MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

Skill Based Mini Project Report

on

2048 Game using Python

Submitted By:

Rakhi Sehrawat

0901CS201095

Faculty Mentor:

Dr. RANJEET KUMAR SINGH

ASSISTANT PROFESSOR, CSE

Submitted to:

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE
GWALIOR - 474005 (MP) est. 1957

JAN-JUNE 2022

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

CERTIFICATE

This is certified that **Rakhi Sehrawat** (0901CS201095) has submitted the project report titled 2048 Game using python under the mentorship of Dr, Ranjeet Kumar Singh, in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Computer Science and Engineering from Madhav Institute of Technology and Science, Gwalior.

R.K. Singh

Dr. Ranjeet Kumar Singh
Faculty Mentor
Assistant Professor
Computer Science and Engineering

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

DECLARATION

I hereby declare that the work being presented in this project report, for the partial fulfilment of requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of Dr. Ranjeet Kumar Singh, Assistant Professor, Computer Science and Engineering.

I declare that I have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.

Rakhi

Rakhi Sehrawat
(0901CS201095)
II Year, 4th SEM
Computer Science and Engineering

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. I am thankful to my institute, **Madhav Institute of Technology and Science** to allow me to continue my disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute. I extend my gratitude to the Director of the institute, **Dr. R. K. Pandit** and Dean Academics, **Dr. Manjaree Pandit** for this.

I would sincerely like to thank my department, **Department of Computer Science and Engineering, for allowing** me to explore this project. I humbly thank **Dr. Manish Dixit**, Professor and Head, Department of Computer Science and Engineering, for his continued support during the course of this engagement, which eased the process and formalities involved.

I am sincerely thankful to my faculty mentors. I am grateful to the guidance of **Dr. Ranjeet Kumar Singh**, Assistant Professor, Computer Science and Engineering, for his continued support and guidance throughout the project. I am also very thankful to the faculty and staff of the department.

[illegible]

Rakhi Sehrawat
0901CS201095
II Year, 4th SEM
Computer Science and Engineering

ABSTRACT

The solitaire game 2048 was developed in 2014 by Gabriele Cirulli, based on another game called Threes developed earlier in 2014 by Asher Vollmer. It is played on a 16-cell square grid, each cell of which can either be empty or contain a tile labeled with a power of two. In
each turn, a tile of value 2 or 4 is placed by the game software on a randomly chosen empty cell. The player then must tilt the board in one of the four cardinal directions, causing its tiles to slide until reaching the edge of the board or another tile. When two tiles of equal value slide into each other, they merge into a new tile of twice the value. The game stops when the whole board fills with tiles, and the goal is to achieve the highest single tile value possible (2048) .

At any step of the game, there must be at least one tile for each nonzero bit in the binary representation of the total tile value. **For total tile values just below a large power of two, the number of ones in the binary representation is similarly large, eventually exceeding the number of cells in the board.**

Recent years have shown increased attention to **machine learning (ML)** for various purposes. ML is an efficient way to solve problems that would take too long or take too much resources otherwise. The game 2048 has complete information, the player can see the entire board, but the consequences of the player's actions are random. Additionally, there are very few possible actions, but many different board states.
Additionally, the game is very popular, which might attract attention to
this field of study, increasing the amount of research of formal verification techniques.

TABLE OF CONTENTS

<u>INTRODUCTION</u>

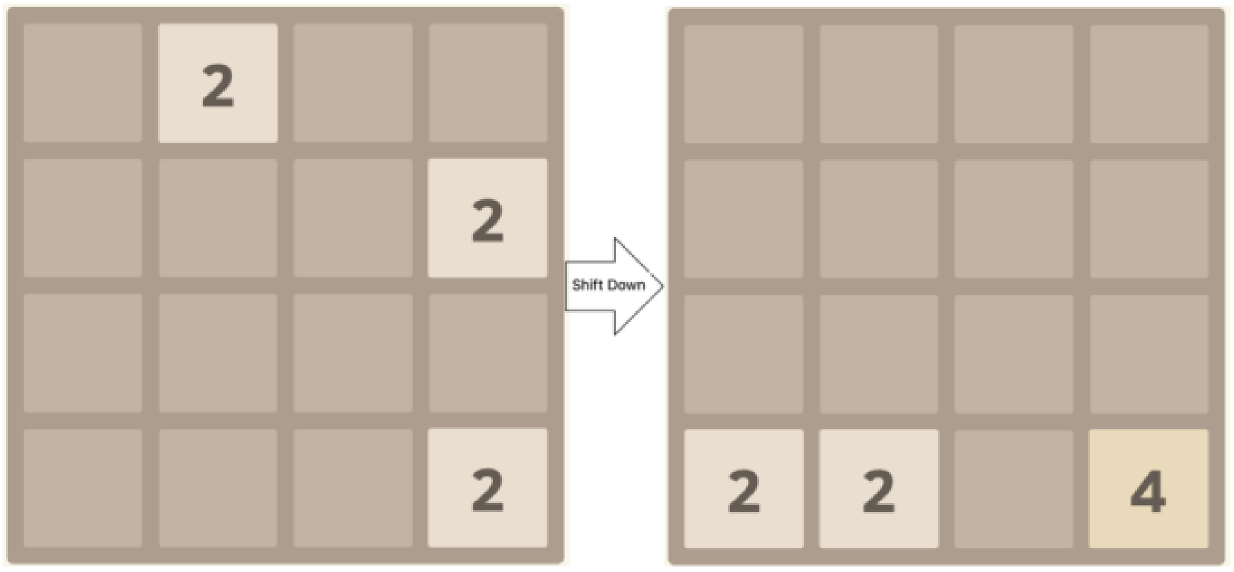
2048 is an exciting tile-shifting game, where we move tiles around to combine them, aiming for increasingly larger tile values.

How to Play 2048

A game of 2048 is played on a 4×4 board. Each position on the board may be empty or may contain a tile, and each tile will have a number on it.

When we start, the board will have two tiles in random locations, each of which either has a “2” or a “4” on it – each has an independent 10% chance of being a “4”, or otherwise a is a “2”.

Moves are performed by shifting all tiles towards one edge – up, down, left, or right. When doing this, any tiles with the same value that are adjacent to each other and are moving together will merge and end up with a new tile equal to the sum of the earlier two:



After we’ve made a move, a new tile will be placed onto the board. This is placed in a random location, and will be a “2” or a “4” in the same way as the initial tiles – “2” 90% of the time and “4” 10% of the time.

The game then continues until there are no more moves possible. **In general, the goal of the game is to reach a single tile with a value of "2048"**.

Problem Explanation

Solving this game is an interesting problem because it has a random component. It's impossible to correctly predict not only where each new tile will be placed, but whether it will be a "2" or a "4".

As such, it is impossible to have an algorithm that will correctly solve the puzzle every time. The best that we can do is determine what is likely to be the best move at each stage and play the probability game.

At any point, there are only four possible moves that we can make. Sometimes, some of these moves have no impact on the board and are thus not worth making – for example, in the above board a move of "Down" will have no impact since all of the tiles are already on the bottom edge.

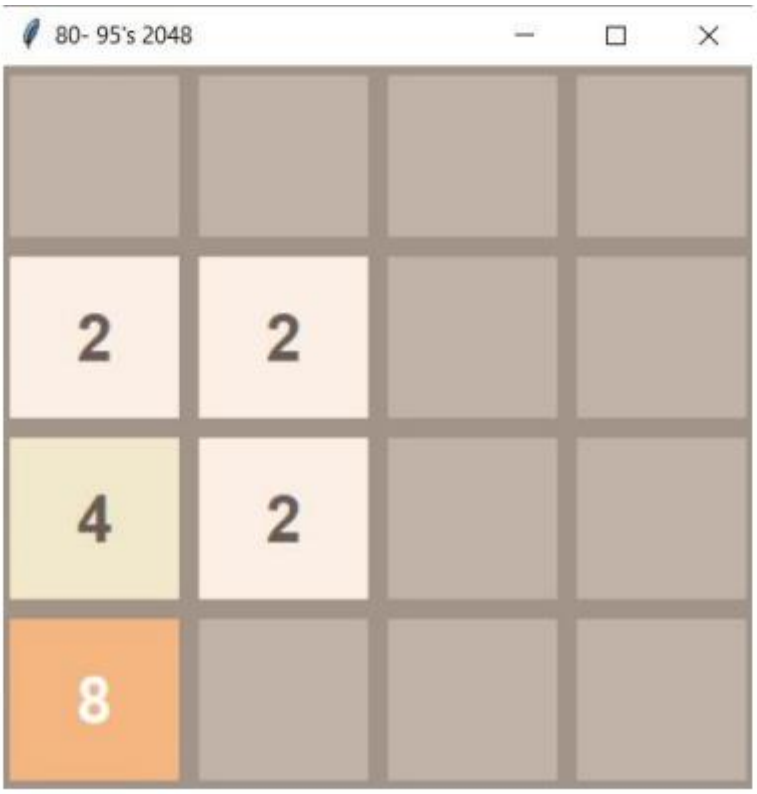
The challenge is then to determine which of these four moves is going to be the one that has the best long-term outcome.

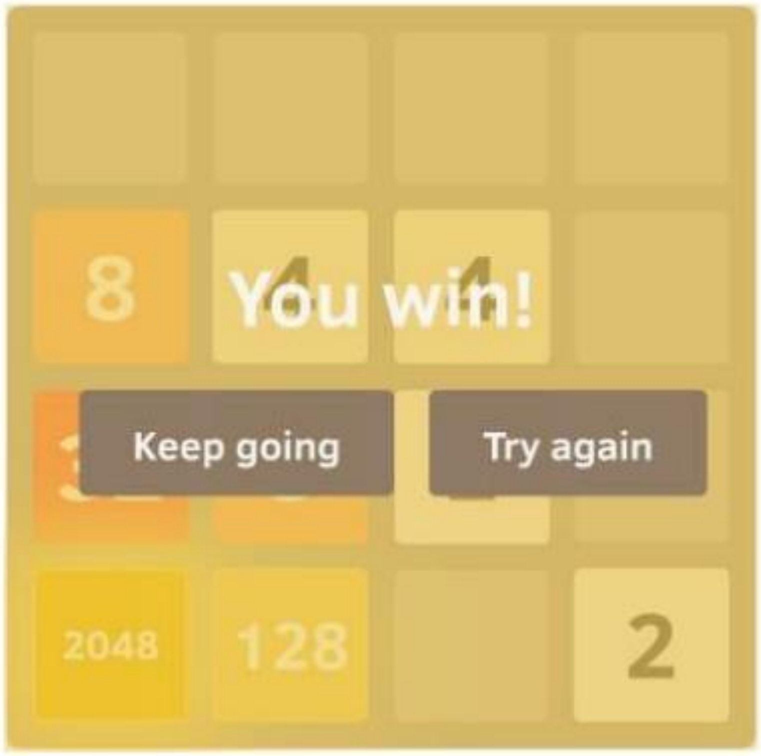
Essentially, we treat the game as a two-player game:

● Player One – the human player – can shift the board in one of four directions

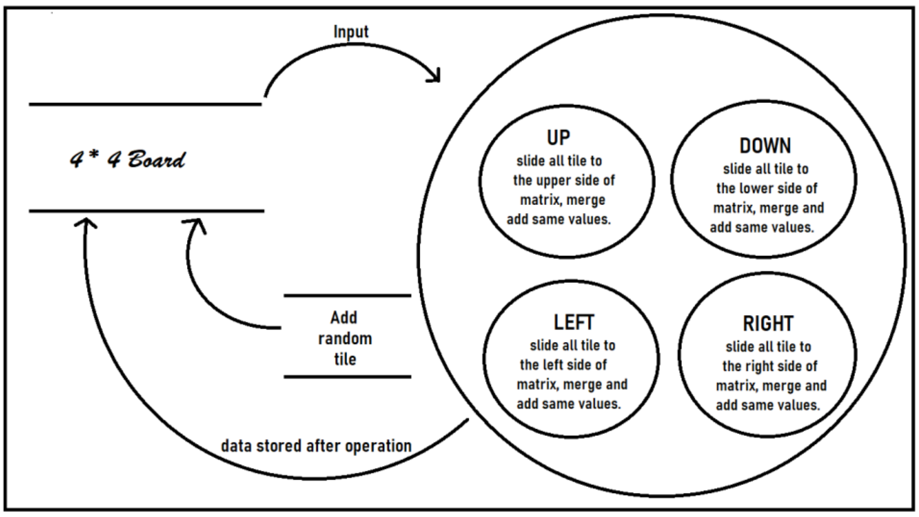
● Player Two – the computer player – can place a tile into an empty location on the board.

This can then give us the details needed to determine which human move is likely to give the best outcome.


80- 95's 2048
2
2
4
2
8


8
4
4
You win!
Keep going
Try again
2048
128
2

DFD (Data Flow Diagram) :


Input
4 * 4 Board
UP
slide all tile to the upper side of matrix, merge add same values.
DOWN
slide all tile to the lower side of matrix, merge and add same values.
LEFT
slide all tile to the left side of matrix, merge and add same values.
RIGHT
slide all tile to the right side of matrix, merge and add same values.
Add random tile
data stored after operation

OBJECTIVE

2048 is a single-player sliding block puzzle game designed by Italian web developer Gabriele Cirulli. The game’s objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048, in order to win the game.

2048 is a game where you combine numbered tiles in order to gain a higher numbered tile. In this game you start with two tiles, the lowest possible number available is two. Then you will play by combining the tiles with the same number to have a tile with the sum of the number on the two tiles. A lot of strategies have been devised in order to obtain the 2048 and be a winner in this game.

HARDWARE AND SOFTWARE USED

HARDWARE:

Processor	Intel(R) Core (TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
Installed RAM	8.00 GB (7.86 GB usable)
Product ID	00327-35896-14599-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

SOFTWARES:

VSCODE

Python 3.10.8

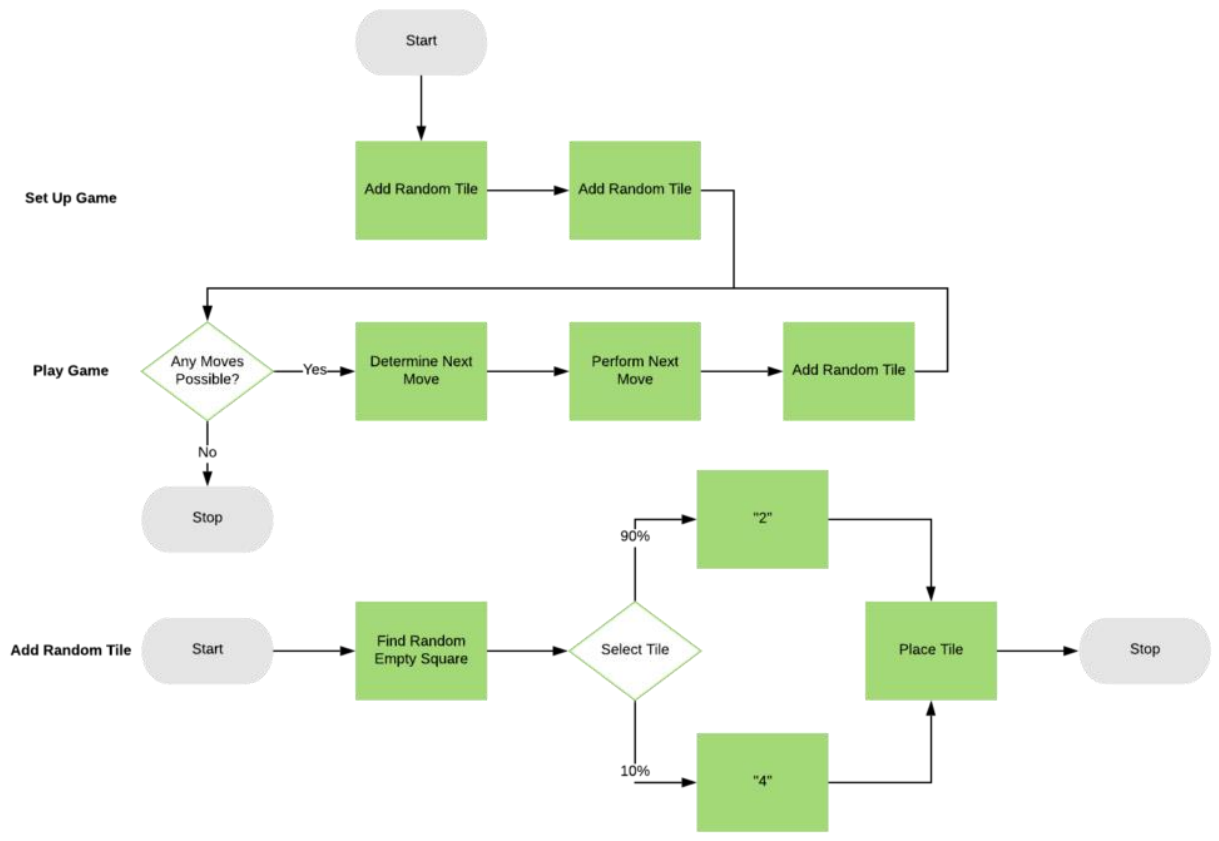
Tkinter

Random

METHODOLOGY AND FLOWCHART

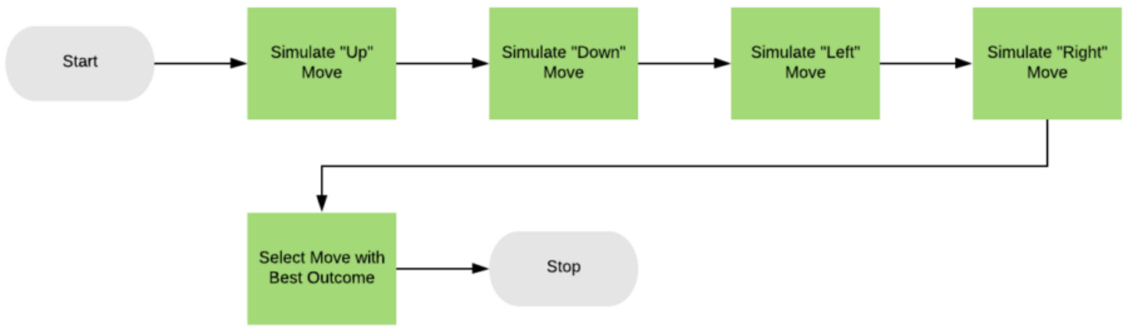
Flowchart of Gameplay

The general flow of how the gameplay works:



We can immediately see the random aspect of the game in the "Add Random Tile" process – both in the fact that we're finding a random square to add the tile to, and we're selecting a random value for the tile.

The general overflow of this seems deceptively simple:



All we need to do is simulate each of the possible moves, determine which one gives the best outcome, and then use that.

So we have now reduced our algorithm into simulating any given move and generating some score for the outcome.

This is a two-part process. The first pass is to see if the move is even possible, and if not, then abort early with a score of “0”. If the move is possible, then we’ll move on to the real algorithm where we determine how good a move this is:

Game Board

Before anything else, we need a game board. This is a grid of cells into which numbers can be placed.

A Computer Player and Placing Tiles

Now that we've got a game board, we want to be able to play with it. **The first thing we want is the computer player because this is a purely random player and will be exactly as needed later on**.

The computer player does nothing more than place a tile into a cell, so we need some way to achieve that on our board. We want to keep this as being immutable, so placing a tile will generate a brand new board in the new state.

A “Human” Player and Shifting Tiles

The next thing we need is a “human” player. **This isn't going to be the end goal, but a purely random player that picks a random direction to shift the tiles every time it makes a move.**
This will then act as a place that we can build upon to make our optimal player.

Playing the Game

We have enough components to play the game, albeit not very successfully. However, soon we will be improving the way that the Human class plays, and this will allow us to see the differences easily.

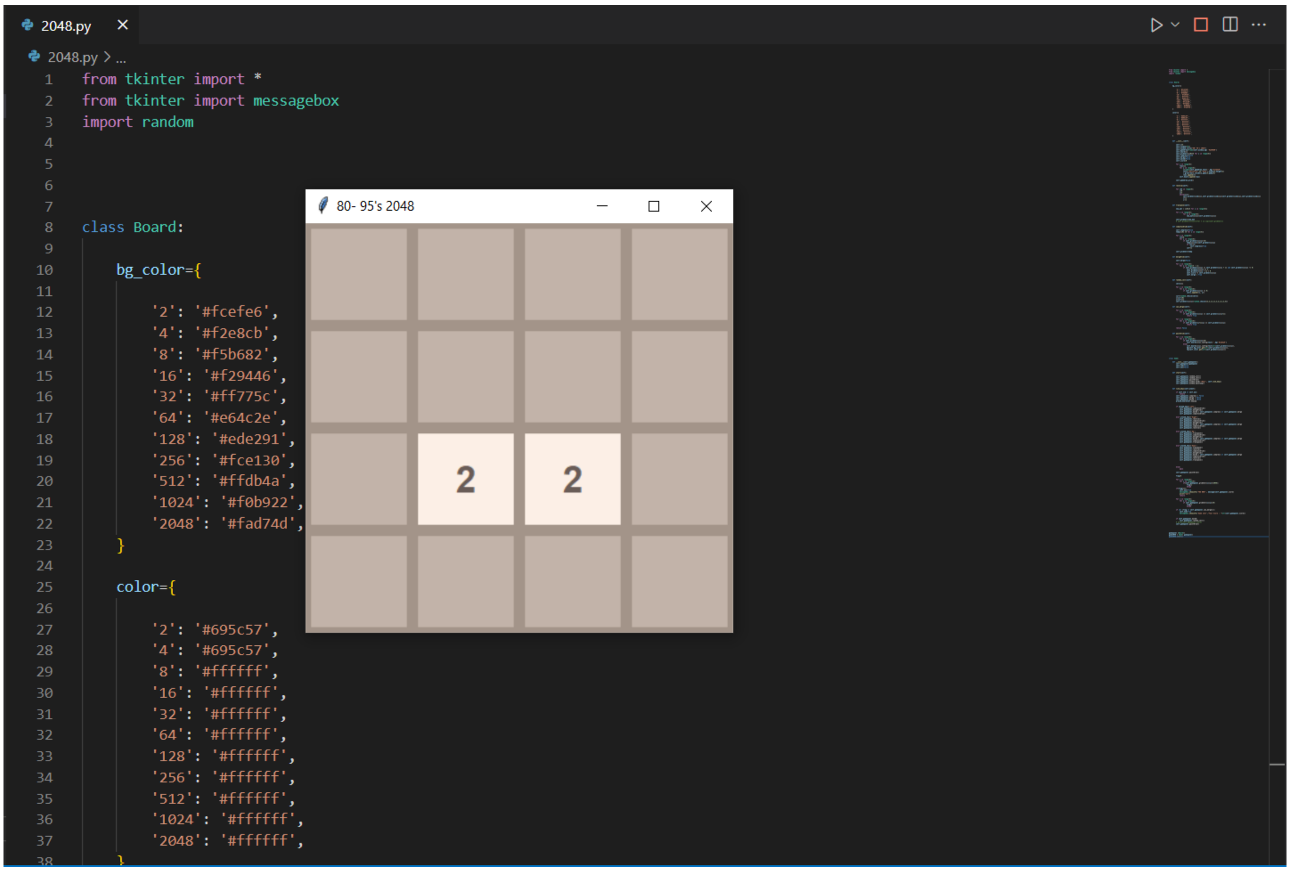
Implementing the 2048 Player

Once we have a base from which to play the game, we can start implementing the “human” player and play a better game than just picking a random direction.

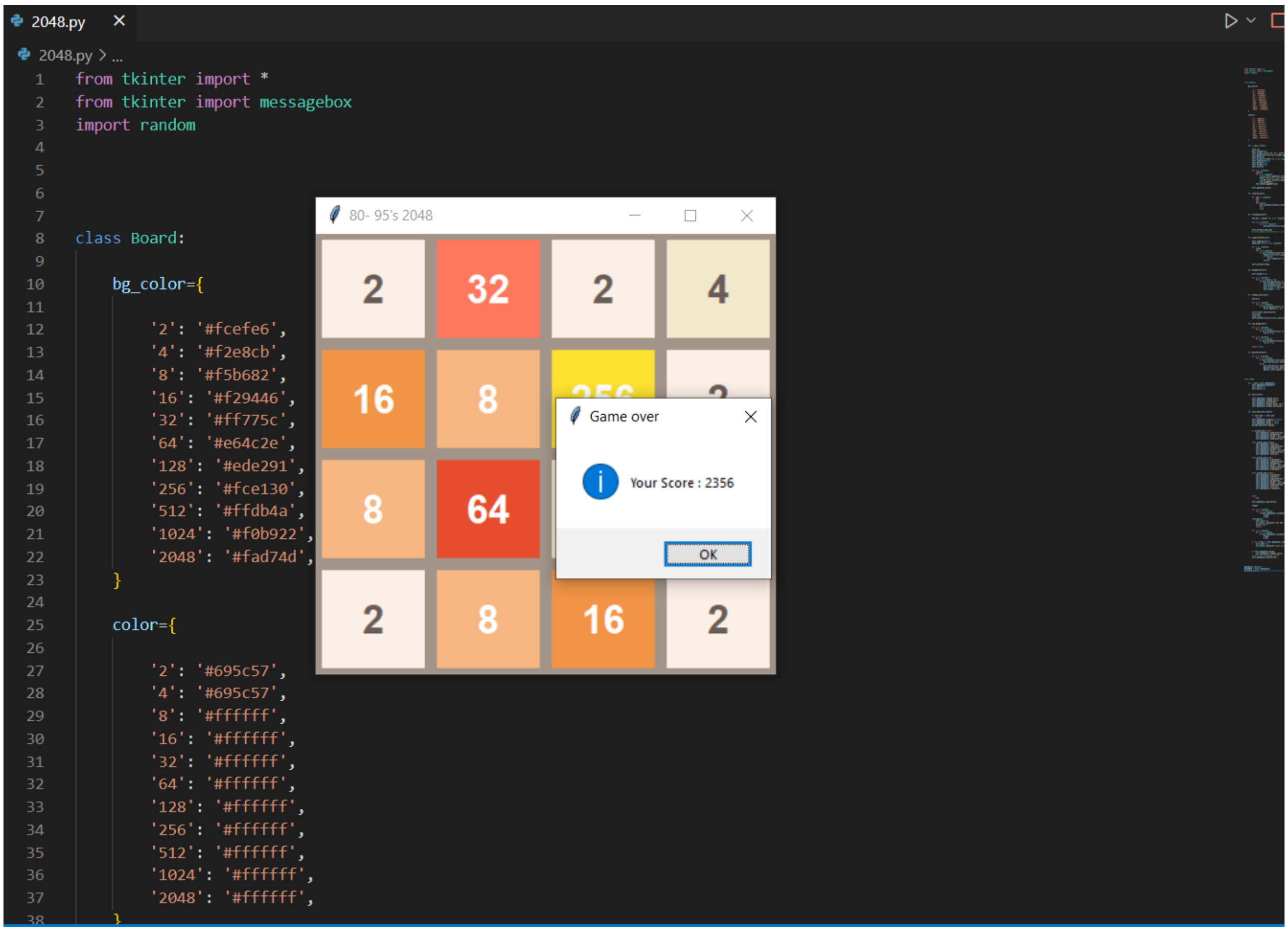
Scoring Final Boards

We're now in a situation where we can simulate moves back and forth by the human and computer players, stopping when we've simulated enough of them. **We need to be able to generate a score for the final board in each simulation branch, so that we can see which branch is the one we want to pursue.**

Our scoring is a combination of factors, each of which we are going to apply to every row and every column on the board. These all get summed together, and the total is returned.



```python
from tkinter import *
from tkinter import messagebox
import random



class Board:

    bg_color={

        '2': '#fcefe6',
        '4': '#f2e8cb',
        '8': '#f5b682',
        '16': '#f29446',
        '32': '#ff775c',
        '64': '#e64c2e',
        '128': '#ede291',
        '256': '#fce130',
        '512': '#ffdb4a',
        '1024': '#f0b922',
        '2048': '#fad74d',
    }

    color={

        '2': '#695c57',
        '4': '#695c57',
        '8': '#ffffff',
        '16': '#ffffff',
        '32': '#ffffff',
        '64': '#ffffff',
        '128': '#ffffff',
        '256': '#ffffff',
        '512': '#ffffff',
        '1024': '#ffffff',
        '2048': '#ffffff',
    }
```



```python
from tkinter import *
from tkinter import messagebox
import random




class Board:

    bg_color={

        '2': '#fcefe6',
        '4': '#f2e8cb',
        '8': '#f5b682',
        '16': '#f29446',
        '32': '#ff775c',
        '64': '#e64c2e',
        '128': '#ede291',
        '256': '#fce130',
        '512': '#ffdb4a',
        '1024': '#f0b922',
        '2048': '#fad74d',
    }

    color={

        '2': '#695c57',
        '4': '#695c57',
        '8': '#ffffff',
        '16': '#ffffff',
        '32': '#ffffff',
        '64': '#ffffff',
        '128': '#ffffff',
        '256': '#ffffff',
        '512': '#ffffff',
        '1024': '#ffffff',
        '2048': '#ffffff',
    }
```

```
Your Score : 2356
```

RESULTS AND DISCUSSION

We created an algorithm which plays 2048, comparing three different value methods: the reverse, transpose, compress which work differently and are used in order to compute the moves entered by users. These three methods are used to manipulate only one move, i.e, left move to evaluate and compute the results for all possible valid moves (left, right, up, down).

Using these moves in order make our algorithm refined, optimized and eradicates the overhead of designing four different algorithms associated with four different moves.

The idea of manipulating all possible moves to left move is :
Left : left_move_computation
Right : reverse, left_move_computation, reverse
Up : transpose, left_move_computation, transpose
Down : transpose, reverse, left_move_computation, reverse, transpose

CONCLUSION AND SCOPE OF THE PROJECT

Recent years have shown increased attention to **machine learning (ML)** for various purposes. ML is an efficient way to solve problems that would take too long or take too much resources otherwise. The game 2048 has complete information, the player can see the entire board, but the consequences of the player's actions are random. Additionally, there are very few possible actions, but many different board states.

Additionally, the game is very popular, which might attract attention to this field of study, increasing the amount of research of formal verification techniques.

2048 is a hugely interesting game to attempt to solve. There is no perfect way to solve it, but we can write heuristics that will search for the best possible routes through the game.

The same general principles work for any two-player game – for example, chess – where you can not predict what the other player will do with any degree of certainty.

REFERENCES

- https://www.w3schools.com/python/
- https://www.python.org/
- https://en.wikipedia.org/wiki/Python_(programming_language)
- https://www.programiz.com/python-programming/online-compiler/
- https://www.tutorialspoint.com/python/index.htm
- https://www.w3schools.in/python/gui-programming
- https://www.youtube.com/playlist?list=PLu0W_9lII9ajLcqRcj4PoEihkukF_OTzA
- https://www.youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV
- https://www.youtube.com/watch?v=YXPyB4XeYLA

APPENDICES :

```
from tkinter import *
from tkinter import messagebox
import random




class Board:

    bg_color={

        '2': '#fcefe6',
        '4': '#f2e8cb',
        '8': '#f5b682',
        '16': '#f29446',
        '32': '#ff775c',
        '64': '#e64c2e',
        '128': '#ede291',
        '256': '#fce130',
        '512': '#ffdb4a',
        '1024': '#f0b922',
        '2048': '#fad74d',
    }

    color={

        '2': '#695c57',
        '4': '#695c57',
        '8': '#ffffff',
        '16': '#ffffff',
        '32': '#ffffff',
        '64': '#ffffff',
        '128': '#ffffff',
        '256': '#ffffff',
    '512': '#ffffff',
    '1024': '#ffffff',
    '2048': '#ffffff',
}


def __init__(self):

    self.n=4
    self.window=Tk()
    self.window.title("80- 95's 2048")
    self.gameArea=Frame(self.window,bg= '#a39489')
    self.board=[]
    self.gridCell=[[0]*4 for i in range(4)]
    self.compress=False
    self.merge=False
    self.moved=False
    self.score=0

    for i in range(4):
        rows=[]
        for j in range(4):
            l=Label(self.gameArea,text='',bg='#c2b3a9',
            font=('Helvetica',25,'bold'),width=4,height=2)
            l.grid(row=i,column=j,padx=5,pady=5)
            rows.append(l)
        self.board.append(rows)

    self.gameArea.grid()


def reverse(self):

    for ind in range(4):
        i=0
        j=3
        while(i<j):
            self.gridCell[ind][i],self.gridCell[ind][j]=self.gridCell[ind][j],self.gridCell[ind][i]
            i+=1
            j-=1


def transpose(self):

    new_mat = [[0]*4 for i in range(4)]

    for i in range(4):
            for j in range(4):
                new_mat[i][j]=self.gridCell[j][i]

    self.gridCell=new_mat
    # self.gridCell=[list(t)for t in zip(*self.gridCell)]


def compressGrid(self):

    self.compress=False
    temp=[[0] *4 for i in range(4)]

    for i in range(4):
        cnt=0
        for j in range(4):
            if self.gridCell[i][j]!=0:
                temp[i][cnt]=self.gridCell[i][j]
                if cnt!=j:
                    self.compress=True
                cnt+=1

    self.gridCell=temp


def mergeGrid(self):

    self.merge=False
    for i in range(4):
        for j in range(4 - 1):
            if self.gridCell[i][j] == self.gridCell[i][j + 1] and self.gridCell[i][j] != 0:
                self.gridCell[i][j] *= 2
                self.gridCell[i][j + 1] = 0
                self.score += self.gridCell[i][j]
                self.merge = True


def random_cell(self):

    cells=[]

    for i in range(4):
        for j in range(4):
            if self.gridCell[i][j] == 0:
                cells.append((i, j))

    curr=random.choice(cells)
    i=curr[0]
    j=curr[1]
    self.gridCell[i][j]=random.choice([2,2,2,2,2,2,2,2,2,2,4])


def can_merge(self):

    for i in range(4):
        for j in range(3):
            if self.gridCell[i][j] == self.gridCell[i][j+1]:
                return True

    for i in range(3):
        for j in range(4):
            if self.gridCell[i+1][j] == self.gridCell[i][j]:
                return True
        return False


    def paintGrid(self):

        for i in range(4):
            for j in range(4):
                if self.gridCell[i][j]==0:
                    self.board[i][j].config(text='',bg='#c2b3a9')
                else:
                    self.board[i][j].config(text=str(self.gridCell[i][j]),
                    bg=self.bg_color.get(str(self.gridCell[i][j])),
                    fg=self.color.get(str(self.gridCell[i][j])))




class Game:

    def __init__(self,gamepanel):
        self.gamepanel=gamepanel
        self.end=False
        self.won=False


    def start(self):

        self.gamepanel.random_cell()
        self.gamepanel.random_cell()
        self.gamepanel.paintGrid()
        self.gamepanel.window.bind('<Key>', self.link_keys)
        self.gamepanel.window.mainloop()


    def link_keys(self,event):

        if self.end or self.won:
```

```
    return
self.gamepanel.compress = False
self.gamepanel.merge = False
self.gamepanel.moved = False
presed_key=event.keysym


if presed_key=='Left':
    self.gamepanel.compressGrid()
    self.gamepanel.mergeGrid()
    self.gamepanel.moved = self.gamepanel.compress or self.gamepanel.merge
    self.gamepanel.compressGrid()

elif presed_key=='Right':
    self.gamepanel.reverse()
    self.gamepanel.compressGrid()
    self.gamepanel.mergeGrid()
    self.gamepanel.moved = self.gamepanel.compress or self.gamepanel.merge
    self.gamepanel.compressGrid()
    self.gamepanel.reverse()

elif presed_key=='Up':
    self.gamepanel.transpose()
    self.gamepanel.compressGrid()
    self.gamepanel.mergeGrid()
    self.gamepanel.moved = self.gamepanel.compress or self.gamepanel.merge
    self.gamepanel.compressGrid()
    self.gamepanel.transpose()

elif presed_key=='Down':
    self.gamepanel.transpose()
    self.gamepanel.reverse()
    self.gamepanel.compressGrid()
    self.gamepanel.mergeGrid()
    self.gamepanel.moved = self.gamepanel.compress or self.gamepanel.merge
    self.gamepanel.compressGrid()
    self.gamepanel.reverse()
    self.gamepanel.transpose()




else:
    pass

self.gamepanel.paintGrid()

flag=0

for i in range(4):
    for j in range(4):
        if(self.gamepanel.gridCell[i][j]==2048):
            flag=1
            break
if(flag==1):
    self.won=True
    messagebox.showinfo('YOU WON!', message=self.gamepanel.score)
    print("won")
    return

for i in range(4):
    for j in range(4):
        if self.gamepanel.gridCell[i][j]==0:
            flag=1
            break

if not (flag or self.gamepanel.can_merge()):
    self.end=True
    messagebox.showinfo('Game over',"Your Score : "+str(self.gamepanel.score))



if self.gamepanel.moved:
    self.gamepanel.random_cell()
# print(self.gamepanel.score)
self.gamepanel.paintGrid()
```

```
gamepanel =Board()
game2048 = Game( gamepanel)
game2048.start()
```