LAPORAN TUGAS BESAR

STRATEGI ALGORITMA

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt

Disusun Oleh:

Afnan Edsa Ramadhan	13521011
Muhammad Haidar Akita Tresnadi	13521025
Muhammad Zulfiansyah Bayu Pratama	13521028

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

DAFTAR ISI

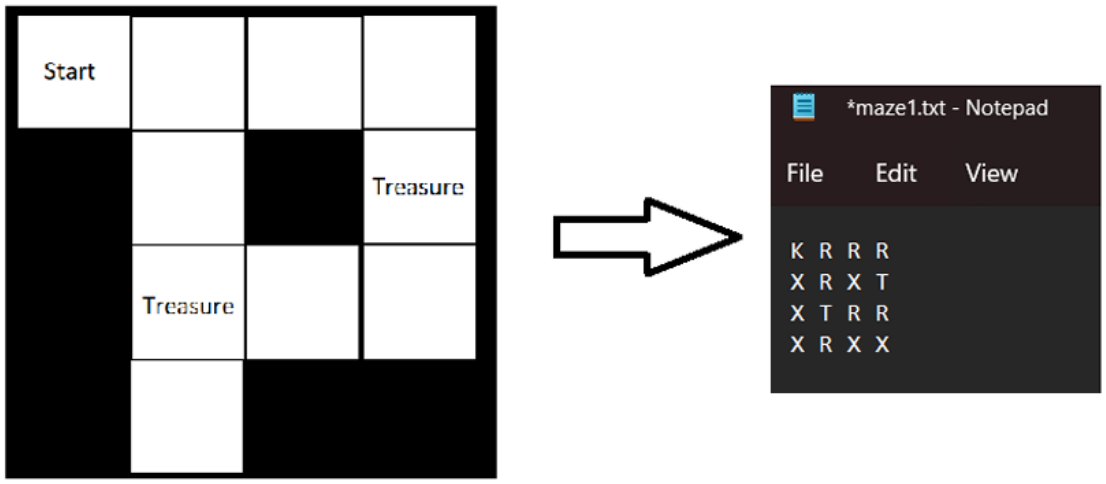
BAB I
DESKRIPSI MASALAH

1.1. Abstraksi

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 :



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 masingmasing kelompok, asalkan dijelaskan di readme / laporan.

BAB II
LANDASAN TEORI

2.1 Graph Traversal

Graph traversal adalah sebuah teknik atau algoritma yang digunakan untuk menelusuri atau melintasi setiap simpul atau node pada sebuah struktur data graph yang terdiri dari banyak simpul dan edge (atau hubungan antar simpul). Graph traversal sangat berguna dalam berbagai bidang seperti pemodelan jaringan, pemrosesan bahasa alami, optimasi, dan pengembangan perangkat lunak. Algoritma pencarian solusi terbagi menjadi dua, yaitu pencarian tanpa informasi (uninformed/blind search) dan pencarian dengan informasi (informed search).Pada blind search, tidak ada informasi tambahan pada simpul tujuan selain yang disediakan di definisi permasalahan, contohnya adalah algoritma breadth first search, depth first search, depth limited search, dan lain-lain. Sedangkan pada informed search mempunyai informasi pada simpul tujuan yang dapat mempercepat pencarian, contohnya adalah algoritma best first search dan A*.

2.1.1 Breadth-First Search

Breadth-First Search (BFS) adalah salah satu algoritma traversal yang digunakan untuk mencari jalur pada sebuah struktur data graph. Algoritma ini bekerja dengan cara menelusuri simpul secara horizontal, artinya mengunjungi semua simpul pada level yang sama terlebih dahulu, baru kemudian bergerak ke level selanjutnya. Pada setiap level, BFS mengunjungi semua simpul yang terhubung dengan simpul sebelumnya dan mencatat jarak atau level dari simpul awal. Algoritma BFS biasanya digunakan untuk mencari jalur antara dua simpul pada grafik, seperti pada masalah navigasi, pemetaan jaringan, atau game dengan ruang keadaan yang kompleks. Dalam implementasinya, BFS menggunakan struktur data antrian (queue) untuk menyimpan simpul yang akan dikunjungi selanjutnya, sehingga simpul yang pertama kali dimasukkan ke antrian akan menjadi yang pertama kali dikunjungi.

2.1.2 Depth-First Search

Depth-First Search (DFS) adalah salah satu algoritma traversal yang digunakan untuk menelusuri setiap simpul pada sebuah struktur data graph secara vertikal atau dalam (depth-first). Algoritma ini bekerja dengan cara memilih simpul awal dan mengunjungi simpul-simpul yang terhubung dengannya secara rekursif, yaitu mengunjungi simpul pertama kemudian melakukan rekursi untuk mengunjungi simpul terdekat dan seterusnya hingga mencapai simpul terakhir. Setelah mencapai simpul terakhir, algoritma akan kembali ke simpul sebelumnya dan mengunjungi simpul terdekat yang belum dikunjungi. Algoritma DFS biasanya digunakan untuk mencari jalur tertentu pada grafik, seperti pada masalah pencarian optimasi, peta jalan, atau pencarian kata dalam kamus. Dalam implementasinya, DFS menggunakan struktur data stack untuk menyimpan simpul yang akan dikunjungi selanjutnya, sehingga simpul yang terakhir dimasukkan ke stack akan menjadi yang pertama kali dikunjungi. Karena algoritma ini hanya mengunjungi simpul terdekat terlebih dahulu, DFS biasanya lebih cepat dan membutuhkan

memori yang lebih sedikit dibandingkan dengan BFS, tetapi belum tentu menemukan jalur terpendek antara dua simpul.

2.2 C# Desktop Application Development

C# Desktop Application Development adalah sebuah software pegembang aplikasi desktop. Pada umumnya, pengembangan aplikasi desktop menggunakan bahasa pemrogramman C++, C#, ataupun Java. Pengembangan desktop application, terlebih dalam hal user interface (UI), dipermudah dengan menggunakan WinForms/WPF dan bantuan integrated development environment (IDE) Visual Studio.

Windows Presentation Foundation (WPF) adalah sebuah kerangka kerja pengembangan aplikasi desktop yang disediakan oleh Microsoft. WPF menggunakan bahasa pemrograman C# dan merupakan bagian dari teknologi .NET Framework. WPF memungkinkan pengembang untuk membuat aplikasi desktop dengan antarmuka pengguna yang modern, termasuk animasi, transisi, efek visual, dan desain responsif. Dalam WPF, antarmuka pengguna (UI) diatur dalam file XAML (eXtensible Application Markup Language) yang memisahkan desain tampilan dari source code. Hal ini memungkinkan pengembang untuk mengubah tampilan UI tanpa harus merubah source code secara langsung. Dengan WPF, pengembang dapat membuat aplikasi desktop yang lebih interaktif, menarik, dan mudah dikembangkan.

BAB III
APLIKASI STRATEGI ALGORITMA BFS DAN DFS

3.1 Langkah-langkah Pemecahan Masalah

Langkah awal dalam proses mendesain solusi dari permasalahan ini adalah dengan membagi permasalahan utama, yaitu pencarian rute, menjadi beberapa permasalahan kecil sehingga dapat mempermudah pencarian solusi. Permasalahan tersebut dapat dibagi menjadi dua permasalahan utama, yaitu pencarian rute menggunakan algoritma BFS dan DFS serta permasalahan tambahan yaitu solusi dari BFS dan DFS harus ditampilkan dalam sebuah aplikasi desktop.

Setelah membagi permasalahan, langkah selanjutnya adalah dengan melakukan pemetaan permasalahan ke dalam elemen-elemen algoritma BFS dan DFS. Setelah melakukan pemetaan elemen-elemen algoritma, dilakukan perancangan algoritma sesuai dengan elemen yang telah dipetakan. Lalu, kami mengimplementasikan algoritma DFS dan BFS pada program, memeriksa, memperbaiki, dan mengembangkan fitur dan program, serta melakukan proses debugging di masing-masing kategori dan program. Terakhir kami mengimplementasikan program yang sudah dibuat ke dalam GUI.

3.2 Proses Mapping

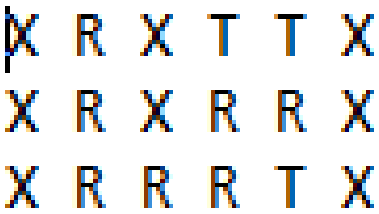
3.2.1. Breadth-First Search

Algoritma breadth-first search pada permasalahan ini menggunakan konsep pencarian solusi dari graf dinamis. Dalam merepresentasikan graf, digunakan daftar ketetanggan atau adjacency list. Elemen antrian atau queue berisi dengan simpul atau node yang akan dikunjungi.

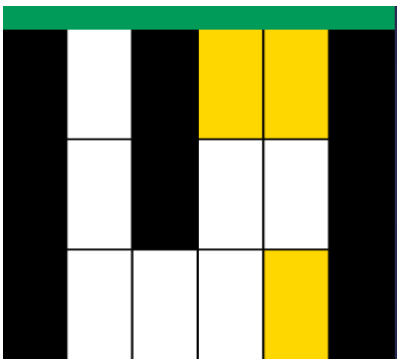
3.2.2. Depth-First Search

Algoritma depth-first search pada permasalahan ini menggunakan konsep pencarian solusi dari graf dinamis. Dalam merepresentasikan graf, digunakan daftar ketetanggan atau adjacency list. Elemen tumpukan atau stack berisi dengan simpul atau node yang akan dikunjungi.

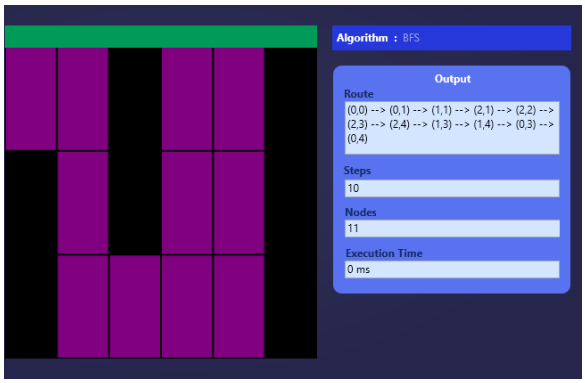
3.3 Ilustrasi Kasus Lain



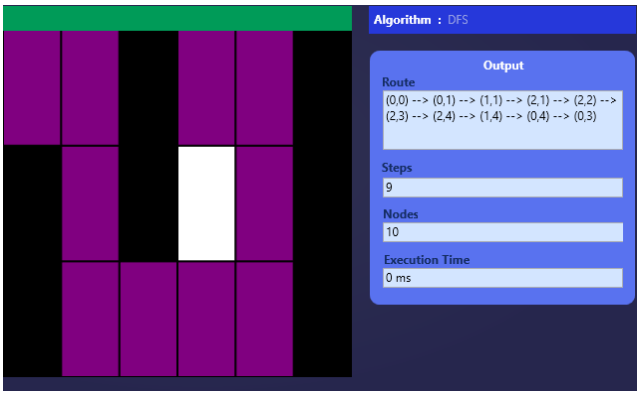
Gambar 3.3.1. Isi dari file txt



Gambar 3.3.2. Visualisasi Maze yang diberikan



Gambar 3.3.3. Hasil BFS



Gambar 3.3.4. Hasil DFS

Pada contoh ilustrasi kasus lain, kami menggunakan map dari file txt sesuai yang terdapat pada gambar 3.3.1. Ketika solusinya dicari menggunakan algoritma DFS dan BFS, kami mendapatkan hasil yang sesuai. Untuk BFS bisa dilihat pada gambar 3.3.3. Sedangkan Untuk DFS solusinya bisa dilihat seperti pada gambar 3.3.4.

BAB IV
ANALISIS PEMECAHAN MASALAH

4.1 Implementasi Program

4.1.1 DFS

```
function DFS(char[,] map)

KAMUS
    treasureCount, count : integer
    currKoor, temp2 : class koor
    avail, visited : stack of class koor
    treasure : list of class koor

ALGORITMA
    avail.Push(start)
    treasureCount <- treasure.Count
    count <- 0

    while(avail.Count>0)do
        currKoor <- new koor(avail.Pop())
        visited.Push(currKoor)
        if(map[currKoor.X,currKoor.Y] = 'T') then
            count++
            treasure.Remove(currKoor)
            if(count = treasureCount)then
                break;
            endif
        endif
        foreach(var i in "DLUR")
            temp2 <- new koor(currKoor)
            if(i='R')then
                if(currKoor.Y < col-1)then
                    if(map[currKoor.X,currKoor.Y+1] != 'X' ) then
                        temp2 <- new koor(currKoor.X,currKoor.Y+1)
                    endif
                endif
            else if(i='L')then
                if(currKoor.Y>0)then
                    if(map[currKoor.X,currKoor.Y-1] != 'X')then
                       temp2 <- new koor(currKoor.X,currKoor.Y-1)
                    endif
                endif
            else if(i='U')then
                if(currKoor.X > 0)then
                    if(map[currKoor.X-1,currKoor.Y] != 'X')then
                       temp2 <- new koor(currKoor.X-1,currKoor.Y)
                    endif
                endif
            else if(i='D') then
                if(currKoor.X < row-1) then
                   if(map[currKoor.X+1,currKoor.Y] != 'X')then
                       temp2 <- new koor(currKoor.X+1,currKoor.Y)
```

```
                endif
            endif
        endif

        if(visited.Contains(temp2))endif
            continue
        endif
        avail.Push(temp2)
    endwhile

    return visited
```

4.1.2 BFS

```
function BFS(char[,] map)

KAMUS
    treasureCount, count : integer
    currKoor, temp2 : class koor
    avail, visited : Queue of class koor
    treasure : list of class koor

ALGORITMA
    avail.Push(start)
    treasureCount <- treasure.Count
    count <- 0

    while(avail.Count>0)do
        currKoor <- new koor(avail.Dequeue)
        visited.Enqueue(currKoor)
        if(map[currKoor.X,currKoor.Y] = 'T') then
            count++
            treasure.Remove(currKoor)
            if(count = treasureCount)then
                break;
            endif
        endif
        foreach(var i in "DLUR")
            temp2 <- new koor(currKoor)
            if(i='R')then
                if(currKoor.Y < col-1)then
                    if(map[currKoor.X,currKoor.Y+1] != 'X' ) then
                        temp2 <- new koor(currKoor.X,currKoor.Y+1)
                    endif
                endif
            else if(i='L')then
                if(currKoor.Y>0)then
                    if(map[currKoor.X,currKoor.Y-1] != 'X')then
                       temp2 <- new koor(currKoor.X,currKoor.Y-1)
                    endif
                endif
            else if(i='U')then
                if(currKoor.X > 0)then
```

```
                if(map[currKoor.X-1,currKoor.Y] != 'X')then
                    temp2 <- new koor(currKoor.X-1,currKoor.Y)
                endif
            endif
        else if(i='D') then
             if(currKoor.X < row-1) then
                if(map[currKoor.X+1,currKoor.Y] != 'X')then
                    temp2 <- new koor(currKoor.X+1,currKoor.Y)
                endif
            endif
        endif

        if(visited.Contains(temp2))endif
            continue
        endif
        avail.Enqueue(temp2)

    endwhile
    return visited
```

4.2 Struktur Data Program

Berikut merupakan struktur data yang digunakan dalam membuat program:

1. Stack

   Implementasi struktur data Stack menggunakan collection yang sudah tersedia pada kelas System.Collectons.Generic. Stack digunakan untuk menyimpan tumpukan Node yang akan dikunjungi oleh program saat melakukan traversal dengan metode DFS. Berikut merupakan implementasi dari method yang digunakan:

   a. Push, berfungsi untuk memasukkan elemen ke dalam tumpukan dengan aturan *last in first out*.

   b. Pop, berfungsi untuk mengeluarkan elemen dari tumpukan dengan aturan *last in first out*.

2. Queue

   Implementasi struktur data Queue menggunakan collection yang sudah tersedia pada kelas System.Collectons.Generic. Queue digunakan untuk menyimpan antrian Node yang akan dikunjungi oleh program saat melakukan traversal dengan metode BFS. Berikut merupakan implementasi dari method yang digunakan:

   a. Enqueue, berfungsi untuk memasukkan elemen ke dalam antrian dengan aturan *first in first out*.

   b. Dequeue, berfungsi untuk mengeluarkan elemen dari antrian dengan aturan *first in first out*.

4.3 Tata Cara Penggunaan Program

4.3.1 Cara untuk menjalankan program

4.3.1.1 Melalui Visual Studio

1. Jalankan Visual Studio yang telah terpasang sebelumnya.
2. Buka file Tubes Stima Maze.sln
3. Klik tombol "Start" pada panel atas.
4. Visual Studio akan secara otomatis menjalankan proses build dan menjalankan aplikasi jika build berhasil.

4.3.1.2 Melalui Executable Code

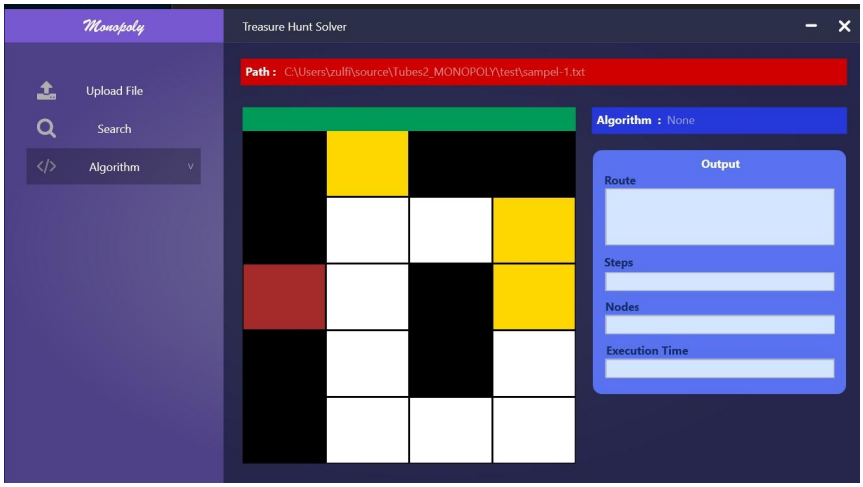
1. Pilih dan klik pada file DBFS.exe yang ada pada folder bin pada repository ini.

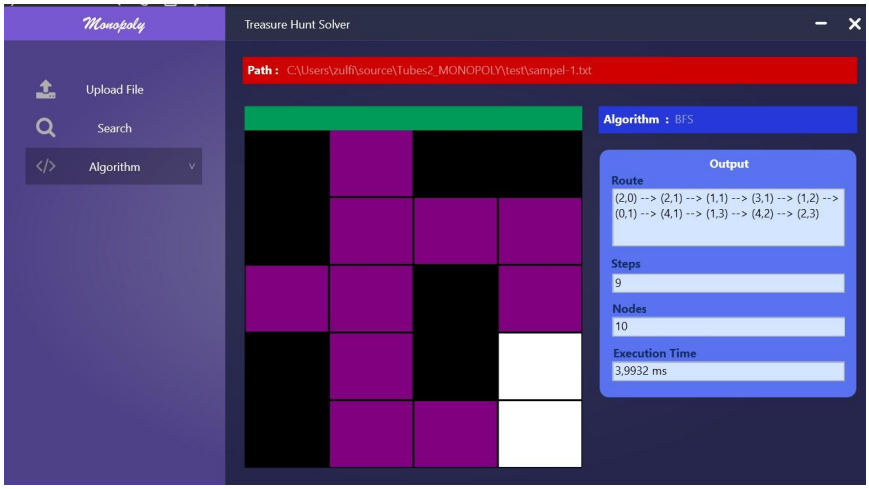
4.3.2 Cara untuk menggunakan program

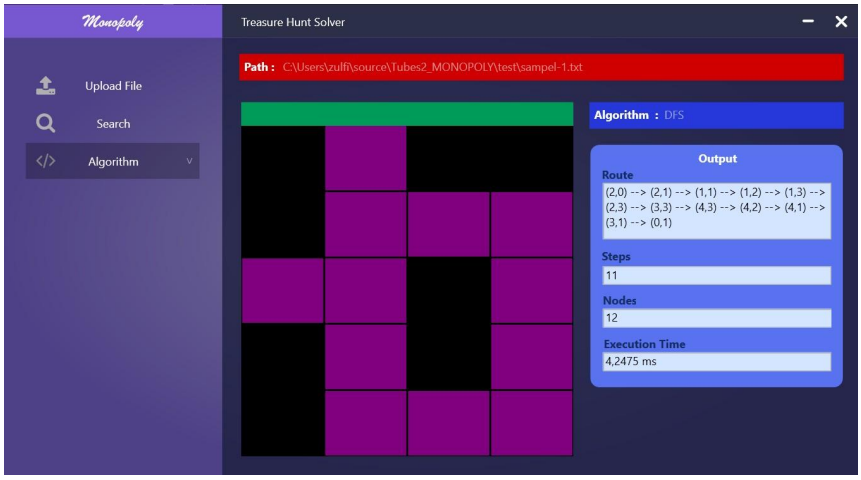
1. Jalankan program seperti pada bagian 4.3.1.
2. Jika aplikasi berhasil dijalankan akan ditampilkan menu utama dari program.
3. Klik tombol "Upload file" untuk memilih maze yang akan digunakan.
4. Pilih method algorithm yang ingin digunakan dalam proses pencarian file tersebut, yakni menggunakan algoritma Breadth First Search (BFS) atau Depth First Search (DFS).
5. Klik tombol “Search” untuk memulai pencarian
6. Layar akan menampilkan hasil pencarian.

4.4 Hasil Pengujian

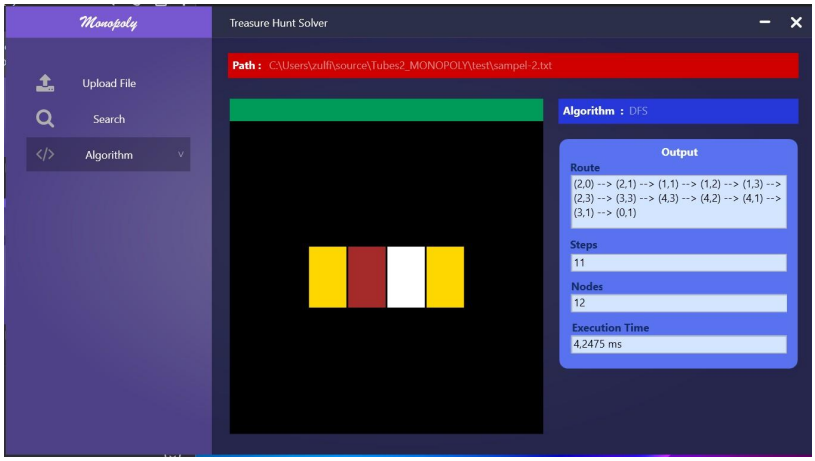
4.4.1 Pengujian 1

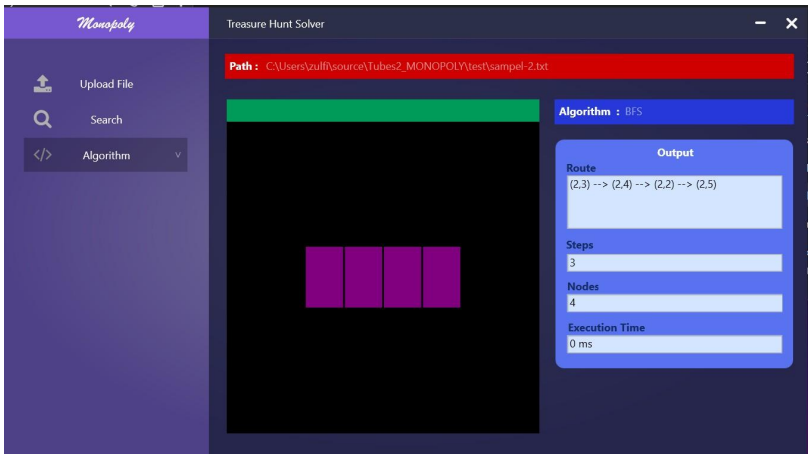


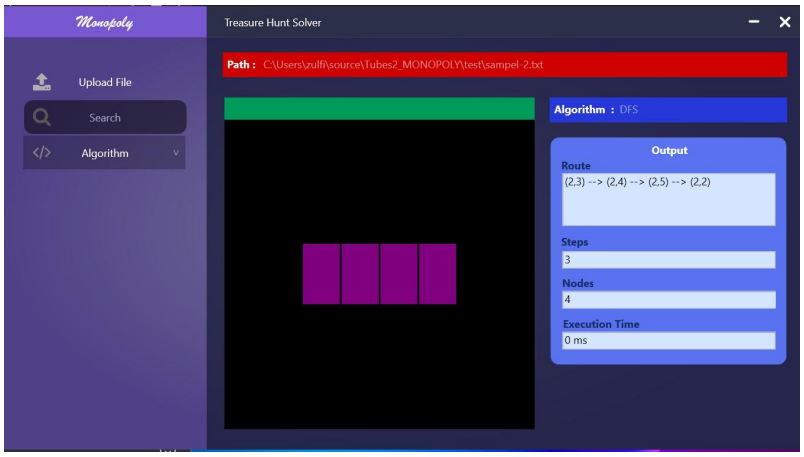




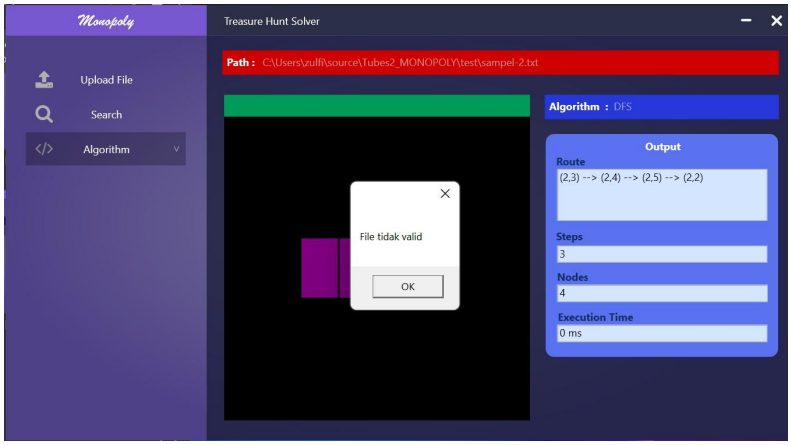
4.4.1 Pengujian 2



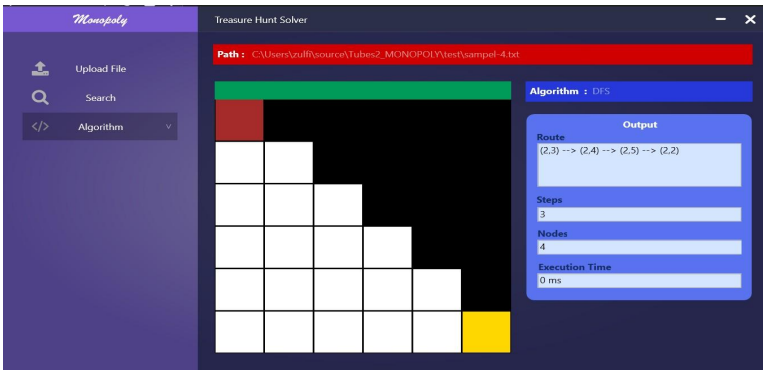


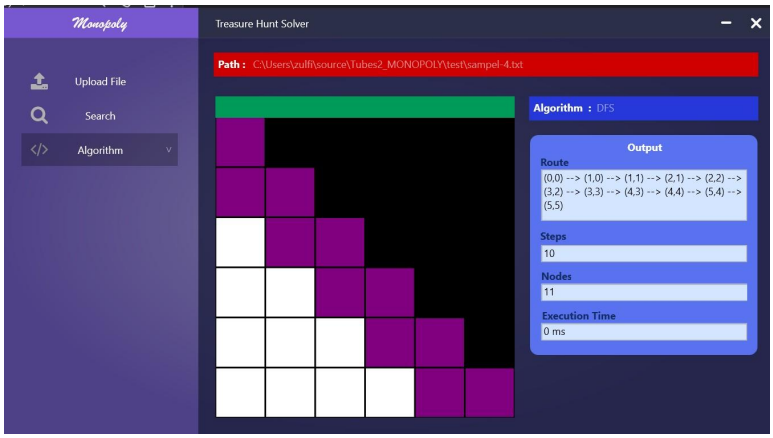


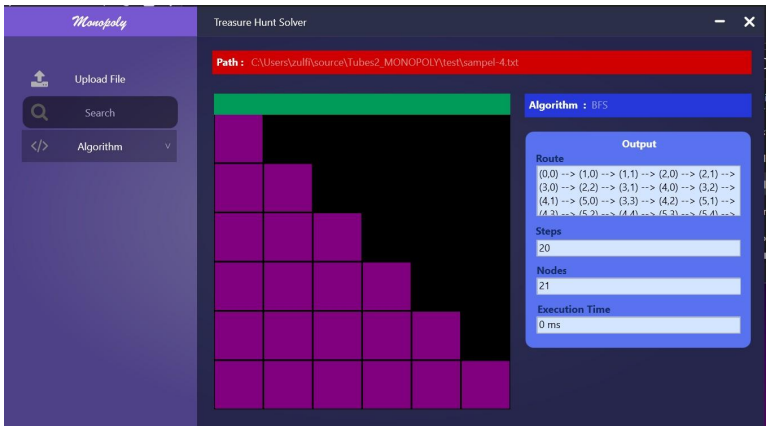
4.4.3 Pengujian 3



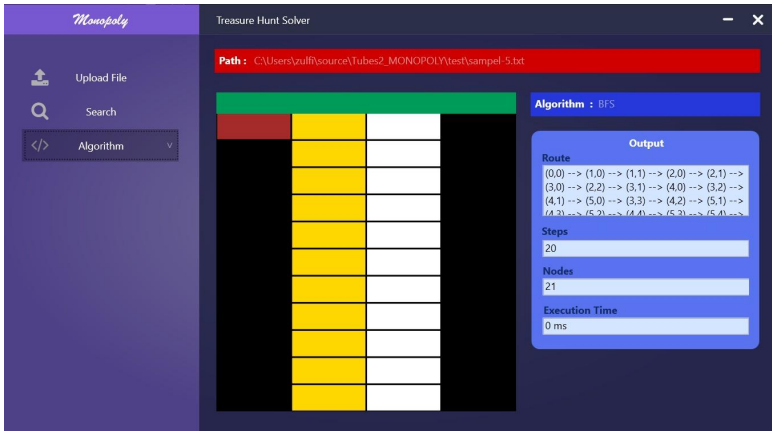
4.4.4 Pengujian 4

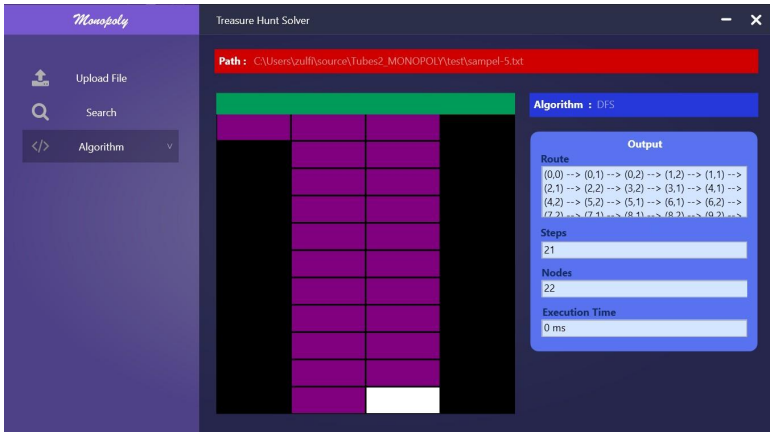


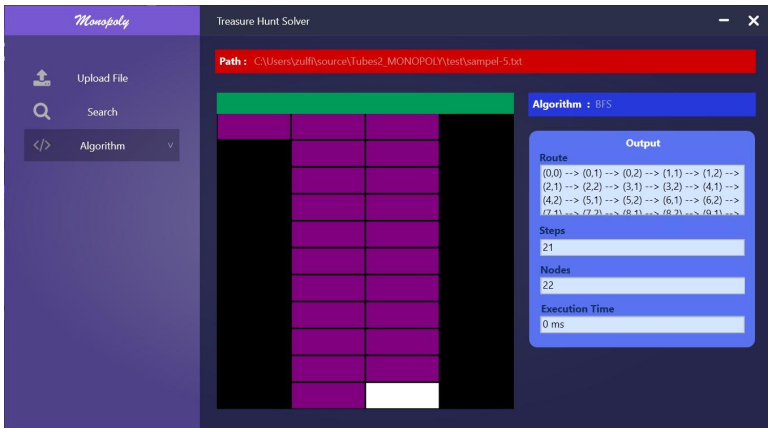




4.4.5 Pengujian 5







4.5 Analisis Desain Solusi

Berdasarkan Pengujian yang kami lakukan dari beberapa contoh kasus yang berbeda, Algoritma DFS memiliki nilai optimasi yang kurang lebih sama dibandingkan Algoritma. Algoritma DFS dan BFS yang kami implementasikan menggunakan prioritas arah simpul RULD (Right Up Left Down). Jika dilakukan analisis lebih mendalam, pada kasus pencarian harta karun di dalam labirin, jika labirin memiliki kedalaman yang cukup dalam dan simpul tujuan berada cukup jauh dari simpul awal, maka BFS dapat menjadi pilihan yang lebih baik karena tidak perlu menyimpan banyak simpul dalam memori. Namun, jika simpul tujuan berada dalam jarak yang cukup dekat dari simpul awal, maka DFS dapat memberikan solusi yang lebih cepat karena hanya mengunjungi simpul pada cabang tertentu. Kedua hal tersebut bisa terjadi dikarenakan perbedaan urutan mengunjungi simpul dari masing masing algoritma sehingga dapat disimpulkan bahwa dalam praktiknya, algoritma solusi yang diberikan DFS dan BFS memiliki kelebihan dan kekurangan masing masing tergantung persoalan yang diselesaikan.

BAB V
PENUTUPAN

5.1 Kesimpulan

Tugas besar program pencarian rute dengan BFS dan DFS berbasis GUI yang mengaplikasikan materi yang berhubungan dengan matriks dalam mata kuliah IF2211 Strategi Algoritma telah berhasil dibuat. Materi-materi yang diimplementasikan pada program ini berupa:

1. Graph Traversal
2. Breadth-First Search
3. Depth-First Search

Melalui penerapan materi-materi tersebut, program ini berhasil diselesaikan sesuai dengan ketentuan yang tertera pada spesifikasi.

5.2 Saran

Kami menyarankan agar pengembangan aplikasi ini fokus pada peningkatan tampilan dan kreativitas desain antarmuka pengguna (GUI). Hal ini diharapkan dapat meningkatkan pengalaman pengguna dalam menggunakan aplikasi.

5.3 Refleksi

Tugas Besar Strategi Algoritma kali ini menjadi pembelajaran yang sangat berharga bagi penulis, banyak hal baru yang dipelajari dalam pembuatan tugas besar ini. Meskipun terdapat beberapa kendala dalam pembuatan algoritma BFS dan DFS serta dalam pembuatan tampilan antar muka GUI namun hal itu malah menjadi pacuan agar penulis lebih semangat dalam mengeksplorasi untuk menemukan solusi yang terbaik.

5.4 Tanggapan Anggota

Menurut kami tugas ini lumayan menantang, karena kami harus mempelajari terlebih dahulu bahasa C# yang masih sangat baru bagi kami. Pembuatan project ini juga dilakukan pada IDE Visual Studio 2022 yang mana itu juga merupakan hal yang baru bagi kami sehingga membuat tugas ini lebih menantang lagi. Pembuatan algoritma BFS dan DFS menurut saya sedikit lebih mudah dibandingan dengan pembuatan GUI karena ada banyak referensi yang tersedia di buku maupun internet serta juga slide perkuliahan. Terakhir kami mengucapkan terima kasih kepada asisten dan dosen pengampu yang sudah memberikan tugas ini, karenanya kami jadi belajar banyak hal baru.

REFERENSI

Munir, Rinaldi. (2021). Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1) (Versi baru 2021). Institut Teknologi Bandung.https://informatika.stei.itb.ac.id/ ~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf . Diakses pada 22 Maret 2023.

Munir, Rinaldi. (2021). Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 2) (Versi baru 2021). Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/ ~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf . Diakses pada 22 Maret 2023.

Microsoft. “C# Documentation”, https://docs.microsoft.com/en-us/dotnet/csharp/, diakses pada 19 Maret 2023.

LAMPIRAN

Link repository Github : https://github.com/zulfiansyah404/Tubes2_MONOPOLY
Link Youtube :