Laporan Tugas Kecil 3
IF 2211 Strategi Algoritma
Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound

Disusun Oleh:
13520163 Frederik Imanuel Louis

Sekolah Tinggi Elektro dan Informatika
Institut Teknologi Bandung
2022

Daftar Isi

Daftar Isi 2
BAB 1 Algoritma Program 3
1.1 File main.py (Interface tkinter) 3
1.2 File puzzlesolver.py (Solver 15-Puzzle) 3
BAB II Source Code Program 5
2.1 Program Puzzle Solver 5
2.2 Program Utama 7
BAB III Pengujian Progam 13
3.1 Puzzle yang dapat disolve 13
3.1.1 Pengujian Puzzle 1 13
3.1.2 Pengujian Puzzle 2 15
3.1.2 Pengujian Puzzle 3 17
3.2 Puzzle yang tidak dapat diselesaikan 19
3.2.1 Pengujian puzzle 4 20
3.2.2 Pengujian puzzle 5 21
Lampiran 22

BAB 1
Algoritma Program

1.1 File main.py (Interface tkinter)

Program main berisi berbagai definisi widget tkinter yang diperlukan untuk menampilkan *graphical user interface* (GUI) dari program 15-puzzle solver. User akan diminta untuk memilih file config dari device user, dan puzzle akan di-load sesuai dengan masukan konfigurasi tersebut. Kemudian, setelah user menekan tombol solve, akan dipanggil fungsi untuk menyelesaikan 15-puzzle dari kelas puzzlesolver yang didefinisikan di file puzzlesolver.py. Jika puzzle berhasil diselesaikan, maka akan muncul tombol animate dan reset yang berguna untuk memperlihatkan lanngkah dari puzzle awal ke simpul akhir solusi. Selain itu, akan muncul pula tombol details yang akan menampilkan rincian eksekusi program yang diminta pada spek tugas besar.

1.2 File puzzlesolver.py (Solver 15-Puzzle)

Program puzzlesolver berisi kelas puzzlesolver dengan rincian atribut sebagai berikut:

1. dxy: const list yang menyimpan arah gerakan yang mungkin (kiri, kanan, atas, bawah)
2. puzzle: menyimpan puzzle yang diload dari file config
3. mincost: menyimpan banyak langkah yang diperlukan untuk menyelesaikan puzzle
4. answer: menyimpan urutan gerakan yang diperlukan untuk menyelesaikan puzzle
5. kurang: menyimpan nilai fungsi kurang tiap masukan untuk keperluan output
6. visited: menyimpan simpul-simpul yang sudah pernah dikunjungi
7. X: menyimpan nilai X awal untuk keperluan output
8. total: menyimpan banyaknya simpul yang dibangkitkan untuk keperluan output

Selain itu, puzzlesolver berisi beberapa method dengan rincian sebagai berikut:

1. Konstruktor: menerima parameter filename, yaitu path file config yang akan diload, dan memanggil method getPuzzle dengan parameter yang sama.
2. getPuzzle: menerima parameter filename, yaitu path file config yang akan diload, dan memasukkan puzzle pada file konfig pada attirbut puzzle jika file konfig valid. Sebaliknya, ia tidak akan mengubah isi attribut puzzle.
3. calcKurang: menghitung jumlah nilai fungsi kurang puzzle dan nilai X awal, sesuai spesifikasi pada slide kuliah. Method ini juga mengubah isi atribut kurang dan X
4. calcCost: menghitung banyak tile pada puzzle yang tidak ada pada posisi yang seharusnya. Nilai ini akan digunakan sebagai fungsi cost pada branch and bound.
5. getZeroPos: mengeluarkan koordinat tile kosong pada puzzle
6. validIndex: mengecek apakah koordinat masukan merupakan koordinat yang valid untuk puzzle dengan ukuran 4 x 4.
7. solve: fungsi utama untuk menyelesaikan puzzle, dengna keluaran jumlah fungsi kurang dan X awal. Fungsi solve pertama-tama mengecek apakah jumlah fungsi kurang dan X awal adalah genap. Jika tidak, fungsi akan langsung mengembalikan nilai tersebut. Kemuadian, fungsi akan membuat minimum heap yang menyimpan informasi semua simpul, yaitu cost dari simpul hidup (calcCost + depth), depth simpul, kondisi puzzle, dan urutan move dari simpul awal untuk mencapai simpul tersebut. Kemudian, fungsi akan

terus mencari solusi sampai heap kosong. Jika heap belum kosong, maka fungsi akan mengambil nilai teratas dari heap (dengan cost terkecil), dan menambahkannya ke daftar simpul yang sudah dikunjungi. Kemudian, jika cost dan depth bernilai sama, maka jawaban ditemukan, dan nilai mincost akan diupdate dengan cost simpul tersebut, dan loop akan berlanjut ke iterasi berikutnya. Sebaliknya, jika cost dan depth berbeda, maka fungsi akan membentuk hingga empat simpul, yaitu simpul yang dapat dicapai dari simpul sekarang, dan memasukkannya ke heap jika simpul tersebut valid dan simpul tersebut belum dikunjungi, dengan nilai cost dan depth yang baru. Jika heap sudah kosong, maka fungsi akan mengembalikan jumlah nilai fungsi kurang dan X awal. Jika jawaban diperoleh, maka list answer akan memuat langkah-langkah yang diperlukan untuk mencapai jawaban, dan sebaliknya, list answer akan kosong jika jawaban tidak dapat diperoleh.

BAB II
Source Code Program

2.1 Program Puzzle Solver

```
import copy
from hashlib import new #deepcopy
import heapq as pq #priority queue

class puzzlesolver:
    '''
    methods to calls:
    constructor, solve
    attributes:
    puzzle: initial puzzle
    mincost: total puzzle solve length
    answer: moves to answer
    '''
    dxy = [(1,0),(0,1),(-1,0),(0,-1)] #move directions
    puzzle = [] #initial puzzle
    mincost = 1e9+7 #minimum cost, initialized at 1e9+7
    answer = [] #answer to puzzle, empty if unsolvable
    kurang = [0 for _ in range(16)] #nilai fungsi kurang tiap entri
    visited = []
    X = 0 #nilai X
    total = 0

    def __init__(self, filename):
        self.getPuzzle(filename)

    def getPuzzle(self,filename): #gets puzzle from test folder
        temp_puzzle = []
        puzzle_set = {}
        with open(filename) as f:
            lines = f.readlines()
            if len(lines)!=4:
                return
            for line in lines:
                if len(line.split())!=4:
                    return
                temp_puzzle.append([int(i) for i in line.split()])
                for i in line.split():
                    puzzle_set.add(int(i))
        if puzzle_set!={i for i in range(16)}:
            return
```

```
        self.puzzle = temp_puzzle

    def calcKurang(self): #calculates sum of KURANG(i) + X
        flat = [i for j in self.puzzle for i in j]
        cost = 0
        for i in range(len(flat)):
            temp = 0
            if flat[i]==0 and (((i//4)%2 == 0 and i%2==1) or (i//4)%2 == 1 and
i%2 == 0):
                self.X = 1
            for j in range(i+1,len(flat)):
                if ((flat[i]>flat[j] or flat[i]==0) and flat[j]!=0):
                    cost += 1
                    temp += 1
            self.kurang[flat[i]] = temp
        return cost + self.X

    def calcCost(self, puzzle): #calculates cost
        flat = [i for j in puzzle for i in j]
        cost = 0
        for i in range(len(flat)):
            if ((i+1)%16 != flat[i]):
                cost+=1
        return cost

    def getZeroPos(self, puzzle):
        for i in range(len(puzzle)):
            for j in range(len(puzzle[0])):
                if puzzle[i][j]==0:
                    return (i,j)
        return (-1,-1)

    def validIndex(self, x, y):
        return (0<=x<4 and 0<=y<4)

    def solve(self): #main puzzle solver, returns kurang value
        kurang = self.calcKurang()
        if kurang%2!=0:
            return kurang

        cost = self.calcCost(self.puzzle)
        heap = [] #current total cost, depth, puzzle, moves to puzzle
        pq.heappush(heap,(cost,0,copy.deepcopy(self.puzzle),[]))

        while len(heap)>0:
```

```
            curCost, depth, curPuzzle, path = pq.heappop(heap)
            self.visited.append(curPuzzle)
            self.total += 1
            if (curCost>self.mincost): #bound
                continue
            if (curCost == depth): #answer found
                if (curCost<self.mincost):
                    self.mincost = curCost
                    self.answer = path
                continue

            #transitions
            x,y = self.getZeroPos(curPuzzle)
            for dx,dy in self.dxy:
                if (not self.validIndex(x+dx, y+dy)):
                    continue
                newPuzzle = copy.deepcopy(curPuzzle)
                newPuzzle[x][y], newPuzzle[x+dx][y+dy] = newPuzzle[x+dx][y+dy],
newPuzzle[x][y]
                newPath = copy.deepcopy(path)
                newPath.append((dx,dy))
                cost = self.calcCost(newPuzzle)
                if (newPuzzle in self.visited):
                    continue
                pq.heappush(heap,(cost+depth+1, depth+1, newPuzzle, newPath))
        return kurang
```

2.2 Program Utama

```
import os
import time
from tkinter import *
from tkinter import ttk
from tkinter import filedialog as fd
from puzzlesolver import *
from tkinter.messagebox import showinfo

### NON-UI VARIABLES
# input
config = ""

# puzzle animation config
puzzle_start = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,0]]
```

```
zero_pos = (3,3)
moves = []

# puzzle solving details
kurang = []
kurangSum = 0
X = []
mincost = 0
duration = 0
total = 0

### BUTTONS
# file select button
def select_file():
    global config
    filetypes = (
        ('text files', '*.txt'),
        ('All files', '*.*')
    )

    filename = fd.askopenfilename(
        title='Open a file',
        initialdir='./test',
        filetypes=filetypes)

    filename_label.config(text = "Config: " + os.path.basename(filename))
    config = filename

#call puzzle solver
def solve():
    global puzzle_start, zero_pos, moves, puzzle_labels
    global kurang, kurangSum, X, mincost, duration, total

    animate_button.place_forget()
    reset_button.place_forget()

    if (len(config)==0):
        status_label.config(text = "File config tidak valid!")
        return
    psolver = puzzlesolver(config)
    if (len(psolver.puzzle)==0):
        status_label.config(text = "File config tidak valid!")
        return
    status_label.config(text = "Solving...")
    # CALL FUCNTION
```

```
    start = time.time()
    kurangSum = psolver.solve()
    end = time.time()

    duration = end - start
    if kurangSum%2!=0:
        status_label.config(text = "Tidak ada solusi!")
        puzzle_start = copy.deepcopy(psolver.puzzle)
        moves = []
    else:
        status_label.config(text = "Solusi ditemukan!")
        puzzle_start = copy.deepcopy(psolver.puzzle)
        moves = copy.deepcopy(psolver.answer)
        animate_button.place(x=110, y=225)
        reset_button.place(x=110, y=260)
    kurang = copy.deepcopy(psolver.kurang)
    X = psolver.X
    mincost = psolver.mincost
    total = psolver.total
    details_button.place(x=20, y=110)
    resetPuzzle()

def details():
    info = "Execution time: "+str(duration*1000)+" ms\n"
    info += "Jumlah simpul yang dibangkitkan: " + str(total) + "\n"
    info += "Jumlah fungsi kurang adalah " + str(kurangSum) + "\n"
    for i in range(1,16):
        info += "Nilai fungsi kurang " + str(i) + ": " + str(kurang[i]) + "\n"
    info += "Nilai fungsi kurang 16: " + str(kurang[0]) + "\n"
    showinfo(
        title='Solution Details',
        message=info
    )

# reset puzzle ui
def resetPuzzle():
    global zero_pos
    for i in range(4):
        for j in range(4):
            if puzzle_start[i][j]==0:
                zero_pos = (i,j)
                puzzle_labels[i][j].config(text = "")
            else:
                puzzle_labels[i][j].config(text = str(puzzle_start[i][j]))
    root.update()
```

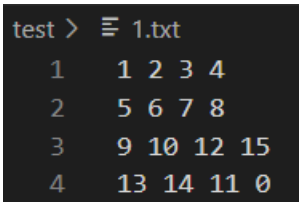
```
# animate solution
def animate():
    resetPuzzle()
    global puzzle_labels
    puzzle = copy.deepcopy(puzzle_start)
    zero = copy.deepcopy(zero_pos)

    for dxy in moves:
        time.sleep(1)
        zx, zy = zero
        dx, dy = dxy
        puzzle[zx][zy], puzzle[zx+dx][zy+dy] = puzzle[zx+dx][zy+dy],
puzzle[zx][zy]
        puzzle_labels[zx+dx][zy+dy].config(text = "")
        puzzle_labels[zx][zy].config(text = str(puzzle[zx][zy]))
        zero = (zx + dx, zy + dy)
        root.update()

### TKINTER SETTINGS
# setting the windows size
root=Tk()
root.title("15 Puzzle Solver")
root.geometry("300x300")

# label
prompt_label = Label(root, text = 'Pilih masukan puzzle:', font=('calibre',8))
prompt_label.place(x=20, y=8)

# open button
open_button = ttk.Button(
    root,
    text='Select file',
    command=select_file
)
open_button.place(x=20, y=30)

#filename
filename_label = Label(root, text = 'File config belum dipilih',
font=('calibre',8))
filename_label.place(x=20, y=56)

# solve button
solve_button = ttk.Button(
    root,
```

```
    text='Solve',
    command=solve
)
solve_button.place(x=20, y=78)

# status_label
status_label = Label(root, text = '', font=('calibre',8))
status_label.place(x=100, y=80)

# kurang_label
kurang_label = Label(root, text = '', font=('calibre',8))
kurang_label.place(x=20, y=104)

# puzzle labels
puzzle_labels=[[0 for _ in range(4)] for _ in range(4)]
for i in range(4):
    for j in range(4):
        puzzle_labels[i][j] = Label(root, text = str(i*4+j+1),
font=('calibre',8),
            height= 1, width=2, anchor=CENTER,
            borderwidth=1, relief="solid")
        puzzle_labels[i][j].place(x=110+20*j, y=135+20*i)
puzzle_labels[3][3].config(text= " ")

# animate button
animate_button = ttk.Button(
    root,
    text='Animate',
    command=animate
)


# reset button
reset_button = ttk.Button(
    root,
    text='Reset',
    command=resetPuzzle
)


# reset button
details_button = ttk.Button(
    root,
    text='Details',
    command=details
```

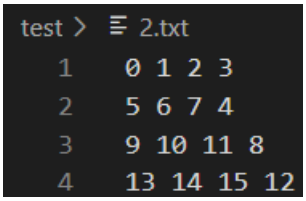
```
)

root.mainloop()
```

BAB III
Pengujian Progam

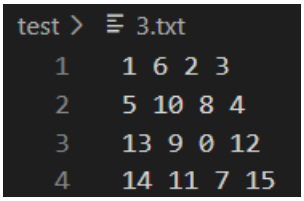
3.1 Puzzle yang dapat disolve



Gambar 1. Config Puzzle 1

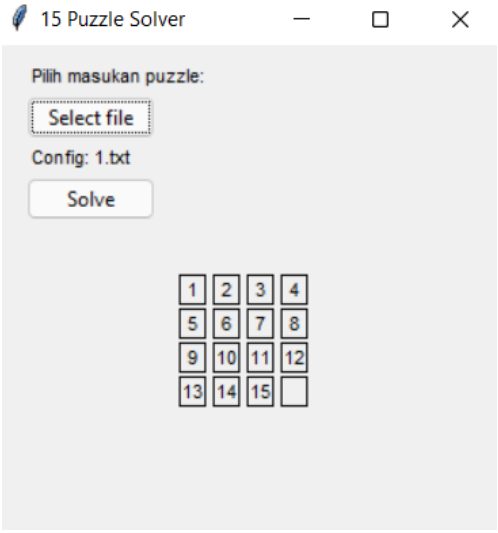


Gambar 2. Config Puzzle 2

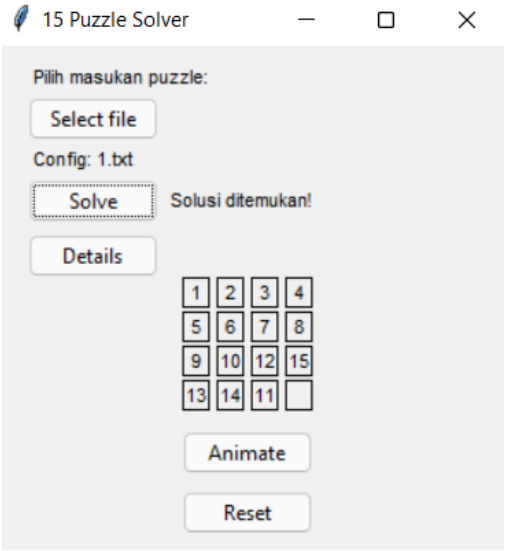


Gambar 3. Config Puzzle 3

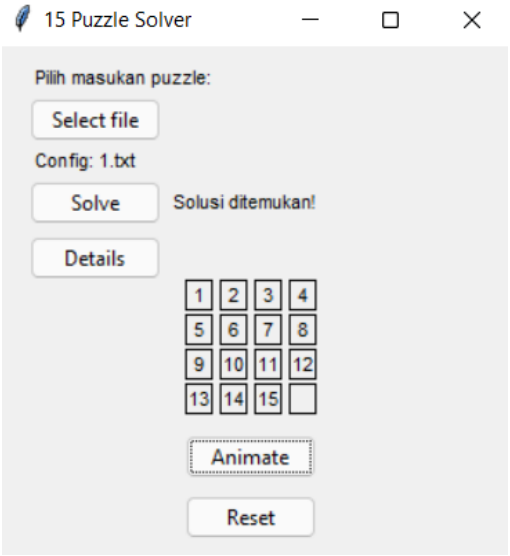
3.1.1 Pengujian Puzzle 1



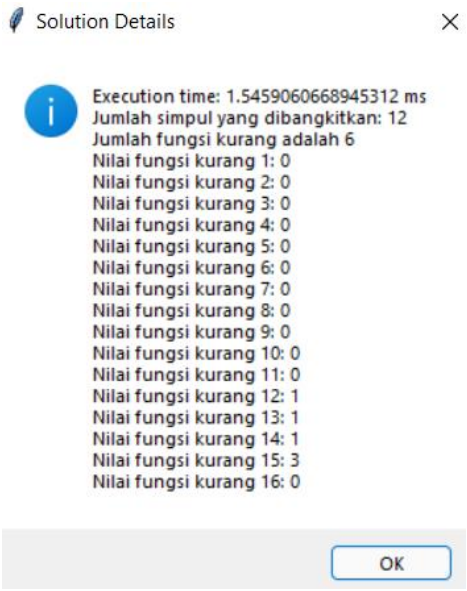
Gambar 4. Masukan config puzzle 1



Gambar 5. Penyelesaian puzzle 1



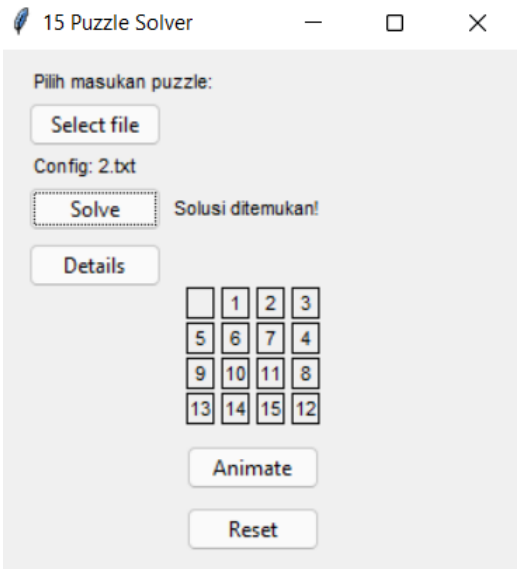
Gambar 6. Akhir animasi solusi puzzle 1



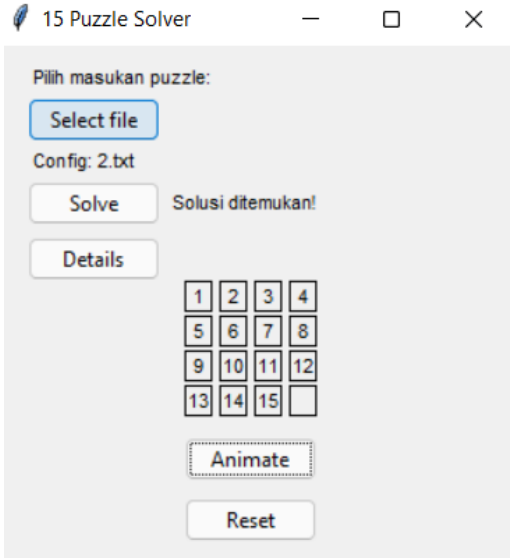
Gambar 7. Rincian puzzle 1

3.1.2 Pengujian Puzzle 2

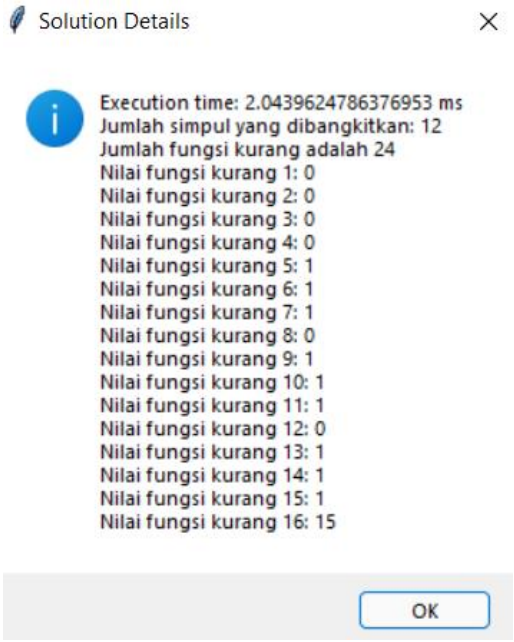
Gambar 8. Masukan config puzzle 2



Gambar 9. Penyelesaian puzzle 2

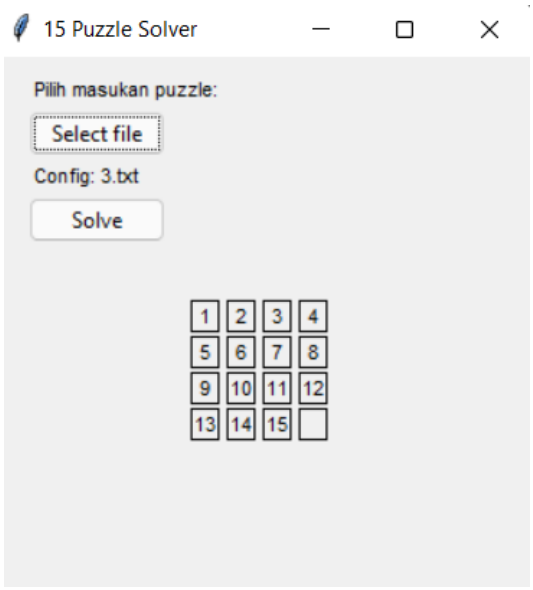


Gambar 10. Akhir animasi solusi puzzle 2

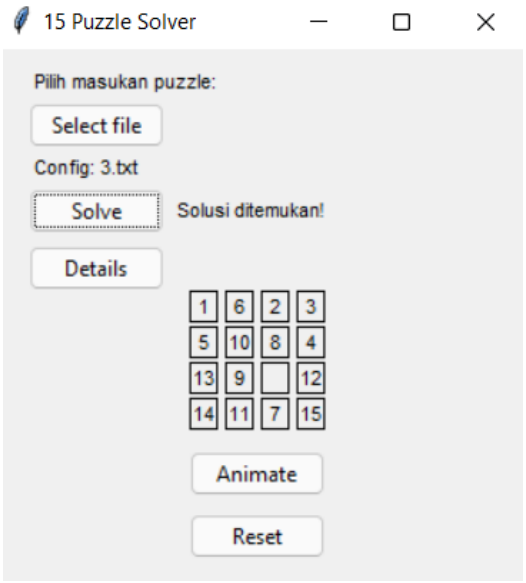


Gambar 11. Rincian puzzle 2

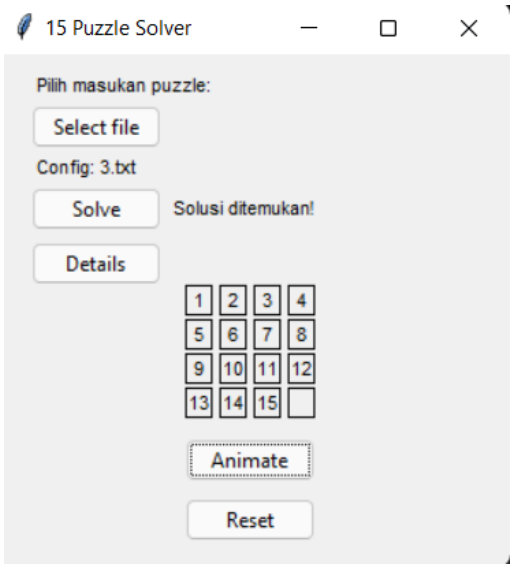
3.1.2 Pengujian Puzzle 3



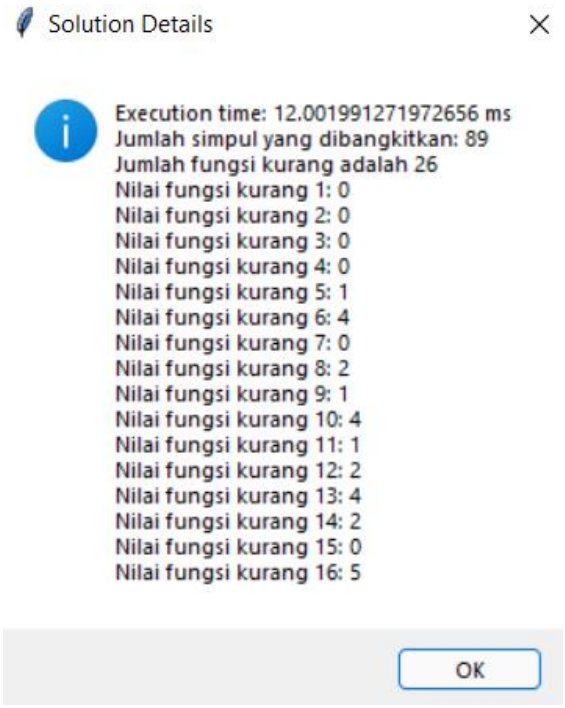
Gambar 12 Masukan config puzzle 3



Gambar 14. Penyelesaian puzzle 3

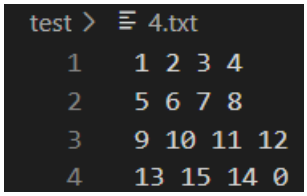


Gambar 15. Akhir animasi solusi puzzle 3

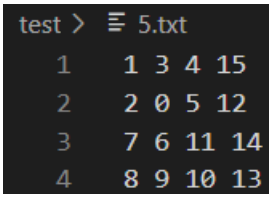


Gambar 16. Rincian puzzle 3

3.2 Puzzle yang tidak dapat diselesaikan

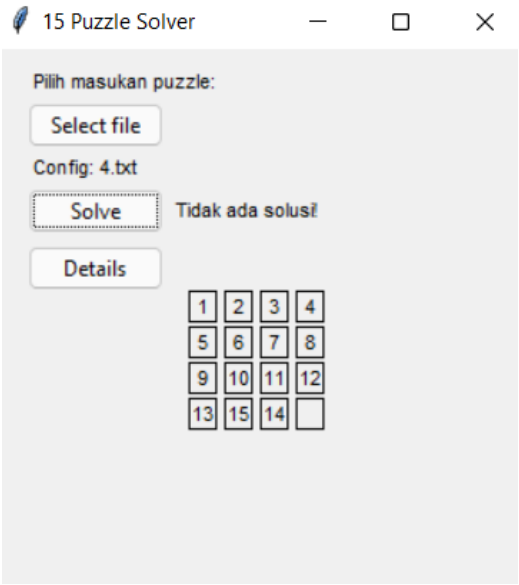


Gambar 17. Config puzzle 4

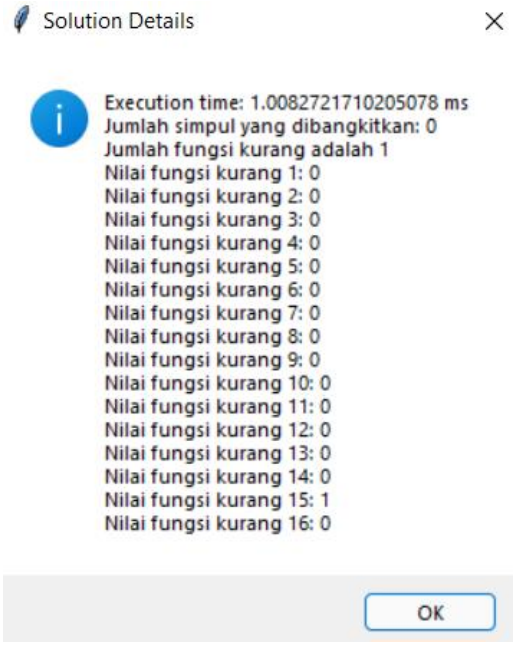


Gambar 18. Config puzzle 5

3.2.1 Pengujian puzzle 4



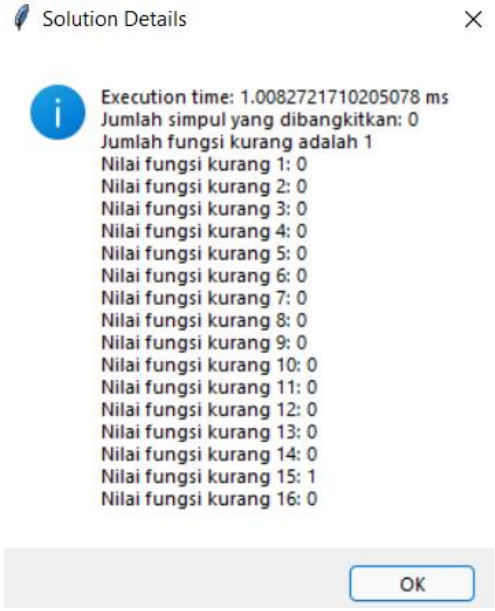
Gambar 19. Penyelesaian puzzle 4



Gambar 20. Rincian puzzle 4

3.2.2 Pengujian puzzle 5

Gambar 21. Penyelesaian puzzle 5



Gambar 22. Rincian puzzle 5

Lampiran

Tabel 1. Cek List Status Program

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil running	✓	
3. Program dapat menerima input dan menuliskan output	✓	
4. Luaran sudah benar untuk semua data uji	✓	
5. Bonus dibuat	✓	

Repositori program: https://github.com/dxt99/STIMA-TC3