УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Предраг Ђорић, ПР100/2018

2Д ВИЗУЕЛИЗАЦИЈА ШЕМЕ ЕЛЕКТРОДИСТРИБУТИВНЕ МРЕЖЕ СА ФОКУСОМ НА НАЧИН ВИЗУЕЛИЗАЦИЈЕ

ПРОЈЕКАТ

- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 2023.

SADRŽAJ

OPIS REŠAVANOG PROBLEMA

Projekat iscrtava 2D graf elektroenergetske mreže sa fokusom na način vizuelizacije.

Elektroenergetske mreže, kraće električne mreže čine elektroenergetski vodovi (nadzemni i kablovski) i pripadna elektroenergetska postrojenja (transformatorska razvodna postrojenja i rasklopna postrojenja - rasklopišta) povezane u okviru određenog naponskog nivoa ili pripadne funkcije unutar EES[1]. Električne mreže služe za prenos električne energije od elektrana do konzumnih područja i distribuciju do krajnjih potrošača[1].

2D računarska grafika ili 2D grafika je prikaz neke slike na računaru predstavljen u dve dimenzije (kao što je tekst ili digitalna slika)[2]. 2D grafika može povezivati geometrijske modele (vektorska grafika), digitalne slike (rasterska grafika), tekst (koji je definisan sadržajem, fontom, veličinom, bojom i orjentacijom), matematičke funkcije i formule i tako dalje[2].

U nastavku rada čitaćete o rešavanju problema učitavanja, aproksimacije i iscrtavanja elektroenergetske mreže pomoću grafičkih elemenata. Pri iscrtavanju vodova je korišćen BFS algoritam o kome će biti reči u nastavku. Pored osnovog problema iscrtavanja mreže rešavaju se dodatni problemi čiji je fokus na vizuelizaciji. U njih spadaju: zumiranje, bojenje dva entiteta spojenih istim vodom, crtanje elipsi i poligona, dodavanje teksta, *undo*, *redo* i *clear* opcije, čuvanje prikaza kao slike, korišćenje slike za prikaz entiteta, sakrivanje/prikazivanje aktivnog dela mreže, promena boje vodova spram materijala, promena boje entiteta spram broja konekcija.

Prilikom izrade korišćen je već postojeći BFS algoritam sa malim modifikacijama. BFS algoritam je posebno koristan za jednu stvar: pronalaženje najkraće putanje na neponderisanim grafovima[8].

OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

Tehnologije i alati koji su korišćeni za izradu projekta su:

1. Microsoft Visual Studio
2. WPF
3. C#

Microsoft Visual Studio - Integrisano programsko okruženje koje se koristi za programiranje računarskih programa, servisa, igara[3]...

WPF - Windows Presentation Foundation je grafički podsistem za renderovanje korisničkog interfejsa u aplikacijama zasnovanim na *Windows*-u[4].

C# - Programski jezik koji je deo razvojnog okruženja *.NET Framework*. Nema ograničenja u pogledu mogućih aplikacija koje možemo napraviti jer koristi okruženje[5].

OPIS REŠENJA PROBLEMA

Model elektroenergetske mreže je zadat pomoću *Geographic.xml* fajla (Slika 3.1.). Svi zadati podaci zajedno predstavljaju šemu elektroenergetske mreže Novog Sada. U njemu se nalaze podaci o svim entitetima i vodovima. Podaci unutar *Geographic.xml* fajla su zabeleženi uz pomoć UTM formata. UTM format predstavlja univerzalni poprečni Merkatorov koordinatni sistem za pozicioniranje na površini Zemlje horizontalnim prikazom položaja koji zanemaruje nadmorsku visinu i tretira zemlju kao savršen elipsoid. Razlikuje se od globalne geografske širine/dužine po tome što deli zemlju na 60 zona i svaku projektuje na ravan kao osnovu za sopstvene koordinate[12].

```
<NetworkModel>
  <Substations>
  <Nodes>
  <Switches>
  <Lines>
    <LineEntity>
      <Id>33947</Id>
      <Name>SEC_137438957044</Name>
      <IsUnderground>true</IsUnderground>
      <R>0.209</R>
      <ConductorMaterial>Steel</ConductorMaterial>
      <LineType>Cable</LineType>
      <ThermalConstantHeat>2400</ThermalConstantHeat>
      <FirstEnd>41990</FirstEnd>
      <SecondEnd>41992</SecondEnd>
      <Vertices>
        <Point>
          <X>407566.68007470988</X>
          <Y>5013899.3558040857</Y>
        </Point>
        <Point>
          <X>407625.00589398207</X>
          <Y>5013876.8697334668</Y>
        </Point>
        <Point>
          <X>407717.51971015992</X>
          <Y>5014160.9525629422</Y>
        </Point>
        <Point>
          <X>407559.40091708023</X>
          <Y>5014220.4665799234</Y>
        </Point>
      </Vertices>
    </LineEntity>
```

Slika 3.1. *Geographic.xml*

Podaci koji su nam dati su: vodovi, trafostanice, čvorovi, prekidači. Na osnovu tih podataka su kreirane klase u projektu i one predstavljaju modele realnog sistema. Klasa *PowerEntity* predstavlja entitet mreže i sadrži polja koja ga opisuju uz *get* i *set* metode svakog polja (Slika 3.2.). Klase koje nasleđuju klasu *PowerEntity* su: *SwitchEntity, NodeEntity, SubstationEntity*. Kako sve te klase predstavljaju jedan entitet imaće ista polja, a *SwitchEntity* će imati dodatno polje *status* koje nam govori da li je prekidač aktivan ili ne. Pored entiteta postoji klasa *Point* čija su polja: *x, y*. Ta polja će nam pomoći da dobijemo stvarne koordinate nakon konvertovanja UTM formata. Poslednja klasa o kojoj imamo podatke u *Geographic.xml* fajlu je klasa *LineEntity* koja sadrži podatke vodova elektroenergetske mreže (Slika 3.3.).

```
public class PowerEntity
{
    private long id;
    private string name;
    private double x;
    private double y;
    private string toolTip;
    private Brush boja;
    private string brojKonekcija;
    private int obojSlikom;
```

Slika 3.2. Klasa *PowerEntity*

```
public class LineEntity
{
    private long id;
    private string name;
    private bool isUnderground;
    private float r;
    private string conductorMaterial;
    private string lineType;
    private long thermalConstantHeat;
    private long firstEnd;
    private long secondEnd;
    private List<Point> vertices;
    //za BFS
    private double pocetakX;
    private double pocetakY;
    private double krajX;
    private double krajY;
    private string prolaz;
```

Slika 3.3 Klasa *LineEntity*

Strukture podataka koje su korišćene pri izradi su liste za skaldištenje entiteta, vodova, elemenata koji se iscrtavaju na *canvas* i *dictionary* kako bi se zapamtilo mesto na kome je iscrtan jedan entitet sa ciljem sprečavanja preklapanja.

Da bismo dobili geografsku širinu i dužinu na početku je bilo potrebno konvertovati UTM format (Slika 3.4.).

```
//From UTM to Latitude and longitude in decimal
public static void ToLatLon(double utmX, double utmY, int zoneUTM, out double latitude, out double longitude)
{
    bool isNorthHemisphere = true;

    var diflat = -0.00066286966871111111111111111111111111;
    var diflon = -0.0003868060578;

    var zone = zoneUTM;
    var c_sa = 6378137.000000;
    var c_sb = 6356752.314245;
    var e2 = Math.Pow((Math.Pow(c_sa, 2) - Math.Pow(c_sb, 2)), 0.5) / c_sb;
    var e2cuadrada = Math.Pow(e2, 2);
    var c = Math.Pow(c_sa, 2) / c_sb;
    var x = utmX - 500000;
    var y = isNorthHemisphere ? utmY : utmY - 10000000;

    var s = ((zone * 6.0) - 183.0);
    var lat = y / (c_sa * 0.9996);
    var v = (c / Math.Pow(1 + (e2cuadrada * Math.Pow(Math.Cos(lat), 2)), 0.5)) * 0.9996;
    var a = x / v;
    var a1 = Math.Sin(2 * lat);
    var a2 = a1 * Math.Pow((Math.Cos(lat)), 2);
    var j2 = lat + (a1 / 2.0);
    var j4 = ((3 * j2) + a2) / 4.0;
    var j6 = ((5 * j4) + Math.Pow(a2 * (Math.Cos(lat)), 2)) / 3.0;
    var alfa = (3.0 / 4.0) * e2cuadrada;
    var beta = (5.0 / 3.0) * Math.Pow(alfa, 2);
    var gama = (35.0 / 27.0) * Math.Pow(alfa, 3);
    var bm = 0.9996 * c * (lat - alfa * j2 + beta * j4 - gama * j6);
    var b = (y - bm) / v;
    var epsi = ((e2cuadrada * Math.Pow(a, 2)) / 2.0) * Math.Pow((Math.Cos(lat)), 2);
    var eps = a * (1 - (epsi / 3.0));
    var nab = (b * (1 - epsi)) + lat;
    var senoheps = (Math.Exp(eps) - Math.Exp(-eps)) / 2.0;
    var delt = Math.Atan(senoheps / (Math.Cos(nab)));
    var tao = Math.Atan(Math.Cos(delt) * Math.Tan(nab));

    longitude = ((delt * (180.0 / Math.PI)) + s) + diflon;
    latitude = ((lat + (1 + e2cuadrada * Math.Pow(Math.Cos(lat), 2) - (3.0 / 2.0) * e2cuadrada * Math.Sin(lat) * Math.Cos(lat) * (tao - lat)) * (tao - lat)) * (180.0 / Math.PI)) + diflat;
}
```

Slika 3.4. Dobijanje geografske širine i dužine iz UTM formata

Nakon što je izvršena konverzija podaci moraju da se skaliraju za prikaz na računaru. Kako bi proračun bio optimalan površina za prikaz je podeljena na zamišljene podeoke kreiranjem matrice 300x300 (Slika 3.5.). Ideja je da se realna mreža predstavi korišćenjem razmere. Suštinski mreža ostaje ista samo je umanjena i svaki objekat pronalazi svoje odgovarajuće mesto na umanjenom prikazu čime se održava tačnost same pozicije i udaljenosti od drugih objekata.

```
private void UcitavanjeMatrice()
{
    //Pravim matricu
    Point rt;

    for (int i = 0; i <= 300; i++)
    {
        for (int j = 0; j <= 300; j++)
        {
            rt = new Point(i, j);
            matrica.Add(rt);
        }
    }
}
```

Slika 3.5. Matrica koja deli *canvas* na zamišljene podeoke

Da bi skaliranje bilo moguće moramo znati razliku najveće i najmanje tačke geografske dužine i geografske širine. Njih nalazimo upoređivanjem koordinata svih objekata. Nakon što odradimo taj korak imamo mogućnost da odredimo koliko prostora zauzimaju geografska širina i geografska dužina i skaliramo ih na matrici (Slika 3.6.).

```
if (checkMinMax == 1)
{
    praviXmax = noviX;
    praviYmax = noviY;
    praviXmin = noviX;
    praviYmin = noviY;

    checkMinMax++;
}
else
{
    //proveraMAX
    if (noviX > praviXmax)
    {
        praviXmax = noviX;
    }

    if (noviY > praviYmax)
    {
        praviYmax = noviY;
    }

    //proveraMIN
    if (noviX < praviXmin)
    {
        praviXmin = noviX;
    }

    if (noviY < praviYmin)
    {
        praviYmin = noviY;
    }
}
odstojanjeLon = (praviXmax - praviXmin) / 300;
odstojanjeLat = (praviYmax - praviYmin) / 300;
```

Slika 3.6. Skaliranje celokupne mreže prema našoj matrici

Nakon što je mreža skalirana sledeći zadatak je iscrtavanje entiteta na toj mreži. Algoritam za iscrtavanje entiteta se ponavlja za svaki entitet koji je učitan iz *Geographic.xml* fajla (Slika 3.7.). Entiteti se iscrtavaju u obliku kvadrata sa tim da ukoliko dođe do preklapanja vrši se postavljanje pozicije na najbliže slobodno mesto.

```
private void CrtajEntitete()
{
    foreach (var element in listaElemenataIzXML)
    {
        ToLatLon(element.X, element.Y, 34, out noviX, out noviY);
        MestoNaCanvasu(noviX, noviY, out relativnoX, out relativnoY);

        Rectangle rect = new Rectangle();
        rect.Fill = element.Boja;
        rect.ToolTip = element.ToolTip; //za hover prikaze info o elementu
        rect.Width = 2;
        rect.Height = 2;

        Bojenje po broju konekcija

        Bojenje slikom

        Bez preklapanja
        //mnozim sa 3 da ne bi bilo preklapanja
        Canvas.SetBottom(rect, mojaTacka.X*3);
        Canvas.SetLeft(rect, mojaTacka.Y*3);
        canvas.Children.Add(rect);
    }
}
```

Slika 3.7. Algoritam za iscrtavanje entiteta

Prolaskom kroz algoritam za iscrtavanje svaki entitet se skalira i dobija odgovarajuću poziciju koja se određuje ulaskom u metodu *MestoNaCnvasu*, a potom se sa njim izvršavaju akcije poput bojenja, dodavanja opisa, podešavanja visine i širine, podešavanja pozicije i na kraju samog iscrtavanja. Entitet dobija svoju poziciju tako što dobija svoje relativne koordinate. One se određuju time što se tačke x i y oduzmu od

minimalnih tačaka x i y koje se javljaju kako bi se ustanovila udaljenost od početne tačke, a potom deljenjem sa odstojanjem već skalirane ose (Slika 3.8.).

```
private void MestoNaCanvasu(double noviX, double noviY, out double relativnoX, out double relativnoY)
{
    relativnoX = Math.Round((noviX - praviXmin) / odstojanjeLon);
    relativnoY = Math.Round((noviY - praviYmin) / odstojanjeLat);
}
```

Slika 3.8. Aproksimacija - pronalaženje pozicije entiteta

Nakon što su iscrtani entiteti, iscrtavaju se vodovi. Algoritam za iscrtavanje vodova prolazi kroz svaki vod koji se nalazi u listi vodova (dobijena čitanjem *Geographic.xml* fajla). Za iscrtavanje vodova potrebno je odrediti početnu i krajnju tačku, a potom se koristi BFS (*Breadth First Search*) algoritam. BFS algoritam pronalazi najkraći mogući put za iscrtavanje vodova tako da nema međusobnog presecanja. U drugom prolazu algoritma se preseci vodova dešavaju, ali je mesto preseka označeno elipsom kako bi se korisniku dalo do znanja da presek postoji.

Za potrebe BFS algoritma kreirane su klase: *PozicijaPolja, BFSProm, BFSPath*. Klasa *PozicijaPolja* sadrži polja *pozX* i *pozY* koje koristimo kao koordinate u toku rada sa BFS-om. Klasa *BFSProm* sadrži promenljive koje se koriste za označavanje početka, kraja, zida ili slobodnog mesta na našoj mapi. Cilj ovih polja je onemogućivanje preseka vodova i spajanje početka i kraja voda. Klasa *BFSPath* sadrži metode za pronalaženje najkraće putanje, proveru da li je validna i rekonstruisanje same putanje pri pozivu iscrtavanja.

Na početku poziva BFS algoritma potrebno je postaviti početnu i krajnju tačku voda. Nakon toga algoritam počinje da traži putanju i skladišti polja koja je prešao u matricu *prev* koju kasnije koristi za rekonstruisanje putanje (Slika 3.9.).

```
PozicijaPolja pocetak = new PozicijaPolja((int)line.PocetakX, (int)line.PocetakY);
PozicijaPolja kraj = new PozicijaPolja((int)line.KrajX, (int)line.KrajY);
PozicijaPolja[,] prev = BFSPath.BFSPronadji(line, BFSprom.BFSlinije);
List<PozicijaPolja> putanja = BFSPath.RekonstruisanjePutanje(pocetak, kraj, prev);
```

Slika 3.9. Promenljive potrebne za BFS algoritam

Sam algoritam počinje pozivanjem metode *BFSPronadji*. Metoda zna polje koje je početak i zna polje koje je kraj. Pre početka traženja se kreiraju matrice posećenih polja gde je pri inicijalizaciji svako polje neposećeno (osim početnog) i matrice prethodnih polja gde nema prethodnih pri incijalizaciji. Matrice su iste veličine kao i početna matrica koja je služila za entitete. U red se smešta polje koje predstavlja početak voda i pretraga počinje. Za svako polje koje se nađe u redu se gleda tačka koja je iznad, ispod, levo ili desno (Slika 3.10.). Vrši se provera validnosti koja proverava da li je polje izvan zadatih dimenzija matrice 300x300 ili je polje zid, to jest već posećeno. Ukoliko se desi neki od slučaja obrada se preskače jer je polje već obrađeno ili nema potrebe da se obrađuje.

Pretraga se vrši sve dok se ne dođe do polja koje predstavlja kraj voda. Tačke koje nisu obrađene se dodaju u red i predstavljaju mogući najkraći put pa se za njih vrši isti postupak obrade kao da su početna polja. Polje dobija oznaku posećeno kako se ne bi proveravalo ponovo, a matrica *prev* dobija polje koje je algoritam posetio.

```
while(queue.Count != 0)
{
    PozicijaPolja polje = queue.Dequeue();

    //dosao do kraja
    if(BFSlinije[polje.PozX,polje.PozY] == 'K')
    {
        BFSlinije[(int)line.PocetakX, (int)line.PocetakY] = ' ';
        BFSlinije[(int)line.KrajX, (int)line.KrajY] = ' ';
        return prev;
    }

    if(isValid(polje.PozX-1, polje.PozY, poseceno, BFSlinije))
    {
        queue.Enqueue(new PozicijaPolja(polje.PozX - 1, polje.PozY));
        poseceno[polje.PozX - 1, polje.PozY] = true;
        prev[polje.PozX - 1, polje.PozY] = polje;
    }

    if (isValid(polje.PozX + 1, polje.PozY, poseceno, BFSlinije))
    {
        queue.Enqueue(new PozicijaPolja(polje.PozX + 1, polje.PozY));
        poseceno[polje.PozX + 1, polje.PozY] = true;
        prev[polje.PozX + 1, polje.PozY] = polje;
    }

    if (isValid(polje.PozX, polje.PozY-1, poseceno, BFSlinije))
    {
        queue.Enqueue(new PozicijaPolja(polje.PozX, polje.PozY-1));
        poseceno[polje.PozX, polje.PozY-1] = true;
        prev[polje.PozX, polje.PozY-1] = polje;
    }
    if (isValid(polje.PozX, polje.PozY + 1, poseceno, BFSlinije))
    {
        queue.Enqueue(new PozicijaPolja(polje.PozX, polje.PozY + 1));
        poseceno[polje.PozX, polje.PozY + 1] = true;
        prev[polje.PozX, polje.PozY + 1] = polje;
    }
}
```

Slika 3.10. Pretraga polja pomoću BFS algoritma

Nakon što algoritam pronađe put od početnog do krajnjeg polja potrebno je rekonstruisati putanju koja predstavlja vod. Metoda *RekonstruisanjePutanje* kao parametre prima početnu tačku, krajnju tačku i tačke koje je algoritam posetio (matricu *prev*). Rekonstruisanje putanje počinje od poslednje tačke i vraća se korak unazad sve dok ne dođe do početne (Slika 3.11.).

```
List<PozicijaPolja> putanja = new List<PozicijaPolja>();
PozicijaPolja polje;

for(polje=kraj; polje != null; polje = prev[polje.PozX, polje.PozY])
{
    putanja.Add(polje);
}
putanja.Reverse();
```

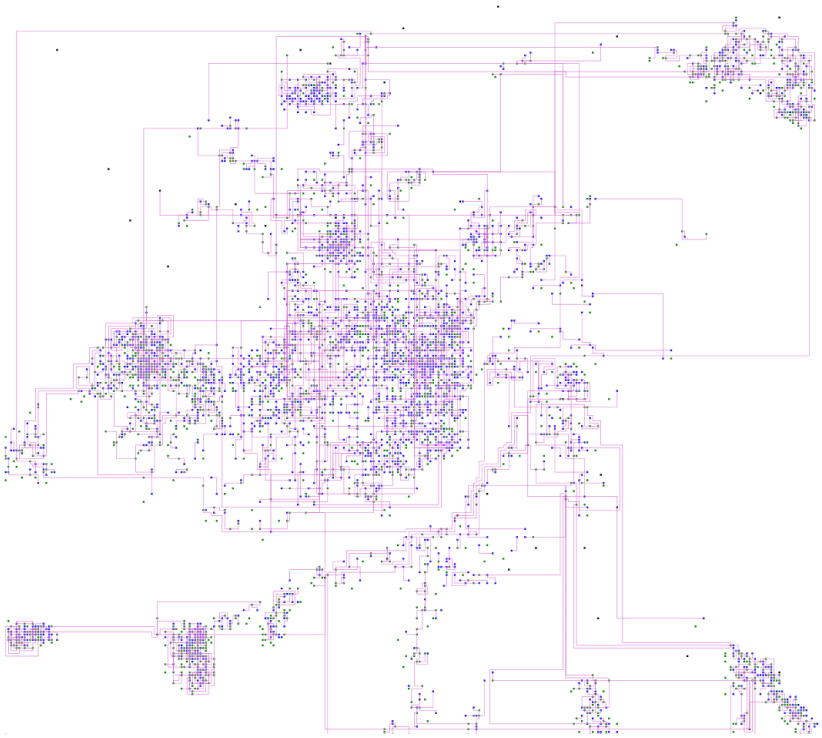
Slika 3.11. Algoritam rekonstruisanja putanje

Pri dodavanju vodova početna i krajnja tačka se uzimaju kao već poznate, a svaka naredna se uzima prolaskom kroz dobijenu putanju. Svaki vod se iscrtava kao *polyline* objekat određen dobijenim tačkama. Svako polje koje se iskoristi za *polyline* objekat se obeležava i predstavlja zid. Nakon prolaska kroz BFS algoritam desiće se da neki vodovi nisu iscrtani jer nije bilo moguće proći kroz zid, to jest preseći drugi već postojeći vod. Zbog toga se kroz BFS algoritam prolazi iznova samo što se u ovom slučaju dozvoljavaju preseci sa već prethodno iscrtanim vodovima i oni se označavaju kao preklapanje crtanjem elipse (Slika 3.12.).

```
foreach (PozicijaPolja zauzeto in putanja)
{
    if(BFSprom.BFSlinije[zauzeto.PozX, zauzeto.PozY] == 'X')
    {
        if (iscrtaoPresek == false)
        {
            Ellipse preklapanje = new Ellipse();
            preklapanje.Height = 0.5;
            preklapanje.Width = 0.5;
            preklapanje.Fill = Brushes.Crimson;
```

Slika 3.12. Obeležavanje preseka vodova

Konačan izgled elektroenergetske mreže sa iscrtanim vodovima i entitetima prikazan je na slici 3.13.

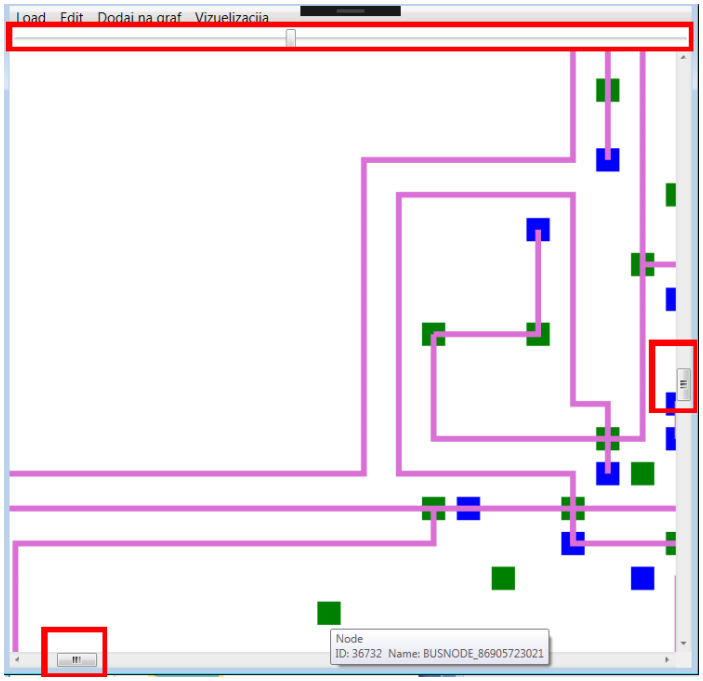


Slika 3.13. Izgled iscrtane elektroenergetske mreže

Kako bi pregled mogao biti detaljniji moguće je približiti ili odaljiti elektroenergetsku mrežu korišćenjem *slider*-a i potom se kretati levo i desno (Slike 3.14. i 3.15.).

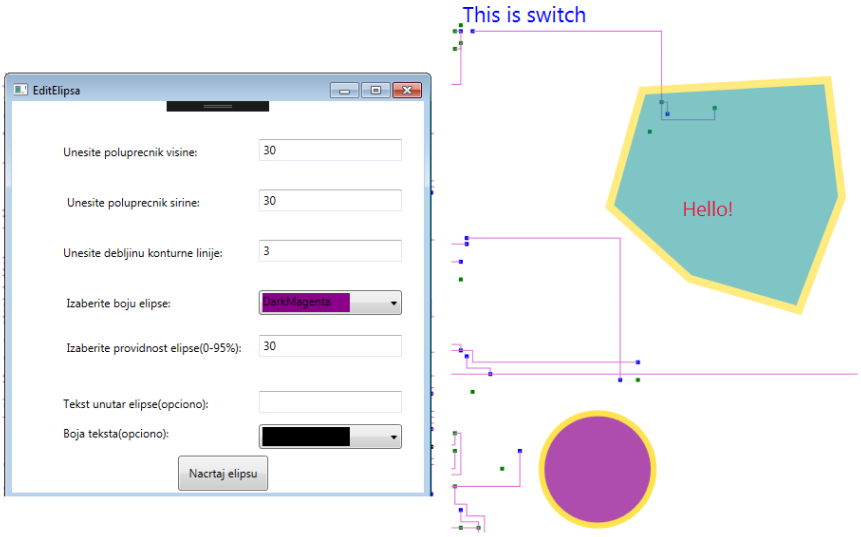
```
<Slider x:Name="slider" Minimum="0.3" Maximum="30" Value="1" DockPanel.Dock="Top"/>
<ScrollViewer HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">
    <Canvas x:Name="canvas" DockPanel.Dock="Bottom" Height="900" Width="905" Focusable="True" MouseLeftButtonDown="LeviPromeniNesto_Click" >
        <Canvas.LayoutTransform>
            <ScaleTransform x:Name="SkaliranjeTransform" ScaleX="{Binding ElementName=slider, Path=Value}"
                            ScaleY="{Binding ElementName=slider, Path=Value}"/>
        </Canvas.LayoutTransform>
    </Canvas>
</ScrollViewer>
```

Slika 3.14. Implementacija *slider*-a



Slika 3.15. Zumiranje i kretanje

Nakon što je kompletna elektroenergetska mreža iscrtana korisnik ima mogućnost dodavanja elipsi, poligona i teksta. Oblici pružaju mogućnost promene atributa i vršenja različitih podešavanja (Slika 3.16.).

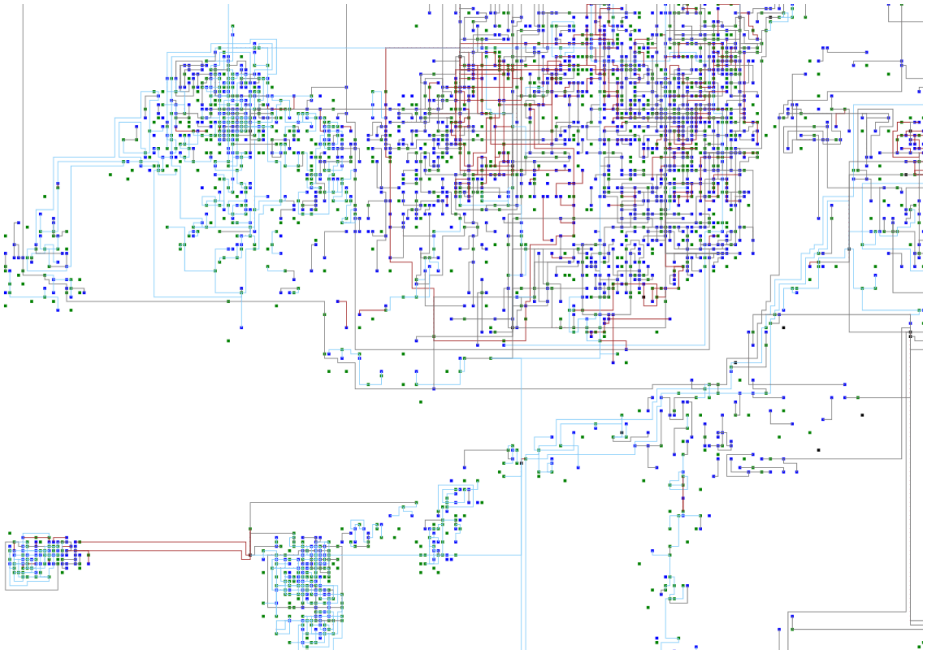


Slika 3.16. Kreiranje i izmena oblika

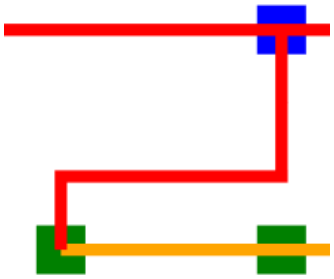
Korisnički meni nad oblicima pruža akcije *undo*, *redo* i *clear*. Korisnik može da obriše sve već iscrtane oblike ili da se vrati nekoliko koraka unazad kako bi sklonio neki određeni oblik.

Mogućnosti prikaza elektroenergetske mreže su sledeće: prikaz aktivnog dela mreže, promena boje entiteta spram broja konekcija (Slika 3.17.), prikaz vodova spram materijala kojim su konstruisani (Slika 3.18.), prikaz boje vodova spram otpornosti (Slika 3.19.).

Slika 3.17. Promena boje entiteta spram broja konekcija



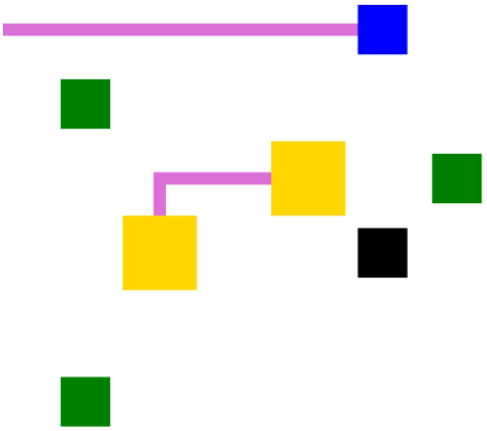
Slika 3.18. Bojenje vodova spram konstrukcionog materijala



Slika 3.19. Promena boje vodova spram otpornosti

Korisnik entitete može prikazati kao sliku (Slika 3.20.) i klikom na vod može naglasiti entitete koji su krajevi tog voda (Slika 3.21.).

Slika 3.20. Entiteti su predstavljeni odabranom slikom



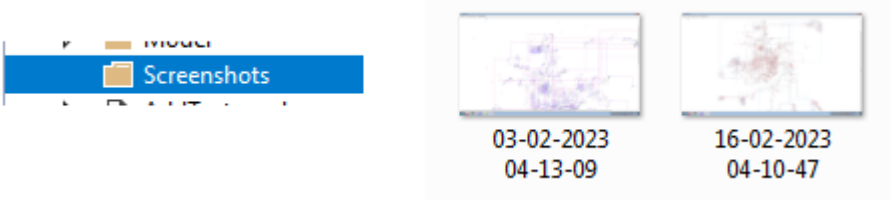
Slika 3.21. Početna i krajnja tačka voda se označavaju zlatnom bojom klikom na vod

Na kraju korisnik može da sačuva trenutni prikaz elektroenergetske mreže kao sliku koja sadrži vremenski datum kada je napravljena (Slika 3.22. i Slika 3.23.).

```
using (System.Drawing.Bitmap bmp = new System.Drawing.Bitmap((int)screenWidth,
    (int)screenHeight))
{
    using (System.Drawing.Graphics g = System.Drawing.Graphics.FromImage(bmp))
    {
        String filename = DateTime.Now.ToString("dd-MM-yyyy hh-mm-ss") + ".png";
        Opacity = .0;
        g.CopyFromScreen((int)screenLeft, (int)screenTop, 0, 0, bmp.Size);
        string pozicijaFoldera = Environment.CurrentDirectory;
        string path = System.IO.Path.GetFullPath(System.IO.Path.Combine(pozicijaFoldera, @"..\..\"));
        path = path+"Screenshots\\"+filename;
        bmp.Save(path);
        Opacity = 1;
    }

}
```

Slika 3.22. Čuvanje slike u *screenshots* folderu



Slika 3.23. Čuvanje trenutnog prikaza *canvas*-a

PREDLOZI ZA DALJA USAVRŠAVANJA

Rezultat rada aplikacije omogućava iscrtavanje grafa mreže na osnovu *Geographic.xml* fajla koji će biti skaliran za prikaz na ekranu i omogućiti korisniku uvid u elektroenergetsku mrežu Novog Sada. Korisniku se pružaju informacije o svakom grafičkom elementu koji predstavlja čvor mreže. Korisnik ima mogućnost dodavanja novih grafičkih elemenata i njihove promene u skladu sa svojim željama.

Prednosti koje su pružene korisniku mogućnostima aplikacije se odlikuju slobodnim korišćenjem akcija koje aplikacija nudi uz mogućnost promene atributa i izgleda oblika i teksta. Pružena je mogućnost popravke grešaka akcijama *undo* i *redo*, ili potpuno brisanje svih oblika i teksta akcijom *clear*. Slobodno korišćenje same elektroenergetske mreže je omogućeno korisničkim menijem koji pruža čekiranje/odčekiranje, to jest odabir stanja u kome se mreža nalazi.

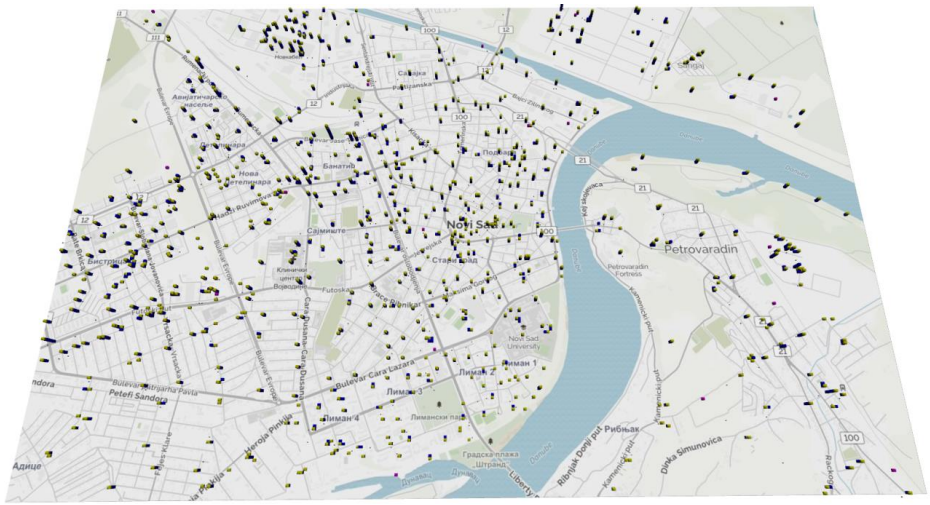
Jedan od nedostataka koji se javlja je vremenski period koji je potreban za odziv aplikacije nakon određenih akcija, koji je nastao usled velikog broja proračuna koje izvršava BFS algoritam.

Predlozi za dalja usavršavanja:

1. Mogućnost daljeg usavršavanja postoji u postavljanju stvarne mape Novog Sada ispod elektroenergetske mreže radi boljeg geografskog snalaženja (Slika 4.1.). Pored toga, moguće je dodati nove elemente ukoliko postoji potreba.

2. Ukoliko bi postojala potreba da se vrši promena nad isključivo jednim delom mreže moguće usavršavanje bi bilo odsecanje dela korišćenjem već postojeće akcije za kreiranje geometrijskih oblika poligona i elipse (Slika 4.2.).

3. Optimizacija izvršavanja svake akcije optimalnijim odzivom uz pažnju da se ne nanesu gubici elektroenergetskoj mreži.



Slika 4.1. Postavljanje mape Novog Sada

Slika 4.2. Korišćenje elipse i poligona za odsecanje određenog dela mape

LITERATURA

[1] ***Jadranka Radović, Elektrodistributivni sistemi, 2017, https://www.ucg.ac.me/skladiste/blog_10913/objava_90140/fajlovi/___!!!IKolEDSPred17_18.pdf***

[2] ***Wikipedia, 2D računarska grafika, 2021, Wikipedia***

[3]***Wikipedia,Microsoft Visual Studio, 2022, Wikipedia***

[4]***Wikipedia, WPF, 2020, Wikipedia***

[5]***Wikipedia, C#, 2022, Wikipedia***

[6] ***Hollie Buckley, WPF for IT Students, May 2016***

[7] ***Jack Xu, Practical WPF Graphics Programming, November 2007***

[8] ***Youtube, William Fiset, Breath First Search Algorythm, April 2018, Youtube***

[9] ***Shortest distance between two cells in a matrix of grid, Jun 2022, www.geeksforgeeks.org***

[10] ***Milan Kilibarda i Dragutin Protić, Geovizuelizacija i web kartografija, 2018, Univerzitet u Beogradu***

[11] ***Vladimir C. Strezorski, Osnovi elektroenergetike, 2014, Fakultet tehničkih nauka u Novom Sadu***

[12] ***Wikipedia, Univerzalni Transverzalni Merkatorov koordinatni sistem, Wikipedia***