Laporan Tugas Besar 2 IF2211 Strategi Algoritma

Semester II tahun 2022/2023

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan *Maze Treasure Hunt*

Kelompok 31 - dicarryVieridanZaki

Anggota Kelompok:

Vieri Fajar Firdaus	13521099
Muhammad Zaki Amanullah	13521146
Mohammad Rifqi Farhansyah	13521166

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

Daftar Gambar

BAB I

1.1. Latar Belakang

Tuan Krabs menemukan sebuah labirin distorsi terletak tepat di bawah Krusty Krab bernama El Doremi yang Ia yakini mempunyai sejumlah harta karun di dalamnya dan tentu saja Ia ingin mengambil harta karunnya. Dikarenakan labirinnya dapat mengalami distorsi, Tuan Krabs harus terus mengukur ukuran dari labirin tersebut. Oleh karena itu, Tuan Krabs banyak menghabiskan tenaga untuk melakukan hal tersebut sehingga Ia perlu memikirkan bagaimana caranya agar Ia dapat menelusuri labirin ini lalu memperoleh seluruh harta karun dengan mudah.

***Gambar 1.1.1**. Ilustrasi Krusty Krab*
Sumber: https://static.wikia.nocookie.net/theloudhouse/images/e/ec/Massive_Mustard_Pocket.png/revision/latest?cb=20180826170029

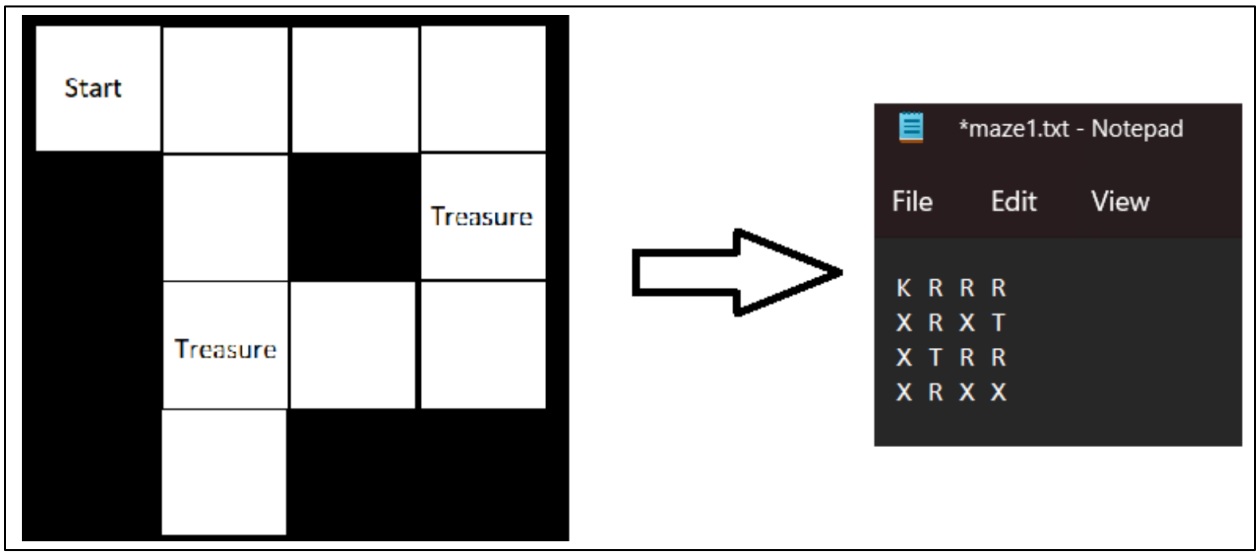
Setelah berpikir cukup lama, Tuan Krabs tiba-tiba mengingat bahwa ketika Ia berada pada kelas Strategi Algoritma-nya dulu, Ia ingat bahwa Ia dulu mempelajari algoritma BFS dan DFS sehingga Tuan Krabs menjadi yakin bahwa persoalan ini dapat diselesaikan menggunakan kedua algoritma tersebut. Akan tetapi, dikarenakan sudah lama tidak menyentuh algoritma, Tuan Krabs telah lupa bagaimana cara untuk menyelesaikan persoalan ini dan Tuan Krabs pun kebingungan. Tidak butuh waktu lama, Ia terpikirkan sebuah solusi yang brilian. Solusi tersebut adalah meminta mahasiswa yang saat ini sedang berada pada kelas Strategi Algoritma untuk menyelesaikan permasalahan ini.

1.2. Deskripsi Tugas

Dalam tugas besar ini, Anda 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 membacapa input sebuha fil etxt yang berisi maze yang akan ditemukan solusi rute mendapatkan *treasure*-nya. Untuk mempemudah, 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

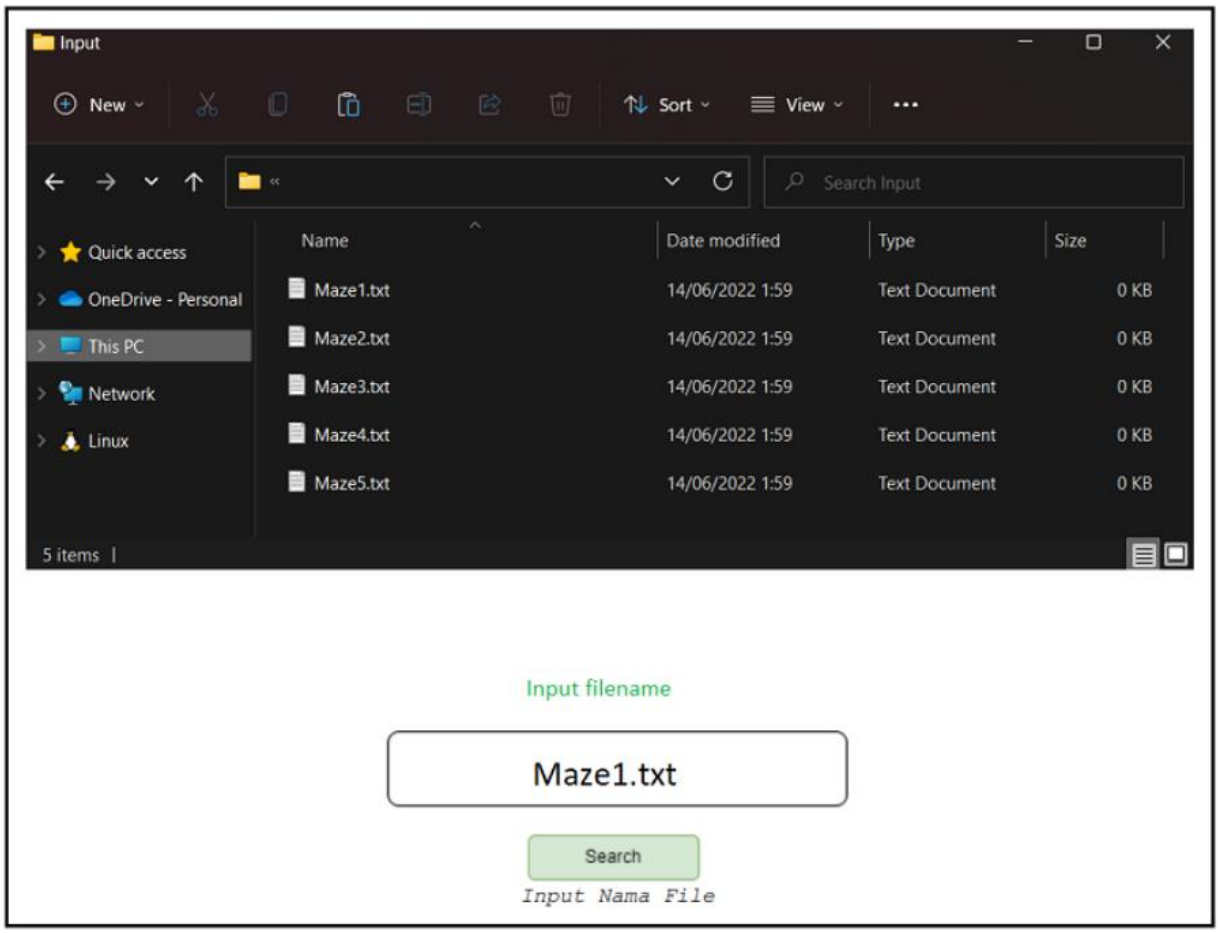
Contoh file input :



***Gambar 1.2.1**. Ilustrasi Input File Maze*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tubes2-Stima-2023.pdf

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 asalakan 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.

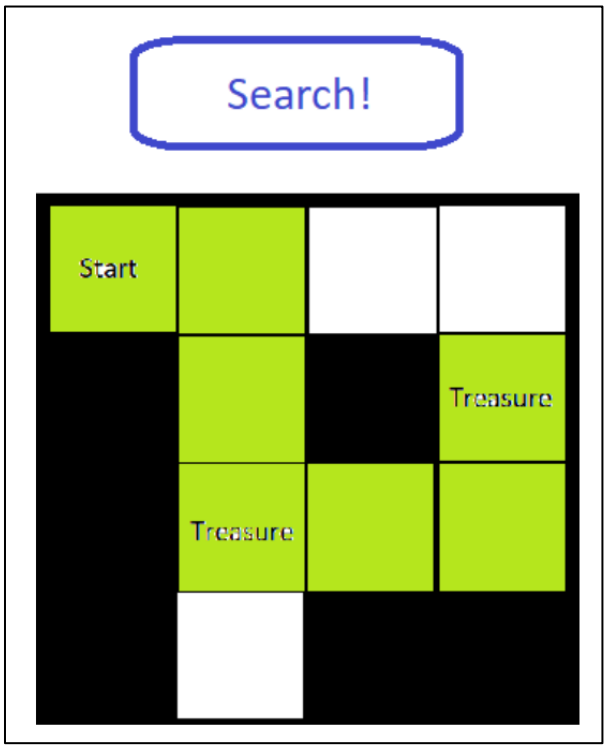
Contoh input aplikasi :



***Gambar 1.2.2**. Contoh Input Program*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tubes2-Stima-2023.pdf

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, hasur menghandle kasus apabila tidak ditemukan dengan anam file tersebut.

Contoh input aplikasi :

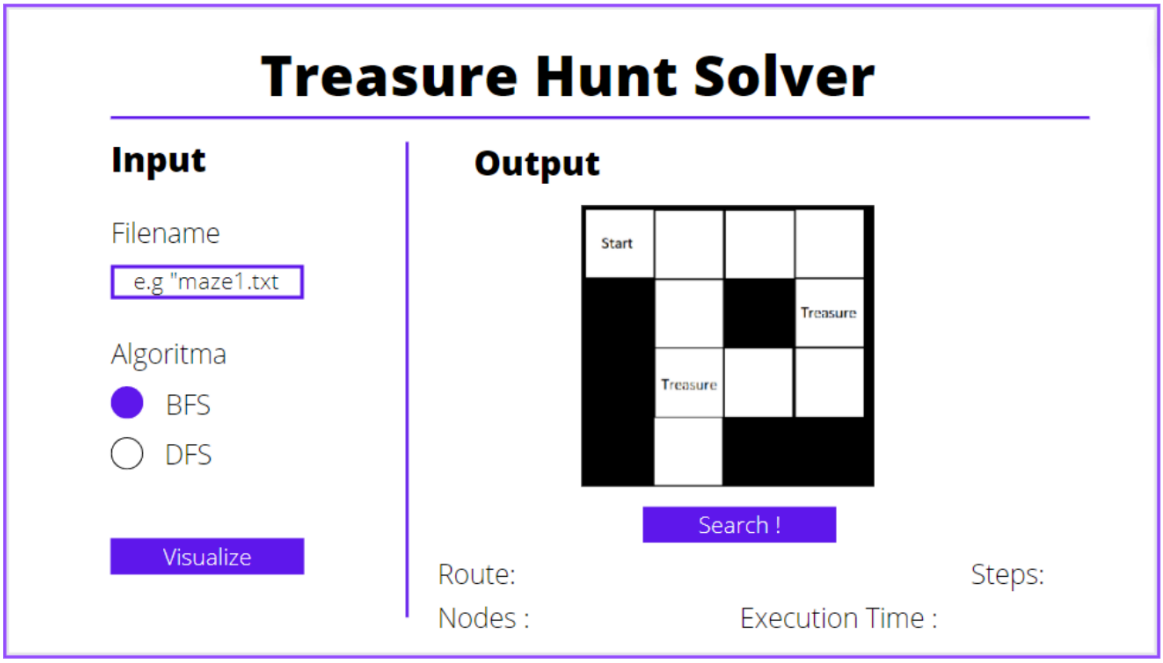


***Gambar 1.2.3**. Contoh Output Program*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tubes2-Stima-2023.pdf

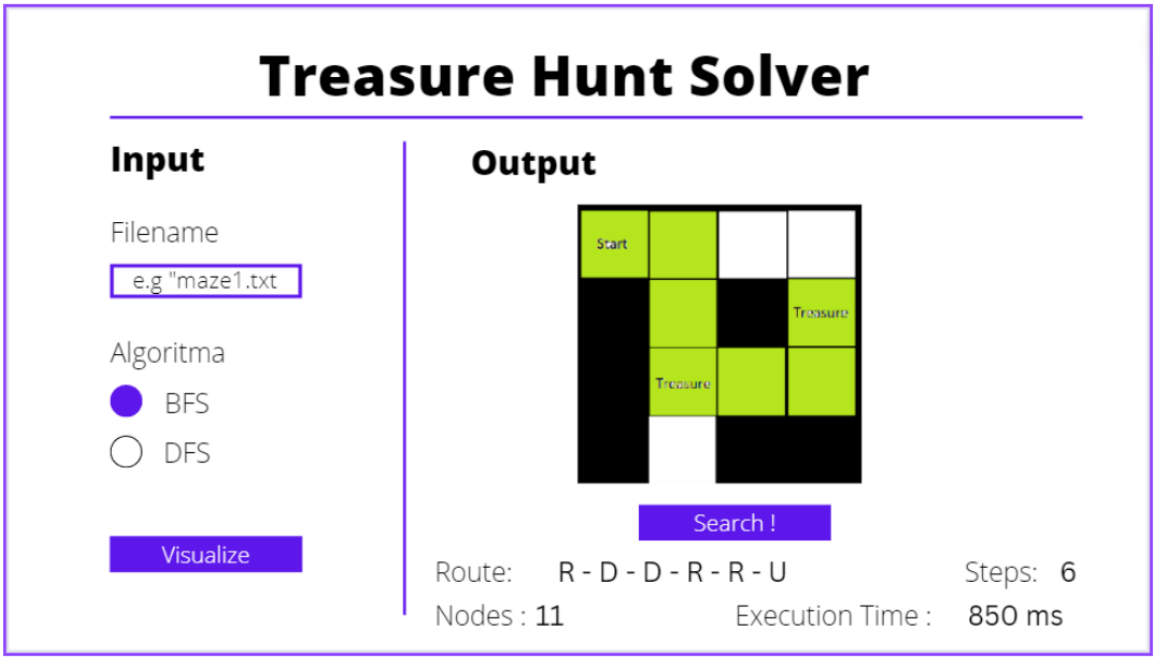
Setelah program melakukan pembacaan input, program akan memvisualisasikan gridnya terlebih dahulu tanpa pemberian rute solusi. Hal tersebut dilakukan agar pengguna dapat mengerjakan terlebih dahulu treasure hunt secara manual jika diinginkan. Kemudian, program menyediakan tombol solve untuk mengeksekusi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

1.3. Spesifikasi Program

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



***Gambar 1.3.1**. Tampilan Program Sebelum dicari Solusinya*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tubes2-Stima-2023.pdf



***Gambar 1.3.2**. Tampilan Program Setelah dicari Solusinya*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tubes2-Stima-2023.pdf

1.3.1. Spesifikasi GUI

1. **Masukan Program** adalah file maze *treasure hunt* tersebut atau nama filenya.
2. Program dapat menampilkan visualisasi dari input file maze dalam bentuk grid dan pewarnaan sesuai deskripsi tugas.
3. Program memiliki toggle untuk menggunakan alternatif algoritma BFS ataupun DFS.
4. Program memiliki tombol search yang dapat mengeksekusi pencarian rute dengan algoritma yang bersesuaian, kemudian memberikan warna kepada rute solusi output.
5. **Luaran program** adalah banyakanya node (grid) yang diperiksa, banyaknya langkah, rute solusinya, dan waktu eksekusi algoritma.
6. (**Bonus**) Program dapat menampilkan progress pencarian grid dengan algoritma yang bersesuaian. Hal tersebut dilakukan dengan memberikan slider / input box untuk menerima durasi jeda tiap step, kemudian memberikan warna kuning untuk setiap grid yang sudah diperiksa dan biru untuk grid yang sedang diperiksa.
7. (**Bonus**) Program membuat toggle tambahan untuk persoalan TSP jadi apabila toggle dinyalakan, rute solusi yang diperoleh juga harus kembali ke titik awal setelah menemukan segala harta karunnya (Tetap dengan algoritma BFS atau DFS).
8. GUI dapat dibuat **sekreatif** mungkin asalkan memuat 5 (7 jika mengerjakan bonus) spesifikasi di atas.

1.3.2. Spesifikasi Wajib Program

1. Buatlah program dalam bahasa **C#** untuk mengimplementasi Treasure Hunt Solver sehingga diperoleh output yang diinginkan. Penelusuran harus memanfaatkan algoritma **BFS** dan **DFS**.
2. Awalnya program menerima file atau nama file maze treasure hunt.
3. Apabila filename tersebut ada, Program akan melakukan validasi dari file input tersebut. Validasi dilakukan dengan memeriksa apakah tiap komponen input hanya berupa K, T, R, X. Apabila validasi gagal, program akan memunculkan pesan bahwa file tidak valid. Apabila validasi berhasil, program akan menampilkan **visualisasi** awal dari maze treasure hunt.
4. Pengguna memilih algoritma yang digunakan menggunakan toggle yang tersedia.
5. Program kemudian dapat menampilkan visualisasi akhir dari maze (dengan pewarnaan rute solusi).
6. Program menampilkan luaran berupa durasi eksekusi, rute solusi, banyaknya langkah, serta banyaknya node yang diperiksa.

   Proses visualisasi ini boleh memanfaatkan pustaka atau kakas yang tersedia. Sebagai referensi, salah satu kakas yang tersedia untuk memvisualisasikan matrix dalam bentuk grid adalah **DataGridView**. Berikut adalah panduan singkat terkait penggunaannya: http://csharp.net-informations.com/datagridview/csharp-datagridview-tutorial.htm .

7. Mahasiswa **tidak diperkenankan** untuk melihat atau menyalin library lain yang mungkin tersedia bebas terkait dengan pemanfaatan BFS dan DFS. Akan tetapi, untuk algoritma lain diperbolehkan menggunakan library jika ada.

   Spesifikasi detail dari Tugas Besar 2 - Strategi Algoritma 2023/2024 dapat diakses melalui pranala berikut ini.

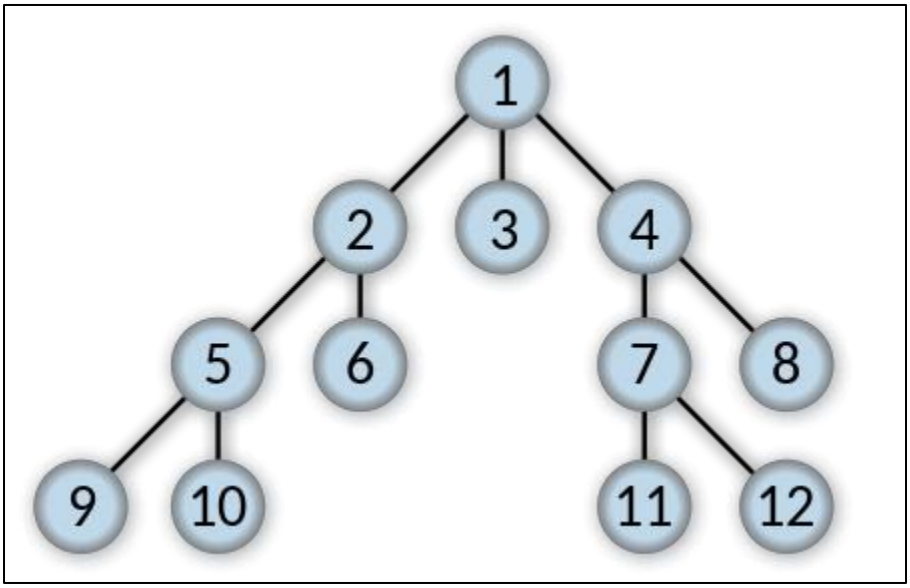
BAB II
Landasan Teori

2.1. Graph Traversal

Graph Traversal adalah teknik untuk mengunjungi setiap node dari graf. Teknik ini juga digunakan untuk menghitung urutan simpul dalam proses traverse. Graph Traversal mengunjungi semua node mulai dari satu node yang terhubung satu sama lain tanpa masuk ke loop. Pada dasarnya dalam graf mungkin terjadi beberapa waktu pengunjung dapat mengunjungi satu node lebih dari sekali sehingga dapat menyebabkan terjadinya *infinite loop*. Jadi, untuk menghindar dari kondisi *infinite loop* ini, graf traversal mencatat setiap simpul. Seperti jika pengunjung mengunjungi vertex maka nilainya akan menjadi nol, jika tidak maka satu.

2.2. Breadth-First Search (BFS)

Breadth-First Search atau BFS merupakan salah satu algoritma yang paling sederhana dalam melakukan pencarian pada graf dan pola dasar pada algoritma pencarian graf lainnya. Algoritma minimum-spanning-tree Prim dan Algoritma pencarian jarak terdekat Djikstra menggunakan ide yang serupa dengan pencarian breadth-first search. Dalam algoritma breadth-first search, graf direpresentasikan dalam bentuk $G = (V, E)$ dengan v merupakan simpul awal penelusuran.



***Gambar 2.2.1**. Ilustrasi Graph Menggunakan BFS*
Sumber: https://upload.wikimedia.org/wikipedia/commons/thumb/3/33/Breadth-first-tree.svg/450px-Breadth-first-tree.svg.png

Secara sederhana, algoritma breadth-first search akan memulai penelusuran dari simpul v. Kemudain, akan dilakukan penelusuran terhadap semua simpul yang bertetangga dengan simpul v terlebih dahulu. Terakhir, akan dilakukan penelusuran terhadap simpul-simpul lain yang belum dikunjungi yang bertetangga dengan simpul-simpul yang telah dikunjungi, Langkah ini akan dilakukan terus-menerus hingga ditemukan simpul yang dicari atau hingga seluruh simpul telah ditelusuri.

Pada prosedur breadth-first search, dapat digunakan adjacency lists dalam merepresentasikan graf $G = (V, E)$. Selain itu, untuk mengetahui simpul yang akan diperiksa atau tidak, akan digunakan struktur data Queue. Terakhir, untuk mengetahui suatu simpul telah diperiksa atau belum, akan digunakan struktur data

array atau hash table yang bertipe boolean. Berikut adalah pseudocode umum untuk prosedur breadth-first search.

```
procedure BFS(input G: graph, input v: vertice)
{ Traversal Graf dengan algoritma penelususran BFS
  I.S. G dan v terdefinisi dan tidak sembarang
  F.S. Semua simpul yang dilalui tercetak pada layar }
KAMUS
  { Variabel }
  q : queue
  w : vertice
  visited : hashTable
  { Fungsi dan Prosedur antara }
  procedure createQueue(input/output q: queue)
  { Inisialisasi queue
    I.S. q belum terdefinisi
    F.S. q terdefinisi dan kosong }
  procedure enqueue(input/output q: queue, input v: vertice)
  { Memasukkan v ke dalam q dengan aturan FIFO
    I.S. q terdefinisi
    F.S. v masuk ke dalam q dengan aturan FIFO }
  function dequeue(q: queue) → vertice
  { Menghapus simpul dari q dengan aturan FIFO
    dan mengembalikan simpul yang dihapus}
  function isQueueEmpty(q: queue) → boolean
  { Mengembalikan True jika q kosong dan sebaliknya }
  procedure createHashTable(input/output T: hashTable)
  { Inisialisasi hashTable
    I.S. T belum terdefinisi
    F.S. T terdefinisi dan terisi False sebanyak simpul }
  procedure setHashTable(input/output T: hashTable, input v: vertice, input val: boolean )
  { Mengubah value dari T
    I.S. T sudah terdefinisi
    F.S. Elemen T dengan key v sudah diubah menjadi val }
  function getHashTable(T: hashTable, input v: vertice) → boolean
  { Mengembalikan elemen T pada key v }

ALGORITMA
  { Inisialisasi visited }
  createHashTable(visited)
  { Inisialisasi q }
  createQueue(q)

  { Mengunjungi simpul pertama (akar) }
  output(v)
  enqueue(q, v)
  setHashTable(T, v, True)

  { Mengunjungi semua simpul }
  while (not isQueueEmpty(q)) do
      v ← dequeue(q)
      for setiap simpul w yang bertetangga dengan simpul v do
          if (not getHashTable(visited, w)) then
              output(w)
              enqueue(q, w)
              setHashTable(T, w, True)

  { q kosong }
```

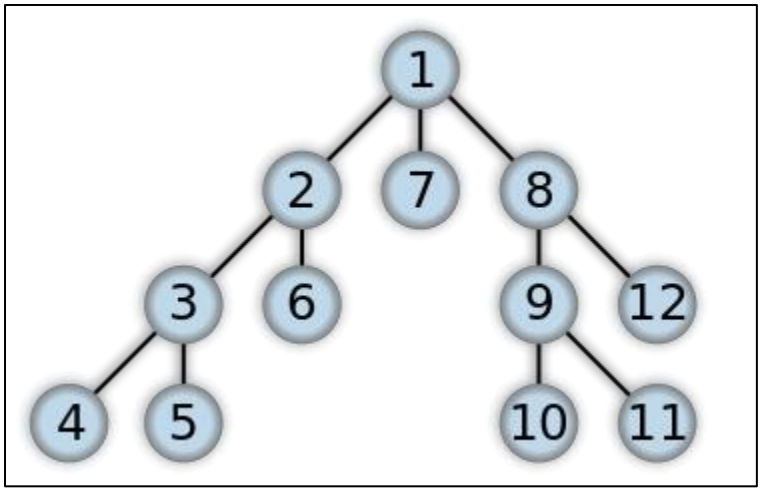
Pseudocode umum untuk prosedur breadth-first search di atas akan menghasilkan seluruh simpul yang ada. Untuk pencarian satu simpul, algoritma dapat ditambahkan menjadi seperti berikut:

```
procedure BFS(input G: graph, input v: vertice, input x: vertice)
{ Traversal Graf dengan algoritma penelususran BFS
   I.S. G dan v terdefinisi dan tidak sembarang
   F.S. Semua simpul yang dilalui tercetak pada layar }
KAMUS
  { Variabel }
  { IDEM }
  found : boolean
  { Fungsi dan Prosedur antara }
  { IDEM }

ALGORITMA
  { Inisialisasi found }
  found ← False
  { Inisialisasi visited }
  createHashTable(visited)
  { Inisialisasi q }
  createQueue(q)

  { Mengunjungi simpul pertama (akar) }
  output(v)
  enqueue(q, v)
  setHashTable(T, v, True)

  { Memeriksa simpul pertama (akar) }
  if (v = x) then
      found ← True

  { Mengunjungi semua simpul }
  while (not isQueueEmpty(q) and not found) do
      v ← dequeue(q)
      for setiap simpul w yang bertetangga dengan simpul v do
          if (not getHashTable(visited, w) and not found) then
          output(w)
          enqueue(q, w)
          setHashTable(T, w, True)
          if (w = x) then
              found ← True

  if (not found) then
      output("Tidak ditemukan")
```

2.3. Depth-First Search (DFS)

Berbeda dengan breadth-first search, algoritma pencarian depth-first search melakukan penelusuran terlebih dahulu ke-"dalam" ketika memungkinkan. Algoritma depth-first search akan menelusuri simpul tetangga dari simpul yang ditelusuri sekarang. Setelah sudah tidak ada simpul tetangga yang tersedia, algoritma melakukan backtracks untuk menelusuri simpul-simpul tetangga dari simpul sebelumnya.

Secara sederhana, algoritma depth-first search akan memulai penelusuran dari simpul v. Kemudian, akan dilanjutkan penelusuran ke simpul w yang bertetangga dengan simpul v. Setelah itu, akan dilakukan pemanggilan kembali algoritma DFS yang dimulai dari simpul w. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga telah dikunjungi, akan dilakukan pencarian runut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya, yaitu simpul yang melakukan pemanggilan algoritma DFS. Pencarian selesai jika tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.



***Gambar 2.3.1**. Ilustrasi Graph Menggunakan DFS*
Sumber: https://upload.wikimedia.org/wikipedia/commons/thumb/1/1f/Depth-first-tree.svg/375px-Depth-first-tree.svg.png

Pada prosedur depth-first search, representasi graf dan struktur data hampir serupa seperti yang digunakan pada prosedur breadth-first search. Akan tetapi, pada depth-first search, tidak akan digunkan struktur data queue, melainkan lebih mirip dengan struktur data stack. Berikut adlaah pseudocode dari prosedur depth-first search.

```
procedure DFS(input G: graph, input/output visited: hashTable, input v: vertice)
{ Traversal Graf dengan algoritma penelususran DFS
  I.S. G, v, dan visited terdefinisi dan tidak sembarang
  F.S. Semua simpul yang dilalui tercetak pada layar }
KAMUS
  { Variabel }
  w : vertice
  { Fungsi dan Prosedur antara }
  procedure setHashTable(input/output T: hashTable, input v: vertice, input val: boolean )
  { Mengubah value dari T
    I.S. T sudah terdefinisi
    F.S. Elemen T dengan key v sudah diubah menjadi val }
  function getHashTable(T: hashTable, input v: vertice) → boolean
  { Mengembalikan elemen T pada key v }

ALGORITMA
  { Mengunjungi simpul sekarang }
  output(v)
  setHashTable(T, v, True)
  { Mengunjungi semua simpul }
  for setiap simpul w yang bertetangga dengan simpul v do
      if (not getHashTable(visited, w)) then
          DFS(G, visited, w)
```

2.4. Pengembangan Aplikasi Desktop

2.4.1. Bahasa Pemrograman C#

C# atau dibaca C Sharp merupakan sebuah bahasa pemrograman berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic, dan lain-lain dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# Language Specification, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023).

Beberapa tujuan dari pembentukan bahasa C# ini diantaranya adalah,

- Dibentuk sebagai bahasa pemrograman yang bersifat genral-purporse, berorientasi objek, modern, dan sederhana.
- Mengembangkan komponen perangkat lunak yang mampu mengambil keuntungan dari lingkungan terdistribusi.
- Supaya cocok digunakan untuk menulis program aplikasi baik dalam sistem klien-server (hosted system) maupun sistem terbenam (embedded system), mulai dari perangkat lunak yang sangat besar yang menggunakan sistem operasi canggih hingga pada perangkat lunak sangat kecil yang memiliki fungsi-fungsi terdedikasi.

2.4.2. C# Desktop Application Development

C# Desktop Application Development adalah sebuah kakas pegembang desktop application yang mampu beroperasi tanpa menggunakan internet. Pada umumnya, pengembangan desktop application 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. WPF digunakan untuk pengembangan desktop application berbasis Windows dengan menggunakan bahasa native-nya, yaitu bahasa pemrogramman C#.

Berikut merupakan langkah-langkah untuk mengembangkan aplikasi desktop menggunakan Windows Presentation Foundation (WPF):

1. Membuat proyek baru:
   a. Buka aplikasi Visual Studio.
   b. Pada jendela awal, pilih Create a new project.
   c. Pada jendela Create a new project, pilih WPF App (.NET Core).
   d. Pada jendela Configure your new project, isi nama proyek pada isian Project name. Kemudian, tekan tombol Create.
2. Membuat aplikasi:
   a. Pilih menu Toolbox untuk membuka jendela Toolbox.
   b. Pada jendela Toolbox, kita dapat memilih komponen-komponen yang ingin kita gunakan pada aplikasi desktop kita. Untuk memodifikasi komponen, dapat mengubah properti pada jendela Properties.
3. Menambahkan kode pada form:
   a. Pada jendela MainWindow.xaml [Design], tekan dua kali untuk membuka jendela MainWindow.xaml.cs.
   b. Kode dapat ditulis pada file MainWindows.xaml.cs.
4. Menjalankan aplikasi:.
   a. Tekan tombol Start pada menu.

BAB III
Analisis Pemecahan Masalah

3.1 Langkah-langkah Pemecahan Masalah

Secara umum, langkah awal dalam proses mendesaian solusi dari permasalaan ini adalah dengan membagi permasalahan utama, yaitu treasure hunt, menjadi beberapa permasalahan kecil sehingga dapat mempermudah pencarian solusi. Permasalahan tersebut dapat dibagi menjadi dua permalasahan utama, yaitu pencarian rute menggunakan algoritma breadth-first search dan depth-first search. Selain itu, ada permasalahan-permasalahan tambahan yang perlu untuk diselesaikan yaitu solusi dari dua permasalahan tersebut harus ditampilkan dalam bentuk desktop application serta pengembangan langkah pemecahan masalahan yang berkaitan dengan Travelling Salesman Problem (TSP).

Kemudian, langkah selanjutnya adalah dengan melakukan pemetaan permasalahan ke dalam elemen-elemen algoritma breadth-first search dan depth-first search. Setelah melakukan pemetaan elemen-elemen algoritma, dilakukan perancangan algoritma sesuai dengan elemen yang telah dipetakan.

Pada penyelesaian permasalahan tambahan, yaitu menampilkan solusi dari pencarian rute menggunakan algoritma BFS dan DFS, kami menggunakan bantuan WPF. Selain itu, kami juga menggunakan bantuan pustaka DataGridView yang dimiliki oleh C# untuk menampilkan visualisasi dari matrix yang telah dibentuk.

Secara singkat, dalam menyelesaikan permasalahan ini, telah dikomposisi beberapa permasalahan sebagai berikut:

1. Memahami dan memplejari struktur folder pada windows untuk memahami cara mengakses direktori dan file yang diperlukan dalam permasalahan ini.
2. Memahami konsep dari algoritma traversal BFS dan DFS untuk menentukan cara terbaik dalam mencari solusi yang diinginkan.
3. Mempelajari syntax bahasa pemrograman C# dan Windows Presentation Foundation (WPF) khususnya dalam mengakses folder dan file, membuat graph dalam C#, dan menyambungkan C# DataGridView untuk menampilkan visualisasi dari peta treasure hunt yang hendak dipecahkan.
4. Membentuk UI design yang diinginkan.
5. Membuat Graphical User Interface (GUI) menggunakan Windows Presentation Foundation (WPF) pada Visual Studio sesuai design yang telah dipersiapkan sebelumnya.
6. Mengimplementasikan algoritma BFS dan DFS pada setiap elemen pencarian rute dalam permasalahan Maze Treasure Hunt.
7. Membuat visualisasi rute dan peta menggunakan C# DataGridView.
8. Mengintegrasikan fungsi-fungsi yang telah dibuat dalam pemecahan masalah dengan GUI.
9. Melakukan pengujian terhadap kasus-kasus yang mungkin ditemui.

3.2 Elemen-elemen Algoritma BFS dan DFS

3.1.1 Breadth-First Search

Algoritma BFS pada permasalahan ini menggunakan konsep pencarian solusi dari graf dinamis. Graf direpresentasikan menggunakan daftar ketetanggan atau adjacency list. Elemen antrian atau queue berisi simpul atau node yang akan dikunjungi, dimana setiap simpul berisi nama folder atau file yang ada. Karena struktur direktori pasti memiliki struktur pohon, tidak diperlukan suatu variabel untuk menyimpan simpul yang telah dikunjungi.

Pada awal pencarian, simpul awal atau root dimasukkan ke dalam antrian atau queue. Selanjutnya, simpul-simpul tetangga dari simpul awal dimasukkan ke dalam antrian. Kemudian simpul pertama di dalam antrian akan diambil dan tetangga-tetangganya akan dimasukkan ke dalam antrian. Proses ini terus dilakukan sampai simpul tujuan atau file yang diinginkan ditemukan atau antrian kosong.

Dalam menggunakan algoritma BFS, pencarian dilakukan secara melebar terlebih dahulu sebelum melanjutkan ke simpul berikutnya. Hal ini berguna untuk menemukan solusi terdekat dari simpul awal ke simpul tujuan. Selain itu, BFS juga memastikan bahwa solusi yang ditemukan adalah yang terdekat dan optimal, karena pencarian dilakukan secara melebar dan tidak akan melompati simpul yang dapat menjadi solusi optimal.

3.1.2 Depth-First Search

Dalam pemecahan masalah Treasure Hunt, kita juga dapat menggunakan algoritma Depth-First Search (DFS) untuk melakukan pencarian solusi pada graf dinamis. Graf direpresentasikan menggunakan daftar ketetanggan atau adjacency list. Elemen stack berisi simpul atau node yang akan dikunjungi, dimana setiap simpul berisi nama folder atau file yang ada.

Pada awal pencarian, simpul awal atau root dimasukkan ke dalam stack. Kemudian, simpul pertama di dalam stack akan diambil dan simpul-simpul tetangganya akan dimasukkan ke dalam stack. Proses ini terus dilakukan sampai simpul tujuan atau file yang diinginkan ditemukan atau stack kosong.

Dalam menggunakan algoritma DFS, pencarian dilakukan secara mendalam terlebih dahulu sebelum melanjutkan ke simpul berikutnya. Hal ini dapat mempercepat pencarian jika simpul tujuan berada dalam kedalaman yang dalam dan jarang memiliki banyak simpul anak. Namun, solusi yang ditemukan mungkin tidak selalu terdekat atau optimal.

Dalam pencarian dengan algoritma DFS, perlu diperhatikan bahwa jika graf memiliki lingkaran atau siklus, maka pencarian akan terjebak dalam lingkaran tersebut dan tidak akan menemukan solusi. Oleh karena itu, perlu dilakukan pengecekan untuk mencegah terjadinya siklus atau lingkaran pada graf.

3.3 Ilustrasi Kasus Lain

Selain algoritma breadth-first search dan algoritma depth-first search dalam penelusuran rute pada permasalahan maze treasure hunt, terdapat algoritma lain yang merupakan gabungan dari breadth-first search dan depth-first search, yaitu BFS-DFS Hybrid Algorithm atau BFS-DFS Gabungan.

Caranya adalah dengan melakukan pencarian BFS dari start point ke end point terlebih dahulu, namun tidak mencatat jalur yang ditemukan. Setelah sampai di end point, dilakukan pencarian DFS dari end point ke start point dengan mencatat jalur yang ditemukan.

Selama proses pencarian DFS, jika ditemukan simpul yang telah dikunjungi sebelumnya melalui BFS, maka simpul tersebut dilewati. Dengan demikian, strategi ini menggabungkan kecepatan pencarian BFS dalam menemukan tujuan dengan keakuratan pencarian DFS dalam mencatat jalur yang ditemukan.

Namun, perlu diperhatikan bahwa strategi ini tidak selalu lebih efeketif daripada algoritma A* atau metode pencarian jalur lainnya. Keefektifannya tergantung pada kompleksitas maze dan kondisi spesifik lainnya.

BAB IV
Implementasi dan Pengujian

4.1 Implementasi Program (Pseudocode)

4.1.1. Implementasi BFS

```
function Node DoAction(goBackHome: bool, mw: MainWindow, delayDuration:int) ->
Node

{Mengembalikan Node yang melewati seluruh Treasure dan kembali ke Titik K jika
goBackHome true, jika goBackHome false akan hanya akan melewati seluruh
Treasure.}

KAMUS
   ResultNode, tempNode : Node
   mainProgramDelayCount, goBackHomeDelayCount, treasureFound : int
   mainStopWatch, tSPStopwatch : Stopwatch

   {Fungsi dan Prosedur Antara}
   procedure BFS()
      I.S. M, CurNode, discovered, liveNode belum terdefinisi
      F.S. M, CurNode, discovered, liveNode telah terdefinisi

   procedure BFS(input/output m : Maze, mw : MainWindow )
      I.S. M, CurNode, discovered, liveNode belum terdefinisi
      F.S. M, CurNode, discovered, liveNode telah terdefinisi seuai dengan m

   procedure EnqueueValidNode(input/output liveNode : Node, discover :
Stack(tuple))
   {Memasukkan Node yang berada di atas, kanan, bawah, kiri yang memenuhi keadaan
    dengan aturan FIFO}
      I.S liveNode terdefinisi
      F.S Node berhasil ditambahkan ke liveNode sesuai dengan keadaan

   procedure InitializeStartingPoint(input/output liveNode : Node, input startI,
   startJ : int)
   {Memasukkan Node yang berada di atas, kanan, bawah, kiri yang memenuhi keadaan
    dengan aturan FIFO dengan titik awal K}
      I.S liveNode terdefinisi
      F.S Node berhasil ditambahkan ke liveNode sesuai dengan keadaan

   function Append(a, b: Node, m: Maze) -> Node
   {Node yang merupakan hasil gabungan dua buah node a dan b}

   function isDiscoveredFirst(n: Node, discoveredNodes : List<Node>) -> bool
   {Mengembalikan true jika Node n merupakan elemen dari discoveredNodes}

   function

ALGORITMA

   {Inisialisasi resultNode}
   resultNode <- null
   {Inisialisasi penghitungan waktu}
   MainStopwatch.StartNew();
   mainProgram <- 0
   tSP <- 0
   treasureFound <- 0
   this.InitializeStartingPoint(m.Start.i, m.Start.j)
   mainProgram <- mainProgram + 1

   while (treasureFound != this.m.TreasureCount) do
      while (true) do
```

```
         Node tempCurNode <- liveNode.Dequeue()
         if (!BFS.IsDiscovered(tempCurNode, discovered)) then
            CurNode <- tempCurNode
            This.discovered.Add(tempCurNode)
            mainProgramDelayCount <- mainProgramDelayCount + 1
         if (this.m.Content[curNode.I][curNode.J] == "T") then
            treasureFound++
            this.m.Content[CurNode.I][CurNode.J] <- "R"
            resultNode <- Append(resultNode, CurNode, m)
            this.InitializeStartingPoint(CurNode.I, CurNode.J)
            mainProgramDelayCount++
            Break
         EnqueueValidNode()
         mainProgramDelayCount <- mainProgramDelayCount + 1
   MainStopwatch.Stop()
   TimeSpan elapsedTime = mainStopwatch.Elapsed -
TimeSpan.FromMilliseconds(mainProgramDelayCount * delayDuration)

   if (goBackHome) then
      tSPStowatch <- Stopwatch.StartNew()
      this.initializeStartingPoint(resultNode.I, resultNode.J)
      tSP++

         while(this.liveNode.Count!=0) do
            tempNode = liveNode.Pop()

            if (!IsDiscovered(tempNode, this.discovered)) then
               tSP++
               this.curNode <- tempNode
               this.discovered.Add(tempNode)

               if (this.m.Content[curNode.I][curNode.J] == "K") break
            EnqueueValidNode()
            tSP++

         resultNode = BFS.Append(resultNode, curNode, this.m)
         tSPStopwatch.Stop()
         TimeSpan elapsedTSP = goBackHomeStopwatch.Elapsed -
TimeSpan.FromMilliseconds(goBackHomeDelayCount * delayDuration)
         elapsedTime += elapsedTSP

   return resultNode
```

4.1.2. Implementasi DFS

```
function Node doAction(goBackHome: bool, mw: MainWindow, delayDuration:int) ->
Node

{Mengembalikan Node yang melewati seluruh Treasure dan kembali ke Titik K jika
goBackHome true, jika goBackHome false akan hanya akan melewati seluruh Treasure}

KAMUS
   ResultNode, tempNode : Node
   mainProgramDelayCount, goBackHomeDelayCount, treasureFound : int
   MainStopWatch, goBackHomeStopwatch : Stopwatch

   {Fungsi dan Prosedur Antara}
   procedure DFS()
      I.S. M, CurNode, discovered, liveNode belum terdefinisi
```

```
      F.S. M, CurNode, discovered, liveNode telah terdefinisi

   procedure DFS(input/output m : Maze, mw : MainWindow )
      I.S. M, CurNode, discovered, liveNode belum terdefinisi
      F.S. M, CurNode, discovered, liveNode telah terdefinisi seuai dengan m

   procedure pushNode(input/output liveNode : Node, discover : Stack(tuple))
   {Memasukkan Node yang berada di atas, kanan, bawah, kiri yang memenuhi keadaan
    dengan aturan LIFO}
      I.S liveNode terdefinisi
      F.S Node berhasil ditambahkan ke liveNode sesuai dengan keadaan

   procedure initializeStartingPoint(input/output liveNode : Node, input startI,
   startJ : int)
   {Memasukkan Node yang berada di atas, kanan, bawah, kiri yang memenuhi keadaan
    dengan aturan LIFO dengan titik awal K}
      I.S liveNode terdefinisi
      F.S Node berhasil ditambahkan ke liveNode sesuai dengan keadaan

   function discoverPoint(i,j : int) -> bool
   {Mengembalikan true jika titik i,j berada pada discover}

   function isDiscoveredFirst(n: Node, discoveredNodes : List<Node>) -> bool
   {Mengembalikan true jika Node n merupakan elemen dari discoveredNodes}

   function
```

ALGORITMA

```
   {Inisialisasi resultNode}
   resultNode <- null
   {Inisialisasi penghitungan waktu}
   mainProgramDelayCount <- 0
   GoBackHomeDelayCount <- 0

   if(curNode != null) then
      resultNode <- BFS.Append(resultNode, curNode, this.m)
      initializeStartingPoint(this.m.Start.i, this.m.Start.j)
      mainProgramDelayCount <- mainProgramDelayCount + 1

      while(this.liveNode.Count != 0 ) do
         Node tempNode = liveNode.Pop()

         if (!BFS.IsDiscovered(tempNode, this.discovered) &&
!isDiscoveredFirst(tempNode, this.discovered)) then
         this.curNode <- tempNode
         this.discovered.Add(tempNode)
         mainProgramDelayCount <- mainProgramDelayCount + 1

         if (this.m.Content[curNode.I][curNode.J] == "T" &&
!curNode.hasInPath(curNode.I, curNode.J)) then

            if (curNode.isSubsetOf(resultNode)) then
               mainProgramDelayCount <- mainProgramDelayCount + 1

            while (Discovered.Count != 0) do
               Discovered.RemoveAt(Discovered.Count - 1)
            this.curNode.TreasureFound++

         if (CurNode.TreasureFound == this.m.TreasureCount) then break
         pushNode()
         mainProgramDelayCount <- mainProgramDelayCount + 1
```

```
   else
      treasureFound <- 0
      resultNode <- null
      this.initializeStartingPoint(m.Start.i, m.Start.j)
      mainProgramDelayCount <- mainProgramDelayCount + 1

      while (treasureFound != this.m.TreasureCount) do
         while (true) do
            Node tempCurNode <- liveNode.Pop()
            if (!BFS.IsDiscovered(tempCurNode, discovered)) then
               CurNode <- tempCurNode
               mainProgramDelayCount <- mainProgramDelayCount + 1
               if (this.m.Content[curNode.I][curNode.J] == "T" &&
!curNode.hasInPath(curNode.I, curNode.J)) then
                  treasureFound++
                  this.m.Content[CurNode.I][CurNode.J] <- "R"
                  resultNode <- BFS.Append(resultNode, CurNode, m)
                  this.initializeStartingPoint(CurNode.I, CurNode.J)
                  while (Discovered.Count != 0) do
                     Discovered.RemoveAt(Discovered.Count - 1)
                  mainProgramDelayCount++
                  Break
               pushNode()
               mainProgramDelayCount <- mainProgramDelayCount + 1
   mainStopwatch.Stop()
   TimeSpan elapsedTime = mainStopwatch.Elapsed -
TimeSpan.FromMilliseconds(mainProgramDelayCount * delayDuration)


   if (goBackHome) then
      goBackHomeStopwatch <- Stopwatch.StartNew()
      this.initializeStartingPoint(resultNode.I, resultNode.J)
      goBackHomeDelayCount++

         while(this.liveNode.Count!=0) do
            tempNode = liveNode.Pop()

         if (this.m.Content[tempNode.I][tempNode.J] == "K") then
            this.curNode <- tempNode
            this.discovered.Add(tempNode)
            Break

         if (!BFS.IsDiscovered(tempNode, this.discovered) &&
!curNode.hasInPath(curNode.I, curNode.J)) then
            goBackHomeDelayCount++
            this.curNode <- tempNode
            this.discovered.Add(tempNode)

            if (this.m.Content[curNode.I][curNode.J] == "K") break
            PushNode()
            GoBackHomeDelayCount++

         resultNode = BFS.Append(resultNode, curNode, this.m)
         goBackHomeStopwatch.Stop()
         TimeSpan elapsedTSP = goBackHomeStopwatch.Elapsed -
TimeSpan.FromMilliseconds(goBackHomeDelayCount * delayDuration)
         elapsedTime -= elapsedTSP

   return resultNode
```

4.2 Penjelasan Struktur Data

Berikut merupakan struktur data beserta spesifikasi program yang digunakan dalam membuat program:

1. Queue

   Implementasi struktur data Queue menggunakan collection pada kelas System.Collections.Generic. Queue digunakan untuk menyimpan antrian Node yang akan dikunjungi oleh program saat melakukan traversal dengan metode BFS (Breadth First Search). Berikut merupakan implementasi dari method yang digunakan.

   a. Enqueue, berfungsi untuk memasukkan Node ke dalam antrian dengan aturan FIFO atau first in first out.
   b. Dequeue, berfungsi untuk mengeluarkan Node dari antrian dengan aturan FIFO atau first in first out.

2. Stack

   Implementasi struktur data Stack menggunakan collection pada kelas System.Collections.Generic. Stack digunakan untuk menyimpan tumpukan Node yang akan dikunjungi oleh program saat melaukan traversal dengan metode DFS (Depth First Search). Berikut merupakan implementasi dari method yang digunakan.

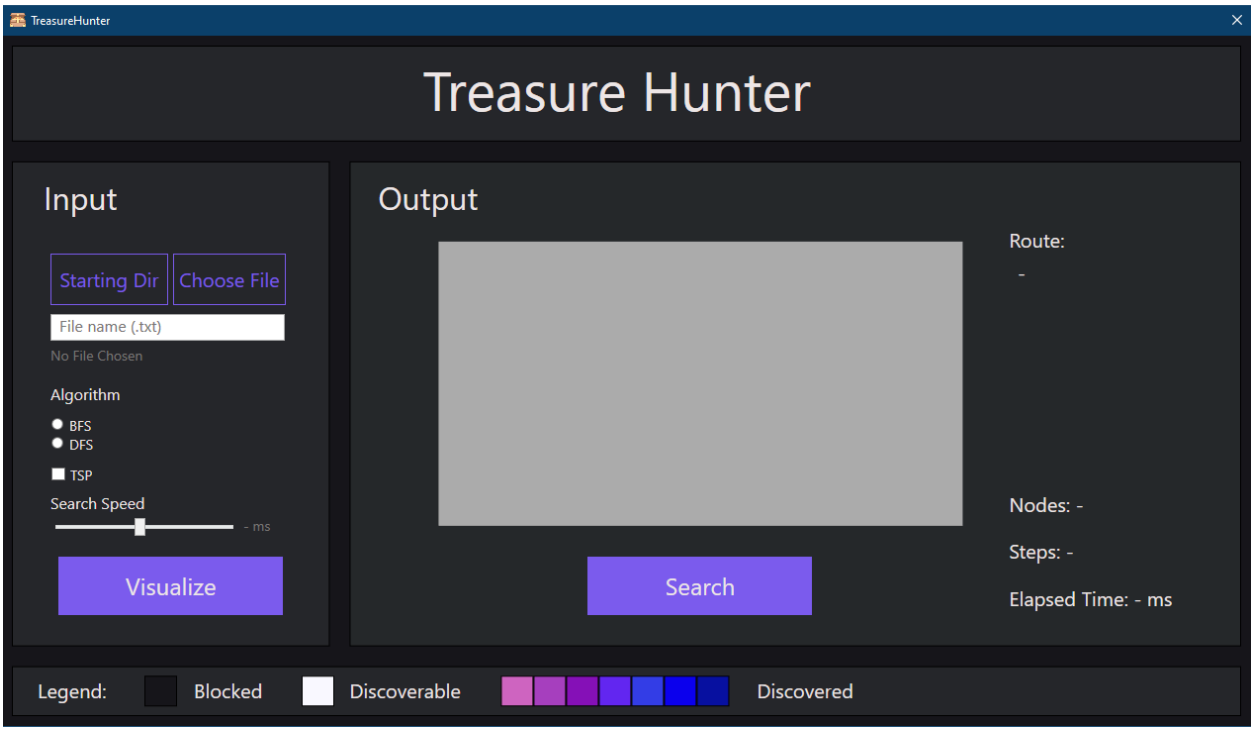
   a. Push, berfungsi untuk memasukkan Node ke dalam tumpukan dengan aturan LIFO atau last in first out.
   b. Pop, berfungsi untuk mengeluarkan Node dari tumpukan dengan aturan LIFO atau last in first out.

3. List & Tuple

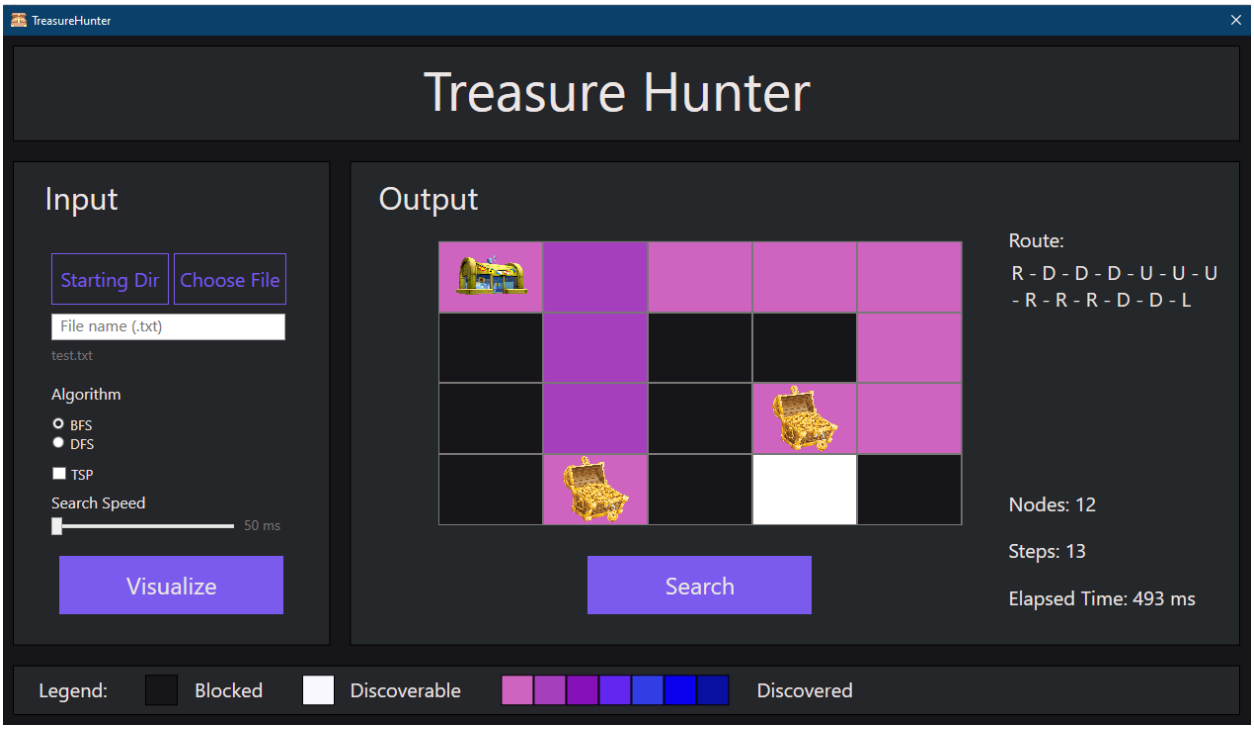
   Implementasi struktur data list dan tuple digunakan dalam kelas Maze. Struktur data list diterapkan untuk menyimpan content yang terdapat pada maze. Sementara struktur data tuple digunakan untuk menyimpan koordinat di maze.

4.3 Penjelasan Tata Cara Penggunaan Program

Untuk menjalankan program, pengguna cukup menjalankan “TreasureHunter.exe”. Setelah menjalankan program, pengguna akan diminta untuk melakukan input direktori awal pada button **Starting Dir**. Setelah itu, pengguna aka diminta kembali untuk melakukan input nama file yang ingin dicari pada bagian **text field** yang telah tersedia. Pengguna juga dapat langsung memilih file config yang hendak digunakan dengan menekan tombol **Choose File**. Kemudian, pengguna dapat memilih algoritma yang hendak digunakan, antara **BFS** atau **DFS** serta terdapat pula pilihan **TSP** yang dapat di-centang jika pengguna hendak menggunakannya. Hal lain yang dapat dikustomisasi pengguna adalah **Search Speed.** Pengguna dapat menyesuaikan search speed yang hendak digunakan dengan melakukan drag pada toggle yang telah tersedia. Ketika pengguna menekan tombol **Visualize** maka akan muncul hasil pembacaan file config yang telah dipilih dalam bentuk map treasure hunt. Apabila pengguna menekan tombol **Search**, maka akan ditampilkan rute terbaik berdasarkan algoritma yang dipilih sekaligus alur penelusuran rute-nya. Ketika penelusuran rute selesai, maka akan ditampilkan rute yang dilewati, jumlah **nodes** dan **steps** yang dilalui, serta **waktu** yang ditempuh. Berikut merupakan tampilan dari program atau desktop application yang telah kami buat.



***Gambar 4.3.1**. Tampilan Awal Program*
Sumber: Dokumen Pribadi Penulis

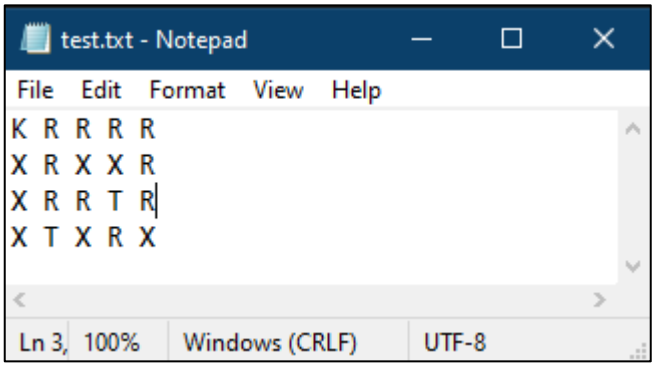


***Gambar 4.3.2**. Tampilan Akhir Program*
Sumber: Dokumen Pribadi Penulis

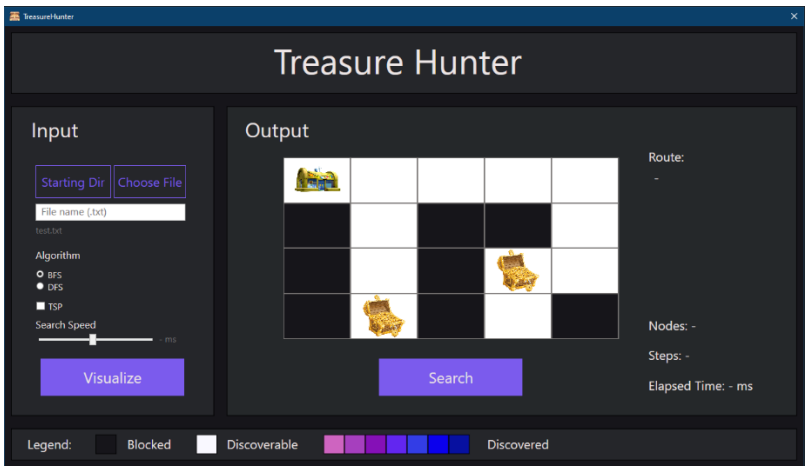
4.4 Hasil Pengujian

4.4.1. BFS

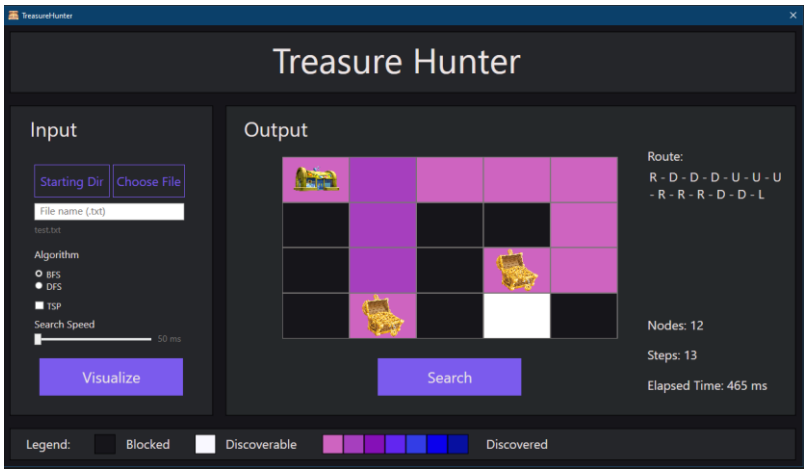
1. Pengujian 1



Gambar 4.4.1.1.1. *Config File Pengujian 1*
Sumber: Dokumen Pribadi Penulis

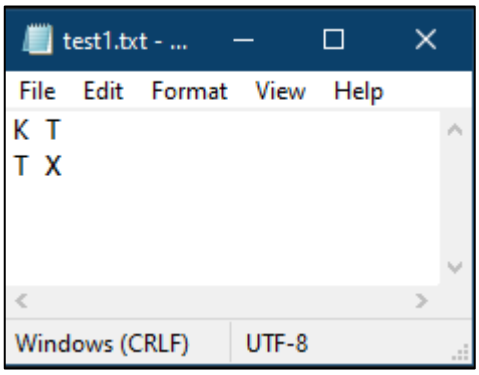


Gambar 4.4.1.1.2. *Visualisasi Pengujian 1*
Sumber: Dokumen Pribadi Penulis

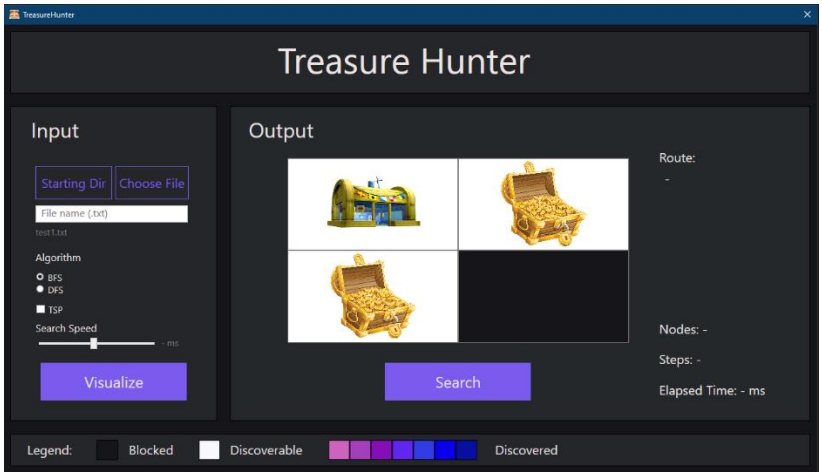


Gambar 4.4.1.1.3. *Pencarian Pengujian 1*
Sumber: Dokumen Pribadi Penulis

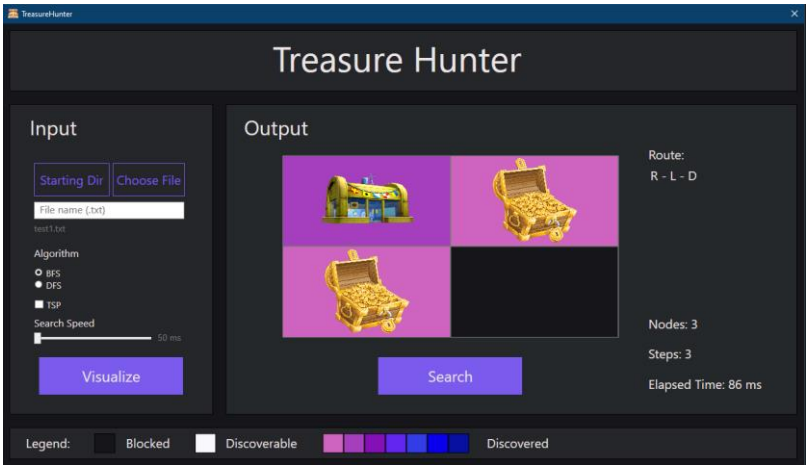
2. Pengujian 2



Gambar 4.4.1.2.1. *Config File Pengujian 2*
Sumber: Dokumen Pribadi Penulis

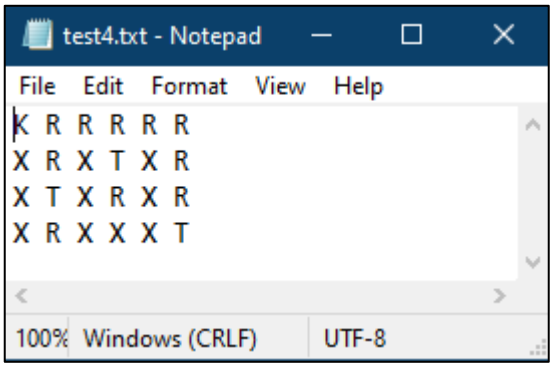


Gambar 4.4.1.2.2. *Visualisasi Pengujian 2*
Sumber: Dokumen Pribadi Penulis

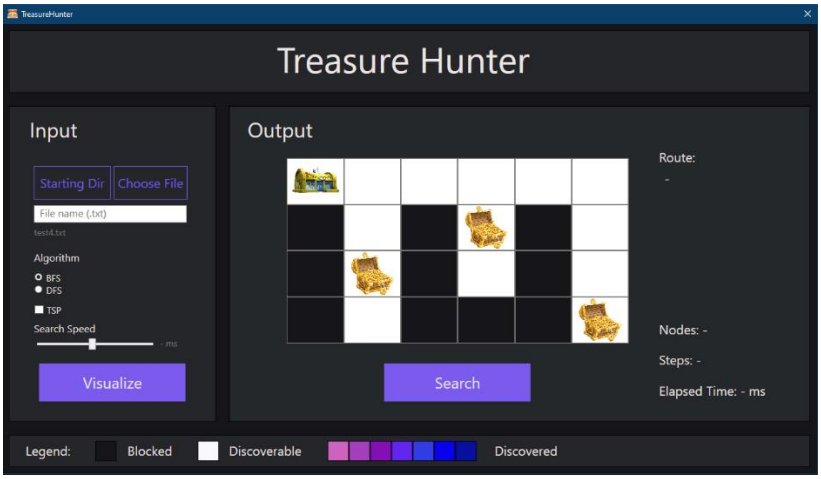


Gambar 4.4.1.2.3. *Pencarian Pengujian 2*
Sumber: Dokumen Pribadi Penulis

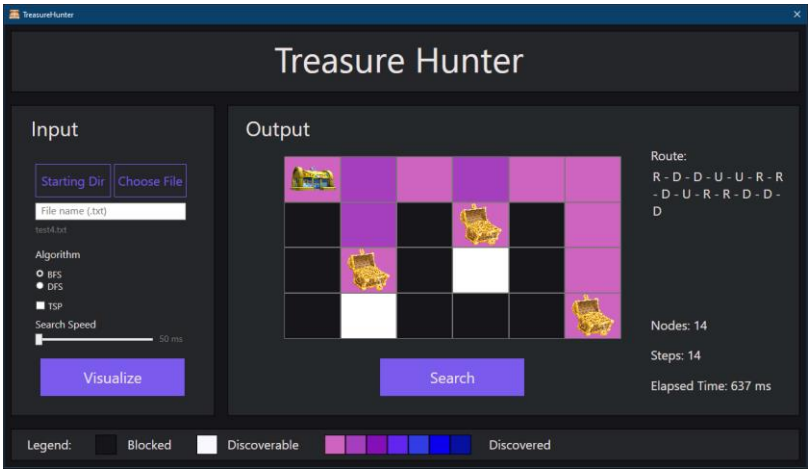
3. Pengujian 3



Gambar 4.4.1.3.1. *Config File Pengujian 3*
Sumber: Dokumen Pribadi Penulis

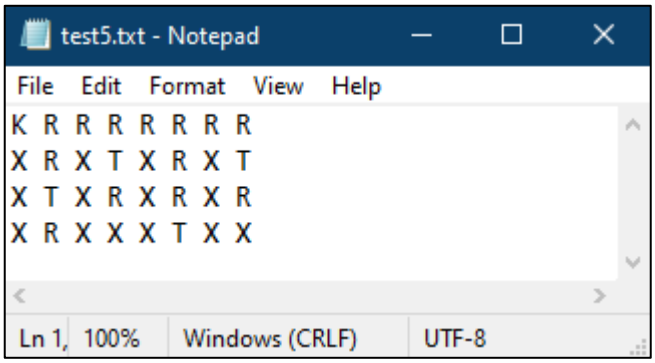


Gambar 4.4.1.3.2. *Visualisasi Pengujian 3*
Sumber: Dokumen Pribadi Penulis

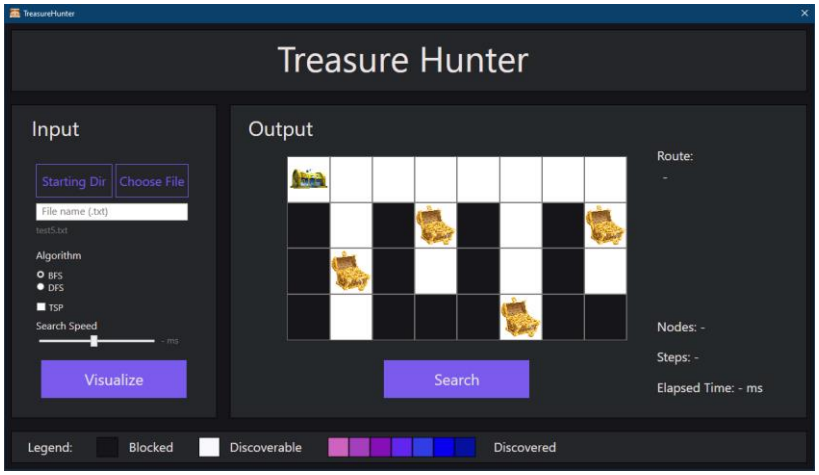


Gambar 4.4.1.3.3. *Pencarian Pengujian 3*
Sumber: Dokumen Pribadi Penulis

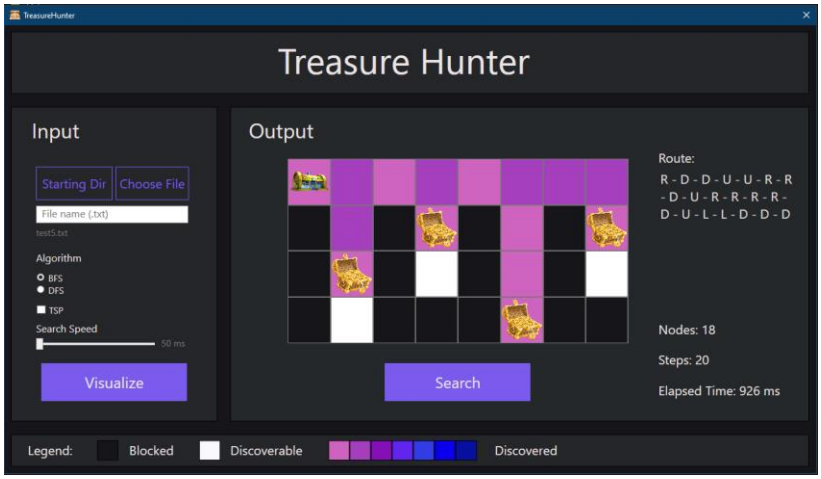
4. Pengujian 4



Gambar 4.4.1.4.1. *Config File Pengujian 4*
Sumber: Dokumen Pribadi Penulis

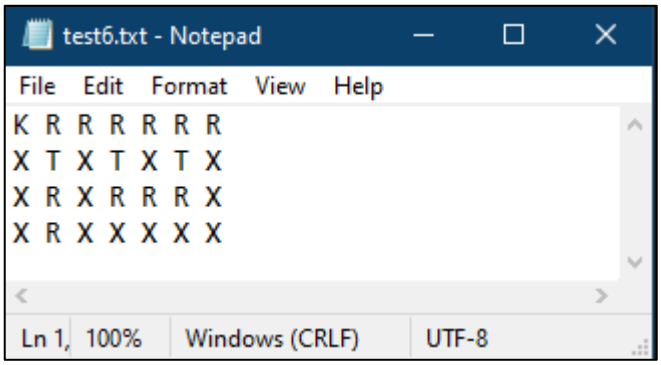


Gambar 4.4.1.4.2. *Visualisasi Pengujian 4*
Sumber: Dokumen Pribadi Penulis

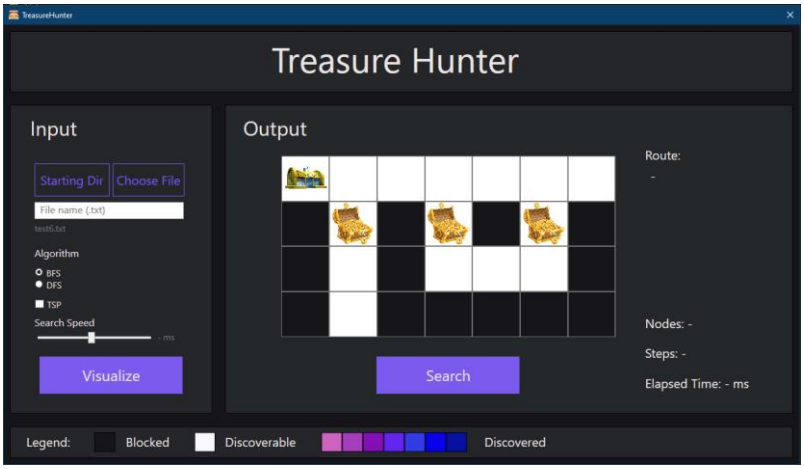


Gambar 4.4.1.4.3. *Pencarian Pengujian 4*
Sumber: Dokumen Pribadi Penulis

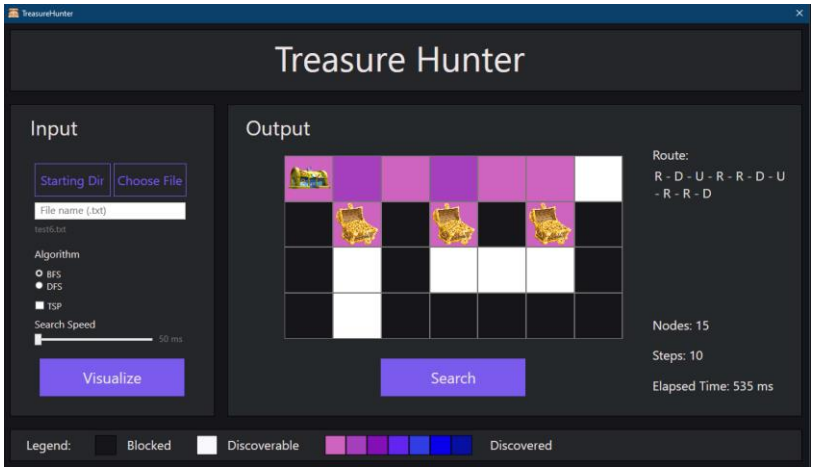
5. Pengujian 5



Gambar 4.4.1.5.1. *Config File Pengujian 5*
Sumber: Dokumen Pribadi Penulis

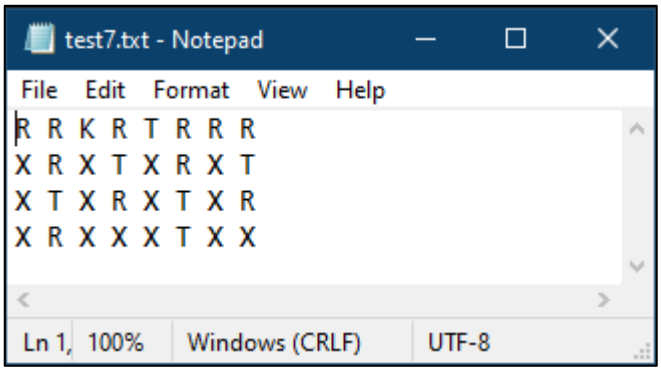


Gambar 4.4.1.5.2. *Visualisasi Pengujian 5*
Sumber: Dokumen Pribadi Penulis

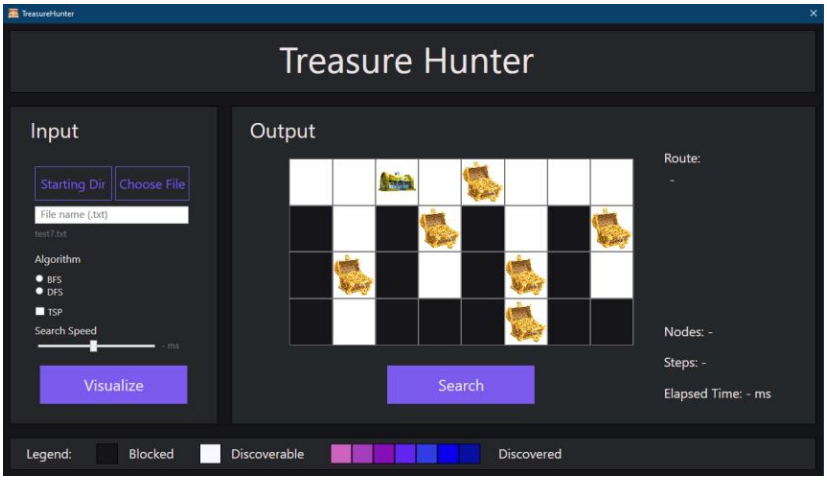


Gambar 4.4.1.5.3. *Pencarian Pengujian 5*
Sumber: Dokumen Pribadi Penulis

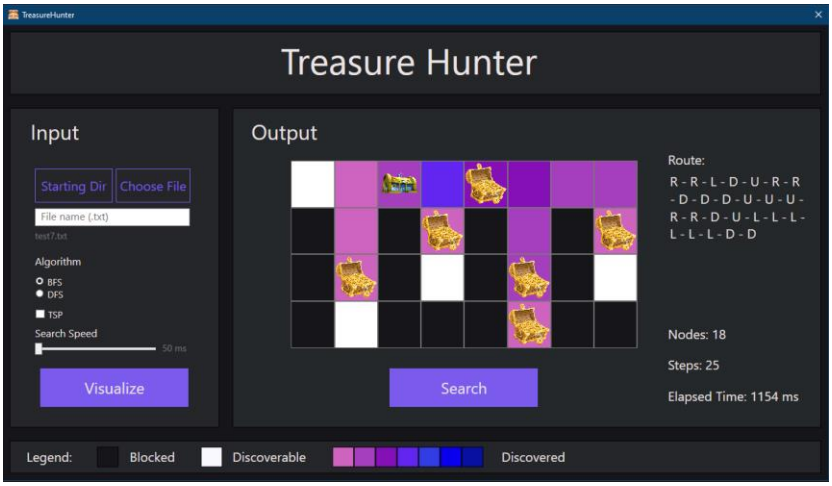
6. Pengujian 6



Gambar 4.4.1.6.1. *Config File Pengujian 6*
Sumber: Dokumen Pribadi Penulis

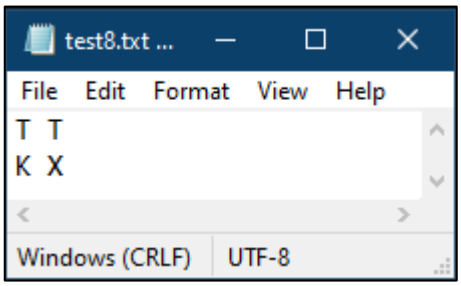


Gambar 4.4.1.6.2. *Visualisasi Pengujian 6*
Sumber: Dokumen Pribadi Penulis

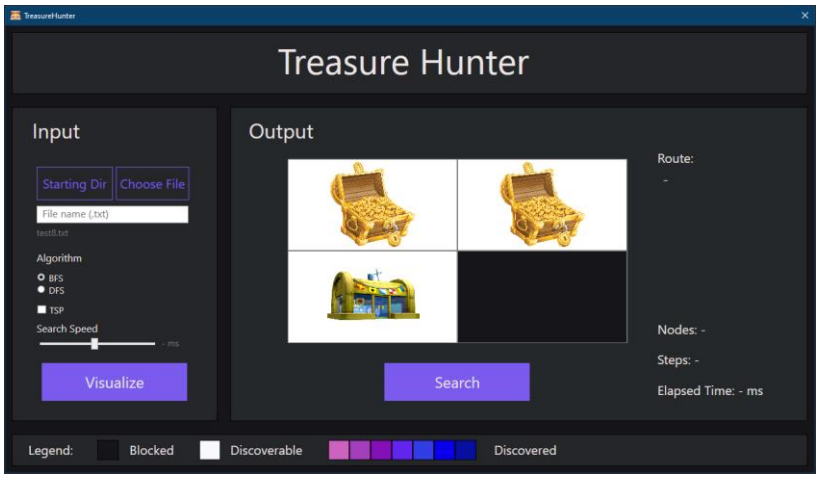


Gambar 4.4.1.6.3. *Pencarian Pengujian 6*
Sumber: Dokumen Pribadi Penulis

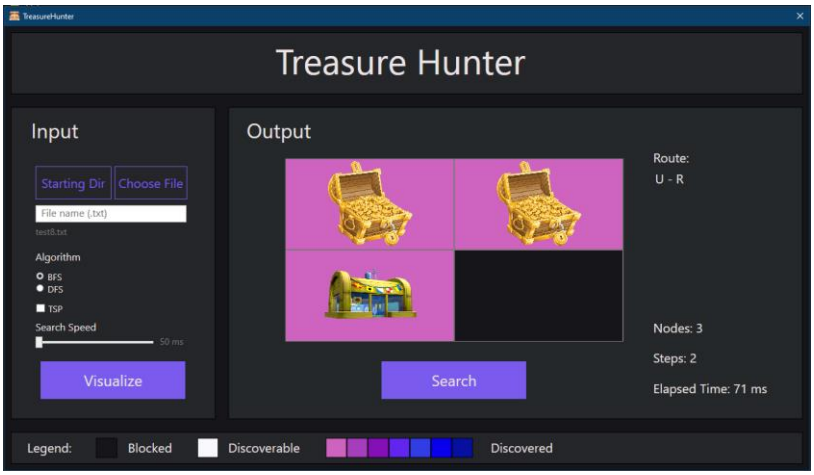
7. Pengujian 7



Gambar 4.4.1.7.1. *Config File Pengujian 7*
Sumber: Dokumen Pribadi Penulis

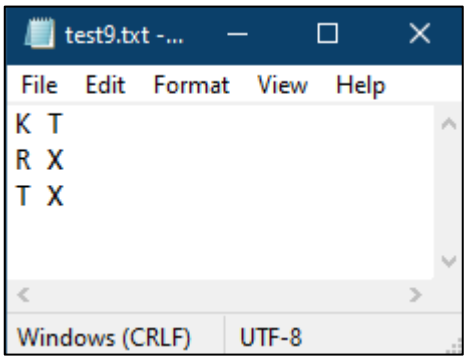


Gambar 4.4.1.7.2. *Visualisasi Pengujian 7*
Sumber: Dokumen Pribadi Penulis

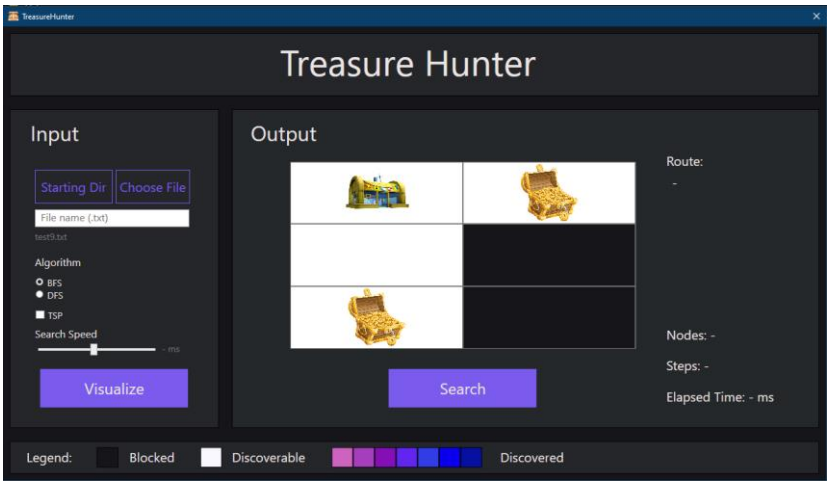


Gambar 4.4.1.7.3. *Pencarian Pengujian 7*
Sumber: Dokumen Pribadi Penulis

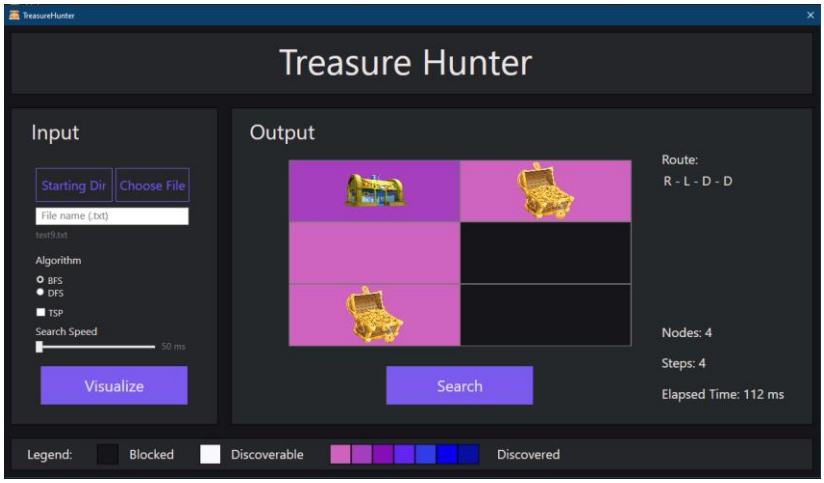
8. Pengujian 8



Gambar 4.4.1.8.1. *Config File Pengujian 8*
Sumber: Dokumen Pribadi Penulis

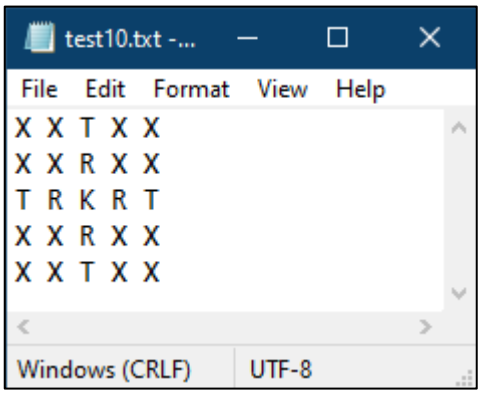


Gambar 4.4.1.8.2. *Visualisasi Pengujian 8*
Sumber: Dokumen Pribadi Penulis

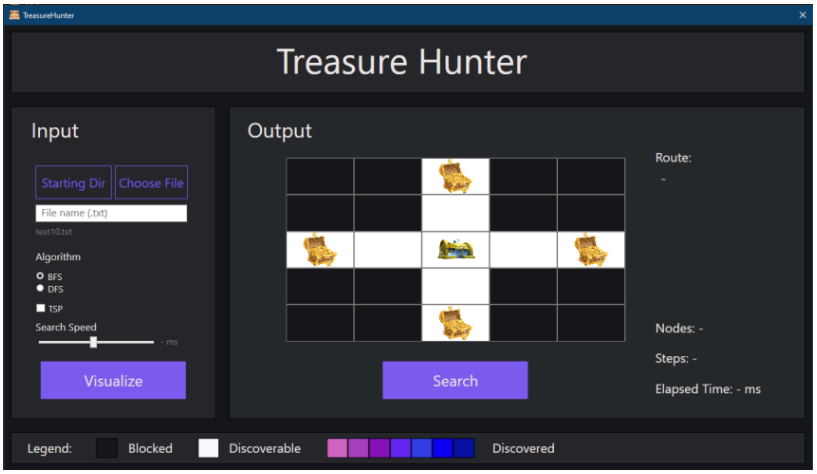


Gambar 4.4.1.8.3. *Pencarian Pengujian 8*
Sumber: Dokumen Pribadi Penulis

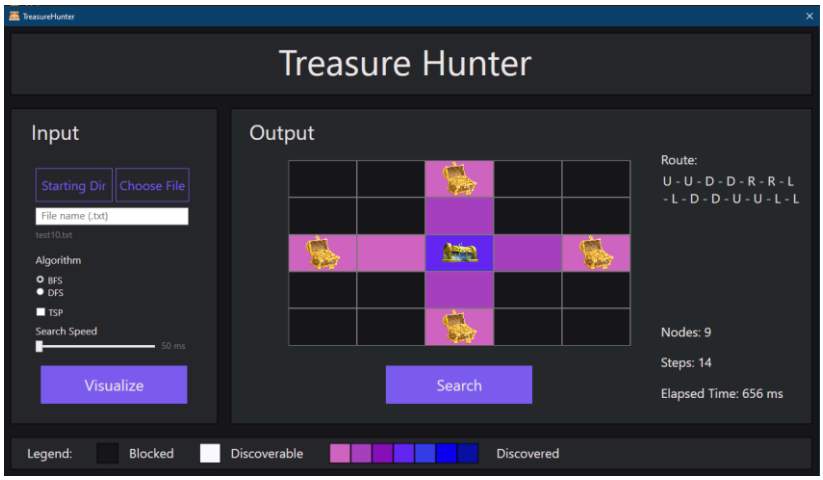
9. Pengujian 9



Gambar 4.4.1.9.1. *Config File Pengujian 9*
Sumber: Dokumen Pribadi Penulis



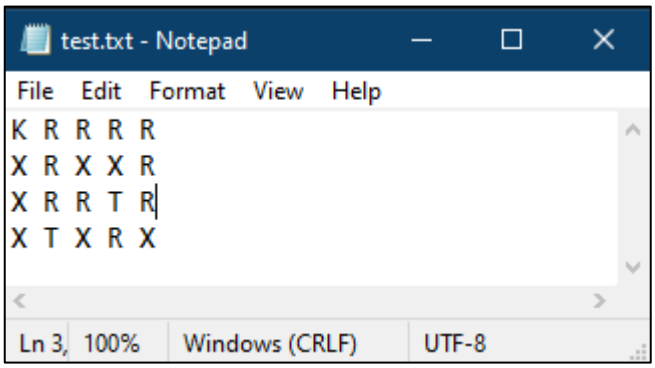
Gambar 4.4.1.9.2. *Visualisasi Pengujian 9*
Sumber: Dokumen Pribadi Penulis



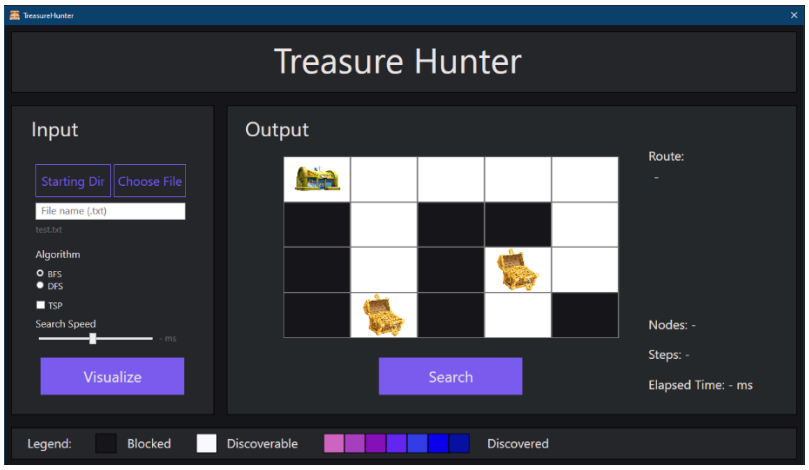
Gambar 4.4.1.9.3. *Pencarian Pengujian 9*
Sumber: Dokumen Pribadi Penulis

4.4.2. DFS

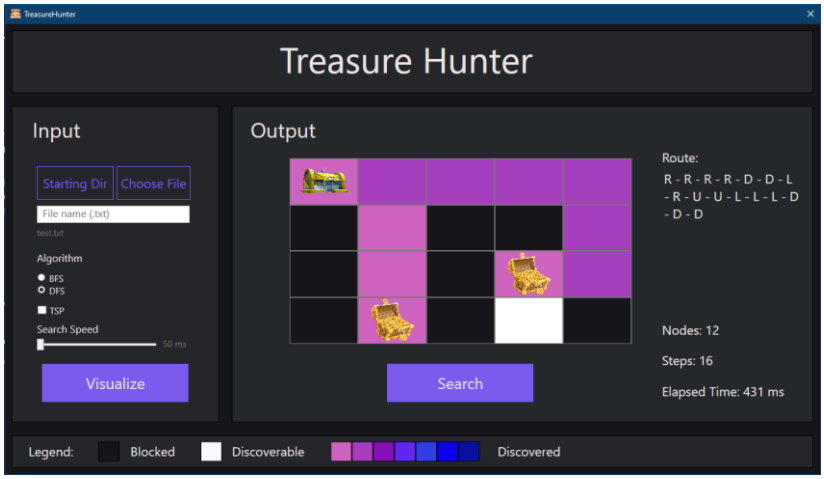
1. Pengujian 1



Gambar 4.4.2.1.1. *Config File Pengujian 1*
Sumber: Dokumen Pribadi Penulis

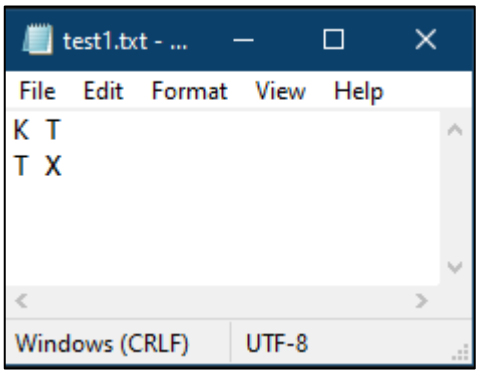


Gambar 4.4.2.1.2. *Visualisasi Pengujian 1*
Sumber: Dokumen Pribadi Penulis

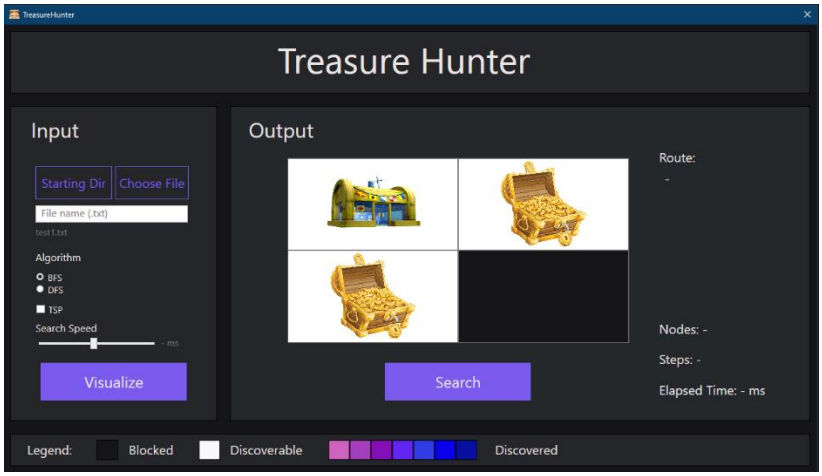


Gambar 4.4.2.1.3. *Pencarian Pengujian 1*
Sumber: Dokumen Pribadi Penulis

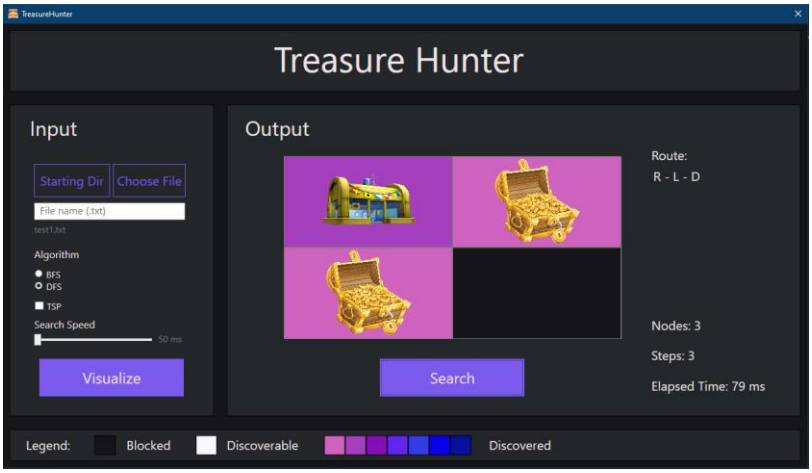
2. Pengujian 2



Gambar 4.4.2.2.1. *Config File Pengujian 2*
Sumber: Dokumen Pribadi Penulis

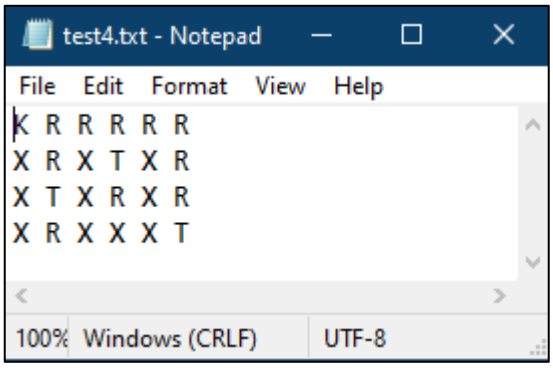


Gambar 4.4.2.2.2. *Visualisasi Pengujian 2*
Sumber: Dokumen Pribadi Penulis

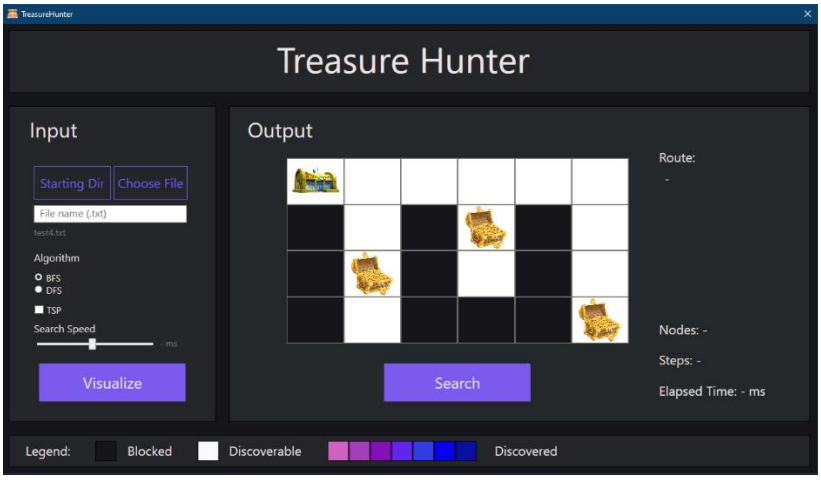


Gambar 4.4.2.2.3. *Pencarian Pengujian 2*
Sumber: Dokumen Pribadi Penulis

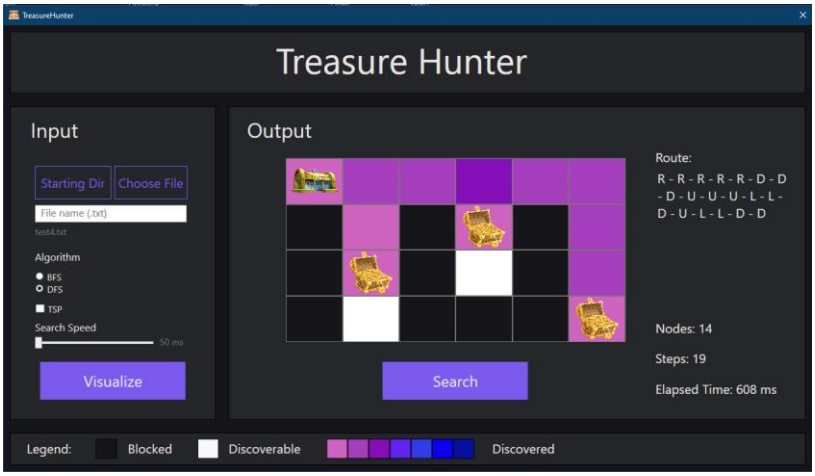
3. Pengujian 3



Gambar 4.4.2.3.1. *Config File Pengujian 3*
Sumber: Dokumen Pribadi Penulis

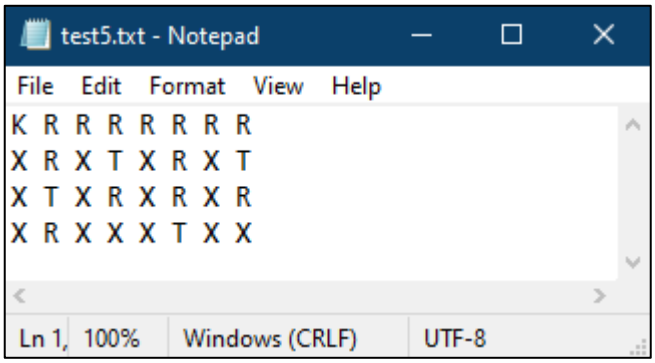


Gambar 4.4.2.3.2. *Visualisasi Pengujian 3*
Sumber: Dokumen Pribadi Penulis

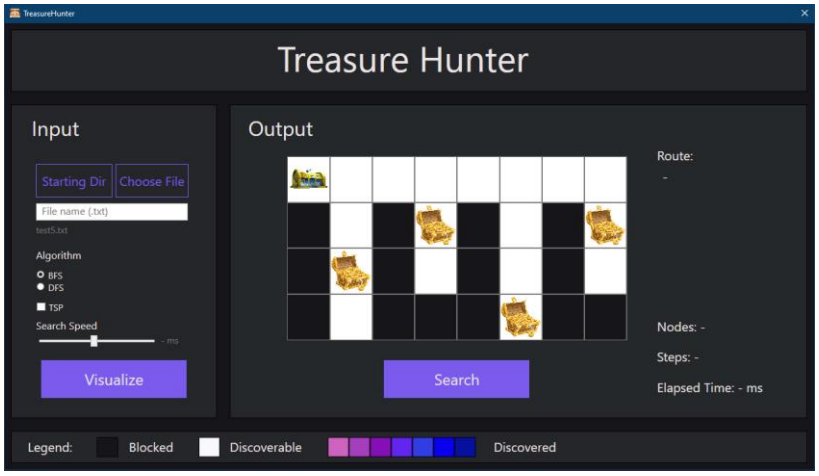


Gambar 4.4.2.3.3. *Pencarian Pengujian 3*
Sumber: Dokumen Pribadi Penulis

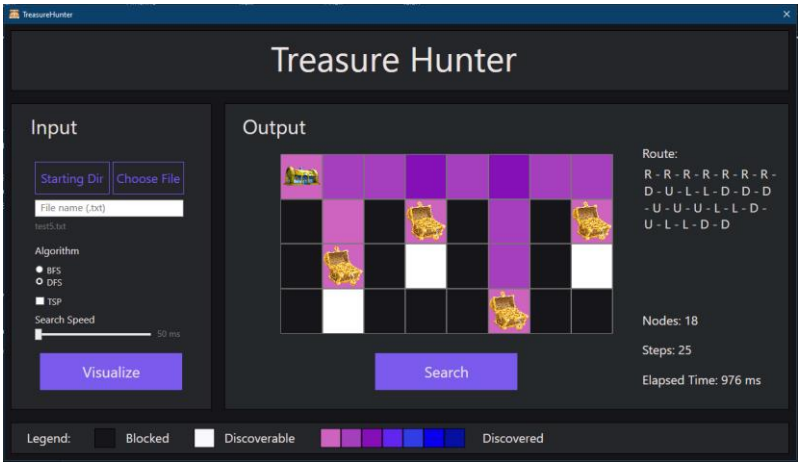
4. Pengujian 4



Gambar 4.4.2.4.1. *Config File Pengujian 4*
Sumber: Dokumen Pribadi Penulis

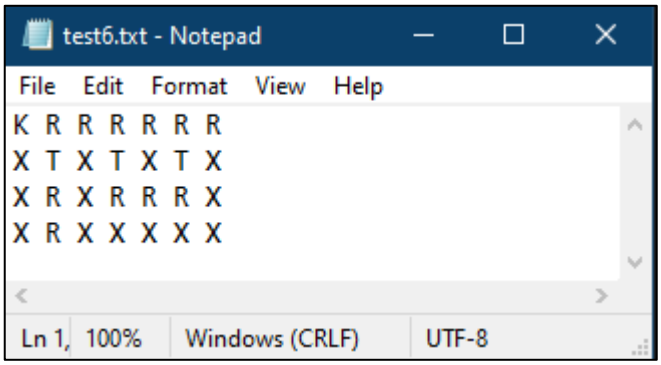


Gambar 4.4.2.4.2. *Visualisasi Pengujian 4*
Sumber: Dokumen Pribadi Penulis

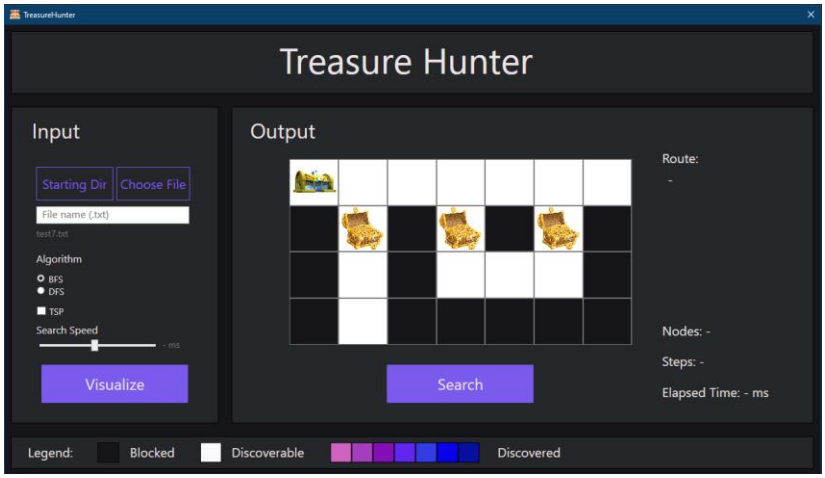


Gambar 4.4.2.4.3. *Pencarian Pengujian 4*
Sumber: Dokumen Pribadi Penulis

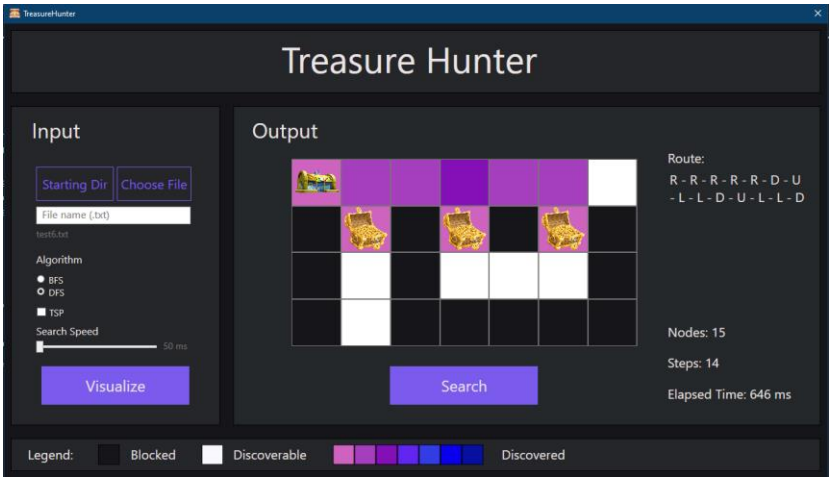
5. Pengujian 5



Gambar 4.4.2.5.1. *Config File Pengujian 5*
Sumber: Dokumen Pribadi Penulis

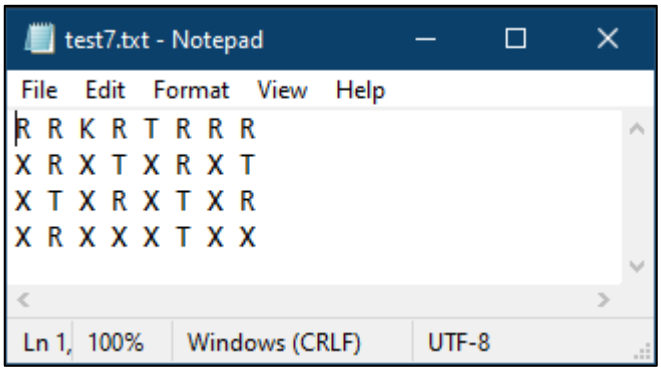


Gambar 4.4.2.5.2. *Visualisasi Pengujian 5*
Sumber: Dokumen Pribadi Penulis

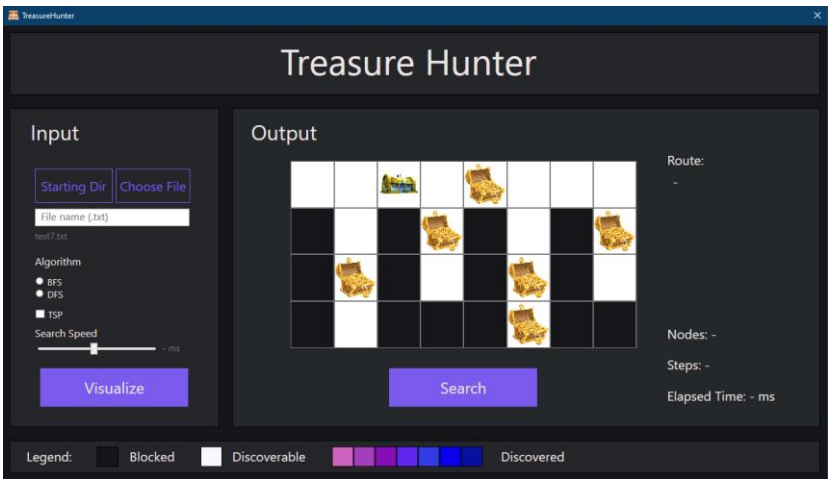


Gambar 4.4.2.5.3. *Pencarian Pengujian 5*
Sumber: Dokumen Pribadi Penulis

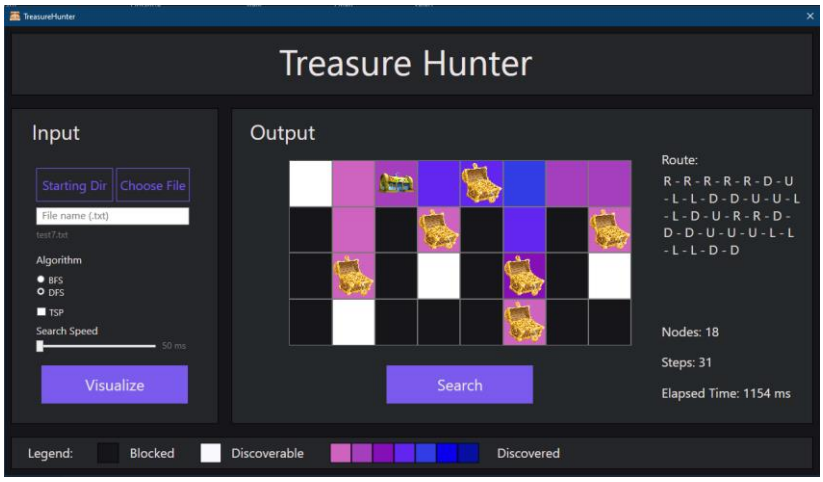
6. Pengujian 6



Gambar 4.4.2.6.1. *Config File Pengujian 6*
Sumber: Dokumen Pribadi Penulis

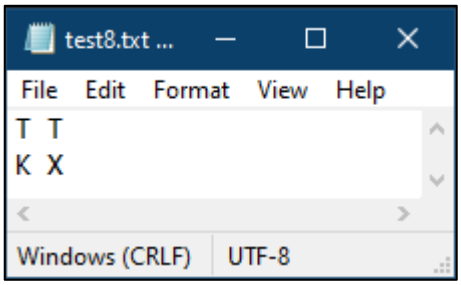


Gambar 4.4.2.6.2. *Visualisasi Pengujian 6*
Sumber: Dokumen Pribadi Penulis

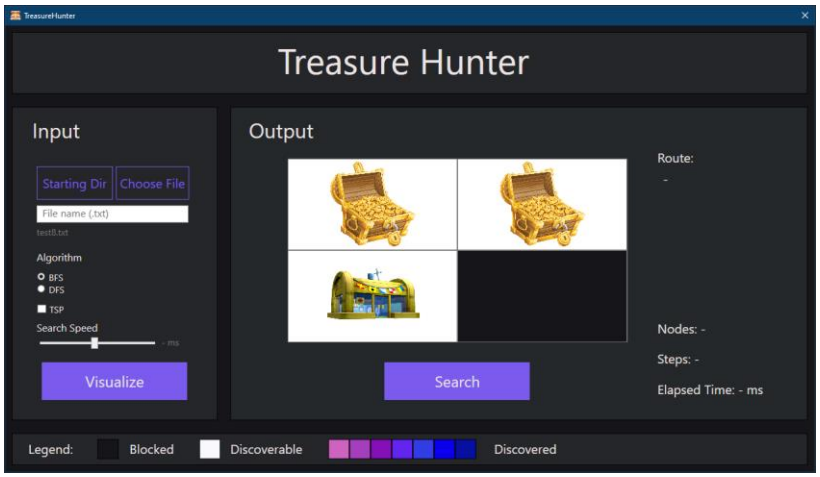


Gambar 4.4.2.6.3. *Pencarian Pengujian 6*
Sumber: Dokumen Pribadi Penulis

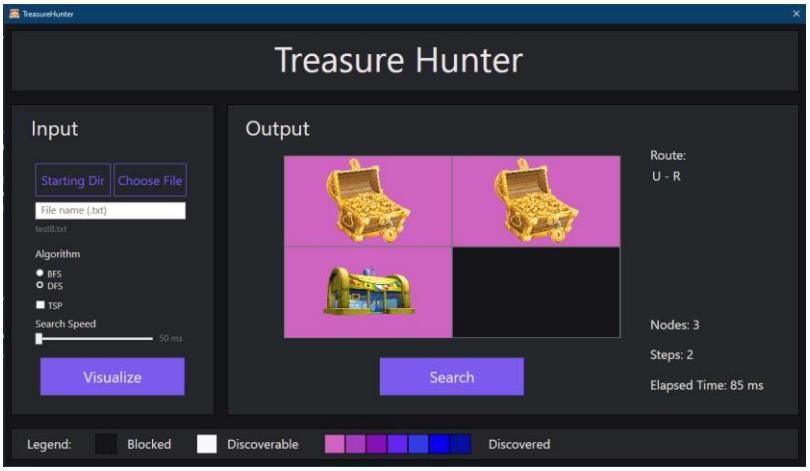
7. Pengujian 7



Gambar 4.4.2.7.1. *Config File Pengujian 7*
Sumber: Dokumen Pribadi Penulis

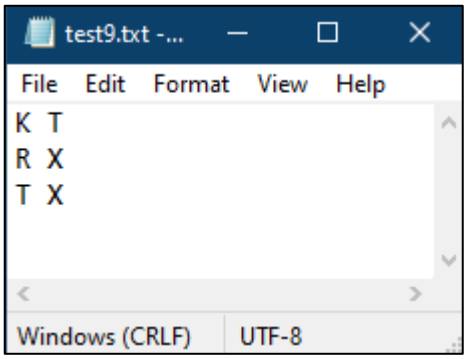


Gambar 4.4.2.7.2. *Visualisasi Pengujian 7*
Sumber: Dokumen Pribadi Penulis

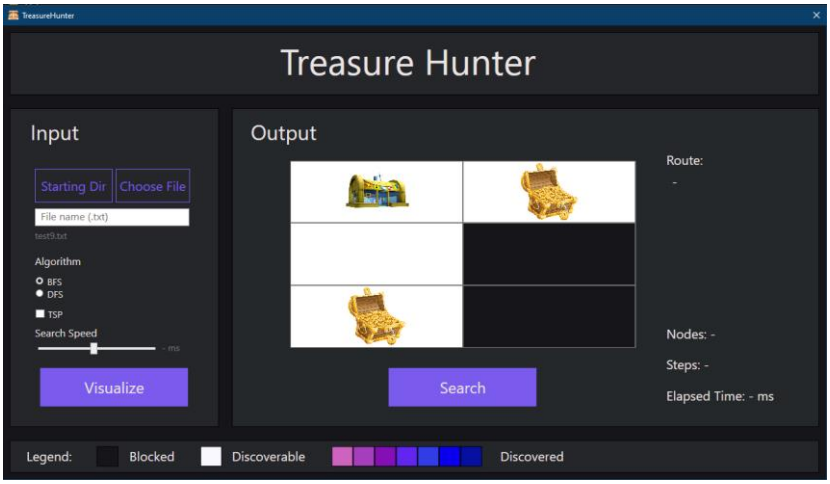


Gambar 4.4.2.7.3. *Pencarian Pengujian 7*
Sumber: Dokumen Pribadi Penulis

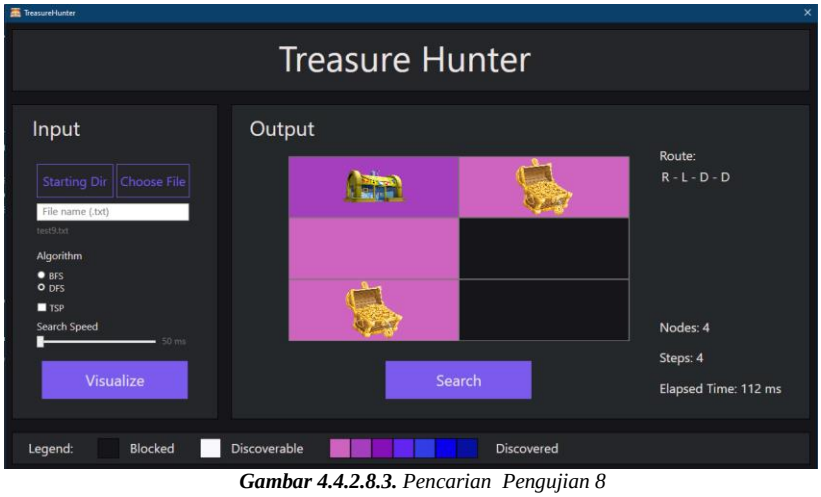
8. Pengujian 8



Gambar 4.4.2.8.1. *Config File Pengujian 8*
Sumber: Dokumen Pribadi Penulis

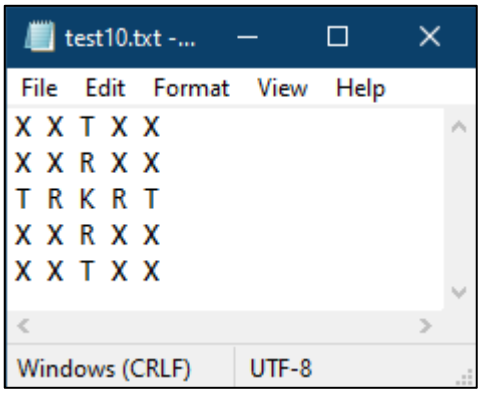


Gambar 4.4.2.8.2. *Visualisasi Pengujian 8*
Sumber: Dokumen Pribadi Penulis

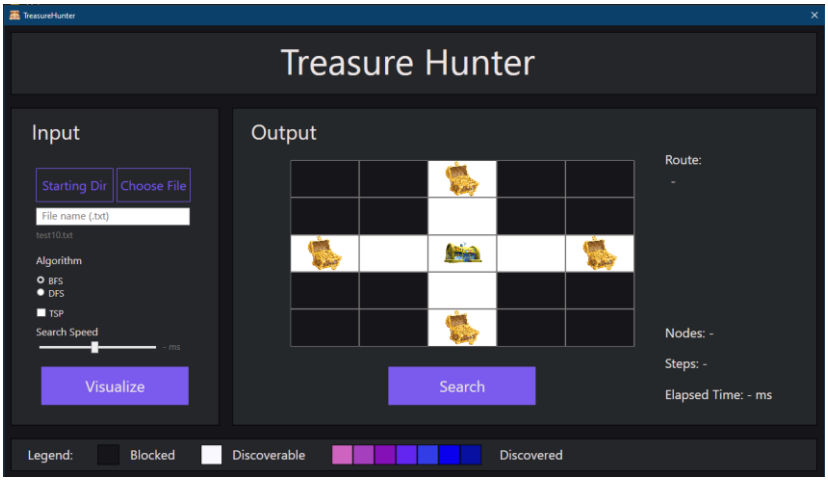


Gambar 4.4.2.8.3. *Pencarian Pengujian 8*
Sumber: Dokumen Pribadi Penulis

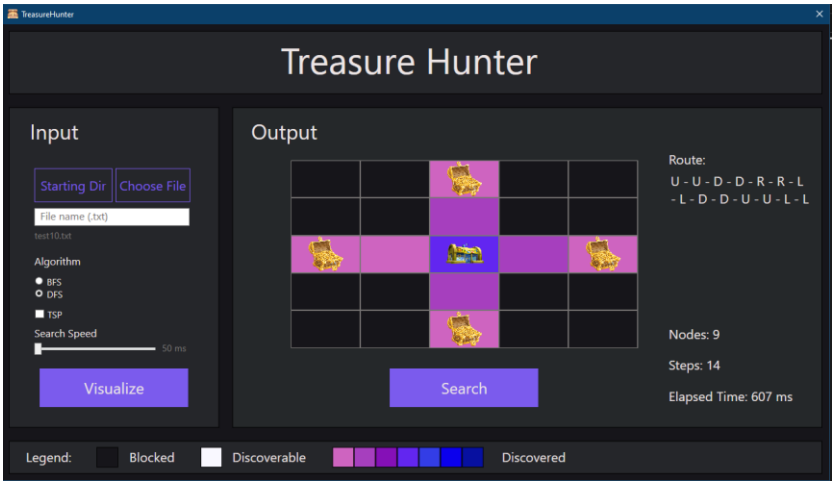
9. Pengujian 9



Gambar 4.4.2.9.1. *Config File Pengujian 9*
Sumber: Dokumen Pribadi Penulis



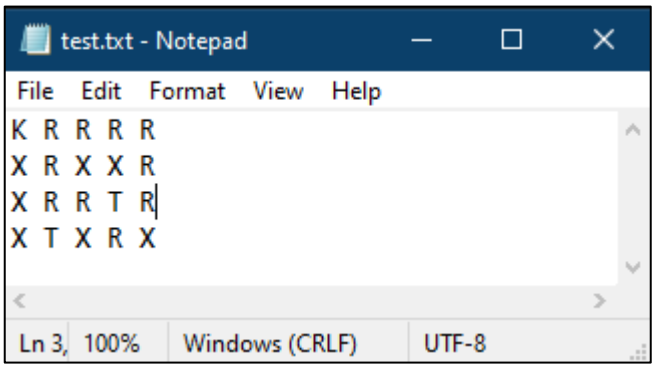
Gambar 4.4.2.9.2. *Visualisasi Pengujian 9*
Sumber: Dokumen Pribadi Penulis



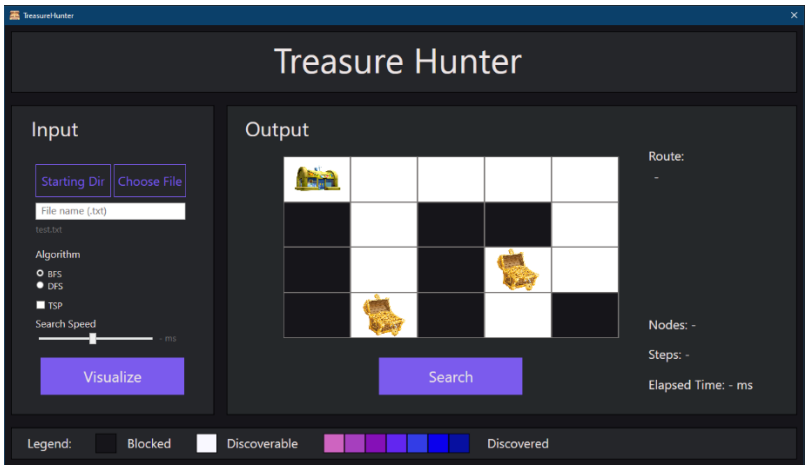
Gambar 4.4.2.9.3. *Pencarian Pengujian 9*
Sumber: Dokumen Pribadi Penulis

4.4.3. TSP

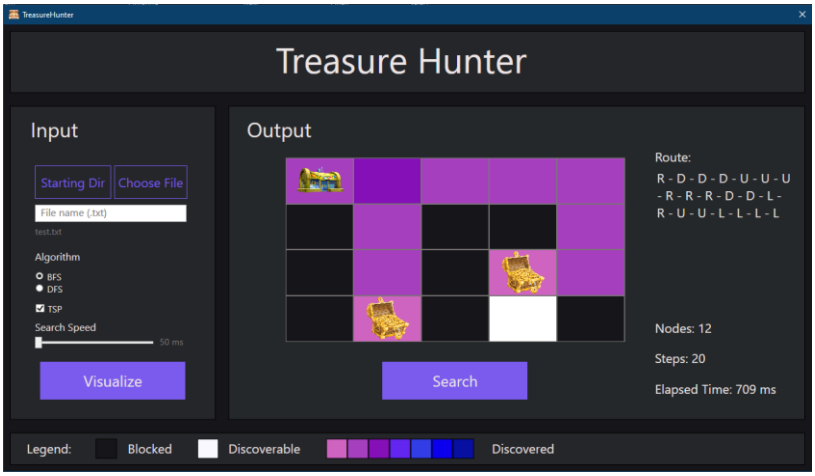
1. Pengujian 1



Gambar 4.4.3.1.1. *Config File Pengujian 1*
Sumber: Dokumen Pribadi Penulis

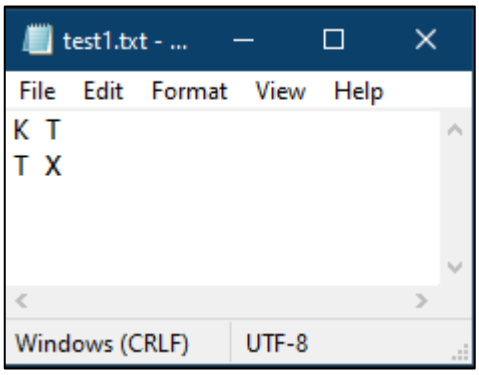


Gambar 4.4.3.1.2. *Visualisasi Pengujian 1*
Sumber: Dokumen Pribadi Penulis

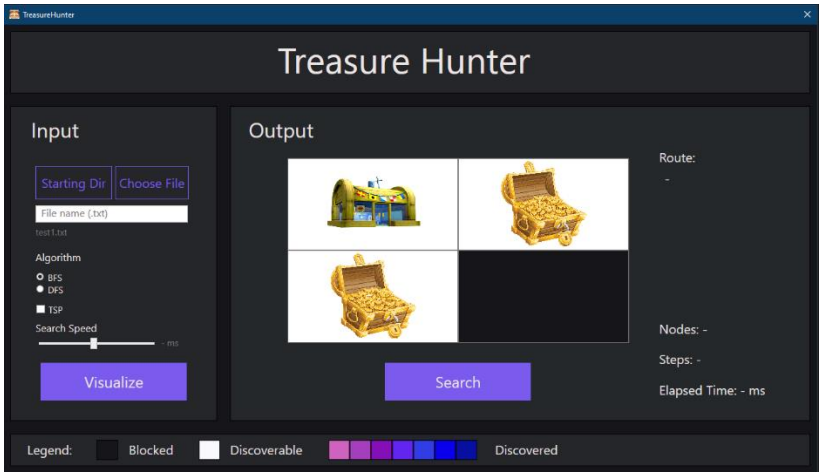


Gambar 4.4.3.1.3. *Pencarian Pengujian 1*
Sumber: Dokumen Pribadi Penulis

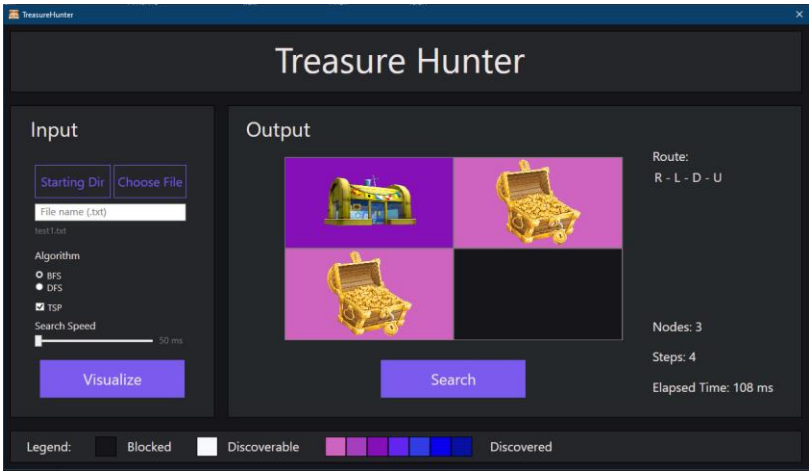
2. Pengujian 2



Gambar 4.4.3.2.1. *Config File Pengujian 2*
Sumber: Dokumen Pribadi Penulis

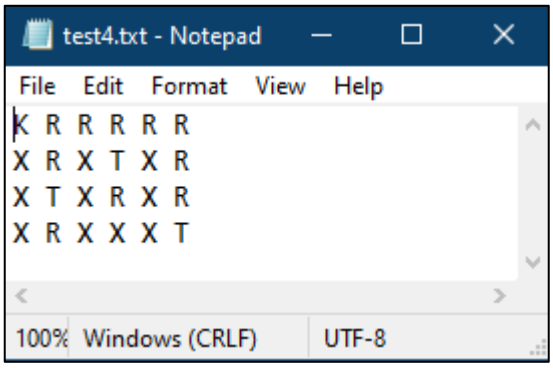


Gambar 4.4.3.2.2. *Visualisasi Pengujian 2*
Sumber: Dokumen Pribadi Penulis

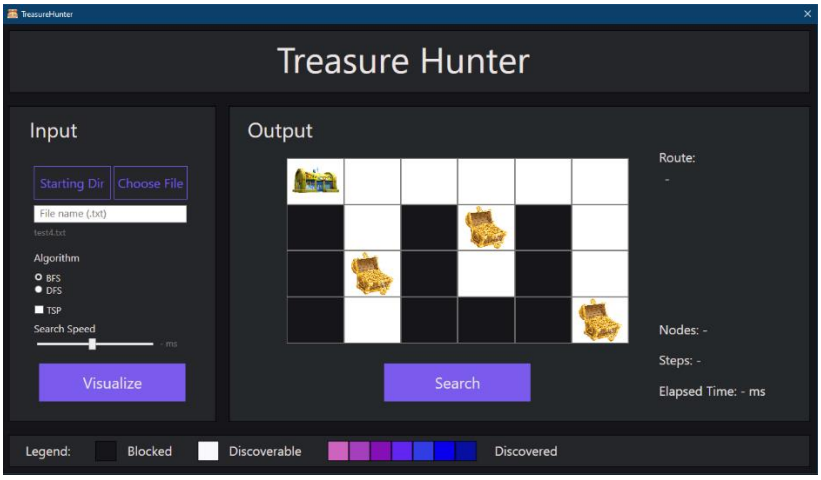


Gambar 4.4.3.2.3. *Pencarian Pengujian 2*
Sumber: Dokumen Pribadi Penulis

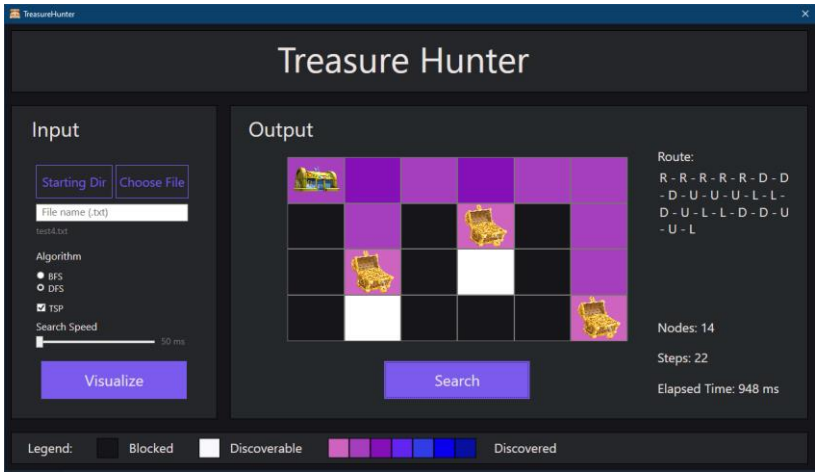
3. Pengujian 3



Gambar 4.4.3.3.1. *Config File Pengujian 3*
Sumber: Dokumen Pribadi Penulis

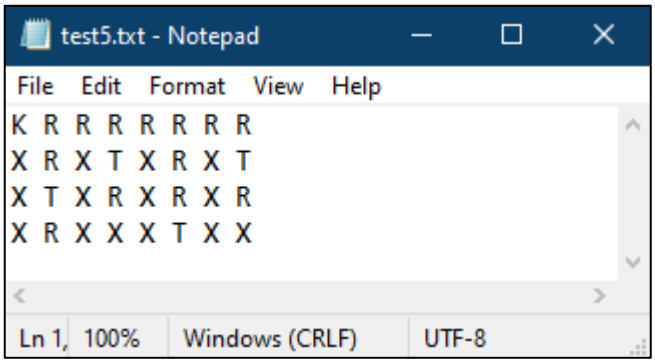


Gambar 4.4.3.3.2. *Visualisasi Pengujian 3*
Sumber: Dokumen Pribadi Penulis

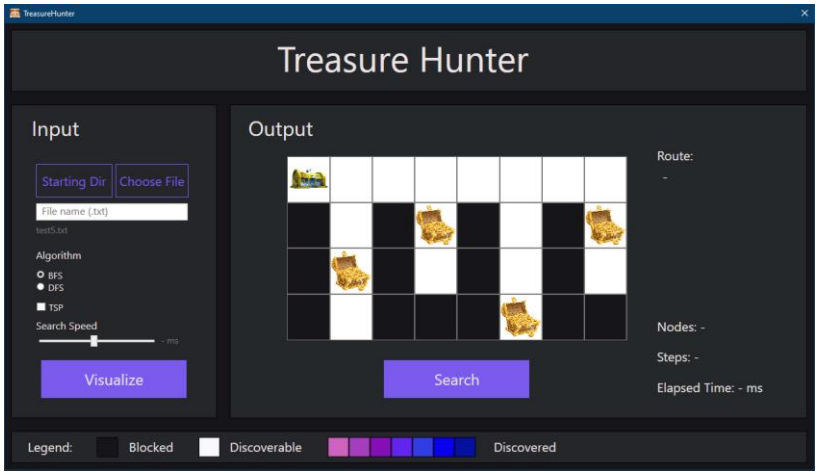


Gambar 4.4.3.3.3. *Pencarian Pengujian 3*
Sumber: Dokumen Pribadi Penulis

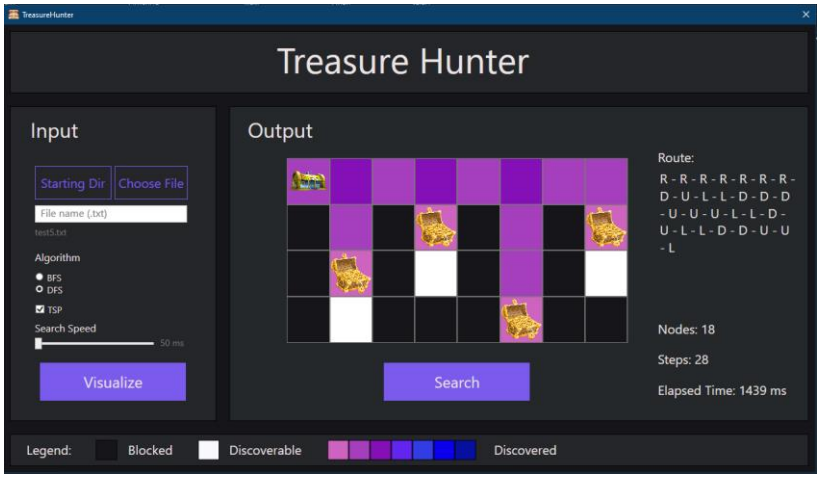
4. Pengujian 4



Gambar 4.4.3.4.1. *Config File Pengujian 4*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.3.4.2. *Visualisasi Pengujian 4*
Sumber: Dokumen Pribadi Penulis



Gambar 4.4.3.4.3. *Pencarian Pengujian 4*
Sumber: Dokumen Pribadi Penulis

4.5 Analisis Desain Solusi Algoritma BFS dan DFS

Berdasarkan hasil pengujian 1, algoritma BFS menempuh 13 steps dengan rute sebagai berikut, R- D - D - D – U – U – U – R – R – R – D – D – L. Sementara algoritma DFS menempuh 16 steps melalui rute sebagai berikut, R – R – R – R – D – D – L – R – U – U – L – L – L – D – D – D. Pada kasus ini, desain solusi algoritma BFS lebih efektif dibanding solusi algoritma DFS. Jika diperhatikan, target treasure yang dicari berada di beberapa cabang yang berbeda-beda, sehingga dapat disimpulkan bahwa penggunaan algoritma BFS lebih efektif digunakan pada pencarian di peta yang memiliki cukup banyak cabang.

Namun, pada pengujian 2, 5, 7, 8, dan 9, keduanya menempuh jumlah langkah yang sama untuk setiap pengujian. Misalnya pada pengujian 9, keduanya menempuh 14 steps, dengan rute yang dilewati BFS adalah sebagai berikut, U – U – D – D – R – R – L – L – D – D – U – U – L – L. Sementara itu rute yang dilewati oleh algoritma DFS sama seperti rute algoritma BFS, yaitu sebagai berikut, U – U – D – D – R – R – L – L – D – D – U – U – L – L. Oleh karena itu, dapat disimpulkan bahwa desain algoritma DFS dan BFS untuk permasalahan ini akan memiliki tingkat efektifitas yang sama pada kasus dimana cabang-cabang peta tidak saling terhubung satu sama lain di tengah-tengah rute. Cabang-cabang tersebut hanya terhubung sekali di awal peta.

Sementara itu, dari 9 kali pengujian, algoritma DFS sama sekali tidak pernah mengungguli jumlah langkah yang ditempuh oleh algoritma BFS. Pada pengujian 6, justru algoritma DFS tertinggal 6 langkah dibandingkan dengan algoritma BFS, dimana BFS hanya menempuh 25 langkah, sedangkan DFS menempuh 31 langkah. Hal ini membuktikan bahwa algoritma BFS lebih efektif untuk menyelesaikan permasalahan maze treasure hunt.

BAB V
Kesimpulan Saran dan Refleksi

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 Aplikasi Desktop yang dapat melakukan pencarian rute dari sebuah titik start tertentu menuju treasure/harta karun yang terdapat dalam map atau peta menggunakan algoritma BFS dan DFS. Kami juga melakukan pengembangan dari konsep tersebut untuk menyelesaikan permasalahan TSP (Travelling Salesman Problem).

Kami juga telah berhasil memanfaatkan Windows Presentation Foundation (WPF) pada bahasa pemrograman C# untuk membuat Graphical User Interface (GUI) dari program kami. GUI akan menerima masukkan file dari user, dilanjutkan dengan menampilkan rute terbaik untuk tiba di treasure-treasure dalam peta, serta memvisualisasikan langkah yang dilewati untuk mendapatkan rute tersebut.

Kesimpulan yang kami dapatkan dari pengerjaan tugas besar ini, antara lain:

- Algoritma breadth-first search dan depth-first search dapat digunakan untuk menyelesaikan permasalahan maze treasure hunt problem. Algoritma breadth-first seacrh melakukan penelusuran rute secara menyebar terlebih dahulu, sedangkan algoritma depth-first seacrh melakukan secara mendalam terlebih dahulu.

5.2 Saran

Pengembangan desktop application menggunakan C# cukup memerlukan pertimbangan yang mendalam. Apabila menggunakan WinForm, maka desktop application yang dibuat hanya dapat digunakan dan dikembangkan melalui sistem operasi Windows. Selain tidak versatile, pengembangan menggunakan WinForm menghambat pengerjaan bagi pengembang yang tidak memiliki computer dengan sistem operasi Windows. Namun, jika menggunakan Windows Presentation Foundation maka performanya akan lebih lambat karena WPF menggunakan teknologi rendering yang kompleks, sehingga membutuhkan sumber daya yang lebih besar dibandingkan dengan WinForms. Oleh karena itu, untuk kedepannya, pengembangan desktop application dapat dibuat dengan menggunakan pustaka yang berbeda sehingga dapat membuat desktop application yang lebih versatile dan performa yang baik.

5.3 Refleksi

Refleksi dari kelompok kami mungkin terkait komunikasi dan sistem pengerjaan yang dilakukan selama rentang waktu tugas besar ini. Dari segi komunikasi, mungkin seharusnya dilakukan lebih sering secara offline atau pertemuan terjadwal, agar komunikasi dapat terjalin dengan lebih baik lagi. Sementara dari sistem pengerjaan, pembagian yang dilakukan di awal mungkin bisa lebih merata, misalnya : seharusnya GUI tidak hanya menjadi tanggung jawab seorang saja karena memiliki tingkat kekompleks-an yang cukup tinggi. Namun, permasalahan sistem pembagian tugas tersebut telah ditangani di pertengahan rentang waktu pengerjaan, sehingga tugas besar ini dapat diselesaikan dengan baik.

5.4 Tanggapan

Tugas Besar 2 IF2211 – Strategi Algoritma 2022/2023 cukup menantang. Hal ini disebabkan oleh kami yang dipacu untuk mempelajari bahasa pemrograman C# yang sebelumnya belum pernah dipelajari secara mendalam. Selain itu, kami juga didorong untuk melakukan eksplorasi cukup mendalam terkait pengembangan Desktop Application melalui visual studio (WPF).

Referensi

Munir, Rinaldi, Nur Ulfa Mauladevi. 2023, “Diktat Breadth/Depth First Search”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf, diakses pada 17 Maret 2023.

Microsoft. “C# Documentation”, https://learn.microsoft.com/en-us/dotnet/csharp/, diakses pada 14 Maret 2023.

Microsoft. “Create a Windows Forms app in Visual Studio with C#”, https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?toc=%2Fvisualstudio%2Fget-started%2Fcsharp%2Ftoc.json&bc=%2Fvisualstudio%2Fget-started%2Fcsharp%2Fbreadcrumb%2Ftoc.json&view=vs-2022, diakses pada 12 Maret 2023.

Microsoft. “Create a simple WPF in Visual Studi with C#”, https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-wpf?view=vs-2022, diakses pada 12 Maret 2023.

Munir, Rinaldi. 2023, “Tugas besar 2: Pengaplikasisan Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tubes2-Stima-2023.pdf, diakses pada 2 Maret 2023.

Pranala Terkait

Link Repository:
https://github.com/rifqifarhansyah/Tubes2_dicarryVieridanZaki
Link Video :
https://drive.google.com/drive/folders/1X1tOkBCgvDR5mcRdpOnLcZ1mk7aLuEYP