МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2021 г.

Создание изометрической кооперативной онлайн-игры для мобильных платформ в жанре Hyper-Casual на основе Voxel Graphics

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____В.А. Парасич
«___»_____2021 г.

Автор работы,
студент группы КЭ-405
_____А.С. Сухарев
«___»_____2021 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2021 г.

Челябинск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2021 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Сухареву Антону Сергеевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Создание изометрической кооперативной онлайн-игры для мобильных платформ в жанре Hyper-Casual на основе Voxel Graphics» утверждена приказом по университету от 26 апреля 2021 года №714-13/12 (приложение №73).
2. **Срок сдачи студентом законченной работы:** 1 июня 2021 г.
3. **Исходные данные к работе:**

- Игра для мобильных платформ Android и IOS
- Кооперативный онлайн-геймплей
- Hyper-Casual жанр
- «Воксельная» графика
- Широкий перечень поддерживаемых мобильных устройств

4. **Перечень подлежащих разработке вопросов:**

- Анализ требований и постановка задачи

- Анализ существующих решений
- Составление технического задания
- Выбор инструментария
- Разработка программной архитектуры будущей игры
- Программная реализация
- Тестирование работоспособности и устранение ошибок
- Размещение игры в магазине мобильных приложений

5. **Дата выдачи задания:** 1 декабря 2020 г.

Руководитель работы _____/*В.А. Парасич*/

Студент _____/*А.С. Сухарев*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2021	
Разработка модели, проектирование	01.04.2021	
Реализация системы	01.05.2021	
Тестирование, отладка, эксперименты	15.05.2021	
Компоновка текста работы и сдача на нормоконтроль	24.05.2021	
Подготовка презентации и доклада	30.05.2021	

Руководитель работы _____*/В.А. Парасич/*

Студент _____*/А.С. Сухарев/*

АННОТАЦИЯ

А.С. Сухарев. Создание изометрической кооперативной онлайн-игры для мобильных платформ в жанре Hyper-Casual на основе Voxel Graphics. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2021, 81с., 29 ил., библиогр. список – 24 наим.

В рамках выпускной квалификационной работы производится разработка мобильной игры в популярном жанре Hyper-Casual. Для возможности кооперативной игры производится создание серверного приложения с доступом по API. В рамках гейм-дизайн и левел-дизайн частей игры рассматриваются проблемы современных Hyper-Casual игр с точки зрения уникальности геймплея и создания особого игрового опыта.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Все чаще игры для мобильных платформ обретают всеобщую популярность, сравнимую с играми схожих жанров для персональных компьютеров. Популярность обусловлена портативностью и повсеместностью смартфонов, почти каждый в наше время имеет собственный компьютер в кармане.

Современные смартфоны обладают внушительными вычислительными способностями, предоставляя разработчикам использовать в играх красивую и впечатляющую графику. Скорость мобильного интернета позволяет делать частые запросы на удаленные сервера и получать оперативные ответы, что способствует созданию онлайн, а также кооперативных игр. Компактность смартфонов позволяет людям играть в мобильные игры в любом удобном для них месте.

Создание онлайн игры с использованием Voxel Graphic имеет возможность разрушения/установку блоков игрового мира (Voxel), что интересно людям, знакомым с Minecraft и прочими аналогами. Также с помощью «вокселей» можно генерировать обширные и красивые интерактивные миры.

Hyper-Casual жанр стал открытием последних лет в игровой индустрии. Сочетая в себе простые механики и способы взаимодействия с игрой, такие игры не требуют от игрока наличия прошлого игрового опыта, обучая его по мере череды проигрышей и побед. На рисунке 2 представлен распространенный граф взаимодействия с игрой в endless и semi-endless Hyper-Casual проектах.

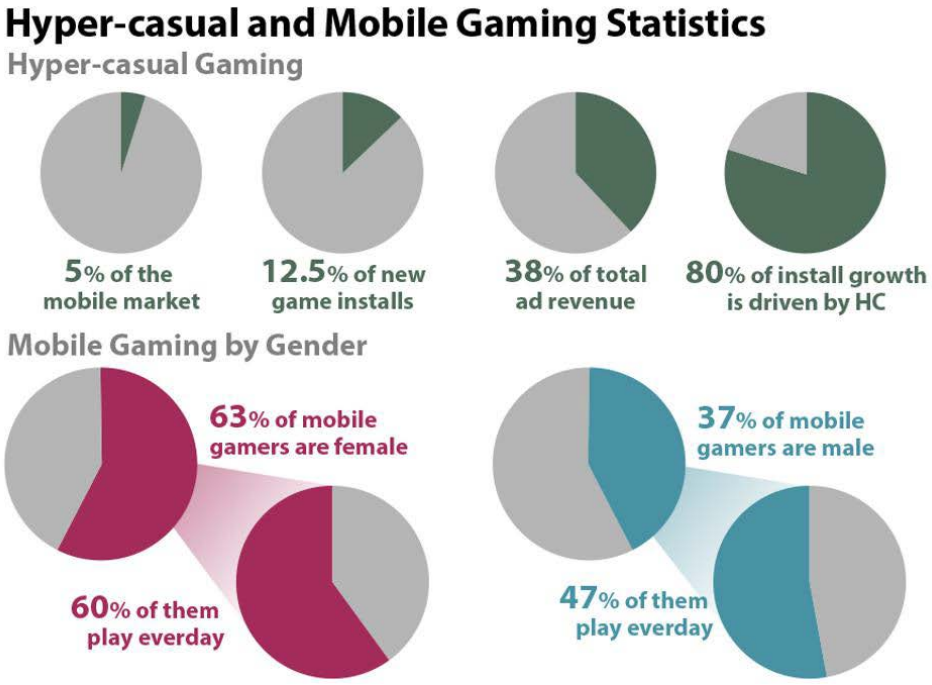

Рисунок 1 - Статистика Hyper-Casual игр

Такие игры способны привлекать к себе «неиграющую» аудиторию, т.е. людей, ранее практически или совсем не играющих в компьютерные игры.

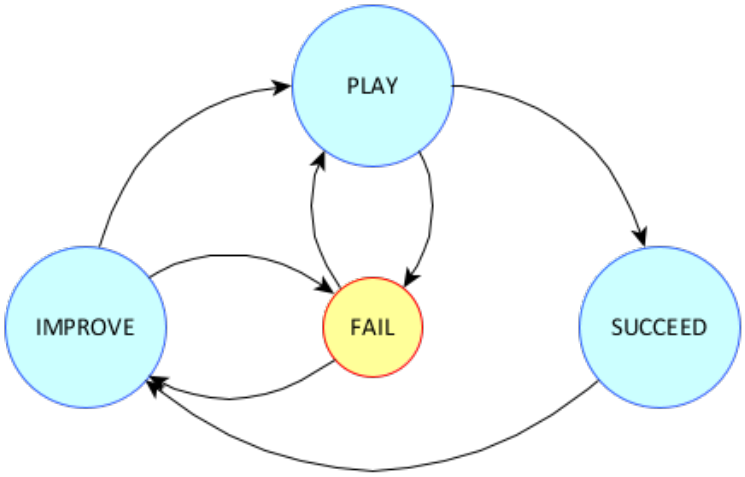

Рисунок 2 - Цикл с обратной связью Hyper-Casual игр

При выполнении работы для получения готового продукта нужно:

1. Выполнить анализ имеющихся игр в данной нише.
2. Проанализировать технологии, позволяющие создать данную игру.

3. Разработать архитектуру проекта с учетом серверной и клиентской частей.
4. Провести анализ полученных результатов, при выявлении каких-либо недостатков - попытаться исправить.

Структура данной выпускной работы, следующая:

1. Анализ предметной области. Подразумевает под собой анализ уже представленных решений и технологий.
2. Определение требований. Необходимо выделить функциональные требования.
3. Проектирование. Провести проектирование всех модулей разрабатываемой системы: серверное ПО, пользовательский интерфейс.
4. Реализация системы.
5. Тестирование полученных результатов.
6. Заключение. Подведение итогов проделанной работы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 АКТУАЛЬНОСТЬ ТЕМЫ

В мобильные игры играет большое количество людей, многие из которых не являются геймерами по своей сути. Играть в игры на телефоны для многих стало обыденностью, необходимостью. Создавая игру – ты создаешь продукт, на который получишь моментальную обратную связь. В одиночной мобильной разработке можно принимать самые смелые решения по поводу игровых механик, архитектурных решений. Создать `Hyper-Casual` игру можно не имея ни гроша в кармане, при этом не тратя годы на создания прототипов.

1.2 АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

`1. Gumslinger`

Мультиплеерная игра в популярном жанре `Battle Royale`, в которой необходимо сражаться против 63 человек за смешно двигающегося человечка.

Игра напоминает .io игру, так как при отсутствии интернета игрок может продолжать играть против ботов. Особенность в том, что игровой процесс не отличается от игры против реальных игроков, создавая впечатление неразрывного геймплея.

В одном раунде ты перестреливаешься с одним из игроков. По его итогу дальше в сетке продвигается только один игрок. Содержит кастомизацию персонажа, ежедневные миссии, соревнования и внутриигровые события.

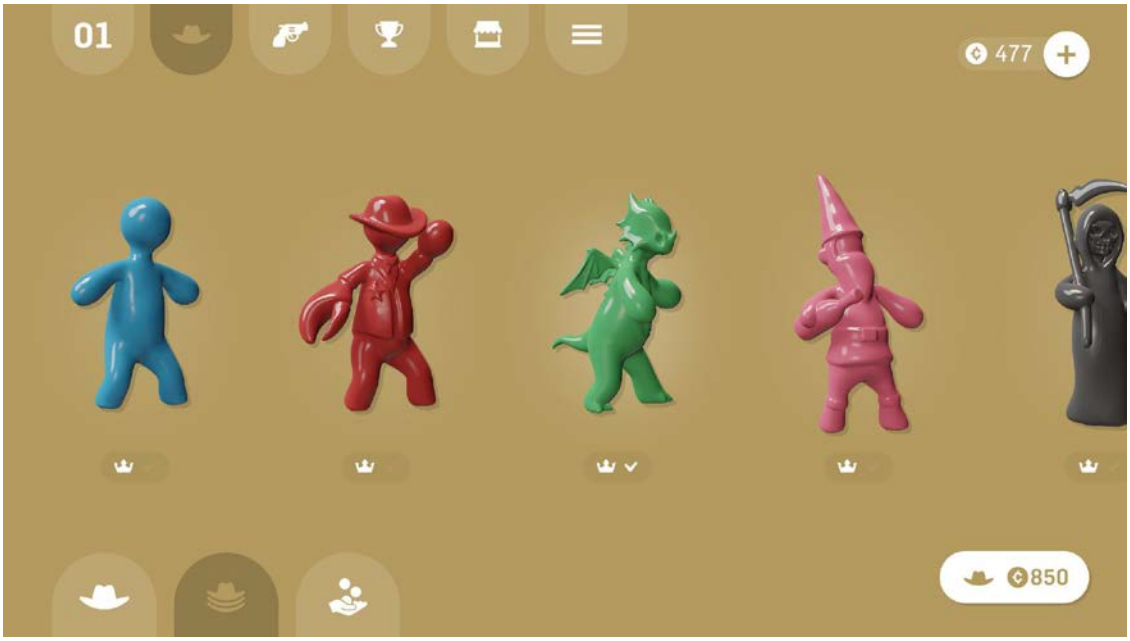

Рисунок 3 - Интерфейс игры Gumslinger

2. Bumper.io

3D мобильная игра .io игра, суть которой в том, чтобы вытолкнуть других игроков с поля. Победителем считается тот, кто последним остался на поле. Так же как и в других .io играх, возможна игра без подключения к сети интернет.

1.3 АНАЛИЗ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

Для создания продукта необходимо определиться со следующими технологическими решениями:

1. Среда разработки для создания и проектирования игры.
2. Выбор средств аналитики для отслеживания внутриигровых действий игрока.
3. Сервис для хостинга серверного клиента игры.
4. Инструмент для создания Мультиплеера для выбранного движка.
5. Инструмент для создания выделенного сервера, который будет принимать запросы от игроков и создавать instance сервера на хосте для матча.

1.3.1 ВЫБОР СРЕДЫ РАЗРАБОТКИ

В настоящее время существует множество инструментов для разработки компьютерных игр, рассмотрим самые популярные из них:

1. `Unity Engine`– кроссплатформенный игровой движок для создания 3D и 2D игр. Основной язык программирования для работы с `Unity - C#`. `Unity` позволяет выпускать игры для компьютеров, игровых приставок, мобильных телефонов.

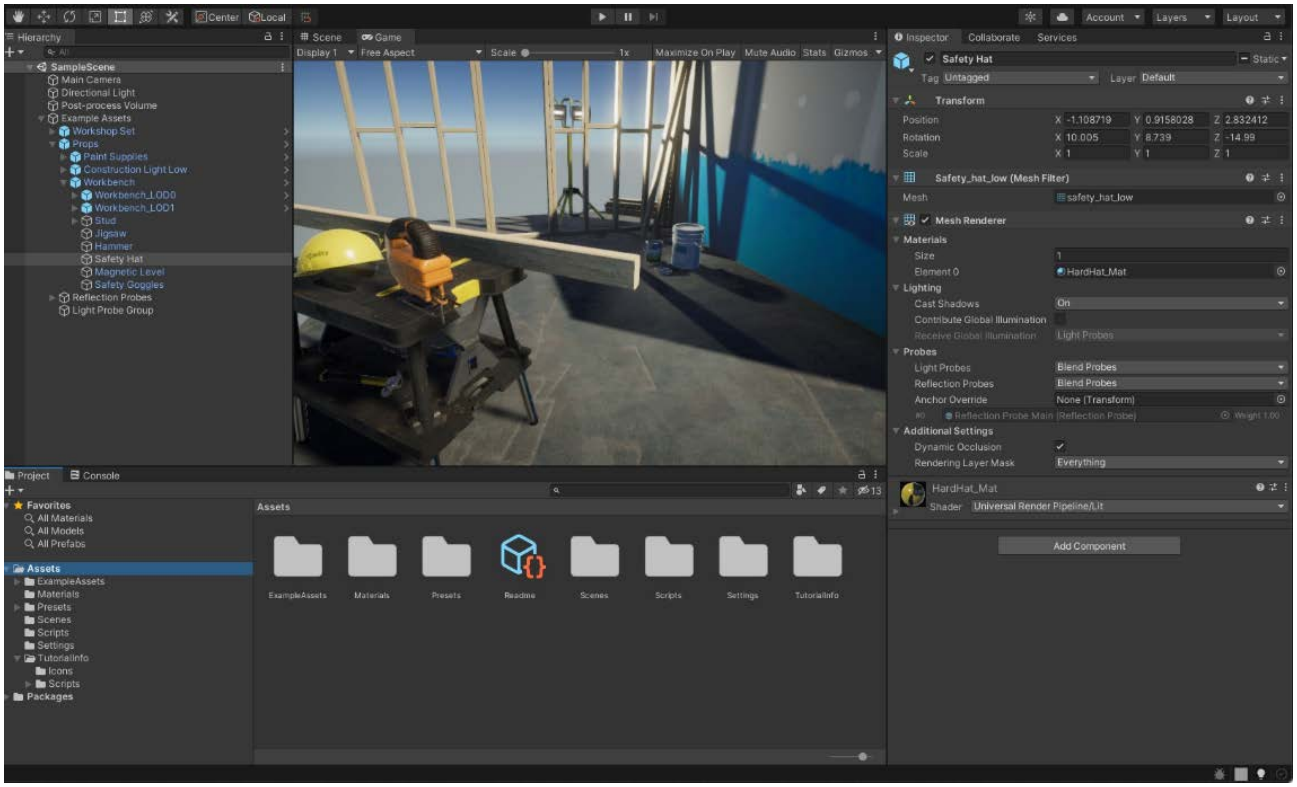

Рисунок 4 - Интерфейс Unity Engine

Достоинства:

- низкий порог входа;
- большое количество книг и курсов;
- модули и плагины для работы с камерой, для создания «катсцен», для работы с сетью, для визуального программирования и другие;
- большой магазин готовых «Ассетов».

Распространение игры, созданной на Unity, бесплатное. “Разработчик не платит деньги с выручки до тех пор, пока заработок от игры не превысит 100 000$ за последние 12 месяцев. При превышении данной суммы разработчик обязан платить 100$ ежемесячно [3].”

2. `Unreal Engine` – кроссплатформенный игровой движок, использующий язык C++ для написания скриптов.

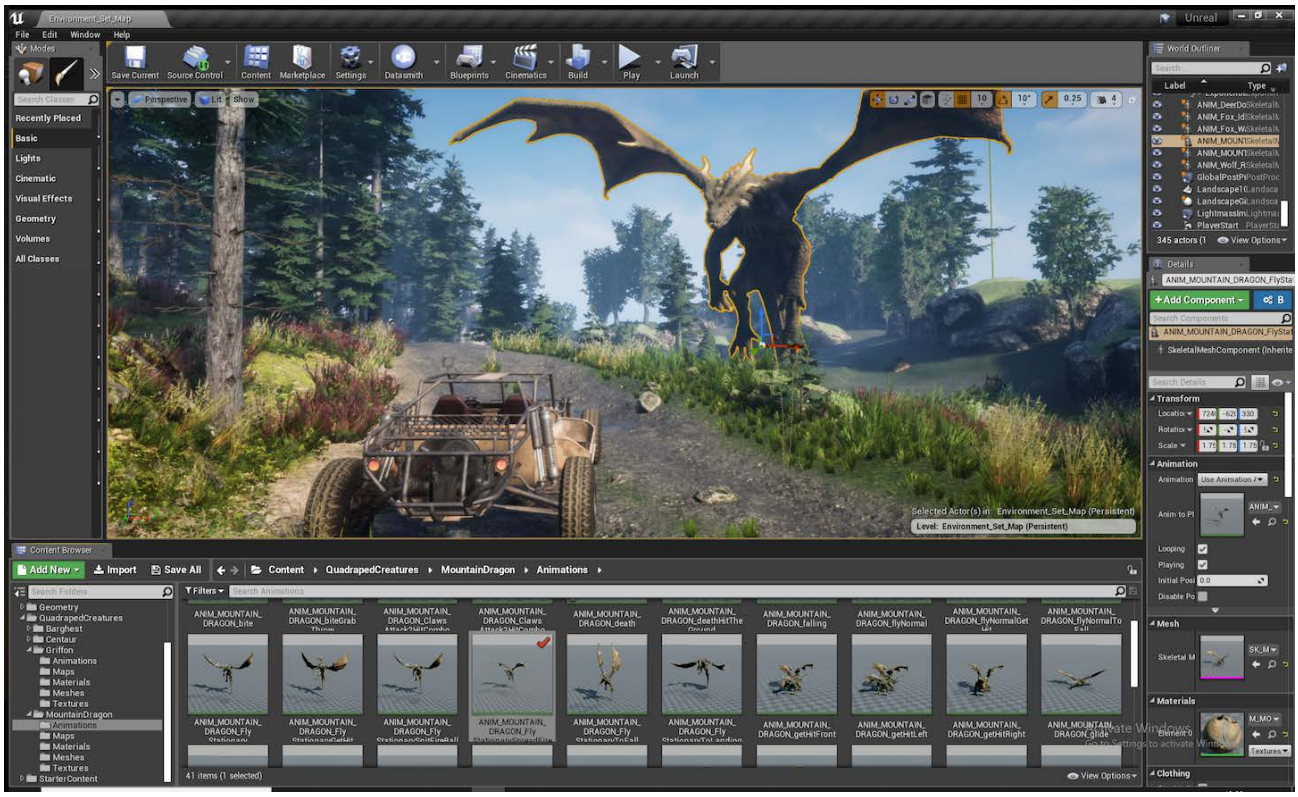

Рисунок 5 - Интерфейс Unreal Engine

Достоинства:

- высокая производительность созданных приложений и игр;
- высокое качество визуальной составляющей игры без особых усилий;
- технология визуального программирования «`Blueprint`», с возможностью генерации исходного `C++` кода;
- возможность легкой интеграции с инструментами для создания стороннего контента, будь то 3D модели или аудио-составляющая.

“Распространение игры, созданной на Unreal Engine, накладывает на доходы от нее налог в пользу «Epic Games» в размере 5% с момента выпуска [4].”

Данные инструменты является лидирующими среди конкурентов. Обширное сообщество и многолетний опыт других разработчик делают создания игры в условиях данного движка комфортным занятием.

1.3.2 ВЫБОР СРЕДСТВ АНАЛИТИКИ

1. Firebase совместно с Google Analytics

Firebase объединяет все сервисы Google в одном аккаунте. Можно подключить аналитику, рекламу, отслеживание вылетов приложения. Интеграция Firebase в проект интуитивно понятная, но несколько сложнее, чем интеграция с Unity Analytics. Поддерживает только мобильные платформы.

2. Unity Analytics

Позволяет отслеживать стандартные действия игроков без написания дополнительного кода, такие как:

- первый запуск приложения;
- время, проведенное в игре;
- внутриигровые покупки, сделанные с помощью Unity IAP. Поддерживает все платформы, под которые Unity может делать сборки. Есть возможность выгрузить «сырые» данные для внешнего анализа.

1.3.3 СЕРВИС ДЛЯ ХОСТИНГА СЕРВЕРНОЙ СБОРКИ ИГРЫ

Azure Cloud и Digital Ocean. Оба решения позволяют проводить удаленную настройку выделенного виртуально средства, настраивать Docker-контейнеры для запуска удаленной серверной сборки. Поэтому сравнение будет идти по тарифам. Помимо хостинга серверной сборки игры необходим хостинг

выделенного сервера для организации «виртуальных комнат», который будет создавать `instance` серверной сборки для каждой комнаты.

1. `Azure Cloud`

Обладает большим авторитетом и узнаваемостью. “Минимальная цена за виртуальную машину с 20GB жестким диском, 1 ядром процессора и 1GB оперативной памяти составляет порядка 10$.” [10] Обладая студенческой подпиской, можно создать виртуальную машину для дипломной работы бесплатно.

2. `Digital Ocean`

Менее популярное решение, но ничем не уступающее решение. “Виртуальная машина в такой же комплектации стоит порядка 5$ в месяц.” [9]

На рисунке 6 представлены расценки `DigitalOcean`.

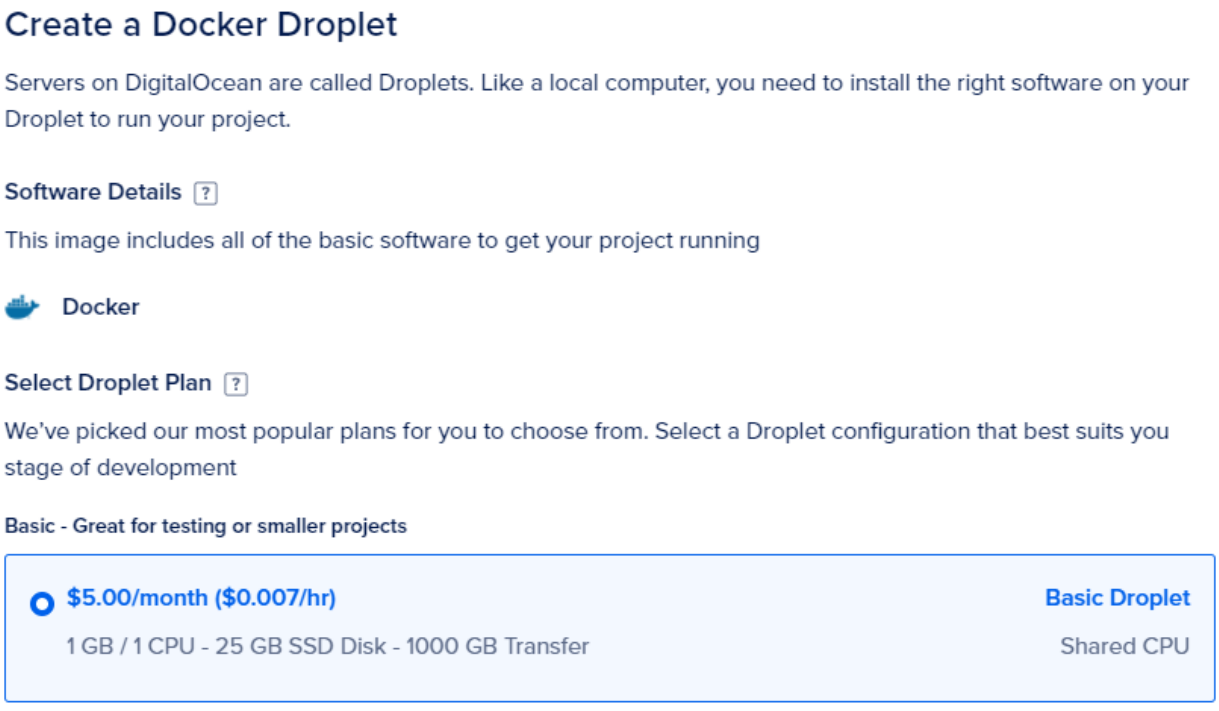

Рисунок 6 - Тарифы Digital Ocean

1.3.4 ИНСТРУМЕНТ ДЛЯ ОРГАНИЗАЦИИ МУЛЬТИПЛЕЕРА

Представленные фреймворки Mirror UNet, Photon Engine и socket.io направлены на упрощение создания «мультиплеерной» игры для заданного количества конкурентных игроков. У обоих есть плюсы и минусы. Рассмотрим их.

1. Mirror UNet

Mirror – инструмент для создания мультиплеера в Unity играх. Он может использоваться для множества общих задач, которые могут вставать перед разработчиками мультиплеерных игр. Mirror позволяет одному из клиентов «хостить» игру, избавляя разработчика от создания выделенного сервера. “Разрабатывая игру совместно с другими сервисами, Mirror позволяет создавать мульиплеерные игры в одиночку.” [8]

Возможности:

- обработка сообщений;
- выделение серверного и клиентского кода через тэги [Server] и [Client] предотвращает вызов методов в ненужном месте;
- тэг [Command] для общения клиент – сервер;
- тэги [SyncVar] и [SyncList] для синхронизации параметров между всеми клиентами;
- высокопроизводительная сериализация;
- управления серверными объектами;
- синхронизация состояния;
- сетевые классы: Server, Client, Connection.

На рисунке 7 представлена уровневая модель, показывающая, что разработчик на Mirror UNet работает с самыми верхними уровнями

абстракции, практически не касаясь реализации подключений и работы сетевых протоколов.

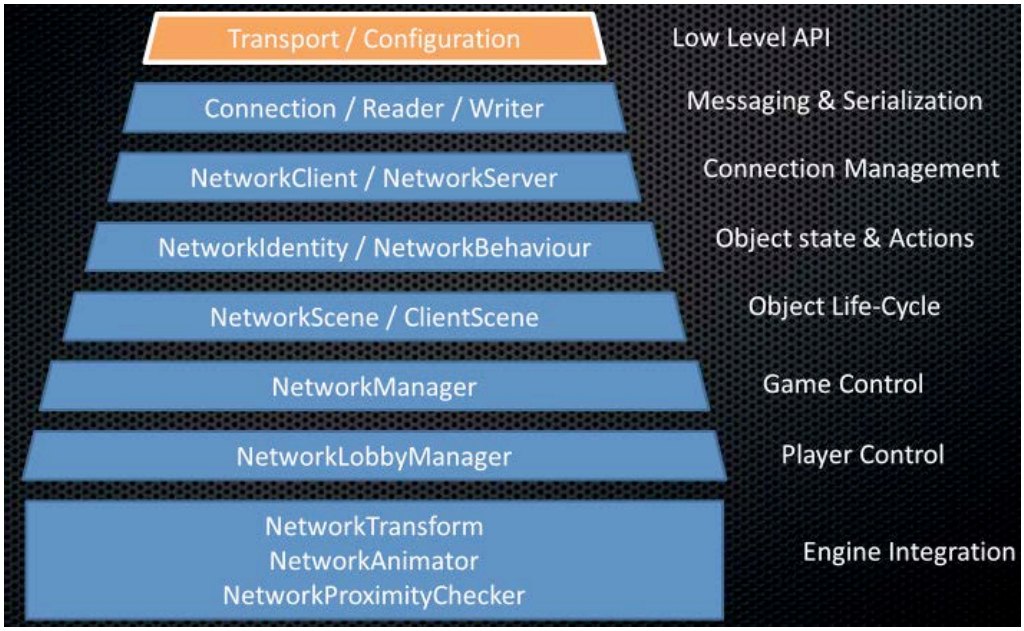

Рисунок 7 - Уровни абстракции Mirror

2. Photon Engine

Photon Engine доступен как для Unreal Engine, так и для Unity Engine. Является инструментом, не требующим выделенного сервера. Разработчик самостоятельно определяет синхронизируемые между игроками параметры, будь то:

1) Позиция в мировых координатах.
2) Поворот.
3) Масштаб.
4) Скорость.
5) Пользовательские параметры.

Поскольку Unreal и Unity являются компонентными инструментами, приведем скриншот классических Photon компонентов из Unity, сославшись на то, что в Unreal Engine аналогичные компоненты. На рисунке 8

представлены компоненты для синхронизации `Transform` – координат, `Animator` – анимации, `Rigidbody` – для синхронизации физического перемещения.

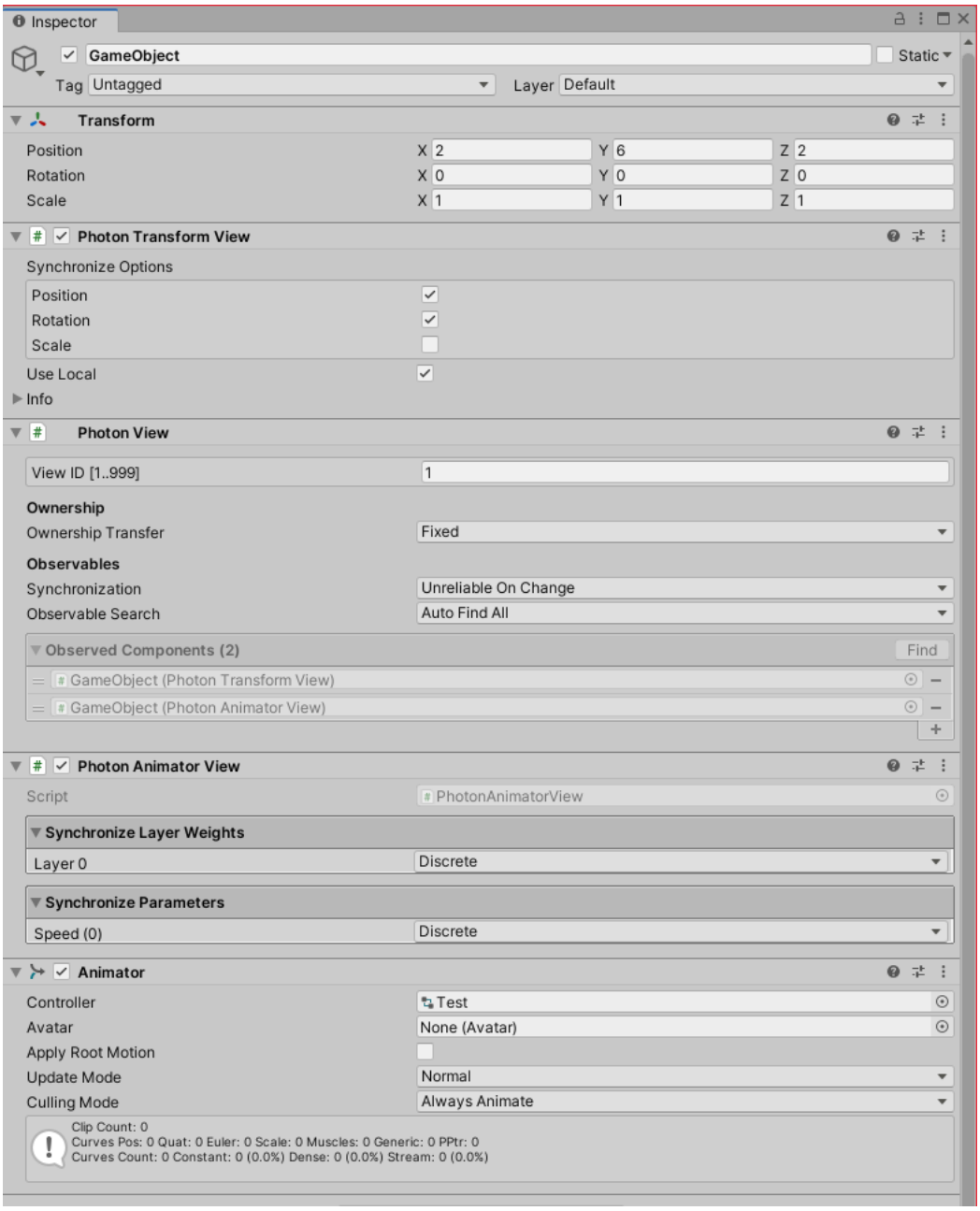

Рисунок 8 – Компоненты для синхронизации параметров

Если в игре присутствуют анимации, разработчик может настроить их синхронизацию между клиентами, чтобы у игрока не создавалось впечатления, что игровой процесс соперников отличается от его игрового процесса.

Основные достоинства:

1) Отсутствие необходимости для выделенного игрового сервера.
2) Легкая настройка, которая заключается в копировании созданного `id` приложения из Панели управления `Photon Engine` [16] в приложение.
3) Синхронизация настроенных разработчиком параметров, а также отправка сообщений между клиентами ложится на плечи сервиса.

Главным минусом является ограничение в 20CCU (20 конкурентных пользователей единовременно), что довольно мало, но вполне достаточно для прототипирования. “Далее действуют тарифы 100$ за 12 месяцев.” [7]. Возможно повысить бесплатный лимит до 100CCU, разместив Photon Cloud на своем хостинге.

3. Socket.io

Подход основан на популярной `node.js` библиотеке `socket.io` [15], написанной на `JavaScript`, позволяющей с помощью веб-сокетов создавать клиент-серверные приложения. В данном случае клиентом будет являться игра, а сервером выделенное `node.js` приложение.

Общение между клиентом и сервером происходит с помощью сообщений, которые могут посылаться как клиентом, так и сервером.

В случае с `Unity`, существует адаптированная под `.net framework` реализация `socket.io-client` [17] библиотеки, написанная на `C#`.

1.4 ВЫВОДЫ

Из-за легкости освоения движка, выберем `Unity Engine`, как основу для создания игры. Для хостинга виртуальной машины выберем `Azure Cloud`, так как для студентов там существует студенческая подписка, чего достаточно для создания и реализации дипломной работы. В качестве инструмента для

создания мультиплеера в игре был выбран Mirror, т.к. у Photon Engine довольно скромные бесплатные тарифы, чего может не хватить даже для качественного тестирования готового дипломного проекта. В качестве технологии для создания выделенного сервера выберем node.js, т.к. сервер на нем создается в пару строчек кода, а написания функций и создание API не занимает много времени.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

Опишем основные функциональные требования к системе в целом, а также к ее составным частям.

Требования к игре в целом:

1) Автоматическое адаптирование элементов управления персонажем под различные разрешения экрана.
2) Звуковое сопровождение основных событий внутри игры.
3) Управление вовлеченностью игроков в геймплей с помощью добавления новых механик, поощрения отзывов.
4) Обеспечение 30 кадров в секунду даже на старых устройствах с `Android 4.4`.
5) Возможность видоизменять аватар игрока.

2.1 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1) Выбор персонажа из списка.
2) Подключение к серверу.
3) Передача сообщения серверу о выбранном персонаже.
4) Настройка уровня сервером и создание нужных сетевых объектов персонажей.
5) Взаимодействие с другими игроками

Необходимо реализовать подсистемы:

1) Подключения к серверу.
2) Систему умений для каждого из персонажей.
3) Управления персонажем.

На рисунке 9 представлена `UseCase` диаграмма разрабатываемого проекта.

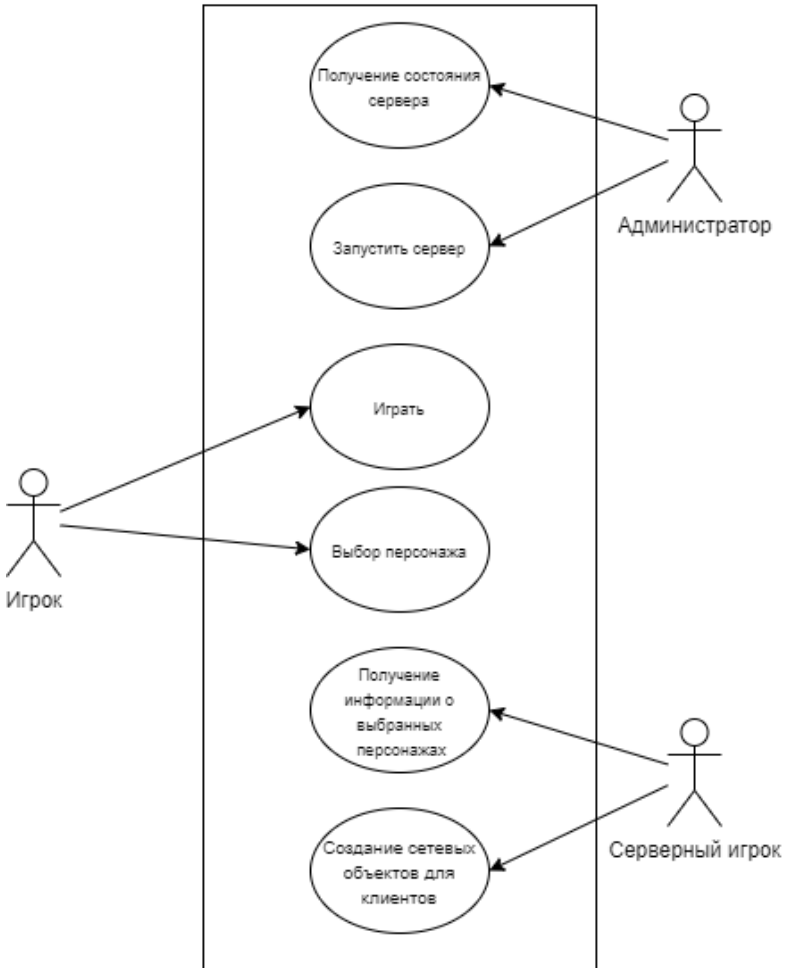

Рисунок 9 - Use-case диаграмма игры как системы

Функциональные требования к подсистеме “Подключение к серверу”:

1) Подключение осуществляется при запуске игры.
2) В случае потери соединения с сервером дать пользователю возможность нажать на кнопку “`Retry`” для повторного подключения.
3) В случае невозможности установить соединение отобразить информацию в пользовательском интерфейсе.
4) Обмен данными с сервером и клиентом.

Функциональные требования к подсистеме “ Развития персонажа”:

Подсистема развития персонажа отвечает за получение игрового опыта, выбора различных навыков и умений персонажа, улучшение игровых предметов и их создание.

1) Получение игрового опыта для получения игровых уровней.
2) Приобретение уникальных навыков и умений персонажа из доступного перечня на доступные очки умений, получаемые после повышения уровня.
3) Улучшение имеющихся предметов с помощью специальных навыков персонажа.
4) Создание предметов на основании рецептов с помощью специальных навыков персонажа.

Функциональные требования к подсистеме “Управления персонажем”:

1) Управление персонажем происходит локально с помощью виртуальных джойстиков и кнопок, расположенных на экране.
2) Управление учитывает непроходимость некоторых препятствий во избежание эффекта “Проваливания под текстуры”.
3) Управление учитывает физические размеры объектов, имеющих коллайдеры, во избежание застревания в игровых локациях.

2.2 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ СИСТЕМЫ

1) Работа игры под мобильные платформы `Google Android` версии `API` не ниже 19 (`Android Kitkat 4.4`) и `Apple IOS` версии не ниже `IOS 8`.
2) Создание и проектирование игры с помощью `Unity Engine`.
3) Использование `C#` для написания игрового кода.
4) Работа удаленного `node.js` сервера.

5) Интерес для неиграющей аудитории.
6) Адекватное отображение пользовательского интерфейса на различных разрешениях экрана мобильных устройств.

2.3 ТРЕБОВАНИЯ К ВИДАМ ОБЕСПЕЧЕНИЯ

1) Наличие удаленного сервера, хранящего всю необходимую информацию о локации для поддержания состояния игрового мира.
2) Хранение информации о пользователях и их аватарах в СУБД на удаленном сервере.
3) Сбор информации о локальном передвижении каждого игрока и предмета на локации.
4) Рассылка собранной информации на конечные устройства для обновления локальных позиций и состояний других игроков и предметов.

3. ПРОЕКТИРОВАНИЕ

По своей сути проект является клиент-серверным приложением, не включая промежуточного `node.js` сервера.

Примерная архитектура инструментов представлена на рисунке 10.

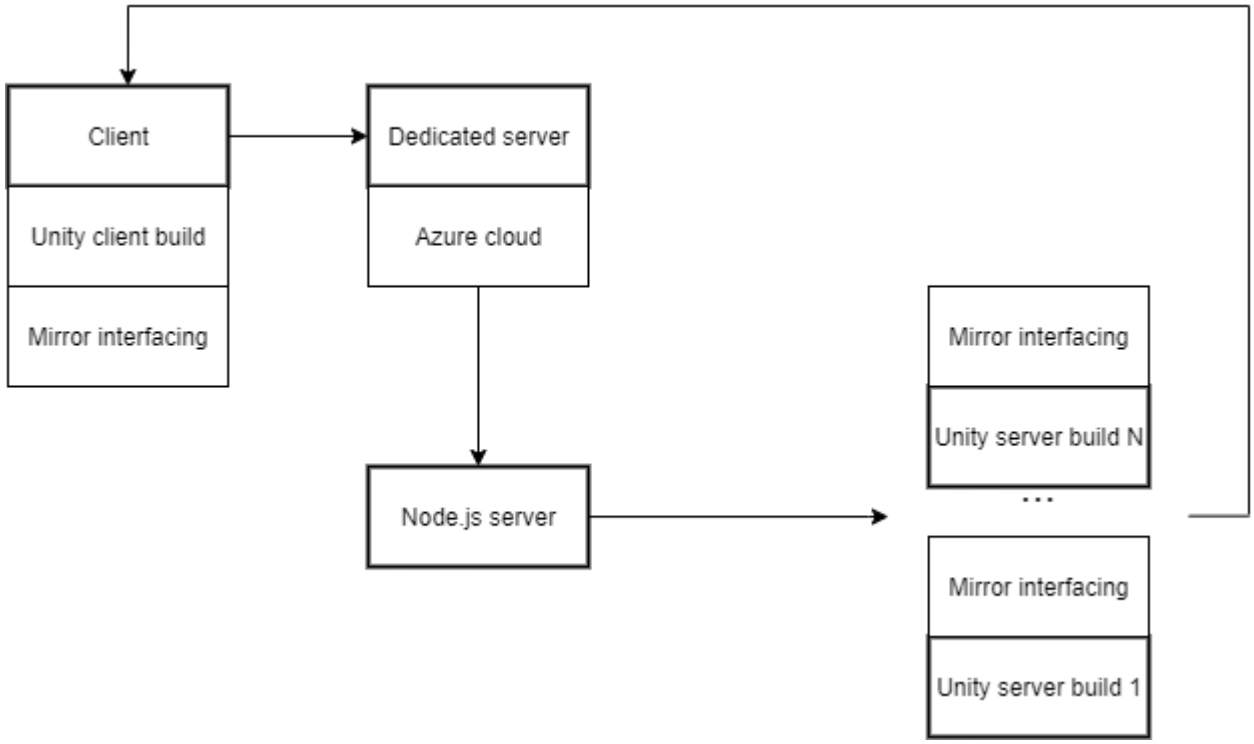

Рисунок 10 - Архитектура инструментов

В архитектуре присутствует удаленный сервер, с запущенным `node.js` приложением для обработки запросов игроков на создание «матчей». Дальше это приложение запускает instance серверной сборки с определенным портом, после чего возвращает клиенту. Далее клиент по полученному порту и статическому ip-адресу подключается к серверной сборке, являющейся `Unity` приложением. Достоинством `Mirror Networking` является то, что клиентская и серверная сборка собираются из одного исходного кода. Отличие заключается в принципе сборки. При сборке серверного клиента создается `Headless` приложение без пользовательского интерфейса.

Далее рассмотрим примерную архитектуру классов поведения персонажа. Существуют 2 базовых класса для персонажей: `Player Controller`, осуществляющий управление персонажем, и `PlayerBaseBehaviour`, осуществляющий поведение персонажа (Обработка столкновений с другими персонажами, обработка звуков, взаимодействие с `GameManager`). Далее для каждого вида персонажей наследуются данные классы, переопределяющие базовые методы, задавая индивидуальное поведение на выбранный функционал.

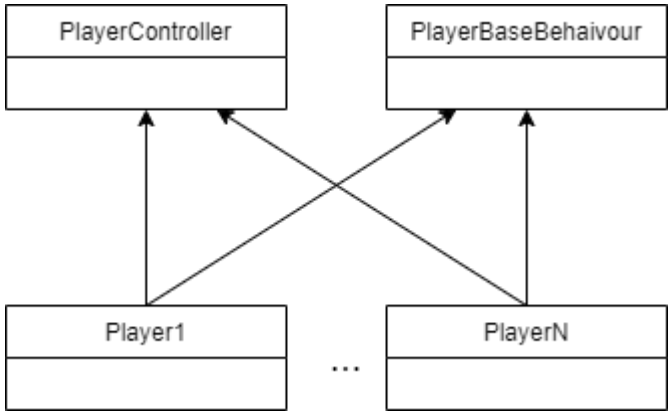

Рисунок 11 - Структура классов персонажа

Основным менеджером в игре является `GameManager`, который инкапсулирует игровые состояния, а также осуществляет взаимодействие менеджеров друг с другом.

`Network Manager`, является наследуемым от базового класса, предоставляемого Mirror Networking. Наследование от класса `NetworkManager` позволяет получить контроль над операциями подключения игроков к серверу, а также ввести промежуточные действия между стадией подключения и входом в игру. Например, для реализации выбора игроком персонажа перед входом в комнату, можно переопределить `Callback OnServerAddPlayer()`, который вызывается после подключения клиента к серверу, но до создания объекта `Player` на сервере и последующего его дублирования на каждом из клиентов.

`PlayersManager` хранит информацию обо всех игроках, позволяет передавать сообщения или `RPC (Remote Procedure Call)` конкретному игроку по информации из его локального объекта без необходимости сетевой идентификации. `PlayersManager` инкапсулирует работу со списком игроков, предоставляемым `NetworkManager`. `PlayersManager` осуществляет поиск по списку и позволяет вести безопасную работу с ним.

Также вся графика в игре должна быть «воксельной» для достижения максимальной производительности из-за небольшого количества полигонов, и следовательно, требующей меньших затрат вычислительных ресурсов для их отрисовки и буферизации.

4. РЕАЛИЗАЦИЯ

В ходе выполнения ВКР был написан `node.js` сервер (в листинге В.1 приложения В), принимающий запросы от игроков и возвращающий им порт сервера, по которому они могут подключиться к своей комнате игры.

Для отслеживания внутриигровых действий игрока был импортирован `Google Firebase`, который после установки не требует настройки и способен подсчитывать количество сыгравших игроков.

`Node.js` сервер совместно с серверной сборкой игры был загружен на арендованную машину `Azure Virtual Box` по студенческой подписке, предоставленной ЮУрГУ. Настройка виртуальной машины проста и банальна. Нужно настроить DNS и открыть порты, а также загрузить на виртуальную машину `node.js` сервер и серверную сборку игры.

На рисунке 12 представлена схема реализованных классов для управления персонажами.

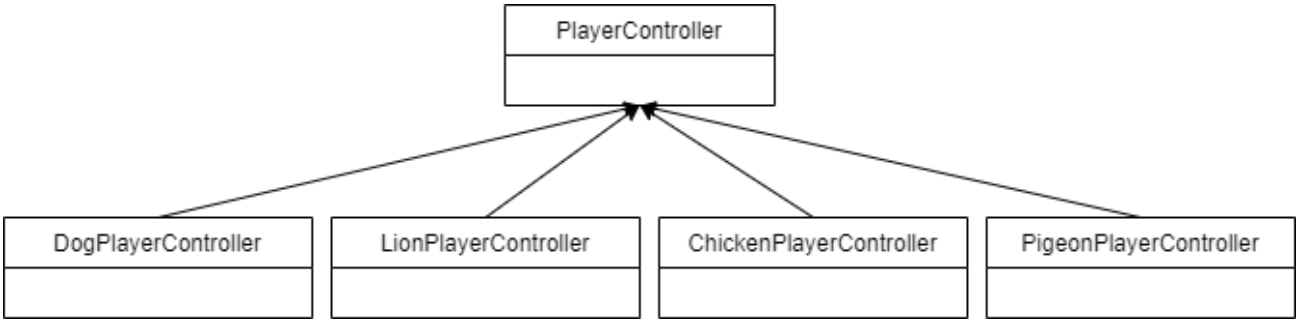

Рисунок 12 – Схема реализованных классов управления персонажами

Опишем логику работы игры. Игра состоит из 2 сцен:

1. Сцена `MainMenu`. Содержит в себе пользовательский интерфейс подключения к игре (рисунок 27), а также кнопку для входа в меню настроек (рисунок 26). На сцене содержатся объекты:
    - `EventSystem`. Используется для обработки событий взаимодействия с пользовательским интерфейсом;
    - `Camera`;

- `Canvas`. Содержит элементы пользовательского интерфейса;
- `NetworkManager`. Объект сетевого менеджера. Помечен как `Don't destroy on load`, что обеспечивает сохранение объекта при переключении между сценами (рисунок 14).

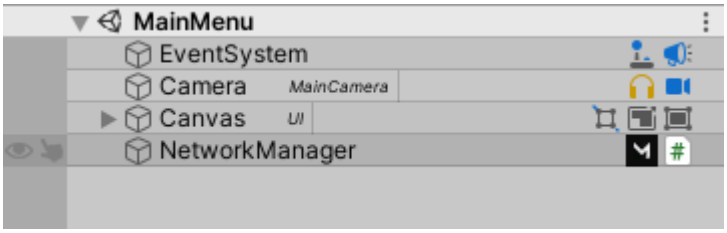

Рисунок 13 – Дерево объектов главного меню

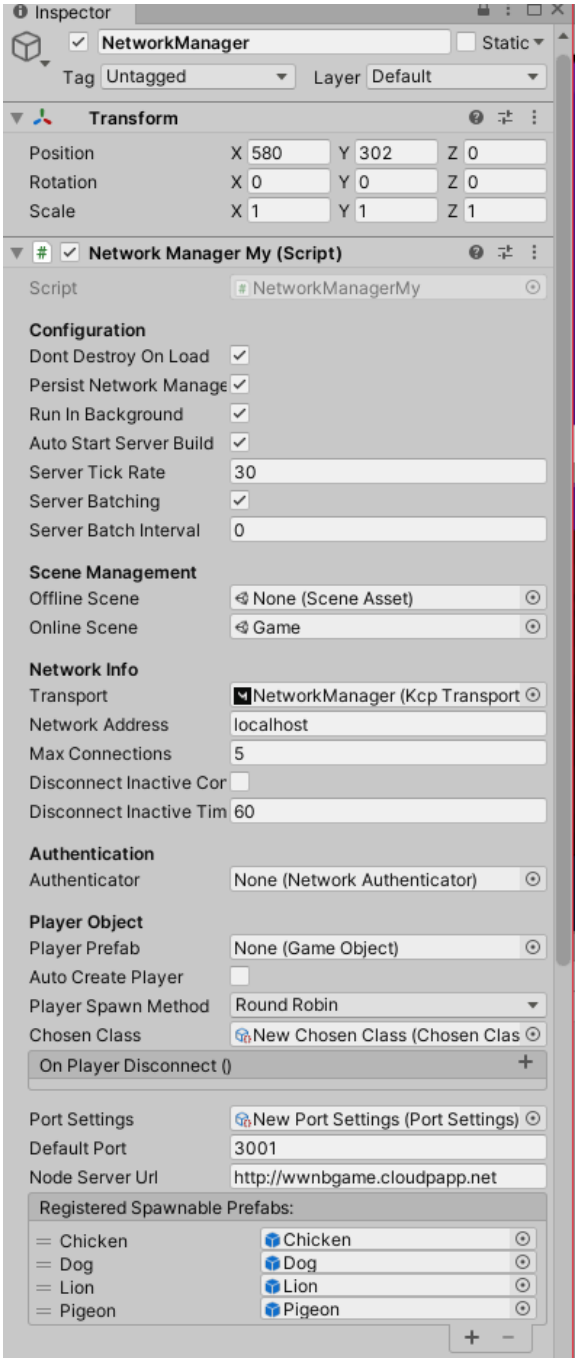

Рисунок 14 – Настройка наследуемого компонента NetworkManager

2. Сцена `Game`. Является основной игровой сценой. Является online-сценой, так как загружается `Mirror Unet` автоматически при подключении к серверу. В сцене использовался Hierarchy 2 для логического отделения объектов по их типу на секции. Так, имеются секции:
    - `CAMERA` — хранящая объекты, управляющие камерой;

- UI — хранящая весь пользовательский интерфейс;
- MANAGERS секцию, хранящую все игровые менеджеры;
- WORLD секцию, хранящую все игровые объекты;
- COINS — хранящая игровые монеты.

Сцена состоит из следующих объектов:

- CM vcam1. Объект Cinemachine для управления камерой;
- Main Camera объект камеры;
- Joystick canvas, которым управляет InputManager; включается автоматически при сборке под мобильные платформы;
- CanvasStats, показывающий информацию о игроке и количестве монет;
- KillChecker, содержащий BoxCollider и проверяющий на столкновение с игроками; является сетевым объектом;
- GameManager, содержит основную игровую логику. Описан далее.
- SceneContext – объект Zenect фреймворка, содержащий «контейнер»;
- AudioManager, отвечает за звуковое сопровождение игры.

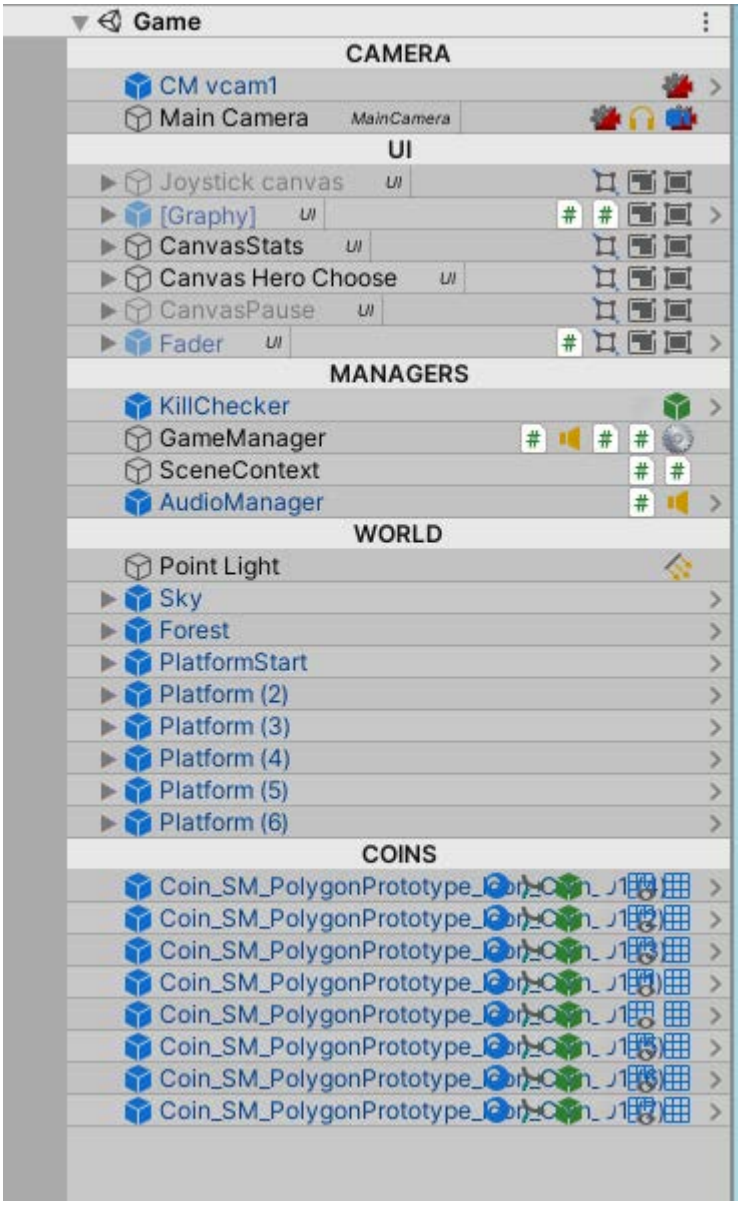

Рисунок 15 – Дерево объектов сцены Game

В ходе реализации ВКР были использованы следующие плагины и фреймворки:

1. `UniRx` [19] – инструмент для внедрения реактивных расширений в `Unity`. Реактивные расширения позволяют не опрашивать свойство или переменную об изменении значения или состояния. При изменении значения реактивная переменная сама вызовет всех подписавшихся слушателей (`Listener`'ов).

Плагин позволяет создавать пользовательские потоки событий, будь то нажатия клавиш, клики мыши или просто созданные UnityEvent, а также предоставляет систему фильтрации и дополнительные условия для потоков.

Инструмент включает в себя таймеры, которые используются в данном проекте. Таймеры UniRx удобны легкой настройкой, фильтрацией и прекращением отсчета. В Unity нет встроенной реализации таймеров, UniRx это компенсирует.

2. DOTween [23] – Unity плагин для создания скриптовых анимаций.
3. Zenject [20] – популярный Dependency Injection фреймворк. Позволяет предоставлять реализацию какого-либо объекта из настроенного «контейнера». Здесь и далее «контейнер» - виртуальное хранилище объектов для разделенного доступа. Удобен тем, что в некоторых местах программы способен убрать жесткие ссылки между системами. В данном проекте используется для предоставления нужной реализации InputManager в классы, управляющие игроком. Является реализацией буквы D принципа S.O.L.I.D [21], означая Dependency inversion (инверсия зависимостей), когда класс, где используется какой-либо объект, не должен думать о методе его получения.

   Его настройка и создание «контейнера» представлены в классе ModuleInstaller. Каждая сцена должна содержать своей ModuleInstaller. Для объектов, которые будут создаваться в Runtime (в момент работы игры), существует ZenAutoInjecter компонент.
4. Hierarchy 2 – расширение для редактора, позволяющее группировать объекты в группы.
5. Serializable Dictionary – предоставляет «сериализуемые» списки.

6. Cinemachine [22] – инструмент для работы с камерой, распространяемый внутри Unity. Плагин берет на себя большинство работы по следованию камеры за указанным объектом. Плагин способен:
    1) Управлять поворотом камеры с помощью мыши.
    2) Двигать камеру в определенных зонах экрана, называемых «мертвыми» зонами.
    3) Настраивать Dumping – скорость изменения позиции камеры за следуемым объектом.

    От пользователя необходимо настроить тип камеры, мертвые зоны экрана, а также Camera Confimer, зоны в мировых координатах, в пределах которой может двигаться камера.

В ходе разработки продукта были реализованы следующие классы на языке C#:

1. PlayerController (в листинге А.1 приложения А) – базовый класс для управления игроком. В нем реализованы методы для полета и прыжка, а также атаки с помощью умений. Абстрактными являются методы, разрешающие или запрещающие полет и прыжок. В классе заданы поля для скорости, доступных умений, звуковых дорожек. Считывание ввода происходит в методе Update через базовый интерфейс InputManager, скрывающий текущую реализацию экранного джойстика или клавиатуры, которые выбираются автоматически в зависимости от типа клиента. Движение и использование умения происходит в FixedUpdate. Различия Update и FixedUpdate в том, что Update вызывается на каждый кадр, тем самым время между вызовами различается, в то время как FixedUpdate вызывается с постоянным интервалом.

    Использование FixedUpdate для движения и умений обусловлено тем, что они используют физический движок Unity, а FixedUpdate

вызывается на каждое обновление физики. Таким путем достигается плавность и производительность физического кода.

2. `PigeonPlayerController` (в листинге А.2 приложения А) – класс, наследуемый от `PlayerController`. Задает возможность полета и прыжка для пингвина.
3. `LionPlayerController` (в листинге А.5 приложения А) – класс, наследуемый от `PlayerController`. Задает возможность полета и прыжка для льва.
4. `ChickenPlayerController` (в листинге А.4 приложения А) – класс, наследуемый от `PlayerController`. Задает возможность полета и прыжка для курицы.
5. `DogPlayerController` (в листинге А.3 приложения А) – класс, наследуемый от `PlayerController`. Задает возможность полета и прыжка для собаки.
6. `FlyerLogic` (в листинге А.6 приложения А) – класс с логикой работы панели для полета. Содержит в себе параметр дальности полета. Проверка на возможность полета происходит непосредственно в методе интерфейса `IFlyer`. Имеет ссылку на `Animator`, для проигрывания анимации срабатывания.
7. `JumperLogic` (в листинге А.7 приложения А) – класс с логикой работы панели для прыжка. Содержит в себе количество возможных прыжков на панели, а также высоту возможного прыжка. Имеет ссылку на `Animator`, для проигрывания анимации срабатывания.
8. `GameManager` (в листинге А.8 приложения А) – основной класс с логикой игры. Является связующим звеном между всеми системами игры, таких как:

1) Хранение информации о собранных монетах.
2) Отслеживание информации о поражении и выигрыше игрока.
3) Связывает пользовательский интерфейс с данными модели.

9. `NetworkManagerMy` (в листинге А.9 приложения А) – сетевой класс, наследуемый от `NetworkManager Mirror` класса. Класс переопределяет базовые методы `OnServerAddPlayer` для возможности выбора разных персонажей на всех клиентах. Также класс получает порт, на котором нужно запустить сервер из командной строки.

   Использует `SerializedDictionary` для создания списка возможных персонажей по ключу/значению и возможности его задания из интерфейса редактора. Содержит методы с тэгами `[Server]` и `[Client]` для выполнения кода на клиенте и сервере соответственно.

10. `CoinsSettings` (в листинге А.10 приложения А) – `ScriptableObject` класс с найстроками текущего количества монет.
11. `ChoosenClass` (в листинге А.11 приложения А) – `ScriptableObject` класс с настройками текущего выбранного персонажа.
12. `BaseBar` (в листинге А.12 приложения А) – базовый класс с логикой работы всех «статус баров» в игре. Использует `DOTween` для `Fade in` и `Fade out` анимаций.
13. `SimpleFader` (в листинге А.13 приложения А) – класс для затемнения экрана в конце игры. Использует `DOTween`.
14. `ProgressBar` (в листинге А.14 приложения А) – наследуемый класс от `BaseBar`, реализующий логику работы со «слайдер» барами.
15. `SkillType` (в листинге А.15 приложения А) – базовый класс для всех умений. Содержит один метод для определения.
16. `MultipleAttack` (в листинге А.16 приложения А) – умение для множества атак вперед. Содержится у всех животных.
17. `HeavyAttack` (в листинге А.17 приложения А) – умение для тяжелого рывка вперед. Содержится у некоторых животных.
18. `Consumable` (в листинге А.18 приложения А) – базовый метод для здоровья, маны и прочих расходных единиц. Содержит события об окончании `Consumable` параметра, а также об его изменении.

19. `Mana` (в листинге А.18 приложения А) – класс, наследуемый от `Consumable`. Содержит в себе логику постепенной регенерации маны. Используется для умений.
20. `Health` (в листинге А.18 приложения А) – класс, наследуемый от `Consumable` для отслеживания здоровья.
21. `EndGameChecker` (в листинге А.19 приложения А) – класс, проверяющий игру на максимальное время жизни и отслеживающий команды от игроков на выигрыш. После чего закрывает игру и разрывает все соединения.
22. `PortSettings` (в листинге А.20 приложения А) – `ScriptableObject` класс, содержащий настройки порта, на котором запущен текущая серверная сборка.
23. `Utils` (в листинге А.9 приложения А) – класс с единственным статическим методом для получения порта из параметра командной строки.
24. `AudioManager` (в листинге А.21 приложения А) – класс для управления звуками в игре. Запускает фоновую музыку при старте игры из случайного списка.
25. `ModuleInstaller` (в листинге А.22 приложения А) – класс `Zenject` фреймворка для создания «контейнера» для дальнейшей инъекции `InputManager` в необходимые места.
26. `InputManager` (в листинге А.23 приложения А) – класс, реализующий кроссплатформенное управление.

Вся графика и ресурсы для игры были заимствованы из `Unity Asset Store`, были проверены лицензии на право использования в учебных и собственных «пет» проектах.

На рисунке 13 представлена файловая структура проекта.

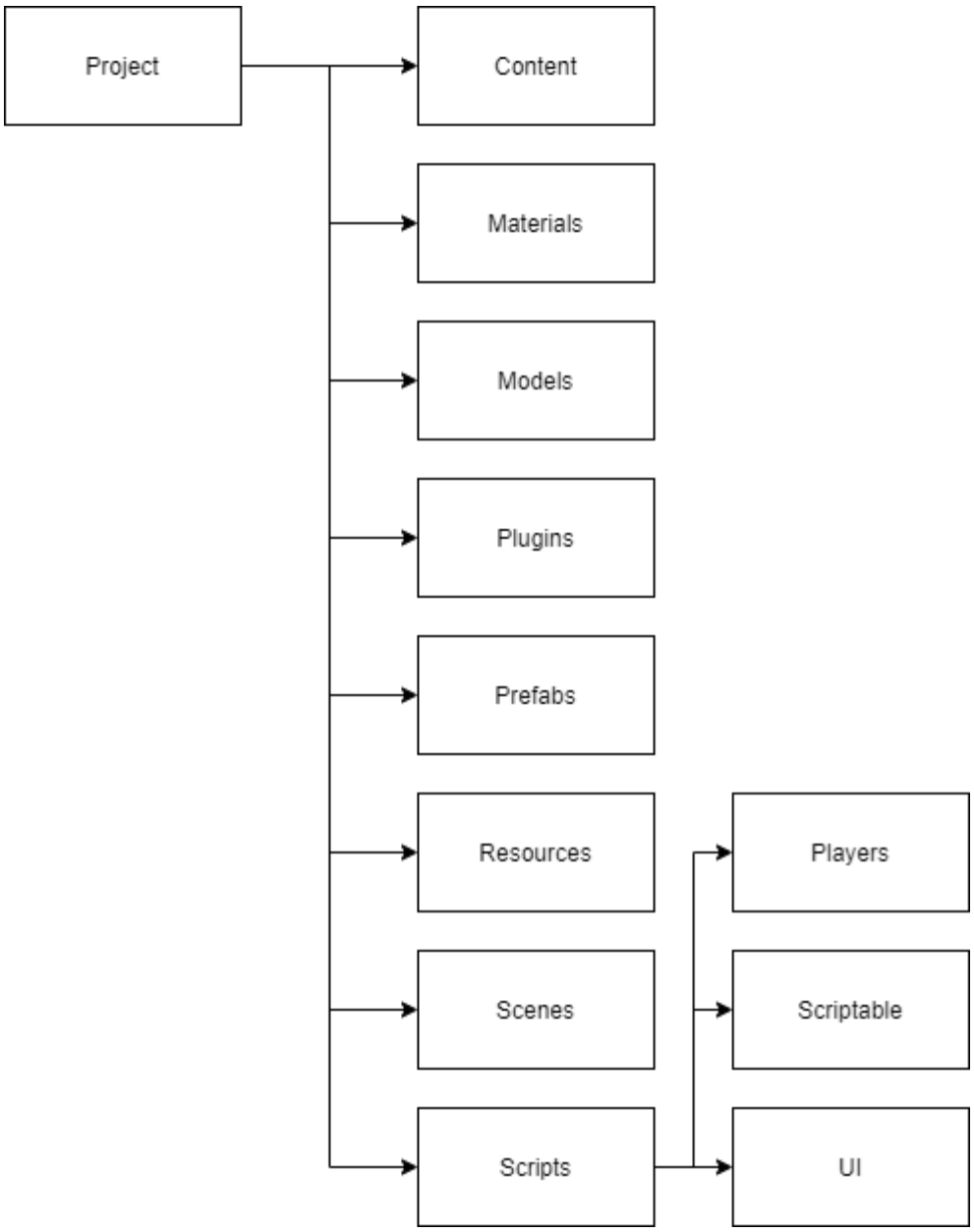

Рисунок 16 – Файловая структура проекта

Были собраны «платформы» для перемещения по уровню из скачанных ассетов.

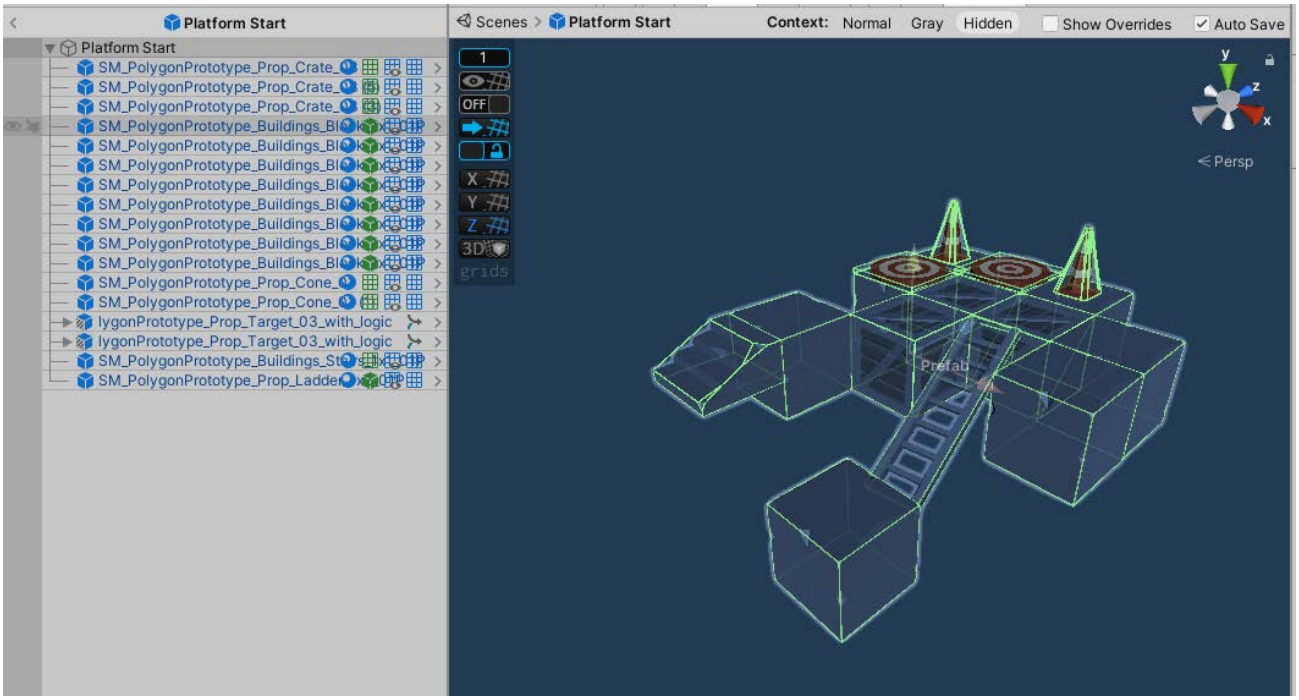

Рисунок 17 – Пример собранной платформы из «воксельных» блоков.

Для прыжков и полетов были добавлены скрипты `FlyerLogic` и `JumperLogic` на круговые платформы. Удобство и простота `Unity` заключается в написании независимых скриптов для создания пользовательского поведения и дальнейшее их добавление на объект. Также на объект круговых платформ был добавлен `BoxCollider` и выставлено поле `isTrigger` для отслеживания факта столкновения с игроком, но отсутствие фактического контакта и «застревания» игрока в платформе. `BoxCollider` является физическим «телом» прямоугольной формы для физических столкновений. После этого в скрипте `PlayerController` в `callback OnTriggerEnter` возможна проверка на столкновение с объектом и проверка на соответствие по тэгу.

Было реализовано 4 персонажа, а также анимации для них:

1) Собака. На рисунке 15 представлено дерево анимаций собаки. Здесь и далее дерево анимаций – настроенные разработчиком анимации игрового объекта, которые являются конечными состояниями абстрактного конечного автомата, а также условия перехода между состояниями.

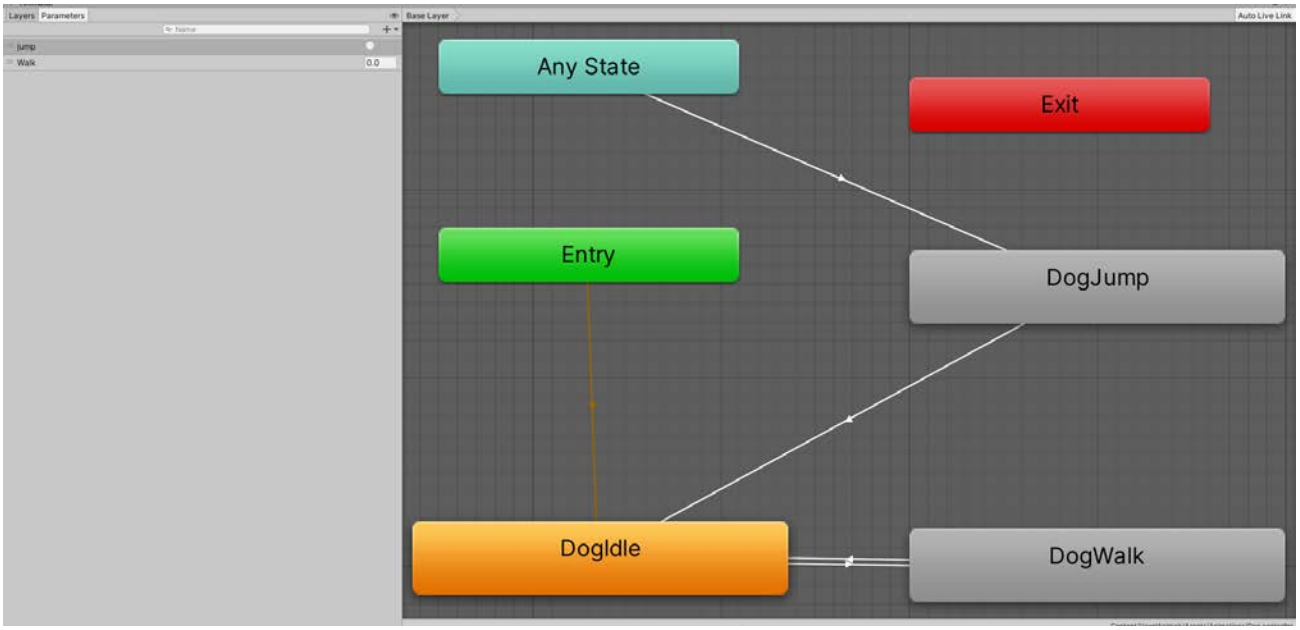

Рисунок 18 – Дерево анимаций собаки

2) Курица. На рисунке 19 представлено дерево анимаций курицы.

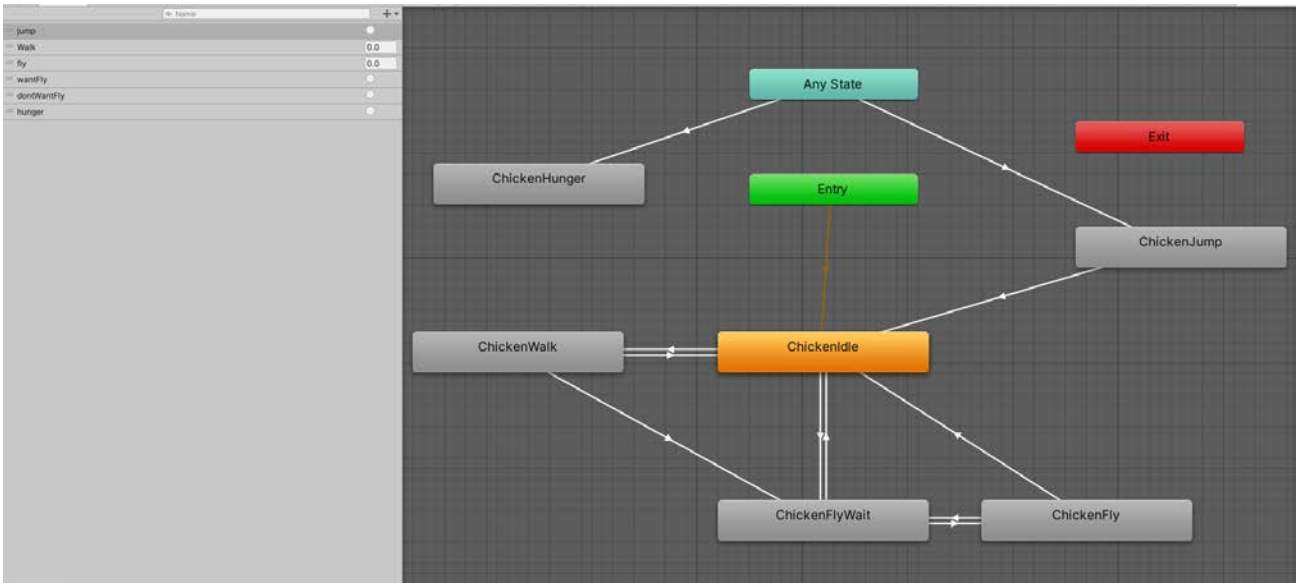

Рисунок 19 – Дерево анимаций курицы

3) Лев. На рисунке 20 представлено дерево анимаций льва.

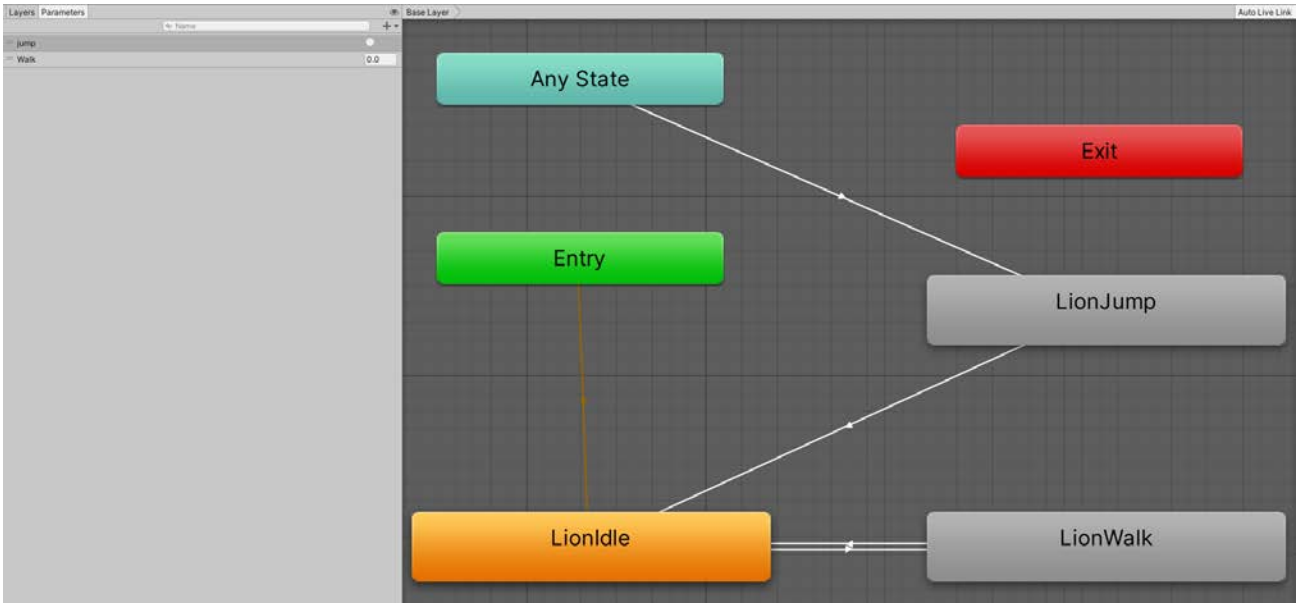

Рисунок 20 – Дерево анимаций льва

4) Пингвин. На рисунке 21 представлено дерево анимаций пингвина.

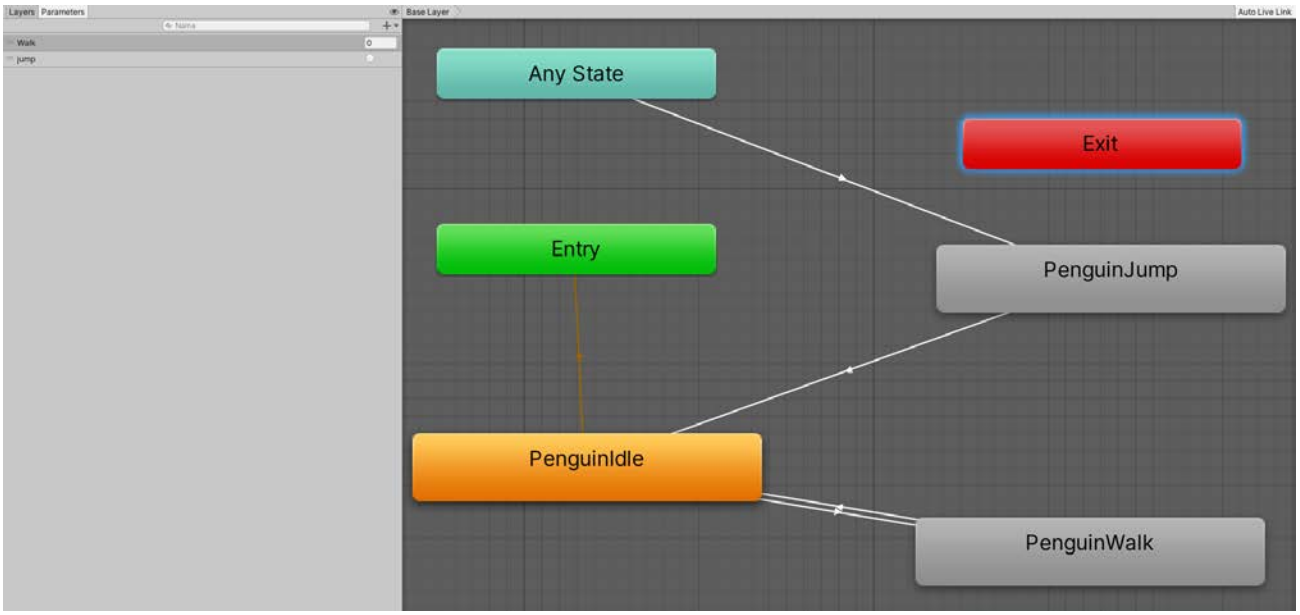

Рисунок 121– Дерево анимаций пингвина

Конечный вид игры представлен на скриншоте 22.

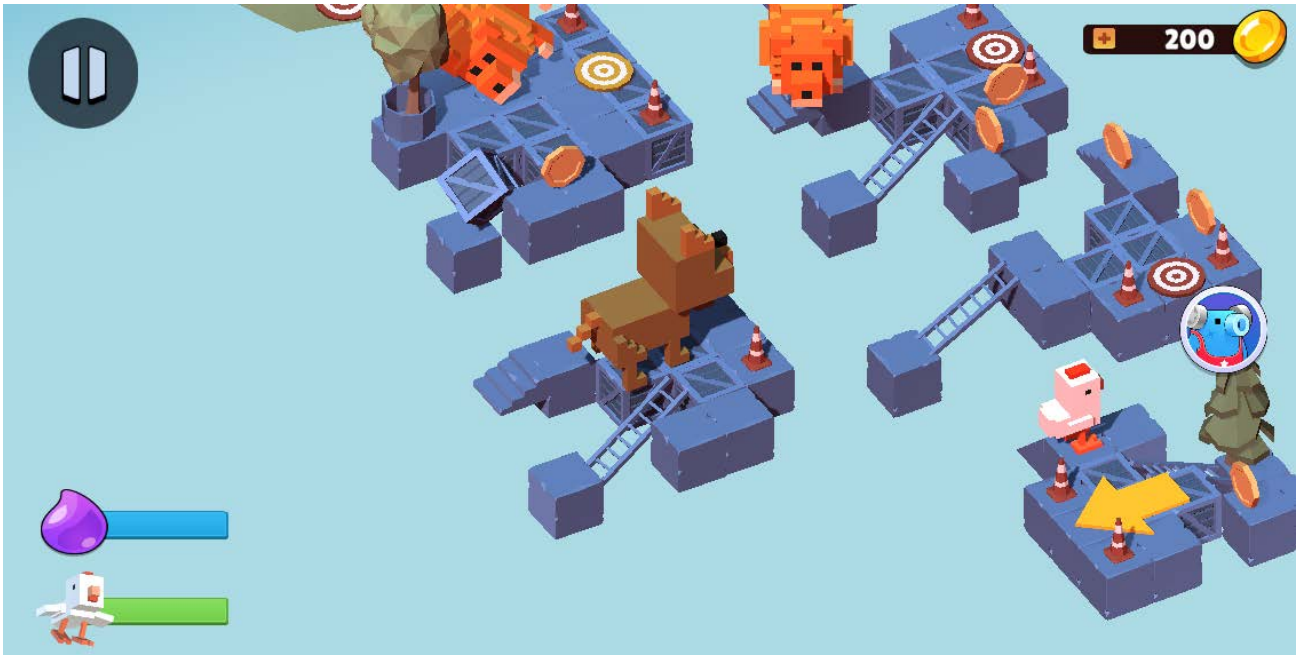

Рисунок 22 – Скриншот игрового процесса

Из пользовательского интерфейса было реализовано:

1) Меню паузы представлено на рисунке 23. При входе в меню паузы игра продолжается для всех остальных игроков, поскольку игра сетевая. Из него можно попасть в главное меню (Рисунок 27), продолжить игру, перезапустить игру (начать заново), а также выйти в меню настроек (Рисунок 26).
2) Экран проигрыша представлен на рисунке 24. Из него можно выйти в главное меню.
3) Экран выигрыша представлен на рисунке 25. Из него можно выйти в главное меню.
4) Экран настроек представлен на рисунке 26. Из него можно выйти в меню паузы, а также изменить некоторые игровые настройки.
5) Экран главного меню представлен на рисунке 27. Из него можно попасть в меню настроек, а также зайти в игру.

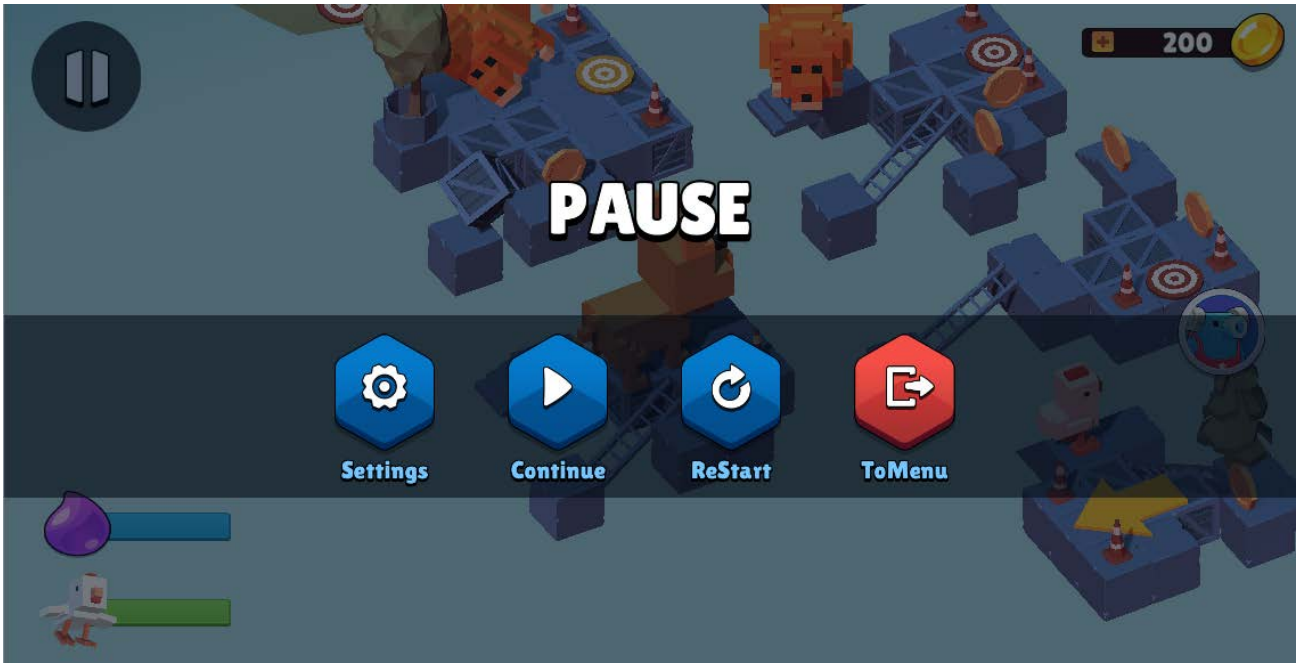

Рисунок 23 – Скриншот паузы

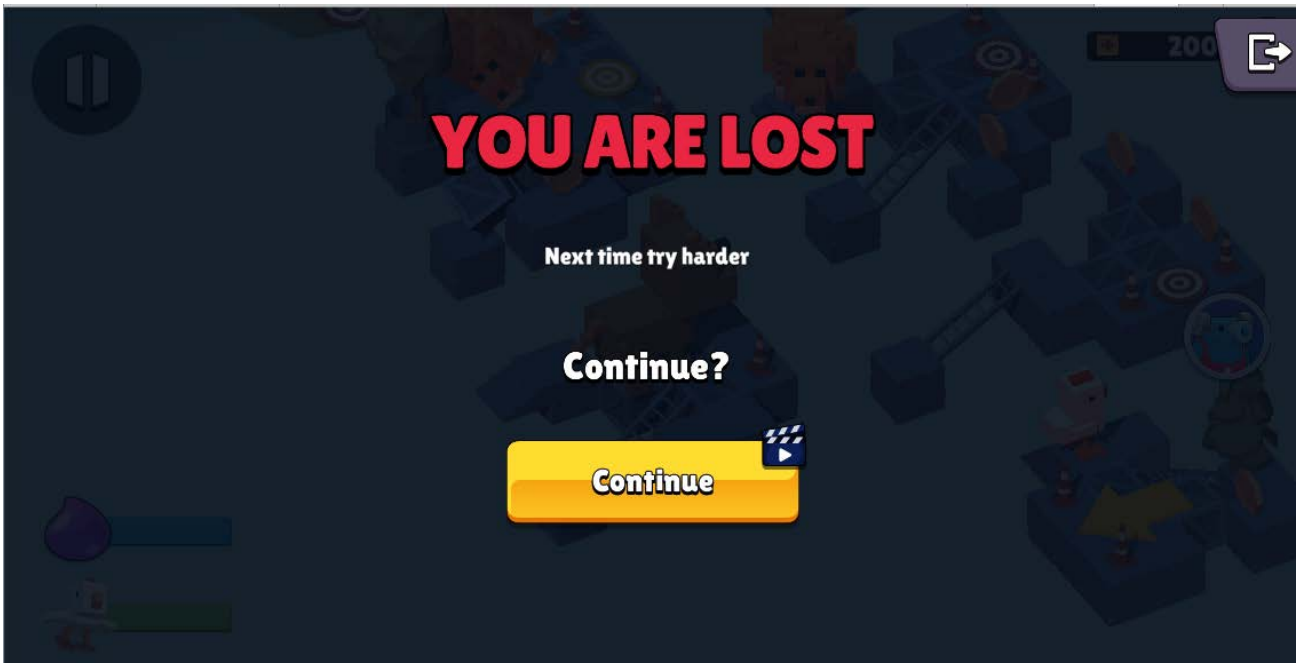

Рисунок 24 – Экран проигрыша

Рисунок 25 – Экран выигрыша

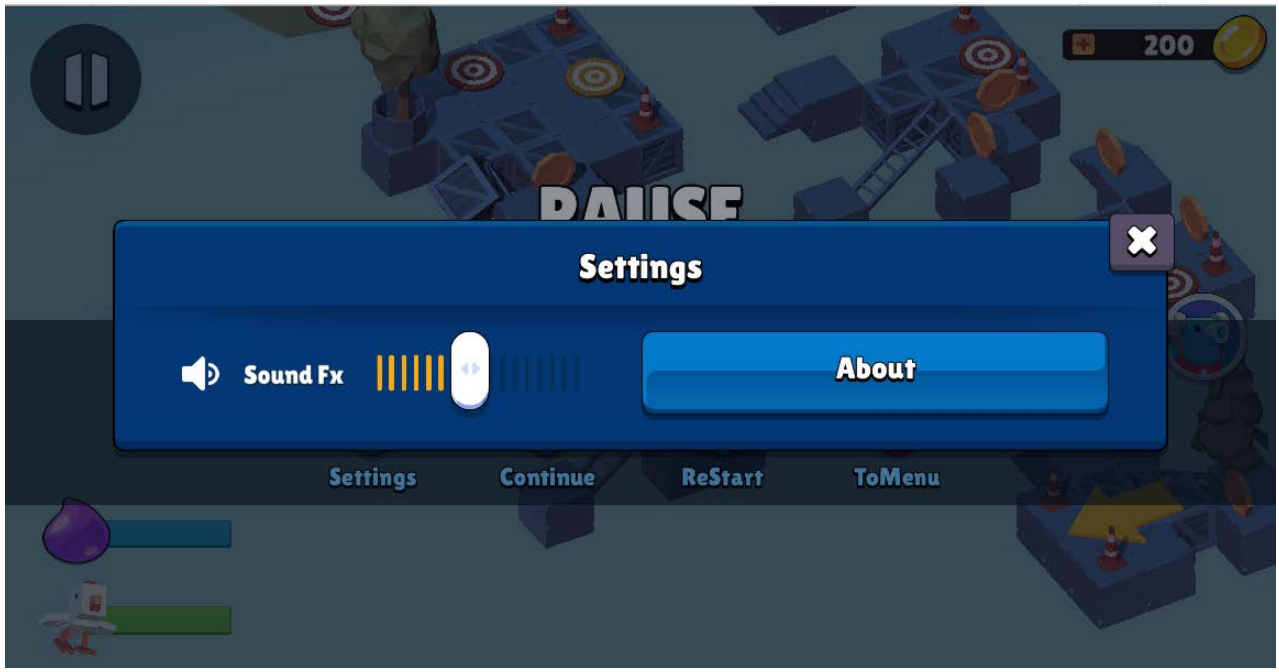

Рисунок 26 – Меню настроек

Рисунок 27 – Главное меню игры

На рисунке 28 представлена карта переходов между экранами. Двойными стрелками указаны экраны, из которых можно возвращаться назад на вызвавший экран.

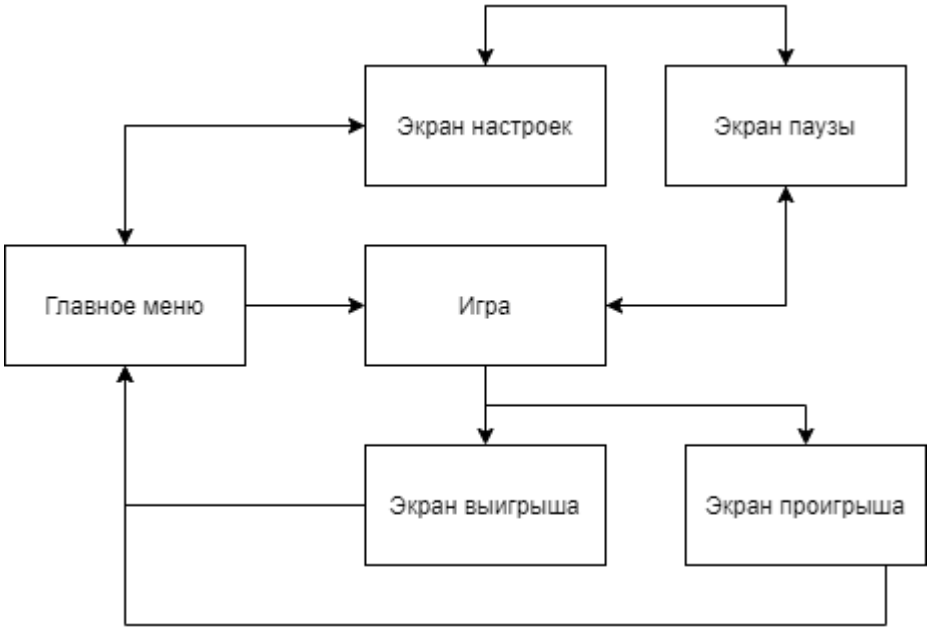

Рисунок 28 – Карта переходов между экранами

5. ТЕСТИРОВАНИЕ

Протестируем работу игры на мобильном устройстве. Для этого установим популярный плагин из `Unity Asset Store` для измерения `FPS` на сборках `Tayx Graphy` [18]. Для его внедрения в игру достаточно просто перетащить объект на игровую «сцену» и настроить измеряемые параметры.

Устройство, на котором замерялось количество кадров в секунду - `Leagoo Kiica Mix`.

Рисунок 29 – Тестирование игры на телефоне

На рисунке 29 мы видим, что количество кадров в секунду даже на мобильном устройстве более 60, чего достаточно для комфортной игры.

Для нагрузочного тестирования было запущено около 20 клиентов, проверена возможность заполнения комнат и открытия новых. Нагрузочное тестирование было пройдено успешно. После превышения игроков 5 в одной комнате, сервер возвращал последующим игрокам другие порты для подключения.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы была реализована 3D изометрическая игра в жанре Hyper-Casual с элементами мультиплеера. Представленное решение отличается от аналогов высокой производительностью работы. Даже на среднем по мощности устройстве наблюдалось количество кадров более 60, что достаточно для комфортной работы. Игра получилось достаточно веселой, чтобы игрокам хотелось в нее вернуться и продолжить играть.

В рамках работы были изучений такие технологии, как Mirror Networking, была проведена конфигурация виртуальной Unix машины в среде Azure Cloud, а также написан node.js сервер для организации «комнат».

В игре использовалась «воксельная» графика, состоящая из минимального количества полигонов. Камера была настроена на изометрический вид для привлекательного вида игры.

У проекта, могут быть множество дальнейших путей развития:

1. Добавление системы аккаунтов для сохранения прогресса игрока, отслеживания количества побед.
2. Создания портативной ПК версии игры с использованием общего сервера для возможности игры с различного типа устройств.
3. Добавление возможности кастомизации и персонализации персонажей для пользователя.
4. Возможность тратить внутриигровую валюту для большего вовлечения игроков;
5. Добавление новых карт.
6. Добавление внутриигровых событий.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Веб-сайт «Forbes». [Электронный ресурс]. URL: https://www.forbes.com/sites/kevinanderton/2020/09/29/hypercasual-female-gamers-are-taking-over-the-industry-infographic/?sh=7ce0a41d7c13 Дата обращения: 14.04.2021
2. Веб-сайт «Habr». [Электронный ресурс]. URL: https://habr.com/ru/post/455624/ Дата обращения: 14.04.2021
3. Веб-сайт «Unity». [Электронный ресурс]. URL: https://store.unity.com/compare-plans Дата обращения: 15.04.2021
4. Веб-сайт «Unreal Engine». [Электронный ресурс]. URL: https://www.unrealengine.com/en-US/faq Дата обращения: 15.04.2021
5. Веб-сайт «Firebase». [Электронный ресурс]. URL: https://firebase.google.com/ Дата обращения: 19.04.2021
6. Веб-сайт «Unity Analytics». [Электронный ресурс]. URL: https://unity.com/features/analytics Дата обращения: 19.04.2021
7. Веб-сайт «Photon Engine». [Электронный ресурс]. URL: https://www.photonengine.com/pun Дата обращения: 20.04.2021
8. Веб-сайт «Mirror-Networking». [Электронный ресурс]. URL: https://mirror-networking.com/ Дата обращения: 20.04.2021
9. Веб-сайт «Digital Ocean». [Электронный ресурс]. URL: https://www.digitalocean.com/ Дата обращения: 20.04.2021
10. Веб-сайт «Azure». [Электронный ресурс]. URL: https://azure.microsoft.com/en-us/services/virtual-machines/ Дата обращения: 20.04.2021
11. Веб-сайт «Node.js». [Электронный ресурс]. URL: https://nodejs.org/en/ Дата обращения: 20.04.2021
12. Веб-сайт «Asp.Net». [Электронный ресурс]. URL: https://dotnet.microsoft.com/apps/aspnet Дата обращения: 20.04.2021
13. Веб-сайт «unity.com». [Электронный ресурс]. URL: https://unity.com/ru/products/machine-learning-agents Дата обращения: 26.05.2021
14. Веб-сайт «Medium.com». [Электронный ресурс]. URL: https://medium.com/ml2vec/reinforcement-deep-q-learning-for-playing-a-game-in-unity-d2577fb50a81 Дата обращения: 26.05.2021

15. Веб-сайт «socket.io». [Электронный ресурс]. URL: https://socket.io/ Дата обращения: 05.06.2021
16. Веб-сайт «Photon engine dashboard». [Электронный ресурс]. URL: https://dashboard.photonengine.com/ Дата обращения: 05.06.2021
17. Веб-сайт «Photon engine dashboard». [Электронный ресурс]. URL: https://github.com/floatinghotpot/socket.io-unity Дата обращения: 05.06.2021
18. Веб-сайт «Asset store Tayx graphy package». [Электронный ресурс]. URL: https://assetstore.unity.com/packages/tools/gui/graphy-ultimate-fps-counter-stats-monitor-debugger-105778 Дата обращения: 06.06.2021
19. Веб-сайт «Github UniRx package». [Электронный ресурс]. URL: https://github.com/neuecc/UniRx Дата обращения: 06.06.2021
20. Веб-сайт «Github Zenject package». [Электронный ресурс]. URL: https://github.com/modesttree/Zenject Дата обращения: 06.06.2021
21. Design principles and design patterns. / Роберт Мартин. - https://fi.ort.edu.uy/innovaportal/file/2032/1/design_principles.pdf. – С. 34
22. Веб-сайт «Unity Cinemachine package». [Электронный ресурс]. URL: https://unity.com/unity/features/editor/art-and-design/cinemachine Дата обращения: 06.06.2021
23. Веб-сайт «DOTween unity package». [Электронный ресурс]. URL: http://dotween.demigiant.com/ Дата обращения: 06.06.2021
24. Веб-сайт «Hierarchy 2 unity package». [Электронный ресурс]. URL: https://github.com/truongnguyentungduy/hierarchy-2 Дата обращения: 06.06.2021

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОЕКТА UNITY

Листинг А.1 – Базовый класс Player Controller для управления персонажами

```
using Mirror;
using MyBox;
using System.Collections;
using UniRx;
using UnityEngine;
using Zenject;
using Cinemachine;

public interface IJumper {
    void Jump(Vector3 jumpForce);
}

public interface IMover {
    void Move(Vector3 movement);
}

public interface IFlyer {
    void Fly(float capacity);
}

public abstract class PlayerController : NetworkBehaviour, IJumper, IMover,
IFlyer {
    /// <summary>
    /// Dependency injection Zenject фреймворк. Бесшовное предоставление ссылок
для объекта
    /// </summary>
    [Inject]
    private IPlayerMover playerMover;

    [Inject]
    [Tooltip("Used to call game manager specific methods")]
    private ICollisionManager collisionManager;

    [Header("Camera movement")]
    [SerializeField]
    private bool cameraRelativeMovement = true;

    [SerializeField]
    [ConditionalField("cameraRelativeMovement")]
    private Camera cam;

    [Header("Audio events")]
    [SerializeField]
    private AudioEvent moveClip;
```

```
    [SerializeField]
    private AudioEvent jumpClip;

    [SerializeField]
    private AudioEvent flyClip;

    [Header("Movement")]
    public float movementSpeed = 3;
    public float attackForce;
    public float jumpFoce;
    private Animator anim;
    private Rigidbody rb;
    private BoxCollider boxCollider;
    private AudioSource playerSource;

    private bool flying = false;

    /// <summary>
    /// Определяют в наследуемых классах, доступен ли полет или прыжок для
животного
    /// </summary>
    #region Allowing
    protected abstract bool IsJumpAllowed();

    protected abstract bool IsFlyAllowed();
    #endregion

    public virtual void Start() {
        if (!isLocalPlayer)
            return;

        if (cam == null) {
            cam = Camera.main;
        }

        var brain = cam.GetComponent<CinemachineBrain>();

        var vCam = (CinemachineVirtualCamera)brain.ActiveVirtualCamera;

        vCam.Follow = transform;

        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody>();
        boxCollider = GetComponent<BoxCollider>();
        playerSource = GetComponent<AudioSource>();
    }

    void Update() {
        if (!isLocalPlayer)
```

```
     return;

    ControllPlayer();
}

public virtual void ControllPlayer() {
    var move = playerMover.Move;

    float moveHorizontal = move.x;
    float moveVertical = move.y;

    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

    Move(movement);

    if (playerMover.Jump) {
        Jump(new Vector3(0f, jumpFoce, 0f));
    }

    if (playerMover.Attack) {
        var attackDirection = transform.forward;
        Attack(attackDirection);
    }
}

/// <summary>
/// Вызывается, когда игрок нажимает на атаку на устройствах ввода
/// </summary>
/// <param name="attackDirection"></param>
private void Attack(Vector3 attackDirection) {
    var attack = attackDirection * attackForce;

    rb.AddForce(attack, ForceMode.Impulse);
}

/// <summary>
/// Вызывается, когда игрок нажимает на ввод, заставляя игрока двигаться
/// </summary>
/// <param name="movement">
/// Текущее движение, снятое с ввода
/// </param>
public virtual void Move(Vector3 movement) {
    if (movement != Vector3.zero) {
        if (cameraRelativeMovement) {
            movement = CameraRelativeMovement(movement);
        }
```

```
    transform.rotation = Quaternion.Slerp(transform.rotation,
Quaternion.LookRotation(movement), 0.30f);

            transform.Translate(movement * movementSpeed * Time.deltaTime,
Space.World);

            if (!flying) {
                moveClip.Play(playerSource);
            }
        }

        anim.SetFloat("Walk", movement.magnitude * movementSpeed);
    }

    /// <summary>
    /// Используется, чтобы игрок двигался в координатах камеры
    /// Векторы forward и right берутся из transform камеры
    /// </summary>
    /// <param name="movement"></param>
    /// <returns></returns>
    private Vector3 CameraRelativeMovement(Vector3 movement) {
        var camTransform = cam.transform;

        var forward = camTransform.forward;
        var right = camTransform.right;
        forward.y = 0f;
        right.y = 0f;

        forward.Normalize();
        right.Normalize();

        return forward * movement.z + right * movement.x;
    }

    public virtual void OnTriggerEnter(Collider other) {
        var gameObj = other.gameObject;

        if (gameObj.HasTag("Killer")) {
            collisionManager.OnPlayerDeath(gameObject);
        }
        else if (gameObj.HasTag("Jump")) {
            var jumper = gameObj.GetComponent<JumperLogic>() as IJumpPlatform;

            Firebase.Analytics.FirebaseAnalytics.LogEvent("custom_jump_event");

            jumper.OnTryJump(this);
        }
        else if (gameObj.HasTag("Fly")) {
            var flyer = gameObj.GetComponent<FlyerLogic>() as IFlyPlatform;
```

```
            flyer.OnTryFly(this);
        }
    }

    #region Jump
    /// <summary>
    /// Вызывается, когда игрок наступает на прыгающую платформу
    /// </summary>
    /// <param name="jumpForce"></param>
    public void Jump(Vector3 jumpForce) {
        if (IsJumpAllowed()) {
            rb.AddForce(jumpForce, ForceMode.Impulse);

            anim.SetTrigger("jump");

            jumpClip.Play(playerSource);
        }
    }
    #endregion

    #region Fly
    /// <summary>
    /// Вызывается, когда игрок наступает на летающую платформу
    /// </summary>
    /// <param name="capacity"></param>
    public void Fly(float capacity) {
        if (IsFlyAllowed()) {
            anim.SetTrigger("wantFly");


MainThreadDispatcher.StartUpdateMicroCoroutine(FlyCoroutine(capacity));
        }
    }

    // Асинхронный метод полета
    private IEnumerator FlyCoroutine(float capacity) {
        while (!playerMover.Press) {
            yield return null;
        }

        rb.useGravity = false;
        boxCollider.isTrigger = true;
        flying = true;

        while (capacity > 0.18f && playerMover.Press) {
            anim.SetFloat("fly", capacity);

            collisionManager.OnPlayerFlying(capacity);
```

```
            capacity -= Time.deltaTime * movementSpeed * 9;
            flyClip.Play(playerSource);

            yield return null;
        }

        flying = false;
        anim.SetTrigger("dontWantFly");
        anim.SetFloat("fly", 0);

        rb.useGravity = true;
        boxCollider.isTrigger = false;
    }
    #endregion
}
```

Листинг А.2 – Pigeon Player Controller класс для управления пингвином

```
public class PigeonPlayerController : PlayerController {
    protected override bool IsFlyAllowed() {
         return true;
    }

    protected override bool IsJumpAllowed() {
        return false;
    }
}
```

Листинг А.3 – Dog Player Controller класс для управления собакой

```
public class DogPlayerController : PlayerController {
    protected override bool IsFlyAllowed() {
        return false;
    }

    protected override bool IsJumpAllowed() {
        return true;
    }
}
```

Листинг А.4 – Chicken Player Controller класс для управления курицей

```
public class ChickenPlayerController : PlayerController {
    protected override bool IsFlyAllowed() {
        return true;
    }

    protected override bool IsJumpAllowed() {
        return false;
    }
```

```
}
```

Листинг А.5 – Lion Player Controller класс для управления львом

```
public class LionPlayerController : PlayerController {
    protected override bool IsFlyAllowed() {
        return false;
    }

    protected override bool IsJumpAllowed() {
        return true;
    }
}
```

Листинг А.6 – Flyer Logic логика работы платформы для взлета

```
using UnityEngine;

public interface IFlyPlatform {
    void OnTryFly(IFlyer flyer);

    void PlayFlyAnimation();
}

public class FlyerLogic : MonoBehaviour, IFlyPlatform {
    #region Assignable
    [SerializeField]
    private Animator animator;

    [SerializeField]
    private int capacity;
    #endregion

    // Вызывается при столкновении с платформой полета из PlayerController
    public void OnTryFly(IFlyer flyer) {
        PlayFlyAnimation();

        flyer.Fly(capacity);
    }

    // Проигрывает анимацию прыжка на платформе полета
    public void PlayFlyAnimation() {
        animator.SetTrigger("Jump");
    }
}
```

Листинг А.7 – Jumper Logic логика работы платформы для прыжка

```
using MyBox;
using UnityEngine;
```

```
public interface IJumpPlatform {
    bool Available {

       get;
    }

    void MakeUnavalaibleVisible();

    void OnTryJump(IJumper jumper);

    void PlayJumpAnimation();
}

public class JumperLogic : MonoBehaviour, IJumpPlatform {
    #region Assignable
    [SerializeField]
    private Animator animator;

    [SerializeField]
    private float force;

    [SerializeField]
    private bool infinityJumps;

    [ConditionalField("infinityJumps", true)]
    public int JumpCount = 1;

    #endregion

    // Используется для проверки доступности прыжка для предотвращения
повторного прыжка до окончания предыдущего
    public bool Available {
        get {
            return JumpCount > 0;
        }
    }

    public void MakeUnavalaibleVisible() {
        animator.SetTrigger("Unavalaible");
    }

    // Вызывается при столкновении с платформой прыжка из PlayerController
    public void OnTryJump(IJumper jumper) {
        if (Available) {
            PlayJumpAnimation();

            jumper.Jump(new Vector3(0, force, 0));

            if (!infinityJumps) {
```

```
                JumpCount--;
            }
        }

        if (!Available) {
            MakeUnavalaibleVisible();

      }
    }

    public void PlayJumpAnimation() {
        animator.SetTrigger("Jump");
    }
}
```

Листинг А.8 – Game Manager главный класс игры

```
using Mirror;
using MyBox;
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public interface ICoinHandler {
    void SaveCoins();

    void BuyCoins(float coins);
}

// Интерфейс для отслеживания столкновений с объектами
public interface ICollisionManager {
    void OnCoinGrab(GameObject coin);

    void OnWin(string levelName, Vector3 winPosition);

    void OnPlayerFlying(float capacity);

    void OnPlayerDeath(GameObject player);

    void OnFoodReached(GameObject food);

    void OnPlayerHungerChanged(float hunger);
}

public class GameManager : MonoBehaviour, ICoinHandler, ICollisionManager {
    [SerializeField]
    private FadeManager fadeManager;

    [SerializeField]
```

```
private Vector3 initalPosition;

#region Bars
[SerializeField]
private ProgressBar staminaFlyBar;

[SerializeField]
private ProgressBar hungerBar;
[SerializeField]
private CoinBar coinBar;

[SerializeField]
private GameObject heroPicker;
#endregion

[SerializeField]
private CoinsSettings coinsSettings;

[SerializeField]
private GameObject looseScreen;

[SerializeField]
private GameObject winScreen;

[ReadOnly]
public int coins;

public bool showDebugInfo;

public int loadDelay;

// Запускает сервер при Headless сборке
private void StartSever() {
    var manager = GetComponent<NetworkManager>();

    manager.StartServer();

    Debug.Log("Starting server");
}

private void Awake() {
    if (showDebugInfo) {
        SceneManager.LoadScene("Shared", LoadSceneMode.Additive);

        fadeManager.Unfade(null);
    }

    var coinsInSettings = coinsSettings.Coins;
```

```
        coinBar.ChangeValue(coinsInSettings);
    }

#region Scene loaders
    private void LoadScene(string levelName) {
        StartCoroutine(LoadSceneAsync(levelName));
    }

    private IEnumerator LoadSceneAsync(string levelName) {
        yield return new WaitForSeconds(loadDelay);

        SceneManager.LoadScene(levelName, LoadSceneMode.Single);
    }

    private void ReloadScene() {

      StartCoroutine(ReloadSceneAsync());
    }

    private IEnumerator ReloadSceneAsync() {
        yield return new WaitForSeconds(loadDelay);

        var index = SceneManager.GetActiveScene().buildIndex;

        SceneManager.LoadScene(index, LoadSceneMode.Single);
    }
#endregion

#region Player event handlers
    /// <summary>
    /// Вызывается, когда игрок умирает открывает экран смерти, с которого можно
    /// в меню
    /// </summary>
    public void OnPlayerDeath(GameObject player) {
        AudioManager.Instance.PlayDeathEvent();
        looseScreen.SetActive(true);
    }

    public void OnPickHero() {
        heroPicker.SetActive(true);
    }

    /// <summary>
    /// Вызывается, когда игрок наступает на монету
    /// </summary>
    /// <param name="coin">Coin GameObject</param>
    public void OnCoinGrab(GameObject coin) {
        var position = coin.transform.position;
```

```
        EffectManager.Instance.PlayCoinEffect(position);
        AudioManager.Instance.PlayCoinEvent();

        coin.SetActive(false);

        coins++;

        coinBar.ChangeValue(coinsSettings.Coins + coins);
    }

    public void OnPlayerHungerChanged(float hunger) {
        hungerBar.ChangeValue(hunger);
    }

    public void OnFoodReached(GameObject food) {
        var position = food.transform.position;
        EffectManager.Instance.PlayFoodEffect(position);
        AudioManager.Instance.PlayFoodEvent();

        food.SetActive(false);
    }

    // Вызывается, когда игрок выигрывает. Активирует экран выигрыша, с которого
можно перейти в главное меню
    public void OnWin(string levelName, Vector3 winPosition) {
        EffectManager.Instance.PlayWinEffect(winPosition);
        AudioManager.Instance.PlayWinEvent();

        SaveCoins();
        winScreen.SetActive(true);
    }

    public void OnPlayerFlying(float capacity) {
        staminaFlyBar.ChangeValue(capacity);
    }
#endregion

#region Coins logic
    public void SaveCoins() {
        coinsSettings.SaveCoins(coins);
    }

    public void BuyCoins(float coins) {
        coinsSettings.SaveCoins(coins);
    }
#endregion
}
```

Листинг А.9 – Network Manager My класс для организации сетевого взаимодействия и выбора персонажа

```
using kcp2k;
using Mirror;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.Networking;

[System.Serializable]
public class ClassTypesDictionary : SerializableDictionary<ClassTypes,
GameObject> {

}

public class Utils {
    // Получает аргументы командной строки при запуске из node.js сервера
    public static string GetArg(string name) {
        var args = System.Environment.GetCommandLineArgs();
        for (int i = 0; i < args.Length; i++) {
            if (args[i] == name && args.Length > i + 1) {
                return args[i + 1];
            }
        }
        return null;
    }
}

public class NetworkManagerMy : NetworkManager {
    [SerializeField]
    private ChosenClass chosenClass;

    [SerializeField]
    private ClassTypesDictionary players;

    [SerializeField]
    private UnityEvent OnPlayerDisconnect;

    public PortSettings portSettings;

    public ushort defaultPort;

    public string nodeServerUrl;

    private Dictionary<int, NetworkConnection> _connections;
```

```
    // Вызывается при старте сервера. При Headless сборке стартуется
автоматически
    public override void OnStartServer() {
        SetServerPort();
        base.OnStartServer();

        _connections = new Dictionary<int, NetworkConnection>();
    }

    private void SetServerPort() {
        var port = Utils.GetArg("port");

        if (port != null) {
            try {
                var portNum = ushort.Parse(port);

                (transport as KcpTransport).Port = portNum;
            }
            catch (System.Exception e) {
                Debug.LogError(e.Message);
            }
            return;
        }

        (transport as KcpTransport).Port = defaultPort;
    }

    // Вызывается при подключении игрока к серверу
    public override void OnServerAddPlayer(NetworkConnection conn) {
        _connections.Add(conn.connectionId, conn);
    }

    // Вызывается при отключении игрока от сервера
    public override void OnServerDisconnect(NetworkConnection conn) {
        base.OnServerDisconnect(conn);

        _connections.Remove(conn.connectionId);
    }

    // Вызывается после подключения игрока к серверу
    [Command]
    private void SpawnPlayer(int chosenPlayerId, int connId) {
        var chosenPlayer = players[(ClassTypes)chosenPlayerId];
        var conn = _connections[connId];

        Transform startPos = GetStartPosition();
        GameObject player = startPos != null
            ? Instantiate(chosenPlayer, startPos.position, startPos.rotation)
            : Instantiate(chosenPlayer);
```

```
        // instantiating a "Player" prefab gives it the name "Player(clone)"
        // => appending the connectionId is WAY more useful for debugging!
        player.name = $"{chosenPlayer.name} [connId={conn.connectionId}]";
        NetworkServer.AddPlayerForConnection(conn, player);
    }

    public override void OnClientConnect(NetworkConnection conn) {
        base.OnClientConnect(conn);

        playerPrefab = players[chosenClass.Class];

        SpawnPlayer((int)chosenClass.Class, conn.connectionId);
    }

    [Server]
    public void OnGameEnd() {
        NetworkServer.Shutdown();

        Application.Quit();
    }

    [Client]
    public void OnPortRetrieve() {
        StartCoroutine(GetPort());
    }

    // Клиентский асинхронный метод для получения порта из node.js сервера
    IEnumerator GetPort() {
        var request = new UnityWebRequest(nodeServerUrl);

        yield return request.SendWebRequest();

        if (request.result != UnityWebRequest.Result.Success) {
            Debug.Log(request.downloadHandler.text);
        }
        else {
            var portStr =
System.Text.Encoding.Default.GetString(request.downloadHandler.data);

            portSettings.port = int.Parse(portStr);
        }

        request.Dispose();
    }
}
```

Листинг А.10 – Coins Settings класс для хранения информации о собранных монетах

```
using MyBox;
using System.Collections;
using UnityEngine;

[CreateAssetMenu(menuName = "Coins")]
public class CoinsSettings : ScriptableObject {
    [SerializeField]
    [ReadOnly]
    private float coins;

    public float Coins {
        get {
            return coins;
        }
    }

    public void SaveCoins(float coins) {
        this.coins += coins;
    }
}
```

Листинг А.11 – Chosen Class класс для хранения информации о выбранном персонаже

```
using UnityEngine;
using MyBox;

[CreateAssetMenu(menuName = "Settings/Choosen class")]
public class ChosenClass : ScriptableObject {
    [ReadOnly]
    public ClassTypes Class;
}
```

Листинг А.12 – Base Bar базовый класс для работы со статус барами

```
using MyBox;
using System;
using UniRx;
using UnityEngine;

public abstract class BaseBar : MonoBehaviour, IBarShow {
    [SerializeField]
    [ReadOnly]
    protected float value;

    [SerializeField]
    protected bool alwaysVisible;
```

```
    [SerializeField]
    [ConditionalField("alwaysVisible", true)]
    protected float fadeSeconds;

    private float lastChangeTime;

    private bool firstChange = true;

    private IDisposable everyTwoSeconds;

    public abstract float MaxValue {
        get; set;
    }

    public virtual void Start() {
        if (!alwaysVisible) {
            gameObject.SetActive(false);

            FadeOutImmediately();
        }
    }

    // Устанавливает таймер, после которого бар пропадает
    public void SetTimer() {
        if (!alwaysVisible) {
            if (everyTwoSeconds != null) {
                everyTwoSeconds.Dispose();
            }

            if (firstChange) {
                firstChange = false;

                gameObject.SetActive(true);

                FadeIn();
            }

            everyTwoSeconds =
Observable.Timer(TimeSpan.FromSeconds(1.1f)).Subscribe(time => {
                var sub = Time.time - lastChangeTime;

                if (sub > 1f) {
                    FadeOut();

                    everyTwoSeconds.Dispose();

                    firstChange = true;
                }
            });
```

```
            }
        }

        public virtual void ChangeValue(float value) {
            SetTimer();

            lastChangeTime = Time.time;
        }

        public abstract void ChangeMaxValue(float value);

        public abstract void FadeIn();

        public abstract void FadeOut();

        public abstract void FadeOutImmediately();
    }
```

Листинг А.13 – Simple Fader класс для затемнения экрана

```
using DG.Tweening;

using UnityEngine;
using UnityEngine.UI;

public interface IBarShow {
    void ChangeValue(float value);

    void ChangeMaxValue(float value);
}

public interface IFader {
    void FadeIn(Image whatToFade, float seconds);

    void FadeOut(Image whatToFade, float seconds);
}

public class SimpleFader : IFader {
    private static SimpleFader _instance;

    public static SimpleFader Instance {
        get {
            if (_instance == null) {
                _instance = new SimpleFader();
            }

            return _instance;
        }
    }
```

```
    // Анимация затухания
    public void FadeIn(Image whatToFade, float seconds) {
        whatToFade.DOFade(1, seconds);
    }

    // Анимация, которая запускается при старте приложения
    public void FadeOut(Image whatToFade, float seconds) {
        whatToFade.DOFade(0, seconds);
    }
}
```

Листинг А.14 – Progress Bar универсальный класс для баров здоровья и маны

```
public class ProgressBar : BaseBar {
    #region Images
    [SerializeField]
    private Image heroImage;

    [SerializeField]
    private Image background;

    [SerializeField]
    private Image fill;
    #endregion

    [SerializeField]
    private Slider progress;

    public override float MaxValue {
        get {
            return progress.maxValue;
        }
        set {
            progress.maxValue = value;
        }
    }

    public override void ChangeMaxValue(float maxValue) {
        progress.maxValue = maxValue;
    }

    public override void ChangeValue(float newValue) {
        base.ChangeValue(newValue);

        progress.value = value = newValue;
    }

    public override void FadeIn() {
```

```
        var fader = SimpleFader.Instance;

        fader.FadeIn(heroImage, fadeSeconds);
        fader.FadeIn(background, fadeSeconds);
        fader.FadeIn(fill, fadeSeconds);
    }

    public override void FadeOut() {
        var fader = SimpleFader.Instance;

        fader.FadeOut(heroImage, fadeSeconds);
        fader.FadeOut(background, fadeSeconds);
        fader.FadeOut(fill, fadeSeconds);
    }

    public override void FadeOutImmediately() {
        gameObject.SetActive(false);

        var transparent = Color.white;
        transparent.a = 0;

        heroImage.color = transparent;
        background.color = transparent;
        fill.color = transparent;
    }
}
```

Листинг А.15 – Skill Type базовый класс для умений персонажей

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

public enum LayerType {
    Damagable,
    Destructable
}

public abstract class SkillType : ScriptableObject {
    public float manaCost;
    public float cooldown;
    public SimpleEffectEvent effect;
    public SimpleAudioEvent sound;

    public abstract void Play(GameObject gameObject, AudioSource source, Vector3
position);
}

[System.Serializable]
```

```
public class LayerHelper {
    [SerializeField] public LayerMask mask;

    public LayerType type;

    public static int GetMask(List<LayerHelper> affectedLayers) {
        int mask = 0;

        foreach (var layer in affectedLayers) {
            mask |= layer.mask;
        }

        return mask;
    }
}
```

Листинг А.16 – Multiple Attack умение для нескольких атак

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[CreateAssetMenu(menuName = "Content/Skills/Multiple Attacks")]
public class MultipleAttack : SkillType {
    public List<LayerHelper> affectedLayers;
    public float castRadius;
    public float damage;

    public override void Play(GameObject gameObject, AudioSource source, Vector3
position) {
        var colliders = Physics.OverlapBoxAll(position, new Vector3(castRadius,
castRadius, castRadius), 0f);

        if (colliders.Length > 0) {
            foreach (var collider in colliders) {
                foreach (var layer in affectedLayers) {
                    if (gameObject.GetInstanceID() !=
collider.transform.root.gameObject.GetInstanceID() &&
                        (1 << collider.gameObject.layer) == layer.mask) {
                        if (layer.type == LayerType.Damagable) {
                            var health =
collider.gameObject.transform.root.GetComponent<Health>();

                            effect.Play(collider.transform.root.position);
                            health.TakeDamage(damage);
                        }
                        else if (layer.type == LayerType.Destructable) {
                            Destroy(collider.gameObject);
                        }
```

```
                }
            }
        }
    }

    sound.Play(source);
  }
}
```

Листинг А.17 Heavy Attack умение для тяжелой атаки

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

[CreateAssetMenu(menuName = "Content/Skills/Heavy Attack")]
public class HeavyAttack : SkillType {
    public float damage;
    public List<LayerHelper> affectedLayers;
    public float castRadius;


public override void Play(GameObject gameObject, AudioSource source, Vector3
position) {
        var colliders = Physics.OverlapBoxAll(position, new Vector3(castRadius,
castRadius, castRadius), 0f);

        if (colliders.Length > 0) {
            foreach (var collider in colliders) {
                foreach (var layer in affectedLayers) {
                    if (gameObject.GetInstanceID() !=
collider.transform.root.gameObject.GetInstanceID() &&
                        (1 << collider.gameObject.layer) == layer.mask) {
                        if (layer.type == LayerType.Damagable) {
                            var health =
collider.gameObject.transform.root.GetComponent<Health>();

                            effect.Play(collider.transform.root.position);
                            health.TakeDamage(damage);
                        }
                        else if (layer.type == LayerType.Destructable) {
                            Destroy(collider.gameObject);
                        }
                    }
                }
            }
        }

        sound.Play(source);
    }
```

```
}
```

Листинг А.18 - Consumable, Mana и Health скрипты

```
using MyBox;
using SukharevShared;
using System;
using System.Collections;
using System.Collections.Generic;
using UniRx;
using UnityEngine;
using UnityEngine.Events;

public abstract class Consumable : MonoBehaviour {
    [SerializeField] protected float value;
    [SerializeField] private bool hasMax = false;

    [ConditionalField(nameof(hasMax))]
    [SerializeField] protected float max;

    public UnityEvent OnEnded;
    public UnityEventFloat OnChange;
    public UnityEventFloat OnMaxChange;
    public UnityEvent OnFull;

    public virtual void Start() {
        if (hasMax) {
            OnMaxChange.Invoke(max);
        }
        else {
            OnMaxChange.Invoke(value);
        }

        OnChange.Invoke(value);
    }

    public void Set(float value) {
        this.value = value;

        OnChange.Invoke(value);
    }

    public void SetMax(float max) {
        hasMax = true;
        this.max = max;
        OnMaxChange.Invoke(max);
    }
```

```
  public void TakeDamage(float damage) {
        value -= damage;

        OnChange.Invoke(value);

        if (value < 0.01f) {
            OnEnded.Invoke();
        }
    }

    public void TakeHealth(float health) {
        value += health;

        if (hasMax && value > max) {
            value = max;

            OnFull.Invoke();
        }
        OnChange.Invoke(value);
    }
}
public class Mana : Consumable {
    public float recoveryAmount;

    private IDisposable recovery;

    public override void Start() {
        base.Start();

        OnFull.AddListener(OnManaFull);
        OnChange.AddListener(OnManaChange);

        recovery = Observable.Interval(TimeSpan.FromSeconds(1)).Subscribe(_ => {
            TakeHealth(recoveryAmount);
        });
    }

    private void OnDisable() {
        recovery?.Dispose();
    }

    public bool HasRequired(float mana) {
        return value > mana;
    }

    private void OnManaChange(float current) {
        if (recovery == null && current < max) {
            recovery = Observable.Interval(TimeSpan.FromSeconds(1)).Subscribe(_
=> {
```

```
                TakeHealth(recoveryAmount);
            });
        }
    }

    private void OnManaFull() {
        if (recovery != null) {
            recovery.Dispose();
            recovery = null;
        }
    }
}

public class Health : Consumable {

}
```

Листинг А.19 – End Game Checker серверный класс

```
using Mirror;
using UnityEngine;
using UniRx;

public class EndGameChecker : NetworkBehaviour {
    [SerializeField] private NetworkManagerMy networkManager;

    /// <summary>
    /// Максимальное время игры в секундах
    /// </summary>
    public int maxGameTime;

    // Start is called before the first frame update
    void Start() {
        if (isServer) {

Observable.Timer(System.TimeSpan.FromSeconds(maxGameTime)).Subscribe(_ => {
                if (NetworkServer.active)
                networkManager.OnGameEnd();
            });
        }
    }

    [Command]
    public void GameEnd() {
        networkManager.OnGameEnd();
    }
}
```

Листинг А.20 – Port Settings класс для хранения порта между сценами

```
using UnityEngine;

[CreateAssetMenu(menuName = "Settings/PortSettings")]
public class PortSettings : ScriptableObject {
    public int port;
}
```

ЛистингА.21 – AudioManager класс для управления звуками в игре

```
using UnityEngine;

public class AudioManager : MonoBehaviour {
    [Header("Background")]
    public AudioSource backgroundSource;

    public AudioEvent backgroundEvent;

    [Header("Interactions")]
    public AudioSource objectInteractionSource;
    public AudioEvent coinEvent;

    public AudioEvent foodEvent;

    public AudioEvent winEvent;

    public AudioEvent deathEvent;

    public static AudioManager Instance;

    void Awake() {
        if (Instance == null) {
            Instance = this;
        }
        else if (Instance == this) {
            Destroy(gameObject);
        }
    }

    void Start() {
        backgroundEvent.Play(backgroundSource);
    }
    public void PlayCoinEvent() {
        coinEvent.Play(objectInteractionSource);
    }

    public void PlayFoodEvent() {
        foodEvent.Play(objectInteractionSource);
```

```
      }

    public void PlayWinEvent() {
        winEvent.Play(objectInteractionSource);
    }

    public void PlayDeathEvent() {
        deathEvent.Play(objectInteractionSource);
    }
}
```

Листинг А.22 – ModuleInstaller класс для Zenject инструмента

```
using SukharevShared;
using UnityEngine;
using Zenject;

public enum InputType {
    JoystickMover,
    KeyboardMover
}

public class ModuleInstaller : MonoInstaller {
    public InputType inputType;

    [SerializeField]
    private GameManager gameManager;

    [SerializeField]
    private GameObject controls;

    public override void InstallBindings() {
        var typeInput = EnumResolver<InputType,
IPlayerMover>.GetType(inputType);

        if (inputType == InputType.KeyboardMover) {
            controls.SetActive(false);
        }

      Container.Bind<IPlayerMover>().To(typeInput.UnderlyingSystemType).AsSingle
      ();
      Container.Bind<ICollisionManager>().FromInstance(gameManager).AsSingle();
    }
}
```

Листинг 23 – InputManager класс с управлением

```
using UnityEngine;
```

```
using Zenject;

/// <summary>
/// So smart
/// </summary>
public interface IPlayerMover {
    Vector2 Move {
        get; set;
    }

    bool Jump {
        get; set;
    }

    bool Attack {
        get; set;
    }

    bool Press {
        get; set;
    }
}

public class KeyboardMover : IPlayerMover {
    public Vector2 Move {
        get {
            var horizontal = Input.GetAxisRaw("Horizontal");
            var vertical = Input.GetAxisRaw("Vertical");

            return new Vector2(horizontal, vertical);
        }
        set {
        }
    }
    public bool Jump {
        get {
            return Input.GetAxisRaw("Jump") > 0;
        }
        set {
        }
    }
    public bool Attack {
        get {
            return Input.GetMouseButtonDown(0);
        }
        set {
        }
    }

    public bool Press {
```

```
        get {
            return Input.GetKey(KeyCode.W);
        }
        set {
        }
    }
}

/// <summary>
/// Don't think that it's really good
/// </summary>
/// <remarks>
/// It's value sets in InputHandler class
/// </remarks>
public class JoystickMover : IPlayerMover {
    public Vector2 Move {
        get; set;
    }

    public bool Jump {
        get; set;
    }

    public bool Attack {
        get; set;
    }
    public bool Press {
        get; set;
    }
}

public class InputManager : MonoBehaviour {
    [Inject]
    private IPlayerMover playerMover;

    public void JoystickInput_Changed(Vector2 value) {
        playerMover.Move = value;
    }

    public void JoystickPressState_Change(bool value) {
        playerMover.Press = value;
    }

    public void JoystickButtonA_Clicked() {
        playerMover.Attack = true;
    }

    public void JoystickButtonA_Up() {
        playerMover.Attack = false;
    }
```

```
    public void JoystickButtonB_Clicked() {
        playerMover.Jump = true;
    }

    public void JoystickButtonB_Up() {
        playerMover.Jump = false;
    }
}
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД NODE.JS СЕРВЕРА

Листинг В.1 – Исходный код node.js сервера

```
const express = require('express');
const app = express();
const { spawn } = require('child_process');

const maxUsersCount = 5
const maxRoomsCount = 5
const serverPort = 3000
const startPort = serverPort + 1
const rooms = []

class Room {
    constructor(maxUsers, childProcess, port) {
        this.currentUsers = 0;
        this.maxUsers = maxUsers;
        this.childProcess = childProcess;
        this.port = port;
    }

    addUser() {
        this.currentUsers++;
    }

    isFree() {
        return this.currentUsers < this.maxUsers;
    }
}

createRooms();

app.get('/', (req, res) => {
    var room = findFreeRoom();

    if (room != null) {
        res.status(404).send("No free rooms");
    } else {
      room.addUser();
        res.send(`${room.port}`);
    }
});
```

```
function createRooms() {
    for (i = 0; i < maxRoomsCount; i++) {
        const roomPort = startPort + i;

        var prc = spawn(“Build/unityBuild”,  [`-port=${roomPort}`]);

        prc.stdout.setEncoding('utf8');
        prc.stdout.on('data', function (data) {
            var str = data.toString()
            var lines = str.split(/(\r?\n)/g);
            console.log(lines.join(""));
        });

        prc.on('close', function (code) {
            const index = rooms.findIndex(room => room.childProcess === prc);

            rooms.splice(index, 1);
        });

        rooms.push(new Room(maxUsersCount, prc, roomPort));
    }
}

function findFreeRoom() {
    for (i = 0; i < maxRoomsCount; i++) {
        const room = rooms[i];

        if (room.isFree()) {
            return room;
        }
    }

    return null;
}

app.listen(serverPort);
```