

LAPORAN TUGAS BESAR
Aplikasi Algoritma BFS dan DFS dalam persoalan Treasure Hunt
IF2211 Strategi Algoritma



Oleh:

13521104 Muhammad Zaydan Athallah

13521138 Johann Christian Kandani

13521170 Haziq Abiyyu Mahdy

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022/2023

BAB 1: Deskripsi Tugas

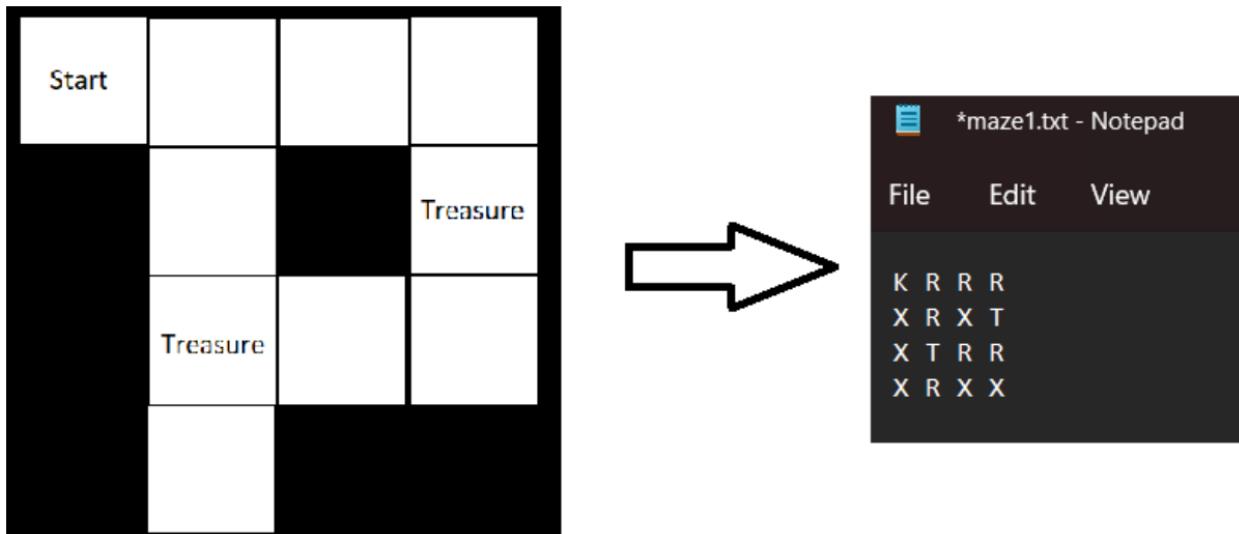
Tuan Krabs menemukan sebuah labirin distorsi terletak tepat di bawah Krusty Krab bernama El Doremi yang Ia yakini mempunyai sejumlah harta karun di dalamnya dan tentu saja Ia ingin mengambil harta karunnya. Dikarenakan labirinnya dapat mengalami distorsi, Tuan Krabs harus terus mengukur ukuran dari labirin 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.



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 :

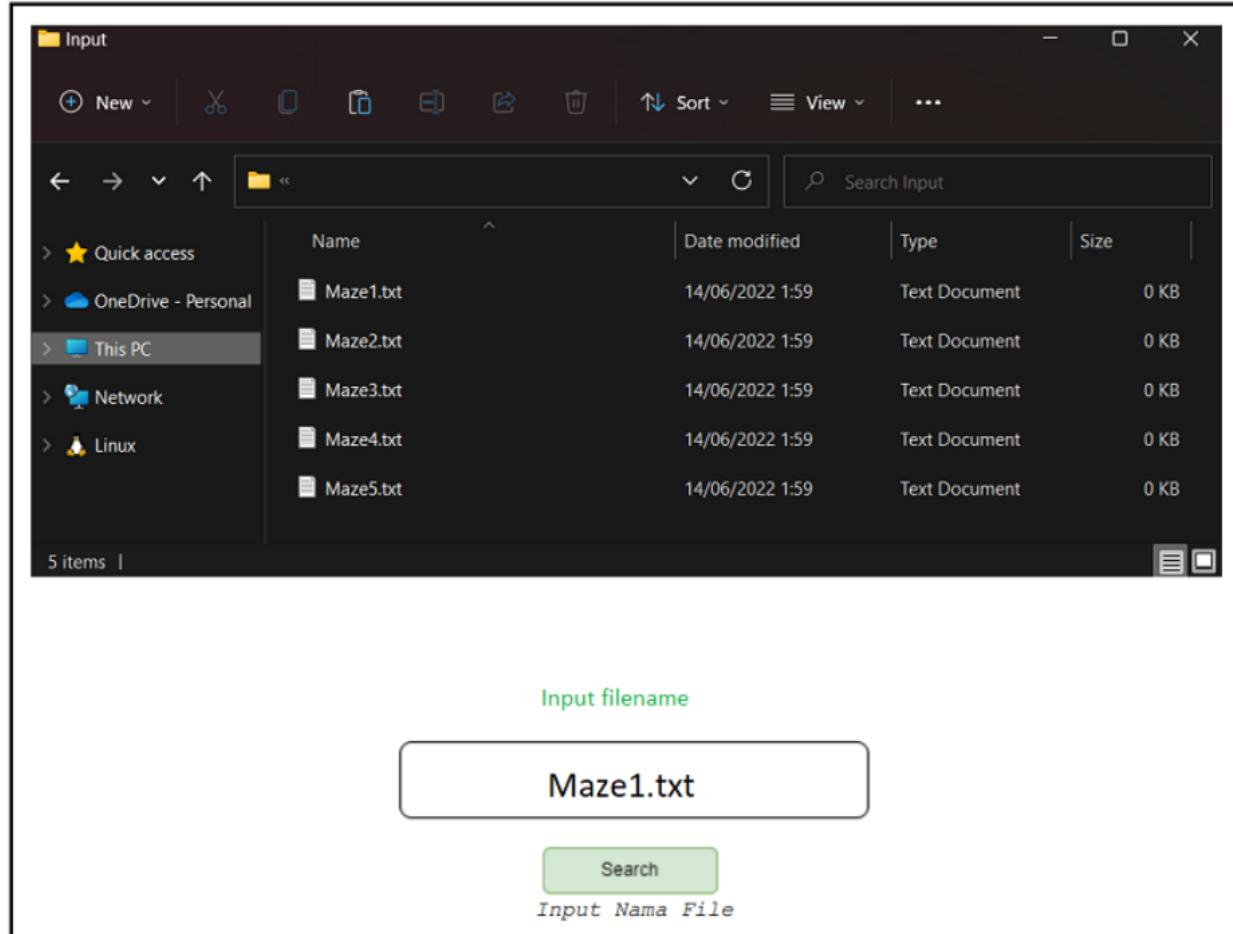
- K : Krusty Krab (Titik awal)
- T : Treasure
- R : Grid yang mungkin diakses / sebuah lintasan
- X : Grid halangan yang tidak dapat diakses

Contoh file input :



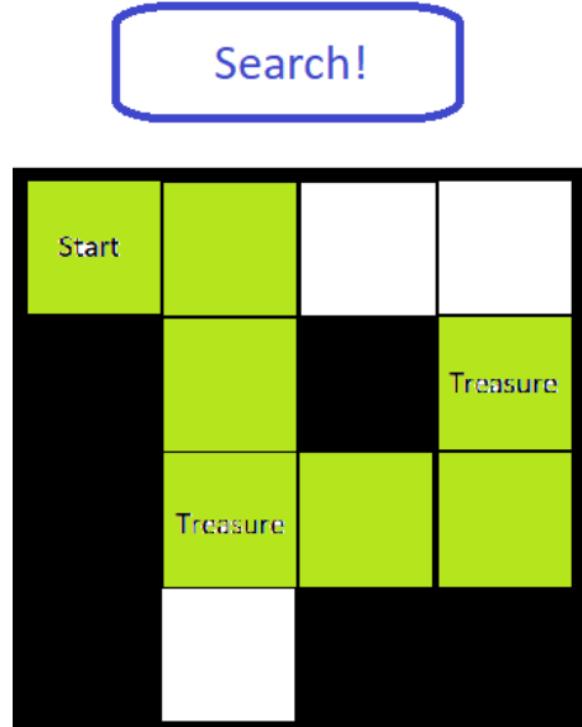
Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), anda dapat menelusuri grid (simpul) yang mungkin dikunjungi hingga ditemukan rute solusi, baik secara melebar ataupun mendalam bergantung alternatif algoritma yang dipilih. Rute solusi adalah rute yang memperoleh seluruh 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 masing - masing kelompok, asalkan dijelaskan di readme / laporan.

Contoh input aplikasi :



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

Contoh output Aplikasi :



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.

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

Treasure Hunt Solver

Input

Filename

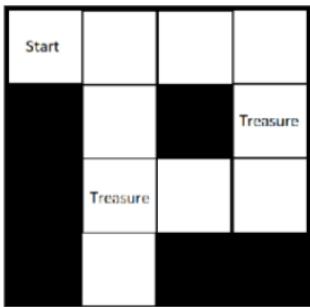
e.g "maze1.txt"

Algoritma

- BFS
- DFS

Visualize

Output



Search !

Route:

Nodes :

Steps:

Execution Time :

Treasure Hunt Solver

Input

Filename

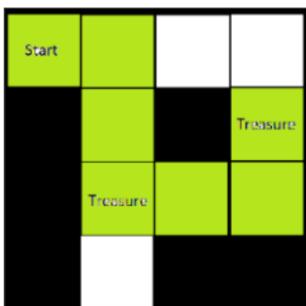
e.g "maze1.txt"

Algoritma

- BFS
- DFS

Visualize

Output



Search !

Route: R - D - D - R - R - U

Nodes : 11

Steps: 6

Execution Time : 850 ms

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

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 banyaknya 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 tiap 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.

Program yang dibuat harus memenuhi spesifikasi wajib sebagai berikut:

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. Apabila filename tersebut ada, Program akan melakukan validasi dari file input tersebut.
3. 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.

BAB 2: Landasan Teori

2.1 Dasar teori

Traversal graf atau algoritma pengunjungan pada graf adalah metode sistematis untuk mengunjungi simpul-simpul. Terdapat dua algoritma traversal pada graf, yaitu Breadth-First Search (BFS) dan Depth-First Search (DFS).

Breadth-First Search atau BFS adalah salah satu algoritma sederhana untuk melakukan pencarian pada graf dan merupakan pola dasar pada algoritma pencarian graf lainnya seperti algoritma minimum-spanning-tree Prim dan algoritma pencarian jarak terdekat Dijkstra yang menggunakan ide serupa dengan BFS. Dalam BFS, graf direpresentasikan sebagai $G=(V,E)$ dengan V sebagai simpul awal penelusuran. Secara sederhana, algoritma BFS memulai penelusuran dari simpul V dan menelusuri semua simpul yang bertetangga dengan V terlebih dahulu, kemudian menelusuri simpul-simpul lain yang belum dikunjungi yang bertetangga dengan simpul-simpul yang telah dikunjungi. Proses ini terus dilakukan hingga simpul yang dicari ditemukan atau seluruh simpul telah ditelusuri.

Algoritma depth-first search berbeda dengan breadth-first search karena melakukan penelusuran ke "dalam" terlebih dahulu jika memungkinkan. Algoritma ini menelusuri simpul tetangga dari simpul yang sedang ditelusuri saat ini dan melakukan backtracking untuk menelusuri simpul-simpul tetangga dari simpul sebelumnya jika sudah tidak ada simpul tetangga lagi yang tersedia.

Secara sederhana, algoritma depth-first search dimulai dari simpul V , kemudian dilanjutkan penelusuran ke simpul W yang bertetangga dengan simpul V . Selanjutnya, algoritma DFS akan dipanggil untuk menelusuri simpul W . Ketika mencapai simpul U dan semua simpul tetangga telah dikunjungi, algoritma akan melakukan pencarian runut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya, yaitu simpul yang melakukan pemanggilan algoritma DFS. Proses ini terus dilakukan sampai tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

2.2 C# desktop application development

C# merupakan salah satu bahasa pemrograman berorientasi objek yang banyak digunakan untuk pengembangan aplikasi *desktop*, pengembangan layanan berbasis *web*, serta pengembangan *game*. Pada tugas besar ini, digunakan bahasa C# serta kakas .NET core versi 7.0. Jenis *project* yang digunakan adalah WinFormsApp. Jenis *project* ini dapat memudahkan proses pengembangan aplikasi *desktop* berbasis GUI karena menyediakan kelas Form untuk tampilan GUI. Dengan menggunakan IDE Visual Studio, proses pembuatan dan modifikasi Form

sangat praktis karena dapat dilakukan dengan *drag-and drop* komponen UI yang diinginkan serta mengubah properti komponen tersebut pada kolom *properties* yang disediakan pada IDE. Setelah membuat tampilan GUI, diperlukan *event handler* untuk merespons *event* yang terjadi pada GUI, misalnya ketika pengguna menekan suatu tombol atau memasukan suatu teks ke dalam *textbox*.

Berikut adalah langkah-langkah untuk membuat WinFormsApp dengan memanfaatkan IDE Visual Studio.

1. Jalankan aplikasi Visual Studio
2. Pada halaman utama, pilih ‘Create a new project’, kemudian pilih Windows Forms App
3. Tuliskan nama dan lokasi penyimpanan *project*
4. Pilih versi kakas .NET core yang akan digunakan
5. Klik tombol ‘Create’
6. IDE akan membuat *project* yang berisi folder bin, obj, Form1.cs, Program.cs, dan beberapa *file* lainnya
7. Untuk mengedit Form, klik Form1.cs pada IDE.
8. Untuk menambahkan kelas, klik kanan pada nama *project* di kolom ‘Solution explorer’, lalu pilih ‘Add’ dan klik ‘Class’.
9. Untuk menambahkan form, klik kanan pada nama *project* di kolom ‘Solution explorer’, lalu pilih ‘Add’ dan klik ‘Form’
10. Untuk melakukan *build* dan *debugging*, klik tombol Run berbentuk segitiga hijau pada IDE. IDE akan membuat *executable file* pada folder bin sekaligus menjalankan *executable file* tersebut.

Pada tugas ini, dibutuhkan konsep pemrograman berorientasi objek untuk membagi tanggung jawab kepada objek-objek. Secara umum, tanggung jawab UI diberikan pada objek dari kelas FileInputGUI dan VisualizeMapGUI, yang merupakan turunan dari kelas Form. Sedangkan tanggung jawab algoritma diberikan pada objek dari kelas TreasureHunt. Dibutuhkan *message passing* antar objek, karena objek dari kelas FileInputGUI akan menerima nama file dan pilihan algoritma dari pengguna, kemudian nama file dan pilihan tersebut diberikan kepada objek dari kelas TreasureHunt untuk melakukan pencarian dengan algoritma BFS atau DFS. Setelah pencarian selesai, hasil pencarian akan diberikan kepada objek dari kelas VisualizeMapGUI untuk dilakukan visualisasi hasil pencarian yang akan ditampilkan kepada pengguna. Terdapat pula kelas-kelas lainnya yang akan dijelaskan lebih lanjut.

BAB 3: Aplikasi Algoritma BFS dan DFS dalam Algoritma Penyelesaian Persoalan Maze Treasure Hunt

3.1 Langkah-langkah pemecahan masalah

Untuk memulai proses mendesain solusi dari permasalahan Treasure-Hunt, langkah pertama yang harus dilakukan adalah membagi permasalahan utama menjadi beberapa permasalahan kecil. Dalam hal ini, permasalahan utama dapat dibagi menjadi dua permasalahan utama, yaitu pencarian *treasure* menggunakan algoritma BFS dan algoritma DFS. Selain itu, ada juga permasalahan-permasalahan tambahan yang perlu diselesaikan, yaitu bagaimana solusi dari dua permasalahan utama dapat ditampilkan dalam bentuk *desktop application* dan versi sederhana *traveling salesman problem* (TSP) yang hanya membutuhkan rute penelusuran seluruh kotak *Treasure* dari titik awal dan kembali ke titik awal. Untuk menampilkan solusi dari pencarian menggunakan algoritma BFS dan DFS, kami memanfaatkan bantuan platform WinForms.

Langkah berikutnya setelah membagi permasalahan menjadi dua, yaitu menggunakan algoritma *breadth-first search* dan algoritma *depth-first search* untuk pencarian, adalah melakukan pemetaan permasalahan ke dalam elemen-elemen algoritma tersebut. Setelah itu, dilakukan perancangan algoritma sesuai dengan elemen yang telah dipetakan. Setiap simpul berisi kotak yang telah dikunjungi.

3.2 Proses *mapping* persoalan menjadi elemen-elemen algoritma BFS dan DFS

Algoritma pencarian search pada persoalan ini menggunakan konsep pencarian solusi dari **pohon dinamis** berupa rute menuju kotak (*grid*) *Treasure* dari kotak awal pencarian. Elemen urutan pencarian berisi kotak yang dikunjungi, dan jalur graf dari akar menuju daun solusi (posisi grid dengan *Treasure*) adalah solusi dari pencarian.

Operator	{up, right, down, left}
Akar	Posisi awal (<i>grid</i> dengan simbol ‘K’)
Simpul	<i>Grid</i> yang dilalui
Daun	<i>Grid Treasure</i> (<i>grid</i> dengan simbol ‘T’)
Solution space	Rute dari akar pohon pencarian menuju simpul daun solusi

3.3 Contoh ilustrasi kasus lain yang berbeda dengan contoh pada spesifikasi tugas

Selain dari metode *breadth-first search* dan *depth-first search* dalam pencarian graf, terdapat metode lain yang merupakan gabungan dari kedua metode tersebut, yaitu *iterative deepening search* atau IDS. IDS menggabungkan efisiensi penyimpanan dari *depth-first search* dan kemampuan pencarian cepat pada simpul yang dekat dengan akar dari *breadth-first search*. IDS menggunakan algoritma *depth-limited search* atau DLS, yang merupakan *depth-first search* dengan pembatasan kedalaman.

BAB 4: Aplikasi Algoritma Penyelesaian Persoalan Maze Treasure Hunt dalam Program yang Dibuat

4.1 Implementasi program (*pseudocode*)

```
/* Program Utama */

Inisialisasi peta
solutionPath <- empty

while peta.treasureCount() > 0
    /* Search(peta) menggunakan algoritma yang dipilih, BFS atau DFS */
    solutionPath.Add(Search(peta))
endwhile

show(solutionPath)
```

```
/* Algoritma BFS */

solutionTree(root) <- startingPosition
searchQueue.Enqueue(startingPosition)

currentPosition <- startingPosition

while notTreasure(currentPosition) do
    currentPosition <- searchQueue.Dequeue()

    for(direction in {up, right, down, left})
        newPosition <- currentPosition + direction
        if positionValid(currentPosition + direction)
            then
                if(not isVisited(newPosition))
                    then
                        isVisited(newPosition) = true
                        solutionTree(currentPosition).AddChild(newPosition)
                        searchQueue.Enqueue(newPosition)
```

```

        endif
    endif

endwhile
finalPosition <- currentPosition
-> solutionTree.getPathTo(finalPosition)

```

```

/* Algoritma DFS */

solutionTree(root) <- startingPosition
searchStack.Push(startingPosition)

currentPosition <- startingPosition

while notTreasure(currentPosition) do

    currentPosition <- searchQueue.Pop()

    if isVisited(currentPosition)
    then continue
    endif

    isVisited(newPosition) = true
    solutionTree(currentPosition).AddChild(newPosition)

    for(direction in {up, right, down, left})
        newPosition <- currentPosition + direction
        if positionValid(currentPosition + direction)
        then
            if(not isVisited(newPosition))
            then
                searchStack.Push(newPosition)
            endif
        endif

endwhile
finalPosition <- currentPosition
-> solutionTree.getPathTo(finalPosition)

```

4.2 Penjelasan struktur data yang digunakan dalam program dan spesifikasi program

- Position

Kelas Position digunakan untuk menyimpan informasi posisi grid peta. Penggunaan kelas Position dapat digantikan dengan Pair, namun tidak digunakan karena digunakan method-method yang lebih mengikuti batasan-batasan penelusuran grid (sebagai kelas *helper*, namun menghindari penggunaan objek tambahan).

```
class Position
```

```
{
    public readonly int row;
    public readonly int col;

    // method tambahan berupa setter
    // dan method pembuat posisi ke arah atas, kanan, bawah, dan kiri
}
```

- Peta

Kelas Peta digunakan untuk menyimpan dan mengolah informasi terkait peta harta karun. Kelas Peta memiliki tanggung jawab dalam memeriksa valid atau tidaknya grid yang akan ditelusuri (di luar batas atau merupakan grid yang tidak dapat diakses (grid dengan simbol ‘X’)) dan pengecekan apakah grid memiliki simbol harta karun.

```
class Peta
{
    private char[,] peta; // menyimpan elemen-elemen peta
    public readonly int nRow; // jumlah baris grid peta
    public readonly int nCol; // jumlah kolom grid peta
    public int nTreasure { get; private set; }
    // jumlah harta karun pada peta pada saat ini
    public Position startPos { get; private set; }
    // posisi start pada peta saat ini

    // method-method terkait pengecekan grid
    // dan method-method terkait modifikasi elemen pada peta
}
```

- PetaVisit (isVisited)

Kelas PetaVisit digunakan untuk menyimpan dan mengolah informasi terkait grid peta yang telah dikunjungi. Kelas PetaVisit memiliki tanggung jawab dalam memeriksa apakah suatu grid telah ditelusuri sebelumnya. Data ini dipisah dari Peta karena tidak selalu digunakan untuk penelusuran bersama-sama dengan peta.

```
class PetaVisit
{
    private bool[,] peta;
    private int nRow;
    private int nCol;
```

```

    // method-method terkait pengecekan grid
    // dan method-method terkait penandaan grid
}

```

- Node

Kelas Node digunakan untuk menyimpan rute pencarian. Penggunaan struktur data Node didasari kebutuhan penyimpanan rute (urutan posisi) yang merupakan salinan dari rute sebelumnya ditambahkan dengan penelusuran saat ini. Alokasi memori dan penyalinan rute lama akan tidak efisien jika terdapat banyak percabangan, sehingga solusinya adalah menggunakan struktur data pohon. Selain itu, menggunakan node memungkinkan membangun pohon dengan menambahkan node baru cukup dengan informasi node sebelumnya, tanpa harus melakukan pencarian sebuah simpul untuk menambah simpul baru seperti model pohon pada umumnya.

```

class Node
{
    public readonly Position position; // menyimpan posisi penelusuran
    public readonly char choice; // menyimpan pilihan penelusuran
    public readonly Node? parent; // menyimpan alamat node sebelumnya
    /* linked-list sebagai akibat dari node akan membentuk pohon
    pencarian dengan node-node anak menunjuk kepada node induk */

    // method-method terkait pengecekan grid
    // dan method-method terkait penandaan grid
}

```

- Struktur data BFS

Dalam menjalankan pencarian dengan BFS, digunakan variabel-variabel lokal berupa

```

Peta searchMap // peta yang digunakan dalam menelusuri grid
PetaVisit isVisited // menyimpan informasi grid telah dikunjungi
Queue<Node> memo // menyimpan urutan grid yang akan ditelusuri secara BFS

int searchCount // menyimpan jumlah grid yang ditelusuri
List<Position> positionSequence // menyimpan urutan penelusuran grid
// hanya akan digunakan jika mode tampilkan penelusuran dipilih

Node currNode; // Node saat ini (sebagai parent dari rute saat ini)
Position currPos; // posisi penelusuran saat ini

```

```
(List<Node>, int) // sebagai return value (message) untuk ditampilkan
```

- Struktur data DFS

Dalam menjalankan pencarian dengan DFS, digunakan variabel-variabel lokal berupa

```
Peta searchMap // peta yang digunakan dalam menelusuri grid
PetaVisit isVisited // menyimpan informasi grid telah dikunjungi
Stack<Node> memo // menyimpan urutan grid yang akan ditelusuri secara DFS

int searchCount // menyimpan jumlah grid yang ditelusuri
List<Position> positionSequence // menyimpan urutan penelusuran grid
// hanya akan digunakan jika mode tampilkan penelusuran dipilih

Node currNode; // Node saat ini (sebagai parent dari rute saat ini)
Position currPos; // posisi penelusuran saat ini

(List<Node>, int) // sebagai return value (message) untuk ditampilkan
```

4.3 Penjelasan tata cara penggunaan program

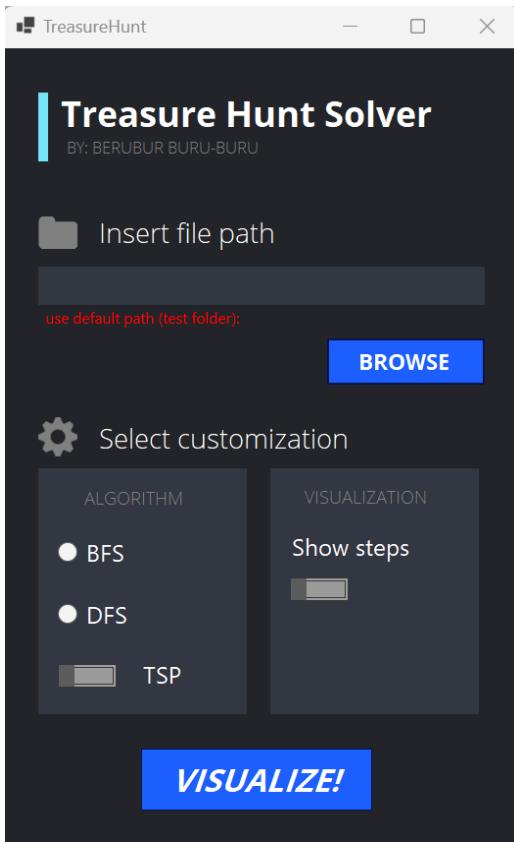
- Untuk menjalankan program, buka folder bin pada repositori. Kemudian, *double-click* pada TreasureHunt.exe. Untuk menjalankan program, dapat juga dilakukan dengan memasukkan *script* berikut pada terminal.

```
cd bin
./TreasureHunt
```

Jika OS pada PC tidak *support* ekstensi .exe, program juga dapat dijalankan dengan kakas .NET dengan cara memasukkan *script* berikut pada terminal. **Metode ini membutuhkan .NET core versi 7.0 yang harus sudah terinstall pada PC.**

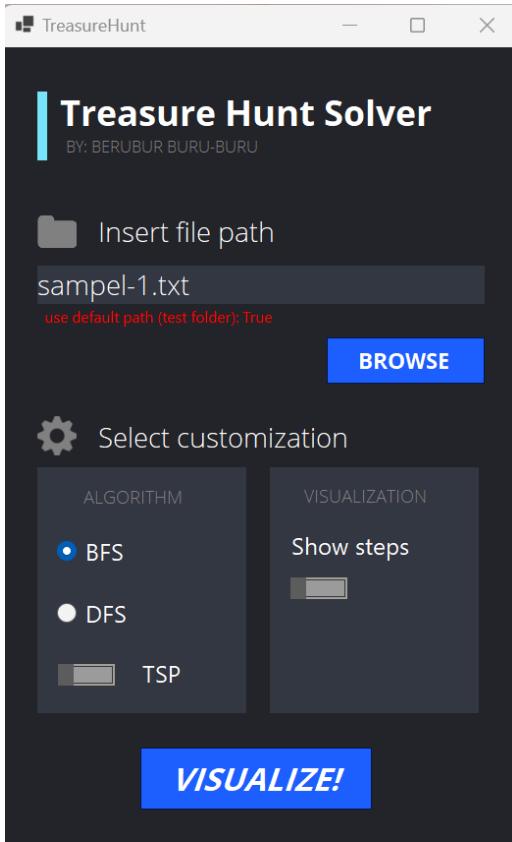
```
cd bin
dotnet TreasureHunt.dll
```

- Berikut adalah tampilan utama program

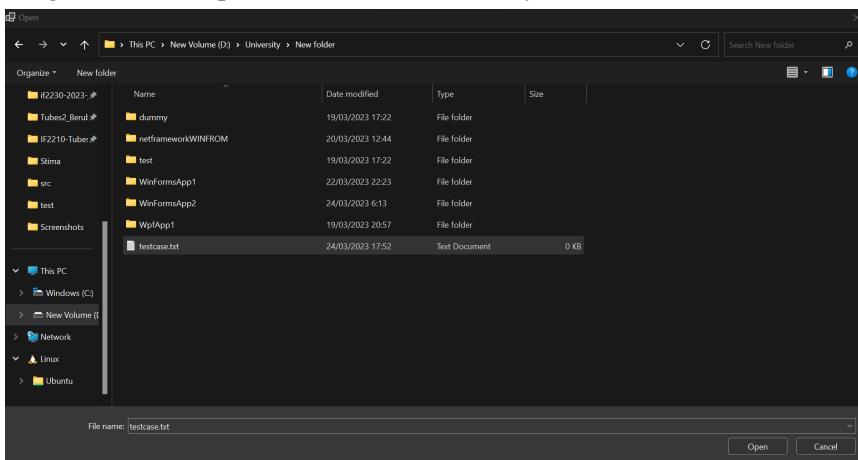


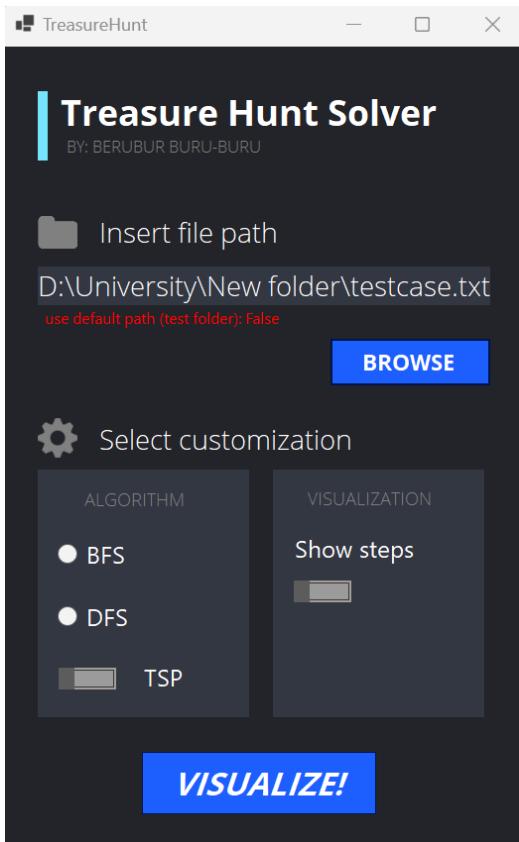
Komponen-komponen pada tampilan utama adalah sebagai berikut.

- *textbox* untuk memasukan nama file
 - *button* ‘BROWSE’ untuk mencari file yang ingin dimasukan
 - *radio button* ‘BFS’ dan ‘DFS’ untuk pilihan algoritma
 - *toggle* ‘TSP’ untuk mengaktifkan mode TSP,
 - *toggle* ‘Show steps’ untuk menampilkan proses pencarian sesuai urutan BFS dan DFS dengan waktu jeda pada setiap langkah.
-
- Masukan nama file ke dalam *textbox*. Jika Anda memasukkan nama *file* langsung dengan *keyboard*, maka nama *file* yang dimasukkan harus berada pada *folder* test.

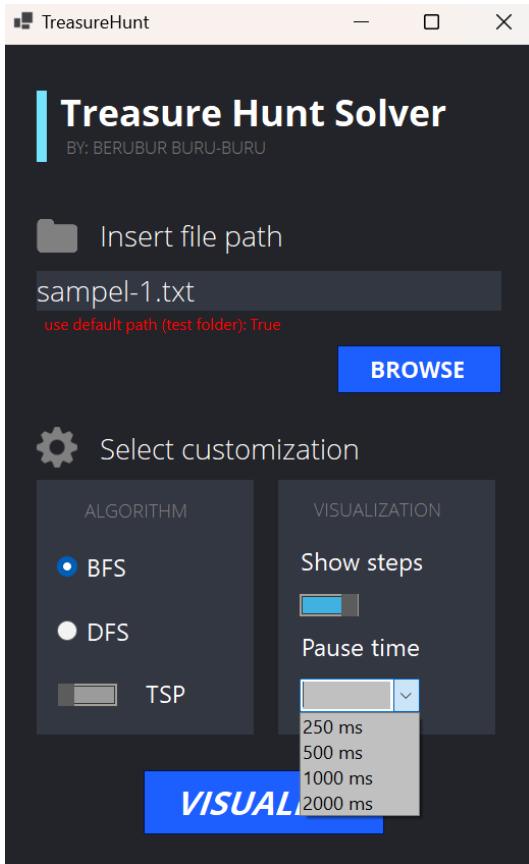


- Alternatif lain adalah dengan menekan button ‘BROWSE’ untuk mencari *file* yang diinginkan. Untuk opsi ini, *file* tidak harus berada dalam *folder* test. Setelah mendapatkan *file* yang diinginkan, klik ‘open’. Kemudian, direktori *file* secara otomatis akan tercantum pada *textbox*.

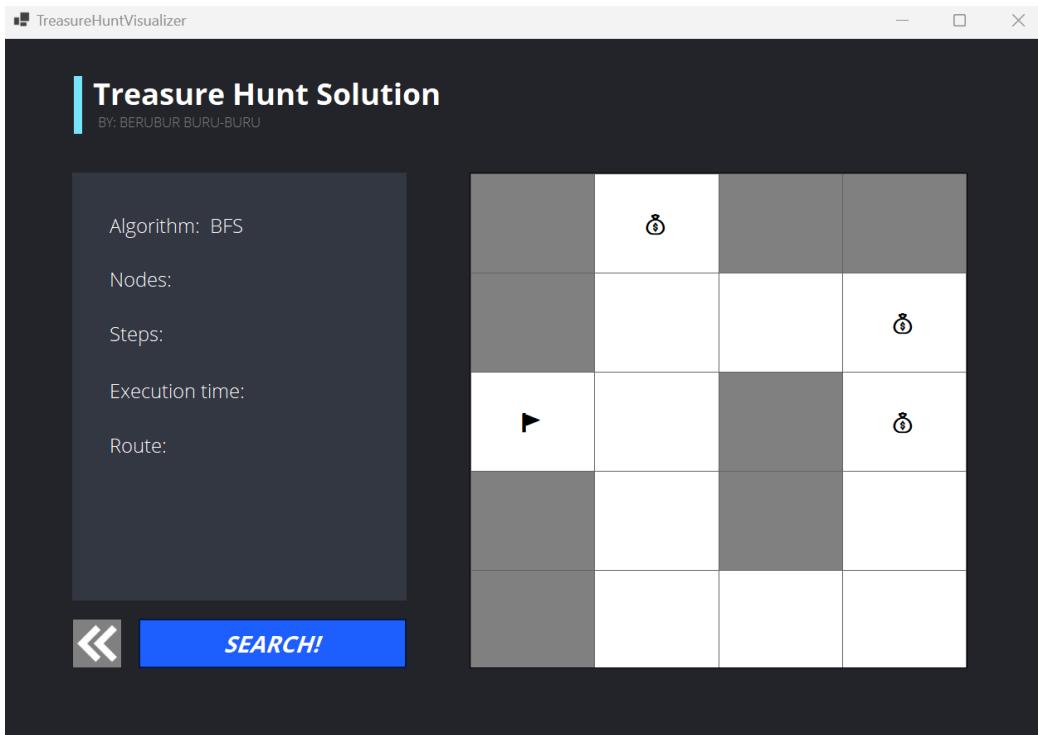




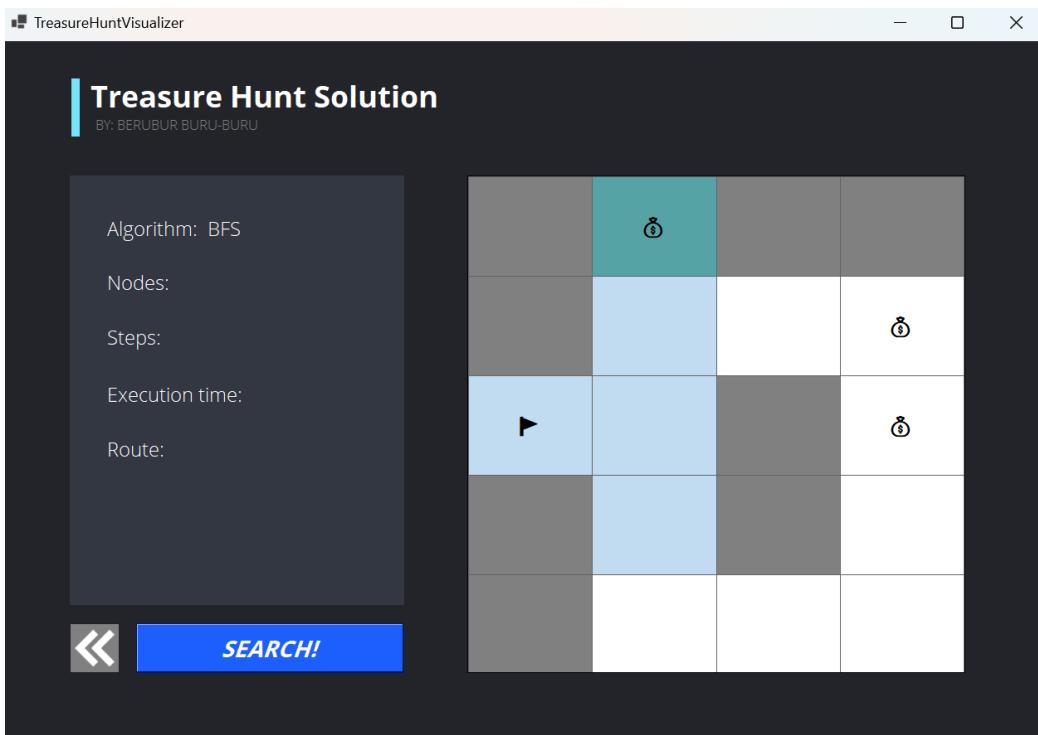
- Setelah memasukkan nama *file*, pilih algoritma yang diinginkan (BFS atau DFS). Jika ingin menghidupkan mode TSP, klik *toggle* ‘TSP’. Jika ingin menampilkan proses pencarian, klik *toggle* ‘Show steps’. Jika *toggle* tersebut diaktifkan, maka program akan menampilkan wakut jeda antarproses pencarian yang harus dipilih, yaitu 250 ms, 500 ms, 1000 ms, atau 2000 ms. Kemudian, klik *button* ‘VISUALIZE!’

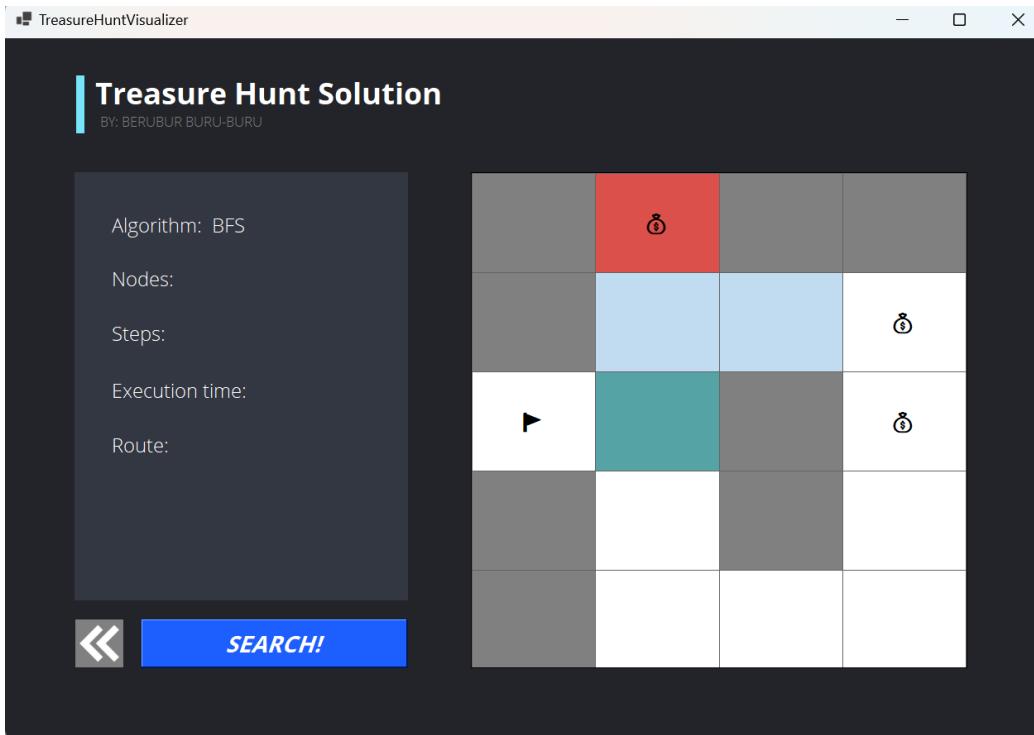


- Setelah itu, program akan menampilkan tampilan awal visualisasi Treasure Hunt. Komponen GUI visualisasi Treasure Hunt adalah sebagai berikut.
 - Tabel (*grid*) yang merepresentasikan peta. Kotak putih melambangkan jalan yang bisa ditempuh, sedangkan kotak abu-abu melambangkan dinding/penghalang. Gambar bendera pada tabel melambangkan posisi *start* (Krusty Krab), sedangkan gambar kantong uang melambangkan *treasure*.
 - *Information box* yang berisi informasi algoritma, *nodes*, *steps*, waktu eksekusi, dan rute solusi.
 - *Button* ‘*SEARCH!*’, untuk memulai pencarian.
 - *Button* ‘‘<<’’, untuk kembali ke tampilan sebelumnya.
- Klik *button* ‘*SEARCH!*’ untuk memulai pencarian.

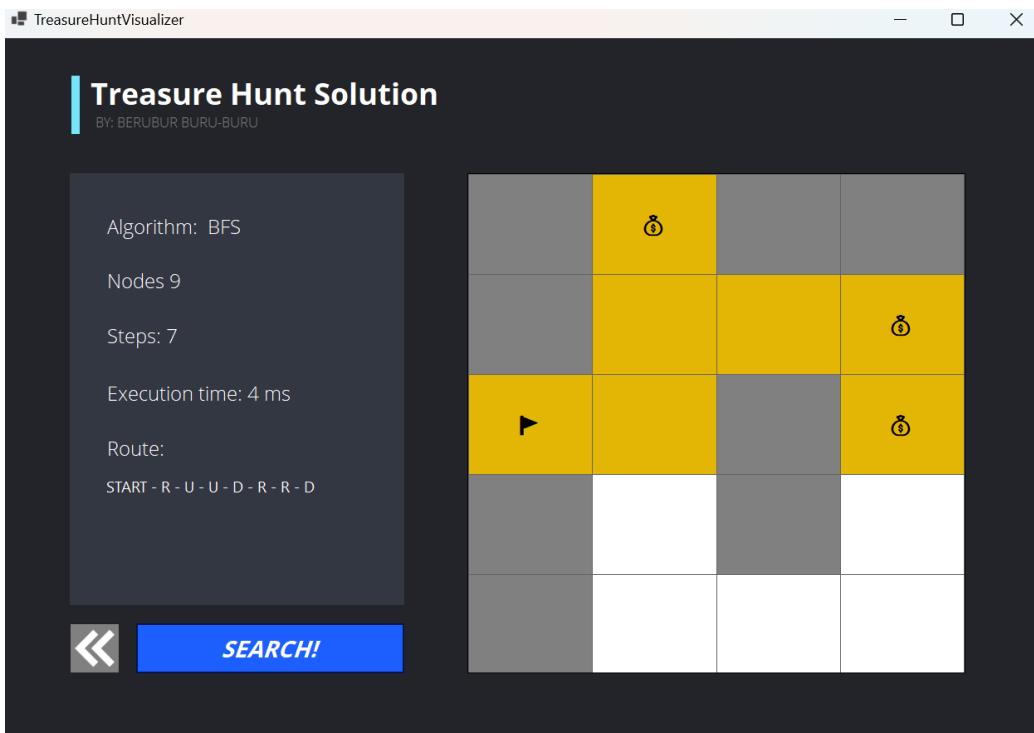


- Jika sebelumnya *toggle* ‘Show steps’ diaktifkan, maka program akan menampilkan langkah-langkah pencarian sesuai algoritma BFS atau DFS. Pada proses tersebut, warna tosca melambangkan jalan yang sedang dicek, warna biru muda melambangkan jalan yang sudah ditempuh, dan warna merah digunakan untuk menandai *treasure* yang sudah didapat.

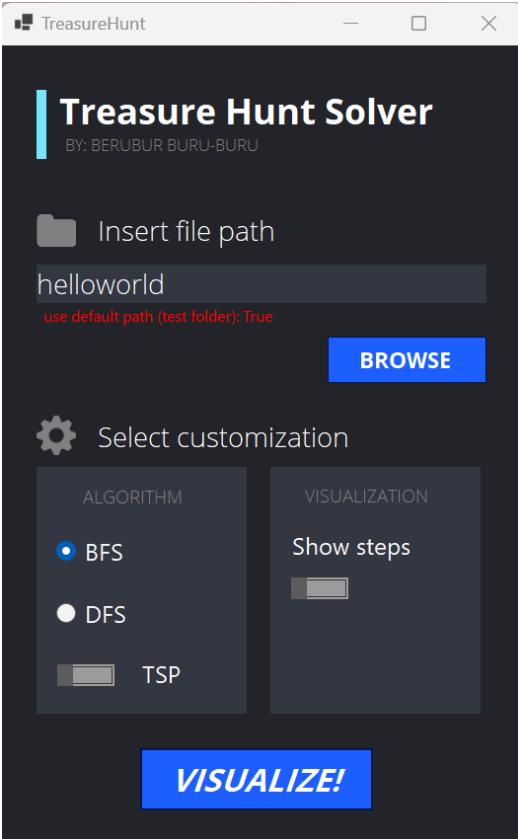


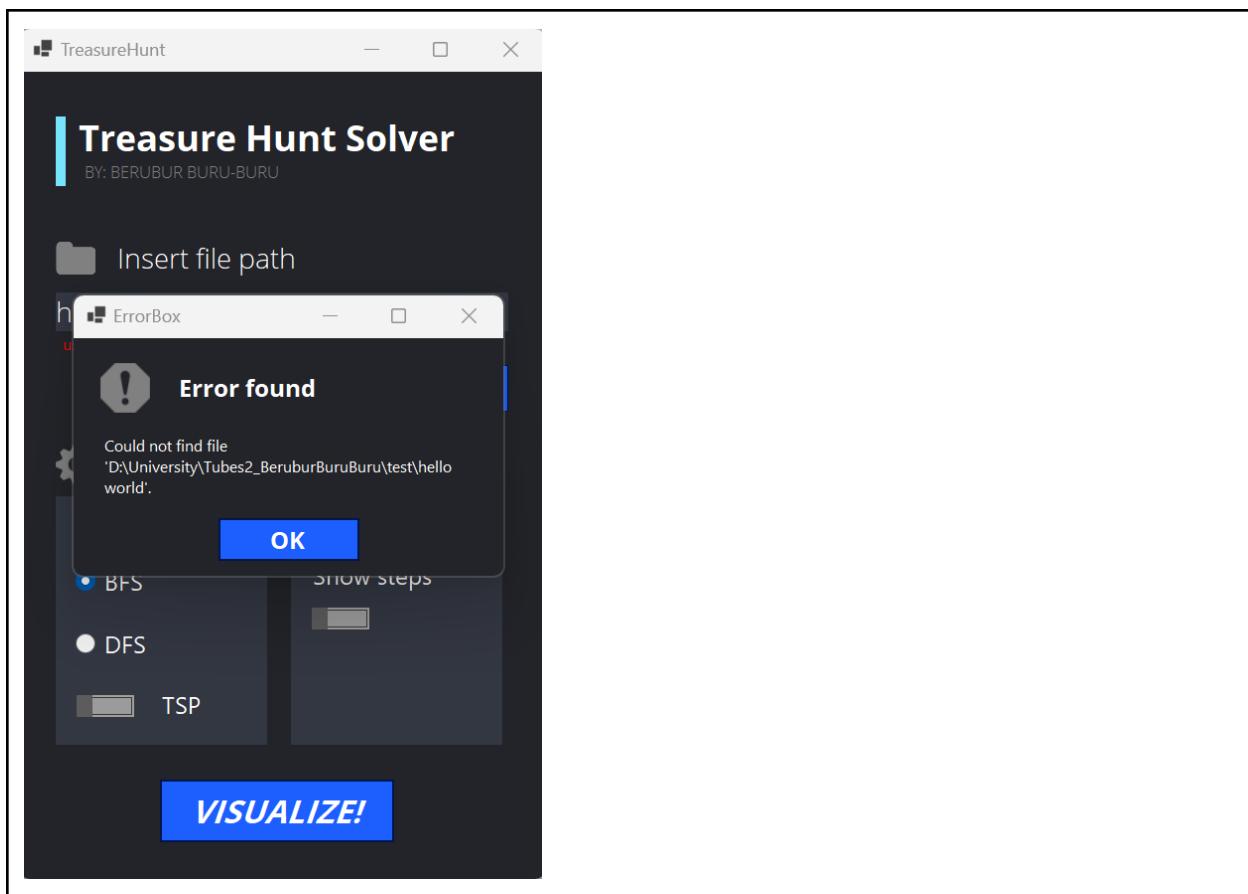


- Setelah pencarian selesai, rute solusi akan ditampilkan dengan warna kuning. Selain itu, informasi banyaknya *node* yang dibuat, banyaknya langkah menuju solusi, waktu eksekusi, serta tahap-tahap menuju rute solusi akan ditampilkan pada *information box*.

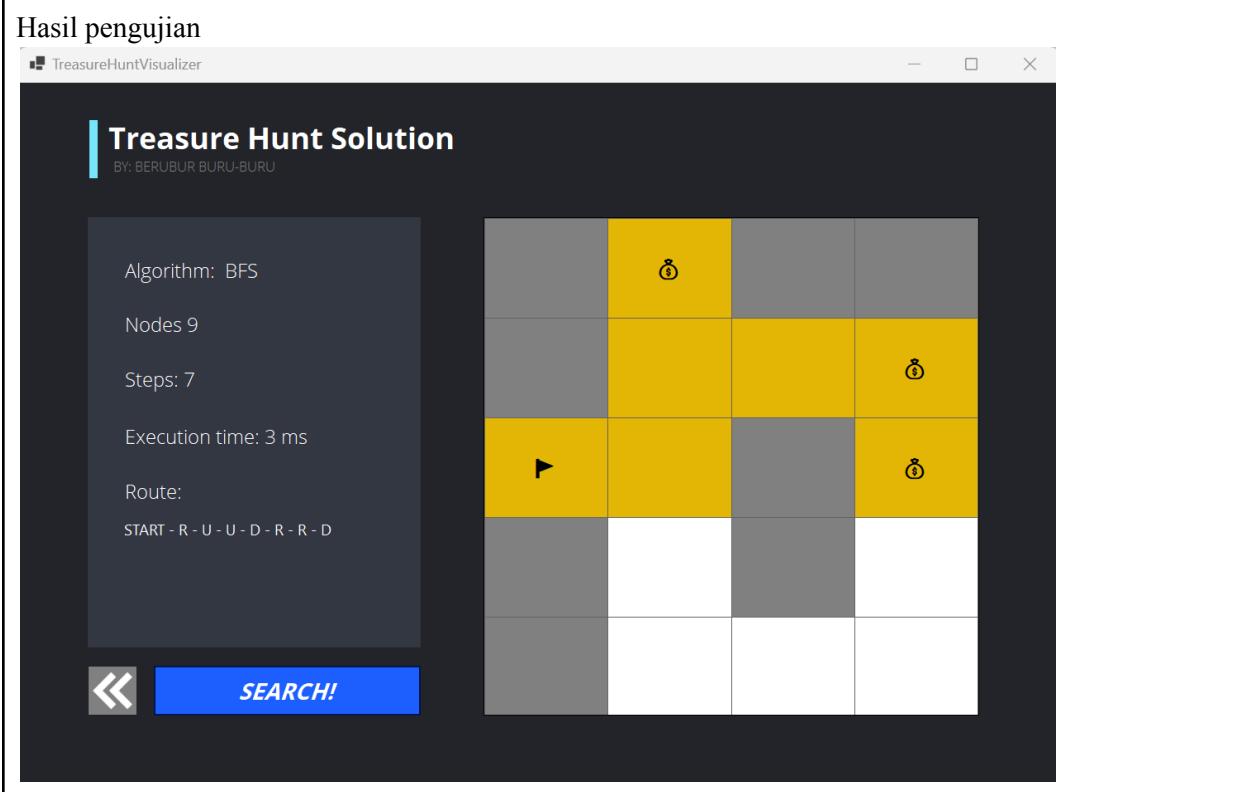
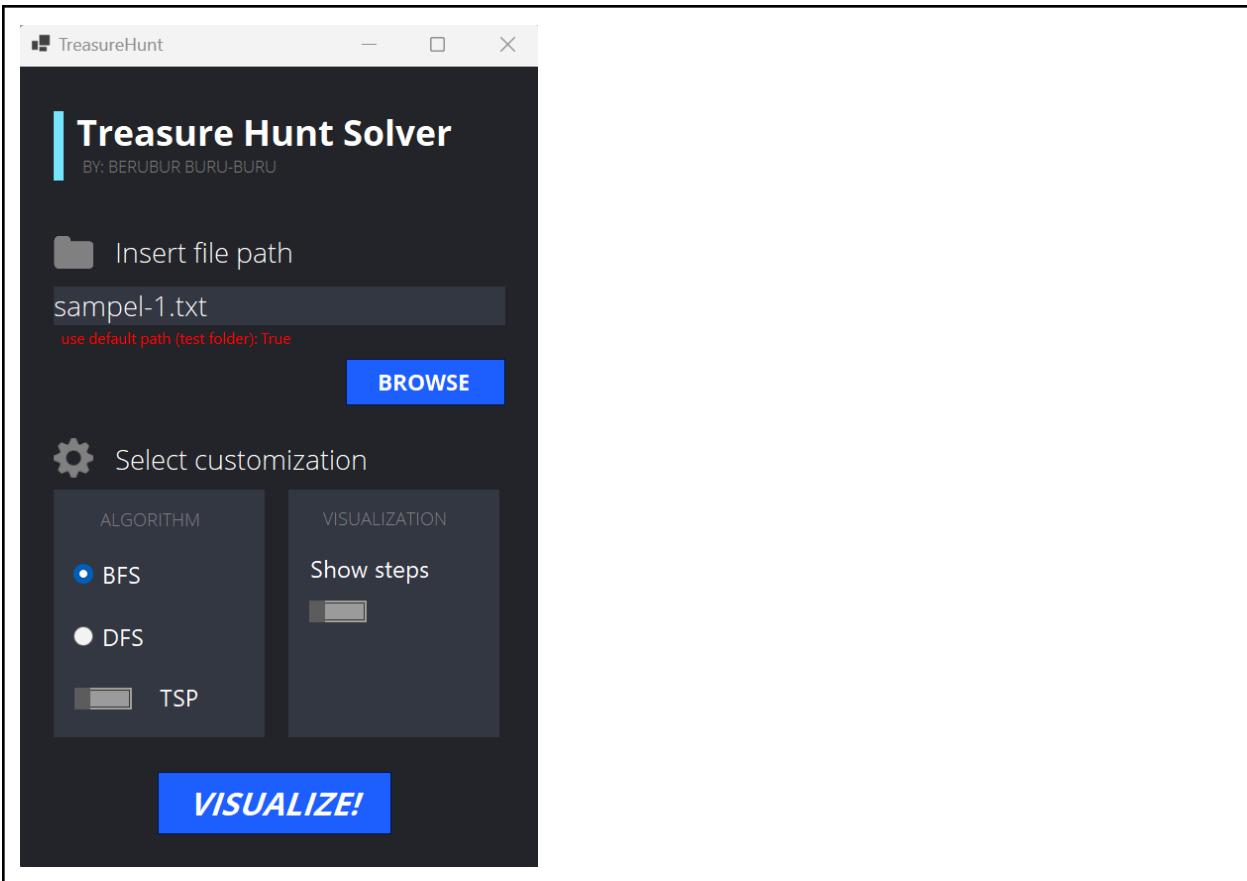


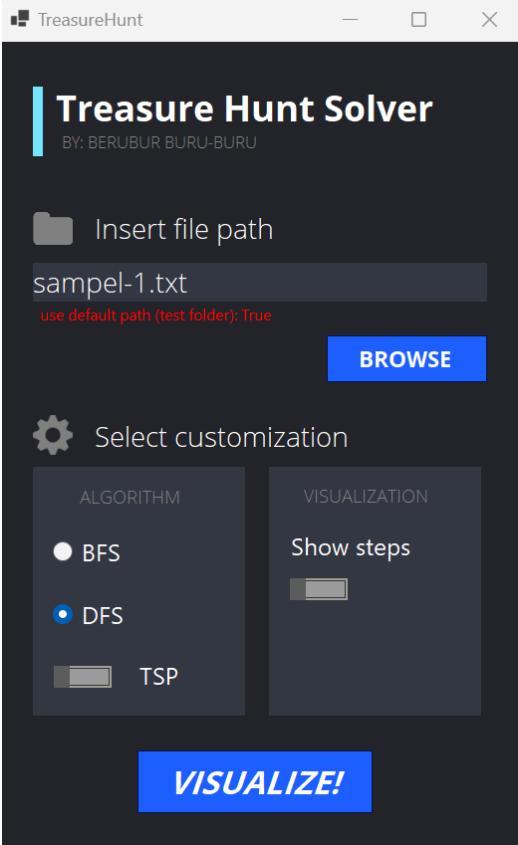
4.4 Hasil pengujian

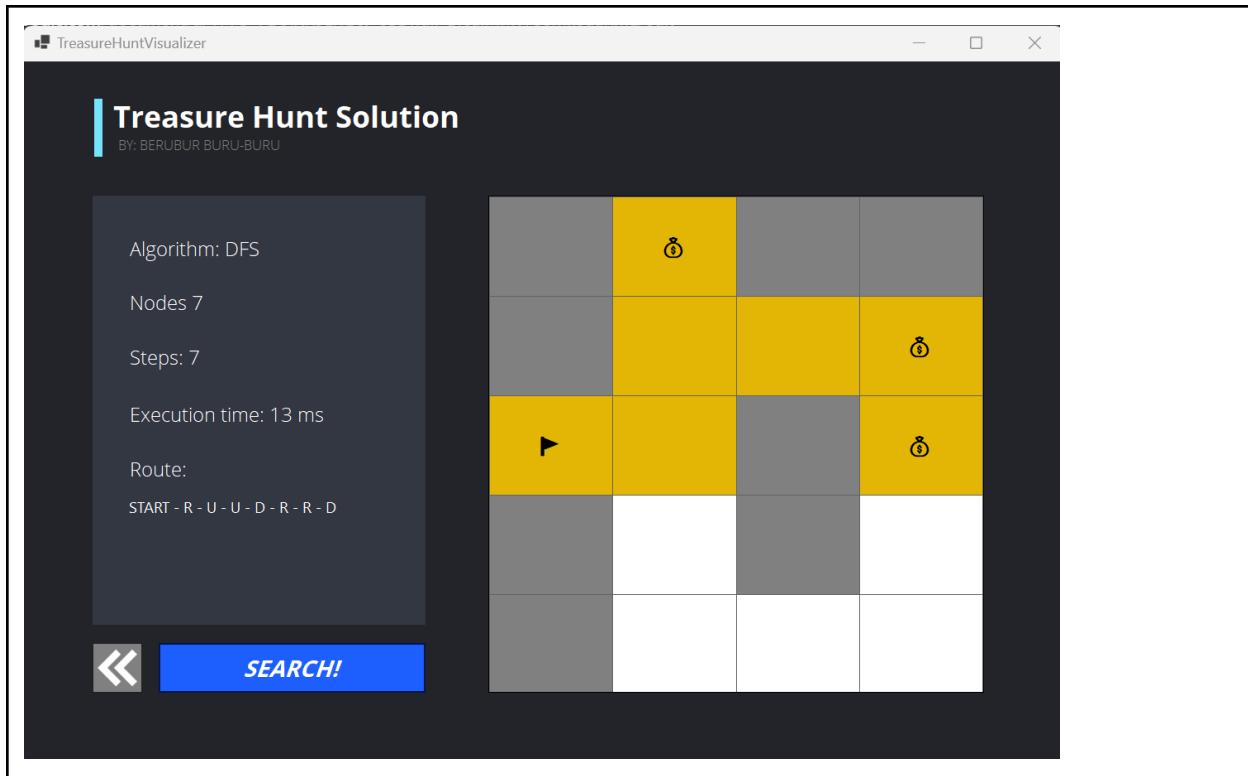
TESTING 1	
Skenario pengujian	Nama <i>file</i> yang dimasukkan tidak valid
Nama <i>file</i>	—
Isi <i>file</i>	—
Tampilan awal	
	
Hasil pengujian	



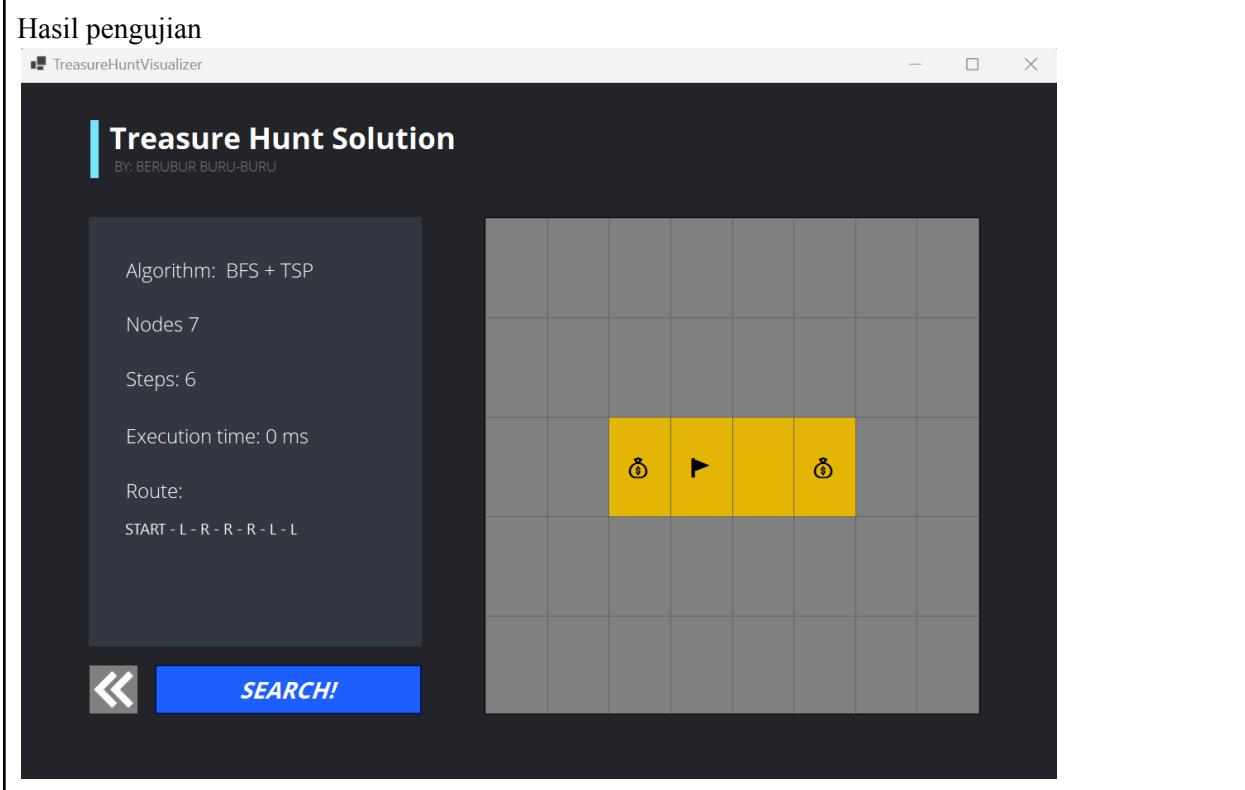
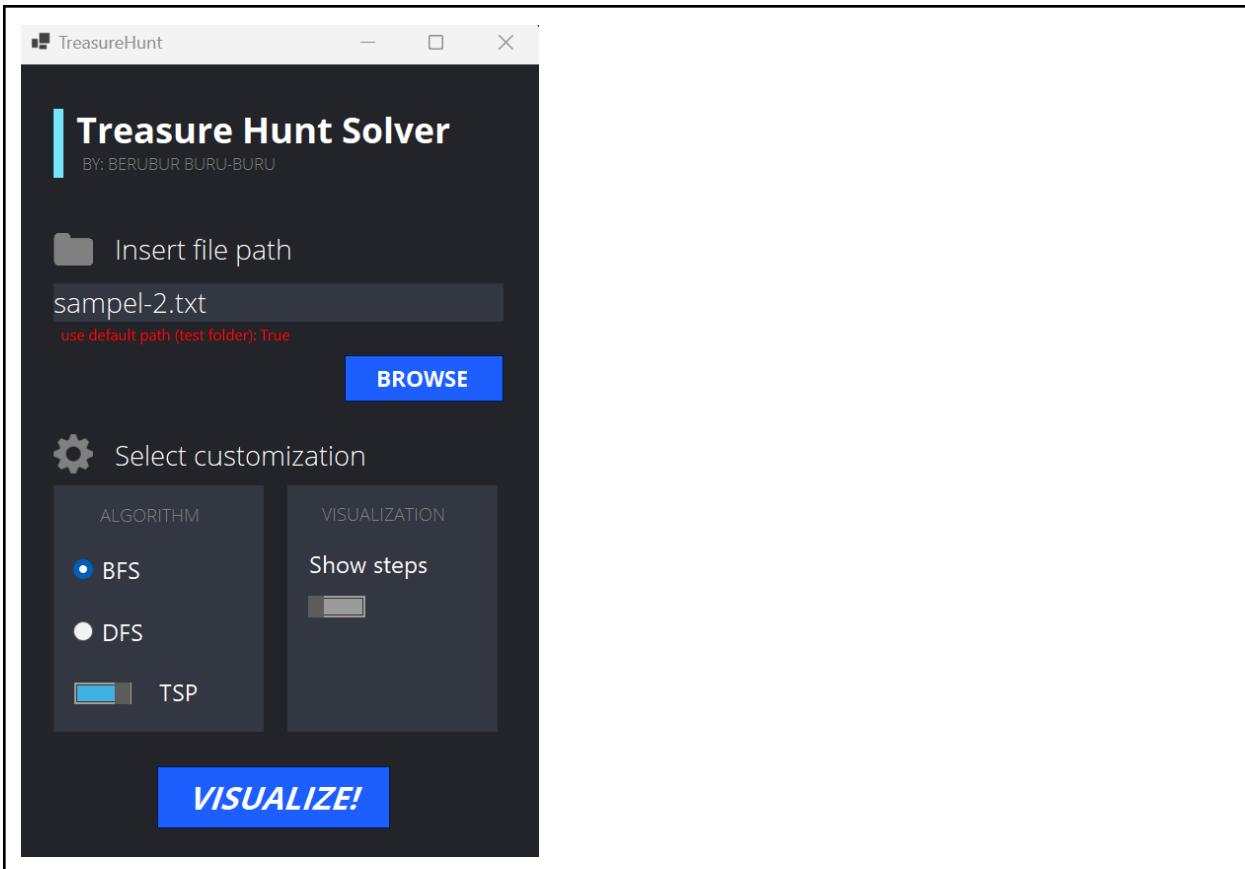
TESTING 2	
Skenario pengujian	Menggunakan algoritma BFS Tidak menggunakan mode TSP Tidak menampilkan progres pencarian Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-1.txt
Isi <i>file</i>	X T X X X R R T K R X T X R X R X R R R
Tampilan awal	



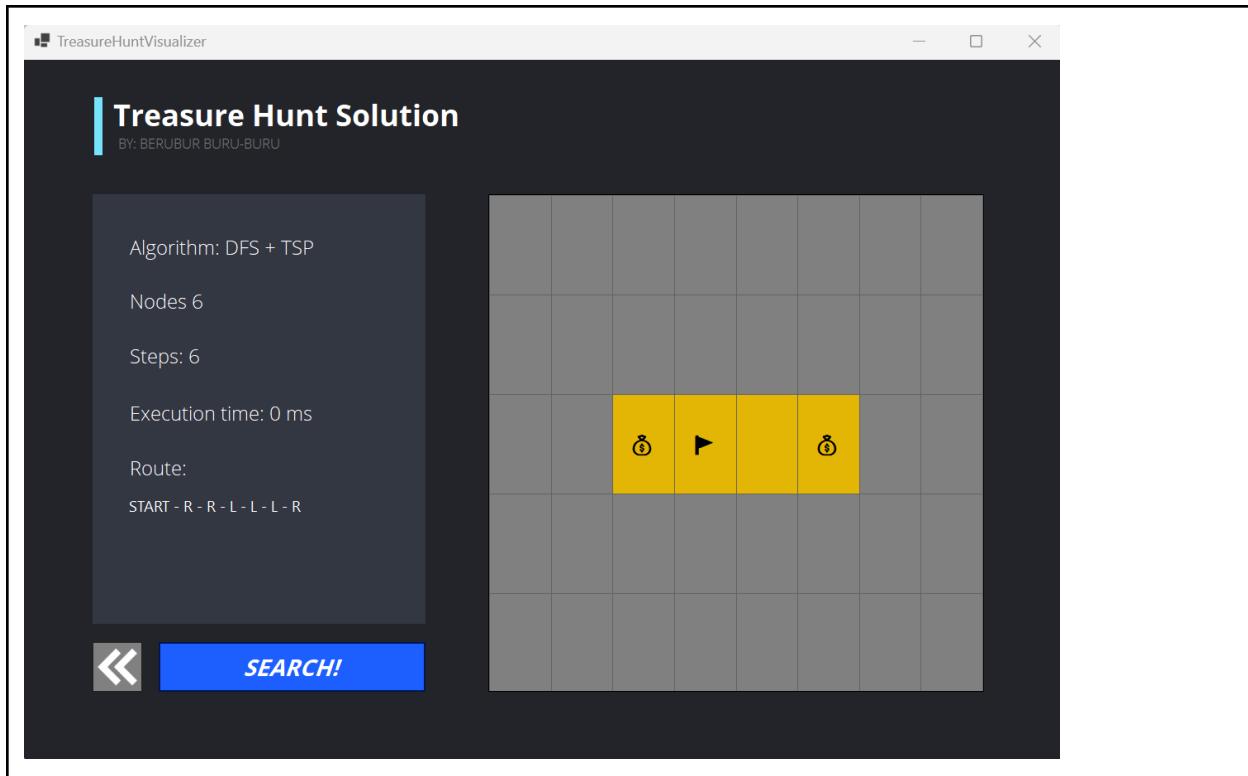
TESTING 3	
Skenario pengujian	Menggunakan algoritma DFS Tidak menggunakan mode TSP Tidak menampilkan progres pencarian Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-1.txt
Isi <i>file</i>	X T X X X R R T K R X T X R X R X R R R
Tampilan awal	
	
Hasil pengujian	



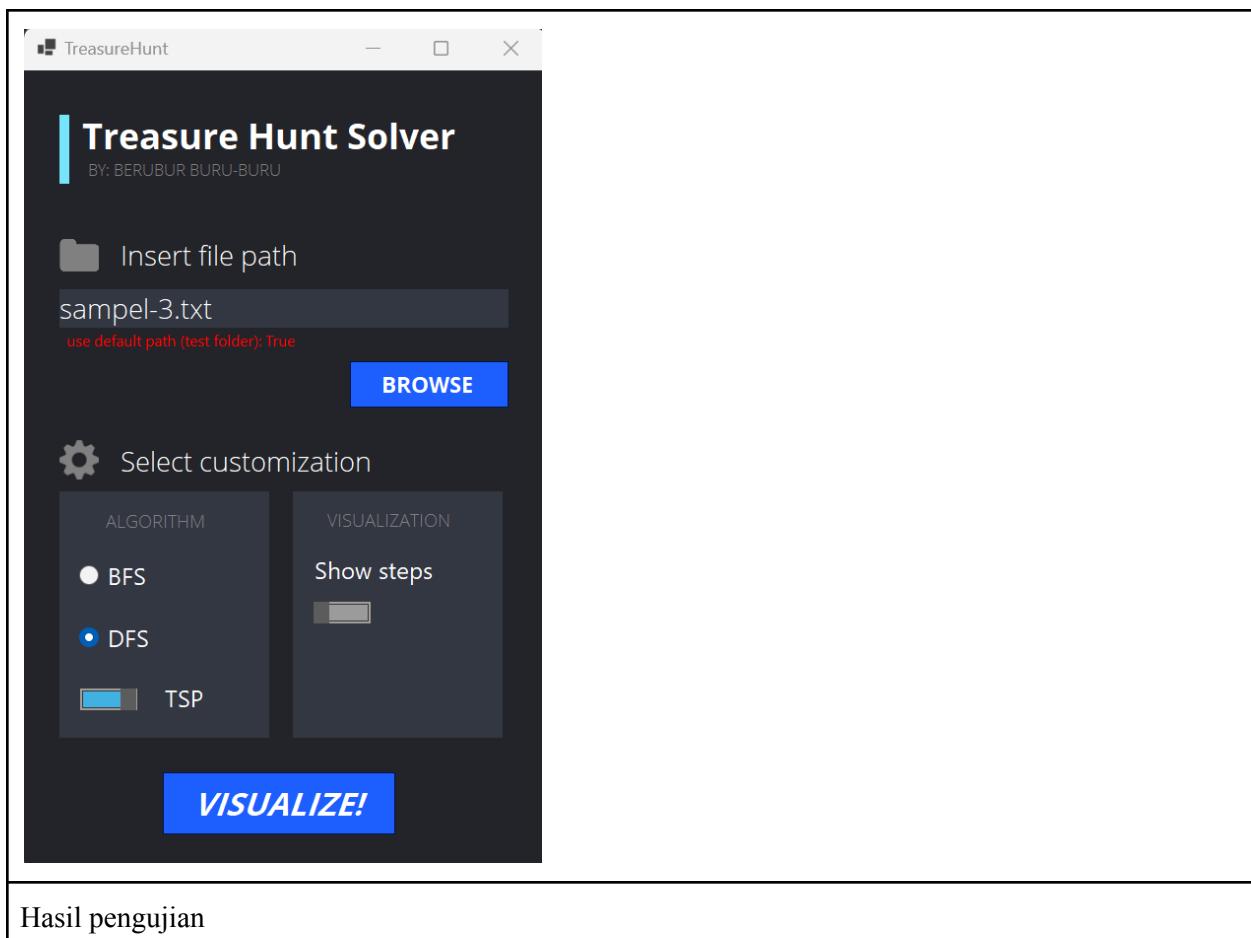
TESTING 4	
Skenario pengujian	Menggunakan algoritma BFS Menggunakan mode TSP Tidak menampilkan progres pencarian Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-2.txt
Isi <i>file</i>	X X X X X X X X X X X X X X X X X X T K R T X X X X X X X X X X X X X X X X X X
Tampilan awal	

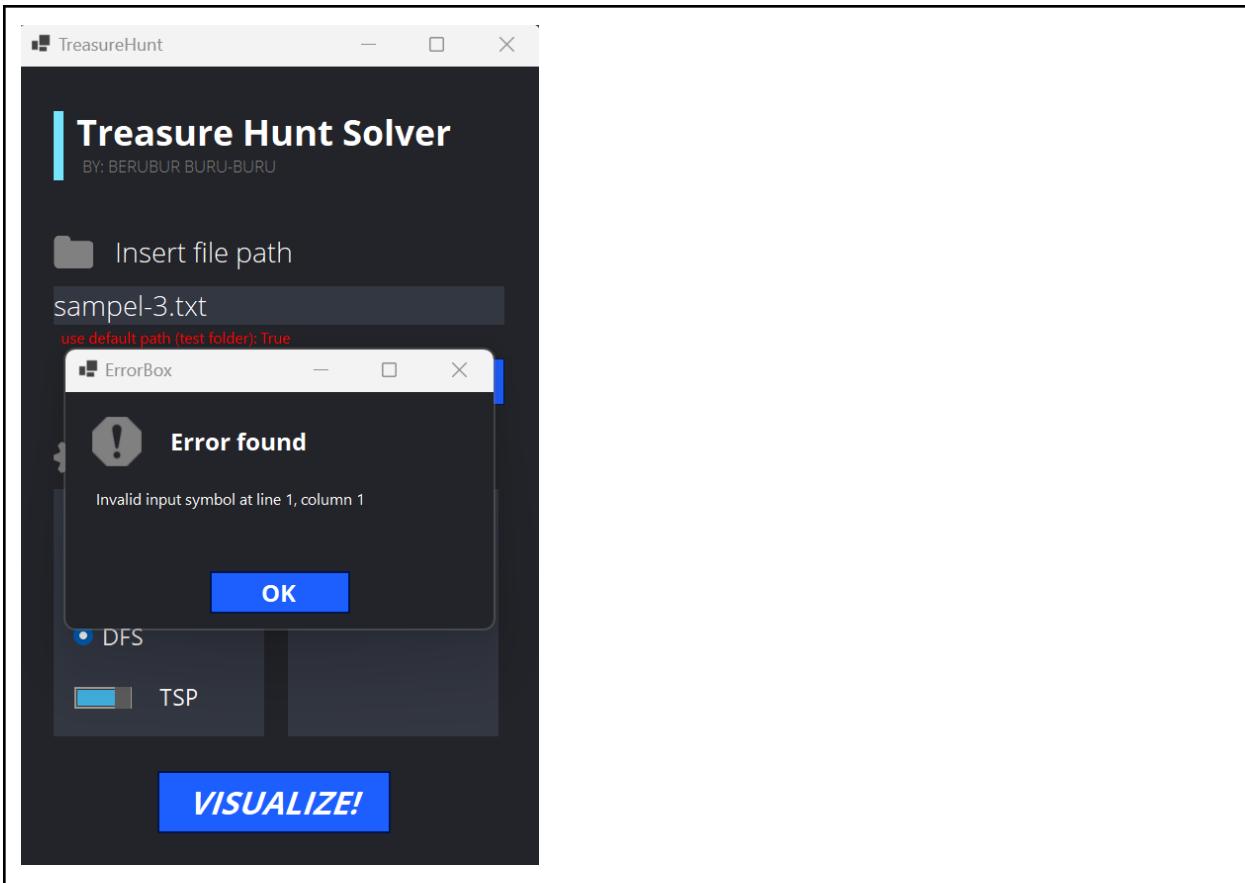


TESTING 5	
Skenario pengujian	Menggunakan algoritma DFS Menggunakan mode TSP Tidak menampilkan progres pencarian Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-2.txt
Isi <i>file</i>	X X X X X X X X X X X X X X X X X X T K R T X X X X X X X X X X X X X X X X X X
<p>Tampilan awal</p>	
Hasil pengujian	

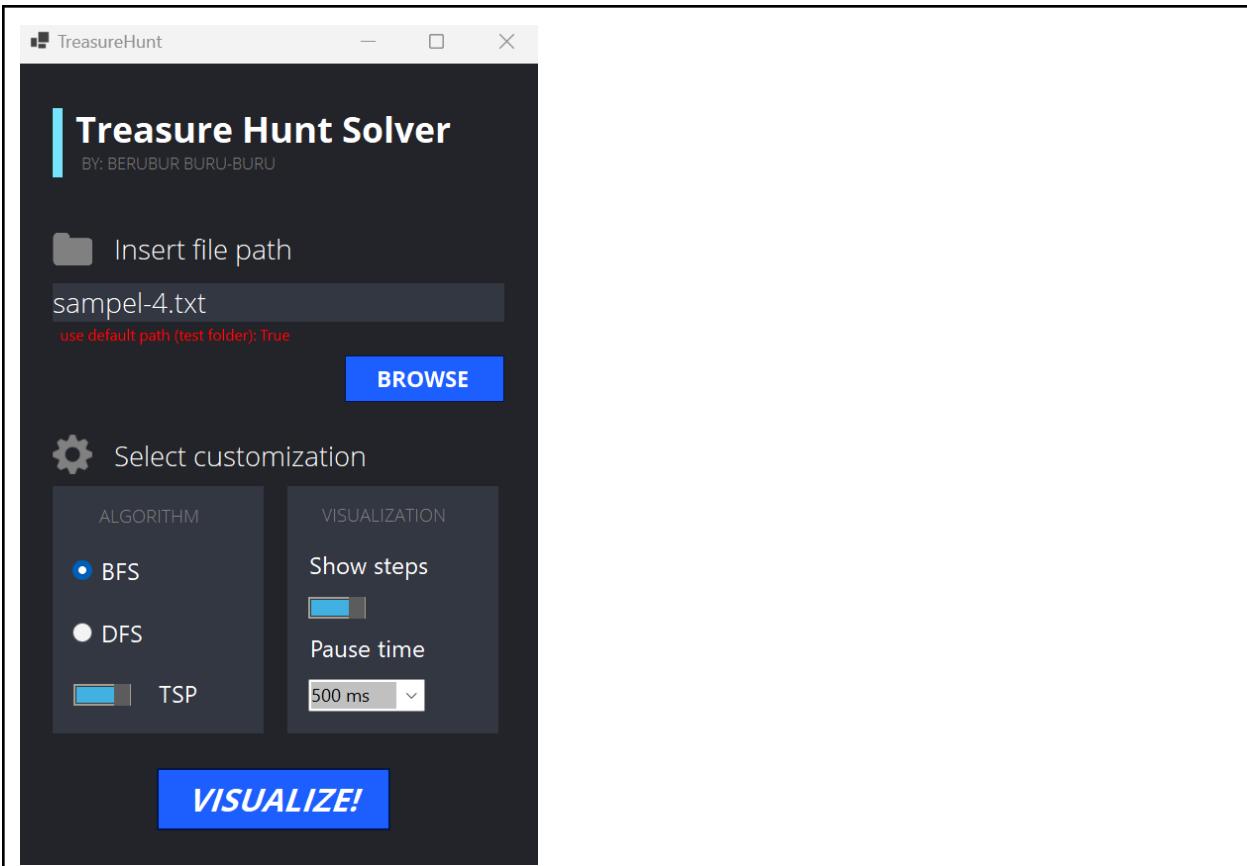


TESTING 6	
Skenario pengujian	Format <i>file</i> yang digunakan tidak valid Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-3.txt
Isi <i>file</i>	J A N G A N L U P A C E K Y A N G B E G I N I Y
Tampilan awal	

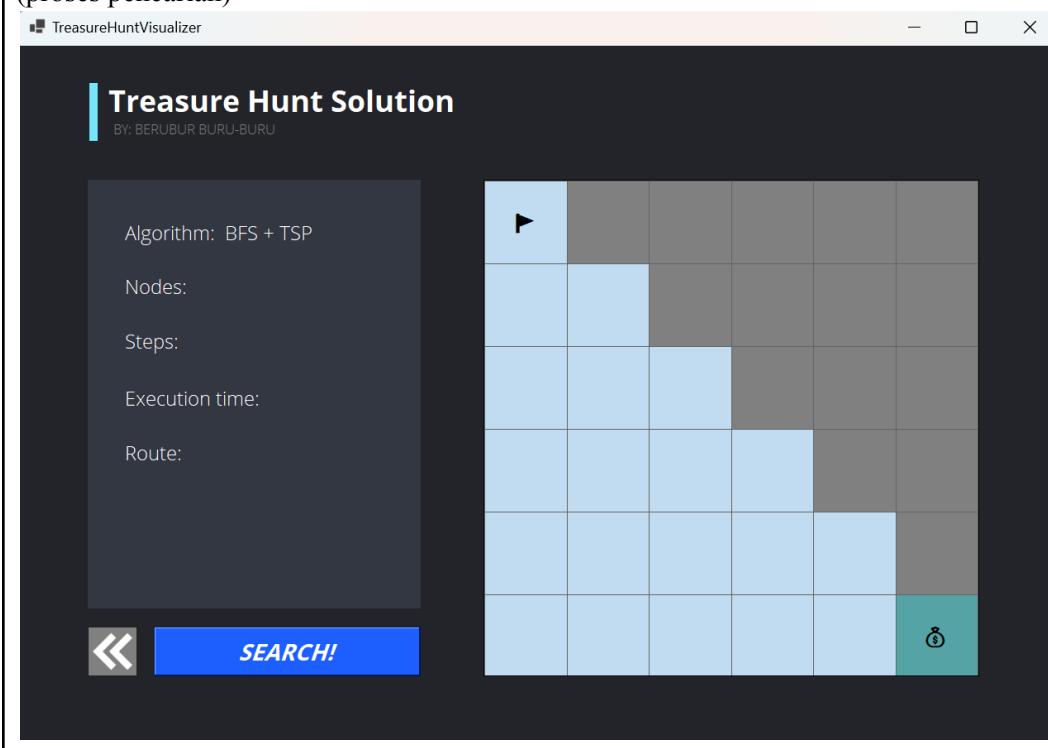




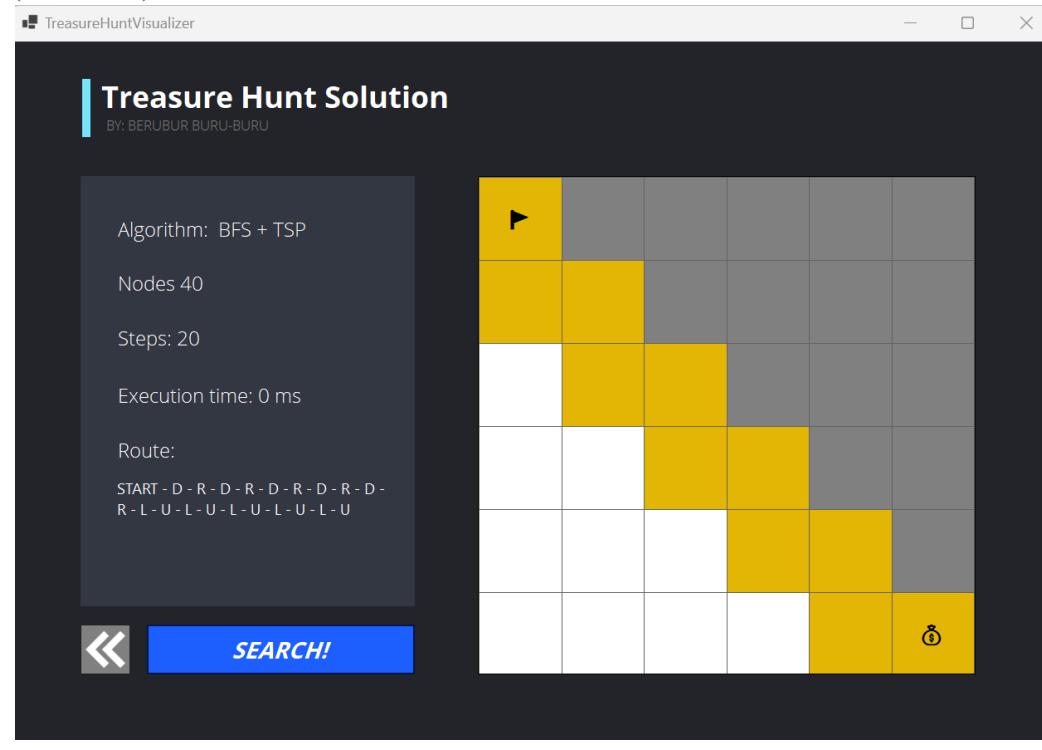
TESTING 7	
Skenario pengujian	Menggunakan algoritma BFS Menggunakan mode TSP Menampilkan progres pencarian Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-4.txt
Isi <i>file</i>	K X X X X X R R X X X X R R R X X X R R R R X X R R R R R X R R R R R T
Tampilan awal	



Hasil pengujian
(proses pencarian)

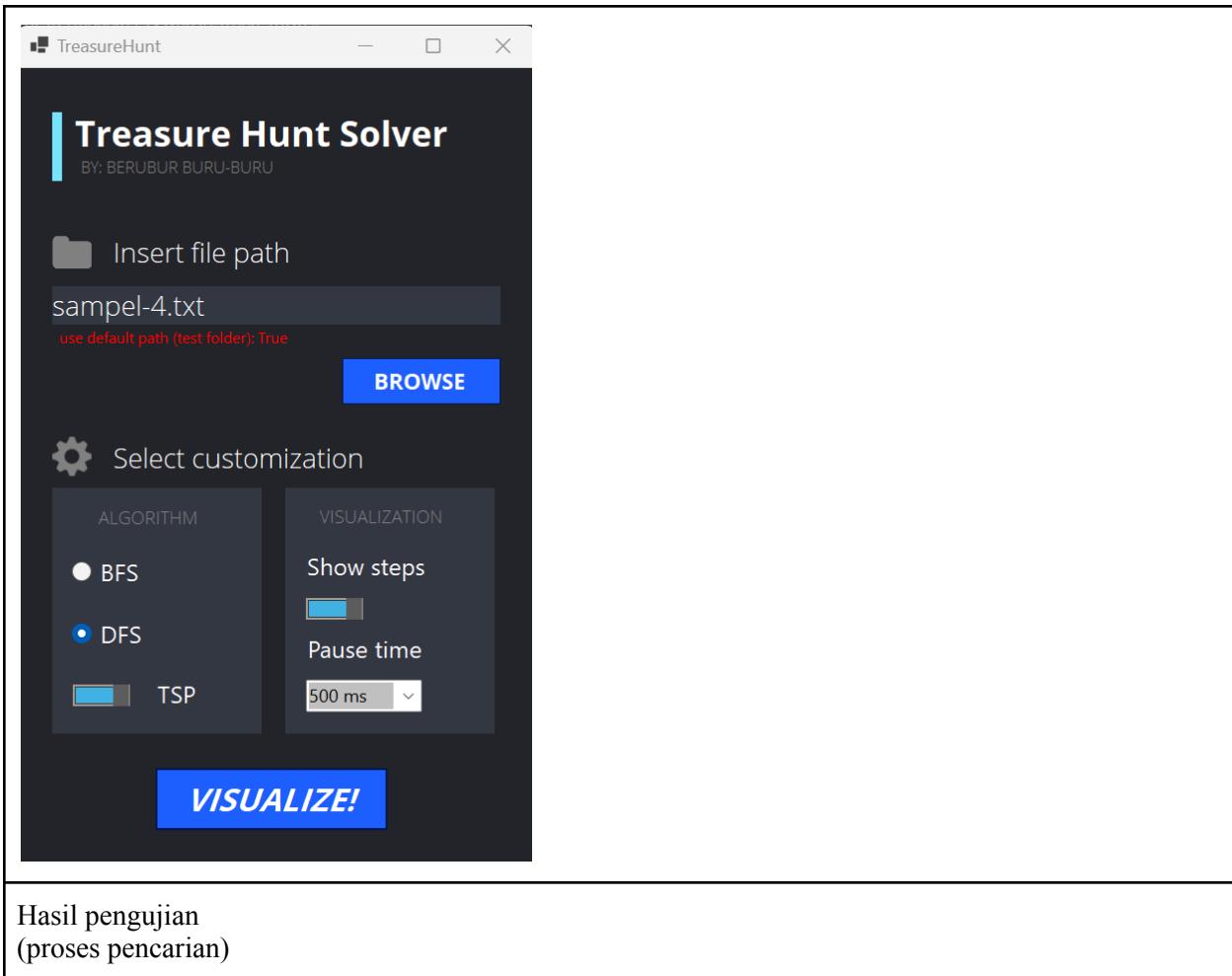


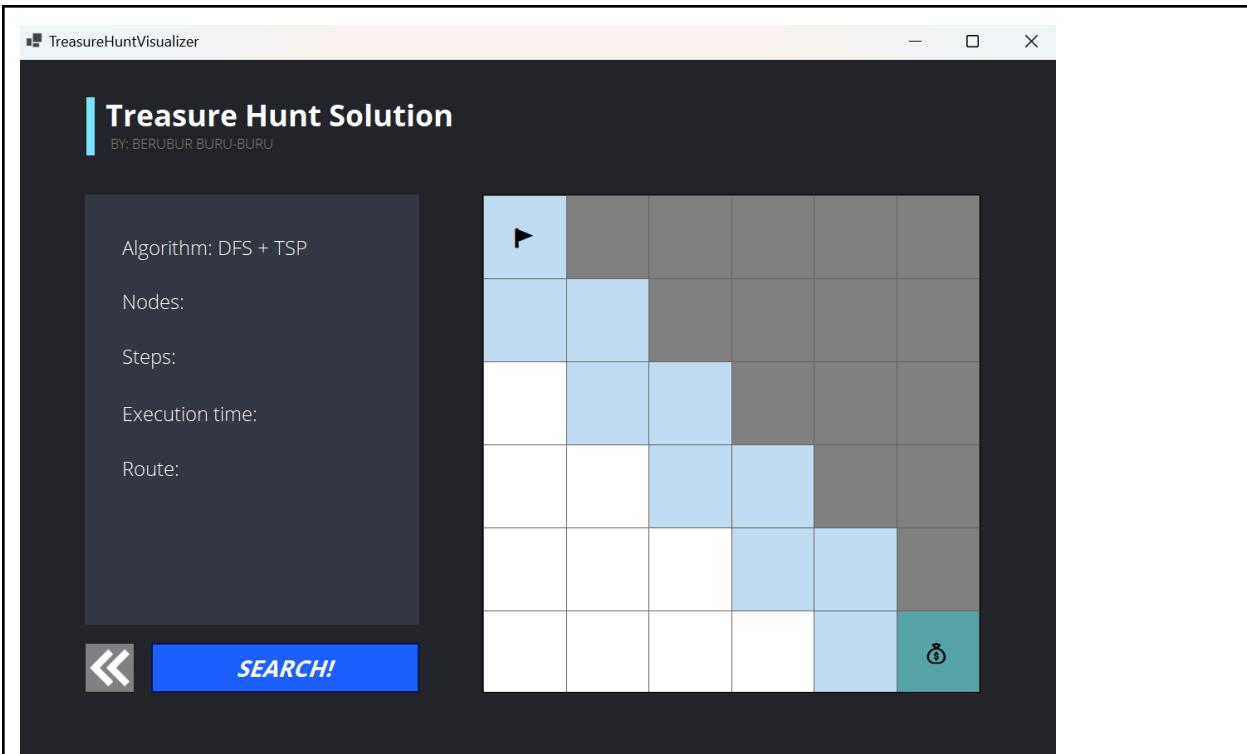
(rute solusi)



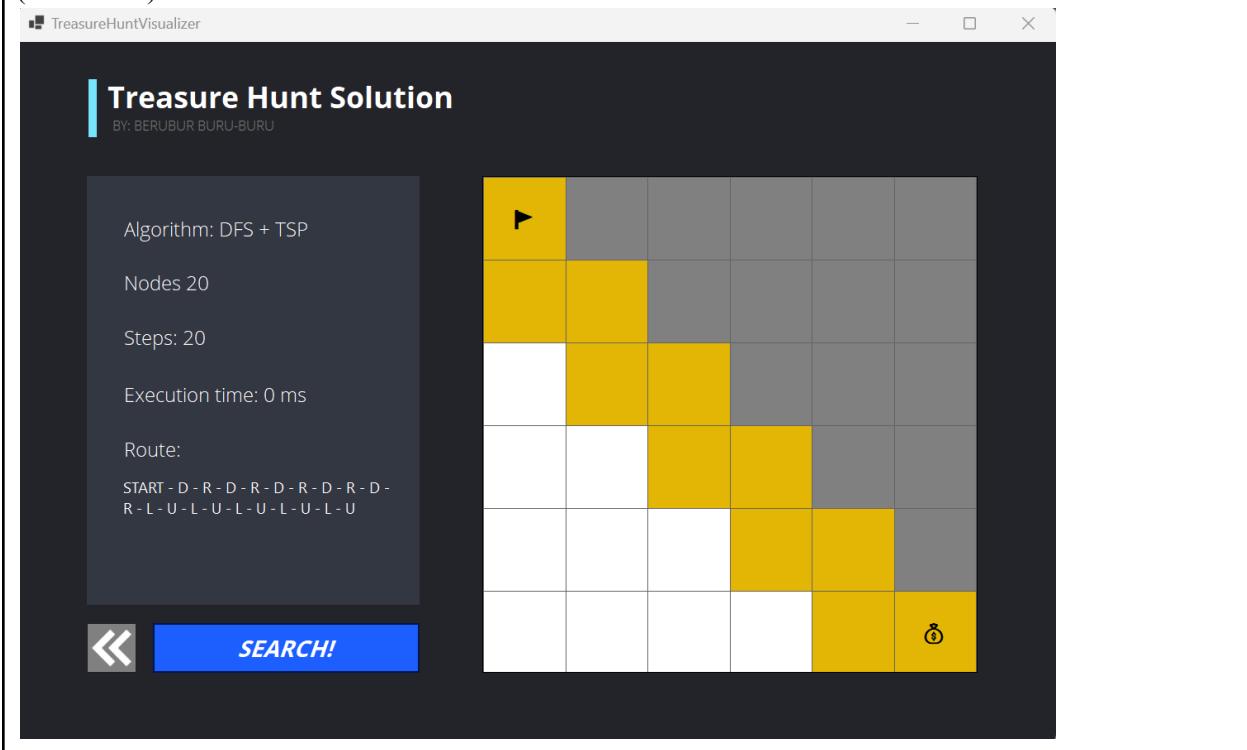
TESTING 8

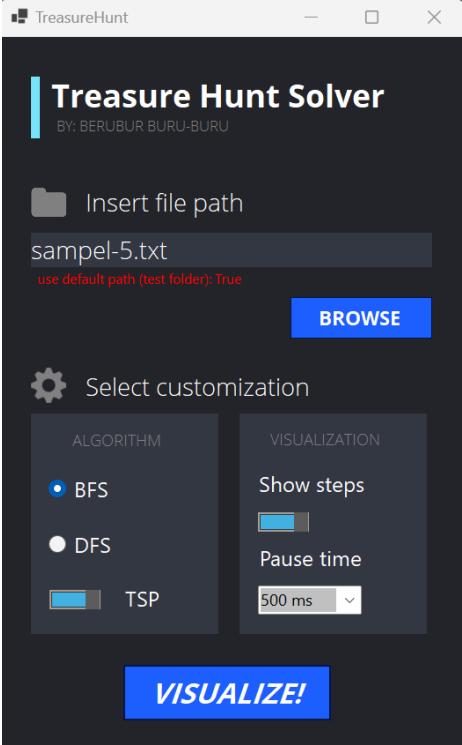
Skenario pengujian	Menggunakan algoritma DFS Menggunakan mode TSP Menampilkan progres pencarian Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-4.txt
Isi <i>file</i>	K X X X X X R R X X X X R R R X X X R R R R X X R R R R R X R R R R R T
Tampilan awal	

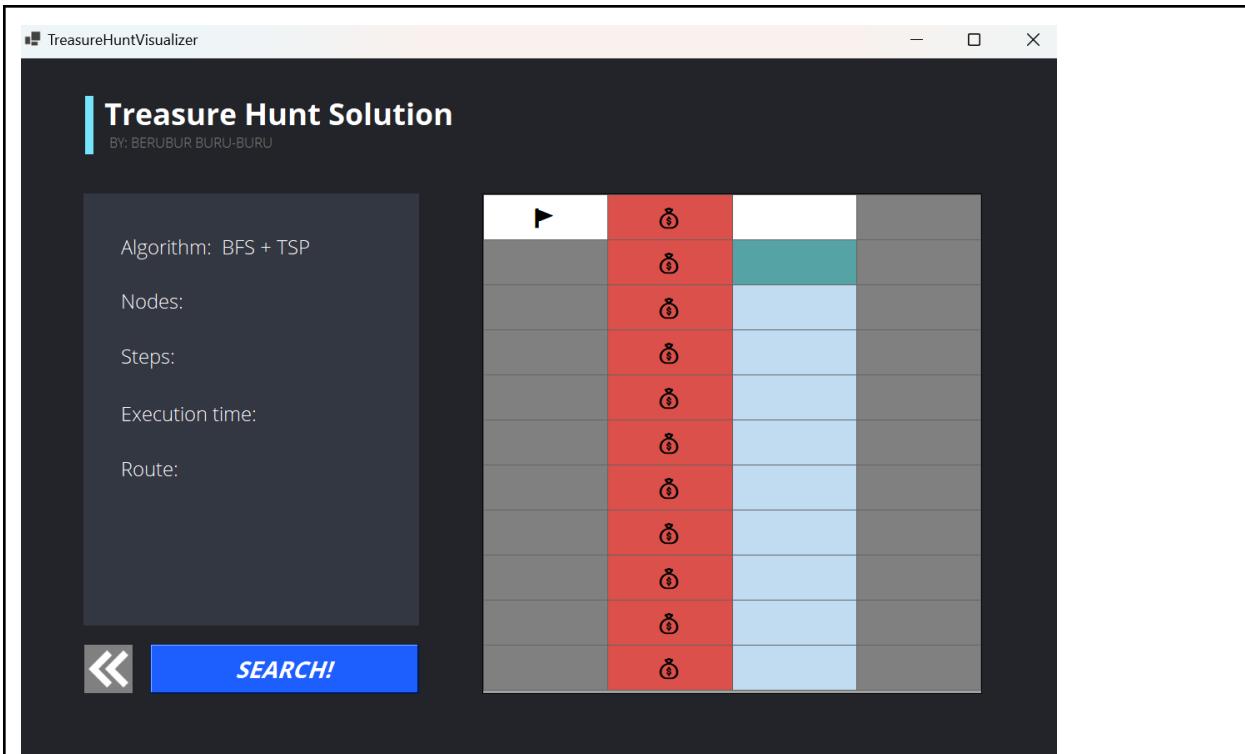




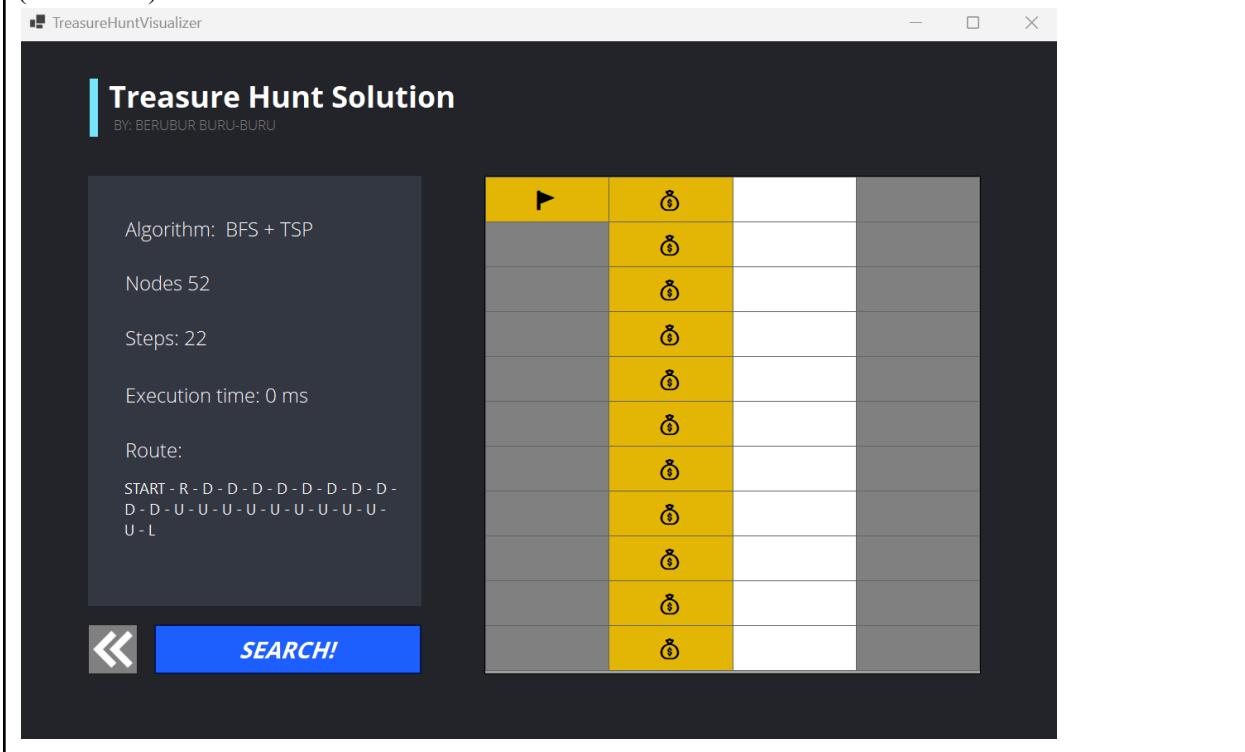
(rute solusi)

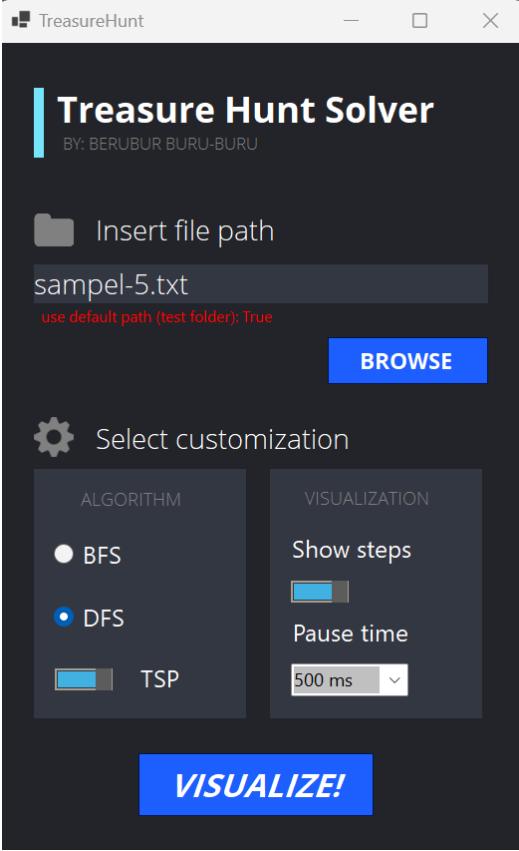


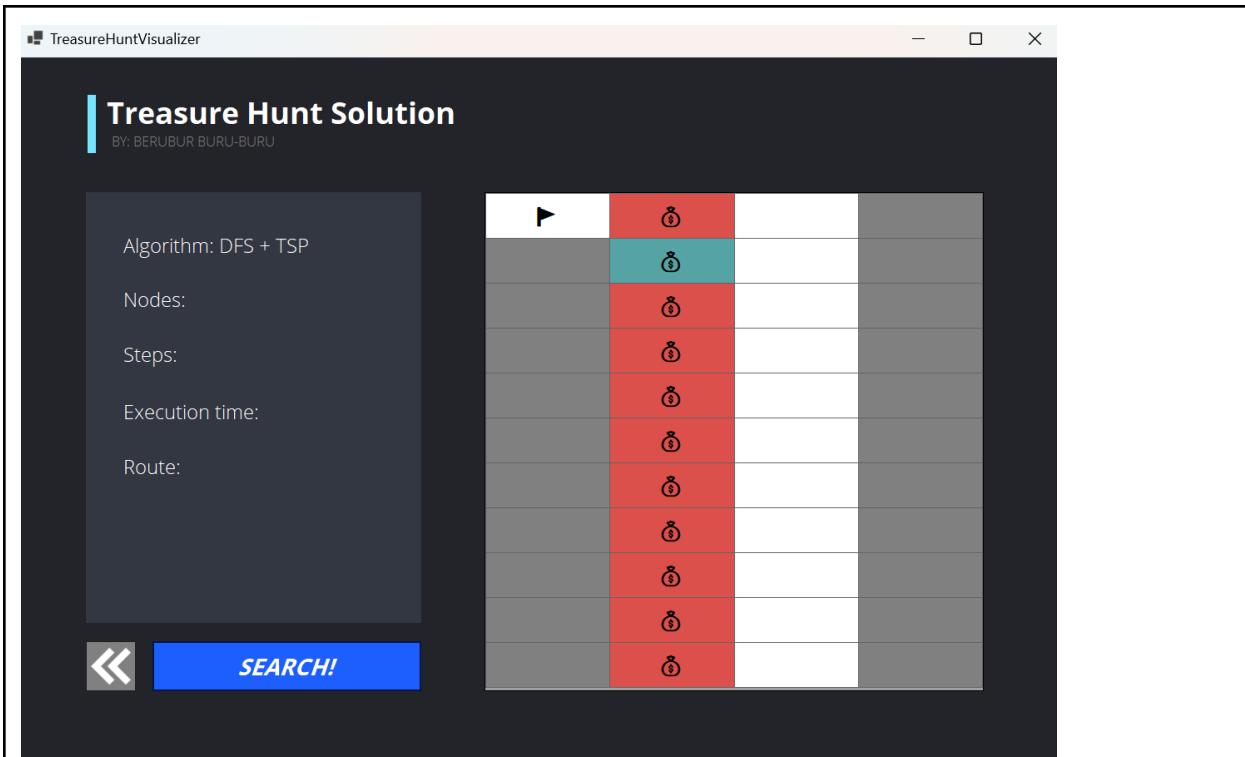
TESTING 9	
Skenario pengujian	Menggunakan algoritma BFS Menggunakan mode TSP Menampilkan progres pencarian Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-5.txt
Isi <i>file</i>	K T R X X T R X
Tampilan awal	
	
Hasil pengujian (proses pencarian)	



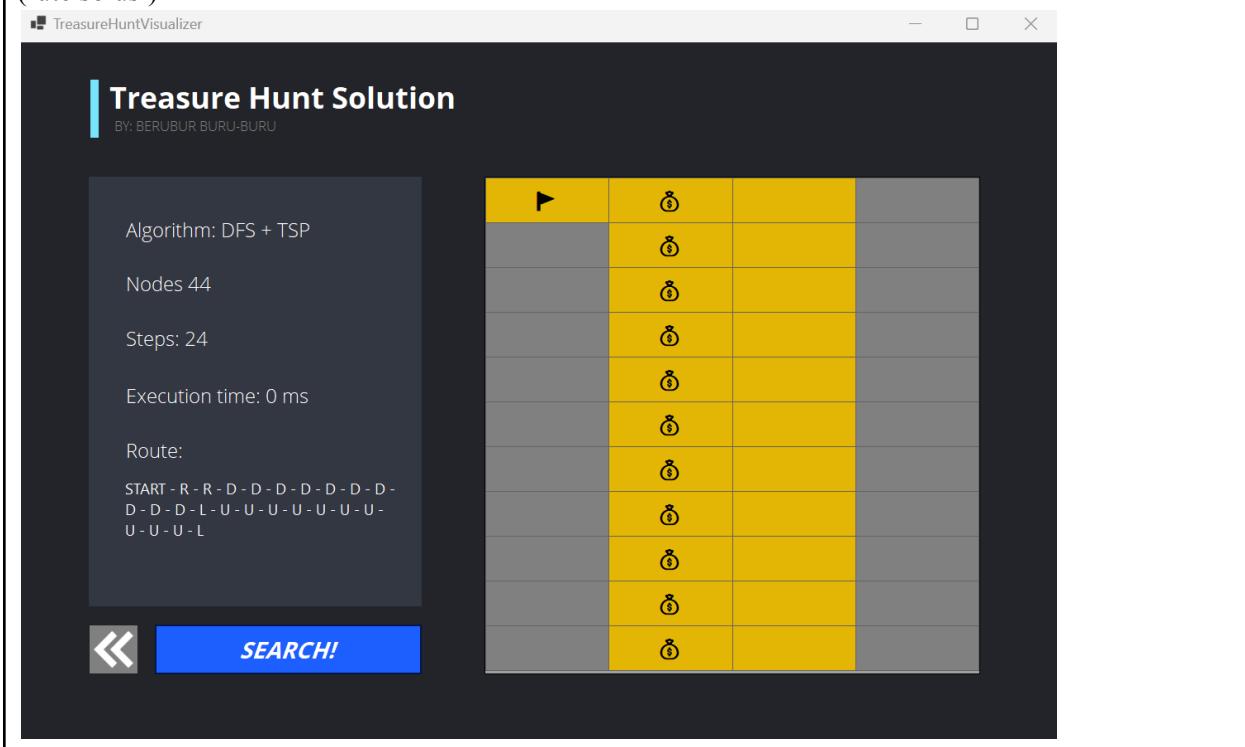
(rute solusi)



TESTING 10	
Skenario pengujian	Menggunakan algoritma DFS Menggunakan mode TSP Menampilkan progres pencarian Menggunakan <i>file</i> pada <i>folder</i> test
Nama <i>file</i>	sampel-5.txt
Isi <i>file</i>	K T R X X T R X
Tampilan awal	
	
Hasil pengujian (proses pencarian)	



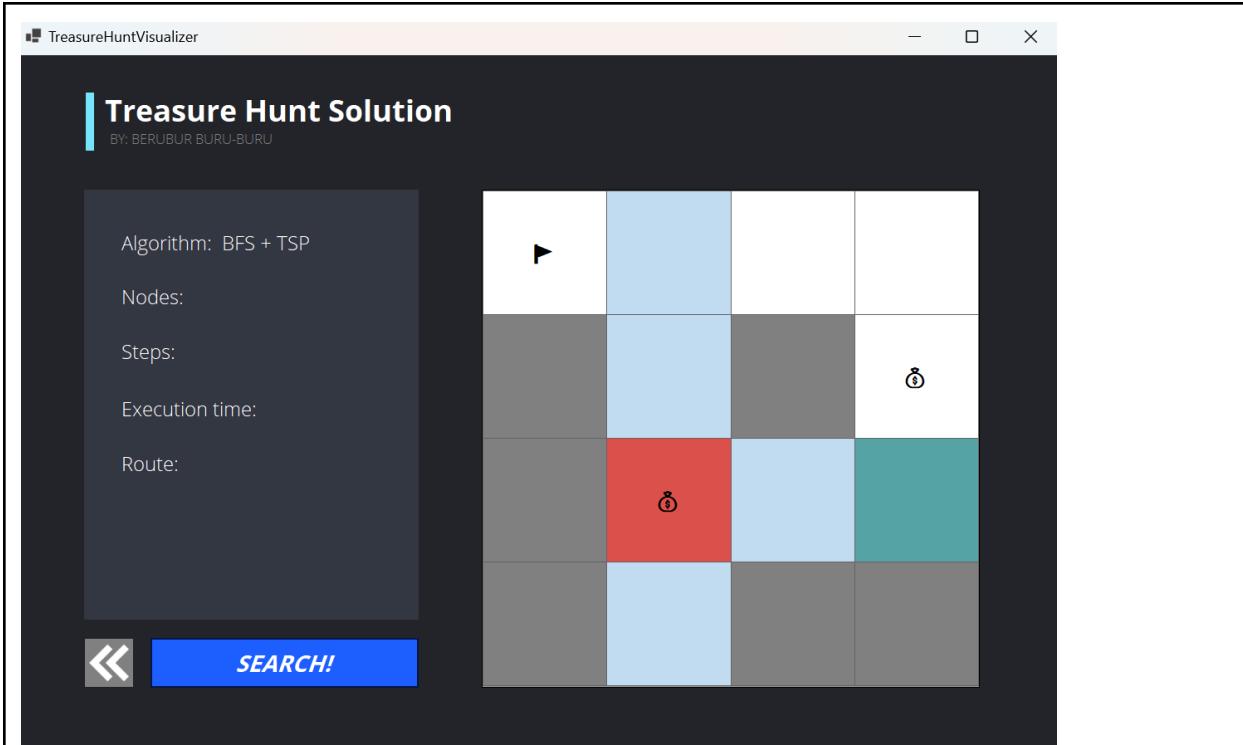
(rute solusi)



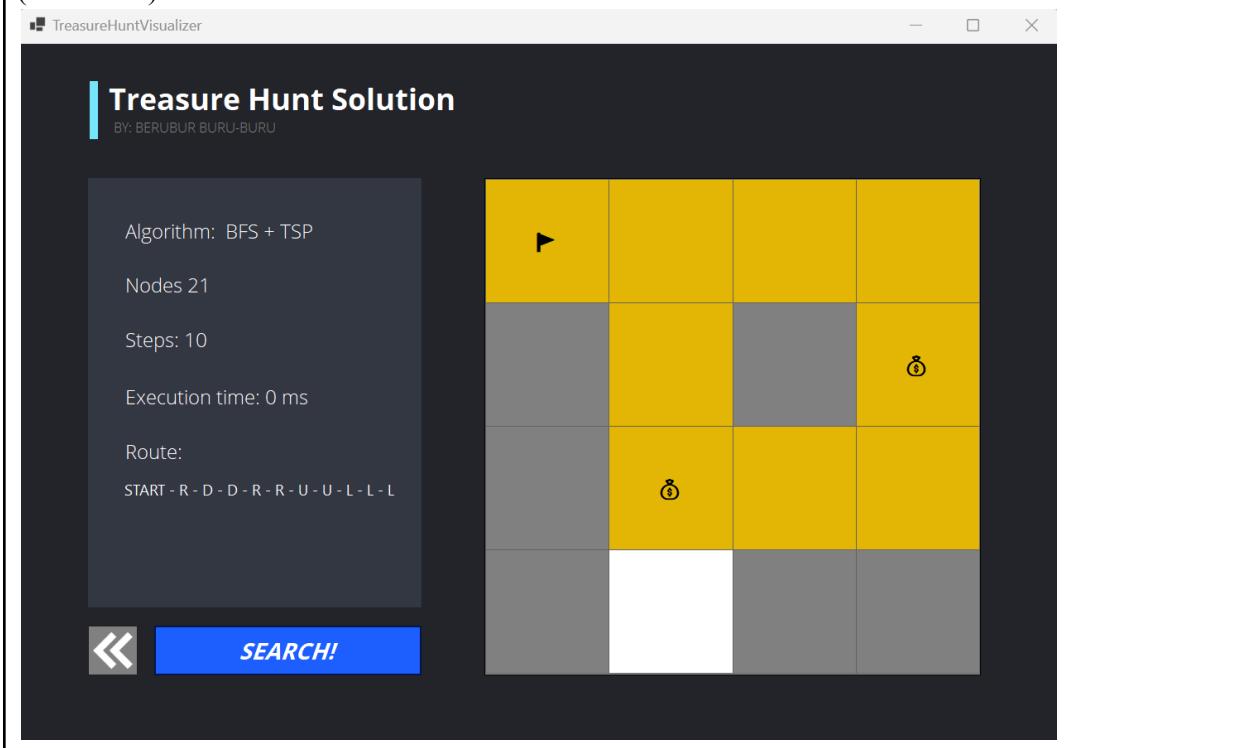
TESTING 11

Skenario pengujian	Menggunakan algoritma BFS
--------------------	---------------------------

	Menggunakan mode TSP Menampilkan progres pencarian Menggunakan <i>file</i> di luar <i>folder</i> test
Nama <i>file</i>	test1.txt
Isi <i>file</i>	K R R R X R X T X T R R X R X X
Tampilan awal	
Hasil pengujian (proses pencarian)	



(rute solusi)



4.5 Analisis desain solusi BFS dan DFS pada kasus uji

Dari hasil pengujian yang sudah dilakukan, algoritma DFS dan BFS sudah menjalankan tugasnya dengan baik dan memberikan hasil yang sesuai ketika mencari satu treasure atau lebih.

Algoritma BFS dan DFS juga sudah dapat digunakan untuk *traveling salesman problem* (TSP) dengan baik. Algoritma DFS dan BFS yang kami lakukan juga berjalan dengan cukup cepat, Namun agar memenuhi bonus, kami sengaja untuk memperlihat langkah-langkah pencarian dengan jeda dalam milidetik setiap langkahnya sehingga dapat terbaca dengan jelas langkah demi langkahnya.

Pada Testing 7 dan 8, dapat dilihat bahwa algoritma DFS (20 nodes) lebih optimal dibandingkan BFS (40 nodes). Berdasarkan pengamatan kami, jika *path* dari *start* ke *treasure* luas/tidak banyak halangan, maka algoritma DFS lebih unggul karena melakukan penelusuran terus-menerus tanpa *backtrack* akibat jalan yang terhalang sementara BFS perlu melakukan penelusuran ke semua percabangan yang mungkin. Namun jika terdapat banyak halangan, maka algoritma BFS lebih unggul karena algoritma DFS akan memerlukan lebih banyak *backtrack* jika menemui banyak jalan buntu.

BAB 5: Kesimpulan dan Saran

5.1. Kesimpulan

Algoritma *breadth-first search* dan *depth-first search* dapat digunakan untuk menyelesaikan permasalahan *Treasure-Hunt*. Algoritma *breadth-first search* melakukan penelusuran secara menyamping atau pada level yang sama terlebih dahulu, sedangkan algoritma *depth-first search* melakukan penelusuran secara mendalam. Pengembangan desktop application dapat menggunakan bantuan Visual Studio dan menggunakan bahasa pemrograman C#. Pengembangan GUI sangat terbantu dengan menggunakan WinForm.

5.2. Saran dan refleksi

Tubes ini adalah salah satu tubes yang berkesan, karena tubes ini adalah tubes pertama yang wajibkan penggunaan GUI, sehingga diperlukan eksplorasi mandiri terutama mengenai *framework* yang dibutuhkan dalam pembuatan GUI. Namun, terbatasnya pengetahuan mengenai GUI membuat kami harus berusaha ekstra untuk membuat tampilan yang menarik, terutama dengan *framework* WinForm dengan komponen UI yang cukup lawas dan menyediakan pilihan kustomisasi yang terbatas. Selain itu, tidak begitu mudah menemukan referensi yang bisa menjawab pertanyaan-pertanyaan mengenai WinForm, melihat pengembangan aplikasi desktop di saat ini sudah jarang yang menggunakan WinForm. Saran dari kami adalah untuk mencoba *framework* yang lebih baru jika ingin mendapatkan lebih banyak fitur UI dan kustomisasi yang lebih luas. Selain itu, kami sangat menyarankan untuk mematangkan perencanaan rancangan kelas dan objek yang diperlukan serta interaksi antarobjek pada program, misalnya data-data apa saja yang dimiliki oleh objek A yang harus disediakan oleh objek B, agar mempermudah proses pengembangan aplikasi. Kurangnya rancangan dapat menyebabkan banyaknya perombakan pada internal kelas seiring berjalannya pengembangan aplikasi.

Daftar Pustaka dan Link Terkait

- [Repository Github](#)
- [Video Bonus](#)
- <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
- <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>