Я Панцулая Темур Генадьевич, взял тему “Создайте класс, который моделирует отправку и получение сообщений через протокол UDP.”

Описание проекта: Создание класса для UDP коммуникации в Unity

Цель проекта:

Разработать класс, который будет моделировать отправку и получение сообщений через UDP протокол в Unity. Это позволит реализовать простую систему связи между различными компонентами в игре или между клиентом и сервером.

Основные задачи:

1) Создание класса:

Определение необходимых полей: IP-адрес, порт, буфер для данных.
Методы для инициализации и очистки.

2) Методы отправки сообщений:

Метод SendMessage, который будет отправлять сообщение на указанный IP и порт.
Обработка возможных ошибок при отправке.

3) Методы получения сообщений:

Метод ReceiveMessage, который будет получать сообщения от указанного IP и порта.
Обработка ошибок и управление потоком данных.

`Update()`: Периодически проверяет `incomingMessages` и извлекает все доступные сообщения, передавая их для дальнейшей обработки.

4) Обработка данных:

Сериализация и десериализация сообщений для передачи объектов (например, игровых объектов или настроек).

Базовая реализация: Изначально класс будет работать со строками (string).

Расширяемость для передачи объектов: Продумать архитектуру, которая позволит в будущем легко интегрировать механизмы сериализации/десериализации для передачи структурированных игровых данных

Ожидаемый результат:

- Рабочий C#-скрипт UdpMessenger, который можно добавить на GameObject в Unity;
- Настраиваемые в Инспекторе Unity поля для портов и IP-адресов;
- Методы SendMessage и ReceiveMessages корректно обрабатывающие UDP-трафик;
- Безопасная передача данных между потоком приема и основным потоком Unity;
- Надежное управление жизненным циклом сокета и потока, предотвращающее ошибки при запуске/остановке приложения;
- Возможность легкой интеграции с UI для отправки и отображения сообщений.

Первый файл

```
using System.Net;
using System.Net.Sockets;
using UnityEngine;


public class UDPCommunicator
{
```

```
    private UdpClient udpClient;
    private IPEndPoint remoteEndPoint;
    private byte[] receiveBuffer;
    private const int BufferSize = 1024;
    private string ipAddress;
    private int port;
    // Инициализация клиента
    public void Initialize(string ipAddress, int port)
    {
        this.ipAddress = ipAddress;
        this.port = port;
        remoteEndPoint = new IPEndPoint(IPAddress.Parse(ipAddress), port);
        udpClient = new UdpClient();
        receiveBuffer = new byte[BufferSize];
    }
    // Очистка ресурсов
    public void Close()
    {
        if (udpClient != null)
        {
            udpClient.Close();
            udpClient = null;
        }
    }
}
```

Второй файл

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class MoveDirectionalPoint : MonoBehaviour
{
    const float velocity = 5f;
    public float mouseSpeed = 5f;
    public float orbitDamping = 10f;
    Vector3 localRot;
    Vector3 startPoint = new Vector3(8, 16, -30);
    private void Start()
    {
        transform.position = startPoint;
    }
    void Update()
    {
        if (Input.GetMouseButton(1))
        {
            localRot.y += Input.GetAxis("Mouse X") * mouseSpeed;
            Quaternion QT = Quaternion.Euler(0, localRot.y, 0f);
            transform.rotation = Quaternion.Lerp(transform.rotation, QT, orbitDamping * Time.deltaTime);
        }
```

```
        if (Input.GetKey(KeyCode.A))
        {
            transform.position -= transform.right * velocity * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.D))
        {
            transform.position += transform.right * velocity * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.W))
        {
            transform.position += transform.forward * velocity * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.S))
        {
            transform.position -= transform.forward * velocity * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.LeftShift))
        {
            transform.position += transform.up * velocity * Time.deltaTime;
        }
        if (Input.GetKey(KeyCode.Space))
        {
            transform.position -= transform.up * velocity * Time.deltaTime;
        }
    }
}
```

Третий файл

```
public class GameNetworkManager : MonoBehaviour
```

```
{
    private UDPCommunicator udp;
    void Start()
    {
        udp = gameObject.AddComponent<UDPCommunicator>();
        udp.Initialize("127.0.0.1", 9050); // Например, локальный адрес и порт
        udp.MessageReceived += OnMessageReceived;

        StartCoroutine(udp.ReceiveMessagesCoroutine());
    }
    void OnDestroy()
    {
        udp.Close();
    }
    private void OnMessageReceived(string message)
    {
        Debug.Log("Получено UDP сообщение: " + message);
    }
}
```

Четвёртый файл

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using UnityEngine;
public class UDPCommunicator
{
    private UdpClient udpClient;
    private IPEndPoint remoteEndPoint;
    private byte[] receiveBuffer;
    private const int BufferSize = 1024;

    private string ipAddress;
    private int port;
    // Инициализация клиента
```

```
public void Initialize(string ipAddress, int port)
{
    this.ipAddress = ipAddress;
    this.port = port;
    remoteEndPoint = new IPEndPoint(IPAddress.Parse(ipAddress), port);
    udpClient = new UdpClient();
    receiveBuffer = new byte[BufferSize];
}
// Очистка ресурсов
public void Close()
{
    if (udpClient != null)
    {
        udpClient.Close();
        udpClient = null;
    }
}
// Отправка сообщения в виде строки
public bool SendMessage(string message)
{
    try
    {
        byte[] data = Encoding.UTF8.GetBytes(message);
        udpClient.Send(data, data.Length, remoteEndPoint);
        return true;
    }
    catch (Exception e)
    {
        Debug.LogError($"Ошибка при отправке UDP сообщения: {e.Message}");
        return false;
    }
}
// Получение сообщения (блокирующий вызов)
public string ReceiveMessage()
{
    try
    {
```

```
            IPEndPoint senderEndPoint = new IPEndPoint(IPAddress.Any, 0);
            byte[] data = udpClient.Receive(ref senderEndPoint);
            string message = Encoding.UTF8.GetString(data);
            return message;
        }
        catch (SocketException e)
        {
            Debug.LogError($"Ошибка при получении UDP сообщения: {e.Message}");
            return null;
        }
        catch (Exception e)
        {
            Debug.LogError($"Общая ошибка при получении UDP сообщения: {e.Message}");
            return null;
        }
    }
}
```

Пятый файл

```
using UnityEngine;


public class GameNetworkManager : MonoBehaviour
{
    private UDPCommunicator udp;
    void Start()
    {
        // Добавляем компонент UDPCommunicator к этому объекту
        udp = gameObject.AddComponent<UDPCommunicator>();
        // Инициализируем с локальным IP и портом 9050 (можно заменить на нужные)
        udp.Initialize("127.0.0.1", 9050);
        // Подписываемся на событие получения сообщения
        udp.MessageReceived += OnMessageReceived;
        // Запускаем корутину для приема сообщений
        StartCoroutine(udp.ReceiveMessagesCoroutine());
    }
    void OnDestroy()
    {
        // Закрываем UDP клиент при уничтожении объекта
        udp.Close();
    }
    // Обработчик входящих сообщений
    private void OnMessageReceived(string message)
    {
        Debug.Log("Получено UDP сообщение: " + message);
        // Здесь можно добавить обработку игрового события или состояния
    }
}
```

Шестой файл

```
using System.Text;
using UnityEngine;
public static class MessageSerializer
{
    // Сериализация объекта в JSON и байты
    public static byte[] Serialize<T>(T obj)
    {
        string json = JsonUtility.ToJson(obj);
        return Encoding.UTF8.GetBytes(json);
    }
    // Десериализация байтов в объект T
    public static T Deserialize<T>(byte[] data)
    {
        string json = Encoding.UTF8.GetString(data);
        return JsonUtility.FromJson<T>(json);
    }
}
```