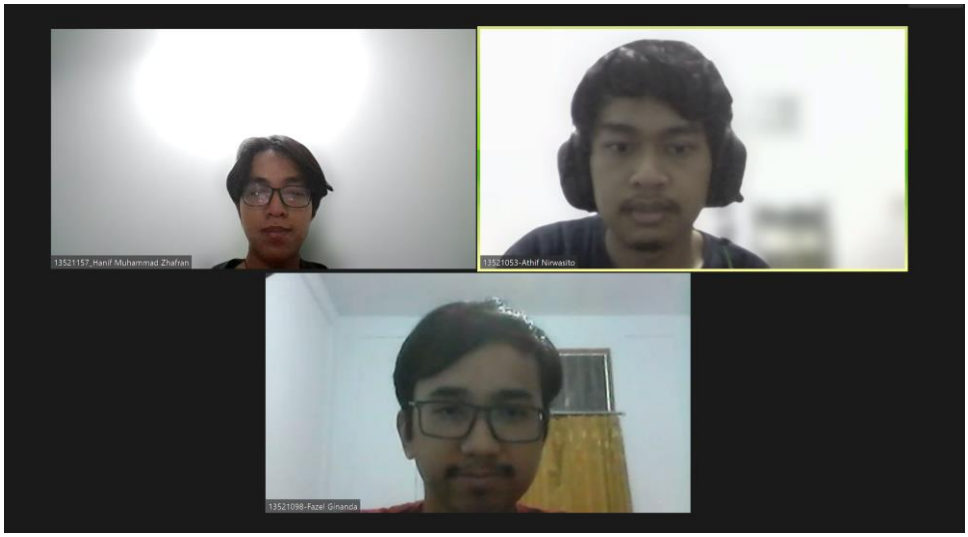
TUGAS BESAR 2 STRATEGI ALGORITMA
IMPLEMENTASI ALGORITMA *BREADTH FIRST SEARCH* DAN *DEPTH FIRST SEARCH*

Disusun untuk memenuhi laporan tugas besar mata kuliah Strategi Algoritma semester 2 Institut Teknologi Bandung



Disusun oleh kelompok *TreasureHunter*:

Athif Nirwasito	13521053
Fazel Ginanda	13521098
Hanif Muhammad Zhafran	13521157

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Jl. Ganesa No. 10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat, 40132

2023

Kata Pengantar

Puji dan syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa, karena berkat-Nya begitu melimpah dalam kehidupan kami. Kami dapat menyelesaikan tugas besar pertama untuk mata kuliah Strategi Algoritma.

Berikut disajikan laporan tugas besar ini. Laporan ini berisi penjelasan terhadap persoalan dari tugas besar pertama

Dengan semua informasi yang kami peroleh

Kami berharap laporan ini dapat menjadi media pembelajaran dan eksplorasi kami khususnya pada mata kuliah Strategi Algoritma dalam implementasi Algoritma *Breadth First Search* dan *Depth First Search*.

Bandung, 22 Maret 2023

Penyusun Laporan

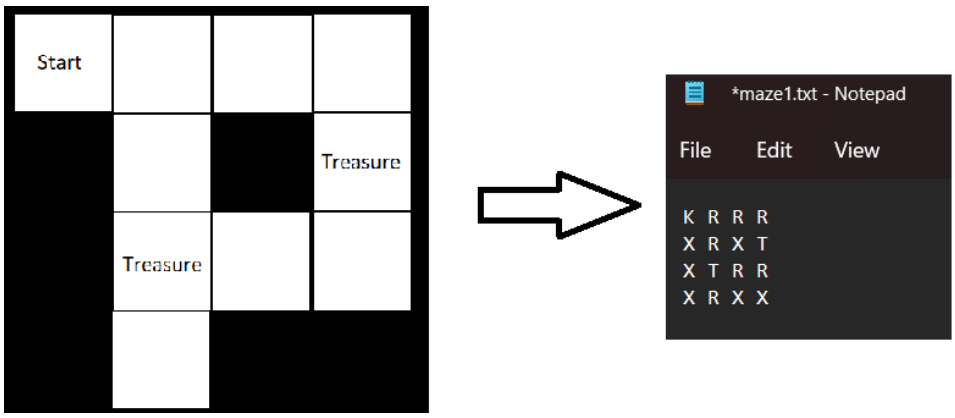
DAFTAR ISI

BAB 1
DESKRIPSI TUGAS

Dalam tugas besar ini, Anda akan diminta untuk membangun sebuah aplikasi dengan GUI sederhana yang dapat mengimplementasikan BFS dan DFS untuk mendapatkan rute memperoleh seluruh treasure atau harta karun yang ada. Program dapat menerima dan membaca input sebuah file txt yang berisi maze yang akan ditemukan solusi rute mendapatkan treasure-nya. Untuk mempermudah, batasan dari input maze cukup berbentuk segi-empat dengan spesifikasi simbol sebagai berikut :

- K : Krusty Krab (Titik awal)
- T : Treasure
- R : Grid yang mungkin diakses / sebuah lintasan
- X : Grid halangan yang tidak dapat diakses

Contoh file input :

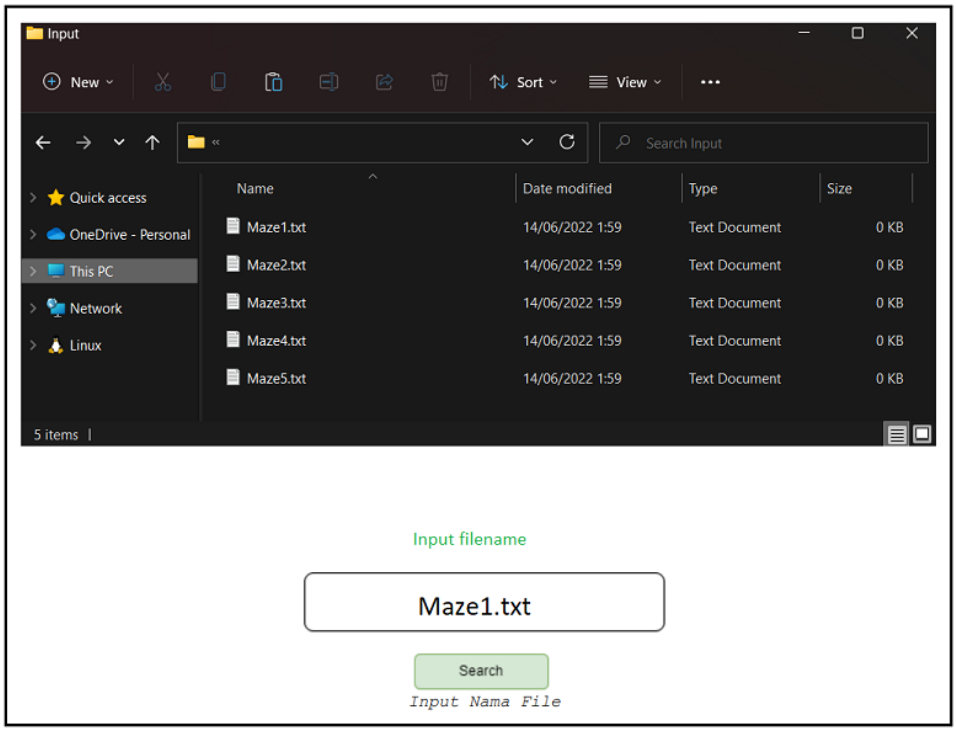


Gambar 1.1. Ilustrasi input file maze

Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), anda dapat menelusuri grid (simpul) yang mungkin dikunjungi hingga ditemukan rute solusi, baik secara melebar ataupun mendalam bergantung alternatif algoritma yang dipilih. Rute solusi adalah rute yang memperoleh seluruh treasure pada maze. Perhatikan bahwa rute yang diperoleh dengan algoritma BFS dan DFS dapat berbeda, dan banyak langkah yang dibutuhkan pun menjadi berbeda. Prioritas arah simpul yang dibangkitkan dibebaskan asalkan ditulis di laporan ataupun readme, semisal LRUD (left right up down). Tidak ada pergerakan secara diagonal. Anda juga diminta untuk memvisualisasikan input txt tersebut menjadi suatu grid maze serta hasil pencarian rute solusinya. Cara visualisasi grid dibebaskan, sebagai contoh

dalam bentuk matriks yang ditampilkan dalam GUI dengan keterangan berupa teks atau warna. Pemilihan warna dan maknanya dibebaskan ke masing - masing kelompok, asalkan dijelaskan di readme / laporan.

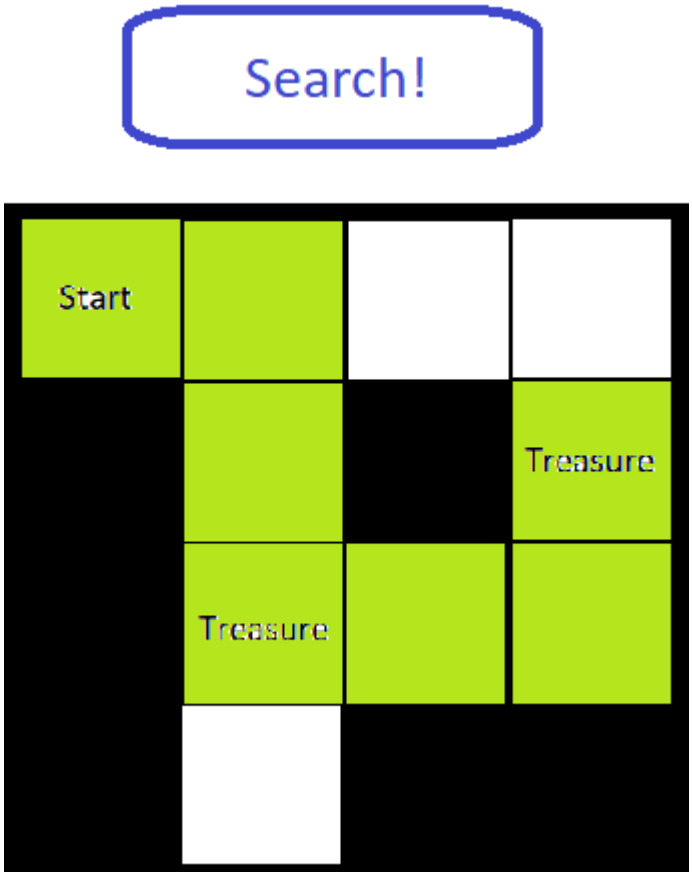
Contoh input aplikasi :



Gambar 1.2. Contoh input program

Daftar input maze akan dikemas dalam sebuah folder yang dinamakan test dan terkandung dalam repository program. Folder tersebut akan setara kedudukannya dengan folder src dan doc (struktur folder repository akan dijelaskan lebih lanjut di bagian bawah spesifikasi tubes). Cara input maze boleh langsung input file atau dengan textfield sehingga pengguna dapat mengetik nama maze yang diinginkan. Apabila dengan textfield, harus menghandle kasus apabila tidak ditemukan dengan nama file tersebut.

Contoh output Aplikasi :



Gambar 1.3. Contoh output program untuk gambar 2

Setelah program melakukan pembacaan input, program akan memvisualisasikan gridnya terlebih dahulu tanpa pemberian rute solusi. Hal tersebut dilakukan agar pengguna dapat mengerjakan terlebih dahulu treasure hunt secara manual jika diinginkan. Kemudian, program menyediakan tombol solve untuk mengeksekusi algoritma DFS dan BFS. Setelah tombol diklik, program akan melakukan pemberian warna pada rute solusi.

BAB 2
LANDASAN TEORI

I. Graph Traversal

Graph traversal adalah sebuah metode untuk mengunjungi semua simpul pada graph secara sistematik. Terdapat 2 cara umum untuk melakukan traversal pada graph, yaitu Breadth First Search (BFS) dan Depth First Search (DFS).

II. Breadth First Search

Breadth First Search adalah sebuah metode pencarian pada graph yang memfokuskan penulusuran pada level per level. Dengan kata lain, Breadth First Search adalah penulusuran graph yang memprioritaskan pencariannya untuk meluas. Umumnya, struktur data BFS terdiri dari matriks ketetanggan untuk menyimpan informasi tetangga, antrian q untuk menyimpan simpul-simpul yang akan dikunjungi, dan tabel boolean untuk menyatakan apakah suatu lokasi telah dikunjungi.

III. Depth First Search

Depth First Search adalah sebuah metode pencarian pada graph yang memfokuskan penulusaran pada cabang per cabang. Dengan kata lain, Depth First Search adalah penulusuran graph yang memprioritaskan pencariannya untuk mendalam. Secara umum, DFS dapat dilakukan dengan dua cara. Cara pertama adalah melakukan backtraking dengan membuat fungsi DFS secara rekursif. Cara kedua adalah dengan menggunakan struktur data yang terdiri dari matriks ketetanggan untuk menyimpan informasi tetangga, stack s untuk menyimpan simpul-simpul yang akan dikunjungi, dan tabel boolean untuk menyatakan apakah suatu lokasi telah dikunjungi.

IV. Penjelasan Singkat Mengenai C# Desktop Application Development

C# merupakan bahasa pemrograman yang merupakan pengembangan dari bahasa C. C# menggunakan paradigma pemrograman berorientasi-objek. C# merupakan salah satu bahasa yang sering digunakan dalam pengembangan aplikasi desktop ataupun aplikasi multi-platform (desktop, Android, iOS) dengan lingkungan .NET *framework*. Untuk memudahkan pengembangan aplikasi dengan bahasa C#, digunakan suatu *IDE* (*integrated desktop environment*). Beberapa *IDE* yang dapat digunakan ialah Visual Studio dan Rider.

Pada aplikasi ini, digunakan *framework WPF* (*Windows Presesntation Foundation*). *WPF* merupakan *framework* yang merupakan bagian dari *.NET framework*. WPF menggunakan XAML untuk mendesain UI dari suatu aplikasi, sehingga memudahkan pemisahan antara proses desain dengan logic dibelakangnya. WPF juga terintegrasi dengan IDE seperti Visual Studio.

BAB 3
APLIKASI BFS DAN DFS

I. Langkah-Langkah Pemecahan Masalah

Maze terdiri dari empat jenis grid, yaitu K (grid start), T (grid treasure), R (grid yang dapat dilewati), dan X (grid yang tidak dapat dilewati). Pencarian dimulai dari grid start. Pencarian dilakukan dengan algoritma DFS atau BFS sesuai dengan pilihan pengguna. Terdapat opsi tambahan dalam permasalahan yaitu permasalahan TSP (*Travelling Salesman Problem*). Jika pengguna tidak memilih TSP, maka pencarian dengan algoritma BFS atau DFS akan berhenti ketika seluruh treasure sudah ditemukan. Jika pengguna memilih TSP, maka pencarian dengan algoritma BFS atau DFS akan berhenti ketika seluruh treasure sudah ditemukan dan setelah itu kembali ke grid start.

a. Langkah-Langkah Algoritma BFS

Berikut langkah-langkah pencarian treasure dengan algoritma BFS:

1. Menetapkan grid start sebagai titik pencarian awal
2. Melakukan pengecekan tetangga dengan prioritas pencarian adalah node dengan level yang sama (Prioritas arah gerak adalah Kanan – Bawah – Kiri – Atas)
3. Ketika ditemukan treasure, tambahkan rute pencarian dari grid awal sampai grid treasure ke dalam suatu list.
4. Ubah grid treasure yang baru saja ditemukan menjadi grid start baru.
5. Lakukan langkah 2-4 sampai semua treasure berhasil ditemukan.
6. Jika persoalan TSP dinyalakan, ubah grid treasure yang terakhir menjadi grid start. Kemudian ubah grid start yang paling awal menjadi grid target yang baru.
7. Lakukan pengecekan dengan cara yang sama dengan langkah 2 sampai ditemukan grid start awal.

b. Langkah-Langkah Algoritma DFS

1. Berikut langkah-langkah pencarian treasure dengan algoritma DFS:
2. Menetapkan grid start sebagai titik pencarian awal.
3. Melakukan pengecekan tetangga dengan prioritas node child merupakan node yang didahulukan (Prioritas arah gerak adalah Kanan – Bawah – Kiri – Atas)
4. Ketika ditemukan treasure, tambahkan rute pencarian dari grid awal sampai grid treasure ke dalam suatu list.
5. Ubah grid treasure yang baru saja ditemukan menjadi grid start baru.
6. Lakukan langkah 2-4 sampai semua treasure berhasil ditemukan.

7. Jika persoalan TSP dinyalakan, ubah grid treasure yang terakhir menjadi grid start. Kemudian ubah grid start yang paling awal menjadi grid target yang baru.
8. Lakukan pengecekan dengan cara yang sama dengan langkah 2 sampai ditemukan grid start awal.

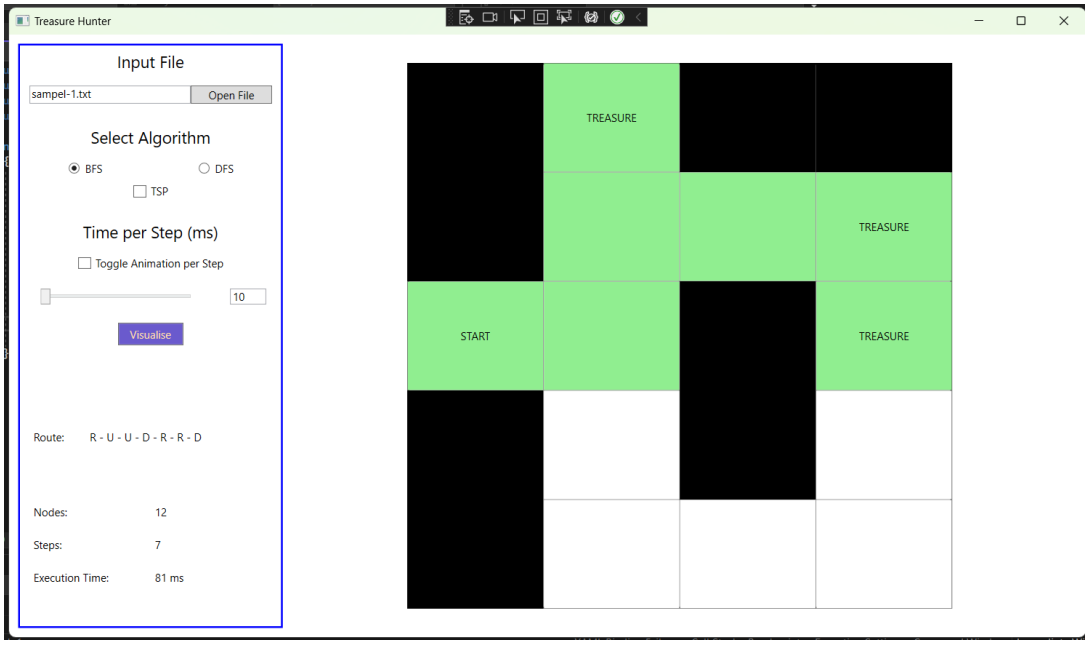
II. Mapping Persoalan

Peta direpresentasikan dengan graf dimana setiap grid yang dapat dilewati merupakan node dari graf tersebut. Node pertama merupakan grid start, sedangkan node goal merupakan grid treasure. Dalam program, graf direpresentasikan sebagai suatu matriks dimana grid hidup (grid yang dapat dilewati) yang bersebelahan (secara horizontal atau vertical, tetapi tidak diagonal) merupakan node yang saling bertetangga.

III. Ilustrasi Kasus Lain

a. Contoh Pencarian yang Berhasil

Berikut merupakan salah satu contoh kasus pencarian berhasil yang berbeda dengan kasus pada spesifikasi tugas.

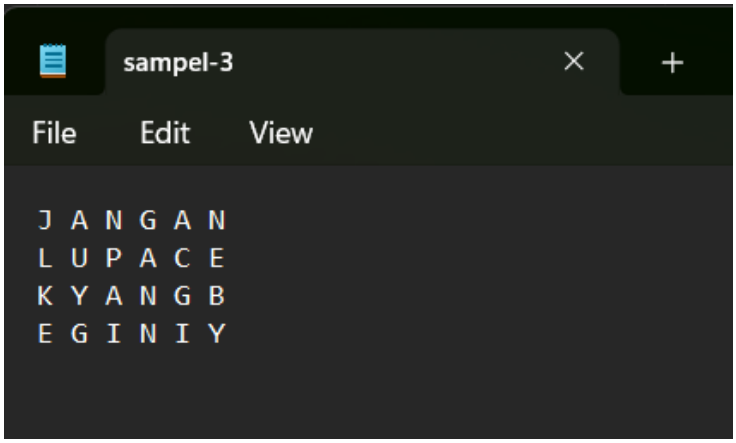


Gambar 3.3.1.1 Contoh program berhasil menemukan solusi

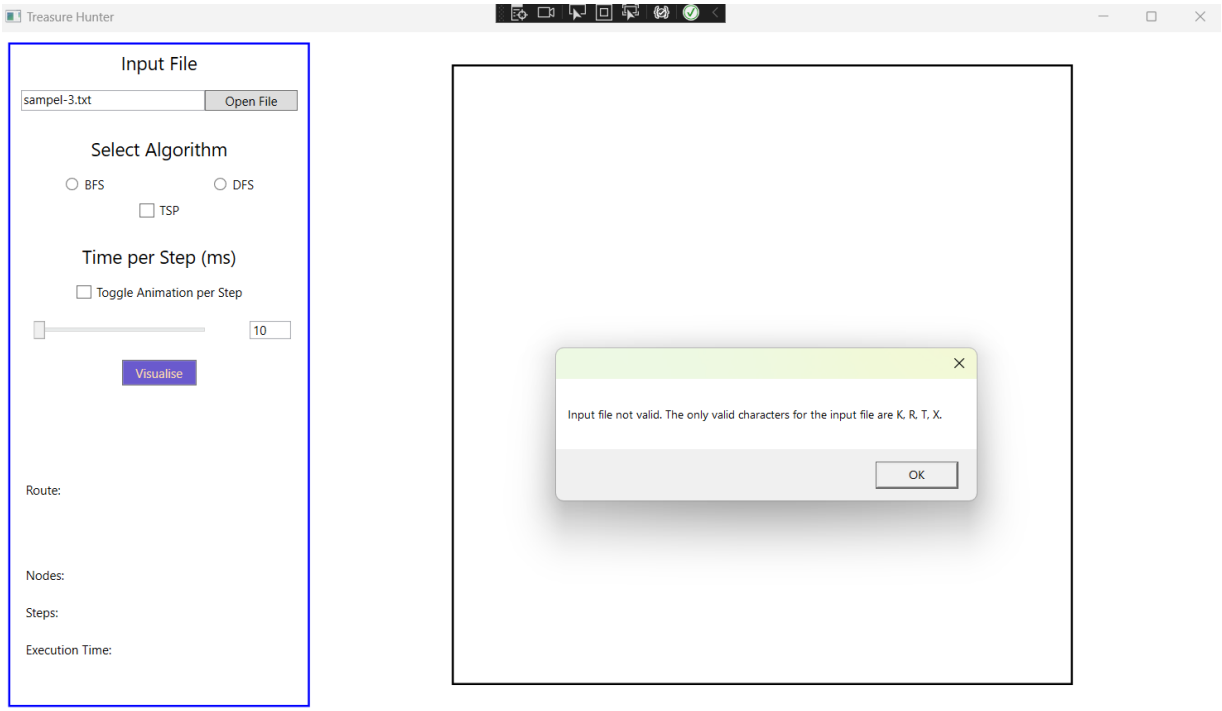
Grid yang diberi warna hijau merupakan grid yang dilewati saat pengambilan treasure. Rincian arah gerak dapat dilihat pada sisi kiri aplikasi. Aplikasi juga menunjukkan berapa node yang dikunjungi dalam proses pencarian, banyak langkah dalam rute pengambilan treasure, dan juga waktu proses pencarian dalam millisecond.

b. Validasi Input File

Berikut merupakan contoh input file yang salah diikuti dengan handling dari aplikasi:

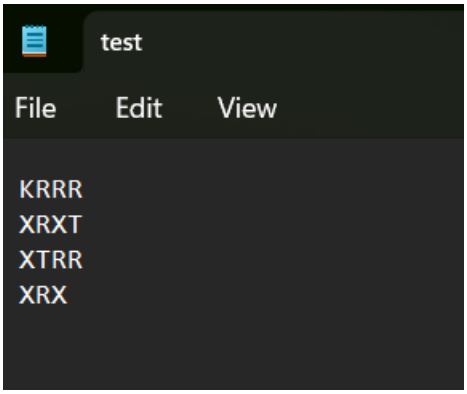


Gambar 3.3.2.1 Contoh input dengan karakter invalid

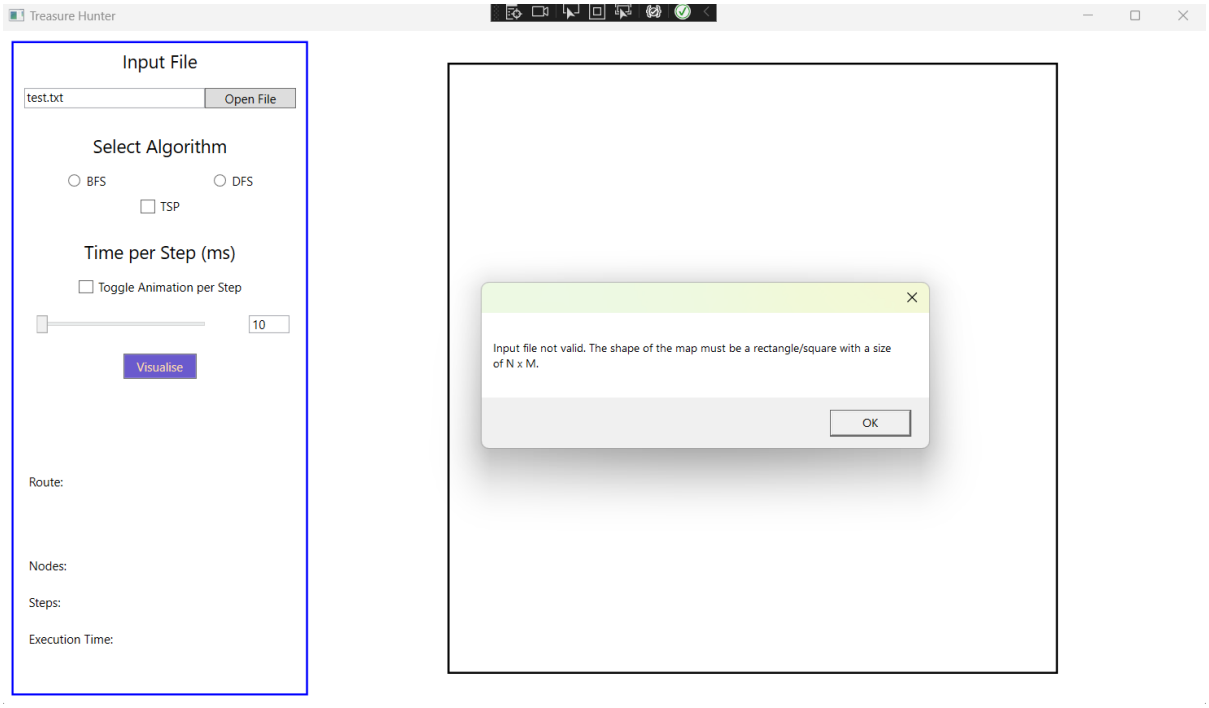


Gambar 3.3.2.2 Contoh program menemukan karakter yang invalid

Aplikasi akan memunculkan Message Box yang berisi pemberitahuan untuk pengguna bahwa input file mengandung karakter diluar K, R, T, atau X. Aplikasi juga akan memvalidasi input file dengan ukuran peta yang tidak N x M (bukan persegi/persegi panjang).

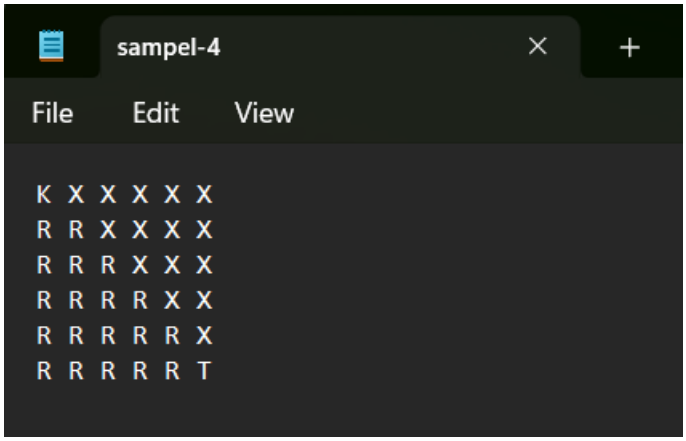


Gambar 3.3.2.3 Contoh ukuran input invalid



Gambar 3.3.2.4 Contoh program mendeteksi ukuran input tidak valid

Contoh input file yang valid akan memunculkan tampilan sebagai berikut:



Gambar 3.3.2.5 Contoh input yang valid

Gambar 3.3.2.6 Contoh program dengan input yang valid

Grid yang dapat dilewati namun bukan start ataupun treasure akan ditandai dengan warna putih. Grid start ditandai dengan warna hijau tosca. Grid treasure ditandai dengan warna kuning. Grid yang tidak dapat dilewati ditandai dengan warna hitam.

BAB 4

ANALISIS PEMECAHAN MASALAH

I. Implementasi program (pseudocode)

1. BFS

```
public override void Solve(bool TSP)
{
   bool repeat = TSP;
   DynamicPeta.Copy(peta);
   queue = new Queue<(Cell,Route)>();
   queue.Enqueue((DynamicPeta.Start, new Route()));
   int count = DynamicPeta.NumTreasure;
   while (count > 0 && queue.Count > 0)
   {

      (Cell,Route) accessing = queue.Dequeue();
      Route temp = new Route(accessing.Item2);
      temp.AddCell(accessing.Item1);
      if(accessing.Item1.Type == 3) {
         count--;
         accessing.Item1.NumAccessed++;
         DynamicPeta[accessing.Item1.X, accessing.Item1.Y].Type = 0;
         DynamicPeta[DynamicPeta.Start.X, DynamicPeta.Start.Y].Type = 2;
         DynamicPeta.Start = DynamicPeta[accessing.Item1.X, accessing.Item1.Y];
         //empty queue
         DynamicPeta.Inaccess();
         if(count <= 0) {

            if (repeat)
            {
               count++;
               DynamicPeta[peta.Start.X, peta.Start.Y].Type = 3;
               DynamicPeta[accessing.Item1.X, accessing.Item1.Y].Type = 0;
               DynamicPeta[DynamicPeta.Start.X, DynamicPeta.Start.Y].Type = 2;
               DynamicPeta.Start = DynamicPeta[accessing.Item1.X, accessing.Item1.Y];
               repeat = false;
               queue.Clear();
               queue.Enqueue((DynamicPeta.Start, accessing.Item2));
            }
            else
            {
               solusi.Copy(temp);
               queue.Clear();
            }
         }
         else
         {
            queue.Clear();
            queue.Enqueue((DynamicPeta.Start, accessing.Item2));
         }


      }
      else//type ==2
      {
         try
         {
            if (DynamicPeta[accessing.Item1.X + 1, accessing.Item1.Y].Type > 1 && !DynamicPeta[accessing.Item1.X + 1,
accessing.Item1.Y].Accessed)
            {
               queue.Enqueue((DynamicPeta[accessing.Item1.X + 1, accessing.Item1.Y], temp));
            }
         }catch (Exception e) { }//ignore
         try
         {
            if (DynamicPeta[accessing.Item1.X, accessing.Item1.Y - 1].Type > 1 && !DynamicPeta[accessing.Item1.X,
accessing.Item1.Y - 1].Accessed)
            {
```

```
                queue.Enqueue((DynamicPeta[accessing.Item1.X, accessing.Item1.Y - 1], temp));
            }
        }
        catch (Exception e) { }//ignore
        try
        {
            if (DynamicPeta[accessing.Item1.X - 1, accessing.Item1.Y].Type > 1 && !DynamicPeta[accessing.Item1.X - 1,
accessing.Item1.Y].Accessed)
            {
                queue.Enqueue((DynamicPeta[accessing.Item1.X - 1 , accessing.Item1.Y], temp));
            }
        }
        catch (Exception e) { }//ignore
        try
        {
            if (DynamicPeta[accessing.Item1.X, accessing.Item1.Y + 1].Type > 1 && !DynamicPeta[accessing.Item1.X,
accessing.Item1.Y + 1].Accessed)
            {


                queue.Enqueue((DynamicPeta[accessing.Item1.X, accessing.Item1.Y + 1], temp));
            }
        }
        catch (Exception e) { }//ignore
        accessing.Item1.NumAccessed++;
    }
    DynamicPeta[accessing.Item1.X, accessing.Item1.Y].Accessed = true;
  }
}
```

2. DFS

```
public override void Solve(bool TSP)
{
  bool repeat = TSP;
  DynamicPeta.Copy(peta);
  stack = new Stack<(Cell, Route)>();
  stack.Push((DynamicPeta.Start, new Route()));
  int count = DynamicPeta.NumTreasure;


  while (count > 0 && stack.Count > 0)
  {
    (Cell, Route) accessing = stack.Pop();
    Route temp = new Route(accessing.Item2);
    temp.AddCell(accessing.Item1);
    if (accessing.Item1.Type == 3)
    {
      count--;
      accessing.Item1.NumAccessed++;
      DynamicPeta[accessing.Item1.X, accessing.Item1.Y].Type = 0;
      DynamicPeta[DynamicPeta.Start.X, DynamicPeta.Start.Y].Type = 2;
      DynamicPeta.Start = DynamicPeta[accessing.Item1.X, accessing.Item1.Y];


      DynamicPeta.Inaccess();
      if (count <= 0)
      {
        if (repeat)
        {
          count++;
          DynamicPeta[peta.Start.X, peta.Start.Y].Type = 3;
          DynamicPeta[accessing.Item1.X, accessing.Item1.Y].Type = 0;
          DynamicPeta[DynamicPeta.Start.X, DynamicPeta.Start.Y].Type = 2;
          DynamicPeta.Start = DynamicPeta[accessing.Item1.X, accessing.Item1.Y];
          repeat = false;
          Console.WriteLine("Masuk");
```

```
                stack.Clear();
                stack.Push((DynamicPeta.Start, accessing.Item2));
            }
            else
            {
                solusi.Copy(temp);
                stack.Clear();
            }
        }
        else
        {
            stack.Clear();
            stack.Push((DynamicPeta.Start, accessing.Item2));
        }
    }
    else //type ==2
    {
        try
        {
            if (DynamicPeta[accessing.Item1.X, accessing.Item1.Y - 1].Type > 1 && !DynamicPeta[accessing.Item1.X,
accessing.Item1.Y - 1].Accessed)
            {
                stack.Push((DynamicPeta[accessing.Item1.X, accessing.Item1.Y - 1], temp));
            }
        }
        catch (Exception e) { }
        try
        {
            if (DynamicPeta[accessing.Item1.X - 1, accessing.Item1.Y].Type > 1 && !DynamicPeta[accessing.Item1.X - 1,
accessing.Item1.Y].Accessed)
            {
                stack.Push((DynamicPeta[accessing.Item1.X - 1, accessing.Item1.Y], temp));
            }
        }
        catch (Exception e) { }
        try
        {
            if (DynamicPeta[accessing.Item1.X, accessing.Item1.Y + 1].Type > 1 && !DynamicPeta[accessing.Item1.X,
accessing.Item1.Y + 1].Accessed)
            {
                stack.Push((DynamicPeta[accessing.Item1.X, accessing.Item1.Y + 1], temp));
            }
        }
        catch (Exception e) { }
        try
        {
            if (DynamicPeta[accessing.Item1.X + 1, accessing.Item1.Y].Type > 1 && !DynamicPeta[accessing.Item1.X + 1,
accessing.Item1.Y].Accessed)
            {
                stack.Push((DynamicPeta[accessing.Item1.X + 1, accessing.Item1.Y], temp));
            }
        }
        catch (Exception e) { }
        accessing.Item1.NumAccessed++;
    }
    DynamicPeta[accessing.Item1.X, accessing.Item1.Y].Accessed = true;
  }
}
```

II. Penjelasan Struktur Data

1. Cell

Kelas Cell merepresentasikan satuan luas dari peta maze.

Atribut	Deskripsi
- int x	Koordinat sumbu x

- int y	Koordinat sumbu y
- int type	Tipe dari Cell. Atribut type bernilai 0 untuk Cell start, 1 untuk Cell mati, 2 untuk Cell hidup, dan 3 untuk Cell harta karun.
- bool accessed	Status apakah Cell telah diakses
- int numAccessed	Berapa kali suatu Cell diakses

Method	**Deskripsi**
+ Cell(char c, int _x, int _y)	Konstruktor dengan 1 parameter char untuk menentukan tipe dan 2 parameter int untuk menentukan koordinat x dan y.
+ Cell(Cell other)	Copy constructor untuk struktur data Cell.
+ int X{get; set;}	Berturut-turut mengambil nilai x dan mengubah nilai x menjadi nilai yang ditentukan
+ int Y{get; set;}	Mengambil nilai y atau mengubahnya.
+ bool Accessed{get;set;}	Mengambil nilai accessed atau mengubahnya
+ int NumAccessed {get;set;}	Mendapatkan atau mengubah data dari attribut numAccessed
+ int Type{get;set;}	Mendapatkan atau mengubah nilai dari attribut type

2. Peta

Kelas Cell merepresentasikan peta harta karun yang akan ditelusuri.

Atribut	**Deskripsi**
- List<List<Cell>> map	Representasi data dari peta
- Cell start	Cell yang menyimpan informasi terkait titik awal dari peta.

- int numTreasure	Jumlah harta karun yang tersebar pada peta

Method	Deskripsi
+ Peta()	**Konstruktor dari peta**
+ Cell Start{get;set}	Mengubah atau mengakses start
+ int NumTreasure{get;set}	Getter dan setter untuk atribut numTreasure
+ void readFrom(string file)	Prosedur menerima parameter nama file dalam string, kemudian membaca string tersebut. Input juga menvalidasi apakah ada karakter invalid atau jumlah baris yang tidak konsisten.
+ Cell GetValue(int x,int y)	Mengakses data Cell berdasarkan parameter koordinat
+ void SetValue (int x,int y,Cell other)	Mengubah data Cell pada koordinat tertentu
+ Cell this[int x,int y]{get;set;}	Mengakses atau mengubah data Cell berdasarakan parameter koordinat
+ int GetNumRow()	Mengembalikan jumlah baris dari peta
+ int GetNumCol()	Mengembalikan jumlah kolom dari peta
+ void Inaccess()	Mengubah nilai accessed pada semua Cell pada peta menjadi false.
+ void Copy(Peta other)	Menyalin data peta lain

3. Route

Kelas Route merepresentasikan rute perjalanan dari sebuah Cell ke Cell lainnya.

Atribut	Deskripsi
- List<Cell> jalur	Menyimpan cell yang membentuk Route

Method	Deskripsi
+ Route()	Konstruktor dari Route

+ Route(Route other)	Copy konstruktor dari struktur data Route
+ void Copy(Route other)	Mengosongkan jalur yang disimpan pada sebuah Route dan menyalin jalur dari Route lain ke Route yang telah dikosongkan tersebut
+ Cell this[int x] {get;set;}	Mengakses atau mengubah data Route berdasarkan parameter indeks
+ void AddCell(Cell cell)	Menambahkan satu Cell ke data Route

4. RouteFinder

Kelas RouteFinder adalah sebuah kelas abstrak yang menyelesaikan persoalan maze dengan harta karun. RouteFinder diturunkan menjadi kelas BFS serta DFS.

Atribut	**Deskripsi**
# Peta peta	Menyimpan peta yang didapat dari input file
# Route solusi	Menyimpan jalur yang ditempuh dalam pencarian treasure dimulai dari titik start
# Peta DynamicPeta	Menyimpan peta yang berubah seiring proses pencarian *treasure.*

Method	**Deskripsi**
+ RouteFinder()	Konstruktor dari RouteFinder
+ PetaAccessor {get}	Mengakses atribut peta dari RouteFinder
+ Solusi {get}	Mengakses atribut solusi dari RouteFinder
+ abstract void Solve(bool TSP)	Method pure virtual untuk menyelesaikan persoalan perncarian treasure
+ void readFrom(string file)	Memanggil fungsi readFrom(string file) dari Peta.

+SumNodesChecked()	Mengembalikan berapa kali pemeriksaan Cell terjadi.

5. BFS

Kelas BFS menyelesaikan persoalan maze dengan harta karun dengan algoritma Breadth First Search.

Atribut	Deskripsi
- Queue <(Cell,Route)> queue	Antrian yang menyimpan Cell yang akan diproses serta Route yang telah dilalui untuk mencapai Cell tersebut.

Method	Deskripsi
+ BFS()	Konstruktor untuk BFS
+ void Solve(bool TSP)	Mencari sebuah rute dengan algoritma BFS untuk mendapatkan semua treasure. Jika TSP bernilai true, alogoritma akan mencari rute balik setelah mendapatkan treasure terakhir. Solusi rute yang didapatkan akan disimpan di atribut solusi

6. DFS

Kelas DFS menyelesaikan persoalan maze dengan harta karun dengan algoritma Depth First Search.

Atribut	Deskripsi
- Stack <(Cell,Route)> stack	Stack yang menyimpan Cell yang akan diproses serta Route yang telah dilalui untuk mencapai Cell tersebut

Method	Deskripsi
+ DFS()	Konstruktor untuk DFS

+ void Solve(bool TSP)	Mencari sebuah rute dengan algoritma DFS untuk mendapatkan semua treasure. Jika TSP bernilai true, algoritma akan mencari rute balik setelah mendapatkan treasure terakhir. Solusi rute yang didapatkan akan disimpan di atribut solusi

7. RowInconsistentException

Kelas RowInconsistentException dibuat untuk menghandle error yang disebabkan oleh input file yang memiliki ukuran peta yang tidak N x M (bukan persegi atau persegi panjang). Kelas ini merupakan turunan dari kelas Exception bawaan C#.

Method	Deskripsi
+ RowInconsistentException()	Konstruktor untuk RowInconsistentException

8. InvalidCharacterExpression

Kelas InvalidCharacterExpression dibuat untuk menghandle error yang disebabkan oleh input file character diluar yang diperbolehkan (yang diperbolehkan adalah K, R, T, X). Kelas ini merupakan turunan dari kelas Exception bawaan C#.

Method	Deskripsi
+ InvalidCharacterExpression ()	Konstruktor untuk InvalidCharacterExpression

9. MainWindow

Kelas ini merupakan kelas UI dari program. Kelas ini terhubung langsung dengan file XAML dari UI program.

Atribut	Deskripsi
+ bool IsBFS	Boolean untuk menandakan apakah algoritma BFS dipilih pada UI program
+ bool IsDFS	Boolean untuk menandakan apakah algoritma DFS dipilih pada UI program
+ bool IsTSP	Boolean untuk menandakan apakah permasalahan TSP dipilih pada UI program
+ static Grid MapGrid	Atribut untuk menyimpan keadaan peta pada UI

+ String filePath	Atribut untuk menyimpan path dari input file txt

Method	Deskripsi
+ MainWindow()	Konstruktor untuk MainWindow
+ void ResetMapColor()	Mengubah pewarnaan dari peta ke keadaan semula
- void OpenFileButton(object sender, RoutedEventArgs e)	Implementasi button untuk memilih file input txt, kemudian menampilkan peta yang sesuai dengan file input.
- void VisualiseButton(object sender, RoutedEventArgs e)	Menjalankan proses pencarian berdasarkan algoritma yang dipilih kemudian memvisualisasikan hasilnya.

III. Tata Cara Penggunaan Program

Berikut beberapa dependencies yang dibutuhkan dalam setup program:

1. .NET Framework 6.0
2. MSBuild 17.4.1 (Dapat diunduh bersamaan dengan Visual Studio 2022)

Dengan Visual Studio, program dapat langsung di build sekaligus dijalankan dengan cara membuka file MazeSolver.sln pada folder src kemudian tekan tombol start pada Visual Studio. Program juga dapat dibuild secara terpisah pada terminal di Visual Studio dengan command msbuild pada folder src.

Program juga dapat dijalankan secara langsung tanpa dibuild dengan menggunakan build yang sudah ada pada folder bin. Masuk ke dalam folder bin, kemudian jalankan MazeSolver.exe.

Berikut tata cara penggunaan program:

1. Buka aplikasi dengan cara menjalankan executable MazeSolver.exe pada folder bin.
2. Klik tombol open file untuk memilih file .txt yang akan digunakan sebagai peta.
3. Pilih algoritma yang akan digunakan untuk proses pencarian. BFS atau DFS harus dipilih salah satu, sedangkan untuk TSP boleh dipilih ataupun tidak.
4. Lakukan pencarian dengan klik tombol “Visualize”

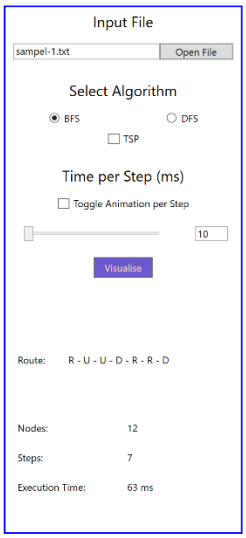
5. Rute yang sudah didapat akan ditampilkan pada peta dengan warna hijau, kemudian ditampilkan juga rute arah gerak, frekuensi pemeriksaan node, panjang rute yang didapat, dan juga *execution time* dari program.

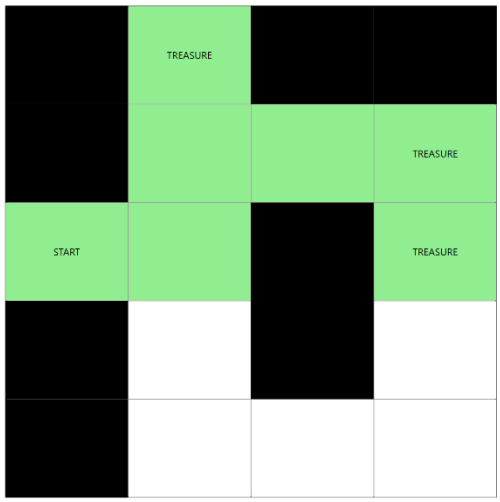
IV. Analisis Desain Solusi Algoritma

1. Test Case

1.1 Test Case 1

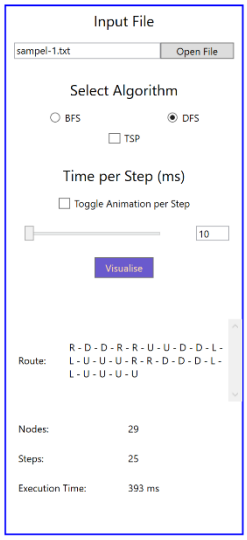
- BFS

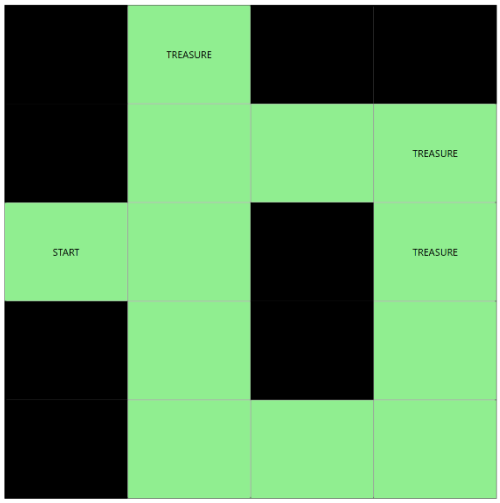




Gambar 4.1.1.1 Test Case 1 dengan BFS

- DFS

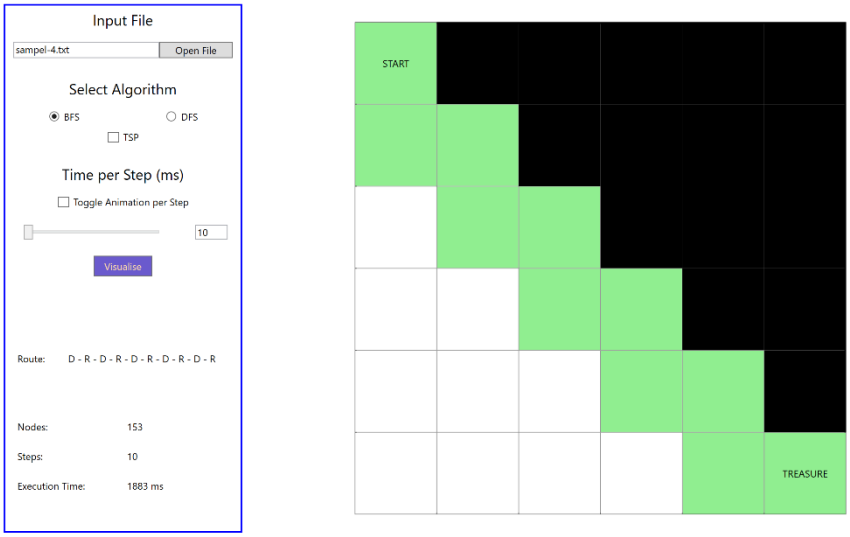




Gambar 4.1.1.2 Test Case 1 dengan DFS

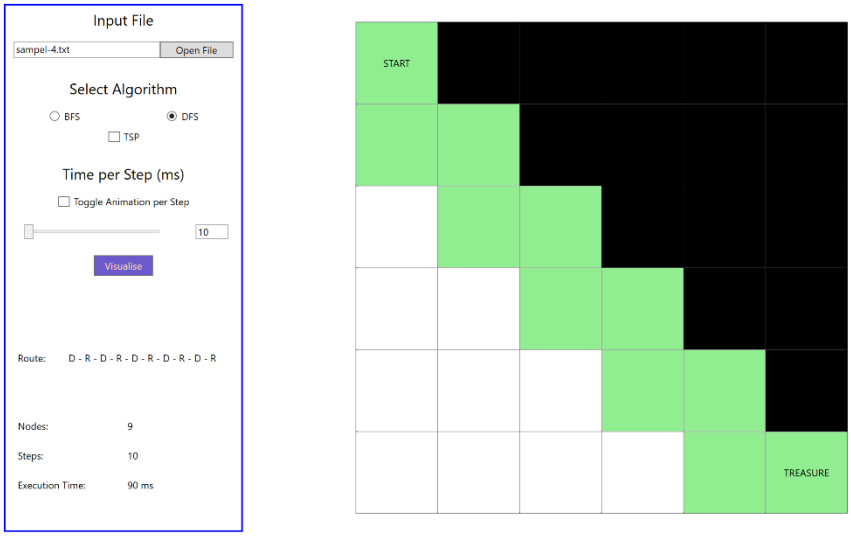
1.2 Test Case 2

- BFS



Gambar 4.1.2.1 Test Case 2 dengan BFS

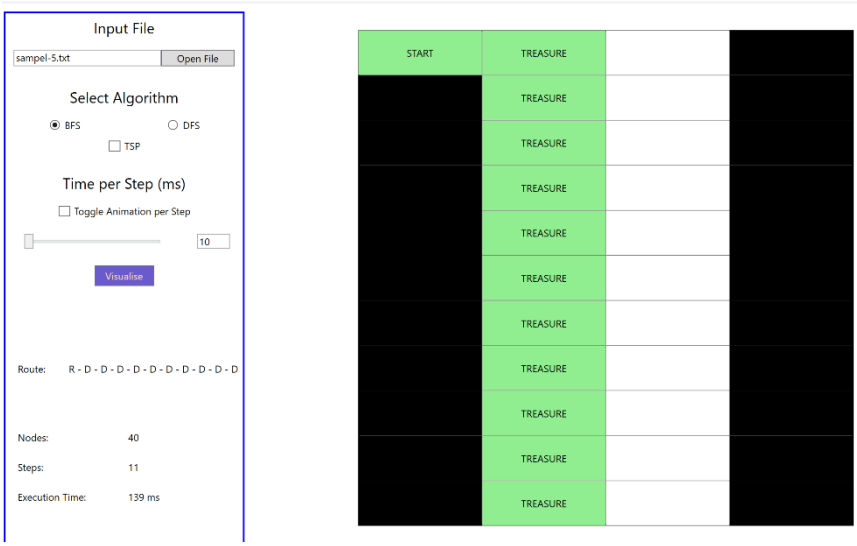
- DFS



Gambar 4.1.2.2 Test Case 2 dengan DFS

1.3 Test Case 3

- BFS



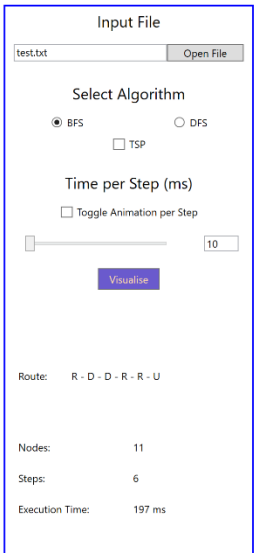
Gambar 4.1.3.1 Test Case 3 dengan BFS

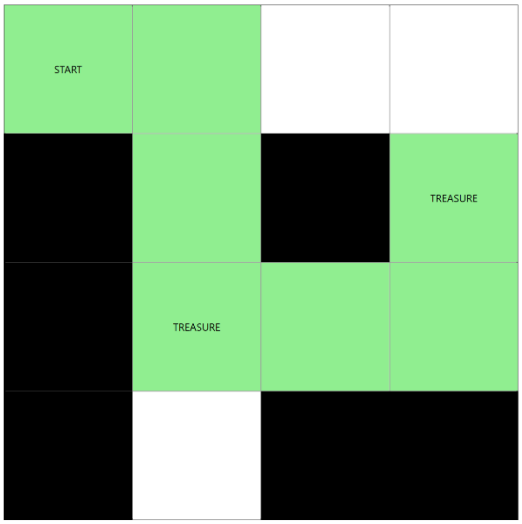
- DFS

Gambar 4.1.3.2 Test Case 3 dengan DFS

1.4 Test Case 4

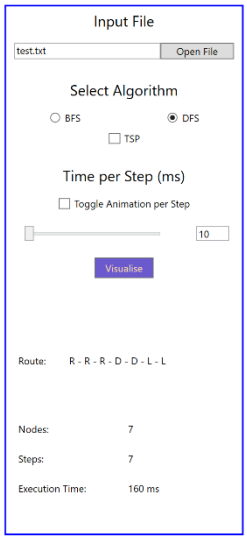
- BFS

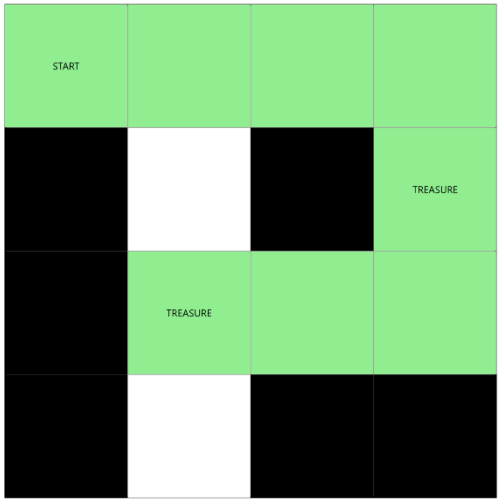




Gambar 4.1.4.1 Test Case 4 dengan BFS

- DFS

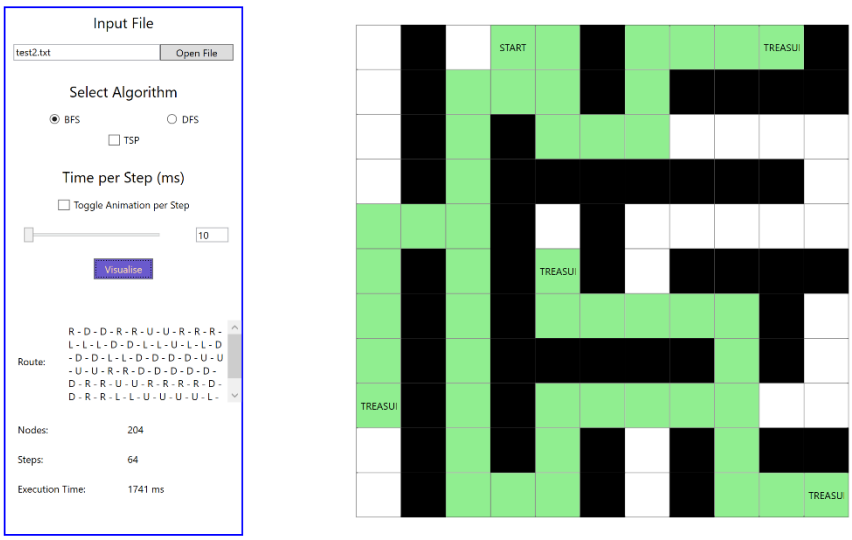




Gambar 4.1.4.2 Test Case 4 dengan DFS

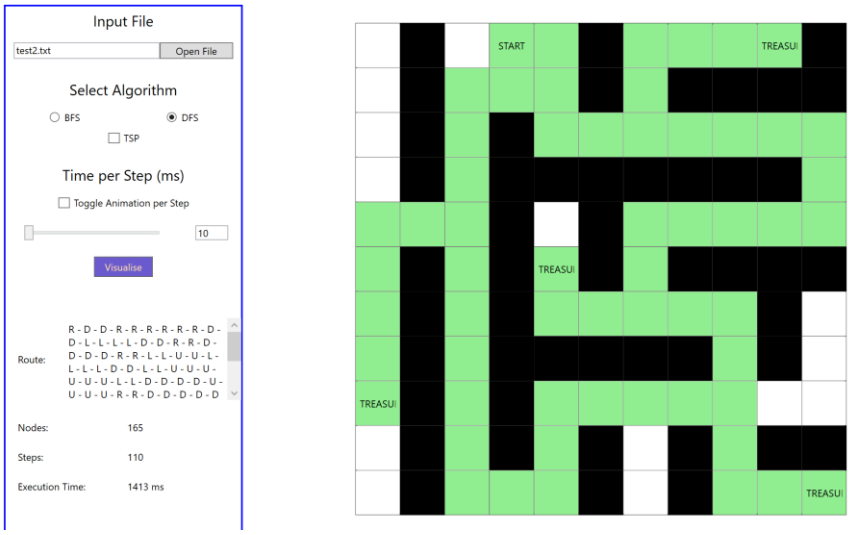
1.5 Test Case 5

- BFS



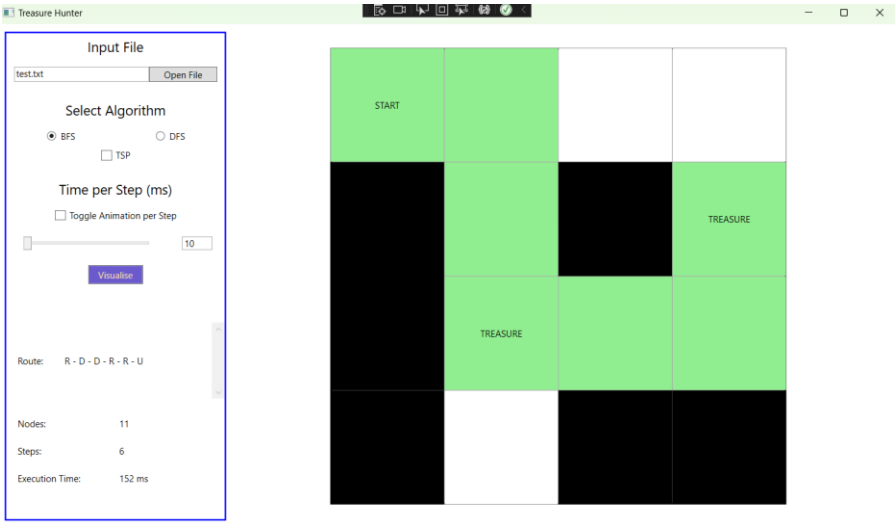
Gambar 4.1.5.1 Test Case 5 dengan BFS

- DFS



Gambar 4.1.5.2 Test Case 5 dengan DFS

1.6 Bonus Test Case untuk TSP



Gambar 4.1.6.1 Bonus Test Case dengan BFS tanpa persoalan TSP

Gambar 4.1.6.2 Bonus Test Case dengan BFS dengan persoalan TSP

2. Analisis

Dari semua test case tersebut, dapat dilihat bahwa rute yang ditemukan dengan algoritma BFS umumnya lebih pendek dibandingkan algoritma DFS. Hal ini terjadi karena BFS mencari node *treasure*, secara level per level. Akibatnya, node *treasure* akan ditemukan dengan jarak yang sama dengan level dari node tersebut terhadap start. Di sisi lain, algoritma DFS akan memprioritaskan pengecekan pada satu arah, sehingga jika node *treasure* berada di posisi dengan prioritas arah terkecil, maka node *treasure* tersebut akan dilewati. Akibatnya, solusi yang diambil oleh DFS sering kali lebih panjang dan berliku-liku. Walaupun itu, solusi yang berliku-liku bukan berarti waktu eksekusi programnya lebih lambat. Dapat dilihat bahwa,

terdapat lebih dari 3 test case dengan solusi BFS lebih pendek, namun dengan waktu eksekusi yang lebih lama. Waktu eksekusi bergantung pada input peta. Jika pada peta terdapat banyak node yang memiliki banyak tetangga, maka waktu eksekusi BFS akan lebih lambat. Di sisi lain, jika terdapat sebuah jalur yang menyesatkan berdasarkan prioritas arah gerak, maka algoritma DFS akan berjalan lebih lambat.

BAB 5
KESIMPULAN DAN SARAN

I. Kesimpulan

Pada tugas ini, telah dibuat sebuah program pencari harta karun pada sebuah peta labirin dengan menggunakan algoritma BFS (Breadth First Search) dan DFS (Depth First Search). Kedua algoritma berhasil menemukan solusi rute dengan baik walaupun rute yang dibuat bukan rute yang paling optimal. Program kemudian dikemas dengan GUI sederhana untuk mempermudah interaksi pengguna dengan program. GUI yang dibuat berhasil menggambarkan jalur yang dapat dilalui untuk mendapatkan semua harta karun, baik secara BFS atau DFS.

II. Saran

Jika memungkinkan, ada baiknya jika status peta dan antrian atau stack tidak direset setiap kali *treasure* ditemukan melainkan dimanfaatkan kembali untuk pencarian berikutnya. Sistem tampilan pada UI juga dapat diubah dengan memanfaatkan skema MVVM (Model-View-ViewModel) supaya terdapat pemisahan antara logic pada UI dan juga data-data pada kelas.

III. Refleksi

Sebaiknya pembagian tugas dilakukan sejak jauh hari supaya pengerjaan bisa lebih terstruktur, terencana, dan tidak mendekati deadline.

IV. Tanggapan pada tugas besar ini

GUI-nya meresahkan.

DAFTAR PUSTAKA

1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf
2. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf
3. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
4. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

Link *Repository* github

hanifmz07/Tubes2_TreasureHunter: Tugas Besar 2 Strategi Algoritma (BFS dan DFS) (github.com)