Министерство образования и науки российской Федерации

КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. А.Н.ТУПОЛЕВА

Кафедра АСОИУ

# КУРСОВОЙ ПРОЕКТ

# по дисциплине

# «Инструментальные средства информационных сетей»

# «Сетевая логическая игра Точки»

**ВЫПОЛНИЛ:** студент гр.4508 Костромин И. А.

**ПРОВЕРИЛ:** доцент кафедры АСОИУ Бондаренко Б.П.

**Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2010 г.**

Казань 2010

Оглавление

[КУРСОВОЙ ПРОЕКТ 1](#_Toc282043832)

[по дисциплине 1](#_Toc282043833)

[«Инструментальные средства информационных сетей» 1](#_Toc282043834)

[«Сетевая логическая игра Точки» 1](#_Toc282043835)

[Введение 3](#_Toc282043836)

[Точки 3](#_Toc282043837)

[Правила 3](#_Toc282043838)

[Комментарии и замечания 4](#_Toc282043839)

[Цель 4](#_Toc282043840)

[Примечания 5](#_Toc282043841)

[Реализация сетевого взаимодействия 6](#_Toc282043842)

[Описание классов 7](#_Toc282043843)

[Пользовательский интерфейс. 8](#_Toc282043844)

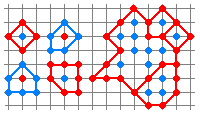
[Диаграмма классов 11](#_Toc282043845)

[Заключение 12](#_Toc282043846)

[ПРИЛОЖЕНИЕ. Исходный код клиента и сервера. 13](#_Toc282043847)

# Введение

## Точки

[](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Gra_w_kropki_bazy01.png)

**То́чки** (реже **города**, **фазенда**, **феодалы**) — [игра](http://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D1%8B_%D0%BD%D0%B0_%D0%B1%D1%83%D0%BC%D0%B0%D0%B3%D0%B5) для двух и более человек на поле в клетку.

## Правила

Правила этой игры похожи на правила игры [го](http://ru.wikipedia.org/wiki/%D0%93%D0%BE) (только камни не убираются с доски), но гораздо более легки для понимания, поэтому игра является некой "облегчённой" версией го. Вместо камней используются точки разных цветов, наносимые на бумагу [ручкой](http://ru.wikipedia.org/wiki/%D0%A0%D1%83%D1%87%D0%BA%D0%B0) или [карандашом](http://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%B0%D0%BD%D0%B4%D0%B0%D1%88). При игре ручками одного цвета для различия противников наравне с точками используются знаки «x», «o» и т. п. Поле игры — лист бумаги (в клетку) любого выбранного размера.

Соперники по очереди ставят точки на пересечении линий листа в клетку, каждый своим цветом.

Официальные правила (по ним проводят международные турниры в Киеве, а также онлайн-турниры)

1. Игра идет на поле размером 39х32 пунктов. Пунктом считается пересечение линий на поле. Играют двое, разными цветами.
2. Игроки ходят по очереди (1 ход — одна точка). Первый ход каждого из соперников происходит в центральной части поля (8х15). Последующие ходы могут быть в любой пункт, если только он не в окруженной области. Возможности пасовать (пропускать ход) нет. Факт окружения даёт право на дополнительный ход.
3. При создании непрерывной (по вертикали, горизонтали, диагонали) замкнутой линии образуется область. Если внутри неё есть точки противника (при этом могут быть пункты, не занятые чьими-либо точками), то это считается областью окружения, в которую далее запрещено ставить точку любому из игроков. Если точек соперника нет, то область свободная и в неё можно ставить точки.
4. При появлении в свободной области точки соперника свободная область будет считаться областью окружения при условии, что точка соперника не была завершающей в его окружении.
5. Точки, попавшие в область окружения, далее не участвуют в образовании линий для окружения.
6. Точки, поставленные на краю поля, не окружаются.
7. Партия заканчивается, когда не осталось свободных мест, по взаимному согласию игроков, либо когда один из игроков говорит/нажимает СТОП. Если игрок A останавливает игру у игрока B будет 3 минуты, в течение которых игрок B будет ставить точки один, доокружая свободные точки игрока А. По истечении 3-х минут игра заканчивается автоматически.
8. Победа определяется при подсчете окружённых точек (побеждает игрок, который окружил большее число точек соперника) или по взаимному согласию игроков.

## Комментарии и замечания

1. Пусть есть непрерывная замкнутая линия, которая ограничивает некоторую область. Но в этой области нет точек противника. Потом противник сделал ход в эту область, эта область будет тогда считаться окружающей, НО только в момент хода игрока, которому принадлежит область. При этом ход может быть в любое другое место на поле (не обязательно входить в состав той окружающей области).
2. Пусть есть окружающая область, которая (в момент Х) попала в окружение. В этом случае количество точек противника, которое область содержала до момента Х, не учитывается при подсчете окруженных точек в конце партии.
3. Игроку, который первый планирует сказать «стоп», следует обезопасить свои точки от окружения выходом к краю.
4. На фактическое обведение точки, поставленную в область, ход не тратится — после обведения можно ходить куда угодно
5. Для каждой партии или турнира устанавливаются временные ограничения, оговариваемые отдельно.

## Цель

В классических точках: точки (окружить, взять в плен больше точек)

В «захвате»: территория (захватить больше территории)

Окружение должно быть построено так, чтобы расстояние между точками составляло не более одной клетки — по прямой или по диагонали. Пленные точки в окружении участвовать не могут. Почти во всех вариантах игры территория без пленных точек противника не захватывается. Ставить точку в территорию, захваченную противником, запрещается либо эта точка считается захваченной, если не завершает собой окружения, разрывающего линию.

Есть вариант правил, когда нельзя окружить группу точек, соединенную с краем листа, если они представляют из себя неразрывную группу или линию без брешей. Называется это «заземлением». (Но если углы при поворотах не заполнены, линию можно все же разбить по диагонали угловой клетки).

## Примечания

В моей реализации есть некоторые ограничения и несоответствия «официальным» правилам, поскольку алгоритмы поиска окружений и заблокированных точек писались давно, когда я еще не знал про существование официальных правил. Возможно, что их тогда еще и не было.

В данной реализации я сконцентрировал усилия на обеспечении сетевого взаимодействия между игроками. Платформой для разработки выступил .NET Framework версии 3.5 SP1. Для разработки графического интерфейса я воспользовался технологией Windows Presentation Foundation (WPF).

# Реализация сетевого взаимодействия

Для реализации сетевого взаимодействия необходимо, во-первых, определиться со способом этого взаимодействия. Это может быть написание клиент-серверного приложения, в котором сервер будет создавать все игры и следить за игрой подключившихся к нему клиентов, но мы выберем в данном случае более простой вариант – клиент-сервер, работающий в локальной сети таким образом, что каждый пользователь может одновременно и создавать игру, и присоединяться к созданной другим пользователем игре.

Для этого была проработана следующая модель поведения. При создании игры программа-хост запускает UDP-клиент, который по таймеру рассылает широковещательные сообщения с информацией о созданной игре. Клиенты, которые слушают указанные UDP-порты, получают эту информацию и могут попытаться соединиться с хостом уже по TCP-протоколу. После того, как клиент подключился к серверу по TCP, рассылка широковещательных UDP-сообщений прекращается. Если же игрок зашел не в ту игру и вышел из неё, рассылка широковещательных сообщений снова возобновляется. После того, как клиент зашел к хосту в игру, они могут обмениваться текстовыми сообщениями друг с другом. Этим способом они могут договориться о старте игры. Хост может стартовать. Когда хост стартует игру, игра начинается, клиенту приходит специальное сообщение о том, что игра началась, и его программа отображает окно с игровым полем. Параллельно, серверное приложение также отображает окно с игровым полем. После этого игроки делают ходы по очереди, а программы автоматически подсчитывают их очки. Игра заканчивается по взаимному согласию или же когда не осталось больше мест, куда ставить точки. Если игрок покидает игру, другому игроку придет об этом сообщение, и он может создать игру снова.

# Описание классов

**GameInfo** – хранит информацию о созданной игре: имя игрока, создавшего игру, его IP-адрес, название игры, количество подключенных игроков.

GameClient-обеспечивает подключение к созданной игре. Технически представляет собой UDP сервер, принимающий широковещательные сообщения о созданных играх плюс TCP клиент, соединяющийся при присоединении к одной из созданных игр.

**GameServer**-обеспечивает создание игры и принимает входящие соединения от игроков. Представляет собой UDP-клиент, рассылающий широковещательные сообщения о созданной игре плюс TCP сервер, принимающий входящие соединения от игроков.

**ICommunicator**- интерфейс, общий для GameClient и GameServer. Используя его, мы можем в некоторых местах абстрагироваться от того, имеем ли мы дело с сервером игры или с клиентом, и написать общую логику для обоих случаев.

**IGameProtocol** – интерфейс, определяющий логику игрового протокола. В нем определены методы наподобие «послать текстовое сообщение другому игроку», «послать координаты следующего хода другому игроку» и события, оповещающие об аналогичных действиях со стороны противника.

GameProtocol-реализация интерфейса IGameProtocol.

Map- инкапсулирует логику по поиску окружений и окруженных точек. Подсчитывает очки.

# Пользовательский интерфейс.

Пользовательский интерфейс сделан с использованием такой мощной штуки как Windows Presentation Foundation, поэтому и код, определяющий внешний вид, очень выразителен:

<Window x:Class="DotsGame.GameWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="GameWindow" xmlns:my="clr-namespace:DotsGame" mc:Ignorable="d" xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

MaxHeight="435" MaxWidth="670" Closed="Window\_Closed">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="408"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="507"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<my:MapControl Margin="5" Grid.Row="0" Grid.Column="0" x:Name="mapControl1" />

<Grid HorizontalAlignment="Stretch" Grid.Row="0" Grid.Column="1" VerticalAlignment="Stretch">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="125"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Label Grid.Row="0" Grid.ColumnSpan="2" Content="Здесь написано кто ходит" FontWeight="Bold" Height="34" Name="labelMove" VerticalAlignment="Top" HorizontalAlignment="Center" />

<Label Grid.Row="1" Grid.Column="0" Content="Ваши очки" HorizontalAlignment="Left" Name="label2" VerticalAlignment="Top" />

<Label Grid.Row="1" Grid.Column="1" Content="0" HorizontalAlignment="Left" Name="labelMyPoints" Height="23" />

<Label Grid.Row="2" Grid.Column="0" Content="Очки противника" HorizontalAlignment="Left" Name="label3" VerticalAlignment="Top" />

<Label Grid.Row="2" Grid.Column="1" Content="0" HorizontalAlignment="Left" Name="labelEnemyPoints" VerticalAlignment="Top" />

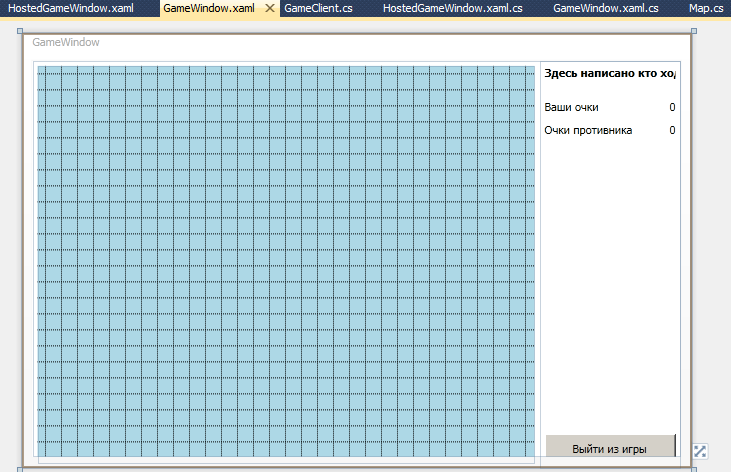
<Button Content="Выйти из игры" Grid.Row="3" Grid.ColumnSpan="2" Height="30" Margin="5" VerticalAlignment="Bottom" Name="buttonLeaveGame" Click="buttonLeaveGame\_Click" />

</Grid>

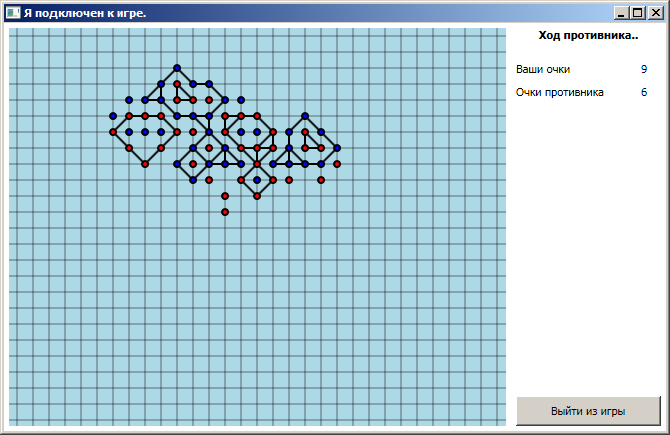
</Grid>

</Window>

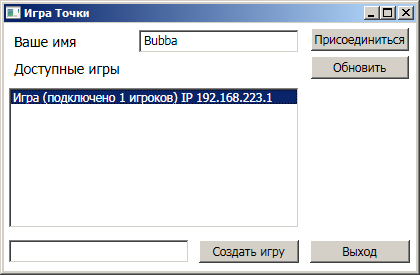
Как можно заметить, определение интерфейса очень лаконично и понятно реализуется с помощью XAML. Вот как выглядит это в дизайнере:



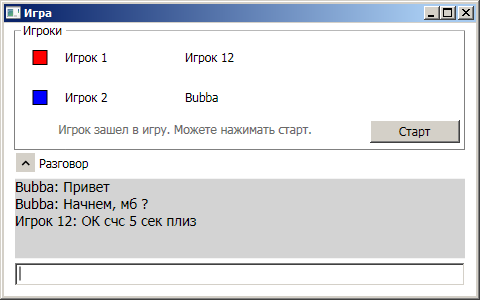
А вот как в процессе выполнения:



Несколько скриншотов пользовательского интерфейса:

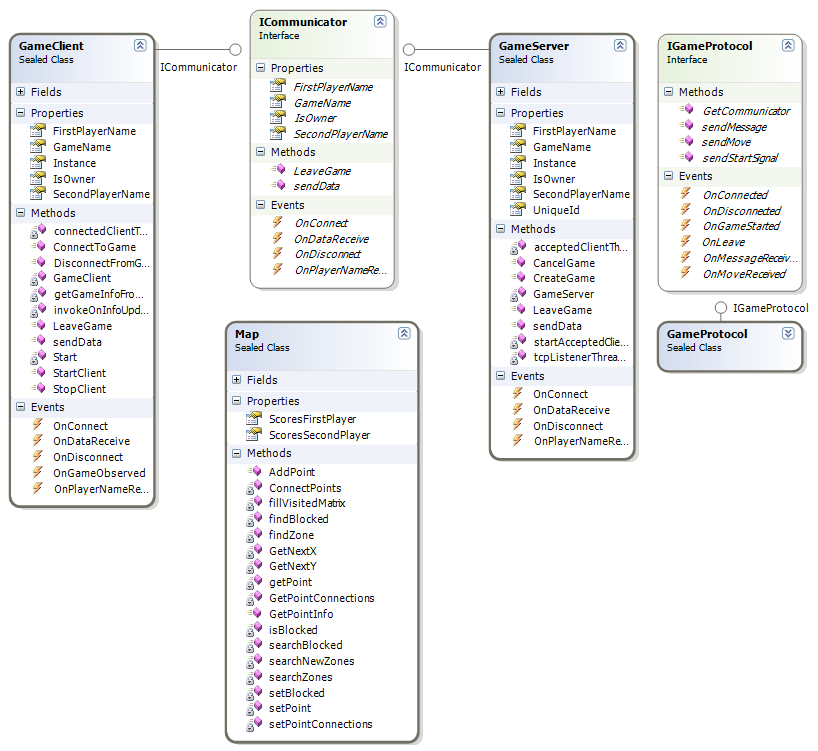


Это – главное окно приложения. Здесь отображается имя игрока, список доступных игр и инструменты для создания своей игры. Для того, чтобы создать игру, нужно ввести ее название (или оставить поле пустым) и нажать кнопку «Создать игру».



Это окно создания игры. В это же окошко пользователь попадает, когда подключается к чьей-то созданной игре.

# Диаграмма классов



На диаграмме представлены основные классы с методами и свойствами, которые они содержат. Все классы, к сожалению, в диаграмму не поместились, поскольку там было множество вложенных классов и классов, отвечающих за пользовательский интерфейс.

# Заключение

Я реализовал сетевое взаимодействие в одноранговой локальной сети и получил опыт программирования сетевых и многопоточных приложений, плюс наконец доработал свою старую программу, написанную еще на Borland C++ Builder, переписав основные алгоритмы с С++ на современный язык C#.

# ПРИЛОЖЕНИЕ. Исходный код клиента и сервера.

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

namespace DotsGame {

public sealed class GameObservedEventArgs : EventArgs {

public GameInfo Game {

get;

private set;

}

public GameObservedEventArgs(GameInfo game) {

this.Game = game;

}

}

public delegate void GameObservedEventHandler(object sender, GameObservedEventArgs args);

/// <summary>

/// Представляет собой UDP сервер, принимающий широковещательные сообщения о созданных играх

/// плюс TCP клиент для подключения к хосту созданной игры.

/// </summary>

public sealed class GameClient : ICommunicator {

public static int[] UDP\_PORTS = new[] {

1435,

1436,

};

private int usedUdpPort;

private GameClient() {

}

private static GameClient instance;

public static GameClient Instance {

get {

return instance ?? (instance = new GameClient());

}

}

public event GameObservedEventHandler OnGameObserved;

private void invokeOnInfoUpdated(GameObservedEventArgs e) {

GameObservedEventHandler handler = this.OnGameObserved;

if (handler != null) {

handler(this, e);

}

}

private Thread listeningThread;

private bool isStopped = false;

private readonly object locker = new object();

private readonly EventWaitHandle waitHandle = new EventWaitHandle(false, EventResetMode.AutoReset);

public void StartClient() {

// Пробуем занять один из портов, на которые будут присылаться широковещательные оповещения

UdpClient client = null;

foreach (int port in UDP\_PORTS) {

try {

client = new UdpClient(port);

usedUdpPort = port;

break;

} catch (SocketException) {

//

}

}

if (client == null) {

throw new InvalidOperationException("Ну все порты заняты, чо."); // Провал

}

listeningThread = new Thread(Start);

listeningThread.Start(client);

}

public void StopClient() {

if (listeningThread != null) {

lock (locker) {

isStopped = true;

}

waitHandle.Set();

listeningThread.Join();

}

}

private TcpClient connectedClient;

private Thread connectedClientThread;

private readonly EventWaitHandle connectedClientThreadWaitHandle = new EventWaitHandle(false,

EventResetMode.AutoReset);

private bool connectedClientThreadStopped;

/// <summary>

/// Извлекает IP адрес хоста игры и пытается с ним соединиться.

/// При успешном соединении запускает поток, который обрабатывает TCP соединения, по кд

/// посылая туда накопившиеся данные и забирая то, что приходит со стороны сервера.

/// </summary>

/// <param name="game">Один из объектов, полученных при прослушивании UDP.</param>

/// <param name="playerName"></param>

public void ConnectToGame(GameInfo game, string playerName) {

try {

connectedClient = new TcpClient();

connectedClient.Connect(new IPEndPoint(game.OwnerIP, GameServer.TCP\_PORT));

// Отсылаем имя подключившегося игрока

connectedClient.Client.Send(Encoding.UTF8.GetBytes("player:" + playerName));

// Стартуем поток, обрабатывающий TCP соединение с сервером игры

connectedClientThread = new Thread(connectedClientThreadFunc);

connectedClientThread.IsBackground = false;

connectedClientThread.Start(connectedClient);

//

firstPlayerName = game.OwnerName;

secondPlayerName = playerName;

EventHandler eventHandler = OnPlayerNameReceived;

if (eventHandler != null) {

eventHandler.Invoke(this, EventArgs.Empty);

}

//

this.gameName = game.Name;

//

EventHandler handler = OnConnect;

if (handler != null)

{

handler.Invoke(this, EventArgs.Empty);

}

//

} catch (Exception) {

connectedClient = null;

connectedClientThread = null;

firstPlayerName = null;

secondPlayerName = null;

gameName = null;

//

throw;

}

}

public void DisconnectFromGame() {

if (connectedClient != null) {

if (connectedClientThread.IsAlive) {

lock (locker) {

connectedClientThreadStopped = true;

}

connectedClientThreadWaitHandle.Set();

connectedClientThread.Join();

//

connectedClient = null;

connectedClientThread = null;

connectedClientThreadStopped = false;

//

firstPlayerName = null;

secondPlayerName = null;

EventHandler eventHandler = OnPlayerNameReceived;

if (eventHandler != null) {

eventHandler.Invoke(this, EventArgs.Empty);

}

//

gameName = null;

}

}

}

private void connectedClientThreadFunc(object param) {

TcpClient client = (TcpClient) param;

bool receivingStarted = false;

//

try {

while (client.Connected) {

//

if (!receivingStarted) {

receivingStarted = true;

Byte[] bytes = new byte[256];

client.GetStream().BeginRead(bytes, 0, bytes.Length, ar => {

try

{

int readed = client.GetStream().EndRead(ar);

// Raise event about new data readed

DataReceiveEventArgs args = new DataReceiveEventArgs(bytes, readed);

DataReceiveEventHandler handler = OnDataReceive;

if (null != handler)

{

handler.Invoke(this, args);

}

//

Debug.Print("Readed {0} bytes : {1}", readed, Encoding.UTF8.GetString(bytes));

Logger.Log("GAME CLIENT : Readed " + readed + " bytes : " +

Encoding.UTF8.GetString(bytes));

receivingStarted = false;

}

catch (ObjectDisposedException)

{

}

catch (IOException)

{

lock (locker)

{

connectedClientThreadStopped = true;

}

}

catch (InvalidOperationException) {

// **TODO : Пофиксить. Если много раз заходить/перезаходить в игру, то здесь падает**

}

}, null);

}

// Записываем все данные из очереди в поток tcp client'а

while (true) {

lock (locker) {

if (queue.Count == 0) {

break;

}

}

//

byte[] bytesToWrite;

lock (locker) {

bytesToWrite = queue[0];

queue.RemoveAt(0);

}

client.Client.Send(bytesToWrite);

Logger.Log("GAME CLIENT : Sent : " + Encoding.UTF8.GetString(bytesToWrite));

}

//

lock (locker) {

if (connectedClientThreadStopped && queue.Count == 0) {

break;

}

}

connectedClientThreadWaitHandle.WaitOne(200);

}

} catch (Exception exc) {

Debug.Print("Unexpected exception in connectedClientThreadFunc : {0}", exc);

} finally {

client.Close();

try {

EventHandler handler = this.OnDisconnect;

if (null != handler) {

handler.Invoke(this, EventArgs.Empty);

}

} catch (Exception) {

Debug.Print("Unexpected exception while handler of OnDisconnected been invoked.");

}

}

}

private void Start(object o) {

Debug.Print("Listening thread begin.");

UdpClient client = (UdpClient) o;

bool clientIsListening = false;

while (true) {

waitHandle.WaitOne(500);

lock (locker) {

if (isStopped) {

break;

}

}

if (!clientIsListening) {

clientIsListening = true;

client.BeginReceive(ar => {

try {

UdpClient udpClient = (UdpClient) ar.AsyncState;

// В эту переменную запишется endpoint хоста игры, которую мы нашли

// Потом мы будем пытаться законнектиться по TCP на этот IP

IPEndPoint gameHostEndPoint = new IPEndPoint(IPAddress.Any, usedUdpPort);

byte[] receivedBytes = udpClient.EndReceive(ar, ref gameHostEndPoint);

GameInfo gameInfo = getGameInfoFromData(receivedBytes);

gameInfo.OwnerIP = gameHostEndPoint.Address;

//

invokeOnInfoUpdated(new GameObservedEventArgs(gameInfo));

//

} catch (Exception) {

Debug.Print("Unexpected exception in UDP listening callback.");

} finally {

clientIsListening = false;

}

}, client);

}

}

Debug.Print("Listening thread finished.");

}

private static GameInfo getGameInfoFromData(byte[] data) {

GameInfo gameInfo = new GameInfo();

string gameInfoString = Encoding.UTF8.GetString(data);

string[] gameInfoStringParts = gameInfoString.Split(new[] {

":"

}, StringSplitOptions.None);

gameInfo.UniqueHostId = new Guid(gameInfoStringParts[0]);

gameInfo.Name = gameInfoStringParts[1];

gameInfo.OwnerName = gameInfoStringParts[2];

gameInfo.PlayersConnectedCount = Int32.Parse(gameInfoStringParts[3]);

return gameInfo;

}

private readonly List<byte[]> queue = new List<byte[]>();

public void sendData(byte[] data) {

lock (locker) {

queue.Add(data);

}

}

public event DataReceiveEventHandler OnDataReceive;

public event EventHandler OnDisconnect;

public event EventHandler OnConnect;

public void LeaveGame() {

sendData(Encoding.UTF8.GetBytes("leave:"));

//

DisconnectFromGame();

//

firstPlayerName = null;

secondPlayerName = null;

EventHandler handler = OnPlayerNameReceived;

if (null != handler) {

handler.Invoke(this, EventArgs.Empty);

}

}

private const string UNKNOWN\_PLAYER\_NAME = "Неизвестный игрок";

private string firstPlayerName;

public string FirstPlayerName {

get {

if (string.IsNullOrEmpty(firstPlayerName)) {

return UNKNOWN\_PLAYER\_NAME;

}

return firstPlayerName;

}

}

private string secondPlayerName;

public string SecondPlayerName {

get {

if (string.IsNullOrEmpty(secondPlayerName)) {

return UNKNOWN\_PLAYER\_NAME;

}

return secondPlayerName;

}

}

public event EventHandler OnPlayerNameReceived;

public bool IsOwner {

get {

return false;

}

}

private string gameName;

public string GameName {

get {

return gameName;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

namespace DotsGame {

/// <summary>

/// Представляет собой UDP клиент, рассылающий по таймеру broadcast-сообщения

/// с информацией о созданной игре плюс TCP сервер, принимающий соединения от игроков и судей.

/// </summary>

public sealed class GameServer : ICommunicator {

/// <summary>

/// Уникальный номер, идентифицирующий этот экземпляр программы, создающей игры.

/// При получении информации о созданной игре мы проверяем, не мы ли собственно сами ее же и создали.

/// Если да, то не выводим в список игр свою игру.

/// Генерируется в конструкторе класса.

/// </summary>

public Guid UniqueId {

get;

private set;

}

private GameServer() {

UniqueId = Guid.NewGuid();

}

private static GameServer instance;

public static GameServer Instance {

get {

return instance ?? (instance = new GameServer());

}

}

public static int TCP\_PORT = 1234;

private Timer timer;

public void CreateGame(string name, string owner) {

// Создаем коннектор UDP, который начинает рассылать широковещательные сообщения о созданной игре

IPAddress broadcastIp = IPAddress.Broadcast;

List<IPEndPoint> broadcastEndPoint = new List<IPEndPoint>(GameClient.UDP\_PORTS.Length);

foreach (int port in GameClient.UDP\_PORTS) {

broadcastEndPoint.Add(new IPEndPoint(broadcastIp, port));

}

UdpClient client = new UdpClient();

timer = new Timer(state => {

if (this.clientsAcceptedCount == 0) {

byte[] bytes =

Encoding.UTF8.GetBytes(UniqueId + ":" + name + ":" + owner + ":" + clientsAcceptedCount + 1);

foreach (IPEndPoint endPoint in broadcastEndPoint) {

client.Send(bytes, bytes.Length, endPoint);

}

}

}, null, TimeSpan.Zero, TimeSpan.FromSeconds(1));

// Создаем TCP сервер, который будет принимать соединения игроков, желающих поиграть

tcpListener = new TcpListener(new IPEndPoint(IPAddress.Any, TCP\_PORT));

tcpListener.Start();

//

tcpListenerThread = new Thread(tcpListenerThreadFunc);

tcpListenerThread.IsBackground = false;

tcpListenerThread.Start(tcpListener);

//

this.firstPlayerName = owner;

EventHandler handler = OnPlayerNameReceived;

if (null != handler) {

handler.Invoke(this, EventArgs.Empty);

}

//

gameName = name;

}

private TcpListener tcpListener;

private Thread tcpListenerThread;

private readonly object locker = new object();

private readonly EventWaitHandle tcpListenerThreadWaitHandle = new EventWaitHandle(false,

EventResetMode.AutoReset);

private bool tcpListenerThreadStopped = false;

private int clientsAcceptedCount = 0;

private Thread acceptedClientThread;

private readonly EventWaitHandle acceptedClientThreadWaitHandle = new EventWaitHandle(false,

EventResetMode.AutoReset);

private bool acceptedClientThreadStopped;

private void tcpListenerThreadFunc(object param) {

Debug.Print("TCP Listener thread has started.");

TcpListener listener = (TcpListener) param;

bool anyListeningStarted = false;

while (true) {

if (!anyListeningStarted) {

anyListeningStarted = true;

listener.BeginAcceptTcpClient(ar => {

try {

TcpClient acceptedClient = listener.EndAcceptTcpClient(ar);

if (clientsAcceptedCount > 0) {

acceptedClient.Close();

} else {

clientsAcceptedCount++;

//

startAcceptedClientThread(acceptedClient);

}

} catch (Exception) {

Debug.Print("Unexpected exception in TCP Listener BeginAccept callback.");

} finally {

anyListeningStarted = false;

}

}, null);

}

tcpListenerThreadWaitHandle.WaitOne(200);

lock (locker) {

if (tcpListenerThreadStopped) {

break;

}

}

}

Debug.Print("TCP Listener thread has finished.");

}

private void startAcceptedClientThread(TcpClient acceptedClient) {

acceptedClientThreadStopped = false;

acceptedClientThread = new Thread(acceptedClientThreadFunc);

acceptedClientThread.IsBackground = false;

acceptedClientThread.Start(acceptedClient);

EventHandler handler = OnConnect;

if (handler != null) {

handler.Invoke(this, EventArgs.Empty);

}

}

/// <summary>

/// Этот поток постоянно прослушивает клиентский сокет.

/// Если от клиента приходят какие-либо данные, то вызывается событие.

/// Если же клиенту нужно отправить данные, они сначала складываются в очередь и потом во время

/// работы потока отправляются.

/// </summary>

/// <param name="param"></param>

private void acceptedClientThreadFunc(object param) {

TcpClient acceptedClient = (TcpClient) param;

//

bool receivingStarted = false;

try {

while (acceptedClient.Connected) {

if (!receivingStarted) {

receivingStarted = true;

Byte[] bytes = new Byte[256];

acceptedClient.GetStream().BeginRead(bytes, 0, bytes.Length, ar => {

try {

int readed = acceptedClient.GetStream().EndRead(ar);

// Raise event about new data readed

DataReceiveEventArgs args = new DataReceiveEventArgs(bytes, readed);

DataReceiveEventHandler handler = OnDataReceive;

if (null != handler) {

handler.Invoke(this, args);

}

//

string s = Encoding.UTF8.GetString(bytes, 0, readed);

if (s.StartsWith("player:")) {

secondPlayerName = s.Split(new[] {

":"

}, StringSplitOptions.None)[1];

//

EventHandler eventHandler = OnPlayerNameReceived;

if (null != eventHandler) {

eventHandler.Invoke(this, EventArgs.Empty);

}

}

//

if (s.StartsWith("leave:")) {

acceptedClientThreadStopped = true;

}

Debug.Print("Readed {0} bytes : {1}", readed, Encoding.UTF8.GetString(bytes));

Logger.Log("GAME SERVER : Readed " + readed + " bytes : " +

Encoding.UTF8.GetString(bytes));

receivingStarted = false;

} catch (ObjectDisposedException) {

} catch (IOException) {

lock (locker) {

acceptedClientThreadStopped = true;

}

}

catch (InvalidOperationException)

{

// **TODO : Пофиксить. Если ливать из игры, иногда падает тут (исключение аналогично**

// **TODO : исключению, возникающему на клиенте в GameClient)**

}

}, null);

}

// Записываем все данные из очереди в поток tcp client'а

while (true) {

lock (locker) {

if (queue.Count == 0) {

break;

}

}

//

byte[] bytesToWrite;

lock (locker) {

bytesToWrite = queue[0];

queue.RemoveAt(0);

}

acceptedClient.Client.Send(bytesToWrite);

Logger.Log("GAME SERVER : Sent : " + Encoding.UTF8.GetString(bytesToWrite));

}

//

acceptedClientThreadWaitHandle.WaitOne(200);

lock (locker) {

if (acceptedClientThreadStopped && queue.Count == 0) {

break;

}

}

}

} catch (Exception) {

Debug.Print("An exception in acceptedClientThreadFunc. May be client lost the connection.");

} finally {

clientsAcceptedCount--;

acceptedClient.Close();

try {

EventHandler handler = this.OnDisconnect;

if (null != handler) {

handler.Invoke(this, EventArgs.Empty);

}

} catch (Exception) {

Debug.Print("Unexpected exception while handler of OnDisconnected been invoked.");

}

}

}

public void CancelGame() {

// Стопим таймер

if (timer != null) {

timer.Dispose();

timer = null;

}

tcpListener.Stop();

// Стопим поток, прослушивающий TCP

lock (locker) {

tcpListenerThreadStopped = true;

}

tcpListenerThreadWaitHandle.Set();

tcpListenerThread.Join();

// Стопим поток, обслуживающий клиента, если кто-то был подключен

if (null != acceptedClientThread) {

lock (locker) {

acceptedClientThreadStopped = true;

}

acceptedClientThreadWaitHandle.Set();

acceptedClientThread.Join();

}

//

firstPlayerName = null;

secondPlayerName = null;

EventHandler handler = OnPlayerNameReceived;

if (handler != null) {

handler.Invoke(this, EventArgs.Empty);

}

//

gameName = null;

//

Debug.Print("Game cancelled.");

}

private readonly List<byte[]> queue = new List<byte[]>();

public void sendData(byte[] data) {

lock (locker) {

queue.Add(data);

}

}

public event DataReceiveEventHandler OnDataReceive;

public event EventHandler OnDisconnect;

public event EventHandler OnConnect;

public void LeaveGame() {

sendData(Encoding.UTF8.GetBytes("leave:"));

//

CancelGame();

}

private const string UNKNOWN\_PLAYER\_NAME = "Неизвестный игрок";

private string firstPlayerName;

public string FirstPlayerName {

get {

if (string.IsNullOrEmpty(firstPlayerName))

{

return UNKNOWN\_PLAYER\_NAME;

}

return firstPlayerName;

}

}

private string secondPlayerName;

public string SecondPlayerName {

get {

if (string.IsNullOrEmpty(secondPlayerName))

{

return UNKNOWN\_PLAYER\_NAME;

}

return secondPlayerName;

}

}

public event EventHandler OnPlayerNameReceived;

public bool IsOwner {

get {

return true;

}

}

private string gameName;

public string GameName {

get {

return gameName;

}

}

}

}