"Peer-to-peer" aplikacija za crtanje

Vukić, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni prijediplomski studij Računarstvo

Peer-to-peer aplikacija za crtanje

Završni rad

Luka Vukić

Osijek, 2024.

SADRŽAJ

1.UVOD

U ovom završnom radu ću prikazati rad aplikacije za „peer-to-peer“ crtanje. Prilikom pokretanja aplikacije jedan korisnik započinje konekciju s drugim korisnikom. Nakon uspješnog stvaranja kanala između dva korisnika oni će moći razmjenjivati poruke preko mreže. Glavni cilj razmjenjivanja poruka će biti reprodukcija nacrtanog sadržaja jednog korisnika na zaslon za crtanje drugog korisnika i obrnuto. Taj prijenos poruka će se odvijati u stvarnom vremenu što će za učinak imati da oba korisnika crtaju na istoj ploči. Osim same implementacije programa ovaj rad će također proći osnovne principe mrežne komunikacije, korištenog protokola te potencijalne probleme koji mogu nastati i način njihovog rješavanja. U drugom poglavlju će se proći aplikacije iste ili slične svrhe te će ih se usporediti. U trećem poglavlju će se kratko opisati korištene tehnologije kao što su programski jezik *C#*, grafička biblioteka *Windows forms.NET* i sistemska biblioteka za mrežnu komunikaciju *System.Net.Sockets*. Osim toga navest će se neke njihove prednosti i mane. Četvrto poglavlje će objasniti naše tehničko rješenje. Objasnit će teoretsku pozadinu rada aplikacije. Pokazat će se kako smo koristili navedene tehnologije kod implementacije programa te kako one rade. U petom poglavlju će se opisati korištenje aplikacije

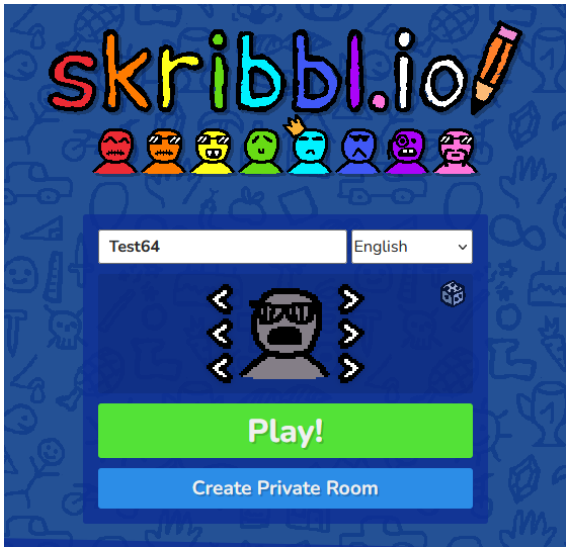
1.1 Zadatak završnog rada

Zadatak završnog rada je pružiti mogućnost peer to peer komunikacije dvaju korisnika. Nakon uspostave komunikacijskog kanala između dva korisnika sve što jedan korisnik nacrta na svom računalu bit će reproducirano drugom korisniku i obrnuto. Zbog toga što će se reproducirati na istu površinu za crtanje dobit će se dojam da oba korisnika crtaju na istoj ploči za crtanje.

2. SLIČNE APLIKACIJE

Postoje brojne aplikacije koje su namijenjene za crtanje i dijeljenje zaslona u isto vrijeme te koje omogućuju da više ljudi crta na njima. One se uglavnom baziraju korištenjem centralnog poslužitelje kojem se pristupa preko mrežnog preglednika. Moje rješenje će za razliku od tih već postojećih i dobro poznatih rješenje koristit izravnu konekciju između klijenta a ne posrednu preko poslužitelje. To znači da će u mojem slučaju aplikacija će sve podatke preko mreže slati izravno drugom klijentu a ne poslužitelju koji će tek onda poslati te podatke drugom klijentu. Također će moja aplikacija biti samostojeća i neće koristiti mrežni preglednik za prikaz grafike. Jedine primjere poznatih aplikacija koje sam pronašao da koriste direktnu konekciju između klijenta su *Skype* i *WhatsApp*. Problem kod tih primjera je taj što su one primarno aplikacije za razmjenu poruka i poziva. One nemaju mogućnost dijeljenja zaslona na kojem se crta ali imaju mogućnost dijeljenja zaslona u sklopu video poziva. Iz tog razloga sam detaljnije odlučio prikazati primjere koje koriste centraliziranu arhitekturu komunikacije ali su u pogledu funkcionalnosti sličniji mojoj aplikaciji.

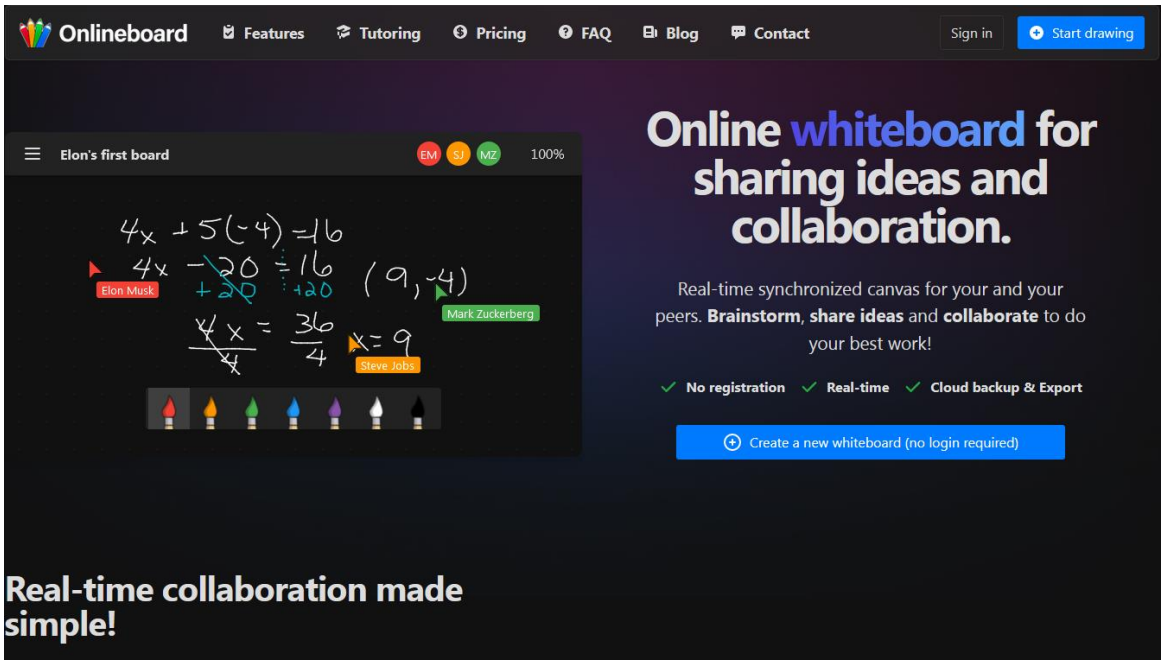
Prvi primjer slične aplikacije je igra *skribbl.io* u kojoj jedna osoba crta a druge osobe pogađaju što osoba crta. Kao što sam već rekao ona se nalazi u obliku mrežne stranice te se njoj pristupa korištenjem mrežnog preglednika i spajanjem na poslužitelja. Crtež se reproducira drugim osobama u stvarno vrijeme to samo jedna osoba može crtati u isto vrijeme. Osoba koja želi igrati ovu igru ima dvije opcije. Prva je da se pridruži već postojećoj sobi za igru ili da kreira svoju privatnu sobu na koju se druge osobe mogu priključiti. Izgled ove stranice možemo vidjeti na slikama 2.1 i 2.2.



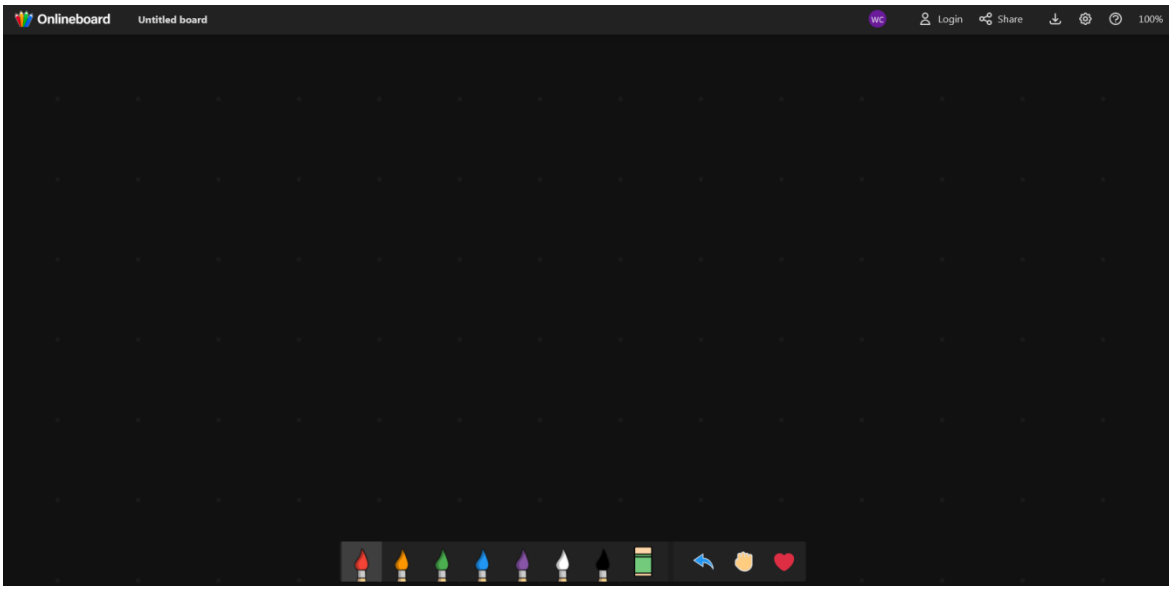
Slika 2.1. *Poćetni zaslon skribbl.io web aplikacije*

Slika 2.2. *Primjer pogađanja crteža unutar spojene sobe*

Drugi primjer je stranica za surađivanje koja omogućuje crtanje i pisanje u stvarnom vremenu jednom ili više korisnika *onlineboard.eu*. Ona također ima koncept privatne sobe u koju se korisnik može priključiti i gledati što osoba crta. U ovom primjer može više osoba crtati u isto vrijeme.



Slika 2.3. *Primjer početnog zaslona stranice*



Slika 2.4. *Primjer dijeljenog zaslona za crtanje i surađivanje*

3. KORIŠTENE TEHNOLOGIJE

Postoje brojne različite tehnologije koje su se mogle koristit za postizanje istog rješenja kao što je moje. Neke od njih su mogle biti različiti programski jezici kao što su: *C++*, *Java*, *Javascript*, *Python*. Različite verzije grafičkih biblioteka kao što su: *WPF*, *Blazor*, *ImGui.NET*. Razlog zašto sam odabrao tehnologije koje ću koristiti je primarno zbog dobrog poznavanja tih tehnologija. Uz to pružaju opširnu *online* dokumentaciju i jednostavno korištenje. Mana odabranih tehnologija je što su one primarno napravljene za korištenje *Windows* operativnom sustav što znači da moja aplikacija neće raditi na *Linux* i *macOS* sustavima.

3.1 C#

C# je visoko-programski jezik s generalnom namjenom razvijen od strane *Microsofta* u 2000. *C#* je najpopularniji jezik koji je baziran na *.NET* platformi[1]. Potpuno je besplatan, otvorenog je koda te se može pokretati na brojnim platformama bilo da se radi o aplikacijama za mobitel, računalo ili server. Zbog svoje popularnosti njegov ekosustav je široko rasprostranjen i pruža gotova rješenja za brojne probleme. Ovo daleko skraćuje potrebno vrijeme za razvoj aplikacija i održavanje istih. Utemeljen je na objektno orijentiranim principima što znači da je srž njegovog programiranja objektno orijentirana(eng: *object oriented programming*). Uz objektno orijentirani način programiranja pruža i ostale mogućnosti poput funkcionalnog programiranja i nisko razinskih mogućnosti koje podržavaju visoko učinkovite scenarije bez da se piše nesiguran kod. On se sam brine o zauzimanju i oslobađanju memorije što samim tim kod čini daleko sigurnijim od mogućih sigurnosnih propusta. Njegova mana u odnosu na programe kao što su *C* i *C++* je da sporiji. Prvo zato što nije direktno kompajliran u binarni kod kao oni. Drugo radi korištenja sakupljanje smeća (eng: *garbage collection*) koja pruža automatski alociranje memorije koje nije učinkovito kao ručno alociranje.

3.2 Windows Forms .NET

Windows forms je okvir korisničkog sučelje koje se koristi za kreiranje aplikacija za *windows* operativni sistem. One pružaju platformu koja podržava bogati skup mogućnosti za razvoj aplikacija kao što su kontrole, grafika, korisnički unos. Uz to također pružaju „*drag-and-drop*“ vizualni dizajner unutar *Visual Studio* programa koji pomaže za razvijanje i pisanje koda što omogućuje lagano kreiranje *windows forms* aplikacija. One su otvorenog koda koji je svima javno dostupan na *githubu* te se pokreće na različitim verzijama *.NET* okvira i *Visual Studio* programa.[2]. Uz to pružaju skup biblioteka koje olakšavaju učestale poslove kao što je čitanje i

pisanje u datotečni sistem, pružanje odgovora na učestale korisnikove radnje kao što su pritisak tipki računalnog miša ili tipkovnice.

Postoje mnogobrojna druga rješenja koja pružaju mogućnosti *windows* formi kao što su *Qt*, *WPF*, *UWP*, itd... Glavni razlog zašto sam izabrao *windows* forme u odnosu na spomenuta rješenja je već prijašnji rad s ovim okvirom te stabilnost samog okvira. *Windows* forme postoje već desetljećima te su prošle brojne cikluse testiranje i primjene u brojnim aplikacijama različitih mogućnosti i namjena. Jedna njihova mana u odnosu na druge okvire je ta što ne posjeduju neke nove mogućnosti tih okvira koje olakšavaju izvršavanje određenih zadataka te pojednostavljuju i ubrzavaju proces razvijanja programa. Uz to naizgled ima zastarjeli dizajn u odnosu na novije okvire.

3.3 System.Net.Sockets

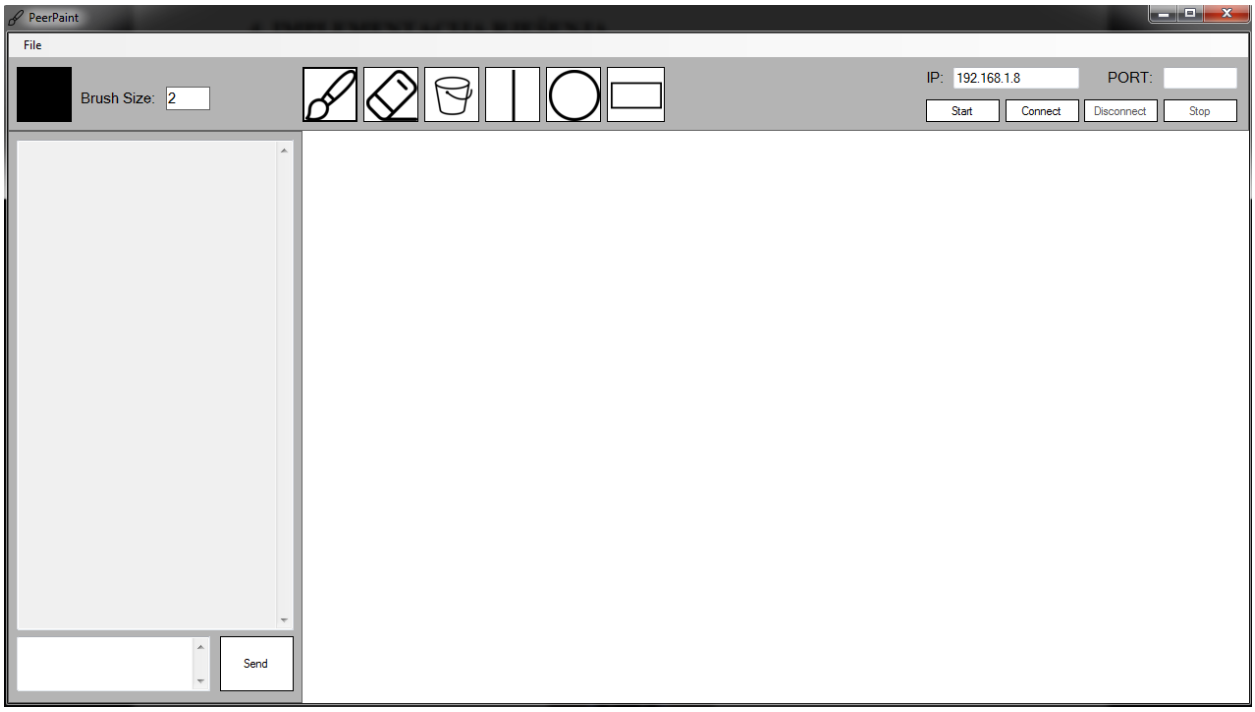
System.Net.Sockets je imenski prostor koji pruža implementaciju *windows socket*(*Winsock*) sučelja za kontrolu pristupa i komuniciranje preko mreže[3]. Za to koristi raznovrsne funkcije i strukture podataka kako bi to postigao. Podržava IPv4 i IPv6 verzije IP(eng: *Internet Protocol*) protokola, sinkrone i asinkrone *sockete*, TCP(eng: *Transmission Control Protocol*) i UDP (eng: *User Datagram Protocol*) protokole za prijenos podataka.

4. TEHNIČKO RJEŠENJE

4.1 Pregled grafičkog rješenja

Rješenje ovog zadatka možemo odvojiti na dvije logičke jedinice. Jedna će biti zadužena za prikazivanje grafike i obrađivanje korisnikovih radnji sa sučeljem. Prilikom toga će omogućiti korisniku da crta po podlozi namijenjenoj za crtanje. Druga logička jedinica će biti zadužena za umrežavanje s drugim korisnikom. Ona će izvršavati korisnikove naredbe bilo prekidanje ili uspostavljanje konekcije. U slučaju da je konekcija ostvarena, cijelo vrijeme će slati korisnikove radnje drugom korisniku i čitati radnje drugog korisnika.

Grafičko sučelje je sačinjeno od dugmadi kojima će korisnik odabirati boju kista, način crtanja, uspostavljanje ili prekidanje konekcije, slušanje za nadolazeće konekcije ili prekidanje slušanja, slanje poruka, spremanje crteža, čišćenje radne plohe te prekidanje same aplikacije. Uz to ima i četiri polja za unos vrijednosti koje su zadužene za podešavanje veličine kista, IP adresu uređaja, port uređaja te unošenje poruka. Postoji jedno posebno polje koje će služiti za prikaz primljenih i poslanih poruka. Uz sve ove elemente postoji i ploha za crtanje. Crtež na plohi možemo spremiti u bilo kojem trenutku odlaskom na *file->save* u formatu .png (eng*: Portable Network Graphics*). Razlog zašto sam odabrao ovaj format je njegova pristupačnost gdje ga gotovo svi preglednici slika razumiju te ne gube nikakvi podatke kod kompresije[4].



Slika 4.1. *prikaz sučelja aplikacije*

Prije nego što možemo krenuti s crtanjem prvo se moraju inicijalizirane neke komponente. Pokretanjem aplikacije odmah se pokreće grafičko sučelje od strane *windows* formi te se definira širina i visina sučelja po kojem ćemo crtati zajedno s našim suradnikom. Za Crtanje koristimo mapu bitova (eng: *bitmap*) iz koje ćemo stvoriti grafiku po kojoj crtamo.

Zatim se postavlja zaglađivajući način rada (eng: *SmoothingMode.AntiAlias*) te se početak i kraj olovke zaokružuje (eng: *LineCap.Round*) kako bi se dobile kontinuirane crte bez prekida. Zatim dohvaćamo i prikazujemo IP adrese našeg uređaja. Ova informacija olakšava korisniku pokretanje poslužitelja na svom računalo na koje se onda druga osoba može spojiti. Uz to isključimo dugme za zaustavljanje servera i odjavljivanje jer korisnik u ovom trenutku nije ni spojen s drugom osobom. Na kraju inicijalizacije omogućujemo prekidanje pozadinskih radnika (eng: *backgroundWorker*).

Pozadinski radnici nam pružaju višenitno (eng: *multithreaded*) okruženje koje je nužno kako bi se ovaj program mogao ispravno izvršavati. U protivnom da ne koristimo višenitnost naš program bi se zablokirao kada bi slušao za nadolazeće konekcije ili poruke jer niti jedan nit ne bi izvršavala grafičko sučelje. Ako bi naša jedna nit izvršavala sučelje onda nitko ne bi slušao za nadolazeće konekcije i poruke te u tom slučaju ne bismo mogli komunicirati s drugim korisnikom.

```
InitializeComponent();
this.KeyPreview = true;
this.Width = 1280;
this.Height = 720;
bm = new Bitmap(picture.Width, picture.Height);
g = Graphics.FromImage(bm);
tempGraphics = picture.CreateGraphics();

g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
pen.StartCap = pen.EndCap = System.Drawing.Drawing2D.LineCap.Round;
eraser.StartCap = eraser.EndCap = System.Drawing.Drawing2D.LineCap.Round;
peersPen.StartCap = peersPen.EndCap = System.Drawing.Drawing2D.LineCap.Round;

g.Clear(Color.White);
picture.Image = bm;

IPAddress[] localIP = Dns.GetHostAddresses(Dns.GetHostName());

foreach (IPAddress address in localIP)
{
      if (address.AddressFamily == AddressFamily.InterNetwork)
      {
```

```
            TextBoxIP.Text = IP = address.ToString();
        }
}

ButtonStop.Enabled = false;
ButtonDisconnect.Enabled = false;

backgroundWorker1.WorkerSupportsCancellation = true;
backgroundWorker2.WorkerSupportsCancellation = true;
backgroundWorker3.WorkerSupportsCancellation = true;
```

Slika 4.2. *Inicijaliziranje komponenata*

Poslije toga smo u našem programu deklarirali i definirali brojne stavke koje ćemo koristiti kroz program. Neke od njih su olovka i gumica za brisanje. Inicijalna boja olovke je manje bitno no boja gumice je bitna te je ona odmah postavljena na bijelu. Naša gumica za brisanje funkcionira da ona ne briše već nacrtan sadržaj već ga boja u boju pozadine naše crtaće plohe. Zatim smo deklarirali paletu boja. Zatim smo definirali varijablu *painting* koja će nam služiti kao nužan uvjet crtanja. Kada je ona neistina onda se crtanje neće izvršavati no pritiskom miša ona će postati istinita i u tom trenutku će se moći crtati. Poslije nje imamo deklariran dva para točaka koje će nam služiti za crtanje različitih oblika i slobodno crtanje. Zatim imamo defiran *enum* koje unutar sebe ima navedene sve načine crtanja. Načini crtanja su: slobodno crtanje, brisanje, punjenje ploha bojom, crtanje ravnih linija, elipsi te pravokutnika.

Potom imamo niz deklariranih varijabli koje su zadužene za mrežni dio programa. Poslije tog dijela imamo dio deklaracija i definicija koje su zadužene za reproduciranje crtanja izvršeno od strane klijenta. Jedna od njih je privremena grafika na kojoj će se određene naredbe izvršavati i crtati po njoj. Te naredbe su vezane za crtanje oblika koje rade na način da prvo pritiskom lijeve tipke miša prema dolje aktiviramo crtanje oblika te sve dok je to dugme pritisnuto oblik će se crtati iznova i iznova te će se prijašnji brisati. Crtanje se izvršava na privremenoj plohi. U trenutku kada otpustimo tu tipku miša onda će se nacrtati taj oblik na stvarnoj plohi koju poslije možemo i spremiti u obliku slike. Zatim ovdje imamo definiranu olovku drugog klijenta i njegovu boju. Njihove inicijalne vrijednosti nisu bitne jer će nam te vrijednosti kasnije pružiti klijent kod uspostave konekcije i crtanja koje će poslati u posebnom zaglavlju. Zatim imamo instancu *HeaderBuilder* klase koja će služiti za stvaranje posebnih zaglavlja koja će se slati drugom korisniku u slučaju uspostave konekcije i izvršavanju različitih operacija.

```
Bitmap bm;
Graphics g;
Pen pen = new Pen(Color.Black, 2);
Pen eraser = new Pen(Color.White, 10);
```

```
ColorDialog colorDialog = new ColorDialog();

bool painting = false;
Point firstPointFreeDraw, secondPointFreeDraw; //freeDrawing
Point firstPointShapeDraw, secondPointShapeDraw; //shapeDrawing
DrawingMode drawingMode;

private TcpClient client;
private TcpListener listener;
public StreamReader STR;
public StreamWriter STW;
public string recieve;
public string textToSend;
public string IP;

Graphics tempGraphics;
SolidBrush peersCursor = new SolidBrush(Color.FromArgb(68, 68, 64, 66));
Pen peersPen = new Pen(Color.Black, 2);
Color peersColor = new Color();

HeaderBuilder customHeader = new HeaderBuilder();
```

Slika 4.3. *Deklaracije i definicije korištene u programu*

Element *windows* formi na kojem crtamo se naziva *picture*. Kako bismo crtali na njemu postavili smo različite slušatelja događaja (eng: *events*) koji će se aktivirati ovisno o radnjama korisnika. Postavljeni događaji su pritisak tipke računalnog miša, pomicanje računalnog miša, pritisak tipke miša prema dolje, otpuštanje tipke miša prema gore.

Kada je tipka miša pritisnuta prema dolje unutar tog panela varijabla *painting* se prebacuje u istinito stanje što bi značilo započinjanje crtanja. Kada se tipka miša otpusti onda se aktivirao drugi događaj koji bi prebacio varijablu *painting* nazad u lažno što znači prekidanje crtanje. U slučaju da je trenutno aktivan neki od crtajućih načina rada za oblike onda bi se taj oblik nacrtao. Razlog tome je što njih crtamo prvo pritisnemo dugme miša u početnu koordinatu iz koje želimo da započne oblik te druga koordinata do koje crtamo je ona u kojoj otpustimo tipku miša prema gore. Unutar ovog događaja kod izvršavanja svih radnji pozivamo funkciju *SendDataIfConnected()* koja provjerava jer korisnik trenutno spojen s drugim korisnikom. Ako je onda će pozvati metodu za gradnju zaglavlja kojeg će potom poslati korištenjem funkcije *WriteLine()*.

```
private void picture_MouseUp(object sender, MouseEventArgs e)
{
      painting = false;

      if (drawingMode == DrawingMode.Line)
```

```
        {
            g.DrawLine(pen, firstPointShapeDraw, secondPointShapeDraw);
            SendDataIfConnected("Up");
        }
        else if(drawingMode == DrawingMode.Ellipse)
        {
            Rectangle rect = SetRectangle();
            g.DrawEllipse(pen, rect);
            SendDataIfConnected("Up");
        }
        else if(drawingMode == DrawingMode.Rectangle)
        {
            Rectangle rect = SetRectangle();
            g.DrawRectangle(pen, rect);
            SendDataIfConnected("Up");
        }
}
```

Slika 4.4. *funkcija koja se aktivira svaki put kada se otpusti tipka miša*

```
private void SendDataIfConnected(string keyState = "none")
{
        if (client != null)
        {
            STW.WriteLine(customHeader.CustomHeaderBuilder(drawingMode,
                                                           firstPointShapeDraw,
                                                           secondPointShapeDraw,
                                                           pen,
                                                           keyState));
        }
}
```

Slika 4.5. *funkcija zadužena za provjeru uspostavljene konekcije i slanje izvršenih naredbi od strane korisnika*

Još jedan od važnih slušatelja događaja koji je vezan za panel *picture* je pomicanje miša. Njegovu funkciju možemo podijeliti na dva dijela. Prvi dio je zadužen za provjeru dali je aktivirano crtanje, drugim riječima je li *painting* istinit te jer je uspostavljena konekcija s korisnikom. U slučaju da je prvi uvjet neistinit dok je drugi istinut u tom slučaju će se izvršavati poseban dio koda koji je zadužen za slanje zaglavlje sačinjeno od strane koordinata miša. Ovo posebno zaglavlje služi programu druge osobe da na temelju njega može reproducirati pokrete miša. To će postići crtanjem male sive kružnice te svakim pomicanjem miša će se slati nove koordinate. Zatim će drugi korisnik obrisati nacrtanu prijašnju točku te će nacrtati novu.

```
if (!painting && client != null)
{
        STW.WriteLine("~Mouse-" + e.X + "-" + e.Y);
}
```

Slika 4.6. *Dio koda zadužen za slanje koordinata računalnog miša*

Drugi dio ovog slušatelja događaja je zadužen za crtanje prostoručnih linija te brisanje nacrtanog sadržaja. Crtanje i brisanje funkcioniraju na isti način. Koriste se iste operacije crtanja pozivanjem funkcije *DrawLine()*. Jedina razlika je u tome što kod crtanja koristimo različite boje dok kod brisanja koristimo boju pozadine po kojoj crtamo. Ovdje također provjeravamo je li uspostavljena veze s drugim korisnikom te ako je ponovo šaljemo zaglavlje tom korisniku koje on može obraditi i reproducirati sve što smo mi napravili. Nakon izvršene operacije crtanje zaslon se osvježava (eng: *refresh*) kako bi se prikazao nacrtani sadržaj. Funkcija *WriteLine()* je jednostavna funkcija koju pozivamo na stvorenu grafiku na kojoj crtamo. Za argumente prima olovku s kojom crtamo, početnu i krajnju točku linije. Zbog toga što mi svakim pomicanjem miša postavljamo novu početnu i krajnju točku dobijemo učinak prostoručnog crtanja. Kako bismo uopće započeli crtanje prvo provjerimo je li varijabla *painting* istinita koja nam govori je li pritisnuto dugme miša. Zatim provjeravamo koja je trenutno aktivirana operacija crtanja. Ovdje provjeravamo crtanje prostoručnih linija koja je u *enumu* definirana kao *FreeDraw* te brisanje koje je definirano kao *Eraser*.

```
if (painting)
{
      if (drawingMode == DrawingMode.FreeDraw)
      {
            firstPointFreeDraw = e.Location;
            g.DrawLine(pen, firstPointFreeDraw, secondPointFreeDraw);
            if (client != null)
            {
                  STW.WriteLine(customHeader.CustomHeaderBuilder(drawingMode,
                                                                 firstPointFreeDraw,
                                                                 secondPointFreeDraw,
                                                                 pen));
            }
      }
      else if (drawingMode == DrawingMode.Eraser)
      {
             firstPointFreeDraw = e.Location;
             g.DrawLine(eraser, firstPointFreeDraw, secondPointFreeDraw);
             if (client != null)
             {
                   STW.WriteLine(customHeader.CustomHeaderBuilder(drawingMode,
                                                                 firstPointFreeDraw,
                                                                 secondPointFreeDraw,
                                                                 eraser));
             }
      }
```

```
        secondPointFreeDraw = firstPointFreeDraw;
}
picture.Refresh();
secondPointShapeDraw = e.Location;
```

Slika 4.7. *Kod zadužen za crtanje prostoručnih linija i brisanje*

Kod crtanja oblika koristimo dva događaja. Prvi je inicijalni klik koji označava početnu koordinatu oblika te drugi je otpuštanje tipke miša koja označava krajnju točku oblika. Stoga kako bismo to mogli izvesti moramo imati slušatelja događaja koji će pratiti kada je tipka miša puštena. Razlog zašto smo koristili pravokutnik za crtanje kružnica i pravokutnika umjesto koordinata točaka je što nam on omogućuje crtati u svim smjerovima.

```
private void picture_MouseUp(object sender, MouseEventArgs e)
{
        painting = false;

        if (drawingMode == DrawingMode.Line)
        {
                g.DrawLine(pen, firstPointShapeDraw, secondPointShapeDraw);
                SendDataIfConnected("Up");
        }
        else if(drawingMode == DrawingMode.Ellipse)
        {
                Rectangle rect = SetRectangle();
                g.DrawEllipse(pen, rect);
                SendDataIfConnected("Up");
        }
        else if(drawingMode == DrawingMode.Rectangle)
        {
                Rectangle rect = SetRectangle();
                g.DrawRectangle(pen, rect);
                SendDataIfConnected("Up");
        }
}
```

Slika 4.8. *Kod zadužen za izvršavanje crtanja oblika*

Osim ove funkcije imamo identičnu funkciju zaduženu za crtanje po panelu *picture*. Ona crta na potpuno isti način ali za razliku od ove funkcije koja crta trajno po mapi bitova ona je zadužena za privremeno crtanje. Ona crta na privremenoj grafici dok je dugme pritisnuto prema dolje i svakim pomicanjem miša crta novi oblik i briše stari. Ovo nam služi kao pokazatelj kako će izgledati naš nacrtani oblik.

Funkcija *SetRectangle()* ovisno o smjeru postavljenih točaka postavlja oblik pravokutnika kojeg onda koristimo za crtanje tih oblika[5]. Logika funkcije se bazira na provjeri pozicije početne i krajnje točke. Ovisno o tome je li X koordinata početne točke manja od druge ili veća mijenjamo

početku točku pravokutnika. Ovisno o tome oduzimamo početne od krajnje točke ili obrnuto. Kako god uvijek imamo situaciju gdje je širina pozitivan broj jer crtanje pravokutnika ne funkcionira s negativnom širinom. Ovo isto napravimo i za Y koordinatu.

```
private Rectangle SetRectangle()
{
	Rectangle rect = Rectangle.Empty;
	if (firstPointShapeDraw.X < secondPointShapeDraw.X)
	{
		rect.X = firstPointShapeDraw.X;
		rect.Width = secondPointShapeDraw.X - firstPointShapeDraw.X;
	}
	else
	{
		rect.X = secondPointShapeDraw.X;
		rect.Width = firstPointShapeDraw.X - secondPointShapeDraw.X;
	}
	if (firstPointShapeDraw.Y < secondPointShapeDraw.Y)
	{
		rect.Y = firstPointShapeDraw.Y;
		rect.Height = secondPointShapeDraw.Y - firstPointShapeDraw.Y;
	}
	else
	{
		rect.Y = secondPointShapeDraw.Y;
		rect.Height = firstPointShapeDraw.Y - secondPointShapeDraw.Y;
	}
	return rect;
}
```

Slika 4.9. *Funkcija zadužena za postavljanje pravokutnika kojeg koristimo kod crtanja elipsi i pravokutnika*

Unutar grafičkog sučelja imamo šest dugmadi koji su zaduženi postavljanje operacije crtanja. Sve što oni rade je kada se dugme pritisne, ovisno o tome koje je dugme pritisnuto oni postave operaciju crtanja.

```
private void ButtonBrush_Click(object sender, EventArgs e)
{
	drawingMode = DrawingMode.FreeDraw;
}

private void ButtonEraser_Click(object sender, EventArgs e)
{
	drawingMode = DrawingMode.Eraser;
}

private void ButtonFill_Click(object sender, EventArgs e)
```

```
{
	drawingMode = DrawingMode.Fill;
}

private void ButtonLine_Click(object sender, EventArgs e)
{
	drawingMode = DrawingMode.Line;
}

private void ButtonEllipse_Click(object sender, EventArgs e)
{
	drawingMode = DrawingMode.Ellipse;
}

private void ButtonRectangle_Click(object sender, EventArgs e)
{
	drawingMode = DrawingMode.Rectangle;
}
```

Slika 4.10. *Postavljanje operacije crtanja*

Za cijelu formu smo dodali slušatelja događaja koji sluša za pritisnuto dugme te ako je pritisnuto dugme *escape*, u tom slučaju poništava crtanje oblika. To postiže tako da postavi početnu točku jednaku krajnjoj.

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
	switch (e.KeyCode)
	{
		case Keys.Escape:
			firstPointShapeDraw = secondPointShapeDraw;
			break;
	}
}
```

Slika 4.11 *Prekidanje crtanja oblika*

Za podešavanje veličine olovke ili kista imamo posebno tekstualno polje za unos veličine. U njega unosimo veličinu od 1 do 100 te pritiskom na *enter* postavljamo veličinu olovke. U slučaju da je krivi unos onda ostavljamo prijašnju veličinu olovke.

```
private void TextBoxBrushSize_KeyPress(object sender, KeyPressEventArgs e)
{
	if (e.KeyChar == (char)Keys.Enter)
	{
		try
		{
			e.Handled = true;
```

```
                if (Int32.Parse(TextBoxBrushSize.Text) < 1 ||
Int32.Parse(TextBoxBrushSize.Text) > 100)
                {
                        MessageBox.Show("Not a valid input, must be number
between 1 and 100!");
                        TextBoxBrushSize.Text = pen.Width.ToString();
                        this.ActiveControl = null;
                }
                else
                {
                        pen.Width = Int32.Parse(TextBoxBrushSize.Text);
                        eraser.Width = pen.Width;
                        TextBoxBrushSize.Text = pen.Width.ToString();
                        this.ActiveControl = null;
                }
            }
            catch (FormatException)
            {
                  MessageBox.Show("Not a valid input, must be number between 1
and 100!");
            }
      }
}
```

Slika 4.12. *Postavljanje veličine olovke*

Za mijenjanje i postavljanje boje olovke koristimo ugrađenu paletu boja pruženu od strane *windows* formi. Za to koristimo klasu *ColorDialog*. Izmjenu boja vršimo preko dugmeta koje je povezano s instancom te klase. Zatim osim što mijenjamo boju olovke mijenjamo i boju dugmeta kako bismo naznačili koja je trenutačno odabrana boja.

```
private void ButtonColor_Click(object sender, EventArgs e)
{
      colorDialog.ShowDialog();
      ButtonColor.BackColor = colorDialog.Color;
      pen.Color = colorDialog.Color;
}
```

Slika 4.13. *Funkcija za izmjenu boja*

Problem popunjavanja oblika bojom je već dobro poznat u računalstvu. Stoga postoji cijeli niz algoritama za rješavanje takvih problema. Jedan od tih algoritama je algoritam poplave (eng: *Flood fill algorithm*)[6]. On se temelji na algoritmu pretrage u širinu (eng: *Breadth first search*) gdje se stara boja zamjenjuje s novom bojom. Pretraživanje u širinu predstavlja postupak pretrage gdje se prvo pretražuju svi elementi trenutne razine te nakon toga se pretraže svi elementi sljedeće razine.

```
private void picture_MouseClick(object sender, MouseEventArgs e)
```

```
{
	if (drawingMode == DrawingMode.Fill)
	{
		FloodFill(bm, e.Location, pen.Color);
		SendDataIfConnected();
	}
}
```

Slika 4.14. *Funkcija koja pokreće punjenje oblika bojom te koja postavlja početnu točku*

```
private void FloodFill(Bitmap bm, Point pt, Color replacementColor)
{
	Stack<Point> pixels = new Stack<Point>();
	Color targetColor = bm.GetPixel(pt.X, pt.Y);
	pixels.Push(pt);

	while (pixels.Count > 0)
	{
		Point a = pixels.Pop();
		if (a.X < bm.Width && a.X > 0 && a.Y < bm.Height && a.Y > 0)
		{
			if (bm.GetPixel(a.X, a.Y) == targetColor)
			{
				bm.SetPixel(a.X, a.Y, replacementColor);
				pixels.Push(new Point(a.X - 1, a.Y));
				pixels.Push(new Point(a.X + 1, a.Y));
				pixels.Push(new Point(a.X, a.Y - 1));
				pixels.Push(new Point(a.X, a.Y + 1));
			}
		}
	}
	picture.Refresh();
	return;
}
```

Slika 4.15. *Implementacija algoritma poplave[7]*

U padajućem izborniku imamo dvije opcije. Možemo spremiti dokument klikom na dugme s nastavkom *png* te možemo očistiti plohu za crtanje. Za spremanje koristimo *SaveFileDialog()* klasu koja je pružena od strane *windows* formi. Spremamo datoteku na način da kopiramo sadržaj mape bitova u novu mapu bitova koju poslje spremamo s pozivom *save()*. Plohu za crtanje čistimo s *clear()* metodom. Kod čišćenja radne plohe ili zatvaranja aplikacije uvijek pitamo korisnika za potvrdu iz razloga kako osoba ne bi slučajno napravila neku operaciju koju poslije ne može poništiti a pri tome je izgubila podatke.

```
private void saveToolStripMenuItem_Click(object sender, EventArgs e)
{
```

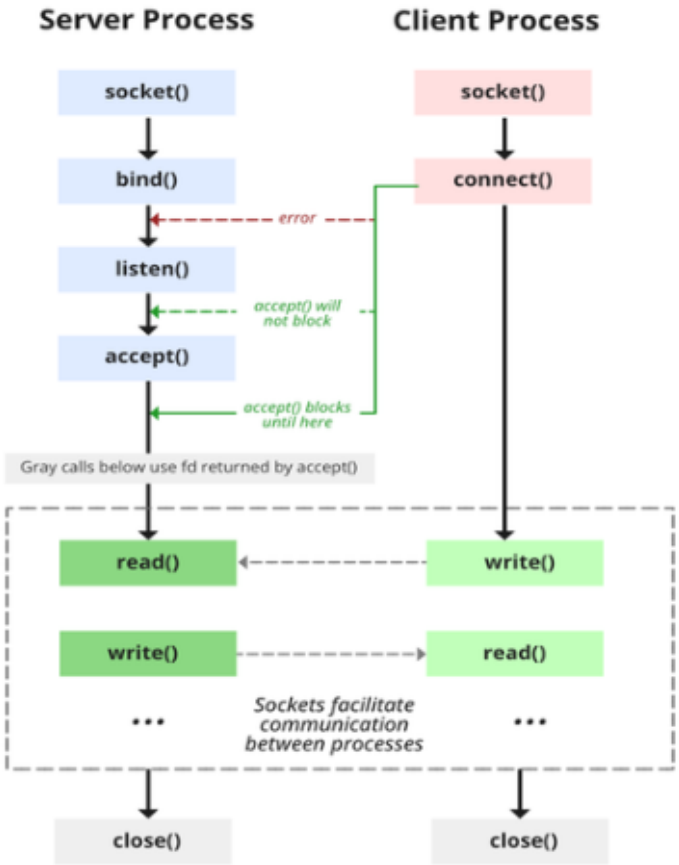
```
        picture.Refresh();
        SaveFileDialog savePicture = new SaveFileDialog();
        savePicture.Filter = "Image(*.png)|*.png|(*.*|*.*";
        if (savePicture.ShowDialog() == DialogResult.OK)
        {
            bm.Save(savePicture.FileName, ImageFormat.Png);
        }
}

private void clearToolStripMenuItem_Click(object sender, EventArgs e)
{
        if(MessageBox.Show("Do you want to clear screen?", "Clear",
MessageBoxButtons.YesNo) == DialogResult.Yes)
        {
            g.Clear(Color.White);
            picture.Image = bm;
            drawingMode = DrawingMode.FreeDraw;
        }
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
        var window = MessageBox.Show("Close the window?", "Are you sure?",
MessageBoxButtons.YesNo);
        e.Cancel = (window == DialogResult.No);
}
```

Slika 4.16. *Kod funkcija zadužen za spremanje radne plohe, čišćenje plohe i isključivanja aplikacije*

4.2 Pregled mrežnog rješenja

Srž mrežnog dijela ovog programa su *socketi* koji omogućuju korisnicima i programu da komuniciraju preko mreže. Kako bi se komunikacija uopće mogla izvršiti potrebno je prvo da jedan od korisnika unese IP adresu svog računala te port na kojem želi ostvariti konekciju. Potom pritiskom dugmeta *Start* on započinje slušati za nadolazeće konekcije. Potom se drugi korisnik može spojiti na prvog korisnika unošenjem IP adrese njegovog računala i port na kojem radi program te pritiskom na dugmeta *Connect*. Za komunikaciju se koristi TCP (eng: *Transmission Control Protocol*) protokol koji za razliku od UDP (eng: *User Datagram Protocol*) protokola omogućuje stabilnu konekciju i razmjenu podataka. Uprava njegova stabilnost i provjera primitka svakog podataka je razlog zašto sam odabrao ovaj protokol u odnosu na UDP bez obzira na to što je sporiji od njega[8][9].

TCP protokol ćemo koristiti u našem program kroz klase *TcpListener* kojeg će poslužitelj koristiti za slušanje i *TcpClient* koji će klijent koristit za spajanje. Jedna prednost korištenja njih u odnosu na klasične *sockete* je u tome što pojednostavljuju međusobnu komunikaciju i upravljanje s konekcijom. Na primjer zajedno s klasama *StreamWriter* i *StreamReader* omogućuju slanje i primanje *string* poruka bez brige oko kodiranje i dekodiranja. Uz to se ne moramo brinuti oko spremnika podataka (eng: *buffer*) te hoće li se njegov kapacitet nadmašiti. Ako bismo koristili dinamičku alokaciju memorije kao u programskim jezicima *C* ili *C++* onda bismo imali novu brigu oko upravljanja s memorijom te bismo morali paziti da ne dođe do curenja memorije (eng: *Memory leak*).



Slika 4.17. *Dijagram rada najčešće korištenih socket* [10]

Osim toga, *TcpListener* smanjuje broj potrebnih poziva metoda kako bismo mogli komunicirati u odnosu na klasične *sockete*. Kod njih, poslužitelj prvo mora kreirati *socket* pozivanjem metode *socket()*. Zatim se mora pozvati *bind()* kako bi se taj *socket* povezao s IP adresom i portom. Potom se moralo pozvati *listen()* kako bi se slušalo za nadolazeću konekciju i na kraju kada se

korisnik želi spojiti na nas morali smo pozvati metodu *accept()*. Zatim bi se komunikacija izvršavala preko metoda *read()* i *write()*. Kod *TcpListener-a* sve što je potrebno je kod kreiranja objekta klase proslijediti IP adresu i broj porta na kojem ćemo slušati i tu smo već kreirali *socket*. Zatim kako bismo slušali za nadolazeće konekcije samo pozovemo metodu *Start()* na kreirani objekt i *AcceptTcpClient()*.

Ovisno o tome hoće li korisnik obnašati ulogu poslužitelja ili klijenta njegov postupak konekcije će biti različiti. Poslužitelj prije uspostavljanja bilo kakve konekcije se otvoriti za nadolazeće konekcije i to će napraviti stvaranjem instance *TcpListener* klase te pozivanjem nje. Nakon toga će program onesposobiti korištenje dugmeta *Start* i *Connect* jer se funkcija poslužitelja već pokrenula. Ujedno će omogućiti dugme *Stop* koje će zaustaviti izvršavanje poslužiteljskih usluga. Nakon podešavanja svojstva ovih dugmadi u prozor za razgovor se ispiše IP adresa poslužitelja i vrijeme pokretanja. Zatim se pokreće pozadinski radnik koji će čekati poruke kako bi glavna nit bila slobodna za izvršavanje grafičkih mogućnosti.

```
private void ButtonStart_Click(object sender, EventArgs e)
{
      try
      {
            listener = new TcpListener(IPAddress.Any,
int.Parse(TextBoxPort.Text));
            listener.Start();

            ButtonStart.Enabled = false;
            ButtonConnect.Enabled = false;
            ButtonStop.Enabled = true;

            TextBoxChatScreen.AppendText("Server started[" + DateTime.Now + "],
with IP:[" + IP + "]" + Environment.NewLine);

            backgroundWorker3.RunWorkerAsync();
      }
      catch (Exception ex)
      {
            MessageBox.Show(ex.Message.ToString());
      }
}
```

Slika 4.18. *Pokretanje poslužitelja*

U pozadinskom radniku imamo prvo „pokušaj i uhvati“ (eng: *try and catch*) dio koda. Kod će se u većini slučajeva normalno izvršavati te će se iznimka (eng: *exception*) jedino pojaviti ako se poslužitelj zaustavi a korisnik je i dalje spojen. Zatim kako bi se radnik izvršavao i u slučaju prekida konekcije smo ostali dio koda stavili unutar beskonačne petlje. Zatim na početku sluša za

nadolazeće konekcije i kada dobije konekciju korisniku šalje inicijalni zaslon kao način sinkroniziranja njihovih ekrana. Ovo se radi kako bi crtali na jednoj te istoj plohi. Slanje inicijalnog zaslona se izvršava pretvaranjem mapa bitova u niz bajtova. Zatim se niz bajtova pretvori u niz karaktera koji se onda oni šalju. Razlog zašto se to mora izvršiti na ovaj način je zbog toga što *StreamWriter* i *SteamReader* rade s karakterima i *stringovima*. Klijent koji prima te podatke samo izvrši obrnuti redoslijed pretvorbi

```
private void SendInitialScreen()
{
      //inicijaliziranje djelove za slanje
      STW = new StreamWriter(client.GetStream());
      STW.AutoFlush = true;

      //pretvaranje bitmape u niz bajtova
      MemoryStream mStream = new MemoryStream();
      bm.Save(mStream, System.Drawing.Imaging.ImageFormat.Bmp);
      byte[] byteImage = mStream.ToArray();

      //pretvaranje bajtova u niz karaktera(string)
      string stringImage = Convert.ToBase64String(byteImage);

      //slanje podataka
      STW.WriteLine(stringImage);
}
```

Slika 4.19. *Funkcija za slanje inicijalnog zaslona*

```
private void GetInitialScreen()
{
      //primanje podataka
      recieve = STR.ReadLine();

      //pretvara string u niz bajtova
      byte[] blob = Convert.FromBase64String(recieve);

      //pretvara niz bajtova u bitmapu
      MemoryStream mStream = new MemoryStream(blob);

      //postavljanje primljene bitmape za trenutnu
      bm = new Bitmap(mStream);
      g = Graphics.FromImage(bm);
      picture.Image = bm;
      picture.Refresh();
}
```

Slika 4.20. *Funkcija za primanje inicijalnog zaslona*

Zatim se kreiraju instance *StreamReader* i *StreamWriter* klase kako bi se mogle razmjenjivati poruke s korisnikom. Zatim se u instanci *StreamWritera* postavlja svojstvo automatskog

pražnjenja kako bismo izbjegli neželjene artefakti prilikom slanja poruka. Zatim se počinje izvršavati nova petlja koja će se izvršavati sve dok je korisnik spojen na poslužitelja. Glavni zadatak ove petlje je čitanje nadolazećih poruka uz pomoć *ReadLine()* metode. Ovdje će se provjeravati nadolazeće poruke te ako dođe prazna vrijednost (eng: *null*) to je znak da je klijent zatvorio konekciju s metodom *close()*. Zatim će se onda i server zatvoriti konekciju s korisnikom.

```
private void backgroundWorker3_DoWork(object sender, DoWorkEventArgs e) //server
{
      try
      {
            while (true) //Održava radnika pokrenutim
            {
                  client = listener.AcceptTcpClient();

                  if(client != null)
                  {
                        SendInitialScreen();
                  }

                  this.TextBoxChatScreen.Invoke(new MethodInvoker(delegate ()
                  {
                        TextBoxChatScreen.AppendText("User connected[" +
DateTime.Now + "]" + Environment.NewLine);
                  }));

                  STR = new StreamReader(client.GetStream());
                  STW = new StreamWriter(client.GetStream());
                  STW.AutoFlush = true;

                 while (client != null) //client.Connected
                 {
                       try
                       {
                             recieve = STR.ReadLine();
                             if (recieve == null) //Aktivira se kada korisnik
zatvori konekciju s .close()
                             {
                                   if (client != null)
                                   {
                                         client.Close();
                                         client = null;
                                   }
```

Slika 4.21. *Prvi dio od tri dijela pozadinskog radnika*

Ovdje ujedno možemo primijeniti da se poruke ispisuju na neobičan način u prostor za poruke. Razlog tome je što mi ispisujemo poruke u grafičko sučelje s kojim radi druga nit. Stoga mi moramo delegirati ovo kako bi pozadinski radnik koja je druga nit mogla ispisati poruku u prostor za poruke grafičkog sučelja.

Zatim možemo vidjeti nakon čitanja poslanih poruka od strani klijenta i provjere je li primljena vrijednost prazna imamo još jednu provjeru. Ova provjera je zadužena za razlikovanje poruka koje je korisnik poslao preko prostora za razgovor i poruka koje su poslane s posebnim zaglavljem koju drugi korisnik treba izvršiti na svom računalu kako bi imali utisak zajedničke plohe za crtanje. Ovdje smo definirali zaglavlje operacija koje se trebaju izvršiti započinju sa znakom tilda(~). Ovaj znak sam po sebi nema nikakvo posebno značenje već razlog zašto sam ga odabrali je zbog toga što se on rijetko koristi prilikom komuniciranja. Samim tim ako korisnik pokuša poslati tekstualnu poruku s tim znakom naš program će izvršiti provjeru i maknut će ga kako ne bi stvorio zabunu programu koji prima poruke. Drugi način na koji smo mogli riješiti problem razlikovanja poruka je da smo koristili neki drugi znak kojeg korisnik nema na tipkovnici te samim tim ga ne bi mogao ni unijeti u prostor za razgovor. Ako je poruka došla koja se treba izvršiti onda se poziva funkcija *ParseAndExecutePeersOperation* gdje ovoj funkciji prosljeđujemo poruku. Ova funkcija je zadužena za tumačenje i izvršavanje nadolazećih operacija. Zatim na kraju ako poruka nije ni prazna ni operacija za izvršavanje onda se ona ispisuje u prostor za poruke te se spremnik u kojem je ona bila isprazni.

```
            this.TextBoxChatScreen.Invoke(new MethodInvoker(delegate ()
            {
                  TextBoxChatScreen.AppendText("User Disconnected[" +
DateTime.Now + "]" + Environment.NewLine);
            }));

            break;
      }
      else if (!string.IsNullOrEmpty(recieve) && recieve[0] == '~')
      {
            ParseAndExecutePeersOperation(recieve);
      }
      else
      {
            this.TextBoxChatScreen.Invoke(new MethodInvoker(delegate ()
            {
                  TextBoxChatScreen.AppendText("You: " + recieve +
Environment.NewLine);
            }));
            recieve = "";
```

```
        }
}
```

Slika 4.22. *Drugi dio od tri dijela pozadinskog radnika*

Na kraju imamo dio koda koji će uhvatiti iznimke kada poslužitelj prekine proces te će zatim zatvoriti konekciju s klijentom.

```
                        catch (Exception) //Aktivira se kada server prekine
proces
                        {
                              if (client != null)
                              {
                                    client.Close();
                                    client = null;
                              }

                              this.TextBoxChatScreen.Invoke(new
MethodInvoker(delegate ()
                              {
                                TextBoxChatScreen.AppendText("User Disconnected["
+ DateTime.Now + "]" + Environment.NewLine);
                             }));

                              break;
                        }
                  }
            }
      }
      catch (Exception ex)
      {
            //Aktivira se kada se server zaustavlja a klijent i dalje slusa
            //MessageBox.Show("Error: " + ex.Message);
      }
}
```

Slika 4.23. *Treći dio od tri dijela pozadinskog radnika*

Slanje poruka se vrši na isti način između korisnika. Pritiskom na dugme *Send* se dohvaća poruka iz prostora za tekst i poziva se pozadinski radnik koji je zaslužan za slanje poruka kako bi nit koja obrađuje grafiku bila slobodna. Slanje poruka osim pritiskom na dugme *Send* možemo poslati i pritiskom tipke *Enter*. Pozadinski radnik prije slanja poruke prvo provjeri dali postoji konekcija te ako postoji zatim provjerava jer prvi znak tilda te ako je onda je uklanja. Zatim na kraju šalje poruku korištenjem *WriteLine()* metode i onda ispiše poruku u prostoru za poruke. Na kraju prekida svoje izvršavanje pozivanjem *CancelAsync()*.

```
private void ButtonSend_Click(object sender, EventArgs e)
{
```

```
      if (TextBoxMessage.Text != "")
      {
            textToSend = TextBoxMessage.Text;
            backgroundWorker2.RunWorkerAsync();
      }
      TextBoxMessage.Text = "";
}

private void TextBoxMessage_KeyPress(object sender, KeyPressEventArgs e)
{
      if (e.KeyChar == (char)Keys.Enter)
      {
            try
            {
                  e.Handled = true;
                  if (TextBoxMessage.Text != "")
                  {
                        textToSend = TextBoxMessage.Text;
                        backgroundWorker2.RunWorkerAsync();
                  }
                  TextBoxMessage.Text = "";
            }
            catch (Exception ex)
            {
                  MessageBox.Show("Error: " + ex.Message);
            }
      }
}
```

Slika 4.24. *Funkcije koje predstavljaju dva načina slanja poruka*

```
private void backgroundWorker2_DoWork(object sender, DoWorkEventArgs e)
{
      if (client.Connected)
      {
            if (textToSend[0] == '~')
            {
                  textToSend = textToSend.Remove(0, 1);
            }

            STW.WriteLine(textToSend);
            this.TextBoxChatScreen.Invoke(new MethodInvoker(delegate ()
            {
                  TextBoxChatScreen.AppendText("Me: " + textToSend +
Environment.NewLine);
            }));
      }
      else
      {
            MessageBox.Show("Sending Failed");
```

```
        }

        backgroundWorker2.CancelAsync();
}
```

Slika 4.25. *Pozadinski radnik koje je zadužen za slanje poruka*

U slučaju da korisnik želi isključiti pokrenutog poslužitelja to će izvršiti dugmetom *Stop*. U tom trenutku se prekida pozadinski radnik koji je slušao nadolazeće poruke. Zatim se isključuje klijent ako je priključen te se na kraju i sam *TcpListener* zaustavi. Zatim se ponovo omogućuje korištenje dugmadi *Start* i *Connect*.

```
private void ButtonStop_Click(object sender, EventArgs e)
{
        try
        {
                backgroundWorker3.CancelAsync();

                if (client != null)
                {
                        client.Close();
                        client = null;
                }

                listener.Stop();

                ButtonStart.Enabled = true;
                ButtonConnect.Enabled = true;
                ButtonStop.Enabled = false;

                TextBoxChatScreen.AppendText("Server stopped[" + DateTime.Now + "]"
+ Environment.NewLine);

        }
        catch (Exception ex)
        {
                MessageBox.Show(ex.Message.ToString());
        }
}
```

Slika 4.26. *Zaustavljanje poslužitelja*

Dugme za priključivanje *Connect* gotovo radi na isti način kao i dugme *Start* kod poslužitelja. Jedna mala razlika je u tome što se ne koristi *TcpListener* već *TcpClient*. Također se mora kreirati instanca *IPEndPoint* klase kojoj prosljeđujemo IP adresu i broj porta na kojeg se želimo spojiti. Zatim na *TcpClient* instancu pozivamo *Connect* te smo ovoj metodi proslijedili instancu *IPEndPoint* klase. Ovdje također podešavamo koja dugmad će se moći koristi a koja neće te

prije nego što pozovemo pozadinskog radnika za klijenta pozivamo funkciju koja će nam dohvati inicijalni zaslon kako bismo se sinkronizirali s drugim korisnikom.

```
private void ButtonConnect_Click(object sender, EventArgs e)
{
	try
	{
		client = new TcpClient();
		IPEndPoint IpEnd = new IPEndPoint(IPAddress.Parse(TextBoxIP.Text),
int.Parse(TextBoxPort.Text));

		client.Connect(IpEnd);

		if (client.Connected)
		{
			ButtonStart.Enabled = false;
			ButtonConnect.Enabled = false;
			ButtonDisconnect.Enabled = true;

			TextBoxChatScreen.AppendText("Connected to the Server[" +
DateTime.Now + "]" + Environment.NewLine);

			STW = new StreamWriter(client.GetStream());
			STR = new StreamReader(client.GetStream());
			STW.AutoFlush = true;

			GetInitialScreen();

			backgroundWorker1.RunWorkerAsync();
		}
	}
	catch (Exception ex)
	{
		client = null;
		MessageBox.Show(ex.Message.ToString());
	}
}
```

Slika 4.27. *Funkcija za priključivanje na drugog klijenta*

Pozadinski radnik klijenta radi gotovo na isti način kao i poslužiteljev. Male razlike se u tome što se ne sluša za nadolazeću konekciju i ne šalje se inicijalni zaslon već se odmah čeka za nadolazeće poruke.

Dugme za prekidanje konekcije radi praktički isto kao i kod poslužitelja samo se ovdje ne mora zaustaviti slušanje na nove konekcije.

```
private void ButtonDisconnect_Click(object sender, EventArgs e)
```

```
{
	try
	{
		backgroundWorker1.CancelAsync();

		if (client != null)
		{
			client.Close();
			client = null;
		}

		ButtonStart.Enabled = true;
		ButtonConnect.Enabled = true;
		ButtonDisconnect.Enabled = false;
	}
	catch (Exception ex)
	{
		MessageBox.Show(ex.Message.ToString());
	}
}
```

Slika 4.28. *Funkcija za prekid konekcije*

Klasa *HeaderBuilder* i njezina metoda *CustomHeaderBuilder* služe za kreiranje zaglavlja koje će se poslati drugom korisniku kad god mi izvršavamo neku naredbu. Ovoj metodi prosljeđujemo način rada, početnu točku, krajnju točku, olovku te vrstu oblika za određene naredbe. Zatim na temelju provjere operacije odabiremo kako ćemo sagraditi zaglavlje. Svako zaglavlje počinje s tildom i načinom rada. Zatim onda ovisno od naredbe mogu ići koordinate početke i krajnje točke te *RGB* (eng: *red, green, blue*) vrijednosti olovke i debljinu olovke. Koje god informacije šaljemo njih odvajamo s crticom kako bismo znali razlikovati koja vrijednost što predstavlja.

```
internal class HeaderBuilder
{
	public string CustomHeaderBuilder(DrawingMode drawingMode, Point
firstPoint, Point secondPoint, Pen drawingPen, string shapeState = "none")
	{
		string customHeader;
		if (drawingMode == DrawingMode.FreeDraw)
		{
			customHeader = "~" + "FreeDraw" +
					"-" + firstPoint.X +
					"-" + firstPoint.Y +
					"-" + secondPoint.X +
					"-" + secondPoint.Y +
					"-" + drawingPen.Color.R +
					"-" + drawingPen.Color.G +
					"-" + drawingPen.Color.B +
```

```
                                "-" + drawingPen.Width;

            }
            else if (drawingMode == DrawingMode.Eraser)
            {
                  customHeader = "~" + "Eraser" +
                                "-" + firstPoint.X +
                                "-" + firstPoint.Y +
                                "-" + secondPoint.X +
                                "-" + secondPoint.Y +
                                "-" + drawingPen.Color.R +
                                "-" + drawingPen.Color.G +
                                "-" + drawingPen.Color.B +
                                "-" + drawingPen.Width;

            }
```

Slika 4.29. *Primjer prvog dijela funkcije za gradnju zaglavlja*

Na kraju programa imamo funkciju koja je zadužena za obrađivanje i izvršavanje poslanih naredbi. Na početku odmah briše prvi znak koji je tilda te potom cijeli primljeni *string* pretvara u listu elemenata. Prilikom pretvorbe elemente koristi se znak crtice za odvajanje dijelova stringa. Zatim zbog već unaprijed poznatog redoslijeda elemenata dohvaćaju se koordinate točaka, *RGB* vrijednosti boje, debljinu olovke i slično. Prvi element predstavlja koja se operacija treba izvršiti te stoga ovisno o njemu će se izvršiti različiti dijelovi koda. Ako se izvršava na primjer gumica prvo će morati posao delegirati zbog toga što se radi o drugoj niti. Zatim će onda koristit već spomenute naredbe za crtanje te će potom osvježiti zaslon. Ako se radi o kompliciranijim naredbama kao što su crtanje oblika onda se one izvršavaju u dva koraka. Od prvog trenutka kada je pritisnuta tipka miša crtat će se po privremenoj plohi te zatim kada se pusti tipka miša izvršit će se crtanje na stvarnoj plohi.

```
private void ParseAndExecutePeersOperation(string recieve)
{
      string recieved = recieve.Remove(0, 1);
      string[] recvTokens = recieved.Split('-');

      if (recvTokens.Length == 9)
      {
            firstPointShapeDraw.X = Int32.Parse(recvTokens[1]);
            firstPointShapeDraw.Y = Int32.Parse(recvTokens[2]);
            secondPointShapeDraw.X = Int32.Parse(recvTokens[3]);
            secondPointShapeDraw.Y = Int32.Parse(recvTokens[4]);

            peersColor = Color.FromArgb(255,
                                        Int32.Parse(recvTokens[5]),
```

```
                                        Int32.Parse(recvTokens[6]),
                                        Int32.Parse(recvTokens[7]));

            peersPen.Color = peersColor;
            peersPen.Width = Int32.Parse(recvTokens[8]);
      }
      if (recvTokens[0] == "Mouse")
      {
            picture.Refresh();
            tempGraphics.FillEllipse(peersCursor, Int32.Parse(recvTokens[1]),
Int32.Parse(recvTokens[2]), 10, 10);
      }
```

Slika 4.30. *Prvi dio koji se bavi tokeniziranje primljene vrijednosti*

```
else if (recvTokens[0] == "LineUp")
{
      this.picture.Invoke(new MethodInvoker(delegate ()
      {
            Try
            {
                  g.DrawLine(peersPen, firstPointShapeDraw,
secondPointShapeDraw);
                  picture.Refresh();
            }
            catch (Exception ex)
            {

            }
      }));
}
else if (recvTokens[0] == "LineDown")
{
      this.picture.Invoke(new MethodInvoker(delegate ()
      {
            try
            {
                  picture.Refresh();
                  tempGraphics.DrawLine(peersPen, firstPointShapeDraw,
secondPointShapeDraw);
            }
            catch (Exception ex)
            {

            }
      }));
}
```

Slika 4.31. *Jedan primjer kako se izvršavaju kompliciranije operacije*

5. OPIS KORIŠTENJA APLIKACIJE

Pokretanjem ovog program dobijemo grafičko sučelje sa zaglavljem na kojem se nalaze različita dugmad. Prvo od njih je dugme *File* koje nam nudi padajući izbornik u kojem možemo spremiti radnu plohu ili je u potpunosti očistiti. Ispod dugmeta koje nam nudi padajući izbornik imamo dugme pomoću kojeg odabiremo boju. Možemo odabrati već gotovo ponuđene boje ili sami odabrati našu klikom na dugme *Defien Custom Colors* koje će nam onda ponuditi unos *RGB* ili *HSL* vrijednosti. Zatim s desne strane imamo prostor za unošenje veličine olovke. Početna je velićina dva te je mi možemo podesiti od 1 do 100. Zatim s desne strane imamo šest dugmadi koja se odnose na načine crtanja. Prvo dugme je odmah aktivirano te ono predstavlja prostoručno crtanje. Drugo dugme je gumica, treće dugme popunjava oblik bojom. Četvrto dugme crta ravne linije, peto kružnice i šesto pravokutnike. Zatim s desne strane imamo dva polja za unos vrijednosti koje se odnose na IP adresu i broj porta. Ispod toga imamo dugme za pokretanje lokalnog poslužitelja koje će omogućiti korisniku da se spoji na našu radnu plohu. Prilikom pokretanja poslužitelja moramo unijeti IP adresu našeg računala i broj porta koji je slobodan. Najčešće su svi portovi od 1024 do 65535 uredu. Drugo dugme služi za spajanje na drugog korisnika. Kako bismo se spojili na korisnika moramo znati njegovu IP adresu i broj porta. Prilikom spajanja sve što smo imali nacrtano će se obrisati i dobit ćemo istu plohu za crtanje kao što je od osobe na koju smo se spojili. Ako se isključimo ta ploha će i dalje ostati na našem zaslonu za crtanje. Ako smo pokrenuli poslužitelja, pritiskom dugmeta *Stop* zaustavljamo naš poslužitelj. To znači da se više nitko neće moći spojiti na našu radnu plohu. U ovim okolnostima će se sve uspostavljene konekcije prekinuti. Kod aplikacije omogućuje da samo jedan korisnik je spojen na drugog korisnika u datom trenutku te da možemo imati samo jednu otvorenu plohu. Ako se korisnik odluči odspojiti pritiskom dugmeta *Disconnect* onda se otvara jedan prostor na kojeg se druga osoba može spojiti. Crtanje po radnoj plohi najnormalnije funkcionira bez obzira dali smo sami ili spojeni na neku drugu plohu ili je netko drugi spojen na nas.

Ploha za crtanje nalazi se ispod zaglavlja i s desne strane od prikaza poruka. Prozor za crtanje je definirane veličine i nije ga moguće mijenjati. S lijeve strane imamo prostor za prikaz poruka i slanje poruka. Sve poslane ili primljene poruke će se prikazati u ovom prostoru. U prostoru za slanje poruka možemo pisati koju god želimo poruku te pritiskom dugmeta na tipkovnici *enter* ili *Send* na grafičkom sučelju šaljemo poruku. Poruke možemo slati jedino kada smo spojeni s drugim korisnikom. Sve poruke ostaju u prozoru za poruke sve dok ne isključimo aplikaciju bez obzira na prekidanje konekcija ili uspostavljanje novih konekcija.

6. ZAKLJUČAK

Tema ovog rada je bila *peer-to-peer* aplikacija za crtanje. Na početku završnog rada smo definirali zadatak. Naveli smo što želimo postići. U našem slučaju to je bila aplikacija koja omogućuje dvojici korisnika crtanje na svom ekranu te sve što jedna osoba nacrta na svom ekranu ona se reproducira drugoj osobi i obrnuto. Potom smo naveli tehnologije koje ćemo koristiti što je u našem slučaju *C#, Windows Forms .NET* i *System.Net.Sockets*. Uz to smo naveli neke njihove prednosti i mane te moguće alternative. Za korišten protokol smo odabrali *TCP* jer nam je bila važnija pouzdanost od brzine. Zatim smo objasnili naše tehničko rješenje programa. Prvi dio programa je bio zadužen za grafiku aplikacije i crtanje dok je drugi bio zadužen za umrežavanje. U prvom djelu programa najveći izazov je bio popunjavanje oblika bojom. Trenutačno rješenje radi kako je zamišljeno ali je njegov problem brzina izvođenja. Osim toga u određenim okolnostima kod iznimno čudnih oblika je čak znalo i usporiti grafičko sučelje. Problem toga je što nit koja je zadužena za implementaciju algoritma je ujedno i ona koja održava grafičko sučelje. Stoga bolje alternative bi bilo da se to računanje preda nekoj drugoj niti ili koristi učinkovitiji algoritam. U drugom tehničkog rješenja prvi problem je stvarala višenitnost. Nužno je bilo koristit više niti kako bi program mogao uopće raditi. Ja sam za to odabrao pozadinske radnike. Moguće je bilo koristit i alternativne za rješavanje tog problema kao što je asinkrono čekanje. Još jedan od problema je bilo upravljati s korisnikovim radnjama. Jedan od tih primjer je korisnik koji želi prekinuti konekciju a nije ni spojen. Moje rješenje na to je bilo da omogućim korisniku samo ono što on uistinu može izvesti u tom trenutku. Također je bilo važno upravljati prekidima konekcije bez da se program sruši što sam je učinio preko upravljanja podignutim iznimkama i korištenja *try catch* blokova. Program ispunjava svoju svrhu ali ima i dalje prostora za njegovo poboljšavanje.

LITERATURA

[1] Microsoft, 2024. C# dokumentacija,, Dostupno na: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/tutorials/ [Pristupljeno: 27.06.2024]

[2] Microsoft, 2023. Windows form dokumentacija, Dostupno na: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-8.0 [Pristupljeno: 27.06.2024]

[3] Microsoft, System.Net.Sockets dokumentacija, Dostupna na: https://learn.microsoft.com/en-us/dotnet/api/system.net.sockets?view=net-8.0 [Pristupljeno: 27.06.2024]

[4] Adobe, PNG vs. TIFF, Dostupno na: https://www.adobe.com/creativecloud/file-types/image/comparison/tiff-vs-png.html [Pristupljeno: 16.09.2024]

[5] Stackoverflow, 2020. How to draw a rectangle in any direction after MouseDown using C#?, Dostupno na: https://stackoverflow.com/questions/60022534/how-to-draw-a-rectangle-in-any-direction-after-mousedown-using-c [Pristupljeno: 16.09.2024]

[6] Geeksforgeeks, 2024. Flood fill algorithm, Dostupno na: https://www.geeksforgeeks.org/flood-fill-algorithm/ [Pristupljeno: 16.09.2024]

[7] K. Oumghar, 2015. Flood Fill algorithm (using C#.Net), Dostupno na: https://simpledevcode.wordpress.com/2015/12/29/flood-fill-algorithm-using-c-net/ [Pristupljeno: 27.06.2024]

[8] Brian „Beej Jorgensen“ Hall, 2024. Beej's Guide to Network Programming, Dostupno na: https://beej.us/guide/bgnet/html/ [Pristupljeno: 27.06.2024]

[9] Geeksforgeek, 2024. Differences between TCP and UDP, Dostupno na: https://www.geeksforgeeks.org/differences-between-tcp-and-udp/ [Pristupljeno: 16.09.2024]

[10] Geeksforgeeks, 2024. Socket Programming in C, Dostupno na: https://www.geeksforgeeks.org/socket-programming-cc/ [Pristupljeno: 27.06.2024]

SAŽETAK

Zadatak ovog završnog rada je bio napraviti peer-to-peer aplikaciju za crtanje. Za rješavanje ovog problema smo koristili tehnologije C#, sockets, TCP protokol, Windows Forms .NET. Uz pomoću Windows formi i ugrađenih biblioteka smo implementirali crtanje u našem programu. Pri tome smo korisniku omogućili veći broj načina crtanja. Korisniku smo pružili jednostavno sučelje za spajanje s drugim korisnikom te mogućnost komunikacije unutar aplikacije. Za mrežnu komunikaciju smo koristili socket i TCP protokol. Prilikom implementacije rješenja naišli smo na više problema u obliku grafike i mrežne komunikacije. Te problem smo riješili na jedan način ali smo ponudili i alternativne pristupe za riješavnje njih. Postojeća implementacija je na kraju u potpunosti izvršila zadani zadatak u pogledu crtanja i umrežavanja.

Ključne riječi: C#, crtanje, peer-to-peer, sockets, TCP

ABSTRACT

PEER-TO-PEER DRAWING APPLICATION

The task of this paper ws to create a peer-to-peer drawing application. To solve this problem we used technologies C#, sockets, TCP protocol, Windows Forms .NET. With the help of Windows forms and built-in libraries, we have implemented drawing in our program. In doing so, we provided the user with a greater number of drawing methods. We have provided the user with a simple interface for connecting with another user and the possibility of communication within the application. We used sockets and TCP protocol for network communication. During the implementation of the solution, we encountered several problems in the form of graphics and network communication. We solved these problems in one way, but we also offered alternative approaches for solving them. The existing implementation ended up fully performing the given task in terms of drawing and meshing.

Keywords: C#, drawing, peer-to-peer, sockets, TCP

ŽIVOTOPIS

Luka Vukić rođen je 25.07.2001. u Osijeku. Pohađao je Osnovnu školu Retfala te zatim Strojarsko tehničku školu Osijek. Za vrijeme pisanja ovog rada pohađa Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Student je treće godine preddiplomskog sveučilišnog studija računarstva smjer programsko inženjerstvo.

_____

Potpis autora