|  |  |  |  |
| --- | --- | --- | --- |
|  | Politechnika Śląska  Wydział Inżynierii Materiałowej  i Metalurgii | | Paweł Wanot  Michał Stanecki  Michał Nabrdalik  Ryszard Janicki |
| Dokumentacja Projektu z:  Zawansowane programowanie w języku C#,  Aplikacje wieloplatformowe | | | |
| **Temat:**  **Projekt systemu składającego się z aplikacji klienckiej oraz serwera pozwalającego na wspólną grę w kółko i krzyżyk przez dwóch graczy przy użyciu biblioteki Winsock** | | | |
| Uwagi prowadzącego: | | | |
| Data przyjęcia: 25 styczeń 2017  Podpis prowadzącego: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Grupa: IPP-30PP  Sekcja: 1 | |
| Ocena: | |

Spis treści

[1. Opis Aplikacji 3](#_Toc474694368)

[1.1. Ogólny opis 3](#_Toc474694369)

[1.2. Klient 3](#_Toc474694370)

[1.3. Serwer 3](#_Toc474694371)

[2. Kod źródłowy aplikacji 3](#_Toc474694372)

[2.1. Klient 3](#_Toc474694373)

[a) Klasa „GetSocket” – odpowiedzialna za połączenie z serwerem 3](#_Toc474694374)

[b) Główny kod programu 4](#_Toc474694375)

[2.2. Serwer 19](#_Toc474694376)

[3. Graf UML 25](#_Toc474694377)

[3.1 Klient 25](#_Toc474694378)

[3.2. Serwer 26](#_Toc474694379)

[4. Screeny aplikacji 26](#_Toc474694380)

[4.1. Klient 26](#_Toc474694381)

[4.2. Serwer 30](#_Toc474694382)

# **Opis Aplikacji**

## **Ogólny opis**

**Stworzenie Systemu składającego się z dwóch aplikacji, klienckiej i serwerowej, która umożliwi grę w kółko i krzyżyk dwóm graczom przez Internet. Każdy gracz musi otrzymać powiadomienie o końcu rozgrywki to jest jedna z trzech opcji:**

* **Wygrana**
* **Przegrana**
* **Brak rozstrzygnięcia**

## **Klient**

**Aplikacja pozwalająca użytkownikowi na połączenie się z serwerem i jeśli pojawi się drugi gracz na serwerze rozpoczęcie rozgrywki. Grę rozpoczyna gracz zaznaczający na planszy „O” natomiast gracz „X” oczekuje na swoją turę. Każdy z graczy dostaje na początku rozgrywki czy gra jako „O” czy jako „X”. Aplikacja została na pisana w języku C# oraz wykorzystuje używanie drugiego wątka do odbierania danych by się nie „zawieszała” na czas oczekiwania na odebranie danych z serwera.**

## **Serwer**

**Aplikacja pośrednicząca w wymianie danych pomiędzy graczami. Jej głównym zadaniem jest połączenie graczy w pary a następnie utworzenie wątku na jakim toczy się gra. Serwer pośredniczy w wymianie danych pomiędzy graczami oraz czuwa nad tym jaki jest obecnie wynik oraz w razie potrzeby informuje o nim graczy. Aplikacja została napisana w języku C oraz wykorzystuje wielowątkowość do obsługo kolejnych gier oraz bibliotekę „Winsock” do komunikacji przez Internet.**

# **Kod źródłowy aplikacji**

## **Klient**

### **Klasa „GetSocket” – odpowiedzialna za połączenie z serwerem**

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

public class GetSocket

{

private Socket socket;

private string server;

private int port;

public void ConnectSocket(string server, int port)

{

Socket s = null;

this.server = server;

this.port = port;

IPEndPoint ipe = new IPEndPoint(IPAddress.Parse(server), port);

s = new Socket(ipe.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

s.Connect(IPAddress.Parse(server), port);

this.socket = s;

}

public void DisconnectSocket()

{

socket.Close();

}

public string SocketReceive()

{

Byte[] bytesReceived = new Byte[256];

int bytes = 0;

string page;

bytes = this.socket.Receive(bytesReceived, bytesReceived.Length, 0);

page = Encoding.ASCII.GetString(bytesReceived, 0, bytes);

return page;

}

public void SocketSend(string request)

{

Byte[] bytesSent = Encoding.ASCII.GetBytes(request);

this.socket.Send(bytesSent, bytesSent.Length, 0);

}

}

### **Główny kod programu**

**using System;**

**using System.Drawing;**

**using System.Threading;**

**using System.Windows.Forms;**

**namespace KlientCS**

**{**

**public partial class Form1 : Form**

**{**

**//obiekt klasy rezlizujący połączenie**

**public GetSocket client = new GetSocket();**

**//Znak używany przez gamecza O lub X**

**private string playerSign;**

**//Znak używany przez drugiego gamecza O lub X**

**private string enemySign;**

**//Kod otrzymany z serwera**

**private int code;**

**//Tablica przycisków do gry**

**private Button[] buttonList;**

**//Blokada klawiszy**

**private bool lockButtons = false;**

**//tablica opisująca kombinacje 1-gamecz 2-wróg**

**private int[] game = new int[9];**

**//Czy czekać na dane z serwera**

**private bool isWin = false;**

**public Form1()**

**{**

**InitializeComponent();**

**buttonList = new Button[] { bt0, bt1, bt2, bt3, bt4, bt5, bt5, bt6, bt7, bt8 };**

**panel1.Location = new Point(0, 0);**

**panel2.Location = new Point(0, 0);**

**this.CenterToScreen();**

**BeginNewGame();**

**}**

**private void DrawEndLine()**

**{**

**foreach (Button bt in buttonList)**

**{**

**int number = Convert.ToInt32(bt.Tag);**

**if (bt.Text == playerSign)**

**{**

**game[number] = 1;**

**}**

**if (bt.Text == enemySign)**

**{**

**game[number] = 2;**

**}**

**}**

**CheckAndDraw();**

**}**

**private void CheckAndDraw()**

**{**

**Point p;**

**//POZIOMO**

**if (game[0] == game[1] && game[1] == game[2])**

**{**

**if (game[0] == 1 || game[0] == 2)**

**{**

**p = buttonList[0].Location;**

**p.Y = -187;**

**pbHor.BeginInvoke(new MethodInvoker(() =>**

**{**

**pbHor.Location = p;**

**pbHor.Visible = true;**

**}));**

**}**

**}**

**if (game[3] == game[4] && game[4] == game[5])**

**{**

**if (game[3] == 1 || game[3] == 2)**

**{**

**p = buttonList[3].Location;**

**p.Y = p.Y - 275;**

**pbHor.BeginInvoke(new MethodInvoker(() =>**

**{**

**pbHor.Location = p;**

**pbHor.Visible = true;**

**}));**

**}**

**}**

**if (game[6] == game[7] && game[7] == game[8])**

**{**

**if (game[6] == 1 || game[6] == 2)**

**{**

**p = buttonList[6].Location;**

**p.Y = p.Y - 100;**

**p.X = p.X - 420;**

**pbHor.BeginInvoke(new MethodInvoker(() =>**

**{**

**pbHor.Location = p;**

**pbHor.Visible = true;**

**}));**

**}**

**}**

**//PIONOWO**

**if (game[0] == game[3] && game[3] == game[6])**

**{**

**if (game[0] == 1 || game[0] == 2)**

**{**

**p = buttonList[0].Location;**

**p.X = p.X + 50;**

**pbVert.BeginInvoke(new MethodInvoker(() =>**

**{**

**pbVert.Location = p;**

**pbVert.Visible = true;**

**}));**

**}**

**}**

**if (game[1] == game[4] && game[4] == game[7])**

**{**

**if (game[1] == 1 || game[1] == 2)**

**{**

**p = buttonList[1].Location;**

**p.X = p.X + 50;**

**pbVert.BeginInvoke(new MethodInvoker(() =>**

**{**

**pbVert.Location = p;**

**pbVert.Visible = true;**

**}));**

**}**

**}**

**if (game[2] == game[5] && game[5] == game[8])**

**{**

**if (game[2] == 1 || game[2] == 2)**

**{**

**p = buttonList[2].Location;**

**p.X = p.X + 50;**

**pbVert.BeginInvoke(new MethodInvoker(() =>**

**{**

**pbVert.Location = p;**

**pbVert.Visible = true;**

**}));**

**}**

**}**

**//NA UKOS**

**if (game[0] == game[4] && game[4] == game[8])**

**{**

**if (game[0] == 1 || game[0] == 2)**

**{**

**pbDiag1.BeginInvoke(new MethodInvoker(() =>**

**{**

**pbDiag1.Visible = true;**

**}));**

**}**

**}**

**if (game[2] == game[4] && game[4] == game[6])**

**{**

**if (game[2] == 1 || game[2] == 2)**

**{**

**pbDiag2.BeginInvoke(new MethodInvoker(() =>**

**{**

**pbDiag2.Visible = true;**

**}));**

**}**

**}**

**}**

**private void IsWin()**

**{**

**foreach (Button bt in buttonList)**

**{**

**int number = Convert.ToInt32(bt.Tag);**

**if (bt.Text == playerSign)**

**{**

**game[number] = 1;**

**}**

**if (bt.Text == enemySign)**

**{**

**game[number] = 2;**

**}**

**}**

**//POZIOMO**

**if (game[0] == game[1] && game[1] == game[2])**

**{**

**if (game[0] == 1 || game[0] == 2)**

**isWin = true;**

**}**

**if (game[3] == game[4] && game[4] == game[5])**

**{**

**if (game[3] == 1 || game[3] == 2)**

**isWin = true;**

**}**

**if (game[6] == game[7] && game[7] == game[8])**

**{**

**if (game[6] == 1 || game[6] == 2)**

**isWin = true;**

**}**

**//PIONOWO**

**if (game[0] == game[3] && game[3] == game[6])**

**{**

**if (game[0] == 1 || game[0] == 2)**

**isWin = true;**

**}**

**if (game[1] == game[4] && game[4] == game[7])**

**{**

**if (game[1] == 1 || game[1] == 2)**

**isWin = true;**

**}**

**if (game[2] == game[5] && game[5] == game[8])**

**{**

**if (game[2] == 1 || game[2] == 2)**

**isWin = true;**

**}**

**//NA UKOS**

**if (game[0] == game[4] && game[4] == game[8])**

**{**

**if (game[0] == 1 || game[0] == 2)**

**isWin = true;**

**}**

**if (game[2] == game[4] && game[4] == game[6])**

**{**

**if (game[2] == 1 || game[2] == 2)**

**isWin = true;**

**}**

**}**

**private void ReciveData()**

**{**

**if (ValidateData(client.SocketReceive()))**

**{**

**//Zaznacz odpowiedznij button**

**foreach (Button bt in buttonList)**

**{**

**if (((string)bt.Tag).Trim() == (code + "").Trim())**

**{**

**bt.BeginInvoke(new MethodInvoker(() =>**

**{**

**bt.Text = enemySign;**

**lockButtons = false;**

**}));**

**}**

**lbMove.BeginInvoke(new MethodInvoker(() =>**

**{**

**lbMove.Text = "Twoja tura - " + playerSign;**

**}));**

**}**

**}**

**//Koniec Gry**

**else**

**{**

**client.SocketSend("ack");**

**string message = "";**

**switch (code)**

**{**

**case 23: message = "Przegrałeś"; break;**

**case 24: message = "Wygrałeś"; break;**

**case 25: message = "Bez roztrzygnięcia"; break;**

**}**

**IsWin();**

**if (isWin)**

**{**

**DrawEndLine();**

**panel3.BeginInvoke(new MethodInvoker(() =>**

**{**

**panel3.Location = new Point(0, 0);**

**}));**

**panel3.BeginInvoke(new MethodInvoker(() =>**

**{**

**panel3.Visible = true;**

**}));**

**lbData.BeginInvoke(new MethodInvoker(() =>**

**{**

**lbData.Text = message + " Nowa game?";**

**}));**

**}**

**else**

**{**

**if (playerSign.Equals("X"))**

**{**

**if (code == 23 || code == 25)**

**{**

**//Odbierz Ostatni**

**code = Int32.Parse(client.SocketReceive());**

**client.SocketSend("ack");**

**//Zaznacz odpowiedznij button**

**foreach (Button bt in buttonList)**

**{**

**if ((string)bt.Tag == (code + "").Trim())**

**{**

**bt.BeginInvoke(new MethodInvoker(() =>**

**{**

**bt.Text = enemySign;**

**DrawEndLine();**

**}));**

**lockButtons = true;**

**}**

**//client.DisconnectSocket();**

**}**

**}**

**}**

**if (playerSign.Equals("O"))**

**{**

**if (code == 23 || code == 25)**

**{**

**//Odbierz Ostatni**

**if (code == 23)**

**code = Int32.Parse(client.SocketReceive());**

**client.SocketSend("ack");**

**//Zaznacz odpowiedznij button**

**foreach (Button bt in buttonList)**

**{**

**if ((string)bt.Tag == (code + "").Trim())**

**{**

**bt.BeginInvoke(new MethodInvoker(() =>**

**{**

**bt.Text = enemySign;**

**DrawEndLine();**

**}));**

**lockButtons = true;**

**}**

**//client.DisconnectSocket();**

**}**

**}**

**}**

**panel3.BeginInvoke(new MethodInvoker(() =>**

**{**

**panel3.Location = new Point(0, 0);**

**}));**

**panel3.BeginInvoke(new MethodInvoker(() =>**

**{**

**panel3.Visible = true;**

**}));**

**lbData.BeginInvoke(new MethodInvoker(() =>**

**{**

**lbData.Text = message + " Nowa game?";**

**}));**

**}**

**}**

**}**

**private void bt\_Click(object sender, EventArgs e)**

**{**

**if (!lockButtons)**

**{**

**if (((Button)sender).Text != "O" && ((Button)sender).Text != "X")**

**{**

**//Zaznacz pole znakiem**

**((Button)sender).Text = playerSign;**

**//Wyslij numer**

**client.SocketSend((string)((Button)sender).Tag);**

**lockButtons = true;**

**//Czekaj na odbior kodu gamecza 2**

**Thread oThread = new Thread(new ThreadStart(ReciveData));**

**// Start the thread**

**lbMove.Text = "Tura przeciwnika - " + enemySign;**

**oThread.Start();**

**oThread.Join();**

**}**

**}**

**}**

**private void BeginNewGame()**

**{**

**//Zeruj dane gry**

**playerSign = "";**

**enemySign = "";**

**code = 0;**

**lockButtons = true;**

**pbDiag1.Visible = false;**

**pbDiag2.Visible = false;**

**pbHor.Visible = false;**

**pbVert.Visible = false;**

**isWin = false;**

**for (int i = 0; i < game.Length; i++)**

**{**

**game[i] = 0;**

**}**

**foreach (Button bt in buttonList)**

**{**

**bt.Text = "";**

**}**

**//Pokaż odpowiednie panele**

**panel1.Visible = true;**

**this.Width = panel1.Width + 10;**

**this.Height = panel1.Height + 35;**

**panel2.Visible = false;**

**panel3.Visible = false;**

**}**

**private bool ValidateData(string v)**

**{**

**//sprawdź kod**

**//21-O,22-X,23-przegamena,24-wygamena,25-remis,26-game**

**code = Int32.Parse(v);**

**if (code > 22 && code < 26)**

**{**

**//Koniec gry**

**return false;**

**}**

**//Graj dalej**

**return true;**

**}**

**private void btConnect\_Click(object sender, EventArgs e)**

**{**

**//Pobierz adres IP**

**string ip = textBox1.Text;**

**//Pobierz numer portu serwera**

**Int32 port = Convert.ToInt32(textBox2.Text);**

**//Połącz z serwerem**

**client.ConnectSocket(ip, port);**

**//Odbierz numer gamecza**

**string player = client.SocketReceive();**

**//Wyslij potwierdzenie otrzymania danych**

**client.SocketSend("ack");**

**//Potwierdź znak gamecza**

**SetPlayerSign(player);**

**if (playerSign == "X") lbMove.Text = "Tura przeciwnika";**

**if (playerSign == "O") lbMove.Text = "Twoja tura";**

**//Start Gry**

**panel1.Visible = false;**

**panel2.Visible = true;**

**this.Width = panel2.Width;**

**this.Height = panel2.Height;**

**lockButtons = true;**

**panel3.Location = new Point(this.Height % 2, this.Width % 2);**

**Thread oThread = new Thread(new ThreadStart(this.GameBegins));**

**oThread.Start();**

**oThread.Join();**

**}**

**private void GameBegins()**

**{**

**//Początek rozgrywki**

**code = Int32.Parse(client.SocketReceive());**

**//Gracz 2**

**if (code != 26)**

**{**

**foreach (Button bt in buttonList)**

**{**

**if ((string)bt.Tag == (code + "").Trim())**

**{**

**bt.BeginInvoke(new MethodInvoker(() =>**

**{**

**bt.Text = enemySign;**

**lockButtons = false;**

**}));**

**}**

**}**

**}**

**else**

**{**

**lockButtons = false;**

**}**

**//Gracz 1**

**}**

**private void SetPlayerSign(string player)**

**{**

**//Ustaw znak jakim posługuje się gamecz**

**if (player == "21")**

**{**

**playerSign = "O";**

**enemySign = "X";**

**}**

**if (player == "22")**

**{**

**playerSign = "X";**

**enemySign = "O";**

**}**

**}**

**private void btYes\_Click(object sender, EventArgs e)**

**{**

**//Nowa game**

**BeginNewGame();**

**}**

**private void btNo\_Click(object sender, EventArgs e)**

**{**

**//Koniec gry**

**Application.Exit();**

**}**

**}**

**}**

## **Serwer**

#pragma warning(disable:4244)

#pragma warning(disable:4996 4005)

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_USE\_MATH\_DEFINES

#ifdef UNICODE

#undef UNICODE

#define \_MBCS

#endif

#include <string>

#include <vector>

#include <cstdio>

#include <cstdlib>

#include <cstring>

using namespace std;

#include <WinSock2.h>

#include <stdio.h>

#include <iostream>

#pragma comment(lib,"ws2\_32.lib")

#define LINE\_MAX 1000

const u\_short iPort = 500;

//KLASA Zajmujaca się przekazaniem połączenia

class Game

{

public:

Game(SOCKET pol1, SOCKET pol2);

~Game();

SOCKET GetPolaczenie(int x);

private:

SOCKET polaczenie1, polaczenie2;

};

Game::Game(SOCKET pol1, SOCKET pol2)

{

this->polaczenie1 = pol1;

this->polaczenie2 = pol2;

}

SOCKET Game::GetPolaczenie(int x) {

if (x == 1) return this->polaczenie1;

if (x == 2) return this->polaczenie2;

}

Game::~Game()

{

}

int CheckWin(int x, int gra[9]) {

bool remis;

//POZIOMO

if (gra[0] == x&& gra[1] == x&&gra[2] == x)

{

cout << "WYGRANA" << endl;

return x;

}

if (gra[3] == x&& gra[4] == x&& gra[5] == x)

{

cout << "WYGRANA" << endl;

return x;

}

if (gra[6] == x&&gra[7] == x&& gra[8] == x)

{

cout << "WYGRANA" << endl;

return x;

}

//PIONOWO

if (gra[0] == x&& gra[3] == x&&gra[6] == x)

{

cout << "WYGRANA" << endl;

return x;

}

if (gra[1] == x&&gra[4] == x&& gra[7] == x)

{

cout << "WYGRANA" << endl;

return x;

}

if (gra[2] == x&& gra[5] == x&& gra[8] == x)

{

cout << "WYGRANA" << endl;

return x;

}

//NA UKOS

if (gra[0] == x&&gra[4] == x&& gra[8] == x)

{

cout << "WYGRANA" << endl;

return x;

}

if (gra[2] == x&& gra[4] == x&& gra[6] == x)

{

cout << "WYGRANA" << endl;

return x;

}

remis = true;

//SPRAWDŹ CZY JEST REMIS

for (int i = 0; i < 9; i++) {

if (gra[i] == 0)

remis = false;

}

if (remis)

{

return 25;

}

//BRAK WYGRANEJ

return 0;

}

//WYSLIJ DANE

int SendData(int x, SOCKET gracz) {

//char \*data = (char\*)&x;

//int zwrot, wszystko = strlen(data), biezacy = 0;

cout << "DO WYSLANIA (INT):" << x << endl;

int zwrot, biezacy = 0, wszystko;

char data[20];

/\*memcpy(&x, data, sizeof(int));

wszystko = strlen(data);\*/

itoa(x, data, 10);

wszystko = strlen(data);

int const nBufor = 10;

char bufor[nBufor] = { '\0' };

//cout << "WYSALANO (CHAR[]): " << data << endl;

//WYSYŁAMY KOD

//while (wszystko > 0) {

zwrot = send(gracz, data, wszystko, 0);

//if (zwrot == 0) break;

/\*else\*/ if (zwrot <= SOCKET\_ERROR) abort;

/\*wszystko -= zwrot;

biezacy += zwrot;\*/

//}

//ODBIERAMY POTWIERDZENIE LUB KOD

zwrot = recv(gracz, bufor, nBufor, 0);

if (zwrot <= SOCKET\_ERROR)abort();

bufor[zwrot] = '\0';

if (bufor == "ack") {

return 30;

//POTWIERDZENIE ŻE JEST GRACZEM NR x

}

else {

int liczba = atoi(bufor);

//cout << "ODEBRANO: " << bufor << endl;

return liczba;

//ZWROT POLA ZAZNACZONEGO

}

}

//Powiadom o końcu gry graczy

void NotifyEnd(int wygral, SOCKET gracz1, SOCKET gracz2) {

if (wygral == 21) {

SendData(24, gracz1);

SendData(23, gracz2);

}

if (wygral == 22)

{

SendData(23, gracz1);

SendData(24, gracz2);

}

if (wygral == 25)

{

SendData(25, gracz1);

SendData(25, gracz2);

}

}

DWORD WINAPI KolkoKrzyzk(LPVOID parametr) {

int kody[6] = { 21,22,23,24,25,26 };//21-O,22-X,23-przegrana,24-wygrana,25-remis,26-gra

Game obecna = \*(Game\*)(parametr);

SOCKET gracz1 = obecna.GetPolaczenie(1), gracz2 = obecna.GetPolaczenie(2);

int gra[9] = { 0,0,0,0,0,0,0,0,0 };

int koniec;

SendData(22, gracz2);

SendData(21, gracz1);

int liczba = SendData(kody[5], gracz1);

//Co zanacza gracz1

gra[liczba] = 21;

while (true) {

//Prześlij do gracza 2 zaznaczony numer gracza 1

//Co zanacza gracz2

liczba = SendData(liczba, gracz2);

gra[liczba] = 22;

//Sprawdź czy jest wygrana

koniec = CheckWin(22, gra);

if (koniec != 0) {

//Powiadom graczy o końcu gry

NotifyEnd(koniec, gracz1, gracz2);

SendData(liczba, gracz1);

//Koniec gry

break;

}

//Prześlij do gracza 1 zaznaczony numer gracza 2

//Co zanacza gracz1

liczba = SendData(liczba, gracz1);

gra[liczba] = 21;

//Sprawdź czy jest wygrana

koniec = CheckWin(21, gra);

if (koniec != 0) {

//Powiadom graczy o końcu gry

NotifyEnd(koniec, gracz1, gracz2);

liczba = SendData(liczba, gracz2);

//Koniec gry

break;

}

}

return 0;

}

int main() {

printf(">> Poczatek programu <<\n\n\n");

WSADATA info;

SOCKET nasluch, polaczenie1, polaczenie2;

sockaddr\_in lokalny, zdalny1, zdalny2; // struktura adres ip + port

int rozmiar1, rozmiar2;

vector<HANDLE> watki;

if (WSAStartup(MAKEWORD(2, 2), &info) != 00) abort();

nasluch = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_IP);

if (nasluch == SOCKET\_ERROR) abort();

lokalny.sin\_family = AF\_INET;

lokalny.sin\_addr.s\_addr = INADDR\_ANY;

lokalny.sin\_port = htons(iPort);

if (bind(nasluch, (sockaddr \*)&lokalny, sizeof lokalny) != 0) abort;

while (true) {

printf("Oczekiwanie na polaczenie \n");

listen(nasluch, SOMAXCONN);

//CZEKAJ NA GRACZ 1

rozmiar1 = sizeof zdalny1;

polaczenie1 = accept(nasluch, (sockaddr \*)&zdalny1, &rozmiar1);

if (polaczenie1 == INVALID\_SOCKET) abort();

printf("Zaakceptowane polaczenie o adresie: %s i porcie: %d - GRACZ 1\n\n", inet\_ntoa(zdalny1.sin\_addr), ntohs(zdalny1.sin\_port));

//Poczekaj na drugiego gracza

Sleep(100);

//CZEKAJ NA GRACZ 2

rozmiar2 = sizeof zdalny2;

polaczenie2 = accept(nasluch, (sockaddr \*)&zdalny2, &rozmiar2);

if (polaczenie2 == INVALID\_SOCKET) abort();

printf("Zaakceptowane polaczenie o adresie: %s i porcie: %d - GRACZ 2\n\n", inet\_ntoa(zdalny2.sin\_addr), ntohs(zdalny2.sin\_port));

////////////////////////////////////////////--KOD DLA TWORZENIA WATKA--\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

//int biezacy = 1;

HANDLE watek; // uchwyt watka

//DWORD dane =

Game nowa = Game(polaczenie1, polaczenie2); // dane wejsciowe dla funkcji watka

DWORD ident; // identyfikator liczbowy watka

// utworzenie watka roboczego w procesie programu:

watek = CreateThread(NULL, // wskaznik do rekordu z atrybutami opisujacymi bezpieczenstwo watka

0, // rozmiar prywatnego stosu watka lub zero dla rozmiaru domyslnego

KolkoKrzyzk, // procedura (funkcja) wykonywana w ramach watka

//(&polaczenie1,&polaczenie2), // wskaznik do wejsciowych danych przekazywanych procedurze lub NULL, gdy nieuzywane

&nowa,

CREATE\_SUSPENDED, // flagi stanu poczatkowego watka, np. CREATE\_SUSPENDED daje watek zawieszony, zero daje watek od razu wykonywany

&ident // identyfikator nowego watka przekazany przez system lub NULL, gdy nieuzywane

);

///////////////////////////////////////////////--------------------------\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

watki.push\_back(watek);

ResumeThread(watek); // dopiero od tego punktu watek bedzie wykonywany

Sleep(100);

}

shutdown(nasluch, SD\_BOTH);

closesocket(nasluch);

if (WSACleanup() != 0) abort;

printf("\n\nKoniec programu.\n");

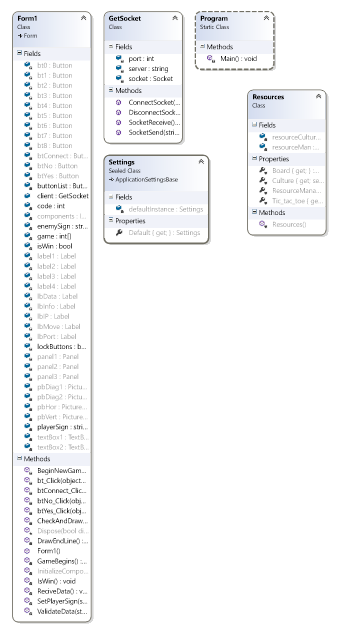
cin.ignore(2);

return 0;

}

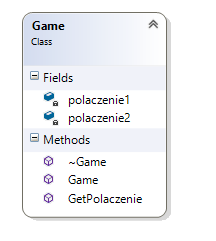
# **Graf UML**

## Klient



Schemat Schemat UML klas aplikacji klienta

## Serwer



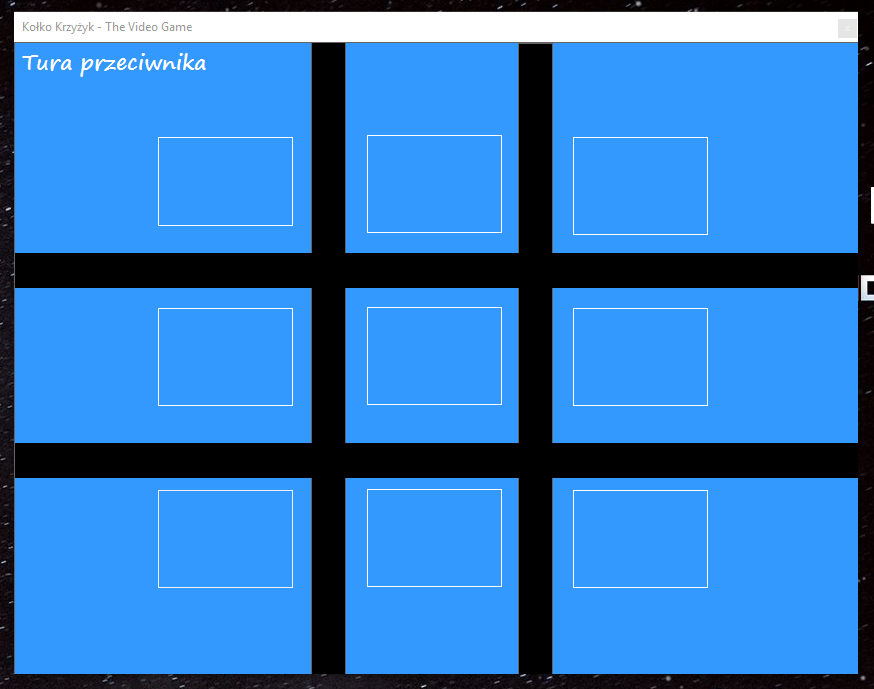
Schemat Schemat klas serwera

# **Screeny aplikacji**

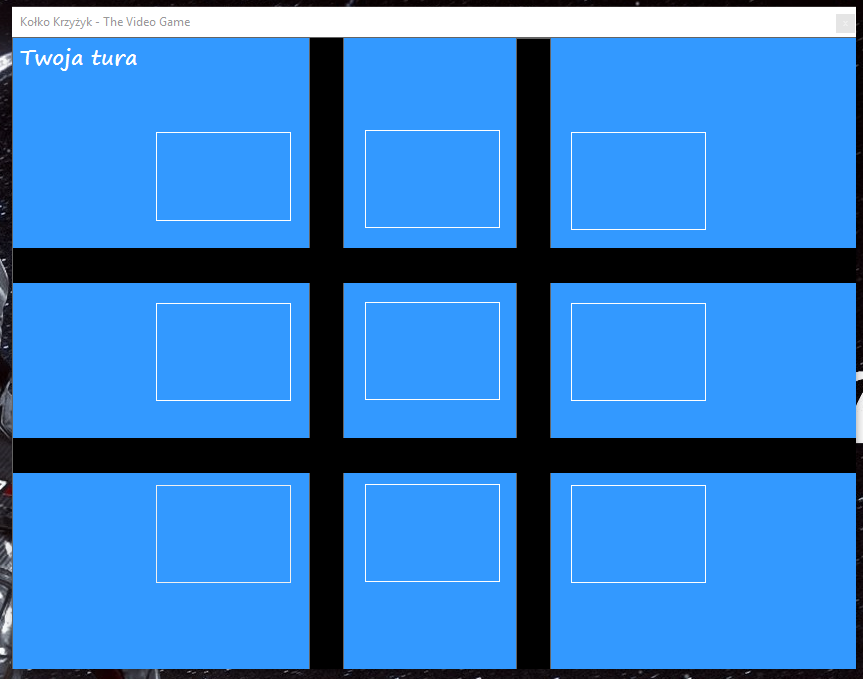
## Klient



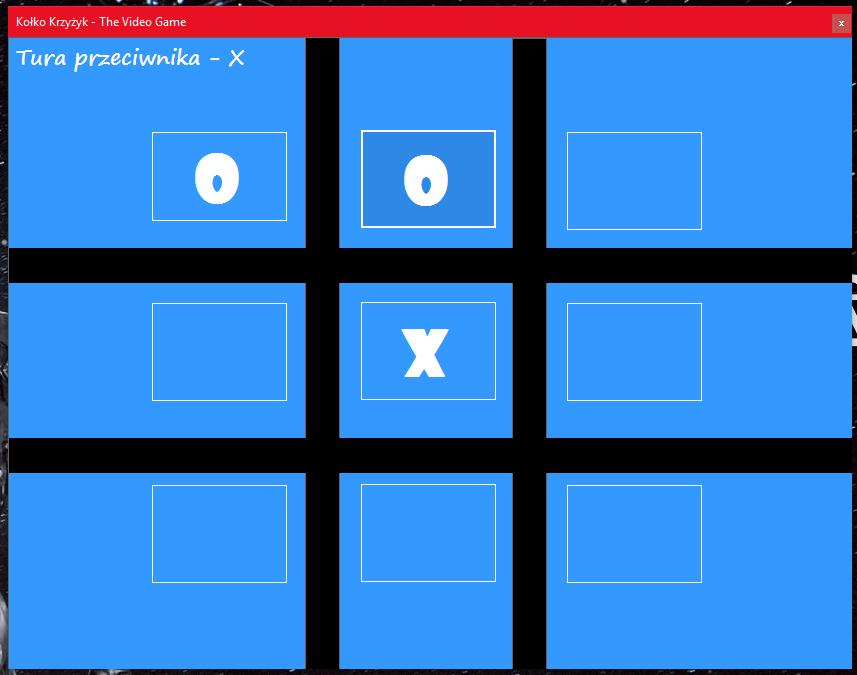
Obraz Ekran logowanie klienta



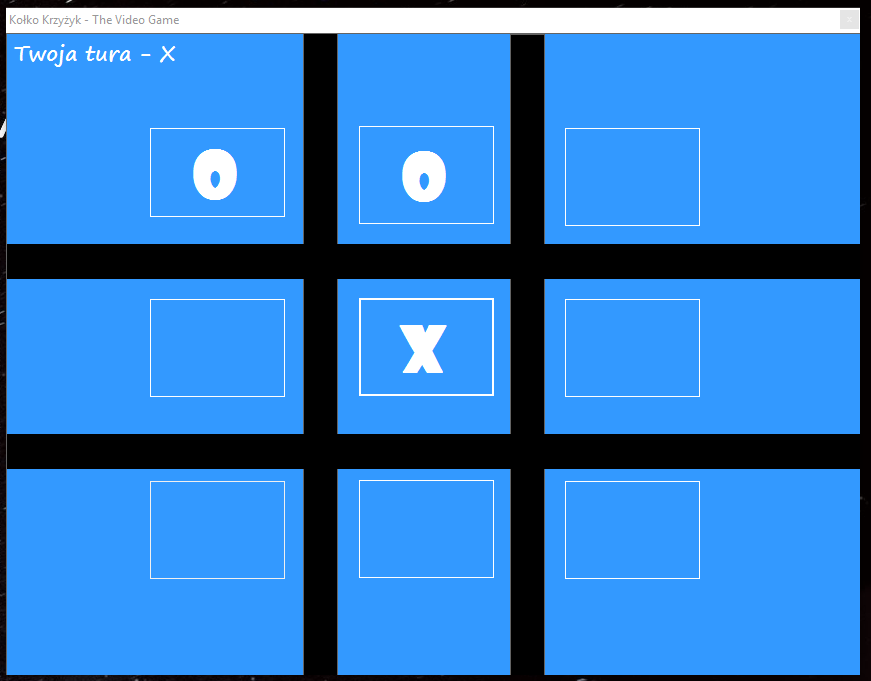
Obraz Ekran oczekiwania na rozpoczęcie gry



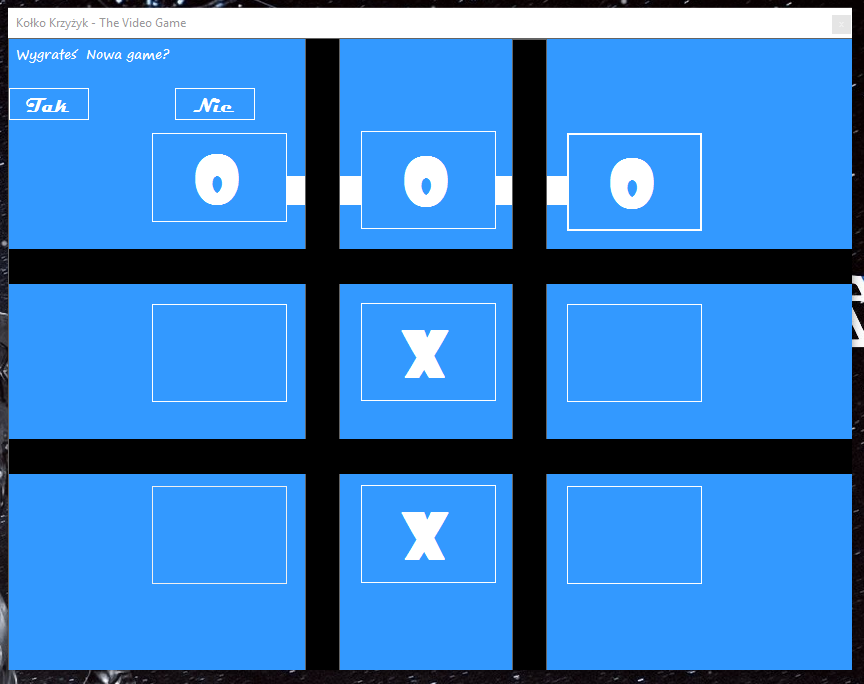
Obraz Ekran rozpoczęcia rozgrywki



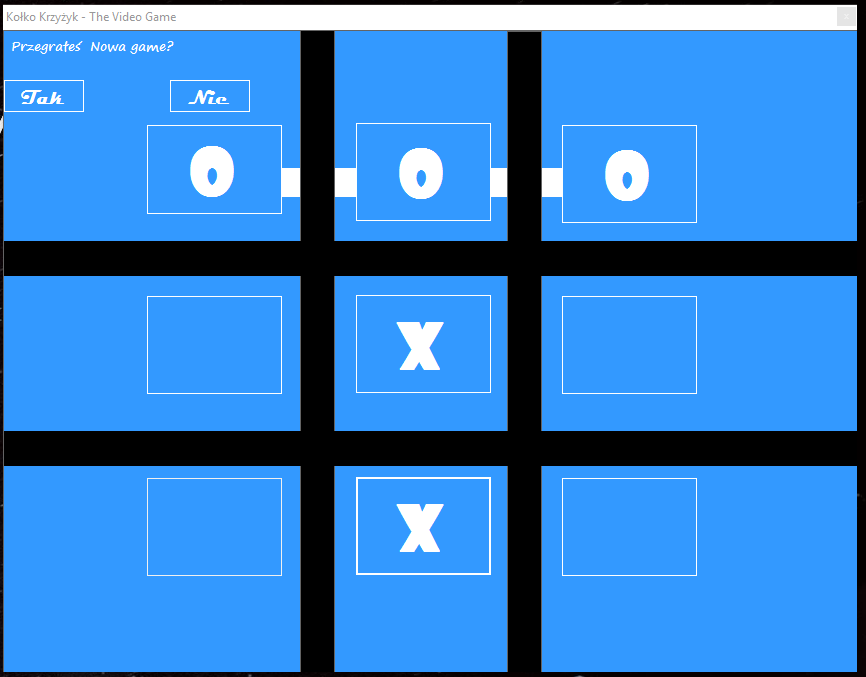
Obraz Ekran rozgrywki gracza O



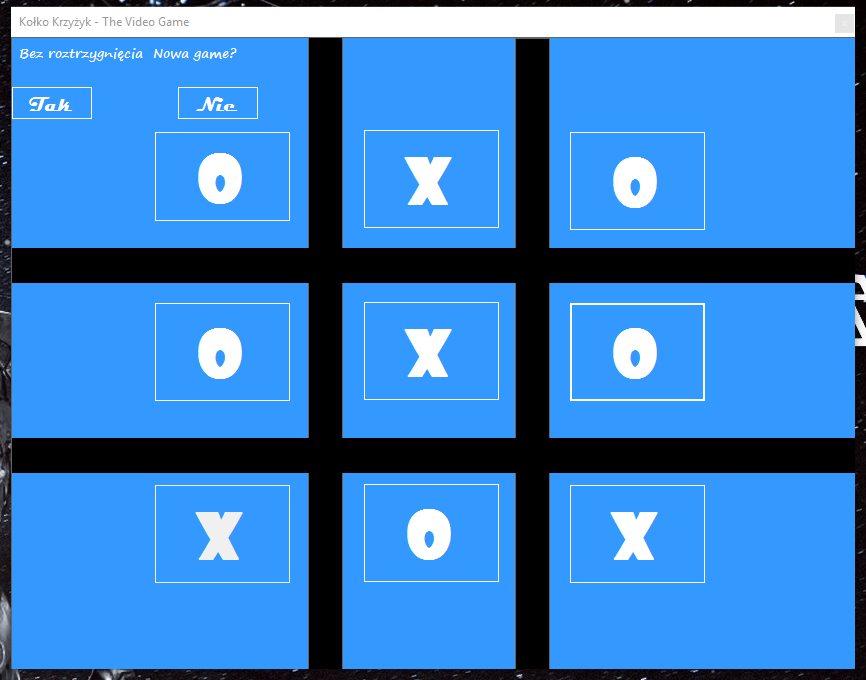
Obraz Ekran rozgrywki gracza X



Obraz Ekran końca rozgrywki gracza wygrywającego

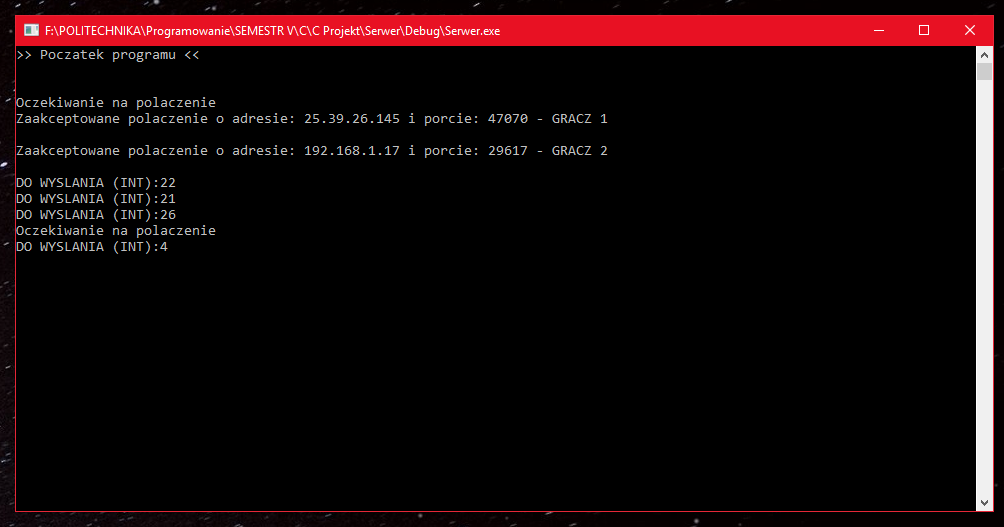


Obraz Ekran końca rozgrywki gracza przegrywającego



Obraz Ekran końca rozgrywki bez rozstrzygnięcia

## Serwer



Obraz Ekran podglądy danych odbieranych i wysyłanych przez serwer