутим фигуру в финишное положение, чтобы создать
        //подсказу пользователю
        figure.transform.rotation = finishPosition;
        //Выставляем время затраченное на прохождение этого уровня в 0:0:00
        CurrentData.Data.currentLevelTime = 0.0f;
    }
```

```
//Базовый метод, срабатывающий через одинаковые малые
//опредленные промежутки времени
private void FixedUpdate()
{
    //Подготовительный этап уровня, создание подсказки
    if (!prepared && !finished
        && mainCamera.GetComponent<Raytracer>().spriteHint != null)
    {
        //Получаем спрайт подсказки от класса управляющего рендерингом
        hint.GetComponent<Image>().sprite =
              mainCamera.GetComponent<Raytracer>().spriteHint;
        //вращаем фигуру в начальное положение
        figure.transform.rotation = Quaternion.Euler(0.0f, 0.0f, 0.0f);
        prepared = true; //Режим подготовки уровня окончен
        loadingScreen.SetActive(false); //Деактивация окна загрузки уровня
        timeCounter.GetComponent<TimeCounterScript>().StartTimer();
        //Запускаме секундомер
    }

    //регулирование слайдера, показывающего близость к финишному
    //положению фигуры
    LevelProgressSlider.value = 1 - (Quaternion.Angle(finishPosition,
        figure.transform.rotation) / 180);
    LevelProgressSliderFill.color = Color.Lerp(
        new Color(0.68f, 0.84f, 1.0f, 1.0f),
        new Color(1.0f, 0.67f, 0.4f, 1.0f),
        LevelProgressSlider.value);//интерполируем цвет
    //Если подготовка заверщена и угол между финишным положением и
    //текущим положением меньше 1,
    //тогда финиш
    if (prepared && !finished && Quaternion.Angle(finishPosition,
        figure.transform.rotation) < 1)
    {
        timeCounter.GetComponent<TimeCounterScript>().isStopped = true;
        //Останавливаем секундомер
        //запоминаем время затраченное на прохождение этого уровня
        CurrentData.Data.currentLevelTime =
              timeCounter.GetComponent<TimeCounterScript>().timer;
        //Если прогресс учитывается и это базовый уровень
        if (CurrentData.Data.isProgress && CurrentData.Data.isBasicLevel)
        {
            //тогда обновляем и сохраняем данные прогресса
            CurrentData.Data.UpdateAndSavePlayerProgress(
              int.Parse(CurrentData.Data.currentLevel.levelName.Split(
              ' ')[1]) - 1);
        }
        CurrentData.Data.isSet = false;
        GetComponent<AudioSource>().Stop();
        //останавливаем фоновую музыку в игре
        //Отображаем время затраченное на прохождение на экране
        result.text +=
              timeCounter.GetComponent<TimeCounterScript>().ToString(
              CurrentData.Data.currentLevelTime);
        finishScreen.SetActive(true);//активируем окно финиша
        finished = true;//устанавливаем флаг окончания уровня
    }
    //Если подготовка завершена и угол меджу винишным и текущим
    //положением меньше допустимой дельты,
    //тогда необходимо автоматически докрутить фигуру до
    //финишного положения
    if (prepared && !finished && Quaternion.Angle(finishPosition,
        figure.transform.rotation) <= acceptableDelta)
    {
```

```
            //вращаем интерполяцией из текущего опложения к инишному
            //с определенной скоростью
            figure.transform.rotation = Quaternion.RotateTowards(
                  figure.transform.rotation, finishPosition, smooth);
            //Отключаем возможность вращения фигуры, чтобы пользователь
            //не мешал
            GameObject.Find(
                  "RotationBase").GetComponent<RotationScript>().isActive =
                  false;
        }
    }
}
```

Level.cs

```
using UnityEngine;

//Пространство имен, в котором находится все для сохранений
namespace SaveData
{
    //Сериализуемый класс, описывающий параметры уровня для его сохранения
    //в файл
    [System.Serializable]
    public class Level
    {
        public string levelName; //Название уровня и файла
        public string roomMaterialName = "RoomMat1";
        //название файла материала комнаты
        public Quaternion finishPosition = Quaternion.Euler(new Vector3(30.0f,
            30.0f, 30.0f)); //финишное положение фигуры
        public string objectMeshName = "CUBE"; //название файла меша фигуры
        public Color[] objectFacesColor = { }; //массив цветов граней фигуры
    }
}
```