УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Немања Лазаревић, PR 32/2019

2Д РАЧУНАРСКА ГРАФИКА ИНТЕРАКЦИЈА

ПРОЈЕКАТ
- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 02.09.2023.

САДРЖАЈ

ОПИС РЕШАВАНОГ ПРОБЛЕМА

Овај пројекат се бави детаљним графичким приказом електроенергетске мреже и пружа различите алате и функционалности за интерактивно уређивање и анализу мреже.

Циљ овог пројектног задатка је стварање визуализације електроенергетске мреже помоћу ортогоналног приказа, користећи податке из Geographic.xml датотеке. Фокус пројекта је интеракција са електроенергетском мрежом ради лакшег коришћења.

Овај пројекат се базира на графичком приказу електроенергетске мреже, где се скуп чворова (географске локације на којима се налазе елементи мреже попут трансформатора, станица итд.) у једном географском простору међусобно повезаних линија мреже користи за пренос електричне енергије од произвођача до крајњих корисника.

Кључни аспекти овог пројекта обухватају:

Исцртавање мреже: Графички приказ електроенергетске мреже се генерише на основу садржаја Geographic.xml датотеке користећи ортогонални приказ.

Ентитети мреже: Ентитети (трафостанице, чворови, прекидачи) се апроксимирају на најближи слободан простор, без преклапања.

Исцртавање веза: Везе које повезују ентитете исцртавају се као праве линије које се крећу из центра сваког ентитета. Линије скрећу под правим углом и уколико дође до њиховог пресека исцртава се чвор.

Примена БФС алгоритма: Овај алгоритам проналази најкраћи пут између ентитета без пресецања већ постојећих веза. У другом пролазу, исцртавају се везе за које није било могуће пронаћи пут без пресецања.

Анимација: Десним кликом на везу покреће се анимација која повећава графичке елементе повезаних ентитета и мења њихову боју, задржавајући промене док се не изабере друга линија.

Зумирање и померање приказа: Омогућено је зумирање и померање приказа електроенергетске мреже како би се боље истраживали детаљи.

Додатне опције за цртање и текст: На врху прозора корисницима су омогућене опције за додавање облика (елипсе, полигона) и текста на исцртану мрежу. Такође, корисницима је дозвољено мењање сваког нацртаног облика, као и опција за мењање исписаних текстова.

Undo, Redo, Clear: Корисницима су омогућене опције за враћање вишеструких корака уназад, враћање обрисаних објеката или брисање свих објеката са платна.

Интеракција: Омогућене су опције сакривања линија и посебно сваког типа ентитета, као и сакривање само неактивног дела мреже. Корисник има могућност да активира неактиван део и обрнуто. Такође, постоји опција за бирање локације и излиставање фајлова који се могу учитати у апликацију.

Пречице: Корисник има опцију клонирања последњег исцртаног облику комбинацијом дугмића CTRL + D, копија се исцртава на локацији миша. Омогућена је опција чувања тренутног изгледа канваса у виду слике.

ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА

Visual Studio је моћан алат за програмере који се користи да цео развојни циклус пројекта буде завршен на једном месту. То је свеобухватно интегрисано развојно окружење које се користи за писање, уређивање, отклањање грешака и прављење кода, а затим и за примену апликације. Осим уређивања кода и отклањања грешака, Visual Studio укључује компајлере, алате за довршавање кода, контролу извора, екстензије и многе друге функције за побољшање сваке фазе процеса развоја софтвера [1].

C# је један од млађих програмских језика. Настао је 2000. Године као саставни део Microsoft-овог развојног окружења .NET Framework 1.0. Подржава више парадигми, објектно оријентисану, императивну, декларативну, генеричку. Одликују га једноставност, интероперабилност, скалабилност, брзина [2].

WPF је развијен од стране Microsoft-а и представља графички подсистем за рендеровање корисничког интерфејса у апликацијама заснованим на Windows-u. WPF користи XAML, изведен из XMLa, да дефинише и повеже различите UI елементе. WPF апликације могу бити развијене као самостални десктоп програми, или као уграђени у вебсајт. Има за циљ да обједини низ заједничких интерфејс елемената, као што су 2Д, 3Д рендеровање, векторска графика, анимације [3].

XAML (Extensible Application Markup Language) је декларативни језик за описивање структуре и изгледа корисничког интерфејса у апликацијама. Омогућава раздвајање дизајна и логике апликације, олакшавајући развој и подржавајући богате могућности као што су анимације, стилови и контроле. XAML се користи у различитим Microsoft технологијама као што су WPF, UWP и Xamarin за креирање интерактивних и визуелно атрактивних корисничких интерфејса

ОПИС РЕШЕЊА ПРОБЛЕМА

У оквиру овог пројекта, датотека *Geographic.xml* садржи сегмент података који се односи на електроенергетску мрежу у Новом Саду. Ова датотека садржи информације о идентификационим бројевима, именима, као и X и Y координатама напојних трафостаница (*SubstationEntity*) и локалних станица (*NodeEntity*), као и статус прекидача (*SwitchEntity*) (Слика 1).

```
<SwitchEntity>
  <Id>39842</Id>
  <Name>loadbreaker_352187340763</Name>
  <Status>Closed</Status>
  <X>409656.4890883537</X>
  <Y>5012678.4995433623</Y>
</SwitchEntity>
```

Слика 1 – Приказ прекидача у *Geographic.xml* фајлу

Линије (*LineEntity*) (Слика 2) обухватају различите атрибуте. Свака линија има идентификациони број, именовање, информацију о отпорности, материјалу од ког је израђена, обележје да ли је смештена испод земље, тип вода, почетни и завршни чвор које повезује, и индикацију које тачке чине ту линију.

```
<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>
```

Слика 2 – Приказ вода у *Geographic.xml* фајлу

У класи MainWindow.xaml.cs се налазе методе које се користе за обраду података из датотеке *Geographic.xml*. Корисник пре ових метода бира директоријум и фајл за учитавање.

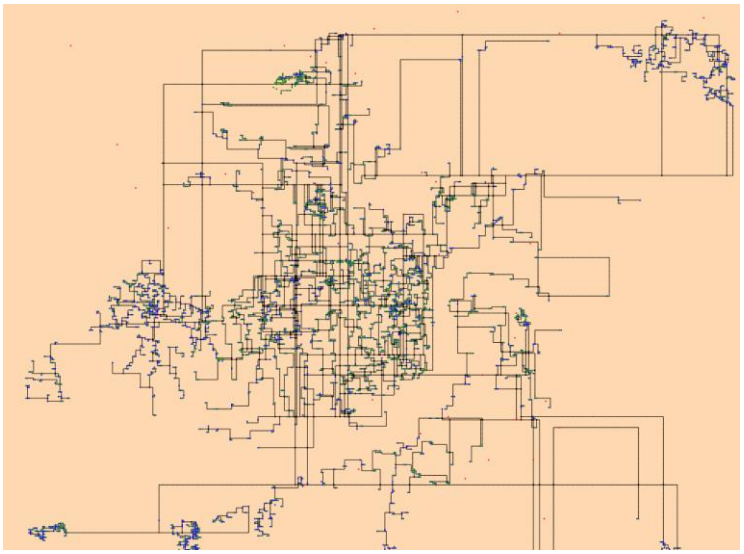
FindScale метода: Ова метода има за задатак да израчуна минималне и максималне вредности координата X и Y из датотеке. Једна ситуација је приказана на слици 3.

```
private void FindScale()
{
    if (substations.Count > 0 && nodes.Count > 0 && switches.Count > 0)
    {
        xMax = Math.Max(Math.Max(substations.Max(item => item.X), nodes.Max(item => item.X)), switches.Max(item => item.X)) + 0.01;
        xMin = Math.Min(Math.Min(substations.Min(item => item.X), nodes.Min(item => item.X)), switches.Min(item => item.X)) - 0.01;
        X = (xMax - xMin) / velicina;
        yMax = Math.Max(Math.Max(substations.Max(item => item.Y), nodes.Max(item => item.Y)), switches.Max(item => item.Y)) + 0.01;
        yMin = Math.Min(Math.Min(substations.Min(item => item.Y), nodes.Min(item => item.Y)), switches.Min(item => item.Y)) - 0.01;
        Y = (yMax - yMin) / velicina;
    }
```

Слика 3 – Један случај скалирања

ReadXML метода: Овој методи је намена да учита податке из различитих секција (Substations, Nodes, Switches, Lines) и на основу тих података креира одговарајуће објекте (SubstationEntity, NodeEntity, SwitchEntity, LineEntity). Након што су подаци учитани из датотеке, следи њихова детаљна обрада.
ToLatLon метода: Ова метода игра кључну улогу у превођењу координата. Након учитавања координата у UTM формату, који је посебан систем координата, користи се математичка функција којом се UTM координате трансформишу у децималне координате. Ове децималне координате се користе за одређивање позиција елемената на мапи. Након обраде, подаци из датотеке морају бити додатно трансформисани коришћењем ToLatLon методе како би се превели у децималне координате, које се потом користе за одређивање позиција на мапи.

Након завршетка обраде ових података, следећи корак је визуализација елемената мапе на корисничком интерфејсу.
Изглед мапе приказан је на слици 4. Додатно, постоје опције за исцртавање геометријских облика као што су елипсе и полигони, као и могућност за испис текста на мапу. Такође, у интерфејсу су доступне опције за "*undo*", "*redo*" и "*clear*", које омогућавају кориснику управљање изменама на мапи.



Слика 4 – Приказ електроенергетске мреже

```
private void Load_Button_Click(object sender, RoutedEventArgs e)
{
    //Load_button.IsEnabled = false;

    if (cmbPutanje.SelectedItem != null && cmbFajlovi.SelectedItem != null)
    {
        ClearData();
        string selectedFolderPath = cmbPutanje.SelectedItem.ToString();
        string selectedFileName = cmbFajlovi.SelectedItem.ToString();
        string completePath = System.IO.Path.Combine(selectedFolderPath, selectedFileName);

        ReadXML(completePath);

        FindScale();
        Draw();
        DrawLine();
    }
    else
    {
        System.Windows.MessageBox.Show("Please select a folder and an XML file.");
    }
}
```

Слика 5 – Дугме за учитавање фајла

Функција приказана на слици 5 реагује на корисников клик на дугме за учитавање. Након што је корисник изабрао фајл, позива претходно поменуте методе за обраду података из фајла, као и методе за исцртавање и додатну методу за чишћење података у случају да је у апликацији већ био коришћен други фајл.

```
private Ellipse DrawElement(double xValue, double yValue, long id, string name, Brush color,string type, string tooltip)
{

    double tmpX = xMax - xValue;
    double tmpY = yMax - yValue;
    int scaledX = (int)(tmpX / X);
    int scaledY = (int)(tmpY / Y);


    Ellipse el = new Ellipse();
    el.Stroke = color;
    el.Fill = color;
    el.Width = 2;
    el.Height = 2;
    el.ToolTip = tooltip;
    el.Name = String.Format("n" + id.ToString());
    if (!velicinaGrida[scaledY, scaledX].SadrziElement)
    {
        velicinaGrida[scaledY, scaledX].SadrziElement = true;
        Canvas.SetLeft(el, scaledY * 2 - 1);
        Canvas.SetTop(el, scaledX * 2 - 1);
        //Console.WriteLine("X: "+scaledY + " Y: "+scaledX);
        if(type is "substation")...
        else if(type is "node")...
        else...
        listaElipsi.Add(el);
        points.Add(id,new Tuple<int,int>(scaledY, scaledX));

        return el;
    }
    else
```

Слика 6 – Исцртавање елемената

DrawElement метода служи за исцртавање графичких елемената који представљају учитане ентитете (Слика 6). Преузима координате од сваког елемента и скалира их на канвас. Креира елипсу на израчунатој локацији коју боји према типу ентитета и додаје јој тултип. Сваки тип ентитета се додаје у одговарајућу листу ради каснијег коришћења. Елементи који се преклапају добијају нову локацију на суседним координатама (Слика 7).

```
else
{
    int brojac = 1;
    while (true)
    {
        if (scaledY - brojac >= 0 && !velicinaGrida[scaledY - brojac, scaledX].SadrziElement) { scaledY -= brojac; break; }
        else if (scaledY + brojac <= 500 && !velicinaGrida[scaledY + brojac, scaledX].SadrziElement) { scaledY += brojac; break; }
        else if (scaledX - brojac >= 0 && !velicinaGrida[scaledY, scaledX - brojac].SadrziElement) { scaledX -= brojac; break; }
        else if (scaledX + brojac <= 500 && !velicinaGrida[scaledY, scaledX + brojac].SadrziElement) { scaledX += brojac; break; }
        else { brojac++; }
    }
    velicinaGrida[scaledY, scaledX].SadrziElement = true;

    Canvas.SetLeft(el, scaledY * 2 - 1);
    Canvas.SetTop(el, scaledX * 2 - 1);
    if(type is "substation")
    {
        substationElipse.Add(el);
    }
    else if(type is "node")
    {
        nodeElipse.Add(el);
    }
    else
    {
        switchElipse.Add(el);
    }
    listaElipsi.Add(el);
    points.Add(id, new Tuple<int, int>(scaledY, scaledX));

    return el;
}
```

Слика 7 – Исцртавање елемената на околној локацији

```
private void DrawLine()
{
    foreach (LineEntity lineEntity in lines)
    {
        if (!points.ContainsKey(lineEntity.FirstEnd) || !points.ContainsKey(lineEntity.SecondEnd))
        {
            continue;
        }
        PozicijaUMatrici pocetnaPozicija = new PozicijaUMatrici()
        {
            Red = points[lineEntity.FirstEnd].Item1,
            Kolona = points[lineEntity.FirstEnd].Item2,
            Roditelj = null
        };
        long krajnjiID = lineEntity.SecondEnd;
        long pocetniID = lineEntity.FirstEnd;

        pocetnaPozicija = BreadthFirstSearch(pocetnaPozicija, krajnjiID);


        while (pocetnaPozicija != null)
        {
            if (pocetnaPozicija.Roditelj == null)
                break;

            int x1 = pocetnaPozicija.Red * 2;
            int y1 = pocetnaPozicija.Kolona * 2;

            int x2 = pocetnaPozicija.Roditelj.Red * 2;
            int y2 = pocetnaPozicija.Roditelj.Kolona * 2;

            bool postojiLinija = false;
            foreach (LinijaNaGridu linijaNaGridu in SveLinije)
            {
```

Слика 8 – Исцртавање линија

DrawLine метода: ова метода служи за исцртавање линија које спајају ентитете. Користи податке од претходно учитаних ентитета вода. Исцртавају се само линије које су спојене са ентитетима на оба краја. Координате сваке линије се додају у матрицу која представља канвас (Слика 10). На основу почетног и крајњег чвора који су спојени водом апроксимира се најкраћа путања употребом BFS алгоритма. Водови су реализовани као спој краћих линија од којих свака има своју позицију у матрици и свака има референцу на претходну. Након што се прође провера да ли нека од линија већ постоји, линије се исцртавају, а на њиховим пресецима се додаје елипса које представља пресечни чвор.

```
foreach (LinijaNaGridu linijaNaGridu in SveLinije)
{
    if (linijaNaGridu.X1 == x1 && linijaNaGridu.Y1 == y1 && linijaNaGridu.X2 == x2 && linijaNaGridu.Y2 == y2) { postojiLinija = true; break; }
    if (linijaNaGridu.X1 == x2 && linijaNaGridu.Y1 == y2 && linijaNaGridu.X2 == x1 && linijaNaGridu.Y2 == y1) { postojiLinija = true; break; }
}
if (!postojiLinija)
{
    Line novaLinija = new Line()
    {
        Stroke = Brushes.Black,
        StrokeThickness = 0.8,
        ToolTip = String.Format("ID: {0} \nName: {1}", lineEntity.Id, lineEntity.Name),
        X1 = x1,
        X2 = x2,
        Y1 = y1,
        Y2 = y2,
    };

    novaLinija.MouseRightButtonDown += LineAnimation;
    SveLinije.Add(new LinijaNaGridu(novaLinija.X1, novaLinija.Y1, novaLinija.X2, novaLinija.Y2, pocetniID, krajnjiID));
    novaLinija.SetValue(TagProperty, lineEntity);
    canvas.Children.Add(novaLinija);
    x1 = pocetnaPozicija.Kolona;
    x2 = pocetnaPozicija.Roditelj.Kolona;
    y1 = pocetnaPozicija.Red;
    y2 = pocetnaPozicija.Roditelj.Red;

    if (!velicinaGrida[y1, x1].SadrziElement && !velicinaGrida[y1, x1].Posecen) { velicinaGrida[y1, x1].Posecen = true; velicinaGrida[y1, x1].PosecenId.Add(lineEntity.
    else if (!velicinaGrida[y1, x1].SadrziElement && velicinaGrida[y1, x1].Posecen && !velicinaGrida[y1, x1].PosecenId.Contains(lineEntity.Id)) { NapraviOblik(lineEnti

    if (!velicinaGrida[y2, x2].SadrziElement && !velicinaGrida[y2, x2].Posecen) { velicinaGrida[y2, x2].Posecen = true; velicinaGrida[y2, x2].PosecenId.Add(lineEntity.
    else if (!velicinaGrida[y2, x2].SadrziElement && velicinaGrida[y2, x2].Posecen && !velicinaGrida[y2, x2].PosecenId.Contains(lineEntity.Id)) { NapraviOblik(lineEnti
}
pocetnaPozicija = pocetnaPozicija.Roditelj;
```

Слика 9 – Исцртавање линија наставак

```
for (int i = 0; i < velicina; i++)
{
    for (int j = 0; j < velicina; j++)
    {
        velicinaGrida[i, j] = new TackaNaGridu();
    }
}
```

Слика 10 – Иницијализација матрице која представља канвас

BreadthFirstSearch метода: проналази накраћу путању између две тачке (Слика 11). У првом пролазу налази путању без пресека са другим линијама, а у другом најкраћу путању са пресеком. Користи претходно поменуту матрицу координата. Алгоритам функционише тако што се од почетне тачке креће до прве суседне у свим правцима и проверава се да ли је суседна тачка крајња тачка. У случају да није поново се помера на суседне тачке које није посетио. Алгоритам се понавља док се не пронађе крајња тачка. Први пут када се та тачка пронађе, то ће значити да је пронађен и најкраћи пут, и функција враћа листу локација у матрици где је потребно исцртати линије. Алгоритам се понавља за сваки вод.

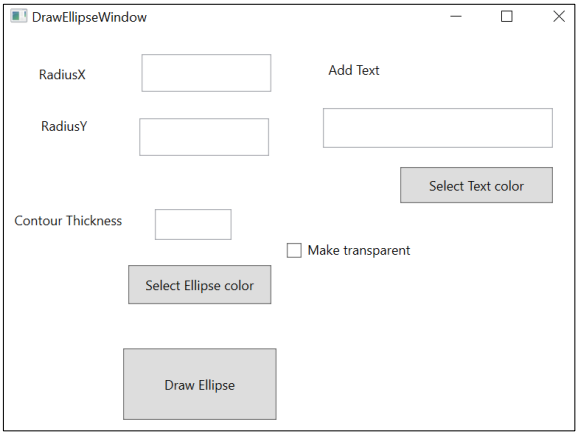
```
private PozicijaUMatrici BreadthFirstSearch(PozicijaUMatrici pocetnaPozicija, long idKrajnjePozicije)
{
    List<PozicijaUMatrici> pozicija = new List<PozicijaUMatrici>();
    pozicija.Add(pocetnaPozicija);
    while (true)
    {
        foreach (var poz in pozicija)
        {
            if (points[idKrajnjePozicije].Item1 == poz.Red && points[idKrajnjePozicije].Item2 == poz.Kolona)
            {
                for (int i = 0; i < velicina; i++)
                {
                    for (int j = 0; j < velicina; j++)
                    {
                        poseceneTacke[i, j] = 0;
                    }
                }
                return poz;
            }
        }
        List<PozicijaUMatrici> listaPozicijaUMatrici = new List<PozicijaUMatrici>();
        foreach (var poz in pozicija)
        {
            if (poz.Red + 1 >= 0 && poz.Kolona >= 0 && poz.Red + 1 < velicina && poz.Kolona < velicina && poseceneTacke[poz.Red + 1, poz.Kolona] == 0)
            {
                poseceneTacke[poz.Red + 1, poz.Kolona] = 1;
                listaPozicijaUMatrici.Add(new PozicijaUMatrici() { Red = poz.Red + 1, Kolona = poz.Kolona, Roditelj = poz });
            }
            if (poz.Red - 1 >= 0 && poz.Kolona >= 0 && poz.Red - 1 < velicina && poz.Kolona < velicina && poseceneTacke[poz.Red - 1, poz.Kolona] == 0)
            {
                poseceneTacke[poz.Red - 1, poz.Kolona] = 1;
                listaPozicijaUMatrici.Add(new PozicijaUMatrici() { Red = poz.Red - 1, Kolona = poz.Kolona, Roditelj = poz });
            }
            if (poz.Red >= 0 && poz.Kolona + 1 >= 0 && poz.Red < velicina && poz.Kolona + 1 < velicina && poseceneTacke[poz.Red, poz.Kolona + 1] == 0)
            {
                poseceneTacke[poz.Red, poz.Kolona + 1] = 1;
                listaPozicijaUMatrici.Add(new PozicijaUMatrici() { Red = poz.Red, Kolona = poz.Kolona + 1, Roditelj = poz });
            }
            if (poz.Red >= 0 && poz.Kolona - 1 >= 0 && poz.Red < velicina && poz.Kolona - 1 < velicina && poseceneTacke[poz.Red, poz.Kolona - 1] == 0)
            {
                poseceneTacke[poz.Red, poz.Kolona - 1] = 1;
                listaPozicijaUMatrici.Add(new PozicijaUMatrici() { Red = poz.Red, Kolona = poz.Kolona - 1, Roditelj = poz });
            }
        }
        pozicija = new List<PozicijaUMatrici>(listaPozicijaUMatrici);
    }
```

Слика 11 – BFS метода за претрагу најкраће путање

Цртање облика и текста на канвасу врши се бирањем једне од три опције и десним кликом миша на жељену локацију (Слика 12). Овако се позива потребан прозор у коме се дефинишу параметри облика или текста (Слика 13).

```
private void mouseRightButtonDown_Canvas(object sender, MouseButtonEventArgs e)
{
    System.Windows.Point point = e.GetPosition((IInputElement)sender);
    if ((bool)Ellipse_RadioButton.IsChecked)
    {
        DrawEllipseWindow drawElipseWindow = new DrawEllipseWindow(point, this);
        drawElipseWindow.Show();
    }
    else if ((bool)Polygon_RadioButton.IsChecked)
    {
        PointsList.Add(point);
    }
    else if ((bool)Text_RadioButton.IsChecked)
    {
        AddTextWindow textWindow = new AddTextWindow(point, this);
        textWindow.Show();
    }
}
```

Слика 12 – Опције за цртање на канвасу



Слика 13 – Прозор за цртање елипсе

Измена нацртаних елемената функционише слично као и њихово цртање. Бирањем опције *Edit* и кликом на нацртани облик добија се исти прозор као при цртању. Разлика је што су одређени параметри (нпр. величина) фиксни и не могу се променити. Параметри који се могу променити су боја, провидност, граница и фонт текста.

HideInactive_Click метода: ова метода сакрива неактиван део мреже. Пошто је сваком исцртаном елементу и линији додељен *TagProperty* у виду свих информација учитаних из фајла, ова метода ће прво пронаћи елементе који су линије, а затим за сваку линију пронаћи прекидач који је на њу повезан и чије је стање *Open*. Након тога ће ту линију додати у листу за брисање и пронаћи сваки ентитет који је на њеном другом крају па и њега додати у засебну листу за брисање. Када се прође кроз све чланове канваса, попуњене листе се користе да уклоне неактивни делови електроенергетске мреже. (Слике 14 и 15)

```
private void HideInactive_Click(object sender, RoutedEventArgs e)
{
    foreach (Shape v in canvas.Children)
    {
        var vod = v.GetValue(TagProperty);
        if(vod is LineEntity)
        {
            LineEntity line = vod as LineEntity;

            foreach (Shape v1 in canvas.Children)
            {
                if (!(v1 is Line))
                {
                    var switchEn = v1.GetValue(TagProperty);
                    if (switchEn is SwitchEntity)
                    {
                        SwitchEntity swE = switchEn as SwitchEntity;
                        if (swE.Status == "Open" && swE.Id == line.FirstEnd)
                        {
                            listaSvihVodovaZaBrisanje.Add((Line)v);

                            foreach (Shape v2 in canvas.Children)
                            {
                                var entitet = v2.GetValue(TagProperty);
                                if (entitet is NodeEntity)
                                {
                                    NodeEntity temp = entitet as NodeEntity;
                                    if (temp.Id == line.SecondEnd)
                                    {
                                        cvoroviZaBrisanje.Add((Ellipse)v2);
                                        break;
                                    }
                                }
                                else if (entitet is SwitchEntity)
                                {
                                    SwitchEntity temp = entitet as SwitchEntity;
                                    if (temp.Id == line.SecondEnd)
                                    {
                                        cvoroviZaBrisanje.Add((Ellipse)v2);
                                        break;
                                    }
                                }
                                else if (entitet is SubstationEntity)
                                {
                                    SubstationEntity temp = entitet as SubstationEntity;
                                    if (temp.Id == line.SecondEnd)
                                    {
                                        cvoroviZaBrisanje.Add((Ellipse)v2);
                                        break;
                                    }
                                }
                            }
                            break;
                        }
                    }
                }
            }
        }
    }
    foreach (Line modelZaBrisanje in listaSvihVodovaZaBrisanje)
    {
        canvas.Children.Remove(modelZaBrisanje);
    }
    foreach(Ellipse ellipse in cvoroviZaBrisanje)
    {
        canvas.Children.Remove(ellipse);
    }
}
```

Слике 14 и 15 – Проналажење и уклањање неактивних делова мреже

ShowInactive_Click metoda: приказује претходно сакривене неактивне делове мреже користећи претходно попуњене листе за брисање.

HideLines_Click, HideSubstations_Click, HideNodes_Click, HideSwitches_Click методе раде на исти начин али за различите ентитете. Проназе све ентитете тог типа на канвасу и уклањају их са канваса. Такође, уклоњене ентитете додају у листе обрисаних које се касније могу користити за поновни приказ тих ентитета.

ShowLines_Click, ShowSubstations_Click, ShowNodes_Click, ShowSwitches_Click методе служе за приказ сакривених делова мреже. Користе претходно поменуте листе обрисаних ентитета. Ако елемент већ постоји на канвасу, неће бити поново додат.

Window_KeyDown метода: ако је корисник претходно нацртао неки облик или текст на канвасу, тај елемент ће бити сачуван као последњи нацртан и његова копија се може цртати преко пречице на тастатури CTRL + D. Функција са слике (Слика 16) проверава да ли постоји претходно исцртан елемент, и ако постоји, клонира га помоћу функције *CloneUIElement* (Слика 17). Клонираном елементу ће бити додељен *event* тако да се може едитовати кликом као и оригинал.

```
private void Window_KeyDown(object sender, System.Windows.Input.KeyEventArgs e)
{
    if (e.Key == System.Windows.Input.Key.D && Keyboard.Modifiers == ModifierKeys.Control)
    {
        System.Windows.Point mousePosition = Mouse.GetPosition(canvas);

        if (lastElement != null)
        {
            Grid grid = new Grid();
            if (isEllipse)
            {
                double height = 0, width = 0;
                foreach (UIElement child in lastElement.Children)
                {
                    if (child is Ellipse)
                    {
                        height = ((Ellipse)child).ActualHeight;
                        width = ((Ellipse)child).ActualWidth;
                    }
                    UIElement clonedChild = CloneUIElement(child);
                    grid.Children.Add(clonedChild);
                }
                foreach(UIElement child in grid.Children)
                {
                    if (child is Ellipse)
                    {
                        Ellipse ellipse = (Ellipse)child;
                        ellipse.Width = width;
                        ellipse.Height = height;
                        grid.Height = height;
                        grid.Width = width;
                        grid.MouseLeftButtonDown += (esender, ee) => EditObjects.UpdateEllipse(esender, ee, this, grid, ellipse);
                    }
                }
                Canvas.SetLeft(grid, mousePosition.X);
                Canvas.SetTop(grid, mousePosition.Y);
                canvas.Children.Add(grid);
            }
```

Слика 16 – Пречица за цртање елемента

```
2 references
private UIElement CloneUIElement(UIElement source)
{
    if (source is FrameworkElement frameworkElement)
    {
        Type elementType = source.GetType();

        FrameworkElement clonedElement = (FrameworkElement)Activator.CreateInstance(elementType);

        foreach (PropertyInfo property in elementType.GetProperties())
        {
            if (property.CanWrite)
            {
                object value = property.GetValue(frameworkElement);
                property.SetValue(clonedElement, value);
            }
        }

        return clonedElement;
    }

    return null;
}
```

Слика 17 – Метода за клонирање било ког елемента канваса

button_SaveCanvasAsImage_Click метода: чува тренутни изглед канваса као *.png* фајл на локацији путање која је подешена на почетку апликације. Као назив слике биће додељен тачан временски тренутак када је слика сачувана.

button_Undo_Click, button_Redo_Click, button_Clear_Click методе: ове методе су унапређене да подржавају целокупну историју додатих објеката. Историја је сачињена од објеката који могу бити елипса, полигон, текст или грид који садржи додатне објекте (Слика 17). Историја се првенствено попуњава при иницијалном креирању сваког објекта.

```
private void button_Undo_Click(object sender, RoutedEventArgs e)
{
    if (UndoRedoPosition > -1)
    {
        if (History[UndoRedoPosition] is Ellipse)
        {
            DependencyObject parent = VisualTreeHelper.GetParent((Ellipse)History[UndoRedoPosition]);
            this.canvas.Children.Remove((Grid)parent);
            UndoRedoPosition--;
        }
        else if (History[UndoRedoPosition] is Polygon)
        {
            DependencyObject parent = VisualTreeHelper.GetParent((Polygon)History[UndoRedoPosition]);
            this.canvas.Children.Remove((Grid)parent);
            UndoRedoPosition--;

        }
        else if (History[UndoRedoPosition] is TextBlock)
        {
            DependencyObject parent = VisualTreeHelper.GetParent((TextBlock)History[UndoRedoPosition]);
            this.canvas.Children.Remove((Grid)parent);
            UndoRedoPosition--;
        }
        else if (History[UndoRedoPosition] is Grid)
        {
            this.canvas.Children.Remove((Grid)History[UndoRedoPosition]);
            UndoRedoPosition--;
        }
    }
    else
    {
        while (History.Count > UndoRedoPosition + 1)
        {
            button_Redo_Click(sender, e);
        }
    }
}
```

Слика 17 – Унапређена *undo* metoda

ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА

- Унапређивање интерфејса: Направити визуелно привлачнији дизајн целокупног интерфејса. Спојити прозоре за цртање различитих елемената у један које ће нудити потребне опције у зависности од типа објекта.

- Статус апликације: Одређене опције у апликацији могу потрајати дуже време, за то време корисник би требао бити обавештен о стању апликације.

- Претрага ентитета: Уместо тренутне претраге свих чланова канваса да би се нашао одређени тип, увести додатне листе или речнике који ће бити подељени по траженим атрибутима ентитета. Овим би се знатно убрзала интеракција са мрежом.

- BFS оптимизација: Алгоритам би могао започети претрагу најкраће путање из почетног и крајњег чвора истовремено. Овим би се скратило време претраге јер би се у истом пролазу кроз листу претраживало из оба смера.

- Клонирање објеката: Тренутно клонирање има посебне случајеве где се познати параметри морају ручно подешавати. Потребно је генерализовати ове случајеве помоћу родитељских објеката као што су грид.

ЛИТЕРАТУРА

[1] *https://visualstudio.microsoft.com/vs/*, ***Visual Studio 2019***
[2] *https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet*, ***C#***
[3] ***Hello World app with WPF in C# - Visual Studio (Windows) | Microsoft Learn***, ***WPF***
[4] ***https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/***, ***Geeks for Geeks***
[5] ***Vladimir C. Strezorski, Osnovi elektroenergetike, 2014, Fakultet tehničkih nauka u Novom Sadu***