



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ

Робототехника и комплексная автоматизация (РК)

КАФЕДРА

Системы автоматизированного проектирования (РК6)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ
НА ТЕМУ:

***«Разработка и интеграция сетевых компонентов
многопользовательской игры на Unreal Engine 4»***

Студент РК6-73Б

(Подпись, дата)

Боженко Д. В.

И.О. Фамилия

Руководитель курсового проекта

(Подпись, дата)

Витюков Ф.А.

И.О. Фамилия

2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой РК6
А.П. Карпенко

« ____ » _____ 2022 г.

ЗАДАНИЕ на выполнение научно-исследовательской работы

по теме: Разработка и интеграция сетевых компонентов многопользовательской игры на Unreal Engine 4

Студент группы РК6-73Б

Боженко Дмитрий Владимирович
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.) учебная
Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения НИР: 25% к 5 нед., 50% к 11 нед., 75% к 14 нед., 100% к 16 нед.

Техническое задание: Изучить принципы создания современных многопользовательских игр. На основе полученных знаний реализовать элементы сетевых компонентов и интегрировать их в простую многопользовательскую игру. Изучить принципы работы с сессиями в движке для дальнейшей интеграции авторизации пользователя на уровне приложения.

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 14 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

Дата выдачи задания «10» сентября 2021 г.

Руководитель НИР

(Подпись, дата)

Витюков Ф.А.

И.О. Фамилия

Студент

(Подпись, дата)

Боженко Д. В.

И.О. Фамилия

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	4
ВВЕДЕНИЕ	5
1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	6
1.1. Программирование сетевых игр	6
1.1.1 Структура мультиплеера	6
1.1.2. Репликация	7
1.1.3. Важнейшие классы в UE 4 для реализации мультиплеера	8
1.1.4. RPC	10
1.1.5. Сетевые роли	12
1.2. Игровые сессии	13
1.2.1 Сессии в рамках UE 4. OnlineSubsystem	14
2. ПРАКТИЧЕСКАЯ ЧАСТЬ	17
2.1. Описание классов, используемых в проекте сетевой игры	17
2.2. Реализация здоровья игрока и его очков	17
2.3. Реализация динамического объявления фрагов и таблицы очков	19
2.4. Реализация респауна игроков	22
ЗАКЛЮЧЕНИЕ	23
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	24

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ЯП — язык программирования.

Многопользовательская игра — режим компьютерной игры, в котором играет более одного пользователя по сети Интернет.

Движок — это программный фреймворк, предназначенный в первую очередь для разработки видеоигр и обычно включающий соответствующие библиотеки, и программы поддержки.

UE 4 — движок Unreal Engine 4.

Клиент — машина, которая получает информацию об игровом мире через сервер и на которой происходит отрисовка игрового процесса.

Сервер — мощная вычислительная машина, через которую происходит обмен информацией об игровом мире без отрисовки графики и воспроизведения звуков.

RPC — Remote procedure call — инструмент в ЯП C++ в UE 4, используемый для обмена информацией между клиентами и сервером об игровом мире.

Удаленный игрок — игрок, который находится на другой машине в пределах одной игровой сети.

Локальный игрок — игрок, который находится на локальной машине.

LAN — локальная вычислительная сеть, где все участники находятся, как правило, в пределах одной ограниченной территории.

AActor — один из основных классов в UE 4, являющийся базовым для всех остальных классов, представленных в игровом мире.

OnlineSubsystem — кроссплатформенная система, позволяющая использовать современные возможности многопользовательских игр.

ВВЕДЕНИЕ

Рынок видеоигр стремительно развивается с каждым годом. На сегодняшний день рынок игр во всем мире является одним из самых больших сегментом мирового рынка цифрового контента, ежегодно генерируя многомиллиардные доходы и привлекая огромную аудиторию. Наибольшая доля в структуре российского рынка приходится на сегмент онлайн-игр. По данным *Mail.ru Group*, в 2019 году его объем увеличился на 9% и составил 56,7 млрд рублей (около \$1 млрд).

Среди всех жанров игр на данный момент самыми популярными являются *ММО* (массовые многопользовательские онлайн игры), которые использует *Real-Time Multiplayer*. *Real-Time Multiplayer* — это тот режим игры, в котором каждый пользователь получает и отправляет данные об игровом мире с выделенного игрового сервера несколько десятков раз за отведенный промежуток времени. Данное понятие называется тикрейт, т. е. какое количество запросов сервер может обрабатывать за установленные промежуток времени.

Целью данной работы является изучение концепций создания многопользовательских игр. С использованием полученных знаний требуется реализовать шаблон сетевой игры, на основе которого в дальнейшем можно будет строить более комплексные проекты. Предполагается поддержка игры по сети Интернет нескольких игроков одновременно, а также развертывание и эксплуатация сетевой инфраструктуры с использованием современных облачных технологий и других методов.

Данная цель является актуальной, так как изучение концепции создания многопользовательских игр является необходимым условием освоения рынка видеоигр, которые в свою очередь стремительно развиваются и набирают большую популярность в сфере информационных технологий.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Программирование сетевых игр

Соединение в многопользовательских играх происходит по клиент-серверной модели, когда несколько клиентов подключаются к выделенному серверу и через него передают друг другу информацию об игровом мире.

На этапе разработки многопользовательской игры, важно понимать, как и с помощью каких инструментов игрового движка реализовано сетевое взаимодействие между игроками, какие при этом создаются объекты классов, сколько копий каждого класса создается и на какой машине они находятся.

1.1.1 Структура мультиплеера

В UE 4 существует четыре основных мода многопользовательской игры:

1. Standalone — автономная игра, где сам экземпляр игры является сервером. Игра, запущенная в таком режиме не принимает никаких подключений от удаленных игроков.
2. Client — режим игры, в котором она имеет роль клиента, и работает только при подключении к игровому серверу.
3. Listen-Server — режим игры, в котором сервером становится один из клиентов и на котором размещена сетевая многопользовательская сессия. В таком режиме игра как принимает запросы от удаленных игроков, так и содержит своих локальных игроков. Такой режим многопользовательской игры хорошо подходит для развертывания кооперативных игр, где все игроки находятся в пределах LAN и сетевые взаимодействия осуществляются по P2P-сети.
4. Dedicated-Server — режим игры, в котором сервер расположен на отдельно выделенной машине. Экземпляр игры, запущенный на выделенном сервере принимает запросы от удаленных игроков, но сам не содержит никаких локальных игроков. Следовательно, на

таком экземпляре игры отсутствуют такие функции, ориентированные на игроков, как отрисовка графики, вывод звуков и пользовательский ввод. Данное решение является основным для большинства многопользовательских игр, где есть много игроков, так как выделенные сервера обладают большой вычислительной мощностью и обеспечивают безопасность от обмана и читерства.

1.1.2. Репликация

Репликация — это синхронизация информации об игровом мире между сторонами. Другими словами, репликация — это механизм, который создает множество копий одного и того же объекта. Объект и его копии хранятся на разных машинах (на клиенте и на сервере), за счет чего между игроками, находящимися на разных игровых машинах, происходит синхронизация информации об одном и том же объекте. Реплицировать можно переменные, события и объекты. Передача измеренной информации об объекте может осуществляться от сервера к одному клиенту, от любого клиента к серверу, и от сервера ко всем клиентам.

В UE 4 для реализации репликации объектов используется возможность класса `AActor`, так как все объекты, представленные в игровом мире являются сущностями этого класса.

Важно понимать, что игрок видит перемещение другого игрока у себя на локальной машине именно за счет репликации. Это происходит за счет того, что при репликации объекта, его копии хранятся на локальных машинах других игроков, а также на сервере. При перемещении игрок посылает серверу свои координаты, тот, в свою очередь, анализируя эти координаты, передает их копиям этого игрока, которые расположены на остальных клиентах. Следовательно, когда игрок видит перемещение другого игрока у себя на локальной машине, он видит перемещение копии этого игрока, которая полностью управляется сервером.

Реплицирование переменных в UE 4 также осуществляется особым образом. Переменную, которую надо реплицировать помечается с помощью макроса `UPROPERTY(Replicated)`. Важно понимать сам механизм реплицирования. Когда переменная, находящаяся на сервере, изменяет свое значение, то ее значение также будет изменено на всех клиентах. При этом также возможно указать функцию, которая будет выполняться при изменении реплицированной переменной через флаг `ReplicatedUsing`. Важно заметить, что данная функция будет выполняться только при изменении реплицированной переменной и только на клиентах, и никогда не будет выполняться на сервере.

1.1.3. Важнейшие классы в UE 4 для реализации мультиплеера

Для базовой реализации простой многопользовательской игры в UE 4 существует несколько базовых классов:

1. `AGameMode` — самый важный класс, который отвечает за правила игры. Важно знать, что экземпляр такого класса находится только на сервере. Чтобы избежать читерства, все действия, связанные с игровой логикой запрашиваются через сущность этого класса. Важно понимать, что попытка получить доступ к сущности класса `AGameMode` с клиента будет безуспешной.
2. `AGameState` — класс, который содержит в себе информацию о текущем состоянии игры, например, о количестве подключенных к сессии игроков. Сущность данного класса располагается на сервере, а также его копия располагается на каждом из клиентов. Таким образом, сущность данного класса самая важная, которая необходима для передачи общей информации между сервером и клиентами.
3. `APlayerState` — класс, который содержит в себе всю текущую информацию об игроке, подключенном к игровой сессии. Сущность данного класса находится на каждом клиенте, а также копия сущности класса `APlayerState` каждого клиента находится на

сервере. Следовательно, сервер знает о сущности APlayerState каждого клиента, а клиент знает о существовании только собственного класса APlayerState.

4. APlayerController — класс, который остается за игроком на протяжении всей игровой сессии. С помощью APlayerController клиента можно легко управлять интерфейсом игрока, когда необходимо освежить новую информацию о состоянии объектов игрового мира, например, изменение очков, изменение здоровья и другое. Распределение по клиентам и серверу такое же, как у APlayerState.
5. ANUD — класс, который существуют только на клиенте, который является владельцем данного класса. ANUD используется для управления виджетами клиентами, управлением данным классом можно осуществлять с клиента, либо же с сервера с помощью RPC.
6. APawn — класс, который представляет из себя объект на сцене, которым управляет игрок либо же сервер. Данный класс является производным от класса AActor, поэтому возможно его реплицирование. Каждый клиент и сервер знает о существовании о каждом объекте класса APawn.

Ниже на рисунке 1 представлена схема репликации классов.

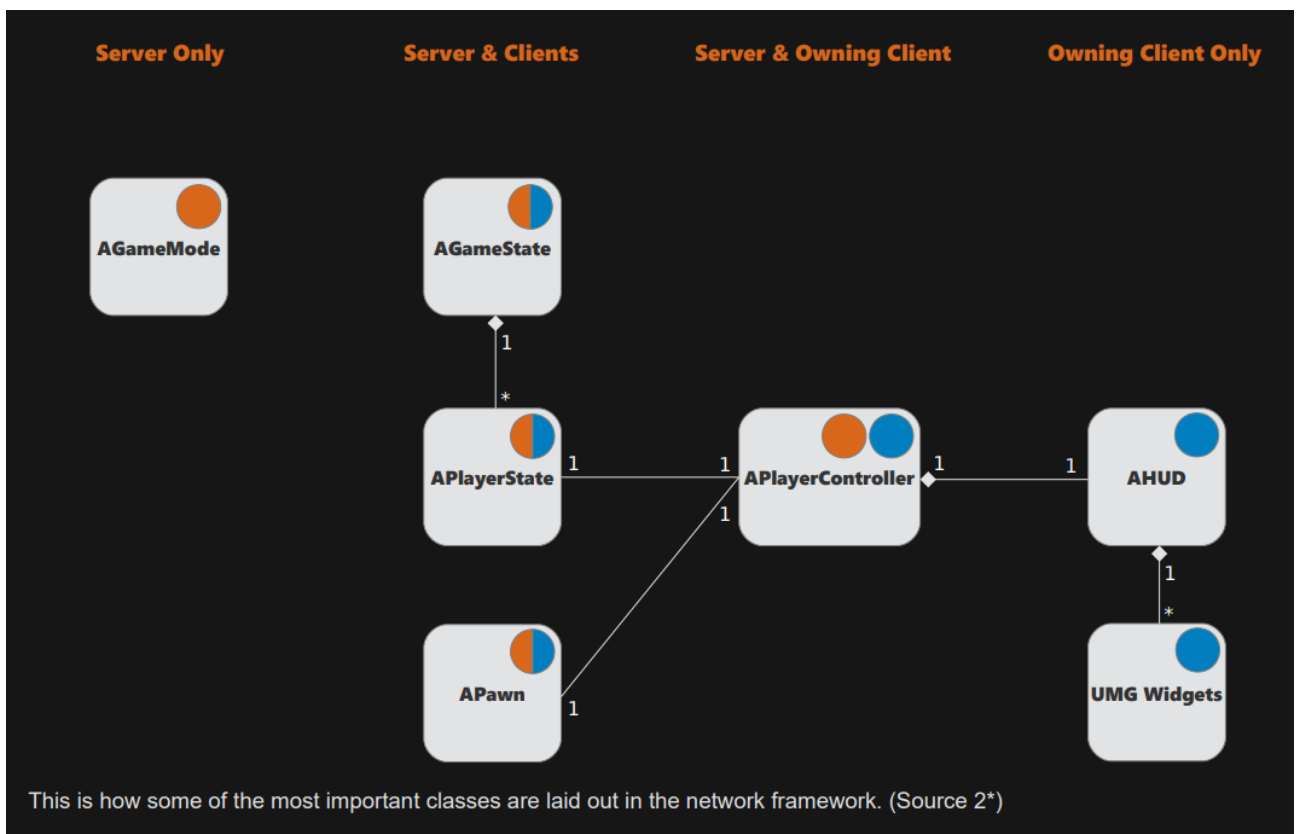


Рисунок 1 — Схема репликации основных классов UE 4

1.1.4. RPC

RPC — это особые функции в UE 4, которые вызываются на одной машине, а выполняются на другой. RPC помечаются через макрос `UFUNCTION()`. Всего существует три вида RPC:

1. **Client** — данный тип RPC, как правило, вызывается с сервера. Модификатор **Client** говорит о том, что данная функция будет выполнена только на том клиенте, который владеет данной RPC-функцией. Данный вид RPC хорошо подходит для обновления виджета в HUD клиента, когда на сервере произошло изменение реплицированной переменной.

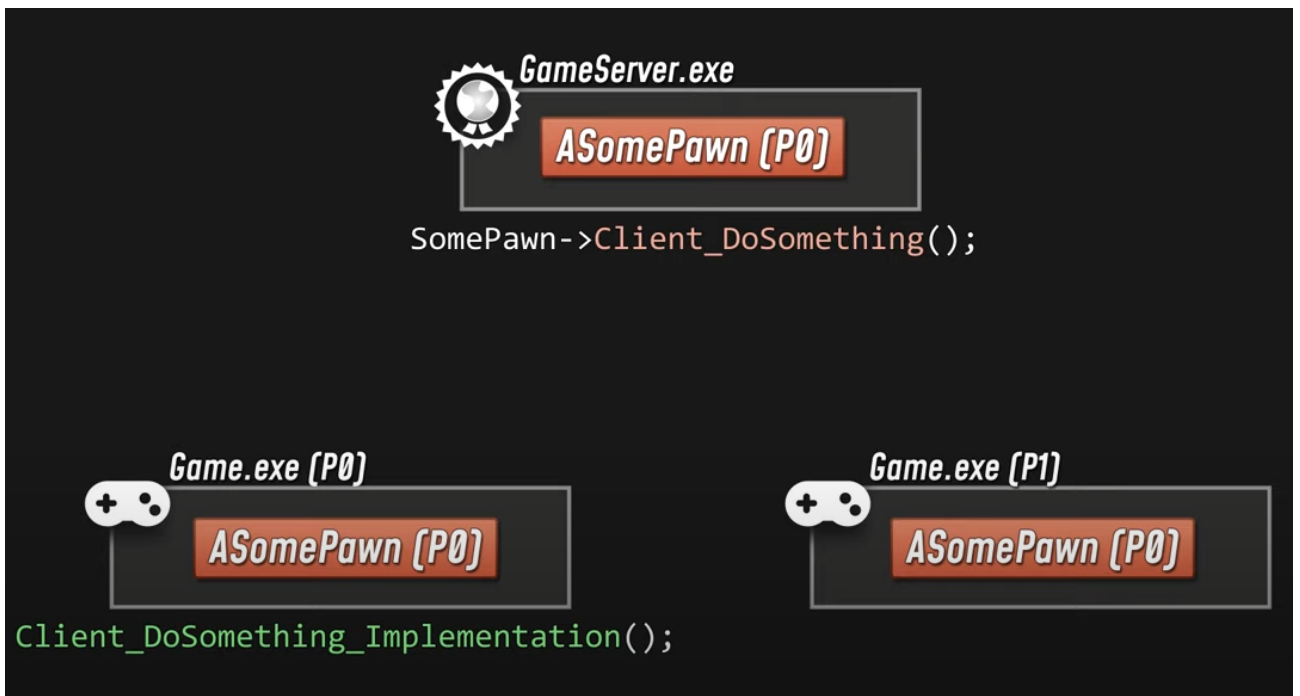


Рисунок 2 — Схема вызова и выполнения Client RPC

2. Server — данный тип RPC вызывается с клиента, а выполнение функции производится только на сервере.

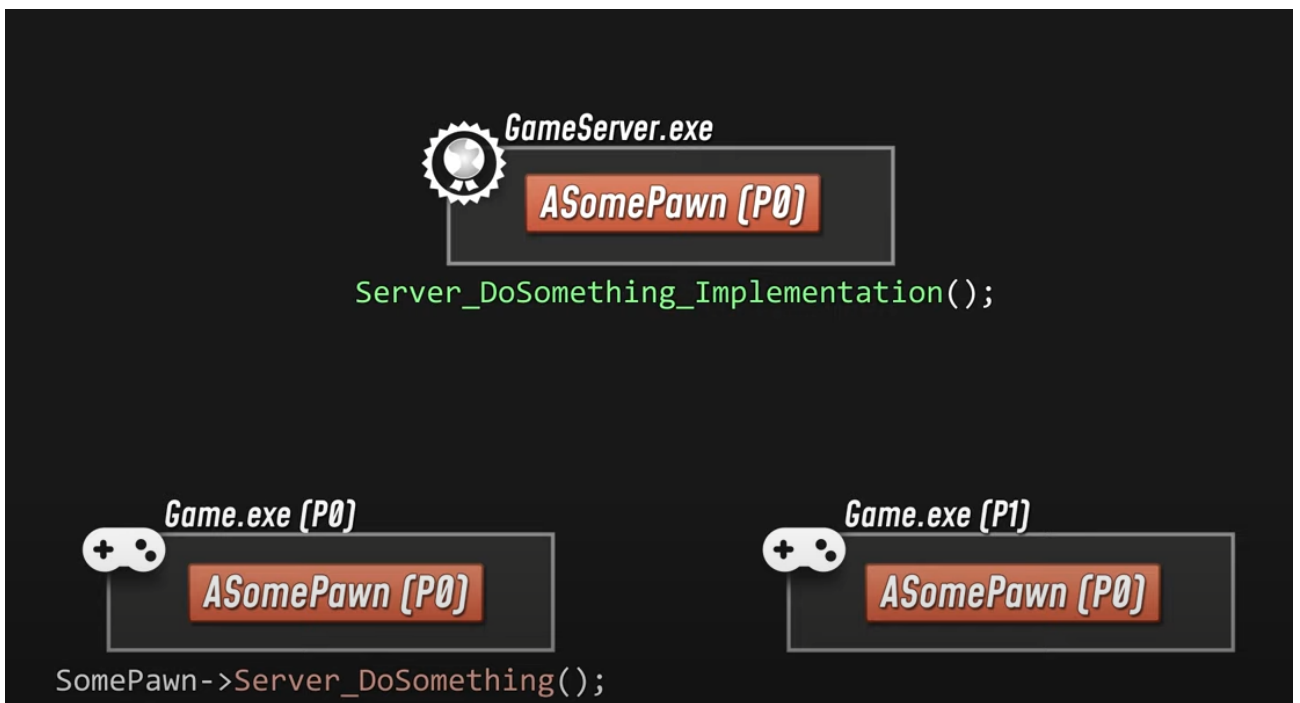


Рисунок 3 — Схема вызова и выполнения Server RPC

3. NetMulticast — данный тип RPC всегда вызывается с сервера и будет выполнен как на всех клиентах, так и на сервере. Такой тип RPC хорошо подходит, например, для проигрывания анимации после смерти игрока.

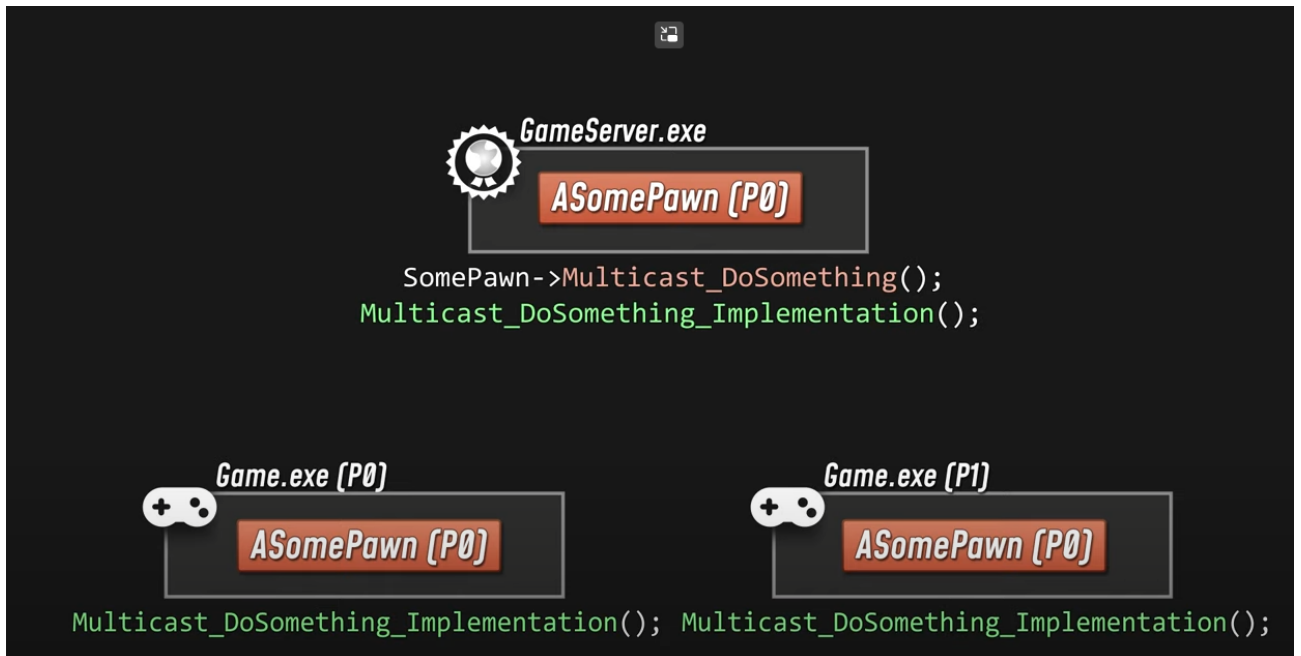


Рисунок 4 — Схема вызова и выполнения NetMulticast RPC

1.1.5. Сетевые роли

В UE 4 существует 4 основных сетевых роли, которые может иметь *AActor*:

1. *ROLE_AutonomousProxy* — сетевая роль, которая показывает, что данный *AActor* находится на клиенте и управляется живым игроком.
2. *ROLE_SimulatedProxy* — сетевая роль, которая показывает, что данный *AActor* находится на клиенте и управляется сервером.
3. *ROLE_Authority* — сетевая роль, которая определяет, что данный *AActor* или его копия находится на сервере.
4. *ROLE_None* — сетевая роль, которая дается *AActor* в случае, если он не обладает ни одной ролью из вышеперечисленных.

На рисунке 5 представлена таблица, с помощью которой можно понять, какую роль получает AActor в зависимости от этого, на какой машине находится он или его копия.

	<i>GameServer.exe</i> NM_DedicatedServer	<i>Game.exe [P0]</i> NM_Client	<i>Game.exe [P1]</i> NM_Client
<i>AGameModeBase</i>	<i>AUTHORITY</i>	[not replicated]	[not replicated]
<i>APlayerController [P0]</i>	<i>AUTHORITY</i>	<i>AutonomousProxy</i>	[not replicated]
<i>APlayerController [P1]</i>	<i>AUTHORITY</i>	[not replicated]	<i>AutonomousProxy</i>
<i>APawn [P0]</i>	<i>AUTHORITY</i>	<i>AutonomousProxy</i>	<i>SimulatedProxy</i>
<i>APawn [P1]</i>	<i>AUTHORITY</i>	<i>SimulatedProxy</i>	<i>AutonomousProxy</i>

Рисунок 5 — Схема распределения сетевых ролей на примере двух игроков

1.2. Игровые сессии

Сессии, также, как и сетевое программирование, является базовым аспектом, с помощью которого строятся многопользовательские игры. С помощью сетевого программирования прорабатывается процесс взаимодействия персонажей друг с другом и с игровым миром в целом. Сессии служат для того, чтобы контролировать подключение игроков к выделенному серверу или хосту.

Сессия — это большая структура данных, которая содержит в себе множество полей, настроек и методов, с помощью которых можно задавать правила подключения игроков к серверу. Базовыми настройками игровой сессии является:

1. Максимальное количество игроков, которые могут одновременно подключиться к игре.

2. Доступность игры из сети Интернет.
3. Возможность использовать лобби.
4. Возможность подключаться к сессии после начала игрового процесса.

Сущность сессии всегда находится на сервере. Клиенты, которые хотят подключиться к серверу, сперва должны подключиться к игровой сессии. Если подключение клиента успешно прошло проверку, ему разрешается присоединиться к игровому серверу. Таким образом роль сессии заключается в том, что она является промежуточным узлом подключения между клиентом и сервером, которые валидирует подключения каждого клиента и управляет ими.

1.2.1 Сессии в рамках UE 4. OnlineSubsystem

В UE 4 есть инструмент, который позволяет запускать и отлаживать многопользовательские игры без создания сессий. Для этого в редакторе UE 4 необходимо выбрать режим запуска игры в секции Multiplayer Options. В качестве параметров необходимо выбрать Number of Players — количество игроков, и параметр Net Mode — режим многопользовательской игры, описанный в пункте 1.1.1. Запуск многопользовательской игры в редакторе UE 4 представлен ниже на рисунке 6.

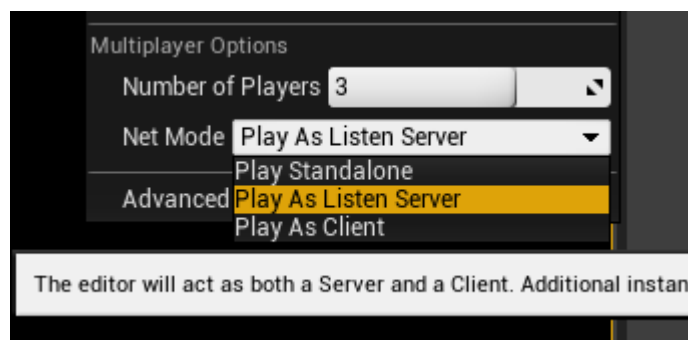


Рисунок 6 — Запуск многопользовательской игры внутри редактора UE 4

Однако такой запуск возможен только внутри редактора UE 4. Для организации многопользовательской игры отдельно от редактора UE 4 необходимы игровые сессии.

В рамках UE 4 существуют отдельные классы, которые позволяют управлять сессиями. Ниже на рисунке 7 представлена схема классов, реализующих механизм сессий в UE 4.

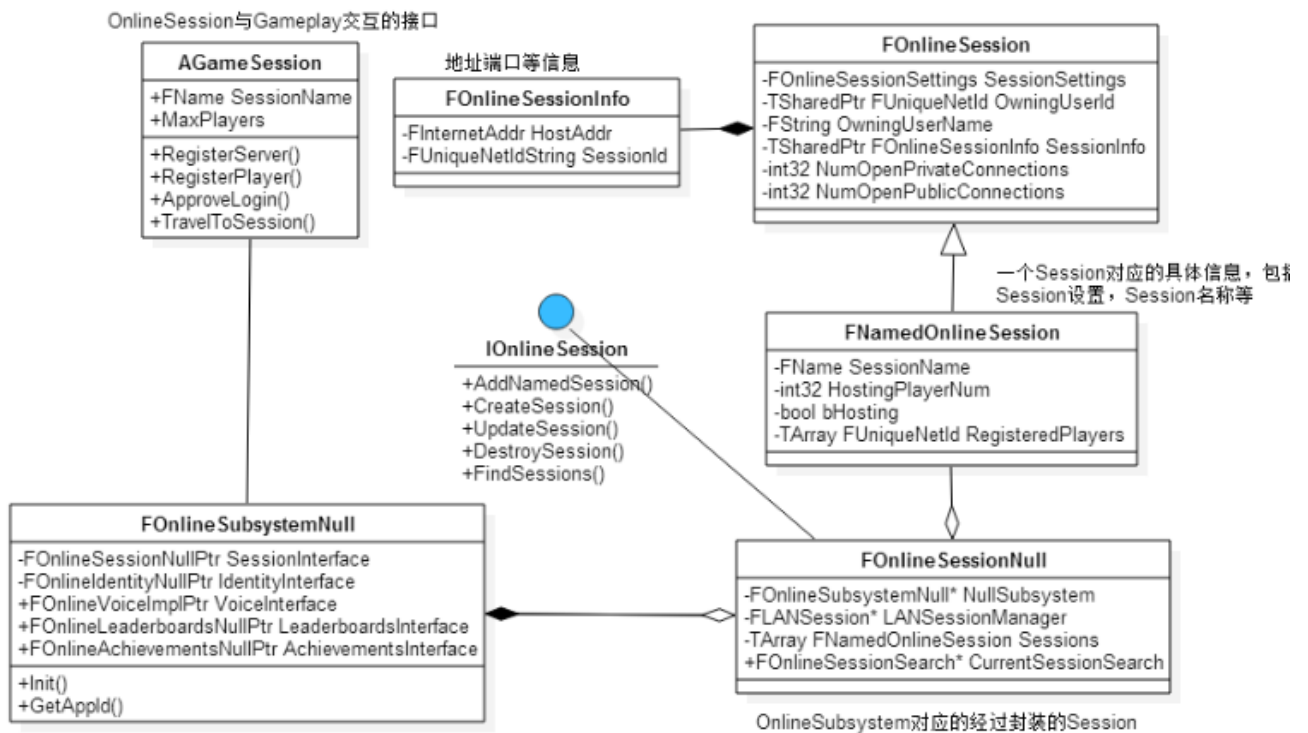


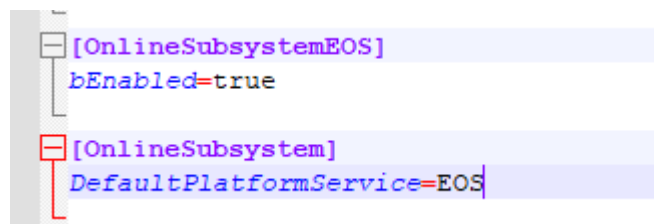
Рисунок 7 — Классы UE 4, используемые для создания сессий

В версиях движка, начиная с UE 4.11 появилась структура OnlineSubsystem, которая напрямую интегрирована в движок и задана как OnlineSubsystem по умолчанию. Данная структура предоставляет интерфейс (SessionInterface), который позволяет работать с сессиями. OnlineSubsystem может предоставлять множество интерфейсов: интерфейс работы с сессиями, интерфейс управления аутентификации пользователей, интерфейс работы с голосовым чатом и многое другое.

В качестве OnlineSubsystem можно использовать OnlineSubsystem движка, а также есть возможность использовать EOS (Epic Online Subsystem), Steam Online Subsystem, GooglePlay OnlineSubsystem и многие другие. OnlineSubsystem в UE 4 предоставляет современные возможности

многопользовательских игр только лишь в пределах LAN, в то время как другие OnlineSubsystem, например, EOS, предоставляют те же самые возможности по сети Интернет, обеспечивая туннельное сетевое соединение между клиентами и сервером. Однако EOS имеет ограниченную поддержку LAN сессий и не может обеспечить подключение игроков, не подключенных к сети Интернет, так как требует обязательной авторизации пользователя через собственные сервисы авторизации. EOS может предоставить игровую сессию, которая видна только для тех клиентов, которые находятся в пределах LAN и скрывает ее видимость для игроков, не находящихся в пределах LAN.

OnlineSubsystem, которая используется в проекте в текущий момент, указывается в настройках игры в файле DefaultEngine.ini в секции OnlineSubsystem (Рисунок 8).



```
[OnlineSubsystemEOS]
bEnabled=true

[OnlineSubsystem]
DefaultPlatformService=EOS
```

Рисунок 8 — Задание OnlineSubsystem в конфигурационном файле проекта DefaultEngine.ini

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Описание классов, используемых в проекте сетевой игры

ALab4Character — производный класс от ACharacter. Сущность данного класса представляет собой объект, который имеет рэгдолл для отображения на сцене, а также кастомные переменные, такие как CurrentHealth — текущее количество здоровья, MaxHealth — максимальное количество здоровья.

ALab4PlayerState — производный класс от класса APlayerState. Класс создан для управления именем игрока, его очками, а также для переопределения виртуальных функций, которые выполняются при изменении реплицированных переменных, встроенных в класс APlayerState.

ALa4PlayerController — производный класс от APlayerContorller. Создан для того, чтобы иметь возможность более гибко управлять HUD игрока.

ALab4HUD — производный класс от AHUD. Данный класс создан для более гибкого управления всеми виджетами игрока.

ALab4GameMode — класс, производный от GameModeBase. Данный класс создан для того, чтобы определять логику игры, такую как запрос респауна игрока, изменение очков и здоровья игрока, а также реализация логики при убийстве игрока.

Перед тем, как использовать свои классы, производные от классов UE 4 необходимо установить их как классы по умолчанию. Это можно сделать в настройках самого движка UE 4, либо же самостоятельно с помощью C++ в конструкторе класса AGameMode.

2.2. Реализация здоровья игрока и его очков

Для реализации здоровья игрока в конечном классе ALab4PlayerState была объявлена переменная float CurrentHealth. С помощью макроса UPROPERTY(ReplicatedUsing) данная переменная была помечена как реплицируемая.

Важно заметить, что переменную здоровья также можно было бы поместить в класс `ALab4PlayerState` игрока. Из-за того, что здоровье игрока необходимо получать быстро и с минимальной задержкой, было принято решение поместить ее непосредственно в сам класс игрока `ALab4Character`, так как для инициализации сущности класса `APlayerState` а также для обращения к его сущности требуется некоторое время, что может образовывать небольшие сетевые задержки.

Важно понимать сам механизм изменения количества здоровья игрока. Здоровье игрока всегда меняется на сервере. Если это клиент, то меняется на его копии, находящейся на сервере. Через функцию UE 4 *bool HasAuthority()* можно определять роль игрока. Когда на сервере произошло изменение здоровья игрока, это изменение сразу же отобразится на всех клиентах, которые используют данную реплицируемую переменную здоровья.

В функции `OnRep_CurrentHealth` происходит изменение количества здоровья только на том клиенте, который владеет данным виджетом.



Рисунок 9 — Виджет здоровья игрока (Healthbar)

Для реализации подсчета очков можно воспользоваться классов `APlayerState`. В данном классе уже реализованы базовые поля такие `Id` игрока, количество очков `Score`, его имя `PlayerName` и другое. Изменение количества очков также, как и здоровье игрока, необходимо совершать с сервера.

Таким образом при изменении количества очков на сервере, данное значение обновится у каждого клиента, у в переопределенной виртуальной

функции `OnRep_Score`, принадлежащей классу `APlayerState`, можно обновить текущее количество очков в HUD игрока.

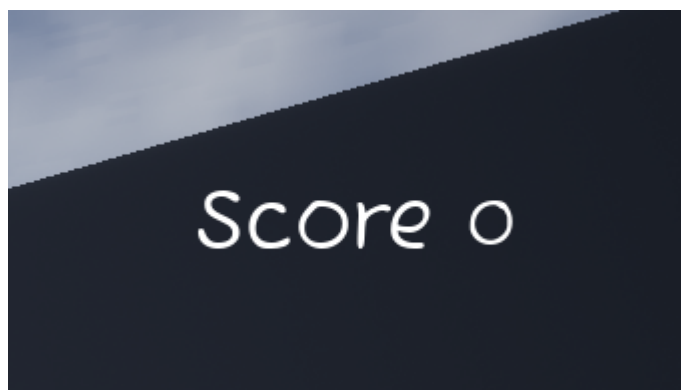


Рисунок 10 — Текущее количество очков в виджете игрока

2.3. Реализация динамического объявления фрагов и таблицы очков

Для динамического объявления фрагов в виджете игрока было принято решение использовать `Client-RPC`. `Client-RPC` — это такое `RPC`, которое вызывается на сервере, а выполняется только на том клиенте, в котором объявлено тело данной функции. `Client-RPC` является хорошим выбором, для решения такой задачи, в отличие от `NetMulticast-RPC`, так как отображение убийств производится с помощью сущности класса `ANHUD`. Данный класс не реплицируется, поэтому данная функция должна выполняться только на тех клиентах, которые обладают сущностью класса `ANHUD`, чтобы реальный игрок видел данные сообщения при обновлении состояния игрового мира.

При очередном фраге, сделанном в игровом мире, на сервере в классе `AGameMode` выполняется функция, которая получает ссылки на игрока, который совершил фраг и которого убили. Далее во вьюпорт игрока выводится динамическое сообщение о совершенном фраге, которое удаляется из вьюпорта спустя заданное количество времени. Пример такого сообщения представлен ниже на рисунке 11.

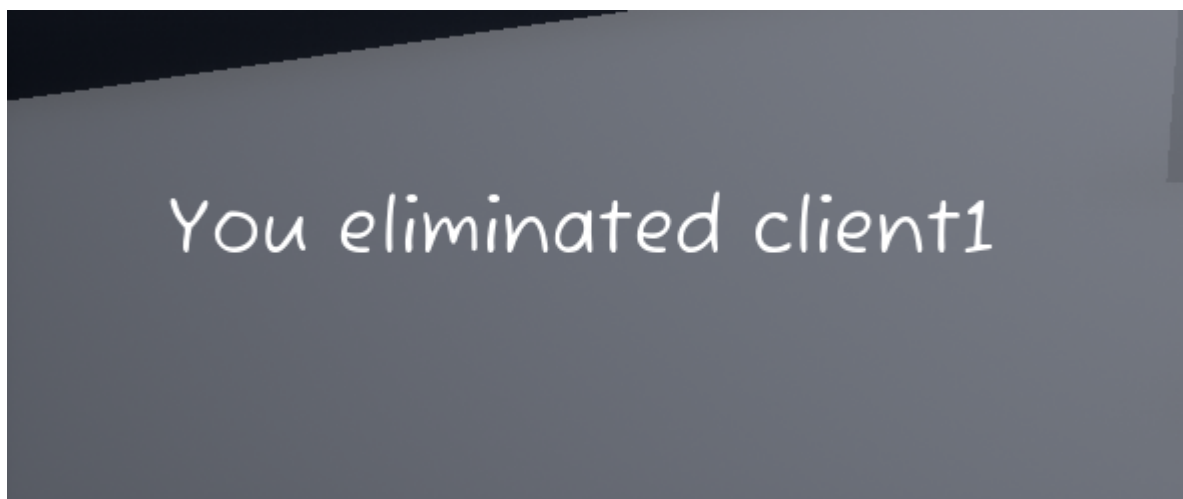


Рисунок 11 — Динамическое сообщение о фраге глазами игрока, совершившего фраг

В итоге у игрока во вьюпорте создается интерфейс, который состоит из нескольких виджетов. Каждый выполняет свой определенный функционал и не зависит от других.

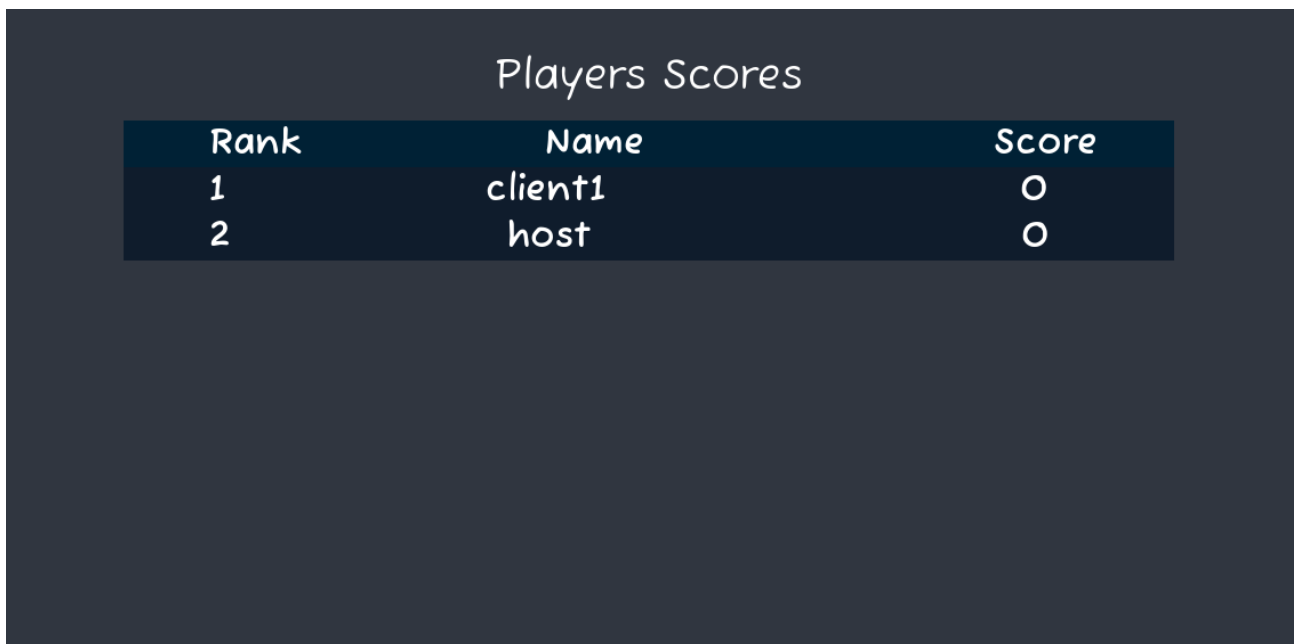


Рисунок 12 — Вьюпорт игрока с созданными в нем виджетами

Также для отображения текущего результата игры был создан виджет в виде таблицы, который появляется при нажатии клавиши Tab. Данное решение используется в современных многопользовательских играх и представляет из себя динамическую таблицу, которая содержит в себе большое количество

информационных полей таких, как имя игрока, его пинг, его количество очков, количество смертей, ассистов и много другой игровой информации.

Для создания аналогичной таблицы было принято решение использовать класс `AGameState`. Данный класс содержит в себе всю информацию об игровом мире, а именно массив всех сущностей `APlayerState TArray<APlayerState*>`. `TArray` — это гибкая структура данных в UE 4. Каждый клиент имеет сущность данного класса, следовательно, каждый игрок имеет доступ к статистике каждого игрока. Ниже на рисунке 13 представлена таблица, в которой отображаются все игроки, их имена, их текущие очки, а также их позиция в игре. Игроки в таблице всегда сортируются в порядке убывания по полю `Score`.



Rank	Name	Score
1	client1	0
2	host	0

Рисунок 13 — Динамическая таблица игроков

2.4. Реализация респауна игроков

Для реализации респауна игрока было принято решение использовать NetMulticast-RPC. NetMulticast-RPC является отличным решением для подобных задач, когда необходимо, чтобы каждый клиент видел изменения игрового мира.

Любые NetMulticast-RPC вызываются на сервере, а их реализация выполнена в классах, принадлежащих клиентам. Алгоритм возрождения игрока можно описать следующим образом:

1. Каждый раз, когда игрок получает урон, необходимо проверять уровень здоровья игрока. Если уровень здоровья игрока упал до нуля, необходимо вызвать функцию, принадлежащую классу AGameMode.
2. Функция, реализованная в классе AGameMode получает референс на APlayerController убитого игрока. Сервер делает респаун убитого игрока, на которого он получил референс. Важно понимать, что если пешка игрока была удалена на сервере, она будет удалена на всех клиентах. Поэтому важно делать респаун игрока именно через сервер.
3. Через указатель на убитого игрока на сервере необходимо вызвать RPC. Как правило, в RPC при респауне игрока вызывается анимация убийства, которая должна быть видна каждому игроку.

ЗАКЛЮЧЕНИЕ

В результате выполнения научной исследовательской работы были получены базовые знания в области сетевого программирования и игровых сессий в рамках UE 4. В области сетевого программирования были изучены такие важные и необходимые темы, как:

1. Режимы мультиплеера
2. Репликации
3. Remote Procedure Calls (RPC)
4. Сетевые роли

Также в рамках работы было изучено, что такое игровые сессии, для чего они используются, принципы работы с игровыми сессиями в рамках UE 4 с помощью OnlineSubsystem.

В результате применения полученных теоретических знаний в практической части были улучшены навыки работы с UE 4 и языком программирования C++, а также выполнено программирование элементов сетевых компонентов мультиплеера и их последующая интеграция в шаблон многопользовательской игры.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Unreal Engine Network Compendium. An Unreal Engine Blog. URL: https://cedric-neukirchen.net/Downloads/Compendium/UE4_Network_Compendium_by_Cedric_eXi_Neukirchen.pdf. Дата обращения 17.10.2022;
2. UE4 Multiplayer Sessions in C++. An Unreal Engine Blog. URL: <https://cedric-neukirchen.net/2021/06/27/ue4-multiplayer-sessions-in-c/>. Дата обращения 01.11.2022;
3. The EOS Online Subsystem (OSS) Plugin. Unreal Engine Documentation. URL: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Online/EOS/>. Дата обращения 02.11.2022;
4. Online Subsystem. Unreal Engine Documentation. <https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Online/>
Дата обращения 02.11.2022;
5. Modifying the DefaultEngine.ini. Epic Games Dev Community. URL: <https://forums.unrealengine.com/t/modifying-the-defaultengine-ini-through-a-utility-widget/482207>. Дата обращения 06.11.2022;
6. Уильям Шериф. Unreal Engine 4.x Scripting with C++ Cookbook / Уильям Шериф, Стивен Уиттл, Джон Доран. — Packt Publishing, 2019 г. — 708 с.
7. Арам Куксон. Unreal Engine 4 Game Development in 24 Hours, Sams Teach Yourself / Арам Куксон, Райан Даулингсока, Клинтон Крамплер. — Москва: Бомбора, 2019 г. — 528 с.
8. Маркус Ромеру. Blueprints Visual Scripting for Unreal Engine 2nd Edition / Маркус Ромеру, Брендэн Сьюэлл. — Packt Publishing, 2019 г. — 380 с.