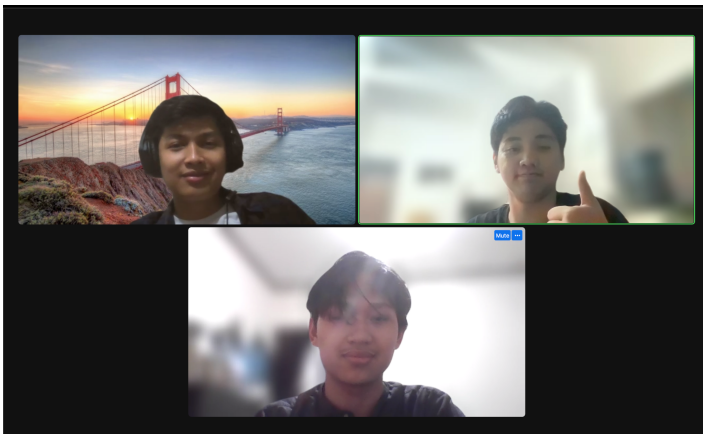
LAPORAN TUGAS BESAR 2 IF2211
APLIKASI ALGORITMA BFS DAN DFS DALAM MENYELESAIKAN PERSOALAN MAZE TREASURE HUNT

Ditujukan untuk memenuhi salah satu
Tugas Besar mata kuliah IF2122 Strategi Algoritma
pada Semester II Tahun Akademik 2022/2023



Oleh:

Fajar Maulana Herawan	13521080
Muhammad Rizky Sya'ban	13521119
Muhammad Naufal Nalendra	13521152

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

2022

DAFTAR ISI

DAFTAR ISI **2**
BAB I
DESKRIPSI MASALAH **3**
BAB II
LANDASAN TEORI **10**
2.1. Algoritma Pencarian Pada Graf 10
2.1.1. Breadth-First Search 10
2.1.2. Depth-First Search 10
2.2. C# Desktop Application Development 11
BAB III
LANGKAH PEMECAHAN MASALAH **12**
3.1. Langkah-langkah Pemecahan Masalah 12
3.2. Elemen-elemen Algoritma BFS dan DFS 12
3.2.1 Breadth-First Search 12
3.2.2 Depth-First Search 13
3.3. Ilustrasi Kasus Lain 13
BAB IV
ANALISIS PEMECAHAN MASALAH **15**
4.1. Implementasi Program 15
4.2. Struktur Data dan Spesifikasi Program 17
4.3. Penggunaan Program 21
4.4. Hasil Pengujian 24
4.5. Analisis Desain Solusi 29
BAB V
KESIMPULAN DAN SARAN **31**
BAB VI
LAMPIRAN **32**

BAB I
DESKRIPSI MASALAH

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 Labirin di Bawah Krusty Krab
(Sumber:https://static.wikia.nocookie.net/theloudhouse/images/e/ec/Massive_Mustard_Pocket.png /revision/latest?cb=20180826170029)

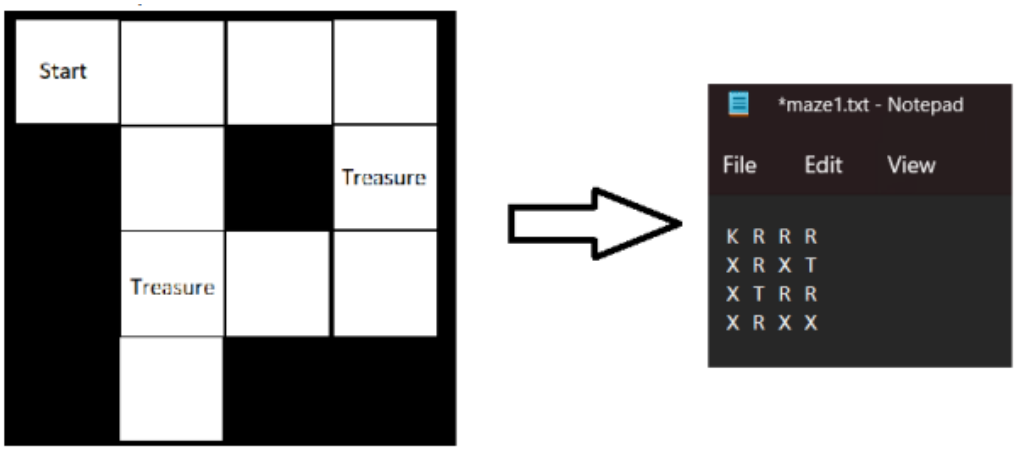
Setelah berpikir cukup lama, Tuan Krabs tiba-tiba mengingat bahwa ketika Ia berada pada kelas Strategi Algoritma-nya dulu, Ia ingat bahwa Ia dulu mempelajari algoritma BFS dan DFS sehingga Tuan Krabs menjadi yakin bahwa persoalan ini dapat diselesaikan menggunakan kedua algoritma tersebut. Akan tetapi, 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.

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

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

Contoh file input :

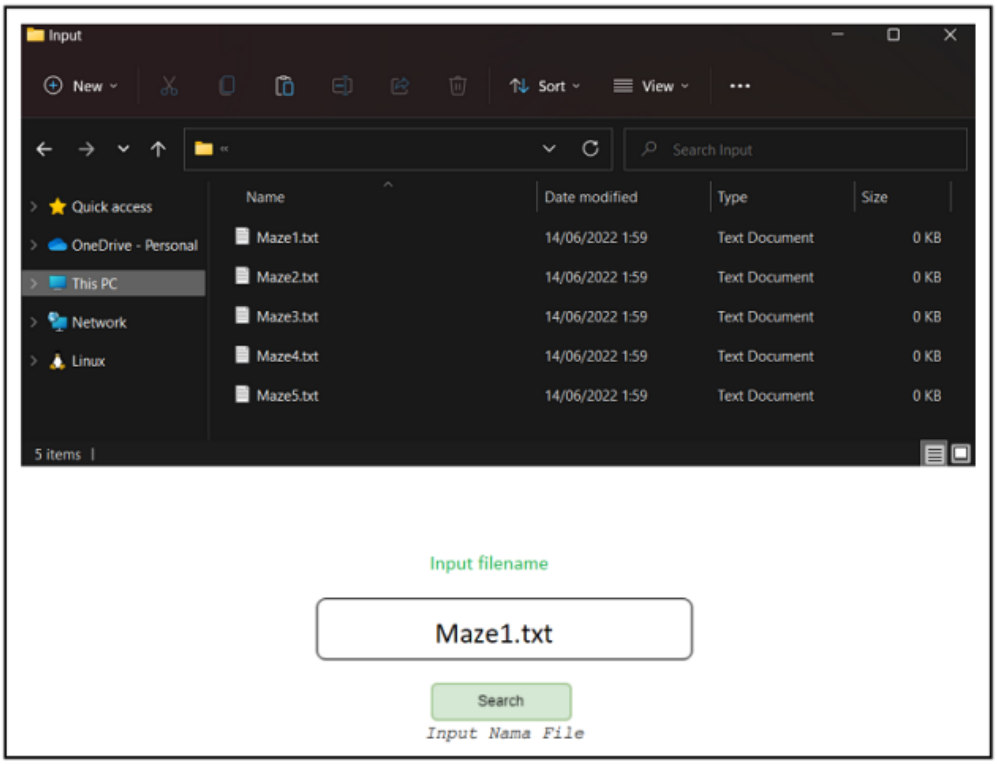


Gambar 2. Ilustrasi input file maze

Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), anda dapat menelusuri grid (simpul) yang mungkin dikunjungi hingga ditemukan rute solusi, baik secara melebar ataupun mendalam bergantung alternatif algoritma yang dipilih. Rute solusi adalah rute yang memperoleh seluruh treasure pada maze. Perhatikan bahwa rute yang diperoleh dengan algoritma BFS dan DFS dapat berbeda, dan banyak langkah yang dibutuhkan

pun menjadi berbeda. Prioritas arah simpul yang dibangkitkan dibebaskan asalkan ditulis di laporan ataupun readme, semisal LRUD (left right up down). Tidak ada pergerakan secara diagonal. Anda juga diminta untuk memvisualisasikan input txt tersebut menjadi suatu grid maze serta hasil pencarian rute solusinya. Cara visualisasi grid dibebaskan, sebagai contoh dalam bentuk matriks yang ditampilkan dalam GUI dengan keterangan berupa teks atau warna. Pemilihan warna dan maknanya dibebaskan ke masing - masing kelompok, asalkan dijelaskan di readme / laporan.

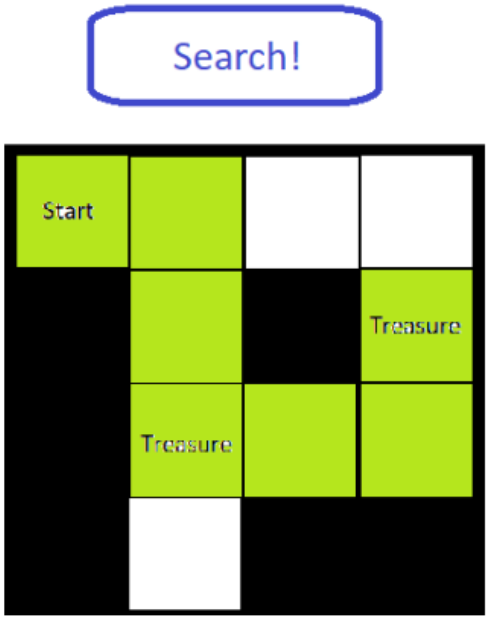
Contoh input aplikasi :



Gambar 3. Contoh input program

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

Contoh output Aplikasi :

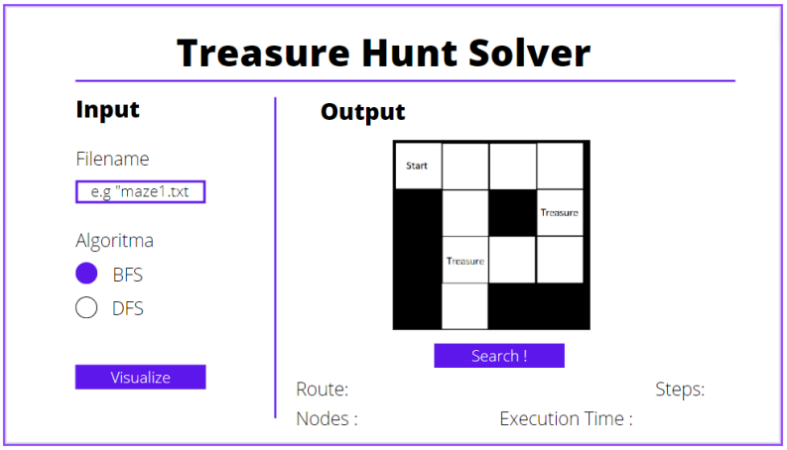


Gambar 4. Contoh output program untuk gambar 2

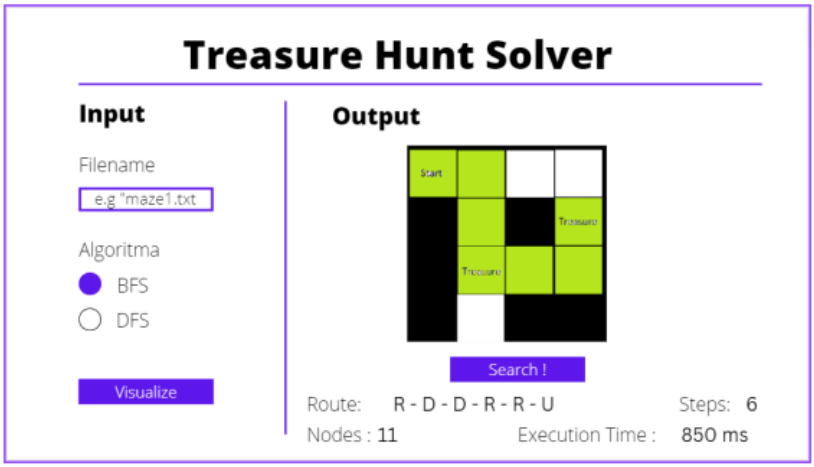
Setelah program melakukan pembacaan input, program akan memvisualisasikan gridnya terlebih dahulu tanpa pemberian rute solusi. Hal tersebut dilakukan agar pengguna dapat mengerjakan terlebih dahulu treasure hunt secara manual jika diinginkan. Kemudian, program menyediakan tombol solve untuk mengeksekusi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

Spesifikasi Program :

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



Gambar 5. Tampilan Program Sebelum dicari solusinya



Gambar 6. Tampilan Program setelah dicari solusinya

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

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.
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

BAB II
LANDASAN TEORI

2.1. Algoritma Pencarian Pada Graf

Algoritma traversal dalam graf adalah algoritma pengunjungan simpul-simpul dengan cara yang sistematik. Algoritma pencarian pada graf dapat dibagi menjadi dua jenis yaitu dengan informasi dan tanpa informasi. Algoritma *Breadth First Search* (*BFS*) dan *Depth First Search* (*DFS*) termasuk dalam jenis algoritma pencarian tanpa informasi. Pencarian pada graf juga bergantung pada jenis grafnya, yaitu graf statis dan tidak statis. Graf statis sudah terbentuk sebelum dilakukan penelusuran, sedangkan graf tidak statis terbentuk selama penelusuran. Keduanya, BFS dan DFS, merupakan algoritma pencarian untuk graf statis.

2.1.1. Breadth-First Search

Breadth-First Search atau BFS merupakan algoritma yang paling sederhana dalam melakukan pencarian pada graf dan pola dasar pada algoritma pencarian graf lainnya. Dalam algoritma *Breadth-First Search*, graf direpresentasikan dalam bentuk $G = (V, E)$ dengan v merupakan simpul awal penulusuran. Secara sederhana, algoritma breadth-first search akan memulai penulusuran dari simpul v. Kemudian, 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. Untuk mengetahui simpul yang akan diperiksa, akan digunakan struktur data Queue.

2.1.2. Depth-First Search

DFS atau *Depth-First Search* adalah algoritma penelusuran graf yang melakukan traversal secara mendalam. Berbeda dengan *DFS*, algoritma pencarian *DFS* melakukan pencarian terlebih dahulu untuk satu kemungkinan simpul tetangga dengan prioritas tertinggi pada setiap pencariannya sampai tidak ada lagi simpul tetangga yang tersedia. Setelah itu akan dilakukan runut-balik (*backtrack*) untuk melanjutkan pencarian ke kemungkinan simpul tetangga yang lain yang belum dikunjungi. Misal pada graf $G = (V, E)$ algoritma *DFS* akan memulai penulusuran dari simpul v. Kemudian, akan dilanjutkan penelusuran ke simpul v yang bertetangga dengan simpul v dan memiliki prioritas tertinggi. Setelah itu akan dilakukan pemanggilan kembali algoritma DFS yang dimulai dari simpul v. Ketika mencapai simpul v 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. Untuk mengetahui simpul yang akan diperiksa, akan digunakan struktur data Stack.

2.2. C# Desktop Application Development

Desktop app ini ditulis dengan bahasa C# (C-Sharp) di dalam aplikasi visual studio menggunakan framework WPF (Windows Presentation Foundation). Windows Presentation Foundation adalah kerangka antarmuka pengguna untuk membuat aplikasi desktop di sistem operasi Windows. Ini menyediakan model pemrograman yang terpadu untuk membangun antarmuka pengguna yang menarik secara visual.

WPF menggunakan XAML (Extensible Application Markup Language) untuk mendefinisikan elemen antarmuka pengguna, yang dapat dimanipulasi dan diatur gayanya menggunakan C# atau bahasa .NET lainnya. Kerangka kerja ini juga mendukung pengikatan data, yang memungkinkan sinkronisasi mudah elemen UI dengan model data.

Secara keseluruhan, WPF menyediakan platform yang kuat dan fleksibel untuk membuat aplikasi desktop yang memberikan pengalaman pengguna yang sangat menarik dan menarik secara visual.

BAB III
LANGKAH PEMECAHAN MASALAH

3.1. Langkah-langkah Pemecahan Masalah

Langkah awal dari proses memecahkan masalah dengan memecah permasalahan *find treasure* ini menjadi bagian yang lebih kecil. Permasalahan kami bagi menjadi dua bagian yakni pencarian solusi menggunakan algoritma *breadth first search* dan dengan menggunakan *depth first search*.Selain itu, ada permasalahan lain yakni menampilkan solusi dari dua permasalahan tersebut dalam bentuk *desktop application*.

Setelah membagi permasalahan menjadi dua bagian, kita harus melakukan pemetaan masalah tersebut agar dapat digunakan oleh masing masing algoritma *breadth first search* dan dengan menggunakan *depth first search.* Kami merepresentasikan masalah ke dalam bentuk objek tile .Semua tile tersebut disimpan dalam tiles.

Dalam menyelesaikan masalah tambahan, yakni menampilkan solusi dari masing masing algoritma. Kami menggunakan GUI dalam Windows Presentation Foundadtion yang dapat menerima masukkan pengguna berupa input file, pilihan algoritma (BFS atau DFS) serta pilihan untuk visualisasi TSP. Dari inputan - inputan tersebut, Kami juga menvisualisai mase pada GUI tersebut. Setelah itu, kami menampilkan solusi yakni jalur untuk menemukan *treasure* dalam *maze* tersebut. Pada akhirnya, kami melakukan *testing* dari GUI yang telah kita buat.

3.2. Elemen-elemen Algoritma BFS dan DFS

3.2.1 *Breadth-First Search*

Algoritma *breadth-first search* pada permasalahan ini menggunakan konsep pencarian solusi pada graf dinamis. Dalam merepresentasikan graf ini, kami menyimpan masing masing simpul atau dalam masalah ini yakni *tile* dan masing *tile* tersebut akan menyimpan beberapa informasi seperti nilai , tile tersebut bernilai T,R,X,atu K. Kita juga menyimpan apakah *tile* tersebut sudah dikunjungi atau belum dan menyimpan alur pencarian dari simpul start.Elemen antrian atau *queue* berisi dengan *tile* yang akan diperiksa.

3.2.2 *Depth-First Search*

Algoritma *depth-first search* pada permasalahan ini menggunakan konsep pencarian solusi pada graf dinamis. Dalam merepresentasikan graf ini, kami menyimpan masing masing simpul atau dalam masalah ini yakni *tile* dan masing *tile* tersebut akan menyimpan beberapa informasi seperti nilai , tile tersebut bernilai T,R,X,atu K. Kita juga menyimpan apakah *tile* tersebut sudah dikunjungi atau belum dan menyimpan alur pencarian dari simpul start.Elemen tumpukan atau *stack* berisi dengan *tile* yang akan diperiksa.

3.3. Ilustrasi Kasus Lain

Dalam pencarian solusi dari suatu maze terdapat banyak *treasure* yang ada " bisa lebih dari satu". Sehingga untuk kasus pencarian solusi dari maze menggunakan algoritma *breadth-first search* perlu dilakukan iterasi untuk setiap penggunaan algoritma *breadth-first search* dan mengganti *start*-nya dengan *tile treasure* yang telah ditemukan sebelumnya. Kita juga tidak lupa untuk *mengatur* ulang kondisi *tile* seperti semula yakni kondisi dimana semua *tile* belum diperiksa.Berikut merupakan *pseudocode* dari algoritma yang digunakan

```
Procedure IterativeBFS(input v:Tile,input treasure:list of
tile,input/output path: list of tuple(string,int,int))
{Masukan : v adalah simpul awal kunjungan 'Krusty Krab'
 Keluaran : path untuk mengunjungi semua treasure
```

```
Deklarasi
Count : integer
Procedure BuatAntrian(input/output q : antrian)
{Membuat antrian kosong}
Procedure MasukAntrian(input/output q : antrian, input v :
Tile)
{memasukan tile v kedalam antrian q}
Procedure HapusAntrian(input/output q : antrian, input v:
Tile)
{menghapus v dari kepala antrian q}
Function AntrianKosong(input q: antrian) -> boolean
{true jika antrian kosong false jika antrian tidak kosong}
Procedure remove(input/ouput l: list of tile, input v : tile)
{menghapus tile v dari list l}
Procedure refresh(input/output l: list of tile, input/output
:antrian)
{ mereset semua tile dan queue ke keadaan semula}
```

```
Procedure visit(input t : tile,input/output l : antrian,input
direction : string):
{kunjungi t dan masukan t ke antrian l,sebelumnya di lakukan
pengecekan apakah t memiliki tetangga dengan arah direction}
```

```
Algoritma
Count <- treasure.count() {banyak treasure}
BuatAntrian(q)
Repeat Count times:
  hasVisited(v)
  MasukAntrian(q,v)
  While not AntrianKosong(q) do
    HapusAntrian(q,a)
    If v ada di treasure then
      tulispath(path)
      refresh(semua tile,q)
      v = a {set v dengan a}
      Berhenti
    Else
       visit(a,bawah)
       visit(a,kanan)
       visit(a,atas)
       visit(a,kiri)
```

BAB IV

ANALISIS PEMECAHAN MASALAH

4.1. Implementasi Program

Breadth-First Search

```
Procedure BFS(input v:Tile,input treasure:list of
tile,input/output path: list of tuple(string,int,int))
{Masukan : v adalah simpul awal kunjungan 'Krusty Krab'
 Keluaran : path untuk mengunjungi semua treasure}
```

```
Deklarasi
Count : integer
Procedure BuatAntrian(input/output q : antrian)
{Membuat antrian kosong}
Procedure MasukAntrian(input/output q : antrian, input v :
Tile)
{memasukan tile v kedalam antrian q}
Procedure HapusAntrian(input/output q : antrian, input v: Tile)
{menghapus v dari kepala antrian q}
Function AntrianKosong(input q: antrian) -> boolean
{true jika antrian kosong false jika antrian tidak kosong}
Procedure remove(input/ouput l: list of tile, input v : tile)
{menghapus tile v dari list l}
Procedure refresh(input/output l: list of tile, input/output
:antrian)
{ mereset semua tile dan queue ke keadaan semula}
Procedure visit(input t : tile,input/output l : antrian,input
direction : string):
{kunjungi t dan masukan t ke antrian l,sebelumnya di lakukan
pengecekan apakah t memiliki tetangga dengan arah direction}
```

```
Algoritma
Count <- treasure.count() {banyak treasure}
BuatAntrian(q)
Repeat Count times:
  hasVisited(v)
  MasukAntrian(q,v)
  While not AntrianKosong(q) do
    HapusAntrian(q,a)
    If v ada di treasure then
      tulispath(path)
```

```
      refresh(semua tile,q)
      v = a {set v dengan a}
      Berhenti
    Else
       visit(a,bawah)
       visit(a,kanan)
       visit(a,atas)
       visit(a,kiri)
```

Depth-First Search

```
Procedure DFS(input v:Tile,input treasure:list of
tile,input/output path: list of tuple(string,int,int))
{Masukan : v adalah simpul awal kunjungan 'Krusty Krab'
 Keluaran : path untuk mengunjungi semua treasure}
```

```
Deklarasi Fungsi
Count : integer
Procedure BuatStack(input/output s :stack)
{Membuat stack kosong}
Procedure MasukStack(input/output s :stack, input v : Tile)
{memasukan tile v kedalam stack q}
Procedure refreshPath(input/output l: list of tile,
input/output :stack)
{ mereset path dalam semua tile dan mengosongkan stack}
Procedure visitDFS(input t : tile,input/output s :stack,input
direction d: string):
{masukan t ke stack s dan set origin tile t dengan d}
```

```
Algoritma Procedure run()
BuatStack(s)
Count <- treasure.count() {banyak treasure}
Repeat Count times:
  hasVisited(v)
  MasukStack(s,v)
  While not StackKosong(s) do
    Pop(s,a)
    hasVisited(a)
    addPath(a, direction)
    If v ada di treasure then
      tulispath(path dari v)
      refreshPath(semua tile,q)
      v = a {set v dengan a}
      Berhenti
```

```
    If (a tidak punya origin)
       visitDFS(a,kiri)
       visitDFS(a,atas)
       visitDFS(a,kanan)
       visitDFS(a,bawah)
    Else
       visitDFS dengan prioritas arah kebalikan origin (menuju
origin)
```

Traveling Salesman Problem

```
Procedure TSP(input v:Tile,input treasure:list of
tile,input/output path: list of tuple(string,int,int))
{Masukan : v adalah simpul awal kunjungan 'Krusty Krab'
 Keluaran : path untuk mengunjungi semua treasure}
```

```
Deklarasi
Procedure refresh(input/output l: list of tile, input/output
:antrian)
{ mereset semua tile dan queue ke keadaan semula}
Procedure add(input/output treasure: list of tile, input home :
tile)
{menambahkan home ke treasure}
Procedure backtohome(input home: tile){
  add(treasure,home)
  refresh(tiles,antrian/stack)
  BFS() atau DFS()
```

```
Algoritma
BFS() atau DFS()
backtohome(home)
```

4.2. Struktur Data dan Spesifikasi Program

Kelompok kami menggunakan konsep pemrograman berbasis objek untuk struktur data beserta spesifikasi program yang digunakan dalam membuat program:

1. Kelas BaseFS
   Kelas ini merupakan *abstract base class* dari kelas BFS dan DFS.Atribut yang disimpan oleh kelas ini meliputi:
   a. path
      Attribut ini memiliki tipe data *list of tuple* yang *tuple* berbentuk <*string,integer,integer*>. *Path* berfungsi untuk menyimpan alur dari start untuk mendapatkan semua *treasure*nya.
   b. pathHistory

Attribut ini memiliki tipe data *list of tuple* yang *tuple* berbentuk <*integer,integer,integer*>.Path berfungsi untuk menyimpan tile apa saja yang sudah diperiksa beserta berapa kali tile tersebut diperiksa.

c. tiles
Attribut ini memiliki tipe data list of tiles. Tiles berfungsi untuk menyimpan semua tile yang ada pada peta kecuali yang memiliki value X

d. treasure
Attribut ini memiliki tipe data list of tiles. treasure berfungsi untuk menyimpan semua tile yang ada pada peta yang memiliki value T

e. start
Attribut ini memiliki tipe data tile. Start berfungsi untuk menyimpan tile yang menjadi simpul awal dalam menjalan algoritma BFS atau DFS

f. home
Attribut ini memiliki tipe data tile. Home berfungsi untuk menyimpan tile yang memiliki value K

Selain itu ada method

a. getResultPath

   Berfungsi untuk mengembalikan path

b. getHistoryPath

   Berfungsi untuk mengembalikan historypath

c. addTile2History

   Berfungsi untuk menambahkan tile ke history

d. appendPath

   Berfungsi untuk menambahkan ke path

e. printStep

   Berfungsi untuk menampilkan path

f. inputTile

   Berfungsi untuk meng-*enqueue* atau *push*

g. refresh

   Berfungsi untuk mengembalikan kondisi tile ke unvisited dan me-reset queue atau stack

h. run

   Menjalankan algoritma pencarian

i. backtoHome

   Menjalankan algoritma pencarian ke home

j. runTSP

Menjalan algoritma TSP

2. Kelas BFS
   Kelas ini merupakan kelas turunan dari kelas BaseFS.Kelas ini memiliki atribut tambahan yakni Queue.Kelas ini berfungsi untuk meng-*override* kelas BaseFS dan mengimplementasikan dengan pendekatan algoritma BFS. Kelas ini memiliki method visit untuk mengunjungi tetangga dari tile
3. Kelas DFS
   Kelas ini merupakan kelas turunan dari kelas BaseFS. Kelas ini memiliki atribut tambahan stack dan lastTile.Kelas ini berfungsi untuk meng-*override* kelas BaseFS dan mengimplementasikan dengan pendekatan algoritma DFS.Kelas ini memiliki method visitDFS untuk mengunjungi tetangga dari tile
4. Kelas Tile
   Kelas ini berfungsi untuk merepresentasikan tile pada map. Kelas ini memiliki beberapa atribut.
   a. Coordinate
      Attribut ini berfungsi untuk menyimpan koordinat dari tile di map. Attribut ini memiliki tipe list of integer
   b. Value
      Attribut ini berfungsi untuk menyimpan value dari tile di map. Attribut in memiliki tipe string.
   c. Visited
      Attribut ini berfungsi untuk mengetahui apakah tile sudah dikunjungi atau belum. Attribut ini memiliki tipe boolean.
   d. path
      Attribut ini berfungsi untuk menyimpan *path* dari tile yang sudah dikunjungi sebelumnya. Attribut ini memiliki tipe list of tuple <string,integer,integer>.
   e. Down
      Attribut ini berfungsi untuk menyimpan tetangga bawah dari tile tersebut. Attribut ini memiliki tipe Tile.
   f. Right
      Attribut ini berfungsi untuk menyimpan tetangga kanan dari tile tersebut. Attribut ini memiliki tipe Tile.
   g. Up
      Attribut ini berfungsi untuk menyimpan tetangga atas dari tile tersebut.
      Attribut ini memiliki tipe Tile.
   h. Left
      Attribut ini berfungsi untuk menyimpan tetangga kiri dari tile tersebut.
      Attribut ini memiliki tipe Tile.

Selain itu, Kelas ini memiliki beberapa method.method tersebut terdiri dari method getter dan setter berikut:

a. getValue
b. isVisited
c. getPath
d. getCoordinate
e. getDown
f. getRight
g. getUp
h. getLeft
i. setCoordinate
j. setValue
k. setDown
l. setRight
m. setUp
n. setLeft
o. hasVisited

Dan method tambahan seperti:

a. addPath
   Method ini untuk menambahkan ke path
b. reset
   Method ini untuk mereset tile
c. resetPath
   Method ini untuk mereset path
d. printInfo
   Method ini untuk menampilkan informasi tile.

5. Kelas Tiles
   Kelas ini merupakan representasi map yang di-input dari file berupa *list of tile*. Kelas ini melakukan *parser* dari file ke tile. Adapun attribute dari kelas ini.
   a. Tiles
      Atribut ini memiliki tipe list of tile. Atribut ini berfungsi untuk menyimpan tile.
   b. Start
      Atribut ini memiliki tipe tile. Attribut ini berfungsi untuk menyimpan tile yang memiliki value K
   c. Treasure
      Atribut ini memiliki tipe list of tile. Attribut ini berfungsi untuk menyimpan tile yang memiliki value T
   d. matrix
      Attribut ini memiliki tipe matrix of string. Attribut ini merupakan representasi matrix dari inputan file

Adapun kelas ini memiliki beberapa method ,yakni
Method dibawah ini merupakan getter

a. getTiles
b. getTreasure
c. getStart
d. getMatrix

Method selanjutnya yakni method tambahan

e. addTile
   Method ini berfungsi untuk menambahkan tile ke tiles
f. parserFile
   Method ini berfungsi untuk mengolah file input
g. convMatrix
   Method ini berfungsi untuk mengubah bentuk matrix ke tiles
h. setAdjacency
   Method ini berfungsi untuk mengeset tetangga dari setiap tiles
i. printMatrix
   Method ini berfungsi untuk menampilkan matrix
j. printTile
   Method ini berfungsi untuk menampilkan Tiles

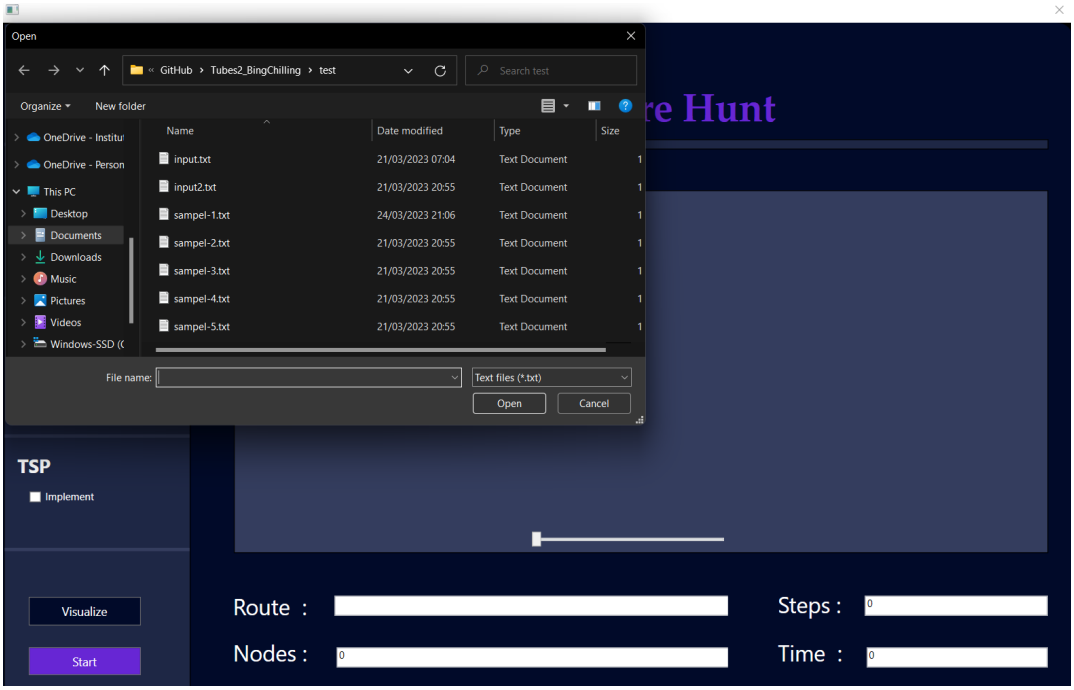
6. Kelas Maze
   Kelas Maze ini berfungsi untuk mengubah inputan file dan ditampilkan dengan GUI.
   Kelas ini memiliki beberapa attribut yakni
   a. mazeContents
   b. startTile
   c. Length
   d. Width
   e. countTreasure
   Dan memiliki method getter dan setter.

4.3. Penggunaan Program

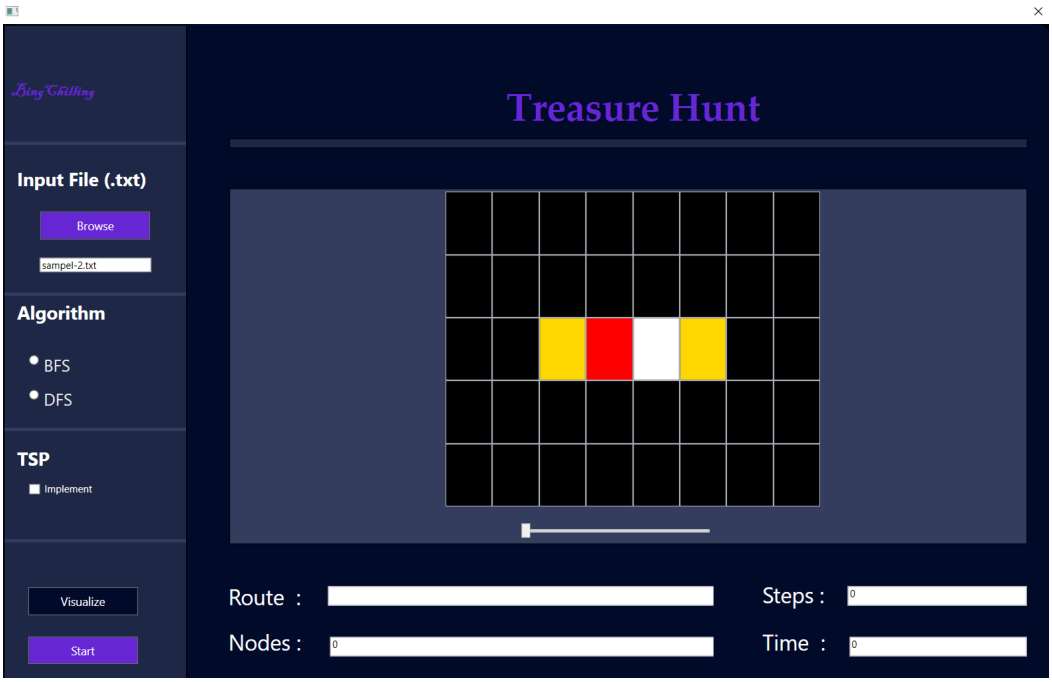
Cara menggunakan program adalah sebagai berikut:

1. Jalankan aplikasi Tubes2_BingChilling
2. Akan muncul window GUI

3. Klik button ‘Browse’ untuk memilih input file

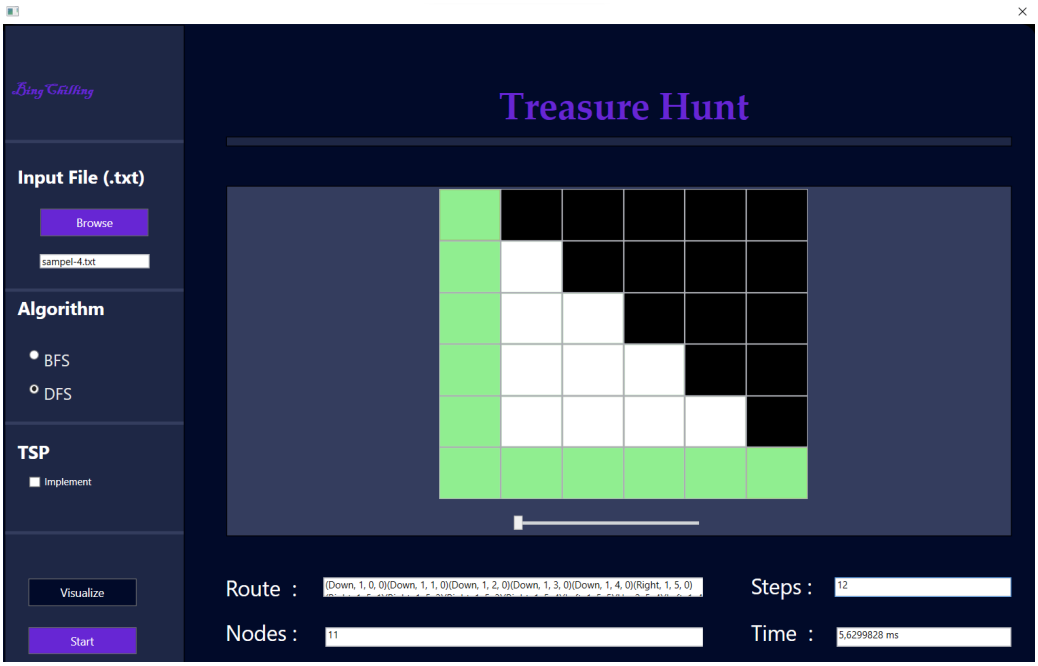


4. Setelah file terpilih klik button ‘visualize’ untuk memunculkan gambar maze ke layar



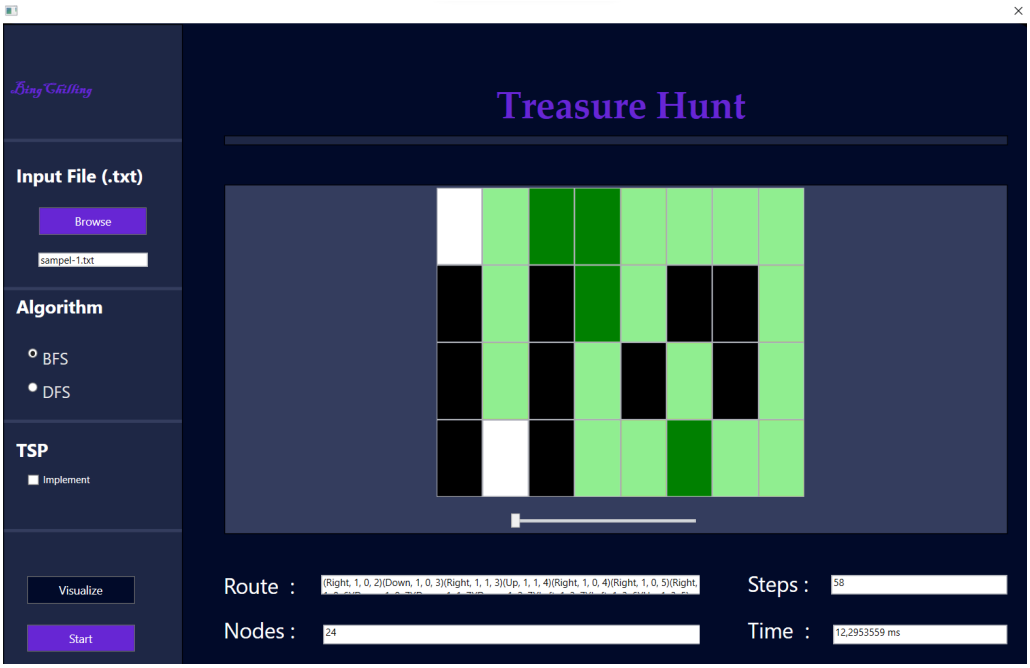
5. Pengguna diharuskan memilih algoritma pencarian, baik itu BFS atau DFS
6. Pengguna juga dapat memilih untuk menerapkan TSP atau tidak
7. Klik tombol ‘Start’ dan langkah pencarian akan muncul di layar

8. Langkah yang muncul pada layar akan berbeda warna sesuai jumlah sebuah tile dilewati mulai dari paling terang hingga paling gelap
9. Pada setiap text box dibawah layar maze akan muncul berbagai output mulai dari rute, jumlah node, jumlah langkah, hingga waktu eksekusi

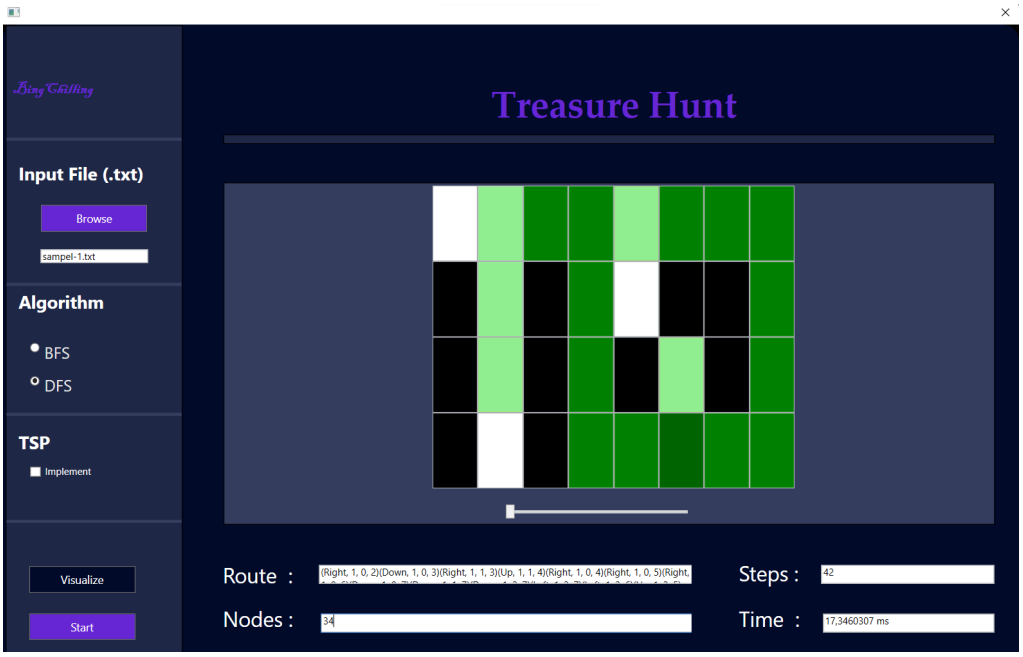


4.4. Hasil Pengujian

1. Test Case 1 - BFS

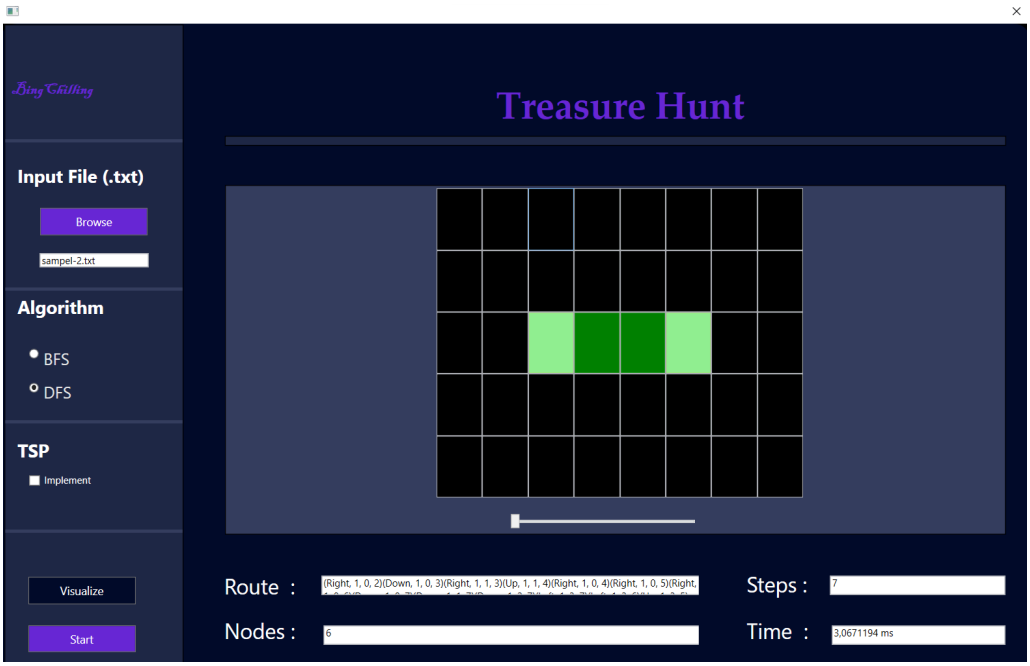


2. Test Case 1 - DFS

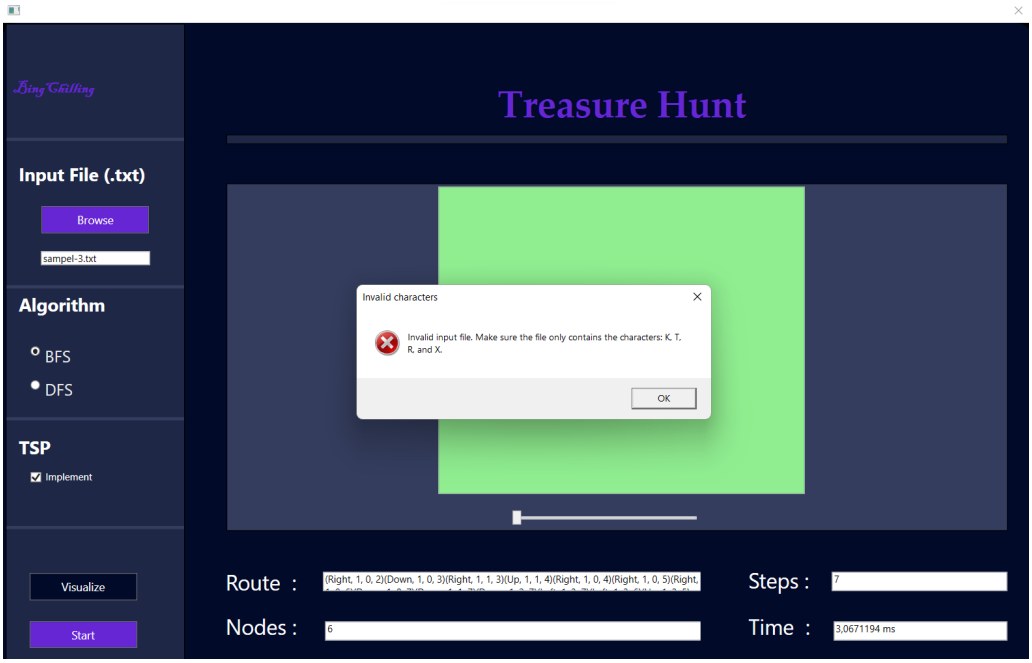


3. Test Case 2 - BFS

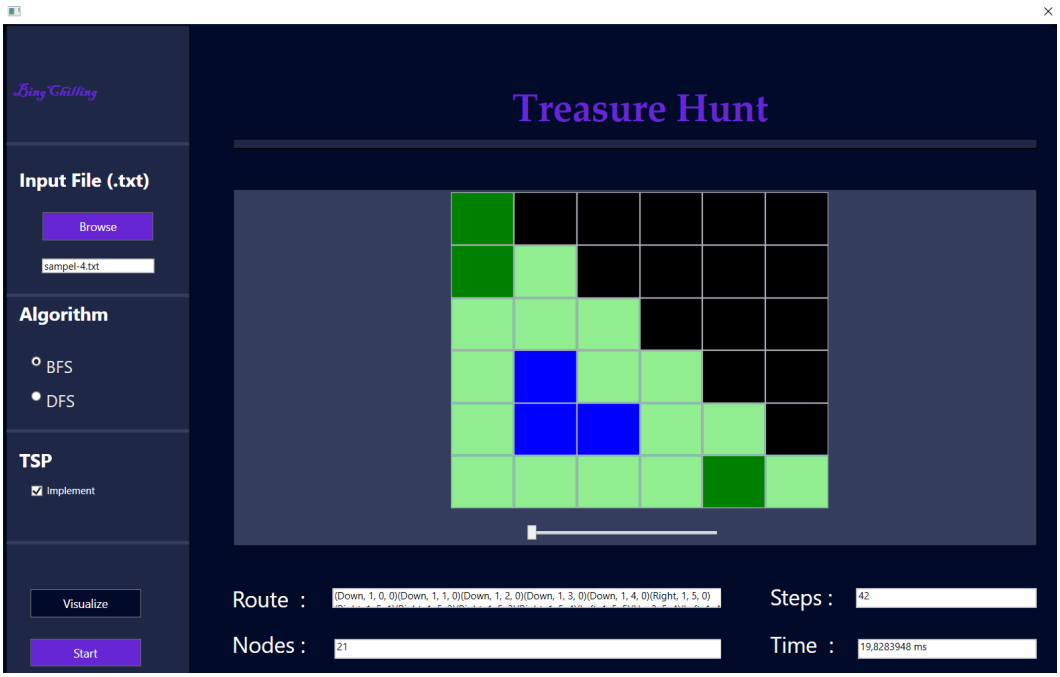
4. Test Case 2 - DFS


Bing Chilling
Treasure Hunt
Input File (.txt)
Browse
sampel-2.txt
Algorithm
BFS
DFS
TSP
Implement
Visualize
Start
Route :
(Right, 1, 0, 2)(Down, 1, 0, 3)(Right, 1, 1, 3)(Up, 1, 1, 4)(Right, 1, 0, 4)(Right, 1, 0, 5)(Right,
Steps :
7
Nodes :
6
Time :
3,0671194 ms

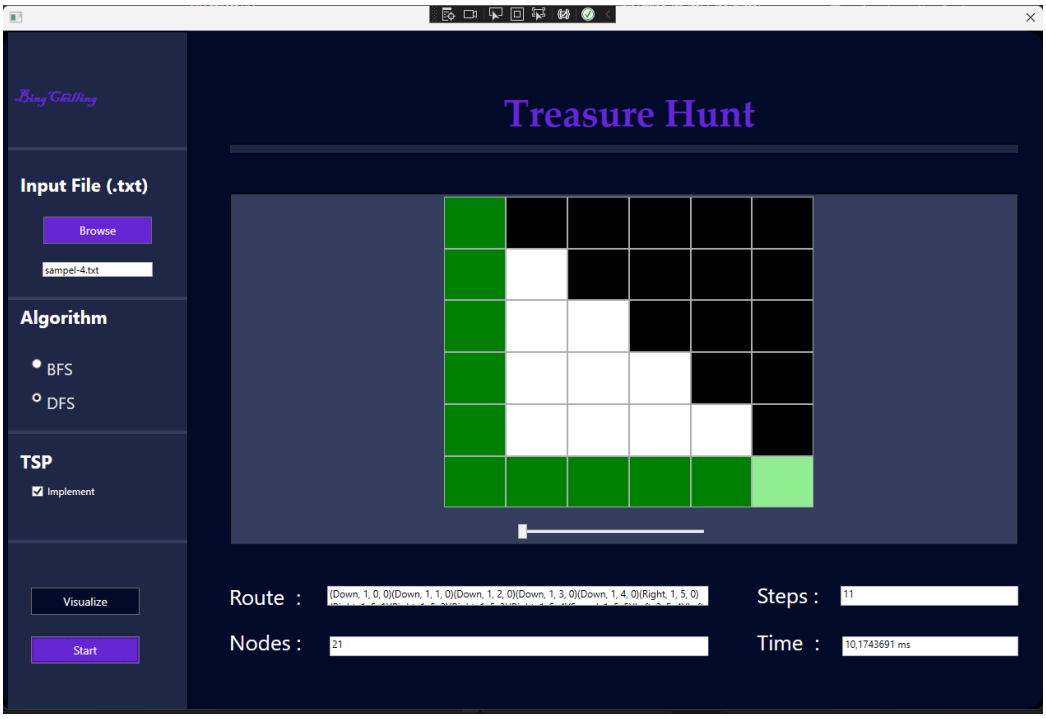
5. Test Case 3


Bing Chilling
Treasure Hunt
Input File (.txt)
Browse
sampel-3.txt
Algorithm
BFS
DFS
TSP
Implement
Visualize
Start
Invalid characters
Invalid input file. Make sure the file only contains the characters: K, T, R, and X.
OK
Route :
(Right, 1, 0, 2)(Down, 1, 0, 3)(Right, 1, 1, 3)(Up, 1, 1, 4)(Right, 1, 0, 4)(Right, 1, 0, 5)(Right,
Steps :
7
Nodes :
6
Time :
3,0671194 ms

6. Test Case 4 - BFS + TSP

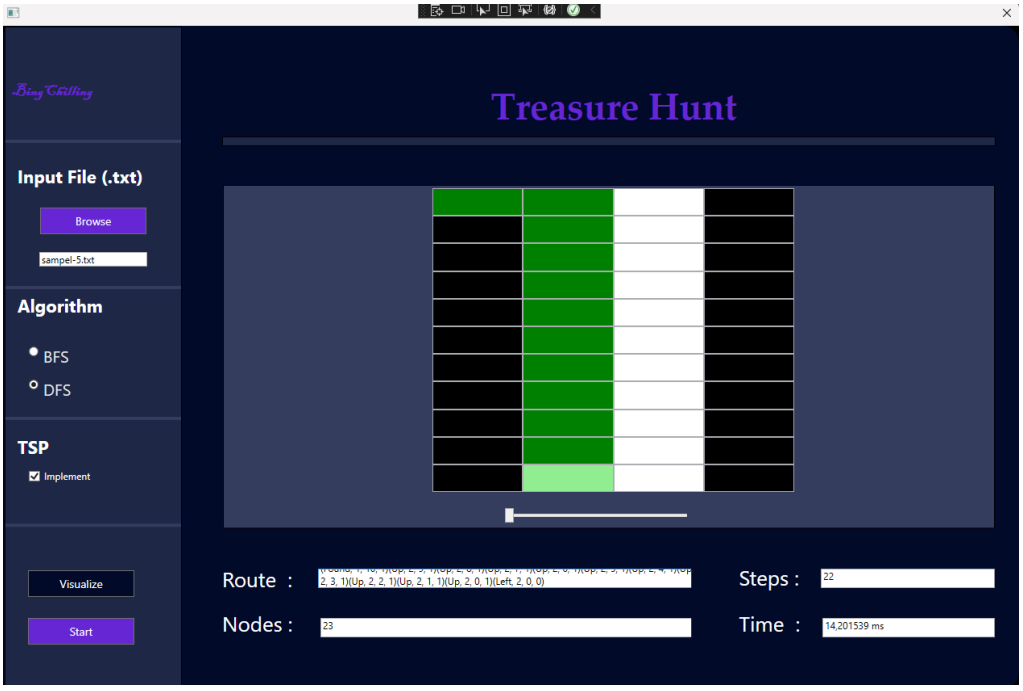


7. Test Case 4 - DFS + TSP



8. Test Case 5 - BFS + TSP

9. Test Case 5 - DFS + TSP



4.5. Analisis Desain Solusi

Algoritma BFS dan DFS dapat berkerja lebih baik satu sama lain tergantung dengan kasus yang diberikan. BFS merupakan penelusuran yang melebar, dimana program mengutamakan penelusuran pada level atau jarak yang sama terlebih dahulu hingga semua simpul pada jarak tersebut sudah dikunjungi. Karena itu BFS akan lebih optimal untuk mencari Treasure yang terdapat pada jarak yang dekat dari Krusty Krab. Sedangkan DFS akan lebih optimal jika Treasure berada pada jarak yang jauh dan tidak terdapat jalur yang sirkular dari Krusty Krab karena DFS mengutamakan untuk menelusuri maze secara mendalam, sehingga jika Treasure berada pada jarak yang dekat ada kemungkinan DFS akan menelusuri semua jalur padahal Treasure yang dituju berada pada jarak yang sangat dekat.

1. Kasus dimana Algoritma BFS lebih unggul
   Berdasarkan hasil test case 1 algoritma BFS melalui path sebagai berikut :

   (Right-Down-Right-Up-Right-Right-Right-Down-Down-Down-Left-Left-Up-Down-Left-Left-Up-Up-Up-Left-Left-Down-Down-Found)
   Dengan waktu 12.3 ms

   dan algoritma DFS melalui path sebagai berikut :

   (Right-Down-Down-Down-Right-Right-Right-Right-Up-Up-Up-Left-Left-Left-Right-Right-Right-Down-Down-Down-Left-Left-Up-Down-Left-Left-Up-Up-Up-Left-Left-Down-Down-Found)
   Dengan waktu 17.3 ms

   Pada kasus ini, desain solusi algoritma BFS lebih efektif dibandingkan solusi algoritma DFS. Jika diperhatikan, treasure yang ada lebih dekat dengan Krusty Krab. Sehingga, dapat disimpulkan bahwa penggunaan algoritma BFS menjadi lebih efektif

2. Kasus dimana Algoritma DFS lebih unggul
   Berdasarkan hasil test case 5 algoritma BFS + TSP melalui path sebagai berikut :

   (Right-Down-Down-Down-Down-Down-Down-Down-Down-Down-Down-Up-Up-Up-Up-Up-Up-Up-Up-Up-Up-Up-Left-Right-Down-Down-Down-Down-Down-Down-Down-Down-Down-Down-Down-Up-Up-Up-Up-Up-Up-Up-Up-Up-Up-Left-Found)
   Dengan waktu 21.3 ms

dan algoritma DFS + TSP melalui path sebagai berikut :

(Right-Down-Down-Down-Down-Down-Down-Down-Down-Down-Down-Up-Up-Up-Up-Up-Up-Up-Up-Up-Up-Up-Left)
Dengan waktu 14.2 ms

Pada kasus ini, desain solusi algoritma DFS lebih efektif dibandingkan solusi algoritma BFS. Jika diperhatikan,pencarian menggunakan DFS menggunakan memori yang lebih sedikit sehingga mempengaruhi waktu eksekusi.

BAB V
KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil pembuatan maze treasure hunt dapat disimpulkan bahwa struktur data graph dan matriks ketetanggaan dapat menggambarkan proses pencarian treasure dengan baik. Penggunaan matriks ketetanggaan dan graph dapat membantu mendata tile yang berada di sekitar posisi sekarang dan belum dilewati, serta membuat pencarian treasure lebih mudah. Proses pencarian ini yang nantinya terpecah menjadi algoritma BFS maupun DFS.

Metode pencarian dengan menggunakan BFS dan DFS dapat diterapkan pada pencarian treasure dari sebuah maze. Kedua algoritma dapat menemukan solusi dengan tepat dan membutuhkan waktu yang kurang lebih sama. Perbedaan dari kedua algoritma terlihat pada penelusuran node, sehingga jalur yang dilalui saat proses traversal berbeda. Algoritma BFS dan DFS memiliki keunggulan pada jenis maze tertentu. Algoritma BFS unggul ketika graph lebih lebar daripada mendalam. Sedangkan DFS unggul untuk graph yang tidak melebar, namun mendalam.

B. Saran

Terdapat beberapa saran yang dapat kami berikan pada pembaca agar pengerjaan tugas dapat lebih baik :

1. Melakukan eksplorasi lebih dalam lagi terkait cara kerja dan feature dari C# Desktop Application Development yang dipilih.
2. Membuat alur kerja yang jelas agar pembuatan GUI dan program utama dapat berjalan secara sinkron dengan baik
3. Melengkapi komentar pada kode program agar memudahkan untuk dipahami oleh anggota kelompok lain

BAB VI

LAMPIRAN

A. Pranala Github

https://github.com/mrsyaban/Tubes2_BingChilling

B. Daftar Pustaka

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf

http://euro.ecom.cmu.edu/people/faculty/mshamos/1976ShamosBentley.pdf

C. Link Youtube

bit.ly/BonusVideoTubesStima2