

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники»  
филиал «Минский радиотехнический колледж»

Допущен к защите  
Преподаватель  
\_\_\_\_\_ /А.В. Батура/

## **ПРОГРАММНОЕ СРЕДСТВО «FPS PROJECT»**

Пояснительная записка  
курсового проекта по учебному предмету  
«Основы алгоритмизации и программирования»  
МРК КП 5-04-0612-02 008ПЗ

Выполнил учащийся группы 4к9341  
\_\_\_\_\_ /А.В. Конон/

## СОДЕРЖАНИЕ

Введение .....	3
1 Постановка задачи .....	5
1.1 Описание предметной области.....	5
1.2 Обзор существующих аналогов .....	6
1.3 Функциональное назначение.....	9
2 Проектирование задачи .....	16
2.1 Алгоритм работы программы.....	16
2.2 Логическое моделирование .....	19
2.3 Описание инструментов разработки.....	21
3 Программная реализация .....	24
3.1 Физическая структура .....	24
3.2 Описание разработанных модулей .....	31
4 Тестирование .....	36
5 Применение .....	41
5.1 Назначение и условия применения.....	41
5.2 Руководство пользователя .....	41
Заключение .....	50
Список использованных источников .....	51
Приложение А(обязательное)Текст программы.....	52
Приложение Б(обязательное)Изображения интерфейса программы .....	67

					МРК КП 5-04-0612-02 008ПЗ						
Изм.	Лист	№ докум	Подпись	Дата							
Разраб		Конон			«FPS Project»			Лит	Лист	Листов	
Пров		Батура								2	70
Реценз.											
Н. Контр.											
Утверд.											
					Пояснительная записка						

## ВВЕДЕНИЕ

Целью курсового проекта является разработка программного средства под названием «FPS Project» для создания интерактивного игрового опыта.

Шутеры от первого лица (First Person Shooter) – один из самых популярных жанров компьютерных игр, сочетающий динамичный игровой опыт, стратегическое планирование и реалистичную визуализацию с помощью 2,5D-графики и/или 3D-графики. Данный проект направлен на создание увлекательной игровой среды, которая позволит игрокам погрузиться в «экшен-сражения», развивать реакцию и тактическое мышление.

Существует немало научных исследований видеоигр от первого лица. Как правило, они изучают восприятие, когнитивные процессы и эмоциональные реакции игроков. Некоторые учёные рассматривают влияние перспективы от первого лица на уровень погружения в игру. Например, исследователи из Университета Индианы в США доказали, что игры от первого лица активируют области мозга, отвечающие за пространственную ориентацию и внимание. Происходит это из-за того, что игрок видит виртуальный мир глазами персонажа.

Исторически первым «шутером» от первого лица считается «Maze War» (1973). Видеоигру для компьютера Imlac PDS-1 разработали студенты Стив Колли, Грег Томпсон и Говард Палмер в Исследовательском центре Эймса космического агентства NASA. Игрок мог ходить по лабиринту, поворачиваться на 90° и заглядывать в дверные проёмы. Тайловая («плиточная») графика была предельно простой, а основной целью игры был поиск выхода из лабиринта. Чуть позже разработчики добавили в игру возможность стрелять.

Разрабатываемое программное средство будет поддерживать не только одиночные сражения, но и многопользовательский режим посредством технологии разделённого экрана, что позволяет одновременно нескольким игрокам взаимодействовать в одном игровом пространстве, используя отдельные части экрана для отображения каждого участника.

Первой видеоигрой, которая поддерживала функцию игры на разделённом экране, стала «Drag Race», выпущенная в 1977 году. Игра представляла собой симулятор гонок. Что соответствовало типичной структуре игровых автоматов в жанре гонки того времени: наличие дисплея, педали для ускорения и руля для управления автомобилем. Однако, игра отличалась от других автоматов того периода важной особенностью – наличием элементов управления для второго игрока, что позволяло двум пользователям соревноваться друг с другом. Несмотря на возможность игры с искусственным интеллектом, настоящее увлечение возникало именно при наличии живого оппонента.

Таким образом, ещё до появления локальных сетей и интернета видеоигры могли предоставлять возможность не только сражаться с компьютерными противниками,

но и взаимодействовать с другими людьми в реальном времени. Разделённый экран, а также элементы управления, позволявшие двум игрокам одновременно участвовать в игровом процессе, стали основой для создания первой многопользовательской видеоигры с режимом разделённого экрана. Хотя в других играх того времени также существовала возможность соревноваться, например, в контексте таблиц рекордов, игра в режиме разделённого экрана позволяла игрокам ощущать волнение и азарт не от результатов в конце игры, а непосредственно от самого процесса соревнования в реальном времени.

В последние годы наблюдается возрождение интереса к видеоиграм с технологией разделённого экрана, что представляет собой не просто ностальгическую ретроспективу, а закономерный ответ на актуальные запросы игрового сообщества. В условиях доминирования онлайн-многопользовательских форматов в современной игровой индустрии, значительная часть пользователей испытывает потребность в восстановлении тех уникальных социально-эмоциональных паттернов взаимодействия, которые характерны для локального кооперативного режима. Создание данного программного средства нацелено на создание игрового опыта, который объединяет игроков в физически общем пространстве, способствуя живому общению, мгновенной обратной связи и укреплению межличностных связей через совместное преодоление игровых испытаний.

# 1 ПОСТАНОВКА ЗАДАЧИ

## 1.1 Описание предметной области

Одной из задач программного средства будет являться реализация концепции «обучение через игру», где развлекательные элементы органично сочетаются с возможностью развития когнитивных и моторных навыков. Такой подход обеспечивает не только увлекательный игровой процесс, но и практическую пользу для пользователей.

Разрабатываемое программное средство предоставит игрокам возможность участвовать в одиночных и многопользовательских сражениях, используя разнообразное вооружение. Удобный интерфейс и классическое управление позволит пользователям быстро освоить игровой процесс.

Программное средство будет сочетать в себе развлекательный и обучающий контент, что позволит создать не просто интересную игру, но и эффективный инструмент для развития важных умений. Пользователи смогут совершенствовать тактические навыки, точность стрельбы и реакцию, анализируя поведение противников с искусственным интеллектом. Развлекательный аспект проекта способствует снижению уровня стресса и позволяет отвлечься от повседневных задач, выполняя функцию психологической разгрузки. Игровой процесс становится способом самовыражения и временного ухода от реальности, что особенно актуально в условиях высокой учебной или профессиональной нагрузки.

Программное средство будет обеспечивать возможность локального кооперативного прохождения с применением «сплит-скрин» технологии (разделенного экрана). Оно позволяет нескольким игрокам одновременно участвовать в игровом процессе на одном устройстве, используя разные части экрана для отображения индивидуальных камер.

Ключевой особенностью реализации проекта будет являться его разработка преимущественно с использованием языка программирования C++ в рамках игрового движка Unreal Engine 5. В отличие от более традиционного подхода с применением визуального программирования на «Blueprints», использование C++ позволит добиться более высокой производительности, гибкости кода и тонкой настройки игровых механик. Прямое использование кода обеспечит возможность создания оптимизированной архитектуры проекта, эффективного управления памятью и более глубокого контроля за логикой поведения игровых объектов, включая искусственный интеллект противников, систему оружия и взаимодействие между игроками.

В рамках программного средства будет реализована локальная система рейтинга, сохраняющая индивидуальные достижения пользователей на локальном устройстве. Система рейтинга служит инструментом для оценки прогресса, позволяя отслеживать динамику развития игрока. Регулярное обновление данных и

отображение ключевых показателей создают условия для здоровой конкуренции и самосовершенствования. Особое внимание в будет уделено созданию надёжной системы фиксации достижений пользователей. После гибели персонажа игроку будет предоставляться возможность ввести своё имя, которое вместе с результатом будет сохранено в бинарном файле. Использование бинарного формата хранения данных обусловлено необходимостью обеспечения быстрого доступа к информации, защиты от несанкционированного редактирования и минимизации объёма хранимых данных. Механизм сериализации и десериализации данных должен быть реализован средствами языка C++ без использования сторонних библиотек.

Для разработки игрового приложения будут использованы современные инструменты и технологии, включая возможности игрового движка Unreal Engine 5. В результате, компьютерное приложение «FPS Project» предоставит пользователям динамичный и увлекательный игровой процесс с возможностью участия в одиночных и многопользовательских сражениях. Программное средство будет направлено на обеспечение высокого уровня производительности, гибкости и контроля над игровыми механиками, обеспечивая комфортное взаимодействие.

## 1.2 Обзор существующих аналогов

В настоящее время на рынке существует широкий спектр кооперативных видеоигр, предлагающих различные функциональные возможности для взаимодействия между игроками. Эти игры играют ключевую роль в сфере развлечений, способствуя улучшению социального взаимодействия и созданию уникальных игровых впечатлений. Каждое игровое приложение обладает собственными особенностями и характеристиками, что делает его привлекательным для определенных групп пользователей. Выбор игры во многом определяется предпочтениями игроков, характеристиками игрового процесса, а также возможностями локального и онлайн-взаимодействия, что в конечном итоге влияет на популярность и востребованность видеоигры.

Рассмотрим существующий аналог в жанре перестрелки от первого лица, представленный на рисунке 1.1 – Far Cry 4. Одним из главных преимуществ Far Cry 4 является обширный и детализированный открытый мир, который предоставляет игрокам возможность свободно исследовать различные локации и участвовать в сражениях. Far Cry 4 поддерживает кооперативный режим, позволяя двум игрокам совместно исследовать мир и выполнять задания. Однако, несмотря на все свои достоинства, Far Cry 4 имеет и несколько недостатков. Во-первых, игра имеет значительные требования к системным ресурсам, что может ограничить доступность на некоторых платформах или у игроков с менее мощными компьютерами или консолями. Во-вторых, многопользовательский режим игры требует подключения к

интернету, что ограничивает возможность полноценной игры в условиях отсутствия сети.



Рисунок 1.1 – Пример игровой сессии Far cry 4

Перейдем к другому аналогу – Fortnite. Этот многопользовательский онлайн-шутер, представленный на рисунке 1.2, разработан компанией Epic Games и предлагает захватывающий игровой процесс с элементами перестрелки и строительных механик. Fortnite также поддерживает кроссплатформенную игру, позволяя пользователям на разных устройствах, включая персональные компьютеры, консоли и мобильные устройства, играть вместе. Одним из недостатков Fortnite является отсутствие локального многопользовательского режима, что ограничивает возможность игры в рамках локальной сети. Все сражения происходят исключительно через интернет-соединение, что требует постоянного подключения к сети и создает зависимость от внешних факторов, таких как качество интернет-соединения и серверная нагрузка.



Рисунок 1.2 – Пример игровой сессии Fortnite

Последний аналог – Wolfenstein: The New Order (2014), представленный на рисунке 1.3. Одним из значительных преимуществ Wolfenstein: The New Order является её внимание к деталям, интересный сюжет, захватывающие персонажи и динамичный игровой процесс. Игра сочетает классические перестрелки с элементами скрытности, а также имеет продуманный выбор оружия, включая уникальные и экспериментальные технологии. Однако у игры есть и несколько недостатков. Во-первых, как и многие современные шутеры, Wolfenstein: The New Order не поддерживает локальный многопользовательский режим. Вся игра предназначена для одиночного прохождения и имеет лишь опцию многопользовательской игры в онлайн режиме. Во-вторых Wolfenstein: The New Order предлагает графику, которая, несмотря на свою высокую детализированность и визуальные эффекты на момент выпуска игры, может быть воспринята как устаревшая по сравнению с современными стандартами.

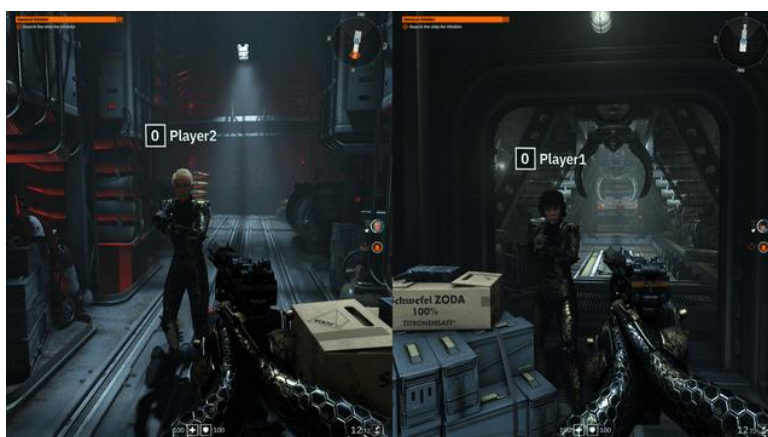


Рисунок 1.3 – Пример игровой сессии Wolfenstein: The New Order

В заключение проведённого сравнения игр, таких как Fortnite, Wolfenstein: The New Order и их особенностей, в контексте разработки аналогичного проекта можно сделать несколько ключевых выводов, которые помогут нам учесть важные аспекты при создании нашего шутера с поддержкой разделенного экрана.

Прежде всего, необходимо отметить, что в отличие от Fortnite и Far Cry 4, наш проект будет ориентирован на локальный кооперативный режим с использованием технологии разделённого экрана. Это позволит создать уникальный опыт взаимодействия между игроками в реальном времени в рамках одного физического пространства. Такой подход поможет восстановить социально-эмоциональные аспекты взаимодействия, которые были характерны для более ранних игр с локальными сетями и разделёнными экранами.

Что касается Wolfenstein: The New Order, несмотря на свои сильные сюжетные линии и качественное исполнение механик, игра имеет устаревшую графику, что



может быть значительным ограничением в условиях быстрого развития технологий и более высоких требований к визуальным эффектам в современных играх.

### **1.3 Функциональное назначение**

Информационная база представляет собой упорядоченный набор данных, сохраняемых в компьютерной системе в виде файлов или записей. Её основное назначение – обеспечение информационных потребностей различных процессов и поддержка принятия решений. Структурированность такой базы позволяет систематизировать данные по заданным критериям, что способствует их эффективному хранению, обработке и последующему извлечению.

Входными данными для разрабатываемого компьютерного приложения являются параметры, получаемые от пользователей в процессе взаимодействия с программой. К ним относятся: действия игроков в ходе игрового процесса (например, передвижение, стрельба, подбор предметов), настройки управления и визуализации, а также информация, вводимая при сохранении результатов, такая как имя пользователя для таблицы рекордов. Все эти данные обрабатываются внутри приложения с целью формирования соответствующего отклика игрового окружения и дальнейшего хранения пользовательских достижений.

Выходными данными компьютерного приложения являются визуальные и звуковые отклики, формируемые в ответ на действия пользователя, а также результаты игрового взаимодействия. К ним относятся: отображение игрового процесса на экране (движение персонажа, стрельба, взаимодействие с объектами), изменения в пользовательском интерфейсе (например, обновление количества боеприпасов или здоровья), звуковые эффекты, а также финальные игровые результаты, такие как набранные очки и записи в таблице рекордов. Эти данные позволяют пользователю отслеживать собственный прогресс и получать обратную связь о ходе игры.

Целью разработки программного средства является создание интерактивной многопользовательской игры с поддержкой локального кооператива посредством технологии разделённого экрана, реализованной на игровом движке Unreal Engine 5 с использованием языка программирования C++. Программное средство ориентировано на обеспечение увлекательного игрового опыта, сочетающего элементы соревнования в условиях одного физического пространства.

Игровой процесс предполагает участие одного или двух игроков в сражениях против противников, управляемых искусственным интеллектом. В рамках игры реализованы механики выбора, использования и пополнения боевого арсенала, включая систему подбора оружия и патронов. Особое внимание уделяется разработке точной, производительной логики поведения игровых объектов, что достигается за

счёт отказа от визуального программирования в пользу прямого написания кода на C++.

Разрабатываемое программное средство представляет собой аркадную игру в жанре шутера от первого лица, ориентированную как на одиночное, так и на локальное многопользовательское прохождение с использованием технологии разделённого экрана (split-screen). Основной целью является создание интерактивной среды, в которой игроки смогут не только сразиться с виртуальными противниками, управляемыми искусственным интеллектом, но и взаимодействовать друг с другом в кооперативном формате. Программное средство обеспечивает реализацию игрового процесса, направленного на развитие координации, реакции и стратегического мышления, а также предоставление развлекательного и обучающего контента.

Перед началом игровой сессии пользователь получает доступ к экрану предварительных настроек. На данном этапе предоставляется возможность тонкой настройки параметров графики и звука. В частности, игрок может регулировать яркость изображения в зависимости от условий освещения в помещении, а также настраивать громкость звуковых эффектов и музыкального сопровождения. Эта настройка позволяет сделать игровой процесс более комфортным и адаптированным к индивидуальным предпочтениям каждого игрока.

Одной из ключевых функций интерфейса предварительного экрана является выбор режима игры. Пользователю предоставляется возможность выбрать количество игроков – один или два. В случае выбора одиночного режима игрок будет взаимодействовать исключительно с противниками, управляемыми искусственным интеллектом. При выборе режима на двоих экран автоматически делится на две части, каждая из которых представляет перспективу одного из игроков. Это позволяет двум пользователям одновременно участвовать в игровом процессе на одном устройстве, используя отдельные контроллеры или клавиши управления на одной клавиатуре.

После запуска игрового процесса игроки оказываются в виртуальной среде, где основной задачей является выживание и уничтожение как можно большего количества врагов. На игровой карте располагаются различные элементы: враги с искусственным интеллектом, оружие, патроны и другие вспомогательные объекты. Противники обладают предопределёнными шаблонами поведения, включающими перемещение, уклонение от выстрелов и атаку игрока.

Игровой процесс сопровождается возможностью подбирать разнообразное оружие на карте. Каждое оружие обладает уникальными характеристиками: скорострельностью, уроном, дальностью стрельбы и объёмом магазина. Игрок может менять оружие в зависимости от ситуации, что вносит дополнительную стратегическую составляющую в игровой процесс. Боезапас не является бесконечным, и для его пополнения пользователю необходимо собирать патроны, находящиеся на территории уровня.

Важно отметить, что в процессе игры система ведёт автоматический учёт действий пользователя. Каждое убийство врага приносит игроку определённое количество очков, которое зависит от типа противника, выбранного оружия и других факторов. Также учитываются иные параметры – например, количество используемых патронов, перезарядок, убитых врагов. Эти показатели формируют итоговую оценку эффективности игрока, которая отображается по завершении уровня или при гибели персонажа.

После окончания игры пользователю предоставляется возможность ввести своё имя, под которым результат будет занесён в таблицу лидеров. Таблица лидеров реализована в виде локального файла, хранящегося на устройстве пользователя. Для повышения надёжности и скорости доступа данные таблицы записываются в бинарном формате. Это позволяет минимизировать объём используемой памяти, обеспечить целостность информации и защитить данные от случайного или преднамеренного редактирования. В случае повторного запуска приложения пользователь может ознакомиться с ранее сохранёнными рекордами, сравнить свои достижения и мотивировать себя к улучшению результатов.

Таким образом, разрабатываемое программное средство предоставляет игрокам не только захватывающий и динамичный игровой процесс, но и возможность соревновательного взаимодействия, как с искусственным интеллектом, так и между собой. Использование современных технологий, таких как разделенный экран, управление на одном устройстве и сохранение результатов в локальном хранилище, делает приложение удобным, доступным и привлекательным как для опытных пользователей, так и для новичков. Кроме того, игровой процесс обладает обучающим потенциалом: он развивает внимание, координацию, точность действий и умение быстро принимать решения. Всё это делает программное средство универсальным решением в области цифровых развлечений с элементами развития когнитивных и моторных навыков.

Компьютерное приложение будет иметь четыре основных окна: окно начала игровой сессии, окно настройки приложения, окно игровой сессии и окно окончания игры.

Окно начала игры представлено на рисунке 1.4. Окно начала игры представляет собой стартовый экран игрового приложения, с которого пользователь начинает взаимодействие с программным продуктом. Данный интерфейс оформлен в соответствии с общим визуальным стилем игры и является центральной точкой доступа ко всем основным функциям перед началом игровой сессии.



Рисунок 1.4 – Окно начала игры

В верхней правой части экрана располагается крупная кнопка «Старт», при нажатии на которую пользователь немедленно переходит к началу самой игры. Эта кнопка запускает основную игровую логику, включая инициализацию карты, генерацию врагов и загрузку игровых объектов.

Ниже размещается кнопка «Настройки игры», открывающая дополнительное диалоговое окно, в котором игрок может изменить параметры аудиовизуального сопровождения. В частности, доступны следующие опции: регулировка громкости фоновой музыки, звуковых эффектов, выбор уровня яркости экрана, а также загрузка собственной фоновой музыки. Эти настройки позволяют адаптировать игру под конкретные технические условия и предпочтения игрока. Особое внимание уделено функции выбора количества игроков, которая представлена в виде кнопки. В зависимости от выбора, игра будет запускаться либо в одиночном режиме, с противниками, управляемыми искусственным интеллектом, либо в кооперативном формате с разделением экрана между двумя игроками. Это делает интерфейс гибким и понятным, позволяя быстро настроить желаемый игровой режим.

Завершает интерфейс кнопка «Выход», которая завершает работу приложения. При нажатии на неё программа корректно закрывается, освобождая ресурсы и завершив все фоновые процессы. Перед выходом возможно появление подтверждающего диалога, предотвращающего случайное завершение сессии.

Окно настроек в игровом приложении представляет собой отдельный интерфейсный раздел, представленный на рисунке 1.5, предоставляющий пользователю возможность индивидуальной конфигурации параметров аудио- и видеосопровождения. Этот экран позволяет адаптировать игру к персональным предпочтениям, техническим особенностям оборудования и создаёт условия для комфортного взаимодействия с программным средством.

Одним из ключевых элементов окна настроек является регулятор громкости, представленный в виде ползунка или шкалы, с помощью которого пользователь может изменять уровень звукового сопровождения в игре. Это касается как фоновой музыки, так и звуковых эффектов, связанных с действиями игрока, окружающей

среды или взаимодействиями с противниками. Такой функционал особенно важен при игре в разных условиях – например, в общественных местах или при необходимости отключить звук без полного прекращения игрового процесса.



Рисунок 1.5 – Окно настроек приложения

Следующей важной функцией является настройка яркости изображения. Этот параметр также представлен в виде ползунка, позволяющего увеличить или уменьшить яркость в пределах заданного диапазона. Возможность регулировки яркости позволяет пользователю адаптировать изображение под освещение в помещении или индивидуальные визуальные предпочтения, обеспечивая комфортное восприятие происходящего на экране.

Дополнительной особенностью окна настроек является функция загрузки музыкальных файлов с диска, которая расширяет стандартный набор опций. Пользователь может выбрать и загрузить собственные музыкальные композиции, которые будут воспроизводиться вместо стандартной фоновой музыки. Для этого предусмотрена кнопка доступа к файловой системе, через которую осуществляется выбор аудиофайлов с локального диска.

Экран окончания игры, представленное на рисунке 1.6, представляет собой интерфейсное окно, которое отображается после завершения игровой сессии. Он выполняет не только информационную функцию, но и предоставляет пользователю ряд опций для дальнейших действий. Центральным элементом экрана является кнопка «Рестарт», позволяющая мгновенно начать новую игру без необходимости возвращаться в главное меню. При нажатии этой кнопки происходит перезапуск текущей игровой сессии с исходными параметрами, такими как режим игры, количество игроков и начальные условия. Это удобно для пользователей, желающих сразу повторить попытку и улучшить свои результаты.

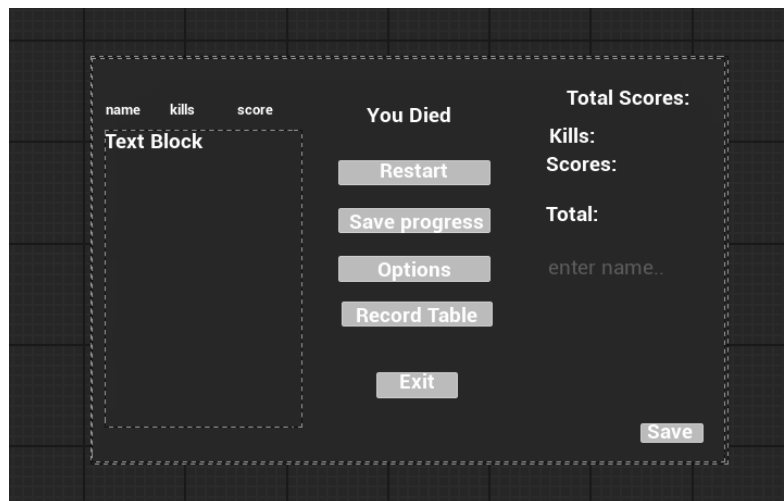


Рисунок 1.6 – Макет окна окончания игры

Кнопка «Сохранить прогресс» служит для записи достигнутых результатов и состояния игрока. Система сохраняет данные, такие как количество набранных очков, количество побежденных противников и имя пользователя. Прогресс записывается во внутреннюю память приложения, что позволяет при следующем запуске продолжить игру с момента последнего сохранения. Такой подход особенно актуален в случае многопользовательских сессий или при использовании рейтинговой системы.

Следующим важным элементом является кнопка «Опции», которая открывает доступ к настройкам игры прямо с экрана завершения. Это позволяет оперативно внести изменения в параметры звука и изображения перед следующей игровой попыткой. Функция особенно полезна, если текущий игровой опыт показал необходимость корректировки настроек.

Кнопка «Показать таблицу лидеров» предоставляет возможность просмотреть локальный рейтинг игроков. Таблица отображает список участников с указанием их результатов: набранные очки, количество убитых противников и имя пользователя.

Кнопка «Выход из игры» позволяет завершить работу приложения. После нажатия пользователь возвращается на рабочий стол операционной системы. Перед окончательным выходом может быть реализован запрос на подтверждение, чтобы избежать случайного завершения сессии без сохранения данных. Окно игрового процесса, представленное на рисунках 1.7 и 1.8, является основным пространством взаимодействия пользователя с программным продуктом, где отображаются ключевые элементы интерфейса, обеспечивающие информирование игрока о текущем состоянии и ходе игры. В верхней части экрана расположены важные индикаторы, такие как счётчик патронов и индикатор жизней. Счётчик патронов отображает количество доступных боеприпасов как в текущем магазине, так и в общем запасе, что помогает игроку планировать свои действия и избегать неожиданных ситуаций в бою. Индикатор жизней отображает текущее здоровье игрока, что представлено в графическом виде, позволяя быстро оценить состояние

персонажа. В центре экрана размещается прицел, который служит для наведения оружия и точного прицеливания. Этот элемент помогает игроку точно ориентироваться при стрельбе и взаимодействии с объектами или противниками. Визуальное оформление прицела может меняться в зависимости от выбранного оружия или состояния игрока, например, при прицеливании через оптический прицел.

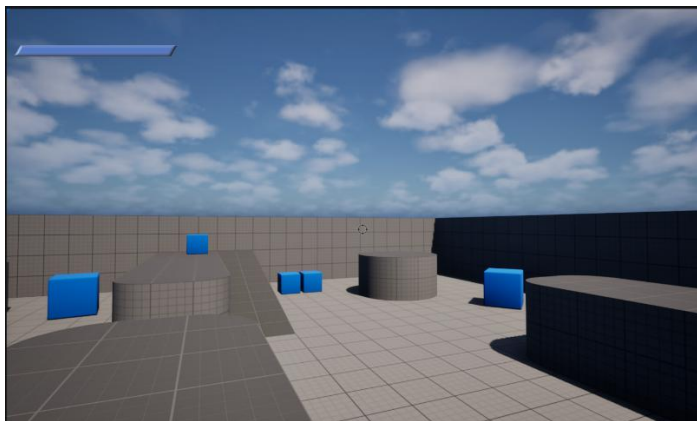


Рисунок 1.7 – Окно игрового процесса(один пользователь)

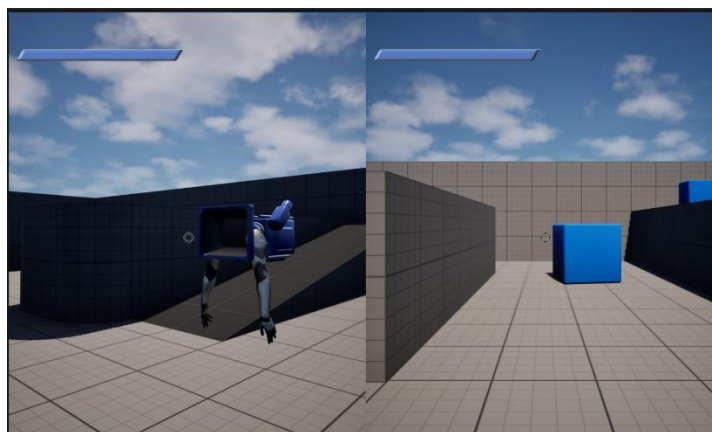


Рисунок 1.8 – Окно игрового процесса(два пользователя)

Все эти элементы вместе обеспечивают игроку полную информацию о его текущем состоянии и позволяют принимать обоснованные тактические решения в ходе игрового процесса. Главная цель разработки заключается в создании динамичного и увлекательного игрового опыта, который сочетает элементы тактики, быстроты реакции и взаимодействия с другими игроками. Игровая механика будет включать такие элементы, как сбор оружия, пополнение боезапаса, а также достижения, фиксируемые в системе таблицы лидеров, что позволяет игрокам соревноваться и отслеживать свои результаты.

## **2 ПРОЕКТИРОВАНИЕ ЗАДАЧИ**

### **2.1 Алгоритм работы программы**

Схема алгоритмов представляет собой графическое отображение процессов и действий, которые происходят в рамках игры, где каждый шаг или действие обозначается определенным графическим элементом, а связи между ними показывают последовательность действий. Этот метод визуализации помогает более наглядно представить логику работы игры и игровой механики, упрощая понимание алгоритмов без необходимости углубляться в детали кода.

Схема алгоритмов является важным инструментом для разработки, отладки и оптимизации игровых процессов. Она помогает в процессе создания игры, а также позволяет легче выявлять и исправлять возможные ошибки и несоответствия в логике игры. Для разрабатываемого программного средства схема алгоритмов будет отображать последовательность действий игрока, взаимодействие с игровым миром, обработку запросов и состояния, такие как перезарядка оружия, изменения в количестве патронов, здоровье персонажа и взаимодействие с противниками.

Схема алгоритмов для компьютерного приложения представляет последовательность операций в зависимости от состояния игры. Она включает в себя различные стадии, такие как начало игры, взаимодействие с противниками, использование оружия, отображение индикаций на экране, а также завершение уровня или игры.

Схема алгоритмов для компьютерного приложения, представленная на рисунке 2.1, отображает последовательность выполняемых функций при различных условиях.

Представленная блок-схема иллюстрирует детализированный алгоритм функционирования игровой программы, охватывающий весь жизненный цикл игрового процесса – от начальной инициализации до завершения сессии и сохранения пользовательских данных. Алгоритм реализован в виде последовательности логических блоков, объединённых управляющими связями, отражающими переходы между различными этапами обработки данных и взаимодействия с пользователем.

Этап инициализации и настройки. Алгоритм начинается с активации стартовой точки – блока «Начало», после чего последовательно выполняется инициализация базовых параметров. В рамках данной процедуры осуществляется:



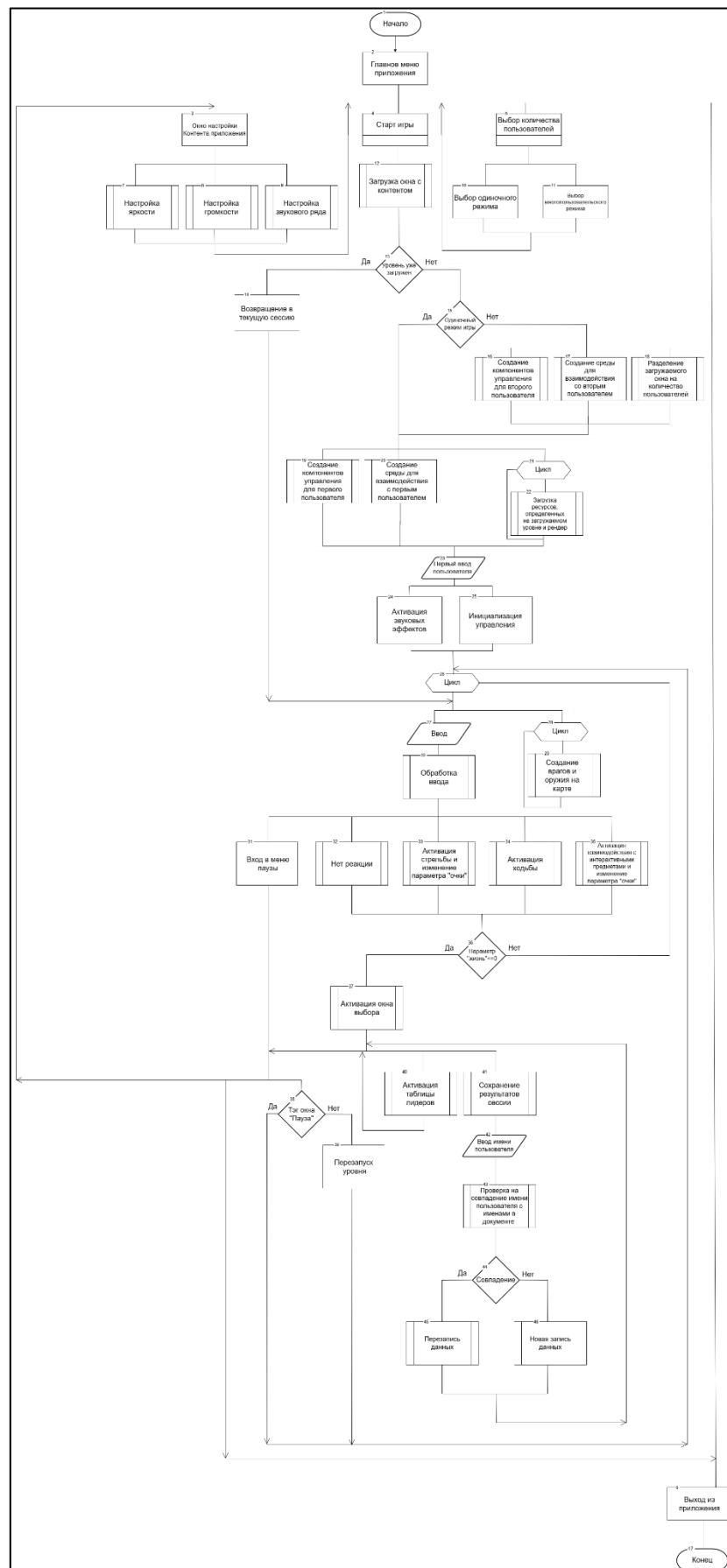


Рисунок 2.1 – Блок-схема алгоритма работы разрабатываемого компьютерного приложения

Настройка яркости – регулировка параметров отображения изображения в зависимости от предпочтений пользователя; Настройка громкости – установка уровня звукового сопровождения; Настройка звукового ряда – выбор и калибровка аудиодорожек, сопровождающих игровой процесс.

После завершения базовой настройки загружается основное окно с контентом, выполняющее функцию стартового интерфейса пользователя. Это окно служит отправной точкой для выбора режима взаимодействия и дальнейшего разветвления логики исполнения.

Определение режима игры. На следующем этапе алгоритм производит проверку наличия второго пользователя, что определяет характер последующего игрового взаимодействия:

В случае однопользовательского режима происходит формирование интерфейса управления и элементов взаимодействия исключительно для одного игрока. Если же второй пользователь подключён, иницируется режим многопользовательской игры, при котором: создаются отдельные компоненты управления для второго игрока; формируется среда взаимодействия между двумя участниками; осуществляется разделение экрана в соответствии с количеством пользователей для обеспечения параллельного отображения игровых действий. Таким образом, система адаптируется к текущей конфигурации взаимодействия и подготавливает соответствующую среду выполнения. Загрузка игровых ресурсов и цикл обработки

После инициализации режима игры начинается цикл загрузки игровых ресурсов, в который входит: Загрузка отображаемых объектов, включая врагов и уровни; Формирование структуры управления на карте.

Затем система ожидает ввод от пользователя, иницируя основной цикл обработки игрового взаимодействия. Этот цикл реализуется как замкнутая система, включающая следующие этапы: Получение ввода с устройств управления; Обработка ввода, результатом которой является либо: Отсутствие реакции (при недопустимом или нераспознанном действии), после чего осуществляется возврат к началу цикла; Либо активация соответствующих игровых действий, включая: Стрельбу и модификацию параметра «очки»; Ходьбу персонажа; Взаимодействие с интерактивными элементами, также влияющее на параметр «очки». Этот цикл продолжается до наступления определённых условий, требующих выхода из него (например, активация дополнительных окон или завершение уровня). Дополнительные функции и пользовательские действия. Параллельно с основным игровым процессом система допускает вызов дополнительных окон, например: Окно выбора, где пользователь может осуществить действия вне основного сценария; В частности, при выборе опции «Пауза» активируются: Окно с таблицей лидеров, отображающее сравнительные результаты; Процедура сохранения результатов текущей игровой сессии. Также предусмотрена возможность ввода имени нового пользователя для последующей персонализации статистики и данных.

Верификация и сохранение данных. После завершения активной игровой сессии производится ввод имени пользователя, который далее проверяется на совпадение с существующими записями в базе данных или документе: В случае совпадения выполняется перезапись данных (обновление ранее сохранённой информации); при отсутствии совпадения создаётся новая запись, соответствующая новому пользователю.

Таким образом, обеспечивается непрерывность статистики и учёт пользовательской активности. Завершение алгоритма. Финальной стадией работы алгоритма является блок «Конец», сигнализирующий о завершении работы приложения. На этом этапе все временные ресурсы освобождаются, и выполнение логики программы прекращается.

Данный алгоритм демонстрирует хорошо структурированный подход к организации игрового процесса, сочетающий гибкость пользовательского взаимодействия, адаптацию под различные режимы игры, а также надёжную систему обработки данных. Блок-схема охватывает не только основные циклы обработки ввода, но и вспомогательные механизмы сохранения информации, что делает её пригодной для реализации в игровых системах с возможностью масштабирования.

Блок-схема алгоритма работы программного средства «FPS Project» представлена в графической части МРКК.507112.001.

## **2.2 Логическое моделирование**

В рамках проектирования игрового приложения было выполнено логическое моделирование пользовательского взаимодействия с системой, изображенной на рисунке 2.2. Это моделирование представлено в виде диаграммы прецедентов UML, на которой отображены ключевые действия пользователя (так называемые «прецеденты») и их взаимосвязи. Основной акцент сделан на описании поведения системы с точки зрения пользователя, без детализации внутренней реализации – именно это и составляет суть логического моделирования.

Главным участником системы выступает пользователь, который в процессе взаимодействия проходит через ряд логически взаимосвязанных этапов. Всё начинается с настройки приложения. Этот шаг представляет собой первоначальную конфигурацию: установка параметров графики, звука, выбора языка, управления, а также возможное подключение учётной записи или профиля. Пользователь выбирает удобный способ управления – клавиатура с мышью или геймпад – и вводит необходимые данные. Это действие сопровождает практически все последующие, так как взаимодействие с системой невозможно без ввода команд.

Завершив настройку, пользователь инициирует запуск основной игры – начинает игровую сессию. Эта точка является ключевой: от неё ответвляются все дальнейшие действия, составляющие игровой процесс. На протяжении сессии игрок исследует

виртуальное пространство, где может подбирать интерактивные предметы. Эти предметы могут представлять собой оружие, патроны, аптечки или другие полезные объекты. При обнаружении оружия или боеприпасов, пользователь имеет возможность поместить их в инвентарь. Наличие этих ресурсов открывает следующий уровень взаимодействия – перезарядку оружия.

После того как оружие готово к использованию, игрок получает возможность вступать в боевые столкновения и наносить урон противникам. Это активное взаимодействие с игровыми объектами – одно из центральных действий в сессии. Как результат успешной боевой активности, пользователь зарабатывает очки. Система начисляет их за каждое поражённую цель.

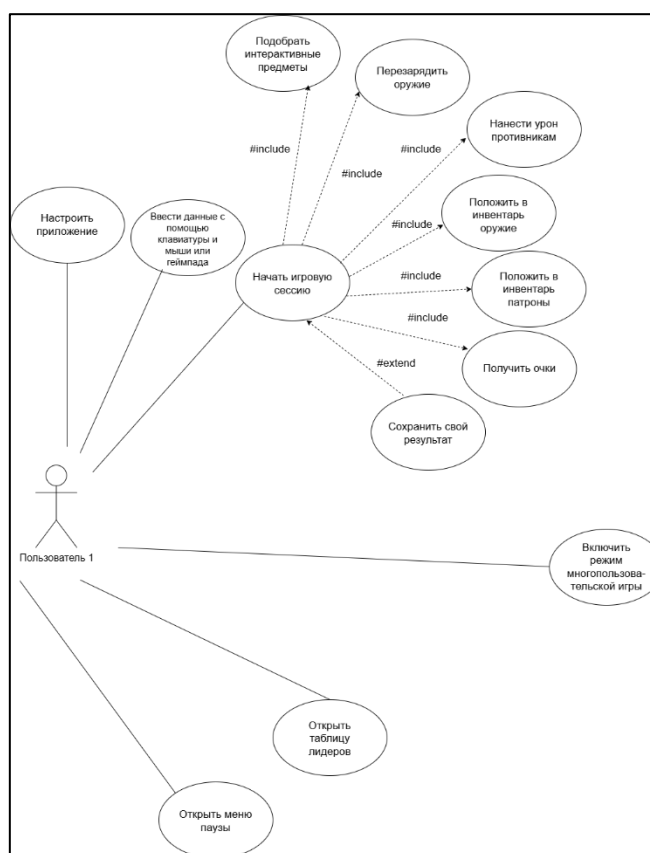


Рисунок 2.2 – Диаграмма вариантов использования для программного средства

Полученные очки могут быть сохранены, что позволяет зафиксировать достигнутый прогресс. Это сохранение особенно важно в контексте многопользовательской игры и сравнения результатов между игроками.

Отдельно стоит отметить, что в любой момент пользователь может приостановить игровой процесс, вызвав меню паузы. Это вспомогательное, но важное действие, предоставляющее доступ к ряду дополнительных возможностей. Одной из них является просмотр таблицы лидеров, где отображаются достижения других игроков и личный рейтинг. Это усиливает мотивацию и добавляет элемент

соревнования. Также из меню паузы можно активировать режим многопользовательской игры. Эта опция предполагает сетевое подключение, выбор или создание игровой комнаты и взаимодействие с другими участниками в реальном времени. Переход в кооперативный режим может полностью изменить характер сессии, предоставляя пользователю новые цели, вызовы и сценарии взаимодействия.

Логические связи между действиями чётко структурированы. Некоторые из них являются обязательными и описываются отношением включения (include) – например, для того чтобы нанести урон противнику, необходимо сначала перезарядить оружие, а до этого – иметь его в инвентаре. Другие связи реализованы как расширения (extend), то есть они необязательны и активируются при определённых условиях. Так, например, возможность открыть таблицу лидеров появляется только в меню паузы, и не обязательно используется в каждой сессии.

Таким образом, логическое моделирование, представленное в виде диаграммы прецедентов, предоставляет системное представление о пользовательских сценариях, подчеркивая основные и вспомогательные функции системы. Оно позволяет точно определить, какие действия пользователь может предпринять, в каком порядке и при каких условиях. Такая модель особенно полезна на ранних этапах проектирования, поскольку она позволяет формализовать требования к функциональности, выявить зависимости между компонентами и избежать логических противоречий в интерфейсе и поведении системы.

Разработанная диаграмма вариантов использования для программного средства «FPS Project» представлена в графической части МРКК.507112.002ПЛ.

## **2.3 Описание инструментов разработки**

Для создания данного программного продукта был выбран язык программирования C++. Этот язык широко используется в индустрии разработки программного обеспечения, особенно в тех областях, где критически важны высокая производительность и полный контроль над аппаратными ресурсами – в частности, в разработке компьютерных игр.

Одним из ключевых факторов выбора C++ для игрового проекта является его высокая скорость выполнения кода. Язык предоставляет разработчику низкоуровневые инструменты для управления памятью, что позволяет эффективно использовать ресурсы компьютера. Это особенно важно в играх, где одновременно обрабатываются физика, анимации, пользовательский ввод, звук, графика и логика искусственного интеллекта. Возможность ручного управления памятью позволяет избежать ненужных затрат ресурсов и точно контролировать поведение системы.

В качестве среды разработки использовалась Microsoft Visual Studio 2022, которая предоставляет мощные инструменты для программирования на C++. Visual Studio – это интегрированная среда разработки (IDE), разработанная компанией

Microsoft. Visual Studio включает в себя редактор кода с подсветкой синтаксиса, интеллектуальную автодополнение (IntelliSense), встроенный отладчик, а также визуальные средства для проектирования интерфейсов. Это позволяет легко создавать проекты, управлять файлами и настройками, а также использовать множество инструментов для разработки, отладки и тестирования. Интерфейс Microsoft Visual Studio 2022 представлен на рисунке 2.3.

Разработка интерфейса и логики игры велась с использованием игрового движка Unreal Engine 5 – одного из самых мощных и широко применяемых инструментов в современной игровой индустрии. Unreal Engine 5 предоставляет встроенные средства для создания как 2D, так и 3D-интерфейсов, включая меню, таблицы рекордов, экраны настроек и другие элементы пользовательского взаимодействия. Благодаря системе UMG (Unreal Motion Graphics) разработчик может визуально проектировать интерфейсы с помощью редактора, перетаскивая элементы на экран и настраивая их свойства без необходимости ручного кодирования базовых функций. Пример работы с движком Unreal Engine 5, представлен на рисунке 2.4.

Unreal Engine 5 предлагает продвинутую визуальную систему разработки, которая позволяет быстро собирать прототипы интерфейсов и интегрировать их с игровыми событиями и логикой. Также поддерживается полное взаимодействие с C++, что даёт возможность разрабатывать как на низком уровне (для оптимизации производительности), так и на высоком (используя визуальный скриптинг в Blueprints).

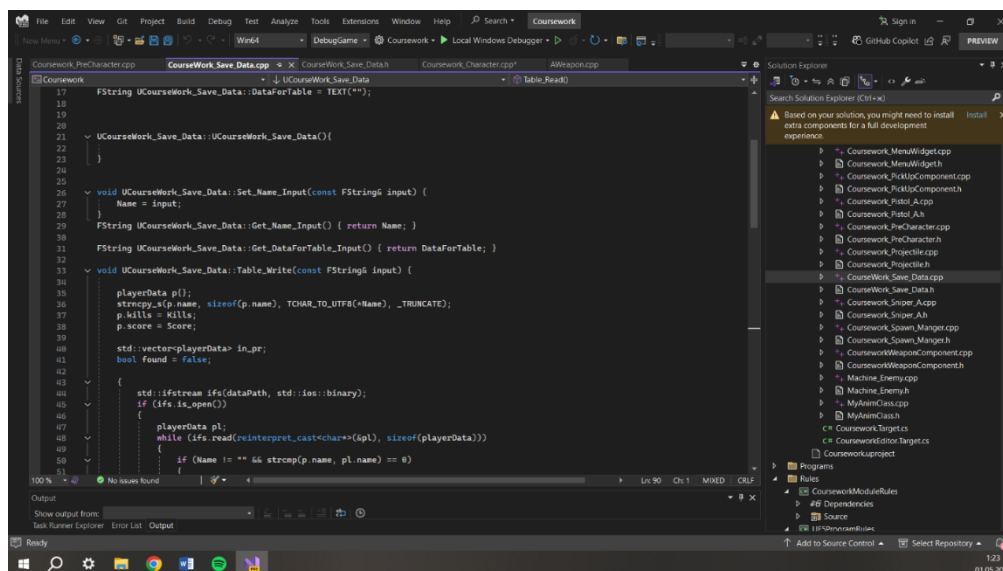


Рисунок 2.3 – Интерфейс Visual Studio 2022

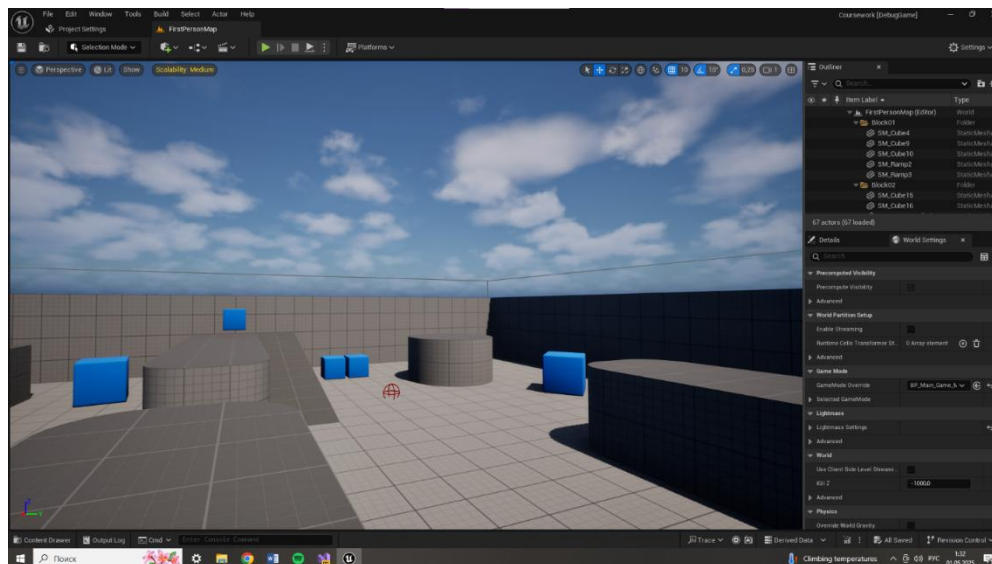


Рисунок 2.4 – Пример работы с Unreal Engine 5

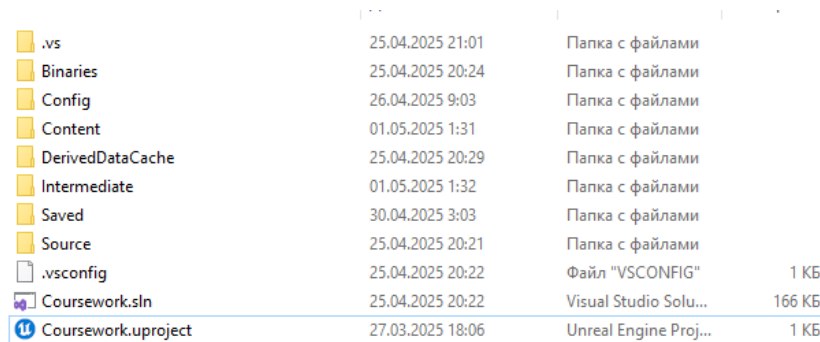
Основные преимущества Unreal Engine 5 при разработке:

- поддержка визуального создания UI с возможностью анимации, адаптации под разные экраны и привязки к игровым событиям;
- поддержка Windows, Mac, Linux, Android, iOS, PlayStation, Xbox и др.
- широкое сообщество, официальная документация и множество обучающих материалов.
- распространяется бесплатно.
- поддержка разработки на C++ даёт полный контроль над игровыми системами.

## 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

### 3.1 Физическая структура


Корневой каталог компьютерного приложения «FPS Project» состоит из восьми папок и трех файлов. Содержание корневой папки представлено на рисунке 3.1.



.vs	25.04.2025 21:01	Папка с файлами	
Binaries	25.04.2025 20:24	Папка с файлами	
Config	26.04.2025 9:03	Папка с файлами	
Content	01.05.2025 1:31	Папка с файлами	
DerivedDataCache	25.04.2025 20:29	Папка с файлами	
Intermediate	01.05.2025 1:32	Папка с файлами	
Saved	30.04.2025 3:03	Папка с файлами	
Source	25.04.2025 20:21	Папка с файлами	
.vsconfig	25.04.2025 20:22	Файл "VSCONFIG"	1 КБ
Coursework.sln	25.04.2025 20:22	Visual Studio Solu...	166 КБ
Coursework.uproject	27.03.2025 18:06	Unreal Engine Proj...	1 КБ

Рисунок 3.1 – Содержание корневой папки

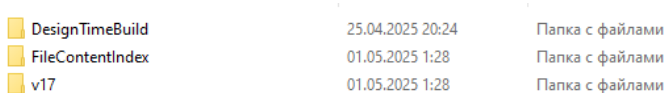
Папка «.vs», содержание которой представлено на рисунке 3.2, содержит настройки проекта интегрированной среды разработки Visual Studio и временные файлы для работы компьютерного приложения. Это служебная директория, автоматически создаваемая Visual Studio при открытии или создании проекта. Она содержит временные файлы и настройки, относящиеся только к определенной среде разработки, а не к самому исходному коду проекта.



Coursework	25.04.2025 20:24	Папка с файлами
ProjectEvaluation	25.04.2025 21:01	Папка с файлами

Рисунок 3.2 – Содержание папки «.vs»

В папке «Coursework» директория «.vs» расположены папки «DesignTimeBuild», «FileContentIndex», «v17», представленные на рисунке 3.3



DesignTimeBuild	25.04.2025 20:24	Папка с файлами
FileContentIndex	01.05.2025 1:28	Папка с файлами
v17	01.05.2025 1:28	Папка с файлами

Рисунок 3.3 – Содержание папки «Coursework»

В папке «DesignTimeBuild» директория «Coursework» расположен служебный файл, создаваемый Visual Studio в рамках папки «.vs», и он отвечает за кэширование данных IntelliSense.



В папке «FileContentIndex» директория «Coursework» расположены служебные индексные файлы Visual Studio, относящийся к системе IntelliSense и поиску по проекту. Содержание папки «FileContentIndex» представлено на рисунке 3.4.

1f89c276-02c8-4dc4-8ebe-f57a5162a3c2.vsidx	25.04.2025 20:28	Файл "VSIDX"	9 334 КБ
4a4eeb87-31c9-42be-9bcb-c0818f77f284.vsidx	25.04.2025 20:26	Файл "VSIDX"	6 647 КБ
5b4064f2-9f02-4b27-bd85-2306450ec0d3.vsidx	25.04.2025 20:27	Файл "VSIDX"	5 589 КБ
5fcec986-81c0-4b7e-8a73-f2b020edf17d.vsidx	26.04.2025 9:03	Файл "VSIDX"	9 681 КБ
6f04843c-134a-470c-9d31-832ac2198753.vsidx	01.05.2025 1:23	Файл "VSIDX"	10 281 КБ
7f150485-39a3-495e-91c8-a0880de539a7.vsidx	25.04.2025 20:28	Файл "VSIDX"	8 911 КБ
22a7c35e-9063-40ad-8e10-ddb94c638238.vsidx	25.04.2025 20:28	Файл "VSIDX"	9 108 КБ
41b27242-89b2-4cdd-8d65-cc02ed23c2e2.vsidx	25.04.2025 20:28	Файл "VSIDX"	9 991 КБ
264e8386-6a2c-4b6f-acf9-e94817fc0387.vsidx	25.04.2025 20:28	Файл "VSIDX"	10 114 КБ
5507b7a9-adcc-4157-be7d-0403d28c4f45.vsidx	25.04.2025 20:27	Файл "VSIDX"	5 837 КБ
a63b697a-b99f-4bb1-8574-20496bb96609.vsidx	01.05.2025 1:23	Файл "VSIDX"	9 517 КБ
ace0a33f-76a4-4abc-8a58-61f7a4ff13cd.vsidx	01.05.2025 1:28	Файл "VSIDX"	10 112 КБ
ade2db41-5eac-45c4-8398-83f4617ec4eb.vsidx	01.05.2025 1:23	Файл "VSIDX"	5 950 КБ
ae902deb-df1d-4687-87cf-2cd15c6dac75.vsidx	25.04.2025 20:28	Файл "VSIDX"	8 291 КБ
b6bb13ff-64ce-43ff-9b3c-393b925b3ef0.vsidx	25.04.2025 20:28	Файл "VSIDX"	9 632 КБ
bae944ed-5b82-4cec-bb69-fe9f27684f52.vsidx	25.04.2025 20:27	Файл "VSIDX"	5 591 КБ
cb67e838-ecce-4913-a72d-b2baf31adeba.vsidx	01.05.2025 1:23	Файл "VSIDX"	5 696 КБ
cdc5536c-cfae-4835-843b-169b615477b5.vsidx	25.04.2025 20:26	Файл "VSIDX"	5 442 КБ
cf21b6ae-3ef3-491a-9b06-e12b04abc733.vsidx	25.04.2025 20:28	Файл "VSIDX"	9 906 КБ
cf232730-a8dc-487e-b272-83e1c189855b.vsidx	25.04.2025 20:27	Файл "VSIDX"	5 700 КБ
e2099655-27b8-45b8-bf43-535f19b392ab.vsidx	01.05.2025 1:23	Файл "VSIDX"	5 787 КБ
eff1f659-2bf2-49ae-bb5f-2de895bcd969.vsidx	25.04.2025 20:28	Файл "VSIDX"	6 094 КБ
fc6a36a7-d3b8-48d6-80c4-1c1ad07e142a.vsidx	25.04.2025 20:25	Файл "VSIDX"	5 589 КБ

Рисунок 3.4 – Содержание папки «FileContentIndex»

Папка «v17» директория «Coursework» является служебной директорией Visual Studio 2022, обозначающая версию IDE (v17 = Visual Studio 2022). Она содержит временные файлы и кэш, используемые Visual Studio для работы с конкретным решением. Содержание папки «FileContentIndex» представлено на рисунке 3.5.

Preview	01.05.2025 1:21	Папка с файлами	
.futdcache.v2	30.04.2025 3:04	Файл "V2"	8 КБ
DocumentLayout.backup.json	30.04.2025 3:04	Файл "JSON"	15 КБ
DocumentLayout.json	01.05.2025 1:28	Файл "JSON"	17 КБ
fileList.bin	30.04.2025 3:04	Файл "BIN"	9 227 КБ
HierarchyCache.v1.txt	26.04.2025 11:05	Текстовый докум...	14 449 КБ

Рисунок 3.5 – Содержание папки «v17»

Папка «Preview» – это служебная папка, используемая Visual Studio для хранения предварительных или временных данных, связанных с открытием, анализом или визуализацией проекта.

Файл «.futdcache.v2» – это служебный кэш-файл Visual Studio, относящийся к IntelliSense и обработке ссылок на типы в проекте, особенно при работе с большими C++-решениями.

Файл «DocumentLayout.backup.json» – это резервная копия конфигурации расположения вкладок и окон в Visual Studio. Он помогает восстановить интерфейс IDE в случае сбоя, зависания или некорректного закрытия проекта.

Файл «DocumentLayout.json» – это основной файл Visual Studio, который хранит состояние открытых документов и интерфейса IDE для конкретного решения. Он

определяет, какие файлы были открыты в редакторе, в каком порядке, и как были расположены окна.

Файл «fileList.bin» – это внутренний бинарный файл Visual Studio, в котором хранится список недавно открытых или отслеживаемых файлов проекта.

Файл «HierarchyCache.v1.txt» – это текстовый кэш, в котором Visual Studio хранит информацию о иерархии проекта (проекты, подпроекты, файлы, связи между ними).

Папка «ProjectEvaluation» директория «.vs» создаётся Visual Studio и используется для внутренней работы системы MSBuild, которая управляет сборкой проекта. Содержимое папки представлено на рисунке 3.6




	coursework.metadata.v9.bin	25.04.2025 21:01	Файл "BIN"	31 КБ
	coursework.projects.v9.bin	25.04.2025 21:01	Файл "BIN"	1 454 КБ
	coursework.strings.v9.bin	25.04.2025 21:01	Файл "BIN"	1 285 КБ

Рисунок 3.6 – Содержание папки «ProjectEvaluation»

Папка «Binaries», представленная на рисунке 3.1, содержание которой представлено на рисунке 3.7, является служебная папка, автоматически создаваемая Unreal Engine при сборке проекта (Build). В ней хранятся скомпилированные двоичные файлы, которые используются.






	CourseworkEditor-Win64-DebugGame.ta...	01.05.2025 1:29	Файл "TARGET"	952 КБ
	UnrealEditor-Coursework-Win64-Debug...	01.05.2025 1:29	Расширение при...	643 КБ
	UnrealEditor-Coursework-Win64-Debug...	01.05.2025 1:29	Exports Library File	74 КБ
	UnrealEditor-Coursework-Win64-Debug...	01.05.2025 1:29	Program Debug D...	63 132 КБ
	UnrealEditor-Win64-DebugGame.modules	01.05.2025 1:29	Файл "MODULES"	1 КБ

Рисунок 3.7 – Содержание папки «Binaries»

Папка «Config», представленная на рисунке 3.1, содержание которой представлено на рисунке 3.8, является одной из ключевых системных директорий, содержащая настройки проекта и движка, оформленные в виде текстовых конфигурационных файлов .ini. Эта папка должна попадать в систему контроля версий Git, так как она влияет на поведение проекта.







	Layouts	20.04.2025 2:15	Папка с файлами	
	DefaultEditor.ini	20.04.2025 3:04	Параметры конф...	34 КБ
	DefaultEditorPerProjectUserSettings.ini	13.03.2025 12:57	Параметры конф...	1 КБ
	DefaultEngine.ini	26.04.2025 9:03	Параметры конф...	4 КБ
	DefaultGame.ini	13.03.2025 12:57	Параметры конф...	1 КБ
	DefaultInput.ini	13.03.2025 12:57	Параметры конф...	13 КБ

Рисунок 3.8 – Содержание папки «Config»

.ini файлы – это текстовый конфигурационный файл, используемый для хранения настроек проекта, движка, редактора, пользовательского ввода и других параметров. Unreal Engine активно использует .ini файлы как основу своей системы конфигураций.

Папка «Layouts» директория «Config» в проекте не является обязательной, но, если она присутствует, она обычно относится к настройкам интерфейса редактора (Unreal Engine 5 Editor). В этой папке хранятся настройки расположения окон редактора и персонализированные рабочие пространства. В данном проекте она пуста.

Папка «Content», представленная на рисунке 3.1, содержание которой представлено на рисунке 3.9, является основной директорией, в которой хранятся все файлы, необходимые для создания игры. Unreal Engine управляет этим содержимым через Content Browser, где отображаются все модели и ресурсы, используемые в проекте. Папки директории содержат игровой контент, такой как: 3D модели, текстуры, изображения, звуки и тд.

Файл .umap – это собственный формат Unreal Engine для карт уровней. В отличие от других типов файлов, .umap файл описывает физическое расположение объектов, освещение, механики, а также сцены, которые будут загружены во время игры. Папка «DerivedDataCache», представленная на рисунке 3.1, используется для хранения кэшированных данных, которые создаются во время работы редактора или сборки проекта. Эти данные ускоряют процесс разработки, поскольку содержат предварительно обработанную информацию, такую как текстуры, шейдеры, и другие игровые объекты, которые могут быть заново сгенерированы или пересчитаны при необходимости.

ExternalActors_	25.04.2025 20:21	Папка с файлами
ExternalObjects_	25.04.2025 20:21	Папка с файлами
AIAnim	25.04.2025 20:21	Папка с файлами
Anim	25.04.2025 20:21	Папка с файлами
AssetsvilleTown	25.04.2025 20:21	Папка с файлами
Basic_VFX	25.04.2025 20:21	Папка с файлами
Blueprints	25.04.2025 20:21	Папка с файлами
Collections	20.04.2025 1:44	Папка с файлами
Decal	25.04.2025 1:10	Папка с файлами
Developers	25.04.2025 20:21	Папка с файлами
FBX	25.04.2025 20:21	Папка с файлами
FirstPerson	25.04.2025 20:21	Папка с файлами
FirstPersonArms	25.04.2025 20:21	Папка с файлами
FPWeapon	25.04.2025 20:21	Папка с файлами
LevelPrototyping	25.04.2025 20:21	Папка с файлами
Levels	27.04.2025 20:37	Папка с файлами
NPC	25.04.2025 20:21	Папка с файлами
Pictures	25.04.2025 20:21	Папка с файлами
Weapon	25.04.2025 20:21	Папка с файлами
Widgets	30.04.2025 1:36	Папка с файлами
Menu_Map.umap	27.03.2025 17:19	Файл "UMAP" 2 КБ

Рисунок 3.9 – Содержание папки «Content»

Папка «Intermediate», представленная на рисунке 3.1, содержание которой представлено на рисунке 3.10, используется для хранения временных файлов, которые создаются в процессе разработки и компиляции, и которые могут быть удалены без ущерба для самого проекта. Эти файлы служат для упрощения и ускорения процессов сборки и производительности при работе с проектом. Папка «Build» директория «Intermediate» хранит собранные бинарные файлы и информацию о процессе сборки проекта. В этой директории могут находиться файлы, которые необходимы для компиляции и сборки игры, а также файлы, создаваемые при работе с Visual Studio или Xcode. Содержание папки «Build» представлено на рисунке 3.11.










	Build	25.04.2025 20:26	Папка с файлами	
	Config	25.04.2025 20:32	Папка с файлами	
	DatasmithContentTemp	25.04.2025 20:28	Папка с файлами	
	PipInstall	25.04.2025 20:32	Папка с файлами	
	ProjectFiles	25.04.2025 20:22	Папка с файлами	
	ReimportCache	25.04.2025 20:32	Папка с файлами	
	ShaderAutogen	25.04.2025 20:29	Папка с файлами	
	CachedAssetRegistry_0.bin	01.05.2025 1:32	Файл "BIN"	42 484 КБ
	TargetInfo.json	25.04.2025 20:22	Файл "JSON"	1 КБ

Рисунок 3.10 – Содержание папки «Intermediate»






	BuildRules	25.04.2025 20:22	Папка с файлами	
	BuildRulesProjects	25.04.2025 20:22	Папка с файлами	
	Win64	25.04.2025 20:24	Папка с файлами	
	SourceFileCache.bin	01.05.2025 1:29	Файл "BIN"	4 057 КБ
	XmlConfigCache.bin	25.04.2025 20:22	Файл "BIN"	1 КБ

Рисунок 3.11 – Содержание папки «Build»

Папка «Config» директория «Intermediate» предназначена для хранения временных файлов, связанных с конфигурационными данными проекта. Эти файлы генерируются при компиляции и сборке проекта и используются движком для временного кэширования конфигурационных настроек, а также для ускорения работы с проектом. Содержание папки «Config» представлено на рисунке 3.12.




	GameUserSettings.ini	25.04.2025 20:32	Параметры конф...	1 КБ
	LocalizationServiceSettings.ini	01.05.2025 1:31	Параметры конф...	1 КБ
	UnrealInsightsSettings.ini	01.05.2025 1:31	Параметры конф...	1 КБ

Рисунок 3.12 – Содержание папки «Config»

Папка «PipInstall» директория «Intermediate» предназначена для процесса установки зависимостей, связанных с Python, например, для создания кэша установленных пакетов, управления зависимостями или подготовки окружения для работы с проектом. Содержание папки «PipInstall» представлено на рисунке 3.13.





 Lib	25.04.2025 20:32	Папка с файлами	
 extra_urls.txt	01.05.2025 1:32	Текстовый докум...	0 КБ
 merged_requirements.in	01.05.2025 1:32	Файл "IN"	0 КБ
 pyreqs_plugins.list	01.05.2025 1:32	Файл "LIST"	0 КБ

Рисунок 3.13 – Содержание папки «PipInstall»

Папка «Lib» директория «PipInstall» используется для хранения библиотек или сторонних зависимостей, которые необходимы для работы проекта. Это могут быть как статические библиотеки, так и динамические библиотеки, используемые в C++ коде проекта. Файл «extra\_urls.txt» директория «PipInstall» – это текстовый файл, который может быть использован для хранения дополнительной информации о URL-ссылках в проекте. Файлы «merged\_requirements.in» и «pyreqs\_plugins.list» директория «PipInstall» – это файлы, которые являются частью процесса управления зависимостями, когда несколько различных файлов requirements.in или зависимостей сливаются в один файл, который затем используется для генерации окончательных зависимостей для установки с помощью pip. Папка «ProjectFiles» директория «Intermediate» используется для хранения файлов, которые могут быть связаны с конкретными инструментами, настройками среды разработки или проектами с нестандартной структурой. Содержание папки «ProjectFiles» представлено на рисунке 3.14.









 AutomationTool.csproj.References	25.04.2025 20:22	Файл "REFERENC...	11 КБ
 Coursework.vcxproj	25.04.2025 20:22	VC++ Project	163 КБ
 Coursework.vcxproj.filters	25.04.2025 20:22	VC++ Project Filte...	6 КБ
 Coursework.vcxproj.user	25.04.2025 20:22	Per-User Project O...	4 КБ
 UE5.vcxproj	25.04.2025 20:22	VC++ Project	9 405 КБ
 UE5.vcxproj.filters	25.04.2025 20:22	VC++ Project Filte...	18 525 КБ
 UE5.vcxproj.user	25.04.2025 20:22	Per-User Project O...	1 КБ
 UECommon.props	25.04.2025 20:22	Project Property File	3 КБ

Рисунок 3.14 – Содержание папки «ProjectFiles»

Папки «ReimportCache» и «ShaderAutogen» директория «Intermediate» связаны с оптимизацией работы с графическими ресурсами. Первая хранит кэшированные данные, ускоряя процесс повторного импорта ассетов, таких как текстуры и модели, при изменении их исходных файлов. Вторая содержит сгенерированные файлы,

которые отвечают за визуализацию материалов, освещения и эффектов на GPU(графический процессор). Файл «TargetInfo.json» директория «Intermediate» содержит информацию о целевых платформах, для которых будет собираться или разрабатываться проект. Этот файл используется для хранения настроек и параметров, связанных с платформой сборки, конфигурациями и целями проекта, такими как Windows, macOS, Linux, консоли или мобильные устройства.

Папка «Saved», представленная на рисунке 3.1, содержание которой представлено на рисунке 3.15, служит для хранения временных, промежуточных и кэшированных данных, которые генерируются во время работы с проектом. Эта папка не содержит исходных игровых объектов или кода, но она критически важна для процессов отладки, сборки, логирования и восстановления сессий.

Autosaves	25.04.2025 20:58	Папка с файлами	
Collections	25.04.2025 20:31	Папка с файлами	
Config	26.04.2025 7:10	Папка с файлами	
Logs	01.05.2025 1:29	Папка с файлами	
ShaderDebugInfo	25.04.2025 20:30	Папка с файлами	
SourceControl	25.04.2025 20:50	Папка с файлами	
UnrealBuildTool	25.04.2025 20:22	Папка с файлами	
AutoScreenshot.png	27.04.2025 20:37	Файл "PNG"	56 КБ

Рисунок 3.15 – Содержание папки «Saved»

Папка «Source», представленная на рисунке 3.1, содержание которой представлено на рисунке 3.16, – это одна из ключевых директорий в проекте Unreal Engine, особенно если проект использует C++. Она содержит исходный код логики проекта, включая ваши собственные игровые классы, модули, компоненты, и все, что связано с программной частью проекта.

Coursework	01.05.2025 1:28	Папка с файлами	
Coursework.Target.cs	13.03.2025 12:57	C# Source File	1 КБ
CourseworkEditor.Target.cs	13.03.2025 12:57	C# Source File	1 КБ

Рисунок 3.16 – Содержание папки «Source»

Папка «Coursework» директория «Source» это основной C++ модуль проекта Unreal Engine. Именно здесь хранятся файлы .cpp и .h, в которых реализуется основная игровая логика, классы и функции. Содержание папки «Coursework» представлено на рисунке 3.17.

Файл «Coursework.Target.cs» директория «Source» – это один из файлов конфигурации сборки в проекте Unreal Engine, написанный на C#.

Файл «CourseworkEditor.Target.cs» – это конфигурационный скрипт Unreal Build Tool, который описывает, как собирать версию проекта с поддержкой редактора Unreal Engine. Файл «.vsconfig», представленная на рисунке 3.1 – это конфигурационный файл Visual Studio, который описывает необходимые рабочие нагрузки и компоненты, нужные для сборки проекта. Он обычно используется в C++-



проектах, чтобы гарантировать, что у всех разработчиков установлены нужные инструменты в Visual Studio. Файл «Coursework.sln», представленная на рисунке 3.1 – это решение Visual Studio, созданное для проекта Unreal Engine. Он представляет собой основной файл, через который Visual Studio управляет проектами, их компиляцией и связями между ними.

* AnimClass.cpp	17.03.2025 16:37	C++ Source	1 KB
AnimClass.h	17.03.2025 16:37	C/C++ Header	1 KB
* AWeapon.cpp	25.04.2025 9:46	C++ Source	5 KB
AWeapon.h	24.04.2025 18:11	C/C++ Header	3 KB
Coursework.Build.cs	24.04.2025 2:57	C# Source File	1 KB
* Coursework.cpp	13.03.2025 12:57	C++ Source	1 KB
Coursework.h	13.03.2025 12:57	C/C++ Header	1 KB
* Coursework_Character.cpp	01.05.2025 1:28	C++ Source	14 KB
Coursework_Character.h	26.04.2025 18:39	C/C++ Header	4 KB
* Coursework_Game_Mode.cpp	27.04.2025 9:48	C++ Source	2 KB
Coursework_Game_Mode.h	27.04.2025 9:47	C/C++ Header	1 KB
* Coursework_MenuWidget.cpp	15.04.2025 7:08	C++ Source	3 KB
Coursework_MenuWidget.h	15.04.2025 6:52	C/C++ Header	2 KB
* Coursework_PickUpComponent.cpp	16.03.2025 23:52	C++ Source	2 KB
Coursework_PickUpComponent.h	16.03.2025 18:22	C/C++ Header	2 KB
* Coursework_Pistol_A.cpp	24.04.2025 13:01	C++ Source	1 KB
Coursework_Pistol_A.h	24.04.2025 12:48	C/C++ Header	1 KB
* Coursework_PreCharacter.cpp	25.04.2025 10:48	C++ Source	2 KB
Coursework_PreCharacter.h	25.04.2025 11:00	C/C++ Header	2 KB
* Coursework_Projectile.cpp	25.04.2025 11:29	C++ Source	3 KB
Coursework_Projectile.h	25.03.2025 2:38	C/C++ Header	2 KB
* CourseWork_Save_Data.cpp	27.04.2025 9:56	C++ Source	3 KB
CourseWork_Save_Data.h	27.04.2025 10:28	C/C++ Header	2 KB
* Coursework_Sniper_A.cpp	24.04.2025 13:01	C++ Source	1 KB
Coursework_Sniper_A.h	24.04.2025 12:48	C/C++ Header	1 KB
* Coursework_Spawn_Manger.cpp	20.04.2025 3:31	C++ Source	2 KB
Coursework_Spawn_Manger.h	23.03.2025 3:28	C/C++ Header	1 KB
* CourseworkWeaponComponent.cpp	17.03.2025 18:23	C++ Source	4 KB
CourseworkWeaponComponent.h	16.03.2025 18:32	C/C++ Header	2 KB
* Machine_Enemy.cpp	25.04.2025 11:25	C++ Source	3 KB
Machine_Enemy.h	24.04.2025 13:15	C/C++ Header	2 KB
* MyAnimClass.cpp	17.03.2025 18:34	C++ Source	1 KB
MyAnimClass.h	17.03.2025 18:49	C/C++ Header	1 KB

Рисунок 3.17 – Содержание папки «Coursework»

Файл «Coursework.uproject», представленная на рисунке 3.1 – это основной файл проекта Unreal Engine. Он является важнейшей частью любого проекта на UE, так как содержит всю информацию о структуре и конфигурации проекта, включая используемые модули, плагины, настройки и другие важные параметры. Этот файл имеет расширение .uproject и используется для открытия проекта в редакторе Unreal Engine.

## 3.2 Описание разработанных модулей

Модули в компьютерном программировании представляют собой независимые части программы, которые выполняют определенные функции. Они служат для организации кода, обеспечивая его модульность и повторное использование. Модули содержат в себе классы, функции, переменные и другие компоненты, необходимые для выполнения определенной задачи или реализации определенного функционала.

В процессе разработки программного продукта были созданы следующие классы и методы на языке C++.

Класс «AWeapon», методы:

- `AWeapon::AWeapon()` – Конструктор класса. В рамках конструктора создаются необходимые компоненты и задаются начальные значения параметров класса.

- `void AWeapon::Attach(USkeletalMeshComponent* arms_mesh)` – Система присоединения оружия к скелету персонажа.

- `void AWeapon::Detach()` – Система отсоединения оружия от скелета персонажа.

- `void AWeapon::Fire(ACharacter* actor)` – Создает объект пули, дает ему импульс и задает вектор направления.

Класс «Coursework», методы:

Отсутствуют.

Класс «ACoursework\_Character», методы:

- `ACoursework_Character::ACoursework_Character()` – Конструктор класса. В рамках конструктора создаются необходимые компоненты и задаются начальные значения параметров класса.

- `void ACoursework_Character::BeginPlay()` – Автоматически активируется в начале игры. Создаются необходимые компоненты и задаются значения параметров класса.

- `void ACoursework_Character::On_Overlap_Begin(UPrimitiveComponent* overlappedComp, AActor* otherActor, UPrimitiveComponent* otherComp, int32 otherBodyIndex, bool bFromSweep, const FHitResult& sweepResult)` – Действия при коллизии объекта класса с объектами любых классов, имеющих компонент, отслеживающий столкновения.

- `void ACoursework_Character::On_End_Overlap(UPrimitiveComponent* overlappedComponent, AActor* otherActor, UPrimitiveComponent* otherComp, int32 otherBodyIndex)` – Действия при окончании события коллизии.

- `void ACoursework_Character::Decal_Correction(AActor* item)` – Изменение расположения и поворота объекта, имеющего компонент Decal.

- `void ACoursework_Character::SetupPlayerInputComponent(UInputComponent* input_component)` – Настройка компонента ввода.

- `void ACoursework_Character::On_Action_Move(const FInputActionValue& value)` – Действия, исполняемые при реакции на определенный ввод. В данном случае на ввод, соответствующий параметру ввода клавиш, определенных в редакторе как Move-клавиши.

- `void ACoursework_Character::On_Action_Look(const FInputActionValue& value)` – Действия, исполняемые при реакции на определенный ввод. В данном случае на ввод, соответствующий параметру ввода клавиш, определенных в редакторе как Look-клавиши.

- `void ACoursework_Character::On_Action_Fire(const FInputActionValue& value)` – Действия, исполняемые при реакции на определенный ввод. В данном



случае на ввод, соответствующий параметру ввода клавиш, определенных в редакторе как Fire-клавиши.

- void ACoursework\_Character::On\_Action\_Use(const FInputActionValue& value) – Действия, исполняемые при реакции на определенный ввод. В данном случае на ввод, соответствующий параметру ввода клавиш, определенных в редакторе как Use-клавиши.

- void ACoursework\_Character::On\_Action\_Restor(const FInputActionValue& value) – Действия, исполняемые при реакции на определенный ввод. В данном случае на ввод, соответствующий параметру ввода клавиш, определенных в редакторе как Restor-клавиши.

- AActor\* ACoursework\_Character::Distance\_Count() – Подсчет расстояния от объекта класса до объекта, которые мы можем перебирать.

- void ACoursework\_Character::Pickup\_Weapon(AWeapon\* weapon) – Активация аналогичного метода у объекта оружия.

- void ACoursework\_Character::Destroy\_Self() – Уничтожение объекта класса.

- void ACoursework\_Character::Check\_Inputs() – Проверка на устройство ввода.

Класс «ACoursework\_Game\_Mode», методы:

- ACoursework\_Game\_Mode::ACoursework\_Game\_Mode() – Конструктор класса. В рамках конструктора создаются необходимые компоненты и задаются начальные значения параметров класса.

- void ACoursework\_Game\_Mode::Creat\_Second\_Player() – Создание объекта второго игрока.

Класс «UCoursework\_MenuWidget», методы:

- void UCoursework\_MenuWidget::NativeConstruct() – Конструктор виджета, унаследованный от базового класса пользовательского виджета.

- void UCoursework\_MenuWidget::Start() – Действия, при нажатии на кнопку Старт.

- void UCoursework\_MenuWidget::Option() – Действия, при нажатии на кнопку Опции.

- void UCoursework\_MenuWidget::Exit() – Действия, при нажатии на кнопку Выход.

Класс «UCoursework\_PickUpComponent», методы:

- UCoursework\_PickUpComponent::UCoursework\_PickUpComponent() – Конструктор класса. В рамках конструктора создаются необходимые компоненты и задаются начальные значения параметров класса.

- void UCoursework\_PickUpComponent::BeginPlay() – Автоматически активируется в начале игры. Создаются необходимые компоненты и задаются значения параметров класса.

- void UCoursework\_PickUpComponent::On\_Sphere\_Begin\_Overlap (UPrimitiveComponent \*overlapped\_component, AActor \*other\_actor, UPrimitiveComponent \*other\_comp, int other\_body\_index, bool from\_sweep, const FHitResult &sweep\_result) – Уведомляет объект, который может быть поднят, о начале коллизии.

Класс «ACoursework\_Pistol\_A», методы:

- ACoursework\_Pistol\_A::ACoursework\_Pistol\_A() – Конструктор объекта класса, унаследованный от базового класса оружия.

- void ACoursework\_Pistol\_A::BeginPlay() – Автоматически активируется в начале игры. Создаются необходимые компоненты и задаются значения параметров класса. Наследуется от базового класса.

Класс «ACoursework\_Sniper\_A», методы:

- ACoursework\_Sniper\_A::ACoursework\_Sniper\_A() – Конструктор объекта класса, унаследованный от базового класса оружия.

- void ACoursework\_Sniper\_A::BeginPlay() – Автоматически активируется в начале игры. Создаются необходимые компоненты и задаются значения параметров класса. Наследуется от базового класса.

Класс «ACoursework\_PreCharacter», методы:

- ACoursework\_PreCharacter::ACoursework\_PreCharacter() – Конструктор класса. В рамках конструктора создаются необходимые компоненты и задаются начальные значения параметров класса.

- void ACoursework\_PreCharacter::BeginPlay() – Автоматически активируется в начале игры. Создаются необходимые компоненты и задаются значения параметров класса.

- void ACoursework\_PreCharacter::Destroy\_Self()– Уничтожение объекта класса.

- void ACoursework\_PreCharacter::Pickup\_Weapon(AWeapon\* weapon) – Активация аналогичного метода у объекта оружия.

Класс «ACoursework\_Projectile», методы:

- ACoursework\_Projectile::ACoursework\_Projectile() – Конструктор класса. В рамках конструктора создаются необходимые компоненты и задаются начальные значения параметров класса.

- void ACoursework\_Projectile::On\_Hit(UPrimitiveComponent\* hit\_comp, AActor\* other\_actor, UPrimitiveComponent\* other\_comp, FVector normal\_impulse, const FHitResult& Hit) – Действия, при активации вхождения в коллизию объекта класса.

Класс «UCourseWork\_Save\_Data», методы:

- UCourseWork\_Save\_Data::UCourseWork\_Save\_Data() – Конструктор класса. В рамках конструктора создаются необходимые компоненты и задаются начальные значения параметров класса.

- void UCourseWork\_Save\_Data::Set\_Name\_Input(const FString& input) – Статический метод, задающий значение полю Name.

- FString UCourseWork\_Save\_Data::Get\_Name\_Input() – Статический метод, возвращающий значение поля Name.

- FString UCourseWork\_Save\_Data::Get\_DataForTable\_Input() – Статический метод, возвращающий значение поля DataForTable.

- void UCourseWork\_Save\_Data::Table\_Write(const FString& input) – Статический метод, реализующий бинарную запись значений в файл с расширением .dat.

- FString UCourseWork\_Save\_Data::Table\_Read() – Статический метод, реализующий чтение значений из файла с расширением .dat.

Класс «ACoursework\_Spawn\_Manger», методы:

- ACoursework\_Spawn\_Manger::ACoursework\_Spawn\_Manger() – Конструктор класса. В рамках конструктора создаются необходимые компоненты и задаются начальные значения параметров класса.

- void ACoursework\_Spawn\_Manger::BeginPlay() – Автоматически активируется в начале игры. Создаются необходимые компоненты и задаются значения параметров класса.

- void ACoursework\_Spawn\_Manger::Spawn() – Создает объекты класса AMachine\_Enemy и задает ему начальные параметры.

Класс «AMachine\_Enemy», методы:

- AMachine\_Enemy::AMachine\_Enemy() – Конструктор объекта класса, унаследованный от базового класса персонажа.

- void AMachine\_Enemy::BeginPlay() – Автоматически активируется в начале игры. Создаются необходимые компоненты и задаются значения параметров класса. Наследуется от базового класса.

- void AMachine\_Enemy::Spawn\_Niagara() – Создает объект анимации и проигрывает его при смерти.

- void AMachine\_Enemy::Destroy\_Self() – Уничтожение объекта класса. Наследуется от базового класса.

Класс «UMyAnimClass», методы:

- UMyAnimClass::UMyAnimClass() – Конструктор объекта класса. Задает флаг для анимации у игрового персонажа.

## 4 ТЕСТИРОВАНИЕ

Проверка качества разработанного программного средства выполнена в соответствии с ГОСТ 28195-99 «Оценка качества программных средств».

Тестирование проводилось на основе следующих критериев оценки:

Оценка языка программирования: разработка приложения выполнена с использованием языка низкого уровня C++.

Надежность: была проведена проверка правильности работы программы при вводе ошибочных данных, с созданием как позитивных, так и негативных тестов (таблицы 4.1 и 4.2). При вводе неверных значений программа не реагирует на ввод. Пользователь не может ввести недопустимые символы или оставить поле пустым.

Удобство применения: протестирована скорость и удобство загрузки и завершения работы программы. Программа запускается по двойному щелчку на ярлык и закрывается при нажатии кнопки закрытия. Также реализована возможность приостановки и повторного запуска программы без потери данных, так как вся необходимая информация сохраняется в файле .dat.

Эффективность: проведены тесты на определение требуемого объема памяти для хранения и работы программы. Результаты показали, что программа занимает 4 ГБ памяти для хранения.

Универсальность: программное обеспечение является многопользовательским, с разделением на первого и второго игроков. Программа предоставляет все необходимые функции и возможности для эффективного решения задач игрового приложения.

Функциональность:

Таблица 4.1 – Проверка работоспособности компьютерного приложения «FPS Project» (позитивное тестирование)

Основные функции программы	Ожидаемый результат	Фактический результат
1	2	3
Работоспособность и корректность исполнения действий кнопок главного меню	Нажатие на кнопки: Старт, Опции, Выбор двух игроков, Выход. Переход в соответствующие окна с рабочим функционалом	Выполнено успешно

Продолжение таблицы 4.1

1	2	3
Генерация игрового контента	Нажатие на кнопку Старт и загрузка игрового контента приложения	Выполнено успешно
Передвижение игрока по игровой локации	Нажатие кнопок, соответствующих кнопкам передвижения. Ответная реакция игрового аватара	Выполнено успешно
Стрельба из оружия	Нажатие кнопки, соответствующей кнопке стрельбы. Ответная реакция игрового аватара. Воспроизведение звукового эффекта	Выполнено успешно
Перезарядка оружия	Нажатие кнопки, соответствующей кнопке перезарядки. Ответная реакция игрового аватара	Выполнено успешно
Выполнение персонажем действия «Прыжок»	Нажатие кнопки, соответствующей кнопке прыжка. Ответная реакция игрового аватара	Выполнено успешно
Подбор боеприпасов и оружия	Нажатие кнопки, соответствующей кнопке подбора. Ответная реакция игрового аватара	Выполнено успешно
Получение урона	Попадание в игрового аватара снаряда. Уменьшение количества очков здоровья.	Выполнено успешно

Продолжение таблицы 4.1

1	2	3
Нанесение урона противнику	Попадание во вражеский объект снаряда. Уменьшение количества очков здоровья соперника	Выполнено успешно
Нанесение критического урона противнику	Проигрывание анимации падения противником. Проигрывание анимации взрыва противника. Уничтожение противника в рамках игрового мира	Выполнено успешно
Изменение громкости звукового ряда	Уменьшение или увеличение громкости звукового ряда. Дальнейшее воспроизведение на выбранной громкости	Выполнено успешно
Изменение звукового ряда	Ввод ссылки на звуковой файл, расположенный на диске. Дальнейшее воспроизведение выбранного звукового ряда	Выполнено успешно
Изменение яркости изображения	Уменьшение или увеличение яркости изображения. Дальнейшее отображение визуального контента с учетом установленного уровня яркости	Выполнено успешно

Продолжение таблицы 4.1

1	2	3
Завершение игровой сессии	Выход на экран завершения игры в результате получения критического урона игровым аватаром. Дальнейшая блокировка управления и отображение итоговой информации	Выполнено успешно
Просмотр таблицы лидеров	Нажатие на кнопку «Таблица лидеров» в интерфейсе. Отображение таблицы с результатами игроков	Выполнено успешно
Сохранение результатов игровой сессии	Нажатие на кнопку «Сохранить» в интерфейсе. Отображение поля ввода и кнопки подтверждения для сохранения введенной информации. Дальнейшая запись данных в файл сохранения	Выполнено успешно

2) Проверка работоспособности разработанного компьютерного приложения «FPS Project» проведена в виде негативного тестирования, представленного в таблице 4.2;

Таблица 4.2 – Проверка работоспособности компьютерного приложения «FPS Project» (негативное тестирование)

Основные функции программы	Ожидаемый результат	Фактический результат
1	2	3
Ввод пользователем пути к звуковому файлу	Ввод неверного, не существующего пути к файлу. Игнорирование программой ввода. Исходный звуковой файл не заменяется	Выполнено успешно
Ввод клавиши без назначения	Нажатие, во время игровой сессии, клавиши без назначения. Игнорирование программой ввода	Выполнено успешно
Некорректный ввод имени пользователя	Ввод в поле имени пользователя пустого значения. Игнорирование программой ввода. Отказ в сохранении данных	Выполнено успешно

В ходе тестирования компьютерного приложения «FPS Project» были обнаружены незначительные ошибки, которые впоследствии были устранены. Согласно требованиям ГОСТ 28195-99 «Оценка качества программных средств», разработанное приложение демонстрирует высокие показатели по всем приведённым критериям качества.



## 5 ПРИМЕНЕНИЕ

### 5.1 Назначение и условия применения

Компьютерное приложение «FPS Project» представляет собой кооперативную игру от первого лица, предоставляющий пользователям динамичный и захватывающий игровой опыт. Основной целью игры является создание увлекательной атмосферы командного взаимодействия, где игроки могут сражаться с противниками, управляемыми искусственным интеллектом, используя разнообразное оружие и тактические возможности.

Ключевые особенности игры включают в себя стрельбу из различных видов оружия, возможность подбирать патроны и менять вооружение, а также реалистичное поведение врагов с продвинутым искусственным интеллектом. В игре предусмотрены два режима: одиночная игра и локальный кооператив, что делает проект доступным как для индивидуального прохождения, так и для совместной игры с друзьями на одном устройстве.

Игра разработана с учётом простоты освоения и интереса для широкой аудитории – от начинающих игроков до опытных игроков. Благодаря интуитивному управлению, проработанному игровому процессу и кооперативным механикам, проект обеспечивает высокий уровень вовлечённости.

Кроме развлекательной ценности, игра может использоваться для развития навыков командной работы, реакции и стратегического мышления. Особенно интересным становится локальный кооператив, где игроки учатся действовать слаженно, деля ресурсы и прикрывая друг друга в бою.

Для корректной работы игры требуется персональный компьютер или ноутбук со следующими минимальными системными требованиями:

Оперативная память: 4 ГБ, Свободное дисковое пространство: 4 ГБ, Разрешение экрана: игра автоматически масштабируется под разрешение экрана,

Операционная система: Microsoft Windows 8/10/11. Компьютерное приложение «FPS Project» – это удобное, захватывающее и технически доступное игровое приложение, созданное для интересного взаимодействия с виртуальной игровой средой и совместного времяпрепровождения.

### 5.2 Руководство пользователя

Компьютерная игра «FPS Project» предназначена для создания захватывающего и интуитивно понятного взаимодействия между игроками и игровой системой. Основная цель игры – предоставить удобный способ управления игровым процессом и взаимодействия с окружающим миром через динамичные сражения и кооперативный игровой процесс.

Игровое приложение предлагает увлекательный опыт, где пользователи могут использовать разнообразное оружие, сражаться с врагами, управляемыми искусственным интеллектом, и развивать командные навыки. Благодаря наличию различных режимов, включая одиночную игру и локальный кооператив, игра становится доступной и интересной как для одиночных игроков, так и для групп игроков.

Начало игрового процесса становится доступным сразу после запуска игрового приложения. Для того чтобы начать игровую сессию, необходимо нажать на кнопку «Старт», расположенную справа от центра экрана. Для удобства навигации пользователя кнопка размещена в первую очередь. Интерфейс главного экрана изображен на рисунке 5.1.



Рисунок 5.1 – Главное компьютерного приложения

В главном меню приложения также предусмотрены функции открытия окна настроек, выбора двух игроков и выхода из приложения, которые активируются при нажатии соответствующих кнопок: «Опции», «Два игрока» и «Выход».

При нажатии на кнопку «Опции», расположенную ниже кнопки «Старт», открывается меню настройки приложения. В данном меню пользователь имеет возможность настроить громкость звукового ряда и яркость изображения с помощью ползунков, а также указать путь к звуковому ряду для воспроизведения через соответствующее поле ввода. Интерфейс меню настройки приложения и процесс ввода пути звукового файла представлены на изображении 5.2. Ниже кнопки «Опции» расположена кнопка «Два игрока». При нажатии на неё активируется режим локального кооператива, и при запуске игры создается второй аватар пользователя. Второму игроку автоматически присваивается режим ввода с геймпада.



Рисунок 5.2 – Процесс ввода пути звукового файла

При нажатии на кнопку «Выход» осуществляется автоматический выход из игрового приложения.

Окно игровой сессии представляет собой игровую локацию. В данном примере используется отладочный полигон. На экране отображается пользовательский интерфейс, включающий индикацию уровня здоровья и прицел. Окно игровой сессии представлено на изображении 5.3.

Во время активной игровой сессии пользователь взаимодействует с различными элементами виртуальной среды в рамках игрового процесса, ориентированного на динамичные боевые действия и тактическое взаимодействие. Игровой процесс начинается с перемещения игрока по виртуальной локации, в данном случае представляющей собой отладочный полигон. Пользователь может свободно исследовать территорию, занимать стратегические позиции, уклоняться от атак и искать ресурсы. Перемещение игроков по игровой локации представлено на рисунке 5.4.

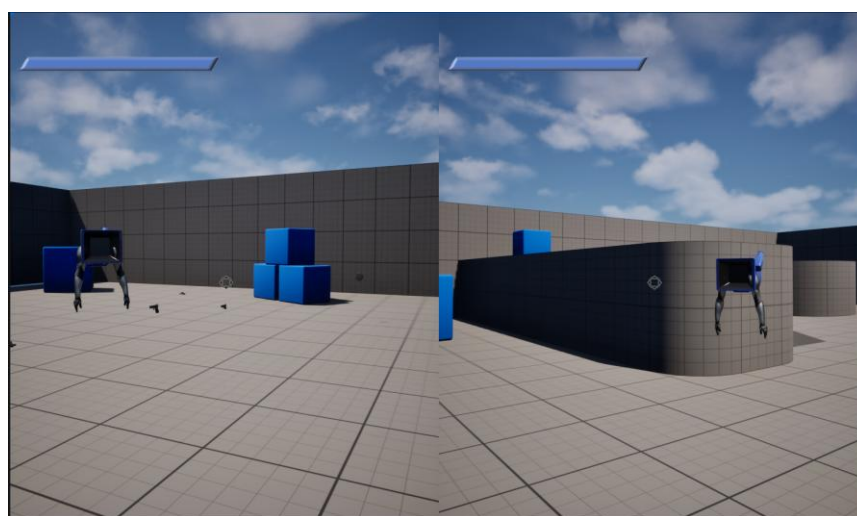


Рисунок 5.3 – Окно игровой сессии. Два игрока

Ключевым элементом игрового процесса является ведение огня по противникам, управляемым искусственным интеллектом. За создание на локации врагов отвечает отдельный менеджер. Пример противника с искусственным интеллектом представлен на рисунке 5.5. У игрока есть возможность победить врага, нанеся ему критические повреждения при помощи точных попаданий во врага. При достижении критического показателя здоровья, враг проиграет ряд анимаций. Первой анимацией идет падение врага, представленное на рисунке 5.6. Затем идет анимация взрыва, исходящего из, непосредственно, самого врага. Данная анимация представлена на рисунке 5.7. Далее персонаж врага удаляется со сцены. Игроку доступен арсенал различного стрелкового оружия, каждое из которых обладает уникальными характеристиками – уровнем урона и объёмом боезапаса. Пример подбора оружия и интерфейса при наличии у игрока боевого оружия, представлен на рисунке 5.8.

В процессе игры осуществляется постоянное взаимодействие с объектами, размещёнными на локации. Пользователь может подбирать боеприпасы и новое оружие, что способствует выживанию в боевых условиях и расширяет тактические возможности. Все действия сопровождаются визуальными индикаторами: на экране отображается пользовательский интерфейс, включающий прицел и индикатор уровня здоровья персонажа, позволяющий оперативно оценивать состояние игрового аватара. Изменение индикатора уровня здоровья, представлено на рисунке 5.9.

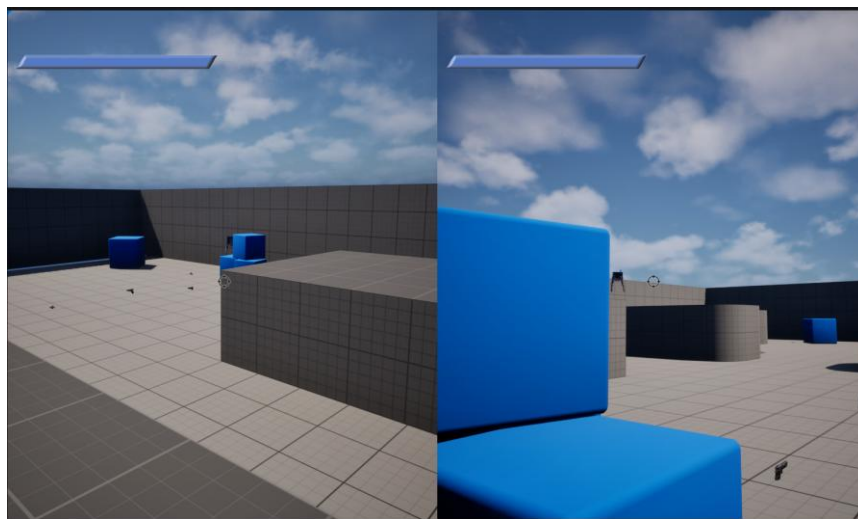


Рисунок 5.4 – Пример смены позиций игроков на игровой локации



Рисунок 5.5 – Пример противника с искусственным интеллектом



Рисунок 5.6 – Пример анимации падения противника

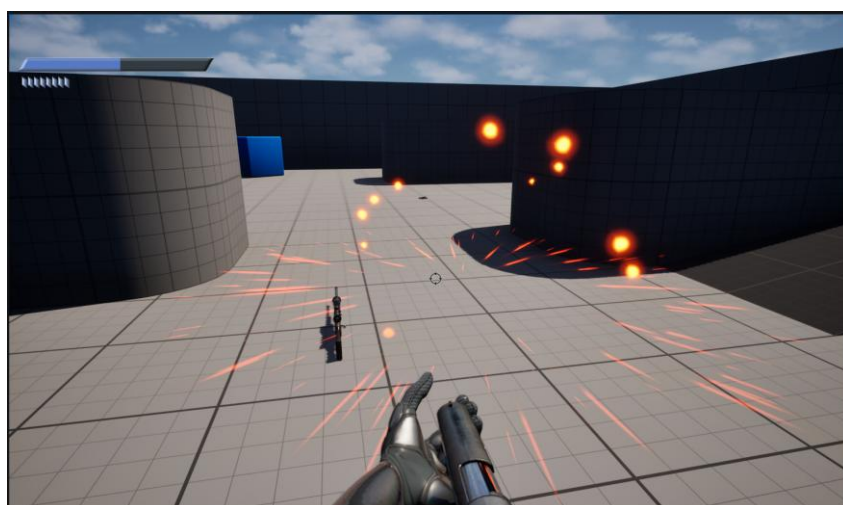


Рисунок 5.6 – Пример анимации взрыва противника



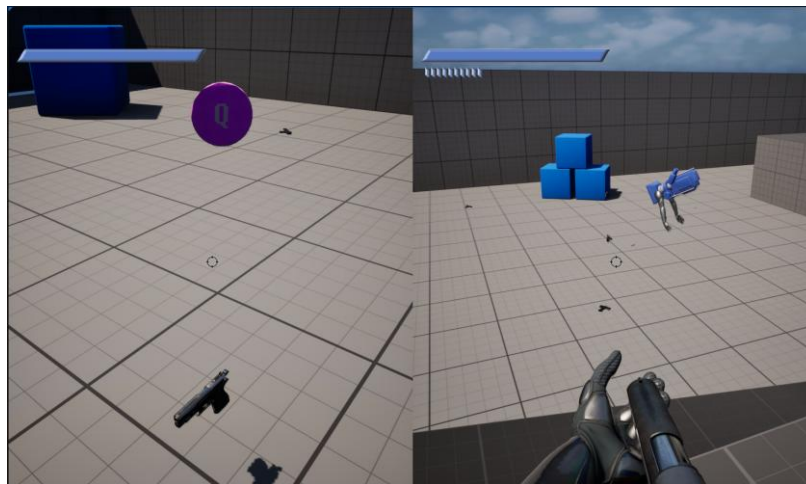


Рисунок 5.8 – Пример подбора оружия и смена графического интерфейса

В случае активации режима локального кооператива посредством кнопки «Два игрока», к игровому процессу присоединяется второй пользователь. Для него автоматически создаётся отдельный аватар, а управление предоставляется через геймпад. Пример одиночной игровой сессии представлен на рисунке 5.10. Таким образом, игровая сессия включает в себя комплекс активностей, направленных на взаимодействие с игровым пространством, сражение с противниками, управление ресурсами и реализацию кооперативного взаимодействия, что в совокупности обеспечивает насыщенный и увлекательный игровой опыт.

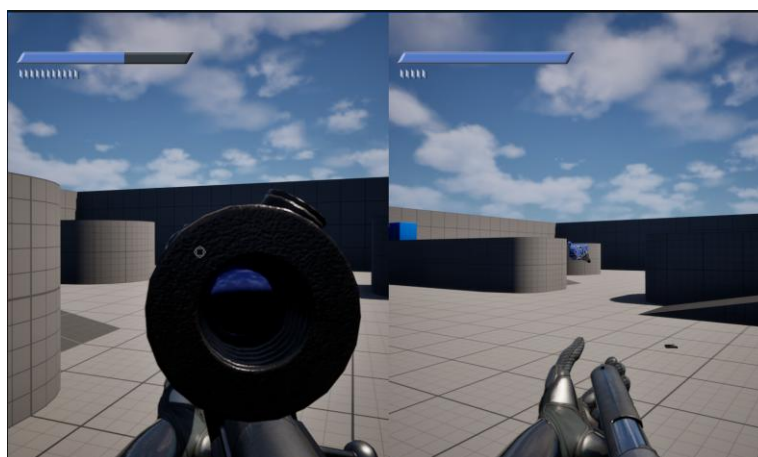


Рисунок 5.9 – Пример изменения индикатора уровня здоровья

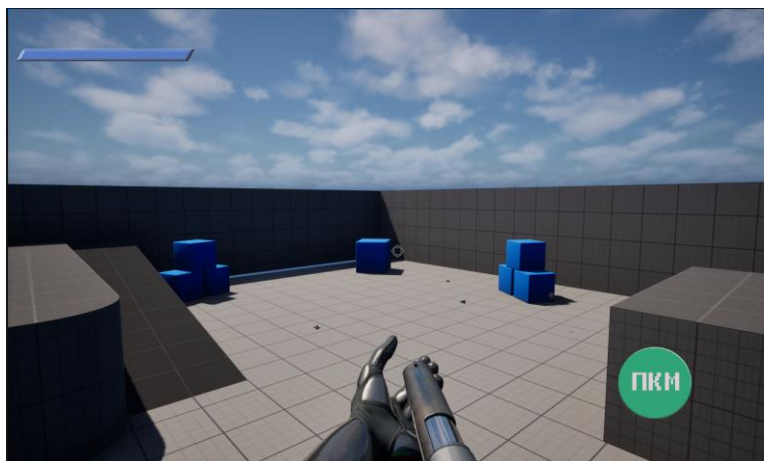


Рисунок 5.10 – Пример одиночной игровой сессии

После гибели игрового персонажа пользователю отображается специальный экран завершения сессии, предназначенный для подведения итогов текущей игры. Интерфейс окна завершения игровой сессии представлен на рисунке 5.11. Данный экран содержит информацию о достигнутом результате, включая количество уничтоженных противников, заработанные очки и возможность ввода имени игрока, для сохранения рекода.

Центральным элементом интерфейса является кнопка Перезапуска уровня. При ее нажатии игровая сессия перезапустится с текущими настройками. Ниже кнопки перезапуска уровня – кнопка сохранения прогресса. При ее нажатии появляется поле ввода и кнопка сохранения. Если пользователь вводит имя, которое уже присутствует в таблице лидеров, то результат перезаписывается. Если же имя встречается впервые, то в таблицу добавляются новые данные. После ввода имени игрок может подтвердить запись, после чего происходит сохранение данных в бинарный файл. Такой способ хранения обеспечивает быстрый доступ и целостность информации, а также исключает возможность её случайного редактирования вне приложения. Также в графическом интерфейсе присутствует кнопка Таблица рекордов. При ее нажатии открывается таблица рекордов. На рисунке 5.12, изображен результат нажатия кнопок Сохранения прогресса и Таблицы лидеров, выполнен ввод имени пользователя.

Кнопка Опции открывает окно настройки приложения, представленное на рисунке 1.5. Экран завершения содержит также кнопку выхода из приложения, что предоставляет пользователю возможность выйти из приложения. Таким образом, данный элемент интерфейса завершает игровой цикл.

Помимо тестового полигона, разработанное компьютерное приложение включает в себя ряд игровых карт, предназначенных для повышения интереса и разнообразия игрового опыта. Эти локации были созданы с целью предоставления пользователю более насыщенного и увлекательного геймплея, отличающегося по структуре, уровню сложности и стилю оформления.

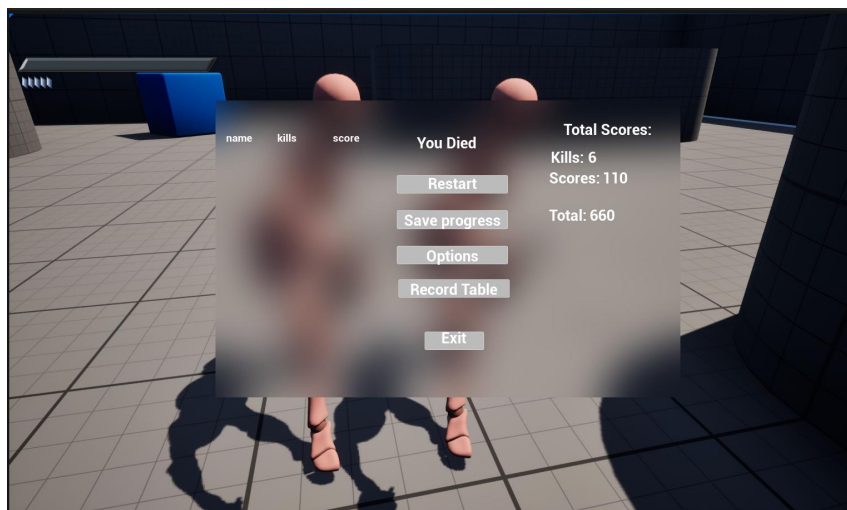


Рисунок 5.11 – Интерфейс окна завершения игровой сессии



Рисунок 5.12 – Результат нажатия кнопок Сохранения прогресса и Таблицы лидеров, выполнен ввод имени пользователя

Это позволяет варьировать стратегию ведения боя, стимулируя игрока адаптироваться к новым условиям и развивать тактическое мышление. Некоторые локации ориентированы на ближний бой в замкнутых пространствах, другие – на дальнобойные сражения на открытых территориях. Наличие нескольких карт существенно расширяет возможности приложения, способствует увеличению продолжительности игрового цикла и формированию устойчивого интереса пользователя к приложению. Варианты игровых локаций представлены на рисунках 5.13 и 5.14.

В приложении Б представлена более подробная информация об интерфейсе программы.





Рисунок 5.13 – Пример игровой локации. Кооперативный режим



Рисунок 5.14 – Пример игровой локации. Кооперативный режим

Компьютерное приложение «Игровое приложение FPS Project» обеспечивает плавное и интуитивно понятное взаимодействие за счёт тщательно продуманной визуальной и навигационной структуры, эффективно обслуживая как пользовательский, так и административный интерфейсы.

## ЗАКЛЮЧЕНИЕ

В процессе разработки компьютерного приложения «Игровое приложение FPS Project» была проведена значительная работа, направленная на достижение поставленных целей и реализацию функционально завершённого продукта.

На этапе постановки задачи была подробно изучена предметная область, а также проанализированы современные аналоги кооперативных шутеров от первого лица. В результате были выделены основные направления проектирования, включая проработку одиночного режима, локального кооператива и базового взаимодействия с элементами окружения.

На этапе проектирования была разработана логическая структура приложения, составлен алгоритм взаимодействия между компонентами игрового процесса, а также выбраны подходящие инструменты разработки. Особое внимание было уделено построению гибкой архитектуры и модульной структуры проекта, что упростило последующую реализацию и обеспечило расширяемость системы.

Реализация проекта включала создание различных игровых механик: стрельба по противникам с искусственным интеллектом, подбор боеприпасов и оружия, отображение пользовательского интерфейса с прицелом и индикатором здоровья. Дополнительно была реализована система регистрации результатов, позволяющая игроку после завершения сессии ввести имя и сохранить рекорд в таблицу, хранящуюся в бинарном формате. При включении режима кооператива автоматически создаётся второй аватар, управление которым осуществляется с помощью геймпада.

Важной частью проекта стало тестирование готового продукта, в ходе которого были выявлены и устранены ошибки, а также подтверждена стабильность и работоспособность всех реализованных функций. Прототип прошёл испытания в различных режимах, включая одиночную игру и кооператив, что подтвердило корректную реализацию игровых сценариев.

На заключительном этапе были определены условия эксплуатации, подготовлено краткое пользовательское руководство, а также оформлена таблица системных требований. Благодаря возможности масштабирования под различные разрешения экрана и невысоким системным требованиям, игра готова к практическому использованию на большинстве современных ПК.

В итоге разработанное приложение представляет собой полностью работоспособную игру в жанре кооперативных перестрелок от первого лица с базовой механикой стрельбы, управлением ресурсами и элементами взаимодействия между пользователями. Таким образом, основная цель курсового проекта по созданию игрового приложения была успешно достигнута.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] История игр от первого лица // dtf.ru[Электронный ресурс]. – 2021. Режим доступа: <https://dtf.ru/games/3197677-istoriya-zhanra-shutery-ot-pervogo-lica/>–Дата доступа: 27.04.2025.
- [2] История кооперативных режимов // stopgame.ru[Электронный ресурс]. – 2025. Режим доступа: [https://stopgame.ru/blogs/topic/107994/split\\_skrin\\_istoriya\\_rezhima/](https://stopgame.ru/blogs/topic/107994/split_skrin_istoriya_rezhima/)–Дата доступа: 28.04.2025.
- [3] История первых кооперативных игр от первого лица coop- // land.ru[Электронный ресурс]. – 2025. Режим доступа: <https://coop-land.ru/helpguides/blogs/19222-igry-kotorye-stoyali-u-istokov-multipleera.html/>–Дата доступа: 28.04.2025.
- [4] Поиск дополнительной информации по теме курсового проектирования // chatgpt.com[Электронный ресурс]. – 2025. Режим доступа: <https://chatgpt.com/>–Дата доступа: 01.05.2025.
- [5] Документация по игровому движку Unreal Engine 5 // dev.epicgames.com[Электронный ресурс]. – 2025. Режим доступа: <https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation/> Дата доступа: 01.05.2025.
- [6] Документация по C++ // learn.microsoft.com[Электронный ресурс]. – 2025. Режим доступа: <https://learn.microsoft.com/fr-fr/cpp/cpp/?view=msvc-170/> Дата доступа: 01.05.2025.

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Текст программы

#### AWeapon.h

```
#pragma once
```

```
#include "Coursework_Projectile.h"
#include "GameFramework/Actor.h"
#include "GameFramework/Character.h"
#include "Components/SphereComponent.h"
#include "Kismet/GameplayStatics.h"
#include "Engine/StaticMeshActor.h"
#include "Components/WidgetComponent.h"
#include "Blueprint/UserWidget.h"
#include "AWeapon.generated.h"
```

```
class ACoursework_Character;
class AMachine_Enemy;
UCLASS()
class COURSEWORK_API AWeapon : public AActor
{
```

```
    GENERATED_BODY()
```

```
public:
```

```
    AWeapon();
```

```
    void Attach(USkeletalMeshComponent* arms_mesh);
    void Detach();
```

```
    TArray<FString> Valid_Weapons;
```

```
    virtual float Get_HP_Rate()const { return 0; };
```

```
    UFUNCTION(BlueprintCallable, Category =
    "Weapon")virtual int Get_Bullet_Count()const { return
    Current_Bullet; };
```

```
    UFUNCTION(BlueprintCallable, Category =
    "Weapon")virtual void Set_Bullet_Count() { Current_Bullet =
    Max_Bullet; };
```

```
    UFUNCTION(BlueprintCallable, Category =
    "Weapon")virtual int Get_Bullet_Store()const { return
    Current_Store; };
```

```
    UFUNCTION(BlueprintCallable, Category =
    "Weapon")virtual void Set_Bullet_Store(int count) {
    Current_Store += count; };
```

```
    const int Max_Bullet = 12;
```

```
    const int Max_Store = 4;
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite)int
    Current_Bullet;
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite)int
    Current_Store;
```

```
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
    Category = Mesh) USkeletalMeshComponent* MeshInd;
```

```
    UPROPERTY(VisibleAnywhere, BlueprintReadWrite,
    Category = "UI") UWidgetComponent* Mark;
```

```
    UFUNCTION(BlueprintCallable) virtual UWidget*
    Get_Mark_BP_Char()const {
        return Mark->GetWidget();
    }
```

```
    UFUNCTION(BlueprintCallable, Category =
    "Weapon") void Fire(ACharacter* actor);
```

```
    UPROPERTY(EditDefaultsOnly, Category =
    Projectile) TSubclassOf<class ACoursework_Projectile>
    Projectile_Class;
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite,
    Category = Gameplay) USoundBase* Fire_Sound;
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite,
    Category = Gameplay) UAnimMontage* Fire_Animation;
```

```
    UPROPERTY(EditAnywhere, BlueprintReadWrite,
    Category = Gameplay) FVector Muzzle_Offset;
```

```
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
    Category = Mesh) USkeletalMeshComponent* Mesh;
```

```
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
    Category = Mesh) USkeletalMeshComponent* Perent_Mesh;
```

```
    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category
    = Mesh)USphereComponent* Sphere;};
```

#### AWeapon.cpp

```
#include "AWeapon.h"
```

```
AWeapon::AWeapon()
```

```
{
```

```
    Muzzle_Offset = FVector(100.0f, 0.0f, 10.0f);
```

```
    Current_Bullet = Max_Bullet;
```

```
    Current_Store = Max_Store;
```

```
    Mesh
```

```
    CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("W
    eapon_Mesh_C"));
```

```
    Mesh->SetupAttachment(RootComponent);
```

```
    Sphere
```

```
    CreateDefaultSubobject<USphereComponent>(TEXT("Weapon
    _Sphere_C"));
```

```
    Sphere->SetupAttachment(Mesh);
```

```
    Mark
```

```
    CreateDefaultSubobject<UWidgetComponent>(TEXT("Weapon
    Widget"));
```

```
    Mark->SetupAttachment(Mesh);
```

```
    Mark->SetDrawSize(FVector2D(500, 500));
```

```
    Mark-
    >SetCollisionEnabled(ECollisionEnabled::NoCollision);
```

```
}
```

```

void AWeapon::Attach(USkeletalMeshComponent* arms_mesh)
{
    USceneComponent* root_component =
    GetRootComponent();
    if (UPrimitiveComponent* prim_component =
    Cast<UPrimitiveComponent>(root_component))
    {
        prim_component-
    >SetSimulatePhysics(false);
        prim_component-
    >SetCollisionProfileName(UCollisionProfile::NoCollision_Profi
    leName);
    }

    FAttachmentTransformRules
    attachment_rules(EAttachmentRule::SnapToTarget, true);
    AttachToComponent(arms_mesh, attachment_rules,
    FName(TEXT("GripPoint")));

    Mark-
    >SetWidgetClass(LoadClass<UUserWidget>(nullptr,
    TEXT("/Game/Widgets/WB_Weapon.WB_Weapon_C")));
}

void AWeapon::Detach()
{
    USceneComponent* root_component =
    GetRootComponent();

    DetachFromActor(FDetachmentTransformRules::Keep
    WorldTransform);

    if (UPrimitiveComponent* prim_component =
    Cast<UPrimitiveComponent>(root_component))
    {
        Sphere-
    >SetCollisionEnabled(ECollisionEnabled::QueryOnly);
        prim_component->SetSimulatePhysics(true);
        prim_component-
    >SetCollisionProfileName(UCollisionProfile::PhysicsActor_Prof
    ileName);
        Mark->SetWidgetClass(nullptr);
    }
}

void AWeapon::Fire(ACharacter* actor)
{
    if (Current_Bullet > 0) {
        if (Projectile_Class != 0)
        {
            if (UWorld* world = GetWorld())
            {
                FRotator spawn_rotation;

                if (APlayerController*
                player_controller = Cast<APlayerController>(actor-
                >GetController())) {
                    spawn_rotation
                =
                player_controller->PlayerCameraManager-
                >GetCameraRotation();
            }
            else {
                spawn_rotation
                = actor->GetViewRotation();
            }
        }
    }
}

```

```

Current_Bullet
+= 1;
    }
    FVector spawn_location
    =
    GetActorLocation() +
    spawn_rotation.RotateVector(Muzzle_Offset);
    FActorSpawnParameters
    actor_spawn_params;

    actor_spawn_params.SpawnCollisionHandlingOverrid
    e
    =
    ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButAl
    waysSpawn;

    ACoursework_Projectile*
    pr
    =
    >SpawnActor<ACoursework_Projectile>(Projectile_Class,
    spawn_location, spawn_rotation, actor_spawn_params); //
    Спавним снаряд на дуле

    pr->HP_Rate
    =
    Get_HP_Rate();

    Current_Bullet -= 1;
}

};
if (Fire_Sound != 0)

    UGameplayStatics::PlaySoundAtLocation(this,
    Fire_Sound, GetActorLocation()); // Пытаемся проиграть звук
    выстрела, если он указан
}

}

```

### Coursework.cpp

```

#include "Coursework.h"
#include "Modules/ModuleManager.h"

IMPLEMENT_PRIMARY_GAME_MODULE(
    FDefaultGameModuleImpl, Coursework, "Coursework");

```

### ACoursework\_Character.h

```

#pragma once

#include "CoreMinimal.h"
#include "Coursework_PreCharacter.h"
#include "InputAction.h"
#include "Machine_Enemy.h"
#include <EnhancedInputSubsystemInterface.h>
#include "MyAnimClass.h"
#include "Components/DecalComponent.h"
#include "Coursework_Character.generated.h"

```

```

class UInputComponent;
class USceneComponent;
class UCameraComponent;
class UWidgetComponent;
class UAnimMontage;
class USoundBase;
class UMyAnimClass;

```

```

UCLASS(config = Game)
class ACoursework_Character : public
ACoursework_PreCharacter

```

```

{
    GENERATED_BODY()

public:
    ACoursework_Character();
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
    UMyAnimClass* Current_Anim;
    UPROPERTY(VisibleAnywhere, BlueprintReadWrite)
    bool Is_GamePad;

    AWeapon* Get_Current_Weapon()const override {
    return Weapon; };
    void Set_Current_Weapon(AWeapon* v) override {
    Weapon = v; };

    void Destroy_Self()override;
    UPROPERTY(EditAnywhere, Category = "UI")
    UWidgetComponent* HealthBar_Widget;
    UPROPERTY(EditAnywhere, Category = "UI")
    UWidgetComponent* Died_Screen_Widget;

protected:
    virtual void BeginPlay();
    virtual void
    SetupPlayerInputComponent(UInputComponent*
    input_component);

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
    TArray<AActor*> Interactable_Actors;
private:
    void On_Action_Move(const FInputActionValue&
    value);
    void On_Action_Look(const FInputActionValue&
    value);
    void On_Action_Fire(const FInputActionValue&
    value);
    void On_Action_Use(const FInputActionValue&
    value);
    void On_Action_Restor(const FInputActionValue&
    value);
    void Pickup_Weapon(AWeapon* weapon);
    void Check_Inputs();

    UFUNCTION() void
    On_Overlap_Begin(UPrimitiveComponent* OverlappedComp,
    AActor* OtherActor, UPrimitiveComponent* OtherComp,
    int32 OtherBodyIndex, bool bFromSweep,
    const FHitResult& SweepResult);
    UFUNCTION() void
    On_End_Overlap(UPrimitiveComponent*
    OverlappedComponent, AActor* OtherActor,
    UPrimitiveComponent* OtherComp, int32
    OtherBodyIndex);

    void Decal_Correction(AActor* item);
    AActor* Distance_Count();

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
    Category = Camera, meta = (AllowPrivateAccess = "true"))
    UCameraComponent* FirstPersonCameraComponent;
    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputMappingContext* DefaultMappingContext;

```

```

    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* Action_Jump;
    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* Action_Move;
    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* Action_Look;
    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* Action_Fire;
    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* Action_Use;
    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* Action_Restor;

    UPROPERTY(VisibleAnywhere)UDecalComponent*
    DecalQ;
    UPROPERTY(VisibleAnywhere)UDecalComponent*
    DecalB;

    UPROPERTY(VisibleAnywhere)UClass*
    DecalInstance;
    AStaticMeshActor* DecalActor;

    AWeapon* Weapon;
};

```

#### **ACoursework\_Character.cpp**

```

#include "Coursework_Character.h"
#include "Coursework_Projectile.h"
#include "AWeapon.h"
#include "Animation/AnimInstance.h"
#include "Camera/CameraComponent.h"
#include "Components/CapsuleComponent.h"
#include "Components/WidgetComponent.h"
#include "Blueprint/UserWidget.h"
#include "EnhancedInputComponent.h"
#include "EnhancedInputSubsystems.h"
#include "GameFramework/InputDeviceSubsystem.h"

ACoursework_Character::ACoursework_Character()
{
    UCourseWork_Save_Data::PlayerCount++;
    UE_LOG(LogTemp, Warning, TEXT("%i"),
    UCourseWork_Save_Data::PlayerCount);

    if (UCourseWork_Save_Data::PlayerCount == 5) {
        Is_GamePad = false;
    }
    else {
        Is_GamePad = true;
    }

    Health_Bar = 100;
    UE_LOG(LogTemp, Warning, TEXT("%f"),
    Health_Bar);
    UCapsuleComponent* Capsule =
    GetCapsuleComponent();
    GetCapsuleComponent()->InitCapsuleSize(55.0f,
    96.0f);
    Capsule-
    >OnComponentBeginOverlap.AddDynamic(this,
    &ThisClass::On_Overlap_Begin);

```

```

Capsule-
>OnComponentEndOverlap.AddDynamic(this,
&ThisClass::On_End_Overlap);

FirstPersonCameraComponent =
CreateDefaultSubobject<UCameraComponent>(TEXT("FirstPer
sonCamera"));
FirstPersonCameraComponent-
>SetupAttachment(GetCapsuleComponent());
FirstPersonCameraComponent-
>SetRelativeLocation(FVector(-10.0f, 0.0f, 60.0f));
FirstPersonCameraComponent-
>bUsePawnControlRotation = true;

Mesh_1P =
CreateDefaultSubobject<USkeletalMeshComponent>(TEXT("C
haracterMesh1P"));
Mesh_1P->SetOnlyOwnerSee(true);
Mesh_1P-
>SetupAttachment(FirstPersonCameraComponent);
Mesh_1P->bCastDynamicShadow = false;
Mesh_1P->CastShadow = false;
Mesh_1P->SetRelativeLocation(FVector(-30.0f, 0.0f, -
150.0f));

HealthBar_Widget =
CreateDefaultSubobject<UWidgetComponent>(TEXT("Charact
erWidget"));
HealthBar_Widget-
>SetupAttachment(RootComponent);
HealthBar_Widget->SetDrawSize(FVector2D(500,
500));
HealthBar_Widget-
>SetWidgetSpace(EWidgetSpace::World);
HealthBar_Widget-
>SetCollisionEnabled(ECollisionEnabled::NoCollision);
HealthBar_Widget->SetOnlyOwnerSee(true);
HealthBar_Widget-
>SetWidgetClass(LoadClass<UUserWidget>(nullptr,
TEXT("/Game/Widgets/WB_HealthBar.WB_HealthBar_C")));

Died_Screen_Widget =
CreateDefaultSubobject<UWidgetComponent>(TEXT("Charact
erWidget2"));
Died_Screen_Widget-
>SetupAttachment(RootComponent);
Died_Screen_Widget->SetDrawSize(FVector2D(500,
500));
Died_Screen_Widget-
>SetWidgetSpace(EWidgetSpace::World);
Died_Screen_Widget-
>SetCollisionEnabled(ECollisionEnabled::NoCollision);
Died_Screen_Widget->SetOnlyOwnerSee(true);
}

void ACoursework_Character::BeginPlay(){
Super::BeginPlay();
DecalInstance = LoadClass<AStaticMeshActor>(nullptr,
TEXT("/Game/FBX/ButtonAct_Blueprint.ButtonAct_Blueprint_
C"));

Current_Anim = Cast<UMyAnimClass>(Mesh_1P-
>GetAnimInstance());
if (APlayerController* player_controller =
Cast<APlayerController>(Controller))
{

```

```

player_controller-
>SetInputMode(FInputModeGameOnly());
player_controller-
>SetShowMouseCursor(false);
if (UEnhancedInputLocalPlayerSubsystem*
input_subsystem =
ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubs
ystem>(player_controller->GetLocalPlayer()))
input_subsystem-
>AddMappingContext(DefaultMappingContext, 0);
}

UUserWidget* Widget_Instance =
Cast<UUserWidget>(HealthBar_Widget->GetWidget());

if (DecalInstance) {
DecalActor = GetWorld()-
>SpawnActor<AStaticMeshActor>(DecalInstance);
DecalActor->SetActorHiddenInGame(true);
if (DecalActor) {
UE_LOG(LogTemp, Warning,
TEXT("DECAL
SPAWNED_____
"));
TArray<UDecalComponent*>
Decals;
DecalActor-
>GetComponents<UDecalComponent>(Decals);
if (Decals.Num() == 2) {
DecalB = Decals[0];
DecalQ = Decals[1];
}
}
else {
UE_LOG(LogTemp, Warning,
TEXT("DECAL
ERROR_____
"));
}
}
else UE_LOG(LogTemp, Warning, TEXT("DECAL
INSTANSE2
ERROR_____
"));
}

void
ACoursework_Character::On_Overlap_Begin(UPrimitiveCompo
nent* overlappedComp, AActor* otherActor,
UPrimitiveComponent* otherComp,
int32 otherBodyIndex, bool bFromSweep, const
FHitResult& sweepResult)
{
Check_Inputs();
if (otherActor && (otherActor != this) && otherComp)
{
Interactable_Actors.AddUnique(otherActor);
UE_LOG(LogTemp, Warning, TEXT("add
to interactable %s"), *otherActor->GetName());
}
Decal_Correction(Interactable_Actors[0]);
DecalActor->SetActorHiddenInGame(false);
}

```

```

void
ACoursework_Character::On_End_Overlap(UPrimitiveComponent* overlappedComponent, AActor* otherActor,
UPrimitiveComponent* otherComp, int32
otherBodyIndex)
{
    if (Interactable_Actors.Contains(otherActor) &&
        Interactable_Actors.Remove(otherActor);
        UE_LOG(LogTemp, Warning,
TEXT("Removed interactable %s"), *otherActor->GetName());
    }
    if (Interactable_Actors.Num() == 0) {
        DecalActor->SetActorHiddenInGame(true);
    }
    else {
        Decal_Correction(Interactable_Actors[Interactable_Actors.Num(
) - 1]);
    }
}

void ACoursework_Character::Decal_Correction(AActor* item)
{
    DecalActor->SetActorLocation(item-
>GetActorLocation() + FVector(0, 0, 100));
    DecalActor->SetActorRotation(this-
>GetActorRotation() + FRotator(0, 90, 0));
}

void
ACoursework_Character::SetupPlayerInputComponent(UInputC
omponent* input_component)
{
    if (UEnhancedInputComponent*
EnhancedInputComponent =
CastChecked<UEnhancedInputComponent>(input_component))
    {
        EnhancedInputComponent-
>BindAction(Action_Jump, ETriggerEvent::Triggered, this,
&ACharacter::Jump);
        EnhancedInputComponent-
>BindAction(Action_Jump, ETriggerEvent::Completed, this,
&ACharacter::StopJumping);
        EnhancedInputComponent-
>BindAction(Action_Move, ETriggerEvent::Triggered, this,
&ACoursework_Character::On_Action_Move);
        EnhancedInputComponent-
>BindAction(Action_Look, ETriggerEvent::Triggered, this,
&ACoursework_Character::On_Action_Look);
        EnhancedInputComponent-
>BindAction(Action_Fire, ETriggerEvent::Triggered, this,
&ACoursework_Character::On_Action_Fire);
        EnhancedInputComponent-
>BindAction(Action_Use, ETriggerEvent::Triggered, this,
&ACoursework_Character::On_Action_Use);
        EnhancedInputComponent-
>BindAction(Action_Restor, ETriggerEvent::Triggered, this,
&ACoursework_Character::On_Action_Restor);
    }
}

void ACoursework_Character::On_Action_Move(const
FInputActionValue& value)
{

```

```

FVector2D movement_vector =
value.Get<FVector2D>();

if (Controller != 0)
{
    AddMovementInput(GetActorForwardVector(),
movement_vector.Y);
    AddMovementInput(GetActorRightVector(),
movement_vector.X);
}

void ACoursework_Character::On_Action_Look(const
FInputActionValue& value)
{
    FVector2D look_axis_vector =
value.Get<FVector2D>();

    if (Controller != 0)
    {
        AddControllerYawInput(look_axis_vector.X);

        AddControllerPitchInput(look_axis_vector.Y);
    }
}

void ACoursework_Character::On_Action_Fire(const
FInputActionValue& value)
{
    Check_Inputs();
    UE_LOG(LogTemp, Warning, TEXT("-----
-----FIRE DETECTED-----"));

    if (Get_Current_Weapon() == 0 || !Current_Anim-
>HasRifle) {
        return;
    }
    Get_Current_Weapon()->Fire(this);
    if (UAnimInstance* anim_instance = Mesh_1P-
>GetAnimInstance())
        anim_instance-
>Montage_Play(Get_Current_Weapon()->Fire_Animation, 1.0f);
    UCourseWork_Save_Data::Score -= 10;
    UE_LOG(LogTemp, Warning,
TEXT("-----%i-----"),
UCourseWork_Save_Data::Score);
}

void ACoursework_Character::On_Action_Use(const
FInputActionValue& value)
{
    if (Interactable_Actors.Num() == 0) {
        return;
    }

    AActor* item = Distance_Count();
    if (AWeapon* weapon = Cast<AWeapon>(item))
        Pickup_Weapon(weapon);

    if (item->ActorHasTag("pick_up")) {
        if (Current_Anim->HasRifle) {
            if (Get_Current_Weapon()-
>Current_Store < Get_Current_Weapon()->Max_Store)

```



```

        Get_Current_Weapon()-
>Set_Bullet_Store(1);
        UE_LOG(LogTemp, Warning,
TEXT("-----PICK_UP--%i-----
----"), Get_Current_Weapon()->Get_Bullet_Store());
        item->Destroy();
    }
    else
    {
        UE_LOG(LogTemp, Warning, TEXT("-----
-----ERROR PICK-----"));
    }
}

void ACoursework_Character::On_Action_Restor(const
FInputActionValue& value) {

    if (Get_Current_Weapon() != 0) {
        AWeapon* weapon =
Get_Current_Weapon();
        if (weapon->Get_Bullet_Store() != 0) {
            weapon->Set_Bullet_Count();
            weapon->Set_Bullet_Store(-1);
            UE_LOG(LogTemp, Warning,
TEXT("-----TAB DETECTED---%i-----
-----"), weapon->Get_Bullet_Store());
            UCourseWork_Save_Data::Score -
= 10;
        }
    }
}

AActor* ACoursework_Character::Distance_Count() {
    int i;
    double distance, min_distance;
    AActor* item, * curr_item;
    FVector player_pos, item_pos;

    if (Interactable_Actors.Num() == 1)
    {
        item = Interactable_Actors[0];
    }
    else
    {
        player_pos = GetActorLocation();

        for (i = 0; i < Interactable_Actors.Num(); i++)
        {
            curr_item = Interactable_Actors[i];

            item_pos = curr_item-
>GetActorLocation();

            distance =
FVector::Distance(player_pos, item_pos);

            if (i == 0 || distance < min_distance)
            {
                min_distance = distance;
                item = curr_item;
            }
        }
        Interactable_Actors.Remove(item);
        Decal_Correction(item);
    }
}

```

```

    }
    return item;
}

void ACoursework_Character::Pickup_Weapon(AWeapon*
weapon)
{
    if (weapon == 0)
        return;
    if (Get_Current_Weapon() != 0) {
        Get_Current_Weapon()->Detach();
    }
    ACoursework_PreCharacter::Pickup_Weapon(weapon)
;
    Current_Anim->HasRifle = true;
    DecalActor->SetActorHiddenInGame(true);

    UCourseWork_Save_Data::Score += 20;
}

void ACoursework_Character::Destroy_Self() {

    ACoursework_PreCharacter::Destroy_Self();
    SetLifeSpan(0.0f);
    Current_Anim->HasRifle = false;

    Died_Screen_Widget-
>SetWidgetClass(LoadClass<UUserWidget>(nullptr,
TEXT("/Game/Widgets/WB_Death_Screen.WB_Death_Screen_
C")));

    UUserWidget* Widget_Instance1 =
Cast<UUserWidget>(Died_Screen_Widget->GetWidget());
    Widget_Instance1->AddToPlayerScreen();

    if (APlayerController* player_controller =
Cast<APlayerController>(Controller))
    {
        player_controller-
>SetIgnoreMoveInput(true);
        player_controller-
>SetIgnoreLookInput(true);
        player_controller-
>SetShowMouseCursor(true);

        player_controller-
>SetInputMode(FInputModeUIOnly());
    }
}

void ACoursework_Character::Check_Inputs() {

    UInputDeviceSubsystem* InputDeviceSubsystem =
GetGameInstance()->GetEngine()-
>GetEngineSubsystem<UInputDeviceSubsystem>();
    if (UCourseWork_Save_Data::PlayerCount == 5) {
        if (InputDeviceSubsystem-
>GetMostRecentlyUsedHardwareDevice(GetPlatformUserId()).
PrimaryDeviceType ==
EHardwareDevicePrimaryType::Gamepad) {
            Is_GamePad = true;
        }
        else {
            Is_GamePad = false;
        }
    }
}

```

```

    }
    if(DecalB)
        DecalB->SetVisibility(Is_GamePad);
}

```

### **ACoursework\_Game\_Mode.h**

```
#pragma once
```

```

#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "InputMappingContext.h"
#include "Coursework_Game_Mode.generated.h"

```

```

UCLASS(minimalapi)
class ACoursework_Game_Mode : public AGameModeBase
{
    GENERATED_BODY()

```

```
public:
```

```
    ACoursework_Game_Mode();
```

```

    UFUNCTION(BlueprintCallable, Category =
"UTIL")void Creat_Second_Player();
    UPROPERTY(EditAnywhere, BlueprintReadOnly,
Category = Input, meta = (AllowPrivateAccess = "true"))
UInputMappingContext* DefaultMappingContext;
    UPROPERTY(EditDefaultsOnly, Category = "Input")
TSoftObjectPtr<UInputMappingContext>
DefaultMappingContextRef;
};

```

### **ACoursework\_Game\_Mode.cpp**

```

#include "Coursework_Game_Mode.h"
#include "Coursework_Character.h"
#include "UObject/ConstructorHelpers.h"
#include "Engine/GameInstance.h"
#include "CourseWork_Save_Data.h"
#include "CoreMinimal.h"
#include "Subsystems/LocalPlayerSubsystem.h"
#include "EnhancedInputSubsystems.h"
#include "InputAction.h"

```

```

ACoursework_Game_Mode::ACoursework_Game_Mode()
: Super()
{
    static ConstructorHelpers::FClassFinder<APawn>
PlayerPawnClassFinder(TEXT("/Game/FirstPerson/Blueprints/B
P_FirstPersonCharacter"));
    DefaultPawnClass = PlayerPawnClassFinder.Class;
    ;
    UCourseWork_Save_Data::Ser_PlayerCount();
}
void ACoursework_Game_Mode::Creat_Second_Player()
{
    FString error;
    ULocalPlayer* NewPlayer =GetGameInstance()-
>CreateLocalPlayer(5, error, true);
    DefaultMappingContextRef =
TSoftObjectPtr<UInputMappingContext>(FSoftObjectPath(TEXT(
"/Game/FirstPerson/Input/IMC_Default.IMC_Default")));

    UInputMappingContext* LoadedContext =
DefaultMappingContextRef.LoadSynchronous();
    NewPlayer-
>GetSubsystem<UEnhancedInputLocalPlayerSubsystem>()-
>AddMappingContext(LoadedContext,0);

```

```
}
```

### **UCoursework\_MenuWidget.h**

```
#pragma once
```

```

#include "CoreMinimal.h"
#include "Blueprint/UserWidget.h"
#include "Components/Button.h"
#include "Components/CanvasPanel.h"
#include "Components/TextBlock.h"
#include "Coursework_MenuWidget.generated.h"

```

```

UCLASS()
class COURSEWORK_API UCoursework_MenuWidget : public
UUserWidget
{
    GENERATED_BODY()

```

```
public:
```

```
    virtual void NativeConstruct() override;
```

```

    UPROPERTY(meta = (BindWidget)) UCanvasPanel*
Root_Canvas;
    UPROPERTY(EditAnywhere, Category = "Level")
TSoftObjectPtr<UWorld> TargetLevel;
    UPROPERTY(EditAnywhere, Category = "Level")
UObject* WorldContextObject;

```

```

    UPROPERTY(meta = (BindWidget)) UButton*
First_Button;
    UPROPERTY(meta = (BindWidget)) UButton*
Second_Button;
    UPROPERTY(meta = (BindWidget)) UButton*
Third_Button;

```

```

    UPROPERTY(meta = (BindWidget)) UTextBlock*
FB_Text;
    UPROPERTY(meta = (BindWidget)) UTextBlock*
SB_Text;
    UPROPERTY(meta = (BindWidget)) UTextBlock*
TB_Text;

```

```

UFUNCTION(BlueprintCallable)void Start( );
UFUNCTION(BlueprintCallable)void Option( );

```

```

UFUNCTION(BlueprintCallable)void Exit( );
TArray<UTextBlock*> Texts;
TArray<UButton*> Buttons;
TArray<FText> Button_Text;

```

```
TArray<FScriptDelegate> Button_Delegates;
```

```
};
```

### **UCoursework\_MenuWidget.cpp**

```

#include "Coursework_MenuWidget.h"
#include "Components/CanvasPanelSlot.h"
#include "Blueprint/WidgetTree.h"
#include "Kismet/GameplayStatics.h"
#include "Engine/World.h"
#include "Engine/Engine.h"
#include "Kismet/KismetSystemLibrary.h"

```

```

void UCoursework_MenuWidget::NativeConstruct() {
    Super::NativeConstruct();

```

```
    Texts={ FB_Text,SB_Text,TB_Text };

```

```

        Buttons={ First_Button,Second_Button,Third_Button
};
        Button_Text =
        {
                FText::FromString(TEXT("Start")),
                FText::FromString(TEXT("Option")),
                FText::FromString(TEXT("Exit"))
        };

        FScriptDelegate Start_Delegate;
        Start_Delegate.BindUFunction(this, FName("Start"));
        FScriptDelegate Option_Delegate;
        Option_Delegate.BindUFunction(this,
FName("Option"));
        FScriptDelegate Exit_Delegate;
        Exit_Delegate.BindUFunction(this, FName("Exit"));

        Button_Delegates = {
                Start_Delegate,
                Option_Delegate,
                Exit_Delegate
        };

        Root_Canvas = NewObject<UCanvasPanel>(this);
        WidgetTree->RootWidget = Root_Canvas;

        if (Root_Canvas) {
                if (Texts.Num() == Buttons.Num()) {

                        for (int i = 0; i < Texts.Num(); i++)

                                Buttons[i] =
NewObject<UButton>(this);
                                Texts[i] =
NewObject<UTextBlock>(this);
                                Texts[i]-
>SetText(Button_Text[i]);
                                UCanvasPanelSlot*
ButtonSlot = Cast<UCanvasPanelSlot>(Buttons[i]->Slot);
                                if (ButtonSlot)
                                {
                                        ButtonSlot-
>SetPosition(FVector2D(100.f, 100.f + i * 60.f));
                                        ButtonSlot-
>SetSize(FVector2D(200.f, 50.f));
                                        Root_Canvas-
>AddChild(Buttons[i]);
                                }

                                Buttons[i]-
>OnClicked.Add(Button_Delegates[i]);

                                UCanvasPanelSlot*
TextSlot = Cast<UCanvasPanelSlot>(Texts[i]->Slot);
                                if (TextSlot)
                                {
                                        TextSlot-
>SetPosition(FVector2D(110.f, 110.f + i * 60.f));
                                        TextSlot-
>SetSize(FVector2D(180.f, 30.f));
                                        Root_Canvas-
>AddChild(Texts[i]);
                                }
                        }
}

```

```

        }
    }
    void UCoursework_MenuWidget::Start()
    {
            if (TargetLevel.IsValid())
            {
                    FString LevelName =
TargetLevel.GetAssetName();

                    UGameplayStatics::OpenLevel(WorldContextObject,
FName(*LevelName));
            }
            else if (TargetLevel.ToSoftObjectPath().IsValid())
            {
                    FString LevelName =
TargetLevel.ToSoftObjectPath().GetAssetName();

                    UGameplayStatics::OpenLevel(WorldContextObject,
FName(*LevelName));
            }
    }
    void UCoursework_MenuWidget::Option( )
    {
            UE_LOG(LogTemp, Warning,
TEXT("Button Clicked"));
    }
    void UCoursework_MenuWidget::Exit( )
    {
            UKismetSystemLibrary::QuitGame(GetWorld(),
nullptr, EQuitPreference::Quit, true);
    }

UCoursework_PickUpComponent.h
#pragma once

#include "CoreMinimal.h"
#include "Components/SphereComponent.h"
#include "Coursework_Character.h"
#include "Coursework_PickUpComponent.generated.h"

DECLARE_DYNAMIC_MULTICAST_DELEGATE_OnePara
m(FOn_Pick_Up, ACoursework_Character*, PickupCharacter);
UCLASS(Blueprintable, BlueprintType, ClassGroup = (Custom),
meta = (BlueprintSpawnableComponent))
class COURSEWORK_API UCoursework_PickUpComponent :
public USphereComponent
{
        GENERATED_BODY()

public:
        UCoursework_PickUpComponent();

        UPROPERTY(BlueprintAssignable, Category =
"Interaction") FOn_Pick_Up On_Pick_Up; // Делегат, на
который может подписаться кто угодно, чтобы получать это
событие

protected:
        virtual void BeginPlay() override;
        UFUNCTION() void
On_Sphere_Begin_Overlap(UPrimitiveComponent*
overlapped_component, AActor* other_actor,
UPrimitiveComponent* other_comp, int other_body_index, bool
from_sweep, const FHitResult& sweep_result);

```

```

};
UCoursework_PickUpComponent.cpp
#include "Coursework_PickUpComponent.h"

UCoursework_PickUpComponent::UCoursework_PickUpComponent()
{
    SphereRadius = 32.0f;
}

void UCoursework_PickUpComponent::BeginPlay()
{
    Super::BeginPlay();

    OnComponentBeginOverlap.AddDynamic(this,
    &UCoursework_PickUpComponent::On_Sphere_Begin_Overlap
    );
}

void
UCoursework_PickUpComponent::On_Sphere_Begin_Overlap(
UPrimitiveComponent *overlapped_component, AActor
*other_actor, UPrimitiveComponent *other_comp, int
other_body_index, bool from_sweep, const FHitResult
&sweep_result)
{
    ACoursework_Character *character =
    Cast<ACoursework_Character>(other_actor);

    if (character != 0)
    {
        On_Pick_Up.Broadcast(character);

        OnComponentBeginOverlap.RemoveAll(this);
    }
}
ACoursework_Pistol_A.h
#pragma once

#include "CoreMinimal.h"
#include "AWeapon.h"
#include "Components/MeshComponent.h"
#include "Coursework_Pistol_A.generated.h"

class USkeletalMeshComponent;

UCLASS()
class COURSEWORK_API ACoursework_Pistol_A : public
AWeapon
{
    GENERATED_BODY()

public:
    ACoursework_Pistol_A();
    bool Valid;

    int Bullet_Store;
    virtual float Get_HP_Rate() const override { return 8; };

protected:

    virtual void BeginPlay();

};
ACoursework_Pistol_A.cpp
#include "Coursework_Pistol_A.h"

ACoursework_Pistol_A::ACoursework_Pistol_A()

```

```

{
    Valid = false;
}

void ACoursework_Pistol_A::BeginPlay() {
    Super::BeginPlay();
    Mesh =
    FindComponentByClass<USkeletalMeshComponent>();

    Mesh-
    >SetSkeletalMesh(LoadObject<USkeletalMesh>(nullptr,
    TEXT("/Game/Weapon/MilitaryWeapSilver/Weapons/Meshes/P
    istols_A")));
    if (Mesh) {

        Valid_Weapons.AddUnique("/Game/Weapon/Military
        WeapSilver/Weapons/Meshes/Pistols_A");
        Valid = true;
    }
}
ACoursework_PreCharacter.h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "AWeapon.h"
#include "Animation/AnimMontage.h"
#include "Animation/AnimInstance.h"
#include "CourseWork_Save_Data.h"
#include "Coursework_PreCharacter.generated.h"

class USkeletalMeshComponent;

UCLASS()
class COURSEWORK_API ACoursework_PreCharacter : public
ACharacter
{
    GENERATED_BODY()

public:
    ACoursework_PreCharacter();
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
    float Health_Bar;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
    Category = Mesh) USkeletalMeshComponent* Mesh_1P;
    UFUNCTION(BlueprintCallable) virtual AWeapon*
    Get_Current_Weapon()const { return 0; };
    UFUNCTION(BlueprintCallable) virtual void
    Set_Current_Weapon(AWeapon* v) {};
    UPROPERTY(EditAnywhere,
    BlueprintReadWrite)UAnimMontage* Dead;

    bool Is_Dead = false;

    virtual void Destroy_Self();
    virtual void Pickup_Weapon(AWeapon* weapon);

protected:
    virtual void BeginPlay();
};
ACoursework_PreCharacter.cpp
#include "Coursework_PreCharacter.h"

ACoursework_PreCharacter::ACoursework_PreCharacter()

```

```

{
    InitialLifeSpan = 0.0f;
}

void ACoursework_PreCharacter::BeginPlay()
{
    Super::BeginPlay();
}
void ACoursework_PreCharacter::Destroy_Self()
{
    if (Dead && GetMesh() && GetMesh()-
>GetAnimInstance())
    {
        if (USkeletalMeshComponent* SkeletalMesh
= Mesh_1P)
        {
            SkeletalMesh-
>SetAnimationMode(EAnimationMode::AnimationSingleNode);

            if (Dead)
            {
                SkeletalMesh-
>SetAnimation(Dead);
                SkeletalMesh-
>Play(true);
            }
        }
    }
    if (Get_Current_Weapon()) {
        Get_Current_Weapon()->Detach();
    }
}

void ACoursework_PreCharacter::Pickup_Weapon(AWeapon*
weapon) {
    Set_Current_Weapon(weapon);
    Get_Current_Weapon()->Attach(Mesh_1P);
}

```

### ACoursework\_Projectile.h

#pragma once

```

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Coursework_Projectile.generated.h"

```

```

class USphereComponent;
class UProjectileMovementComponent;
UCLASS(config=Game)
class ACoursework_Projectile : public AActor
{

```

GENERATED\_BODY()

public:

```

    UPROPERTY(VisibleDefaultsOnly,
Category=Projectile)USphereComponent* Collision_Comp;

```

```

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly,
Category = Movement, meta = (AllowPrivateAccess = "true"))
    UProjectileMovementComponent*
Projectile_Movement;

```

ACoursework\_Projectile();

```

    UFUNCTION()void On_Hit(UPrimitiveComponent*
hit_comp, AActor* other_actor, UPrimitiveComponent*
other_comp, FVector normal_impulse, const FHitResult& Hit);

```

```

    UPROPERTY(VisibleDefaultsOnly, Category =
Projectile) float HP_Rate;

```

};

### ACoursework\_Projectile.cpp

```

#include "Coursework_Projectile.h"
#include "Coursework_PreCharacter.h"
#include "GameFramework/ProjectileMovementComponent.h"
#include "Components/SphereComponent.h"

```

ACoursework\_Projectile::ACoursework\_Projectile()

```

{
    Collision_Comp
CreateDefaultSubobject<USphereComponent>(TEXT("SphereC
omp"));

```

```

    Collision_Comp->InitSphereRadius(1.16f);
    Collision_Comp-
>BodyInstance.SetCollisionProfileName("Projectile");
    Collision_Comp-
>OnComponentHit.AddDynamic(this,
&ACoursework_Projectile::On_Hit);

```

```

    Collision_Comp-
>SetWalkableSlopeOverride(FWalkableSlopeOverride(Walkable
Slope_Unwalkable, 0.f));
    Collision_Comp->CanCharacterStepUpOn = ECB_No;

```

RootComponent = Collision\_Comp;

```

    Projectile_Movement
CreateDefaultSubobject<UProjectileMovementComponent>(TE
XT("ProjectileComp"));

```

```

    Projectile_Movement->UpdatedComponent
Collision_Comp;
    Projectile_Movement->InitialSpeed = 6000.f;
    Projectile_Movement->MaxSpeed = 6000.f;
    Projectile_Movement->bRotationFollowsVelocity

```

true;

Projectile\_Movement->bShouldBounce = true;

InitialLifeSpan = 3.0f;

}

```

void ACoursework_Projectile::On_Hit(UPrimitiveComponent*
hit_comp, AActor* other_actor, UPrimitiveComponent*
other_comp, FVector normal_impulse, const FHitResult& Hit)
{

```

```

    if (ACoursework_PreCharacter* character
Cast<ACoursework_PreCharacter>(other_actor))
    {

```

character->Health\_Bar -= HP\_Rate;

```

    if (character->Health_Bar <= 0&&
!character->Is_Dead) {

```

```

        character-
>Destroy_Self();

```

```

        character->Is_Dead
true;

```

}

}

```

    if ((other_actor != 0) && (other_actor != this) &&
(other_comp != 0) && other_comp->IsSimulatingPhysics())

```

```

        {
            other_comp-
>AddImpulseAtLocation(GetVelocity() * 1000.0f,
GetActorLocation());
        }
        Destroy();
    }
UCourseWork_Save_Data.h
#pragma once

#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include "CourseWork_Save_Data.generated.h"

UCLASS(Blueprintable, BlueprintType, ClassGroup = (Custom),
meta = (BlueprintSpawnableComponent))
class COURSEWORK_API UCourseWork_Save_Data:public
UObject
{
    GENERATED_BODY()

public:
    UCourseWork_Save_Data();
    struct playerData {
        char name[5];
        int kills;
        int score;
    };

};

    static int PlayerCount;
    UFUNCTION(BlueprintCallable, Category = "Utils")
    static int Get_PlayerCount() {
        return PlayerCount;
    };
    UFUNCTION(BlueprintCallable, Category = "Utils")
    static void Ser_PlayerCount() {
        PlayerCount = 4;
    };

    UFUNCTION(BlueprintCallable, Category = "Utils")
    static void Set_Name_Input(const FString& input);
    UFUNCTION(BlueprintCallable, Category = "Utils")
    static FString Get_Name_Input();

    UFUNCTION(BlueprintCallable, Category = "Utils")
    static FString Get_DataForTable_Input();

    UFUNCTION(BlueprintCallable, Category = "Utils")
    static void Table_Write(const FString& input);

    UFUNCTION(BlueprintCallable, Category = "Utils")
    static FString Table_Read();

    UFUNCTION(BlueprintCallable, Category = "Utils")
    static bool Get_SecondPlayer() {
        return SecondPlayerON;
    };

    UFUNCTION(BlueprintCallable, Category = "Utils")
    static void Ser_SecondPlayer(bool setting) {
        SecondPlayerON = setting;
    };

```

```

    UFUNCTION(BlueprintCallable, Category = "Utils")
    static int Get_Kills() { return Kills; }
    UFUNCTION(BlueprintCallable, Category = "Utils")
    static int Get_Score() { return Score; }

    static FString Name ;
    static int Kills;
    static int Score;
    static bool SecondPlayerON;

    static FString DataForTable;

private:
    static std::string dataPath;

};
UCourseWork_Save_Data.cpp
#include "CourseWork_Save_Data.h"

FString UCourseWork_Save_Data::Name = TEXT("");
int UCourseWork_Save_Data::Kills = 0;
int UCourseWork_Save_Data::Score = 0;
int UCourseWork_Save_Data::PlayerCount = 0;
bool UCourseWork_Save_Data::SecondPlayerON = false;

FString fullPath = FPaths::ProjectSavedDir() +
TEXT("playerDataFile.dat");
std::string UCourseWork_Save_Data::dataPath=
TCHAR_TO_UTF8(*fullPath);
FString UCourseWork_Save_Data::DataForTable = TEXT("");

UCourseWork_Save_Data::UCourseWork_Save_Data() {

}

void UCourseWork_Save_Data::Set_Name_Input(const
FString& input) {
    Name = input;
}
FString UCourseWork_Save_Data::Get_Name_Input() { return
Name; }
FString UCourseWork_Save_Data::Get_DataForTable_Input() {
return DataForTable; }
void UCourseWork_Save_Data::Table_Write(const FString&
input) {

    playerData p{ };
    strncpy_s(p.name, sizeof(p.name),
TCHAR_TO_UTF8(*Name), _TRUNCATE);
    p.kills = Kills;
    p.score = Score;
    std::vector<playerData> in_pr;
    bool found = false;
    {
        std::ifstream ifs(dataPath, std::ios::binary);
        if (ifs.is_open())
        {
            playerData pl;
            while (ifs.read(reinterpret_cast<char*>(&pl),
sizeof(playerData)))
            {
                if (Name != "" && strcmp(p.name, pl.name) == 0)
                {
                    pl = p;

```

```

        found = true;
    }
    in_pr.push_back(pl);
}
ifs.close();
}

if (!found && Name != "")
{
    in_pr.push_back(p);
}
{
    std::ofstream ofs(dataPath, std::ios::binary | std::ios::trunc);
    if (ofs.is_open())
    {
        for (const auto& player : in_pr)
        {
            ofs.write(reinterpret_cast<const char*>(&player),
sizeof(playerData));
        }
        ofs.close();
    }
}
Kills = 0;
Score = 0;
}

```

```

FString UCourseWork_Save_Data::Table_Read() {
    DataForTable = "";
    std::vector<playerData> in_pr;
    std::ifstream ifs(dataPath, std::ios::in | std::ios::binary);

```

```

    if (ifs.is_open()) {
        playerData p;
        while (ifs.read((char*)&p, sizeof(playerData))) {

            in_pr.push_back(p);
        }
    }

```

```

    ifs.close();
}

```

```

        for (playerData pl : in_pr)
        {
            FString first = FString::ChrN(12 - FString(pl.name).Len(), '
');
            FString sec = FString::ChrN(12 -
FString::FromInt(pl.kills).Len(), ' ');

            DataForTable +=
FString::Printf(TEXT("%s%s%i%s\n"), *FString(pl.name),
*first, pl.kills, *sec, pl.score);
        }
        return TEXT("");
    }
}

```

```

ACoursework_Sniper_A.h

```

```

#pragma once

```

```

#include "CoreMinimal.h"
#include "AWeapon.h"
#include "Components/MeshComponent.h"
#include "Coursework_Sniper_A.generated.h"

```

```

class USkeletalMeshComponent;

```

```

UCLASS()

```

```

class COURSEWORK_API ACoursework_Sniper_A : public
AWeapon
{
    GENERATED_BODY()

public:
    ACoursework_Sniper_A();
    bool Valid;
    virtual float Get_HP_Rate() const override { return 15;
};

```

```

protected:
    virtual void BeginPlay();

```

```

};
ACoursework_Sniper_A.cpp
#include "Coursework_Sniper_A.h"

```

```

ACoursework_Sniper_A::ACoursework_Sniper_A()
{
    Valid = false;
}

```

```

void ACoursework_Sniper_A::BeginPlay() {
    Super::BeginPlay();
    Mesh

```

```

    FindComponentByClass<USkeletalMeshComponent>();

```

```

    Mesh-
>SetSkeletalMesh(LoadObject<USkeletalMesh>(nullptr,
TEXT("/Game/Weapon/MilitaryWeapSilver/Weapons/Meshes/S
niper_Rifle_A"));
    if (Mesh) {

```

```

        Valid_Weapons.AddUnique("/Game/Weapon/Military
WeapSilver/Weapons/Meshes/Sniper_Rifle_A");
        Valid = true;
    }
}

```

```

ACoursework_Spawn_Manger.h

```

```

#pragma once

```

```

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "Machine_Enemy.h"
#include "Coursework_Spawn_Manger.generated.h"

```

```

UCLASS()

```

```

class COURSEWORK_API ACoursework_Spawn_Manger :
public AActor
{
    GENERATED_BODY()

```

```

public:
    ACoursework_Spawn_Manger();

```

```

protected:
    virtual void BeginPlay();
    void Spawn();
    FTimerHandle Current_Timer;

```

```

};
ACoursework_Spawn_Manger.cpp
#include "Coursework_Spawn_Manger.h"
#include "TimerManager.h"

```

```

ACoursework_Spawn_Manger::ACoursework_Spawn_Manger()
{}

```

```

void ACoursework_Spawn_Manger::BeginPlay()
{
    Super::BeginPlay();

    GetWorld()->GetTimerManager().SetTimer(
        Current_Timer,
        this,
        &ACoursework_Spawn_Manger::Spawn,
        15,
        true
    );
}

void ACoursework_Spawn_Manger::Spawn() {
    FVector SpawnLocation = GetActorLocation();
    FRotator SpawnRotation = FRotator::ZeroRotator;

    TSubclassOf<AMachine_Enemy>BPClass =
    StaticLoadClass(AMachine_Enemy::StaticClass(), nullptr,
    TEXT("/Game/NPC/PB_NPC_Machine.PB_NPC_Machine_C")
    );
    AMachine_Enemy* enemy = GetWorld()-
    >SpawnActor<AMachine_Enemy>(BPClass, SpawnLocation,
    SpawnRotation);

    TSubclassOf<AActor>AABP =
    StaticLoadClass(AActor::StaticClass(), nullptr,
    TEXT("/Game/Weapon/MilitaryWeapSilver/Pickups/Pistol_Stor
    .Pistol_Stor_C"));
    AActor* SpawnAA = GetWorld()-
    >SpawnActor<AActor>(AABP, SpawnLocation,
    SpawnRotation);
}
UCourseworkWeaponComponent.h
#pragma once

#include "Coursework_Character.h"
#include "Components/SkeletalMeshComponent.h"
#include "CourseworkWeaponComponent.generated.h"

UCLASS(Blueprintable, BlueprintType, ClassGroup = (Custom),
meta = (BlueprintSpawnableComponent))
class COURSEWORK_API UCourseworkWeaponComponent :
public USkeletalMeshComponent
{
    GENERATED_BODY()

public:
    UCourseworkWeaponComponent();

    UFUNCTION(BlueprintCallable, Category =
    "Weapon") void AttachWeapon(ACoursework_Character*
    TargetCharacter); // Приаттачить пушку к персонажу
    UFUNCTION(BlueprintCallable, Category =
    "Weapon") void Fire(); // Выстрелить снарядом

    UPROPERTY(EditDefaultsOnly, Category =
    Projectile) TSubclassOf<class ACoursework_Projectile>
    ProjectileClass;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,
    Category = Gameplay) USoundBase* FireSound;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,
    Category = Gameplay) UAnimMontage* FireAnimation;
    UPROPERTY(EditAnywhere, BlueprintReadWrite,
    Category = Gameplay) FVector MuzzleOffset; // Смещение дула
    пушки от позиции персонажа

```

```

    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputMappingContext* FireMappingContext;
    UPROPERTY(EditAnywhere, BlueprintReadOnly,
    Category = Input, meta = (AllowPrivateAccess = "true"))
    UInputAction* FireAction;

protected:
    virtual void EndPlay(const EEndPlayReason::Type
    end_play_reason) override;

private:
    ACoursework_Character* Character;
};
UCourseworkWeaponComponent.cpp
#include "CourseworkWeaponComponent.h"
#include "Coursework.h"
#include "Coursework_Projectile.h"
#include "GameFramework/PlayerController.h"
#include "Camera/PlayerCameraManager.h"
#include "Kismet/GameplayStatics.h"
#include "EnhancedInputComponent.h"
#include "EnhancedInputSubsystems.h"

UCourseworkWeaponComponent::UCourseworkWeaponCompo
nent()
{
    MuzzleOffset = FVector(100.0f, 0.0f, 10.0f);
}

void
UCourseworkWeaponComponent::AttachWeapon(ACoursework
_Character* target_character)
{
    Character = target_character;
    if (Character == 0)
        return;

    FAttachmentTransformRules
    attachment_rules(EAttachmentRule::SnapToTarget, true);
    AttachToComponent(Character->Mesh_1P,
    attachment_rules, FName(TEXT("GripPoint")));

    if (APlayerController* player_controller =
    Cast<APlayerController>(Character->GetController()))
    {
        if (UEnhancedInputLocalPlayerSubsystem*
        input_subsystem =
        ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubs
        ystem>(player_controller->GetLocalPlayer()))
            input_subsystem-
            >AddMappingContext(FireMappingContext, 1);

        if (UEnhancedInputComponent*
        enhanced_input_component =
        Cast<UEnhancedInputComponent>(player_controller-
        >InputComponent))
            enhanced_input_component-
            >BindAction(FireAction, ETriggerEvent::Triggered, this,
            &UCourseworkWeaponComponent::Fire);
    }
}

void UCourseworkWeaponComponent::Fire()
{
    if (Character == 0 || Character->GetController() == 0)

```



```

        return;
    if (ProjectileClass != 0)
    {
        if (UWorld* world = GetWorld())
        {
            APlayerController*
player_controller = Cast<APlayerController>(Character-
>GetController());

            FRotator spawn_rotation =
player_controller->PlayerCameraManager-
>GetCameraRotation();

            FVector spawn_location =
GetOwner()->GetActorLocation() +
spawn_rotation.RotateVector(MuzzleOffset);
            FActorSpawnParameters
actor_spawn_params;

            actor_spawn_params.SpawnCollisionHandlingOverrid
e =
ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButDo
ntSpawnIfColliding;

            world-
>SpawnActor<ACoursework_Projectile>(ProjectileClass,
spawn_location, spawn_rotation, actor_spawn_params);
        }

        if (FireSound != 0)

            UGameplayStatics::PlaySoundAtLocation(this,
FireSound, Character->GetActorLocation());

        if (FireAnimation != 0)
        {
            if (UAnimInstance* anim_instance =
Character->Mesh_1P->GetAnimInstance())
            {
                anim_instance-
>Montage_Play(FireAnimation, 1.0f);
            }
        }

        void UCourseworkWeaponComponent::EndPlay(const
EEndPlayReason::Type end_play_reason)
        {
            if (Character == 0)
                return;

            if (APlayerController* player_controller =
Cast<APlayerController>(Character->GetController()))
            {
                if (UEnhancedInputLocalPlayerSubsystem*
input_subsystem =
ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubs
ystem>(player_controller->GetLocalPlayer()))
                {
                    input_subsystem-
>RemoveMappingContext(FireMappingContext);
                }
            }
        }
    }
}

AMachine_Enemy.h
#pragma once

#include "CoreMinimal.h"
#include "Coursework_Projectile.h"
#include "Coursework_PreCharacter.h"
#include "Perception/PawnSensingComponent.h"

```

```

#include "Components/TimelineComponent.h"

#include "NiagaraFunctionLibrary.h"
#include "NiagaraComponent.h"
#include "Machine_Enemy.generated.h"

UCLASS(config = Game)
class COURSEWORK_API AMachine_Enemy : public
ACoursework_PreCharacter
{
    GENERATED_BODY()

public:
    AMachine_Enemy();

    UPROPERTY(VisibleAnywhere,
BlueprintReadOnly)UPawnSensingComponent*
PawnSensingComponent;
    UPROPERTY(VisibleAnywhere,
BlueprintReadOnly)UNiagaraComponent* NiagaraComponent;

    AWeapon* Get_Current_Weapon()const override {
return Weapon; };
    void Set_Current_Weapon(AWeapon* v) override {
Weapon = v; };

    virtual void Destroy_Self() override;

    void Spawn_Niagara();

protected:
    virtual void BeginPlay();

private:
    UPROPERTY()UTimelineComponent* FallTimeline;

    AWeapon* Weapon;
};

AMachine_Enemy.cpp
#include "Machine_Enemy.h"
#include "Components/CapsuleComponent.h"
#include "AIController.h"
#include "TimerManager.h"
#include "Components/MeshComponent.h"

AMachine_Enemy::AMachine_Enemy()
{
    Health_Bar = 20;
    Mesh_1P = GetMesh();
    UE_LOG(LogTemp, Warning, TEXT(("f")),
Health_Bar);

    PawnSensingComponent =
CreateDefaultSubobject<UPawnSensingComponent>(TEXT(("Pa
wnSensingComponent")));

    PawnSensingComponent->SetActive(true);
    PawnSensingComponent->bAutoActivate = true;
    PawnSensingComponent-
>SetPeripheralVisionAngle(90.0f);
    PawnSensingComponent->SightRadius = 1900.0f;
    PawnSensingComponent->HearingThreshold =
2800.0f;
    PawnSensingComponent-
>SetPeripheralVisionAngle(70);

```

```

        NiagaraComponent =
CreateDefaultSubobject<UNiagaraComponent>(TEXT("Niagara
Component"));
        NiagaraComponent-
>SetupAttachment(RootComponent);

    }
    void AMachine_Enemy::Spawn_Niagara() {
    if (UNiagaraSystem* NiagaraEffect =
LoadObject<UNiagaraSystem>(nullptr,
TEXT("/Game/Basic_VFX/Niagara/NS_Basic_2.NS_Basic_2"))
    )
        {

            FVector SpawnLocation = Mesh_1P-
>GetSocketLocation("SpineSocket");
            NiagaraComponent =
UNiagaraFunctionLibrary::SpawnSystemAtLocation(
                GetWorld(),
                NiagaraEffect,
                SpawnLocation,
                FRotator::ZeroRotator
            );
            NiagaraComponent-
>SetWorldScale3D(FVector(10.0f));
            NiagaraComponent->ActivateSystem(true);
            NiagaraComponent->SetAutoDestroy(true);
        }
    }

    void AMachine_Enemy::BeginPlay()
    {
        Super::BeginPlay();

        FVector SpawnLocation = GetActorLocation();
        FRotator SpawnRotation = FRotator::ZeroRotator;

        TSubclassOf<AWeapon>BPClass =
StaticLoadClass(AWeapon::StaticClass(), nullptr,
TEXT("/Game/Weapon/MilitaryWeapSilver/Weapons/BP_Pistol
_A.BP_Pistol_A_C"));
        if (BPClass) {
            AWeapon* SpawnedWeapon = GetWorld()-
>SpawnActor<AWeapon>(BPClass, SpawnLocation,
SpawnRotation);
            Pickup_Weapon(SpawnedWeapon);
        }
    }

    void AMachine_Enemy::Destroy_Self() {
        ACoursework_PreCharacter::Destroy_Self();

        FTimerHandle TimeHendler;
        GetWorldTimerManager().SetTimer(
            TimeHendler,
            [this]() {
                Spawn_Niagara();
                SetLifeSpan(1.5f);
            },

```

```

            5.0f,
            false
        );
        UCourseWork_Save_Data::Kills++;
        UCourseWork_Save_Data::Score+=100;
    }
UMyAnimClass.h
#pragma once

#include "CoreMinimal.h"
#include "MyAnimClass.generated.h"

class ACoursework_Character;
UCLASS()
class COURSEWORK_API UMyAnimClass : public
UAnimInstance
{
    GENERATED_BODY()

public:
    UMyAnimClass();
    UPROPERTY(BlueprintReadWrite, EditAnywhere,
Category = "Rifle") bool HasRifle;
};
UMyAnimClass.cpp
#include "MyAnimClass.h"
UMyAnimClass::UMyAnimClass()
{
    HasRifle = false;
}
Coursework.Build.cs
using UnrealBuildTool;

public class Coursework : ModuleRules
{
    public Coursework(ReadOnlyTargetRules Target) :
base(Target)
    {
        PCHUsage =
PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicDependencyModuleNames.AddRange(new
string[] {
            "Core",
            "CoreUObject",
            "Engine",
            "InputCore",
            "EnhancedInput",
            "UMG",
            "AIModule",
            "Niagara",
        });

        PrivateDependencyModuleNames.AddRange(new
string[] {
            "Slate",
            "SlateCore"
        });
    }
}

```

## ПРИЛОЖЕНИЕ Б

(обязательное)

### Изображения интерфейса программы



Рисунок Б.1 – Главное меню приложения

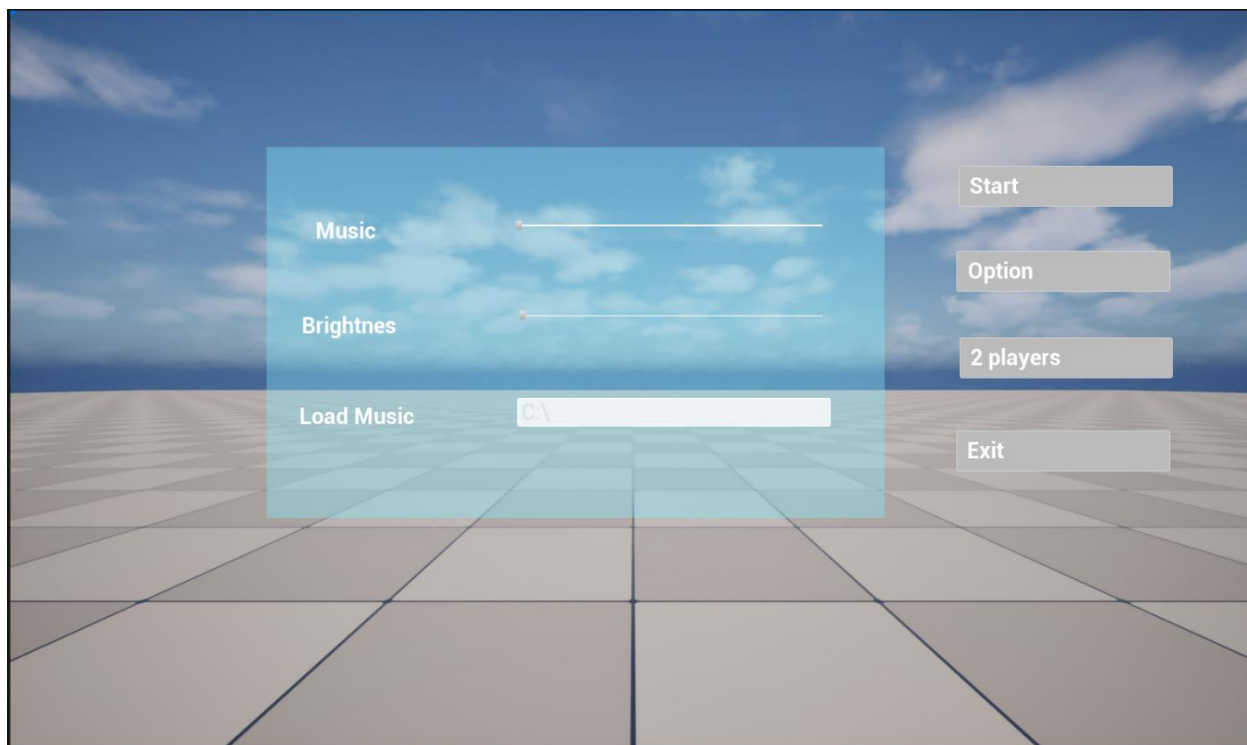


Рисунок Б.2 – Окно настроек приложения



Рисунок Б.3 – Окно окончания игры



Рисунок Б.4 – Окно окончания игры с таблицей лидеров и полем ввода

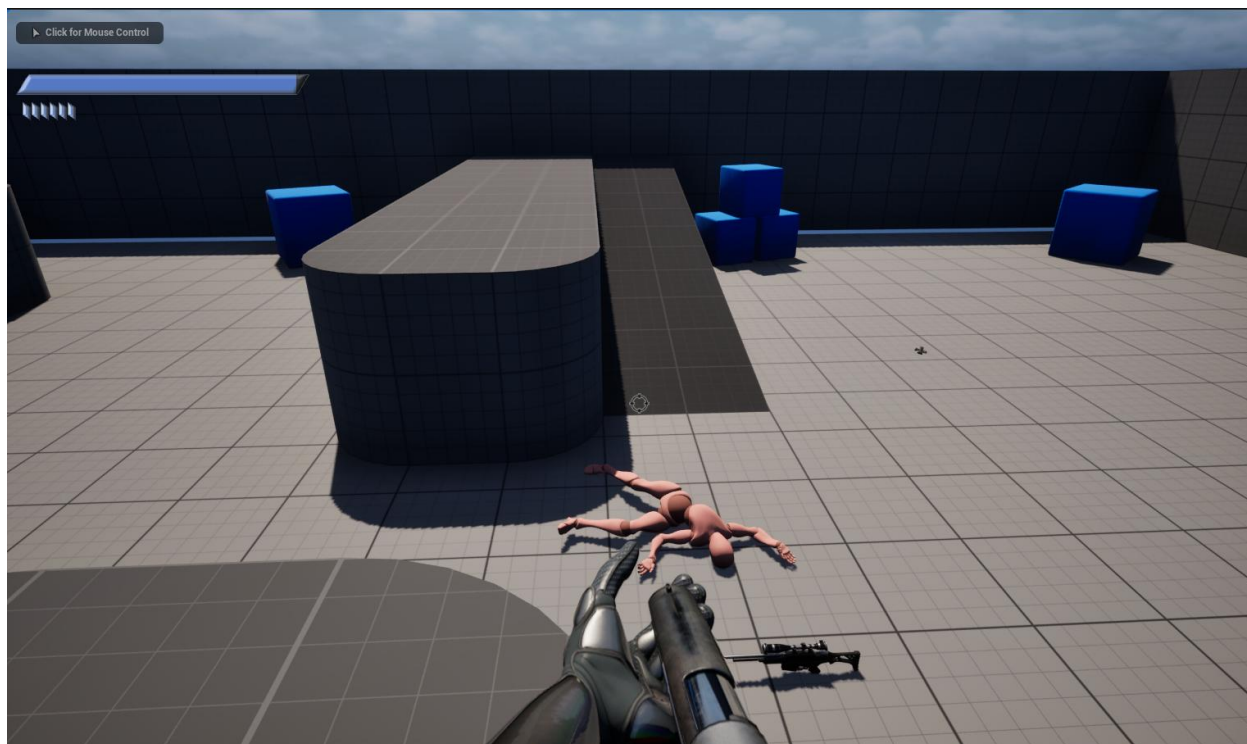


Рисунок Б.5 – Интерфейс игровой сессии в одиночном режиме



Рисунок Б.6 – Интерфейс игровой сессии в кооперативном режиме