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*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ НА ТЕМУ

«Разработка сетевых компонентов и их интеграция в шаблон многопользовательской игры на Unreal Engine 4»

Студент РК6-83Б — Д. В. Боженко
(Группа) (подпись, дата) (инициалы и фамилия)

Руководитель — Ф. А. Витюков
(подпись, дата) (инициалы и фамилия)

Москва, 2023 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

«_____» _____ 2023 г.

ЗАДАНИЕ
на выполнение научно-исследовательской работы

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

Студент группы РК6-83Б

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 виды лобби, которые используются в современных многопользовательских играх. На основе полученных знаний реализовать лобби с автоматическом подсчетом подключившихся игроков. Также реализовать admin-style + crowd-style лобби на пустом уровне.

Оформление научно-исследовательской работы:
Расчетно-пояснительная записка на 22 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):
_____
_____
_____
_____

Дата выдачи задания «10» февраля 2023 г.

Руководитель НИР _____ **Витюков Ф.А.**
(Подпись, дата) И.О. Фамилия

Студент _____ **Боженко Д. В.**
(Подпись, дата) И.О. Фамилия

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

СОДЕРЖАНИЕ

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 *MMO* (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 практической работой является изучение одной из доступных подсистем для движка Unreal Engine 4, а именно предоставляемых ей программных интерфейсов, таких как матчмейкинг, лобби, таблицы лидеров и т. п., которые широко применяются в современных многопользовательских играх. На основе полученных знаний разработать вышеперечисленные сетевые компоненты и внедрить их в шаблон многопользовательской игры.

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. ОСНОВНЫЕ ВИДЫ ЛОББИ

Лобби — это отдельная комната, куда может подключиться ограниченное количество игроков перед началом матча. Как правило, для лобби выделяется отдельный игровой уровень и отдельный класс AGameMode, так как для лобби на стороне сервера существуют свои правила, отличные от правил, определенных на игровом уровне. Есть два варианта лобби по реализации игрового процесса:

1. Пользователь имеет возможность управлять персонажем и перемещаться по карте во время ожидания остальных игроков (далее GameAndUI).
2. На экран пользователя добавляется множество виджетов с информацией о текущем уровне, пользователю доступно только управления данными виджетами (дале UIOnly).

Первый тип характерен для проектов, где пользователю необходимо время, чтобы ознакомиться с принципами взаимодействия с игровым миром в предстоящем матче на следующем уровне.

Второй тип характерен для остальных проектов, где на экран пользователя необходимо вывести больше информации о текущем состоянии уровня, например, текущее количество подключившихся игроков, имя каждого игрока и его статус готовности начать предстоящий матч.

Также существует два типа лобби по стилю управления: admin-style лобби и crowd-style лобби.

Admin-style лобби — это вид лобби, в котором игрок, находящийся на машине, на которой запущена сессия, может самостоятельно запустить матч, нажав, например, на кнопку, расположенную в его HUD. Такой вид лобби может подойти только для игры типа Listen-Server, так как в таком случае сервер, расположенный на машине одного из пользователей, имеет графическое отображение и правилами игры можно управлять с помощью графического интерфейса.

Crowd-style лобби — это вид лобби, в котором система автоматически начинает игру, как только количество подключившихся в лобби игроков достигло заданного количества и все игроки подтвердили свой статус готовности начать матч, нажав на кнопку подтверждения статуса, расположенную в HUD. Такой вид лобби может использовать в многопользовательской игре как типа Dedicated-Server, так и типа Listen-Server, так как для управления правилами игры в таком стиле не нужно никакого графического интерфейса. Большинство современных многопользовательских игр используют именно crowd-style лобби, потому что в них используются отдельные сервера.

Для проектирования шаблона многопользовательской игры важно учитывать возможность использования альтернативных решений на том или ином уровне. Для этого необходимо реализовать все вышеперечисленные виды уровня лобби.

2. ОСОБЕННОСТИ НАСТРОЙКИ ИГРОВОГО УРОВНЯ

Для того, чтобы создать лобби, для начала необходимо создать отдельный игровой уровень и провести его настройки (Рисунок 1).

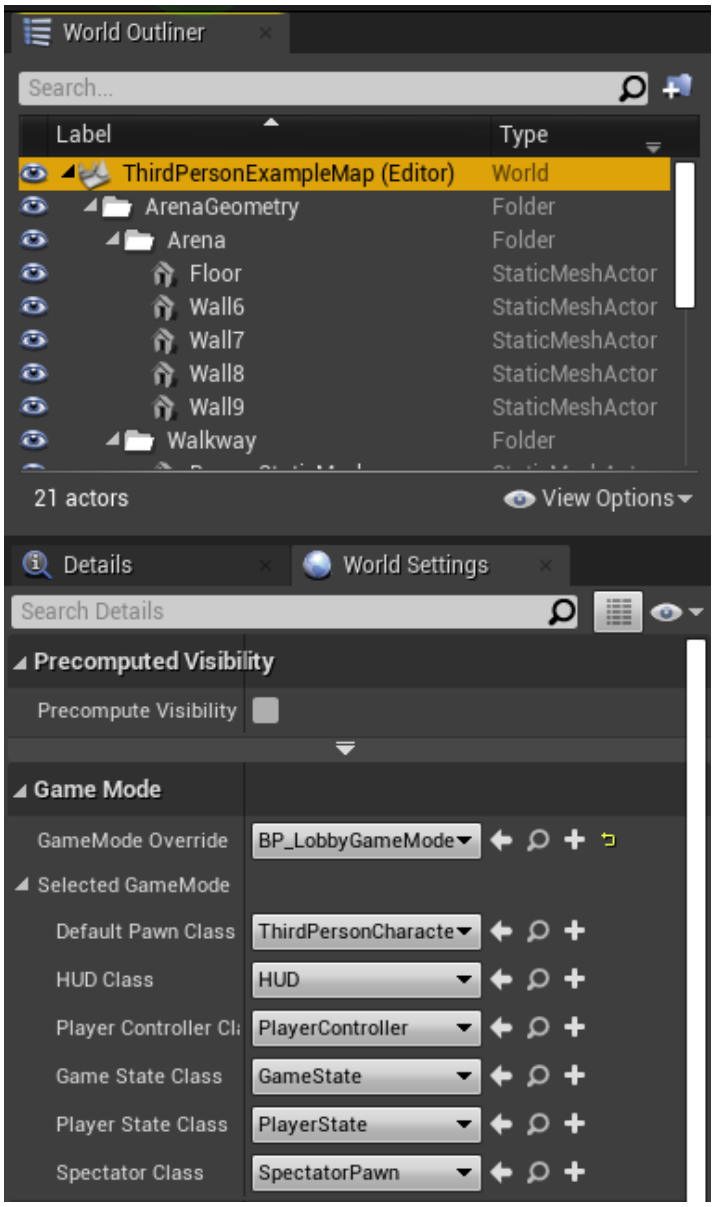

Рисунок 1 — Настройка GameAndUI лобби как отдельного игрового уровня

Отдельный игровой уровень представляет собой карту (файл с расширением. umap). Как видно на рисунке, представленном выше, карта имеет список геометрии, которая будет отображаться на сцене, и свои настройки в разделе World Outliner. Данный вариант настройки уровня характерен для GameAndUI лобби, так как на уровне присутствует геометрия и в качестве Default Pawn Class задан класс персонажа, который имеет меш для отображения

на сцене. Вариант настройки уровня с UIOnly лобби представлен ниже (Рисунок 2).

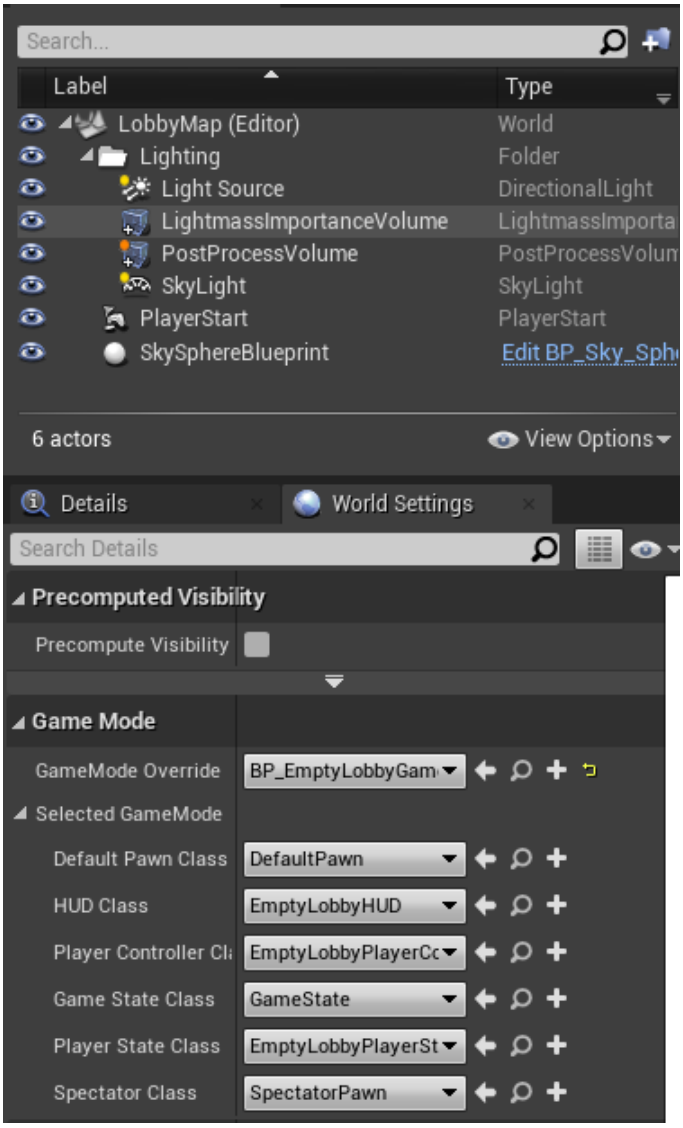

Рисунок 2 — Настройка UIOnly лобби как отдельного игрового уровня

В варианте лобби UIOnly видно, что на уровне нет геометрии и в качестве Default Pawn Class задана пешка по умолчанию. Самыми важными настройками для карты является настройка GameMode, которая определяет, какой экземпляр класса AGameMode будет задаваться для уровня по умолчанию. Также в разделе Selected GameMode присутствует перечисление классов, экземпляры которых будут созданы на уровне по умолчанию. В итоге для уровня необходимо задать класс ALobbyGameMode, производный от класса GameMode, который будет определять правила подключения игроков к лобби. Также для пункта Default

Pawn Class необходимо указать пешку, которую сервер будет создавать в игровом уровне на месте компонента NetworkPlayerStart (Рисунок 3).

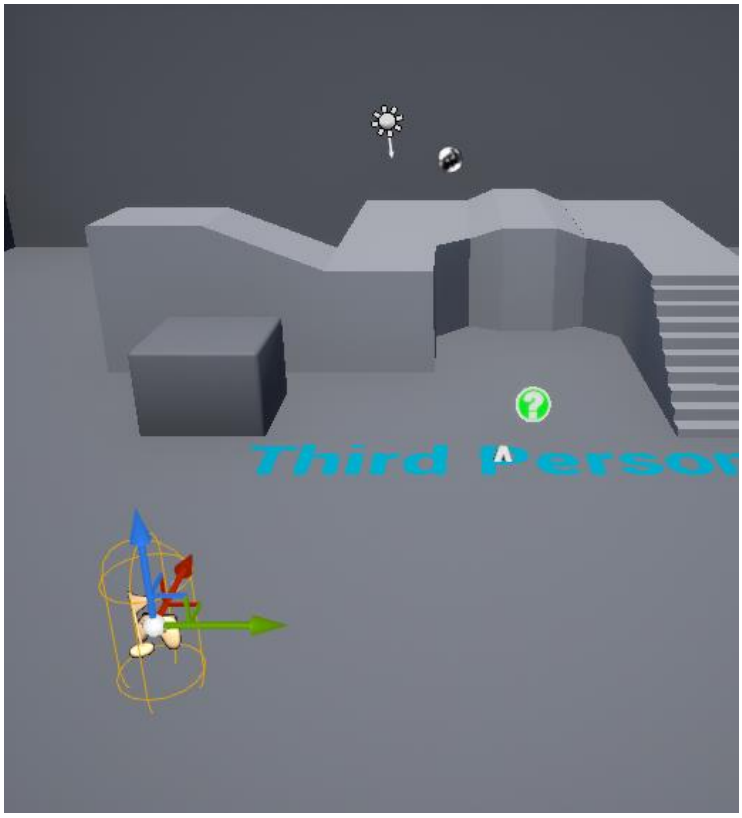

Рисунок 3 — Компонент NetworkPlayerStart на уровне лобби

Для GameAndUI лобби для управления пешкой задан режим управления FInputModeGameOnly — режим управления по умолчанию, в котором игроки могут управлять виджетами и своим персонажем. Для UIOnly лобби необходимо предоставить режим доступа управления только с помощью курсора мыши, для этого необходимо выполнять ряд действий в функции virtual APlayerController::ReceivedPlayer (Листинг 1).

```
void AEmptyLobbyPlayerController::ReceivedPlayer()
{
    Super::ReceivedPlayer();
    FInputModeUIOnly InputModeUIOnly;
    InputModeUIOnly.SetLockMouseToViewportBehavior(
        EMouseLockMode::DoNotLock);
    SetInputMode(InputModeUIOnly);
    bShowMouseCursor = true;
}
```

Задание классов по умолчанию также можно задать в конструкторе самого класса ALobbyGameMode (Листинг 2).

Листинг 2 — Задание классов по умолчанию в конструкторе класса

```
ALobbyGameMode::ALobbyGameMode()
{
    static ConstructorHelpers::FClassFinder<APawn>
    PlayerPawnBPClass(
    TEXT("/Game/ThirdPersonCPP/Blueprints/ThirdPersonCharacter"));

    if (PlayerPawnBPClass.Class != nullptr)
    {
        DefaultPawnClass = PlayerPawnBPClass.Class;
    }
}
```

3. УПРАВЛЕНИЕ ПОДКЛЮЧЕНИЯМИ ПОЛЬЗОВАТЕЛЕЙ В ЛОББИ

Управление подключениями пользователей, которые подключаются к уровню лобби, осуществляется с помощью функции virtual void AGameMode::PostLogin(APlayerController* NewPlayer). Данная функция, как видно, из ее определения, является виртуальной, и выполняется каждый раз на сервере, когда подключился новый игрок. Данная функция также принимает в качестве параметра указатель APlayerController подключившегося игрока. С помощью доступа к экземпляру класса APlayerController можно с помощью RPC типа Client вывести на экран каждого пользователя имя игрока, который только что подключился к уровню лобби (Рисунок 4).

Рисунок 4 — Отслеживание подключающихся игроков к уровню лобби

Для того, чтобы отследить общее количество подключившихся игроков, необходимо воспользоваться классом AGameState. Экземпляр данного класса доступен на сервере и содержит в себе поле PlayerArray — структура данных типа TArray, которая содержит в себе список всех игроков, подключенных к игре на момент обращения к члену класса.

Каждый раз, когда новый пользователь подключается к лобби, происходит сравнение общего количества игроков в поле PlayerArray с заранее заданным

количеством игроков. Когда количество игроков совпадает необходимо выполнять команду ServerTravel(FString("/Game/Maps/GamePlayMap?listen")), заранее задав свойство класса AGameMode bUseSeamlessTravel = true. Функция ServerTravel выполняется на сервере и выполняет перемещение сервера на другой уровень или карту. Также, важно знать, что все подключенные клиенты также последуют за сервером и они переподключатся к новому уровню. Это происходит из-за того, что сервер неявно вызывает у каждого подключенного клиента функцию ClientTravel(FString("/Game/Maps/GamePlayMap?listen")).

Общее количество игроков, которое необходимо для подключения задается с помощью настроек сессии, которые находятся в структуре FOnlineSesstionSettings в члене класса NumPublicConnections.

Свойство bUseSeamlessTravel необходимо для того, что уже ранее подключенные к серверу клиенты переподключились к новому уровне, сохраняя при этом подключение к текущему серверу. Для использования такого типа переподключения необходимо создать TransitionMap — пустой уровень, который является промежуточным между старым уровнем карты и новым игровым уровнем, к которому будут подключены все игроки, находящиеся в лобби.

4. КРУГОВАЯ ЗАДЕРЖКА (ROUND TRIP TIME)

Для того, чтобы все клиенты имели достаточное количество времени для подключения к уровню, где будет происходить игра, необходимо перед началом матча добавить промежуточное состояние матча с таймером, в котором каждый игрок имеет возможность осмотреть карту с помощью элементов управления и подождать других игроков для подключения (Рисунок 5).

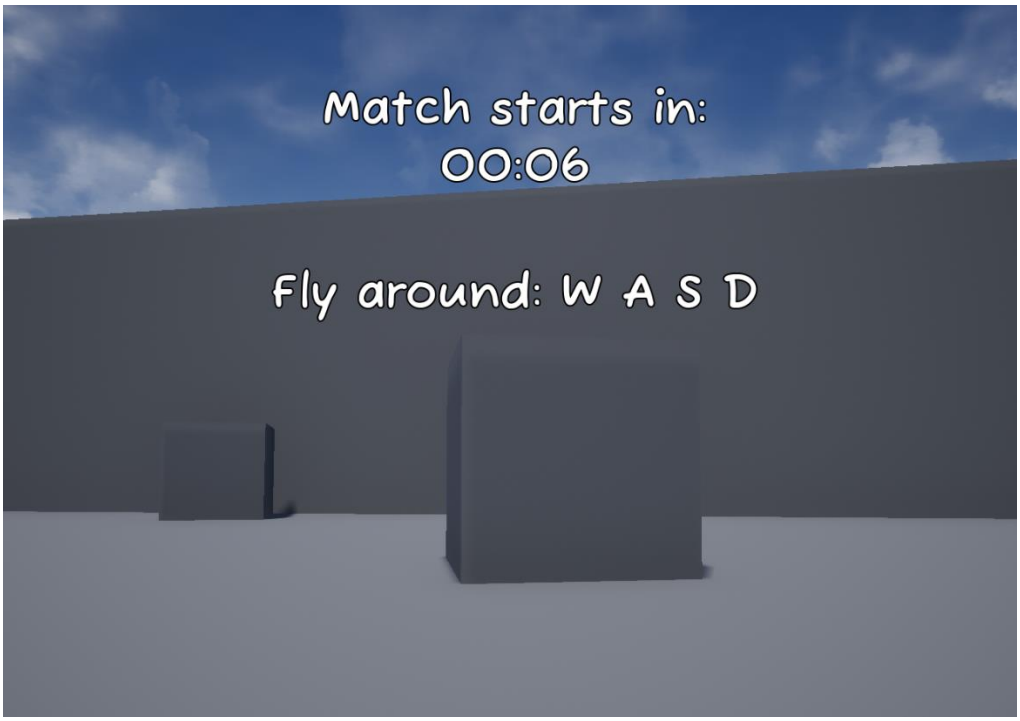

Рисунок 5 — Ожидание подключения всех игроков на игровой уровень

Как видно на рисунке, представленном выше, в HUD игрока используется таймер. Есть несколько проблем, которые могут возникнуть во время использования таймера:

1. Клиенты должны использовать время загрузки уровня, которое определяется на сервере, так как каждый клиент использует разное время загрузки игрового уровня.
2. При запросе серверного времени клиент может получить неправильное время, вызванное присутствием межсетевых задержек, возникающее при выполнении запросов от сервера к клиенту и от клиента к серверу.

Проблемы, представленные выше, описывают проблему круговых

задержек в сети, которые необходимо решить с помощью множественных вызовов RPC (Рисунок 6).

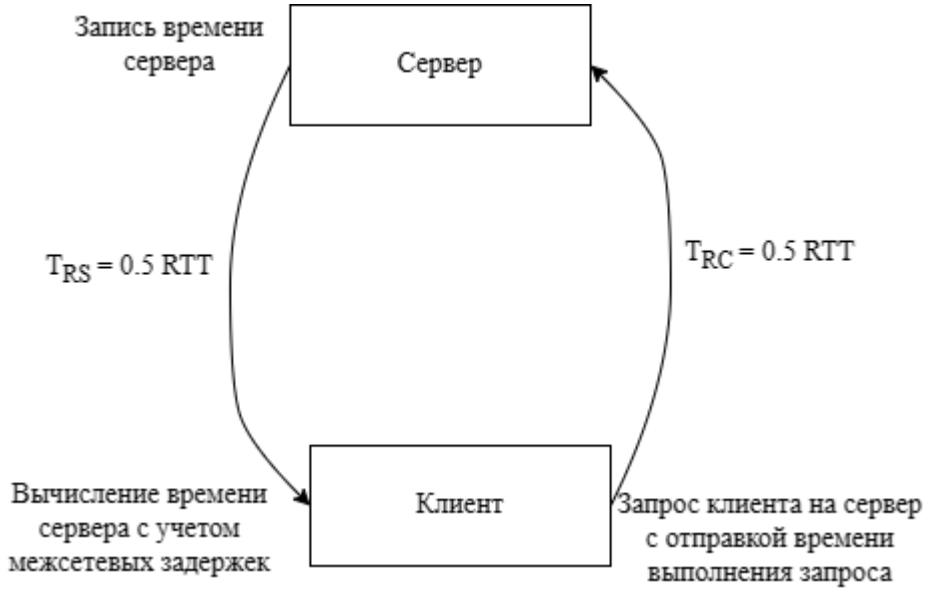

Рисунок 6 — Схема межсетевых задержек при выполнении клиент-серверных запросов

Для того, чтобы найти разницу во времени, с которой игровой уровень был загружен на сервере и на клиенте необходимо выполнить последовательный вызов RPC типа Client и Server, вызов которых представлен выше на схеме:

1. Отправить запрос типа Server RPC с клиента ServerRequestServerTime(GetWorld->GetTimeSeconds), где в качестве единственного аргумента вызова передается время, когда клиент сделал запрос.
2. Отправить запрос типа Client RPC ClientResponseServerTime (GetWorld->GetTimeSeconds, ClientRequestTime), где в качестве аргументов вызова передаются время, когда сервер получил запрос, и время, когда клиент сделал запрос.
3. Рассчитать время круговой задержки (далее RTT), сложив время, которое потребовалось серверу, чтобы получить запрос от клиента (T_{RC}), и время, которое потребовалось клиенту, чтобы получить ответ от сервера (T_{RC}).

4. Оценить реальное текущее время сервера на клиенте с учетом межсетевых задержек, прибавляя к текущему времени сервера половину RTT.
5. Вычислить на клиенте разницу во времени, с которой игровой уровень был запущен на сервере и на клиенте.

Оценив разницу времени загрузки игрового уровня на сервере и на клиенте, можно инициализировать таймер в HUD клиента. В итоге в HUD каждого клиента в промежуточное состояние матча будет отображаться таймер, который синхронизирован с серверным временем загрузки игрового уровня.

5. ADMIN-STYLE ЛОББИ

Для реализации admin-style лобби необходимо в HUD игрока, который является владельцем сессии, добавить соответствующие элементы управления в виде дополнительного графического интерфейса.

Игроку, который является владельцем сессии, добавляется в его HUD, кнопка, по нажатии которой происходит перемещение сервера на другой уровень. У других пользователей, которые являются клиентами, не должно быть соответствующего элемента управления лобби.

Чтобы однозначно определить пользователя, который является владельцем лобби, можно вызвать функцию UGamePlayStatics::GetPlayerController(this, 0) в экземпляре классе ALobbyGameMode, где 0 — это индекс элемента в массиве всех экземпляров класса APlayerController. Так как владелец лобби подключается к уровню лобби самым первым, ему всегда будет соответствовать индекс 0.

Получив доступ к экземпляру APlayerController владельца лобби, можно добавить в его HUD ранее созданный с помощью средств UMG виджет, в котором будет содержаться кнопка для начала игрового процесса и соответствующее информационное сообщение (Рисунок 7).

Рисунок 7 — Создание виджета для admin-style лобби

При перемещении сервера на конечный игровой уровень из экрана игрока необходимо удалить элемент управления матчем. В итоге в HUD пользователя, являющегося владельцем лобби, при первой загрузке соответствующего уровня появляется элемент управления лобби, с помощью которого возможно заранее, не дожидаясь заданного количества игроков, запустить матч (Рисунок 8).

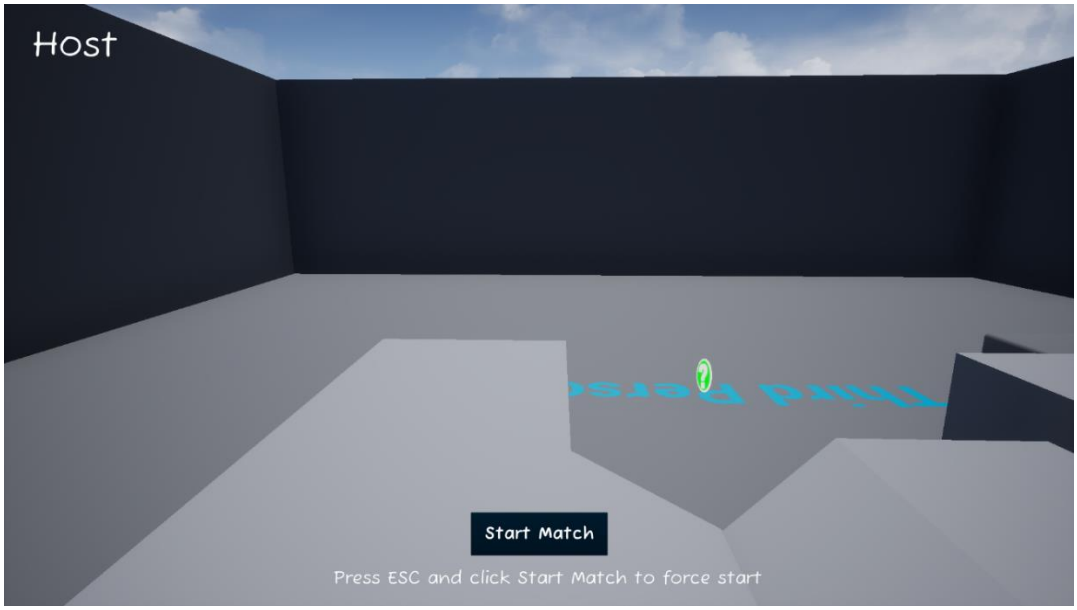

Рисунок 8 — Экран игрока, владеющего GameAndUI лобби

6. ADMIN-STYLE + CROWD-STYLE ЛОББИ

Для реализации admin-style лобби и crowd-style лобби необходимо в HUD каждого игрока добавить кнопки для изменения статуса готовности, а также необходимо добавить таблицу со всеми подключившимися игроками, где отображается имя каждого игрока и его статус готовности запустить матч.

Управление статусом готовности можно осуществить с помощью двух кнопок Ready и Cancel. Для того, чтобы пользователь случайно не смог нажать на элемент управления готовностью, необходимо также добавить обработку статуса длительного нажатия и полосу прокрутки, которая показывает, какое количество времени осталось производить нажатие (Рисунок 9).

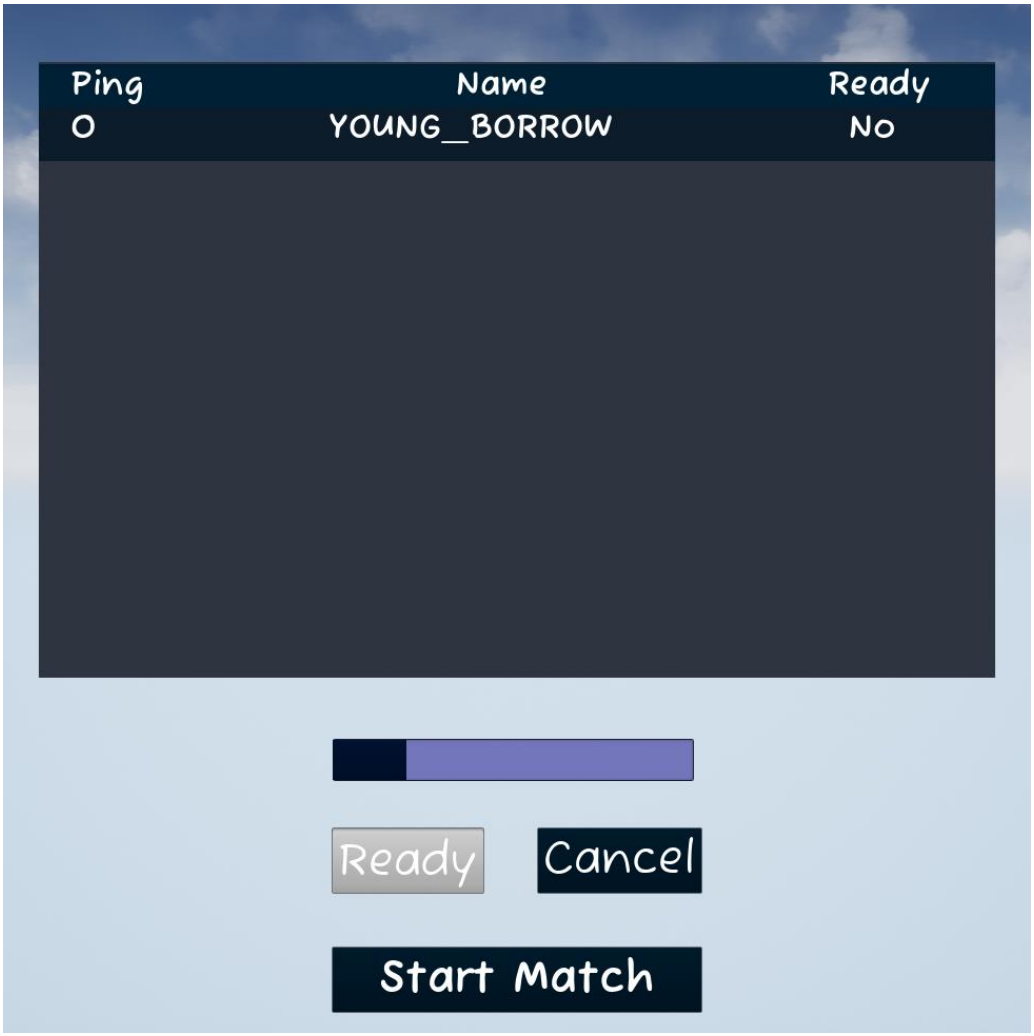

Рисунок 9 — Виджет пользователя для admin-style + crowd-style лобби

Кнопку “Start Match” также необходимо отображать только в HUD того игрока, кто является хостом сессии. Для определения сетевой роли игрока, владеющего HUD, можно использовать условие с проверкой сетевой роли (Листинг 3).

Листинг 3 — Определение сетевой роли игрока, владеющего виджетом

```
EmptyLobbyOwningController =
Cast<AEmptyLobbyPlayerController>(GetOwningPlayer());
if (EmptyLobbyOwningController &&
        EmptyLobbyOwningController->GetLocalRole() ==
        ENetRole::ROLE_AutonomousProxy &&
        EmptyLobbyOwningController->GetRemoteRole() ==
        ENetRole::ROLE_Authority)
{
    StartMatchButton->SetVisibility(ESlateVisibility::Hidden);
}
```

Чтобы обеспечить покадровое обновление полосы прокрутки, представленной выше на рисунке, необходимо расширить возможности класса, отвечающего за управление данным виджетом. Так как каждый виджет, производный от класса UUserWidget не является потомком класса AActor, в котором реализована виртуальная функция void Tick(float DeltaSeconds), необходимо выполнить множественное наследование и самостоятельно реализовать в классе виджета функцию, альтернативную функции void Tick(float) (Листинг 4).

Листинг 4 — Расширение функционала класса виджета, производного от UObject

```
class LAB4_API UStatusControll : public UUserWidget, public
FTickableGameObject
{
    protected:
    virtual bool Initialize() override;
    virtual void Tick(float DeltaTime) override;
    virtual bool IsTickable() const override;
};
void UStatusControll::Tick(float DeltaTime)
{
    if (bIsProgressbarVisible)
    {
        SetPorgressbarPercantage();
    }
}
bool UStatusControll::IsTickable() const
{
    return true;
}
TStatId UStatusControll::GetStatId() const
{
    return TStatId();
}
```

После добавления и реализации функций, представленных выше, в теле функции void UStatusControll::Tick(float DeltaTime) необходимо выполнять код, который должен повторяться каждый тик.

ЗАКЛЮЧЕНИЕ

Лобби является важным промежуточным уровнем, который требует гибкой настройки для правильной работы. Так как существует несколько видов лобби, то шаблон многопользовательской игры должен предоставлять возможность управления лобби несколькими способами.

Для ознакомления пользователя с игровым процессом на следующем уровне шаблон многопользовательской игры должен предоставлять вид лобби, в котором игрок может управлять своим персонажем и взаимодействовать с элементами карты, дожидаясь других игроков.

В случае, если пользователю важно получать много информации о текущем игровом уровне (количество игроков, состояние готовности каждого игрока, количество времени до старта игрового процесса) и нет необходимости с ознакомлением предстоящего игрового процесса, шаблон многопользовательской игры должен предоставлять возможность использования пустого уровня с большом количеством информативных интерактивных виджетов.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Traveling in Multiplayer. Unreal Engine Documentation [Электронный ресурс] // Режим доступа: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/Travelling/ (дата обращения 20.04.23).
2. Расширяем возможности UObject в Unreal Engine 4. Хабр [Электронный ресурс] // Режим доступа: https://habr.com/ru/companies/pixonic/articles/475622/ (дата обращения 24.04.23).
3. RPCs. Unreal Engine Documentation [Электронный ресурс] // Режим доступа: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/Actors/RPCs/ (дата обращения: 25.04.23).
4. Replication. Unreal Engine Community Wiki [Электронный ресурс] // Режим доступа: https://www.unrealcommunity.wiki/replication-vyrv8r37 (дата обращения: 26.04.23)