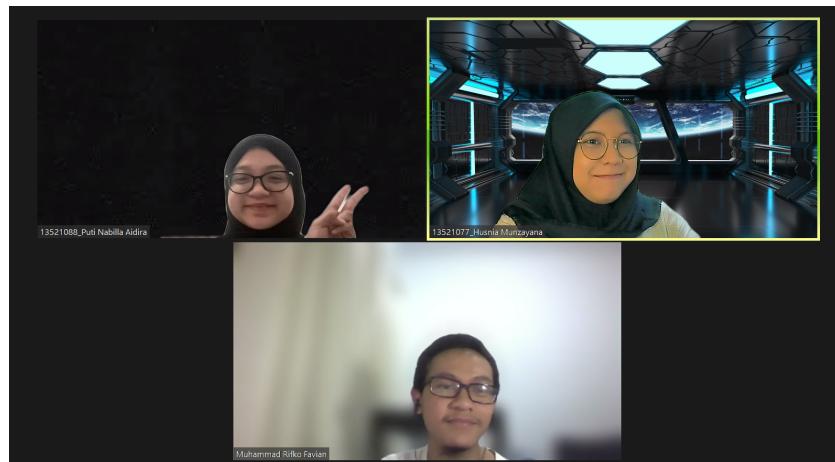


LAPORAN TUGAS BESAR II
IF2211 Strategi Algoritma Semester II Tahun 2022/2023
Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze
Treasure Hunt



Disusun Oleh:

Kelompok TubesHunting

Muhammad Rifko Favian 13521075

Husnia Munzayana 13521077

Puti Nabilla Aidira 13521088

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung
2022/2023

DAFTAR ISI

DAFTAR ISI.....	2
BAB 1 DESKRIPSI TUGAS.....	3
BAB 2 LANDASAN TEORI.....	6
2.1. Dasar Teori.....	6
2.1.1. Graf Traversal.....	6
2.1.2. Breadth-First-Search.....	6
2.1.3. Depth-First-Search.....	7
2.2. C# Desktop Application Development.....	8
BAB 3 PENJELASAN ALGORITMA.....	10
3.1. Langkah Pemecahan Masalah.....	10
3.2. Mapping Elemen BFS dan DFS.....	11
3.3. Kasus Lain.....	11
BAB 4 ANALISIS PEMECAHAN MASALAH.....	13
4.1. Implementasi Program.....	13
4.1.1. BFSAlgorithm.cs.....	13
4.1.2. DFSAlgorithm.cs.....	14
4.2. Struktur Data dan Spesifikasi Program.....	15
4.2.1. Struktur data yang digunakan:.....	15
4.2.2. Struktur Data Pada Kelas:.....	16
4.3. Tata Cara Penggunaan Program.....	18
4.4. Hasil Pengujian.....	22
4.4.1. sampel-1.txt.....	22
4.4.2. sampel-2.txt.....	23
4.4.3. sampel-3.txt.....	23
4.4.4. sampel-4.txt.....	24
4.4.5. sampel-5.txt.....	25
4.5. Analisis Desain Algoritma.....	25
BAB 5 KESIMPULAN DAN SARAN.....	27
5.1. Kesimpulan.....	27
5.2. Saran.....	27
5.3. Refleksi.....	27
5.4. Tanggapan.....	28
DAFTAR PUSTAKA.....	29
TAUTAN REPOSITORY.....	30
TAUTAN VIDEO DEMO.....	30

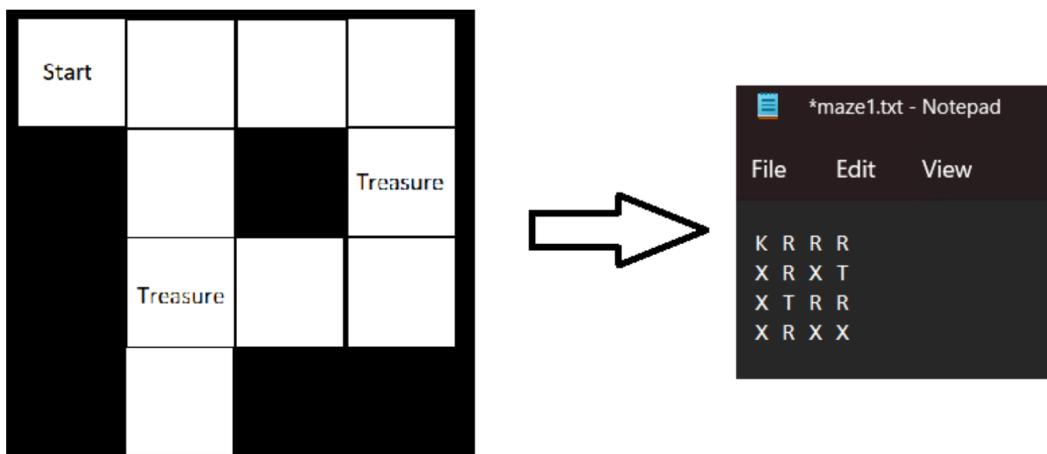
BAB 1

DESKRIPSI TUGAS

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

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

Contoh file input :

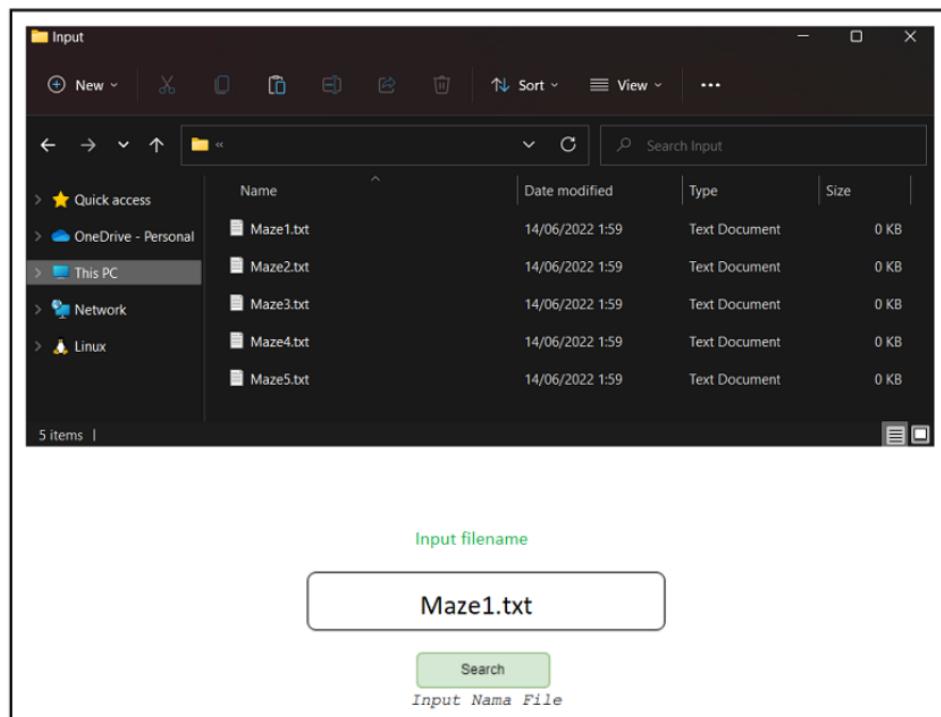


Gambar 1.1 Ilustrasi input file maze

Dengan memanfaatkan algoritma Breadth First Search (BFS) dan Depth First Search (DFS), anda dapat menelusuri grid (simpul) yang mungkin dikunjungi hingga ditemukan rute solusi, baik secara melebar ataupun mendalam bergantung alternatif algoritma yang dipilih. Rute solusi adalah rute yang memperoleh seluruh treasure pada maze. Perhatikan bahwa rute yang diperoleh dengan algoritma BFS dan DFS dapat berbeda, dan banyak langkah yang dibutuhkan pun menjadi berbeda. Prioritas arah simpul yang dibangkitkan dibebaskan asalkan ditulis di laporan ataupun readme, semisal LRUD (left right up down). Tidak ada pergerakan secara diagonal. Anda juga diminta untuk memvisualisasikan input txt tersebut menjadi suatu grid

maze serta hasil pencarian rute solusinya. Cara visualisasi grid dibebaskan, sebagai contoh dalam bentuk matriks yang ditampilkan dalam GUI dengan keterangan berupa teks atau warna. Pemilihan warna dan maknanya dibebaskan ke masing - masing kelompok, asalkan dijelaskan di readme / laporan.

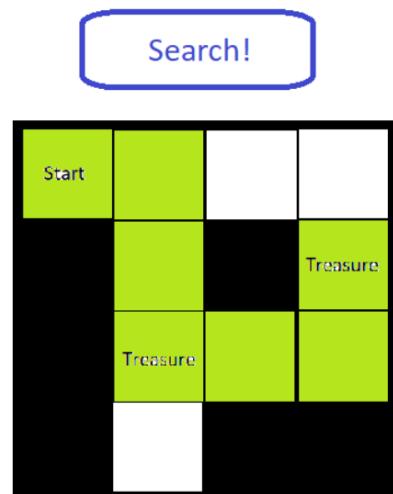
Contoh input aplikasi :



Gambar 1.2 Contoh input program

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

Contoh output Aplikasi :



Gambar 1.3 Contoh output program

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.

BAB 2

LANDASAN TEORI

2.1. Dasar Teori

2.1.1. Graf Traversal

Algoritma traversal graf adalah algoritma untuk mengunjungi simpul dengan cara sistematik. Dalam algoritma ini persoalan direpresentasikan dalam bentuk graf terhubung untuk kemudian pencarian solusi dilakukan dengan traversal graf. Algoritma proses pencarian dengan graf traversal dibagi menjadi dua :

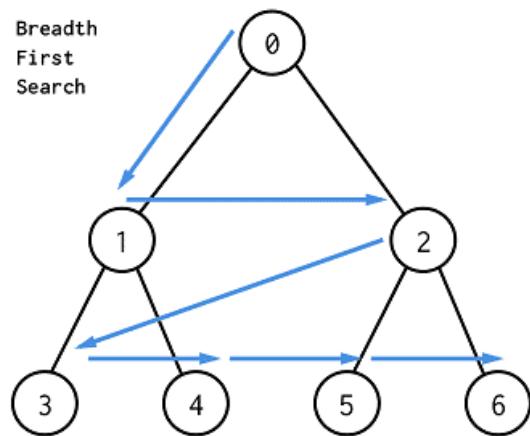
- a. Pencarian melebar (*breadth first search/BFS*)
- b. Pencarian mendalam (*depth first search/DFS*)

2.1.2. Breadth-First-Search

Breadth-First Search adalah algoritma pencarian solusi melebar secara traversal untuk melintasi atau mencari simpul tujuan atau goal node dari suatu graf atau tree. Pada algoritma ini, pencarian solusi dimulai dari suatu node untuk kemudian dilakukan pencarian secara bertahap setiap level pada graf hingga ditemukan simpul tujuan atau *goal node*. Pada setiap tahapan pencarian, simpul yang bertetangga dengan simpul yang saat ini dikunjungi menjadi prioritas simpul yang dikunjungi berikutnya. Algoritma atau langkah dalam pencarian traversal pada graf secara BFS dapat dirumuskan sebagai berikut:

1. Kunjungi sebuah simpul sebagai node awal, kita misalkan node yang sedang dikunjungi sebagai *currentNode*
2. Kunjungi semua simpul yang bertetangga dengan *currentNode* terlebih dahulu.
3. Setelah semua simpul tetangga telah dikunjungi, kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi.
4. Lakukan pengunjungan setiap tetangga hingga ditemukan simpul solusi atau *goal node*.

Sebagai ilustrasi, berikut adalah contoh traversal graf dengan algoritma Breadth-First-Search :



Gambar 2.1.2 Ilustrasi Breadth-First-Search (Sumber : www.freelancinggig.com)

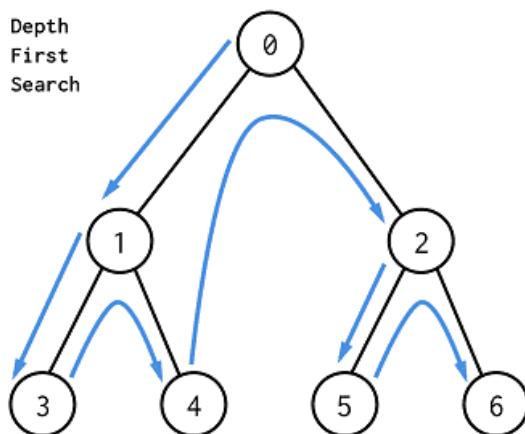
2.1.3. Depth-First-Search

Depth-First Search adalah algoritma pencarian solusi secara traversal mendalam untuk melintasi atau mencari simpul tujuan atau goal node dari suatu graf atau tree. Pada algoritma ini, pencarian solusi dimulai dari suatu node untuk kemudian dilakukan pencarian pada salah satu tetangga atau anak(*child*) hingga ditemukan simpul tujuan atau goal node. Pada setiap tahapan pencarian, simpul yang akan dikunjungi adalah simpul anak dari simpul yang saat ini dikunjungi hingga mencapai simpul daun. Akan tetapi, dalam pencarian solusi, tidak semua simpul daun merupakan simpul tujuan atau *goal node*, sehingga diperlukan proses runut balik atau *backtracking* untuk kembali ke simpul parent. Proses *backtracking* ini berakhir ketika ditemukan suatu simpul yang memiliki simpul anak lain yang belum dikunjungi untuk kemudian dilakukan pencarian mendalam pada simpul anak tersebut. Proses ini dilakukan berulang hingga setiap node telah dikunjungi atau telah dicapai simpul tujuan atau *goal node*. Algoritma atau langkah dalam pencarian traversal pada graf secara DFS dapat dirumuskan sebagai berikut:

1. Kunjungi sebuah simpul sebagai node awal, kita misalkan node yang sedang dikunjungi sebagai *currentNode*

2. Kunjungi sebuah simpul yang bertetangga dengan currentNode.
3. Ulang proses 2 hingga tidak ada lagi simpul tetangga yang belum dikunjungi.
4. Ketika mencapai sebuah simpul sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi namun belum ditemukan simpul tujuan atau *goal node*, dilakukan proses runut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul tetangga lain yang belum dikunjungi.
5. Ulangi proses tersebut hingga tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi atau simpul tujuan/*goal node* ditemukan.

Sebagai ilustrasi, berikut adalah contoh traversal graf dengan algoritma Depth-First-Search :



Gambar 2.1.2 Ilustrasi Depth-First-Search (Sumber : www.freelancinggig.com)

2.2. C# Desktop Application Development

C# Desktop Application Development adalah proses pengembangan aplikasi *desktop* menggunakan bahas pemrograman C#. Bahasa pemrograman C# (C sharp) adalah bahasa pemrograman yang dikembangkan oleh Microsoft. C# termasuk dalam bahasa pemrograman berorientasi objek, sehingga seperti bahasa dengan paradigma berorientasi objek lain, implementasi program dalam bahasa pemrograman ini juga menggunakan konsep *inheritance*, *class*, *polymorphism* dan *encapsulation*. Selain itu, bahasa C# juga bersifat *general-purpose* yang berarti dapat digunakan untuk pemrograman *server-side*

pada *website*, membangun aplikasi *desktop* maupun *mobile*, pemrograman game, dan sebagainya. Dengan menggunakan C# untuk pengembangan aplikasi *desktop*, *programmer* dapat lebih mudah dalam mengembangkan aplikasi yang *scalable*, fleksibel, dan efisien dengan *deployment-friendly environment*.

Bahasa C# dirancang untuk berjalan dalam *framework* .NET. Salah satu *framework* .NET yang tersedia adalah .NET MAUI (Multi-platform App UI). .NET MAUI merupakan *framework open-source* yang memungkinkan pengembangan aplikasi *multiplatform* (iOS, Android, Windows, MacOs) dengan menggunakan satu *source code* dan *development tools* yang sama.

Pengembangan aplikasi desktop menggunakan C# juga tidak lepas dari salah satu tools IDE yaitu Visual Studio. Visual studio adalah IDE yang populer dan *powerful* untuk pengembangan aplikasi Desktop C#. Visual studio menyediakan berbagai fitur dan alat untuk mendukung seluruh siklus pengembangan perangkat lunak, termasuk merancang, menulis kode, melakukan pengujian, dan *debugging* aplikasi. Beberapa fitur penting visual studio yang mendukung C# Desktop Application Development diantaranya: *integrated debugger*, *code analysis*, *code navigation*, *designer tools*, dan *deployment tools*.

BAB 3

PENJELASAN ALGORITMA

3.1. Langkah Pemecahan Masalah

Pada tugas besar ini, dibutuhkan suatu pencarian *multi-treasure* pada suatu maze yang berbentuk segi empat dengan algoritma BFS dan DFS dan dapat divisualisasikan dengan GUI. Maka dari itu, kelompok kami melakukan analisis pemecahan masalah dalam penggerjaan tugas besar ini untuk mendapat solusi yang sesuai. Analisis ini menghasilkan beberapa bagian/langkah.

Pertama kami membuat kelas MazeMap yang bertujuan untuk menyimpan bentuk dan isi maze yang akan digunakan untuk mencari treasure-nya. Dari MazeMap yang sudah disimpan, kami kemudian membuat GameState untuk menyimpan berapa banyak sisa treasure yang ada. Kami juga membuat kelas Explorer yang menyimpan titik node awal mulai, titik node saat ini, dan berapa banyak suatu node sudah dilewati. Dari kelas Explorer, kami membuat turunan kelas yaitu ExplorerState yang menyimpan rute yang dijalani oleh Explorer dan banyak node yang dilewati.

Dari kelas-kelas tersebut kemudian kami manfaatkan untuk membuat algoritma BFS dan DFS nya. Pada BFS kami menggunakan queue untuk mempermudah pencarian, visualisasi, dan agar tidak melakukan pencarian dua kali pada suatu node. Untuk DFS, kami menggunakan cara *hardcoded* pencarian traversal DFS biasa.

Terakhir adalah pembuatan GUI kami. GUI ini pertama akan menerima *input* file maze berupa txt, pemilihan algoritma antara BFS atau DFS, pemilihan apakah mau TSP atau tidak, maze position untuk menyesuaikan posisi maze sesuai dengan kenyamanan Anda, dan time interval untuk menentukan kecepatan antar step pencarian. Setelah semua pemilihan tinggal menekan tombol visualize untuk memvisualisasi step-step nya dan mengeluarkan output berupa route yang dilewati menuju treasure, steps yang diperlukan, banyak node yang dikunjungi dalam pencarian treasure, dan execution time yaitu waktu yang diperlukan untuk menjalankan algoritma.

3.2. Mapping Elemen BFS dan BFS

Setelah program menerima input peta nya berupa txt, maze akan dijadikan suatu matriks dalam kelas MazeMap. Matriks ini bisa diilustrasikan seperti sebuah graf. Di sini, simpulnya adalah node yang sedang dikunjungi saat ini dan sisi adalah semua arah simpul yang dapat dikunjungi. Kami membuat prioritas arah simpul nya adalah LDRU (left down right up). Jadi Setelah mapping, pencarian kemudian dilakukan sesuai dengan algoritma yang dipilih oleh user.

Pada BFS, pencarian dilakukan dengan sekaligus mencari rute yang terpendek untuk ke semua *goal*. Pencarian dimulai dengan mencari satu *treasure*. Semua arah simpul akan dicek apakah itu arah yang bisa dikunjungi atau tidak, kemudian dimasukkan ke queue apabila simpul dapat dikunjungi. Setelah itu akan dilakukan dequeue untuk mengecek apakah itu *treasure* atau bukan. Setelah ditemukan, pencarian selanjutnya dilakukan dengan membuat semua node yang sudah diakses menjadi dapat diakses kembali, atau bisa disebut mengulang *state*. Hal ini dilakukan sampai semua *goal* telah tercapai, atau semua node sudah dikunjungi namun semua *goal* belum tercapai (harusnya tidak mungkin karena semua *goal accessible*).

Kemudian pada DFS, pencarian dilakukan seperti traversal DFS pada umumnya. Setelah satu *treasure* ditemukan, ia akan melanjut mengunjungi simpul yang menjadi prioritas tertinggi. Apabila semua simpul tetangga sudah dikunjungi, ia baru akan *backtrack* sampai terdapat simpul tetangga yang belum dikunjungi. Proses ini dilakukan sampai semua *goal* telah tercapai, atau semua node sudah dikunjungi.

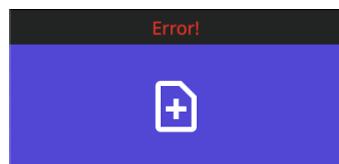
3.3. Kasus Lain

Beberapa ilustrasi kasus lain sebagai berikut:

- Map maze yang diinput oleh user dalam txt harus berbentuk segi empat, maupun persegi atau persegi panjang. Dan di dalam isi txt tersebut huruf yang diperbolehkan hanya ‘K’, ‘T’, ‘R’, dan ‘X’; di mana jumlah K wajib hanya 1 dan jumlah T minimal 1. Apabila terdapat kesalahan dalam isi txt maka program akan mengeluarkan tulisan error (Gambar 3.3.2).



Gambar 3.3.1 Contoh file input map maze yang tidak sesuai format.



Gambar 3.3.2 Output label error

- Map maze yang diinput oleh user harus memungkinkan *treasure* tercapai dari titik *start*. Jika tidak sesuai, tidak dilakukan penanganan exception karena pada spesifikasi tugas dijamin *treasure* selalu dapat tercapai dari *start*. Karena tidak dilakukan penanganan, program akan *crash* ketika StackOverflow Exception dilemparkan (Gambar 3.3.4).



Gambar 3.3.3 Contoh file input *map maze* yang tidak memungkinkan treasure tercapai dari titik start.



Gambar 3.3.4 Notifikasi StackOverflow Exception pada Visual Studio saat aplikasi crash.

BAB 4

ANALISIS PEMECAHAN MASALAH

4.1. Implementasi Program

4.1.1. BFSalgorithm.cs

```
procedure breadthFirstSearch(pos:Position, maze:MazeMap, game:GameState)
queue <- new
queue.Enqueue(pos, getRouteNow())
visited[pos]++
onQueue[pos] <- true

while(queue.Count > 0) do
    current <- queue.Dequeue()
    setCurrentPos(current.pos)
    setRoute(current.route)
    visited[current.pos]++

    if current.pos = treasure then
        maze.changeMazeElementToRoad(current.pos)
        game.setTreasureCount(game.treasureCount-1)
        break
    endif

    for tiap semua arah simpul do
        if (not canBeVisited) or (onQueue) then
            continue
        endif
        current.route.Add(arah saat ini)
        onQueue[currentPos] <- true
        queue.Enqueue(currentPos, routeAfterAdded)
    endfor
endwhile

if (game.treasureCount > 0) then
    breadthFirstSearch(current.pos, maze, game)
endif

procedure runTSPforBFS(pos:Position, maze:MazeMap, game:GameState)
maze.setFirstPosToTreasure()
maze.setCurrentPosToFirstPos(pos)
game.setTreasureCount(1)
breadthFirstSearch(pos,maze,game)
```

4.1.2. DFSalgorithm.cs

```
procedure depthFirstSearch(pos:Position, maze:MazeMap, game:GameState)
    if game.treasureCount > 0 then
        if maze.getMapElement(pos) = treasure then
            maze.changeMazeElementToRoad(pos)
            game.setTreasureCount(game.treasureCount-1)
            if (game.treasureCount = 0) then
                visited[pos]++
                return
            endif
        endif

        if AllSimpulTetanggaVisited then
            startBackTrack()
        else
            if onBackTrack then
                stopBackTrack()
                addRoute(backTrackRoute)
                clearBackTrackRoute()
            endif

            if (isLeftVisitable()) then goToLeft();
            else if (isDownVisitable()) then goToDown();
            else if (isRightVisitable()) then goToRight();
            else if (isUpVisitable()) then goToUp();
            endif
            visited[currentPos]++
        endif

        if onBackTrack then
            addBackTrackRoute(getLastRoute())
            removeLastRoute()
        endif

        breadthFirstSearch(currentPos(), maze, game)
    endif

procedure TSPSetupDFS(currentPos:Position, maze:MazeMap, game:GameState)
    maze.setFirstPosToTreasure()
    maze.setCurrentPosToFirstPos(currentPos)
    game.setTreasureCount(1)
    setInitVisitedMap(maze)
```

4.2. Struktur Data dan Spesifikasi Program

4.2.1. Struktur data yang digunakan:

Nama Struktur Data	Keterangan
Array2D	Struktur data Array2D digunakan sebagai representasi <i>map treasure</i> . Sehingga, atribut/variabel yang berkaitan dengan mengakses, menelusuri, dan menampilkan elemen-elemen <i>map</i> juga menggunakan struktur data Array2D.
List	Struktur data List digunakan sebagai representasi rute dan urutan nodes pada algoritma pencarian. List digunakan pada pencatatan koordinat rute, koordinat <i>visited nodes</i> , rute arah gerak, dan rute backtrack pada DFS.
String	Struktur data String digunakan dalam penanganan input file <i>map treasure</i> dan <i>user interface text</i> lainnya.
Tuple	Struktur data Tuple digunakan sebagai representasi koordinat untuk pencatatan koordinat rute dan koordinat <i>visited nodes</i> (pencatatan dilakukan untuk keperluan visualisasi output).
Queue	Queue digunakan sebagai representasi koordinat rute khusus untuk algoritma BFS.

4.2.2. Struktur Data Pada Kelas:

Nama Kelas	Struktur Atribut dan Method Kelas
ExplorerState.Explorer	<pre> Explorer # currentPos # firstPos # visited # coorVisited + Explorer() + setCurrentPosition() + getCurrentPosition() + setFirstPosition() + getFirstPosition() + setCoorVisited() + getCoorVisited() + setVisitedMap() + getVisitedMap() + setInitVisitedMap() + printVisitedMap() + printCoorVisited() + Visit() </pre>
ExplorerState.ExplorerAction	<pre> ExplorerState.ExplorerAction # route # coorRoute # nodes + ExplorerAction() + setRoute() + getRoute() + setCoorRoute() + getCoorRoute() + addCoorRoute() + printCoorRoute() + setNodes() + getNodes() + countNodes() + printRoute() + toStringRoute() + goToUp() + goToDown() + goToRight() + goToLeft() + isAllAdjVisited() + isUpVisitable() + isDownVisitable() + isLeftVisitable() + isRightVisitable() + setCurrentAction() </pre>

DFSalgorithm.DFS	<pre> DFSalgorithm.DFS - back - backRoute + DFS() + getBackTrackRoute() + getBackTrackRoute() + startBackTrack() + stopBackTrack() + isBackTrack() + printBackRoute() + backTrack() + setCurrentAction() + depthFirstSearch() + TSPSetupDFS() </pre>
BFSalgorithm.BFS	<pre> BFSalgorithm.BFS + BFS() + setCurrentAction() + breadthFirstSearch() + runTSPforBFS() </pre>
Game.GameState	<pre> Game.GameState + ROAD + OBSTACLES + TREASURE_PLACE - treasureCount + GameState() + setTreasureCount() + getTreasureCount() </pre>

MazeMap.Maze	<pre> MazeMap.Maze + mapMatrix - rows - cols - countK - countT - countBlank + Maze() + Maze() + setRows() + setCols() + getRows() + getCols() + setMapMatrix() + getMapMatrix() + setMapElement() + getMapElement() + printMap() + validation() </pre>
MazeMap.MazeException	- msg()

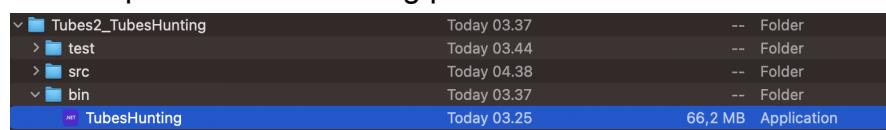
Spesifikasi program:

Program dikembangkan dengan .NET Multi-platform App UI (MAUI) dengan target framework .NET versi 7.0. Saat ini, program kompatibel dengan sistem operasi MacOS.

4.3. Tata Cara Penggunaan Program

Cara menggunakan aplikasi TubesHunting adalah sebagai berikut:

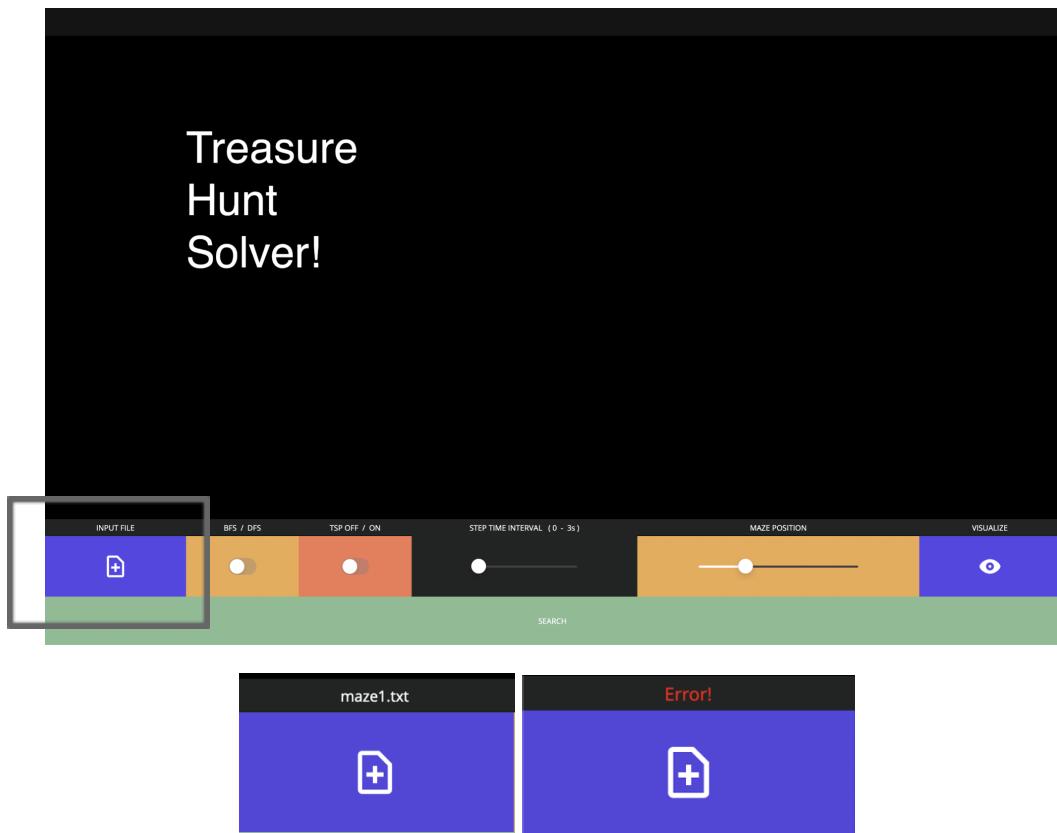
1. Jalankan aplikasi TubesHunting pada folder bin



Gambar 4.3.1 Aplikasi TubesHunting pada folder bin

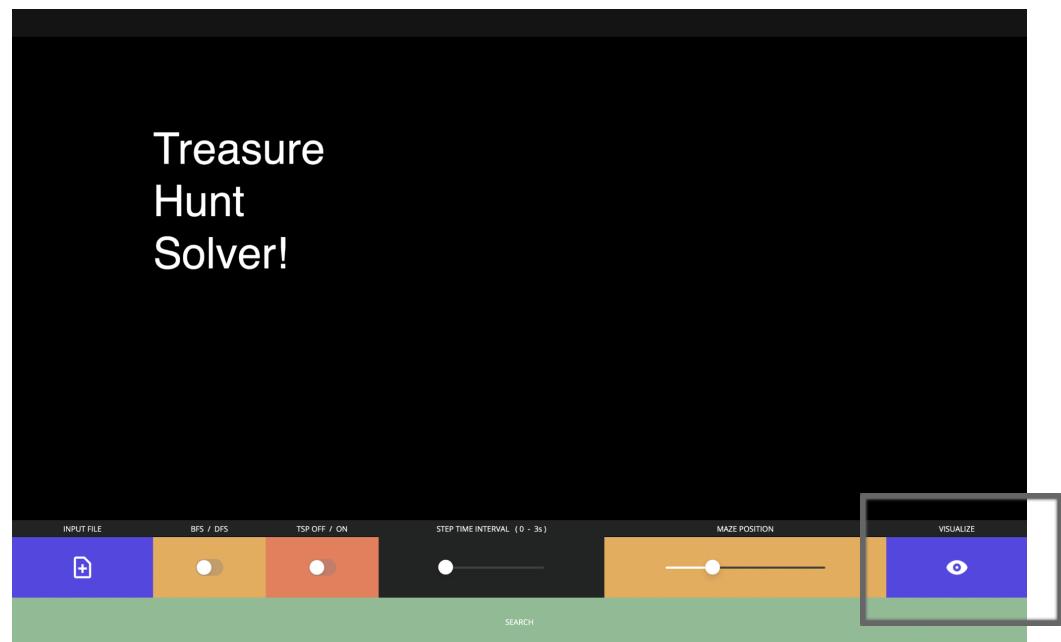
2. Masukan file txt pada tombol “INPUT FILE”

Jika masukan sesuai (tidak melanggar *constraint map*) nama file akan tertulis menggantikan text “INPUT FILE”, sebaliknya jika masukan tidak sesuai akan muncul text “Error!”.



Gambar 4.3.2 Tombol “INPUT FILE” sebelum dan sesudah masukan file

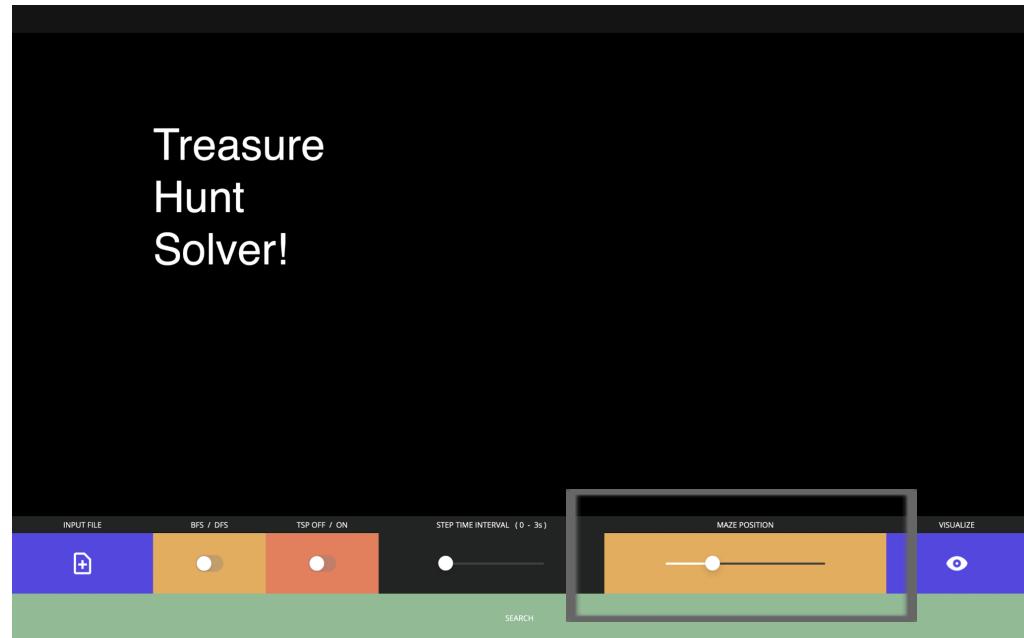
3. Untuk melakukan visualisasi tekan tombol “VISUALIZE”.



Gambar 4.3.3 Tombol “VISUALIZE”

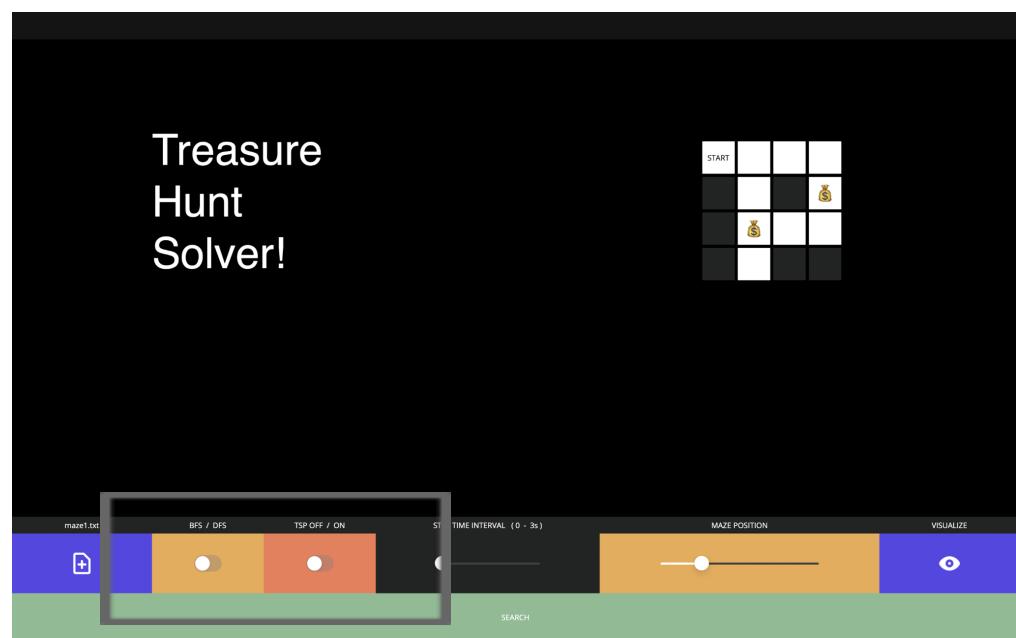
4. Posisi *map* pada visualisasi dapat diatur melalui *slider* “MAZE POSITION”

Posisi *map* yang diatur adalah *margin map*, sehingga pergerakan posisi akan bersifat diagonal.



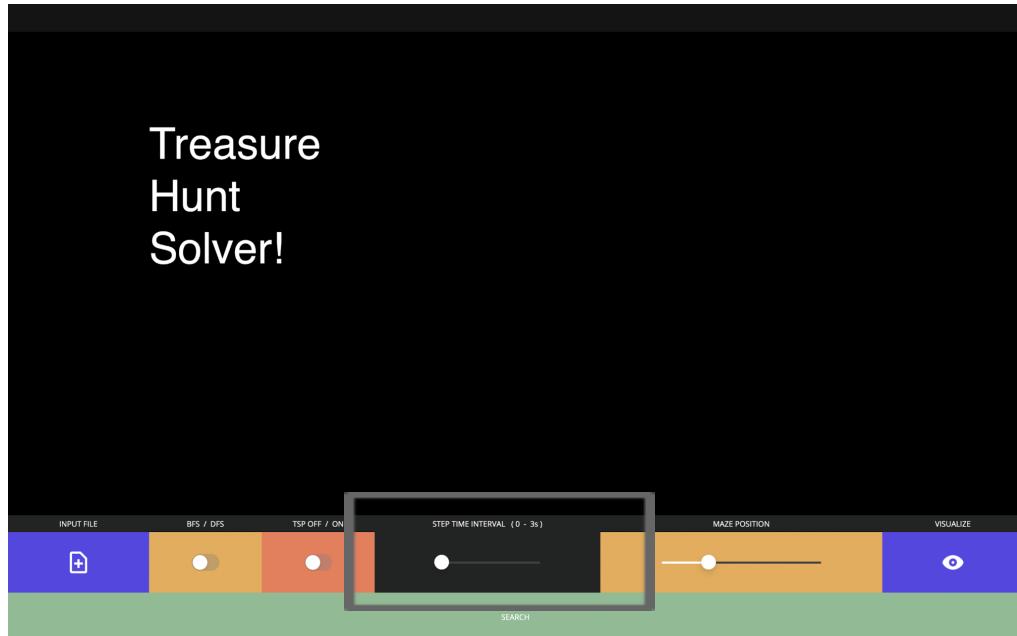
Gambar 4.3.4 *Slider* “MAZE POSITION”

5. Atur pilihan algoritma pada *toggle* “BFS/DFS” dan “TSP OFF/ON”



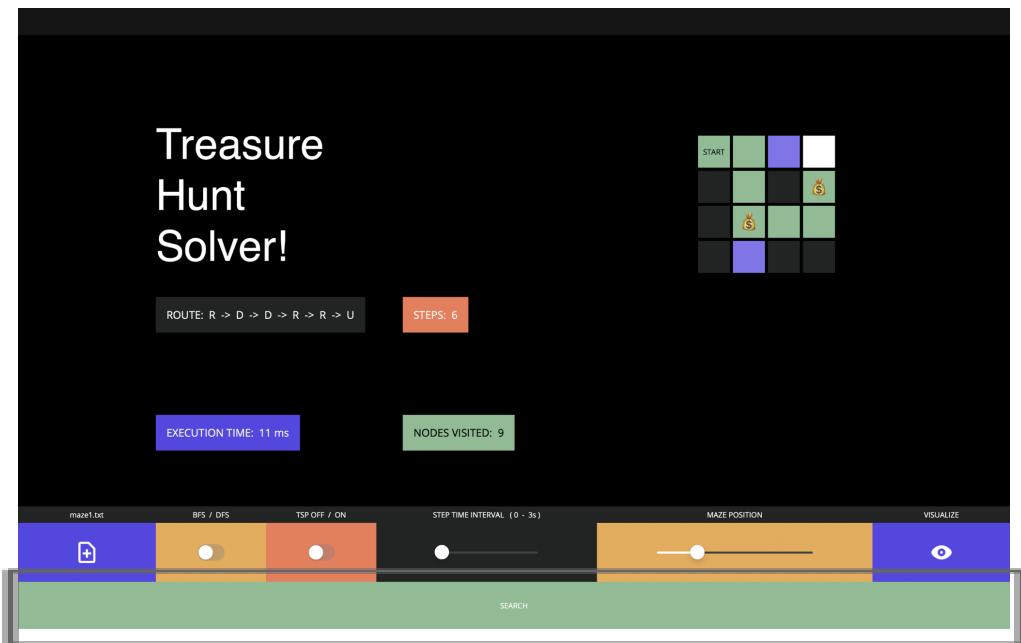
Gambar 4.3.5 Tombol pengaturan algoritma

- Atur interval waktu untuk memperlihatkan proses pencarian dengan slider “STEP TIME INTERVAL”.
Rentang waktu adalah dari 0 sampai 3 detik. Interval waktu dapat diubah secara dinamis pada saat pencarian berlangsung.



Gambar 4.3.6 Slider “STEP TIME INTERVAL”

- Tekan tombol “SEARCH” untuk memulai pencarian.
Setelah tombol “SEARCH” ditekan, hasil pencarian akan ditampilkan. Pada pencarian dengan interval waktu, warna biru menunjukkan *node* yang sudah diperiksa (semakin gelap semakin sering diperiksa berulang) sedangkan warna kuning menunjukkan *node* yang sedang diperiksa. Pada akhir pencarian, warna hijau menunjukkan rute hasil pencarian. Setelah pencarian dilakukan, tombol “SEARCH” akan *disabled*. Untuk melakukan pencarian kembali, ulangi langkah dari poin 3.

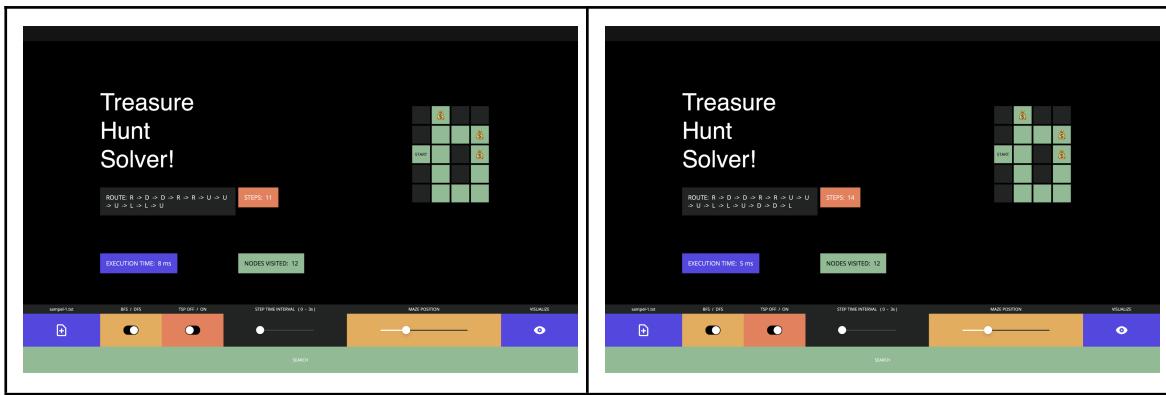


Gambar 4.3.7 *User Interface* program state setelah tombol “SEARCH” ditekan

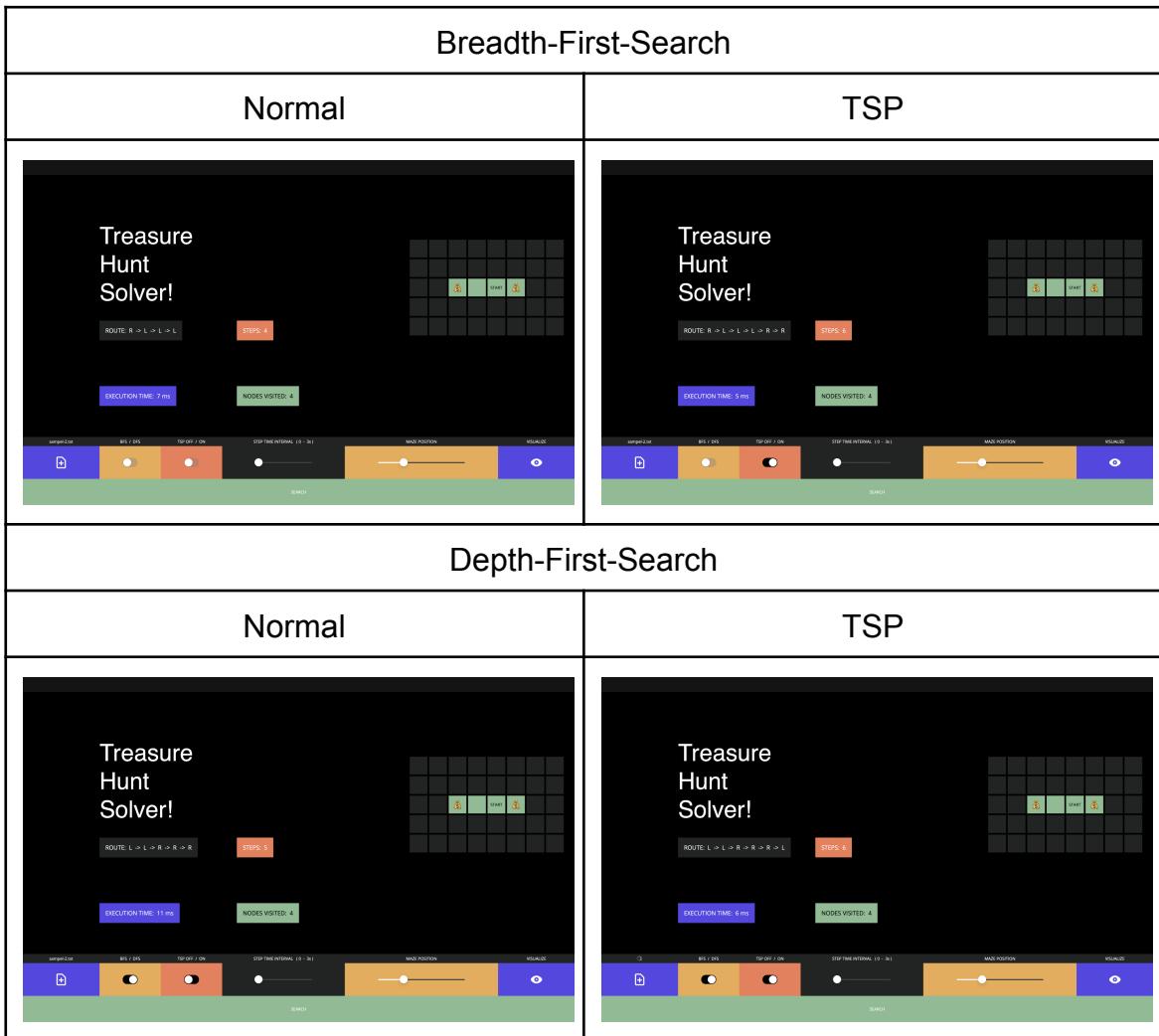
4.4. Hasil Pengujian

4.4.1. sampel-1.txt

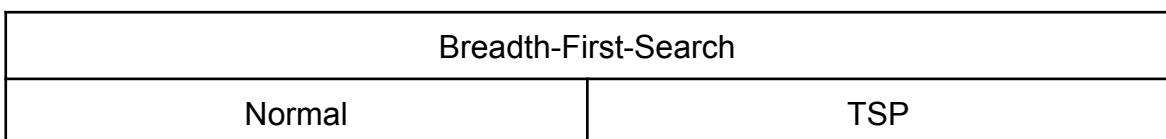
Breadth-First-Search	
Normal	TSP
<p>Treasure Hunt Solver!</p> <p>ROUTE: R -> U -> U -> D -> R -> R -> D STEPS: 7</p> <p>EXECUTION TIME: 7 ms NODES VISITED: 9</p> <p>maze1.txt BFS / DFS TSP OFF / ON STEP TIME INTERVAL (0 - 3s) MAZE POSITION VISUALIZE</p> <p>SEARCH</p>	<p>Treasure Hunt Solver!</p> <p>ROUTE: R -> U -> U -> D -> R -> R -> D STEPS: 12</p> <p>EXECUTION TIME: 2 ms NODES VISITED: 12</p> <p>maze1.txt BFS / DFS TSP OFF / ON STEP TIME INTERVAL (0 - 3s) MAZE POSITION VISUALIZE</p> <p>SEARCH</p>
Depth-First-Search	
Normal	TSP

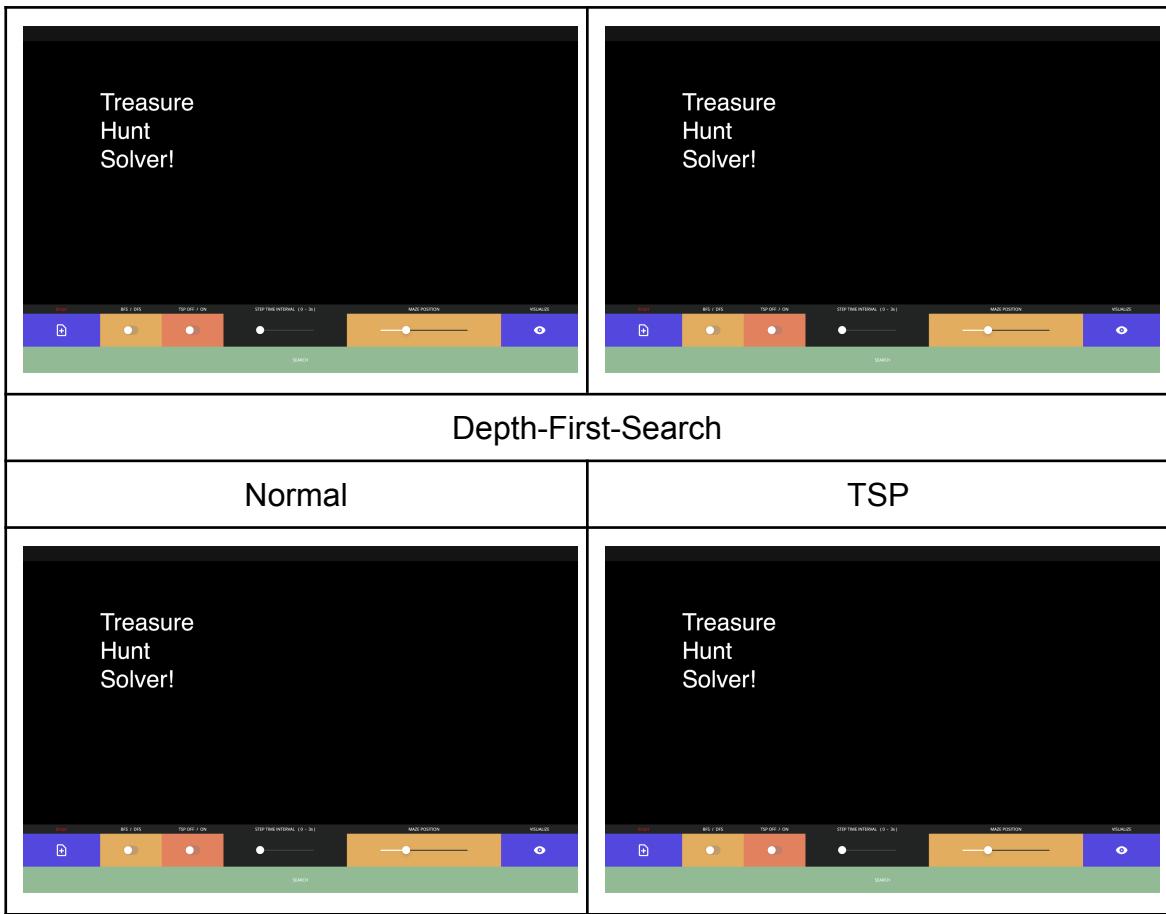


4.4.2. sampel-2.txt

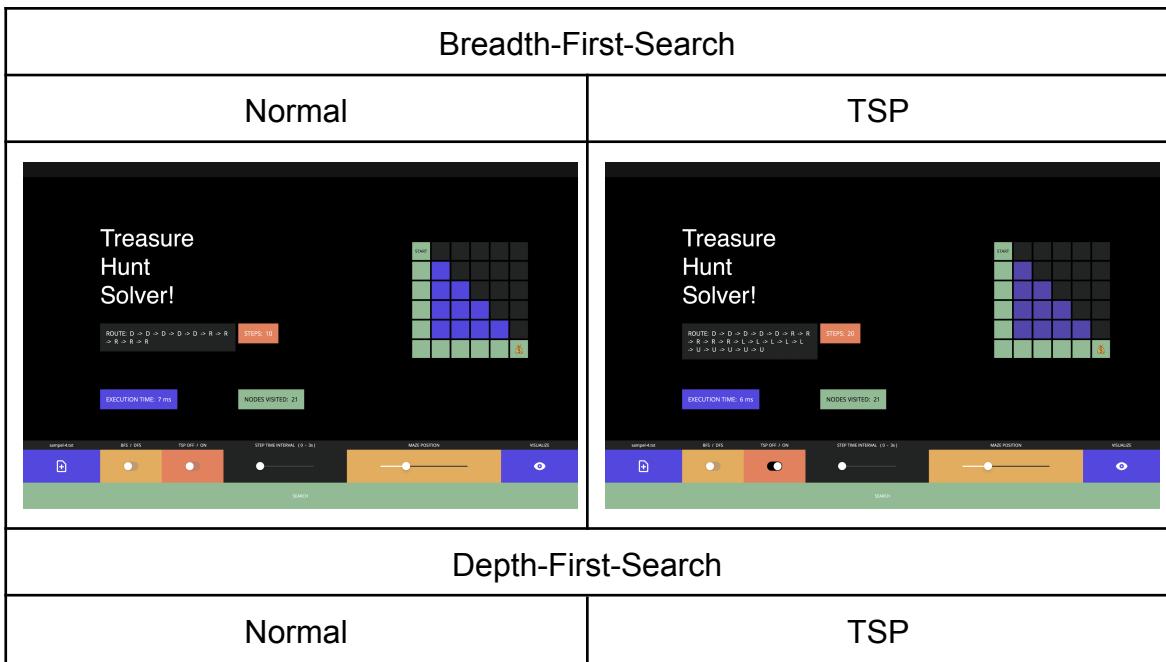


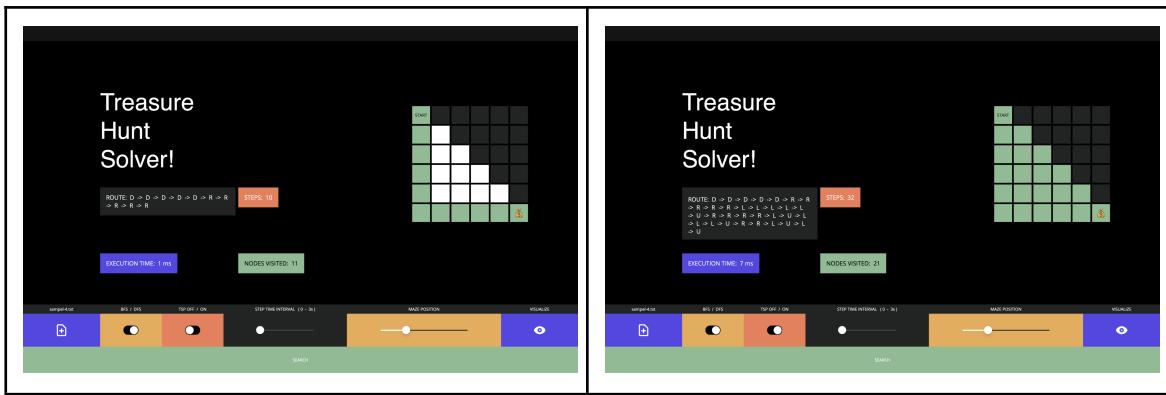
4.4.3. sampel-3.txt



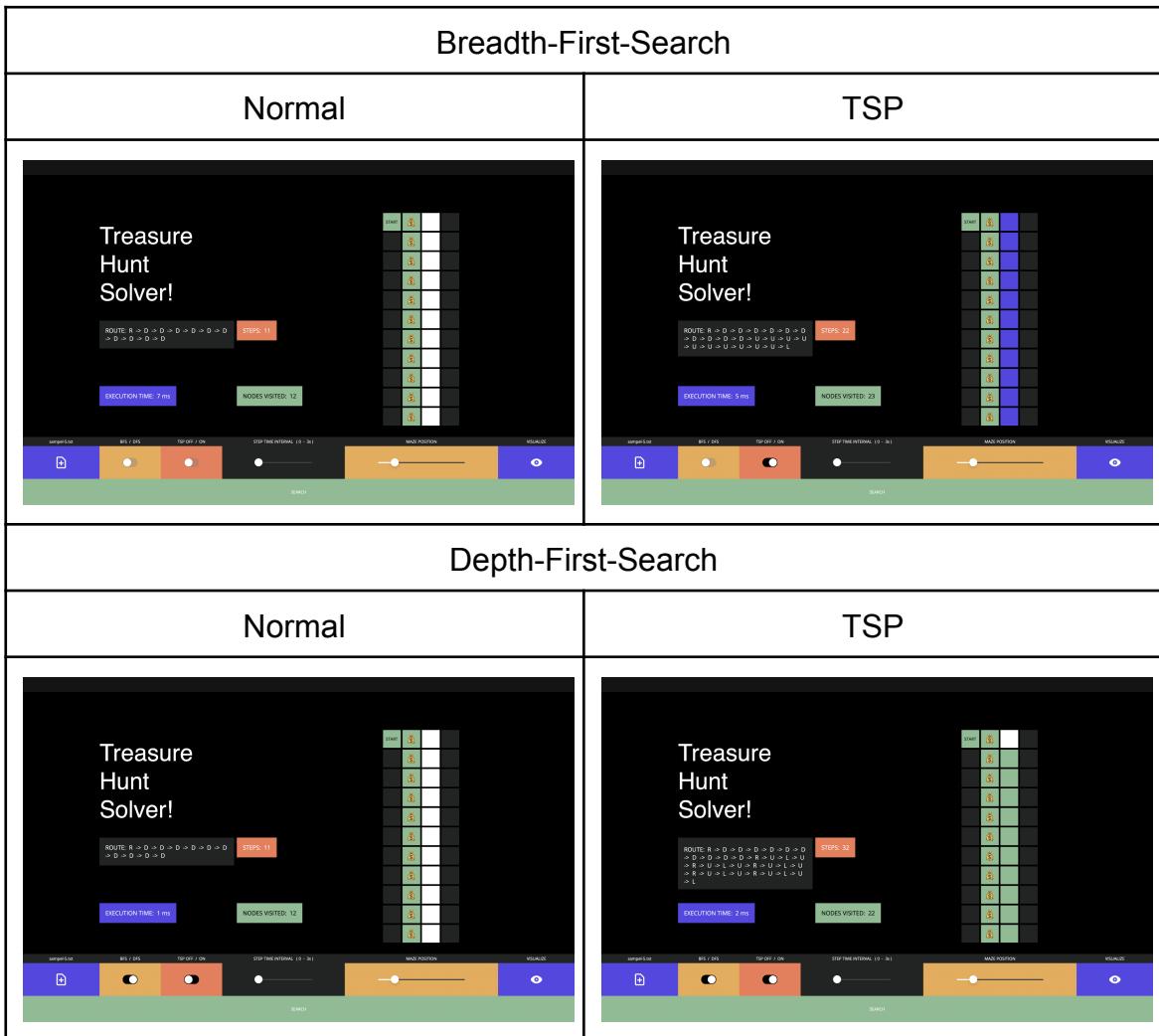


4.4.4. sampel-4.txt





4.4.5. sampel-5.txt



4.5. Analisis Desain Algoritma

Algoritma BFS diimplementasikan dengan menelusuri semua simpul dalam jarak yang sama dari simpul awal terlebih dahulu sebelum melanjutkan ke simpul yang lebih jauh. Sedangkan, algoritma DFS diimplementasikan dengan menelusuri

simpul secara rekursif dari simpul awal sampai tidak ada lagi simpul yang dapat dikunjungi lalu melakukan *backtracking* sampai terdapat simpul yang bisa dikunjungi. Karena perbedaan ini, terdapat perbedaan efisiensi algoritma pada kasus-kasus ekstrim tertentu.

Secara umum, algoritma BFS bekerja lebih baik pada graf (map) yang melebar dengan kedalaman yang tidak terlalu besar. Dengan kata lain, BFS bekerja lebih baik jika target (*treasure*) lebih dekat dengan titik mulai dari segi kedalaman. Contohnya pada graf sampel-1, terlihat bahwa lebih efisien untuk menelusuri nodes-nodes berjarak sama (dari titik *start*) terlebih dahulu daripada menelusuri kedalaman. Hal ini karena, *treasure-treasure* terletak pada kedalaman yang kurang lebih sama.

Sedangkan, algoritma DFS bekerja lebih baik pada graf (map) yang mampang (mendalam) dengan lebar yang tidak terlalu besar. Contohnya pada garf sampel-4, terlihat bahwa lebih efisien untuk menelusuri nodes terdalam terlebih dahulu daripada menelusuri nodes-nodes berjarak sama (dari titik start). Hal ini karena, satu *treasure* terletak pada kedalaman yang paling dalam.

Meskipun begitu, efisiensi algoritma BFS dan DFS tidak hanya bergantung pada bentuk graf melainkan penempatan treasure untuk kasus-kasus ekstrim. Contohnya pada sampel-5, terlihat bahwa graf berbentuk memanjang (mendalam). Namun, DFS pada sampel ini sama efisennya dengan BFS karena *treasure* ada pada setiap level kedalaman. Selain itu, prioritas arah gerak juga menjadi penentu efisiensi algoritma.

Secara umum, pada kasus-kasus normal, BFS dan DFS memiliki efisiensi yang sama. Begitu pula dengan akurasi, BFS dan DFS sama-sama akurat dalam menentukan rute menuju target.

BAB 5

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil pembuatan program TubesHunting ini, dapat disimpulkan bahwa algoritma BFS dan DFS dapat digunakan sebagai algoritma pencarian rute untuk semua treasure serta pencarian rute TSP yang melewati semua treasure. Pencarian rute dengan algoritma BFS dan DFS sama-sama menghasilkan rute yang akurat dengan waktu yang kurang lebih sama.

Perbedaan antara kedua algoritma ini terlihat pada proses penelusuran node, yaitu urutan pencarian node yang berbeda. Karena perbedaan urutan pencarian ini, rute akhir yang dihasilkan berbeda pula. Rute DFS akan mengandung *backtracking* sehingga rute pencarian dan rute akhir sama. Sedangkan rute BFS tidak mengandung *backtracking*, melainkan mengandung proses mematikan simpul sehingga rute pencarian dan rute akhir berbeda.

Secara umum, BFS lebih efektif dalam menangani graf dengan lebar yang besar namun tidak terlalu dalam. Sementara itu, DFS lebih unggul dalam menangani graf dengan kedalaman yang besar namun tidak terlalu lebar.

5.2. Saran

Setelah melakukan pengerojan Tugas Besar ini, kami ingin menyampaikan saran kepada pihak asisten agar dapat memperjelas spesifikasi, karena terdapat beberapa spesifikasi yang rancu. Kami juga menyarankan agar asisten dapat mengadakan asistensi selama pengerojan tugas agar dapat lebih membantu pengerojan.

5.3. Refleksi

Setelah melakukan pengerojan Tugas Besar ini, refleksi yang dapat kami lakukan adalah terkait proses sinkronisasi antara pengerojan GUI dan algoritma pencarian yang masih buruk. Hal ini karena, algoritma pencarian sebelumnya dirancang untuk CLI sebelum template folder GUI dikerjakan. Sehingga, proses sinkronisasi folder dan repositori git menimbulkan beberapa masalah. Terdapat

pula beberapa masalah karena perbedaan sistem operasi yang digunakan dalam pengembangan GUI .NET MAUI. Kedepannya kami berharap dapat mengorganisasi penggerjaan tugas dengan lebih baik.

5.4. Tanggapan

Tugas Besar ini cukup efektif untuk memperluas pemahaman kami terkait algoritma BFS dan DFS. Selain itu, Tugas Besar ini juga menambah wawasan terkait C# desktop development. Akan tetapi, dibutuhkan waktu lebih untuk melakukan eksplorasi terkait C# desktop *development* ini, terlebih terkait variasi-variasi pemilihan *framework* serta kompatibilitas dan variasi tata cara *build* aplikasi untuk sistem operasi yang berbeda.

DAFTAR PUSTAKA

- Filus, T. (2017, January 18). Pengenalan Bahasa Pemrograman C#. codepolitan.com. Retrieved March 23, 2023, from <https://codepolitan.com/blog/pengenalan-bahasa-pemrograman-c-587effa1cb95b>
- Gadhavi, M. (2023, FebruBahasa Pemrograman C#: Apa Itu, Contoh, dan Mengapa Ia Pentingary 16). *Why is .NET the Best Desktop Application Development Technology?* Radixweb. Retrieved March 23, 2023, from <https://radixweb.com/blog/net-preferred-desktop-application-development-technology>
- Hidayati, K. F. (2023, March 14). Bahasa Pemrograman C#: Apa Itu, Contoh, dan Mengapa Ia Penting. Glints Blog. Retrieved March 23, 2023, from <https://glints.com/id/lowongan/bahasa-pemrograman-c-sharp/#.ZB2-XXZBy3>
- Munir, Rinaldi. (2021). Algoritma Greedy Bagian 1. [https://informatika.stei.itb.ac.id.](https://informatika.stei.itb.ac.id/) Diakses pada 14 February 2023 pukul 18.32, dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy -\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy -(2021)-Bag1.pdf)
- Munir, Rinaldi, & Maulidevi, Nur Ulfa. (2021). Breadth/Depth First Search Bagian 1. [https://informatika.stei.itb.ac.id/.](https://informatika.stei.itb.ac.id/) Retrieved March 21, 2023, from <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf>
- Munir, Rinaldi, & Maulidevi, Nur Ulfa. (2021). Breadth/Depth First Search Bagian 2. [https://informatika.stei.itb.ac.id/.](https://informatika.stei.itb.ac.id/) Retrieved March 21, 2023, from <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>
- Patel, T. (2017, August 18). *What is C# Desktop Application Development?* Blog Concetto Labs. Retrieved March 23, 2023, from <https://www.concettolabs.com/blog/c-desktop-application-development/>

TAUTAN REPOSITORY

https://github.com/munzayanahusn/Tubes2_TubesHunting.git

TAUTAN VIDEO DEMO

<https://youtu.be/MM30Xk74Dzc>