Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
"___"_____2023 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

Разработка игры на Unity «Lost Soul»

Руководитель _____ _____ старший преподаватель Т.С. Титовская
подпись *дата* *должность, ученая степень*

Выпускник _____ _____ Д.В. Сбитнев
подпись *дата*

Нормоконтролёр _____ _____ Т.С. Титовская
подпись *дата*

Красноярск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Игры стали неотъемлемой частью жизни множества людей в современном обществе в различных странах и на всевозможных платформах. Для одних пользователей это способ развлечения в свободное время, для других — способ получения стабильного заработка, третьи преследуют цель получения мирового признания путем участия в турнирах.

В игровой индустрии существует великое множество жанров и их сочетаний. Чтобы погрузиться в мир игр от пользователя не требуется ничего, кроме подходящего устройства, наличия игры и желания играть.

Общий объём игрового рынка в 2022-м составил $184,4 миллиарда — это на 4,3 % меньше, чем в прошлом году. Распределение следующее [1]:

– Мобильные игры (50 %) — $92,2 млрд;

– Консольные игры (28 %) — $51,8 млрд;

– Полноценные ПК-игры для цифрового издания и розницы (21 %) — $38,2 млрд;

– Браузерные ПК-игры (1 %) — $2,3 млрд.

Мобильные игры заняли половину от общего объема игрового рынка в связи с их доступностью. Наиболее популярные, прибыльные и обсуждаемые игры приходятся на крупные игровые студии, что неудивительно, так как это более качественный продукт, данное утверждение справедливо для всех платформ.

Для того, чтобы удовлетворить потребности игроков на различных игровых платформах необходимо выпускать игры как на мобильные устройства, так и на ПК.

Для реализации данной цели следует выбрать наиболее популярный движок, который обеспечит кроссплатформенность. Данным движком является Unity.

В выпускной квалификационной работе планируется выполнить следующие задачи:

– обзор игрового движка Unity;

– анализ архитектур для создания игр;

– обзор клиент-серверных решений;

– разработка концепта игры;

– составление задания на проектирование и разработку;

– проектирование игры;

– отрисовка дизайна;

– реализация игры;

– адаптация игры для телефонов;

– тестирование и отладка продукта.

1 Анализ предметной области и составление задания на проектирование и реализацию

1.1 Игровой движок Unity

Unity — это кроссплатформенный движок разработанный Unity Technologies. Движок поддерживает различные настольные, мобильные, консольные приложения и приложения виртуальной реальности. На сегодняшний день Unity позволяет обеспечить максимальных охват аудитории благодаря поддержке более 25 платформ и технологий. Движок может использоваться для создания трехмерных и двумерных игр, а также для симуляций. Помимо игр, Unity также используется в архитектуре, машиностроении, строительстве, кино.

Unity используется не только среди начинающих программистов и инди-студий. Среди крупных проектов есть студии, которые разрабатывали игры на Unity. Например: Genshin Impact, Hearthstone, Outlast, Cuphead, Pokemon Go и другие.

Если разработчик выбирает данную платформу, то после регистрации необходимо выбрать тарифный план. Сетка тарифных планов приведена в Таблица 1.

Таблица 1 — Тарифы и основные возможности

Возможности	**Тарифный план**		
	Personal	**Plus**	**Pro**
Цена в год	Бесплатно	399 $	2040 $
Доступ к платформе Unity	+	+	+
Визуальный скриптинг	+	+	+
Управление заставочным экраном	-	+	+
Интеграция рекламы в проект	+	+	+

Окончание таблицы 1

Возможности	**Тарифный план**		
	Personal	**Plus**	**Pro**
Полный доступ к исходному коду Unity	-	-	+
Развертывание на игровых консолях	-	-	+
Havok Physics Unity	-	-	+
Интеграция встроенных покупок в игры	+	+	+

Разработчики вместе с пользователями создали динамичное творческое сообщество. Были реализованы такие вспомогательные сервисы как [2]:

– Unity Learn: сервис, позволяющий развивать навыки в Unity с помощью онлайн-занятий и обучающего контента общей длительностью более 750 часов, специально разработанного для любителей и профессионалов.

– Asset Store: магазин, включающий каталог платных и бесплатных вспомогательных инструментов, наборов графики и библиотек, позволяющих облегчить разработку.

– Unity Documentation: руководство, предоставляющее разработчику информацию о новых функциях и рабочих процессах Unity, а также о том, как создавать и использовать скрипты.

– Gaming service: сервис, в котором находятся проверенные решения для каждого этапа разработки.

1.2 Архитектуры при разработке игр

Рассмотрим основные подходы к архитектуре при разработке игр.

1.2.1 Entity Component System

Entity Component System (ECS) — это архитектурный паттерн, специально созданный для разработки игр, он отлично подходит для описания динамического игрового мира. ECS является частным случаем подхода, ориентированного на работу не с объектом, а с данными.

Entity — сущность. Контейнер для свойств, определяющих чем будет являться эта сущность.

Component — компонент, свойство с данными объекта. Компоненты не содержат никакой логики, в них хранятся исключительно данные.

System — система или же логика обработки данных. Системы содержат только логику обработки данных.

Исходя из того, что из себя представляет ESC видно, что данные от логики строго отделены. Поведение объекта определяется присвоенными объекту свойствами с данными и отдельной логикой обработки.

Плюсы данной архитектуры:

– хорошая модульность и тестируемость логики;

– комбинаторика свойств;

– прирост производительности в больших проектах;

– широкие возможности для параллельной обработки данных;

– масштабируемость проекта.

Минусы данной архитектуры:

– высокий порог вхождения;

– системы логики работают исключительно в потоке, друг за другом;

– плохо работает рекурсивная логика;

– больше файлов и классов.

1.2.2 HMVC

HMVC — это иерархический подход, который использует триады модель-вид-контроллер. Рассмотрим данную архитектуру (Рисунок 1).

Рисунок 1 — Пример архитектуры HMVC

Модель отвечает за обработку данных и бизнес-логику. Она содержит данные, методы для доступа и изменения данных, а также логику, связанную с обработкой и манипуляцией этими данными.

Вид отвечает за отображение данных и пользовательский интерфейс. Он представляет данные, полученные от модели, в удобном для восприятия пользователем виде.

Контроллер отвечает за обработку пользовательского ввода и взаимодействие между моделью и видом. Он принимает пользовательский ввод, обрабатывает его и обновляет модель или вид в соответствии с этим вводом.

Существует пустая сцена с GameInstaller, который подгружает контейнеры отдельно для каждой сцены. Класс GameInstaller хранит глобальные триады, которые отвечают за крупные системы. Далее GameInstaller загружает необходимый контейнер для конкретной сцены, который инициализирует верхнеуровневые триады внутри себя, а тот, в свою очередь, если это необходимо, будет инициализировать внутри себя дочерние контроллеры. И так продолжается по нисходящей. Все взаимодействие между ветвями происходит через события и поля, хранящие в себе некое значение, на изменение которого подписываются отдельные члены ветви.

Благодаря такому подходу легко разделять триады, при этом сохранять адекватную связь между её дочерними элементами [3].

Плюсы данной архитектуры:

– сцены проекта загружаются практически мгновенно;

– жесткая структурированность;

– достаточно легкая отладка по ветвям триад.

Минусы данной архитектуры:

– сложность при возникновении необходимости обработать событие в дереве триады;

– объемная совокупность программного кода;

– сложности при связывании ветвей между собой.

1.2.3 MonoBehaviour

Архитектура разработки игры на Unity с использованием MonoBehavior представляет собой объектно-ориентированный подход, в котором зачастую каждый элемент игры представлен в виде компонентов, наследующихся от базового класса MonoBehavior.

MonoBehavior — это базовый класс, который предоставляет основные методы и свойства для управления жизненным циклом компонентов, таких как Start(), Update(), FixedUpdate() и другие.

Архитектура игры на Unity состоит из нескольких основных элементов:

– сцены (scenes): контейнеры, которые содержат набор объектов, созданных в Unity. Каждая сцена представляет собой уровень или экран игры.

– игровые объекты (game objects): строительные блоки игры, представляющие собой контейнеры для компонентов. Каждый игровой объект может содержать несколько компонентов, которые управляют его поведением.

– компоненты (components): представляют собой модули функциональности, которые могут быть прикреплены к игровым объектам. Они расширяют возможности игровых объектов, добавляя различные поведения,

свойства и функциональность. Компоненты могут взаимодействовать друг с другом и с другими системами в Unity для управления игровым процессом. Например, компоненты могут управлять физикой, анимацией, визуальными эффектами и т.д.;

– скрипты (scripts): представляют собой программный код, который определяет поведение и функциональность компонентов или игровых объектов. Скрипты могут содержать игровую логику, обработку пользовательского ввода, взаимодействие с другими объектами, алгоритмы и многое другое. В Unity, скрипты обычно написаны на языке программирования C#, хотя также поддерживаются и другие языки, такие как JavaScript (устаревший) и Boo. Скрипты в Unity обычно связываются с компонентами, путем прикрепления скрипта, который наследуется от MonoBehaviour, к игровому объекту в качестве компонента. В результате, скрипт становится частью компонента и приобретает возможность взаимодействовать с другими компонентами и системами Unity;

– ресурсы (resources): файлы, используемые в игре, такие как модели, текстуры, звуки и т.д. Они могут быть загружены в игру и использованы в компонентах и скриптах.

Архитектура игры на Unity с использованием MonoBehavior позволяет разработчикам создавать гибкие и масштабируемые игры с помощью компонентного подхода. Каждый компонент может быть использован в различных игровых объектах и сценах, что упрощает процесс разработки и повторного использования кода.

1.3 Клиент-серверное взаимодействие

Для обеспечения в игре клиент-серверного взаимодействия необходимо решить, за счет чего клиент и сервер будут обмениваться данными. Рассмотрим, какие решения возможно интегрировать в Unity.

1.3.1 Mirror

Mirror позволяет встроить в игру систему multiplayer. Сервер для работы онлайн-сессий может быть развернут локально. В данном случае один из игроков является одновременно сервером и клиентом, иначе говоря хостом. Все сетевые данные передаются через хост.

В случае с выделенным сервером на отдельное независимое оборудование разворачивается сервер, к которому подключаются игроки. В данном случае все сетевые данные между игроками передаются через него. Игровая логика также обрабатывается на данном сервере.

При локальном сервере возникает проблема в подключении игроков вне данной сети, хоть и существуют программы, эмулирующие локальную сеть между ними, но никакие рядовые пользователи не захотят заниматься этим. Выделенный сервер является идеальным решением, но для стабильной работы сервера понадобятся привлекать дополнительные денежные средства.

Бесплатный и удобный вариант размещения сервера — Mirror в связке с игровой платформой Steam. В данном случае игрок-хост создает онлайн-сессию, а остальные игроки подключаются к нему через лобби. Стоит отметить, что игра не публикуется на игровой платформе как продукт, а добавляется как сторонняя игра и отображается в библиотеке игр только у тех пользователей, которые сделают это вручную.

Достоинства Mirror:

– бесплатно;

– неограниченное количество игроков для подключения, все зависит от мощности сервера;

– хорошая синхронизация между клиентами.

1.3.2 Firebase

Firebase — это сервис, разработанный компанией Google, хранящий данные в режиме реального времени. Благодаря данному решению можно

встраивать в игру системы, способные обновлять свои данные на всех устройствах, которые пользуются этой системой.

С помощью Firebase не стоит разрабатывать масштабные игры в жанре «шутер» или «MMO». Если же необходимо интегрировать в свою игру, например авторизацию, чат или таблицу лидеров, то Firebase может оказаться хорошим решением.

В Firebase есть как бесплатная версия, так и платная, со всеми расценками можно ознакомиться на официальном сайте [4].

1.3.3 LootLocker

LootLocker — это кроссплатформенный сервис, с помощью которого возможно интегрировать различные игровые системы, требующие взаимодействие с сервером. Сервис бесплатный до тех пор, пока в игре не будет десять тысяч активных пользователей в месяц [5].

LootLocker хранит все данные игроков вместе в одном месте. Для разработчика есть доступ к их профилю и списку друзей на всех платформах.

Благодаря данному сервису реализуются такие сетевые взаимодействия как:

– регистрация и авторизация игроков;

– система наград;

– системы прогресса и опыта;

– таблица лидеров;

– система монетизации и прочее.

1.4 Задание на проектирование и разработку

1.4.1 Назначение разработки

Выпускная квалификационная работа «Разработка игры на Unity «Lost Soul»:

– является комплексным проектом, охватывающим различные аспекты разработки программного обеспечения;

– подразумевает организацию клиент-серверного взаимодействия;

– является продуктом в сфере компьютерных развлечений.

1.4.2 Концепт разрабатываемой игры

Жанр игры — платформер, вид игры — 2D, целевая платформа — Windows и Android.

Игровой персонаж, которым пользователь управляет, обладает «энергией тьмы», с помощью которой способен поглощать элементы стихий, тем самым побеждать врагов. Эскиз игрового персонажа (Рисунок 2).

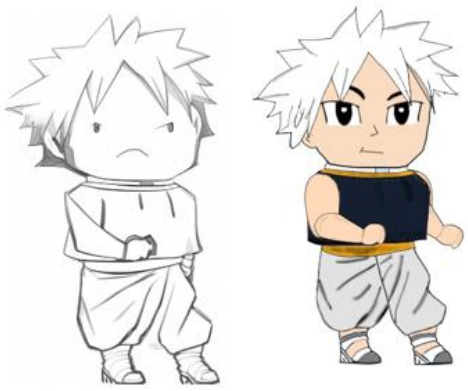

Рисунок 2 — Эскиз игрового персонажа

Игровой уровень — это одна из стихий: дендро, ветер, вода, земля. На локации находятся враги, побеждая которых главный герой накапливает энергию элемента, которую возможно потратить на замораживающую атаку.

Планируется 4 игровых уровня с различным графическим оформлением в соответствии со стихией уровня. Также, на трех локациях будет доступна своя уникальная механика: крюк-кошка, потребление воздуха, ограниченная видимость.

Цель уровня — используя механики, доступные для конкретного уровня дойти до портала. В конце уровня игроку начисляется опыт и очки,

отображаемые в таблице лидеров, которые складываются из затраченного времени и подобранных монет на уровне.

1.4.3 Требования к функциональным характеристикам

Игра предусматривает как наличие клиента, так и взаимодействие с сервером. В процессе работы приложения пользователь является участником игрового процесса — оказывает непосредственное влияние как на него, так и на данные на сервере.

На стороне клиента должен присутствовать такой функционал как:

– интерфейс регистрации и авторизации;

– интерфейс главного меню;

– музыкальное сопровождение;

– настройка звука;

– игровые сцены;

– интерфейсы пользователя для игровых сцен;

– взаимодействие с игровыми объектами;

– боевая система;

– специальные механики.

На стороне сервера должен присутствовать такой функционал как:

– система регистрации и авторизации пользователей;

– система подтверждения почты;

– система установки и замены игрового имени;

– система игрового опыта и наград, отображаемых в главном меню;

– система таблицы лидеров, основанная на игровым опыте, который выдается по завершению игрового уровня.

1.5 Выводы по первой главе

После изучения основных сведений о игровом движке, основных архитектурных решений и сервисов в области клиент-серверного

взаимодействия, резюмируем, что у каждого подхода есть свои достоинства и недостатки. Выбор архитектуры зависит от размеров проекта и команды разработчиков. Если проект масштабный, с большим количеством объектов и сложной логикой, то ECS может быть более подходящим выбором. Это позволит более эффективно управлять ресурсами и обеспечить высокую производительность. Если же проект небольшой и команда разработчиков небольшая, то использование классических подходов на MonoBehaviour может быть проще и удобнее. Исходя из выше сказанного было принято решение, что проектирование и дальнейшая реализация будут выполнены, исходя из классического подхода к архитектуре на MonoBehaviour. После анализа существующих решений было решено использовать сервис LootLocker, так как данный сервис позволит реализовать игровые механики, описанные в задании на проектирование и разработку эффективней, чем аналоги в области клиент-серверного взаимодействия. Составлено задание на проектирование и разработку.

2 Проектирование игры

2.1 Проектирование игровых сцен и игровые механики

2.1.1 Стартовая сцена

При запуске игры пользователя встречает сцена, которой управляет один класс StartPageManager, он представляет собой логику управления объектами, обработку пользовательского ввода и взаимодействие с сервером для аутентификации, создания игрока, установки имени и получения данных игрока.

Класс StartPageManager (Рисунок 3).

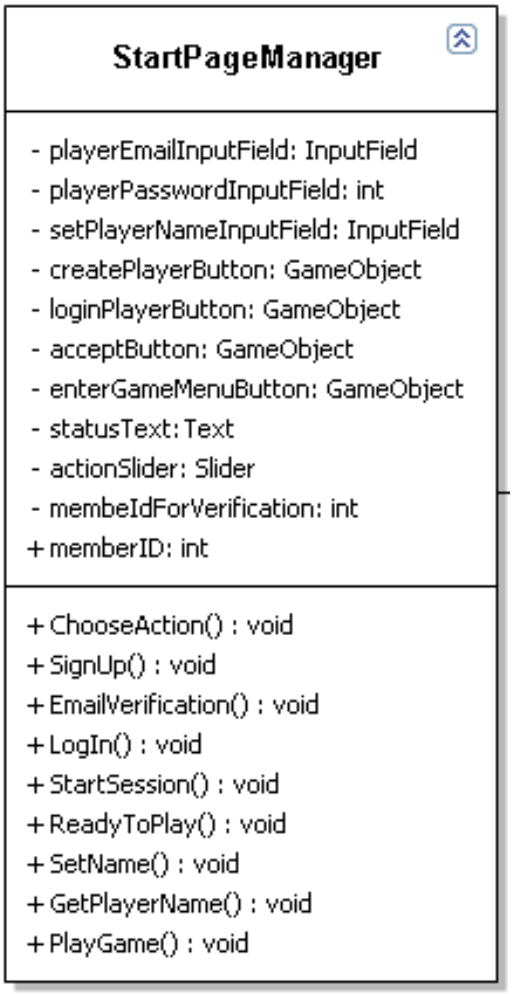

Рисунок 3 — Класс StartPageManager

Описание методов класса приведены в приложении А.

Диаграмма последовательности авторизации (Рисунок 4).

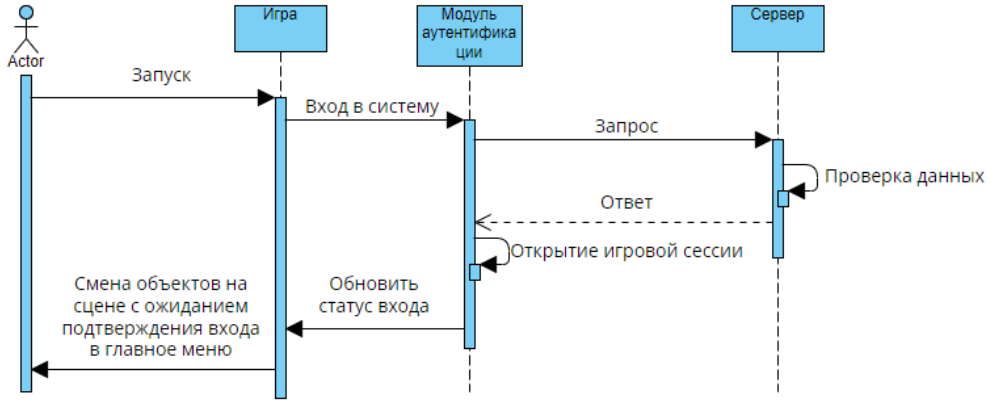

Рисунок 4 — Диаграмма последовательности авторизации

После ввода данных в поля «Email» и «Password» игра отправляет запрос на сервер об авторизации игрока, если от сервера приходит положительный ответ, то открывается игровая сессия и на интерфейсе авторизированного пользователя появляется кнопка с подтверждением входа в главное меню игры.

Диаграмма последовательности регистрации (Рисунок 5).

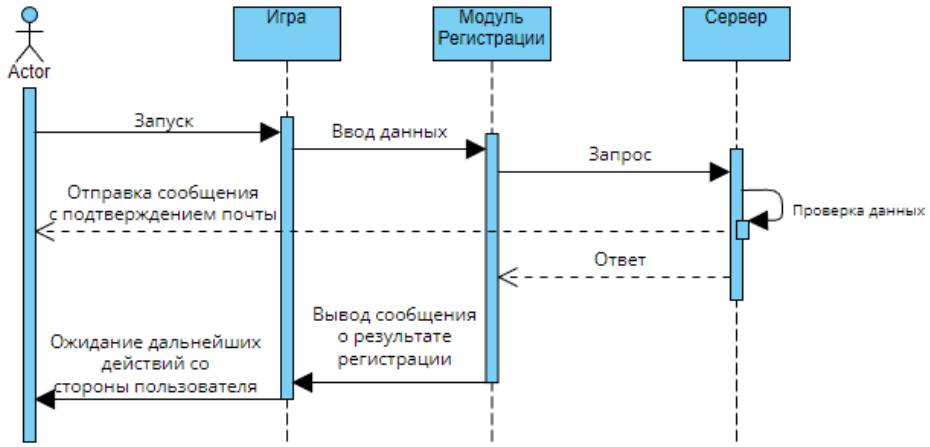

Рисунок 5 — Диаграмма последовательности регистрации

После ввода данных в поля «Email» и «Password» игра отправляет запрос на сервер об регистрации игрока, если от сервера приходит положительный ответ, то выводится сообщение об успешной регистрации и на почту игрока приходи сообщение о подтверждении регистрации, затем игра ожидает дальнейших действий со стороны пользователя.

2.1.2 Сцена главного меню

После успешной авторизации и запуска игровой сессии пользователь может взаимодействовать с сервисом LootLocker, пока не покинет игру. В главном меню для пользователя доступен следующий функционал:

– игровое имя;

– система опыта за прохождение уровней;

– таблица лидеров по наилучшему результату на уровне;

– список наград;

– выбор игрового уровня.

Класс MenuManager (Рисунок 6).

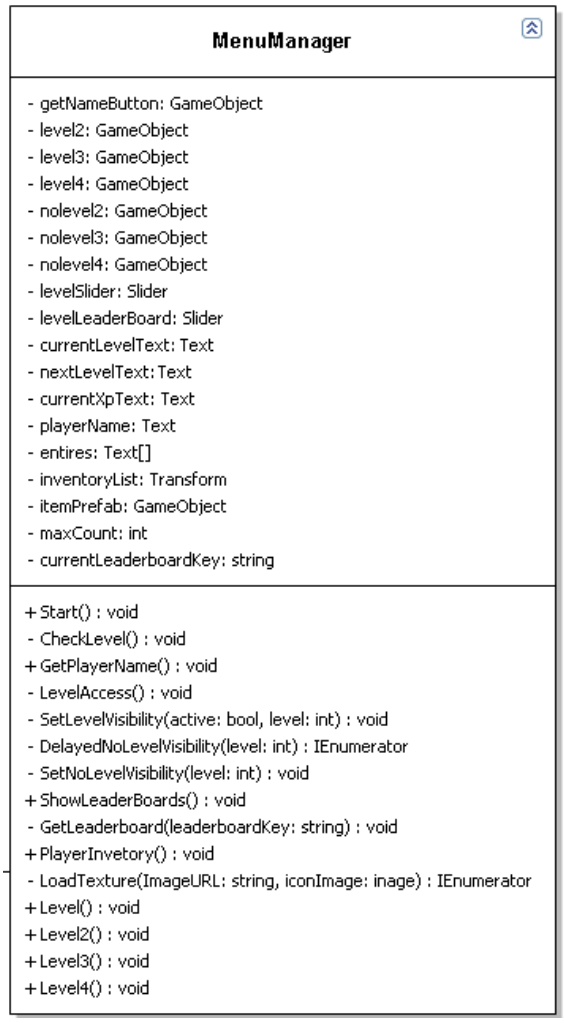

Рисунок 6 — Класс MenuManager

Описание методов класса приведены в приложении А.

2.1.3 Общие механики персонажа для игровых сцен

На диаграмме состояний (Рисунок 7) для игрового персонажа существуют следующие возможные состояния:

– бездействие;

– бег;

– прыжок;

– атака;

– атака специальная.

Если же состояние «Смерть (Интерфейс GameOverMeu)» становится активным и означает, что игрок не справился с прохождением игрового уровня.

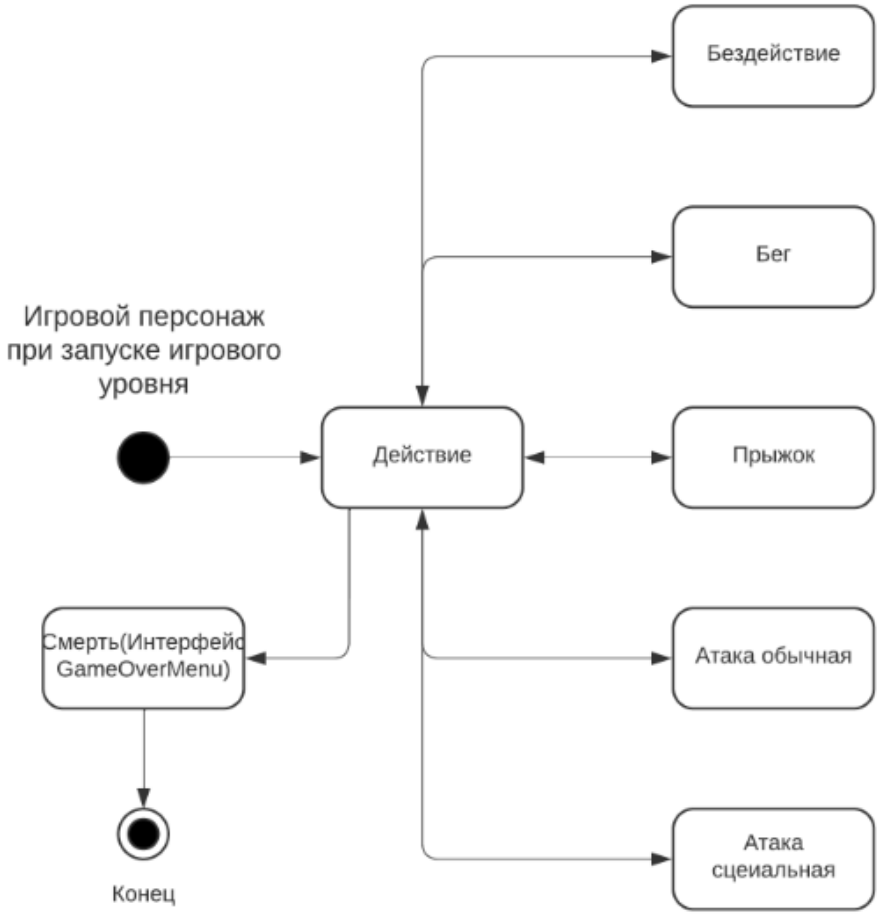

Рисунок 7 — Диаграмма состояний игрового персонажа

Класс PlayerController отвечает за управление игровым персонажем.

Класс Health отвечает за здоровье игрового персонажа.

Класс Health взаимодействует с классом HealthBar, обеспечивая работу событий.

Класс HealthBar обновляет изображение шкалы здоровья на экране в соответствии с изменением здоровья объекта, к которому прикреплен компонент Health.

Класс AttackSystem отвечает за боевую систему игрового персонажа (обычный и замораживающий выстрел).

Класс BulletSystem является абстрактным и наследуется от MonoBehaviour. Он содержит параметры и логику для атаки.

Экземпляр обычной пули содержит класс SimpleBullet и представляет снаряд, который наносит обычный урон.

Экземпляр замораживающей пули содержит класс FreezBullet и наследуется от представляет снаряд, который наносит замораживающий урон.

Класс TimerController необходим для того, чтобы подсчитать, сколько времени игрок потратил на прохождение уровня.

Диаграмма классов общих механик игрового персонажа (Рисунок 8).

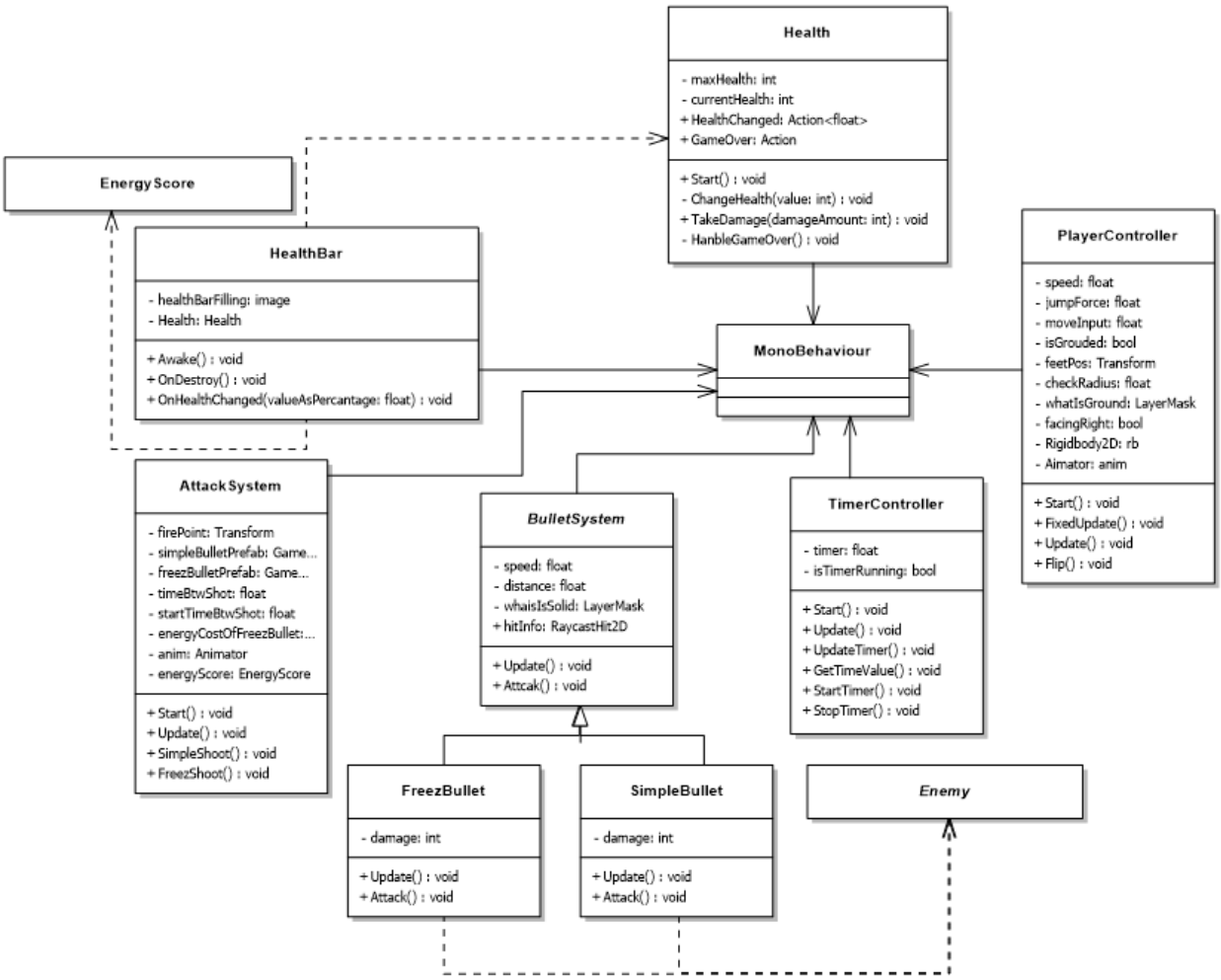

Рисунок 8 — Диаграмма классов общих механик игрового персонажа

Описание методов классов общих механик персонажа для игровых сцен приведены в приложении А.

2.1.4 Ресурсы

В игре существует три типа ресурсов, которые игрок может собрать. Это энергия, монеты и сферы воздуха. Данные ресурсы могут находиться как на игровом поле, путем размещения префабов этих ресурсов, так и «выпадать» при уничтожении противников — за это отвечает класс SpawnAttributes.

В классе Collector находится список объектов, реализующих интерфейс IResourceCollector.

В классе ItemCollector регистрирует себя в Collector, а также находит и получает ссылки на объекты AirField, CoinScore и EnergyScore.

Интерфейс IResourceCollector содержит в себе типы ресурсов и служит для сбора ресурса. Реализуется классами, которые выполняют сбор ресурсов.

В классе EnergyScore реализовано хранилище энергии.

В классе CoinScore реализован счетчик очков с монетами.

Диаграмма классов ресурсов (Рисунок 9).

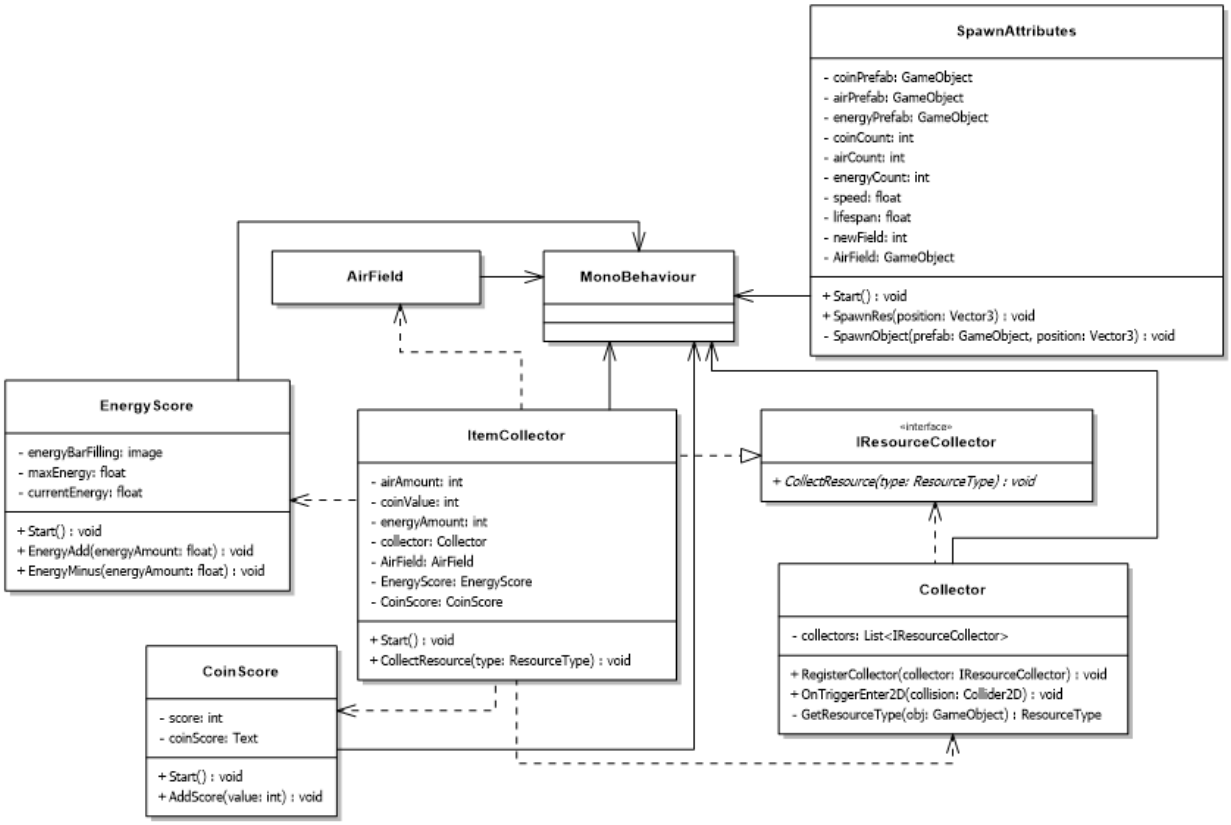

Рисунок 9 — Диаграмма классов ресурсов

Описание методов классов для ресурсов приведены в приложении А.

2.1.5 Логика поведения для противников

На диаграмме состояний (Рисунок 10) представлено поведение противников. Патрулирование территории становится активным сразу при появлении объекта на активном кадре локации, в случае обнаружения игрока переключается в «Преследование игрока», в случае доступа к атаке становится активным «Атака». Состояние «Заморожен» означает, что противник обездвижен.

В состояние «Смерть (Уничтожение объекта)» переключается в случае уровня здоровья противника меньшим или равно нулю.

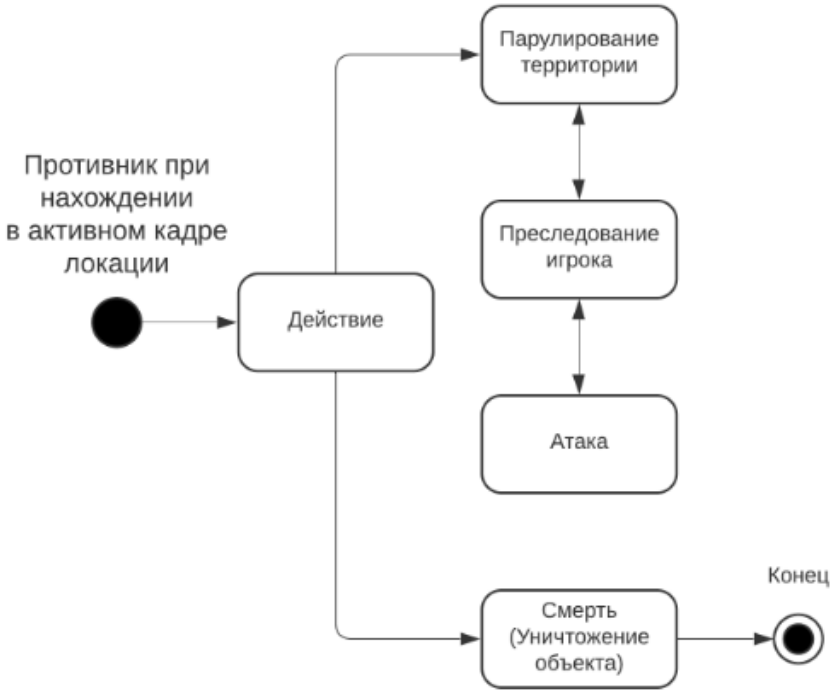

Рисунок 10 — Диаграмма состояний «Enemy»

Логика поведения для различных типов противников содержится в абстрактном классе Enemy. Так как класс Enemy является абстрактным, поэтому он не может быть использован напрямую в игре. Вместо этого его наследники должны реализовать метод EnemyAttack() согласно своей уникальной логике атаки.

Классы Slime и Skeleton соответствуют типам противников и реализуют класс Enemy в соответствии со своей уникальной атакой.

За здоровье противника отвечает класс EnemyHealth.

Диаграмма классов противников (Рисунок 11).

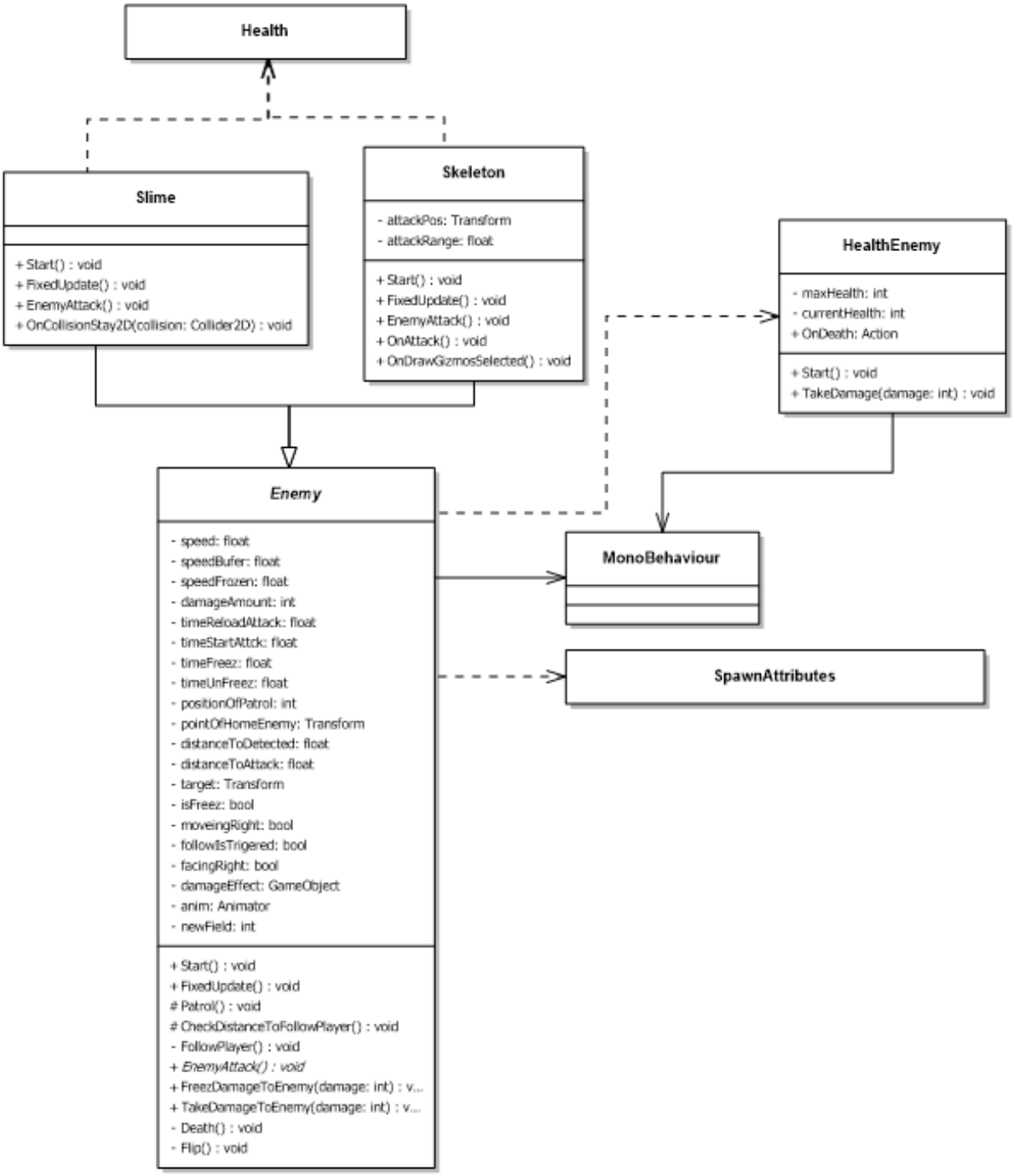

Рисунок 11 — Диаграмма классов противников

Описание методов классов для противников приведены в приложении А.

2.1.6 Служебные классы

Служебные классы представляют собой классы, которые управляют такими событиями, как открытие и закрытие интерфейса меню паузы,

интерфейса при смерти игрока и интерфейса конца уровня, а также событиями смены активных участков игровой сцены.

Класс PauseMenu отвечает за управление интерфейса паузы.

Класс GameOverMenu отвечает за управление при смерти игрока.

Класс EndLevel отвечает за завершение уровня игры и взаимодействует с таблицей лидеров и начисляет опыт.

Класс FrameSwitch позволяет контролировать активацию и деактивацию участков игровой сцены.

Диаграмма классов служебных классов (Рисунок 12).

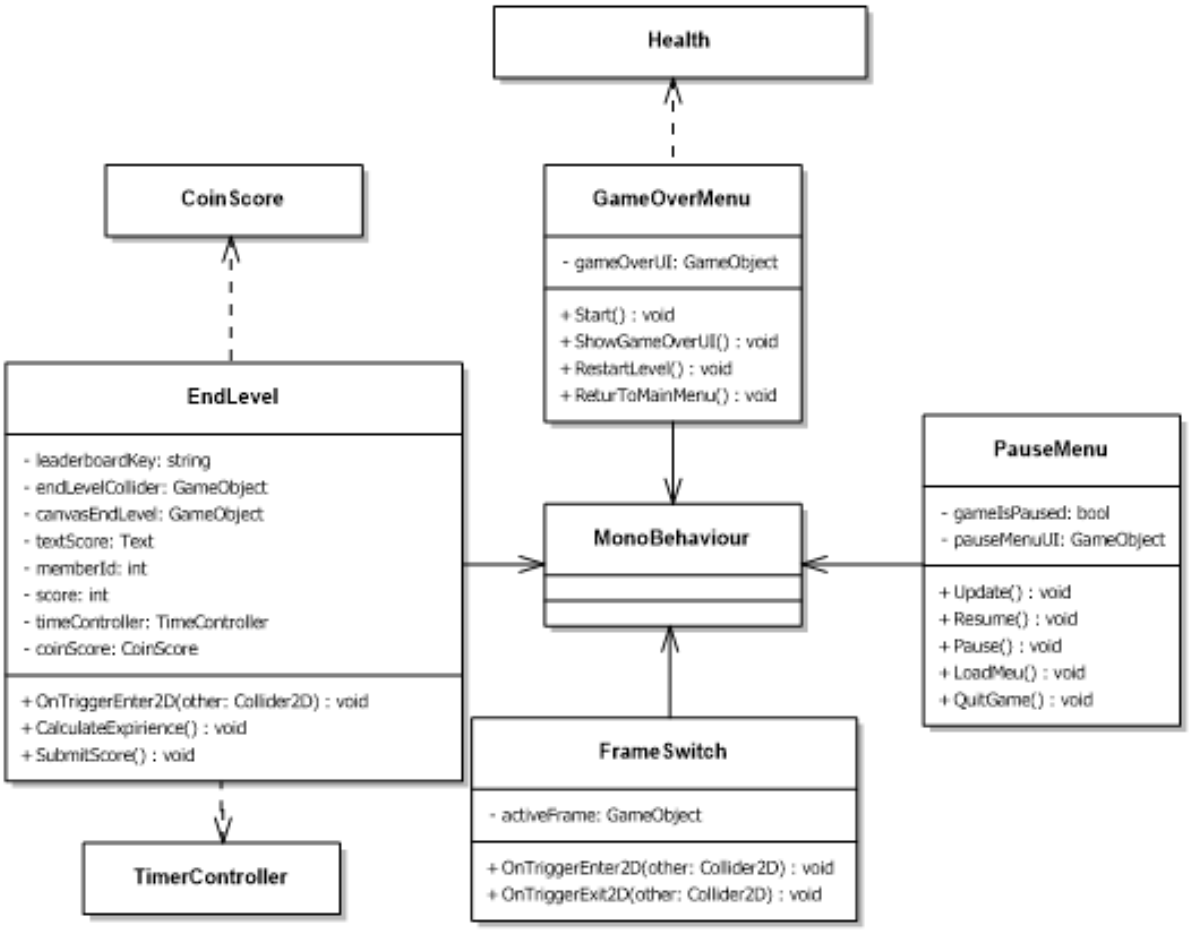

Рисунок 12 — Диаграмма классов служебных классов.

Описание методов служебных классов приведены в приложении А.

2.1.7 Специальные механики для игровых уровней

При разработке концепта игры было решено использовать специальные механики для игровых уровней, что позволит разнообразить процесс игры.

Для подводного уровня будет реализована механика потребления воздуха. За это отвечает класс AirField (Рисунок 13).

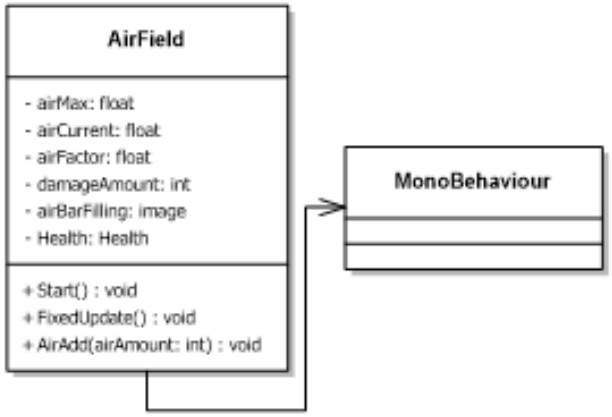

Рисунок 13 — Класс AirField

Описание методов класса приведены в приложении А.

Для уровня в горах будет реализована механика крюк-кошка. Класс GrabHook отвечает за создание и управление крюка, используемым для захвата объектов.

Класс HookRayCast возвращается информацию о столкновении в виде RaycastHit2D, включая информацию о столкнувшемся объекте и точке столкновения.

Класс HookRender отвечает за отображение линии крюка.

Диаграмма классов механики крюк-кошка (Рисунок 14).

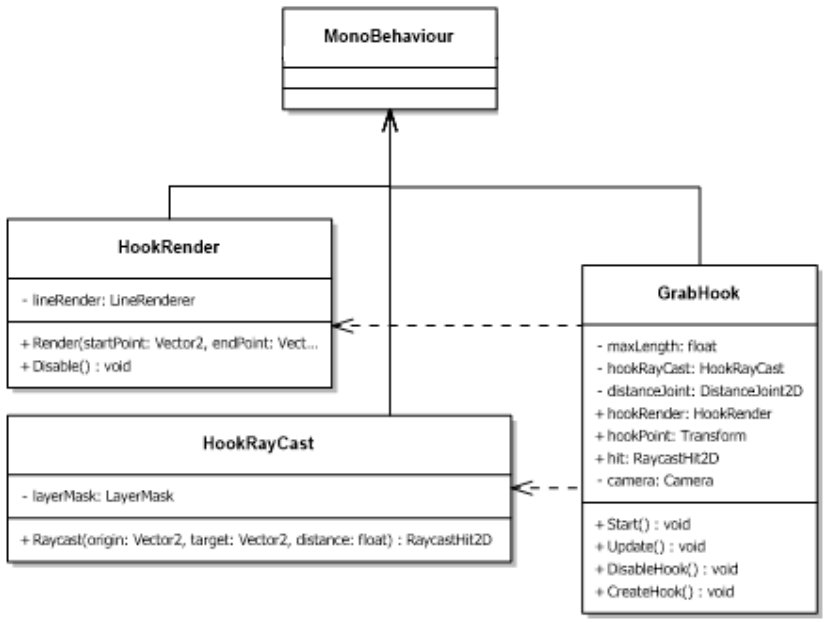

Рисунок 14 — Диаграмма классов механики крюк-кошка

Описание методов классов для механики крюк-кошка приведены в приложении А.

Для подземного уровня будет реализована механика свет, для данной механики нет необходимости писать код. Объект «Свет» будет дочерним для объекта «Player». Исходя из этого, при движении персонажа свет будет следовать за игроком. Реализуется за счет дополнительного пакета Universal Render Pipeline.

2.2 Запросы на сервер

В код интегрированы такие запросы на сервер как:

– LootLockerSDKManager.WhiteLabelSignUp отвечает за регистрацию;

– LootLockerSDKManager.WhiteLabelRequestVerification отвечает за отправку сообщения с подтверждением почты;

– LootLockerSDKManager.WhiteLabelLogin отвечает за авторизацию;

– LootLockerSDKManager.StartWhiteLabelSession отвечает за открытие игровой сессии;

– LootLockerSDKManager.SetPlayerName установка игрового имени;

– LootLockerSDKManager.GetPlayerName получение игрового имени;

– LootLockerSDKManager.GetPlayerInfo получает информацию об опыте и уровне;

– LootLockerSDKManager.GetScoreList получает таблицу лидеров;

– LootLockerSDKManager.GetInventory получает список вещей в инвентаре;

– LootLockerSDKManager.SubmitScore начисляет опыт игроку в таблицу лидеров текущего уровня;

– LootLockerSDKManager.SubmitXp начисляет опыт игроку в систему опыта.

2.3 Итоговая диаграмма классов

Итоговая диаграмма классов (Рисунок 15).

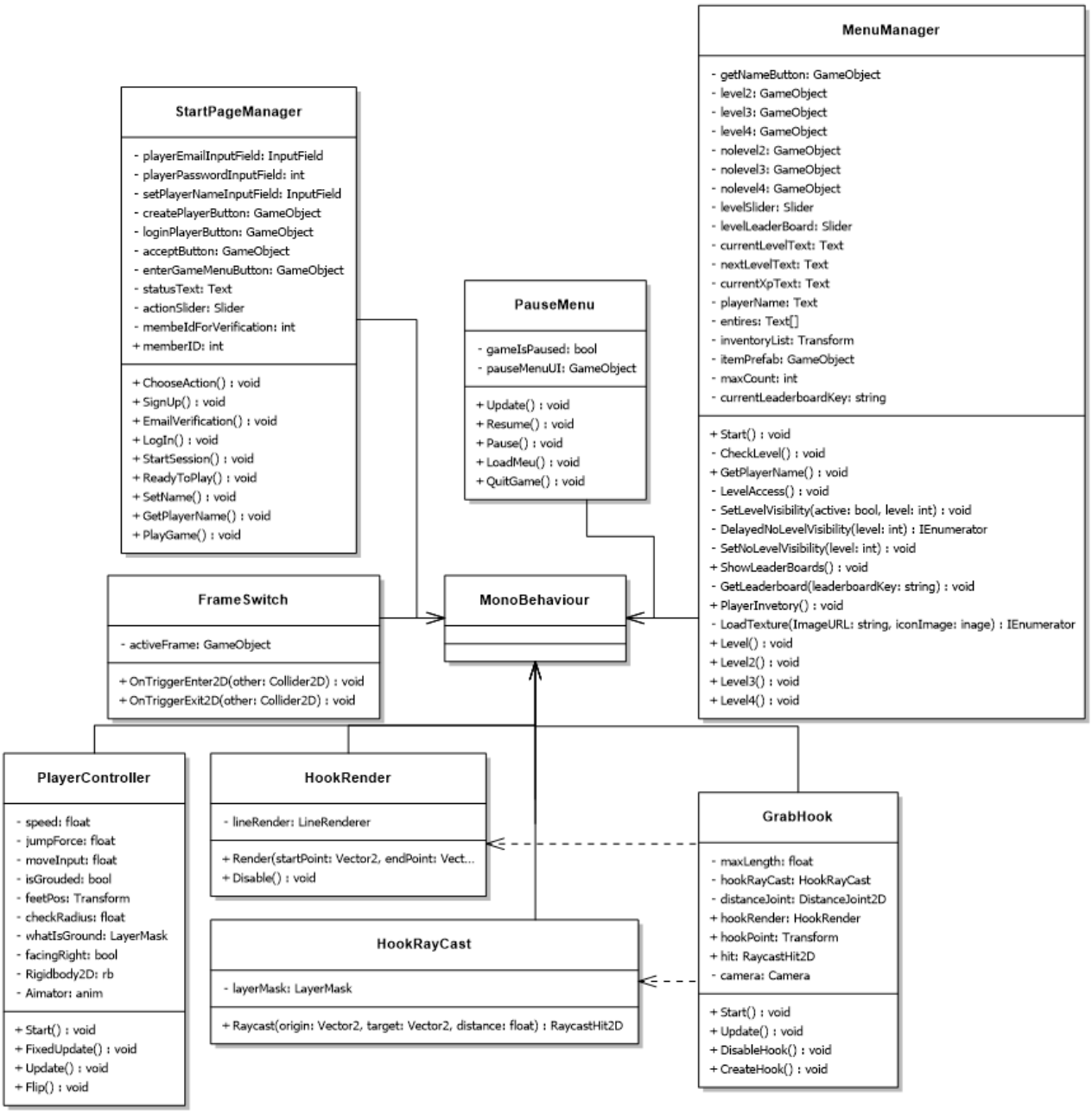

Рисунок 15 — Итоговая диаграмма классов, лист 1

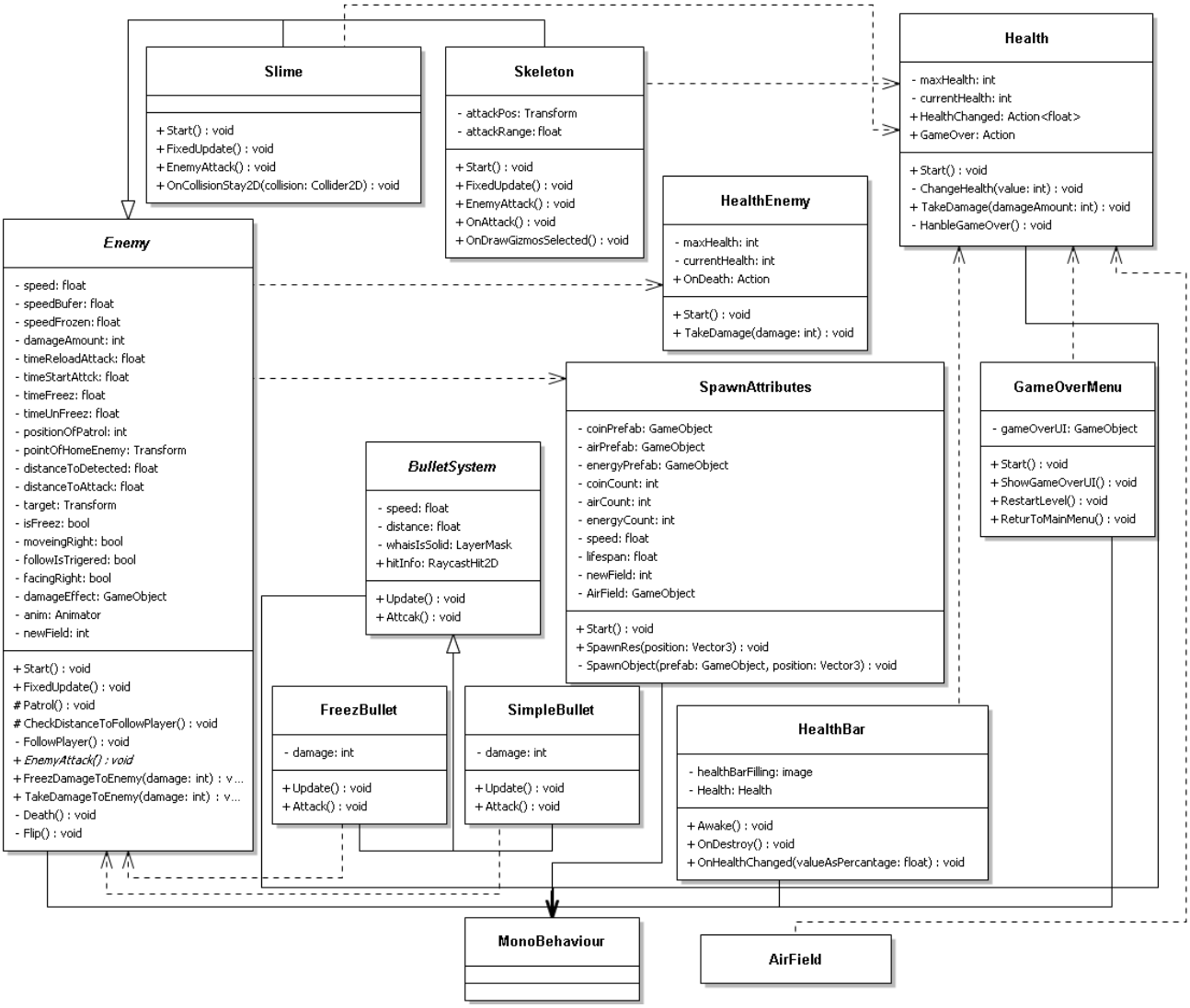

Рисунок 15, лист 2

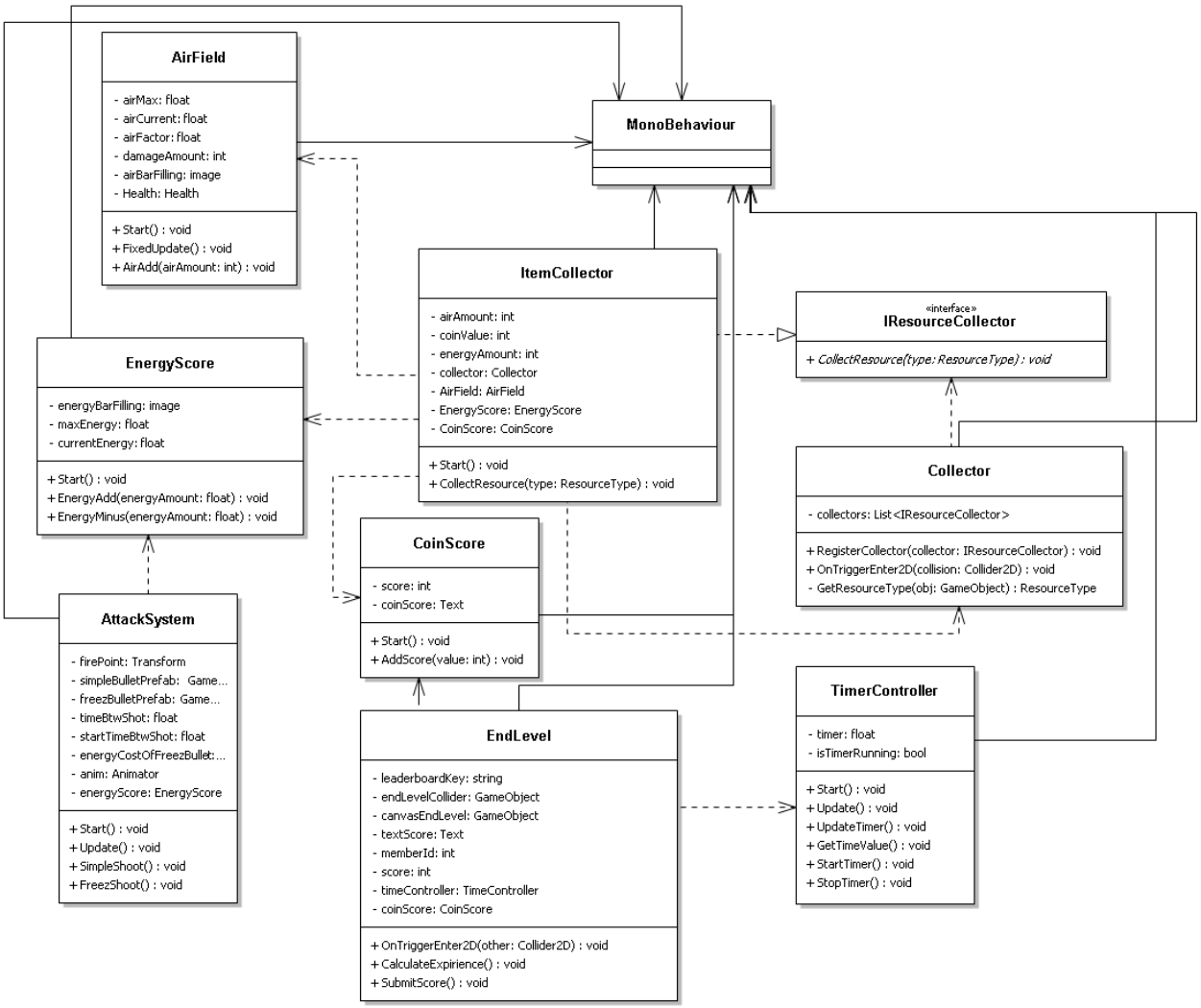

Рисунок 15, лист 3

2.4 Выводы по второй главе

В ходе выполнения второй главы выпускной квалификационной работы были выполнены такие задачи как:

– проектирование игры: были спроектированы классы и методы, созданы диаграммы последовательности для авторизации и регистрации, диаграммы состояний для игрового персонажа и противников, итоговая диаграмма классов;

– отрисовка дизайна: были отрисованы все необходимые объекты.

3 Реализация игры

3.1 Общие моменты реализации

3.1.1 Назначение скриптов и компонентов к объектам

Для того чтобы объект исполнял какую-либо логику, необходимо во вкладке Inspector назначить тот или иной компонент. Пример назначения компонентов (Рисунок 16).

Рисунок 16 — Пример назначения компонентов

Настройка переменных скриптов в Inspector осуществляется за счет модификатора доступа public или же [SerializeField] private.

Объектам возможно назначить компоненты, отвечающие за звук, видео, камеру, слои, физику, свет, события.

3.1.2 Основные настройки сцены

Находясь на игровой сцене, пользователь переходит между локациями. Объекты прорисовываются только на активной локации.

Создается пустой объект на сцене, назовем его Frame, для которого назначается Polygon Collider2D для ограничения отображаемого пространства в игре, иначе говоря, это зона, за которую камера не сможет выйти.

В дальнейшем все объекты, отвечающие за игровое окружение, размещаются в объект Frame, это необходимо для того, чтобы при работе скрипта FrameSwitch объекты отключались и включались группами.

Само переключение считывается за счет дополнительных объектов с компонентом Box Collider2D.

Для того чтобы игровое пространство не было статичным был использован сторонний скрипт, обеспечивающий параллакс эффект [5]. Объекту Frame назначается компонент ParallaxBackgound. А всем дочерним объектам компонент ParallaxLayer, в котором устанавливается необходимое значение Parallax Factor, обеспечивающее разную скорость перемещения окружения при передвижении игрока по игровому пространству. За счет разной скорости передвижения объектов создается впечатление, что одни объекты находятся близко, другие далеко.

В объект Frame также необходимо поместить еще один пустой объект, который будет иметь компонент Polygon Collider2D для обозначения зоны, по которой игрок может передвигаться.

Пример настройки сцены с игровым пространством (Рисунок 17).

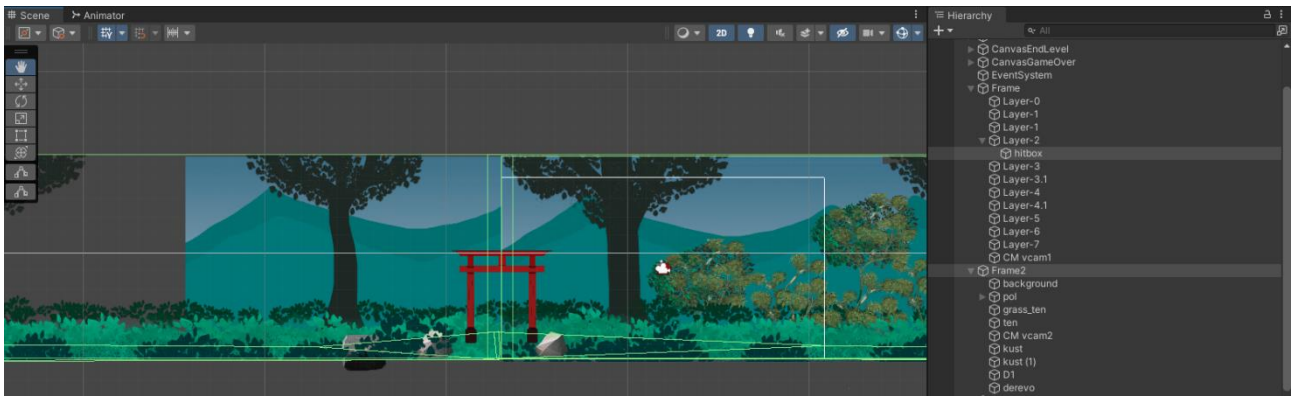

Рисунок 17 — Пример настройки сцены с игровым пространством

3.1.3 Анимация объектов

Для более реалистичного поведения объектов их необходимо анимировать. Выбираем необходимый нам объект, переходим в окно Animation, после нажатия кнопки «запись» двигаем слои объекта. На 0 кадре слои находятся в исходном положении, на 20 кадре происходит изменение, на 40 возвращается в исходное положение (Рисунок 18).

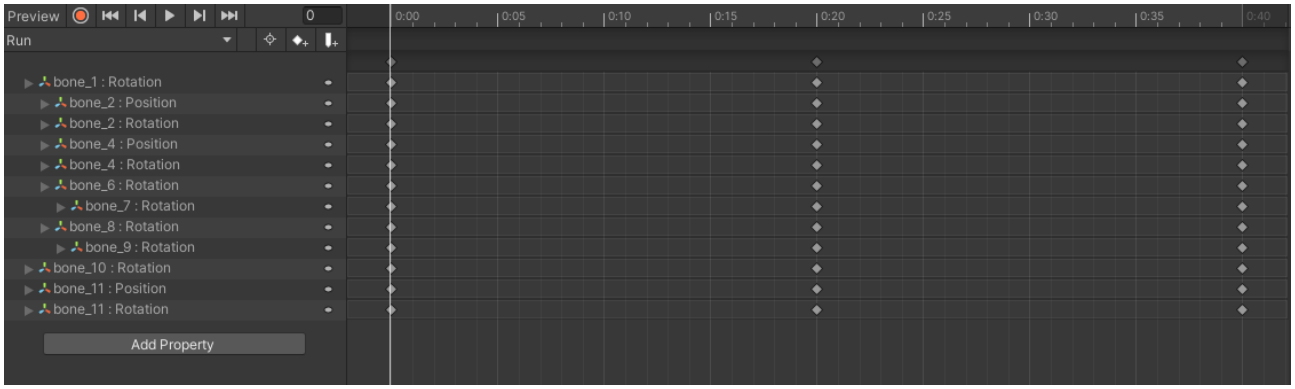

Рисунок 18 — Запись анимации

В Unity возможно анимировать объекты с помощью костной анимации. Пример костной анимации (Рисунок 19).

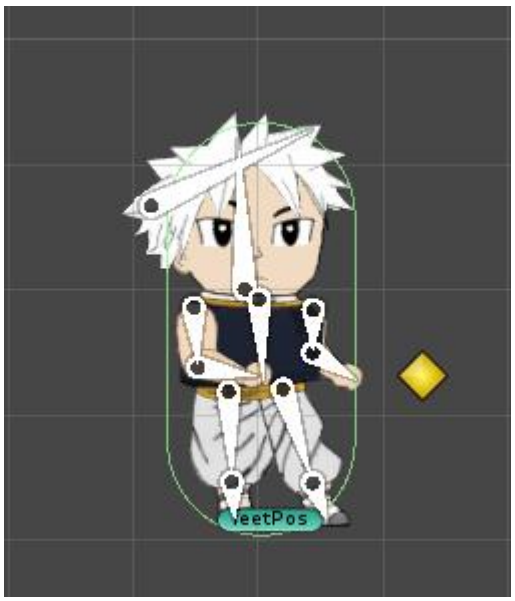

Рисунок 19 — Костная анимация

После создания всех необходимых анимационных файлов необходимо простроить между ними связи для переключения. В дальнейшем на состояния анимации возможно обращаться в скриптах. Пример взаимодействия состояний анимации (Рисунок 20).

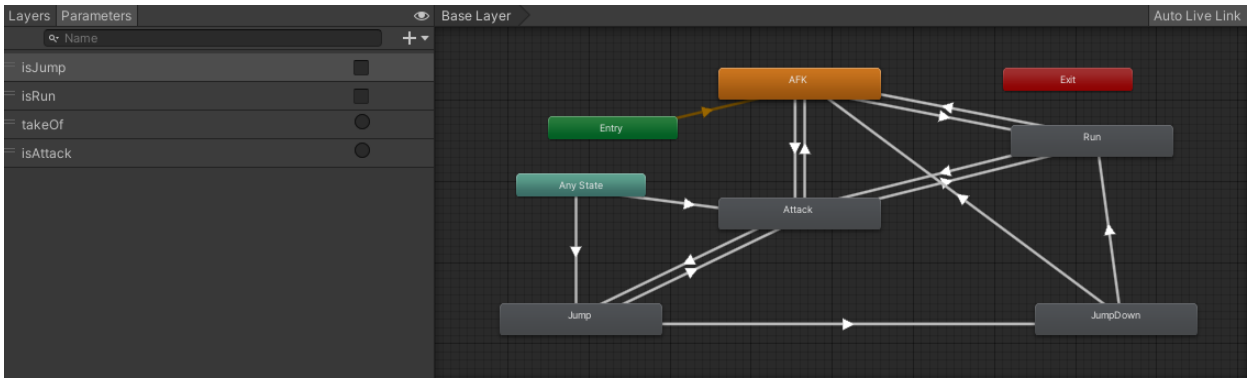

Рисунок 20 — Взаимодействие состояний анимации

3.2 Реализация интерфейсов

UI Canvas в Unity 2D является мощным инструментом для создания пользовательского интерфейса. Доступны следующие элементы: кнопка, текст, изображение, текстовое поле ввода, ползунок, выпадающий список и другие.

3.2.1 Интерфейсы на игровых сценах

Интерфейс игрового персонажа (Рисунок 21) содержит:

– health основанный на элементах Image, показывающих уровень здоровья игрока;

– air основанный на элементах Image, показывающий запас воздуха игрока;

– energy основанный на элементах Image, показывающий запас энергии игрока;

– coin основанный на элементе Text, показывающий количество собранных монет.

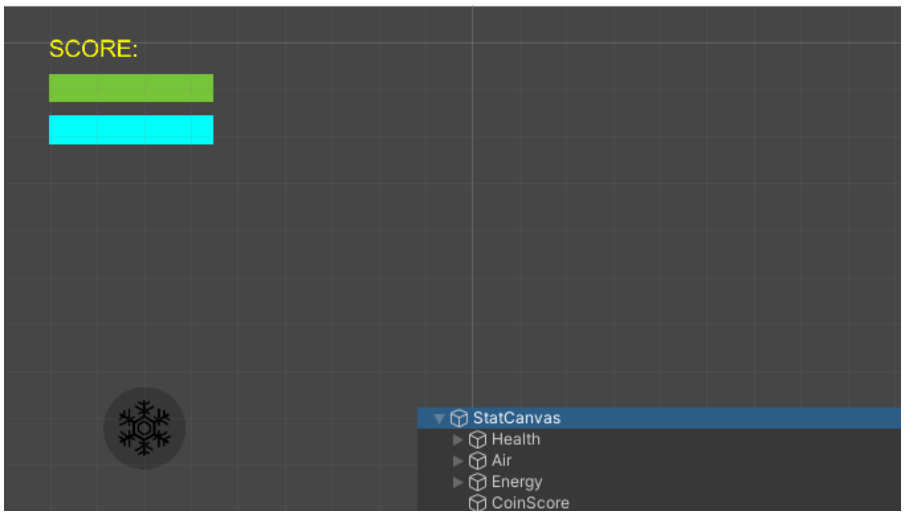

Рисунок 21 — Интерфейс игрового персонажа

Интерфейс паузы (Рисунок 22) содержит:

– Panel заполняет интерфейс определенным цветом по всей площади;

– ResumeButton основанный на элементе Button, обеспечивающий возобновление игры;

– MenuButton основанный на элементе Button, обеспечивающий выход в игровое меню;

– QuitButton основанный на элементе Button, обеспечивающий выход из игры.

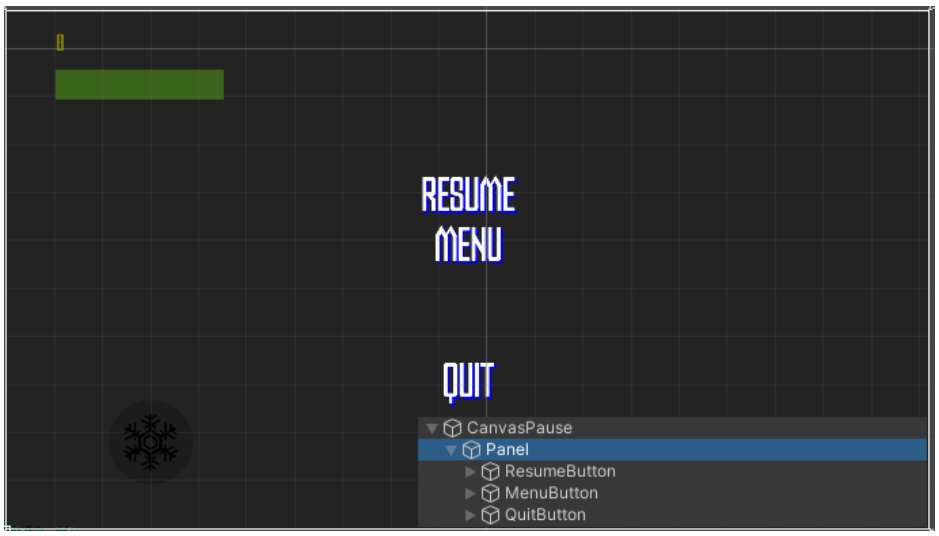

Рисунок 22 — Интерфейс паузы

Интерфейс в случае поражения (Рисунок 23) содержит:

– Panel заполняет интерфейс определенным цветом по всей площади;

– RestartButton основанный на элементе Button, обеспечивающий перезапуск текущей игровой сцены;

– MenuButton основанный на элементе Button, обеспечивающий выход в игровое меню.

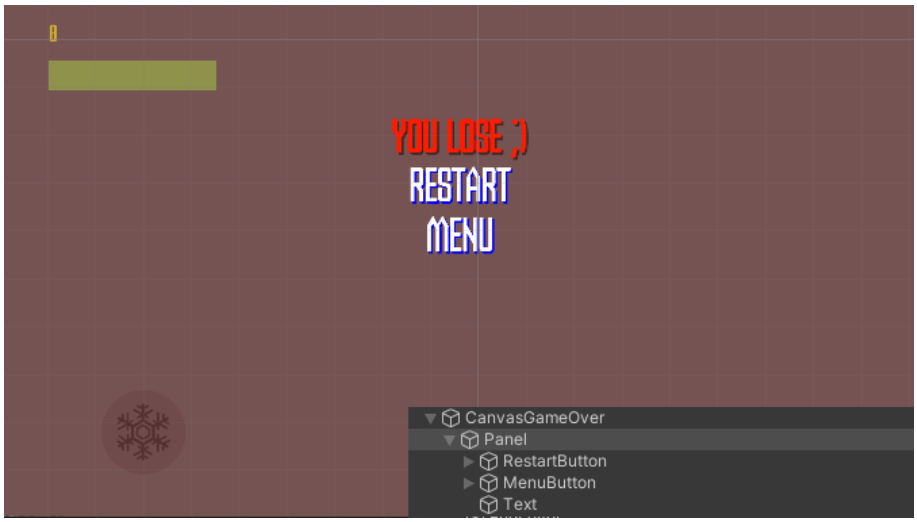

Рисунок 23 — Интерфейс в случае поражения

Интерфейс в случае победы (Рисунок 24) содержит:

– Panel заполняет интерфейс определенным цветом по всей площади;

– MenuButton основанный на элементе Button, обеспечивающий выход в игровое меню;

– TextScore основанный на элементе Text, показывающий количество заработанных очков.

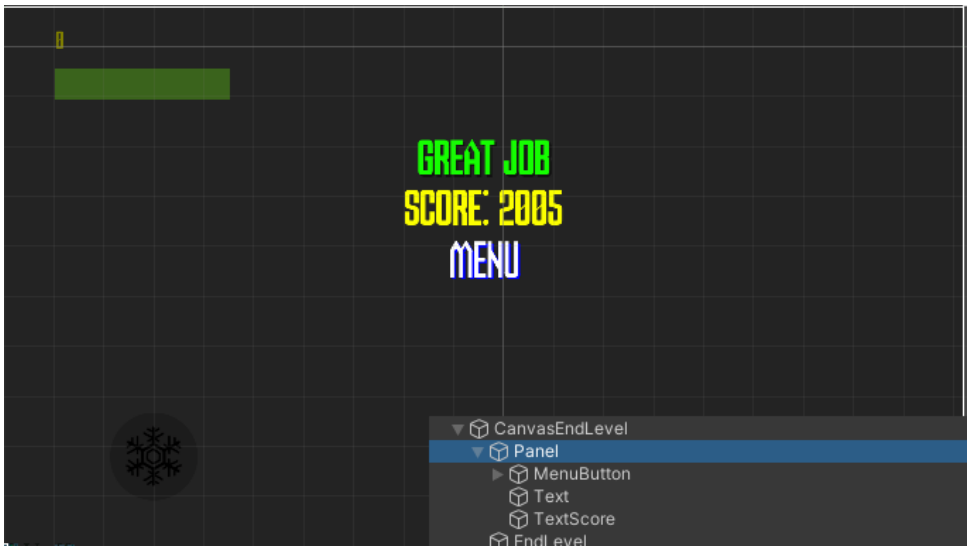

Рисунок 24 — Интерфейс в случае победы

3.2.2 Интерфейсы стартовой сцены и главного меню

Интерфейс стартовой сцены необходим для авторизации, регистрации, установки игрового имени (Рисунок 25). Интерфейс содержит:

– PlayerEmailInputField, PlayerPasswordInputField, SetPlayerNameInputField основанные на элементе InputField, обеспечивающие ввод данных;

– EnterGameMenuButton, AcceptButton, CreatePlayerButton, LoginPlayerButton основанные на элементе Button, обеспечивающие вход в игру, подтверждение установки игрового имени, регистрацию и авторизацию соответственно;

– Slider переключает активацию элементов для регистрации и авторизации;

– Video содержит повторяющийся видеофрагмент для визуального оформления заднего плана;

– объект gameManeger содержит скрипт StartPageManager, который управляет всем происходящим на сцене.

Рисунок 25 — Интерфейс стартовой сцены

Интерфейс главного меню необходим для отображения игрового имени, инвентаря игрока, таблицы лидеров, уровня игрока и его прогресса (Рисунок 26). Интерфейс содержит:

– PlayerName основанный на элементе Text, показывает игровое имя;

– SliderXp основанный на элементе Slider, показывает текущее количество опыта и необходимое количество опыта для повышения уровня;

– LevelSelector основанных на элементах Button, обеспечивающий вход на один из четырех игровых уровней;

– LeaderBoard основанный на элементах Text, выводит на экран таблицу лидеров, максимум 6 лучших игроков;

– SliderLeaderBoard основанный на элементе Slider, переключает выводимую таблицу лидеров;

– Inventory основанный на Scrollbar Vertical и Viewport, отображает на экране инвентарь игрока;

– Video содержит повторяющийся видеофрагмент для визуального оформления заднего плана;

– объект MenuManeger содержит скрипт MenuManager, который управляет всем происходящим на сцене.

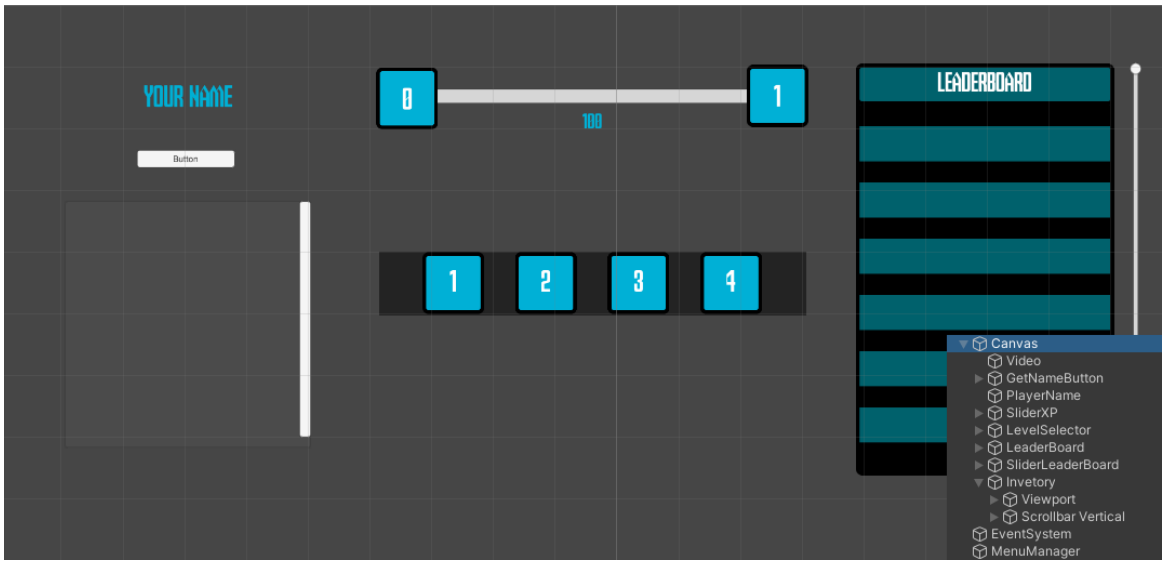

Рисунок 26 — Интерфейс главного меню

3.3 Реализация игровых уровней

3.3.1 Первый уровень

Первый уровень графически оформлен в стиле стихии дендро (от греческого dendron - дерево). Вся локация выполнена в зеленых тонах (Рисунок 27). Данный уровень знакомит игрока с боевой системой.

Рисунок 27 — Первый уровень

3.3.2 Второй уровень

Второй уровень графически оформлен в стиле стихии ветра. Локация в горах. На уровне игроку доступна специальная механика крюк-кошка, предназначенная для преодоления обрывов (Рисунок 28).

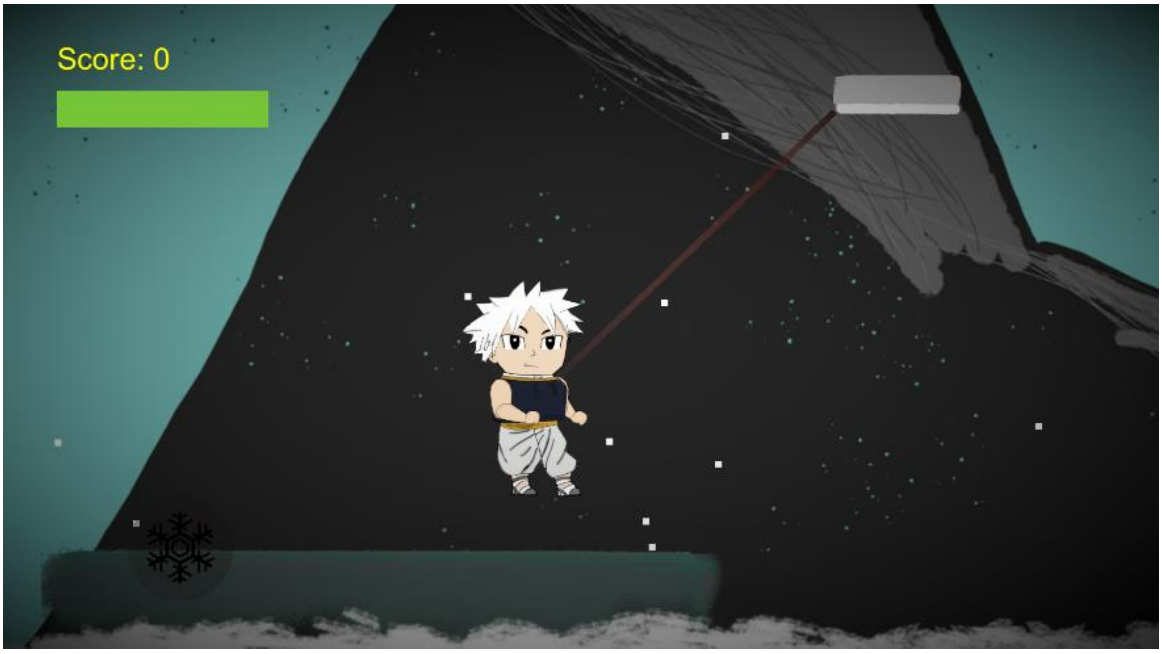

Рисунок 28 — Второй уровень

3.3.3 Третий уровень

Третий уровень графически оформлен в стиле стихии воды. Вся локация находится под водой. На уровне игроку необходимо следить за запасом воздуха (Рисунок 29).

Рисунок 29 — Третий уровень

3.3.4 Четвертый уровень

Четвертый уровень графический оформлен в стиле стихии земли. Локация находится под землей. На уровне игроку необходимо в условиях ограниченной видимости пройти уровень (Рисунок 30).

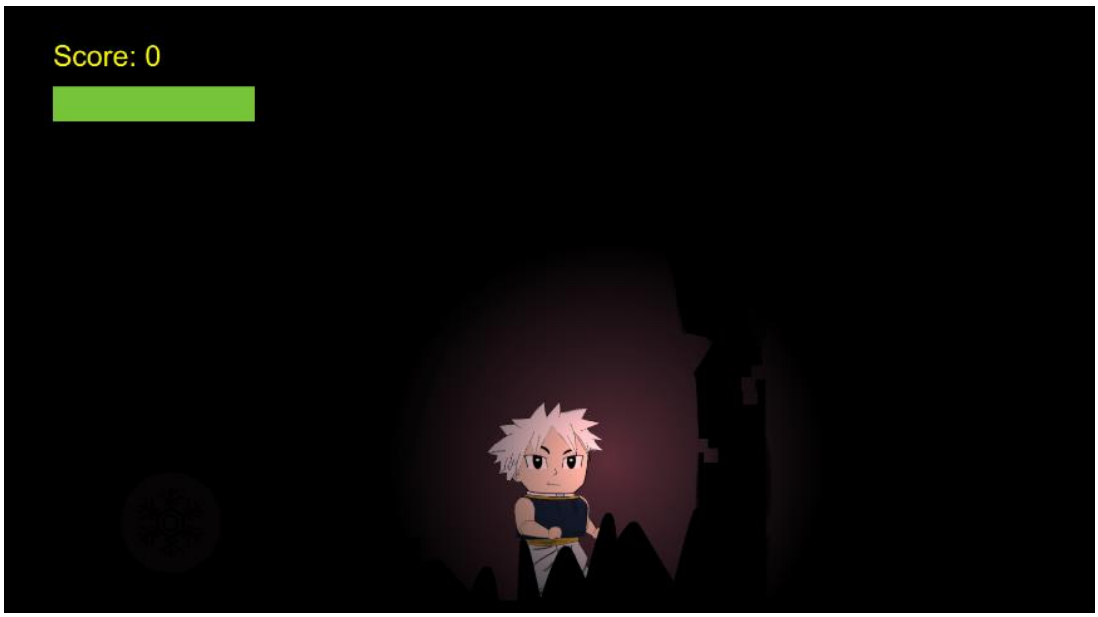

Рисунок 30 — Четвертый уровень

3.4 Реализация врагов

На всех игровых уровнях игроку придется столкнуться с такими противниками как слизь и скелет (Рисунок 31).

Рисунок 31 — Противники

Слизь может атаковать только при контакте с игроком. Скелет наносит урон в радиусе атаки. Из поверженных противников «выпадают» монеты, энергия, на подводном уровне к получаемым ресурсам добавляются сферы воздуха.

3.5 Освещение, шейдеры и постобработка

Для улучшения визуальной составляющей в проект был интегрирован пакет Universal Render Pipeline.

3.5.1 Освещение

На четвертом уровне вся сцена затемнена. На персонажа помещен источник света. Также в зонах ямы размещены источники света красного цвета. Размещенные источники света (Рисунок 32).

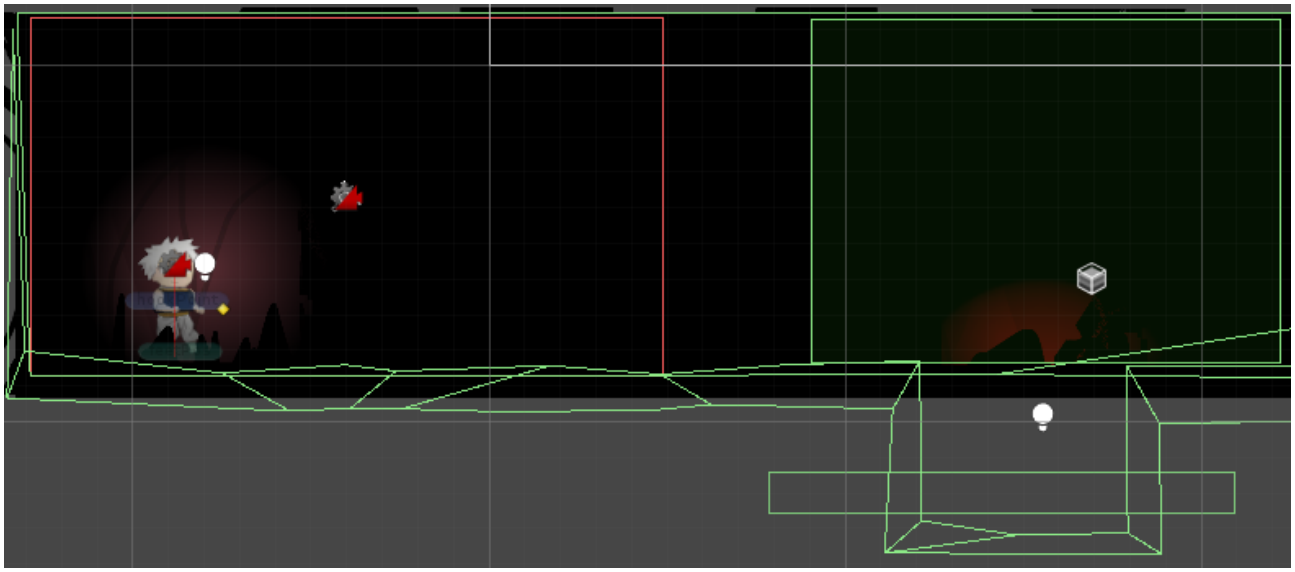

Рисунок 32 — Размещенные источники света

Свет и тени могут значительно нагружать GPU. Использовав меньшее количество источников света и уменьшив качество теней, можно ускорить производительность.

3.5.2 Шейдер портала

Объектом, сигнализирующем, что персонаж достиг конца уровня является анимированных портал, сознанный при помощи Shader Graph. Вместо написания кода шейдер реализуется за счет соединения различных узлов в структуре графа [7]. Реализация шейдера портала (Рисунок 33).

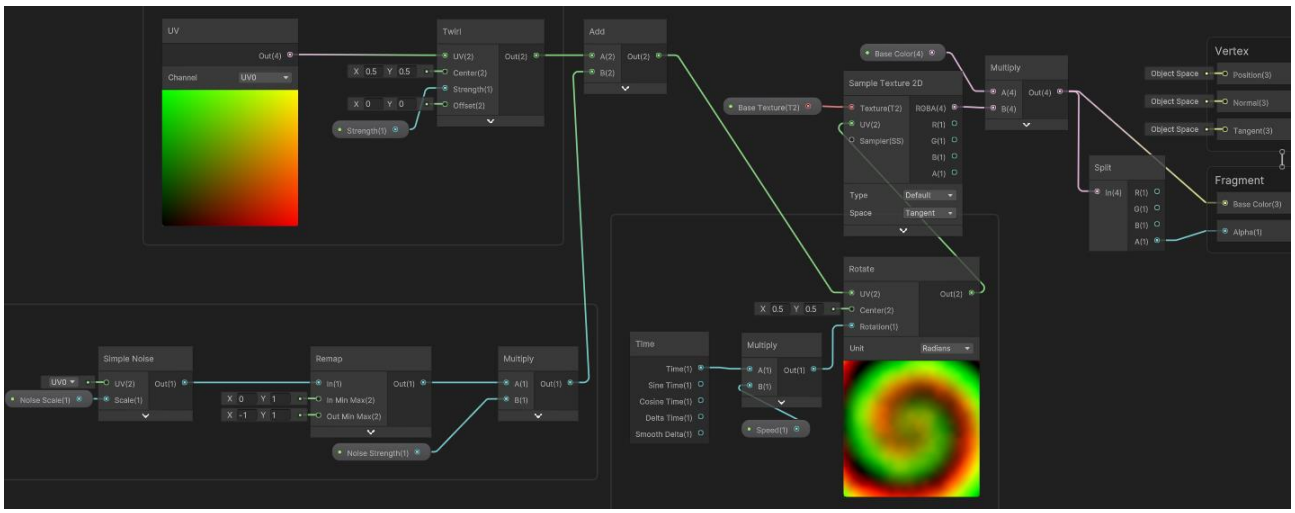

Рисунок 33 — Реализация шейдера портала

Шейдер портала (Рисунок 34).

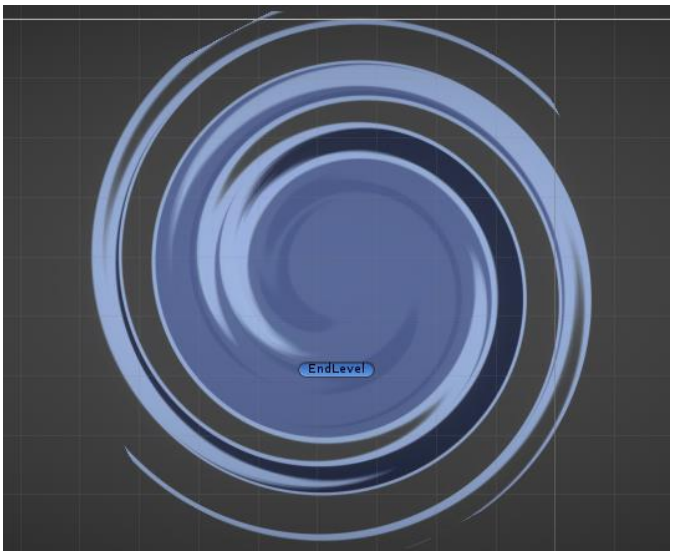

Рисунок 34 — Шейдер портала

3.5.3 Постобработка

Для всех уровней на камеру (Main Camera) наложены такие эффекты как:

– vignette: снижение яркости изображения по краям камеры;

– bloom: эффект размытости.

Настройка эффектов камеры (Рисунок 35).

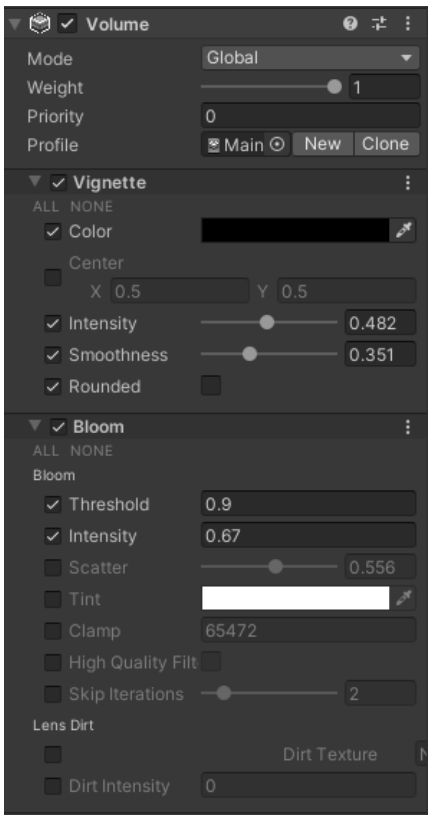

Рисунок 35 — Настройка эффектов камеры

На втором и четвертом уровнях в зонах, где игрок может упасть и как следствие проиграть размещены Box Volume. Box Volume – это объекты на сцене, заходя в зону (определяется с помощью Box Collider) которых изменяется графический эффект. Пример размещения Box Volume и его настройка (Рисунок 36).

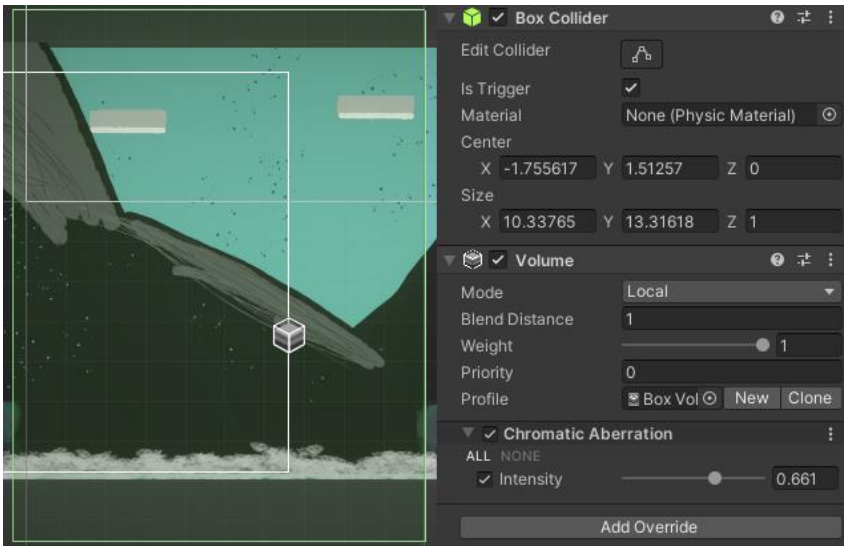

Рисунок 36 — Пример размещения Box Volume и его настройка

На втором уровне это эффект хроматической аберрации, на четвертом – изменение баланса белого цвета.

Частицы и другие эффекты могут нагружать GPU. При меньшем количестве эффектов и частиц можно ускорить производительность.

3.6 Музыкальное сопровождение

За музыкальное сопровождение отвечает класс MusicPlayer, в котором находится логика воспроизведения фоновой музыки и музыкальных эффектов.

Настройка громкости музыки и эффектов реализована за счет слайдеров в меню паузы, значения которых загружаются и сохраняются с помощью PlayerPrefs.

Для каждого уровня предусмотрена своя фоновая музыка. Присвоены музыкальные эффекты для прыжка, атаки и нажатия кнопок на игровых сценах. [9].

3.7 Адаптация игры для телефона

Так как Unity является кроссплатформенным движком, портирование игры не вызывает никаких сложностей, необходимо адаптировать управление и изменить настройки сборки проекта.

Передвижение и прыжок игрока реализованы за счет Joystick Pack, интегрированного в проект с Unity Asset Store [8]. Также в классе PlayerController изменена переменная moveInput для передвижения и создана verticalMove для считывания прыжка, когда джойстик находится в верхнем положении.

Созданы кнопки: пауза, атака и атака заморозкой. Код не изменялся, так как активация метода вызывается из настроек кнопки в инспекторе. Пример вызова метода по нажатию кнопки (Рисунок 37).

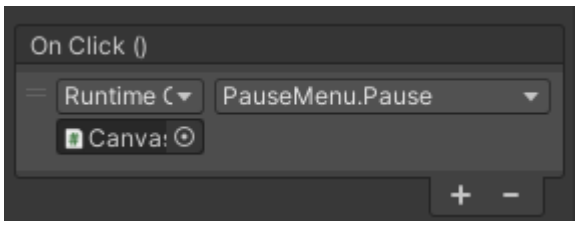

Рисунок 37 — Вызов метода по нажатию кнопки

Для второго уровня изменен способ активации крюка-кошки. В методе Update заменен участок кода, отвечающий за считывание нажатия правой кнопки мыши на прикосновение. Если это новое прикосновение (TouchPhase.Began) и нет активного прикосновения, то создается крюк с помощью метода CreateHook(), передавая ему позицию прикосновения. Если прикосновение закончилось (TouchPhase.Ended) или было отменено (TouchPhase.Canceled), и прикосновение соответствует активному прикосновению, то крюк отключается с помощью метода DisableHook(). В методе CreateHook() переменная target получает вместо mousePosition touchPosition.

Интерфейс управления с телефона (Рисунок 38).

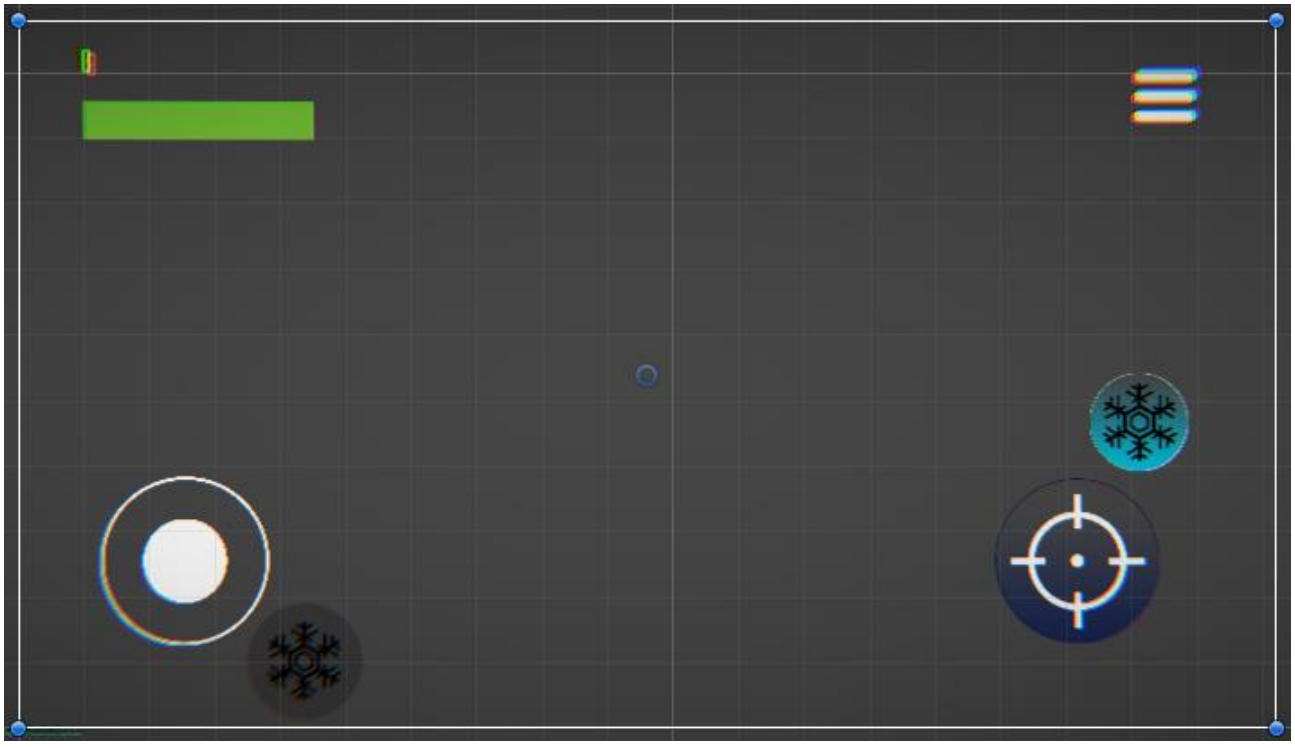

Рисунок 38 — Интерфейс управления с телефона

3.8 Созданные префабы

Prefabs в Unity — это предварительно созданные объекты, которые можно использовать в нескольких местах в вашей игре. Их можно сравнить с шаблонами или макетами, которые можно повторно использовать для создания однотипных объектов. Преимущества использования префабов в Unity:

– эффективность разработки: создание объектов вручную для каждой сцены или уровня может быть трудоемким и затратным по времени. Использование префабов позволяет значительно ускорить процесс разработки, поскольку они могут быть созданы заранее и повторно использованы в разных частях игры.

– легкость изменения: изменения, сделанные в префабе, автоматически применяются ко всем объектам, созданным на его основе. Это означает, что можно легко изменять, обновлять и дорабатывать префабы, чтобы улучшить игровой процесс или внешний вид игры.

– улучшенная масштабируемость: с помощью префабов вы можете легко создавать множество объектов с одинаковыми параметрами и функциональностью.

Список созданных префабов:

– skeleton (противник);

– slime (противник);

– coin (ресурс);

– energy (ресурс);

– airSphere (ресурс);

– bullet (атака игрока);

– freezBullet (замораживающая атака игрока);

– particleEffect’s (визуальные эффекты).

3.9 Реализация клиент-серверного взаимодействия

В проект был интегрирован LootLockerSDK, который обеспечивает функционал в сфере клиент-серверного взаимодействия. В настройках проекта Unity указаны ApiKey и DomainKey, сгенерированные в личном кабинете на сайте LootLocker. Вся информация по работе с сервисом была получена с официальной документации [5].

Сконфигурирована система опыта (Рисунок 39).

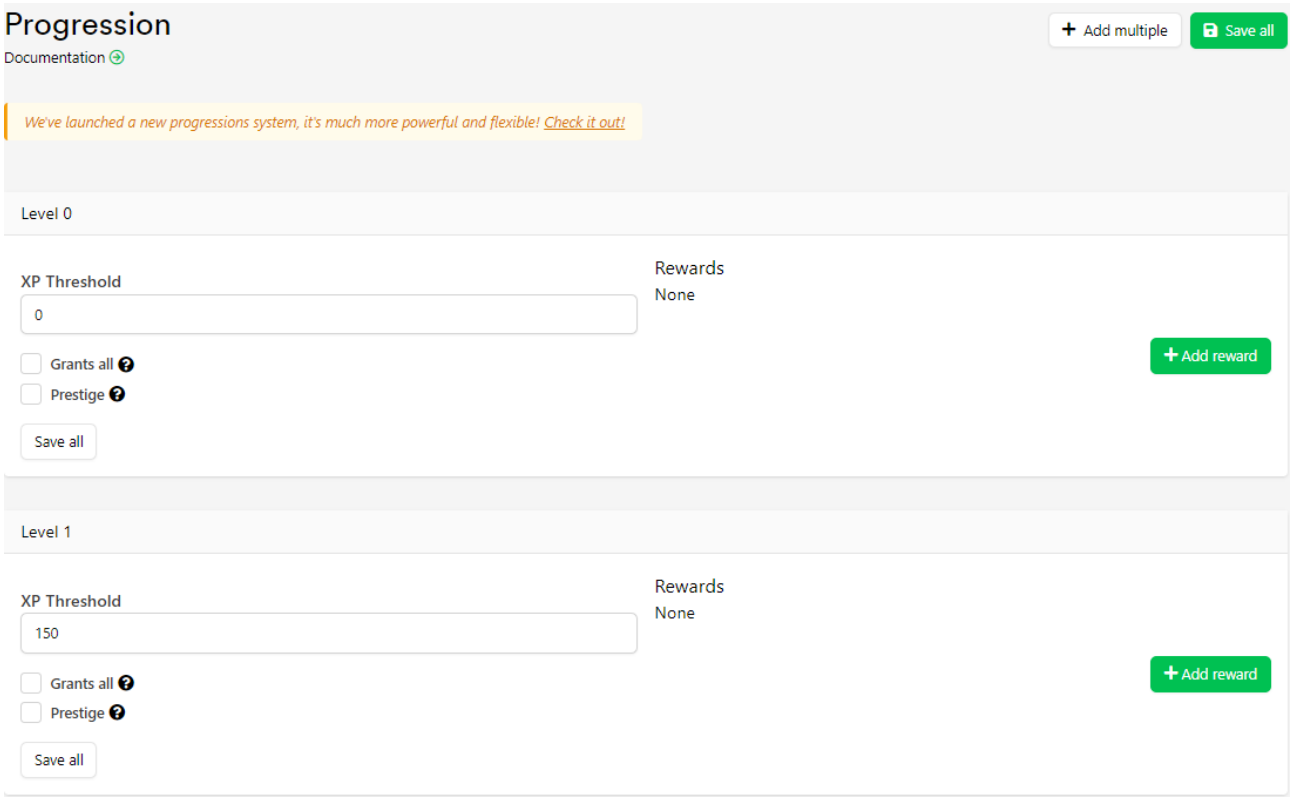

Рисунок 39 — Система опыта

Созданы таблицы лидеров для четырех уровней (Рисунок 40).

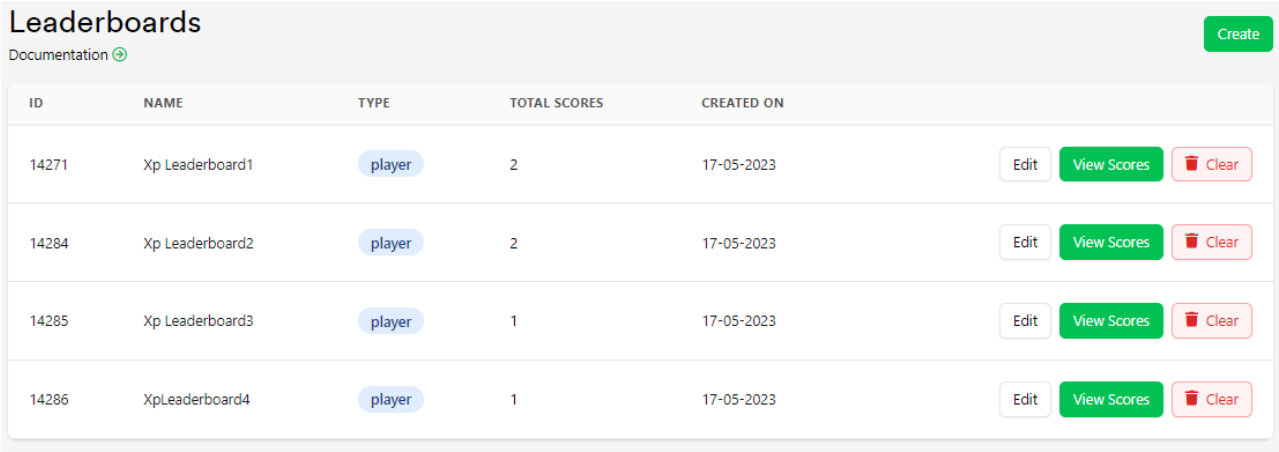

Рисунок 40 — Таблицы лидеров для четырех уровней

Созданы награды, получаемые при повышении уровня (Рисунок 41).

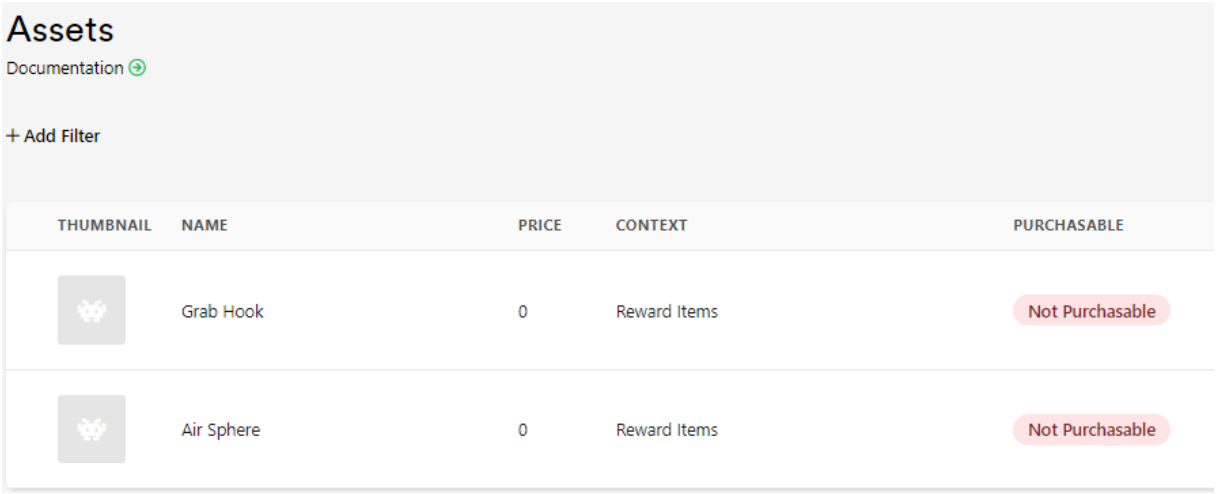

Рисунок 41 — Список наград

3.10 Выводы по третьей главе

В ходе выполнения третьей главы выпускной квалификационной работы были решены такие задачи как:

– реализация игры: созданы все необходимые сцены и интерфейсы, вся визуальная составляющая интегрирована в проект, объектам назначены необходимые компоненты, клиент-серверное взаимодействие интегрировано в проект;

– адаптация игры для телефонов;

– тестирование и отладка игры.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы были выполнены следующие этапы:

– обзор игрового движка Unity;

– анализ архитектур для создания игр;

– обзор клиент-серверных решений;

– разработка концепта игры;

– составление задания на проектирование и разработку;

– проектирование игры;

– отрисовка дизайна;

– реализация игры;

– адаптация игры для телефонов;

– тестирование и отладка игры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Stopgame.ru: русскоязычный интернет-ресурс о компьютерных играх [Электронный ресурс]. — Режим доступа: https://stopgame.ru/newsdata/56350/2022_y_v_cifrah_samye_prodavaemye_igry_i_rost_pk_geyminga (дата обращения: 24.11.2022).

2. Игровой движок Unity: официальный сайт [Электронный ресурс]. — Режим доступа: https://unity.com/ru (дата обращения: 27.11.2022).

3. Dtf.ru: русскоязычный интернет-ресурс о компьютерных играх [Электронный ресурс]. — Режим доступа: https://dtf.ru (дата обращения: 29.11.2022).

4. Firebase: сервис баз данных в режиме реального времени [Электронный ресурс]. — Режим доступа: https://firebase.google.com (дата обращения: 09.12.2022).

5. LootLocker: официальный сайт [Электронный ресурс]. — Режим доступа: https://lootlocker.com (дата обращения: 12.12.2022).

6. UnityDiscussions: статья посвященная параллакс эффекту [Электронный ресурс]. — Режим доступа: https://discussions.unity.com/t/parallax-scrolling-using-orthographic-camera/83699 (дата обращения: 18.03.2023).

7. UnityManualShaderGraph: официальная документация [Электронный ресурс]. — Режим доступа: https://docs.unity3d.com/Packages/com.unity.shadergraph@16.0/manual/index.html (дата обращения: 28.04.2023).

8. UnityAssetStore: магазин дополнительных инструментов для проектов Unity [Электронный ресурс]. — Режим доступа: https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631 (дата обращения: 25.05.2023).

9. Zvukipro: интернет ресурс с музыкой и звуками [Электронный ресурс]. — Режим доступа: https://zvukipro.com (дата обращения: 10.06.2023).

ПРИЛОЖЕНИЕ А

Описание методов классов

Таблица А.1 – Описание методов классов

Название класса	
Название метода	**Описание метода**
StartPageManager	
ChooseAction()	Отвечает за выбор между регистрацией и авторизацией. Если значение слайдера равно 0, то активируется кнопка для создания нового пользователя, а кнопка для авторизации скрывается, и наоборот, если значение слайдера равно 1.
SingUp()	Отправляет запрос на сервер для создания нового пользователя. Для этого используются данные, введенные в поля ввода адреса электронной почты и пароля, при положительном ответе от сервера вызывается метод EmailVerification() и на почту игрока приходит письмо с подтверждением регистрации.
LogIn()	Отправляет запрос на сервер для авторизации пользователя. При успешной авторизации вызывается метод StartSession(), который инициализирует сессию.
StartSession()	Запускает сессию на сервере, и если она успешно создана, то вызываются методы ReadyToPlay() и GetPlayerName().
ReadyToPlay()	Скрывает поля ввода адреса электронной почты и пароля, кнопку для авторизации, слайдер, и активирует поле для ввода имени пользователя, кнопку для подтверждения имени и кнопку для входа в меню игры.
SetName()	Отправляет запрос на сервер для установки имени пользователя и при успешном ответе вызывает метод GetPlayerName().
GetPlayerName()	Отправляет запрос на сервер для получения имени пользователя и при успешном ответе отображает его в текстовом поле.
PlayGame()	Загружает следующую сцену с игровым меню.
MenuManager	
Start()	Вызывает все методы в классе.
CheckLevel()	Получает информацию о текущем уровне игрока, а затем обновляет текстовые поля CurrentLevelText, NextLevelText и CurrentXpText в соответствии с полученной информацией. Кроме того, он обновляет значение LevelSlider, который представляет текущий уровень игрока. Если текущий опыт равен текущему порогу уровня, то максимальное значение LevelSlider устанавливается на следующий порог уровня, а текущее значение LevelSlider устанавливается на 0. В противном случае, текущее значение LevelSlider вычисляется как разница между текущим опытом и текущим порогом уровня.
GetPlayerName()	Получает имя пользователя от сервера и выводит информацию на экран.

Продолжение приложения А

Название класса	
Название метода	**Описание метода**
MenuManager (продолжение)	
LevelAccess()	Используется для определения доступных уровней игрока и установки видимости соответствующих элементов интерфейса. Например, SetLevelVisibility() показывает или скрывает определенные игровые уровни на основе текущего уровня игрока, а SetNoLevelVisibility() показывает или скрывает элементы интерфейса для доступа к уровням, которые игрок еще не достиг.
ShowLeaderboards()	Вызывается при изменении значения ползунка sliderLeaderBoard и используется для отображения соответствующего списка лучших результатов игроков на основе выбранного значения.
GetLeaderboard()	Получает список лучших результатов игроков для определенного ключа доски лидеров. Он обрабатывает полученный ответ, заполняет текстовые поля интерфейса соответствующими результатами и обрабатывает ситуацию, когда количество результатов меньше максимально допустимого значения.
PlayerInventory()	Получает информацию об инвентаре игрока. Он обрабатывает полученный ответ, создает элементы интерфейса для каждого предмета в инвентаре и загружает их иконки из указанных URL.
LoadTexture()	Используется для асинхронной загрузки текстуры по указанному URL и установки ее в качестве иконки для соответствующего элемента интерфейса.
Level(),Level2(), Level3() и Level4()	Вызывают функцию LoadScene() из SceneManager, чтобы загрузить соответствующий игровой уровень при нажатии на соответствующие кнопки интерфейса.
PlayerController	
Start()	Start() метод, вызываемый при старте игры. Он инициализирует Rigidbody2D и Animator для игрока.
FixedUpdate()	Метод, вызываемый в каждом фиксированном кадре игры. Он определяет перемещение игрока по горизонтали, используя Input.GetAxis("Horizontal"), устанавливает соответствующую скорость Rigidbody2D, а также проигрывает анимацию бега или остановки.
Update()	Метод, вызываемый каждый кадр игры. Он проверяет, касается ли игрок земли с помощью Physics2D.OverlapCircle() и происходит ли нажатие на клавишу Space, то устанавливает скорость Rigidbody2D в направлении вверх с помощью произведения значений Vector2.up на jumpForce и воспроизводит анимацию прыжка.
Flip()	Метод, который переворачивает игрока в зависимости от направления его движения.

Продолжение приложения А

Название класса	
Название метода	**Описание метода**
Health	
Start()	Метод, вызываемый при старте игры. Он инициализирует текущее здоровье объекта значением максимального здоровья.
ChangedHealth()	Метод, который изменяет значение текущего здоровья объекта и вызывает событие HealthChanged, чтобы уведомить другие компоненты об изменении здоровья.
AttackSystem	
Update()	Обновляет состояние пули. Он использует Physics2D.Raycast для проверки столкновения снаряда с преградами. Если столкновение произошло и преграда имеет тег "Enemy", вызывается метод Attack(). После этого сам снаряд уничтожается.
Attack()	Абстрактный метод не имеет реализации в этом классе. Он ожидает, что его реализуют подклассы SimpleBullet и FreezBullet.
SimpleBullet	
Update()	Переопределенный метод вызывает родительскую реализацию Update().
Attack()	Переопределенный метод вызывает метод TakeDamageToEnemy() у компонента Enemy, связанного с коллайдером, с которым столкнулся снаряд. Он передает значение damage для нанесения обычного урона.
FreezBullet	
Update()	Переопределенный метод вызывает родительскую реализацию Update().
Attack()	Переопределенный метод вызывает метод FreezDamageToEnemy() у компонента Enemy, связанного с коллайдером, с которым столкнулся снаряд. Он передает значение damage для нанесения замораживающего урона.
TimerController	
Start()	Вызывается при старте объекта и запускает таймер с помощью метода StartTimer().
Update()	Вызывается на каждом кадре и обновляет значение таймера с помощью метода UpdateTimer().
UpdateTimer()	Обновляет значение таймера при условии, что флаг isTimerRunning установлен в true. Значение таймера увеличивается на Time.deltaTime, что представляет время, прошедшее с предыдущего кадра.
GetTimerValue()	Возвращает текущее значение таймера.
StartTimer()	Запускает таймер путем установки флага isTimerRunning в true.
StopTimer()	Останавливает таймер путем установки флага isTimerRunning в false.
SpawnAttributes	
Start()	Вызывается при старте объекта и ищет объект с тегом "AirField" для дальнейшего использования.
SpawnRes(Vector3 position)	Метод для создания ресурсов. Создает заданное количество объектов каждого типа ресурса на указанной позиции.

Продолжение приложения А

Название класса	
Название метода	**Описание метода**
SpawnAttributes (продолжение)	
SpawnObject()	Создает объект ресурса на указанной позиции. Задает случайное направление и скорость движения объекта ресурса. Также добавляет компонент Destroy с заданным временем жизни для объекта ресурса.
Collector	
RegisterCollector()	Метод для регистрации объекта сборщика ресурсов. Добавляет объект в список collectors.
OnTriggerEnter2D()	вызывается при соприкосновении коллайдера с другим коллайдером. Проверяет, является ли столкнувшийся объект сборщиком ресурсов (IResourceCollector). Если да, вызывает метод CollectResource() у сборщика ресурсов, передавая тип ресурса, который был задан через тег объекта. ResourceType.
GetResourceType()	Возвращает тип ресурса на основе тега объекта.
EnergyScore	
Start()	Устанавливает начальное значение currentEnergy равное startEnergy, которое по умолчанию равно 0.
EnegryAdd()	Увеличивает текущую энергию currentEnergy на указанное значение energyAmount. Если текущая энергия превышает максимальное значение maxEnergy, то она устанавливается равной maxEnergy. Затем обновляется заполнение полоски энергии energyBarFilling в соответствии с текущим значением энергии.
EnergyMinus	Уменьшает текущую энергию currentEnergy на указанное значение energyAmount. Затем обновляется заполнение полоски энергии energyBarFilling в соответствии с текущим значением энергии.
CoinScore	
Start()	В методе происходит инициализация. Компонент Text на сцене получается с помощью GetComponent<Text>(), и затем обновляется текст счетчика на основе начального значения score. Текст счетчика форматируется в виде строки "Score: [значение счетчика]".
AddScore()	Увеличивает значение счетчика score на заданную величину value. Затем обновляется текст счетчика coinScore.text с новым значением счетчика.
Enemy	
Start()	Вызывается при старте объекта. В нем инициализируются необходимые переменные, такие как скорость, ссылка на игрока (target), а также устанавливается обработчик события OnDeath из компонента HealthEnemy. Этот метод также используется для подписки на событие OnDeath путем добавления метода Death() в список обработчиков.
FixedUpdate()	Отвечает за выполнение основного поведения противника. Внутри метода проверяется состояние "заморозки" (isFreez) противника и обновляется таймер заморозки. Затем вызывается метод CheckDistanceToFollowPlayer().

Продолжение приложения А

Название класса	
Название метода	**Описание метода**
Enemy (продолжение)	
CheckDistanceToFollowPlayer()	Проверяет расстояние между противником и игроком. Если расстояние меньше заданного значения distanceToDetected, то переменная followIsTrigered устанавливается в true, что указывает на необходимость следования за игроком. Если расстояние между противником и игроком также меньше значения distanceToAttack, то вызывается метод EnemyAttack() для атаки игрока.
Patrol()	Отвечает за патрулирование территории. Противник перемещается влево и вправо в заданном диапазоне патруля (positionOfPatrol) относительно точки дома противника (pointOfHomeEnemy). При достижении крайних точек патруля происходит разворот противника.
FollowPlayer()	Отвечает преследование игрока. Противник перемещается к игроку с помощью Vector2.MoveTowards, а также происходит разворот противника, если игрок находится слева или справа от противника.
TakeDamageToEnemy(int damage)	Принимает значение урона и передающий его в компонент HealthEnemy. Также создается эффект повреждения (damageEffect), который инстанцируется в позиции противника.
FreezDamageToEnemy(int damage)	Вызывается при получении заморозки. В свою очередь он вызывает метод TakeDamageToEnemy(int damage).
TakeDamageToEnemy(int damage)	Вызывается для нанесения урона и устанавливает скорость противника (speed) в значение заморозки (speedFrozen);
Death()	Вызывается при уничтожении противника. Внутри метода создается экземпляр класса SpawnAttributes и вызывается его метод SpawnRes().Затем метод удаляется из списка обработчиков события OnDeath и сам объект противника уничтожается.
SpawnRes()	Метод для создания ресурсов на позиции противника
Flip()	Отвечает за поворот противника. Он изменяет флаг facingRight, отражает объект по горизонтали и поворачивает его на 180 градусов.
Slime	
EnemyAttack()	Метод, отвечающий за атаку.
Start()	Вызывает базовой метод Start() из класса Enemy
FixedUpdate()	Вызывает базовый метод FixedUpdate() из класса Enemy

Продолжение приложения А

Название класса	
Название метода	**Описание метода**
Skeleton	
EnemyAttack()	Метод, отвечающий за атаку.
Start()	Вызывает базовой метод Start() из класса Enemy
FixedUpdate()	Вызывает базовый метод FixedUpdate() из класса Enemy
PauseMenu	
Update()	Update() обновляет состояние игры при нажатии клавиши Escape. Если игра находится на паузе, вызывается метод Resume(), иначе вызывается метод Pause();
Resume()	Возобновляет игру после паузы. Отключает интерфейс меню паузы (pauseMenuUI.SetActive(false)), возвращает нормальное время проигрывания (Time.timeScale = 1f) и устанавливает флаг gameIsPaused в значение false.
Pause()	Приостанавливает игру, вызывая паузу. Он включает интерфейс меню паузы (pauseMenuUI.SetActive(true)), устанавливает время проигрывания на 0, что останавливает все движение в игре (Time.timeScale = 0f), и устанавливает флаг gameIsPaused в значение true.
LoadMenu()	Загружает главное меню игры. Возвращается нормальное время проигрывания (Time.timeScale = 1f), и вызывается метод LoadScene() для загрузки сцены "GameMenu".
QuitGame()	Завершает работу приложения. Вызывается метод Quit() из класса Application, который завершает выполнение игры.
GameOverMenu	
Start()	Ищет объект с компонентом Health на сцене и подписывается на событие GameOver этого объекта. Если объект Health найден, метод ShowGameOverUI будет вызываться при наступлении события GameOver;
ShowGameOverUI()	Отображает интерфейс меню проигрыша. Он активирует объект gameOverUI, который содержит интерфейс проигрыша, и устанавливает время проигрывания на 0, что останавливает все движение в игре.
RestartLevel()	Перезапускает текущий уровень. Он загружает текущую активную сцену, что приводит к перезапуску уровня. Затем он возвращает нормальное время проигрывания (Time.timeScale = 1f), чтобы игра продолжилась.
ReturnToMainMenu()	Возвращает к главному меню игры. Он загружает сцену "GameMenu", переключая игру на главное меню.

Продолжение приложения А

Название класса	
Название метода	**Описание метода**
EndLevel	
OnTriggerEnter2D()	Вызывается, когда объект соприкасается с коллайдером, на котором данный скрипт находится. Активируется объект canvasEndLevel, который содержит интерфейс окончания уровня; устанавливается время игры в 0, останавливая все движение в игре (Time.timeScale = 0f).
StopTimer()	Останавливает таймер (timerController.StopTimer()).
CalculateExperience()	Вычисляет опыт игрока на основе его времени прохождения уровня и количества собранных монет. Значение таймера получается с помощью метода GetTimerValue() из timerController. Вычисляется обратное значение времени и умножается на 1000 для получения опыта (inverseTime = (100f / timerValue) * 1000f). Буферное значение количества собранных монет получается из coinScore.score. Итоговый опыт вычисляется путем суммирования буферного значения монет и обратного значения времени. Обновляется текст в интерфейсе, отображающий итоговый опыт игрока.
SubmitScore()	Отправляет результаты игрока на таблицу лидеров текущего уровня и обновляет прогресс опыта. Для этого используется LootLockerSDKManager.SubmitScore() и LootLockerSDKManager.SubmitXp().
FrameSwitch	
OnTriggerEnter2D()	Проверяет, если объект, сталкивающийся с коллайдером, имеет тег "Player", то активируется объект activeFrame. Это означает, что при столкновении игрока с коллайдером, устанавливается активное состояние для activeFrame.
OnTriggerExit2D	Проверяет, если объект, покидающий коллайдер, имеет тег "Player", то деактивируется объект activeFrame. Это означает, что при выходе игрока из коллайдера, устанавливается неактивное состояние для activeFrame.
AirField	
Start()	Устанавливает начальное количество воздуха airCurrent равным максимальному airMax, а также получается компонент Health у игрока с тегом "Player".
FixedUpdate()	Уменьшает количества воздуха airCurrent с течением времени. Если значение airCurrent становится меньше или равным 0 и компонент Health не равен null, вызывается метод TakeDamage у игрока с передачей значения damageAmount для нанесения урона.
AirAdd()	Используется для добавления воздуха в хранилище. Он увеличивает значение airCurrent на указанное количество airAmount.
GrabHook	
Update()	Проверяет нажатия и отпускания кнопки мыши. При нажатии кнопки создается крюк с помощью метода CreateHook(), а при отпускании кнопки крюк отключается с помощью метода DisableHook(). Если крюк активен, то вызывается метод hookRender.Render().

Окончание приложения А

Название класса	
Название метода	**Описание метода**
GrabHook (продолжение)	
Update()	Проверяет нажатия и отпускания кнопки мыши. При нажатии кнопки создается крюк с помощью метода CreateHook(), а при отпускании кнопки крюк отключается с помощью метода DisableHook(). Если крюк активен, то вызывается метод hookRender.Render().
DisableHook()	Отключает крюк путем отключения компонента DistanceJoint2D и вызова метода hookRender.Disable(), чтобы отключить отображение линии.
CreateHook()	Создает крюк путем выпуска луча из начальной точки hookPoint в направлении указателя мыши и проверяет столкновение с объектами на пути луча. Если столкновение происходит, то активируется компонент DistanceJoint2D и устанавливается точка захвата для крюка.
hookRender.Render()	Метод для отображения линии между начальной точкой крюка и точкой захвата.
HookRayCast	
Raycast()	Выпуская луч из указанной начальной точки origin в направлении цели target на заданное расстояние distance. Возвращается информация о столкновении в виде RaycastHit2D, включая информацию о столкнувшемся объекте и точке столкновения.
HookRender	
Render()	Включает отображение линии LineRenderer и устанавливает начальную и конечную точки линии в соответствии с заданными значениями startPoint и endPoint.
Disable()	Отключает отображение линии путем отключения компонента LineRenderer.

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
"19" 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

Разработка игры на Unity «Lost Soul»

Руководитель _____ 19.06.2023г. старший преподаватель Т.С. Титовская
подпись дата должность, ученая степень

Выпускник _____ 19.06.2023г. Д.В. Сбитнев
подпись дата

Нормоконтролёр _____ 19.06.2023г. Т.С. Титовская
подпись дата

Красноярск 2023