Частное учреждение образования

«Колледж бизнеса и права»

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ «First Person Shooter»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

«Конструирование программ и языки программирования»

КП Т.592020.401

Выполнил А.И. Сизов

Руководитель проекта Е.В. Багласова

2018

Содержание

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

*КП Т.592020.401 ГЧ*

Разраб.

*Сизов А.И.*

Провер.

Багласова Е.В

Т. Контр.

Н. Контр.

Утверд.

*Программная реализация игры «First Person Shooter»*

Лит.

Листов

50

КБП

[Введение 4](#_Toc517935875)

[1 Объектно-ориентированный анализ и проектирование системы 5](#_Toc517935876)

[1.1 Назначения и цели создания 5](#_Toc517935877)

[1.2 Проектирование модели 5](#_Toc517935878)

[2 Вычислительная система 9](#_Toc517935879)

[2.1 Требования к аппаратным и операционным ресурсам 9](#_Toc517935880)

[2.2 Инструменты разработки 9](#_Toc517935881)

[3 Проектирование задачи 11](#_Toc517935882)

[3.1 Требования к приложению 11](#_Toc517935883)

[3.2 Концептуальный прототип 12](#_Toc517935884)

[3.3 Организация данных 13](#_Toc517935885)

[3.4 Функции и элементы управления 15](#_Toc517935886)

[4 Описание программного средства 16](#_Toc517935887)

[4.1 Общие сведения 16](#_Toc517935888)

[4.2 Функциональное назначение 16](#_Toc517935889)

[4.3 Входные и выходные данные 16](#_Toc517935890)

[5 Методика испытаний 17](#_Toc517935891)

[5.1 Технические требования 17](#_Toc517935892)

[5.2 Функциональное тестирование 17](#_Toc517935893)

[6 Применение 18](#_Toc517935894)

[6.1 Назначение программы 18](#_Toc517935895)

[6.2 Условия применения 18](#_Toc517935896)

[Заключение 19](#_Toc517935897)

[Список информационных источников 20](#_Toc517935898)

[Приложение А Текст программы 21](#_Toc517935899)

# Введение

Целями практики по разработке и сопровождению программного обеспечения являются:

* закрепление знаний, связанных с технологией обработки информации;
* приобретение навыков разработки программ с использованием современных средств обработки экономической и деловой информации;
* выработка умений применять средства стандартных библиотек в практических задачах.

Задачами практики по разработке и сопровождению ПО являются:

* углубленное изучение принципов организации программного обеспечения и технологии его проектирования;
* умение разрабатывать программы в соответствии с промышленными требованиями, обеспечивая высокий уровень качества программного обеспечения и экономической эффективности;
* изучение способов определения экономической себестоимости и эффективности разработки программного обеспечения;
* оформление комплекта документации на созданное программное обеспечение.

##### Таблица 1.1 – Календарный график работы

|  |  |
| --- | --- |
| Период | Наименование и содержание работ |
| 09.02.18 | Изучение требований к рабочей дисциплине и технике безопасности.  Изучение нормативных документов. |
| 11.02.18-  17.02.18 | Ознакомление с различным программным обеспечением, используемым для разработки программ |
| 18.02.18-  24.02.18 | Получение индивидуальных заданий по созданию программ для обработки экономической и деловой информации. Обсуждение тем индивидуальных заданий. |
| 25.02.18-  02.03.18 | Изучение общих требований к программному обеспечению обработки экономической и деловой информации. Требования к интерфейсу пользователя, выходным документам (печатные, экранные формы, форматы результирующих файлов с результатами расчетов). |
| 03.03.18-  23.03.18 | Анализ предметной области индивидуального задания. Выбор и обоснование языка и среды программирования. Описание функциональных возможностей задачи. |
| 24.03.18-  06.04.18 | Технология создания программного обеспечения. Правила построения диаграмм UML. Проектирование ПО при помощи CASE-средства Rational Rose |
| 07.04.18-  20.04.18 | Конструирование алгоритма и его реализация в выбранной визуальной среде программирования. Разработка интерфейса. Проектирование классов, определение свойств и методов, определение связей между классами. |
| 21.04.18-  04.05.18 | Создание информационной базы задачи. Тестирование и отладка программы с использованием подготовленных тестовых примеров. Проектирование модулей, позволяющих осуществить защиту программного продукта от несанкционированного доступа. |
| 05.05.18-  25.05.18 | Критерии оценки качества программного продукта. Изучение нормативных документов, регламентирующих оформление технического проекта. Создание документации по эксплуатации и сопровождению разработанного ПС. Проведение испытания ПС в нормальных и критических условиях. |
| 26.05.18-  01.06.18 | Анализ возможности модернизации проекта с учетом альтернативных решений. Оформление аннотации на программное средство. |
| 02.06.18-  15.06.18 | Оформление отчета по практике. Подготовка листингов программ с дополнением комментариев. Распечатка экранных форм. |
| 16.06.18-  21.06.18 | Демонстрация программ по индивидуальным заданиям.  Защита отчета по практике. |

# 1 Объектно-ориентированный анализ и проектирование системы

## 1.1 Назначения и цели создания

Тема курсового проекта - программная реализация игры «First Person Shooter». Данная игра будет относиться к жанру шутер и будет реализована на игровом движке Unity. Шу́тер (Стрелялка, [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) shooter — «стрелок») — жанр [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0). На момент зарождения жанра за рубежом укрепилось слово «шутер», как вариант описания игрового процесса и перевод для слова shooter, в России и некоторых других странах СНГ жанр изначально был назван в среде любителей как «стрелялка».

Основные признаки шутера:

* игрок находится в трёхмерном пространстве и имеет некоторую свободу передвижения;
* [уровни](https://ru.wikipedia.org/wiki/%D0%A3%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D0%B8%D0%B3%D1%80%D1%8B)), как правило, являют собой ограниченный лабиринт, в котором расположены враги и союзники;
* действие большинства шутеров разворачивается в анизотропном пространстве (помещения имеют очевидные пол и потолок, в них действует гравитация), хотя существуют и исключения, в которых пространство [изотропно](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%BE%D1%82%D1%80%D0%BE%D0%BF%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%B0) ([Descent](https://ru.wikipedia.org/wiki/Descent));
* [геймплей](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%B9%D0%BC%D0%BF%D0%BB%D0%B5%D0%B9) [канонического](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%BE%D0%BD_(%D0%B8%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%BE)) шутера сводится к поиску выхода с уровня, с устранением всех препятствующих противников и препятствий;
* в расширенном жанре ставятся дополнительные цели, такие как установка бомбы на вражеское укрепление, решение паззлов за отведённое время ([Call of Duty](https://ru.wikipedia.org/wiki/Call_of_Duty));
* наличие на экране таких отличительных индикаторов как жизнь, броня, количество патронов, режим стрельбы, время матча, цель мисси, мини карта.

Проект «FPS» будет многопользовательской компьютерной игрой в жанре шутер от первого лица. Основная идея игры – противостояние роботов. Каждый из них будет иметь вооружение, выбираемое пользователем. Цель игры — набрать большее количество очков до истечения времени, либо набрать максимум очков, установленных картой.

В проекте «FPS» будут реализованы такие отличительные качества жанра «шутер» как наличие в нижнем левом углу экрана индикатора количества жизней как в графическом, так и в процентном соотношении, в нижнем правом углу экрана индикатор количества патронов как вообще, так и в обойме, сверху будет находиться время до окончания матча и количество очков до победы. Действия будут разворачиваться в анизотропном пространстве.

## 1.2 Проектирование модели

Главной целью проектирования моделей является отображение функциональной структуры объекта, то есть производимые ими действия и связи между этими действиями. Наиболее распространенным средством моделирования данных являются диаграммы «сущность-связь», которые предназначены для графического представления моделей данных разрабатываемой программной системы и предлагают некоторый набор стандартных обозначений для определения данных и отношений между ними. С помощью этого вида диаграмм можно описать отдельные компоненты концептуальной модели данных и совокупность взаимосвязей между ними, имеющих важное значение для разрабатываемой системы. Основными понятиями данной нотации являются понятия сущности и связи. При этом под сущностью понимается произвольное множество реальных или абстрактных объектов, каждый из которых обладает одинаковыми свойствами и характеристиками. В этом случае каждый рассматриваемый объект может являться экземпляром одной и только одной сущности, должен иметь уникальное имя или идентификатор, а также отличаться от других экземпляров данной сущности. Связь определяется как отношение или некоторая ассоциация между отдельными сущностями. Примерами связей могут являться родственные отношения типа «отец-сын» или производственные отношения типа «начальник-подчиненный». Другой тип связей задается отношениями «иметь в собственности» или «обладать свойством».

Графическая модель данных строится таким образом, чтобы связи между отдельными сущностями отражали не только семантический характер соответствующего отношения, но и дополнительные аспекты обязательности связей, а также кратность участвующих в данных отношениях экземпляров сущностей.

Исходя из исследования предметной области, можно выделить следующие сущности разработки: Карта, Персонаж, Оружие.

Для сущности «Персонаж» существует такой атрибут как тип персонажа, которым будет управлять пользователь

Для сущности «Оружие» существуют такие атрибуты как урон, скорострельность, точность.

Диаграмма «Сущность-связь» представлена на рисунке 1.1.

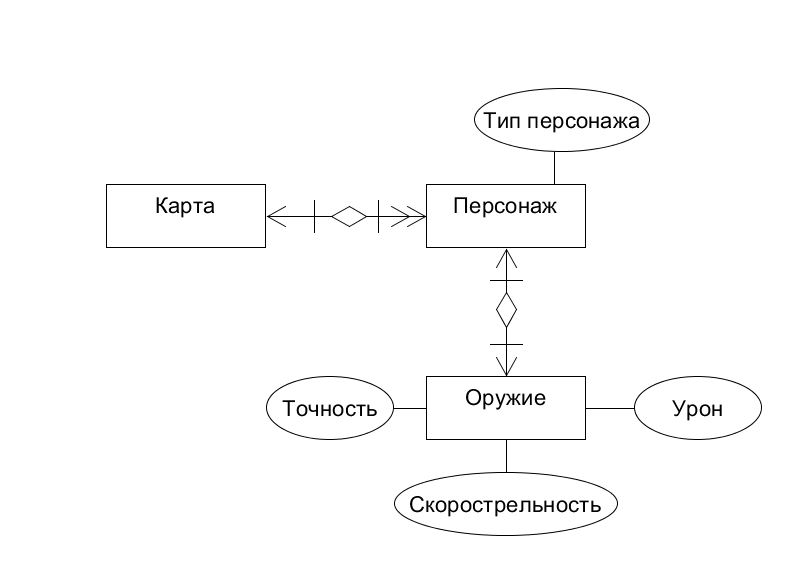


Рисунок 1.1 – Диаграмма «Сущность-связь»

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций – диаграмм. В терминах языка UML определены следующие виды диаграмм: диаграмма вариантов использования, диаграмма классов, диаграмма деятельности, диаграмма компонентов.

Перечень этих диаграмм представляет собой неотъемлемую часть графической нотации языка UML, сам процесс объектно-ориентированного программирования (ООП) неразрывно связан с процессом построения этих диаграмм. Совокупность построенных таким образом диаграмм содержит всю информацию, необходимую для реализации проекта сложной системы.

Диаграмма – граф специального вида, состоящий из вершин в форме геометрических фигур, которые связаны между собой ребрами или дугами.

Суть диаграммы вариантов использования состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования.

Варианты использования описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности. Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы. Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы.

В данной проектируемой системе в качестве актера выступает обычный пользователь, который служит источником воздействия на моделируемую систему.

К основным функциям разрабатываемой программы относятся следующие функции:

* создание сервера;
* синхронизация с сервером;
* стрельба;
* перемещения.

Диаграмма вариантов использования представлена в графической части на листе 1.

Диаграмма компонентов описывает объекты реального мира – компоненты программного обеспечения. Эта диаграмма позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код.

Диаграмма компонентов представлена в графической части на листе 2.

Диаграмма деятельности — [UML](https://ru.wikipedia.org/wiki/UML)-диаграмма, на которой показаны действия, состояния которых описано на диаграмме состояний. Под деятельностью ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) activity) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Диаграммы деятельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений.

Диаграмма деятельности представлена в графической части на листе 3

Если рассматривать взаимодействия объектов во времени, тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Временной аспект поведения имеет существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Именно для этой цели и используются диаграммы последовательности, в которых ключевым моментом является динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения: одно - слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии; второе - вертикальная временная ось, направленная сверху вниз, на которой начальному моменту времени соответствует самая верхняя часть диаграммы.

Диаграмма последовательности представлена в графической части на листе 4.

Центральное место в ООП занимает разработка логической модели системы в виде диаграммы классов. Нотация классов в языке UML проста и интуитивно понятна. Схожая нотация применяется и для объектов — экземпляров класса, с тем различием, что к имени класса добавляется имя объекта, и вся надпись подчеркивается.

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами кото­рого являются элементы типа «классификатор» и которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. Поэтому диаграмму классов принято считать графическим представлением таких структурных взаимосвязей логической модели системы, которые не зависят от времени.

Класс в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линями на разделы или секции. Здесь указывается: имя класса, атрибуты и операции (методы).

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельный класс может обозначаться простым прямоугольником с указанием только имени соответствующего класса. По мере проработки отдельных компонентов диаграммы классов дополняются атрибутами и операциями. Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций.

При моделировании поведения проектируемой или анализируемой системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, переход в следующее состояние срабатывает только при завершении этой операции. Графически диаграмма деятельности представляется в форме графа, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому.

Диаграмма классов использования представлена в графической части на листе 5.

# 2 Вычислительная система

## 2.1 Требования к аппаратным и операционным ресурсам

При разработке программы выдвигаются следующие аппаратные требования:

* процессор Intel(R) Core(TM) i5-3210M CPU @ 2.50 ГГц и лучше;
* оперативная память DDR3 6 Гбайт и лучше;
* свободное место на жёстком диске 3Гбайт;
* видеокарта Intel(R) HD 4000 и лучше;
* монитор, мышь, клавиатура;
* подключение к Интернету 56 Кбит/секунду или быстрее.

## 2.2 Инструменты разработки

К инструментам разработки являются:

* операционная система Windows 10;
* игровой движок Unity3D;
* среда Microsoft Visual Studio 2017;
* язык программирования С#.

Операционная система Windows 10 – это продукт семейства [коммерческих](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D0%B5%D1%80%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) (OC) корпорации [Microsoft](https://ru.wikipedia.org/wiki/Microsoft), ориентированных на применение [графического интерфейса](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) при управлении[8].

Система призвана стать единой для разных устройств, таких как [персональные компьютеры](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), [планшеты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD%D1%88%D0%B5%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), [смартфоны](https://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82%D1%84%D0%BE%D0%BD), консоли [Xbox One](https://ru.wikipedia.org/wiki/Xbox_One) и пр. Доступна единая платформа разработки и единый [магазин](https://ru.wikipedia.org/wiki/Windows_Store) [универсальных приложений](https://ru.wikipedia.org/wiki/%D0%A3%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0_Windows), совместимых со всеми поддерживаемыми устройствами. Windows 10 [поставляется в качестве услуги](https://ru.wikipedia.org/wiki/SaaS) с выпуском обновлений на протяжении всего цикла поддержки.

Unity – это [межплатформенная](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0). Unity позволяет создавать приложения, работающие под более чем 20 различными [операционными системами](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), включающими [персональные компьютеры](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), [игровые консоли](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BA%D0%BE%D0%BD%D1%81%D0%BE%D0%BB%D1%8C), [мобильные устройства](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0), [интернет-приложения](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) и другие.

Основными преимуществами Unity являются наличие [визуальной среды разработки](https://ru.wikipedia.org/wiki/RAD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [межплатформенной](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних [библиотек](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))[9].

Microsoft Visual Studio 2017 – Интерактивная интегрированная среда разработки Visual Studio — это оригинальная среда запуска, которая позволяет просматривать и изменять практически любой код, а также отлаживать, выполнять сборку и публиковать приложения для устройств с Android, iOS, Windows, а также в Интернете и облаке. Доступны версии как для компьютеров Mac, так и для компьютеров с Windows. В этой статье вы ознакомитесь с возможностями интегрированной среды разработки Visual Studio. Мы рассмотрим некоторые возможности Visual Studio, установку и использование этой среды, создание простых проектов, отладку и развертывание кода, а также различные окна инструментов.

Visual Studio предлагает набор инструментов, позволяющих с легкостью создавать облачные приложения на базе Microsoft Azure. Она упрощает настройку, сборку, отладку, упаковку и развертывание приложений и служб в Microsoft Azure прямо из IDE[10].

C# (произносится как "си шарп") — простой, современный объектно-ориентированный и типобезопасный язык программирования. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript [11].

C# является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности. Важная особенность таких компонентов — это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также встроенные элементы документации. C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# отлично подходит для создания и применения программных компонентов.

# 3 Проектирование задачи

## 3.1 Требования к приложению

Данное приложение имеет ограничение на число человек, не более 20, и подключение к Интернету.

Разрабатываемое приложение должно иметь понятный и удобный в использовании интерфейс, чтобы взаимодействие между программой и пользователем было максимально упрощено. Для того чтобы интерфейс был понятен пользователю, необходимо оформить диалог приложения с пользователем, обладающий кнопочным меню.

Функциональные требования представлены на диаграмме SADT, которая отображает вначале всю систему в виде простейшей компоненты – одного блока и дуг, изображающих интерфейсы с функциями вне системы. Единственный блок представляет всю систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг – они также представляют полный набор внешних интерфейсов системы в целом. Диаграмма нулевого уровня представлена на рисунках 3.1 и 3.2.

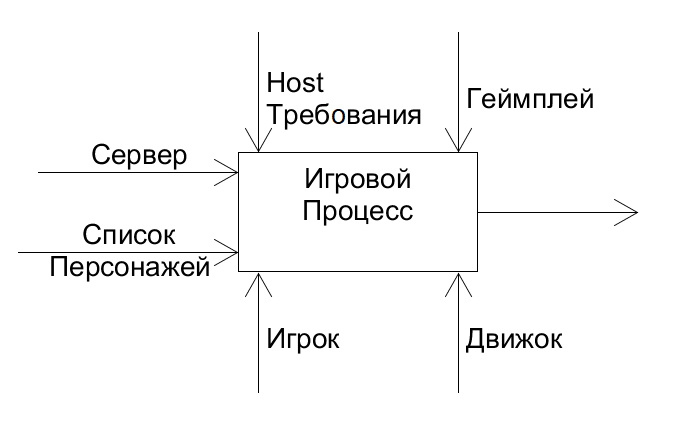


Рисунок 3.1 – Функциональная SADT диаграмма нулевого уровня

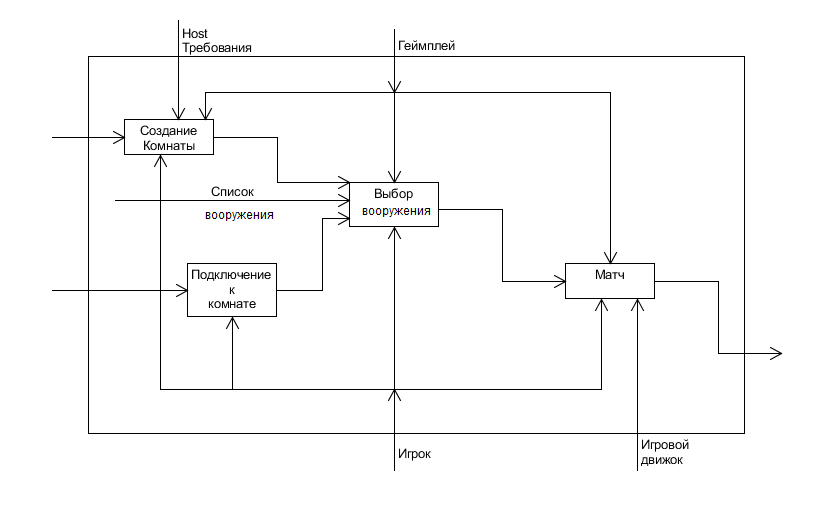


Рисунок 3.2 – Развёрнуая функциональная SADT диаграмма нулевого уровня

## 3.2 Концептуальный прототип

Концептуальный прототип представляет собой описание внешнего пользовательского интерфейса – систему меню и форм.

Все действия можно будет выполнить с помощью главной формы, которая будет появляться после логотипа игрового движка Unity. Главная форма будет иметь кнопочное меню программы позволяющее создать или подключится к уже созданному серверу, изменить имя пользователя. Используя меню, пользователь должен иметь возможность завершить работу всего приложения. Главное меню программы будет представлено формой Main на которой находятся основные элементы управления программой. С помощью кнопок на главной форме можно начать игру и выйти из игры.

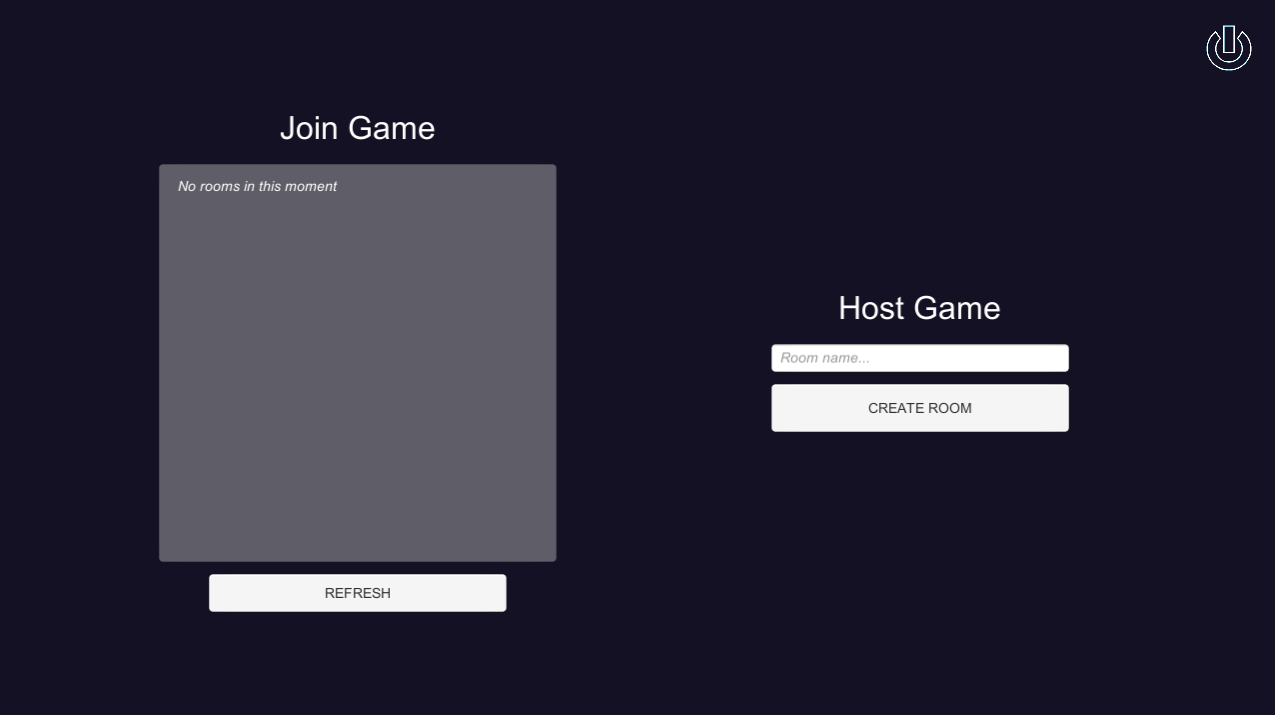


Рисунок 3.3 – Главное меню

Непосредственно во время игры в центре экрана будет перекрестие прицела – место куда летит пуля при выстреле. В нижней части экрана будут находится индикаторы количества жизней, количество патронов. В верхней части экрана время игры и количество очков до победы. При нажатии кнопки Esc открывается меню «Паузы» на которой будет возможность выхода из игры.

## 3.3 Организация данных

В программе данные организованы посредством классов.

GameManager.cs – отвечает за отслеживание персонажей, а также переключение на камеру сцены.

public void SetSceneCameraActive(bool \_isActive) – включение, выключение камеры сцены

public static void RegisterPlayer(string \_netID, Player \_player) – регистрация персонажа для отслеживания

public static void UnRegisterPlayer(string \_playerID) – удаление персонажа из отслеживания

public static bool IsRegistered(string \_playerID) – проверка, существует ли персонаж с таким именем

public static Player GetPlayer(string \_playerID) – получить персонажа

public static Player[] GetAllPlayers() – получить всех персонажей

public static int ReturnTopScore() – вернуть наилучший счет

public static bool IsMatchEnd() – возвращает true, если наилучшй счет равен победному

public static string ReturnTopScorePlayerName() – возвращает имя персонажа с наилучшим счетом

public static int ReturnPlayerScore(string \_playerID) – возвращает счет заданного персонажа

HostGame.cs – отвечает за создание комнаты, и пару общих функций

public void SetRoomName(string \_name) – задает имя комнаты

public void CreateRoom() – создание комнаты

public void Exit() – выход из игры

JoinGame.cs – отвечает за присоединение в комнату

public void RefreshRoomList() – начало процесса обновления комнат

public void OnMatchList(bool success, string extendedInfo, List<MatchInfoSnapshot> matches) – поиск комнат и их вывод на экран

private void ClearRoomList() – отчистка списка комнат

public void JoinRoom(MatchInfoSnapshot \_match) – вход в комнату

RoomListItem.cs – хранит в себе значения и методы, необходимые для JoinGame.cs

public void Setup(MatchInfoSnapshot \_match, JoinRoomDelegate \_joinRoomCallback) – задание значений для поля комнаты

public void JoinRoom() – вход в комнату

Killfeed.cs – отвечает за вывод киллфида на экране справа сверху

public void OnKill(string player, string source) – вызывается при убийстве, выводит киллфид на экран

KillfeedItem.cs – хранит в себе значения, требуемые для Killfeed.cs

public void Setup(string player, string source) – задает значения, которые будут выводиться на экран

MatchSettings.cs – хранит в себе настройки матча

PauseMenu.cs – отвечат за меню паузы

public void LeaveRoom() – выход из комнаты

Player.cs – отвечает за смерть и возрождение персонажа, конец игры

public void SetupPlayer() – выключает камеру сцены для локального персонажа и вызывает команду ниже

public void CmdBroadCastNewPlayerSetup() – вызывает Rpc метод ниже

private void RpcSetupPlayerOnAllClients() – выключает необходимые компоненты

public void RpcTakeDamage(int \_amount, string \_sourceID) – получение урона

private void Die(string \_sourceID) – смерть персонажа

private void CmdDie(string \_sourceID) – команда для смерти персонажа

private void RpcDie(string \_sorceID) – rpc метод смерти

private void CmdMatchEnd(bool boolean) – команда для окончания матча

private void RpcMatchEnd(bool boolean) – rpc метод окончания матча

private IEnumerator Respawn() – возрождение персонажа

public void SetDefaults() – выставления настроек по умолчанию для персонажа

public float GetHealthPct() – возвращает количество жизней от 0 до 1

public void EndMatchForPlayer() – окончание игры для персонажа

PlayerController.cs – отвечает за все поступающие действия из вне, своего рода API

private void Update() – принимает все входящие нажатия клавиш и в соответствии с этим выполняет определенные команды

private void SetJointSettings(float \_jointSpring) – установка уровня полета персонажа

public float GetEndurancePct() – возвращает количество топлива от 0 до 1

PlayerMovement.cs – примениение команды от PlayerController.cs

public void Move(Vector3 \_velocity) – задает вектор движения

public void Rotate(Vector3 \_rotation) – задает угол по Х

public void RotateCamera(float \_cameraRotationX) – задает угол по Y

public void ApplyThruster(Vector3 \_thrusterForce) – задает вектор полета

private void PerformMovement() – применение движения

private void PerformRotation() – применение поворота

ScoreBoard.cs – отвечает за вывод счета при нажатии клавиши TAB

private void OnEnable() – при включении выводится список персонажей и их счет

private void OnDisable() – отчистка при отключении

PlayerScoreboardItem.cs – хранит в себе значения, требуемые для ScoreBoard.cs

public void Setup(string username, string kills, string deaths) – задает значения, которые будут выводится на экран

PlayerShoot.cs – отвечает за выстрелы в игре

private void CmdOnShoot() – команда, вызываемая при выстреле

private void RpcDoShootEffect() – эффект выстрела

private void CmdOnHit(Vector3 \_pos, Vector3 \_normal) – вызывается, когда персонаж попал во что-то

private void RpcDoHitEffect(Vector3 \_pos, Vector3 \_normal) – эффект на поверхности попадания

private void Shoot() – сделать выстрел

private void CmdPlayerShot(string \_playerID, int \_damage, string \_sourceID) – команда выстрела по игроку

PlayerWeapon.cs – хранит характеристики оружия

WeaponGraphics.cs – хранит объекты, необходимые для отображения графических эффектов оружия

WeaponManager.cs – хранит все оружие, а также отвечает за его перезарядку

public PlayerWeapon GetCurrentWeapon() – получение текущего оружия

public WeaponGraphics GetCurrentGraphics() – получение текущего объекта графики

private void EquipWeapon(PlayerWeapon \_weapon) – установка оружия

public void Reload() – запуск коротины перезарядки

private IEnumerator Reload\_Coroutine() - перезарядка

private void CmdOnReload() – команда, вызываемая при перезарядке

private void RpcOnReload() – rpc метод, вызываемый при перезарядке

PlayerUI.cs – отвечает за интерфейс

public void SetPlayer(Player \_player) – выбор персонажа, для которого будет отображаться UI

private void TogglePauseMenu() – включение/выключение меню паузы

private void SetEnduranceAmount(float \_amount) – установка количества топлива

private void SetHealthAmount(float \_amount) – установка количества жизней

private void SetAmmoAmount(int \_amount, int \_maxAmount) – установка количества патронов

private void SetCrosshairColor() – установка цвета прицела

private void UpdateScore() – установка счета

public void UIEnd() – меню при окончании игры

Utility.cs – хранит полезные методы для программы

public static void SetLayerRecursively(GameObject \_obj, int \_newLayer) – устанавливает слои для объекта и его наследников

## 3.4 Функции и элементы управления

На основании диаграммы вариантов использования в программном приложении «FPS» были реализованы функции стрельбы и перемещения, создание и подключение к серверу.

Функция стрельбы активируется при нажатии левой кнопки мыши, персонаж выстрелит в перекрестие прицела. Для перезарядки необходимо нажать клавишу «К»(«R»). Во время перезарядки оружия главный герой не может стрелять.

Функция передвижения является основной функцией проекта. Персонаж может: идти, летать, бежать. Для передвижения используются стандартные для всех игр жанра «шутер» кнопки: вперед «Ц»(«W»), назад «Ы»(«S»), вправо «В»(«D»), влево «Ф»(«A»). Кнопка «Shift» отвечает за ускорение. При нажатии клавиши «Пробел»(«Space») персонаж сможет летать.

Функция создания сервера является основой для игры с друзьями. Сервер может быть создан только при подключенном интернет соединении. Ограничение сервера – 16 человек.

Функция подключения к серверу осуществляется при наличии уже созданной комнаты. Для подключения к комнате существует список комнат, в котором вы можете выбрать сервер и увидеть краткую информацию о нем (название сервера, количество человек на сервере).

# 4 Описание программного средства

## 4.1 Общие сведения

«FPS» – компьютерная игра, основанная на движке Unity3D. Является типичным представителем жанра «шутер». Весит данная игра – 40 Мбайт.

## 4.2 Функциональное назначение

Данная программа предназначена для хорошего времяпровождения. В программном средстве «FPS» реализованы функции, которые были определены на этапе объектно-ориентированного проектирования и анализа. Разрабатываемое программное приложение позволяет развить реакцию, мелку моторику рук, стратегическое мышление.

## 4.3 Входные и выходные данные

Входные и выходные данные как таковые отсутствуют.

# 5 Методика испытаний

## 5.1 Технические требования

Минимальные системные требования к приложению:

* 3 Гбайт свободного места на винчестере;
* 300 Mбайт свободной оперативной памяти;
* процессор Intel(R) Core(TM) i3-2140 2 ГГц или совместимый аналог;
* видеокарта с объемом памяти не менее 512 Мбайт;
* операционная система семейства Microsoft Windows;
* клавиатура, мышь, монитор;
* подключение к Интернету.

## 5.2 Функциональное тестирование

В процессе написания программного средства необходимо производить тестирование на правильность работы приложения. Одной из основных задач тестирования является устранение ошибок, происходящих при вводе данных.

Функциональное тестирование – это тестирование функций приложения на соответствие требованиям. Оценка производится в соответствии с ожидаемыми и полученными результатами (на основании функциональной спецификации), при условии, что функции отрабатывали на различных значениях.

Тестирование программы будет производиться последовательно, переходя из одной части программы в другую. Во время теста будут проверяться все действия с программой, навигация пунктам меню, которые может произвести пользователь. После чего все собранные и найденные ошибки будут исправлены.

В результате тестирования игры были обнаружены следующие ошибки:

В полете, при нажатии паузы, персонаж останавливался. Решение: был переработан скрипт управления персонажа.

Оружие могло войти в стену при достаточном приближении. Добавление отдельной камеры, которая видит только оружие и установление первого места в очереди отрисовки объектов.

# 6 Применение

## 6.1 Назначение программы

Данная программа предназначена для хорошего времяпрепровождения людей всех возрастных категорий.

Разрабатываемое программное приложение позволяет развить реакцию, мелкую моторику рук, стратегическое мышление.

## 6.2 Условия применения

Необходимо скопировать проект «FPS». Данный проект должен содержать следующие папки:

* Assets;
* Build;
* Library;
* obj
* Packages
* ProjectSettings;
* UnityPackageManager.

Для запуска игры должен быть «Build.exe».

Необходимо отметить, что данное программное средство не обладает средствами, ограничивающими доступ к нему, а также средствами, препятствующими свободному распространения программы. Для работы с приложением не требуется установка дополнительных компонентов и специальных технических и программных ресурсов.

# Заключение

В рамках выполнения проекта был совершен анализ предметной области, была спроектирована модель, были поставлены требования к аппаратным и операционным ресурсам, были выдвинуты функциональные требования, было приведено функционально тестирование.

В программе были реализованы такие задачи, как:

* стрельба;
* передвижения;
* создание сервера;
* подключение к серверу;
* синхронизация с сервером.

В процессе проектирования программное приложение прошло функциональное и полное тестирование, которые доказали корректность выполнения всех определенных функций, отсутствие сбоев и неадекватных реакций со стороны программы. В процессе разработки программы использовался в большом объеме тематический материал и исследована предметная область разработки аналогичных программных средств. В приложении реализован далеко не весь желаемый разработчиком функционал, поэтому оно будет поддерживаться и улучшаться.

# Список информационных источников

1. Багласова, Т.Г. Методические указания по выполнению дипломного проекта для учащихся по специальности 2-40 01 01 «Программное обеспечение технологий» / Т.Г. Багласова. – Минск : КБП, 2017. – 30 c.
2. Багласова, Т.Г. Методические указания по оформлению курсовых и дипломных проектов / Т.Г. Багласова, К.О. Якимович. – Минск : КБП, 2013. – 29 c.
3. Вендров, А.М. Проектирование программного обеспечения экономических информационных систем: Учебник / А.М. Вендров. – 2-е изд., перераб. и доп.– М. : Финансы и статистика, 2006. — 544 с.
4. Общие требования к тестовым документам : ГОСТ 2.105-95. – Введ. 01.01.1996. – Минск : Межгос. совет по стандартизации, метрологии и сертификации, 1995. – 84 с.
5. Описание программы. Требования к содержанию, оформлению и контролю качества : ГОСТ 19.402-2000. – Введ. 01.09.2001. – Минск : Межгос. совет по стандартизации, метрологии и сертификации, 2000. – 14 с.
6. Программа и методика испытаний. Требования к содержанию, оформлению и контролю качества : ГОСТ 19.301-2000. – Введ. 01.09.2001. – Минск : Межгос. совет по стандартизации, метрологии и сертификации, 2000. – 14 с.
7. Текст программы. Требования к содержанию, оформлению и контролю качества : ГОСТ 19.401-2000. – Введ. 01.09.2001. – Минск : Межгос. совет по стандартизации, метрологии и сертификации, 2000. – 16 с.
8. Операционная система Windows 10 [Электронный ресурс] – Режим доступа: http:// docs.microsoft.com/ru-ru/dotnet/windows-10/tour-of-windows-10/
9. Пакет офисных программ и текстовых редакторов Unity3D [Электронный ресурс]. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/index.html. – Дата доступа: 25.05.2018.
10. Среда разработки Microsoft Visual Studio 2017[Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/ide/visual-studio-ide. – Дата доступа: 15.06.2018.
11. Язык программирования C#[Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/. – Дата доступа: 14.04.2018.

# Приложение А

(обязательное)

Текст программы

public class GameManager : NetworkBehaviour{

public static GameManager instance;

public MatchSettings matchSettings;

[SerializeField]

private GameObject sceneCamera;

public delegate void OnPlayerKilledCallback(string player, string source);

public OnPlayerKilledCallback onPlayerKilledCallback;

private void Awake()

{

if (instance != null)

{

Debug.LogError("more than one gamemanager");

}

else

{

instance = this;

}

}

public void SetSceneCameraActive(bool \_isActive)

{

if (sceneCamera == null)

return;

sceneCamera.SetActive(\_isActive);

}

#region Player Tracking

private static Dictionary<string, Player> players = new Dictionary<string, Player>();

public static void RegisterPlayer(string \_netID, Player \_player)

{

string \_playerID = "Player " + \_netID;

players.Add(\_playerID, \_player);

\_player.transform.name = \_playerID;

}

public static void UnRegisterPlayer(string \_playerID)

{

players.Remove(\_playerID);

}

public static bool IsRegistered(string \_playerID)

{

Player[] \_players = GetAllPlayers();

for(int i=0;i<\_players.Length;i++)

{

if (\_players[i].name == \_playerID)

return true;

}

return false;

}

public static Player GetPlayer(string \_playerID)

{

return players[\_playerID];

}

public static Player[] GetAllPlayers()

{

return players.Values.ToArray();

}

public static int ReturnTopScore()

{

Player[] players = GetAllPlayers();

int score = 0;

foreach (Player player in players)

{

if (score < player.kills)

score = player.kills;

}

return score;

}

public static bool IsMatchEnd()

{

if (ReturnTopScore() >= GameManager.instance.matchSettings.scoreToWin)

return true;

else

return false;

}

public static string ReturnTopScorePlayerName()

{

Player[] players = GetAllPlayers();

int score = 0;

string name = "";

foreach (Player player in players)

{

if (score < player.kills)

{

score = player.kills;

name = player.name;

}

}

return name;

}

public static int ReturnPlayerScore(string \_playerID)

{

return players[\_playerID].kills;

}

#endregion

}

public class HostGame : MonoBehaviour

{

[SerializeField]

private uint roomSize = 16;

private string roomName;

private NetworkManager networkManager;

private void Start()

{

networkManager = NetworkManager.singleton;

if(networkManager.matchMaker == null)

{

networkManager.StartMatchMaker();

}

}

public void SetRoomName(string \_name)

{

roomName = \_name;

}

public void CreateRoom()

{

if (roomName != "" && roomName != null)

{

networkManager.matchMaker.CreateMatch(roomName, roomSize, true, "", "", "", 0, 0, networkManager.OnMatchCreate);

}

}

public void Exit()

{

Application.Quit();

}

}

public class JoinGame : MonoBehaviour {

List<GameObject> roomList = new List<GameObject>();

[SerializeField]

private Text status;

[SerializeField]

private GameObject roomListItemPrefab;

[SerializeField]

private Transform roomListParent;

private NetworkManager networkManager;

private void Start()

{

networkManager = NetworkManager.singleton;

if (networkManager.matchMaker == null)

networkManager.StartMatchMaker();

RefreshRoomList();

}

public void RefreshRoomList()

{

networkManager.matchMaker.ListMatches(0, 20, "", true, 0, 0, OnMatchList);

status.text = "Loading...";

}

public void OnMatchList(bool success, string extendedInfo, List<MatchInfoSnapshot> matches)

{

status.text = "";

if (matches == null)

{

status.text = "Couldn't get room list.";

return;

}

ClearRoomList();

foreach(MatchInfoSnapshot match in matches)

{

GameObject \_roomListItemGO = Instantiate(roomListItemPrefab);

\_roomListItemGO.transform.SetParent(roomListParent);

RoomListItem \_roomListItem = \_roomListItemGO.GetComponent<RoomListItem>();

if(\_roomListItem!=null)

{

\_roomListItem.Setup(match, JoinRoom);

}

roomList.Add(\_roomListItemGO);

}

if(roomList.Count == 0)

{

status.text = "No rooms in this moment";

}

}

private void ClearRoomList()

{

for(int i=0;i<roomList.Count;i++)

{

Destroy(roomList[i]);

}

roomList.Clear();

}

public void JoinRoom(MatchInfoSnapshot \_match)

{

networkManager.matchMaker.JoinMatch(\_match.networkId, "", "", "", 0, 0, networkManager.OnMatchJoined);

ClearRoomList();

status.text = "Joining...";

}

}

class Killfeed : MonoBehaviour

{

[SerializeField]

GameObject killfeedItemPrefab;

private void Start()

{

GameManager.instance.onPlayerKilledCallback += OnKill;

}

public void OnKill(string player, string source)

{

GameObject go = (GameObject)Instantiate(killfeedItemPrefab, this.transform);

go.GetComponent<KillfeedItem>().Setup(player, source);

go.transform.SetAsFirstSibling();

Destroy(go, 4f);

}

}

class KillfeedItem : MonoBehaviour

{

[SerializeField]

Text text;

public void Setup(string player, string source)

{

text.text = source + " killed " + player;

}

}

public class MatchSettings

{

public float respawnTime = 3f;

public int scoreToWin = 50;

public float timeRemaining = 600f;

}

public class PauseMenu : MonoBehaviour {

public static bool IsOn = false;

private NetworkManager networkManager;

private void Start()

{

networkManager = NetworkManager.singleton;

}

public void LeaveRoom()

{

MatchInfo matchInfo = networkManager.matchInfo;

networkManager.matchMaker.DropConnection(matchInfo.networkId, matchInfo.nodeId, 0, networkManager.OnDropConnection);

networkManager.StopHost();

}

}

[RequireComponent(typeof(PlayerSetup))]

public class Player : NetworkBehaviour {

[SyncVar]

private bool \_isDead = false;

public bool isDead

{

get { return \_isDead; }

protected set { \_isDead = value; }

}

[SerializeField]

private int maxHealth = 100;

[SyncVar]

private int currentHealth;

public int kills;

public int deaths;

[SerializeField]

private Behaviour[] disableOnDeath;

[SerializeField]

private GameObject[] disableGameObjectsOnDeath;

public bool[] wasEnabled;

[SerializeField]

private GameObject deathEffect;

[SerializeField]

private GameObject spawnEffect;

[SerializeField]

private GameObject Graphics;

private bool firstSetup = true;

private bool matchEnd = false;

private void Update()

{

if (isLocalPlayer)

{

CmdMatchEnd(GameManager.IsMatchEnd());

if (matchEnd && !isDead)

{

EndMatchForPlayer();

}

if(matchEnd)

{

GetComponent<PlayerSetup>().playerUIInstance.SetActive(true);

GetComponent<PlayerSetup>().playerUIInstance.GetComponent<PlayerUI>().UIEnd();

if (Cursor.lockState != CursorLockMode.None)

{

Cursor.lockState = CursorLockMode.None;

Cursor.visible = true;

}

}

}

}

public void SetupPlayer()

{

if (isLocalPlayer)

{

GameManager.instance.SetSceneCameraActive(false);

GetComponent<PlayerSetup>().playerUIInstance.SetActive(true);

CmdBroadCastNewPlayerSetup();

}

}

[Command]

public void CmdBroadCastNewPlayerSetup()

{

RpcSetupPlayerOnAllClients();

}

[ClientRpc]

private void RpcSetupPlayerOnAllClients()

{

if (firstSetup)

{

wasEnabled = new bool[disableOnDeath.Length];

for (int i = 0; i < wasEnabled.Length; i++)

{

wasEnabled[i] = disableOnDeath[i].enabled;

}

firstSetup = false;

}

SetDefaults();

}

[ClientRpc]

public void RpcTakeDamage(int \_amount, string \_sourceID)

{

if (isDead)

return;

currentHealth -= \_amount;

if (currentHealth <= 0)

{

Die(\_sourceID);

}

}

private void Die(string \_sourceID)

{

isDead = true;

deaths++;

GameManager.GetPlayer(\_sourceID).kills++;

GameManager.instance.onPlayerKilledCallback(gameObject.transform.name, GameManager.GetPlayer(\_sourceID).transform.name);

for (int i = 0; i < disableOnDeath.Length; i++)

{

disableOnDeath[i].enabled= false;

}

for (int i = 0; i < disableGameObjectsOnDeath.Length; i++)

{

disableGameObjectsOnDeath[i].SetActive(false);

}

Collider \_col = GetComponent<Collider>();

if (\_col != null)

\_col.enabled = false;

gameObject.GetComponent<Rigidbody>().useGravity = false;

gameObject.GetComponent<Animator>().enabled = false;

if (!matchEnd)

{

GameObject \_gfxIns = (GameObject)Instantiate(deathEffect, transform.position, Quaternion.identity);

Destroy(\_gfxIns, 2f);

}

if (isLocalPlayer)

{

GameManager.instance.SetSceneCameraActive(true);

GetComponent<PlayerSetup>().playerUIInstance.SetActive(false);

}

Debug.Log(transform.name + "is dead!");

Transform \_spawnPoint = NetworkManager.singleton.GetStartPosition();

transform.position = \_spawnPoint.position;

transform.rotation = \_spawnPoint.rotation;

GetComponent<ConfigurableJoint>().targetPosition = new Vector3(0, -transform.position.y + 1, 0);

if(!matchEnd)

StartCoroutine(Respawn());

}

[Command]

private void CmdDie(string \_sourceID)

{

RpcDie(\_sourceID);

}

[ClientRpc]

private void RpcDie(string \_sorceID)

{

Die(\_sorceID);

}

[Command]

private void CmdMatchEnd(bool boolean)

{

RpcMatchEnd(boolean);

}

[ClientRpc]

private void RpcMatchEnd(bool boolean)

{

matchEnd = boolean;

}

private IEnumerator Respawn()

{

yield return new WaitForSeconds(GameManager.instance.matchSettings.respawnTime);

SetupPlayer();

}

public void SetDefaults()

{

isDead = false;

currentHealth = maxHealth;

for (int i = 0; i < disableOnDeath.Length; i++)

{

disableOnDeath[i].enabled = wasEnabled[i];

}

for (int i = 0; i < disableGameObjectsOnDeath.Length; i++)

{

disableGameObjectsOnDeath[i].SetActive(true);

}

Collider \_col = GetComponent<Collider>();

if (\_col != null)

\_col.enabled = true;

gameObject.GetComponent<Rigidbody>().useGravity = true;

gameObject.GetComponent<Animator>().enabled = true;

if (!matchEnd)

{

GameObject \_gfxIns = (GameObject)Instantiate(spawnEffect, transform.position, Quaternion.identity);

Destroy(\_gfxIns, 2f);

}

}

public float GetHealthPct()

{

return (float)currentHealth / maxHealth;

}

public void EndMatchForPlayer()

{

CmdDie(transform.name);

GetComponent<PlayerSetup>().playerUIInstance.SetActive(true);

GetComponent<PlayerSetup>().playerUIInstance.GetComponent<PlayerUI>().UIEnd();

if (Cursor.lockState != CursorLockMode.None)

{

Cursor.lockState = CursorLockMode.None;

Cursor.visible = true;

}

}

}

public class PlayerController : MonoBehaviour

{

[SerializeField]

private float speed = 5f;

[SerializeField]

private float sensetivity = 3f;

[SerializeField]

private float enduranceBurnSpeed = 2f;

[SerializeField]

private float enduranceRegenSpeed = 0.5f;

private float enduranceMaxAmount = 10f;

private float enduranceAmount = 10f;

private float sprintMultiplier = 2f;

[SerializeField]

private float thrusterForce = 1000f;

[Header("Spring settings:")]

[SerializeField]

private JointDriveMode jointMode = JointDriveMode.Position;

[SerializeField]

private float jointSpring = 20f;

[SerializeField]

private float jointMaxForce = 40f;

private PlayerMovement movement;

private ConfigurableJoint joint;

private Animator animator;

private void Start()

{

movement = GetComponent<PlayerMovement>();

joint = GetComponent<ConfigurableJoint>();

animator = GetComponent<Animator>();

SetJointSettings(jointSpring);

}

private void Update()

{

if (PauseMenu.IsOn)

{

if (Cursor.lockState != CursorLockMode.None)

{

Cursor.lockState = CursorLockMode.None;

Cursor.visible = true;

}

movement.Move(Vector3.zero);

movement.Rotate(Vector3.zero);

movement.RotateCamera(0f);

animator.SetFloat("Forward", 0, 0.1f, Time.deltaTime);

animator.SetFloat("Right", 0, 0.1f, Time.deltaTime);

enduranceAmount += enduranceRegenSpeed \* Time.deltaTime;

enduranceAmount = Mathf.Clamp(enduranceAmount, 0f, enduranceMaxAmount);

return;

}

if (Cursor.lockState != CursorLockMode.Locked)

{

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible = false;

}

RaycastHit \_hit;

if (Physics.Raycast(transform.position, Vector3.down, out \_hit, 100f))

{

joint.targetPosition = new Vector3(0, -\_hit.point.y, 0);

}

else

{

joint.targetPosition = new Vector3(0, 1, 0);

}

//Вычисление направления движения

float \_xMov = Input.GetAxis("Horizontal");

float \_zMov = Input.GetAxis("Vertical");

Vector3 \_movHorizontal = transform.right \* \_xMov;

Vector3 \_movVertical = transform.forward \* \_zMov;

//Расчет скорости движения с учетом спринта

float \_sprintMultiplier = 1f;

if (Input.GetKey(KeyCode.LeftShift) && \_zMov > 0 && \_xMov == 0)

{

if (enduranceAmount > 0f)

{

enduranceAmount -= enduranceBurnSpeed \* Time.deltaTime;

\_sprintMultiplier = sprintMultiplier;

}

}

Vector3 \_thrusterForce = Vector3.zero;

if (Input.GetKey(KeyCode.Space) && enduranceAmount >= 0.1f)

{

enduranceAmount -= enduranceBurnSpeed \* Time.deltaTime;

\_thrusterForce = Vector3.up \* thrusterForce;

SetJointSettings(0f);

}

else

{

SetJointSettings(jointSpring);

}

if (!Input.GetKey(KeyCode.LeftShift) && !Input.GetKey(KeyCode.Space))

{

enduranceAmount += enduranceRegenSpeed \* Time.deltaTime;

}

movement.ApplyThruster(\_thrusterForce);

enduranceAmount = Mathf.Clamp(enduranceAmount, 0f, enduranceMaxAmount);

//Конечный вектор движения

Vector3 \_velocity = (\_movHorizontal + \_movVertical) \* speed \* \_sprintMultiplier;

//Анимирование движения

animator.SetFloat("Forward", \_zMov, 0.1f, Time.deltaTime);

animator.SetFloat("Right", \_xMov, 0.1f, Time.deltaTime);

//Применение изменения движения

movement.Move(\_velocity);

//Вычисление поворота по горизонтали

float \_yRot = Input.GetAxisRaw("Mouse X");

Vector3 \_rotation = new Vector3(0f, \_yRot, 0f) \* sensetivity;

//Применение изменения поворотов по горизонтали

movement.Rotate(\_rotation);

//Вычисление поворотов для камеры по вертикали

float \_xRot = Input.GetAxisRaw("Mouse Y");

float \_cameraRotationX = \_xRot \* sensetivity;

//Применение изменения поворотов для камеры по вертикали

movement.RotateCamera(\_cameraRotationX);

}

private void SetJointSettings(float \_jointSpring)

{

joint.yDrive = new JointDrive

{

mode = jointMode,

positionSpring = \_jointSpring,

maximumForce = jointMaxForce

};

}

public float GetEndurancePct()

{

return enduranceAmount / enduranceMaxAmount;

}

}

[RequireComponent(typeof(Rigidbody))]

public class PlayerMovement : MonoBehaviour

{

[SerializeField]

private Camera cam;

private Vector3 velocity = Vector3.zero;

private Vector3 rotation = Vector3.zero;

private Vector3 thrusterForce = Vector3.zero;

private float cameraRotationX = 0f;

private float currentCameraRotationX = 0f;

[SerializeField]

private float cameraRotationLimit = 85f;

private Rigidbody rb;

private void Start()

{

rb = GetComponent<Rigidbody>();

}

//Получает вектор движения

public void Move(Vector3 \_velocity)

{

velocity = \_velocity;

}

//Получает вектор поворота

public void Rotate(Vector3 \_rotation)

{

rotation = \_rotation;

}

//Получает вектор поворота для камеры

public void RotateCamera(float \_cameraRotationX)

{

cameraRotationX = \_cameraRotationX;

}

public void ApplyThruster(Vector3 \_thrusterForce)

{

thrusterForce = \_thrusterForce;

}

//Вызывается на каждой физической итерации

private void FixedUpdate()

{

PerformMovement();

PerformRotation();

}

//Выполняет движения на основе переменной velocity

private void PerformMovement()

{

if (velocity != Vector3.zero)

{

rb.MovePosition(rb.position + velocity \* Time.fixedDeltaTime);

}

if(thrusterForce!=Vector3.zero)

{

rb.AddForce(thrusterForce \* Time.fixedDeltaTime, ForceMode.Acceleration);

}

}

//Выполнение поворота

private void PerformRotation()

{

rb.MoveRotation(rb.rotation \* Quaternion.Euler(rotation));

if (cam != null)

{

currentCameraRotationX -= cameraRotationX;

currentCameraRotationX = Mathf.Clamp(currentCameraRotationX, -cameraRotationLimit, cameraRotationLimit);

cam.transform.localEulerAngles = new Vector3(currentCameraRotationX, 0f, 0f);

}

}

}

[RequireComponent(typeof(Player))]

[RequireComponent(typeof(PlayerController))]

public class PlayerSetup : NetworkBehaviour

{

[SerializeField]

Behaviour[] componentsToDisable;

[SerializeField]

string remoteLayerName = "RemotePlayer";

[SerializeField]

string dontDrawLayerName = "DontDraw";

[SerializeField]

GameObject playerGraphics;

[SerializeField]

GameObject playerUIPrefab;

public GameObject playerUIInstance;

private void Start()

{

//выключаем компоненты на других игроках,

//которые мы можем использовать

if (!isLocalPlayer)

{

DisableComponents();

AssignRemoteLayer();

}

else

{

//выключаем графику для локального персонажа

SetLayerRecursively(playerGraphics,LayerMask.NameToLayer(dontDrawLayerName));

//Создаем интерфейс

playerUIInstance = Instantiate(playerUIPrefab);

playerUIInstance.name = playerUIPrefab.name;

//Настраиваем интерфейс

PlayerUI ui = playerUIInstance.GetComponent<PlayerUI>();

if (ui != null)

ui.SetPlayer(GetComponent<Player>());

GetComponent<Player>().SetupPlayer();

}

}

public void SetLayerRecursively(GameObject obj, int newLayer)

{

obj.layer = newLayer;

foreach(Transform child in obj.transform)

{

SetLayerRecursively(child.gameObject, newLayer);

}

}

public override void OnStartClient()

{

base.OnStartClient();

string \_netID = GetComponent<NetworkIdentity>().netId.ToString();

Player \_player = GetComponent<Player>();

GameManager.RegisterPlayer(\_netID, \_player);

}

private void AssignRemoteLayer()

{

SetLayerRecursively(gameObject, LayerMask.NameToLayer(remoteLayerName));

}

private void DisableComponents()

{

for (int i = 0; i < componentsToDisable.Length; i++)

{

componentsToDisable[i].enabled = false;

}

}

//Когда персонаж не в игре

private void OnDisable()

{

Destroy(playerUIInstance);

//включаем камеру сцены

if(isLocalPlayer)

GameManager.instance.SetSceneCameraActive(true);

GameManager.UnRegisterPlayer(transform.name);

}

}

[RequireComponent(typeof(WeaponManager))]

public class PlayerShoot : NetworkBehaviour

{

[SerializeField]

private Camera cam;

[SerializeField]

private LayerMask mask;

private PlayerWeapon currentWeapon;

private WeaponManager weaponManager;

private PlayerMovement playerMovement;

private void Start()

{

if (cam == null)

{

Debug.LogError("Playershoot: no cam");

}

weaponManager = GetComponent<WeaponManager>();

playerMovement = GetComponent<PlayerMovement>();

}

private void Update()

{

currentWeapon = weaponManager.GetCurrentWeapon();

if (PauseMenu.IsOn)

return;

if(Input.GetKeyDown(KeyCode.R))

{

if (currentWeapon.bullets < currentWeapon.maxBullets)

{

weaponManager.Reload();

return;

}

}

if (currentWeapon.fireRate == 0f)

{

if (Input.GetButtonDown("Fire1"))

{

Shoot();

}

}

else

{

if (Input.GetButtonDown("Fire1"))

{

InvokeRepeating("Shoot", 0f, 1f/currentWeapon.fireRate);

}

else if(Input.GetButtonUp("Fire1"))

{

CancelInvoke("Shoot");

}

}

if(Input.GetKeyDown(KeyCode.Alpha1))

{

weaponManager.EquipPrimaryWeapon();

}

if(Input.GetKeyDown(KeyCode.Alpha2))

{

weaponManager.EquipSecondaryWeapon();

}

}

//Вызывается на сервере, когда игрок стреляет

[Command]

private void CmdOnShoot()

{

RpcDoShootEffect();

}

//Вызывается на всех клиентах когда нужно сделать

//эффект выстрела

[ClientRpc]

private void RpcDoShootEffect()

{

weaponManager.GetCurrentGraphics().muzzleFlash.Play();

weaponManager.GetCurrentGraphics().gunAnimator.Play("Shoot");

}

//Вызывается на сервере когда игрок поподает во что-то

//передаются координаты попадания, и нормаль поверхности

[Command]

private void CmdOnHit(Vector3 \_pos, Vector3 \_normal)

{

RpcDoHitEffect(\_pos, \_normal);

}

//Вызывается на всех клиентах, чтобы запустить эффект попадания

[ClientRpc]

private void RpcDoHitEffect(Vector3 \_pos, Vector3 \_normal)

{

GameObject \_hitEffect = (GameObject)Instantiate(weaponManager.GetCurrentGraphics().hitEffectPrefab, \_pos, Quaternion.LookRotation(\_normal));

Destroy(\_hitEffect, 1f);

}

[Client]

private void Shoot()

{

if (!isLocalPlayer || weaponManager.isReloading)

return;

if(currentWeapon.bullets<=0)

{

weaponManager.Reload();

return;

}

currentWeapon.bullets--;

//Вызываем метод OnShoot на сервере

CmdOnShoot();

RaycastHit \_hit;

if (Physics.Raycast(cam.transform.position, cam.transform.forward, out \_hit, currentWeapon.range, mask))

{

if (\_hit.collider.tag == "Player")

{

CmdPlayerShot(\_hit.collider.name, currentWeapon.damage, transform.name);

}

//Мы попали во что-то, поэтому вызываем метод OnHit на сервере

CmdOnHit(\_hit.point, \_hit.normal);

}

if (currentWeapon.bullets <= 0)

{

weaponManager.Reload();

}

}

[Command]

private void CmdPlayerShot(string \_playerID, int \_damage, string \_sourceID)

{

Player \_player = GameManager.GetPlayer(\_playerID);

\_player.RpcTakeDamage(\_damage, \_sourceID);

}

}

public class PlayerUI : MonoBehaviour

{

[SerializeField]

private RectTransform enduranceFill;

[SerializeField]

private RectTransform healthBarFill;

[SerializeField]

private Text ammoText;

[SerializeField]

private GameObject pauseMenu;

[SerializeField]

private GameObject scoreboard;

[SerializeField]

private Text nameText;

private Camera playerCamera;

[SerializeField]

private LayerMask mask;

[SerializeField]

private Image crosshair;

[SerializeField]

private Text topScoreText;

[SerializeField]

private Text youScoreText;

[SerializeField]

private GameObject endPanel;

[SerializeField]

private Text winnerText;

[SerializeField]

private GameObject killfeed;

private Player player;

private PlayerController controller;

private WeaponManager weaponManager;

public void SetPlayer(Player \_player)

{

player = \_player;

controller = player.GetComponent<PlayerController>();

weaponManager = player.GetComponent<WeaponManager>();

playerCamera = player.GetComponentInChildren<Camera>();

}

private void Start()

{

PauseMenu.IsOn = false;

nameText.text = player.name;

}

private void Update()

{

SetEnduranceAmount(controller.GetEndurancePct());

SetHealthAmount(player.GetHealthPct());

SetAmmoAmount(weaponManager.GetCurrentWeapon().bullets, weaponManager.GetCurrentWeapon().maxBullets);

SetCrosshairColor();

UpdateScore();

if(Input.GetKeyDown(KeyCode.Escape))

{

TogglePauseMenu();

}

if(Input.GetKeyDown(KeyCode.Tab))

{

scoreboard.SetActive(true);

} else if(Input.GetKeyUp(KeyCode.Tab))

{

scoreboard.SetActive(false);

}

}

private void TogglePauseMenu()

{

pauseMenu.SetActive(!pauseMenu.activeSelf);

PauseMenu.IsOn = pauseMenu.activeSelf;

}

private void SetEnduranceAmount(float \_amount)

{

enduranceFill.localScale = new Vector3(\_amount, 1f, 1f);

}

private void SetHealthAmount(float \_amount)

{

healthBarFill.localScale = new Vector3(\_amount, 1f, 1f);

}

private void SetAmmoAmount(int \_amount, int \_maxAmount)

{

ammoText.text = \_amount + " / " + \_maxAmount;

}

private void SetCrosshairColor()

{

RaycastHit \_hit;

if (Physics.Raycast(playerCamera.transform.position, playerCamera.transform.forward, out \_hit, 1000, mask))

{

if(\_hit.collider.tag=="Player")

{

crosshair.color = Color.red;

}

else

{

crosshair.color = Color.white;

}

}

else

{

crosshair.color = Color.white;

}

}

private void UpdateScore()

{

topScoreText.text = GameManager.ReturnTopScore().ToString();

youScoreText.text = GameManager.ReturnPlayerScore(player.name).ToString();

}

public void UIEnd()

{

enduranceFill.parent.gameObject.SetActive(false);

healthBarFill.parent.gameObject.SetActive(false);

nameText.transform.parent.gameObject.SetActive(false);

ammoText.transform.parent.gameObject.SetActive(false);

crosshair.gameObject.SetActive(false);

youScoreText.transform.parent.gameObject.SetActive(false);

killfeed.SetActive(false);

endPanel.SetActive(true);

winnerText.text = GameManager.ReturnTopScorePlayerName();

}

}

[System.Serializable]

public class PlayerWeapon

{

public string name = "";

public int damage = 10;

public float range = 100f;

public float fireRate = 0f;

public int maxBullets = 30;

public int bullets;

public GameObject graphics;

public float reloadTime = 2f;

public PlayerWeapon()

{

bullets = maxBullets;

}

}

public class RoomListItem : MonoBehaviour {

public delegate void JoinRoomDelegate(MatchInfoSnapshot \_match);

private JoinRoomDelegate joinRoomCallback;

[SerializeField]

private Text roomNameText;

MatchInfoSnapshot match;

public void Setup(MatchInfoSnapshot \_match, JoinRoomDelegate \_joinRoomCallback)

{

match = \_match;

joinRoomCallback = \_joinRoomCallback;

roomNameText.text = match.name + " (" + match.currentSize + "/" + match.maxSize + ")";

}

public void JoinRoom()

{

joinRoomCallback.Invoke(match);

}

}

public class ScoreBoard : MonoBehaviour {

[SerializeField]

GameObject playerScoreboardItemPrefab;

[SerializeField]

Transform playerList;

private void OnEnable()

{

Player[] players = GameManager.GetAllPlayers();

foreach(Player player in players)

{

GameObject itemGO = (GameObject)Instantiate(playerScoreboardItemPrefab, playerList);

PlayerScoreboardItem item = itemGO.GetComponent<PlayerScoreboardItem>();

if (item != null)

item.Setup(player.name, player.kills.ToString(), player.deaths.ToString());

}

}

private void OnDisable()

{

foreach(Transform child in playerList)

{

Destroy(child.gameObject);

}

}

}

public class Utility{

public static void SetLayerRecursively(GameObject \_obj, int \_newLayer)

{

if (\_obj == null)

return;

\_obj.layer = \_newLayer;

foreach(Transform \_child in \_obj.transform)

{

if (\_child == null)

continue;

SetLayerRecursively(\_child.gameObject, \_newLayer);

}

}

}

public class WeaponGraphics : MonoBehaviour {

public ParticleSystem muzzleFlash;

public GameObject hitEffectPrefab;

public Animator gunAnimator;

}

public class WeaponManager : NetworkBehaviour

{

[SerializeField]

private Transform weaponHolder;

[SerializeField]

private PlayerWeapon primaryWeapon;

[SerializeField]

private PlayerWeapon secondaryWeapon;

private PlayerWeapon currentWeapon;

private WeaponGraphics currentGraphics;

GameObject \_weaponIns;

public bool isReloading = false;

private void Start()

{

EquipWeapon(primaryWeapon);

}

public PlayerWeapon GetCurrentWeapon()

{

return currentWeapon;

}

public WeaponGraphics GetCurrentGraphics()

{

return currentGraphics;

}

public void EquipPrimaryWeapon()

{

if (weaponHolder.GetChild(0).gameObject != null)

Destroy(weaponHolder.GetChild(0).gameObject);

EquipWeapon(primaryWeapon);

}

public void EquipSecondaryWeapon()

{

if (weaponHolder.GetChild(0).gameObject != null)

Destroy(weaponHolder.GetChild(0).gameObject);

EquipWeapon(secondaryWeapon);

}

private void EquipWeapon(PlayerWeapon \_weapon)

{

currentWeapon = \_weapon;

\_weaponIns = (GameObject)Instantiate(\_weapon.graphics, weaponHolder.position, weaponHolder.rotation);

\_weaponIns.transform.SetParent(weaponHolder);

currentGraphics = \_weaponIns.GetComponent<WeaponGraphics>();

if (isLocalPlayer)

Utility.SetLayerRecursively(\_weaponIns, LayerMask.NameToLayer("Weapon"));

}

public void Reload()

{

if (isReloading)

return;

StartCoroutine(Reload\_Coroutine());

}

private IEnumerator Reload\_Coroutine()

{

CmdOnReload();

yield return new WaitForSeconds(currentWeapon.reloadTime);

currentWeapon.bullets = currentWeapon.maxBullets;;

CmdOnReload();

}

[Command]

private void CmdOnReload()

{

RpcOnReload();

}

[ClientRpc]

private void RpcOnReload()

{

isReloading = !isReloading;

if (isReloading)

currentGraphics.gunAnimator.SetTrigger("IsReloading");

else

currentGraphics.gunAnimator.ResetTrigger("IsReloading");

}

}



Реценз.

Т. Контр.

Н. Контр.

ОП Т.592020.401 ГЧ

Утверд.

Разраб.

Сизов А.И.

Листов 5

№ докум.

Подпись

Дата

Изм.

Лист

Программная реализация игры «First Person Shooter»

Диаграмма вариантов использования

У

Масса

Лит.

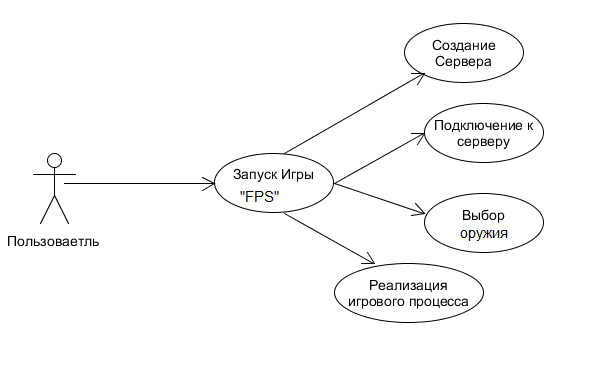
Масштаб

КБП

Взам.инв.№

Подп. и дата

Инв.№дубл.



Подп. и дата

Инв.№подл.

Лист 1

Провер.

Багласова Е.В



Реценз.

Т. Контр.

Н. Контр.

КП Т.592020.401 ГЧ

Утверд.

Разраб.

Сизов А.И.

Листов 5

Лист 2

№ докум.

Подпись

Дата

Изм.

Лист

Программная реализация игры «First Person Shooter»

Диаграмма компонентов

У

Масса

Лит.

Масштаб

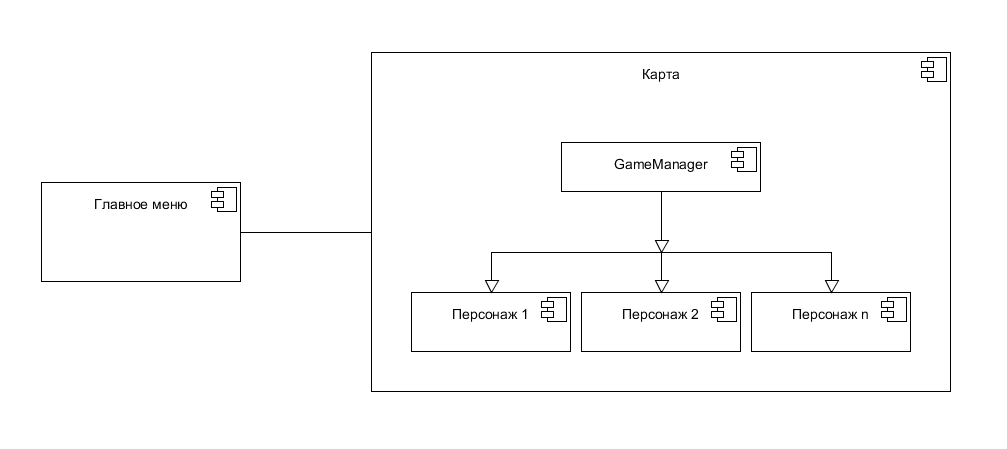
КБП

Взам.инв.№

Подп. и дата

Подп. и дата

Инв.№дубл.



Провер.

Багласова Е.В

Инв.№подл.



Реценз.

Т. Контр.

Н. Контр.

КП Т.592020.401 ГЧ

Утверд.

Разраб.

Сизов А.И.

Листов 5

№ докум.

Подпись

Дата

Изм.

Лист

Программная реализация игры «First Person Shooter»

Диаграмма деятельности

У

Масса

Лит.

Масштаб

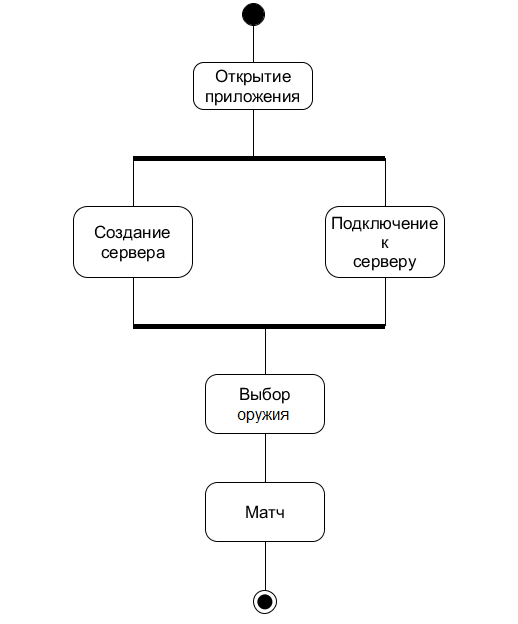
КБП

Взам.инв.№

Подп. и дата

Подп. и дата

Инв.№дубл.



Инв.№подл.

Лист 3

Провер.

Багласова Е.В

Разраб.

Сизов А.И.

Провер.

Багласова Е.В

Т. Контр.

Утверд.

№ докум.

Подпись

Дата

Изм.

Лист

Н. Контр.

Масса

Лит.

Масштаб

КБП

КП Т.592020.401 ГЧ



Реценз.

Листов 5

Программная реализация игры «First Person Shooter»

Диаграмма последовательности

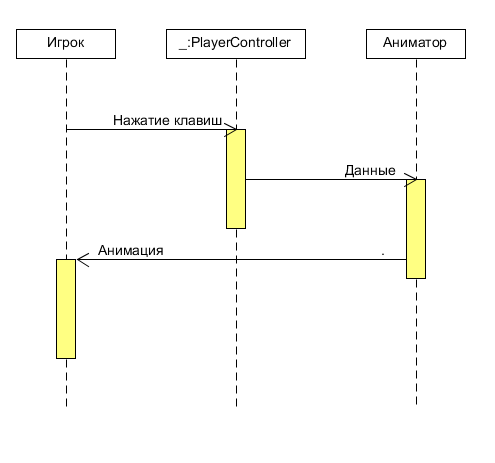
У

Взам.инв.№

Подп. и дата

Подп. и дата

Инв.№дубл.



Инв.№подл.

Лист 4

Разраб.

Сизов А.И.

Провер.

Багласова Е.В

Т. Контр.

Утверд.

№ докум.

Подпись

Дата

Изм.

Лист

Н. Контр.

Масса

Лит.

Масштаб

КБП

КП Т.592020.401 ГЧ



Реценз.

Листов 5

Программная реализация игры «First Person Shooter»

Диаграмма классов

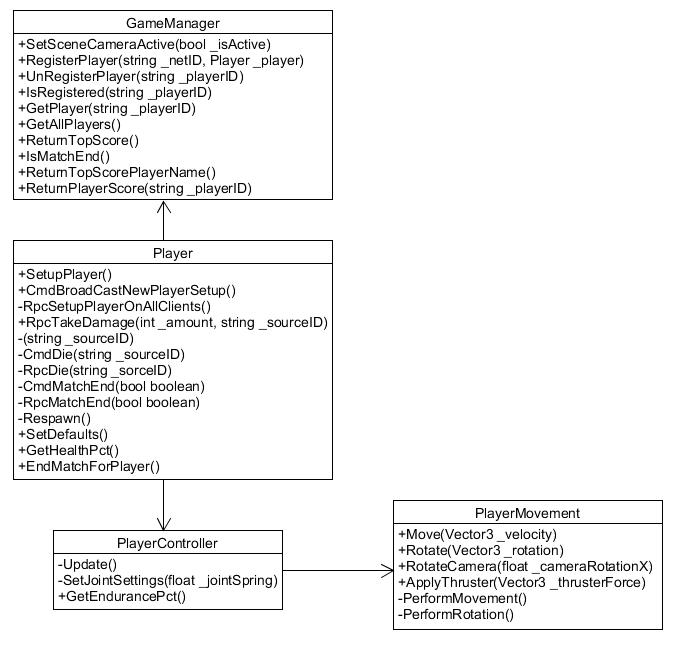
У

Взам.инв.№

Подп. и дата

Подп. и дата

Инв.№дубл.



Лист 5

Инв.№подл.