

Laporan Tugas Besar 2
IF2211 Strategi Algoritma
Pengaplikasian Algoritma BFS dan DFS
dalam Menyelesaikan Persoalan Maze Treasure Hunt



Oleh:

Kelompok 50 - Acetone

Christophorus Dharma Winata	13521009
Asyifa Nurul Shafira	13521125
Cetta Reswara Parahita	13521133

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG BANDUNG
2023

DAFTAR ISI

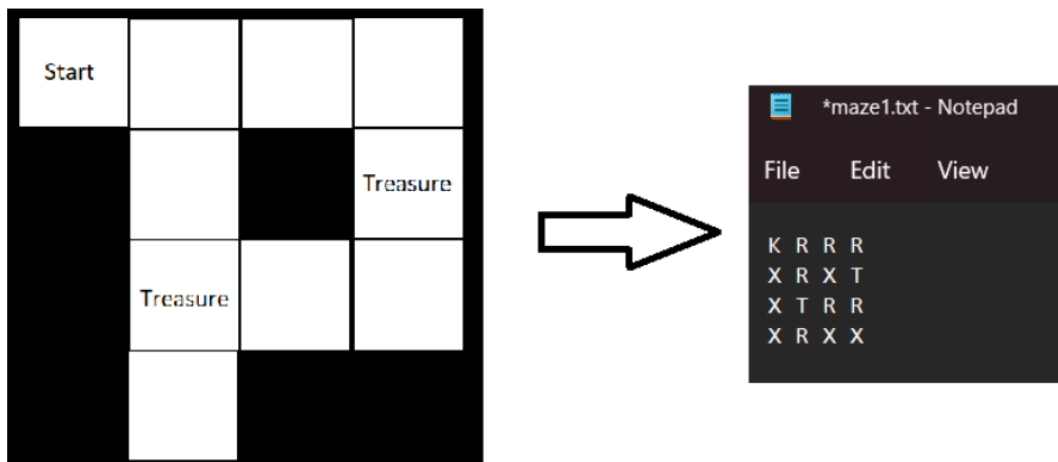
DAFTAR ISI	2
Bab 1 Deskripsi tugas	3
Bab 2 Landasan Teori	7
2.1 Graph Traversal, BFS, dan DFS	7
2.2 C# Desktop Application Development	10
2.3 Travelling Salesman Problem(TSP)	10
Bab 3 Pembahasan	12
3.1 Langkah-langkah Pemecahan Masalah	12
3.1.1 Pencarian Solusi dengan BFS	12
3.1.2 Pencarian Solusi dengan DFS	12
3.1.3 Pencarian Solusi dengan TSP	13
3.2 Proses Mapping Persoalan Menjadi Elemen-elemen Algoritma BFS dan DFS	14
3.3 Contoh Ilustrasi Kasus Lain	14
Bab 4 Analisis Pemecahan Masalah	15
4.1 Implementasi program	15
4.2 Struktur Data dan Spesifikasi Program	18
4.3 Tata Cara Penggunaan Program (interface program, fitur-fitur yang disediakan program, dan sebagainya)	21
4.4 Hasil pengujian	21
4.5 Analisis Desain Solusi	23
Bab 5 Kesimpulan dan Saran	24
5.1 Kesimpulan	24
5.2 Saran	24
5.3 Refleksi	24
5.4 Tanggapan	24
Daftar Pustaka	25
Lampiran	26

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 :

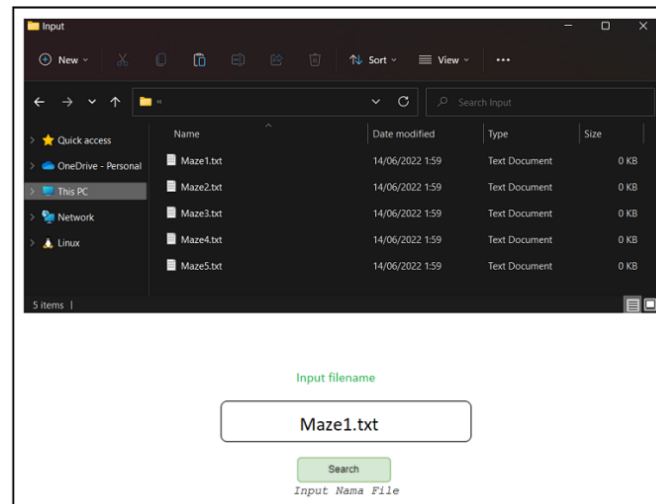


Gambar 1. Ilustrasi input file maze

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.

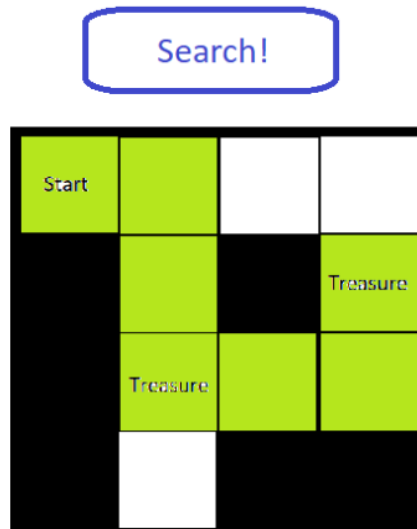
Contoh input aplikasi :



Gambar 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 handle kasus apabila tidak ditemukan dengan nama file tersebut.

Contoh output Aplikasi :

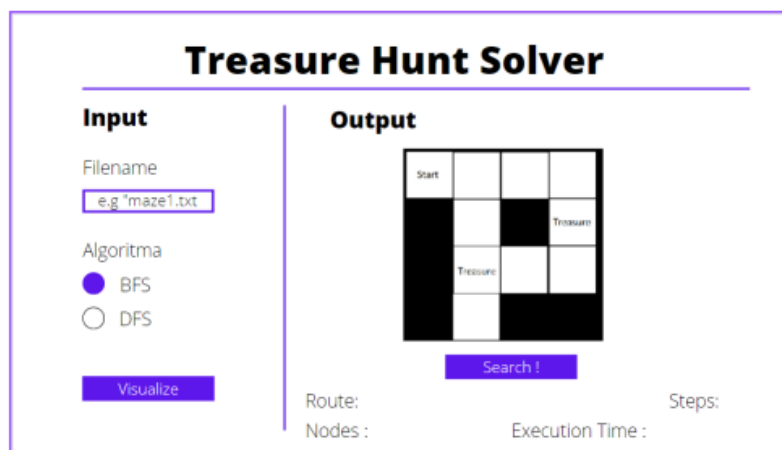


Gambar 3. Contoh output program untuk gambar 2

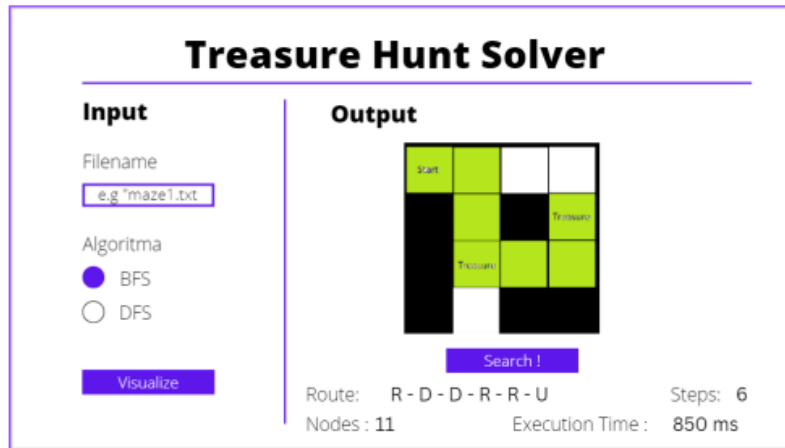
Setelah program melakukan pembacaan input, program akan memvisualisasikan gridnya terlebih dahulu tanpa pemberian rute solusi. Hal tersebut dilakukan agar pengguna dapat mengerjakan terlebih dahulu treasure hunt secara manual jika diinginkan. Kemudian, program menyediakan tombol solve untuk mengeksekusi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

Spesifikasi Program:

Aplikasi yang akan dibangun dibuat berbasis GUI. Berikut ini adalah contoh tampilan dari aplikasi GUI yang akan dibangun



Gambar 4. Tampilan Program Sebelum dicari solusinya



Gambar 5. Tampilan Program setelah dicari solusinya

Catatan: Tampilan diatas hanya berupa contoh layout dari aplikasi saja, untuk design layout aplikasi dibebaskan dengan syarat mengandung seluruh input dan output yang terdapat pada spesifikasi.

Bab 2 Landasan Teori

2.1 Graph Traversal, BFS, dan DFS

Graf adalah sekumpulan objek terstruktur dimana beberapa pasangan objek mempunyai hubungan atau keterkaitan tertentu. Graf terdiri dari himpunan objek-objek yang dinamakan titik, simpul, atau sudut, yang dihubungkan oleh penghubung yang disebut garis atau sisi. Traversal adalah proses kunjungan dalam graf atau pohon, dengan setiap node hanya dikunjungi tepat satu kali.

Traversal graf memiliki arti mengunjungi simpul-simpul dengan cara yang sistematis. Dalam traversal graf ini terdapat dua cara, yaitu:

1. Pencarian Melebar (Breadth First Search / BFS)
2. Pencarian Mendalam (Depth First Search / DFS)

Dalam kedua cara traversal graf ini, digunakan asumsi bahwa tiap simpul/node/titik setidaknya terhubung dengan satu simpul/node/titik lainnya dan graf merupakan graf terhubung. Traversal graf ini termasuk algoritma pencarian solusi, dimana algoritma pencarian solusi ini terbagi menjadi dua jenis, yaitu:

1. Tanpa Informasi (uninformed / blind search): Tidak ada informasi tambahan. Contohnya adalah DFS, BFS, Depth Limited Search, Iterative Deepening Search, dan Uniform Cost Search.
2. Dengan Informasi (informed search): Pencarian berbasis heuristik. Pada algoritma ini, diketahui non-goal state yang “lebih menjanjikan” daripada yang lain. Contohnya adalah Best First Search dan A*.

Dalam proses pencarian solusi, terdapat dua pendekatan:

1. Graf Statis: Graf sudah terbentuk sebelum proses pencarian dilakukan(direpresentasikan dengan struktur data).
2. Graf Dinamis: Graf baru terbentuk saat proses pencarian sedang dilakukan(tidak tersedia sebelum pencarian).

BFS

Algoritma BFS (Breadth First Search) adalah salah satu algoritma pencarian jalur sederhana, dimana pencarian dimulai dari simpul awal, kemudian dilanjutkan ke semua simpul

yang bertetangga dengan simpul awal secara terurut. Jika simpul tujuan belum ditemukan, maka perhitungan akan diulang lagi ke masing-masing simpul cabang dari masing-masing simpul, sampai simpul tujuan ditemukan.

Berikut ini adalah mekanisme umum pencarian menggunakan algoritma BFS:

- Traversal dimulai dari simpul v .
- Algoritma:
 1. Kunjungi simpul v
 2. Kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu.
 3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul -simpul yang tadi dikunjungi, demikian seterusnya.

Berikut adalah algoritma BFS secara umum:

ALGORITHM *BFS*(G)

//Implements a breadth-first search traversal of a given graph

//Input: Graph $G = \langle V, E \rangle$

//Output: Graph G with its vertices marked with consecutive integers

// in the order they are visited by the BFS traversal

mark each vertex in V with 0 as a mark of being “unvisited”

$count \leftarrow 0$

for each vertex v in V **do**

if v is marked with 0

$bfs(v)$

$bfs(v)$

//visits all the unvisited vertices connected to vertex v

//by a path and numbers them in the order they are visited

//via global variable $count$

$count \leftarrow count + 1$; mark v with $count$ and initialize a queue with v

while the queue is not empty **do**

for each vertex w in V adjacent to the front vertex **do**

if w is marked with 0

$count \leftarrow count + 1$; mark w with $count$

 add w to the queue

 remove the front vertex from the queue

DFS

DFS (Depth First Search) adalah salah satu algoritma penelusuran struktur graf / pohon berdasarkan kedalaman. Simpul ditelusuri dari root kemudian ke salah satu simpul anaknya (misalnya prioritas penelusuran berdasarkan anak pertama / simpul sebelah kiri), maka penelusuran dilakukan terus melalui simpul anak pertama dari simpul anak pertama level sebelumnya hingga mencapai level terdalam.

Setelah sampai di level terdalam, penelusuran akan kembali ke 1 level sebelumnya untuk menelusuri simpul anak kedua pada pohon biner / simpul sebelah kanan, lalu kembali ke langkah sebelumnya dengan menelusuri simpul anak pertama lagi sampai level terdalam dan seterusnya.

Berikut ini adalah mekanisme umum pencarian menggunakan algoritma DFS:

- Traversal dimulai dari simpul v.
- Algoritma:
 1. Kunjungi simpul v
 2. Kunjungi simpul w yang bertetangga dengan simpul v.
 3. Ulangi DFS mulai dari simpul w.
 4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
 5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Berikut adalah algoritma DFS secara umum:

ALGORITHM *DFS*(*G*)

```
//Implements a depth-first search traversal of a given graph
//Input: Graph  $G = \langle V, E \rangle$ 
//Output: Graph  $G$  with its vertices marked with consecutive integers
//      in the order they are first encountered by the DFS traversal
mark each vertex in  $V$  with 0 as a mark of being “unvisited”
count  $\leftarrow 0$ 
for each vertex  $v$  in  $V$  do
    if  $v$  is marked with 0
        dfs( $v$ )

dfs( $v$ )
//visits recursively all the unvisited vertices connected to vertex  $v$ 
//by a path and numbers them in the order they are encountered
//via global variable count
count  $\leftarrow$  count + 1; mark  $v$  with count
for each vertex  $w$  in  $V$  adjacent to  $v$  do
    if  $w$  is marked with 0
        dfs( $w$ )
```

2.2 C# Desktop Application Development

C# Desktop Application Development adalah salah satu dari banyak cara untuk membuat aplikasi desktop dengan bahasa C#. Terdapat beberapa framework untuk membuat Desktop Application, salah satunya adalah menggunakan .NET MAUI. Framework .NET MAUI sendiri memungkinkan kita untuk mengembangkan aplikasi yang dapat berjalan di Android, iOS, macOS, dan Windows dari satu basis kode bersama. IDE yang digunakan untuk membuat C# Desktop Application Development ini salah satunya adalah Microsoft Visual Studio. Pada Tugas Besar ini, kami menggunakan .NET MAUI App untuk merealisasikan GUI (Graphical User Interface) dari program yang dibuat. Dengan memanfaatkan Visual Studio.

2.3 Travelling Salesman Problem(TSP)

Travelling Salesman Problem (TSP) adalah permasalahan optimasi klasik yang melibatkan seorang salesman yang harus melakukan kunjungan sekali pada semua kota dalam

sebuah rute sebelum kembali ke titik awal. Penentuan rute perjalanan merupakan salah satu permasalahan yang sering dihadapi dalam kehidupan sehari-hari, seperti menentukan rute yang paling murah untuk dilalui seorang salesman. Permasalahan TSP dapat diselesaikan dengan beberapa cara, salah satunya adalah dengan menggunakan metode Nearest Neighbour.

Metode Nearest Neighbour merupakan metode sederhana untuk menyelesaikan masalah TSP. Langkah-langkah dalam pembentukan rute dengan metode ini diawali dengan inisialisasi, seperti menentukan satu titik awal dan himpunan titik yang akan dikunjungi. Kemudian, langkah selanjutnya adalah memilih titik tujuan yang memiliki jarak terdekat dengan titik sebelumnya. Setelah seluruh titik dikunjungi, rute perjalanan ditutup dengan kembali ke titik asal.

Bab 3 Pembahasan

3.1 Langkah-langkah Pemecahan Masalah

Implementasi pemecahan masalah menggunakan algoritma DFS, BFS, dan TSP untuk menyelesaikan permasalahan *treasure hunt* diselesaikan pada file spesifik yakni MainPage.xaml dan MainPage.xaml.cs. Kedua file ini menyimpan fungsi yang memberikan kembalian DFS, BFS, dan TSP untuk selanjutnya divisualisasikan pada tampilan MAUI.

3.1.1 Pencarian Solusi dengan BFS

Untuk menyelesaikan pencarian harta karun menggunakan dasar algoritma BFS, maka akan dibuat beberapa fungsi penunjang antara lain *checkTreasure(Node, int)*, *checkAnyChild(Node)*, dan *BFSSearch(Node[,], int)*. Fungsi *BFSSearch* adalah fungsi utama yang akan melakukan pencarian jumlah harta karun yang telah ditemukan, node-node pada satu level yang sama dan sedang dicari, serta total semua node yang dicari. Akan dilakukan pengulangan pencarian pada level yang lebih bawah untuk setiap node apabila belum ditemukan harta karun sejumlah dengan jumlah hartakarun yang dimasukkan pada parameter.

3.1.2 Pencarian Solusi dengan DFS

Untuk menyelesaikan pencarian harta karun menggunakan dasar algoritma DFS, maka akan dibuat beberapa fungsi penunjang antara lain *nowChecking(Node)*, *DFSSearchRecursive(Node[,], int, Node, Node)*, dan *DFSSearch(Node[,], int)*. Fungsi *nowChecking* adalah fungsi yang menerima sebuah node dari peta dan mengembalikan node yang akan dicek selanjutnya pada pencarian DFS. Fungsi ini memeriksa node anak yang belum pernah di cek mulai dari kanan, bawah, kiri, dan atas secara berurutan. Jika tidak ada anak node yang belum pernah dicek, maka fungsi ini akan mengembalikan node yang diberikan sebagai input.

DFSSearchRecursive adalah sebuah fungsi yang melakukan pencarian DFS rekursif pada peta dan mengembalikan array dari node yang telah dilewati serta jumlah harta karun yang telah ditemukan. Fungsi ini menerima beberapa parameter yaitu peta yang akan dicari, jumlah harta karun yang ingin ditemukan, node saat ini, dan node induk. Fungsi ini melakukan pencarian pada node saat ini dengan memeriksa apakah node tersebut merupakan harta karun atau bukan. Jika iya, maka jumlah harta karun yang

ditemukan akan di-increment. Jika tidak, maka fungsi *nowChecking* akan dipanggil untuk mendapatkan node anak yang akan dicek selanjutnya. Jika node anak yang ditemukan berbeda dengan node saat ini, maka fungsi *DFSSearchRecursive* akan dipanggil kembali pada node anak tersebut. Fungsi ini akan diulang terus menerus sampai semua harta karun yang diinginkan telah ditemukan atau tidak ada node lagi yang dapat dikunjungi. Fungsi ini mengembalikan array dari node yang telah dilewati dan jumlah harta karun yang telah ditemukan.

DFSSearch adalah sebuah fungsi yang melakukan pencarian DFS pada peta dengan memanggil fungsi *DFSSearchRecursive* pada node awal (biasanya node pertama pada peta). Fungsi ini mengembalikan array dari node yang telah dilewati. Fungsi ini digunakan untuk memulai pencarian DFS pada peta.

3.1.3 Pencarian Solusi dengan TSP

Untuk menyelesaikan pencarian harta karun menggunakan dasar algoritma TSP, maka akan dibuat beberapa fungsi penunjang antara lain *restartCheck(Node[,])*, *TSPSteps(Node, Node)*, dan *TSPSearch(Node[,], int)*. Fungsi *restartCheck* akan mengulang status cek pada semua node dalam map. Kegunaannya adalah untuk mengembalikan node pada kondisi semula sebelum dilakukan pencarian jalur terpendek. Fungsi *TSPSteps* akan mencari jalur terpendek dari suatu node start ke node finish pada peta. Kegunaannya adalah untuk mencari jalur terpendek dari satu titik ke titik lainnya. Jika node yang dicek sama dengan finish, fungsi akan menghentikan pencarian.

TSPSearch adalah fungsi utama yang akan melakukan pencarian jalur terpendek untuk mengunjungi beberapa titik pada peta. *TSPSearch* akan mencari jalur terpendek dengan cara mengecek setiap node yang merupakan titik-titik harta karun (yang ingin dikunjungi). Kemudian node-node tersebut diurutkan berdasarkan jarak terdekat dan diproses satu per satu untuk mencari jalur terpendek yang melalui semua titik harta karun tersebut. Setelah itu, akan dilakukan kembali langkah yang sama untuk kembali ke titik awal pada map. Dalam program ini, terdapat beberapa fungsi lain seperti *checkNode()* untuk mengecek node, *concatNode()* untuk menggabungkan node, dan *relativeDistance()* untuk menghitung jarak relatif antara suatu node dengan node lainnya.

3.2 Proses Mapping Persoalan Menjadi Elemen-elemen Algoritma BFS dan DFS

Tabel 3.2 Mapping Elemen-elemen Algoritma BFS dan DFS

Elemen	BFS	DFS
Pohon ruang status	Graf yang terbentuk dari simpul-simpul yang disimpan pada simpul ruang status	
Simpul	Akar : node dengan status startStatus true Daun : node dengan status treasureStatus true	
Cabang	Setiap node yang belum pernah dikunjungi dari <i>parent node</i> di dalam array	Menambahkan setiap node yang telah dikunjungi ke dalam array
Ruang status	Simpul-simpul yang terdapat dalam array	
Ruang solusi	Kumpulan semua jalur yang mungkin menuju setiap harta karun yang terdapat pada maze	

3.3 Contoh Ilustrasi Kasus Lain

Ilustrasi dengan menggunakan Algoritma A* yang merupakan gabungan dari algoritma uniform-cost search dan heuristic search. Dalam menyelesaikan permasalahan pencarian jalur atau pathfinding, algoritma A* akan berulang kali memeriksa lokasi yang paling bernilai untuk dikunjungi. Dalam kasus yang lain, algoritma ini juga membantu mencatat semua tetangga simpul/lokasi sebagai eskplorasi tambahan. A* merupakan algoritma pathfinding yang paling populer pada game AI. Kompleksitas dari algoritma ini tergantung pada fungsi heuristik yang digunakan pada programnya.

Ilustrasi dengan Pendekatan Exhaustive Search Untuk pendekatan dengan exhaustive search, dilakukan enumerasi untuk semua kemungkinan path yang dapat dibentuk dari posisi 'K' hingga mencapai semua 'T' dalam maze. Kemudian simpan path-path tersebut dan jika semua kemungkinan path sudah dienumerasi, pilih 1 path yang memenuhi aturan yang sebelumnya (dari posisi 'K' hingga mencapai semua 'T' dalam maze). Kemudian tampilkan route-nya

Bab 4 Analisis Pemecahan Masalah

4.1 Implementasi program

```
using System.ComponentModel;
using Microsoft.Maui.Graphics;
namespace Tubes2_Acetone;

public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
        BindingContext = new MainViewModel();
    }
    public class MainViewModel : INotifyPropertyChanged
    {
        private string _filePath;

        public string FilePath
        {
            get { return _filePath; }
            set
            {
                if (_filePath != value)
                {
                    _filePath = value;
                    OnPropertyChanged(nameof(FilePath));
                }
            }
        }
        private string _stepSolution;
        public string StepSolution
        {
            get { return _stepSolution; }
            set
            {
                if (_stepSolution != value)
                {
                    _stepSolution = value;
                    OnPropertyChanged(nameof(StepSolution));
                }
            }
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

```

[Obsolete]
private async void OnChooseFileClicked(object sender, EventArgs e)
{
    var result = await FilePicker.PickAsync(new PickOptions
    {
        FileTypes = new FilePickerFileType(new Dictionary<DevicePlatform, IEnumerable<string>>
        {
            { DevicePlatform.macOS, new[] { "txt" } },
            { DevicePlatform.iOS, new[] { "public.plain-text" } },
            { DevicePlatform.Android, new[] { "text/plain" } },
            { DevicePlatform.UWP, new[] { ".txt" } },
        })
    });

    if (result != null)
    {
        FilePathEntry.Text = result.FullPath;
    }
    ((MainViewModel)BindingContext).StepSolution = "0";
}

private void OnLoadFileClicked(object sender, EventArgs e)
{
    string filePath = FilePathEntry.Text;
    ((MainViewModel)BindingContext).FilePath = FilePathEntry.Text;

    if (!string.IsNullOrEmpty(filePath))
    {
        try
        {
            // Read the text file contents
            string[] lines = File.ReadAllLines(filePath);
            for (int i = 0; i < lines.Length; i++)
            {
                MapGrid.RowDefinitions.Add(new RowDefinition() { Height = GridLength.Auto });
            }
            int numColumns = lines[0].Split(' ').Length;
            for (int i = 0; i < numColumns; i++)
            {
                MapGrid.ColumnDefinitions.Add(new ColumnDefinition() { Width = GridLength.Auto });
            }

            for (int row = 0; row < lines.Length; row++)
            {
                string[] words = lines[row].Split(' ');
                for (int col = 0; col < words.Length; col++)
                {

```



```

        /*var label = new Label
        {
            Text = words[col],
            TextColor = Colors.Black,
            BackgroundColor = words[col] == "X" ? Colors.Gray : Colors.White
        };
        MapGrid.Children.Add(label);*/

        // Create an image for each word and add it to the grid
        var image = new Image();
        if (words[col] == "X")
        {
            image.Source = "wall.png";
            image.BackgroundColor = Colors.Gray;
        }
        else if (words[col] == "T")
        {
            image.Source = "treasure.png";
            image.BackgroundColor = Colors.Yellow;
        }
        else if (words[col] == "R")
        {
            image.Source = "visited.png";
            image.BackgroundColor = Colors.Black;
        }
        else if (words[col] == "K")
        {
            image.Source = "treasure.png";
            image.BackgroundColor = Colors.White;
        }

        // Add the image to the grid
        Grid.SetRow(image, row);
        Grid.SetColumn(image, col);
        MapGrid.Children.Add(image);
    }
}
}
catch (Exception ex)
{
    Console.WriteLine($"Error reading file: {ex.Message}");
}
}
}
}
}

```

4.2 Struktur Data dan Spesifikasi Program

Dalam mengimplementasikan program, kami menggunakan beberapa struktur data, diantaranya:

1. Array 2 dimensi untuk menyimpan berbagai entitas yang dihasilkan dan diperlukan dalam keberjalanan program
2. Node secara spesifik untuk merepresentasikan data titik pada peta dengan struktur sebagai berikut:

```
public class Node {  
    // Attribut  
    int i;  
    int j;  
    bool startStatus;  
    bool treasureStatus;  
    int timesChecked;  
    Node? left;  
    Node? right;  
    Node? up;  
    Node? down;  
  
    // Constructor  
    public Node(bool treasureStatus, int i, int j) {  
        this.i = i;  
        this.j = j;  
        this.treasureStatus = treasureStatus;  
        this.timesChecked = 0;  
        this.left = null;  
        this.right = null;  
        this.up = null;  
        this.down = null;  
        this.startStatus = false;  
    }  
  
    // Getter  
    public int getI() {  
        return this.i;  
    }  
  
    public int getJ() {  
        return this.j;  
    }  
  
    public string getPosition() {  
        return "(" + this.i + "," + this.j + ")";  
    }  
  
    public Node? getLeft() {  
        return this.left;  
    }  
}
```

```
}

public Node? getRight() {
    return this.right;
}

public Node? getUp() {
    return this.up;
}

public Node? getDown() {
    return this.down;
}

public bool isChecked() {
    return this.timesChecked > 0;
}

public int getTimesChecked() {
    return this.timesChecked;
}

public bool isTreasure() {
    return this.treasureStatus;
}

public bool isStart() {
    return this.startStatus;
}

// Setter
public void setLeft(Node newleft) {
    this.left = newleft;
}

public void setRight(Node newright) {
    this.right = newright;
}

public void setUp(Node newup) {
    this.up = newup;
}

public void setDown(Node newdown) {
    this.down = newdown;
}

public void checkNode() {
    this.timesChecked++;
}
```

```

public void setStart() {
    this.startStatus = true;
}

public void restartCheck()
{
    this.timesChecked = 0;
}
// Method
public void print() {
    Console.WriteLine("-----");
    Console.WriteLine("This node is in position" + this.getPosition() + " This is a ");
    if (this.startStatus) {
        Console.WriteLine("Start");
    } else {
        if (this.treasureStatus) {
            Console.WriteLine("Treasure");
        } else {
            Console.WriteLine("Road");
        }
    }
    if (this.left != null)
    {
        Console.WriteLine("Left: " + left.isTreasure());
    }

    if (this.right != null)
    {
        Console.WriteLine("Right: " + right.isTreasure());
    }

    if (this.up != null)
    {
        Console.WriteLine("Up: " + up.isTreasure());
    }

    if (this.down != null)
    {
        Console.WriteLine("Down: " + down.isTreasure());
    }
    Console.WriteLine("-----");
}

public double relativeDistance(Node x) {
    double distance = Math.Sqrt((this.i - x.i)*(this.i - x.i) + (this.j - x.j)* (this.j - x.j));
    return distance;
}

// Main check

```

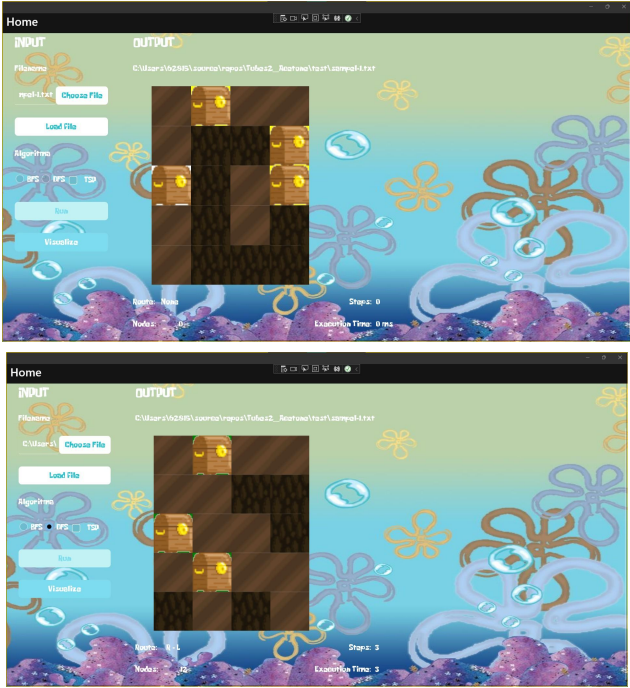
```
//public static void Main(String[] args) {  
  
    //}  
}
```

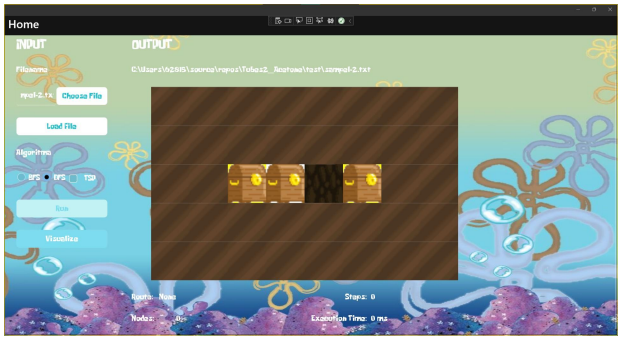
4.3 Tata Cara Penggunaan Program (interface program, fitur-fitur yang disediakan program, dan sebagainya)

Tampilan awal program:

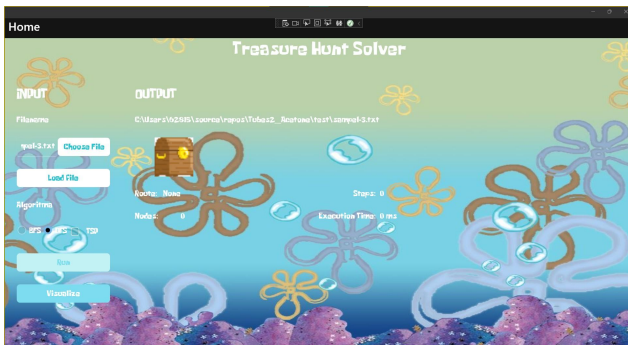
- 1. Pastikan device Anda telah memenuhi pre-requisite untuk menjalankan program kemudian clone repository pada local folder Anda.
- 2. Buka repository yang telah di clone dengan Visual Studio
- 3. Masuk pada bagian folder Program dan pilih child folder yang sesuai dengan device yang Anda gunakan.
- 4. Lakukan debuggin dan running program dengan menekan tombol start di bagian atas.
- 5. Program akan berjalan dan Anda dapat menjalankan program dengan memilih button jenis penelusuran dan menekan button run dan visualize.

4.4 Hasil pengujian

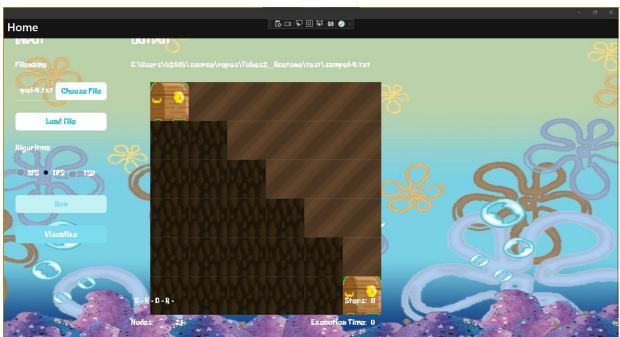
Antarmuka Program	Data Uji
	<div>X T X X</div> <div>X R R T</div> <div>K R X T</div> <div>X R X R</div> <div>X R R R</div>



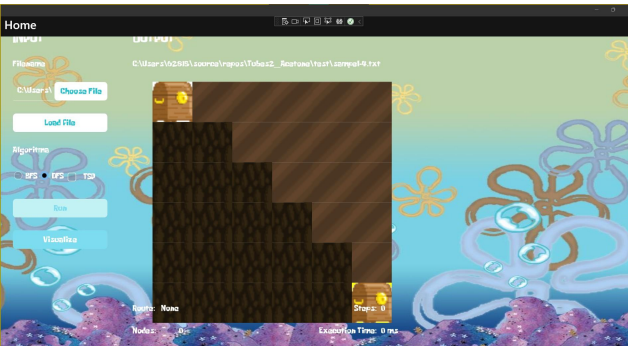
```
X X X X X X X X
X X X X X X X X
X X T K R T X X
X X X X X X X X
X X X X X X X X
```



```
J A N G A N
L U P A C E
K Y A N G B
E G I N I Y
```



```
K X X X X X
R R X X X X
R R R X X X
R R R R X X
R R R R R X
R R R R R T
```





4.5 Analisis Desain Solusi

Design solusi dengan menggunakan algoritma Breadth First Search(BFS) dan Depth First Search(DFS) memiliki kelebihan pada persoalan kasus yang berbeda. Untuk algoritma BFS memiliki kelebihan pada kasus-kasus dimana posisi setiap harta karun dekat dengan titik *start*. Sedangkan, untuk algoritma DFS memiliki kelebihan pada kasus-kasus dimana posisi setiap harta karun berada jauh dari titik *start*.

Bab 5 Kesimpulan dan Saran

5.1 Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma semester 2 2022/2023 berjudul “Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt”, kami berhasil membuat solusi permasalahan yang memanfaatkan algoritma pencarian *Breadth First Search*(BFS) dan *Depth First Search*(DFS). Kedua algoritma tersebut memiliki kelebihan pada kasus yang berbeda. Algoritma BFS akan lebih optimal pada kasus dimana semua *treasure* berada dekat dengan titik *start*. Sedangkan, algoritma DFS lebih optimal pada kasus dimana semua *treasure* berada jauh dengan titik *start*.

5.2 Saran

Saran yang dapat kami berikan untuk tugas besar ini adalah:

1. Lebih banyak melakukan eksplorasi terkait C# Desktop Application Development, mulai dari struktur file, cara kerja, serta kreativitas dalam pembuatan UI
2. Melengkapi code program dengan komentar, untuk mempermudah *debugging*
3. Membuat code program dengan lebih modular

5.3 Refleksi

Setelah menyelesaikan tugas besar 2 IF2211 Strategi Algoritma ini, kami mendapatkan banyak pengetahuan baru, mulai dari lebih mengenal bahasa C# dan penggunaannya dalam pembuatan GUI yang memiliki berbagai macam jenis serta penerapan algoritma BFS, DFS, dan TSP untuk menyelesaikan persoalan pada tugas besar ini.

5.4 Tanggapan

Dari tugas besar 2 IF2211 Strategi Algoritma yang telah diberikan, kenapa harus C# 🤔(◉_◉)🐼📖. Hehe, tapi sebenarnya asik kok kak, cuma C# aja ini sih. Algoritma udah jadi, masukin ke MAUI nya lagi bikin nangis wkwkwk. Tidak apa, namanya juga proses belajar. Terima kasih :D

Daftar Pustaka

Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2021). “Breadth/Depth First Search (BFS/DFS) Bagian 1”. Diakses online dari

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf> pada 24 Maret 2023.

Munir, Rinaldi dan Maulidevi, Nur Ulfa. (2021). “Breadth/Depth First Search (BFS/DFS) Bagian 2”. Diakses online dari

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf> pada 24 Maret 2023.

Davidbritch, Banovvv, dan Jconrey (2023). “Apa itu .NET MAUI?”. Diakses online dari

<https://learn.microsoft.com/id-id/dotnet/maui/what-is-maui#:~:text=.NET%20Multi%2Dplatform%20App%20UI,dari%20satu%20basis%20kode%20bersama>. pada 24 Maret 2023.

Lampiran

Link repository: [christodharma/Tubes2_Acetone \(github.com\)](https://github.com/christodharma/Tubes2_Acetone)

Link youtube: bit.ly/VideonyaAcetone