Петрозаводский государственный университет Институт математики и информационных технологий Кафедра прикладной математики и кибернетики

09.03.02 – Информационные системы и технологии

Отчет о практике по научно-исследовательской работе

Изучение инструмента Netcode for GameObjects для создания многопользовательских игр на Unity

Выполнил:

студент 2 курса группы 22205

И. О. Левицкий

*подпись*

Руководитель:

Р. В. Сошкин, к. т. н., старший преподаватель

Итоговая оценка

*подпись*

*оценка*

Петрозаводск — 2022

Содержание

[1 Введение 3](#_Toc135205914)

[1.1 Список используемых терминов 3](#_Toc135205915)

[1.2 Введение 3](#_Toc135205916)

[2 Обзор Netcode для создания многопользовательских игр на Unity 5](#_Toc135205917)

[2.1 Unity и его возможности 5](#_Toc135205918)

[2.2 Поддержка мультиплеера в Unity и выбор Netcode для создания игры 5](#_Toc135205919)

[2.3 Основные возможности Netcode 6](#_Toc135205920)

[3 Разработка многопользовательской игры на Unity с использованием Netcode 9](#_Toc135205921)

[3.1 Настройка сетевой части игры 9](#_Toc135205922)

[3.2 Создание персонажей и управление ими 15](#_Toc135205923)

[3.3 Создания игрового объекта, использующего передачу сообщений между клиентами 19](#_Toc135205924)

[3.4 Результат 22](#_Toc135205925)

[4 Заключение 24](#_Toc135205926)

[5 Список использованных источников 25](#_Toc135205927)

# Введение

Цель практики: изучить инструмент Netcode for GameObjects

Задачи практики:

1. Получить опыт работы с Unity
2. Исследовать возможности Netcode for GameObjects
3. Получить опыт разработки многопользовательской игры

## Список используемых терминов

**Игровой движок** — это программное обеспечение, которое облегчает разработку видеоигр. Он предоставляет набор инструментов и библиотек для создания игрового мира, включая графику, звук, физику, искусственный интеллект и другие функциональные возможности. Игровой движок упрощает процесс разработки, позволяя разработчикам сконцентрироваться на создании игрового контента, а не на написании базовых компонентов игры с нуля.

**Мультиплеер** (англ. multiplayer — «множество игроков») — режим компьютерной игры, во время которого играет больше одного человека**.**

**Netcode, NGO** – Netcode for GameObjects

**Ассет** (англ. Asset — «объект») — это любой объект, который может быть использован в игре, включая модели персонажей, текстуры, звуки, скрипты, анимации и другие материалы.

**Префаб** (Prefab) — это предопределенный шаблон игрового объекта, который может быть многократно использован в игре. Он представляет собой готовую конфигурацию объекта, включая его компоненты, свойства, относительное положение в пространстве и другие атрибуты.

## Введение

В наше время многопользовательские игры пользуются большой популярностью среди игроков и разработчиков. Они позволяют людям со всего мира играть вместе, общаться и создавать новые знакомства. Разработка многопользовательской игры является сложным и трудоемким процессом, требующим знания многих технологий и инструментов. В рамках этой работы планируется изучать инструмент для создания многопользовательских игр на Unity, одной из самых популярных игровых платформ на сегодняшний день.

Целью данной курсовой работы является изучение инструмента для создания многопользовательских игр на Unity и разработка простейшей многопользовательской игры с использованием выбранного инструмента. Для достижения данной цели в работе будут рассмотрены основные инструменты для создания многопользовательских игр на Unity, проведен обзор возможностей выбранного инструмента, описан процесс разработки многопользовательской игры на Unity с использованием выбранного инструмента и проведено тестирование игры.

# Обзор Netcode для создания многопользовательских игр на Unity

## Unity и его возможности

Unity — это мощный и гибкий игровой движок, который широко используется для создания игр различных жанров и платформ. Unity поддерживает создание игр для ПК, мобильных устройств, игровых консолей и виртуальной реальности.

Unity имеет встроенные инструменты для создания анимаций, света, физики и звука, что упрощает процесс создания игр и позволяет разработчикам быстро прототипировать свои идеи.

Кроме того, Unity имеет огромное сообщество разработчиков и множество ресурсов, таких как форумы и видеоуроки, которые помогут быстро начать работу с движком и решить возникающие проблемы. Unity также имеет свой магазин ассетов (Asset Store), где можно найти различные ресурсы, такие как готовые модели, текстуры, звуки, скрипты и другие материалы, которые можно использовать для создания игр.

Структура Unity завязана на работе с игровыми объектами (Game Objects), которые играют ключевую роль в игровой среде. Каждый игровой объект может иметь набор компонентов - основных элементов, которые определяют взаимодействия игрового объекта с окружающим миром. Некоторые компоненты встроены в Unity изначально, такие как Transform, Rigidbody и Collider, которые определяют положение, движение и физическое поведение объектов в игре. Однако, разработчики игры могут написать собственные компоненты, чтобы расширить возможности игры и определить свое уникальное поведение объектов.

## Поддержка мультиплеера в Unity и выбор Netcode для создания игры

При установке Unity, инструменты для разработки многопользовательских игр не являются частью стандартного набора функций. Однако, Unity имеет встроенный менеджер пакетов, который позволяет легко найти и установить необходимые инструменты. На сегодняшний день, существует несколько библиотек для создания многопользовательских игр на Unity, наиболее популярными из них являются:

* Netcode for GameObjects
* Photon
* Mirror

Из перечисленных вариантов я выбрал Netcode, так как это инструмент, который разрабатывается и поддерживается самим Unity. Он обладает многими полезными функциями, которые позволяют легко создавать многопользовательские игры.

## Основные возможности Netcode

Netcode for GameObjects (Netcode) — это высокоуровневая сетевая библиотека, созданная для Unity, которая позволяет абстрагировать логику сетевого взаимодействия. Она позволяет отправлять игровые объекты и данные игрового мира через сетевую сессию для одновременной игры многих игроков. С помощью Netcode можно сосредоточиться на создании игры, а не на работе с низкоуровневыми протоколами и сетевыми фреймворками.

Компоненты, предоставляемые Netcode:

* NetworkManager — это обязательный компонент Netcode, который содержит все настройки, связанные с сетевым взаимодействием нашего проекта.
* NetworkTransform – это компонент, который автоматически синхронизирует параметры компонента Transform на всех клиентах в игре. Этот компонент позволяет выбрать синхронизацию позиции, поворота и размера игрового объекта по всем нужным осям, что делает его достаточно гибким. Кроме того, компонент NetworkTransform поддерживает интерполяцию, что позволяет сгладить рывки при передвижении объекта и улучшить качество игрового процесса для игроков.
* NetworkAnimator – обеспечивает синхронизацию анимации во время сетевой игры. Состояния анимации синхронизируются с игроками, которые присоединяются к уже существующей сетевой игре, а также с любыми клиентами, которые уже были подключены до изменения состояния анимации.

**Система сообщений в Netcode**

В Netcode используются два варианта для синхронизации состояния игры или событий:

* RPC (Remote Procedure Call)
* NetworkVariables

RPC может рассматриваться как способ отправить оповещение о каком-либо событии от сервера клиенту или наоборот.

Есть два типа RPC методов:

* ServerRpc
* ClientRpc

ServerRpc – это метод, вызываемый клиентом, который будет получен и исполнен сервером.

ClientRpc – это метод, вызываемый сервером, который будет получен и исполнен на одном или более клиентах.

Стоит заметить, что клиенты, которые подключатся после вызова RPC методов, не получат их, из-за чего состояние игры на разных клиентах может отличаться.

NetworkVariable чаще всего используется для синхронизации состояний между уже подключенными клиентами и теми клиентами, которые подключились позже.

Какую систему сообщений применять зависит от архитектуры проекта. Если игра будет начинаться только после подключения всех игроков, и подключиться после начала игры нельзя, то можно обойтись только использованием RPC методов.

Очень важной частью Netcode является сериализация данных. RPC методы принимают только те аргументы, которые могут быть сериализованы. Netcode способен автоматически сериализовать:

* примитивные типы из C# (bool, char, sbyte, byte, short, ushort, int, uint, long, ulong, float, double, string)
* Примитивные типы Unity (Color, Color32, Vector2, Vector3, Vector4, Quaternion, Ray, Ray2D)
* Массивы из перечисленных выше типов данных
* Собственные структуры, в которых будет реализован интерфейс INetworkSerializable

# Разработка многопользовательской игры на Unity с использованием Netcode

## Настройка сетевой части игры

При выполнении исследования необходимо создать игру, в которой один из клиентов будет хостом, то есть будет также выполнять роль сервера. Благодаря этому в проекте не придется использовать выделенный сервер и следить за его работоспособностью.

Однако если клиент и хост не находятся в одной локальной сети, то для подключения придется выбрать один из следующих вариантов:

* Ручная настройка перенаправления портов у хоста
* Использование Relay

Первый вариант очень неудобен, так как требует некоторых технических знаний от пользователей игры, и маловероятно, что игрокам понравится заниматься настройкой своего маршрутизатора.

Второй вариант предусматривает использование Relay сервера. Это сервер в интернете с публичным IP адресом, к которому могут подключится хост и клиенты без дополнительных условий. После подключения все данные будут проходить через этот сервер.

Unity предоставляет бесплатное использования такого сервера, с ограничением в 50 одновременных подключений. Именно этот вариант будет использован при выполнении работы.

**Создание игровых объектов и компонентов, отвечающих за сетевую настройку игры**

**Network Manager**

Для создания объекта Network Manager нужно завести пустой игровой объект на сцене Unity. Назовем его NetworkManager. После этого требуется прикрепить к нему компонент NetworkManager, который имеется в библиотеке Netcode. Unity автоматически добавит компонент Unity Transport, в котором нужно выбрать тип используемого протокола. Для данной работы было выбрано использование Relay сервера, поэтому выбирается соответствующий вариант. Остальные параметры выбраны оптимально и не требуют изменения.

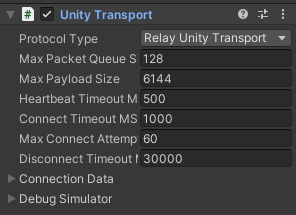


Рисунок 1 - компонент Unity Transport

**Игровой объект Relay и его компонент**

Для обеспечения функционирования Relay сервера в проекте будет создан пустой игровой объект с названием "Relay". К этому объекту будет прикреплен компонент Relay, написанный вручную для реализации необходимой функциональности.

Особенностью объекта Relay является его единственный экземпляр в проекте. Для достижения этой цели будет применен паттерн проектирования Singleton. Этот паттерн позволяет гарантировать, что объект Relay будет создан только один раз, и будет доступен для обращения из любой части кода в проекте.

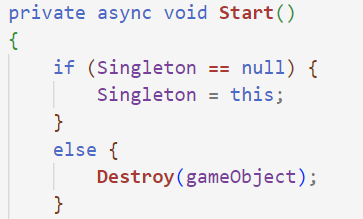


Рисунок 2 - реализация паттерна Singleton

Функция Start играет важную роль в жизненном цикле игрового объекта в Unity, так как она автоматически вызывается при инициализации объекта на сцене. Созданная реализация функции Start имеет особое значение, поскольку гарантирует наличие только одного игрового объекта с компонентом Relay на сцене в любое время.

Для обеспечения выделения и настройки сервера Relay будет разработана асинхронная функция под названием CreateRelay. Данная функция не будет принимать аргументы и будет открытой для доступа из других компонентов, чтобы обеспечить возможность ее вызова из различных частей игры.

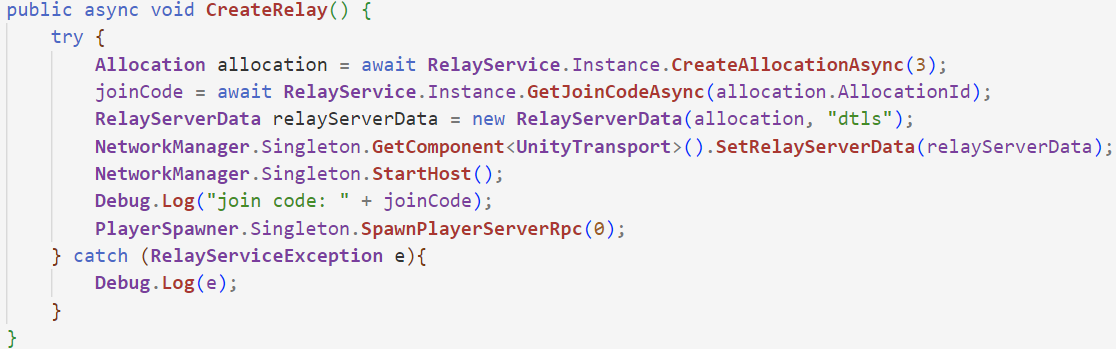


Рисунок 3 - реализация функции CreateRelay

Эта функция предназначена для запроса выделения сервера и выполняет следующие шаги для обеспечения правильной работы сетевого взаимодействия в игре.

Вначале функция запрашивает выделение сервера, используя соответствующие сервисы и инструменты, предоставляемые Unity. Затем она сохраняет полученный код, который будет использоваться клиентами для подключения к серверу. Кроме того, функция сохраняет необходимые данные о самом сервере, которые понадобятся компоненту NetworkManager для правильной инициализации и установки сетевых параметров.

Однако, в процессе работы с сервисами Unity могут возникать ошибки, которые не зависят от игрового клиента. Поэтому вся функция окутана конструкцией для обработки исключений, чтобы перехватывать и обрабатывать такие ошибки. Это позволяет более гибко управлять и обрабатывать исключительные ситуации, которые могут возникнуть в процессе выполнения этой функции.

Для подключения к уже созданному серверу будет реализована функция JoinRelay, которая будет принимать код для подключения в виде строки

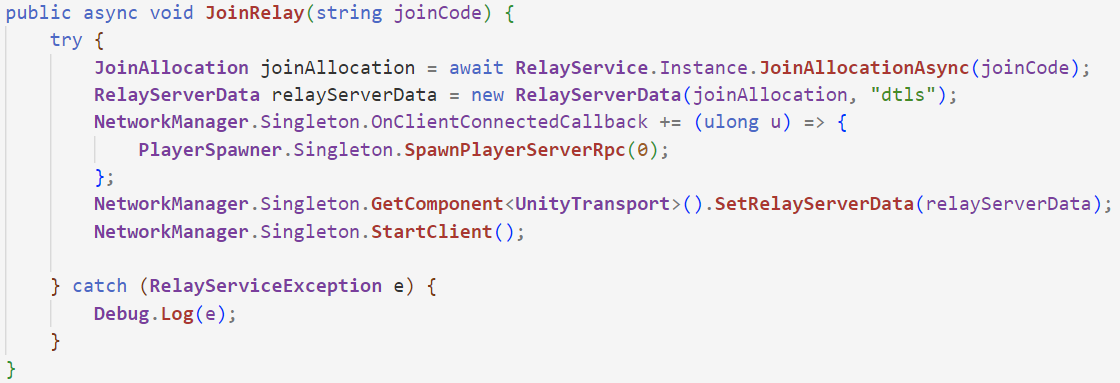


Рисунок 4 - реализация функции JoinRelay

Функция JoinRelay выполняет следующие задачи:

* Принимает код для подключения в виде строки, который предоставляется клиентом. Этот код представляет собой уникальный идентификатор сервера Relay, который позволяет клиенту успешно установить соединение с сервером.
* Использует полученный код для подключения к серверу Relay. Для этого функция будет использовать соответствующие сетевые протоколы и механизмы, предоставляемые Unity, для установления соединения с сервером.
* Обрабатывает возможные ошибки, которые могут возникнуть при подключении к серверу Relay. Для этого функция будет использовать конструкцию try-catch, чтобы перехватывать и обрабатывать исключения, связанные с сетевыми ошибками или неправильным кодом подключения.
* После успешного подключения к серверу Relay функция JoinRelay выполняет необходимые действия для продолжения сетевого взаимодействия с сервером. На текущий момент это включает в себя запуск клиента и инициализацию персонажа игрока, реализация которой будет написана позже.

Для создания и инициализации игрового объекта персонажа будет реализован компонент PlayerSpawner, который будет прикреплен к одноименному пустому объекту на сцене. В этом компоненте также будет реализован паттерн Singleton.

Данный компонент будет иметь одну функцию, реализующую создание персонажа и привязку его к игроку.

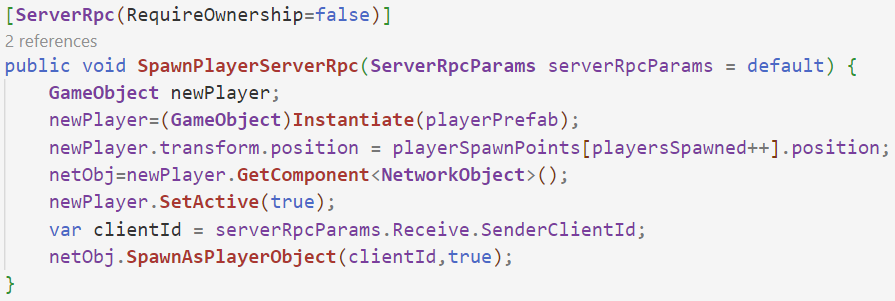


Рисунок 5 - реализация функции SpawnPlayerServerRpc

Данная функция выполняет следующие действия:

Создание игрового объекта "newPlayer" путем создания экземпляра префаба игрока (playerPrefab).

Установка позиции игрового объекта "newPlayer" на одну из точек появления игрока (playerSpawnPoints) в порядке их использования. Каждый раз, когда игрок создается, значение переменной "playersSpawned" увеличивается для выбора следующей точки появления.

Получение компонента NetworkObject (netObj) из созданного игрового объекта "newPlayer". NetworkObject отвечает за сетевую функциональность и синхронизацию объекта между клиентами и сервером.

Активация игрового объекта "newPlayer".

Получение идентификатора клиента (clientId) из параметров серверного RPC (serverRpcParams). Этот идентификатор указывает, какому клиенту должен быть назначен созданный игровой объект.

Вызов метода SpawnAsPlayerObject() на компоненте NetworkObject (netObj) для созданного игрового объекта "newPlayer". Этот метод сообщает сетевой системе, что объект должен быть инициализирован как объект игрока для указанного клиента (clientId) и передает информацию о том, что объект является объектом игрока.

На данном этапе разработки проекта, этих игровых объектов и компонентов будет достаточно для дальнейшей работы.

## Создание персонажей и управление ими

Для реализации персонажа игрока планируется создать специальный префаб, который будет применяться в функции SpawnPlayerServerRpc. В рамках начальной версии проекта требуется разработать игровой объект, представляющий собой двумерную капсулу и выполняющий функции игрового персонажа. В качестве необходимых компонентов следует добавить Collider, Rigidbody и NetworkTransform отвечающие за физическое моделирование данного игрового объекта и синхронизацию компонента Transform среди всех клиентов. Дополнительно будет создан пользовательский компонент PlayerController, который обеспечит функциональность управления персонажем.

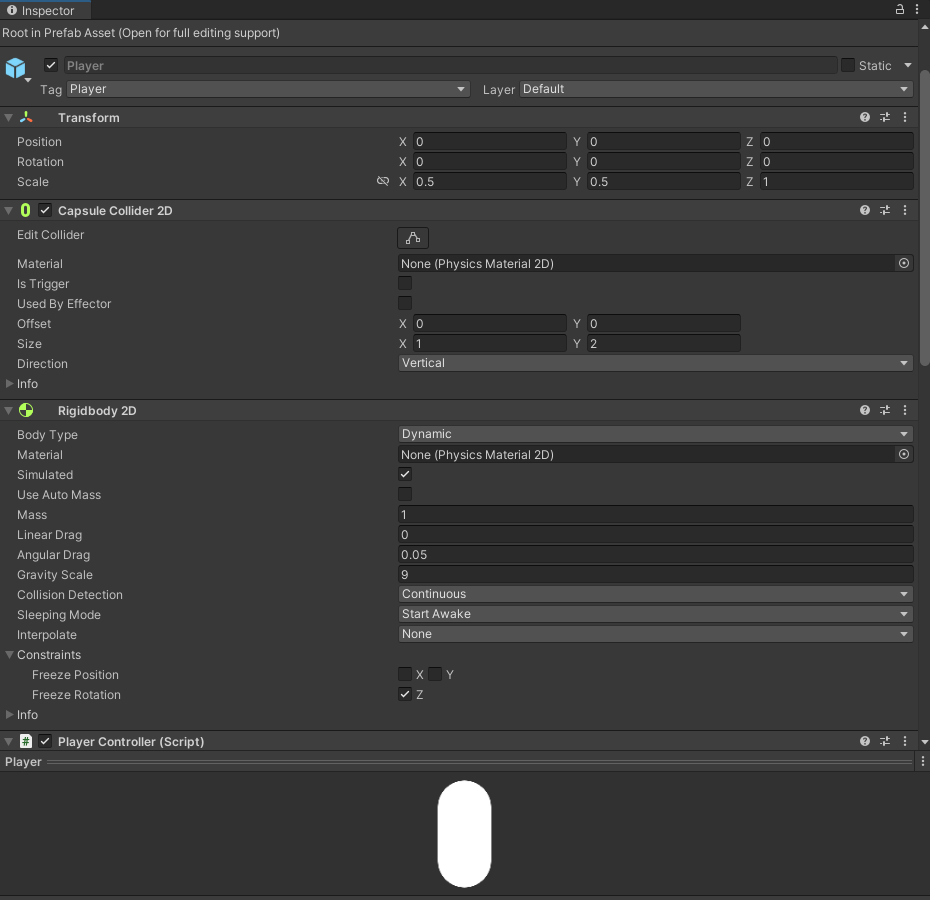


Рисунок 6 – список основных компонентов персонажа

**Реализация компонента PlayerController**

В функции Start требуется сохранить ссылки на компоненты Rigidbody и Collider, чтобы не вызывать функцию GetComponent при каждом обращении к этим компонентам. Это оптимизирует производительность проекта и уменьшит количество написанного кода.



Рисунок 7 - реализация функции Start в PlayerController

В контексте текущей реализации игры, на данном этапе, достаточно предусмотреть движение игрового персонажа по горизонтали и его способность к прыжку. Функция HandlePlayerMovement будет использована для обработки проверки нажатия клавиш и реализации функциональности прыжка.

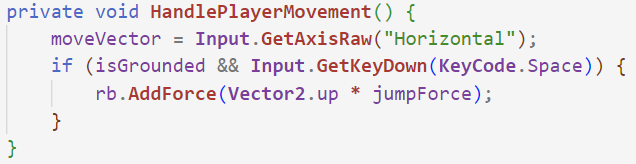


Рисунок 8 - реализация функции HandlePlayerMovement

Для определения направления движения персонажа в этом компоненте использовалась функция GetAxisRaw, доступная в классе Input, реализованном в среде разработки Unity. В данном контексте функция возвращает значение 1 при нажатии клавиши "D" или стрелки вправо, а также значение -1 при нажатии клавиши "A" или стрелки влево. В случае, когда ни одна из указанных клавиш не нажата, функция возвращает значение 0.

В дальнейшем производится проверка состояния, при котором игровой персонаж находится на земле, и клавиша пробел была нажата. В случае выполнения данных условий, будет применена функция AddForce компонента Rigidbody. Данная функция применяет к игровому объекту силу, определенную вектором, переданным в качестве аргумента функции.

Далее будут представлены реализации функций Update и FixedUpdate, которые автоматически вызываются в среде разработки Unity. Функция Update вызывается на каждом кадре отображения, в то время как функция FixedUpdate вызывается с равными, предварительно заданными интервалами времени, определенными в настройках проекта.

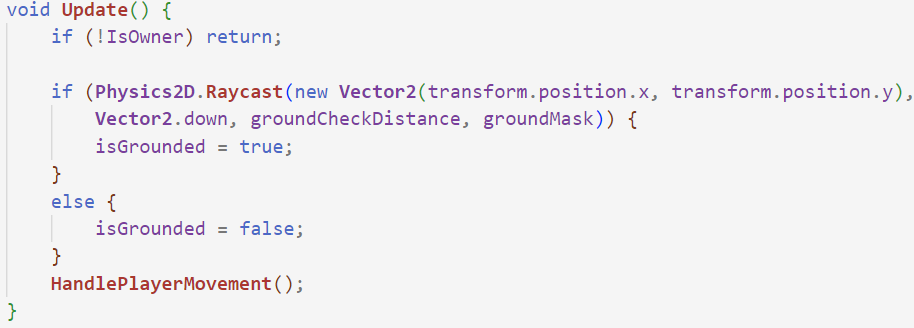


Рисунок 9 - реализация функции Update в PlayerController

В начале данной функции осуществляется проверка на принадлежность персонажа к клиенту, чтобы предотвратить ситуацию, в которой один игрок контролирует всеми персонажами на сцене. В случае, если клиент не является владельцем, функция завершает свое выполнение.

Затем, с использованием встроенной функции Raycast в Unity, выпускается луч. При вызове функции задаются параметры луча, такие как его исходная точка, направление, длина и слои, с которыми необходимо взаимодействовать при столкновении.

В данном случае производится проверка находится ли персонаж на земле.

В конечной части функции Update выполняется обработка нажатия клавиш.

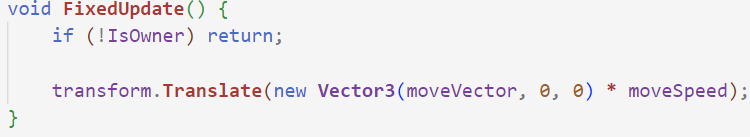


Рисунок 10 - реализация функции FixedUpdate в PlayerController

В функции FixedUpdate также присутствует проверка владения персонажем. При успешном выполнении данной проверки происходит перемещение персонажа по горизонтали с использованием функции Translate компонента Transform. Данная функция принимает вектор, который определяет смещение игрового объекта относительно его текущего положения.

Реализация перемещения осуществляется в функции FixedUpdate с целью обеспечения постоянной скорости перемещения персонажа независимо от частоты кадров. Если бы этот код был размещен в функции Update, то функция перемещения вызывалась бы чаще на компьютерах с более высокой частотой кадров, что привело бы к увеличению скорости перемещения в игре.

## Создания игрового объекта, использующего передачу сообщений между клиентами

Для примера создания игрового объекта, который будет синхронизирован на всех клиентах, был выбран лифт, который активируется нажатием персонажа кнопку.

Он будет реализован в виде синего и красного прямоугольника, где синий прямоугольник – это сам подъемник, а красный – кнопка, активирующая его.



Рисунок 11 - демонстрация лифта на игровой сцене

**Реализация компонента кнопки для лифта**

Для синхронизации состояния лифта понадобится знать только одно значение – нажата ли кнопка в данный момент. Для этого будут использованы ServerRpc и ClientRpc методы.



Рисунок 12 - реализация RPC методов для кнопки лифта

В данном компоненте реализована переменная класса логического типа, отвечающая за то, нажата ли кнопка. RPC методы просто реализуют изменение значения этой переменной на всех клиентах игры.

Далее требуется написать методы, в которых будут вызываться данные RPC.



Рисунок 13 - методы, в которых вызываются RPC

Данные методы реализованы в классе Collider, который является компонентом лифта и кнопки лифта, с целью проверки пересечения с другими игровыми объектами на сцене. Эти два метода вызываются в случае, когда другой игровой объект входит в коллайдер данного объекта или покидает его.

Внутри этих методов осуществляется проверка, является ли игровой объект, взаимодействующий с кнопкой, игроком. В противном случае функция завершает свое выполнение. Затем вызывается ServerRpc, который изменяет состояние кнопки на всех клиентах.

**Реализация компонента лифта**

Для реализации компонента лифта потребуется только изменение положения лифта в зависимости от текущего состояния кнопки.

Это будет реализовано сразу в функции FixedUpdate



Рисунок 14 - реализация компонента лифта

В данном компоненте будут настраиваемы несколько параметров:

* Игровой объект кнопки, для гибкого изменения того, какая кнопка отвечает за определенный лифт
* Максимальная высота подъёма лифта
* Скорость лифта

В приведённой реализации компонента используется атрибут SerializeField. Он нужен для отображения переменной в инспекторе, при этом оставляя её приватной.

В функции Start идёт получение ссылки на компонент, реализованный ранее.

Функция FixedUpdate реализует перемещение лифта с помощью функции Translate, зависящее от текущего состояния кнопки, привязанной к лифту.

## Результат

В результате разработки тестовой многопользовательской игры в рамках данной исследовательской работы получилось создать начальную версию, в которой реализовано:

* Подключение игроков друг к другу
* Персонажи, которыми игроки могут управлять
* Синхронизация перемещения персонажей
* Лифт, синхронизированный на всех клиентах





Рисунок 15 - демонстрация перемещения лифта

# Заключение

В ходе выполнения данной курсовой работы были сформулированы цели и задачи работы, рассмотрены основные аспекты создания многопользовательской игры на платформе Unity с использованием Netcode.

Краткий обзор на инструмент Netcode позволил понять возможности и преимущества данной библиотеки. Было отмечено, что Unity является мощным инструментом для создания игр и предоставляет широкий набор возможностей. Было рассмотрено, как Netcode интегрируется в Unity и было принято решение использовать Netcode для разработки многопользовательской игры.

В разделе "Разработка многопользовательской игры на Unity с использованием Netcode" были представлены основные шаги и этапы разработки игры. Описана настройка сетевой части игры, создание персонажей и управление ими, а также создание игрового объекта, использующего передачу сообщений между клиентами.

В целом, выполнение данной курсовой работы позволило получить практические навыки разработки многопользовательских игр на Unity с использованием Netcode, а также понимание основных принципов и возможностей данного инструмента. Полученные знания и опыт могут быть полезными для дальнейшей работы в области разработки игр и создания многопользовательских игровых проектов.

# Список использованных источников

1. Документация Netcode for GameObjects. Электронный ресурс URL: <https://docs-multiplayer.unity3d.com/netcode/current/about/> (Дата обращения 15.04.2023)
2. Документация Unity. Электронный ресурс URL: <https://docs.unity3d.com/Manual/index.html> (Дата обращения 15.04.2023)
3. <https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0>
4. Видеоуроки по Unity. Электронный ресурс URL: <https://www.youtube.com/@CodeMonkeyUnity> (Дата обращения 15.04.2023)