



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ
НАУКА У НОВОМ САДУ



Јована Лажетић, ПР111/2019

2Д РАЧУНАРСКА ГРАФИКА
BFS

ПРОЈЕКАТ

- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 31.8.2023.

САДРЖАЈ

1. ОПИС РЈЕШАВАНОГ ПРОБЛЕМА.....	3
2. ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА.....	4
3. ОПИС РЈЕШЕЊА ПРОБЛЕМА.....	5
4. ПРИЈЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА.....	17
5. ЛИТЕРАТУРА.....	18

ОПИС РЈЕШАВАНОГ ПРОБЛЕМА

Овај пројектни задатак има за циљ визуализацију електроенергетске мреже на основу података из "*Geographic.xml*" датотеке, користећи ортогонални приказ. Нагласак је на ефикасној примјени и оптимизацији **BFS** алгоритма за исцртавање и повезивање различитих компоненти мреже.

Кључни аспекти овог пројекта укључују:

Визуализација мреже: Електроенергетска мрежа се визуализује користећи ортогонални приказ, тежећи вјерном приказу података из "*Geographic.xml*" датотеке.

Приказ ентитета: Елементи мреже (*Substation*, *Node*, *Switch*) се позиционирају на најближи слободан простор, избјегавајући преклапање. Корисне информације о сваком ентитету приказују се преласком миша.

Повезивање водова: Водови који повезују ентитете приказују се као праве линије, стартујући из средишта сваког ентитета.

BFS алгоритам: За исцртавање водова примењује се **BFS** алгоритам који је оптимизован како би вријеме исцртавања мреже било мање од 15 секунди. Ово укључује проналажење најкраћег пута између ентитета без пресјека са већ постојећим водовима, а у случају немогућности, водови се исцртавају са пресјецима.

Анимација: Десним кликом на вод покреће се анимација која приказује згушњавање графичких елемената повезаних ентитета и промјену боје. Анимација траје док се не изабере други вод.

Зумирање и помјерање: Могуће је зумирати и помјерити приказ мреже како би се детаљније испитали детаљи.

Додатне опције: Корисници имају опцију додавања облика (елипсе, полигона) и текста на мрежу. Исто тако, могуће је мијењати облике и текст накнадно.

Поништавање, поновно враћање и брисање: Корисницима је омогућено поништавање последње акције, враћање обрисаних објеката или чишћење цјелокупног приказа.

Оваква апликација омогућава брзо и ефикасно исцртавање електроенергетске мреже, оптимизовано користећи **BFS** алгоритам, са могућношћу интеракције и истраживања детаља и адаптације приказа.

ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА

При разради овог задатка коришћени су следећи алати и технологије:

C# : Током израде пројектног задатка коришћен је C#, модеран и лако разумљив програмски језик, за запис важеће логике пројекта. C# је објектно оријентисан језик и припада .NET окружењу. Користи се за развој разноврсних апликација, укључујући веб, десктоп и мобилне апликације. Пружа програмерима богат сет алата за развој, укључујући напредне могућности управљања меморијом, асинхронно програмирање и обраду грешака.

.NET : Платформа .NET садржи разноврстан скуп компоненти и библиотека које обезбеђују основе за развој различитих апликација, као и њихово управљање и извршавање на различитим оперативним системима. Подржава библиотеке за различите програмске језике и обезбеђује алатке за обраду података, управљање базама података и управљање грешкама.

WPF: (*Windows Presentation Foundation*) је графичка технологија развијена од стране *Microsoft-a* за креирање богатих и интерактивних корисничких интерфејса у апликацијама. Она користи XAML (*Extensible Application Markup Language*) за декларативно дефинисање изгледа и структуре интерфејса, а омогућава и интеграцију са логиком апликације путем кода. WPF пружа могућности као што су анимације, стилови, контроле и могућност дизајнирања апликација које раде на више различитих уређаја и резолуција.

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.

Visual Studio 2019: За програмирање и развој пројекта коришћен је *Visual Studio 2019*. Ово интегрисано радно окружење обезбеђује алатке за писање, тестирање, дебаговање и оптимизацију софтвера. Такође, омогућава означавање верзија, анализу кода, креирање корисничких интерфејса и ефикасно управљање пројектима, чиме олакшава и убрзава процес развоја софтвера.

Ове технологије и алати су помогли у стварању стабилне и функционалне апликације која задовољава потребе пројектног задатка.

ОПИС РЈЕШЕЊА ПРОБЛЕМА

Циљ овог пројекта је надоградња почетног задатка у којем се врши визуализација графа електроенергетске мреже Новог Сада. Нагласак је на оптимизацији и резултатима *BFS (Breadth-First Search)* алгоритма.

Програма на основу информација из датотеке "*Geographic.xml*" исцртава граф мреже у ортогоналном приказу. Површина за цртање је подијељена на замишљене "подиоке", што омогућава детаљнији приказ.

Ентитети мреже (*Substation, Node, Switch*) приказују се као графички облици на најближем слободном подиоку, избјегавајући преклапање. Информације о сваком ентитету доступне су преко *ToolTip-a*.

Водови који повезују ентитете приказују се као праве линије које потичу из центра ентитета. Путање водова рачунају се коришћењем *BFS* алгоритма како би се избјегло пресјецање већ постојећих водова.

Десним кликом на вод покреће се анимација која увећава приказ повезаних ентитета и мијења њихову боју, како би се истакли повезани елементи.

Приказ мреже омогућава зумирање и помјерање погледа, док корисници могу додавати облике (елипсе, полигоне) и текст на мрежу, као и мијењати их.

Додатне опције укључују могућност поништавања (*Undo*), поновног извршавања (*Redo*) и брисања свих објеката (*Clear*) са платна.

Суштина алгоритама је оптимизација времена исцртавања и постизање мапе у оквиру 15 секунди за различите сценарије и димензије платна.

```
<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>
```

Дати *XML* код (Слика 1) представља информације о једном воду (линији) у електроенергетској мрежи. Елементи као што су идентификациони број, име, подземност, резистенција, материјал проводника, тип линије и термална константа карактеришу вод, док тачке координата *X* и *Y* описују врхове који одређују путању линије на мрежи.

Слика 1 – Приказ вода

"SwitchEntity" класа представља ентитет прекидача (свича) у контексту електроенергетске мреже. Ова класа наслеђује другу класу, "PowerEntity", која садржи основне атрибуте и функционалности за ентитете у електроенергетској мрежи.

У овој класи постоји својство "Status" које описује статус свича, односно његово тренутно стање. Свичеви су у суштини прекидачи који омогућавају контролисање протока електричне струје у мрежи, па је статус битан податак да би се знало да ли је свич укључен или искључен. Ова информација је важна за прецизну контролу и управљање мрежом, како би се осигурало равнотежно и ефикасно распоређивање електричне енергије.

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

Слика 2 – Приказ прекидача

Овај *XML* код (Слика 2) описује основне атрибуте и позицију свича у контексту електроенергетске мреже, укључујући његов идентификациони број, име, статус (отворен/затворен) и координате позиције на површини.

Пројекат садржи класу NodeEntity која описује ентитет *node* (Слика 3) у електроенергетској мрежи. Ова класа наслеђује основну класу PowerEntity и садржи атрибуте *Id* и *Name* за идентификацију и име, као и координате *X* и *Y* за његову позицију на мрежи. Цјелокупна информација о *node* се представља у *XML* облику, гдје се *Id* одређује именом *"Id"*, *Name* одређује именом *"Name"*, *X* одређује именом *"X"*, а *Y* одређује именом *"Y"*.

```
<NodeEntity>
  <Id>36305</Id>
  <Name>BUSNODE_86905008121</Name>
  <X>406794.287305533</X>
  <Y>5011422.29316307</Y>
</NodeEntity>
```

Слика 3 – Приказ локалне трафостанице у *Geographic.xml*

```
<SubstationEntity>
  <Id>144396663052602591</Id>
  <Name>EMS_TSH_17_2</Name>
  <X>414825.46321875858</X>
  <Y>5006154.1329433313</Y>
</SubstationEntity>
```

Слика 4 – Приказ подстанции

Пројекат такође укључује класу *SubstationEntity* која представља ентитет подстанции у електроенергетској мрежи. Ова класа наслеђује основну класу *PowerEntity* и садржи атрибуте *Id* и *Name* за идентификацију и име подстанце, као и координате *X* и *Y* које означавају њену позицију на мрежи. Опис ентитета подстанции дат је у *XML* формату (Слика 4) у коме се *Id* одређује тагом "*Id*", *Name* одређује тагом "*Name*", *X* одређује тагом "*X*", а *Y* одређује тагом "*Y*".

```

4 references
public class ImportHelper
{
    2 references
    public static void FindMinMaxCoordinates()...

    2 references
    public static void ImportData() ...

    3 references
    static void ToLatLon(double utmX, double utmY, int zoneUTM, out double latitude, out double longitude)...
}

```

Слика 5 – Класа *ImportHelper*

Класа *ImportHelper* (Слика 5) садржи методе за учитавање и конверзију географских података из *XML* формата у интерни формат модела података који се користи у апликацији. Ове методе се користе за обраду и прилагођавање географских података прије него што буду уписани у интерну базу података или кориснички интерфејс.

FindMinMaxCoordinates() метода има задатак да пронађе минималне и максималне координате тачака на *canvasu* како би се израчунали одговарајући *offseti*. То је корисно за приказивање података на графици тако да сви подаци буду видљиви на екрану. Метода прво издваја *X* и *Y* координате свих ентитета из колекције *PowerEntities*, а затим рачуна минималне и максималне вриједности за обе координате. На крају, израчунава *offsete* за *X* и *Y* координате *canvasa*.

ImportData() је метода која врши учитавање података из *XML* фајла "*Geographic.xml*" и конвертује их у интерне моделе ентитета. Сваки тип ентитета (*Substations*, *Nodes*, *Switches*, *Lines*) има своје чворове у *XML* документу. Метода пролази кроз сваку групу ентитета у *XML-у*, чита њихове атрибуте и вриједности и креира одговарајуће инстанце модела ентитета у складу са тим подацима. Након конверзије, модели ентитета се додају у колекцију *PowerEntities* за даљу обраду или упис у базу података.

ToLation() је помоћна метода за конверзију координата из *UTM* формата у географски формат (*latitude i longitude*). Метода се користи за трансформацију координата у одговарајући формат како би се подаци тачно приказали на мапи или графици. Метода узима *UTM* координате, зону *UTM*, и израчунава географске координате (*latitude i longitude*) уз помоћ математичких формула.

Ове методе раде заједно како би омогућиле конверзију и прилагођавање географских података према потребама апликације, као и да омогуће тачно приказивање тих података на корисничком интерфејсу.

Након што су подаци обрађени, следћи корак је приказивање елемената мапе на платну (Слика 6).

Undo
Redo
Clear
Render Model
Render Model without Nodes
Add Ellipse
Add Polygon
Add Text
Dimensions 200x200

Корисницима се пружа могућност да визуализују мапу са или без елемената типа “*Node*”, уз избор димензија платна: 200x200, 750x750, и 1500x1500 пиксела. Осим тога, омогућено је и исцртавање геометријских облика попут елипси и полигона, као и додавање текстуалних ознака на мапу. Поред тога, корисници могу користити опције “*Undo*”, “*Redo*” и “*Clear*” како би управљали претходним акцијама и стањем мапе.

Слика 6 – Опције корисника

```
private void MapSize_Click(object sender, RoutedEventArgs e)
{
    switch (CanvasHelper.Size)
    {
        case 200:
            CanvasHelper.Size = 750;
            break;
        case 750:
            CanvasHelper.Size = 1500;
            break;
        case 1500:
            CanvasHelper.Size = 200;
            break;
        default:
            CanvasHelper.Size = 200;
            break;
    }

    MapSize.Content = $"Map Size ({CanvasHelper.Size}x{CanvasHelper.Size})";
    MainCanvas.Children.Clear();

    MainCanvas.Width = MainCanvas.Height = 2 * CanvasHelper.Size;
    CanvasHelper.Move = 2;

    AlgorithmHelper.map = new bool[CanvasHelper.Size, CanvasHelper.Size];
}
```

Слика 7 – Приказ методе *MapSize_Click*

Ова метода (Слика 7) реагује на клик на дугме за промјену величине мапе. У зависности од тренутне величине мапе, она прелази на следећу величину из списка: 200x200, 750x750, 1500x1500. Ако се величина мапе не налази на овом списку, подразумејевана величина поставља се на 200x200. Након промјене величине, ажурира се приказ на дугмету са новом величином мапе, бришу се сви елементи са главног платна и ажурирају се димензије главног платна и помоћних

промјенљивих. На крају, алоцира се нови низ булових вриједности за мапу са новом величином.

```
private void LoadModel_Click(object sender, RoutedEventArgs e)
{
    MainCanvas.Children.Clear();
    CanvasHelper.RenderNodes = true;
    CanvasRendererHelper.AppMainWindow = this;
    ImportHelper.ImportData();
    ImportHelper.FindMinMaxCoordinates();
    CanvasRendererHelper.DisplayPowerEntitiesOnCanvas();
    CanvasRendererHelper.DisplayLinesOnCanvas();

    Entities.Lines.Clear();
    Entities.PowerEntities.Clear();
}
```

Слика 8 – Метода за учитавање мапе

```
private void LoadModelNoNodes_Click(object sender, RoutedEventArgs e)
{
    MainCanvas.Children.Clear();
    CanvasHelper.RenderNodes = false;
    CanvasRendererHelper.AppMainWindow = this;
    ImportHelper.ImportData();
    ImportHelper.FindMinMaxCoordinates();
    CanvasRendererHelper.DisplayPowerEntitiesOnCanvas();
    CanvasRendererHelper.DisplayLinesOnCanvas();

    Entities.Lines.Clear();
    Entities.PowerEntities.Clear();
}
```

Слика 9 – Метода за учитавање мапе без нодова

Метода *LoadModel_Click* (Слика 8) се позива када се кликне на дугме "*LoadModel*". Она извршава низ операција које укључују брисање свих постојећих елемената на главном канвасу, подешавање вриједности за приказивање елемената, увоз података, приказивање ентитета *power* и линија на канвасу, као и брисање постојећих линија и ентитета *power* из одговарајућих листи.

Садржај ове методе се користи за управљање приказом и увозом података у апликацију.

LoadModelNoNodes_Click (Слика 9) метода има сличну структуру као и претходна, *LoadModel_Click*, са разликом у томе што она поставља вриједност за приказивање елемената на "*false*", чиме искључује приказивање нодова на *canvasu*.

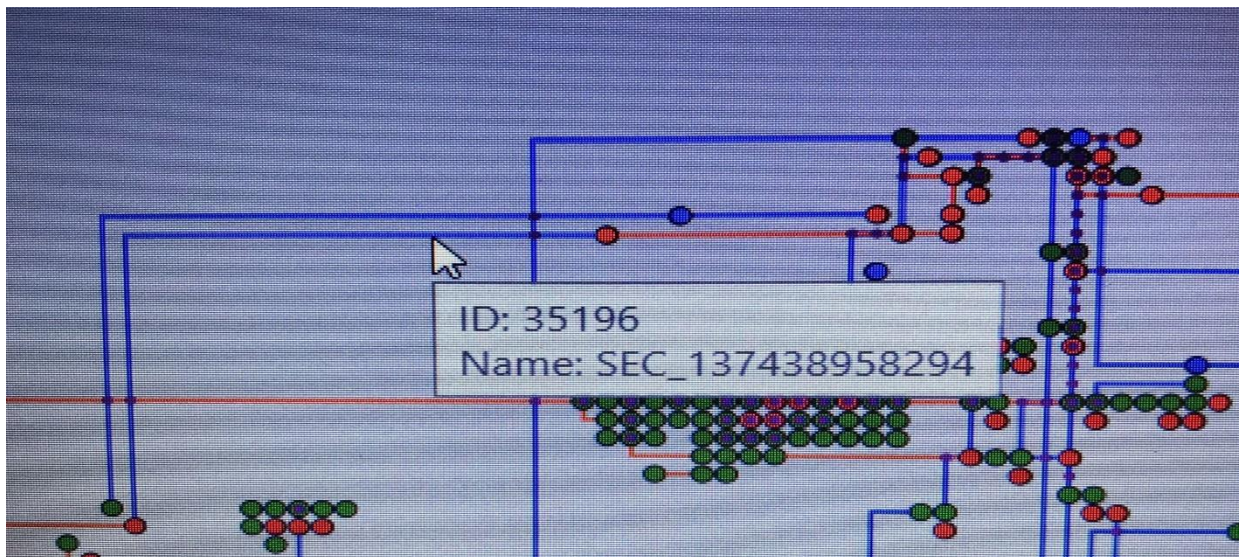
```
private static void DetermineAndDrawPath(LineEntity line)
{
    if (DrawnLines.Any(t => (t.Item1 == line.FirstEnd && t.Item2 == line.SecondEnd)
        || (t.Item2 == line.FirstEnd && t.Item1 == line.SecondEnd)))
        return;
    DrawnLines.Add((line.FirstEnd, line.SecondEnd));

    var start = positionCoordinates[line.FirstEnd];
    var end = positionCoordinates[line.SecondEnd];

    List<(int, int)> linePoints = AlgorithmHelper.FindFirstLinePoints(start, end);
    if (linePoints == null)
    {
        pendingLines.Add(line);
        return;
    }
    RenderPathOnCanvas(line, linePoints);
}
```

Слика 10 – Метода за одређивање и приказивање путање

Ова метода (Слика 10) се користи за одређивање и приказивање путање између два крајња појединачна чвора, на графу (Слика 11). Метода провјерава да ли је путања већ нацртана између датих чворова (крајњих тачака), и ако није, додаје те чворове као нацртану путању. Затим се приступа добијању координата почетног и крајњег чвора и користе алгоритми да се нађу тачке које чине путању између тих чворова. Уколико путања не може да се нађе, путања се додаје на листу чекања. Ако је путања успјешно нађена, користи се функција "RenderPathOnCanvas" да се путања прикаже на *canvasu*.



Слика 11 – Приказ *Tool* типа

```
private static List<(int, int)> GetAdjacentNeighbors((int, int) pos, (int, int) end)
{
    List<(int, int)> neighbors = new List<(int, int)>();

    if (pos.Item1 > end.Item1 && pos.Item1 > 0)
        neighbors.Add((pos.Item1 - 1, pos.Item2));

    if (pos.Item2 > end.Item2 && pos.Item2 > 0)
        neighbors.Add((pos.Item1, pos.Item2 - 1));

    if (pos.Item1 < end.Item1 && pos.Item1 < CanvasHelper.Size - 1)
        neighbors.Add((pos.Item1 + 1, pos.Item2));

    if (pos.Item2 < end.Item2 && pos.Item2 < CanvasHelper.Size - 1)
        neighbors.Add((pos.Item1, pos.Item2 + 1));

    return neighbors;
}
```

Слика 12 – Метода за добијање сусједних чворова

Ова метода, GetAdjacentNeighbors, се користи за добијање листе суседних тачака у односу на дату почетну тачку *pos* и крајњу тачку *end*. Ова метода помаже да се одреде сусједне тачке за тренутну тачку на мапи. У зависности од положаја почетне тачке у односу на крајњу тачку, додају се одговарајуће тачке као сусједне у листу. На крају, метода враћа листу сусједних тачака које се послије користе у алгоритму претраге и у дијеловима кода гдје је потребно разматрати околне тачке.

```

public static List<(int, int)> FindFirstLinePoints((int, int) start, (int, int) end)
{
    bool[,] visited = new bool[CanvasHelper.Size, CanvasHelper.Size];
    queue.Enqueue(new List<(int, int)>() { start });

    while (queue.Count > 0)
    {
        var path = queue.Dequeue();
        var last = path.Last();

        if (visited[last.Item1, last.Item2])
            continue;

        visited[last.Item1, last.Item2] = true;

        var neighbors = GetAdjacentNeighbors(last, end);
        for(int i = 0; i < neighbors.Count; i++)
        {
            if (neighbors[i] == end)
            {
                path.Add(neighbors[i]);
                return FilterAndFillMappedPath(path);
            }

            if (!visited[neighbors[i].Item1, neighbors[i].Item2] && !map[neighbors[i].Item1, neighbors[i].Item2])
            {
                List<(int, int)> newPath = new List<(int, int)>(path);
                newPath.Add(neighbors[i]);
                queue.Enqueue(newPath);
            }
        }
    }

    ClearQueue();
    return null;
}

```

Слика 13 – Метода за проналажење прве путање између тачака на мапи

Овај дио кода (Слика 13) представља методу *FindFirstLinePoints*, која претрагом у ширину (*BFS*) пролази кроз мрежу тачака у потрази за првом путањом између двије задате тачке, *start* (почетне) и *end* (крајње). Путања се налази тражењем путање кроз сусједне тачке користећи ред чекања. Ако путања није пронађена, метода враћа *null*.

```

public static (List<(int, int)>, List<(int, int)>) FindSecondLinePointsWithMarks((int, int) start, (int, int) end)
{
    bool[,] visited = new bool[CanvasHelper.Size, CanvasHelper.Size];
    queue.Enqueue(new List<(int, int)>() { start });

    while (queue.Count > 0)
    {
        var path = queue.Dequeue();
        var last = path.Last();

        if (visited[last.Item1, last.Item2])
            continue;

        visited[last.Item1, last.Item2] = true;

        var neighbors = GetAdjacentNeighbors(last, end);
        for (int i = 0; i < neighbors.Count; i++)
        {
            if (neighbors[i] == end)
            {
                path.Add(neighbors[i]);
                var marks = ExtractMarkedPoints(path);
                return (marks, FilterAndFillMappedPath(path));
            }

            if (!visited[neighbors[i].Item1, neighbors[i].Item2])
            {
                List<(int, int)> newPath = new List<(int, int)>(path);
                newPath.Add(neighbors[i]);
                queue.Enqueue(newPath);
            }
        }
    }

    ClearQueue();
    return (null, null);
}

```

Слика 14 - Метода за проналажење друге путање између тачака на мапи

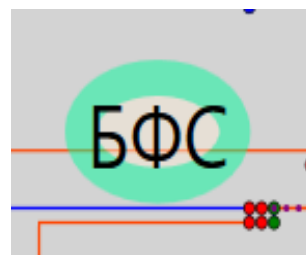
Ова метода (Слика 14) користи алгоритам претраге у ширину (*BFS*) за проналазак друге путање између почетне и крајње тачке на мапи. Истовремено, она издваја означене тачке на путањи користећи методу *ExtractMarkedPoints*, и враћа те издвојене тачке заједно са путањом. Ако путања није пронађена, метода враћа (*null, null*).

```
private void drawEllipseButton_Click(object sender, RoutedEventArgs e)
{
    if (AssociatedEllipse != null)
    {
        AssociatedEllipse.Fill = new SolidColorBrush(fillColor.SelectedColor ?? Colors.Black);
        AssociatedEllipse.Stroke = new SolidColorBrush(strokeColor.SelectedColor ?? Colors.Black);
        AssociatedEllipse.Width = double.Parse(radiusX.Text);
        AssociatedEllipse.Height = double.Parse(radiusY.Text);
        AssociatedEllipse.StrokeThickness = double.Parse(strokeThickness.Text);
        if ((bool)ellipseIsTransparent.IsChecked) AssociatedEllipse.Opacity = 0.5;
        else AssociatedEllipse.Opacity = 1;
        AssociatedTextBlock.Text = textBox.Text;
        AssociatedTextBlock.Foreground = new SolidColorBrush(textColor.SelectedColor ?? Colors.Black);
        Close();
        return;
    }

    if (fillColor.SelectedColor == null || strokeColor.SelectedColor == null || radiusX == null || radiusY == null
        || strokeThickness == null)
        return;

    try
    {
        FillColor = new SolidColorBrush(fillColor.SelectedColor ?? Colors.Black);
        StrokeColor = new SolidColorBrush(strokeColor.SelectedColor ?? Colors.Black);
        Width = double.Parse(radiusX.Text);
        Height = double.Parse(radiusY.Text);
        StrokeThickness = double.Parse(strokeThickness.Text);
        if ((bool)ellipseIsTransparent.IsChecked) OpacityValue = 0.5;
        else OpacityValue = 1;
        TextContent = textBox.Text;
        TextColor = new SolidColorBrush(textColor.SelectedColor ?? Colors.Black);
    }
    catch (Exception)
    {
        MessageBox.Show("Please fill out all fields");
        return;
    }
    Close();
}
```

Слика 15 – Креирање елипсе



Слика 16 - Приказ елипсе

Овај код (Слика 15) представља обраду догађаја када се кликне на дугме *drawEllipseButton*. Метода провјерава да ли постоји везана елипса (*AssociatedEllipse*). Ако постоји, ажурира податке о елипси на основу корисничког уноса. Ако не постоји, прима кориснички унос и креира нову елипсу са датим подацима. Ако постоји *AssociatedEllipse*:

Ажурира боју унутрашњости, ивице, величину, дебљину ивице, провидност и текст елипсе.

Затим ажурира текстовни блок (*AssociatedTextBlock*) са текстом из текстуалног поља (*textBox*). Ако не постоји *AssociatedEllipse*:

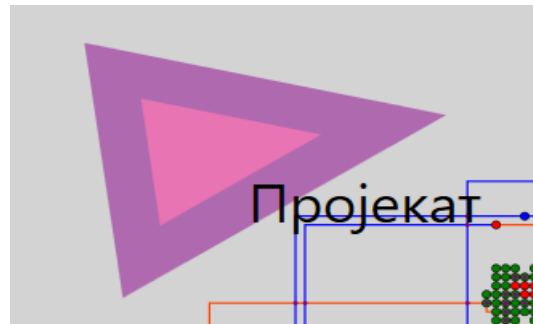
Провјерава да ли су изабране боје, величина и дебљина ивице. Податке узима из текстуалних поља и креира нову елипсу. Ако је омогућена провидност (*ellipseIsTransparent*), подешава вриједност провидности на 0.5, иначе на 1. Подешава текст, боју текста и остале податке. У случају грешке, приказује поруку о недостатку података. Ова метода се користи за додавање или ажурирање података о елипси и приказује је на корисничком интерфејсу (Слика 16).


```
private void drawPolygonButton_Click(object sender, RoutedEventArgs e)
{
    if (AssociatedPolygon != null)
    {
        AssociatedPolygon.Fill = new SolidColorBrush(fillColor.SelectedColor ?? Colors.Black);
        AssociatedPolygon.Stroke = new SolidColorBrush(strokeColor.SelectedColor ?? Colors.Black);
        AssociatedPolygon.StrokeThickness = double.Parse(strokeThickness.Text);
        if ((bool)polygonIsTransparent.IsChecked) AssociatedPolygon.Opacity = 0.5;
        else AssociatedPolygon.Opacity = 1;
        AssociatedTextBlock.Text = textBox.Text;
        AssociatedTextBlock.Foreground = new SolidColorBrush(textColor.SelectedColor ?? Colors.Black);
        Close();
        return;
    }

    if (fillColor.SelectedColor == null || strokeColor.SelectedColor == null || strokeThickness == null)
        return;

    try
    {
        FillColor = new SolidColorBrush(fillColor.SelectedColor ?? Colors.Black);
        StrokeColor = new SolidColorBrush(strokeColor.SelectedColor ?? Colors.Black);
        StrokeThickness = double.Parse(strokeThickness.Text);
        if ((bool)polygonIsTransparent.IsChecked) OpacityValue = 0.5;
        else OpacityValue = 1;
        TextContent = textBox.Text;
        TextColor = new SolidColorBrush(textColor.SelectedColor ?? Colors.Black);
    }
    catch (Exception)
    {
        MessageBox.Show("Please fill out all fields");
        return;
    }
    Close();
}
```

Слика 17 – Цртање полигона

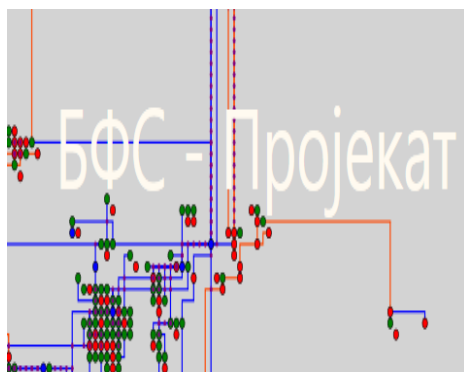


Слика 18 – Приказ полигона

Део кода (Слика 17) представља обраду догађаја када се кликне на дугме *drawPolygonButton*, које се користи за цртање многоугла (полигона). Метода провјерава да ли постоји везани полигон (*AssociatedPolygon*). Ако постоји, ажурира податке о полигону на основу корисничког уноса. Ако не постоји, прима кориснички унос и креира нови полигон са датим подацима. Ако постоји *AssociatedPolygon*:

Ажурира боју унутрашњости, ивице, дебљину ивице, провидност и текст полигона. Затим ажурира текстовни блок (*AssociatedTextBlock*) са текстом из текстуалног поља (*textBox*). Ако не постоји *AssociatedPolygon*:

Провјерава да ли су изабране боје и дебљина ивице. Податке узима из текстуалних поља и креира нови полигон. Ако је омогућена провидност (*polygonIsTransparent*), подешава вриједност провидности на 0.5, иначе на 1. Подешава текст, боју текста и остале податке. У случају грешке, приказује поруку о недостатку података. Ова метода се користи за додавање или ажурирање података о полигону и приказује га на корисничком интерфејсу (Слика 18).



Слика 19 – Приказ текста

```
private void addText_Click(object sender, RoutedEventArgs e)
{
    if (AssociatedTextBlock != null)
    {
        AssociatedTextBlock.Text = textBox.Text;
        AssociatedTextBlock.Foreground = new SolidColorBrush(textColor.SelectedColor ?? Colors.Black);
        AssociatedTextBlock.FontSize = double.Parse(textSize.Text);
        Close();
        return;
    }

    if (string.IsNullOrEmpty(textBox.Text) || textColor.SelectedColor == null || textSize.Text == null)
        return;

    try
    {
        TextContent = textBox.Text;
        TextColor = new SolidColorBrush(textColor.SelectedColor ?? Colors.Black);
        TextSize = double.Parse(textSize.Text);
        if (TextSize < 0 || string.IsNullOrEmpty(TextContent))
            throw new Exception();
    }
    catch (Exception)
    {
        MessageBox.Show("Please fill out all fields");
        return;
    }
    Close();
}
```

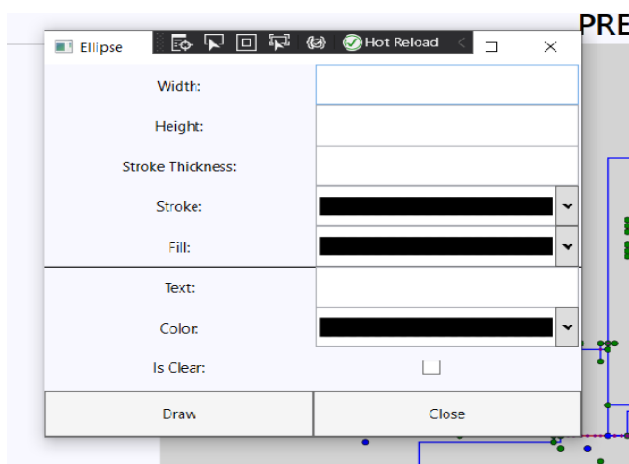
Слика 20 – Метода за приказ текста

Приказана метода (Слика 20) представља обраду догађаја када се кликне на дугме *addText*, које се користи за додавање текста на кориснички интерфејс. Метода провјерава да ли постоји везани текстовни блок (*AssociatedTextBlock*). Ако постоји, ажурира текстовни блок на основу корисничког уноса. Ако не постоји, прима кориснички унос и креира нови текстовни блок са датим подацима. Ако постоји *AssociatedTextBlock*:

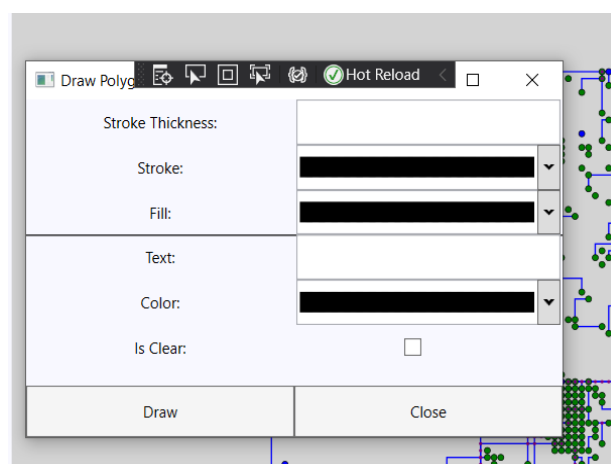
Ажурира текст и боју текста у текстовном блоку. Ажурира величину фонта текста на основу унијете вриједности (*textSize.Text*). Ако не постоји *AssociatedTextBlock*:

Провјерава да ли је унијет текст, боја текста и величина фонта. Мијења вриједност величине фонта из текстуалног поља у тип *double*. Податке мијења и креира нови текстовни блок. Провјерава да ли је величина фонта негативна или је текст празан. У случају грешке, приказује поруку о недостатку података.

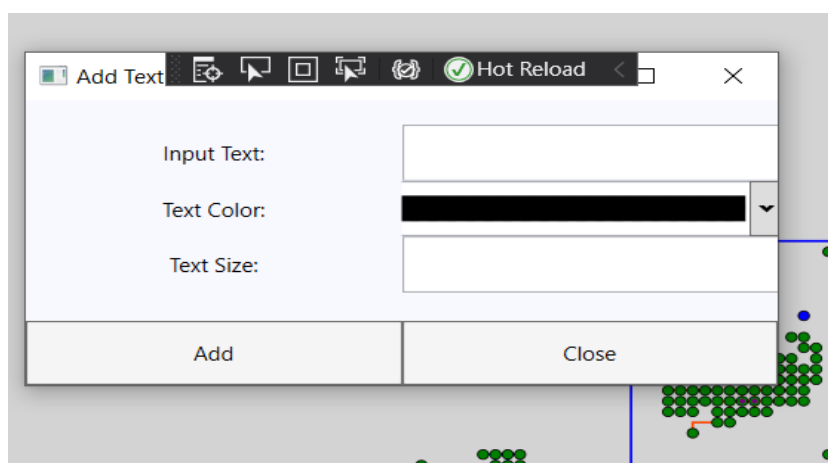
Ова метода се користи за додавање или ажурирање текста и приказује је на корисничком интерфејсу.



Слика 21 – Прозор за цртање елипсе



Слика 22 – Прозор за цртање полигона



Слика 23 – Прозор за унос текста

```

private void Undo_Click(object sender, RoutedEventArgs e)
{
    if (wasCleared)
    {
        ClearList.ForEach(t => t.Redo());
        UndoList.AddRange(ClearList);
        ClearList.Clear();

        wasCleared = false;
    }

    else if (UndoList.Count > 0)
    {
        var command = UndoList.Last();
        UndoList.Remove(command);
        command.Undo();
        RedoList.Add(command);
    }

    UpdateHistoryButtonAvailability();
}

1 reference
private void Redo_Click(object sender, RoutedEventArgs e)
{
    if (RedoList.Count > 0)
    {
        var command = RedoList.Last();
        RedoList.Remove(command);
        command.Redo();
        UndoList.Add(command);
    }

    UpdateHistoryButtonAvailability();
}

1 reference
private void Clear_Click(object sender, RoutedEventArgs e)
{
    ClearList.AddRange(UndoList);
    UndoList.ForEach(t => t.Undo());
    UndoList.Clear();

    wasCleared = true;
    UpdateHistoryButtonAvailability();
}

```

Слика 24 – Методе Undo, Redo и Clear

Ове методе (Слика 24) приказују обраду догађаја за разне акције које су повезане са праћењем и додавањем/опозивањем акција у корисничком интерфејсу.

Undo (Поништи): Операција која отказује последњу извршену акцију.

Redo (Поново враћање): Операција која понавља акцију која је раније отказана.

Clear (Очисти): Операција која уклања све акције из историје.

Undo Click:

Ако је претходно извршено брисање акција (*Clear*), враћају се опозване акције назад на стање прије брисања. Ако постоје опозване акције у *UndoList*, извлачи последњу акцију из листе. Опозива последњу акцију (користећи методу **Undo()**) која је дефинисана за ту акцију).

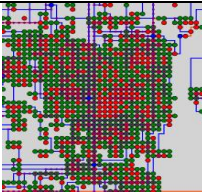
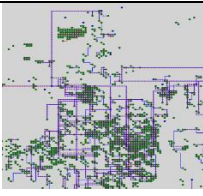
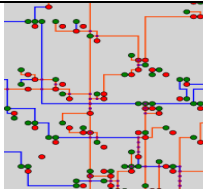
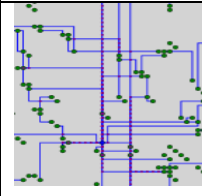
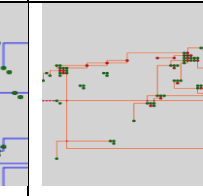
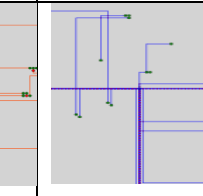
Додаје опозвану акцију у *RedoList* за случај да корисник жели да поново изврши ту акцију. Ажурира доступност дугмади историје.

Redo Click:

Ако постоје акције за поново извршавање у **RedoList**, извлачи посљедњу акцију из листе. Поново извршава акцију (користећи методу **Redo()** која је дефинисана за ту акцију). Додаје извршену акцију у **UndoList** за случај да корисник жели да опозове ту акцију. Ажурира доступност дугмади историје.

Clear Click:

Додаје све акције из **UndoList** у **ClearList**. Опозива све акције из **UndoList** (користећи методу **Undo()** за сваку акцију). Брише све акције из **UndoList**. Означава да је брисање извршено (узимајући у обзир да се операције могу вратити назад). Ажурира доступност дугмади историје. Овај код омогућава кориснику да контролише историју акција, поништава и поново извршава акције, као и да врши брисање акција. Такође, методе **UpdateHistoryButtonAvailability()** се користе да би се ажурирала доступност дугмади у зависности од стања историје акција.

200x200 Render Model	200x200 Render Model Without Nodes	750x750 Render Model	750x750 Render Model Without Nodes	1500x1500 Render Model	1500x1500 Render Model Without Nodes
					
10,81 s	1,62 s	13,5 s	3,82 s	14,47 s	12,10 s

Табела 1 – Времена учитавања мапе за дату величину *canvasa*

На основу анализе свих претходно објашњених метода, корисницима је омогућено да учитавају мапу на више начина. Сваки од ових начина учитавања је оптимизован тако да пружи што брже вријеме учитавања мапе, са циљем да корисницима буду осигурана боља и ефикаснија корисничка искуства. Вријеме учитавања је мање од 15 секунди (Табела 1). Сваки од ових начина учитавања има своје предности и пружа брзе резултате, што корисницима омогућава да брзо и лако прегледају мапу у зависности од својих потреба. Истовремено, сав код је дизајниран да буде ефикасан и да обезбиједи оптимално корисничко искуство у складу са задатим временским ограничењем од 15 секунди.

ПРИЈЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА

У овом пројектном задатку успјешно је извршена оптимизација рада *BFS* алгоритма, резултујући брзим исцртавањем мапе. Опције за додавање различитих облика и текста омогућило је корисницима да креативно прилагођавају мапу. Поред тога, функционалности поништавања и враћања претходних акција доприносе бољем контролисању и управљању измјенама.

Додатне оптимизације на *BFS* алгоритму могу се имплементирати са циљем да се вријеме исцртавања смањи, доприносећи бржем и учинковитијем исцртавању мреже. Ово би побољшало корисничко искуство, осигуравајући да апликација одговара на захтјеве корисника за брзим и ефикасним исцртавањем.

Чување исцртаних мапа у различитим форматима би омогућило корисницима да их касније прегледају или дијеле са другима. Ова опција доприноси бољем организовању и дијељењу података, чиме се омогућава већа практичност у коришћењу апликације.

Имплементација *Drag-and-Drop* функционалности би значајно обогатила корисничко искуство. Ова могућност омогућава лако додавање нових облика и текста на мрежу, при чему корисници могу ефикасно манипулисати садржајем мреже према својим жељама.

Корисници могу имати потпуну слободу прилагођавања апликације према својим потребама. Опције као што су избор језика, промјена изгледа и конфигурација поставки за складиштење омогућило би корисницима да апликацију учине угодном и функционалном у складу са њиховим представама и захтјевима.

Ове имплементације и опције могу допринијети једноставном, функционалном и задовољавајућем корисничком искуству.

ЛИТЕРАТУРА

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

[2] **WIKIPEDIA**

[3] <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>

[4] [Hello World app with WPF in C# - Visual Studio \(Windows\) | Microsoft Learn](#)

[5] [HTTPS://WWW.GEEKSFORGEES.ORG/TYPES-OF-POLYGONS/](https://www.geeksforgeeks.org/types-of-polygons/)

[6] <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

[7] <https://www.w3schools.com/cs/index.php>