МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ  
«ПОЛЕССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИСТЕТ»

|  |  |
| --- | --- |
| Факультет банковского дела | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (отметка о допуске курсовой работы  (проекта) к защите) |
|  | \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись научного руководителя) (Ф.И.О.)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ г. |

Дата регистрации  
работы на кафедре \_\_\_\_\_\_\_\_\_\_\_\_

Оценка  
курсовой работы (проекта) \_\_\_\_\_\_\_\_\_\_

**КУРСОВАЯ РАБОТА (ПРОЕКТ)**

**ПО ДИСЦИПЛИНЕ:** Программирование сетевых приложений

На тему: Разработка сетевого приложения «Многопользовательский FPS» с использованием протокола TCP

Исполнитель:  
студент группы 14ИТ-1  
Домась А.Н.

Научный руководитель:  
Пигаль П.Б, старший преподаватель

Пинск, 2016

**РЕФЕРАТ**

Курсовой проект: 30с, 3 рисунка, 4 таблицы, 21 источник, 3 приложения.

ПРОТОКОЛ, ОТПРАВКА, FPS, ИГРА, СЕРВЕР

Объектом исследования являются многопользовательские игры с управлением от первого лица.

Предметом исследования является система соединения игроков в многопользовательской игре с помощью сервера.

Целью проекта обеспечение возможности нескольким игрокам установить связь во время сеанса игры.

При выполнении проекта были использованы методы: среда для разработки игры “Unity 5.5”, паттерн одиночка(Singleton), консольное приложения для обмена данными.

В процессе работы проведены следующие исследования и разработки: спроектирована схема отправки и приема сообщений между игроками и создана возможность обмена данными между игроками посредством сервера.

Результатами внедрения явилась возможность обмениваться данными между игроками во время игры.

Студент подтверждает, что приведенный в дипломном проекте расчетно-аналитический материал объективно отражает состояние разрабатываемого объекта, все заимствованные из литературных и других источников теоретические и методологические положения и концепции сопровождаются ссылками на их авторов.

\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись студента)*

**Содержание**

[Введение 4](#_Toc469286330)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc469286331)

[1.1 Общие положения 6](#_Toc469286332)

[1.2 Сведения из теории 6](#_Toc469286333)

[1.3 Постановка задачи 7](#_Toc469286334)

[2 ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ 8](#_Toc469286335)

[2.1 Алгоритм решения 8](#_Toc469286336)

[2.2 Макет приложения 8](#_Toc469286337)

[2.3 Описание программы 9](#_Toc469286338)

[3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 11](#_Toc469286339)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 12](#_Toc469286340)

[ПРИЛОЖЕНИЕ А 13](#_Toc469286341)

# **Введение**

В основе сетевой поддержки языков программирования лежит концепция сокета (socket). Сокет идентифицирует конечную точку сети. Сокеты – основа современных сетей, поскольку сокет позволяет отдельному компьютеру обслуживать одновременно множество разных клиентов и множество различных типов информации. Это достигается за счет использования порта (port) – нумерованного сокета на определенной машине. Считают, что серверный процесс «слушает» порт до тех пор, пока клиент не соединится с ним. Сервер в состоянии принять множество клиентов, подключенных к одному и тому же номеру порта, хотя каждый сеанс является уникальным. Чтобы обработать множество клиентских соединений, серверный процесс должен быть многопоточным либо обладать какими-то другими средствами обработки одновременного ввода-вывода. Сетевые коммуникации происходят по протоколу IP (Internet Protocol – IP) – это низкоуровневый маршрутизирующий протокол, который разбивает данные на небольшие пакеты и посылает их через сеть по определенному адресу, что не гарантирует доставки всех этих пакетов по этому адресу. Протокол управления передачей (Transmission Control Protocol – TCP) является протоколом более высокого уровня, обеспечивающий надежную сборку этих пакетов, сортировку и повторную передачу, необходимую для надежной доставки данных. Третий протокол — протокол пользовательских дейтаграмм (User Datagram Protocol – UDP), стоящий непосредственно за TCP, может быть использован непосредственно для поддержки быстрой, не требующей постоянного соединения и ненадежной транспортировки пакетов. Как только соединение установлено, применяется высокоуровневый протокол, зависящий от используемого порта. TCP/IP резервирует первые 1024 порта для специфических протоколов. Порт номер 21 – для FTP, 23 – для Telnet, 25 – для электронной почты, 80 – для HTTP, 119 – для netnews. Например, HTTP – это протокол, используемый Web-браузерами и серверами для передачи гипертекста и графических изображений. Он работает следующим образом. Когда клиент запрашивает файл с сервера HTTP, это действие известно как попадание (hit) и состоит в простой отправке имени файла в определенном формате на предопределенный порт с последующим чтением содержимого этого файла. Сервер также сообщает код состояния, чтобы известить клиента о том, был ли запрос обработан или нет, и по какой причине. Ключевым компонентом Интернет является адрес. Каждый компьютер в Интернет обладает собственным адресом. Адрес Интернет представляет собой число, уникально идентифицирующее каждый компьютер в Интернет. Изначально все адреса состояли из 32-битных значений, организованных в четыре 8-битных значения. Адрес такого типа определен IPv4 (Интернет-протокол версии 4). Однако в последнее время все чаще используется новая схема адресации, называемая IPv6, которая предназначена для того, чтобы поддержать гораздо большее адресное пространство, чем IPv4. Точно так же, как IP-адрес описывает сетевую иерархию, имя адреса Интернет, называемое доменным именем, описывает местонахождение машины в пространстве имен. Например, www.polessu.by относится к домену by (зарезервированному для сайтов Республики Беларусь), имеет имя polessu (по названию организации), a www идентифицирует сервер, обрабатывающий Web-запросы. Доменное имя Интернет ПолесГУ отображается на IP-адрес посредством службы доменных имен (Domain Name Service – DNS). Это позволяет пользователям работать с доменными именами, в то время как Интернет оперирует IP-адресами. Все вышеописанные средства необходимы для написания сетевых приложений.

# **1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **Общие положения**

На сегодняшний день около 35% людей на земле имеют компьютер, из них значительная часть могут выйти в интернет. В видеоигры играют 44% интернет-пользователей. Исходя из этой статистики в мире видеоигры довольно популярный вид развлечения. Статистика последних лет показывает, что у 35% людей имеется компьютер, причем большинство из них имеет возможность выхода в Интернет. 44% интернет-пользователей играют видеоигры, в том числе через глобальную сеть. Отсюда имеем, что видеоигры являются весьма популярным видом развлечения, причем среди абсолютно различной аудитории.

Многопользовательские игры собирают большую аудиторию нежели однопользовательские, а сложность в создании многопользовательских игр значительно больше. Следует отметить, что аудитория у многопользовательских игр шире, нежели у однопользовательских. А процесс создания многопользовательских игр, разумеется, значительно сложнее.

Объектом исследования являются многопользовательские игры с управлением от первого лица.

Предметом исследования является система соединения игроков в многопользовательской игре с помощью сервера.

Целью проекта является объединить игроков в одной игре. (см выше)

Задачи:

- разработка многопользовательского FPS;

- разработка серверной части игры;

- разработка клиентской части игры;

- соединение игроков посредством сервера.

## **1.2 Сведения из теории**

Существует два варианта общения между пользователями: посредством сервера (такие приложения называют клиент-серверными) и Peer-to-peer. Что касается Peer-to-peer, это обмен данными напрямую между клиентами. В клиент-серверном приложении обмен данными между клиента, проходит через сервер. Разработка клиент-серверного приложения дает лучшую защищенность нежели peer-to-peer, так как все просчёты организуются на сервере.

Выбор стал на разработке клиент-серверного приложения. Так же стоял вопрос, какой использовать протокол для создания клиент-серверного приложения. Протокол UDP не дает гарантий на доставку данных, так же может смениться порядок доставки этих же данных, что касается TCP, эти нюансы исправлены, но есть минус этого протокола ­– он медленнее UDP.

## **1.3 Постановка задачи**

Название приложения – “Многопользовательский FPS”.

Назначение приложения – развлечение создание дополнительной функции для многопользовательских игр, позволяющей связывать игроков между собой.

Входные данные серверной части приложения, прием от клиента значений: координаты позиции, уровня здоровья и сообщений в чате. Входные данные клиентской части, прием от сервера значений другого игрока, таких как: координаты позиции, уровня здоровья и сообщений в чате. Выходные данные серверной части, совпадают с входными, только они адресованы определенному пользователю. Выходные данные клиентской части, отправка на сервер значений: координаты позиции, уровня здоровья и сообщений в чате.

Для того чтобы упростить создание 3D игры использовалась среда разработки игр Unity, она предоставляет широкий функционал для создания как 3D, так и 2D игр. Если бы игра создавалась с нуля потребовалось бы значительно больше времени, а также графическая часть игры явно отставала бы от современной. Что касается среды разработки, выбор был очевиден – это Visual Studio 2015.

Для работы приложения требуется: компьютер, подключение к интернету, запущенная серверная часть и клиентская часть приложения.

# **2 ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ**

## **2.1 Алгоритм решения**

Для создания сервера использовались два класса: Connection, ClientInfo. Класс Connection предоставляет функции:

* AddConnection – добавляет в список подключенного клиента
* BroadcastMessage – рассылка сообщений, кроме от кого было добавлено
* Disconnect – отключение сервера
* Listen – прослушивание входящих сообщений
* RemoveConnection – удаление из списка подключенных клиентов

Класс ClientInfo:

* ClientInfo – конструктор, генерирует ID клиента и добавляет подключение
* Close – закрытие сетевого потока
* GetMessage – прослушивание входящих сообщений
* Process – отсылка сообщений определенным клиентам и вывод их на консоль

Классы которые использовались при создании приложения для работы сети: TcpListener, TcpClient, NetworkStream. TcpListener использовался на серверной части приложения, для прослушивания клиентов. TcpClient также использовался на серверной части, для создания объекта подключенного клиента, а так же использовался на клиентской части для соединения с сервером. NetworkStream предназначен для передачи данных от клиента серверу, и наоборот.

## **2.2 Макет приложения**

После запуска приложения увидим такой экран:



Рисунок 2.1 – Меню приложения

После нажатия кнопки Play:

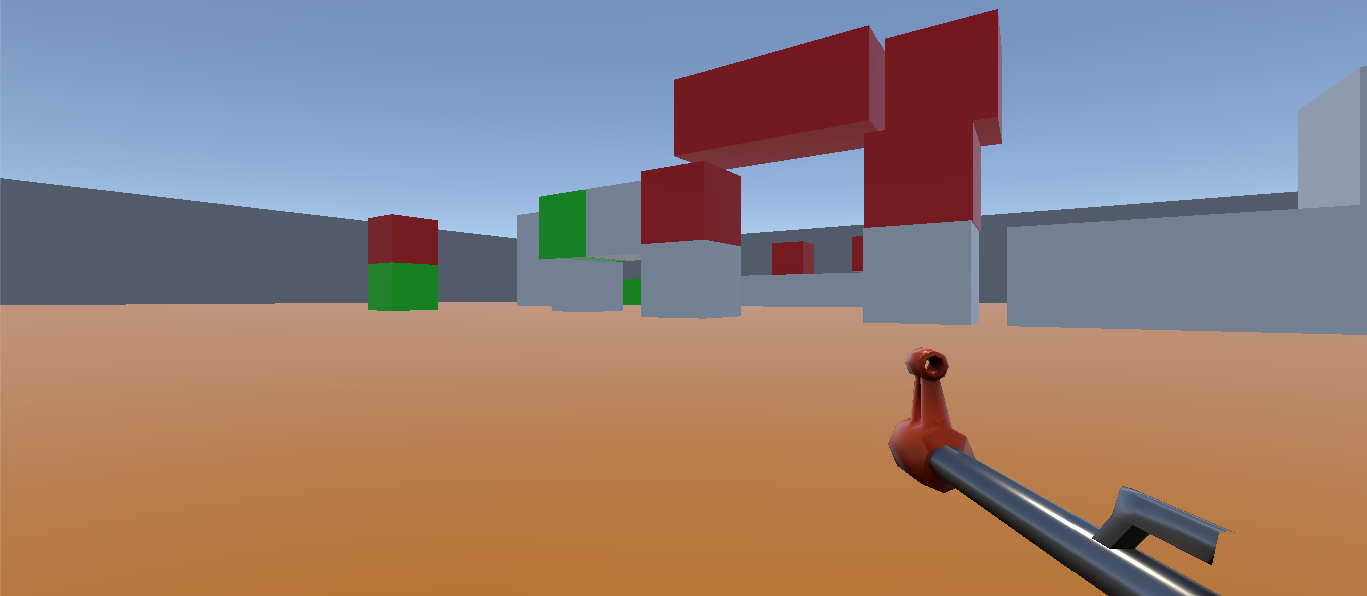


Рисунок 2.2 – Процесс игры

Серверная часть приложения:

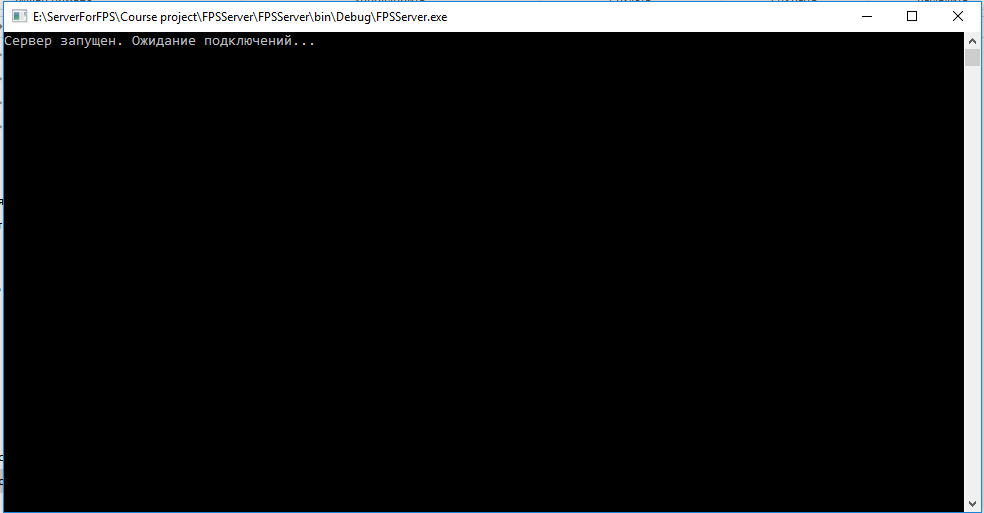


Рисунок 2.3 – Сервер

## **2.3 Описание программы**

# “Многопользовательский FPS” изначально создавался с развлекательной целью. Конечно эту видеоигру в будущем есть возможность монетизировать, что принесет некоторые средства, но для развития такого вида проекта, понадобится как минимум три человека в команду.

Ниже представлена диаграмма классов серверной части приложения.

# C:\Users\cimiox\AppData\Local\Microsoft\Windows\INetCacheContent.Word\диаграмма классов.png

Рисунок 2.4 – диаграмма классов

# **3** **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

## **3.1 Запуск приложения.**

Для работы клиентской части приложения, требуется запуск сервера, Рисунок 2.3, о готовности сервера нас проинформирует надпись “Сервер запущен”. Далее открывается возможность запуска самой видеоигры Рисунок 2.1.

## **3.2 Меню игры**

После запуска игры, требуется ввести ваш Nickname Рисунок 2.1. И наконец мы можем насладится игрой, после нажатия клавиши Play.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Wikipedia: свободная энциклопедия [Электронный ресурс] // URL: https://ru.wikipedia.org /wiki/База\_данных
2. Web Adequate: блог программиста [Электронный ресурс] // URL: <http://webadequate.ru/rabota-s-bd-mysql/16-identifikaciya-i-tipizaciya-obektov-v-mysql.html>
3. Библиотека MSDN [Электронных ресурс] // URL: <https://msdn.microsoft.com/ru-ru/library/ms233763.aspx>
4. Библиотека MSDN [Электронных ресурс] // URL: https://msdn.microsoft.com/ru-ru/library/vstudio/ms171925%28v=vs.110%29.aspx

# **ПРИЛОЖЕНИЕ А**

using System;

using System.Threading;

namespace FPSServer

{

class Program

{

static Connection server; // сервер

static Thread listenThread; // потока для прослушивания

static void Main(string[] args)

{

try

{

server = new Connection();

listenThread = new Thread(new ThreadStart(server.Listen));

listenThread.Start(); //старт потока

}

catch (Exception ex)

{

server.Disconnect();

Console.WriteLine(ex.Message);

}

}

}

}

using System;

using System.Net.Sockets;

using System.Text;

namespace FPSServer

{

class ClientInfo

{

protected internal string Id { get; private set; }

protected internal NetworkStream Stream { get; private set; }

string userName;

TcpClient client;

Connection server; // объект сервера

public ClientInfo(TcpClient tcpClient, Connection serverObject)

{

Id = Guid.NewGuid().ToString();

client = tcpClient;

server = serverObject;

serverObject.AddConnection(this);

}

public void Process()

{

try

{

Stream = client.GetStream();

// получаем имя пользователя

string message = GetMessage();

userName = message;

message = userName + " вошел в чат";

// посылаем сообщение о входе в чат всем подключенным пользователям

server.BroadcastMessage(message, this.Id);

Console.WriteLine(message);

// в бесконечном цикле получаем сообщения от клиента

while (true)

{

try

{

message = GetMessage();

message = String.Format("{0}: {1}", userName, message);

Console.WriteLine(message);

server.BroadcastMessage(message, this.Id);

}

catch

{

message = String.Format("{0}: покинул чат", userName);

Console.WriteLine(message);

server.BroadcastMessage(message, this.Id);

break;

}

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

finally

{

// в случае выхода из цикла закрываем ресурсы

server.RemoveConnection(this.Id);

Close();

}

}

// чтение входящего сообщения и преобразование в строку

private string GetMessage()

{

byte[] data = new byte[64]; // буфер для получаемых данных

StringBuilder builder = new StringBuilder();

int bytes = 0;

do

{

bytes = Stream.Read(data, 0, data.Length);

builder.Append(Encoding.UTF8.GetString(data, 0, bytes));

}

while (Stream.DataAvailable);

return builder.ToString();

}

// закрытие подключения

protected internal void Close()

{

if (Stream != null)

Stream.Close();

if (client != null)

client.Close();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

namespace FPSServer

{

class Connection

{

static TcpListener tcpListener; // сервер для прослушивания

List<ClientInfo> clients = new List<ClientInfo>(); // все подключения

protected internal void AddConnection(ClientInfo clientObject)

{

clients.Add(clientObject);

}

protected internal void RemoveConnection(string id)

{

// получаем по id закрытое подключение

ClientInfo client = clients.FirstOrDefault(c => c.Id == id);

// и удаляем его из списка подключений

if (client != null)

clients.Remove(client);

}

// прослушивание входящих подключений

protected internal void Listen()

{

try

{

tcpListener = new TcpListener(IPAddress.Loopback, 11000);

tcpListener.Start();

Console.WriteLine("Сервер запущен. Ожидание подключений...");

while (true)

{

TcpClient tcpClient = tcpListener.AcceptTcpClient();

ClientInfo clientObject = new ClientInfo(tcpClient, this);

Thread clientThread = new Thread(new ThreadStart(clientObject.Process));

clientThread.Start();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Disconnect();

}

}

// трансляция сообщения подключенным клиентам

protected internal void BroadcastMessage(string message, string id)

{

byte[] data = Encoding.UTF8.GetBytes(message);

for (int i = 0; i < clients.Count; i++)

{

if (clients[i].Id != id) // если id клиента не равно id отправляющего

{

clients[i].Stream.Write(data, 0, data.Length); //передача данных

}

}

}

// отключение всех клиентов

protected internal void Disconnect()

{

tcpListener.Stop(); //остановка сервера

for (int i = 0; i < clients.Count; i++)

{

clients[i].Close(); //отключение клиента

}

Environment.Exit(0); //завершение процесса

}

}

}

using UnityEngine;

using UnityEngine.UI;

public class Manager : MonoBehaviour

{

public static Manager Instance;

public GameObject NamePrefab;

public GameObject NicknamePanel;

public GameObject ConnectionPanel;

public GameObject ConReadyText;

public Text ErrorNickname;

public InputField NicknameInputField;

void Start()

{

if (Instance == null)

{

Instance = this;

}

else

{

Destroy(this);

}

DontDestroyOnLoad(this);

ConnectionPanel.SetActive(true);

ServerConnection.Instance.MyPlayer.SetActive(false);

}

private void Update()

{

if (ServerConnection.Instance.IsConnect)

{

ServerConnection.Instance.IsConnect = false;

ConReadyText.SetActive(true);

NicknamePanel.SetActive(true);

ConnectionPanel.SetActive(false);

}

}

public void Play()

{

try

{

if (NicknameInputField.text.Length != 0)

{

ServerConnection.Name = NicknameInputField.text;

ServerConnection.Instance.SendMessage(NicknameInputField.text);

ErrorNickname.gameObject.SetActive(false);

ServerConnection.Instance.MyPlayer.SetActive(true);

ServerConnection.Instance.SendMessage(10, 10, 10);

ServerConnection.Instance.SendMessage("<InstantiatePlayer>");

NicknamePanel.SetActive(false);

}

else

{

ErrorNickname.gameObject.SetActive(true);

}

}

catch (System.Exception)

{

print("Exception");

}

}

}

using System;

using System.Net.Sockets;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading;

using UnityEngine;

public class ServerConnection : MonoBehaviour

{

public static ServerConnection Instance;

public GameObject EnemyPrefab;

public GameObject EnemyGameObject;

public static bool SynchronizedPosition = false;

public static bool instantiatePlayer = false;

public GameObject MyPlayer;

static string Nickname;

private const string host = "127.0.0.1";

private const int port = 11000;

static TcpClient client;

static NetworkStream stream;

public bool IsConnect { get; set; }

private void Start()

{

IsConnect = false;

if (Instance == null)

{

Instance = this;

}

else

{

Destroy(this);

}

DontDestroyOnLoad(this);

Nickname = UnityEngine.Random.RandomRange(1, 100).ToString();

client = new TcpClient();

try

{

client.Connect(host, port);

if (client.Connected)

{

IsConnect = true;

}

stream = client.GetStream();

string message = Nickname;

byte[] data = Encoding.UTF8.GetBytes(message);

stream.Write(data, 0, data.Length);

Thread receivedThread = new Thread(new ThreadStart(ReceiveMessage));

receivedThread.Start();

print("enter " + Nickname);

}

catch (Exception ex)

{

print(ex.Message);

//Disconnect();

}

}

~ServerConnection()

{

Disconnect();

}

void Update()

{

if (instantiatePlayer)

{

InstantiatePlayer();

instantiatePlayer = false;

}

if (SynchronizedPosition)

{

SendMessage(MyPlayer.transform.position.x, MyPlayer.transform.position.y, MyPlayer.transform.position.z);

SendMessage(MyPlayer.transform.rotation.x, MyPlayer.transform.rotation.y, MyPlayer.transform.rotation.z, MyPlayer.transform.rotation.w);

}

if (EnemyGameObject != null)

{

EnemyGameObject.transform.position = new Vector3(PosX, PosY, PosZ);

EnemyGameObject.transform.rotation = new Quaternion(RotX, RotY, RotZ, RotW);

}

}

public new void SendMessage(string message)

{

EncodingAndWrite(message);

}

public new void SendMessage(float posX, float posY, float posZ)

{

string message = "<Position>" + "/<PosX>" + posX.ToString() + "/<PosY>" + posY.ToString() + "/<PosZ>" + posZ.ToString();

message.Trim();

EncodingAndWrite(message);

}

public new void SendMessage(float rotX, float rotY, float rotZ, float rotW)

{

string message = "<Rotation>" + "/<RotX>" + rotX.ToString() + "/<RotY>" + rotY.ToString() + "/<RotZ>" + rotZ.ToString() + "/<RotW>" + rotW.ToString();

message.Trim();

EncodingAndWrite(message);

}

private void EncodingAndWrite(string message)

{

byte[] data = Encoding.UTF8.GetBytes(message);

stream.Write(data, 0, data.Length);

}

private void ReceiveMessage()

{

while (true)

{

try

{

byte[] data = new byte[1024];

StringBuilder builder = new StringBuilder();

int bytes = 0;

do

{

bytes = stream.Read(data, 0, data.Length);

builder.Append(Encoding.UTF8.GetString(data, 0, bytes));

} while (stream.DataAvailable);

string message = builder.ToString();

if (message.Contains("<InstantiatePlayer>"))

{

instantiatePlayer = true;

}

else if (CheckMessage(message, "<Position>"))

{

ParseMessage(message, "<Position>");

}

else if (CheckMessage(message, "<Rotation>"))

{

ParseMessage(message, "<Rotation>");

}

else if (CheckMessage(message, "<EnemyHealth>"))

{

ParseMessage(message, "<EnemyHealth>");

}

}

catch (Exception ex)

{

print(ex);

Disconnect();

}

}

}

private bool CheckMessage(string message, string value)

{

return message.Contains(value);

}

private void ParseMessage(string message, string typeMsg)

{

if (message != null)

{

switch (typeMsg)

{

case "<Position>":

PosX = FindValuesInMessage("PosX", message);

PosY = FindValuesInMessage("PosY", message);

PosZ = FindValuesInMessage("PosZ", message);

break;

case "<Rotation>":

RotX = FindValuesInMessage("RotX", message);

RotX = FindValuesInMessage("RotY", message);

RotX = FindValuesInMessage("RotZ", message);

RotX = FindValuesInMessage("RotW", message);

break;

case "<EnemyHealth>":

foreach (Match item in new Regex(Pattern("EnemyHealth")).Matches(message))

{

MyPlayer.GetComponent<PlayerHealth>().currentHealth = Convert.ToInt32(item.Value.Remove(0, 13));

}

break;

}

}

}

private float FindValuesInMessage(string pattern, string message)

{

foreach (Match item in new Regex(Pattern(pattern)).Matches(message))

{

return Convert.ToSingle(item.Value.Remove(0, 6));

}

return 0;

}

private string Pattern(string value)

{

return string.Format(@"<{0}>.?\d+.?\d+", value);

}

public void InstantiatePlayer()

{

EnemyGameObject = Instantiate(EnemyPrefab);

EnemyGameObject.gameObject.transform.position = new Vector3(PosX, PosY, PosZ);

EnemyGameObject.gameObject.transform.rotation = new Quaternion(RotX, RotY, RotZ, RotW);

EnemyGameObject.name = Name;

SynchronizedPosition = true;

}

void Disconnect()

{

if (stream != null)

{

stream.Close();

}

if (client != null)

{

client.Close();

}

}

public static float PosX { get; set; }

public static float PosY { get; set; }

public static float PosZ { get; set; }

public static float RotX { get; set; }

public static float RotY { get; set; }

public static float RotZ { get; set; }

public static float RotW { get; set; }

public static string Name { get; set; }

}

using UnityEngine;

[RequireComponent(typeof(CharacterController))]

[AddComponentMenu("Control Scripts/PlayerMovement")]

public class PlayerMovement : MonoBehaviour

{

public float speed = 6.0f;

public float gravity = -9.8f;

public CharacterController \_characterController;

private void Start()

{

\_characterController = GetComponent<CharacterController>();

}

private void Update()

{

float deltaX = Input.GetAxis("Horizontal") \* speed;

float deltaZ = Input.GetAxis("Vertical") \* speed;

Vector3 movement = new Vector3(deltaX, 0, deltaZ);

movement = Vector3.ClampMagnitude(movement, speed);

movement.y = gravity;

movement \*= Time.deltaTime;

movement = transform.TransformVector(movement);

\_characterController.Move(movement);

}

}

using UnityEngine;

public class MouseLook : MonoBehaviour

{

public enum RotationAxes

{

MouseXAndY = 0,

MouseX = 1,

MouseY = 2

}

public RotationAxes axes = RotationAxes.MouseXAndY;

public float sensitivityHor = 9.0f;

public float sensitivityVer = 9.0f;

public float minimumVert = -45f;

public float maximumVert = 45f;

private float \_rotationX = 0;

void Start()

{

Rigidbody body = GetComponent<Rigidbody>();

if (body != null)

{

body.freezeRotation = true;

}

}

void Update()

{

if (axes == RotationAxes.MouseX)

{

transform.Rotate(0, Input.GetAxis("Mouse X") \* sensitivityHor, 0);

}

else if (axes == RotationAxes.MouseY)

{

\_rotationX -= Input.GetAxis("Mouse Y") \* sensitivityVer;

\_rotationX = Mathf.Clamp(\_rotationX, minimumVert, maximumVert);

float rotationY = transform.localEulerAngles.y;

transform.localEulerAngles = new Vector3(\_rotationX, rotationY, 0);

}

else

{

\_rotationX -= Input.GetAxis("Mouse Y") \* sensitivityVer;

\_rotationX = Mathf.Clamp(\_rotationX, minimumVert, maximumVert);

float delta = Input.GetAxis("Mouse X") \* sensitivityVer;

float rotationY = transform.localEulerAngles.y + delta;

transform.localEulerAngles = new Vector3(\_rotationX, rotationY, 0);

}

}

}