

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Научный руководитель, приглашенный
преподаватель департамента программной
инженерии

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, канд. техн. наук

_____ Н.И. Веселко
«____» _____ 2023 г.

_____ В.В. Шилов
«____» _____ 2023 г.

МНОГОПОЛЬЗОВАТЕЛЬСКАЯ ИГРА В ЖАНРЕ ШУТЕР НА
UNREAL ENGINE 5

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.05.03-01 81 01-1-ЛУ

Инв. № подл.	Подп. и дата

Исполнитель
студент группы БПИ201
/А.Д. Зубко/
«____» _____ 2023 г.

Москва 2023

УТВЕРЖДЕН
RU.17701729.05.03-01 81 01-1-ЛУ

МНОГОПОЛЬЗОВАТЕЛЬСКАЯ ИГРА В ЖАНРЕ ШУТЕР НА UNREAL ENGINE 5

Пояснительная записка

RU.17701729.05.03-01 81 01-1

Листов 38

Инв. № подп.	Подп. и дата	Бзак. инв. №	Инв. №	Подп. и дата

Москва 2023

СОДЕРЖАНИЕ

1. Введение	4
1.1. Наименование программы	4
1.1.1. Наименование программы на русском языке	4
1.1.2. Наименование программы на английском языке	4
1.2. Документы, на основании которых ведется разработка	4
2. Назначение и область применения	5
2.1. Функциональное назначение	5
2.2. Эксплуатационное назначение	5
3. Технические характеристики	6
3.1. Постановка задачи на разработку программы	6
3.2. Описание алгоритма и функционирования программы с обоснованием выбора	6
3.2.1. Система игровых персонажей	6
3.2.1.1. Вид от первого лица	6
3.2.1.2. Графический интерфейс	6
3.2.1.3. Здоровье и смерть	7
3.2.1.4. Управление персонажем	8
3.2.2. Боевая система	8
3.2.2.1. Класс BP_BaseWeapon	8
3.2.2.2. Механика выстрела	9
3.2.2.3. Механика отдачи	12
3.2.2.4. Динамические характеристики	16
3.2.2.5. Механика перезарядки	17
3.2.2.6. Подсистема антчита	17
3.2.3. Игровые режимы	18
3.2.3.1. Лобби ожидания начала матча	18
3.2.3.2. Основной игровой режим	19
3.2.3.2.1. Команды	20
3.2.3.2.2. Автоматическое сбалансированное разделение по командам	20
3.2.3.2.3. Система игровых точек	21
3.2.3.2.3.1. Граф точек и провода	22

3.2.3.2.3.2. Захват точек	22
3.2.3.2.3.2.1. CaptureTriggers	23
3.2.3.2.3.2.2. Индикаторы захвата	23
3.2.3.2.4. Миникарта	24
3.2.3.2.4.1. WBP_Minimap	25
3.2.3.2.4.2. BP_BlipComponent	25
3.2.3.2.4.3. WBP_BaseBlip	26
3.2.3.2.4.4. WBP_CharacterBlip	26
3.2.3.2.4.5. WBP_PointBlip	26
3.2.3.2.4.6. WBP_WiresLayer	26
3.2.4. Главное меню и многопользовательская система	26
3.2.4.1. Главное меню	26
3.2.4.2. Выделенный сервер и подключение к нему	27
3.2.4.3. Интегрированные со Steam игровые сессии	28
3.2.5. Прочее	30
3.2.5.1. Меню «паузы»	31
3.3. Описание и обоснование выбора метода организации входных и выходных данных	31
3.3.1. Входные данные	31
3.3.2. Выходные данные	31
3.4. Описание и обоснование выбора состава технических и программных средств	32
3.4.1. Unreal Engine	32
4. Ожидаемые технико-экономические показатели	33
4.1. Предполагаемая потребность	33
4.2. Ориентировочная экономическая эффективность	33
4.3. Экономические преимущества разработки по сравнению с аналогами	33
Перечень использованных источников	34
Приложение. Описание и функциональное назначение классов	35

1. ВВЕДЕНИЕ

1.1. Наименование программы

1.1.1. Наименование программы на русском языке

«Многопользовательская игра в жанре шутер на Unreal Engine 5».

1.1.2. Наименование программы на английском языке

«Multiplayer Shooter Game in Unreal Engine 5».

1.2. Документы, на основании которых ведется разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденная академическим руководителем тема курсового проекта.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Функциональное назначение

Функциональное назначение разрабатываемого программного средства «Многопользовательская игра в жанре шутер на Unreal Engine 5» заключается в создании игры, в которой игроки смогут сражаться в команде против другой команды, используя игровое оружие. В процессе игры игроки могут захватывать игровые точки; команда, которая захватит главную точку команды противника побеждает. Главное назначение игры - обеспечить игрокам разнообразие и интересный геймплей, а также позволить им получить положительные эмоции от командной игры в жанре шутера.

2.2. Эксплуатационное назначение

Разрабатываемое программное средство может эксплуатироваться любым рядовым пользователем персонального компьютера, отвечающему программным и аппаратным ограничениям проектируемого программного обеспечения. При этом основной целевой аудиторией программного средства являются любители игр следующих жанров:

- 1) Шутер
- 2) Стратегия
- 3) Action

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Данная часть программы разработана в соответствии с техническим заданием «Много-пользовательская игра в жанре шутер на Unreal Engine 5» студента БПИ201 Зубко Александра Дмитриевича.

3.2. Описание алгоритма и функционирования программы с обоснованием выбора

3.2.1. Система игровых персонажей

Подключившийся к серверу игрок получает своего игрового персонажа, которым может управлять.

3.2.1.1. Вид от первого лица

Когда игрок владеет персонажем, на экране своего монитора он наблюдает вид этого персонажа от первого лица. В то же время другие игроки видят его от третьего лица. Реализовано это следующим образом. У каждого персонажа одновременно есть два меша¹⁾: первый отвечает за то, что видит сам персонаж (только руки), второй за то, что видят остальные (все туловище). Меш, который не должен быть виден – не рендерится[2].

3.2.1.2. Графический интерфейс

Как и многие другие экторы[3], персонаж имеет связанный с собой виджет. В данном случае речь идет о `WBP_Character`. То есть при уничтожении персонажа (например, при смерти) данный виджет также уничтожается. Вот, что содержит этот виджет:

¹⁾Меш – полигональная сетка[1].

Виджет персонажа



1 - прицел, 2 - индикатор количества патронов, 3 - индикатор здоровья

Рис. 1.

3.2.1.3. Здоровье и смерть

У персонажа есть здоровье, которое может уменьшаться, например, при попадании в него из оружия. На экране персонажа отображается полоска здоровья (рис. 1). Когда здоровье доходит до нуля – персонаж умирает. При смерти происходит анимация затемнения экрана, меш персонажа начинает симулировать Ragdoll[4], камера переключается на вид от третьего лица, следуя за мешем персонажа и игрок больше не может управлять персонажем (рис. 2).

Смерть персонажа



Рис. 2.

3.2.1.4. Управление персонажем

Управление персонажем осуществляется стандартно мышкой и клавиатурой. Клавиша W - двигаться вперед, S - назад, A - влево, D - вправо. Мышкой можно осматриваться. Также персонаж может прыгать (на пробел) и приседать (на левый Control).

3.2.2. Боевая система

3.2.2.1. Класс BP_BaseWeapon

Вся основная логика боевой системы содержится в классе `BP_BaseWeapon`. Данный класс является абстрактным и в нем содержится общая логика оружия в игре (на момент написания данного документа таковое только одно). Этот класс наследуют различные виды оружия, например `BP_Rifle` (рис. 3):

BP_Rifle

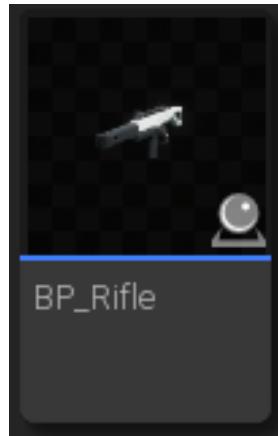


Рис. 3.

3.2.2.2. Механика выстрела

Игрок может осуществить выстрел из оружия при условии, что в магазине содержится хотя бы 1 патрон. Для выстрела используется левая кнопка мыши. Оружие может быть как автоматическим, так и полуавтоматическим. За это отвечает переменная-перечисление типа E_FireMode. Если оружие автоматическое, оно будет продолжать стрелять до тех пор, пока игрок не отпустит кнопку стрельбы или пока у оружия не закончатся патроны в магазине. Если оружие полуавтоматическое, каждое зажатие кнопки стрельбы не сделает более одного выстрела.

Когда игрок нажимает кнопку выстрела, сначала выполняются некоторые проверки на стороне клиента, такие как проверка на непустой магазин и проверка на то, что с предыдущего выстрела прошло достаточное количество времени. Это необходимо в том числе, чтобы лишний раз не нагружать сеть. Если все проверки на стороне клиента выполнены, клиент делает «ложный» выстрел у себя. Под «ложным» выстрелом подразумевается выстрел, который видит только этот клиент, в то же время сервер может отклонить этот выстрел (не зарегистрировать попадание) по некоторым причинам и тогда этот «ложный» выстрел не будет прореплицирован на остальные клиенты. Однако, при нормальных условиях (отсутствие читерства (подробнее в подразделе «Подсистема античита»), сравнительно небольшой пинг) сервер будет засчитывать до 100% попаданий. Некоторые серверные архитектуры могут не делать «ложного» выстрела, а ждать точного ответа от сервера, однако при таком подходе игроки с высоким пингом будут страдать от долгой задержки от нажатия кнопки выстрела до косметического выстрела.

Вместе с «ложным» выстрелом клиент вызывает серверный RPC²⁾. В этот RPC он передает Origin выстрела (место откуда должна вылететь «пуля») и направление выстрела. При этом направление квантуется встроенной в Unreal Engine функцией *Make Vector Net*

²⁾RPC – Remote procedure call[5].

Quantize Normal. Это делается для сетевой оптимизации. Когда сервер выполняет этот RPC, он выполняет те же проверки, что и должен был выполнить клиент. Если проверки выполнены успешно, сервер заносит полученные от клиента данные об Origin'e и направлении выстрела в структуру `S_ShotData` и реплицирует[6] ее на все остальные клиенты (на клиент, который и является инициатором этого выстрела эта структура не реплицируется, поскольку он и так знает эти данные). Когда эта переменная прореприлировалась на клиент, этот клиент также выполняет «ложный» выстрел у себя.

Непосредственная логика серверного выстрела и «ложного» клиентского выстрела расположена в функции `ShotInternal`. Эта функция осуществляет `LineTrace`[7] по данным выстрела (Origin и направление) по специально созданному Trace каналу *Bullet*:

- 1) Если код исполняется на сервере:

Декрементируется количество патронов

Если `LineTrace` определил, что под траекторию пули попал игровой персонаж, сервер наносит этому персонажу урон (уменьшает здоровье) в соответствии с характеристиками данного оружия. При этом урон умножается на мультипликатор части тела, в которую попала пуля. Так, например, попадание персонажу в голову умножает урон на 4. Помимо меша персонажа существует физический аксет персонажа (рис. 4).

Физический аксет персонажа «PA_Mannequin»



Рис. 4.

Этот аксет содержит физические примитивы, такие как капсулы. Каждому отдельному физи-

ческому примитиву можно задать физический материал. Поэтому для физических материалов персонажа я выделил абстрактный класс `PM_Damageable`, который содержит в себе публичное свойство `DamageMultiplier` (мультипликатор урона). От этого класса наследуются два класса: `PM_Player_WeakSpot` (слабые места (голова и шея), мультипликатор урона 4) и `PM_Player` (прочие места, мультипликатор урона 1). В свою очередь `LineTrace` в своей структуре возврата возвращает в том числе физический материал. Этот материал приводится к вышеописанному типу `PM_Damageable` и таким образом сервер получает множитель урона для конкретной части тела, в которую попала пуля.

Помимо этого урон падает тем больше, чем больше пролетела пуля. Так самым простейшим образом симулируется сопротивление воздуха в результате которого пуля теряет скорость во время полета и вследствие энергии при контакте с поверхностью. Для этой сцены класс `BP_BaseWeapon` содержит в свойствах содержит ссылку на ассет типа `FloatCurve`. В этом ассете содержится кривая графика зависимости мультипликатора урона от расстояния, которое пролетела пуля. Более конкретно от доли расстояния к максимальной дальности оружия. Например, вот как выглядит этот ассет у класса `BP_Rifle`:

Curve_RifleDamageFalloff_Float

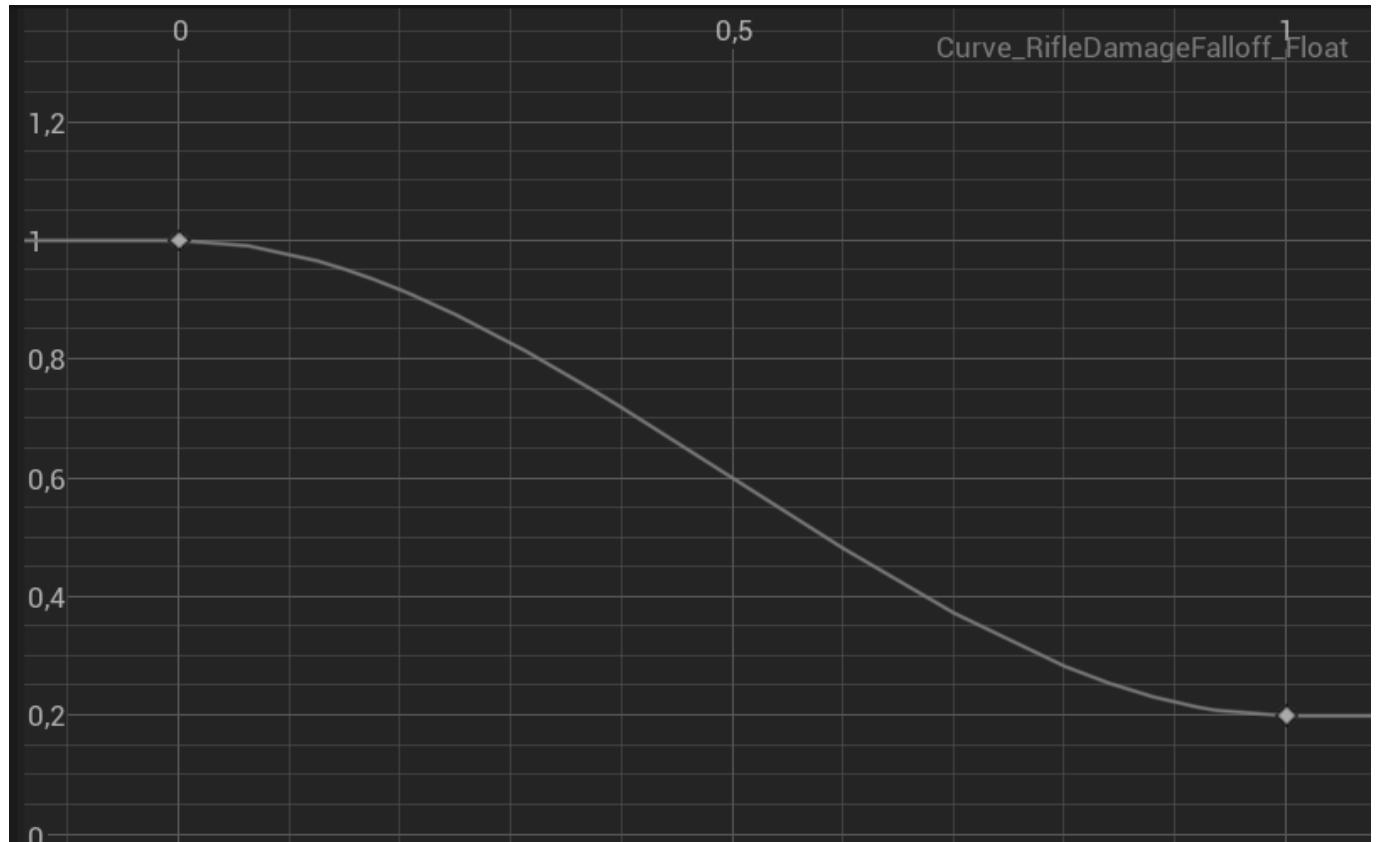


Рис. 5.

Как можно видеть, мультипликатор у винтовки начинается с 1 и постепенно падает до 0.2, когда пуля пролетела максимально возможное расстояние. Таким образом, при рассчете, если

дальность полета пули BP_Rifle равна 100м, то если пуля до контакта с персонажем пролетит 50м, то в доле от максимальной дальности это составит 0.5, а на графике 0.5 соответствует мультипликатору урона 0.6. Стоит также заметить, что пуля не может пролететь расстояние более дальности своего оружия, так как это ограничивается в вызове функции LineTrace.

2) Если код исполняется для эктора оружия, персонаж-владелец которого контролируется игроком, который играет на данном клиенте:

Декрементируется количество патронов

Симулируется отдача (подробнее в подразделе «Механика отдачи»)

3) Если код исполняется на любой машине, в том числе на клиенте, который в то же время является сервером, но не на выделенном сервере (чистый сервер с вырезанными косметическими эффектами и без возможности играть на нем не в качестве клиента):

Воспроизводятся косметические эффекты. А именно:

Анимация выстрела оружия

Анимация выстрела персонажа

Эффект дульной вспышки

Эффект выброса снаряда

Пространственный звук выстрела

Выстрел от третьего лица



Рис. 6.

3.2.2.3. Механика отдачи

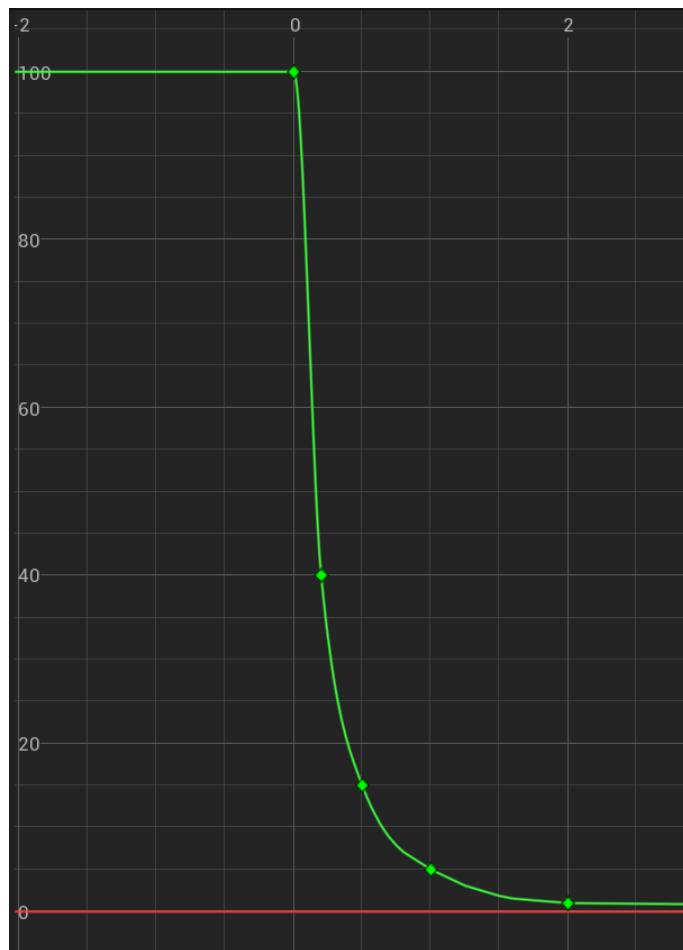
Также была реализована механика отдачи. Отдача в данном проекте является недетерминированной, то есть каждый раз она случайная. При этом она также может настраиваться как статически, так и динамически:

1) Статическая настройка:

Класс BP_BaseWeapon содержит два свойства типа VectorCurve: YawPitchMeanCurve и YawPitchStdCurve. YawPitchMeanCurve определяет среднее арифметическое отклонений угла рысканья (Yaw) и тангажа (Pitch)[8]. А YawPitchStdCurve определяет стандартное отклонение тех же углов. Эти данные используются для сэмплирования отклонения углов по нормальному распределению[9] с данными параметрами. За это отвечает функция SampleNormalDistribution из BPFL_MathLibrary. Кривая отображает зависимость среднего арифметического и стандартного отклонения углов от стабильности оружия (подробнее в следующем пункте: «Динамическая настройка»).

Вот как выглядят эти кривые для BP_Rifle:

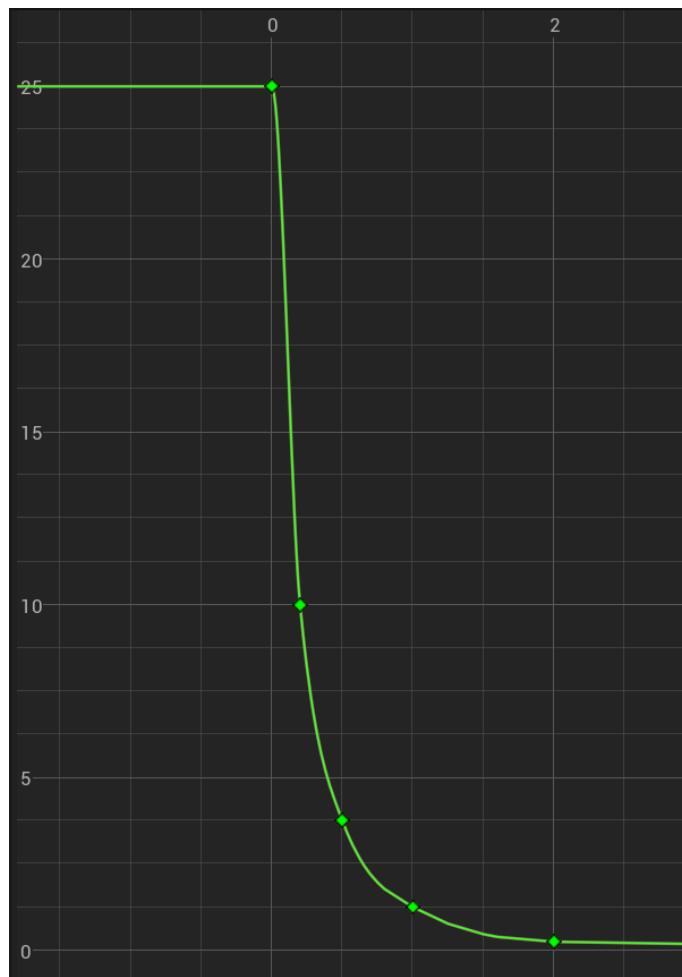
Среднее арифметическое отклонения углов при отдаче: Curve_RifleYawPitchMean_Vector



Красная кривая - угол Yaw, зеленая - угол Pitch. По данной картинке видно, что среднее арифметическое угла Pitch всегда положительное, что значит, что прицел в среднем всегда будет уводить вверх. А среднее арифметическое угла Yaw равно нулю, что значит, что в какую-то определенную сторону влево-вправо постоянно прицел смещаться не будет (но будет случайно дергаться влево-вправо из-за положительности стандартного отклонения, об этом на следующей картинке)

Рис. 7.

Стандартное отклонение отклонения углов при отдаче: Curve_RifleYawPitchStd_Vector



Зеленая кривая - и угол Yaw, и угол Pitch.

Благодаря положительности стандартного отклонения отдача будет непредсказуемой.

Рис. 8.

2) Динамическая настройка:

Как уже было сказано, графики выражают зависимость отклонений углов от стабильности оружия. Стабильность - еще одно (на самом деле 2 (`StaticStability` и `DynamicStability`)) свойство, определенное в классе `BP_BaseWeapon`. `DynamicStability` может изменяться в процессе игры из-за чего прямо по ходу игры возможно автоматическое уменьшение отдачи при увеличении этого свойства.

3) Результат:

В результате эта механика может быть легко настроена дизайнером для любого нового оружия. Все, что нужно - это настроить две вышеперечисленные кривые. Вся остальная логика динамического изменения скрыта от дизайнера и ему не нужно будет отдельно настраивать отдачу для каждого нового уровня прокачки характеристик оружия. Сами углы отклонения отдачи при каждом новом выстреле аккумулируются к соответствующей переменной `PendingRecoilYawPitchOffset` и постепенно «поглащаются» (применяются к углам

игроком в таймере `AbsorbPartOfRecoilTimerHandle` и функции `OnAbsorbPartOfRecoilTimerElapsed`, которая вызывает `ControllerAbsorbPartOfRecoil`.

3.2.2.4. Динамические характеристики

Как было вкратце упомянуто в предыдущем пункте, оружие помимо статических характеристик имеет также и динамические. Статические характеристики задаются разработчиками для каждого отдельного оружия. Некоторые характеристики являются полностью статическими, то есть не могут изменяться по ходу игры, например, `FireMode` (режим стрельбы). Некоторые характеристики динамические. Они также содержат статическое свойство для базовой настройки дизайнером. Но помимо этого они содержат и динамическое свойство, которое может меняться по ходу игры. Конкретнее, к ним по ходу игры может быть применен динамический конфиг, т. е. структура `S_WeaponDynamicConfig`. На момент написания данного документа эта структура выглядит так:

`S_WeaponDynamicConfig`

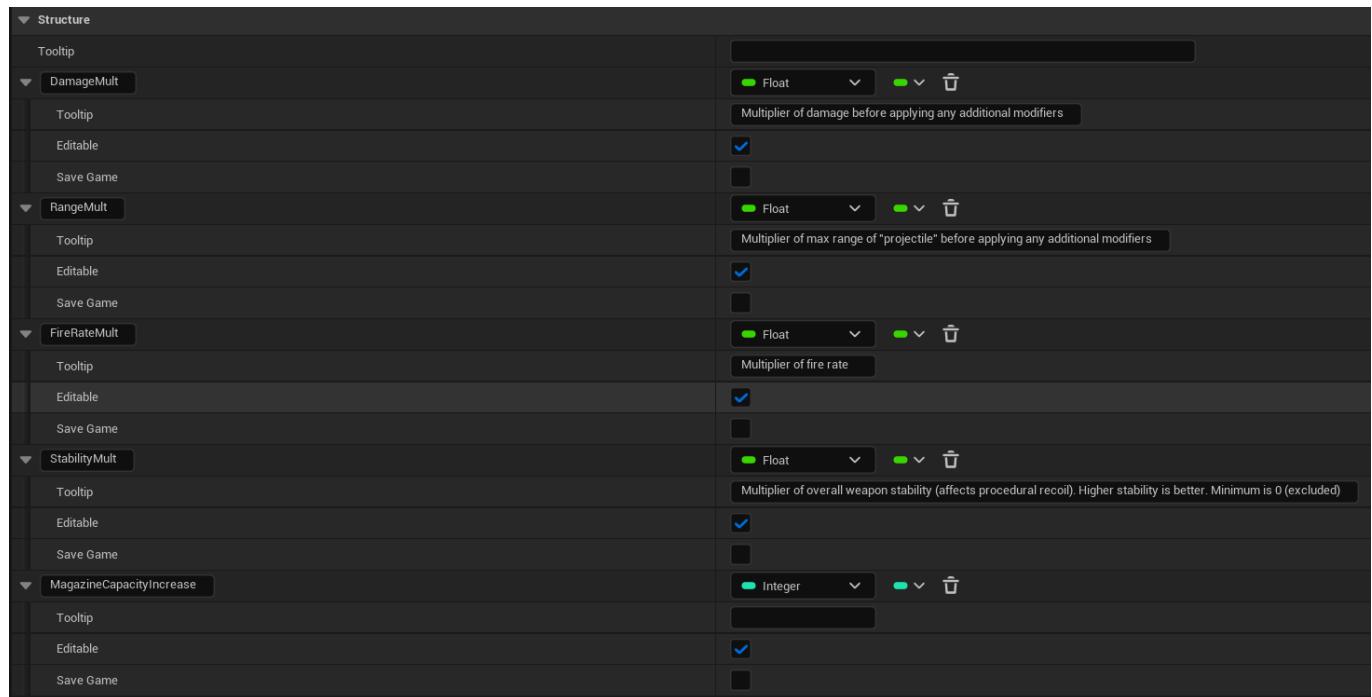


Рис. 9.

Динамический конфиг приходит извне (из других систем). За применение динамического конфига отвечает функция `UpdateWithNewDynamicConfig`. Так, например, если статическое свойство урона оружия (`StaticDamage`) равно 20, а в новом динамическом конфиге `DamageMult` равен 1.5, то динамическое свойство урона оружия (`DynamicDamage`) станет равным 30 и базовым (до применения остальных модификаторов, таких как снижение урона на расстоянии) уроном оружия будет 30.

Эта механика позволяет динамически улучшать характеристики оружия по ходу игры. Сейчас динамических характеристик 5:

- 1) Урон
- 2) Дальность
- 3) Скорострельность
- 4) Стабильность
- 5) Ёмкость магазина

3.2.2.5. Механика перезарядки

Игрок может осуществлять перезарядку оружия своего персонажа по нажатию на клавишу R. При перезарядке воспроизводится соответствующая анимация вместе со звуком. При перезарядке клиент также вызывает серверное RPC, чтобы сервер реплицировал эту перезарядку на остальные клиенты, а также обновил у себя и реплицировал количество активных в магазине патронов данного оружия на ёмкость этого магазина, когда анимация перезарядки сообщила о наступлении момента перезарядки через AnimNotify[10] AN_Reload.

В моей реализации также используется «ложная» перезарядка у клиента, который инициирует сие действие, по аналогии с механикой выстрела. То есть, клиент так же обновляет количество патронов у себя из AN_Reload. В теории это также уменьшает визуальную задержку у игроков с большим пингом, однако может вызвать некритичные проблемы с рассинхронизацией данных переменных у сервера и клиента, в связи с чем сервер может не засчитывать первые выстрелы после перезарядки клиента с большим пингом. Хоть проблема и некритичная, так как клиент в любом случае не потеряет патроны в своем магазине из-за нерегистрации выстрелов (сервер все равно среплицирует актуальное количество патронов на клиент, чей выстрел он не засчитал), в дальнейшем это решение может быть пересмотрено и переделано.

3.2.2.6. Подсистема античита

Как известно, одна из проблем многопользовательских игр – читерство. Игроки могут модифицировать свой клиент игры, тем самым получая преимущество над другими игроками. В таких играх, как многопользовательские шутеры, полностью избавиться от этой проблемы невозможно. Однако можно минимизировать потенциальный эффект от читерства при грамотном проектировании серверной архитектуры. Серверная архитектура данного проекта изначально проектировалась из принципа недоверия к клиенту, поэтому любой критический вызов серверного RPC из клиента осуществит дополнительные проверки на стороне сервера и если проверки пройдены не будут, RPC проигнорируется, тем самым другие клиенты не испытают отрицательного эффекта от некоторых видов читов.

Например, если игрок изменит свой клиент и сделает себе бесконечное количество патронов, сервер просто проигнорирует все выстрелы, которые были отстреляны после опустошения магазина с точки зрения сервера, нежели клиента. При этом он выведет в консоль информацию

о подозрении в читерстве:

Сообщение античита о невозможности выстрелов

```
[2023.04.10-08.56.34:875][649]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.35:041][654]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.35:240][660]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.35:407][665]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.35:607][671]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.35:773][676]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.35:938][681]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.36:105][686]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.36:270][691]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.36:470][697]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.36:635][702]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.36:835][708]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
[2023.04.10-08.56.37:003][713]LogBlueprintUserMessages: [Weapon_GEN_VARIABLE_BP_Rifle_C_CAT_2] [Anticheat] Shooting when unable. MagazineAmmo=0. IsReloading=false
```

Рис. 10.

3.2.3. Игровые режимы

В игре существует несколько игровых режимов. Основные из них: лобби (`BP_GM_ConnectiaLobby`) и основной игровой режим (`BP_GM_ConnectiaMain`). Игровой режим – это эктор. Как и многие другие экторы, эти два имеют общую логику. Поэтому они оба наследуются от абстрактного класса `BP_GM_ConnectiaBase`.

3.2.3.1. Лобби ожидания начала матча

Основная серверная карта – лобби. Это означает, что при запуске сервера он открывает эту карту. Лобби предназначено для ожидания подключения игроков перед стартом основного матча. Так как класс `BP_GM_ConnectiaLobby` наследует `BP_GM_ConnectiaBase`, в лобби присутствует часть логики основного режима, например, возможность побегать, пострелять, поумирать. В лобби еще нет разделения по командам, поэтому нет какой-то определенной цели, кроме как размяться перед основным матчем.

В лобби можно проголосовать за начало основного матча. Когда все игроки проголосовали за начало, сервер начинает обратный отсчет до начала матча, после чего карта меняется на основную.

Вся необходимая информация выводится в графический интерфейс, в том числе подсказка кнопки для голосования и отмены голосования. Когда игрок нажимает необходимую кнопку, вызывается серверное RPC, сервер обновляет данные о проголосовавших и проверяет, следует ли ему начать матч. Помимо этого сервер реплицирует новые данные о проголосовавших или данные об отсчете до начала матча на остальные клиенты, чтобы они так же обновили свой графический интерфейс о ходе голосования:

Виджет голосования (игрок еще не проголосовал)



Рис. 11.

Виджет голосования (игрок уже проголосовал)



Рис. 12.

Виджет обратного отсчета до начала матча



Рис. 13.

3.2.3.2. Основной игровой режим

Основной игровой режим представляет из себя игру команды на команду. Карта основного игрового режима содержит несколько игровых точек. При смерти через некоторое время персонажи возрождаются на своей главной точке. Игра заканчивается, когда команда захватывает главную точку противника. Эта команда побеждает. Подробнее описано в подпунктах данного пункта. Когда игра завершается, все персонажи игроков из проигравшей команды мгновенно умирают, игрокам в графический интерфейс выводится сообщение с информацией о победившей команде (рис. 14), сервер начинает отсчет, после которого возвращает карту обратно на Лобби.

Уведомление о победившей команде



Рис. 14.

3.2.3.2.1. Команды

За команды отвечает Primary Data Asset[11] PDA_Team, который реализуют следующие Data Asset'ы:

- 1) Team_None – фиктивная команда, в основном отвечающая за нейтральный захват игровой точки (подробнее в «Система игровых точек»)
- 2) Team_Unassigned – команда, которая означает, что игроку еще не назначена конкретная команда, за которую можно играть
- 3) Team_Blue – команда синих
- 4) Team_Red – команда красных

В PDA_Team содержится цвет команды и ее название, а также логика по применению цвета команды к экторам, таким как персонаж игрока и оружие персонажа игрока. В этой логике определяются все меши данного эктора, для каждого меша получаются все его материалы[12], каждый материал приводится к MaterialInstanceDynamic (динамическому материалу)[13] и, если преобразование прошло успешно, цвет применяется для заданных в коде параметров этого динамического материала.

Возможные команды хранятся массивом в BP_GM_ConnectiaMain. Однако, сейчас на всех основных картах содержится только две команды (синие и красные). Но в будущем есть возможность расширения карт для игры более чем двух команд, однако на данный момент это выглядит нецелесообразно с точки зрения игрового дизайна.

3.2.3.2.2. Автоматическое сбалансированное разделение по командам

При подключении нового игрока, сервер распределяет игрока в команду. Если число игроков в командах равное, это распределение случайно. Если в одной команде меньше игроков, игрок будет распределен в эту команду для баланса.

При отключении игрока, игрок из его команды убирается.

Для удобства обеспечения этой и некоторой другой логики, в BP_GS_ConnectiaMain хранится словарь с ключами типа PDA_Team и значениями типа массивов BP_PS_ConnectiaMain (массив PlayerState[14]: PlayerState по сути содержит информацию об игроке).

3.2.3.2.3. Система игровых точек

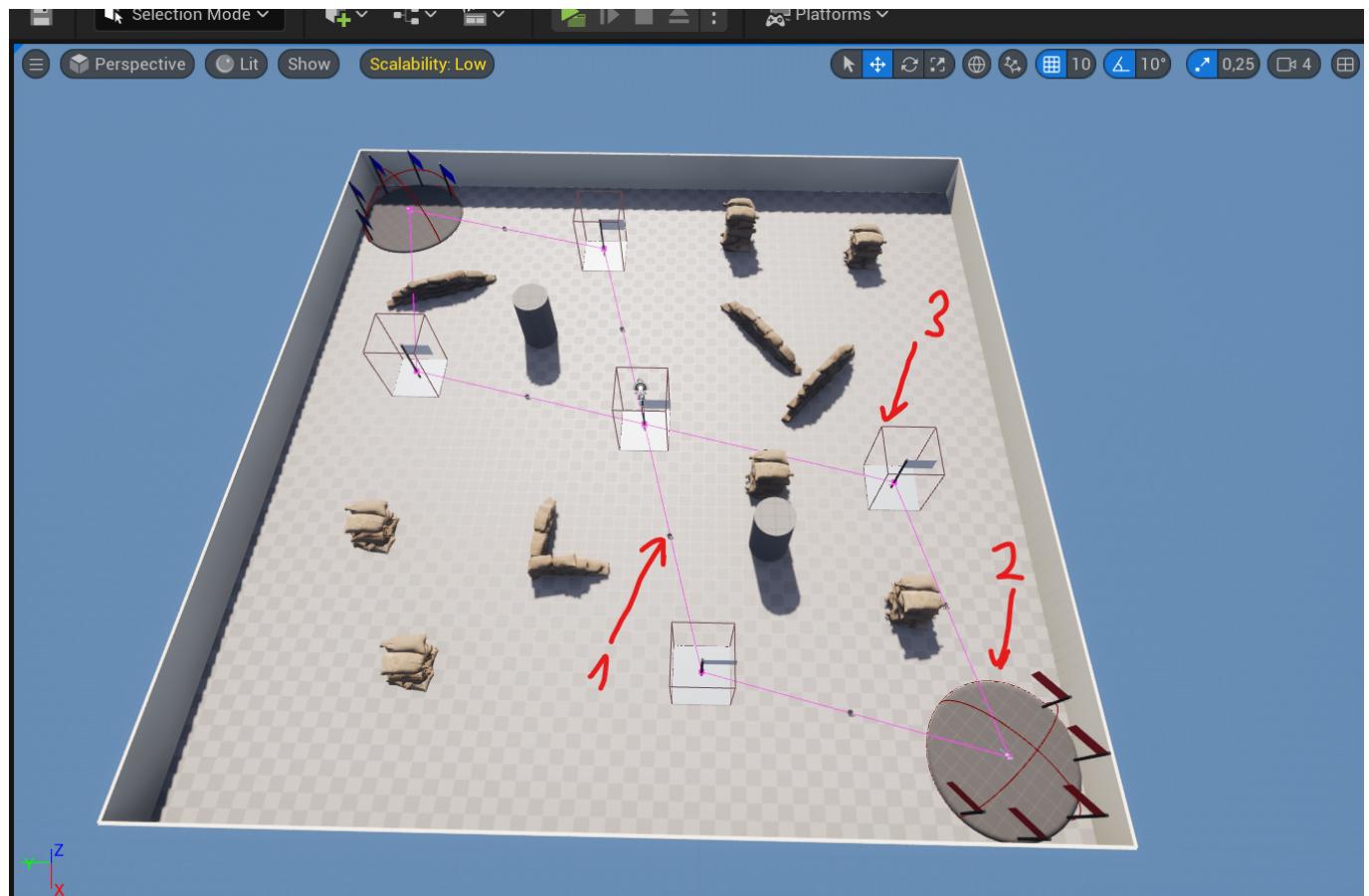
Ключевой частью основного игрового режима являются игровые точки. Как было упомянуто, захват главной точки выигрывает игру. Точки представляют собой экторы.

Основная логика, такая как захват точек (подробнее в подпункте «Захват точек») встроена в абстрактный класс `BP_BasePoint`.

Этот класс реализует еще один абстрактный класс `BP_BaseMainPoint`. `BP_BaseMainPoint` отвечает за логику главной точки, в том числе содержит список `PlayerStart`'ов[15], которые относятся к этой главной точке (эта информация нужна игровому режиму для определения, где возраждать персонажа). Далее `BP_BaseMainPoint` наследует неабстрактный класс `BP_MainPoint`, экземпляры этого класса размещаются непосредственно на карте как главные точки (рис. 15-2). В текущей реализации `BP_MainPoint` содержит в себе дочерние `PlayerStart`'ы и дочерние `FlagIndicator`'ы (подробнее в подпункте «Индикаторы захвата»), а также эктор-компонент[16] `BP_ColliderCaptureTriggerComponent` (подробнее в подпункте «CaptureTriggers»).

Абстрактный класс `BP_BasePoint` также реализует неабстрактный класс `BP-RegularPoint`. В текущей реализации `BP-RegularPoint` также содержит в себе дочерние `FlagIndicator`'ы и эктор-компонент `BP_ColliderCaptureTriggerComponent`, но не содержит `PlayerStart`'ы. Экземпляры этого класса размещаются на карте как обычные точки (рис. 15-3).

Карта в редакторе



1 - провод BP_Wire на карте (видим только в редакторе), 2 - главная точка BP_MainPoint, 3 - обычная точка BP_RegularPoint

Рис. 15.

3.2.3.2.3.1. Граф точек и провода

Точки могут соединяться с другими точками проводами. Провода сугубо «виртуальные», то есть не имеют визуального представления в самой игре. За провода отвечает класс BP_Wire. Провод по сути является ребром в графе игровых точек, где вершинами являются сами точки. То есть провод хранит ссылки на обе точки, которые он соединяет. Для удобства дизайна уровня, провод подсвечивается в редакторе в окне с картой с помощью встроенного в Unreal Engine компонента SplineComponent (рис. 15-1).

Каждая точка содержит массив смежных с ней точек (применение этого массива будет описано ниже). Этот массив инициализируется автоматически по проводам. То есть чтобы добавить новый провод, достаточно просто добавить на сцену актор BP_Wire и задать у него свойства точек, которые он соединяет.

3.2.3.2.3.2. Захват точек

Команда может захватывать нейтральные точки и точки вражеских команд. Сами классы точек не содержат в себе логику начала захвата и отмены захвата точек (при отмене прогресс захвата обнуляется). За это отвечают отдельные компоненты, которые «навешиваются» на актор точки (см. «CaptureTriggers»). Сами же точки оперируют со структурой S_CaptureState. Данная структура содержит следующие свойства:

- 1) CurrentTeam типа PDA_Team – команда в данный момент контролирующая данную точку или Team_None, если точка нейтральна
- 2) RecapturingTeam типа PDA_Team – команда, которая захватывает данную точку
- 3) IsBeingCaptured типа Boolean – ведется ли захват данной точки
- 4) RecapturingProgress типа Float – какой прогресс захвата данной точки (от 0 до 1)

Данная структура реплицируется с сервера на все клиенты.

Если точка захватывается, прогресс ее захвата увеличивается со скоростью CaptureSpeedPerSecond. Игрокам, которые начали захват точки НЕ обязательно оставаться на ней, чтобы захват продолжался.

Когда состояние захвата точки изменяется, точка транслирует event dispatcher[17] OnCaptureStateChanged, на которые могут подписываться индикаторы захвата (см. ниже) и миникарта (см. ниже).

Аналогично, когда точка полностью захвачена, транслируется event dispatcher `OnCaptureStateChanged`, на который подписывается, например, `BP_GM_ConnectiaMain` (только для главных точек) для завершения игры.

3.2.3.2.3.2.1. CaptureTriggers

Для начала захвата точки и отмены используется эктор-компонент `BP_ColliderCaptureTriggerComponent`. В будущем возможно добавление других таких компонентов, например, для захвата по нажатию кнопки. Вкратце, логика компонента `BP_ColliderCaptureTriggerComponent` заключается в том, что захват точки начинается, когда игрок из чужой команды заходит в триггер точки, где нет игроков из команды, которая владеет этой точкой; а отмена захвата – когда игрок из команды, которая эту точку теряет заходит в этот триггер при условии, что на ней нет захватывающих врагов или он персонажей таковых убил.

3.2.3.2.3.2.2. Индикаторы захвата

Индикаторы захвата визуализируют состояние захвата точек в игровом мире. Индикатором может являться любой эктор, который реализует интерфейс `BPI_Indicator`. Для реализации этого интерфейса нужно определить метод `Update`, который будет принимать в себя вышеописанную структуру `S_CaptureState`. В качестве реализации этого интерфейса в данный момент реализован 1 класс `BP_FlagIndicator`, который из себя представляет флаг, отображающий состояние захвата точки. Благодаря встроенному в Unreal Engine функционалу симуляции ткани и установленном на карте источнику ветра, во время игры флаг будет развиваться по ветру (рис. 16). Материал для флага реализован в `M_Flag`. В себя он принимает параметры схожие со структурой `S_CaptureState`.

Игрок из красной команды захватывает точку синей команды

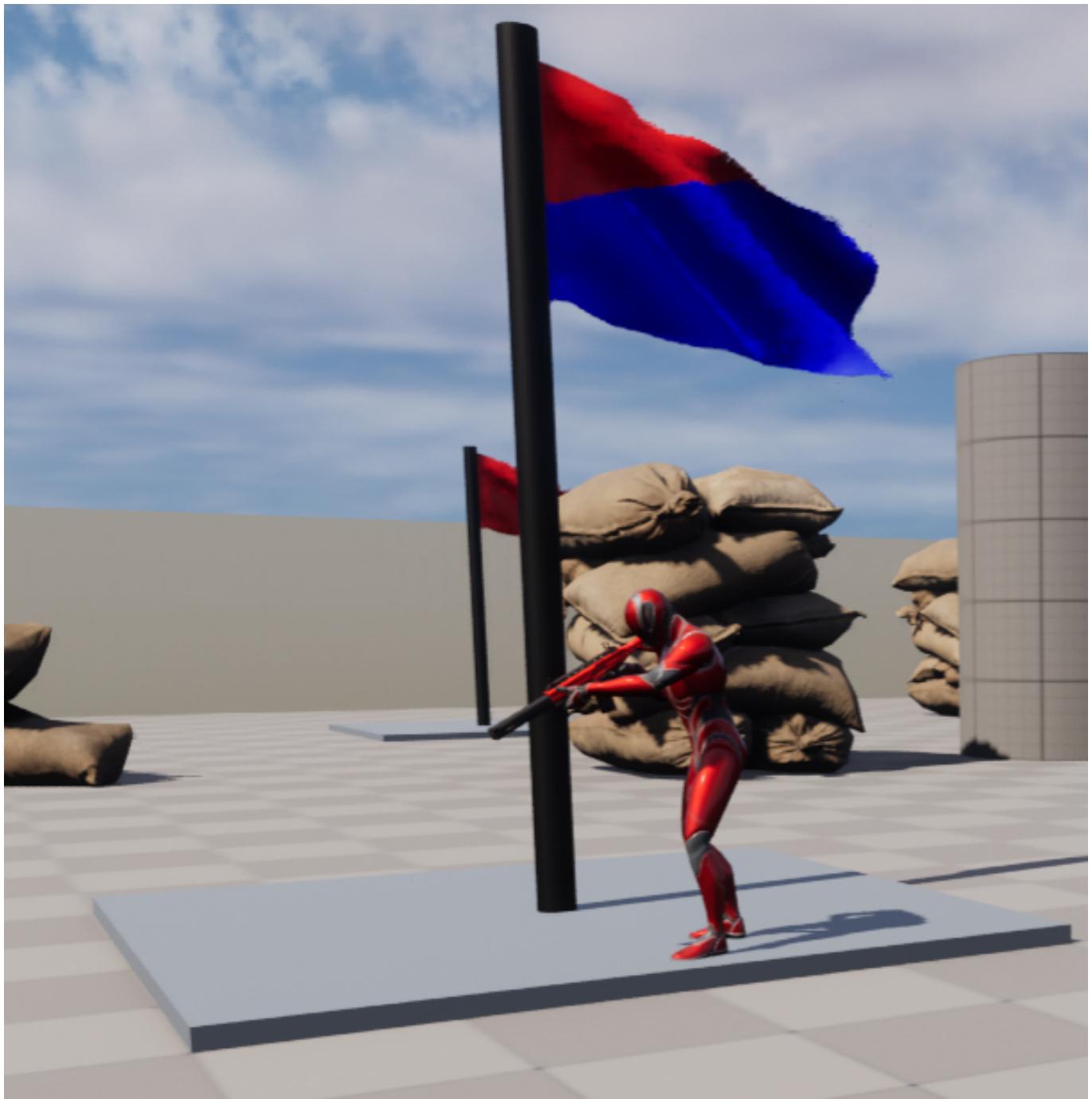


Рис. 16.

3.2.3.2.4. Миникарта

Для удобства ориентирования на карте и получения информации о состоянии точек реализована миникарта (рис. 17).

Миникарта

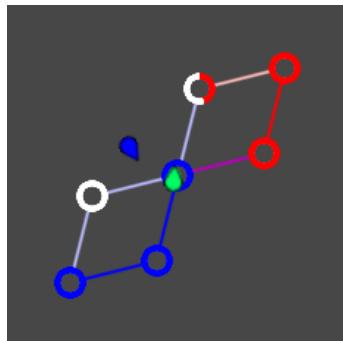


Рис. 17.

Миникарта отображает:

- 1) Позицию и угол поворота локального персонажа
- 2) Позицию и угол поворота персонажей из команды локального игрока
- 3) Точки и состояние их захвата
- 4) Провода

3.2.3.2.4.1. WBP_Minimap

WBP_Minimap – виджет самой миникарты, который расположен на виджете WBP_PlayerMain.

На данном виджете располагаются виджеты, которые должны быть на миникарте, т. н. блипы. Также этот виджет содержит логику по преобразованию координат из мировых в координаты на этом виджете (миникарте), аналогично с углом поворота.

3.2.3.2.4.2. BP_BlipComponent

Чтобы объект появился на миникарте (то есть на миникарте отобразился его блин), нужно повесить на него эктор-компонент BP_BlipComponent, в этом компоненте задать следующие свойства:

- 1) **WidgetClass** типа класса (не экземпляра класса, а именно самого класса) WBP_BaseBlip (подробнее ниже) – то, какой конкретно виджет должен показывать этот блин на миникарте
- 2) **IsStatic** типа Boolean – опционально: если false, то блин будет обновлять позицию и угол поворота каждый кадр, а если true, то сохранит эти значения при инициализации и не будет их обновлять
- 3) **ZOrder** типа Integer – опционально: очередность рендеринга блипа, например, блипы персонажей имеют более высокий ZOrder чем блипы точек, чтобы они рендерились поверх них

Когда этот компонент создается, он добавляет на миникарту виджет типа WidgetClass. Когда компонент уничтожается, он уничтожает виджет вместе с собой.

3.2.3.2.4.3. WBP_BaseBlip

Это абстрактный класс виджета, который должны наследовать все виджеты блипов. Данный виджет непосредственно зависит от `BP_BlipComponent` и, по сути, реализовывает логику свойств этого компонента.

3.2.3.2.4.4. WBP_CharacterBlip

Это виджет блипа персонажа. Наследует `WBP_BaseBlip`. При инициализации и при изменении связанным персонажем команды меняет свой цвет на цвет команды персонажа. При этом если это блин локального персонажа, то всегда принимает цвет `LocalCharacterColor` (сейчас зеленый). Также если `HideEnemyPlayers` = true, то скрывается если это блин вражеского персонажа. Когда связанный с блипом персонаж умирает перед уничтожением виджета проигрывается анимация повышения прозрачности блипа (он медленно исчезает).

3.2.3.2.4.5. WBP_PointBlip

Этот блин связан с игровой точкой. Работает достаточно схоже с `BP_FlagIndicator`, но в отличие от него не реализует интерфейс `BPI_Indicator`, а подписывается на event dispatcher `OnCaptureStateChanged` связанной точки. Использует материальный инстанс[13] `MI_PointCaptureStateMinimapBlip`, который наследует материальный инстанс `MI_PointCaptureStateBagelLike`, который наследует материал `M_PointCaptureState`.

3.2.3.2.4.6. WBP_WiresLayer

Этот виджет рендерит сразу все провода. Из-за особенности рендеринга линий на виджетах в Unreal Engine (они игнорируют ZOrder) сделать это по аналогии с `WBP_PointBlip` не представляется возможным. В связи с этим этот виджет располагается статически на ту же панель, куда и располагаются все блипы, однако за счет статического размещения он становится раньше всех остальных блипов (они располагаются динамически, то есть после) в порядке обхода в глубину[18], вследствие рендерится первее всех блипов, что и является желаемым поведением.

3.2.4. Главное меню и многопользовательская система

3.2.4.1. Главное меню

При запуске непосредственно игры (например, из предоставленного бинарного файла) открывается главное меню (рис. 18).

Главное меню

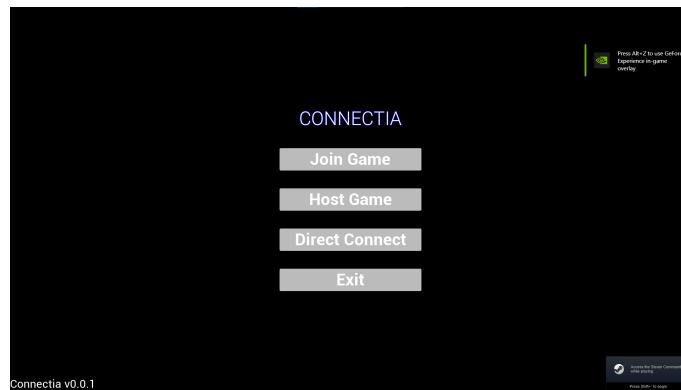


Рис. 18.

На данный момент оно содержит 4 кнопки:

- 1) Join Game – подключение к сессии (подробнее в «Интегрированные со Steam игровые сессии»)
- 2) Host Game – создание игры вместе с сессией (подробнее в «Интегрированные со Steam игровые сессии»)
- 3) Direct connect – подключение к серверу по айпи адресу (подробнее в «Выделенный сервер и подключение к нему»)
- 4) Exit – выход из игры

Также слева внизу отображается текущая версия сборки игры.

3.2.4.2. Выделенный сервер и подключение к нему

Основной (но не единственный) сценарий игры, работа которого гарантируется – запуск выделенного сервера и подключение к нему. Сборка выделенного сервера выполнялась через редактор Unreal Engine, собранный из исходных кодов из официального репозитория. К исходному коду Unreal Engine было применено небольшое изменение, позволяющее вести логирование на выделенном сервере из блюпринтов[19] (это поведение желалось получить в том числе для вывода в логи выделенного сервера информации от «античита», который также реализован на блюпринтах). После запуска выделенного сервера к нему можно подключиться из главного меню (рис. 18) нажатием кнопки «Direct Connect», откроется подменю (рис. 19).

Подменю «Direct Connect»

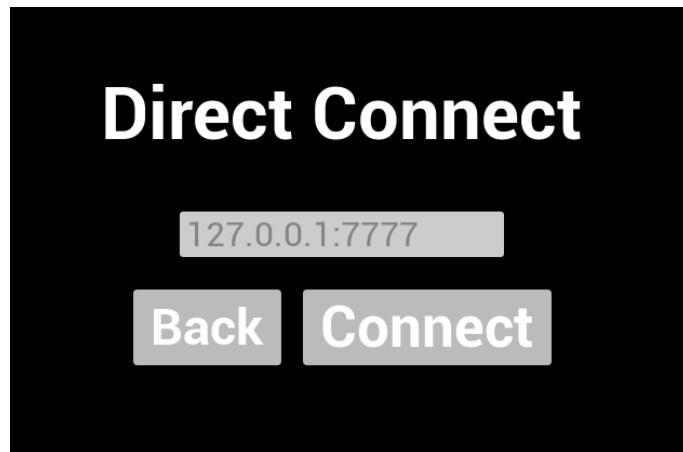


Рис. 19.

Если в поле ввести адрес сервера, например, если сервер запущен на локальной машине, можно ввести 127.0.0.1 и нажать «Connect», произойдет подключение к серверу.

Однако, не обязательно запускать выделенный сервер. Если запускать игру с выключенным Steam или с параметром запуска `--nosteam`, то нажатие на кнопку «Host Game» в главном меню (рис. 18) запустит сервер, к которому также можно будет подключиться через его адрес. Это удобно, например, при игре через локальную сеть. В таком случае хосту не нужно будет запускать помимо игры выделенный сервер. А клиенты смогут подключиться через «Direct Connect», зная IPv4[20] адрес хоста.

3.2.4.3. Интегрированные со Steam игровые сессии

Благодаря Advanced Sessions Plugin[21], в игру также интегрирован Steam. Этот плагин реализует бэкэнд сессий, а фронтэнд реализован мной в виджетах `WBP_HostMenu`, `WBP_JoinMenu`, `WBP_JoinServerListing`. Благодаря этому можно играть друг с другом из любой точки мира, где есть Интернет, не находясь с хостом в одной сети и при этом хосту не обязательно будет иметь «белый»[22] айпи адрес. Для этого и хосту, и клиенту необходим будет скачанный и запущенный клиент Steam. В этом случае в отличие от подраздела «Выделенный сервер и подключение к нему», игра НЕ должна запускаться с параметром запуска `--nosteam`.

Игрок-хост в главном меню (рис. 18) также нажимает кнопку «Host Game». Это создаст новую Steam-сессию через вышеупомянутый плагин.

Игроки-клиенты в том же меню для подключения к хосту нажимают кнопку «Join Game». Начнется сканирование Steam-сессий (рис. 20).

Подменю «Join Game» (сканирование сессий)

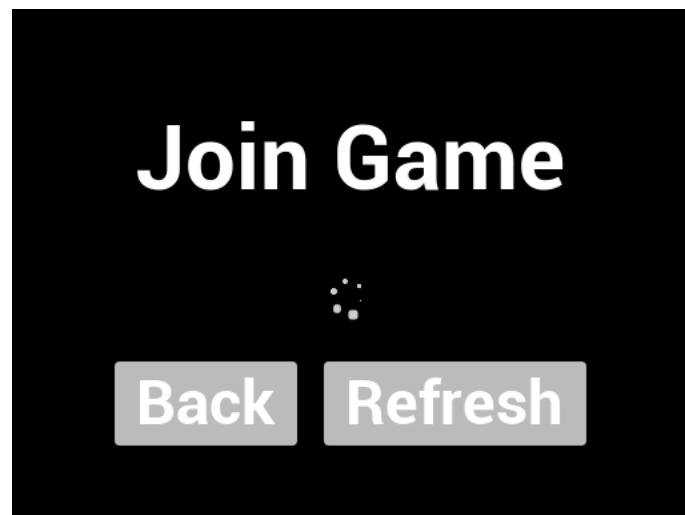


Рис. 20.

Подменю «Join Game» (сессии не найдены)

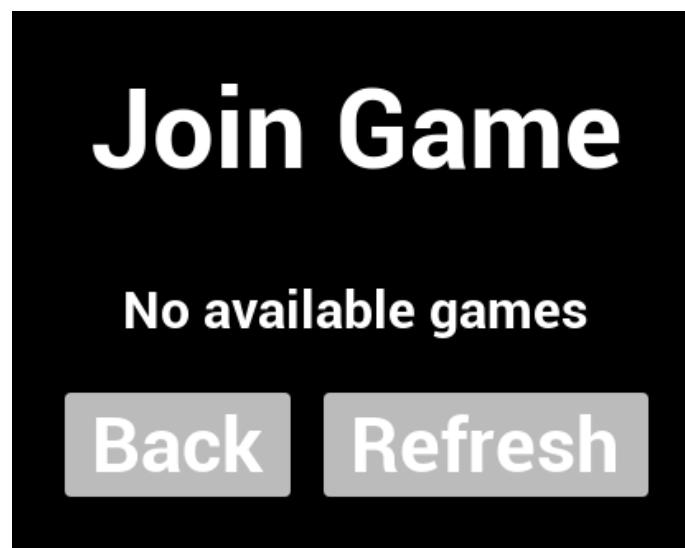


Рис. 21.

Если сессии не найдены (рис. 21), можно нажать кнопку «Refresh». Это запустит сканирование еще раз.

Также стоит заметить, что если у хоста и у клиента версии сборки отличаются (рис. 18 текст слева внизу), клиент НЕ будет видеть сессию хоста.

Если все прошло успешно, клиент увидит список доступных сессий (рис. 22).

Подменю «Join Game» (сессия найдена)

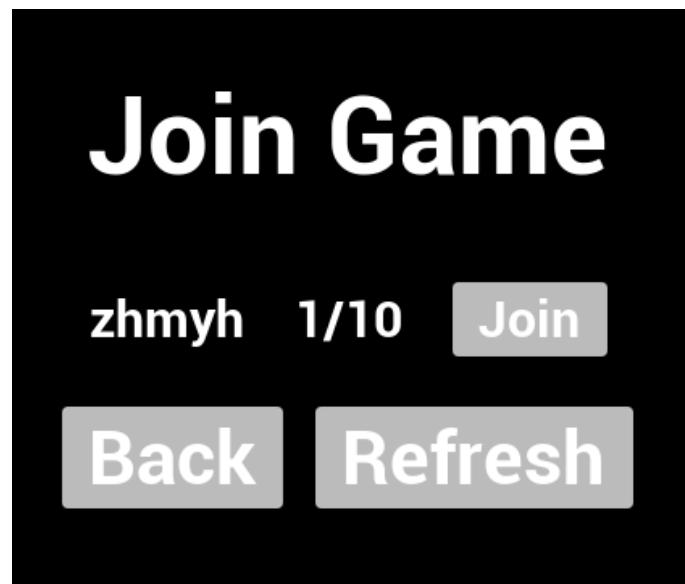


Рис. 22.

Нажатие на кнопку Join напротив определенной сессии подключит игрока к ней (рис. 23) и загрузит карту хоста.

Подменю «Join Game» (подключение к сессии)

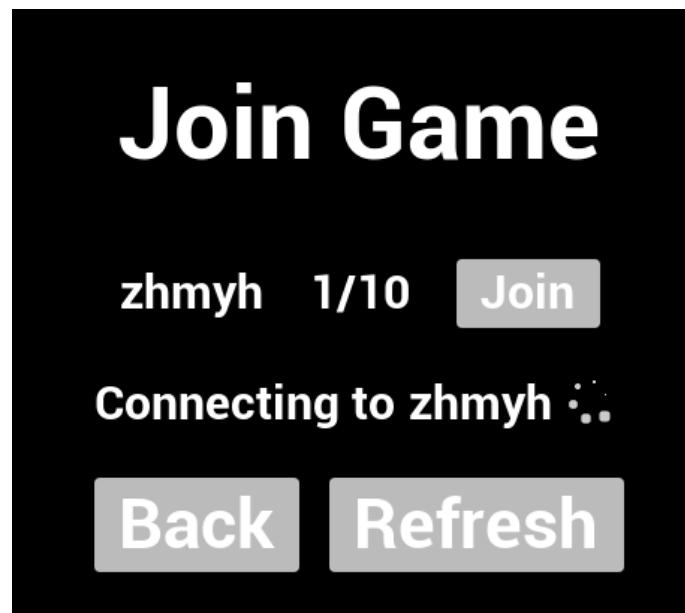


Рис. 23.

3.2.5. Прочее

3.2.5.1. Меню «паузы»

Также в игре присутствует меню «паузы» (конечно, это не совсем можно назвать паузой, ведь на самом деле игра не останавливается). Вызывается это меню нажатием клавиши Esc в игре (рис. 24). Виджет этого меню размыает экран и добавляет две кнопки:

- 1) Resume – закрыть меню паузы и вернуться в игру
- 2) Exit to menu – закрыть меню паузы и вернуться в главное меню

Меню «паузы»



Рис. 24.

3.3. Описание и обоснование выбора метода организации входных и выходных данных

3.3.1. Входные данные

Входными данными для данной программы являются нажатия на клавиатуру и клавиши мыши, а также движения мыши. Описание и обоснование выбора метода организации входных данных подробно указаны для каждой подсистемы в пункте 3.1.

3.3.2. Выходные данные

Выходными данными для данной программы является вывод изображения на экран и воспроизведение звуков. Описание и обоснование выбора метода организации выходных данных подробно указаны для каждой подсистемы в пункте 3.1.

3.4. Описание и обоснование выбора состава технических и программных средств

3.4.1. Unreal Engine

Unreal Engine был выбран для реализации этой программы, поскольку он предоставляет огромный инструментарий для разработки в особенности многопользовательских игр, а также имеет обширную библиотеку бесплатных ассетов.

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Предполагаемая потребность

Предполагаемыми пользователями данного приложения является целевая аудитория, которая увлекается компьютерными играми в жанрах «шутер» или «стратегия».

4.2. Ориентировочная экономическая эффективность

Разрабатываемый продукт подразумевает базовую реализацию, которая в дальнейшем будет улучшаться и дополняться функционалом. Проводя поверхностное сравнение с некоторыми аналогами в пункте «Экономические преимущества разработки по сравнению с аналогами», можно выделить хотя бы несколько преимуществ у разрабатываемой программы. Таким образом, беря во внимание экономическую успешность многих аналогов, при небольшом бюджете на разработку данного продукта можно ожидать некоторую экономическую эффективность.

4.3. Экономические преимущества разработки по сравнению с аналогами

Разработка игры шутера на Unreal Engine 5 может принести следующие экономические преимущества по сравнению с аналогами, такими как Counter Strike, diep.io, Apex Legends, Valorant, Unreal Tournament:

- 1) Снижение затрат на разработку - Unreal Engine 5 предлагает разнообразные инструменты для разработки игр, что может упростить и ускорить процесс создания игры и снизить затраты на разработку.
- 2) Высокое качество графики - Unreal Engine 5 позволяет создавать игры с высоким качеством графики, что может привлечь большую аудиторию и увеличить прибыль.
- 3) Возможность создания многопользовательской игры может привлечь большую аудиторию и увеличить доход разработчиков за счет продаж игры и платных дополнений.
- 4) Возможность расширения игры - Unreal Engine 5 позволяет легко добавлять новые уровни, механики и другие элементы в игру, что может увеличить доход за счет продаж дополнений и расширений игры.
- 5) Широкие возможности для разработки игровых механик - Unreal Engine 5 позволяет разработчикам создавать разнообразные игровые механики, такие как система захвата игровых точек, боевая система игры и система развития персонажа. Это может привести к большей уникальности игры и повысить ее привлекательность для игроков.

ПЕРЕЧЕНЬ ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Полигональная сетка. — https://en.wikipedia.org/wiki/Polygon_mesh.
2. Рендеринг. — [https://en.wikipedia.org/wiki/Rendering_\(computer_graphics\)](https://en.wikipedia.org/wiki/Rendering_(computer_graphics)).
3. Unreal Engine Actors. — <https://docs.unrealengine.com/5.1/en-US/actors-in-unreal-engine/>.
4. Ragdoll. — https://en.wikipedia.org/wiki/Ragdoll_physics.
5. Remote procedure call (удаленный вызов процедуры). — https://en.wikipedia.org/wiki/Remote_procedure_call.
6. Unreal Engine репликация свойств. — <https://docs.unrealengine.com/5.1/en-US/property-replication-in-unreal-engine/>.
7. Unreal Engine LineTrace. — <https://docs.unrealengine.com/5.1/en-US/using-a-single-line-trace-raycast-by-channel-in-unreal-engine/>.
8. Названия углов в авиации. — https://en.wikipedia.org/wiki/Aircraft_principal_axes.
9. Нормальное распределение. — https://en.wikipedia.org/wiki/Normal_distribution.
10. Unreal Engine Animation Notifies. — <https://docs.unrealengine.com/5.1/en-US/animation-notifies-in-unreal-engine/>.
11. Unreal Engine Assets. — <https://docs.unrealengine.com/5.1/en-US/asset-management-in-unreal-engine/>.
12. Unreal Engine Materials. — <https://docs.unrealengine.com/5.1/en-US/unreal-engine-materials/>.
13. Unreal Engine Instanced Materials. — <https://docs.unrealengine.com/5.1/en-US/instanced-materials-in-unreal-engine/>.
14. Unreal Engine PlayerState. — <https://docs.unrealengine.com/5.1/en-US/API/Runtime/Engine/GameFramework/APlayerState/>.
15. Unreal Engine PlayerStart. — <https://docs.unrealengine.com/5.1/en-US/player-start-actor-in-unreal-engine/>.
16. Unreal Engine Components. — <https://docs.unrealengine.com/5.1/en-US/components-in-unreal-engine/>.
17. Unreal Engine Event Dispatchers. — <https://docs.unrealengine.com/5.1/en-US/event-dispatchers-in-unreal-engine/>.
18. Depth-first search. — https://en.wikipedia.org/wiki/Depth-first_search.
19. Unreal Engine blueprints. — <https://docs.unrealengine.com/5.1/en-US/introduction-to-blueprints-visual-scripting-in-unreal-engine/>.
20. IPv4. — https://en.wikipedia.org/wiki/Internet_Protocol_version_4.
21. Unreal Engine Advanced Sessions Plugin. — <https://forums.unrealengine.com/t/advanced-sessions-plugin/30020>.
22. «Белый» айпи адрес. — <https://192-168-1-1.ru/ip-address-classify/>.

ПРИЛОЖЕНИЕ

Описание и функциональное назначение классов

Класс	Назначение
BPFL_GlobalConstants	Содержит глобальные константы, такие как <code>GAME_VERSION</code> (версия игры) и <code>MAX_PLAYERS</code> (максимальное количество игроков в сессии).
BPFL_MaterialHelpers	Содержит вспомогательные функции для материалов. Например, преобразование всех материалов меша или одного материала меша в <code>DynamicMaterialInstance</code> .
BPFL_MathLibrary	Содержит некоторые математические функции, например, сэмплирование нормального распределения и обратная линейная интерполяция.
BPML_ActorComponentHelpers	Содержит вспомогательные макросы для <code>ActorComponent</code> .
BPML_ActorHelpers	Содержит вспомогательные макросы для <code>Actor</code> .
BPML_Anticheat	Содержит вспомогательные макросы для античита.
BPML_Limits	Содержит константные пределы, например, максимальное значение типа <code>Integer</code> .
BPML_PawnHelper	Содержит вспомогательные макросы для <code>Pawn</code> .
BPML_PlayerControllerHelper	Содержит вспомогательные макросы для <code>PlayerController</code> .
EditorMaxFpsSetterUtility	Повышает лимит кадров в секунду при старте редактора Unreal Engine.
BP_ConnectiaCharacter	Содержит логику персонажа.
BP_DeathCamera	Содержит логику камеры смерти.
PM_Damageable	Базовый класс физических материалов, способных получать урон.
BP_GI_Connectia	<code>GameInstance</code> игры.
BP_GM_ConnectiaBase	Базовый <code>GameMode</code> .
BP_GM_ConnectiaLobby	<code>GameMode</code> лобби.
BP_GM_ConnectiaMain	<code>GameMode</code> основного режима игры.
BP_GM_ConnectiaMainMenu	<code>GameMode</code> главного меню игры.
BP_GS_ConnectiaBase	Базовый <code>GameState</code> .
BP_GS_ConnectiaLobby	<code>GameState</code> лобби.
BP_GS_ConnectiaMain	<code>GameState</code> основного режима игры.
BP_PC_ConnectiaBase	Базовый <code>PlayerController</code> .
BP_PC_ConnectiaLobby	<code>PlayerController</code> лобби.

Класс	Назначение
BP_PC_ConnectiaMain	PlayerController основного режима игры.
BP_PC_ConnectiaMainMenu	PlayerController главного меню игры.
BP_PS_ConnectiaBase	Базовый PlayerState.
BP_PS_ConnectiaMain	PlayerState основного режима игры.
BP_GameOverDamageType	Урон, получаемый персонажами, когда их команда проиграла игру.
BP_InstantDamageType	Урон, получаемый персонажами, при попадании в них из оружия.
BP_LobbyPlayerStart	PlayerStart в лобби.
BPI_Initializable	Интерфейс, который реализуют некоторые классы, которые содержат в себе логику инициализации. Необходим для инициализации объектов в строго требуемом порядке.
BP_BaseMainPoint	Базовый класс главной игровой точки.
BP_BasePoint	Базовый класс игровой точки.
BP_ColliderCaptureTriggerComponent	Компонент, позволяющий начать захват игровой точки, путем захода в её триггер.
BP_FlagIndicator	Флаг-индикатор состояния захвата игровой точки.
BP_MainPoint	Главная игровая точка.
BP_PointsManagerComponent	Менеджер игровых точек и проводов.
BP-RegularPoint	Обычная игровая точка.
BP_Wire	Провод, соединяющий две игровые точки.
BPI_Indicator	Интерфейс, который реализуют индикаторы состояния захвата игровой точки.
PDA_Team	PrimaryDataAsset команды.
BP_Testing_ThirdPerson_Character	Персонаж для тестирования третьего лица.
BP_Testing_ThirdPerson_GameMode	Игровой режим для тестирования третьего лица.
BP_Testing_ThirdPerson_Rifle	Винтовка для тестирования третьего лица.
BP_BaseWeapon	Базовый класс оружия.
BP_Rifle	Винтовка.
BP_BlipComponent	Компонент блипа (изображения на миникарте).
AN_FootPlant_Base	Базовый класс AnimNotify'я шагов.
AN_FootPlant_Left	AnimNotify левого шага.
AN_FootPlant_Right	AnimNotify правого шага.
AN_PlayWeaponMontage	AnimNotify для проигрывания анимации оружия, монтаж меша персонажа которого проигрывается.
AN_Reload	AnimNotify для перезарядки оружия.
WBPI_CharacterOwned	Интерфейс, который реализуют виджеты персонажа.

Класс	Назначение
WBPI_PlayerOwned	Интерфейс, который реализуют виджеты игрока.
WBP_Ammo	Виджет количества патронов.
WBP_Blackscreen	Виджет эффекта затемнения экрана при смерти персонажа.
WBP_Character	Виджет игрока.
WBP_Crosshair	Виджет прицела.
WBP_GameMain	Интерфейс, который реализуют виджеты состояния игры.
WBP_HealthBar	Виджет полоски здоровья.
WBP_LobbyStatus	Виджет состояния голосования за начало матча в лобби.
WBP_PlayerBase	Виджет игрока в базовом игровом режиме.
WBP_PlayerMain	Виджет игрока в основном игровом режиме.
WBP_MainGameOver	Виджет уведомления об окончании матча в основном игровом режиме.
WBP_DirectConnectMenu	Виджет главного меню для подключения к серверу по его адресу.
WBP_GameVersion	Виджет главного меню для отображения текущей версии игры.
WBP_HostMenu	Виджет главного меню для хоста новой игры и создания многопользовательской сессии.
WBP_JoinMenu	Виджет главного меню для поиска многопользовательских сессий.
WBP_JoinServerListing	Виджет главного меню для подключения к многопользовательской сессии.
WBP_MainMenu	Виджет главного меню.
WBP_BaseBlip	Базовый виджет блипа.
WBP_CharacterBlip	Виджет блипа персонажа.
WBP_Minimap	Виджет миникарты.
WBP_PointBlip	Виджет блипа игровой точки.
WBP_WiresLayer	Виджет проводов на миникарте.
WBP_PauseMenu	Виджет меню «паузы».

Лист регистрации изменений