TUGAS KECIL IF2211
STRATEGI ALGORITMA

PENYELESAIAN PERSOALAN 15-PUZZLE DENGAN ALGORITMA BRANCH AND BOUND

Disusun oleh
Farrel Farandieka Fibriyanto - 13520054

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

Semester II Tahun 2021/2022

Daftar Isi

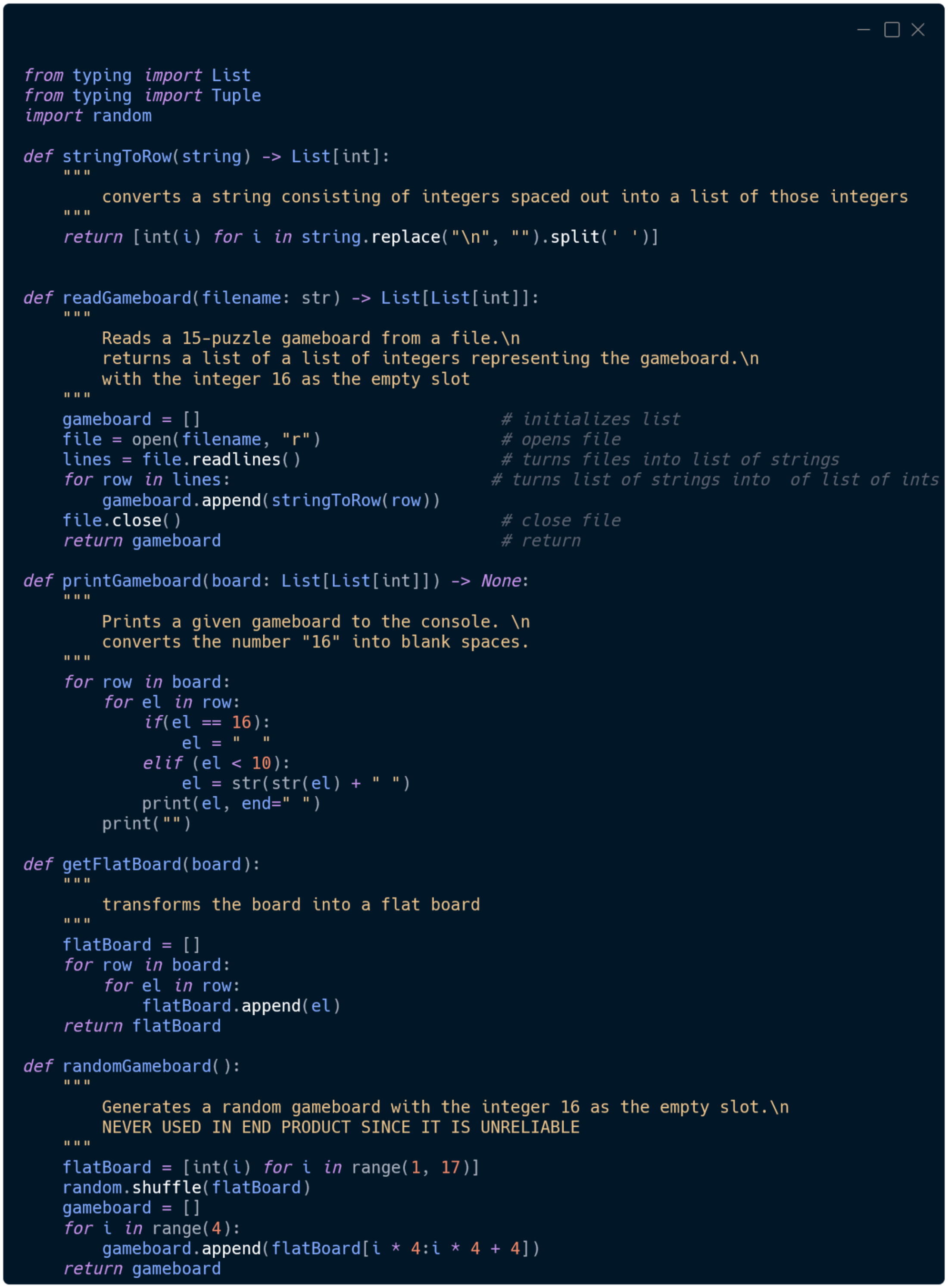
Algoritma Branch & Bound

Dalam laporan ini, saya akan memanggil kotak-kotak dalam permainan 15-puzzle sebagai *game board* dan pergerakan kotak kosong sebagai *move*. Algoritma branch & bound akan meneruskan pembangunan simpul dari semua simpul hidup yang memiliki *cost* terkecil. Dalam permainan 15-puzzle, *cost* merupakan ongkos *move* yang dibutuhkan untuk mencapai simpul tersebut dari simpul akar ditambah dengan ongkos perkiraan *move* untuk mencapai simpul tujuan dari simpul tersebut. Ongkos perkiraan move untuk mencapai simul tujuan menggunakan *heuristics Misplaced Tile*. Algoritma yang saya buat bekerja sebagai berikut:

1. Masukkan kondisi *game board* awal pada *queue* kosong sebagai simpul akar
2. Bangun semua kemungkinan *game board* sebagai simpul yang bisa didirikan berdasarkan pergerakan kotak kosong yang memungkinkan pada *game board* di awal *queue* (apabila kotak kosong berada di atas *game board*, tentu saja tidak bisa bergerak ke atas, dst).
3. Tambahkan informasi pergerakan yang dilakukan, ongkos untuk mencapai kondisi *game board* tersebut, serta perkiraan ongkos untuk mencapai simpul tujuan
4. Apabila terdapat simpul yang memiliki ongkos untuk mencapai simpul tujuan berupa 0, maka hentikan program, kita sudah mendapatkan jawabannya. Apabila tidak, lanjut ke langkah 5
5. Urutkan *queue* simpul berdasarkan *cost* mereka, dari *cost* yang paling rendah ke yang paling tinggi dan kembali ke langkah 2

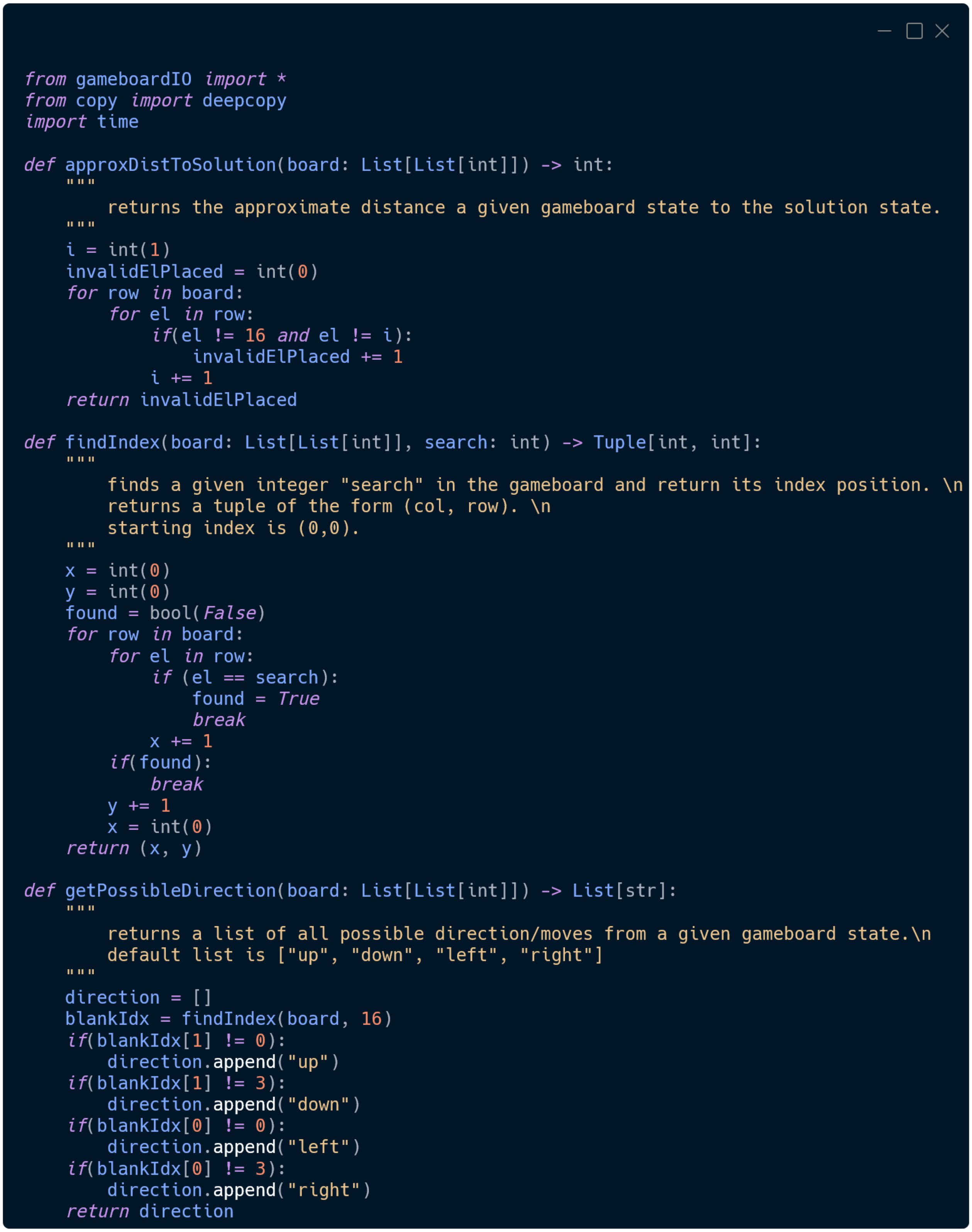
Source Code Program

gameboardIO.py

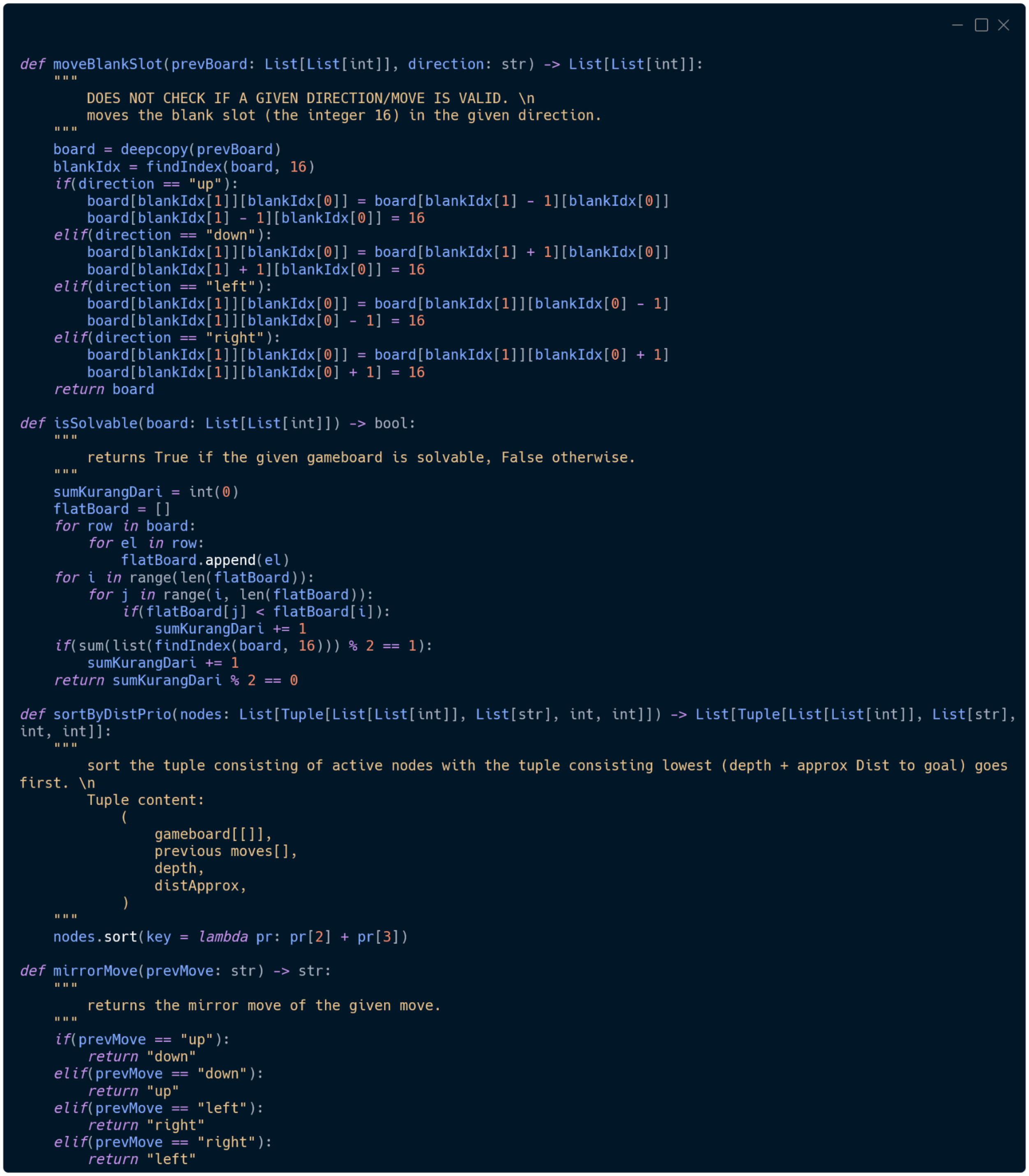


```python
from typing import List
from typing import Tuple
import random

def stringToRow(string) -> List[int]:
    """
        converts a string consisting of integers spaced out into a list of those integers
    """
    return [int(i) for i in string.replace("\n", "").split(' ')]


def readGameboard(filename: str) -> List[List[int]]:
    """
        Reads a 15-puzzle gameboard from a file.\n
        returns a list of a list of integers representing the gameboard.\n
        with the integer 16 as the empty slot
    """
    gameboard = []                              # initializes list
    file = open(filename, "r")                  # opens file
    lines = file.readlines()                    # turns files into list of strings
    for row in lines:                          # turns list of strings into  of list of ints
        gameboard.append(stringToRow(row))
    file.close()                                # close file
    return gameboard                            # return

def printGameboard(board: List[List[int]]) -> None:
    """
        Prints a given gameboard to the console. \n
        converts the number "16" into blank spaces.
    """
    for row in board:
        for el in row:
            if(el == 16):
                el = "  "
            elif (el < 10):
                el = str(str(el) + " ")
            print(el, end=" ")
        print("")

def getFlatBoard(board):
    """
        transforms the board into a flat board
    """
    flatBoard = []
    for row in board:
        for el in row:
            flatBoard.append(el)
    return flatBoard

def randomGameboard():
    """
        Generates a random gameboard with the integer 16 as the empty slot.\n
        NEVER USED IN END PRODUCT SINCE IT IS UNRELIABLE
    """
    flatBoard = [int(i) for i in range(1, 17)]
    random.shuffle(flatBoard)
    gameboard = []
    for i in range(4):
        gameboard.append(flatBoard[i * 4:i * 4 + 4])
    return gameboard
```

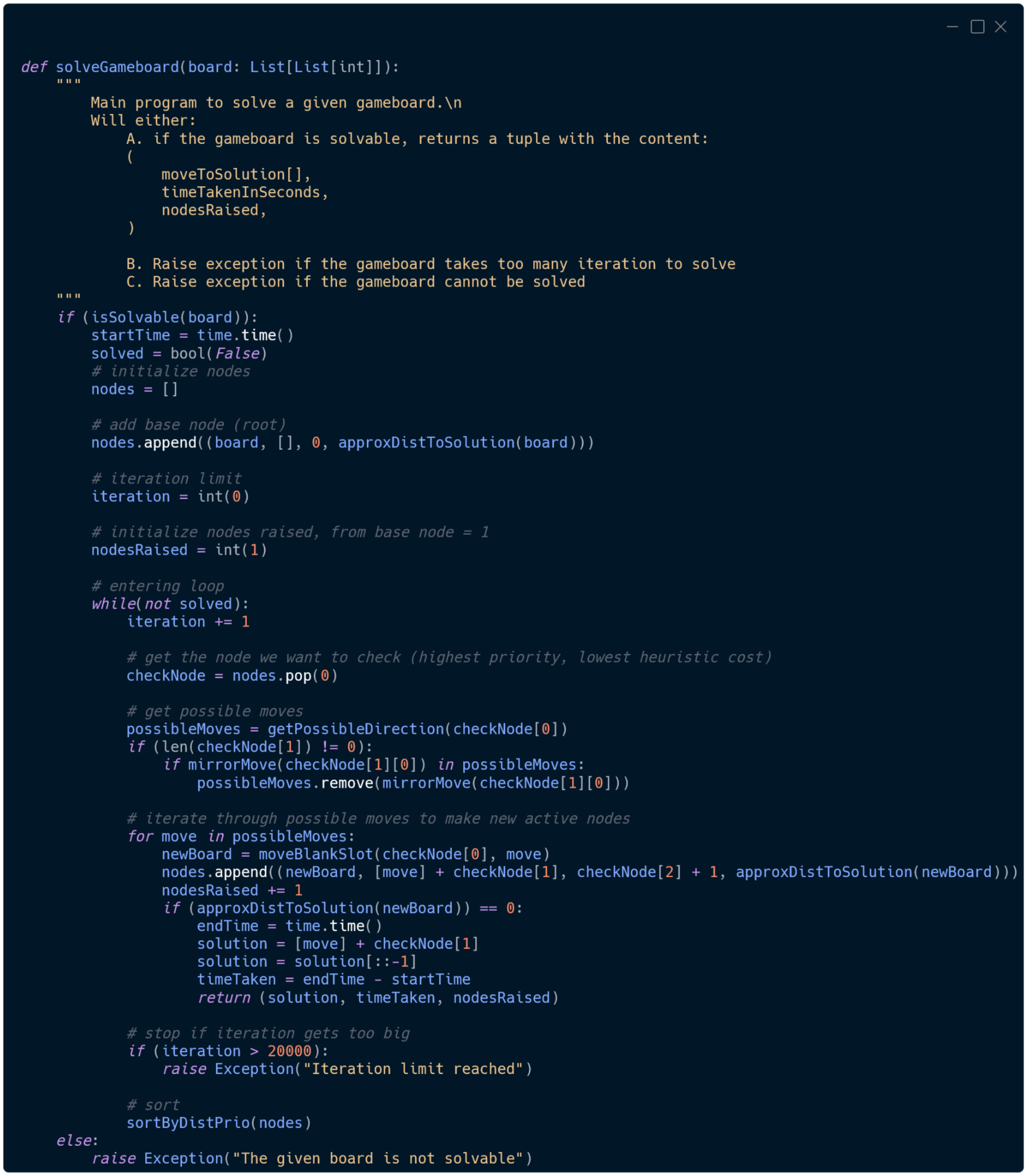
puzzlesolver.py



```python
from gameboardIO import *
from copy import deepcopy
import time

def approxDistToSolution(board: List[List[int]]) -> int:
    """
        returns the approximate distance a given gameboard state to the solution state.
    """
    i = int(1)
    invalidElPlaced = int(0)
    for row in board:
        for el in row:
            if(el != 16 and el != i):
                invalidElPlaced += 1
            i += 1
    return invalidElPlaced

def findIndex(board: List[List[int]], search: int) -> Tuple[int, int]:
    """
        finds a given integer "search" in the gameboard and return its index position. \n
        returns a tuple of the form (col, row). \n
        starting index is (0,0).
    """
    x = int(0)
    y = int(0)
    found = bool(False)
    for row in board:
        for el in row:
            if (el == search):
                found = True
                break
            x += 1
        if(found):
            break
        y += 1
        x = int(0)
    return (x, y)

def getPossibleDirection(board: List[List[int]]) -> List[str]:
    """
        returns a list of all possible direction/moves from a given gameboard state.\n
        default list is ["up", "down", "left", "right"]
    """
    direction = []
    blankIdx = findIndex(board, 16)
    if(blankIdx[1] != 0):
        direction.append("up")
    if(blankIdx[1] != 3):
        direction.append("down")
    if(blankIdx[0] != 0):
        direction.append("left")
    if(blankIdx[0] != 3):
        direction.append("right")
    return direction
```

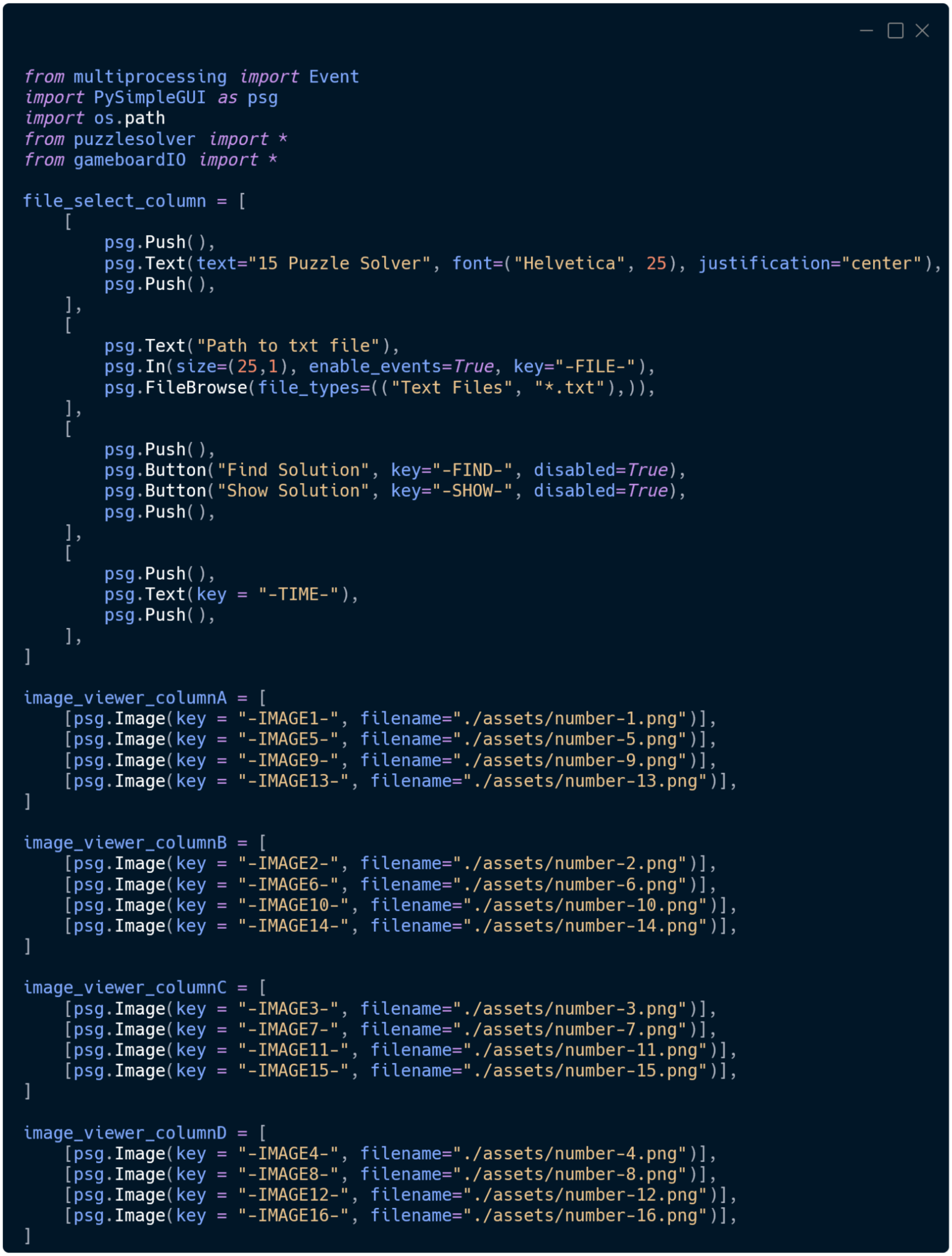


```python
def moveBlankSlot(prevBoard: List[List[int]], direction: str) -> List[List[int]]:
    """
        DOES NOT CHECK IF A GIVEN DIRECTION/MOVE IS VALID. \n
        moves the blank slot (the integer 16) in the given direction.
    """
    board = deepcopy(prevBoard)
    blankIdx = findIndex(board, 16)
    if(direction == "up"):
        board[blankIdx[1]][blankIdx[0]] = board[blankIdx[1] - 1][blankIdx[0]]
        board[blankIdx[1] - 1][blankIdx[0]] = 16
    elif(direction == "down"):
        board[blankIdx[1]][blankIdx[0]] = board[blankIdx[1] + 1][blankIdx[0]]
        board[blankIdx[1] + 1][blankIdx[0]] = 16
    elif(direction == "left"):
        board[blankIdx[1]][blankIdx[0]] = board[blankIdx[1]][blankIdx[0] - 1]
        board[blankIdx[1]][blankIdx[0] - 1] = 16
    elif(direction == "right"):
        board[blankIdx[1]][blankIdx[0]] = board[blankIdx[1]][blankIdx[0] + 1]
        board[blankIdx[1]][blankIdx[0] + 1] = 16
    return board

def isSolvable(board: List[List[int]]) -> bool:
    """
        returns True if the given gameboard is solvable, False otherwise.
    """
    sumKurangDari = int(0)
    flatBoard = []
    for row in board:
        for el in row:
            flatBoard.append(el)
    for i in range(len(flatBoard)):
        for j in range(i, len(flatBoard)):
            if(flatBoard[j] < flatBoard[i]):
                sumKurangDari += 1
    if(sum(list(findIndex(board, 16))) % 2 == 1):
        sumKurangDari += 1
    return sumKurangDari % 2 == 0

def sortByDistPrio(nodes: List[Tuple[List[List[int]], List[str], int, int]]) -> List[Tuple[List[List[int]], List[str],
int, int]]:
    """
        sort the tuple consisting of active nodes with the tuple consisting lowest (depth + approx Dist to goal) goes
first. \n
        Tuple content:
            (
                gameboard[[]],
                previous moves[],
                depth,
                distApprox,
            )
    """
    nodes.sort(key = lambda pr: pr[2] + pr[3])

def mirrorMove(prevMove: str) -> str:
    """
        returns the mirror move of the given move.
    """
    if(prevMove == "up"):
        return "down"
    elif(prevMove == "down"):
        return "up"
    elif(prevMove == "left"):
        return "right"
    elif(prevMove == "right"):
        return "left"
```

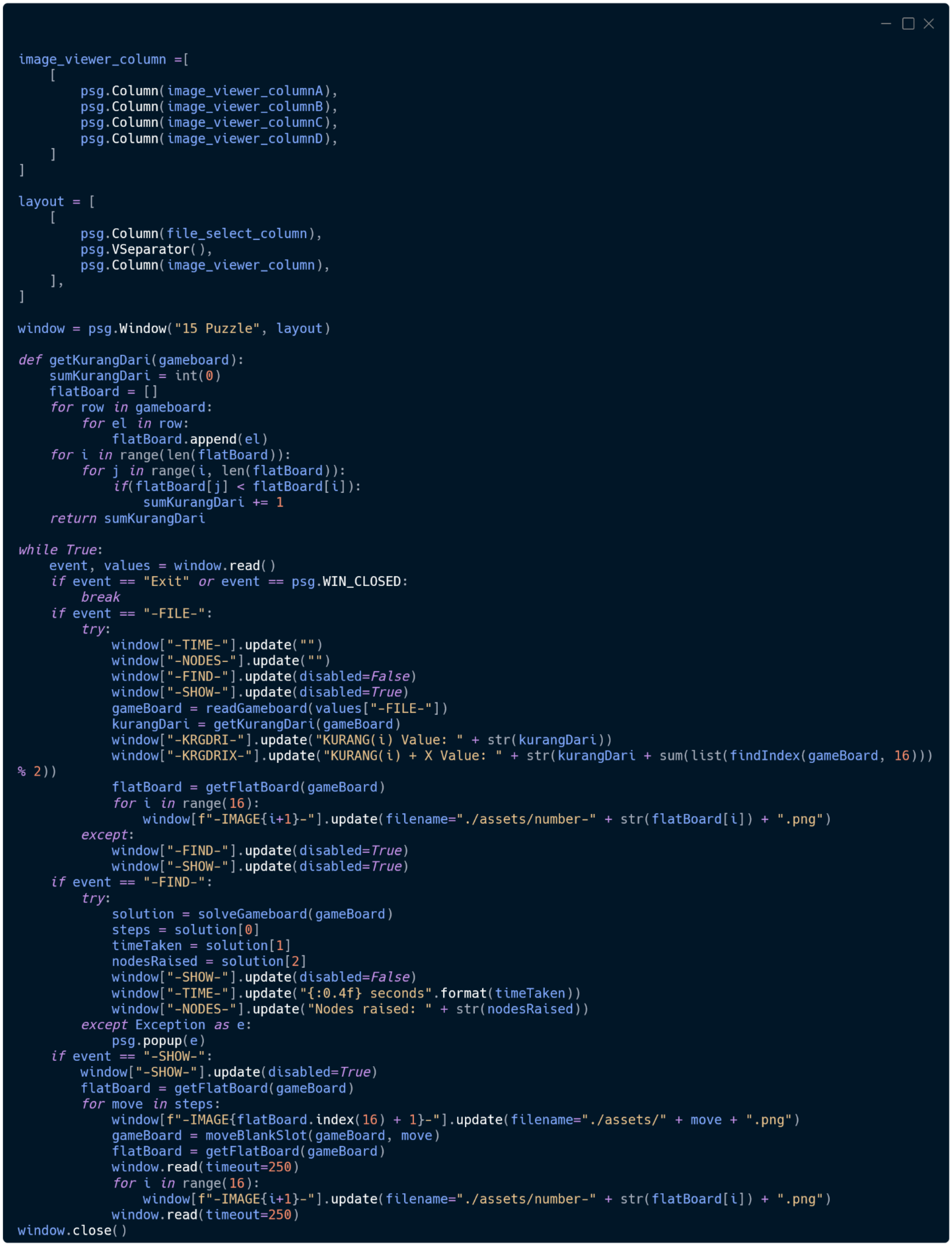


```python
def solveGameboard(board: List[List[int]]):
    """
        Main program to solve a given gameboard.\n
        Will either:
            A. if the gameboard is solvable, returns a tuple with the content:
            (
                moveToSolution[],
                timeTakenInSeconds,
                nodesRaised,
            )

            B. Raise exception if the gameboard takes too many iteration to solve
            C. Raise exception if the gameboard cannot be solved
    """
    if (isSolvable(board)):
        startTime = time.time()
        solved = bool(False)
        # initialize nodes
        nodes = []

        # add base node (root)
        nodes.append((board, [], 0, approxDistToSolution(board)))

        # iteration limit
        iteration = int(0)

        # initialize nodes raised, from base node = 1
        nodesRaised = int(1)

        # entering loop
        while(not solved):
            iteration += 1

            # get the node we want to check (highest priority, lowest heuristic cost)
            checkNode = nodes.pop(0)

            # get possible moves
            possibleMoves = getPossibleDirection(checkNode[0])
            if (len(checkNode[1]) != 0):
                if mirrorMove(checkNode[1][0]) in possibleMoves:
                    possibleMoves.remove(mirrorMove(checkNode[1][0]))

            # iterate through possible moves to make new active nodes
            for move in possibleMoves:
                newBoard = moveBlankSlot(checkNode[0], move)
                nodes.append((newBoard, [move] + checkNode[1], checkNode[2] + 1, approxDistToSolution(newBoard)))
                nodesRaised += 1
                if (approxDistToSolution(newBoard)) == 0:
                    endTime = time.time()
                    solution = [move] + checkNode[1]
                    solution = solution[::-1]
                    timeTaken = endTime - startTime
                    return (solution, timeTaken, nodesRaised)

            # stop if iteration gets too big
            if (iteration > 20000):
                raise Exception("Iteration limit reached")

            # sort
            sortByDistPrio(nodes)
    else:
        raise Exception("The given board is not solvable")
```

gui.py



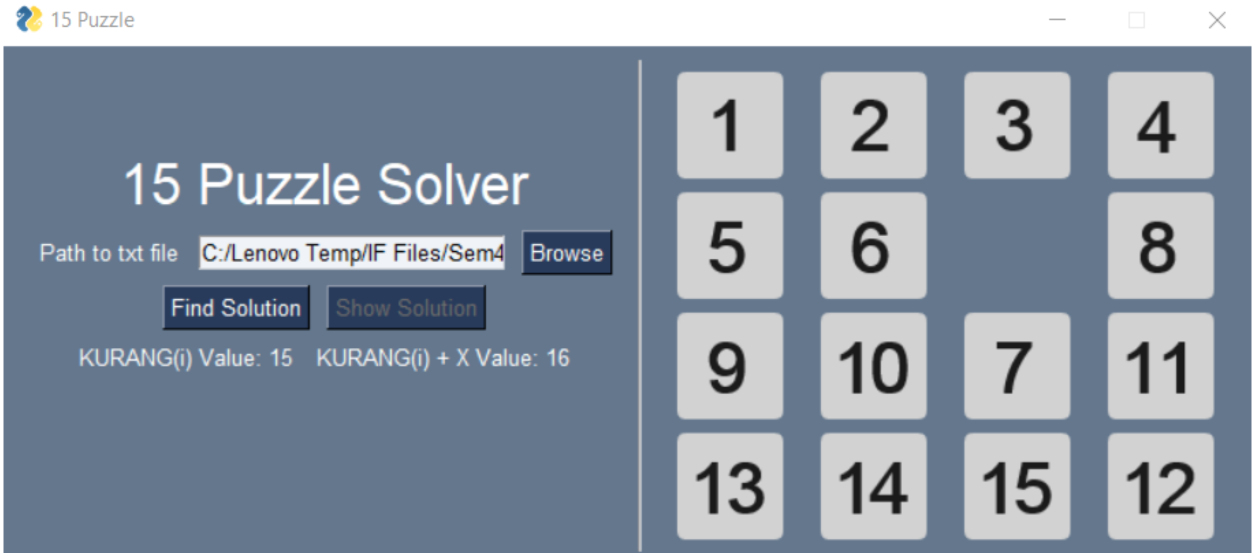
```
from multiprocessing import Event
import PySimpleGUI as psg
import os.path
from puzzlesolver import *
from gameboardIO import *

file_select_column = [
    [
        psg.Push(),
        psg.Text(text="15 Puzzle Solver", font=("Helvetica", 25), justification="center"),
        psg.Push(),
    ],
    [
        psg.Text("Path to txt file"),
        psg.In(size=(25,1), enable_events=True, key="-FILE-"),
        psg.FileBrowse(file_types=(("Text Files", "*.txt"),)),
    ],
    [
        psg.Push(),
        psg.Button("Find Solution", key="-FIND-", disabled=True),
        psg.Button("Show Solution", key="-SHOW-", disabled=True),
        psg.Push(),
    ],
    [
        psg.Push(),
        psg.Text(key = "-TIME-"),
        psg.Push(),
    ],
]

image_viewer_columnA = [
    [psg.Image(key = "-IMAGE1-", filename="./assets/number-1.png")],
    [psg.Image(key = "-IMAGE5-", filename="./assets/number-5.png")],
    [psg.Image(key = "-IMAGE9-", filename="./assets/number-9.png")],
    [psg.Image(key = "-IMAGE13-", filename="./assets/number-13.png")],
]

image_viewer_columnB = [
    [psg.Image(key = "-IMAGE2-", filename="./assets/number-2.png")],
    [psg.Image(key = "-IMAGE6-", filename="./assets/number-6.png")],
    [psg.Image(key = "-IMAGE10-", filename="./assets/number-10.png")],
    [psg.Image(key = "-IMAGE14-", filename="./assets/number-14.png")],
]

image_viewer_columnC = [
    [psg.Image(key = "-IMAGE3-", filename="./assets/number-3.png")],
    [psg.Image(key = "-IMAGE7-", filename="./assets/number-7.png")],
    [psg.Image(key = "-IMAGE11-", filename="./assets/number-11.png")],
    [psg.Image(key = "-IMAGE15-", filename="./assets/number-15.png")],
]

image_viewer_columnD = [
    [psg.Image(key = "-IMAGE4-", filename="./assets/number-4.png")],
    [psg.Image(key = "-IMAGE8-", filename="./assets/number-8.png")],
    [psg.Image(key = "-IMAGE12-", filename="./assets/number-12.png")],
    [psg.Image(key = "-IMAGE16-", filename="./assets/number-16.png")],
]
```



```
image_viewer_column =[
    [
        psg.Column(image_viewer_columnA),
        psg.Column(image_viewer_columnB),
        psg.Column(image_viewer_columnC),
        psg.Column(image_viewer_columnD),
    ]
]

layout = [
    [
        psg.Column(file_select_column),
        psg.VSeparator(),
        psg.Column(image_viewer_column),
    ],
]

window = psg.Window("15 Puzzle", layout)

def getKurangDari(gameboard):
    sumKurangDari = int(0)
    flatBoard = []
    for row in gameboard:
        for el in row:
            flatBoard.append(el)
    for i in range(len(flatBoard)):
        for j in range(i, len(flatBoard)):
            if(flatBoard[j] < flatBoard[i]):
                sumKurangDari += 1
    return sumKurangDari

while True:
    event, values = window.read()
    if event == "Exit" or event == psg.WIN_CLOSED:
        break
    if event == "-FILE-":
        try:
            window["-TIME-"].update("")
            window["-NODES-"].update("")
            window["-FIND-"].update(disabled=False)
            window["-SHOW-"].update(disabled=True)
            gameBoard = readGameboard(values["-FILE-"])
            kurangDari = getKurangDari(gameBoard)
            window["-KRGDRI-"].update("KURANG(i) Value: " + str(kurangDari))
            window["-KRGDRIX-"].update("KURANG(i) + X Value: " + str(kurangDari + sum(list(findIndex(gameBoard, 16)))
% 2))
            flatBoard = getFlatBoard(gameBoard)
            for i in range(16):
                window[f"-IMAGE{i+1}-"].update(filename="./assets/number-" + str(flatBoard[i]) + ".png")
        except:
            window["-FIND-"].update(disabled=True)
            window["-SHOW-"].update(disabled=True)
    if event == "-FIND-":
        try:
            solution = solveGameboard(gameBoard)
            steps = solution[0]
            timeTaken = solution[1]
            nodesRaised = solution[2]
            window["-SHOW-"].update(disabled=False)
            window["-TIME-"].update("{:0.4f} seconds".format(timeTaken))
            window["-NODES-"].update("Nodes raised: " + str(nodesRaised))
        except Exception as e:
            psg.popup(e)
    if event == "-SHOW-":
        window["-SHOW-"].update(disabled=True)
        flatBoard = getFlatBoard(gameBoard)
        for move in steps:
            window[f"-IMAGE{flatBoard.index(16) + 1}-"].update(filename="./assets/" + move + ".png")
            gameBoard = moveBlankSlot(gameBoard, move)
            flatBoard = getFlatBoard(gameBoard)
            window.read(timeout=250)
            for i in range(16):
                window[f"-IMAGE{i+1}-"].update(filename="./assets/number-" + str(flatBoard[i]) + ".png")
            window.read(timeout=250)
window.close()
```

Screenshot

Found-1.txt

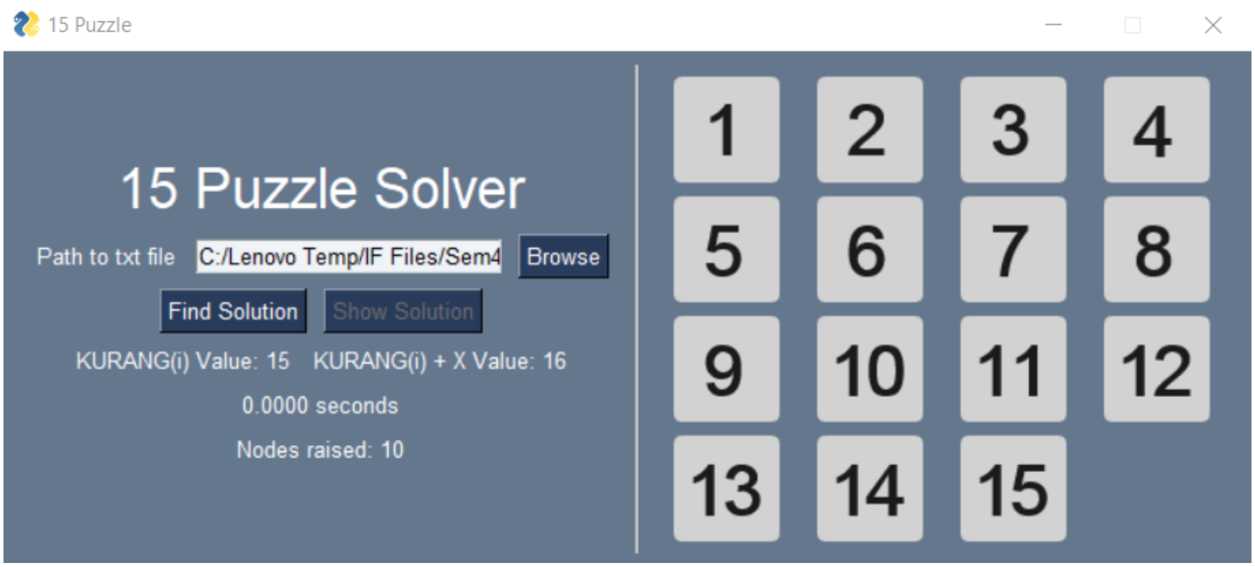


15 Puzzle Solver

Path to txt file: C:/Lenovo Temp/IF Files/Sem4

KURANG(i) Value: 15   KURANG(i) + X Value: 16

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12



15 Puzzle Solver

Path to txt file: C:/Lenovo Temp/IF Files/Sem4

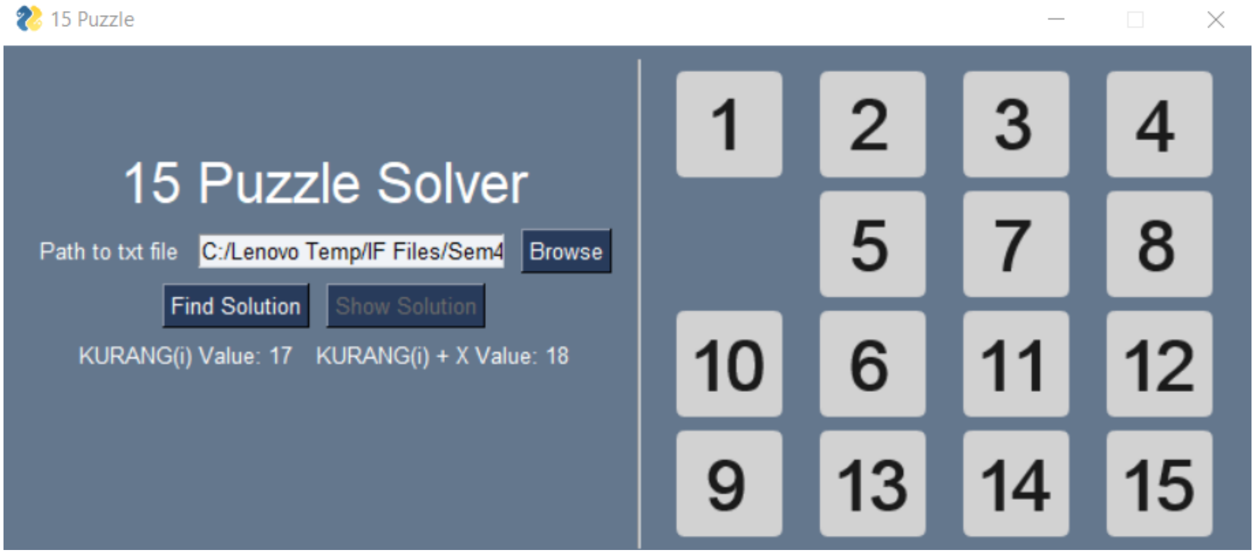
KURANG(i) Value: 15   KURANG(i) + X Value: 16

0.0000 seconds

Nodes raised: 10

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Found-2.txt

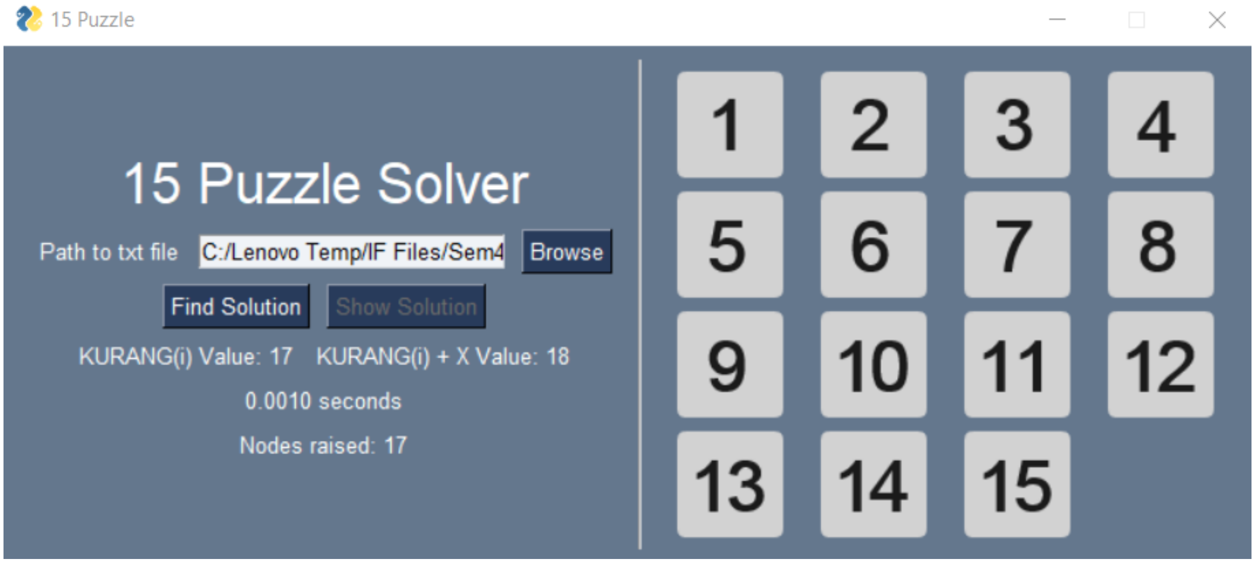


15 Puzzle Solver

Path to txt file C:/Lenovo Temp/IF Files/Sem4

KURANG(i) Value: 17   KURANG(i) + X Value: 18

1	2	3	4
	5	7	8
10	6	11	12
9	13	14	15



15 Puzzle Solver

Path to txt file C:/Lenovo Temp/IF Files/Sem4

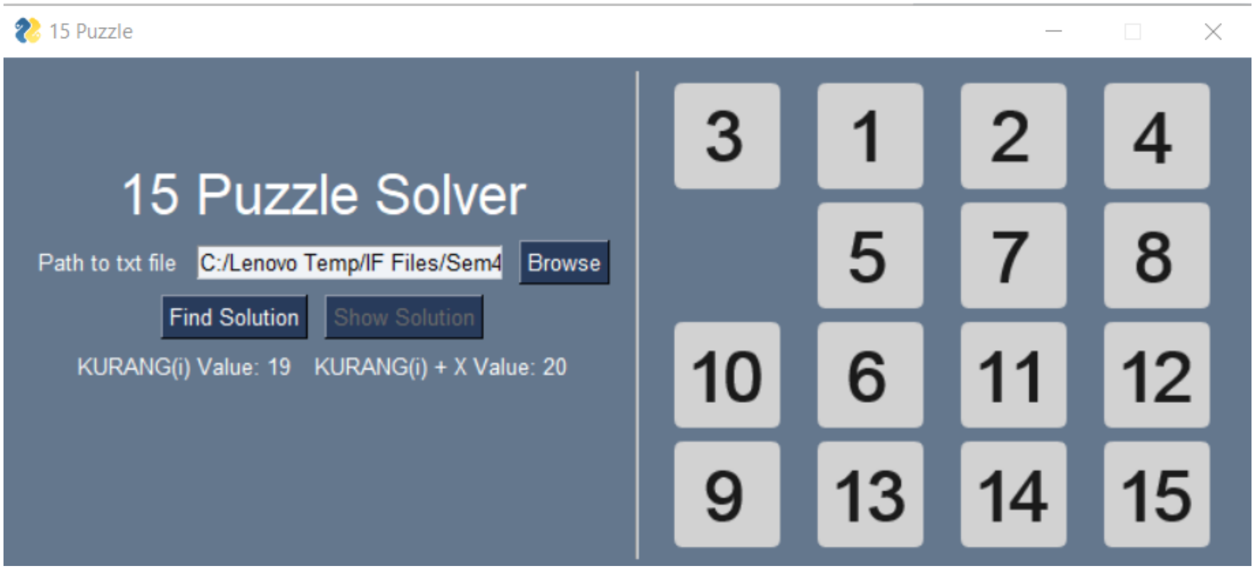
KURANG(i) Value: 17   KURANG(i) + X Value: 18

0.0010 seconds

Nodes raised: 17

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Found-3.txt

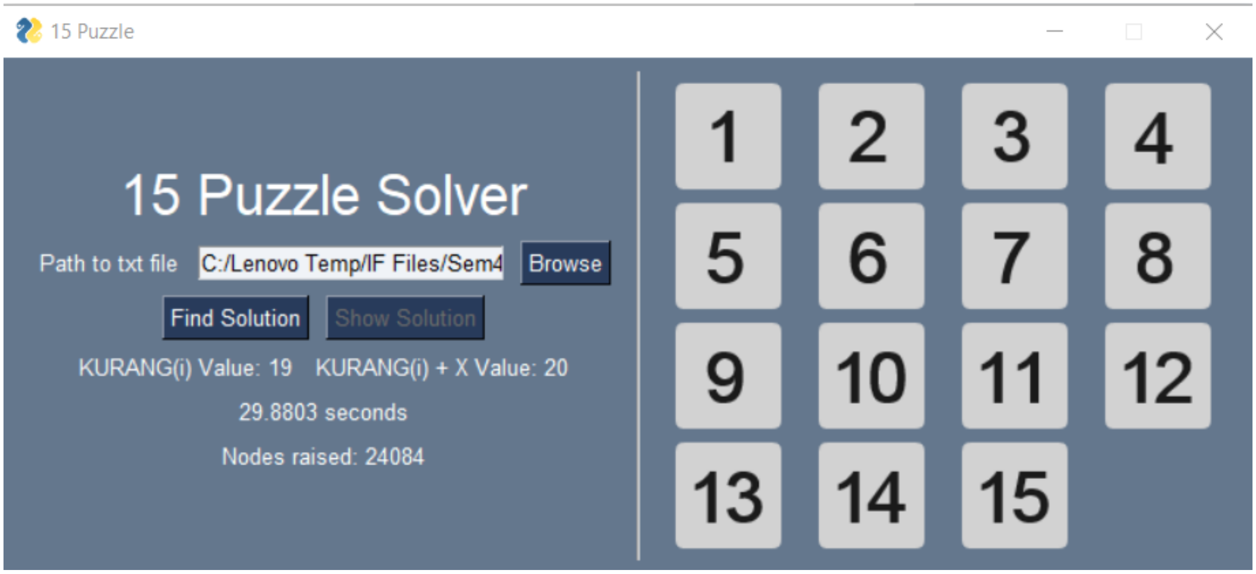


15 Puzzle Solver

Path to txt file C:/Lenovo Temp/IF Files/Sem4

KURANG(i) Value: 19 KURANG(i) + X Value: 20

3	1	2	4
	5	7	8
10	6	11	12
9	13	14	15



15 Puzzle Solver

Path to txt file C:/Lenovo Temp/IF Files/Sem4

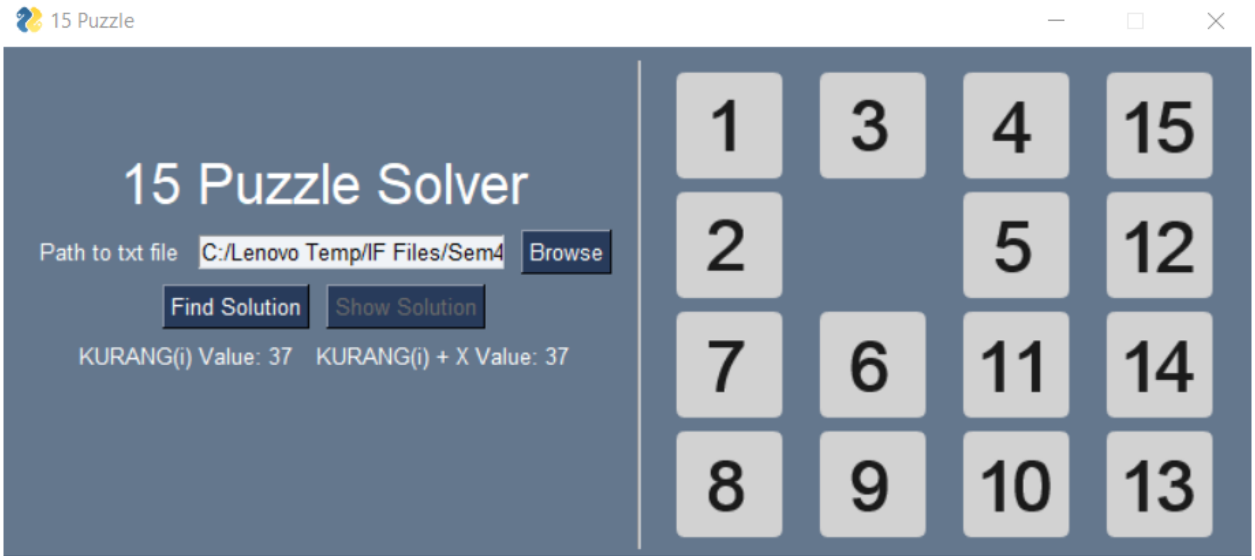
KURANG(i) Value: 19 KURANG(i) + X Value: 20

29.8803 seconds

Nodes raised: 24084

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

NotFound-1.txt

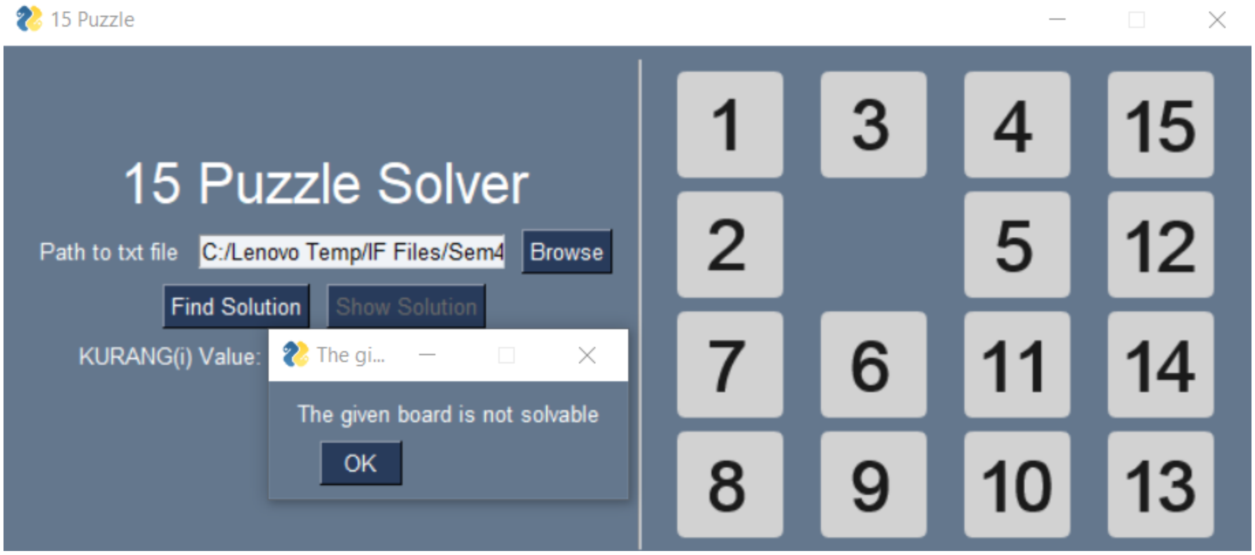


15 Puzzle Solver

Path to txt file: C:/Lenovo Temp/IF Files/Sem4

KURANG(i) Value: 37   KURANG(i) + X Value: 37

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13



15 Puzzle Solver

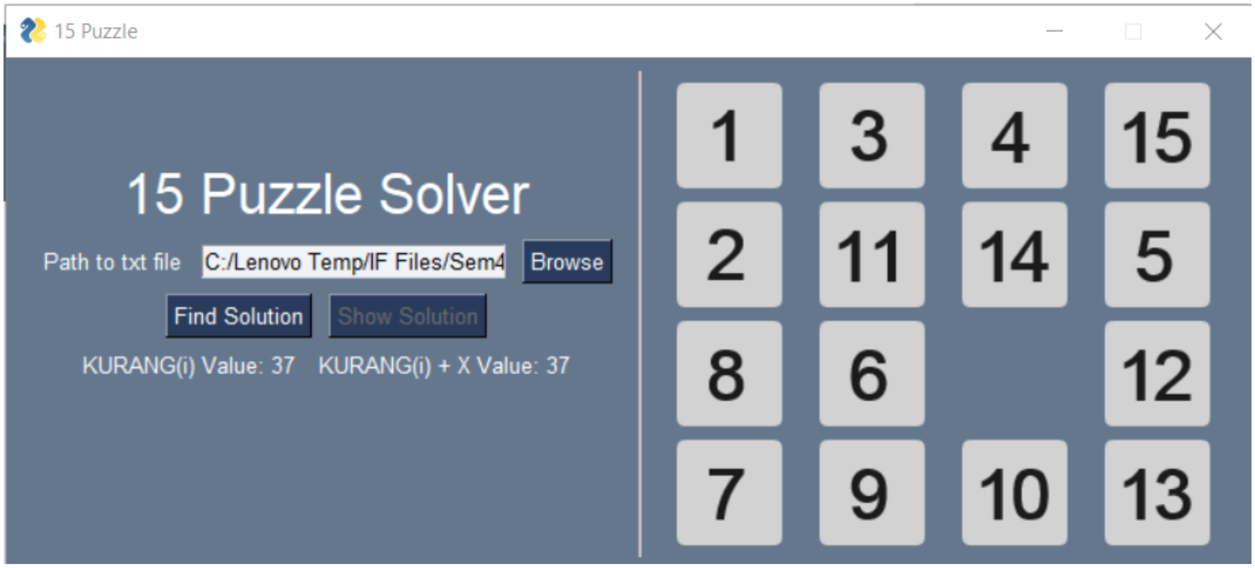
Path to txt file: C:/Lenovo Temp/IF Files/Sem4

KURANG(i) Value:

The given board is not solvable

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

NotFound-2.txt

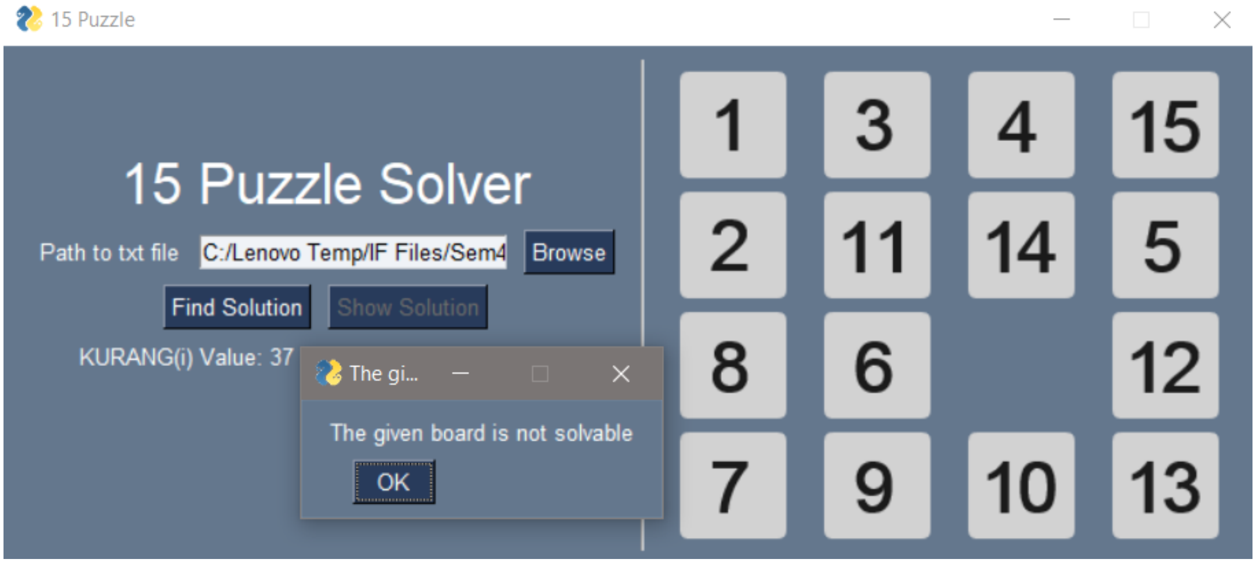


15 Puzzle Solver

Path to txt file: C:/Lenovo Temp/IF Files/Sem4

KURANG(i) Value: 37 KURANG(i) + X Value: 37

1	3	4	15
2	11	14	5
8	6		12
7	9	10	13



15 Puzzle Solver

Path to txt file: C:/Lenovo Temp/IF Files/Sem4

KURANG(i) Value: 37

The given board is not solvable

1	3	4	15
2	11	14	5
8	6		12
7	9	10	13

Catatan Kaki:

- *Step-By-Step* penyelesaian tidak ditangkap layar (*screenshoot*) karena akan memperbanyak gambar yang perlu diperiksa sehingga hanya diambil *end-state* setelah melakukan “Find Solution” dan “Show Solution”

Checklist

No.	Poin	Keberhasilan Poin
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 (GUI)	☑

Link Penting

Drive Source Code:
https://drive.google.com/drive/folders/1qtRjjcrlh4FXSNIvn8jsjDNIvg4WDSqe?usp=sharing

Repository Github:
https://github.com/Noxira/15PuzzleSolver