



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Никола Сердар, ПР48/2019

2Д ВИЗУЕЛИЗАЦИЈА ШЕМЕ ЕЛЕКТРОДИСТРИБУТИВНЕ МРЕЖЕ

ПРОЈЕКАТ

- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 12.12.2023

САДРЖАЈ

1. ОПИС РЕШАВАНОГ ПРОБЛЕМА
2. ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА
3. ОПИС РЕШЕЊА ПРОБЛЕМА
4. ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА
5. ЛИТЕРАТУРА

ОПИС РЕШАВАНОГ ПРОБЛЕМА

Главни циљ пројекта јесте решавање изазова које су везане за визуализацију, апроксимацију и приказ електроенергетске мреже употребом графичких елемената у 2D формату.

Обезбеђују се могућности попут уношења облика и текста, приказа мреже, имплементирање undo/redo функционалности, додавање слика на чворове мреже као и чишћење свих претходних измена како би се електроенергетска мрежа вратила у првобитно стање.

Полигон, елипса представљају могући избор облика којим корисник може да манипулише. То укључује одређивање њихове величине, боје, транспарентности, дебљине ивица, као и писање текста унутар њих.

У рачунарској графичкој, електроенергетска мрежа подразумева графички приказ различитих водова, чворова и постојећим постројења у систему електроенергетске мреже. Ова визуализација помаже инжењерима и оператерима да брже и лакше разумеју њену структуру и компоненте. [1]

Као последица тога, пројекат нуди кориснику опције као што су померање слајдера не би ли повећао односно смањивао приказ мреже, приказивање/сакривање активне мреже, бојење ентитета по броју конекција, бојење линија по отпорности, бојење анимације као и одређивање временског опсега трајања анимације да би могли што ефикасније да прате ток енергије, оптерећење, напоне и друге параметре. [2]

Поред испуњавања визуелних функционалности, битно је нагласити решавања проблема перформанси при исцртавању електричне мреже. За ту потребу је примењен BFS алгоритам са мањим изменама. Главни разлог одабира BFS алгоритма у поређењу са сличним начинима претраге графа (Depth First Search, Dijkstra) лежи у чињеници да је бржи у проналажењу најкраће путање од почетне до крајње тачке у непондерисаном графу, где је од интереса да се пронађе одговор на питање: “Да ли су чворови (у овом случају ентитети), повезани или не”. [8]

ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА

Технологије и алати који су коришћени са израду пројекта су:

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

Microsoft Visual Studio - јесте IDE (Integrated Development Environment) тј. окружење које обезбеђује интеграцију различитих алатки за развој и тестирање софтвера. Поддржава више програмских језика као што су C#, C++, Visual Basic, Python и други. Такође има улогу у развоју за различите платформе, односно десктоп, веб, мобилне апликације. [3]

C# - представља објектно-оријентисани програмски језик који се користи за развој различитих апликација. Пример јесу веб апликације(где се C# користи уз ASP.NET за развој динамичких сајтова). Поддржава основне концепте објектно-оријентисаног програмирања као што су класе, објекти, наслеђивање и полиморфизам. [9]

WPF - Windows Presentation Foundation је графички фрејмворк који омогућава креирање богатих и интерактивних корисничких интерфејса. Интерфејси се дефинише помоћу XAML-а, што обезбеђује раздвајање између дизајна и логике апликације. [5]

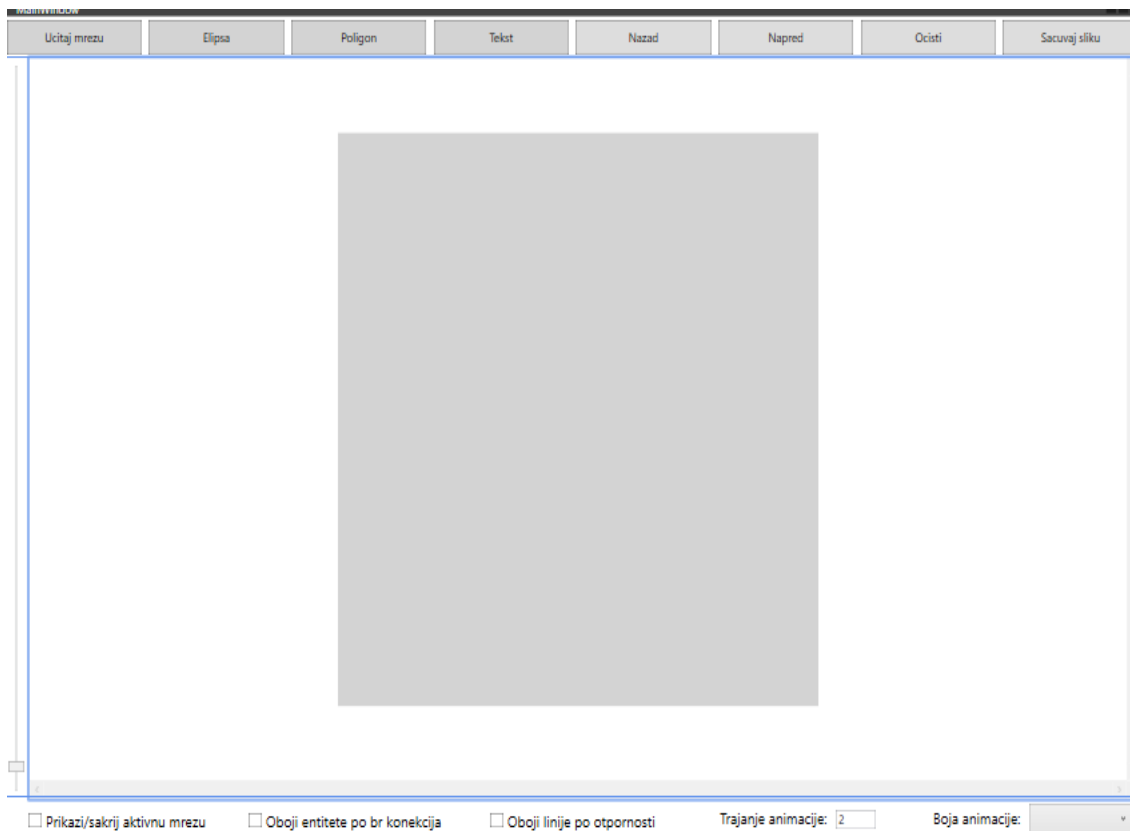
XAML (Extensible Application Markup Language) - је декларативан језик који се користи за развој корисничког интерфејса.

Он подржава развој апликација заснованих на догађајима.

Такође се користи за дефинисање графичких елемената и анимација на корисничком интерфејсу. Велика предност XAML-а је та што се лако интегрише са кодом тј. Омогућава развојницима и дизајнерима да сарађују при чему је дизајнерска логика у XAML-у, а програмска логика у коду. [10]

ОПИС РЕШЕЊА ПРОБЛЕМА

Након покретања пројекта, корисник се сусреће са следећим прозором. (Слика 3.1)



. Слика 3.1 Почетни прозор

“Учитај мрежу” представља дугме која позива функцију **Load_Click**.

Она прима два параметара и за циљ има да формира матрицу 300x300 након чега позива следеће три функције: **ParseXml**, **LoadXML**, **FormMatrix** и **FormLines**. (Слика 3.2)

```
private void Load_Click(object sender, RoutedEventArgs e)
{
    for (int i = 0; i < 300; i++)
    {
        for (int j = 0; j < 300; j++)
        {
            matrica[i, j] = false;
        }
    }

    xmlEntities = ParseXml();
    LoadXML();
    FormMatrix();
    FormLines();
}
```

Слика 3.2 Почетни прозор

Прва функција **ParseXml** служи да пронађе путању и учита Geographic.xml. Пре самог учитавања одвија се провера да ли фајл постоји. У случају да не постоји, функција враћа null. У супротном извршава се учитавање.

Geographic.xml јесте структура електроенергетске мреже. Садржи информације о ентитетима и водовима унутар мреже као и детаљне податке који описују шему електроенергетске мреже у Новом Саду. У њему се налазе листе о нодовима, субстанцијама и свичевима.

Пошто су сви неопходни подаци које требамо да прикажемо записани у UTM (Универзални Трансверзални Меркаторов) формату, користи се функција ToLatLon која претвара UTM у децималне вредности географске ширине и дужине. (Слика 3.3)


```

public void FormMatrix()
{
    int newX;
    int newY;
    List<Tuple<int, int>> reservedPosition = new List<Tuple<int, int>>();

    double firstX = listaPowerEntities.Min(xx => xx.X);
    double firstY = listaPowerEntities.Min(yy => yy.Y);
    double lastX = listaPowerEntities.Max(xx => xx.X);
    double lastY = listaPowerEntities.Max(yy => yy.Y);

    foreach (var element in listaPowerEntities)
    {
        newY = (int)((299 * (element.Y - firstY)) / (lastY - firstY));
        newX = (int)((299 * (element.X - firstX)) / (lastX - firstX));

        int korak = 1;
        bool validPositionFound = false;

        while (!validPositionFound)
        {
            reservedPosition.Clear();

            reservedPosition.Add(new Tuple<int, int>(newX + korak, newY + korak));
            reservedPosition.Add(new Tuple<int, int>(newX + korak, newY - korak));
            reservedPosition.Add(new Tuple<int, int>(newX - korak, newY + korak));
            reservedPosition.Add(new Tuple<int, int>(newX - korak, newY - korak));

            for (int i = korak - 1; i > -korak; i--)
            {
                reservedPosition.Add(new Tuple<int, int>(newX + korak, newY + i));
                reservedPosition.Add(new Tuple<int, int>(newX + i, newY - korak));
                reservedPosition.Add(new Tuple<int, int>(newX - korak, newY + i));
                reservedPosition.Add(new Tuple<int, int>(newX + i, newY + korak));
            }
        }
    }
}

```

Слика 3.4 *FormMatrix*

У случају да није, креира се објекат класе **Ellipse**, пошто су чворови који се мапирају елипсоидног облика. Иницијализују се поља везана за њихову висину и ширину. После, се врши провера о типу чвора. Постоје три такве провере за субстанице, node и свичеве где се и додају у SubstationEntities, NodeEntities и SwitchEntities. За свичеве се врши провера о статусу активности(Слика 3.5).


```

foreach (var position in reservedPosition)
{
    if (position.Item1 >= 0 && position.Item1 < 300 && position.Item2 >= 0 && position.Item2 < 300)
    {
        if (matrica[position.Item1, position.Item2] == false)
        {
            matrica[position.Item1, position.Item2] = true;

            Ellipse ellipse = new Ellipse();
            ellipse.Width = 2;
            ellipse.Height = 2;
            ellipse.ToolTip = element.ToolTip;
            ellipse.MouseLeftButtonDown += Entity_MouseLeftButtonDown;
            ellipse.MouseRightButtonDown += Entity_MouseRightButtonDown;

            if (element.GetType() == typeof(SubstationEntity))
            {
                ellipse.Fill = System.Windows.Media.Brushes.Blue;
                ellipse.Tag = $"substation";
                SubstationEntities.Add(ellipse);
                OtherEntites.Add(ellipse);
            }
            else if (element.GetType() == typeof(NodeEntity))
            {
                ellipse.Fill = System.Windows.Media.Brushes.Crimson;
                ellipse.Tag = $"node";
                NodeEntities.Add(ellipse);
                OtherEntites.Add(ellipse);
            }
            else if (element.GetType() == typeof(SwitchEntity))
            {
                ellipse.Fill = System.Windows.Media.Brushes.Green;
                ellipse.Tag = $"switch";
                SwitchEntities.Add(ellipse);

                if (((SwitchEntity)element).Status == "Open")
                {
                    OpenSwitches.Add(ellipse);
                }
                else
                {
                    ClosedSwitches.Add(ellipse);
                }
            }
        }
    }
}

```

Слика 3.5 Прављење чворова и одређивање типа

За додатну функционалност врши се провера о броју конекција. (Слика 3.6)

```

switch (element.NumberOfConnections)
{
    // 0-3
    case 0:
    case 1:
    case 2:
        EntitiesWith03Connections.Add(ellipse);
        break;
    // 3-5
    case 3:
    case 4:
        EntitiesWith35Connections.Add(ellipse);
        break;
    // 5+
    default:
        EntitiesWithMoreThan5Connections.Add(ellipse);
        break;
}

```

Слика 3.6 Одређивање број конекција чвора

Пролази се на крају кроз функцију **FormLines** не би ли се унети чворови повезали и формирали крајњу електроенергетску мрежу. Повезивање се постиже употребом **BFS (Breadth-First Search)** алгоритма.

Алгоритам за цртање водова пролази кроз сваку линију из листе водова, која је добијена из Geographic.xml фајла, представљајући везу између ентитета и повезујући почетну и крајњу тачку.

Први задатак је одређивање почетних и крајњих тачака. Након итерације кроз листу линија, координате се иницијализују. Затим се пролази кроз листу скалираних ентитета, прослеђују x и y координате почетним и крајњим ентитетима. У случају да су x и y координате негативне, ентитети не могу бити скалирани на канвасу, те се прескаче на следећу итерацију. На самом крају одређивања позиције ентитета позива се BFS алгоритам. (Слика 3.7а) (Слика 3.7б)

```

1 reference
private void FormLines()
{
    foreach (LineEntity line in listaLines)
    {
        double firstX = -1;
        double firstY = -1;
        double secondX = -1;
        double secondY = -1;

        foreach (var ent in listaScaledPowerEntities)
        {
            if (line.FirstEnd == ent.Id)
            {
                firstX = ent.X;
                firstY = ent.Y;
            }
            if (line.SecondEnd == ent.Id)
            {
                secondX = ent.X;
                secondY = ent.Y;
            }
        }

        if (firstX == -1 || firstY == -1 || secondX == -1 || secondY == -1)
        {
            continue;
        }

        line.PocetakX = firstX;
        line.PocetakY = firstY;
        line.KrajX = secondX;
        line.KrajY = secondY;

        System.Windows.Point start = new System.Windows.Point(firstX, firstY);
        System.Windows.Point end = new System.Windows.Point(secondX, secondY);

        BFSPassage1(line, start, end);
    }
}

```

Слика 3.7а Први пролазак

```

foreach (LineEntity line in listaLines)
{
    double firstX = -1;
    double firstY = -1;
    double secondX = -1;
    double secondY = -1;

    foreach (var ent in listaScaledPowerEntities)
    {
        if (line.FirstEnd == ent.Id)
        {
            firstX = ent.X;
            firstY = ent.Y;
        }
        if (line.SecondEnd == ent.Id)
        {
            secondX = ent.X;
            secondY = ent.Y;
        }
    }

    if (firstX == -1 || firstY == -1 || secondX == -1 || secondY == -1)
    {
        continue;
    }

    System.Windows.Point start = new System.Windows.Point(firstX, firstY);
    System.Windows.Point end = new System.Windows.Point(secondX, secondY);

    if (line.Prolaz != "1")
    {
        BFSPassage2(line, start, end);
    }

    line.Prolaz = "0";
}
iscrtaoPresek = true;

```

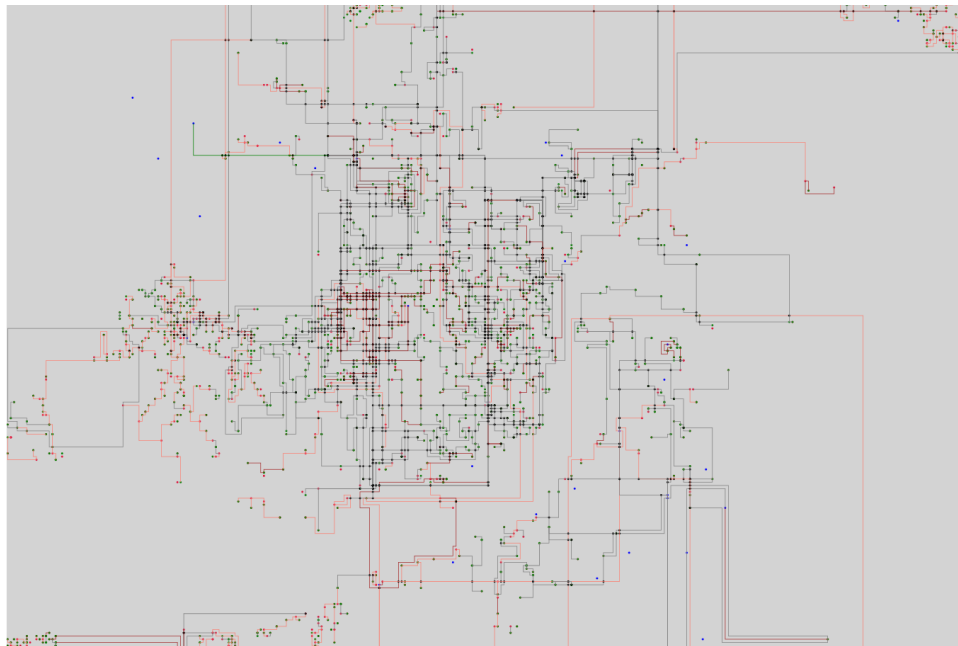
Слика 3.76 Други пролазак

Смисао првог BFS проласка јесте пронаћи путању од почетне до крајње тачке, избегавајући пресеке између њих. Проверава се да ли је пролазак једнак нули. У случају да јесте, ради се о једном, изолованом ентитету. У супротном, претрага почиње тако што се прво поље поставља за почетно. Затим, за свако од суседних поља, врши се провера валидности. Проверава се да ли се поље налази унутар задатих димензија матрице (300x300) и да ли није већ посећено. Процес се наставља док се не стигне до поља које представља крај вода. Тачке које нису обрађене додају се у ред и третирају се као почетна поља. Кроз ову итерацију, BFS алгоритам тражи и бележи путању између почетне и крајње тачке вода, осигуравајући да путања не пресече друге путање.

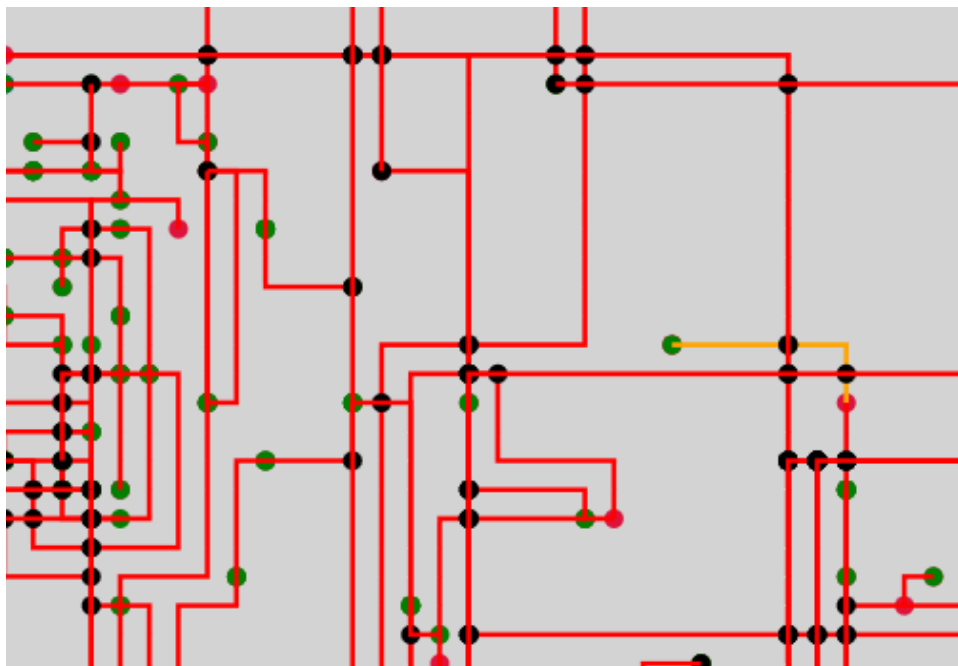
Други BFS пролазак подразумева укључивање пресечених водова, који се обележавају елипсама. Имплементација друге путање BFS-а идентична је првој, с изузетком провере на пресецима и

њиховог локализовања. Пресечени ентитети се третирају као почетна поља, одакле се поново тражи путања до крајњег ентитета.

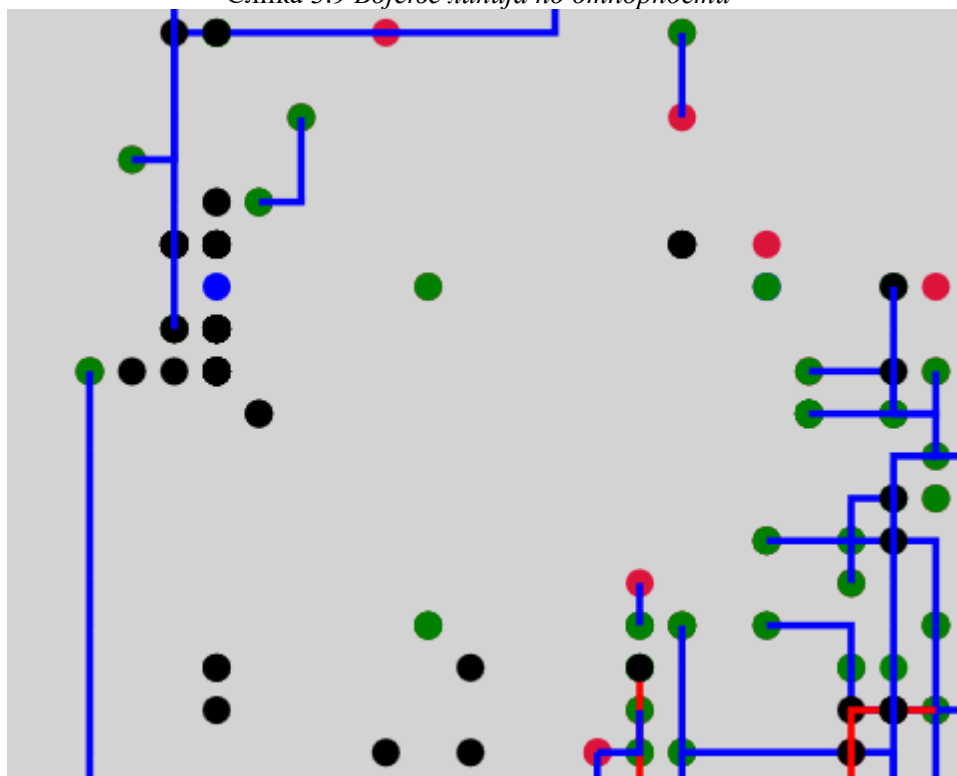
Након ових корака добија се повезана електроенергетска мрежа. (Слика 3.8)



Слика 3.8 *Електроенергетска мрежа*

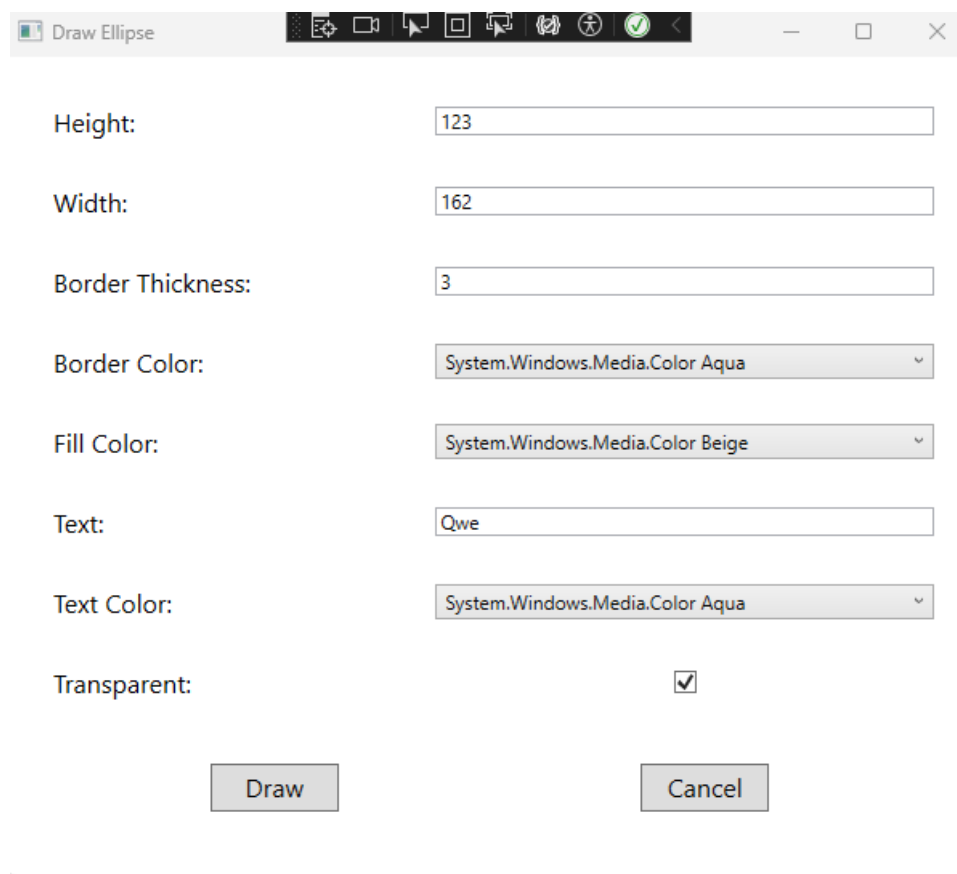


Слика 3.9 Бојење линија по отпорности



Слика 3.10 Приказ/Сакривање активне мреже

Левим кликом на дугме “Елипса” затим десним на канвас отвара се прозор за скицирање елипсе.
(Слика 3.11)



Draw Ellipse

Height: 123

Width: 162

Border Thickness: 3

Border Color: System.Windows.Media.Color Aqua

Fill Color: System.Windows.Media.Color Beige

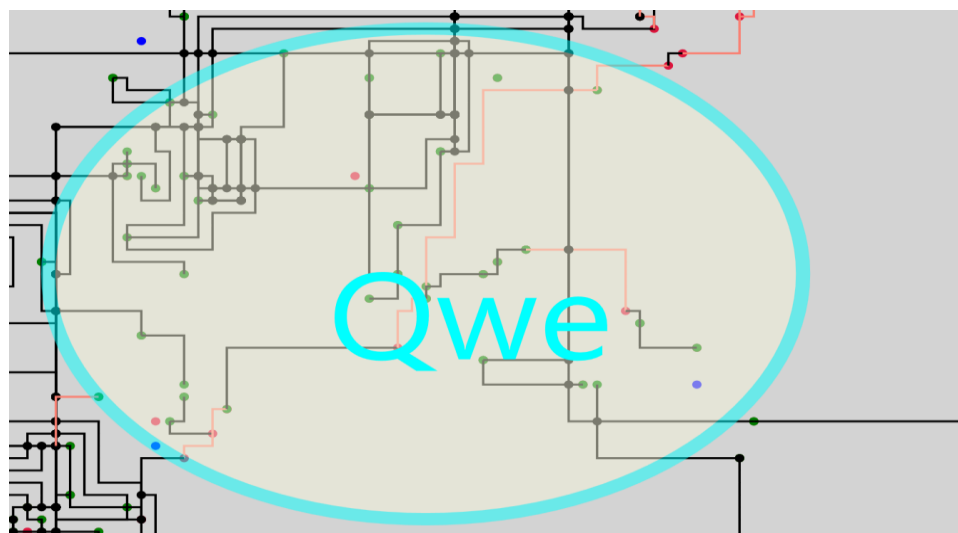
Text: Qwe

Text Color: System.Windows.Media.Color Aqua

Transparent: ☒

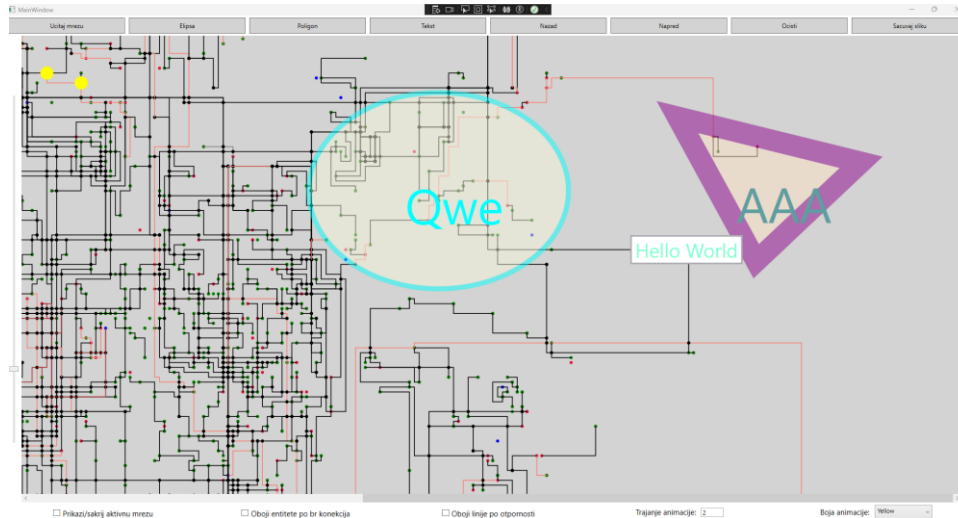
Draw Cancel

Слика 3.11 Прозор за креирање Елипсе



Слика 3.12 Креирана елипса

Слично елипси левим кликом на дугме “Полигон” креира се полигон. Главна разлика јесте што је неопходно да се на канвасу кликне минимум три пута десним кликом да би се одабрале три или више тачака, пошто је троугао најједноставнији полигон. Када су се тачке одабрале, левим кликом на канвас се отвара прозор за његово креирање.



Слика 3.13 Креирана елипса, полигон и текст, као и анимација чвора

На послетку присутне су опције **Clear**, **Undo** и **Redo**.

Clear брише све нацртане елипсе, полигоне и текст.

Undo поништава последње промене на канвасу тј. исцртавање последњег облика или текста, док **Redo** враћа претходни обрисан облик или текст.


```

private void Undo_Click(object sender, RoutedEventArgs e)
{
    if (WasCleared)
    {
        foreach (UIElement element in UndoHistory)
        {
            if (ShapeTextPairs.ContainsKey(element))
            {
                canvas.Children.Add(ShapeTextPairs[element]);
            }
            canvas.Children.Add(element);
            CanvasObjects.Add(element);
        }

        UndoHistory.Clear();
        WasCleared = false;

        return;
    }

    if (canvas.Children.Count > 0)
    {
        var element = canvas.Children[canvas.Children.Count - 1];
        UndoHistory.Add(element);
        CanvasObjects.Remove(element);
        canvas.Children.Remove(element);
        if (ShapeTextPairs.ContainsKey(element))
        {
            canvas.Children.Remove(ShapeTextPairs[element]);
        }
    }
}

```

Слика 3.14 *Имплементација Undo*

```

private void Redo_Click(object sender, RoutedEventArgs e)
{
    if (UndoHistory.Count > 0 && WasCleared == false)
    {
        var element = UndoHistory[UndoHistory.Count - 1];
        if (ShapeTextPairs.ContainsKey(element))
        {
            canvas.Children.Add(ShapeTextPairs[element]);
        }
        canvas.Children.Add(element);
        CanvasObjects.Add(element);
        UndoHistory.RemoveAt(UndoHistory.Count - 1);
    }
}

1 reference
private void Clear_Click(object sender, RoutedEventArgs e)
{
    UndoHistory.Clear();

    foreach (UIElement element in CanvasObjects)
    {
        UndoHistory.Add(element);
        canvas.Children.Remove(element);
        if (ShapeTextPairs.ContainsKey(element))
        {
            canvas.Children.Remove(ShapeTextPairs[element]);
        }
    }

    CanvasObjects.Clear();
    WasCleared = true;
}

```

Слика 3.15 *Имплементација Redo и Clear*

ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА

Израдом овог пројекта успешно је постигнута 2D визуелизација шеме електроенергетске мреже са додатним опцијама за исцртавањем облика, текста као и могућност брисања и враћања мреже у првобитно стање.

Захваљујући додатним опцијама за бојање водова на основу отпорности, броју конекција, сакривање активне мреже као и увођење анимације за селектоване чворове, и њихово трајање, значајно је побољшано искуство корисника приликом њеног приказа и манипулисања.

Осим тога, увођењем BFS алгоритма, осигран је проналазак најоптималнијег решења у колико их има више. Велика предност овог алгоритма је та што се неће извршавати заувек у корисној путањи тако да је време значајно уштеђено.

Међутим постоје неколико предлога за побољшање пројекта:

1. Имплементација тестирања апликације како би се исправиле тј. уклониле грешке
2. Постоје други алгоритми претраге (DFS) које могу бити боље у погледу брзине извршавања
3. Привремено уклањање неактивних чворова мреже ради убрзавања процеса претраге
4. Унапређивање анимације чворова

ЛИТЕРАТУРА

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

[2] **Wikipedia, 2D računarska grafika, 2021,**
https://www.wikiwand.com/sh/2D_ra%C4%8Dunarska_grafika

[3] **Wikipedia, Microsoft Visual Studio, 2022, [Microsoft Visual Studio — Википедија \(wikipedia.org\)](#)**

[5] **Wikipedia, C#, 2022, [Windows Presentation Foundation — Википедија \(wikipedia.org\)](#)**

[7] **Youtube, William Fiset, Breadth First Search Algorithm, April 2018, [YouTube](#)**

[8] **Shortest distance between two cells in a matrix of grid, Jun 2022, [Breadth First Search Algorithm | Shortest Path | Graph Theory - YouTube](#)**

[9] **What is C#: Definitions, Strengths & Usages, 2022,**
<https://www.designveloper.com/blog/what-is-c-sharp/>

[10] <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-8.0>