KOCAELİ ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BLM PROGRAMLAMA LAB. II
PROJE I

GEZGİN ROBOT PROJESİ

Ahmet Göçmen
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
ahmetgocmeen@outlook.com

I. ÖZET

Bu dökumantasyon Kocaeli Üniversitesi Programlama Laboratuvarı II dersinin I. projesinin çözümüne yönelik hazırlanmıştır. Bu dökumanda projenin tanıtımı, çözümü için yapılan araştırmalar, proje hazırlanırken kullanılan yöntemler,U ML diyagramı, kod ve algoritma bilgisi, proje hazırlanırken kullanılan geliştirme ortamı gibi oluşturduğumuz programı açıklayan alt başlıklara yer verilmiştir. Proje boyunca yararlandığımız kaynaklar dökümanın en sonunda yer almaktadır.

II. PROJE TANITIMI

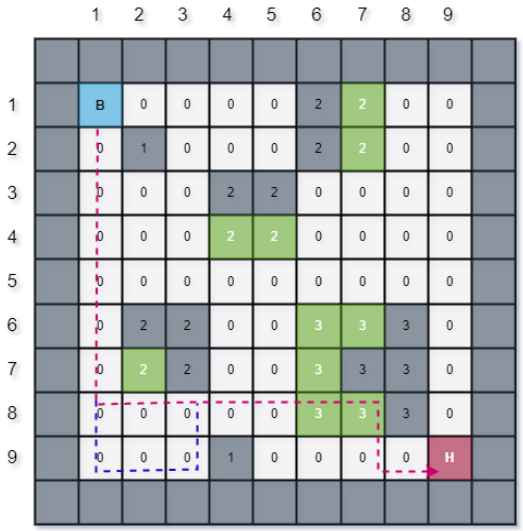
Bu projede bizden belirli kurallara göre hareket eden bir robotun önündeki engelleri aşarak istenen hedefe ulaşmasını sağlayan bir oyun tasarlanması beklenmektedir. Oyunda iki adet problemin çözülmesi gerekmektedir. Problemlerin çözümü için nesneye yönelik programlama ve veri yapıları bilgilerinin kullanılması beklenmektedir. Bu projede amaç proje gerçekleştirimi ile öğrencilerin nesneye yönelik programlama ve veri yapıları bilgisinin pekiştirilmesi ve problem çözme becerisinin gelişimi amaçlamaktadır. Projede aşağıda tanımlanan iki probleme çözüm bulmamız beklenmektedir.

Problem 1:

Bu problemde bizden robotu ızgara (grid) üzerinde verilen hedefe engellere takılmadan en kısa sürede ve en kısa yoldan ulaştırmamız beklenmektedir. Robot tüm ızgarayı gezerek değil, yalnızca gerekli yolları gezerek hedefe ulaşmasını sağlamamız gerekmektedir.

Problemde grid oluşturabilmek için URL adresinden .txt dosyası okunması gerekmektedir. Bu yapıldıktan sonra uygun boyutlarda karesel bir grid oluşturulmalıdır. Daha sonra oluşturulan grid üzerine okunan dosyada yerleri ve türleri verilen engeller yerleştirilmelidir. Okunan dosyadaki 1,2,3 değerleri üç farklı türdeki engelleri temsil etmektedir. Bu engeller birbirinden farklı karesel alan temsil etmektedir. 1 değeri 1x1 karelik alanı, 2 değeri 2x2 karelik alan ve 3 değeri 3x3 karelik alan kaplayan engelleri ifade etmektedir.

Robot başlangıçta tüm ızgarayı bilmemekte sadece ona komşu olan gidebileceği yerleri bilmektedir. Bunlardan hareketle robotun hedefe ulaşabileceği en kısa yol adım adım grid üzerinde gösterilmelidir. Hedefe ulaşıldığında ise başlangıç noktasından hedef konuma giden robota göre en kısa yol ızgara üzerinde ayrıca çizdirilmelidir. Geçen toplam süre (sn cinsinden) ve kaç kare üzerinden geçildiği bilgileri ekranda gösterilmelidir.



Problem 1 Örnek Görünüm

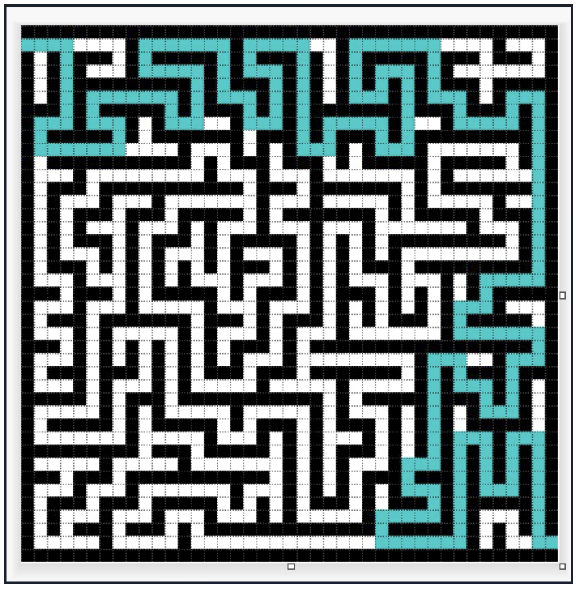
Problem 2:

Bu problemde bizden robotu labirentteki çıkış noktasına ulaştırmanız beklenmektedir.

Öncelikle kullanacıdan alınan kenar uzunluğu bilgisine göre karesel grid oluşturulması gerekmektedir. Grid üzerine Problem 1' de kullanılan 1x1 birimlik engeller konularak labirent oluşturulmalıdır. Labirentte mutlaka çıkmaz yollar olmalısı gerekmektedir.

Labirentin giriş noktası sol üst köşe, çıkış noktası ise sağ alt köşedir. Robot başlangıçta labirenti bilmemektedir. Labirentte yanlış girilen bir yol algılandığında robotun doğru olarak tespit ettiği en son konuma giderek buradan itibaren yol aramaya devam etmesi gerekmektedir.

Tüm bu bilgiler doğrultusunda, robotun çıkışa ulaşmak için izlediği yol adım adım ızgara üzerinde gösterilmelidir. Her adımda robotun daha önce geçtiği yollar üzerinde iz bırakması gerekmektedir. Robot hedefe ulaştığında giriş noktasından çıkış noktasına giden yol ızgara üzerinde çizilmelidir. Geçen toplam süre (sn cinsinden), kaç kare üzerinden geçildiği bilgileri ekranda gösterilmelidir.



Problem 2 Örnek Gösterim

III. YÖNTEM VE SONUÇLAR

Bu projeyi ilk okuduğumda beni daha çok zorlayacağını düşündüğüm kısım Problem 2 kısmıydı. Çünkü Problem 1'e kıyasla burda engeller rastgele oluşturulmalıydı ve labirent mantığında olmalıydı. Labirent oluşturma kısmında hangi algoritmayı kullanacağımı bilmiyordum. Ayrıca labirent çözülme algoritması da yazmam gerekiyordu. Bunun iki Problem içinde ortak olması işimi daha da kolaylaştırır diye düşündüm.

İlk olarak kullanabileceğim arama algoritmalarına baktım. Burada dikkat ettiğim konular aramanın robotmuş gibi yapılması ve süre konularıydı. DFS, BFS, Dijkstra, A* ve Greedy algoritmalarını bir örnek üzerinde denedim. BFS,Dijkstra ve A* çok fazla kısımdan arama yaptığı için projeye uygun olmayacaktı. Özellikle BFS neredeyse tüm gridi dolaştığı için zamansal olarak çok uzun sürüyordu. En son DFS ve Greedy algoritmalarını karşılaştırdım. İkisi benzer yollar denese de Greedy genellikle daha hızlı sonuca ulaşıyordu. Ben de Greedy algoritmasını kullanmaya karar verdim.

Bir sonraki adım labirent oluşturma kısmında ne yapacağıma karar vermekti. Çok fazla araştırma yaptım ancak tam olarak işime yarayacak bir şey bulamadım. Yazdığım algoritmalar tam olarak istediğim gibi çalışmıyordıu. Araştırmaya devam ettiğimde kaynakça kısmında paylaşacağım sitede bir algoritma buldum. Denediğimde istediğim sonucu almıştım. Algoritma temelde DFS kullanıyordu ve Backtracking uygulanmıştı. Algoritma şu şekilde çalışmaktadır:

1. Rastgele bir düğüm seçilir.
2. Seçilen düğüm ziyaret edilmiş olarak işaretlenir.
3. Düğümün tüm komşularının listesi alınır.
4. Rastgele seçilen bir komşudan başlayarak:
    1. Komşunun ziyaret edilip edilmediği kontrol edilir.
    2. Eğer ziyaret edilmemişse düğüm ile arasındaki duvar kaldırılır.
    3. Komşu mevcut düğüm yapılarak algortima recursive şekilde devam eder.

Algortima oldukça iyi çalışıyordu. Ancak console uygulaması için yazılmış olduğu için kendi koduma göre ayarlamam gerekiyordu. Ben de döndürülen string değerini engel olması için gereken string değeri ile aynı mı diye Regex.IsMatch kullanarak kontrol ettim. Eğer eşleşiyorsa bunu engel olarak tanımladım.

```
maze = new Maze(edge/2);
for (int i = 0; i < edge; i++)
{
    for (int j = 0; j < edge; j++)
    {
        if (Regex.IsMatch(maze.mazestr.Substring(i * edge + j, 1), "[+-|]"))
            raster[i, j] = problemForm.IMPDMT;
    }
}
```

Regex.IsMatch

Buradan sonra URL adresinden .txt okuma kısmına geçtim. Okuduğum .txt dosyasını bir stringe atadım. Daha sonra bunu stringi split ederek string diziye atadım. Bu dizini her elemanını için öncelikle elemanları int diziye dönüştürdüm.

Burdan sonra ise int dizinin elemanlarını matris1 ve matris2 değişkenlerine atadım.

```
public void TextTo2DArray()
{
    int i = 0;
    theTextFile = wc.DownloadString("http://bilgisayar.kocaeli.edu.tr/prolab2/url1.txt");
    newText = theTextFile.Split(new Char[] { '\n' });
    foreach (string author in newText)
    {
        var intArray = author.Select(c => c - '0').ToArray();
        for (int a = 0; a < intArray.Length; a++)
            matris1[i, a] = intArray[a];
        i++;
    }
    i = 0;
    theTextFile = wc.DownloadString("http://bilgisayar.kocaeli.edu.tr/prolab2/url2.txt");
    newText = theTextFile.Split(new Char[] { '\n' });
    foreach (string author in newText)
    {
        var intArray = author.Select(c => c - '0').ToArray();
        for (int a = 0; a < intArray.Length; a++)
            matris2[i, a] = intArray[a];
        i++;
    }
}
```

URL Adresinde Matrise Aktarma

Daha sonra kullanıcının "URL Değiştir" butonuna basmasına göre URL class'ından uygun matrisi return ettim ve bu matrisi iki boyutlu olan raster değişkenine klonladım. Problem 1'de 2x2 ve 3x3'lük engellerin tamamının dolu olmaması bekleniyordu. Bunun için rastgele olarak buralardan engelleri kaldırım. Daha sonra okuduğum değere göre uygun olan engel değişkenine atama yaptım. Burda sabit değişkenler kullanma sebebim Form Paint kısmında daha rahat ediyor olmamdı.

```
if (i % 2 == 0)
{
    raster = (int[,])url.Return1().Clone();
}
else
{
    raster = (int[,])url.Return2().Clone();
}
i++;
for (int i = 0; i < edge; i++)
{
    for (int j = 0; j < edge; j++)
    {
        if (raster[i, j] == 2)
        {
            if (rnd.Next(1, 5) == 1)
                raster[i, j] = problemForm.EMPTY;
            else
                raster[i, j] = problemForm.IMPDMT2;
        }
        else if (raster[i, j] == 3)
        {
            if (rnd.Next(1, 5) == 1)
            {
                if (!(i > 0 && j > 0 && raster[i + 1, j] == 3 &&
                    raster[i - 1, j] == 3 && raster[i, j + 1] == 3 &&
                    raster[i, j + 1] == 3))
                    raster[i, j] = problemForm.EMPTY;
            }
            else
                raster[i, j] = problemForm.IMPDMT3;
        }
    }
}
```

Klonlama ve Engel Ataması

Bunları tamamladıktan sonra problemleri çözme aşamasına geçtim. Yukarıda belirttiğim gibi Greedy algoritmasını kullanmaya karar verdim. Bu algoritma DFS ve BFS algoritmalarının birleşimidir. Best-First Search, her iki algoritmanın da avantajlarından yararlanmamızı sağlar. Greedy(Best-First Search) arama algoritması her zaman o anda en iyi görünen yolu seçer. Best-First Search ile hedef düğüme en yakın düğüm genişletilir ve yakınlık maliyeti bir fonksiyonla tahmin edilir.

$$f(n) = h(n)$$

Burada h(n) mevcut düğümden hedef düğüme olan maliyettir. h(n) değerini hesaplamak için oluşturduğum tüm düğümlerin merkez noktalarını hesapladım. Bu değerleri centerpoints değişkeninde tuttum. Daha sonra başlangıç noktasından başlayarak bulunduğum düğümün etrafındaki engel olmayan komşu düğümlerin listesini buldum. Her engel olmayan komşu için h değerini hesapladım. h değerini, bulunulan düğüm ile varış noktasının x, y değerlerinin karesi olarak hesapladım.

```
List<Node> neighbors = FindNeighbors(current);
for (int i = 0; i < neighbors.Count; i++)
{
    double diffxlength = Math.Abs(problemForm.centerpoints[problemForm.destinationPos.row,
        problemForm.destinationPos.col].X
        - problemForm.centerpoints[neighbors[i].row,
        neighbors[i].col].X);
    double diffylength = Math.Abs(problemForm.centerpoints[problemForm.destinationPos.row,
        problemForm.destinationPos.col].Y
        - problemForm.centerpoints[neighbors[i].row,
        neighbors[i].col].Y);
    neighbors[i].h = diffxlength + diffylength;
```

Maliyet için h değerinin hesaplanması

Algoritmanın çalışması iki adet liste kullanılmıştır. openList ve closedList isminde iki listem vardı. Bunlar algoritma için gerekli listelerdir. Algoritmanın çalışma mantığı şu şekilde ilerlemektedir:

1. Başlangıç noktası openList'e yerleştirilir.
2. Eğer openList boş ise arama durdurulur ve return edilir.
3. En düşük maliyetli düğüm openList h değerine göre sort edilerek bulunur.
4. En düşük maliyetli düğüm 0. indekste olacağı için openList listesinden kaldırılır.
5. Kaldırılan düğüm closedList değişkenine yerleştirilir.
6. Şu anki düğümün etrafındaki engel olmayan düğümlerin listesi bulunur.

7.Listedeki her düğüm için h değeri hesaplanır ve h değeri 0 olan varsa arama durdurulur ve başarıyla sonlandırılır. Eğer hiçbiri sıfır değilse bir sonraki aşamaya geçilir
8.Listedeki her düğümün closedList'te olup olmadığı kontrol edilir eğer yoksa openList'e eklenir ve 2. adıma dönülür.

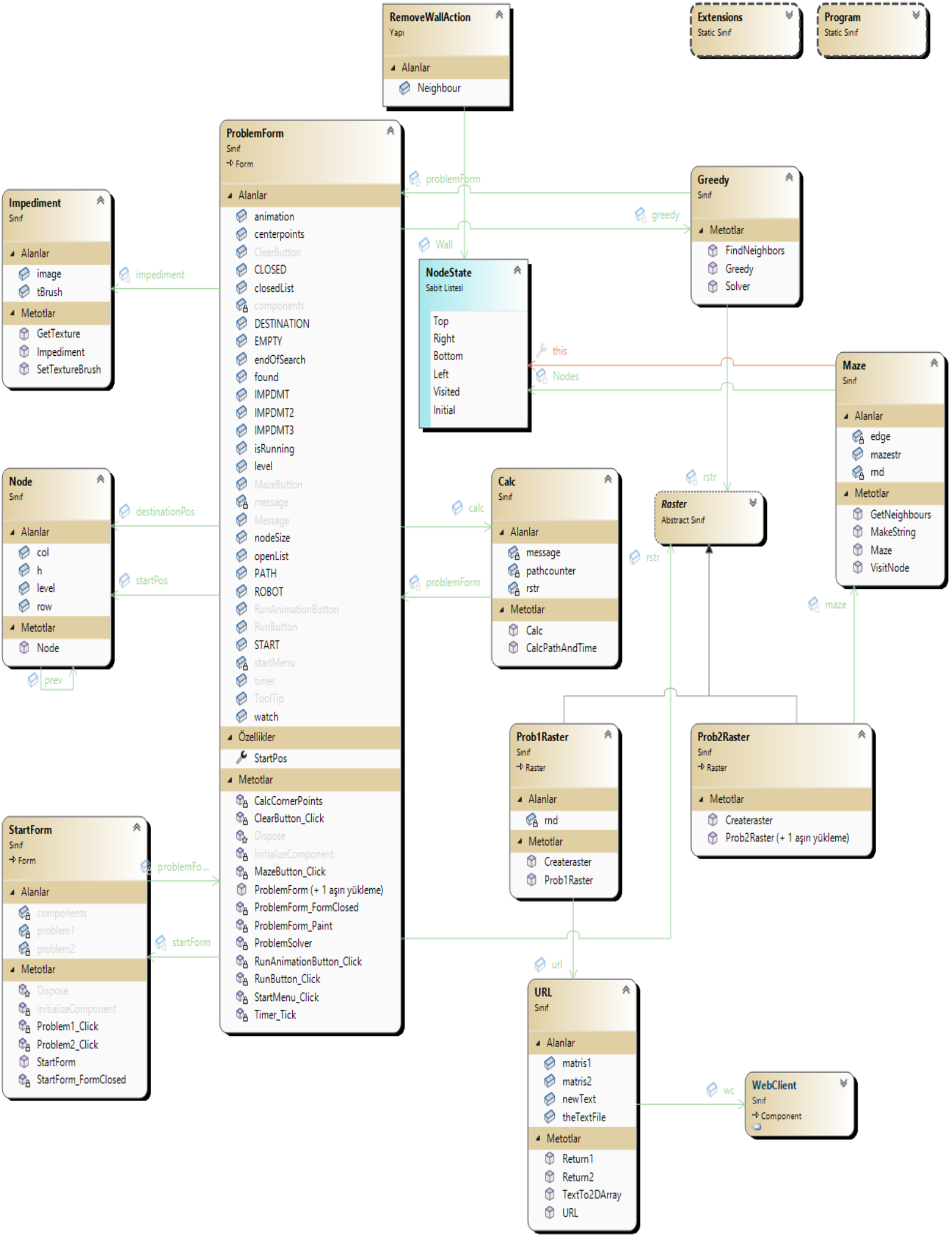
```
if (neighbors[i].h == 0)
{
    problemForm.closedList.Add(current);
    problemForm.found = true;
    problemForm.endOfSearch = true;
    return;
}
else
{
    int index = -1;
    for (int j = 0; j < problemForm.closedList.Count; j++)
    {
        if (problemForm.closedList[j].row == neighbors[i].row && problemForm.closedList[j].col == neighbors[i].col)
        {
            index = j;
            break;
        }
    }
    if (index == -1)
    {
        problemForm.openList.Add(neighbors[i]);
        gr.raster[neighbors[i].row, neighbors[i].col] = problemForm.ROBOT;
    }
}
```

h değeri kontrolü

Çıkış yolu bulunduktan sonra arama sonlandırılır ve bulunurken geçen süre ve geçilen kare sayısı hesaplanması için Calc class'ının CalcPathAndTime fonksiyonu çalıştırılır. closedList'in ilk elemanından başlayarak düğüm son noktaya gelmediği sürece bir düğümü bir sonrakine taşıyarak geçilen kare sayısı hesaplanır. Tüm bu süreçte çalıştır butonuna basıldıktan sonra buraya kadar geçen süre watch değişkeniyle hesaplanır ve burada durdurulur.
Geçen süre ve geçilen kare sayısı ekranda gösterilir.

IV. KAYNAKÇA

- https://en.wikipedia.org/wiki/Maze_generation_algorithm
- https://www.javatpoint.com/ai-informed-search-algorithms
- https://rosettacode.org/wiki/Maze_generation
- https://en.wikipedia.org/wiki/Maze-solving_algorithm
- https://stackoverflow.com/questions/61218945/best-algorithm-for-maze-solving


RemoveWallAction
Yapı
Alanlar
Neighbour
Extensions
Static Sınıf
Program
Static Sınıf
ProblemForm
Sınıf
Form
Alanlar
animation
centerpoints
ClearButton
CLOSED
closedList
components
DESTINATION
EMPTY
endOfSearch
found
IMPDMT
IMPDMT2
IMPDMT3
isRunning
level
MazeButton
message
Message
nodeSize
openList
PATH
ROBOT
RunAnimationButton
RunButton
START
startMenu
timer
ToolTip
watch
Özellikler
StartPos
Metotlar
CalcCornerPoints
ClearButton_Click
Dispose
InitializeComponent
MazeButton_Click
ProblemForm (+ 1 aşırı yükleme)
ProblemForm_FormClosed
ProblemForm_Paint
ProblemSolver
RunAnimationButton_Click
RunButton_Click
StartMenu_Click
Timer_Tick
problemForm
Greedy
Sınıf
Metotlar
FindNeighbors
Greedy
Solver
greedy
Wall
Impediment
Sınıf
Alanlar
image
tBrush
Metotlar
GetTexture
Impediment
SetTextureBrush
impediment
NodeState
Sabit Listesi
Top
Right
Bottom
Left
Visited
Initial
this
Nodes
Maze
Sınıf
Alanlar
edge
mazestr
rnd
Metotlar
GetNeighbours
MakeString
Maze
VisitNode
Node
Sınıf
Alanlar
col
h
level
row
Metotlar
Node
destinationPos
startPos
prev
Calc
Sınıf
Alanlar
message
pathcounter
rstr
Metotlar
Calc
CalcPathAndTime
calc
problemForm
rstr
Raster
Abstract Sınıf
rstr
maze
Prob1Raster
Sınıf
Raster
Alanlar
rnd
Metotlar
Createraster
Prob1Raster
Prob2Raster
Sınıf
Raster
Metotlar
Createraster
Prob2Raster (+ 1 aşırı yükleme)
StartForm
Sınıf
Form
Alanlar
components
problem1
problem2
Metotlar
Dispose
InitializeComponent
Problem1_Click
Problem2_Click
StartForm
StartForm_FormClosed
problemFo...
startForm
url
URL
Sınıf
Alanlar
matris1
matris2
newText
theTextFile
Metotlar
Return1
Return2
TextTo2DArray
URL
wc
WebClient
Sınıf
Component