**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе «Star Wars» по дисциплине ИКСиС

**Выполнил**:

студенты гр. ИСТд-31

*Желепов Алексей Сергеевич*

*Харитонов Алексей Александрович*

**Проверил**:

доцент каф. ИВК, к.т.н., доцент

*Кандаулов Валерий Михайлович*

**Ульяновск, 2014**

# Содержание

[Содержание 2](#_Toc389687809)

[Список использованных сокращений и обозначений 3](#_Toc389687810)

[Список исполнителей 4](#_Toc389687811)

[Введение 5](#_Toc389687812)

[1.Описание проекта 6](#_Toc389687813)

[1.1 Описание клиентской части 6](#_Toc389687814)

[1.2 Описание серверной части 6](#_Toc389687815)

[1.2.1. Общее описание сервера приложения 6](#_Toc389687816)

[1.2.2. Описание серверного API приложения 7](#_Toc389687817)

[2.Руководство пользователя 11](#_Toc389687818)

[2.1 Общие сведения 11](#_Toc389687819)

[2.2. Описание общего интерфейса системы. 11](#_Toc389687820)

[2.2.1 Пользовательский интерфейс приложения 11](#_Toc389687821)

[Заключение 20](#_Toc389687822)

[Список использованных источников 21](#_Toc389687823)

[Приложение А. Исходные тексты программных модулей 22](#_Toc389687824)

# Список использованных сокращений и обозначений

ЗВ – Звездные войны

# Список исполнителей

1. Желепов Алексей Сергеевич
2. Харитонов Алексей Александрович

# Введение

Звездные войны – культовая эпическая фантастическая сага, включающая в себя 6 кинофильмов, а также анимационные сериалы, книги, комиксы, мультфильмы, телефильмы, книги, комиксы видео игры - все пронизанные единой сюжетной линией и созданные в единой фантастической вселенной “Звездных воин”. Звездные войны (в дальнейшем ЗВ) создали полноценную индустрию развлечений. По ней было создано много игр и разработанное приложение не исключение. Нашей работой является готовое клиент-серверная игра, способная обеспечивать игру для многих пользователей одновременно.

Приложение было создано на игровом движке Unity3D. Основной из причин выбора данного движка была возможность создавать приложения реального времени. Также в Unity3D существует возможность создавать 2d сцены. В нашем приложении каждый пользователь может стать сервером, либо присоединиться к уже ранее созданному.

После того как  пользователь зашел на сервер у него есть возможность выбрать сторону. Его кораблю дается ограниченный запас жизней и патронов. Кроме этого  на карте раскиданы бонусы в виде здоровья и пуль для пушек корабля.

Основная задача игры – уничтожить корабль противника.

# 1.Описание проекта

## 1.1 Описание клиентской части

При запуске приложения клиент может выбрать уровень на котором играть, на данный момент существует один уровень: “Dukuland”. После входа пользователь может присоединиться к серверу или создать свой. Для этого нужно указать IP адрес созданного сервера в текстовом поле в верхнем левом углу, и нажать присоединиться. После этого игрок попадает на игровое поле. Игровое поле состоит из двух баз, разбросанных по разные стороны карты, карта имеет ограничения поля для боевых действий и ограничена желтой линией. На карте присутствуют второстепенные элементы, такие как планеты, астероиды, звезда смерти. Например, если корабль столкнется с астероидом, то корабль разрушится. Второстепенным объектам свойственно перемещаться по игровой карте.

## 1.2 Описание серверной части

### 1.2.1. Общее описание сервера приложения

Сервер является важнейшей частью клиент-серверного приложения, потому что он обеспечивает синхронизацию между всеми игроками в процессе игры. Например, рассылка и синхронизация боевых кораблей игроков, выпущенных пуль, движение планет и астероидов. При этом сервером является один из игроков, т.о. когда кто-то из участников запускает приложение он имеет две возможности:

* создать свой сервер;
* подключиться к уже созданному серверу, указав ip-адрес сервера.

### 1.2.2. Описание серверного API приложения

Сервер игрового приложения написан при помощи средств Unity 3d и языка высокого уровня C# (код серверной части программы представлен в приложении настоящей пояснительной записки).

**Класс Network**

Данный класс осуществляет организацию соединения «клиент-сервер». Основные функции: создание сервера, подключение к серверу, создание сетевого экземпляра префаба игрока.

**Основные методы:**

**Network.Connect (string host, int remotePort, string password = "")** – выполняет подключение к серверу *host* с портом *remotePort* и паролем *password*. Метод возвращает перечисление NetworkConnectionError.

**Network.InitializeServer(int connections, int listenPort, bool useNat)** – создает сервер с максимально разрешенным количеством подключений *connections*; порт входящих подключений *listenPort*, а также *useNat*. Также возвращает перечисление *NetworkConnectionError*.

**Network.Instantiate(Object prefab, Vector3 position, Quaternion rotation, int group)** – создает экземпляр префаба *prefab* в сети в позиции *position* с поворотом *rotation* и группой *group*. Возвращает весь созданный объект, с которым после создания можно выполнить дополнительные действия.

**Основные свойства:**

**bool Network.isClient** и **bool Network.isServer** – определяют, является ли игра сервером или клиентом. Изначально оба  свойства установлены в значение *false*.

**string Network.incomingPassword** – свойство, задающее пароль для входящих подключений.

**NetworkPlayer Network.player** – возвращает экземпляр локального игрока *NetworkPlayer*.

**NetworkPeerType Network.peerType** – возвращает текущее состояние подключения: *Disconnected* (отключен), *Server* (запущен как сервер), *Client* (подключен к серверу), *Connecting* (попытка, в процессе подключения).

**NetworkPlayer[] Network.connections** – возвращает всех подключенных игроков. В игре-клиенте возвращается только игрок с сервера.

**Основные серверные события:**

**OnConnectedToServer()** – вызывается на клиенте при успешном подключении к серверу.

**OnDisconnectedFromServer(NetworkDisconnection info)** – вызывается на клиенте при отключении от сервера и на сервере при завершении подключений *Network.Disconnect()*. В *info* содержится причина отключения: *LostConnection* (потеря связи) и *Disconnected* (при успешном отключении).

**OnFailedToConnect(NetworkConnectionError error)** — вызывается на клиенте при ошибке подключения. Свойство *error* содержит ошибку типа *NetworkConnectionError*.

**OnNetworkInstantiate(NetworkMessageInfo info)** — вызывается на клиенте и сервере, если был создан новый экземпляр методом *Network.Instantiate()*. Содержит свойство *info* типа *NetworkMessageInfo*.

**OnPlayerConnected(NetworkPlayer player)** — вызывается на сервере при успешном подключении клиента и содержит *player* типа *NetworkPlayer*.

**OnPlayerDisconnected(NetworkPlayer player)** — вызывается на сервере при отключении клиента и содержит информацию об игроке, представленную типом *NetworkPlayer*.

**OnServerInitialized()** — вызывается на сервере, после того как сервер был успешно создан.

**OnSerializeNetworkView(BitStream stream, NetworkMessageInfo info)** — событие синхронизации компонента с сетью.

**Класс NetwokView**

Данный класс существует как компонент для игрового движка Unity 3D и предназначен для синхронизации компонентов в сети и для вызова RPC (Remote Procedure Call). Класс обладает такими свойствами синхронизации *NetworkStateSynchronization*:

* **Off** — не выполняет синхронизацию объекта, однако позволяет вызывать удаленные процедуры.
* **ReliableDeltaCompressed** — выполняет передачу пакетов поочередно и проверяет, доставлен ли пакет (*подобно протоколу TCP*).
* **Unreliable** — выполняет быструю отправку пакетов, не гарантируя доставки (*подобно протоколу UDP*).

**Основные методы:**

**networkView.RPC(string name, RPCMode mode, params object[] args)** — вызывает удаленную процедуру name, mode определяет получателей, *args* – аргументы для передачи процедуре.

**networkView.RPC(string name, NetworkPlayer target, params object[] args)** – то же, что и предыдущий метод, однако выполняет отправку конкретному игроку типа *NetworkPlayer*.

**Основные свойства:**

**bool networkView.isMine** – свойство, определяющее, является ли объект локальным. Весьма часто используется для проверки владельца объекта.

**Component networkView.observed** – компонент, который будет синхронизироваться. Если это скрипт, то он должен содержать метод*OnSerializeNetworkView(BitStream stream, NetworkMessageInfo info)*, упомянутый выше.

**NetworkPlayer networkView.owner** – свойство, возвращающее владельца объекта.

**NetworkStateSynchronization networkView.stateSynchronization** — тип синхронизации: *Off*, *ReliableDeltaCompressed*, *Unreliable*.

**NetworkViewID networkView.viewID** — уникальный идентификатор в сети для *NetworkView*.

# 2.Руководство пользователя

## 2.1 Общие сведения

Разработанное приложение – онлайн-игра, которая предназначена для поклонников линейки фильмов и игр “Star Wars”. Тестированием подтверждено, что в игру одновременно могут играть до восьми человек.

## 2.2. Описание общего интерфейса системы.

В текущем разделе будут описаны порядок пользования пользовательским интерфейсом приложения и игровое управление.

### 2.2.1 Пользовательский интерфейс приложения

1) Для запуска приложения необходимо запустить .exe файл программы (см. рис. 2.2.1.1).

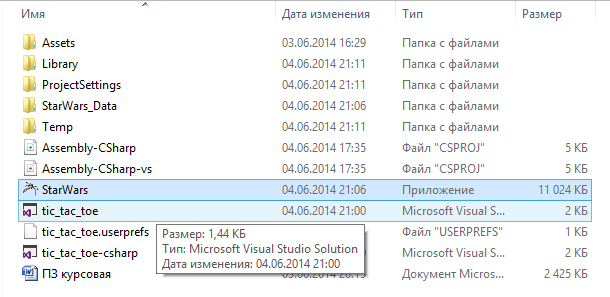


Рис. 2.2.1.1. Запуск .exe файла приложения

2)  В появившемся окне предлагается установка графических параметров приложения. В нашем случае установлен оконный режим и разрешение 800x600 пикселей (см. рис. 2.2.1.2). Для запуска игры необходимо нажать кнопку “Play!”.

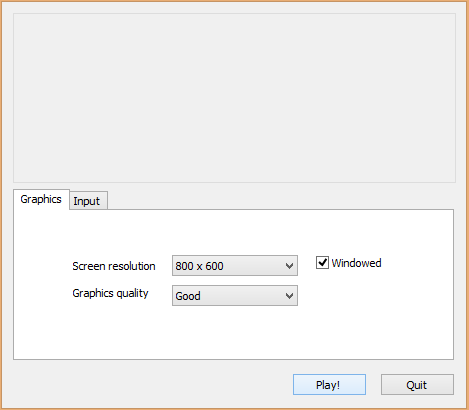


Рис. 2.2.1.2. Окно настройки графических параметров приложения

3) В результате должно открыться окно приложения, показанное на рисунке 2.2.1.3.



Рис. 2.2.1.3. Главное меню приложения Star Wars.

4) В пункте меню “Authors” пользователь может посмотреть информацию о разработчиках игры (рис. 2.2.1.4).

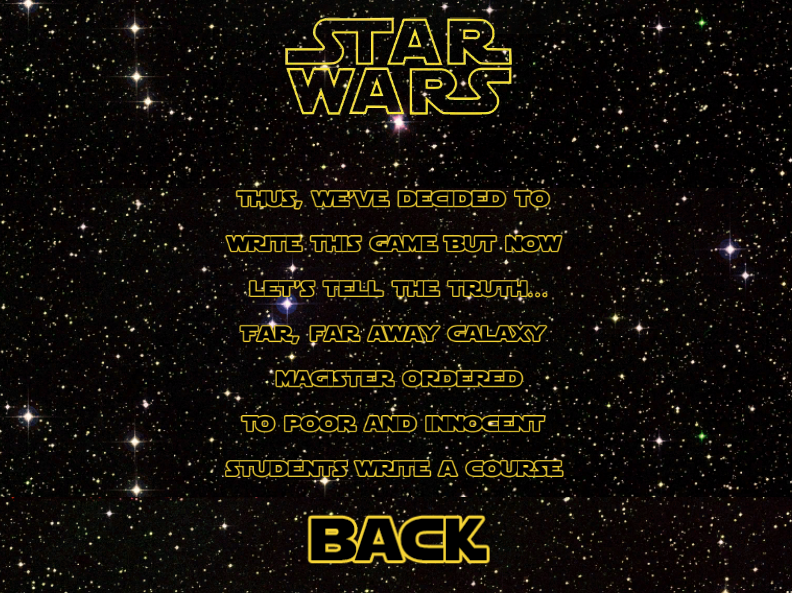


Рис. 2.2.1.4. Информация о разработчиках программы

5) Нажав на кнопку “Back”, можно вернуться в главное меню игры. Пункт главного меню “Exit” позволяет пользователю выйти из программы.

6) Пункт меню “Level Choice” позволяет пользователю выбрать карту на которой он сможет играть с другими пользователями (см. рис. 2.2.1.5).



Рис. 2.2.1.5. Выбор карты игрового мира

7) После выбора игрового мира игрок может создать свой сервер для игры с другими игроками, нажав на кнопку “Host”, либо присоединиться к уже существующему, указав ip-адрес сервера (см. рис. 2.2.1.6).



Рис. 2.2.1.6. Создание сервера

8) Далее пользователь должен выбрать сторону, за которую он хочет играть. Выбор осуществляется кликом по соответствующей иконке (см. рис. 2.2.1.7).



Рис. 2.2.1.7. Создание сервера

9) Все игроки начинают игру на своей базе (см. рис. 2.2.1.8). Управлять боевым истребителем нужно при помощи клавишей стрелок.

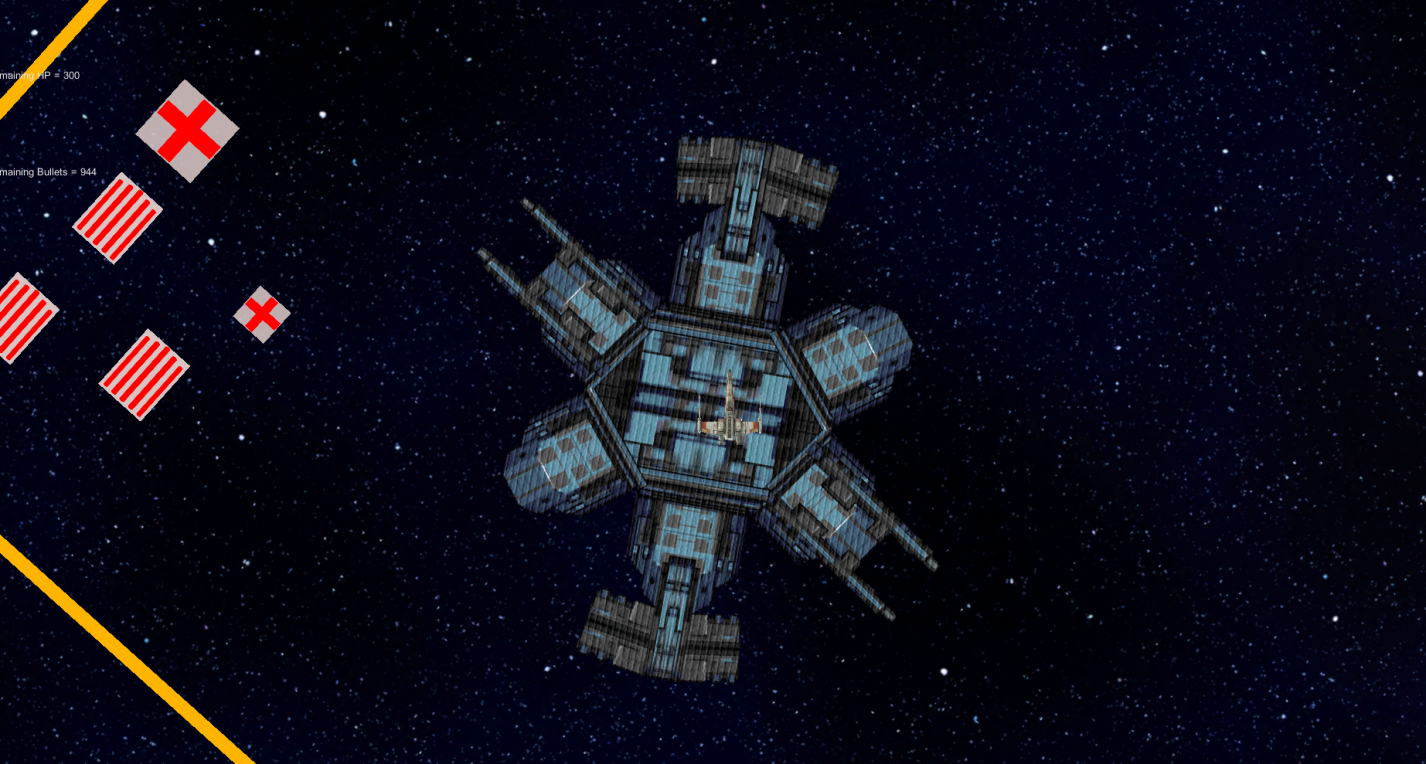


Рис. 2.2.1.8. Старт игры

10) Основная цель игры – уничтожение противников. При этом если игрок уничтожил чей-то истребитель или его самого сбили, он воскрешается на своей базе и после может продолжать игру. Стрельба осуществляется нажатием на клавишу “P” (см. рис. 2.2.1.9).



Рис. 2.2.1.9. Процесс стрельбы в игре

11) Процесс коллективной игры отображен на рисунке 2.2.1.10.



Рис. 2.2.1.10 Многопользовательский режим

# Заключение

Реализация курсовой работы выполнена в полном соответствии основных и дополнительных требований, которые соответствуют определенному уровню сдачи. В результате проведенной работы было реализовано клиент-серверное приложение. В будущем возможна доработка проекта до полноценной игры с добавлением новых уровней и игровых режимов.

# Список использованных источников

1. Unity 3d Network[Электронный ресурс] // К.Федоров. – Режим доступа: http://unity3dscripts.narod.ru/03/index.html. – Загл. с экрана.
2. Основы многопользовательской игры для Unity3D [Электронный ресурс] // В. Углицкий  – Режим доступа: http://habrahabr.ru/post/211202/–Загл. с экрана.
3. Unity3D Network [Электронный ресурс] // Б. Фаррис. - Режим доступа: http://www.youtube.com/channel/UCStD9dM0fAPxyJqrWWwFHPQ/ - Загл. с экрана.
4. Unity3d 2D development mode [Электронный ресурс] // М. Слободянюк. - Режим доступа: http://www.gamedev.ru/code/forum/?id=167237/- Загл. с экрана.
5. Unity3d Tutorial - Server Client tutorial series  [Электронный ресурс] // К. Паркер – Режим доступа: http://forum.unity3d.com/threads/unity3d-tutorial-server-client-tutorial-series-new.119990/ - Загл. с экрана.

# Приложение А. Исходные тексты программных модулей

В приложении приведены основные скрипты программы на языке высокого уровня C#.

1. PlayerScript.cs – отвечает за управление истребителем игрока и внутри игрового меню.

using UnityEngine;

using System.Collections;

public class PlayerScript : MonoBehaviour {

public Vector2 speed = new Vector2(50,50);

public Vector2 rotationSpeed = new Vector2(50,50);

public GameObject spawnPoint;

public float HP = 100.0f;

public Camera camera;

public float armour;

public int bulletsStorage;

public float coolDownTime;

private float HP\_start;

private Camera gameCamera;

private bool showCoolDown = false;

private bool isKilled = false;

public bool IsKilled{ get { return isKilled; } set { isKilled = value; } }

//GUI

public Texture2D quitTextureMessage;

public Texture2D yesTextureMessage;

public Texture2D noTextureMessage;

private bool isExitMenu = false;

private float widthExitMenu = 700;

private float heightExitMenu = 200;

// Use this for initialization

private void Start () {

if(networkView.isMine) {

HP\_start = HP;

gameCamera = GameObject.Find ("GameCamera").camera;

gameCamera.enabled = false;

camera.enabled = true;

}

else {

camera.enabled = false;

}

}

private void PlayerMove() {

float inputX = Input.GetAxis("Horizontal");

float inputY = Input.GetAxis("Vertical");

Vector3 movement = new Vector3(0, speed.y \* inputY, 0);

Vector3 rotation = new Vector3(0, 0, rotationSpeed.x \* inputX);

movement \*= Time.deltaTime;

rotation \*= Time.deltaTime;

this.transform.Translate(movement);

this.transform.Rotate(rotation);

}

// Update is called once per frame

private void Update () {

if(networkView.isMine) {

//Player movements

PlayerMove();

//ATTACK

if(Input.GetKey(KeyCode.P) && bulletsStorage > 0) {

WeaponScript weapon = this.GetComponent<WeaponScript>();

weapon.Attack();

}

//RESPAWN SPACESHIP

if(isKilled) {

isKilled = false;

showCoolDown = true;

camera.enabled = true;

gameCamera.enabled = false;

/\*this.transform.position = new Vector3(-10000,-10000,0);

float timer = coolDownTime;

Debug.Log ("Start" + coolDownTime);

while(timer >= 0.0f) {

timer -= Time.deltaTime;

Debug.Log (timer);

}

Debug.Log ("Finish" + timer);

showCoolDown = false;\*/

HP = HP\_start;

this.transform.position = spawnPoint.transform.position;

}

if(Input.GetKey(KeyCode.Escape)) {

isExitMenu = true;

}

}

}

private void OnCollisionEnter2D(Collision2D collider) {

if(networkView.isMine) {

if(collider.gameObject.tag == "Bullet") {

Network.Destroy(collider.gameObject);

HP -= collider.gameObject.GetComponent<ShotScript>().damage;

if(HP <= 0) {

/\*Network.RemoveRPCs(networkView.viewID);

Network.Destroy(this.gameObject);\*/

gameCamera.enabled = true;

isKilled = true;

}

}

if(collider.gameObject.tag == "Armour") {

}

if(collider.gameObject.tag == "BulletsBox") {

}

}

}

void OnGUI() {

if(networkView.isMine) {

GUILayout.Space(100);

GUILayout.Label("Remaining HP = " + HP.ToString());

GUILayout.Space(100);

GUILayout.Label("Remaining Bullets = " + bulletsStorage.ToString());

if(showCoolDown) {

GUILayout.Space(100);

GUILayout.Label("Please wait!");

}

if(isExitMenu) {

GUILayout.BeginArea(new Rect((Screen.width - widthExitMenu)/2, (Screen.height - heightExitMenu)/2, widthExitMenu, heightExitMenu));

{

GUILayout.Label(quitTextureMessage, GUIstyles.GetStyle());

GUILayout.BeginHorizontal();

{

if(GUILayout.Button (yesTextureMessage, GUIstyles.GetStyle())) {

Network.Destroy(this.gameObject);

Network.Disconnect();

network\_menu\_script.isChoiceDone = false;

isExitMenu = false;

Application.LoadLevel("menu");

}

if(GUILayout.Button (noTextureMessage, GUIstyles.GetStyle())) {

isExitMenu = false;

}

}

GUILayout.EndHorizontal();

}

GUILayout.EndArea();

}

}

if(!network\_menu\_script.Connected) {

Network.Destroy(this.gameObject);

Network.Disconnect();

network\_menu\_script.isChoiceDone = false;

isExitMenu = false;

Application.LoadLevel("menu");

}

}

}

2. NetworkMenuScript.cs – отвечает за создание сервера или присоединение к уже существующему игровому серверу.

using UnityEngine;

using System.Collections;

public class network\_menu\_script : MonoBehaviour {

public string connectionIP = "127.0.0.1";

public int portNumber = 8679;

public Texture2D btnConnectTexture;

public Texture2D btnHostTexture;

public Texture2D labelIPTexture;

public Texture2D labelPortTexture;

public Texture2D btnDarkSide;

public Texture2D btnLightSide;

private static bool connected = false;

public static bool choiceLightDark = false;

public static bool isChoiceDone = false; //true - light side of power, false - dark side of power

private static string myName = "Default";

private float tuneWindowWidth = 300;

private float tuneWindowHeight = 300;

public static bool Connected { get { return connected; } }

//public static string MyName { get { return myName; } }

private void OnConnectedToServer() {

//Somebody joined to the server

connected = true;

}

private void OnServerInitialized() {

//Master has initialized server

connected = true;

}

private void OnDisconnectedFromServer() {

//Someone exited from the chat

connected = false;

}

private void OnGUI() {

if(!connected) {

GUILayout.BeginArea(new Rect((Screen.width - tuneWindowWidth)/2, (Screen.height - tuneWindowHeight)/2, tuneWindowWidth, tuneWindowHeight));

{

GUILayout.BeginHorizontal();

{

GUILayout.Label(labelIPTexture, GUIstyles.GetStyle());

connectionIP = GUILayout.TextField(connectionIP);

}

GUILayout.EndHorizontal();

GUILayout.Space(10);

GUILayout.BeginHorizontal();

{

GUILayout.Label(labelPortTexture, GUIstyles.GetStyle());

int.TryParse(GUILayout.TextField (portNumber.ToString()), out portNumber);

}

GUILayout.EndHorizontal();

GUILayout.Space(10);

if(GUILayout.Button(btnConnectTexture, GUIstyles.GetStyle())) {

Network.Connect(connectionIP, portNumber);

}

GUILayout.Space(10);

if(GUILayout.Button(btnHostTexture, GUIstyles.GetStyle())) {

Network.InitializeServer(10, portNumber, true);

}

}

GUILayout.EndArea();

}

else {

if(!isChoiceDone) {

if(GUI.Button(new Rect(0,0,Screen.width/2, Screen.height), "", GUIstyles.SetImageOnBG(btnLightSide))) {

choiceLightDark = true;

isChoiceDone = true;

}

if(GUI.Button(new Rect(Screen.width/2,0,Screen.width/2, Screen.height), "", GUIstyles.SetImageOnBG(btnDarkSide))) {

choiceLightDark = false;

isChoiceDone = true;

}

}

//GUILayout.Label("Connections: " + Network.connections.Length.ToString());

}

}

}