Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего профессионального 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

Клиент-серверная игра «Змейка». Клиентская часть

Руководитель	_____	_____	ст. преподаватель	В.С. Васильев
	подпись	*дата*	*должность, ученая степень*	
Выпускник	_____	_____		А.Р. Шарафеева
	подпись	*дата*		
Нормоконтролер	_____	_____		В.С. Васильев
	подпись	*дата*		

Красноярск 2023

РЕФЕРАТ

Выпускная квалификационная работа по теме «Клиент-серверная игра «Змейка». Клиентская часть» содержит 50 страниц текстового документа, 1 таблицу, 47 рисунков, 21 использованных источников.

ИГРА, UNITY, MICROSOFT WINDOWS, ЗМЕЙКА, АРКАДА, КЛИЕНТ-СЕРВЕР

Объект выпускной работы: процесс создания программного продукта.

Предмет выпускной квалификационной работы: разработка приложения для устройств на ОС Microsoft Windows.

Целью данной выпускной квалификационной работы является разработка клиентского приложения для клиент-серверной игры «Dragon.io» в жанре аркада для устройств на ОС Microsoft Windows.

Выпускная квалификационная работа выполнена в соответствии с индивидуальным заданием. В процессе выполнения решены следующие задачи:

1. Выполнен анализ предметной области проекта.
2. Сформировано техническое задание на проект.
3. Проанализирован выбор программных средств разработки клиентской части приложения.
4. Создано клиентское приложение.
5. Выполнено тестирование приложения.
6. Составлена инструкция разработчика.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По данным аналитических ресурсов pr-cy.ru [1] сайт Snake.io [2] реализующий игру «Змейка» просматривает 1.3 миллиона пользователей в день. Разработчик получает оплату за каждый просмотр страницы при размещении на ней рекламы.

Целью работы является создание клиентской части игры Змейка, подобной популярным реализациям игр Snake.io [2] и Wormax.io [3]. Структура работы отражает решаемые **задачи**.

В рамках **первой главы** проведен сравнительный анализ аналогов, отмечены их недостатки и предлагаются варианты улучшения игры. С их учетом, разработана спецификация требований для клиентской части приложения.

Во второй главе приведены результаты проектирования клиентской части системы, включая ER-диаграммы базы данных, диаграммы последовательности и диаграммы классов. Спецификация требований оформлена с использования стандарта OpenAPI [4].

В третьей главе работы описаны особенности реализации и тестирования приложения. На основе спецификации OpenAPI, с использованием существующих инструментальных средств, сгенерирована часть кода системы. Приведены инструкции по сборке системы, а также ее тестирования.

В процессе выполнения работы решены следующие **задачи**:

– разработано техническое задание для проекта;

– на основе технического задания выполнено проектирование системы;

– в соответствии с проектом выполнена реализация программы, а также ее тестирование.

1 Анализ предметной области

1.1 Анализ существующих аналогов

Для обеспечения конкурентоспособности приложения, выполнен анализ сильных и слабых сторон аналогичных приложений. С их учетом составлена спецификация требований приложения. Результаты сравнительного анализа приведены в таблице 1.

Таблица 1 — Сравнение аналогов

Критерий сравнения	**snake.io[2]**	**wormax.io[3]**	**slither.io[5]**
Количество просмотров страницы, млн. человек/день	1,3	2,2	19,6
Монетизация	Показ и скрытие сторонних объявлений	Показ и скрытие сторонних объявлений	Показ и скрытие сторонних объявлений
Создание учетной записи	+	+	-
Общение с другими игроками	-	-	-
Возможность поделиться результатом в соц. сети	+	+	+
Возможность играть в однопользовательском режиме	-	-	+
Использование платных дополнений	+	+	-
Просмотр общей статистики игроков	+	+	-
Просмотр статистика игроков в текущей игровой сессии	+	+	+
Выполнение ежедневных заданий	-	+	-
Повышение ранга	-	+	-
Просмотр мини карты	-	+	+
Выбор управления	+	-	-
Использование особых предметов	-	+	-
Использование особые возможностей персонажа	Ускорение	Ускорение, остановка, неуязвимость	Ускорение

Окончание таблицы 1

Критерий сравнения	snake.io[2]	wormax.io[3]	slither.io[5]
Выбор игровой модели персонажа	+	+	+
Создание игровой модели персонажа	-	-	+
Возможность узнать направление лучшего игрока в текущей игровой сессии	+	-	-

Исходя из анализа сильных и слабых сторон аналогов определены возможности, которые может иметь приложение.

1.2 Спецификация требований для клиентского приложения

Пользователь программы – человек, который играет в многопользовательскую игру, управляя драконом на игровом поле. Его основная цель – заработать как можно больше очков, съедая еду и увеличивая размер своего дракона. Игрок сражается с соперниками (другими пользователями) в режиме реального времени. Во время игры он может следить за своим результатом, общим рейтингом и текущими лидерами, для того чтобы знать какие игроки занимают первые места и постараться опередить их. После завершения игры (проигрыша) пользователь видит достигнутый результат и может начать игру заново.

1.2.1 Функциональные требования

На рисунке 1.1 приведена диаграмма прецедентов создаваемого приложения.

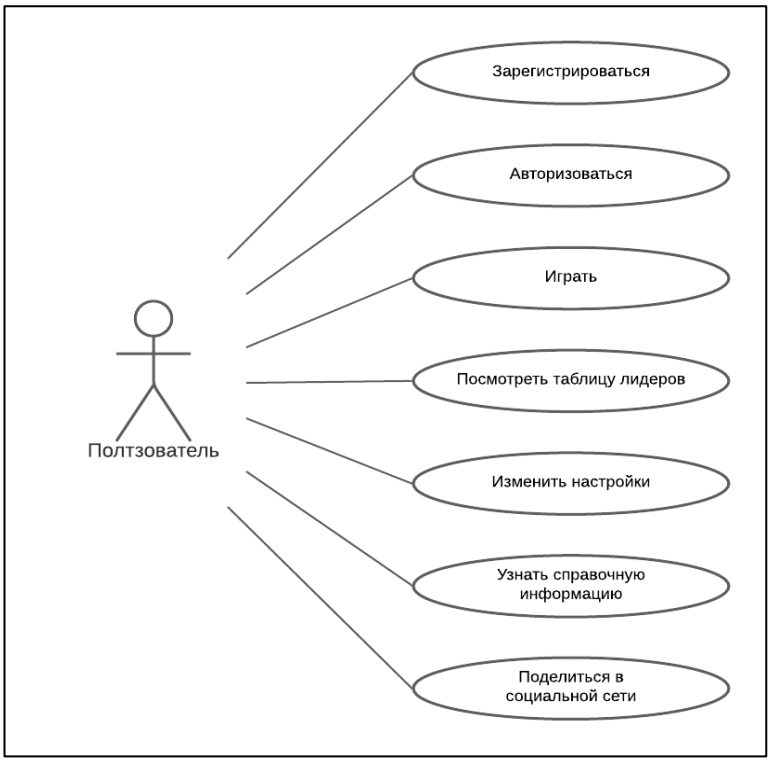

Рисунок 1.1 — Диаграмма прецедентов

Ниже приведено текстовое описание прецедентов и макеты пользовательского интерфейса (Рисунок 1.2).

Рисунок 1.2 — Макет пользовательского интерфейса для основного меню

Прецедент 1. Зарегистрироваться.

Цель: пользователь создает новую учетную запись в системе.

Предусловия: пользователь находится в основном меню.

Основной сценарий:

1) пользователь нажимает кнопку «Регистрация» (Рисунок 1.3);

2) пользователь вводит данные: адрес электронной почты, пароль, имя персонажа;

3) система проверяет уникальность введенного адреса электронной почты;

4) пользователь подтверждает свое согласие с политикой конфиденциальности и правилами использования системы;

5) пользователь нажимает кнопку «Зарегистрироваться»;

6) отправляется запрос с данными на сервер;

7) система создает новую учетную запись для пользователя;

8) пользователь получает сообщение о том, что регистрация прошла успешно;

9) пользователь перенаправляется на главную страницу.

Постусловия: пользователь успешно создал новую учетную запись в системе и может использовать ее для авторизации в системе.

Условия ввода в действие альтернативных сценариев

Условие 1. Введенный адрес электронной почты уже используется другим пользователем.

1) выводится сообщение с просьбой ввести другой адрес электронной почты или авторизоваться, если у него уже есть учетная запись;

2) пользователь перенаправляется в окно с вводом данных.

Условие 2. Сервер недоступен.

1) выводится сообщение об ошибке.

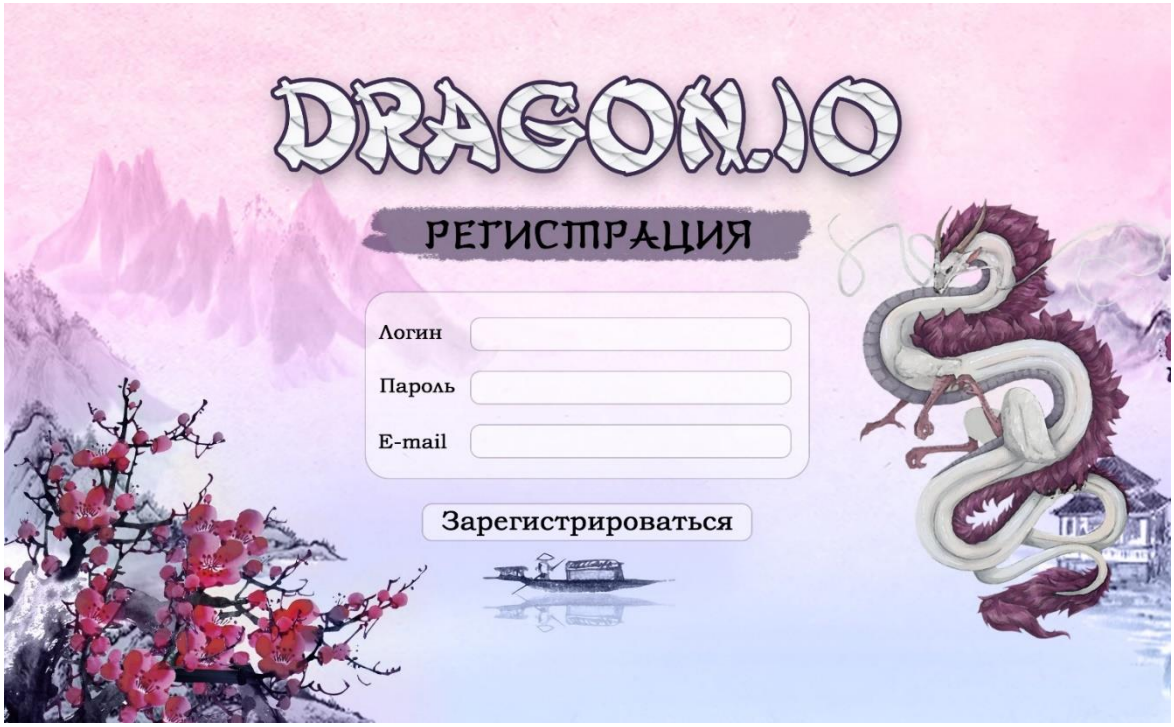

Рисунок 1.3 — Окно регистрации

Прецедент 2. Авторизоваться.

Цель: пользователь вводит свои учетные данные, чтобы получить доступ к своему аккаунту в игре.

Предусловия: пользователь находится в основном меню.

Основной сценарий:

1) пользователь нажимает кнопку «Вход»;

2) система отображает на экране форму авторизации, которая включает в себя поля для ввода адреса электронной почты и пароля (Рисунок 1.4);

3) пользователь вводит свои данные: адрес электронной почты и пароль;

4) пользователь нажимает кнопку «Войти»;

5) система проверяет, что введенные учетные данные являются действительными и находятся в базе данных;

6) система открывает доступ к аккаунту пользователя и переводит его на экран основного меню.

Постусловия: пользователь авторизовался и получил доступ к своему аккаунту в игре.

Условия ввода в действие альтернативных сценариев

Условие 1. Данные введены неправильно.

1) система выводит сообщение об ошибке и предлагает пользователю повторить попытку;

2) пользователь перенаправляется в окно с вводом данных.

Условие 2. У пользователя нет учетной записи.

1) пользователь может нажать кнопку «Регистрация» и создать новую учетную запись.

Условие 3. Пользователь забыл пароль.

1) пользователь может нажать на кнопку «Забыли пароль?» на экране авторизации, чтобы сбросить пароль и получить новый.

Условие 4. Сервер недоступен.

1) выводится сообщение об ошибке.

Рисунок 1.4 — Окно авторизации

Прецедент 3. Играть.

Цель: Игрок начинает игру, управляет драконом и соревнуется с другими игроками.

Предусловия: Игрок находится в основном меню.

Основной сценарий:

1) игрок нажимает кнопку «Играть»;

2) система загружает игровую карту и выводит ее на экран (Рисунок 1.5);

3) система создает дракона и размещает его на карте;

4) игрок управляет драконом и собирает еду, чтобы увеличивать длину дракона;

5) когда игрок нажимает определенную кнопку, его скорость увеличивается на n секунд;

6) когда дракон сталкивается с препятствием, либо другим игроком то игра заканчивается.

Постусловия:

1) игрок завершил игру;

2) выводится окно с результатами игры;

3) результат игрока сохраняется в таблицу лидеров, если он вошел в n – лучших результатов;

4) игрок перенаправляется на страницу основного меню.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер недоступен.

1) выводится сообщение об ошибке.

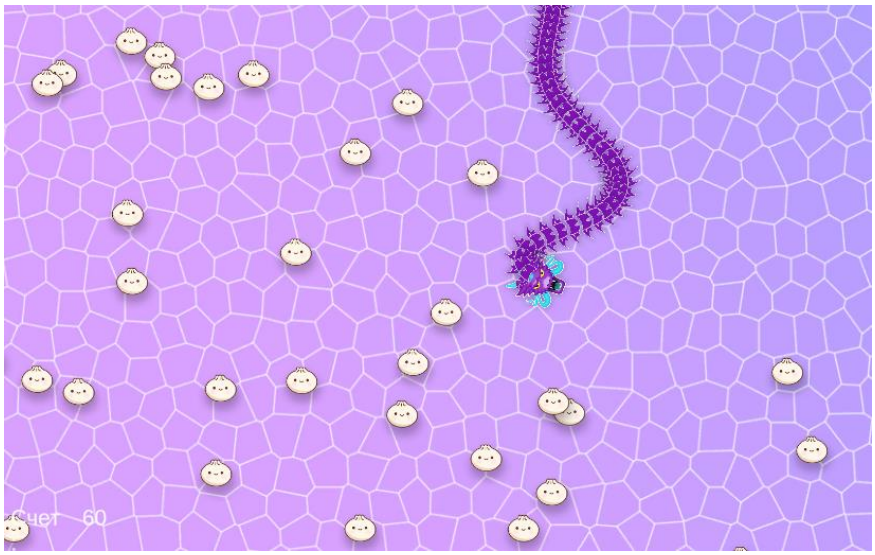

Рисунок 1.5 — Окно игрового поля

Прецедент 4. Посмотреть таблицу лидеров.

Цель: пользователь просматривает список лучших игроков в игре.

Предусловия: пользователь находится в основном меню.

Основной сценарий:

1) пользователь нажимает на кнопку «Лучшие игроки»;

2) система отображает на экране страницу с таблицей лидеров, в которой отображаются лучшие игроки в порядке убывания их рейтинга (Рисунок 1.6);

3) таблица лидеров включает в себя информацию об имени игрока и его максимальном рейтинге;

4) пользователь может выбрать определенный период времени, чтобы просмотреть результаты игроков за этот период;

5) пользователь может вернуться на главный экран, нажав на кнопку «Назад».

Постусловия: пользователь просмотрел таблицу лидеров по результатам игры.

Условия ввода в действие альтернативных сценариев

Условие 1. Таблица лидеров пуста.

1) система выводит сообщение о том, что в таблице лидеров пока что нет результатов и предлагает начать игру, чтобы создать результаты для заполнения таблицы.

Рисунок 1.6 — Окно с таблицей лидеров

Прецедент 5. Изменить настройки.

Цель: пользователь меняет настройки своего профиля в игре, в том числе имя персонажа, модель персонажа и способ управления.

Предусловия: пользователь авторизовался и находится в основном меню.

Основной сценарий:

1) пользователь нажимает на кнопку «Настройки»;

2) система открывает экран настроек профиля (Рисунок 1.7);

3) пользователь выбирает опцию «Изменить имя», вводит новое имя для своего профиля и нажимает кнопку «Подтвердить» (Рисунок 1.8);

4) пользователь выбирает опцию «Персонаж» и выбирает одну из доступных моделей персонажа (Рисунок 1.10);

5) пользователь выбирает опцию «Управление» и выбирает один из доступных способов управления (мышь или клавиатура) (Рисунок 1.9);

6) система сохраняет изменения;

7) пользователь может вернуться на главный экран, нажав на кнопку «Назад».

Постусловия: пользователь успешно изменил настройки своего профиля в игре.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер недоступен.

1) выводится сообщение об ошибке.

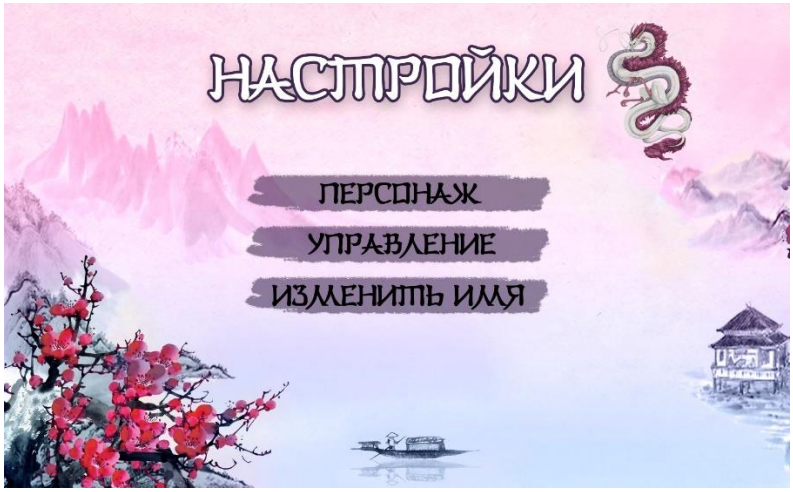

Рисунок 1.7 — Окно настроек

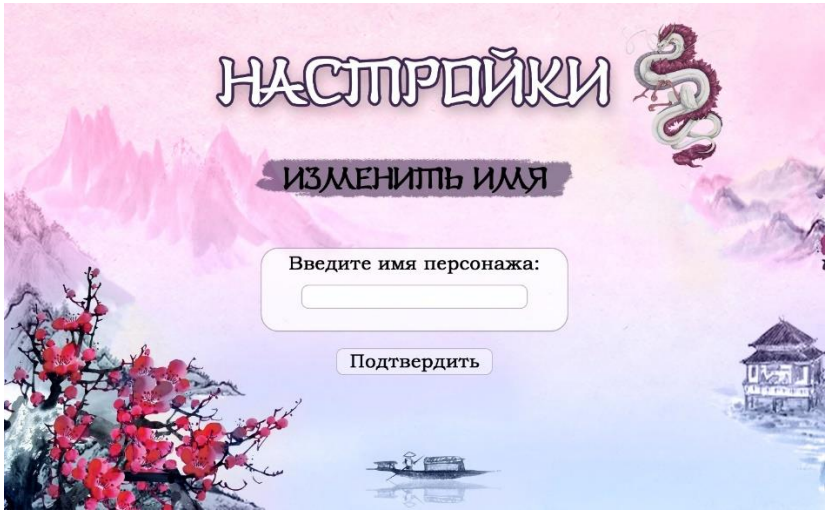

Рисунок 1.8 — Окно изменения имени персонажа

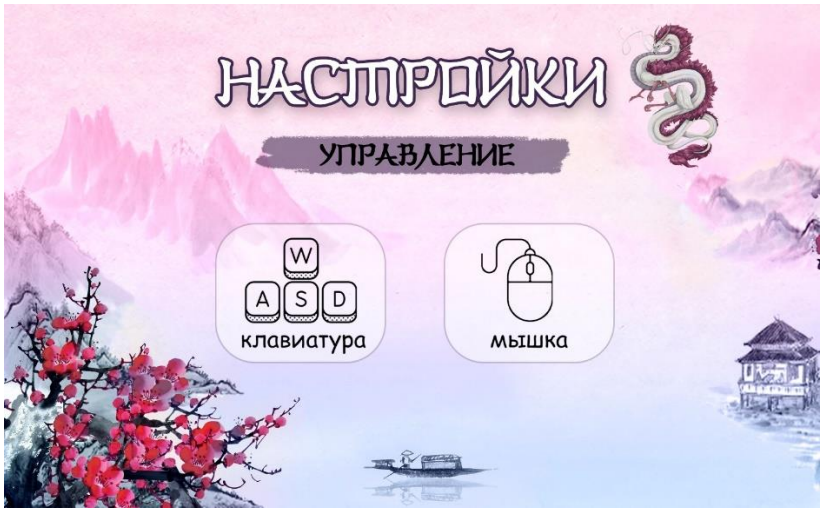

Рисунок 1.9 — Окно выбора управления

Рисунок 1.10 — Окно выбора модели персонажа

Прецедент 6. Узнать справочную информацию.

Цель: пользователь просматривает справочную информацию о правилах игры, о возможностях персонажа, способах управления.

Предусловия: пользователь находится в основном меню.

Основной сценарий:

1) пользователь нажимает на кнопку «Справка»;

2) система открывает экран справочной информации (Рисунок 1.11);

3) система отображает информацию об игре на экране;

4) пользователь просматривает информацию и, если нужно, прокручивает страницу вниз или вверх;

5) пользователь может вернуться на главный экран, нажав на кнопку «Назад».

Постусловия: пользователь получил необходимую справочную информацию о правилах игры, о возможностях персонажа и способах управления.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер недоступен.

1) выводится сообщение об ошибке.

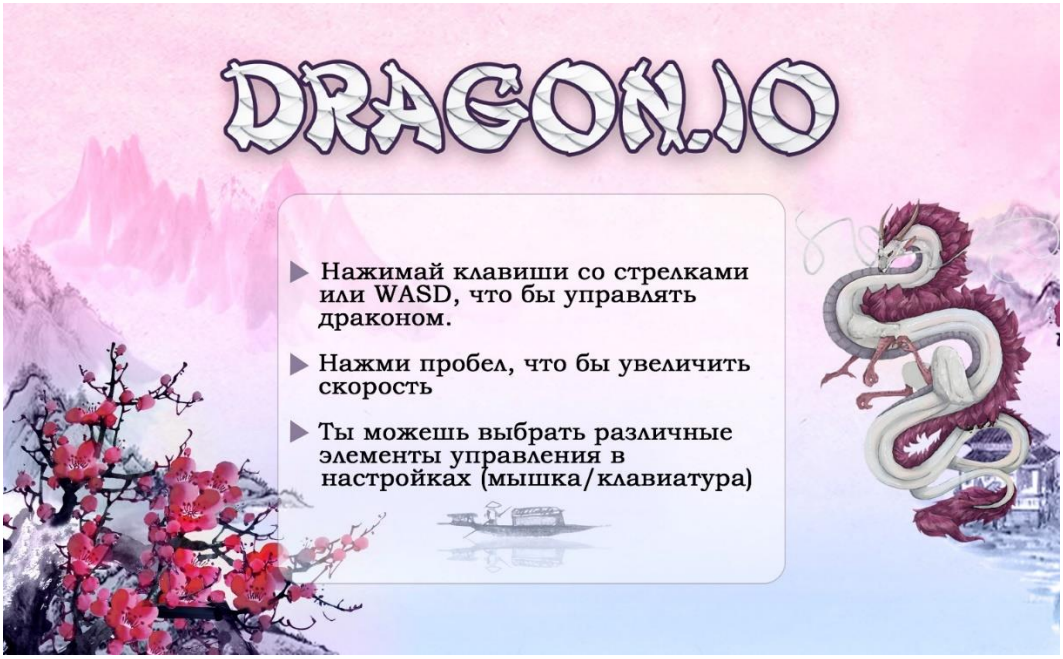

Рисунок 1.11 — Окно справочной информации

Прецедент 7. Поделиться в социальной сети.

Цель: пользователь публикует информацию с ссылкой на игру в своих социальных сетях.

Предусловия:

– пользователь находится в основном меню. (пользователь находится на экране с результатами игры);

– у пользователя есть учетная запись в социальной сети, куда он хочет опубликовать информацию.

Основной сценарий:

1) пользователь нажимает на кнопку «поделиться в социальной сети»;

2) система открывает экран с выбором социальной сети, куда пользователь хочет опубликовать информацию;

3) пользователь выбирает нужную социальную сеть из списка;

4) система открывает страницу в выбранной социальной сети для создания новой публикации;

5) пользователь видит информацию и может добавить к ней комментарий;

6) пользователь нажимает кнопку «Опубликовать»;

7) система публикует информацию о результате игры в выбранной социальной сети;

8) пользователь перенаправляется на страницу основного меню.

Постусловия: информация об игре была опубликована в выбранной социальной сети.

Условия ввода в действие альтернативных сценариев

Условие 1. Пользователь не авторизован в выбранной социальной сети.

1) система социальной сети выводить сообщение о том, что необходимо авторизоваться или зарегистрироваться в данной социальной сети.

1.2.1 Нефункциональные требования

Клиентское приложение должно соответствовать следующим требованиям:

– совместимость: приложение должно работать на разных устройствах и операционных системах, чтобы пользователи могли играть на любом устройстве;

– удобство использования: интерфейс приложения должен быть интуитивно понятным и удобным для использования, чтобы игроки могли быстро начать играть.

1.3 Диаграмма потока экранов

Основываясь на прецедентах, был разработан список необходимых страниц для приложения. На рисунке 1.12 представлена диаграмма потока экранов, на которой показана взаимосвязь страниц и переходов между ними.

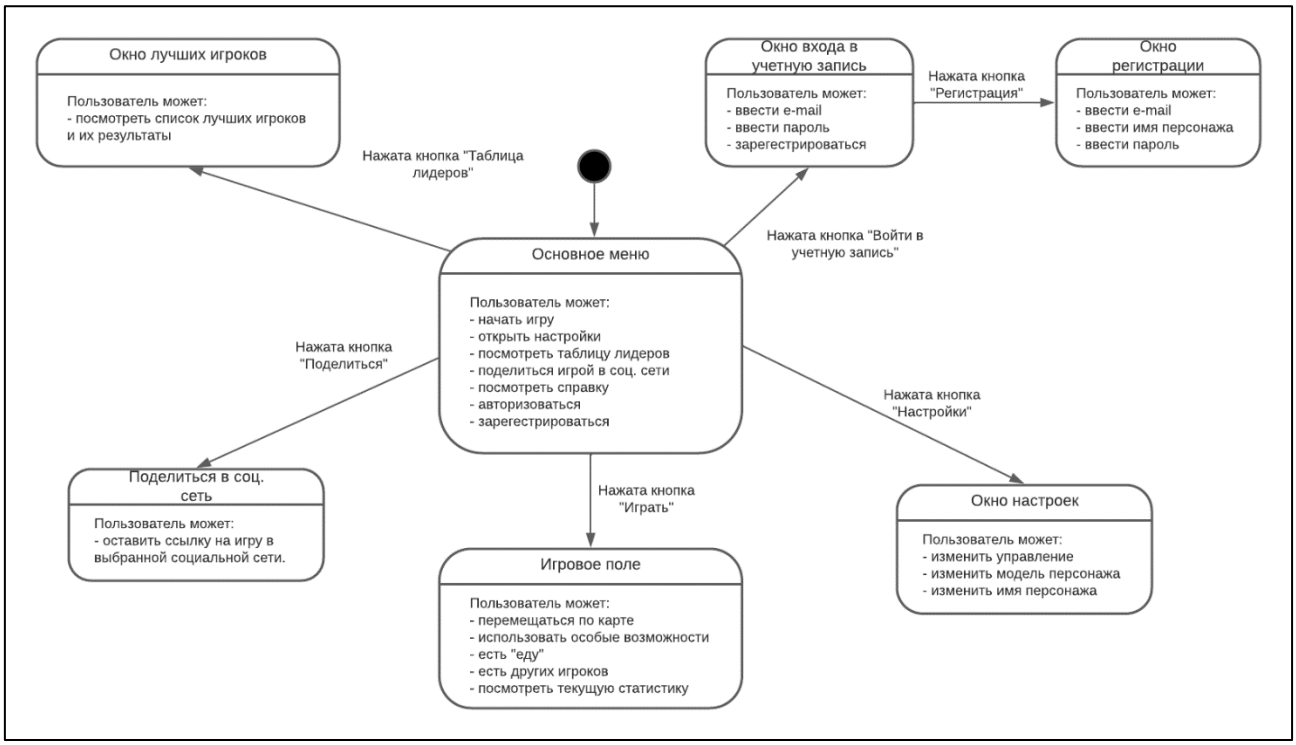

Рисунок 1.12 — Диаграмма потока экранов

1.4 Выводы по главе

1. Проведен анализ сравнения популярных аналогов с целью поиска успешных решений для возможности заимствования.
2. Сформулированы функциональные требования к приложению в виде диаграммы прецедентов и их текстового описания.
3. Сформулированы не функциональные требования к приложению.

2 Проектирование

Архитектура разрабатываемого приложения составлена на основе спецификации требований (Рисунок 2.1). Система состоит из клиентского и серверного приложения.

Клиентское приложение взаимодействует серверным приложением по протоколу TCP для получения данных о пользователе. Серверное приложение возвращает клиентскому приложению эти данные. Для хранения данных на сервере используется база данных, которая содержит информацию об имени пользователя, пароле, адресе электронной почты и лучшем игровом результате. Во время игры клиентское приложение взаимодействует с серверным приложением, используя Unity Transport, работающий по протоколу UDP. Сервер хранит информацию о подключенных к нему клиентах, обрабатывает исходящую от них информацию и рассылает эту информацию другим клиентам.

Клиентское приложение содержит экран меню, игровой экран, сетевые объекты и пользовательские настройки.

Основное меню содержит в себе подменю, такие как форма авторизации, таблица лидеров, справочная информация и меню настроек. Пользовательские настройки, такие как способ управления и модель персонажа, содержатся в файле с разрешением ini. При нажатии кнопки играть происходит переключение на игровой экран. Он взаимодействует с сетевыми объектами. Это игровые объекты, требующие сетевой синхронизации, такие как еда и персонажи.

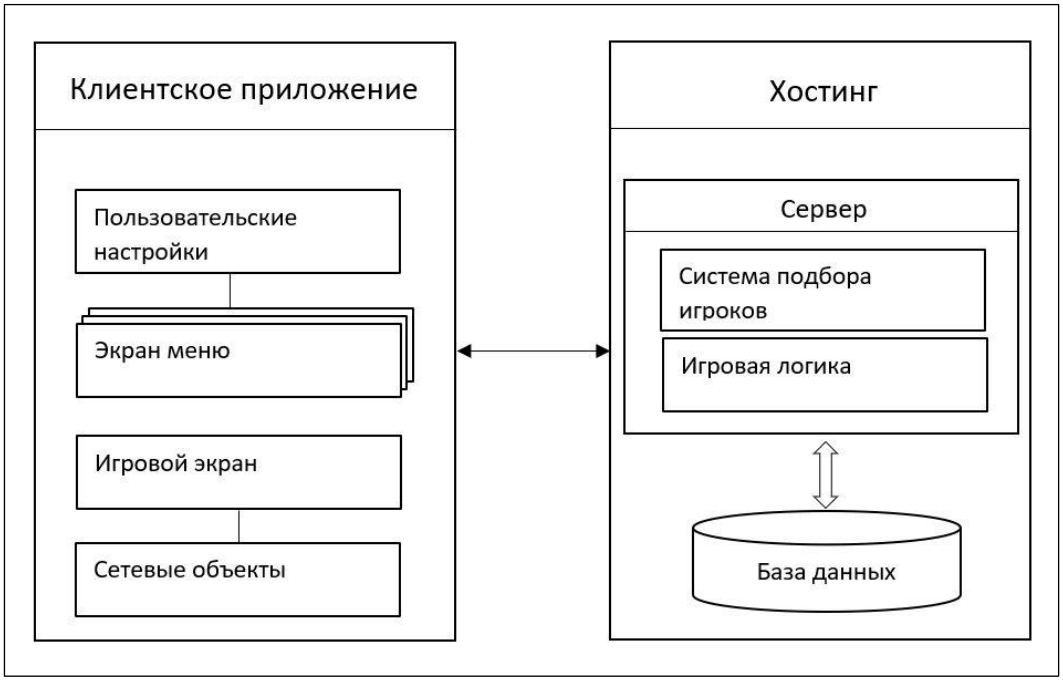

Рисунок 2.1 — Архитектура приложения

Для того, чтобы результат разработки соответствовал составленной спецификации требований был выбран процесс ICONIX [6], основанный на прецедентах. В приложении есть операции, которые применяются к нескольким сущностям, такие как: присоединение к игре, управление персонажем, обработка игровых событий, визуализация и обновление игрового состояния. Поэтому для основных прецедентов были разработаны диаграммы пригодности и последовательности. На основе диаграмм последовательности были разработаны диаграммы классов, которые могут быть использованы при написании кода.

2.1 Диаграммы пригодности

2.1.1 Прецедент «Играть»

Пользователь запускает приложение и нажимает кнопку «Играть». Программа отправляет запрос на подключение к серверу и подключает пользователя к выделенному серверу. В случае корректного соединения происходит загрузка игрового мира и создание персонажа в текущей игровой сессии. Пользователь начинает управлять своим персонажем, взаимодействовать с игровыми

объектами и другими пользователями. В случае некорректного соединения, выводится сообщение об ошибке сети (Рисунок 2.2).

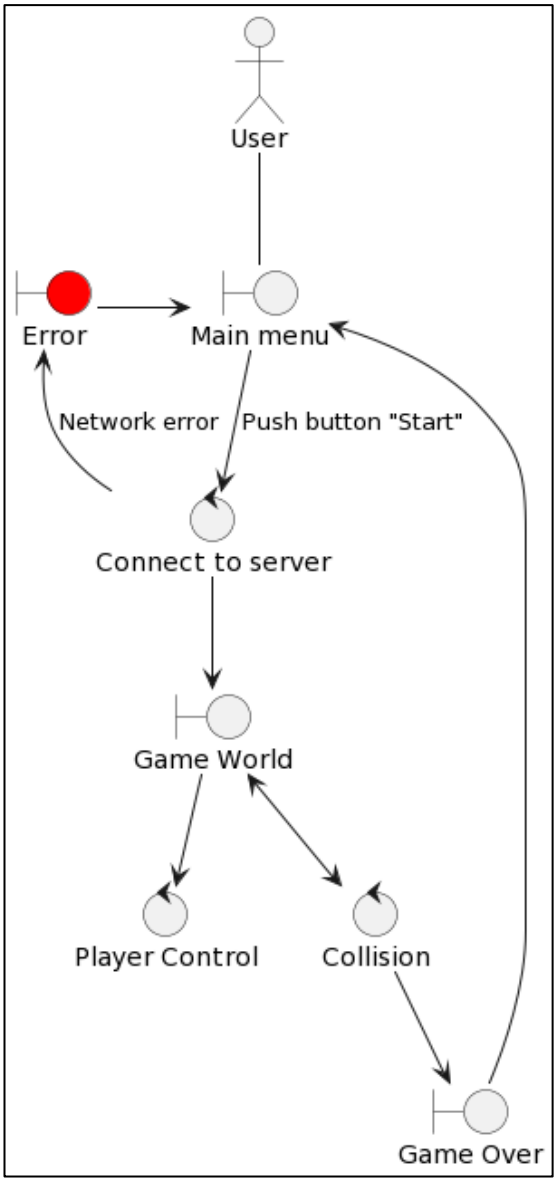

Рисунок 2.2 — Диаграмма пригодности прецедента «Играть»

2.1.2 Прецедент «Авторизоваться»

Пользователь запускает приложение и нажимает кнопку «Войти». В открывшемся окне вводит данные: адрес электронной почты, пароль и имя пользователя. На сервер отправляется запрос с данными, проводится их проверка и возвращается ответ и на клиенте появляется сообщение о том, что авторизация прошла успешно, либо об ошибке. Если у пользователя нет учетной записи он

регистрируется в системе – вводит данные, после чего на сервер отправляется запрос о регистрации. Сервер возвращает ответ и на клиенте появляется сообщение об успешной регистрации или об ошибке (Рисунок 2.3).

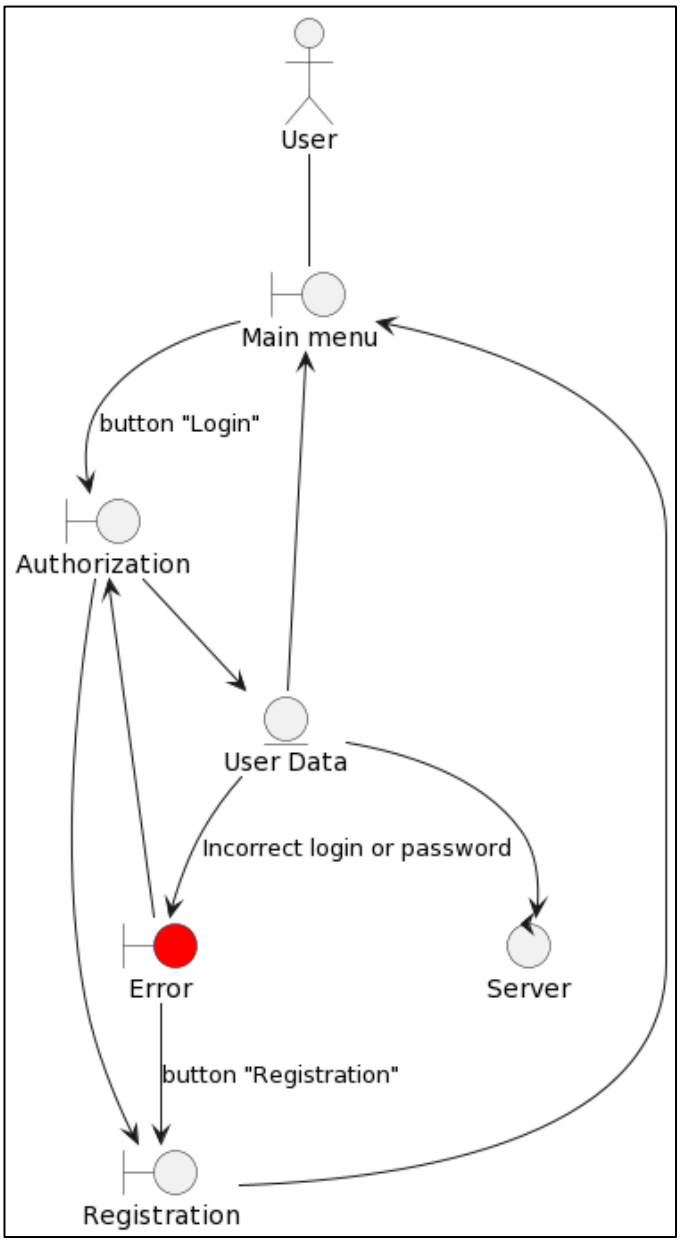

Рисунок 2.3 — Диаграмма пригодности прецедента «Авторизоваться»

2.1.3 Изменить настройки

Пользователь переходит в меню настроек, в котором может изменить способ управления, имя персонажа или выбрать модель для персонажа (Рисунок 2.4).

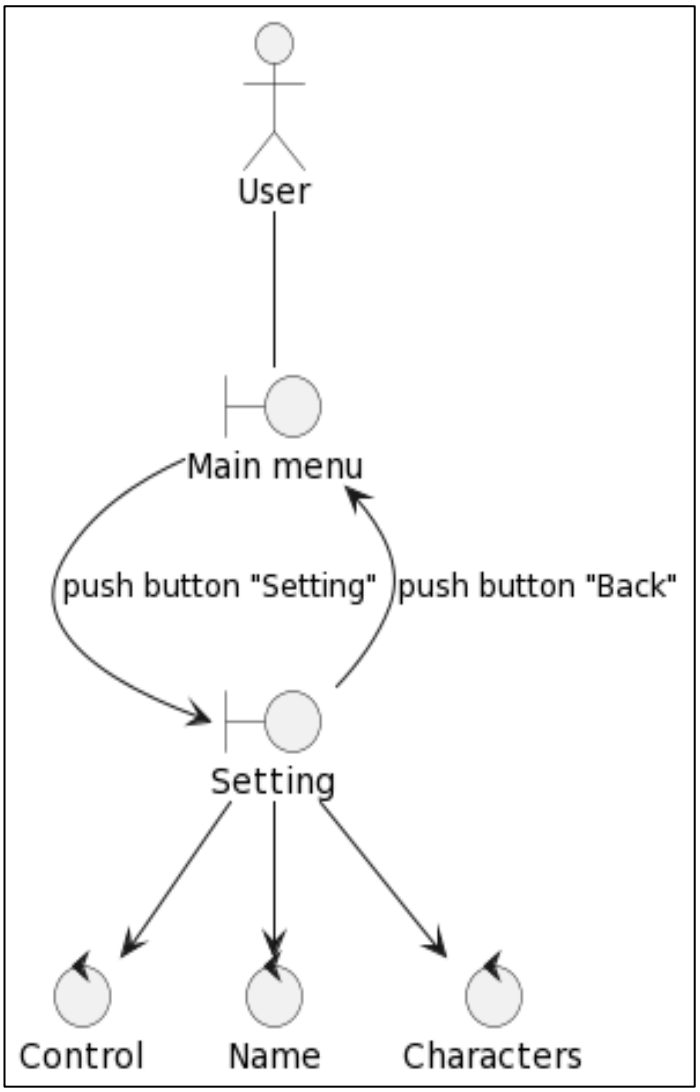

Рисунок 2.4 — Диаграмма пригодности прецедента «Изменить настройки»

2.2 Диаграммы последовательности

Значительная часть логики системы связана с меню и игровым полем. В данном разделе приведены диаграммы последовательностей для основных прецедентов, связанных со взаимодействием компонентов системы.

На рисунке 2.5 изображена диаграмма последовательности для варианта использования «Играть», представляющая процесс взаимодействия клиентской программы с сервером, для получения позиции еды, позиции персонажа и его длины, а также получения своего итогового результата.

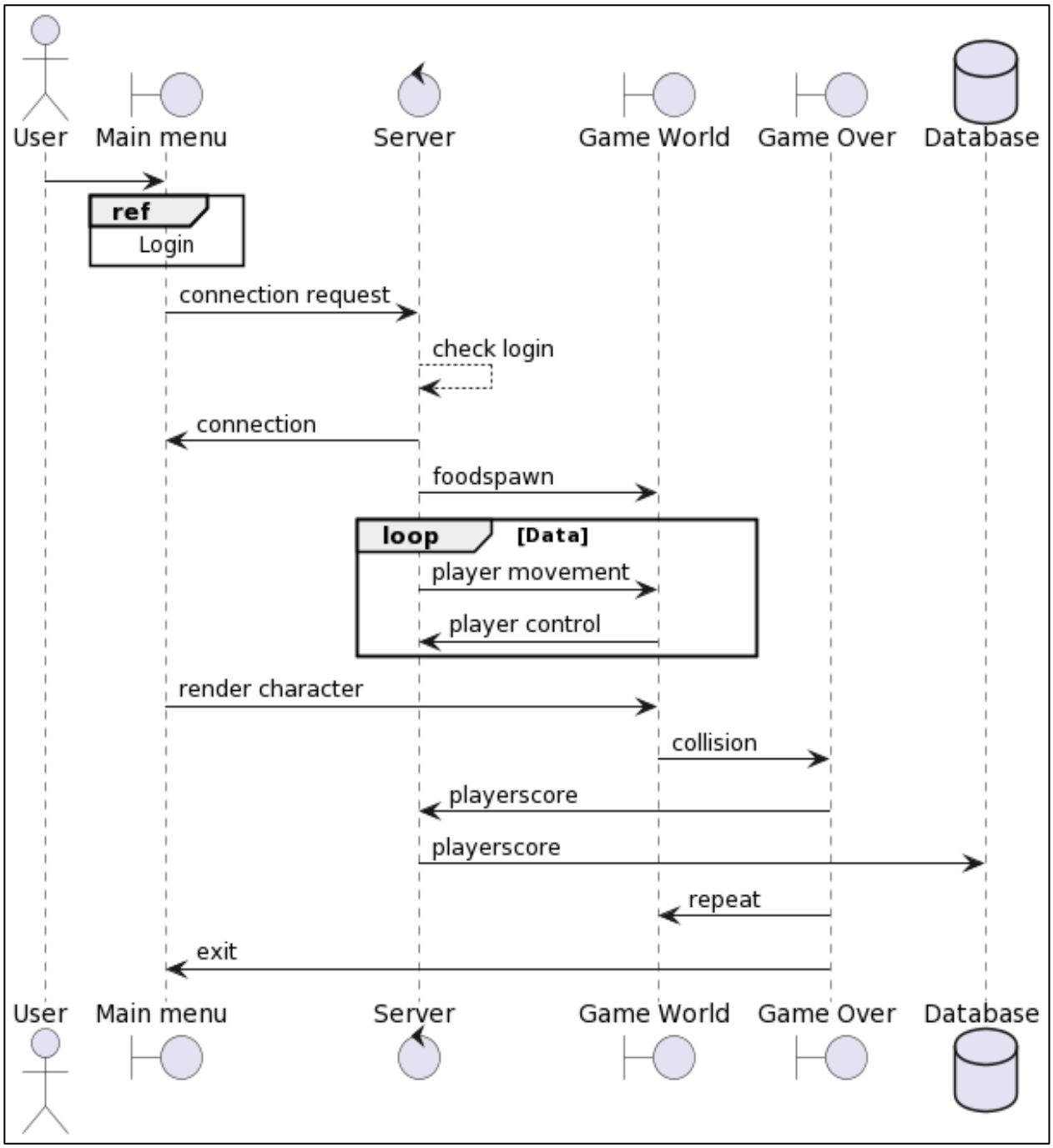

Рисунок 2.5 — Диаграмма последовательности «Играть»

На рисунке 2.6 изображена диаграмма последовательности для варианта использования «Авторизация», представляющая процесс авторизации и регистрации пользователя, передачи данных на сервер и их проверки.

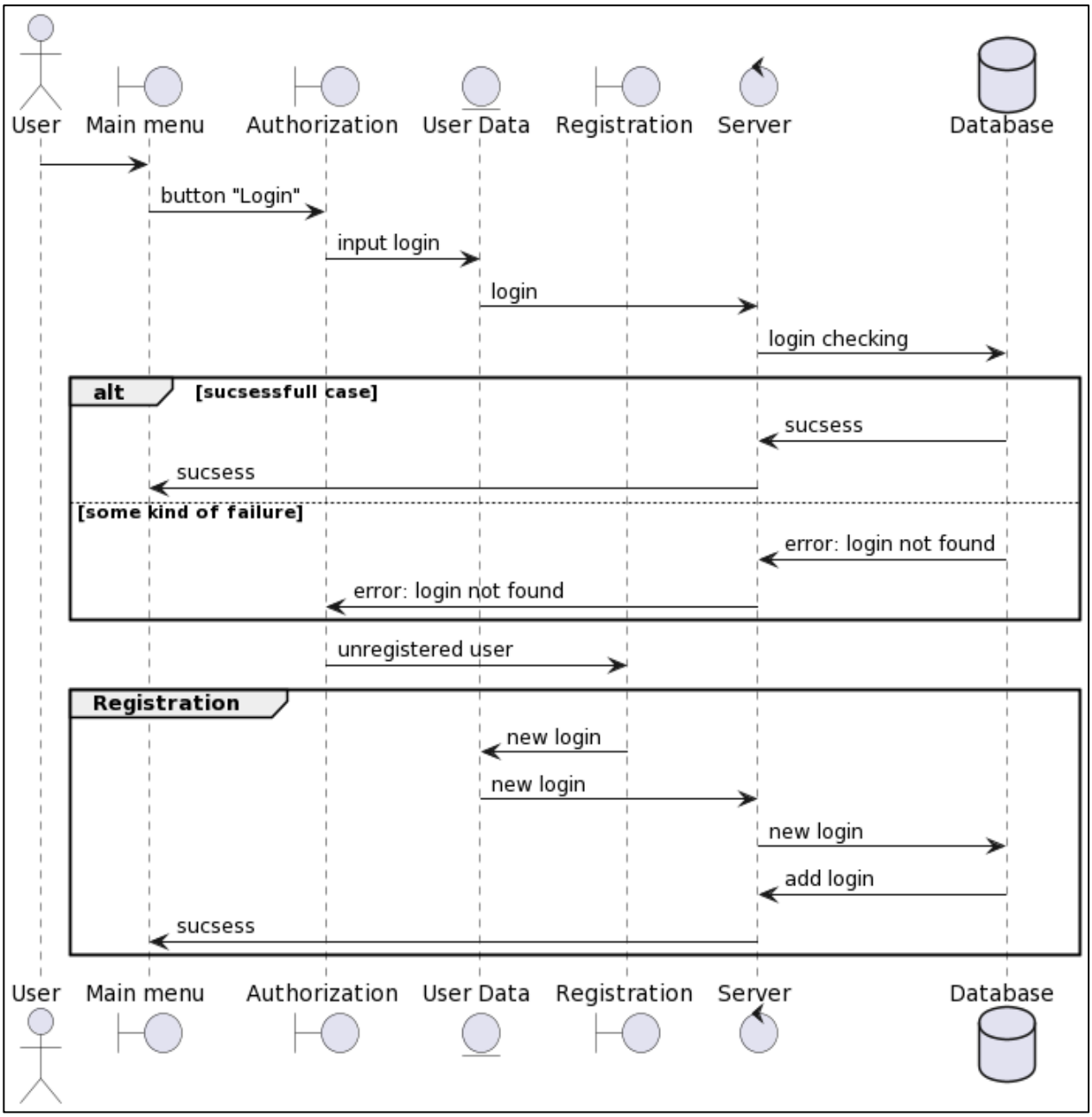

Рисунок 2.6 — Диаграмма последовательности «Авторизация»

На рисунке 2.7 изображена диаграмма последовательности для варианта использования «Изменить настройки», представляющая процесс изменения пользовательских настроек, в которые входят возможности: смены управления, изменения имени и выбор модели персонажа.

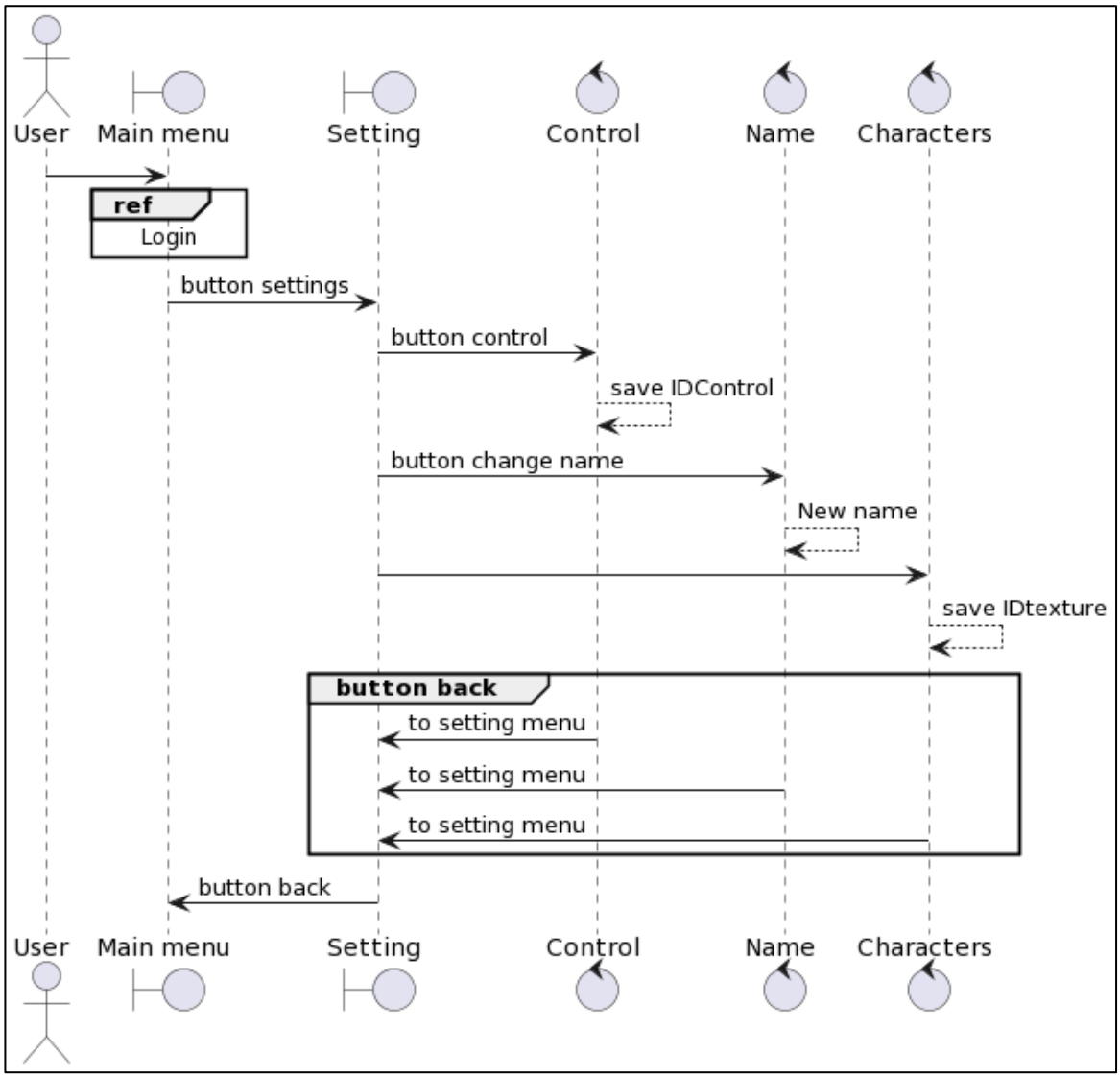

Рисунок 2.7 — Диаграмма последовательности «Изменить настройки»

2.3 Диаграммы классов

Все игроки имеют одинаковый набор параметров таких, как: положение головы, положение хвоста, длина хвоста, скорость передвижения по игровому полю, текущее количество заработанных очков. Поэтому целесообразно создать класс Player, от которого будут наследоваться игроки. В классе происходит валидация данных. Таким образом, если кто-то попытается из вне подменить значение скорости или длину хвоста, то настройки методов, обращающихся с сервером, не позволят изменить данные.

Для того, чтобы игрок мог передвигаться по игровому полю класс player_control передает серверу положение мышки, после чего сервер

передвигает голову персонажа в сторону направления мышки. А класс camera_controller выполняет слежение за игровым персонажем.

Для того, чтобы игрок мог увеличивать длину своего персонажа сервер обрабатывает событие столкновения с едой методом collisions и передает значение длины классу player в переменную length, после чего вызывается класс player_lenght, который добавляет новое звено хвоста.

Класс tail выполняет передвижение звеньев хвоста по направлению, заданному головой персонажа.

Класс player_stats хранит и отображает текущую статистику игрока.

Диаграмма классов архитектуры игрока представлена на рисунке 2.8.

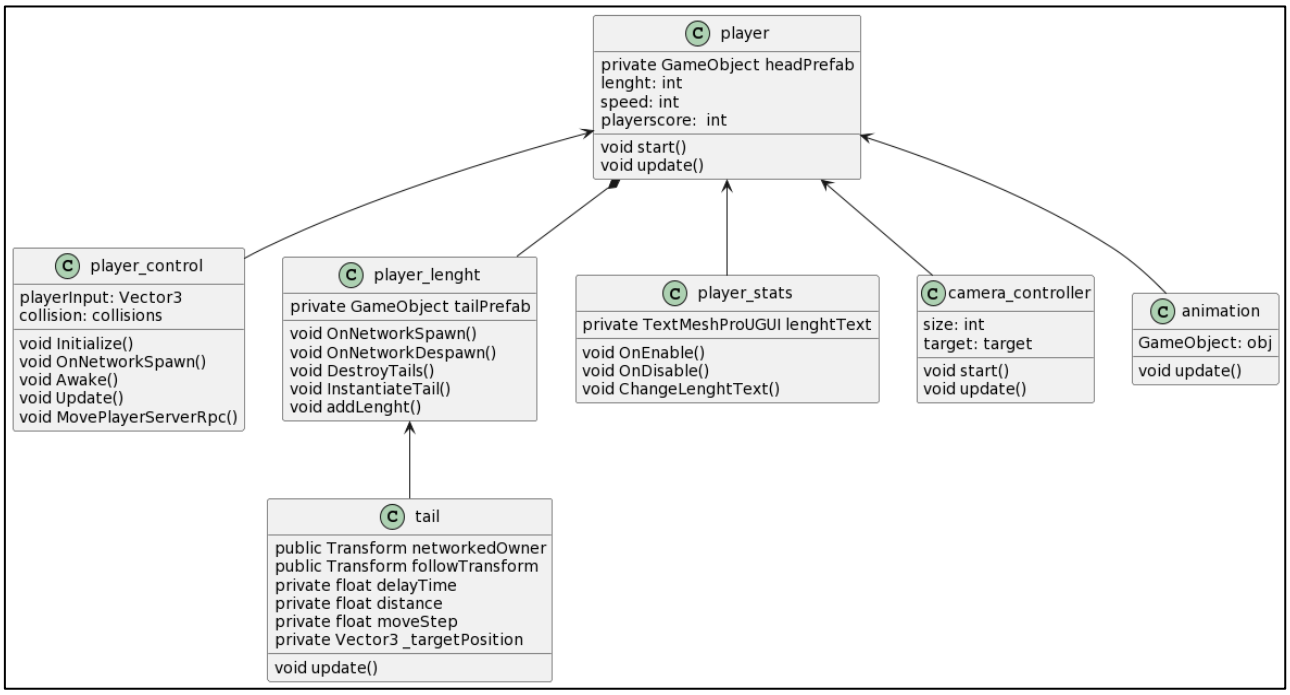

Рисунок 2.8 — Диаграмма классов архитектуры игрока

Для того, чтобы на игровом поле появились объекты – игроки и еда, выполняется связь с сервером посредством класса networkManager, который принимает значение положения и количества еды, а также позицию появления игрока.

Диаграмма классов появления объектов на игровой сцене представлена на рисунке 2.9.

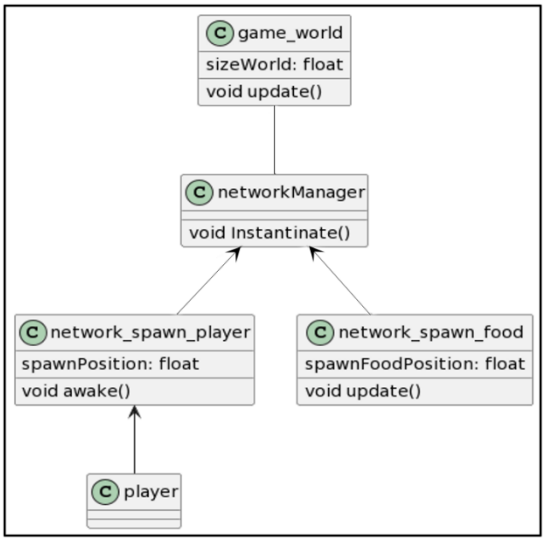

Рисунок 2.9 — Диаграмма классов для процесса появления объектов на игровой сцене

Класс authorization_controller производит передачу введенных данных на игровой сервер, где происходит проверка. Если данные введены, верно, пользователь авторизуется и перенаправляется в главное меню. Иначе выводится сообщение о том, что произошла ошибка авторизации, пользователь остается в меню авторизации и может перейти в меню регистрации, если у него нет учетной записи.

Диаграмма классов процесса авторизации представлена на рисунке 2.10.

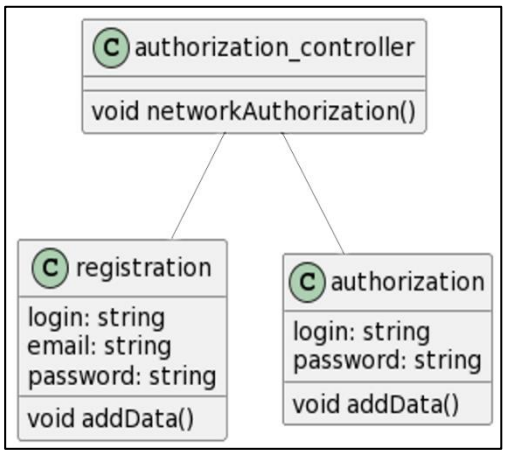

Рисунок 2.10 — Диаграмма классов для процесса авторизации

2.4 Диаграмма базы данных

На сервере используется база данных для хранения информации о пользователе (Рисунок 2.11).

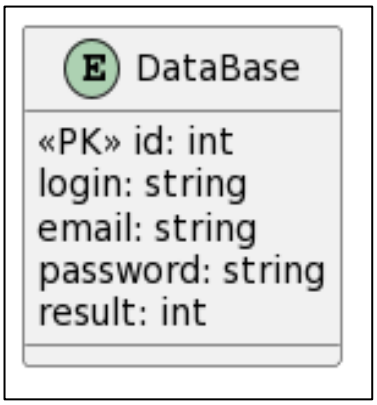

Рисунок 2.11 — ER-диаграмма базы данных

База данных содержит информацию об идентификаторе пользователя, его лучшем результате, и регистрационных данных, таких как логин, адрес электронной почты и пароль.

2.5 Выводы по главе

В соответствии со спецификацией требований:

1) составлены диаграммы пригодности для наиболее важных прецедентов. На их основе составлены диаграммы последовательности и разработана диаграмма классов;

2) предложена архитектура системы и структура базы данных.

3 Реализация и тестирование

3.1 Выбор инструментов

Для реализации клиент – серверного приложения выбран инструмент Unity [7], поскольку он обеспечивает поддерживает большое количество платформ, включая Windows, macOS, Linux, iOS, Android и другие, что позволяет создавать приложения, которые работают на разных устройствах. Unity имеет мощный движок, который позволяет разрабатывать игровую логику и реализовывать игровые механики. Так же, для реализации сетевой функциональности была выбрана библиотека NetCode for GameObject [8], которая позволяет легко создавать и настраивать сетевые объекты, обрабатывать взаимодействие между ними и синхронизировать состояние игры между клиентами и сервером, что обеспечивает высокую производительность в онлайн-режиме.

3.2 Игровой процесс

Игра Dragon.io представляет собой многопользовательскую онлайн-игру, в которой игроки управляют драконами, стремясь стать самыми крупными и доминировать на игровом поле.

Начало игры:

– игроки заходят в игру и подключаются к серверу Dragon.io;

– игровое поле представляет собой прямоугольную область, на которой будут развиваться события игры.

Управление драконом:

– каждый игрок контролирует своего дракона с помощью клавиатуры, мышки или сенсорного экрана;

– дракон может двигаться вверх, вниз, влево и вправо, его цель - собирать еду для увеличения своего размера.

Сбор еды:

– на игровом поле расположены объекты пищи, такие как пельмени;

– дракон должна съесть эти объекты, чтобы увеличить свой размер и набрать очки.

Рост дракона:

– после съедения пищи дракон растет в длину;

– управление длинной дракона становится сложнее, поскольку игроку необходимо управлять всем его телом.

Столкновения:

– драконы не должны сталкиваться с препятствиями: границами игрового поля и другими драконами;

– если дракон сталкивается с препятствием он погибает, и игрок выбывает из текущей игровой сессии.

Лидерство и рейтинг:

– игроки могут соревноваться за занятие высоких мест в рейтинге игры;

– очки и место в рейтинге определяются размером дракона и количеством съеденной пищи;

– текущий размер дракона отображается в нижней левой части экрана.

Завершение игры:

– игра завершается после гибели дракона, либо при нажатии кнопки «Выход»;

– на экран выводится достигнутый результат в текущей игре.

3.3 Дизайн приложения

3.3.1 Выбор игрового стиля

Игра реализована в стиле «Hyper Casual 2D» [9]. Это обосновано такими факторами, как:

1) привлекательность и популярность: Жанр «Hyper Casual 2D» очень популярный и привлекательный для широкой аудитории игроков. Этот стиль игр характеризуется простотой геймплея, яркими визуальными эффектами и удобством управления, что делает их доступными и привлекательными для игроков всех возрастных групп;

2) простота и интуитивность: Стиль «Hyper Casual 2D» отличается простыми правилами и механиками игры;

3) мобильная оптимизация: «Hyper Casual 2D» игры широко распространены на мобильных устройствах. Они имеют компактный размер и не требуют высоких системных требований, что делает их идеальным выбором для мобильной платформы;

4) гибкость и возможности развития: «Hyper Casual 2D» игры могут быть легко расширены и доработаны с добавлением новых уровней, функций и контента. Это дает возможность постепенного развития игры.

3.3.2 Отрисовка дизайна

Для отрисовки спрайтов, фона и элементов меню был использован графический редакторы Adobe Photoshop 2022 [10]. Все изображения были сохранены в растровом формате PNG, так как движок Unity не воспринимает векторные форматы. На рисунке 3.1 представлены спрайты игровых объектов.

Рисунок 3.1 — Спрайты игровых объектов

На рисунке 3.2 представлен фон для игрового поля.

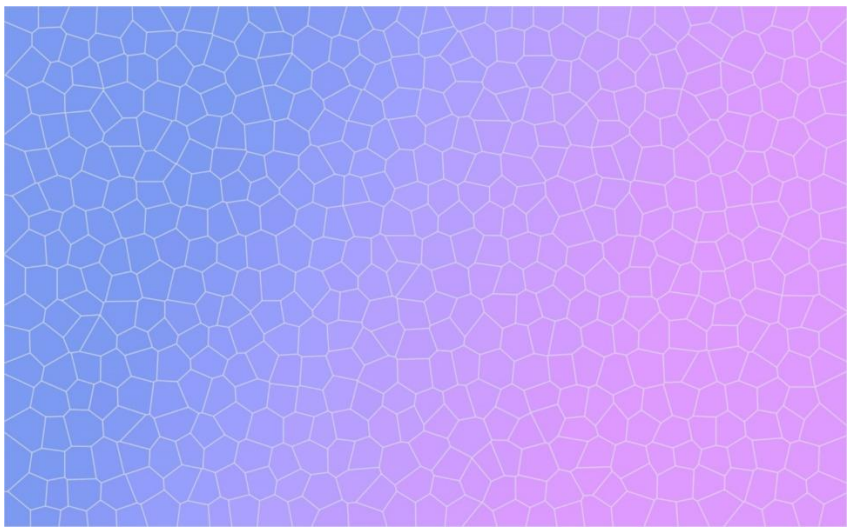

Рисунок 3.2 — Игровой фон

На рисунке 3.3 представлены спрайты для элементов интерфейса – основные кнопки, иконки и модели персонажей.

Рисунок 3.3 — Спрайты

3.4 Создание пользовательского интерфейса

Создается 2 сцены:

– оффлайн сцена – для отображения игрового меню;

– онлайн сцена – для отображения поля, на котором происходит игровая активность.

Для формирования оффлайн сцены создается Canvas (полотно), на котором размещены основные элементы меню (Рисунок 3.4).

Окно Меню содержит следующее:

– название игры;

– фоновый рисунок;

– кнопку «Играть», для подключения к серверу и перехода на онлайн сцену;

– кнопку «Настройки», для перехода в подменю с настройками;

– кнопку «Лучшие игроки», для просмотра таблицы лидеров;

– кнопку «Справка», для просмотра окна справочной информации;

– кнопку «Поделиться», для размещения ссылки в социальной сети «VK» (ВКонтакте);

– кнопку «Вход», для авторизации.

Настраиваются окна для настроек, таблицы лидеров, справки и переходы между ними. На рисунке 3.5 представлен пример настроек для перехода между окнами. Так же настраивается переход на онлайн сцену и подключение к серверу, посредством кнопки «Играть» (Рисунок 3.6).

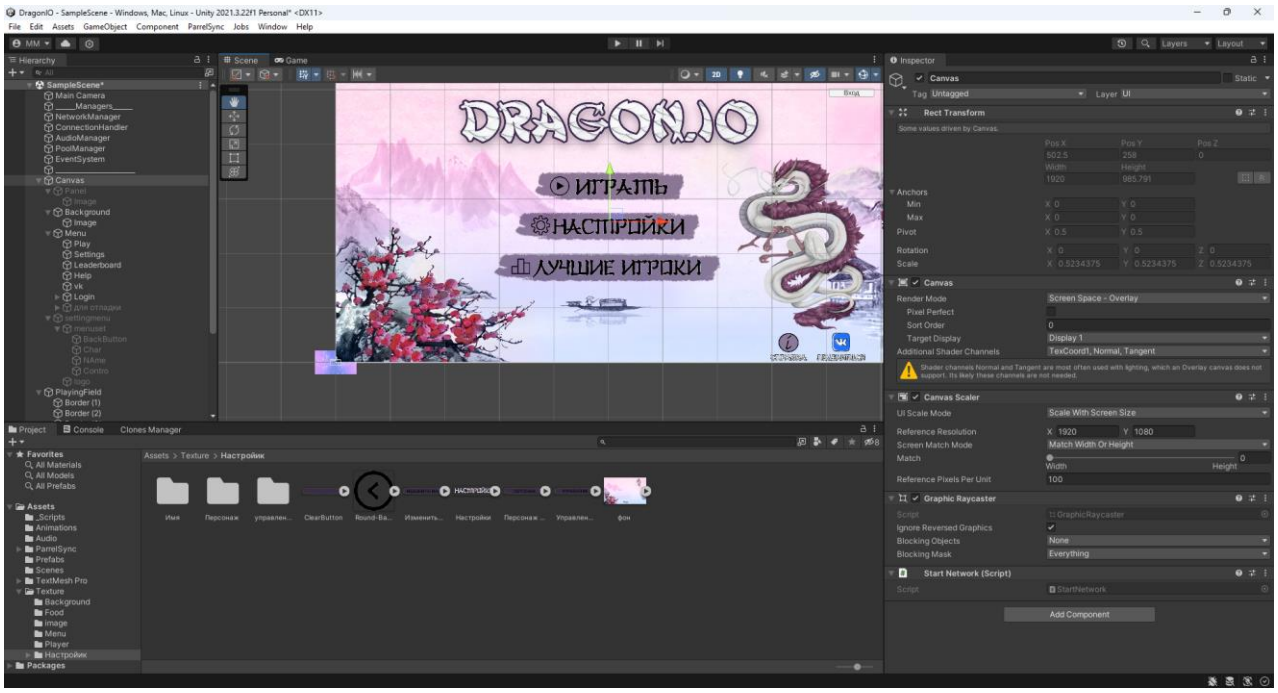

Рисунок 3.4 — Формирование оффлайн сцены

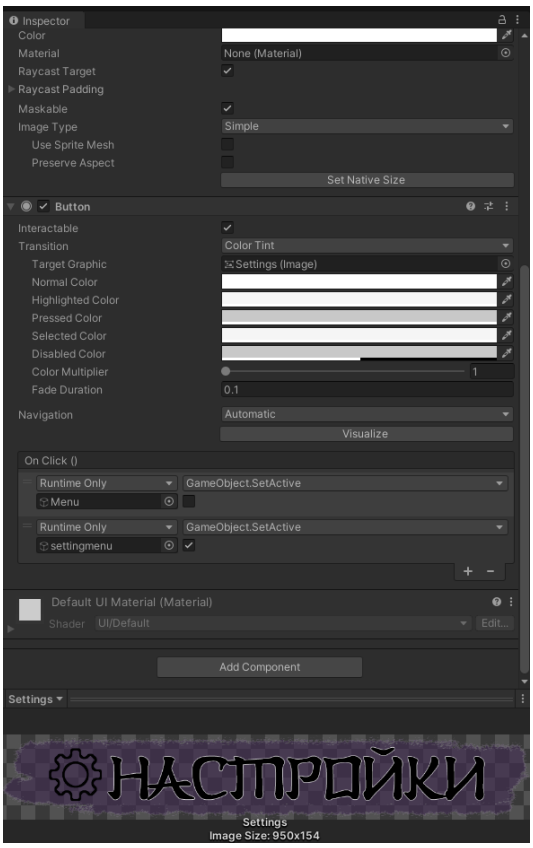

Рисунок 3.5 — Пример настроек перехода между окнами

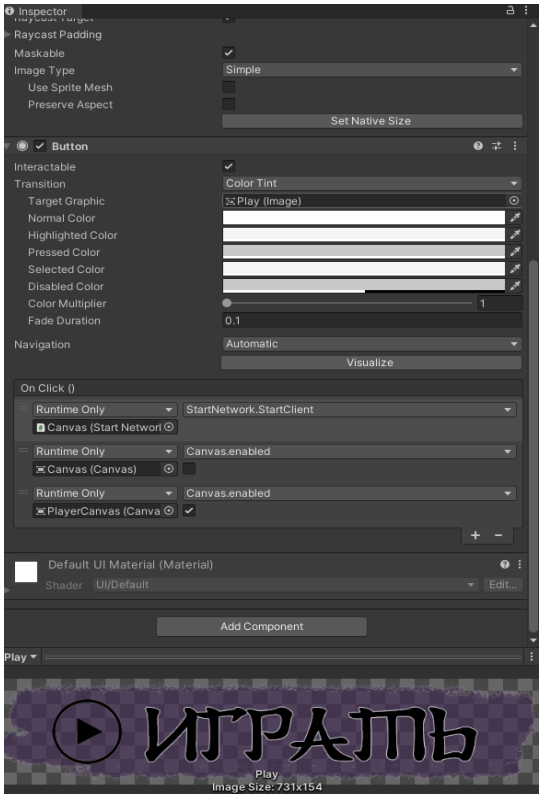

Рисунок 3.6 — Настройка кнопки «Играть»

Таким же образом формируется онлайн сцена, на которой размещены фоновый рисунок, текущая таблица лидеров, игровой счет, еда и персонажи. На рисунке 3.7 представлен итоговый вид онлайн сцены.

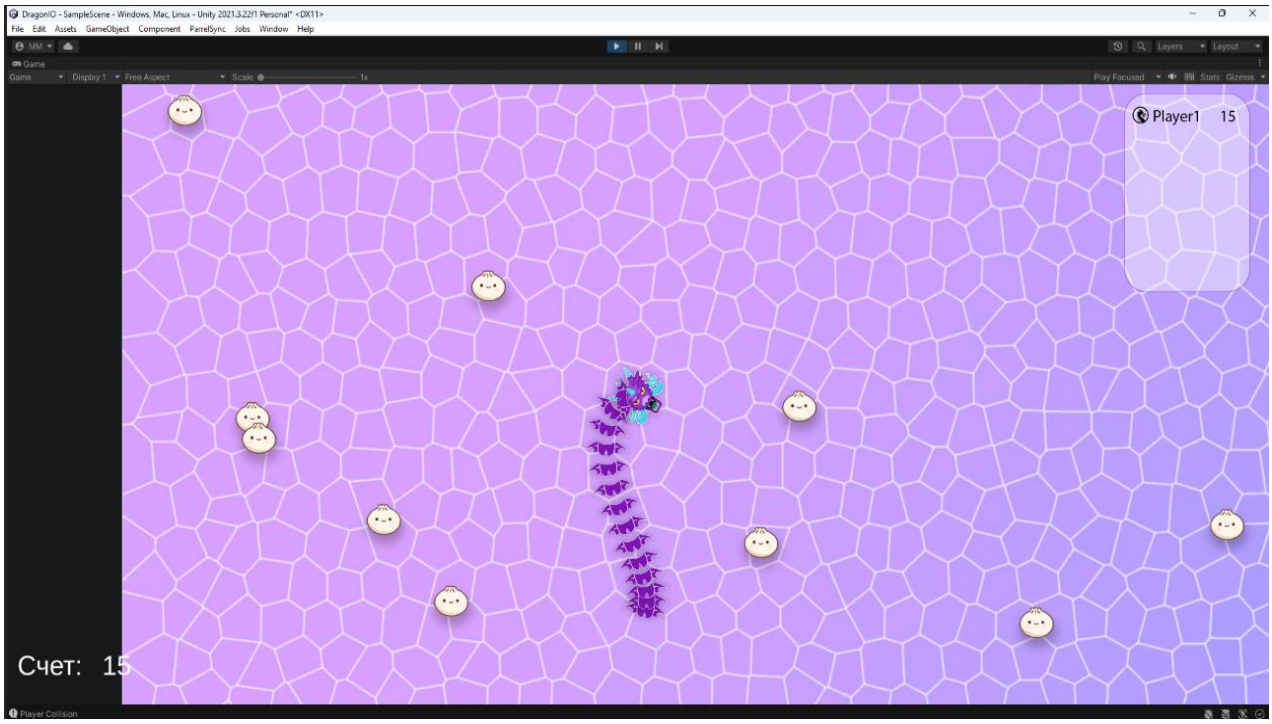

Рисунок 3.7 — Онлайн сцена

3.5 Реализация мультиплеера

Так, как игра Dragon.io многопользовательская, необходимо что бы пользователи могли играть одновременно и видеть друг друга.

Для реализации мультиплеера создан пустой объект Network, который отвечает за связь клиента с сервером. Он содержит в себе два компонента – NetworkManager и UnityTransport.

NetworkManager обеспечивает функции для управления подключениями игроков к серверу. Он обрабатывает подключение новых игроков, отключение игроков и обработку ошибок сетевого взаимодействия.

UnityTransport предоставляет набор функций и классов для управления сетевыми соединениями, передачи данных между клиентом и сервером и обработки сетевых событий.

На рисунке 3.8 представлены настройки компонента Network.

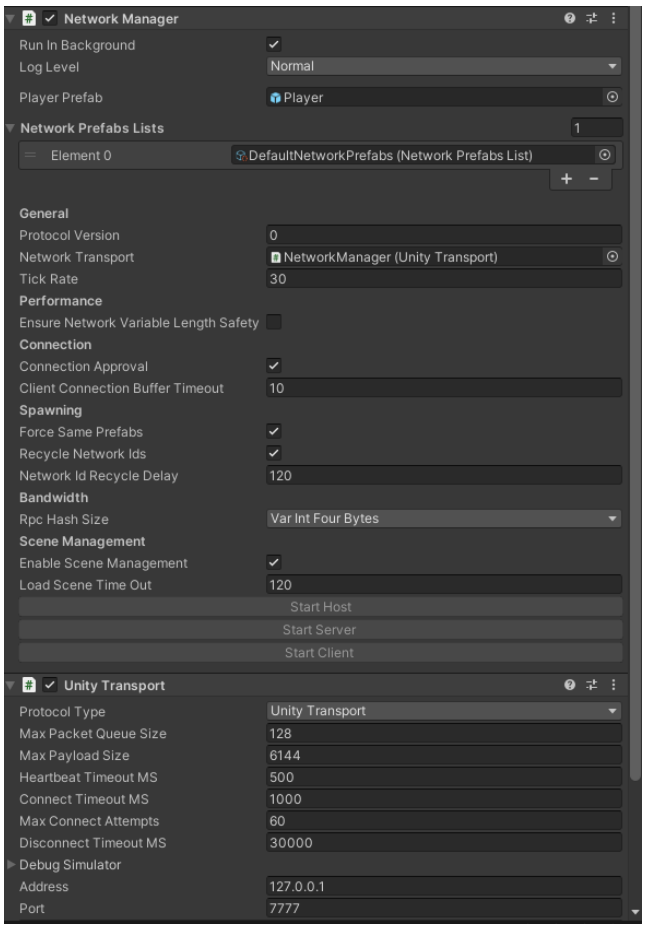

Рисунок 3.8 — Настройки компонента Network

После создаются префабы для основных объектов.

Префаб – это готовый шаблон объекта, который представляет собой заранее настроенный объект со всеми его компонентами, свойствами и настройками.

Player – префаб для игрока (Рисунок 3.9). Данный префаб размещается в специальном поле созданного ранее объекта Network. К префабу добавляются компоненты:

– Sprite Renderer, который отвечает за визуализацию игрового персонажа;

– Circle Collider 2D, который отвечает за взаимодействия персонажей на основе формы круга в двухмерном пространстве;

– Rigidbody 2D, который, используется для добавления физического поведения к персонажу в двухмерном пространстве;

– Animator, который отвечает за анимацию объектов;

– Player Controller, который отвечает за перемещение персонажа;

– Player Length, который отвечает за размер и отображение хвоста персонажа;

– Network Animator, который отвечает за сетевую анимацию;

– Network Object, который отвечает за идентификацию объекта внутри сети;

– Network Transform, отвечает за синхронизацию положения и вращения объекта между клиентами и сервером.

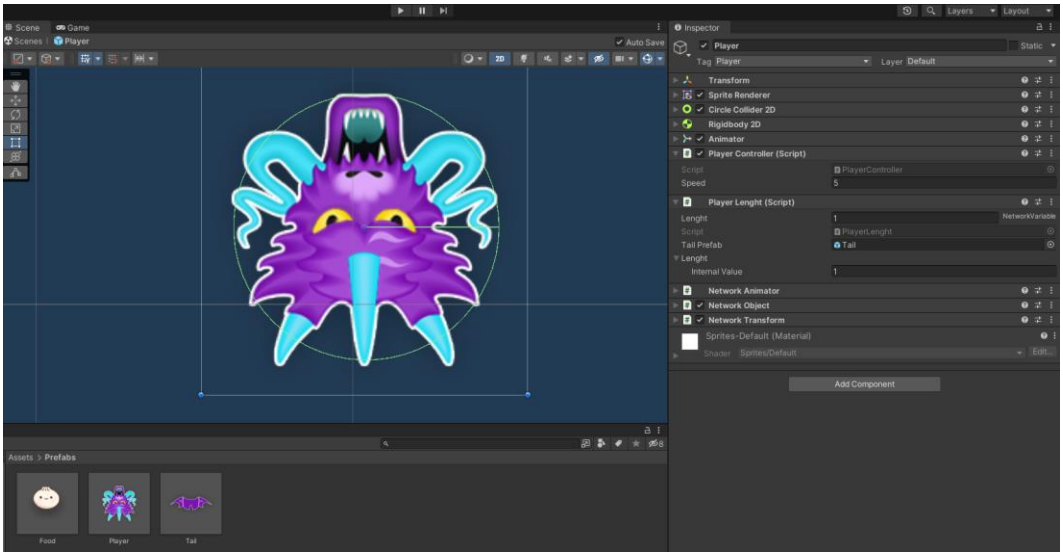

Рисунок 3.9 — Префаб игрока

Tail - префаб для хвоста (Рисунок 3.10). К нему добавляются компоненты:

– Sprite Renderer;

– Circle Collider 2D;

– Rigidbody 2D;

– Tail, который отвечает за перемещение хвоста.

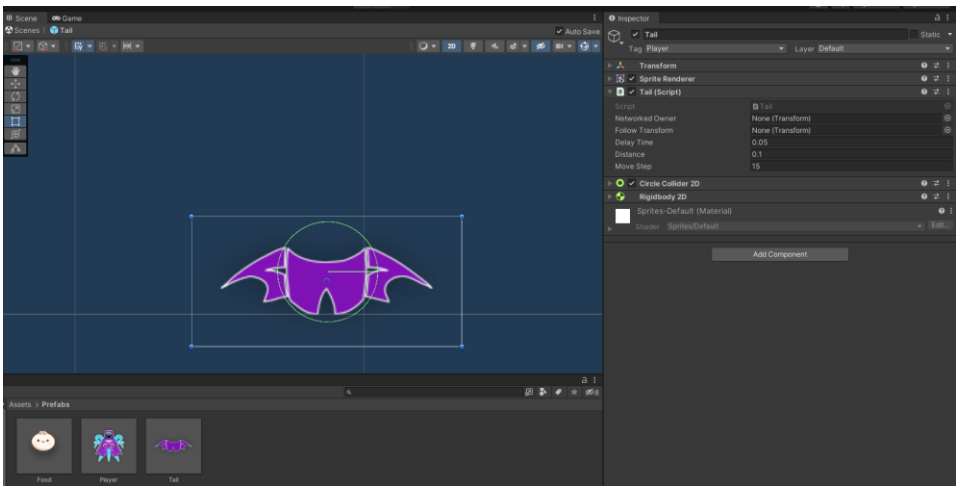

Рисунок 3.10 — Префаб хвоста

Food – префаб для еды (Рисунок 3.11). К нему добавляются компоненты:

– Sprite Renderer;

– Polygon Collider 2D, который отвечает за взаимодействия персонажей на основе формы произвольного многоугольника в двухмерном пространстве;

– Rigidbody 2D;

– NetworkObject;

– Food, который отвечает за отображение еды.

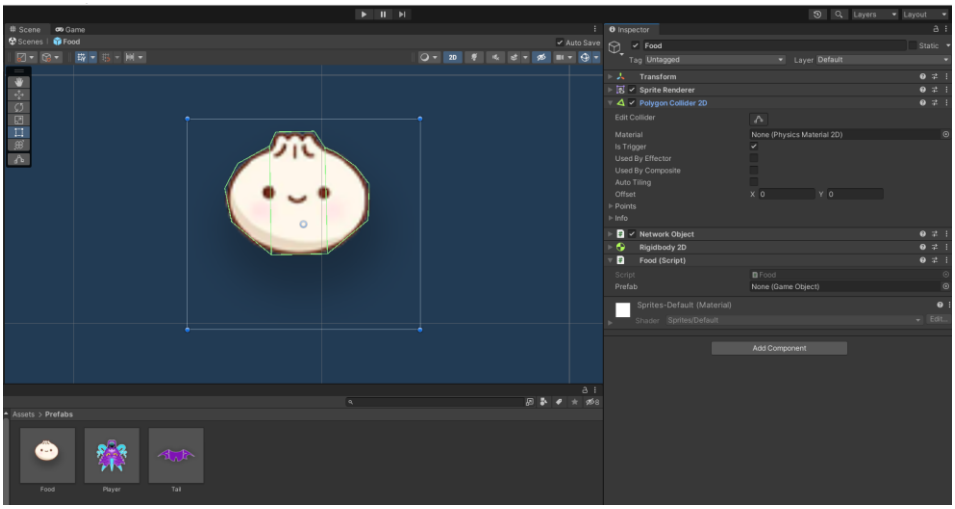

Рисунок 3.11 — Префаб еды

Работа мультиплеера представлена на рисунке 3.12.

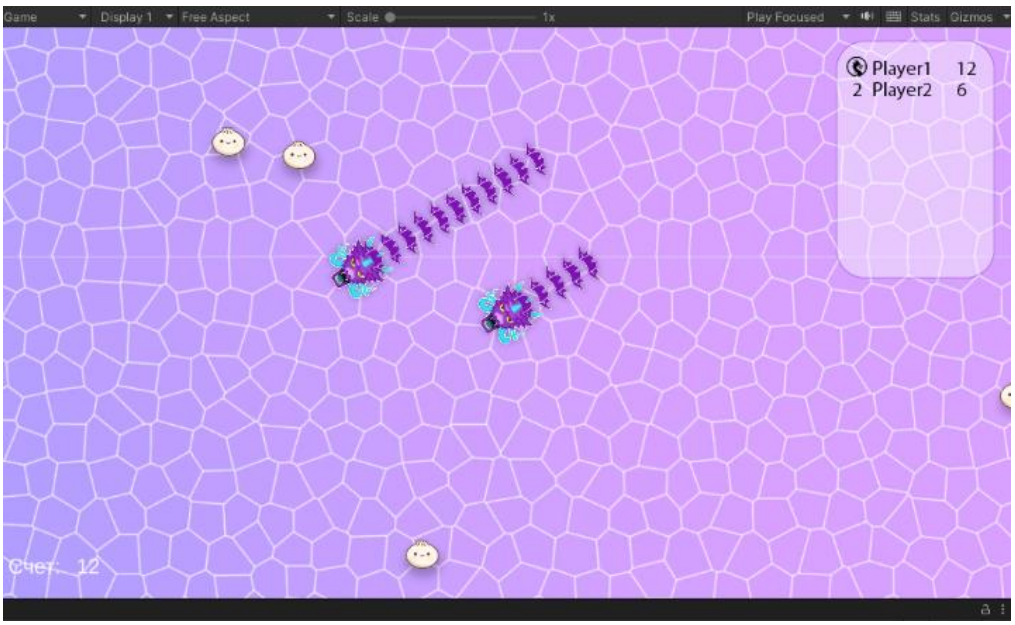

Рисунок 3.12 — Мультиплеер.

3.6 Скриптинг

В Unity скрипты рассматриваются в качестве объектов, которые используются для создания функциональности и поведения объектов в игре. Как и другие компоненты в Unity, они могут быть прикреплены к объектам. Это видно в окне инспектора.

Скрипты позволяют управлять поведением объектов, обрабатывать входные данные, управлять анимацией, взаимодействовать с другими объектами и другое.

Ниже представлены основные скрипты.

3.6.1 PlayerController

Данный скрипт отвечает за управление персонажем. Он привязан к префабу Player.

Скрипт содержит:

– запрос к серверу на перемещение персонажа (Рисунок 3.13);

– запрос к серверу на обработку столкновений (Рисунок 3.14);

– слежение камеры за персонажем (Рисунок 3.15).

```
private void MovePlayerServerRpc(Vector3 mouseWorldCoor-
dinates)
    {
        // Перемещение в сторону курсора
        transform.position = Vector3.MoveTowards(trans-
form.position, mouseWorldCoordinates, Time.deltaTime *
speed);
              // Поворот в сторону курсора
        if (mouseWorldCoordinates != transform.position)
        {
            Vector3 targetDirection = mouseWorldCoordi-
nates - transform.position;
            targetDirection.z = 0f;
            transform.up = targetDirection;
        }
    }
```

Рисунок 3.13 — Запрос на перемещение персонажа

```
private void WinInformationServerRpc(ulong winner, ulong
loser)
    {
        _targetClientsArray[0] = winner;
        ClientRpcParams clientRpcParams = new Cli-
entRpcParams
        {
            Send = new ClientRpcSendParams
            {
                TargetClientIds = _targetClientsArray
            }
        };
        AtePlayerClientRpc(clientRpcParams);

        _targetClientsArray[0] = loser;
        clientRpcParams.Send.TargetClientIds = _tar-
getClientsArray;
        GameOverClientRpc(clientRpcParams);
    }
```

Рисунок 3.14 — Запрос на обработку столкновений

```
    private void CameraMovement()
    {
        _mainCamera.transform.localPosition = new Vec-
tor3(transform.position.x, transform.position.y, -1f);
        transform.position =
            Vector2.MoveTowards(transform.position,
                _mainCamera.transform.localPosition,
Time.deltaTime);
    }
```

Рисунок 3.15 — Слежение камеры

3.6.2 Tail

Данный скрипт отвечает за поведение хвоста (Рисунок 3.16). Он привязан к префабу Player.

```
public class Tail : MonoBehaviour
{
    // Which networked owner these tails belong to
    public Transform networkedOwner;
    // The Transform object that this tail is following
    public Transform followTransform;
    [SerializeField, Tooltip("Задержка между игровыми
обьектами")] private float delayTime = 0.1f;
    [SerializeField, Tooltip("Расстояние, между игровыми
объектами")] private float distance = 0.3f;
    [SerializeField, Tooltip("Скорость движения")] pri-
vate float moveStep = 10f;
    private Vector3 _targetPosition;
    /// <summary>
    /// Движение частей хвоста
    /// </summary>
    private void Update()
    {
        _targetPosition = followTransform.position - fol-
lowTransform.forward * distance;
        _targetPosition += (transform.position - _target-
Position) * delayTime;
        _targetPosition.z = 0f;

        transform.position = Vector3.Lerp(transform.posi-
tion, _targetPosition, Time.deltaTime * moveStep);
    }}
```

Рисунок 3.16 — Поведение хвоста

3.6.3 PlayerLenght

Данный скрипт отвечает за размер хвоста (Рисунок 3.17). С сервера принимается значение длины и новая часть хвоста отрисовывается на клиенте. Скрипт привязан к префабу Player.

```
/// <summary>
    /// Спавн новой части хвоста
    /// </summary>
    private void InstantiateTail()
    {
        GameObject tailGameObject = Instantiate(tailPrefab,
transform.position, Quaternion.identity);
        tailGameObject.GetComponent<SpriteRenderer>().sort-
ingOrder = -length.Value;
        if (tailGameObject.TryGetComponent(out Tail tail))
        {
            tail.networkedOwner = transform;
            tail.followTransform = _lastTail;
            _lastTail = tailGameObject.transform;
            Physics2D.IgnoreCollision(tailGameObject.GetCom-
ponent<Collider2D>(), _collider2D);
        }
        _tails.Add(tailGameObject);
    }
```

Рисунок 3.17 — Размер хвоста

3.6.4 StartNetwork

Данный скрипт отвечает за подключение к серверу (Рисунок 3.18).

```
private void TicketAssigned(MultiplayAssignment assignment)
    {
        Debug.Log($"Ticket Assigned: {assignment.Ip}:{as-
signment.Port}");
        NetworkManager.Singleton.GetComponent<Uni-
tyTransport>().SetConnectionData(assignment.Ip, (ushort)as-
signment.Port);
        NetworkManager.Singleton.StartClient();
    }
```

Рисунок 3.18 — Подключение к серверу

3.6.5 Food

Данный скрипт запускает триггер столкновения с едой (Рисунок 3.19). Если персонаж столкнулся с едой, а не с другим игроком, то на сервер посылается запрос на увеличение длины.

```
/// <summary>
/// Входит в Networked Food object.
/// </summary>
public class Food : NetworkBehaviour
{
    /// <summary>
    /// При столкновении с игроком добавляет длину хвоста и
удаляет объект со сцены.
    /// Мы проверяем, являемся ли мы сервером, прежде чем до-
бавлять длину хвоста
    /// </summary>
    private void OnTriggerEnter2D(Collider2D col)
    {
        if (!IsServer) return;
        if (!col.CompareTag("Player")) return;

        if (col.TryGetComponent(out PlayerLength playerLength))
        {
            playerLength.AddLengthServer();
        }
        else if (col.TryGetComponent(out Tail tail))
        {
            tail.networkedOwner.GetComponent<Player-
Length>().AddLengthServer();
        }
        if (NetworkObject.IsSpawned) NetworkObject.Despawn();
    }
}
```

Рисунок 3.19 — Триггер столкновения с едой

3.7 Сборка

Для сборки приложения были использованы средства движка Unity. Целевая платформа сборки – Microsoft Windows. Параметры сборки представлены на рисунке 3.20.

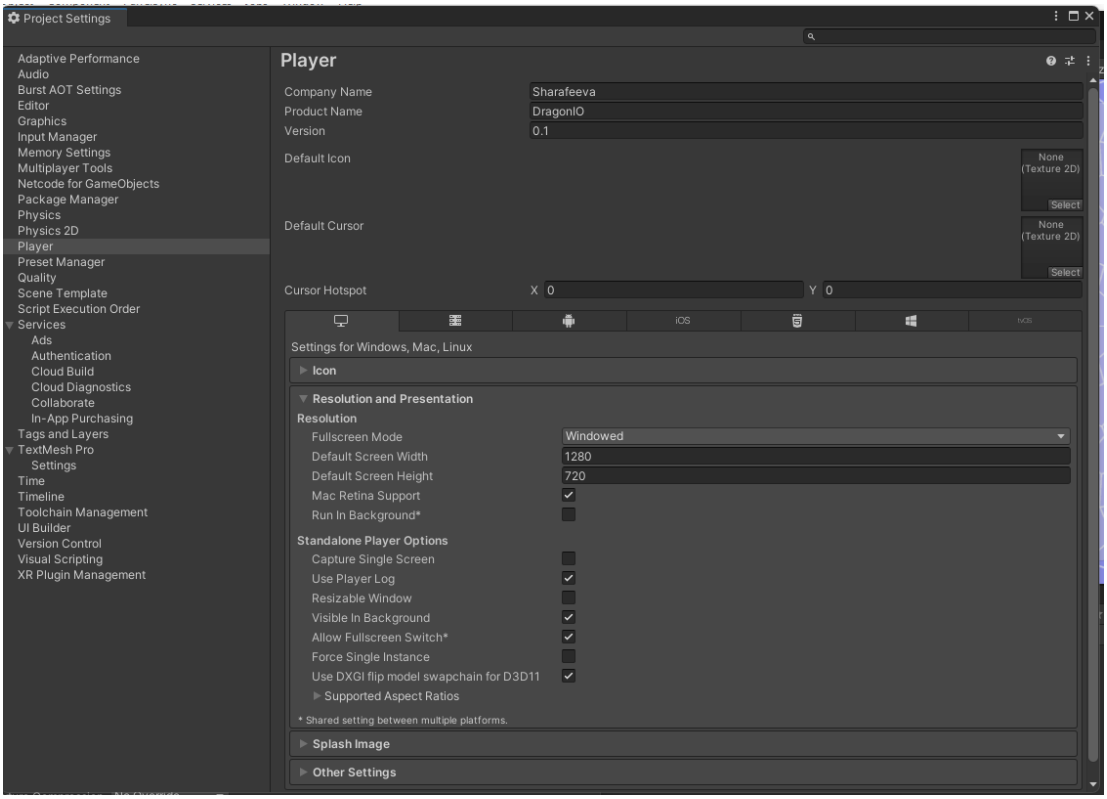

Рисунок 3.20 — Параметры сборки приложения

Окно сборки приложения представлено на рисунке 3.21

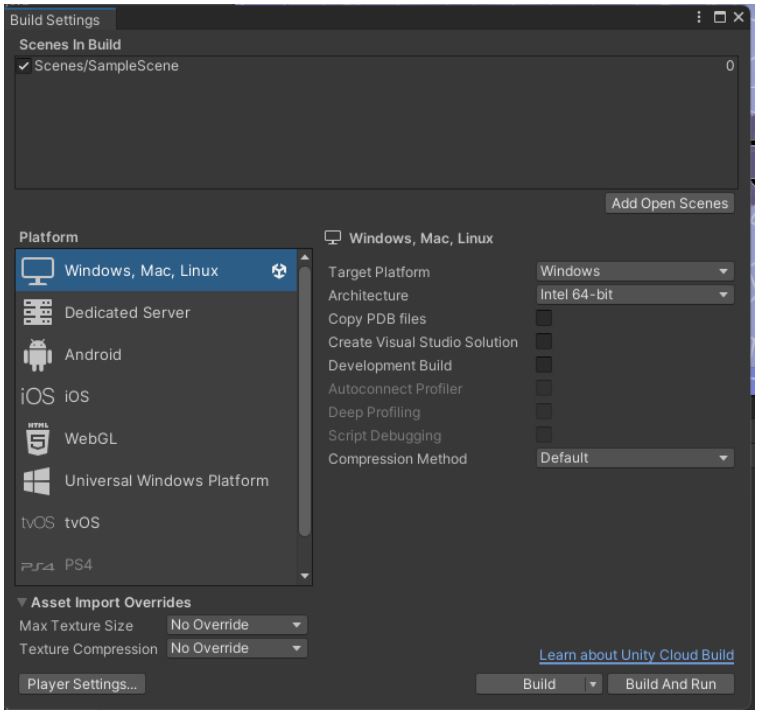

Рисунок 3.21 — Окно сборки

3.8 Тестирование

Первый этап тестирования приложения проводился в ручном режиме.

Второй этап осуществлялся с помощью загрузки сборки приложения на тематический сервер в Discord, посвященный Unity разработке. В тестировании приняли участие 9 человек, был выявлен и исправлен ряд ошибок.

Например, в определенной точке карты персонаж мог уйти за границу игрового поля (Рисунок 3.22). Ошибка возникала из-за неправильно настроенной границы.

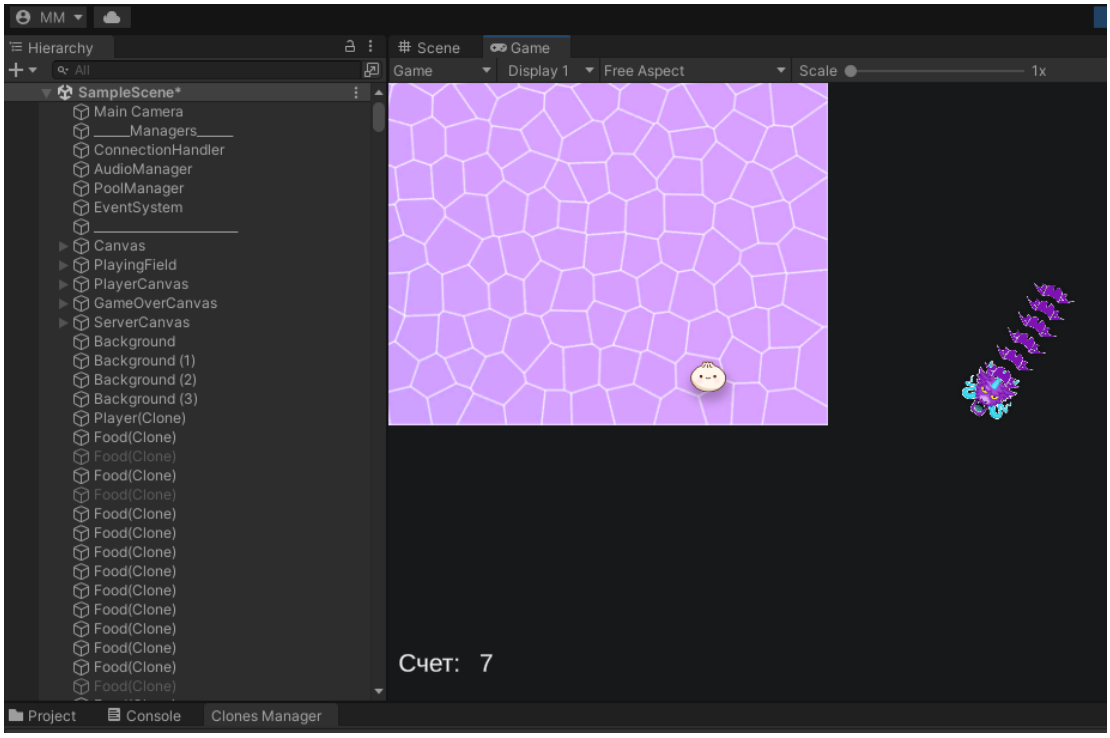

Рисунок 3.22 — Пример ошибки

3.9 Инструкция разработчика

По ходу написания кода программы использовались XML-теги. Это такие специальные теги XML, которые содержаться в комментариях и описывают свойства или методы в конкретном файле. На рисунке 3.23 представлен пример комментария для тега.

```
/// <summary>
/// Перемещает игрока, если он является владельцем.
/// </summary>
private void Update()
{
if (!IsOwner || !Application.isFocused) return;
MovePlayerServer();
}
```

Рисунок 3.23 Пример комментария для тега

Документация к коду сгенерирована автоматически с помощью системы Doxygen [11] в html-формате (Рисунок 3.24).

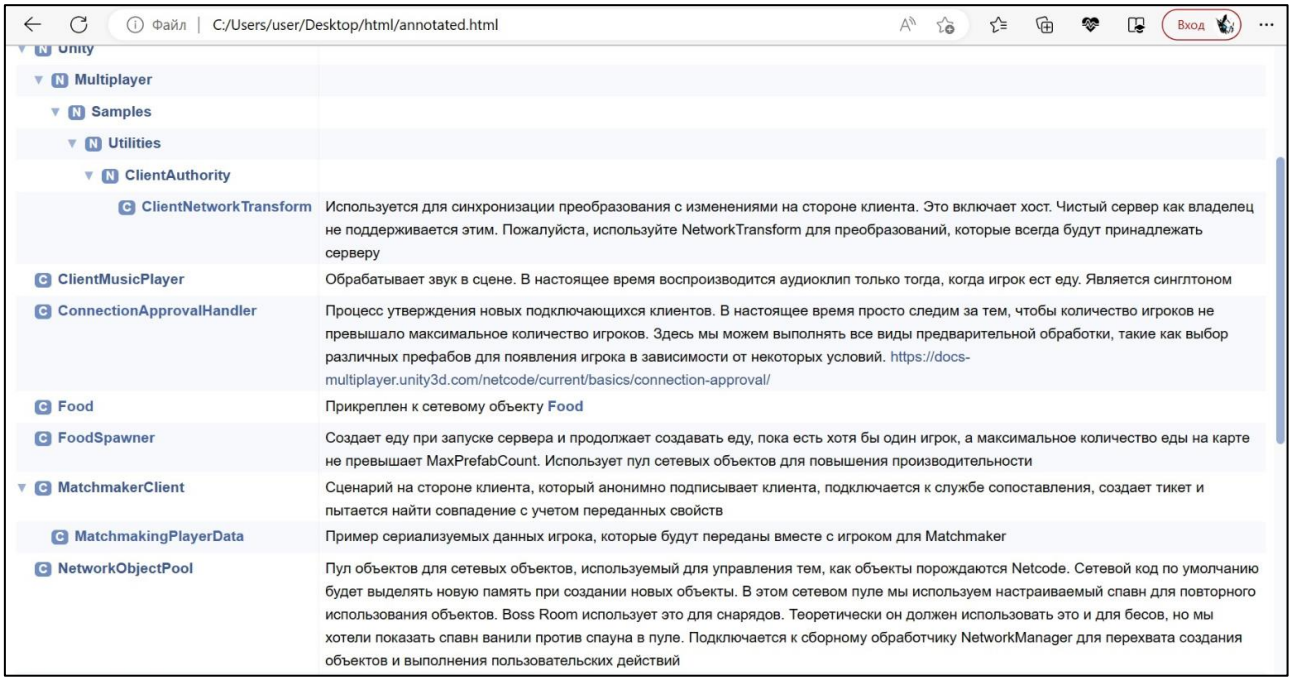

Рисунок 3.24 — Пример документации

3.10 Выводы по главе

1. Реализован прототип клиентского приложения «Dragon.io».
2. Создана сборка клиентского приложения.
3. Проведено тестирование приложения.
4. Составлена инструкция разработчика на основе системы Doxygen.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы:

1. Спроектировано, реализовано и протестировано клиентское приложение «Dragon.io».
2. Создано удобное окружение для разработки с использованием движка Unity.

В разработанном приложении присутствуют недостатки, исправить их можно путем пополнения функциональных возможностей:

– добавить режим хоста, для игры на закрытом для других пользователей поле;

– добавить систему рангов;

– добавить систему ежедневных заданий и наград;

– добавить дополнительные возможности для персонажей;

– добавить нестандартные модели для персонажей;

– улучшить взаимодействие с базой данных, сделать авторизацию через социальные сети;

– другого рода улучшения.

Исходный код приложения доступен для скачивания с git-репозитория [12].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сервис, анализирующий сайты. PR-CY. — Режим доступа: https://a.pr-cy.ru/ (дата обращения: 12.12.2022).

2. Многопользовательская браузерная игра. Snakeio. — Режим доступа: https://snake.io/ (дата обращения: 5.12.2022).

3. Многопользовательская браузерная игра. Wormax. — Режим доступа: https://wormax.io/ (дата обращения: 5.12.2022).

4. Спецификация OpenApi. OpenApi. — Режим доступа: https://www.open-apis.org/ (дата обращения: 20.05.2023)

5. Многопользовательская браузерная игра. Slitherio. — Режим доступа: http://slither.io/ (дата обращения: 5.12.2022).

6. Методология разработки программного обеспечения ICONIX. ICONIX. — Режим доступа: https://iconix.ru (дата обращения: 20.05.2023)

7. Документация Unity. Unity. — Режим доступа: https://docs.unity3d.com/Manual (дата обращения 9.03.2022)

8. Netcode. Netcode for GameObjects. — Режим доступа: https://unity.com/products/netcode (дата обращения 25.04.2023)

9. Hyper Casual Games. Gillion. — Режим доступа: https://gdcuffs.com/gd-stuff-hyper-casual (дата обращения 5.05.2023)

10. Photoshop. Adobe. — Режим доступа: https://www.adobe.com/ru/products/photoshop.html (дата обращения 10.10.2023)

11. Система документирования Doxygen. Doxygen— Режим доступа: https://www.doxygen.nl/

12. Dragon.io. GitHub. — Режим доступа: https://github.com/AKASer2022/DragonIO (дата обращения: 20.05.2023)

13. Гурвиц, Г. Разработка реального приложения в среде клиент-сервер: учебное пособие / Г. Гурвиц. — Москва : ДВГУПС, 2005. — 206 с.

14. Буч, Г. Введение в UML от создателей языка: учебное пособие / Г. Буч, Д. Рамбо, И. Якобсон : Издательство «ДМК-Пресс», 2015. — 496 с.

15. PlantUml. Plantumlwebserver. — Режим доступа: http://www.plantuml.com (дата обращения 5.05.2023)

16. UML. Блог программиста. — Режим доступа: https://proprof.com/archives/2594 (дата обращения: 20.05.2023)

17. Нотации модели сущность – связь. Блог программиста. — Режим доступа: https://pro-prof.com/archives/8126 (дата обращения: 20.05.2023)

18. ГОСТ 7.32-2001. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления.

19. ГОСТ 7.9-95 (ИСО 214-76). Система стандартов по информации, библиотечному и издательскому делу. Реферат и аннотация. Общие требования.

20. ГОСТ 7.1-2003. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

21. СТО 4.2-07-2014. Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности.

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
"12" 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

Клиент-серверная игра «Змейка». Клиентская часть

Руководитель	подпись	12.06.23 дата	ст. преподаватель должность, ученая степень	В.С. Васильев
Выпускник	подпись	12.06.23 дата		А.Р. Шарафеева
Нормоконтролер	подпись	12.6.23 дата		В.С. Васильев

Красноярск 2023