COMPETITIVE CODING

An **Industrial Training Report** Submitted

in Partial Fulfilment of the Requirements

for the Degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

By

HARIPRIYA YADAV

(2002840109003)

Department of Computer Science

United Institute of Technology, Allahabad (Code 284)

Dr. A.P.J. Abdul Kalam Technical University, Lucknow

Under the Supervision of

Mr. DHANANJAY KUMAR SHARMA

UNITED INSTITUTE OF TECHNOLOGY, PRAYAGRAJ

60 hours (4 week -2 hours per day)

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my teacher Mr. Dhananjay Sir, who gave me golden opportunity to do this wonderful summer training internship on ‘COMPETATIVE CODING’.

Who also help me in completing my project. I came to know about so many new things I am really thankful to him.

HARIPRIYA YADAV
CSE 3RD YEAR

TOPIC: 2048

AGENDA

- Introduction
- History
- Method/Module Used
- Features of Backtracking
- About the game
- How to play
- Programming approach

INTRODUCTION:

The project aims to simulates the board game 2048. As indicated in the project requirement:

- Its output should always be a board in some state.
- The game should start in a random state, with the two starting blocks somewhere on the board.
- The user can then use the current board as input, together with their next move (up/down/left/right), with which the board is updated, and returned to the reader.
- Make use of colors to make the game look attractive and easier to read.
- Send the user a congratulatory message when they reach 2048, but the game may continue if you want.

History

2048 is a single-player sliding tile puzzle video game written by Italian web developer Gabriele Cirulli and published on GitHub. It was originally written in JavaScript and CSS over a weekend, and released on 9 March 2014 as free and open-source software subject to the MIT License. Nineteen-year-old Gabriele Cirulli created the game in a single weekend as a test to see if he could program a game from scratch. "It was a way to pass the time", he said. He described it as being "conceptually similar" to the recently released iOS game Three*s*, and a clone of another game, 1024. Developed by Veewo Studio, 1024 is itself a clone of Three*s*, with its App Store description once reading "no need to pay for Threes". Cirulli's README for 2048 cites another 1024 clone as influence: the homonymous but slightly different in terms of mechanics 2048 by Saming.

Cirulli was surprised when his weekend project received over 4 million visitors in less than a week, The game is free to play, Cirulli having said that he was unwilling to make money "from a concept that [he] didn't invent". He released ports for iOS and Android in May 2014.

Method/Module Used:

1. **Tkinter (tkinter):**
   Module for coding a user-friendly interface as introduced in class. It contains many function such as create_text(), create_rectangle(), label(), button(), bind_al()*l* that makes the user interface interactive.

. 2**. Backtracking**:

Backtracking is a complete backup solution with an array of features that allow you to easily schedule and retrieve the backup of your data.

<u>Features of Backtracking</u>

Backtracking is a complete backup solution with an array of features that allow you to easily schedule and retrieve the backup of your data. You can back up files and folders of any number of employees with the folder structure and filename preserved in the backup and use a one-point mechanism for LAN-based and online backup. You can choose either a full back up to be doubly sure that even unchanged data is backed up, or incremental backup where only changed data is backed up. We have included various features like Individual Folder selection, Individual file backup and Files in use backup. Apart from this, you can automate the process by scheduling a backup at pre-defined, regular intervals to reduce manual intervention, and you can choose either a repetitive or an absolute scheduling. Backtrack also offers various restoration features. You can restore file paths, or individual file/folders or restore information to a separate location. In effect, you are getting a central storage repository, as backup of data from all users is stored at one central location – added convenience to manage backup data. What's more, you can protect stored data with backup password protection and encryption. Other thoughtful features are data compression before backup to minimize both the amount of bandwidth used and storage space, and file filtering where you choose the type of files to be excluded for backup and filter out unnecessary content. We understand the need for organizations to have a smart, secure and reliable backup solution. Backtrack is exactly that. With its easy to use interface, you will be set up in record time – you can even go under the hood and make changes any time you wish. Its single administration and monitoring window makes it easy to manage.

About the Game:

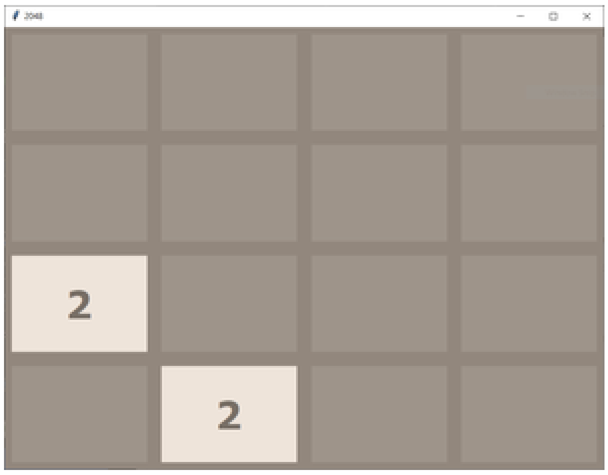
2048 is a popular single-player game for Web and mobile. It's a type of "sliding block puzzle" — think Threes! on which 2048 is based, or the old-timey game klotski — that's played on an almost Sudoku-like grid. Like Sudoku, it also involves some math. The object of the game is to combine the numbers displayed on the tiles until you reach 2048.

Basically, 2048 presents with with a 4×4 grid. When you start the game, there will be two "tiles" on the grid, each displaying the number 2 or 4. You hit the arrow keys on your keyboard to move the tiles around — and also to generate new tiles, which will also be valued at 2 or 4. When two equal tiles collide, they combine to give you one greater tile that displays their sum. The more you do this, obviously, the higher the tiles get and the more crowded the board becomes. Your objective is to reach 2048 before the board fills up.

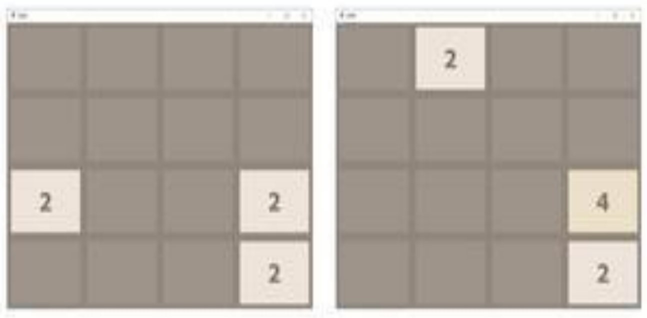
Yeah, so, the math behind 2048 is easy — but the game itself is very hard.

How to play:

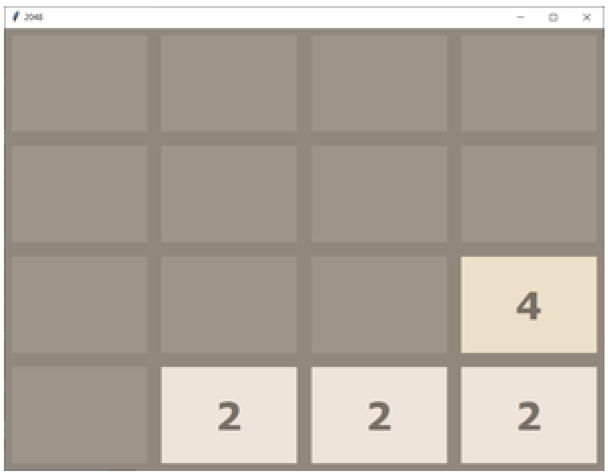
1. There is a 4*4 grid which can be filled with any number. Initially two random cells are filled with 2 in it. Rest cells are empty.



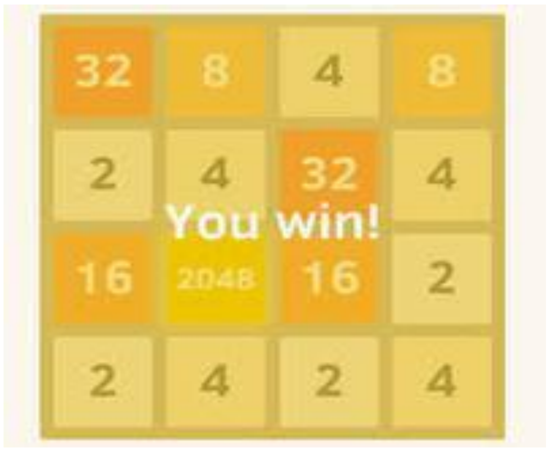
2. we have to press any one of four keys to move up, down, left, or right. When we press any key, the elements of the cell move in that direction such that if any two identical numbers are contained in that particular row (in case of moving left or right) or column (in case of moving up and down) they get add up and extreme cell in that direction fill itself with that number and rest cells goes empty again.



3. After this grid compression any random empty cell gets itself filled with2.



4. Following the above process we have to double the elements by adding up and make 2048 in any of the cell. If we are able to do that we wins.



Programming approach:

```
import tkinter as tk
import random
import time

class Game(tk.Tk):
    board = []
    new_tile_selection = [2,2,2,2,2,2,4]
    score = 0
    highscore = 0
    scorestring = 0
    highscorestring = 0

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.scorestring = tk.StringVar(self)
        self.scorestring.set("0")
        self.highscorestring = tk.StringVar(self)
        self.highscorestring.set("0")
        self.create_widgets()
        self.canvas = tk.Canvas(self, width=410, height=410, borderwidth=5,
highlightthickness=0)
```

```
        self.canvas.pack(side="top", fill="both", expand="false")
        self.new_game()

    #Adds 1 new tiles to board in empty spaces, highlights tile
    def addNewTile(self):
        index = random.randint(0,6)
        x = -1
        y = -1
        while self.isFull() == False:
            x = random.randint(0,3)
            y = random.randint(0,3)
            if (self.board[x][y] == 0):
                self.board[x][y] = self.new_tile_selection[index]
                x1 = y*105
                y1 = x*105
                x2 = x1 + 105 - 5
                y2 = y1 + 105 - 5
                num = self.board[x][y]
                if num == 2:
                    self.square[x,y]                                =
self.canvas.create_rectangle(x1,y1,x2,y2,    fill="#e0f2f8",    tags="rect",
outline="", width=0)
                    self.canvas.create_text((x1    +    x2)/2,    (y1+y2)/2,
font=("Arial", 36), fill="#f78a8a", text="2")
                elif num == 4:
                    self.square[x,y]                                =
self.canvas.create_rectangle(x1,y1,x2,y2,    fill="#b8dbe5",    tags="rect",
outline="", width=0)
                    self.canvas.create_text((x1    +    x2)/2,    (y1+y2)/2,
font=("Arial", 36), fill="#f78a8a", text="4")


                break
    #Returns True if board is full
    def isFull(self):
        for i in range(0,4):
            for j in range(0,4):
                if (self.board[i][j] == 0):
                    return False
        return True
    #Prints game board
    def printboard(self):
        cellwidth = 105
        cellheight = 105
        self.square = {}

        for column in range(4):
            for row in range(4):
                x1 = column*cellwidth
                y1 = row*cellheight
                x2 = x1 + cellwidth - 5
                y2 = y1 + cellheight - 5
                num = self.board[row][column]
                if num == 0:
                    self.print0(row, column, x1, y1, x2, y2)
                elif num == 2:
                    self.print2(row, column, x1, y1, x2, y2)
                elif num == 4:
                    self.print4(row, column, x1, y1, x2, y2)
                elif num == 8:
                    self.print8(row, column, x1, y1, x2, y2)
                elif num == 16:
                    self.print16(row, column, x1, y1, x2, y2)
                elif num == 32:
                    self.print32(row, column, x1, y1, x2, y2)
                elif num == 64:
                    self.print64(row, column, x1, y1, x2, y2)
                elif num == 128:
                    self.print128(row, column, x1, y1, x2, y2)
                elif num == 256:
                    self.print256(row, column, x1, y1, x2, y2)
                elif num == 512:
                    self.print512(row, column, x1, y1, x2, y2)
                elif num == 1024:
                    self.print1024(row, column, x1, y1, x2, y2)
                elif num == 2048:
                    self.print2048(row, column, x1, y1, x2, y2)
    def print0(self, row, column, x1, y1, x2, y2):
        self.square[row,column] = self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#f5f5f5", tags="rect", outline="")
    def print2(self, row, column, x1, y1, x2, y2):
        self.square[row,column] = self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#e0f2f8", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 36),
fill="#494949", text="2")
    def print4(self, row, column, x1, y1, x2, y2):
        self.square[row,column] = self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#b8dbe5", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 36),
fill="#494949", text="4")
    def print8(self, row, column, x1, y1, x2, y2):
        self.square[row,column] = self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#71b1bd", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 36),
fill="white", text="8")
    def print16(self, row, column, x1, y1, x2, y2):
        self.square[row,column] = self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#27819f", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 36),
fill="white", text="16")
    def print32(self, row, column, x1, y1, x2, y2):
        self.square[row,column] = self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#0073b9", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 36),
fill="white", text="32")
    def print64(self, row, column, x1, y1, x2, y2):
        self.square[row,column] = self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#7fa8d7", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 36),
fill="white", text="64")
    def print128(self, row, column, x1, y1, x2, y2):
        self.square[row,column] = self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#615ea6", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 32),
fill="white", text="128")
    def print256(self, row, column, x1, y1, x2, y2):
        self.square[row,column]  =  self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#2f3490", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 32),
fill="white", text="256")
    def print512(self, row, column, x1, y1, x2, y2):
        self.square[row,column]  =  self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#1c1862", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 32),
fill="white", text="512")
    def print1024(self, row, column, x1, y1, x2, y2):
        self.square[row,column]  =  self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#9c005d", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 30),
fill="white", text="1024")
    def print2048(self, row, column, x1, y1, x2, y2):
        self.square[row,column]  =  self.canvas.create_rectangle(x1,y1,x2,y2,
fill="#c80048", tags="rect", outline="")
        self.canvas.create_text((x1 + x2)/2, (y1+y2)/2, font=("Arial", 30),
fill="white", text="2048")

    #Creates buttons at top of screen
    def create_widgets(self):
        self.buttonframe = tk.Frame(self)
        self.buttonframe.grid(row=2, column=0, columnspan=4)
        tk.Button(self.buttonframe,           text           =           "New
Game",command=self.new_game).grid(row=0, column=0)
        tk.Label(self.buttonframe, text = "Score:").grid(row=0, column=1)
        tk.Label(self.buttonframe,
textvariable=self.scorestring).grid(row=0, column=2)
        tk.Label(self.buttonframe, text = "Record:").grid(row=0, column=3)
        tk.Label(self.buttonframe,
textvariable=self.highscorestring).grid(row=0, column=4)
        self.buttonframe.pack(side="top")

    #executes moves based on arroy keys pressed
    def keyPressed(self,event):
        shift = 0
        if event.keysym == 'Down':
            for j in range(0,4):
                shift = 0
                for i in range(3,-1,-1):
                    if self.board[i][j] == 0:
                        shift += 1
                    else:
                        if i - 1 >= 0 and self.board[i-1][j] ==
self.board[i][j]:
                            self.board[i][j] *= 2
                            self.score += self.board[i][j]
                            self.board[i-1][j] = 0
                        elif i - 2 >= 0 and self.board[i-1][j] == 0 and
self.board[i-2][j] == self.board[i][j]:
                            self.board[i][j] *= 2
                            self.score += self.board[i][j]
                            self.board[i-2][j] = 0
                        elif i == 3 and self.board[2][j] + self.board[1][j]
== 0 and self.board[0][j] == self.board[3][j]:
                            self.board[3][j] *= 2
                            self.score += self.board[3][j]
                            self.board[0][j] = 0
                        if shift > 0:
                            self.board[i+shift][j] = self.board[i][j]
                            self.board[i][j] = 0
            self.printboard()
            self.addNewTile()
            self.isOver()
        elif event.keysym == 'Right':
            for i in range(0,4):
                shift = 0
                for j in range(3,-1,-1):
                    if self.board[i][j] == 0:
                        shift += 1
                    else:
                        if j - 1 >= 0 and self.board[i][j-1] ==
self.board[i][j]:
                            self.board[i][j] *= 2
                            self.score += self.board[i][j]
                            self.board[i][j-1] = 0
                        elif j - 2 >= 0 and self.board[i][j-1] == 0 and
self.board[i][j-2] == self.board[i][j]:
```

```
                            self.board[i][j] *= 2
                            self.score += self.board[i][j]
                            self.board[i][j-2] = 0
                        elif j == 3 and self.board[i][2] + self.board[i][1]
== 0 and self.board[0][j] == self.board[3][j]:
                            self.board[i][3] *= 2
                            self.score += self.board[i][3]
                            self.board[i][0] = 0
                        if shift > 0:
                            self.board[i][j+shift] = self.board[i][j]
                            self.board[i][j] = 0
            self.printboard()
            self.addNewTile()
            self.isOver()
        elif event.keysym == 'Left':
            for i in range(0,4):
                shift = 0
                for j in range(0,4):
                    if self.board[i][j] == 0:
                        shift += 1
                    else:
                        if j + 1 < 4 and self.board[i][j+1] ==
self.board[i][j]:
                            self.board[i][j] *= 2
                            self.score += self.board[i][j]
                            self.board[i][j+1] = 0
                        elif j + 2 < 4 and self.board[i][j+1] == 0 and
self.board[i][j+2] == self.board[i][j]:
                            self.board[i][j] *= 2
                            self.score += self.board[i][j]
                            self.board[i][j+2] = 0
                        elif j == 0 and self.board[i][1] + self.board[i][2]
== 0 and self.board[i][3] == self.board[i][0]:
                            self.board[i][0] *= 2
                            self.score += self.board[i][0]
                            self.board[i][3] = 0
                        if shift > 0:
                            self.board[i][j-shift] = self.board[i][j]
                            self.board[i][j] = 0
            self.printboard()
            self.addNewTile()
            self.isOver()
        elif event.keysym == 'Up':
            for j in range(0,4):
                shift = 0
                for i in range(0,4):
                    if self.board[i][j] == 0:
                        shift += 1
                    else:
                        if   i   +   1   <   4   and   self.board[i+1][j]   ==
self.board[i][j]:
                            self.board[i][j] *= 2
                            self.score += self.board[i][j]
                            self.board[i+1][j] = 0
                        elif  i  +  2  <  4  and  self.board[i+1][j]  ==  0  and
self.board[i+2][j] == self.board[i][j]:
                            self.board[i][j] *= 2
                            self.score += self.board[i][j]
                            self.board[i+2][j] = 0
                        elif i == 0 and self.board[1][j] + self.board[2][j]
== 0 and self.board[3][j] == self.board[0][j]:
                            self.board[0][j] *= 2
                            self.score += self.board[0][j]
                            self.board[3][j] = 0
                        if shift > 0:
                            self.board[i-shift][j] = self.board[i][j]
                            self.board[i][j] = 0
            self.printboard()
            self.addNewTile()
            self.isOver()
        self.scorestring.set(str(self.score))
        if self.score > self.highscore:
            self.highscore = self.score
            self.highscorestring.set(str(self.highscore))


    def new_game(self):
        self.score = 0
        self.scorestring.set("0")
        self.board = []
        self.board.append([0,0,0,0])
        self.board.append([0,0,0,0])
        self.board.append([0,0,0,0])
        self.board.append([0,0,0,0])
        while True:
            x = random.randint(0,3)
            y = random.randint(0,3)
            if (self.board[x][y] == 0):
                self.board[x][y] = 2
                break

        index = random.randint(0,6)
        while self.isFull() == False:
            x = random.randint(0,3)
            y = random.randint(0,3)
            if (self.board[x][y] == 0):
                self.board[x][y] = self.new_tile_selection[index]
                break
        self.printboard()


    #Returns True if board is full & has no more moves
    def isOver(self):
        for i in range(0,4):
            for j in range(0,4):
                if (self.board[i][j] == 2048):
                    self.youWon()
        for i in range(0,4):
            for j in range(0,4):
                if (self.board[i][j] == 0):
                    return False
        for i in range(0,4):
            for j in range(0,3):
                if (self.board[i][j] == self.board[i][j+1]):
                    return False
        for j in range(0,4):
            for i in range(0,3):
                if self.board[i][j] == self.board[i+1][j]:
                    return False
        gameover = [["G", "A", "M", "E",],["O", "V", "E", "R"], ["", "", "",
""],  ["", "", "", ""]]
        cellwidth = 105
        cellheight = 105
        self.square = {}

        for column in range(4):
            for row in range(4):
                x1 = column*cellwidth
                y1 = row*cellheight
                x2 = x1 + cellwidth - 5
                y2 = y1 + cellheight - 5
                self.square[row,column] =
self.canvas.create_rectangle(x1,y1,x2,y2, fill="#e0f2f8", tags="rect",
outline="")
                self.canvas.create_text((x1 + x2)/2, (y1+y2)/2,
font=("Arial", 36), fill="#494949", text=gameover[row][column])
        return True
    def youWon(self):
        gameover = [["Y", "O", "U", "",],["W", "O", "N", "!"], ["", "", "",
""],  ["", "", "", ""]]
        cellwidth = 105
        cellheight = 105
        self.square = {}
        for column in range(4):
            for row in range(4):
                x1 = column*cellwidth
                y1 = row*cellheight
                x2 = x1 + cellwidth - 5
                y2 = y1 + cellheight - 5
                self.square[row,column] =
self.canvas.create_rectangle(x1,y1,x2,y2, fill="#e0f2f8", tags="rect",
outline="")
                self.canvas.create_text((x1 + x2)/2, (y1+y2)/2,
font=("Arial", 36), fill="#494949", text=gameover[row][column])
```

```
if __name__ == "__main__":
    app = Game()
    app.bind_all('<Key>', app.keyPressed)
    app.wm_title("2048")
    app.minsize(420,450)
    app.maxsize(420,450)
    app.mainloop()
```

REFERENCE:

1. Google.com
2. Slide Share
3. Slide In
4. Coursera
5. P4YE
6. Wikipedia

7. Python
8. https://www.geeksforgeeks.org/
9. https://mrcet.com/downloads/digital_notes/CSE/IV%20Year/MACHINE%20LEARNING(R17A0534).pdf
10. https://allsoftsolutions.skillsnetwork.site/
11. http://bit.ly/ibm-intro-ds
12. https://www.javatpoint.com/
13. https://github.com/
14. https://www.quora.com/
15. https://www.edureka.co/