# DATA STRUCTURES & ALGORITHMS 2

[19AIE203]

**S3 B.TECH CSE (AIE A)**

# 2048 GAME

## A Project Report

*submitted by members of* ***Group 26***

| S.NO | ROLL NUMBER | NAME |
|---|---|---|
| 1 | AM.EN.U4AIE20042 | KONIJETI SRI VYSHNAVI |
| 2 | AM.EN.U4AIE20049 | METHUKU SAMHITHA |


AMRITA SCHOOL OF ENGINEERING
AMRITA VISHWA VIDYAPEETHAM

**AMRITAPURI 690 525**

**FEBRUARY 2021**

# ***ABSTRACT :***

*2048 is an easy and fun puzzle game. It is a fun and easy-to-play board game , appropriate for all ages, which aims to become a modern classic. Join the numbers and get to the 2048 tile. Even if you don't love numbers you will love this game. It is played on a 4x4 grid using the arrows or keys alternatively. Every time you press a key - all tiles slide.*

*Tiles with the same value that bump into one-another are merged. Although there might be an optimal strategy to play, there is always some level of chance.* ***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".*

## Contents

# INTRODUCTION :

*2048 is a single-player sliding tile puzzle video game written by Italian web developer Gabriele Cirulli and published on GitHub. The objective of the game is to slide numbered tiles on a grid to combine them to create a tile with the number 2048; however, one can continue to play the game after reaching the goal, creating tiles with larger numbers. 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. Versions for iOS and Android followed in May 2014. 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. 2048 is an exciting tile-shifting game, where we move tiles around to combine them, aiming for increasingly larger tile values. If you beat the game and would like to master it, try to finish with a smaller score. That would mean that you finished with fewer moves. This game is mobile compatible and you can play it on any device - iPhone, iPad or any other smartphone.*


App icon

# HOW TO PLAY :

*2048 is a one-player puzzle slide game where the purpose of the game is to slide numbered numbers into a grid to be assembled to produce a 2048 value tile, or higher. The game is played on a 4x4 grid with smooth numbered numbers in one of the four areas where the tiles slide as much as possible in the selected area until the other tile or edge is suspended, and when two tiles with the same*

*number collide, they combine with double the value of the tile. When a movement creates three consecutive tiles of the same value to slide together only two tiles will come together. If all four spaces in a row or column have the same value the first two and the last two will meet. Each time a new tile will appear randomly in a blank space on the board. In this version the game is won when the tile reaches the value of 2048.*

## LOGIC OF THE CODE :

***We used 3 files namely constants.py, logic.py and 2048.py and their roles are written below:***

- ***constants.py :***

*We initialise the size, length of the grid and grid padding along with the colour of the cell, grid and also the font of the numbers.*

*Also specifying the "KEY_QUIT", "KEY_BACK", "KEY_UP", "KEY_DOWN", "KEY_LEFT", "KEY_RIGHT".*

- ***logic.py :***

❖ *logic.py to be imported in the 2048.py file.*

*Importing packages like random and initialises a 4x4 null matrix in the "start_game()" and adding 2 after every movement. We get the current state of the game through "get_current_state()" which returns "WON", "GAME NOT OVER" and "LOST" under required conditions*

*To perform every operation we have to compress, merge and then compress the grid again. The compress can be done through reversing and transposing the matrix. Therefore "transpose()"," cover_up()", "merge()" and "reverse()" are used accordingly to perform up, down, left and right operations.*

- ***2048.py :***

*Importing the logic.py file where we have written all the logic functions is done in 2048. Along with that we also look into the*

*visualizations of the grid like the title, size of the grid and cell, grid padding, colour of empty cell and a cell containing item in it.*

*We used history_matrixs so that we can perform the back operations using "b" and print "YOU WIN!" and "YOU LOSE!" at specified positions.*

## FUTURE WORKS :

- *In our version of the game our code only adds on a 2 in an empty cell for every move. In our future works we are planning in such a way that every turn a new tile will randomly appear in an empty spot on the board with a 90% chance of being a 2 and 10% chance of being a 4.*
- *We are planning to develop an AI for 2048 in an efficient way to solve the puzzle on its own through expectimax algorithm.*

## CODE :

### Constants.py :

```
SIZE = 400

GRID_LEN = 4

GRID_PADDING = 10


BACKGROUND_COLOR_GAME = "#92877d"

BACKGROUND_COLOR_CELL_EMPTY = "#9e948a"


BACKGROUND_COLOR_DICT = {

2:      "#eee4da",

4:      "#ede0c8",

8:      "#f2b179",

16:     "#f59563",
```

```
32:    "#f67c5f",
64:    "#f65e3b",
128:   "#edcf72",
256:   "#edcc61",
512:   "#edc850",
1024:  "#edc53f",
2048:  "#edc22e",
4096:  "#eee4da",
8192:  "#edc22e",
16384: "#f2b179",
32768: "#f59563",
65536: "#f67c5f",
}

CELL_COLOR_DICT = {
2:   "#776e65",
4:     "#776e65",
8:     "#f9f6f2",
16:    "#f9f6f2",
32:    "#f9f6f2",
64:    "#f9f6f2",
128:   "#f9f6f2",
256:   "#f9f6f2",
512:   "#f9f6f2",
1024:  "#f9f6f2",
2048:  "#f9f6f2",
4096:  "#776e65",
```

```
8192:   "#f9f6f2",
16384:  "#776e65",
32768:  "#776e65",
65536:  "#f9f6f2",
}

FONT = ("Verdana",40,"bold")

KEY_QUIT = "Escape"
KEY_BACK = "b"

KEY_UP = "Up"
KEY_DOWN = "Down"
KEY_LEFT = "Left"
KEY_RIGHT = "Right"

KEY_UP_ALT1 = "w"
KEY_DOWN_ALT1 = "s"
KEY_LEFT_ALT1 = "a"
KEY_RIGHT_ALT1 = "d"

KEY_UP_ALT2 = "i"
KEY_DOWN_ALT2 = "k"
KEY_LEFT_ALT2 = "j"
KEY_RIGHT_ALT2 = "l"
```

## logic.py :

```
import random
```

```
import constants as c


def new_game(n):
        matrix = []
        for i in range(n):
        matrix.append([0] *n)
        matrix = add_two(matrix)
        matrix = add_two(matrix)
        return matrix


def add_two(mat):
        a = random.randint(0, len(mat)-1)
        b = random.randint(0, len(mat)-1)
        while mat[a][b] != 0:
        a = random.randint(0, len(mat)-1)
        b = random.randint(0, len(mat)-1)
        mat[a][b] = 2
        return mat


def game_state(mat):
        for i in range(len(mat)):
        for j in range(len(mat[0])):
        if mat[i][j] == 2048:
        return 'win'
        for i in range(len(mat)):
        for j in range(len(mat[0])):
        if mat[i][j] == 0:
        return 'not over'
        for i in range(len(mat)-1):
```

```
            for j in range(len(mat[0])-1):
            if mat[i][j] == mat[i+1][j] or mat[i][j+1] == mat[i][j]:
            return 'not over'
            for k in range(len(mat)-1):
            if mat[len(mat)-1][k] == mat[len(mat)-1][k+1]:
            return 'not over'
            for j in range(len(mat)-1):
            if mat[j][len(mat)-1] == mat[j+1][len(mat)-1]:
            return 'not over'
            return 'lose'


def reverse(mat):
            new = []
            for i in range(len(mat)):
            new.append([])
            for j in range(len(mat[0])):
          new[i].append(mat[i][len(mat[0])-j-1])
            return new


def transpose(mat):
            new = []
            for i in range(len(mat[0])):
            new.append([])
            for j in range(len(mat)):
            new[i].append(mat[j][i])
            return new


def cover_up(mat):
            new = []
```

```
        for j in range(c.GRID_LEN):
        partial_new = []
        for i in range(c.GRID_LEN):
        partial_new.append(0)
        new.append(partial_new)
        done = False
        for i in range(c.GRID_LEN):
        count = 0
        for j in range(c.GRID_LEN):
        if mat[i][j] != 0:
        new[i][count] = mat[i][j]
        if j != count:
                done = True
        count += 1
        return new, done


def merge(mat, done):
        for i in range(c.GRID_LEN):
        for j in range(c.GRID_LEN-1):
        if mat[i][j] == mat[i][j+1] and mat[i][j] != 0:
        mat[i][j] *= 2
        mat[i][j+1] = 0
        done = True
        return mat, done


def up(game):
        print("up")
        print(game)
        game = transpose(game)
```

```
        game, done = cover_up(game)
        game, done = merge(game, done)
        game = cover_up(game)[0]
        game = transpose(game)
        return game, done

def down(game):
        print("down")
        game = reverse(transpose(game))
        game, done = cover_up(game)
        game, done = merge(game, done)
        game = cover_up(game)[0]
        game = transpose(reverse(game))
        return game, done

def left(game):
        print("left")
        game, done = cover_up(game)
        game, done = merge(game, done)
        game = cover_up(game)[0]
        return game, done

def right(game):
        print("right")
        game = reverse(game)
        game, done = cover_up(game)
        game, done = merge(game, done)
        game = cover_up(game)[0]
        game = reverse(game)
```

```
        return game, done
```

## 2048.py :

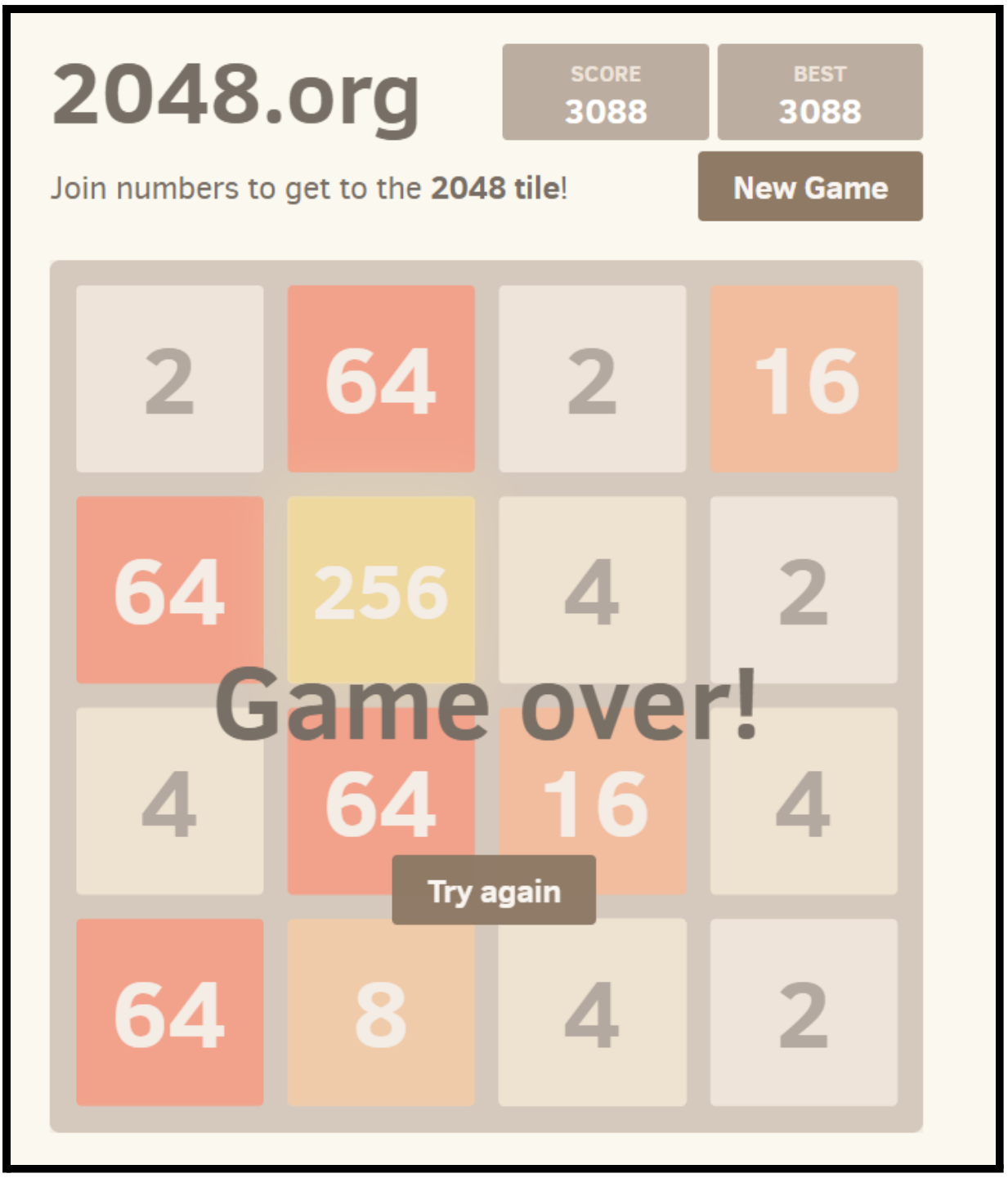
```
 from tkinter import Frame, Label, CENTER
import random
import logic
import constants as c


def gen():
        return random.randint(0, c.GRID_LEN - 1)


class GameGrid(Frame):
        def __init__(self):
        Frame.__init__(self)


        self.grid()
        self.master.title('2048')
      self.master.bind("<Key>", self.key_down)


        self.commands = {
        c.KEY_UP: logic.up,
        c.KEY_DOWN: logic.down,
        c.KEY_LEFT: logic.left,
        c.KEY_RIGHT: logic.right,
        c.KEY_UP_ALT1: logic.up,
        c.KEY_DOWN_ALT1: logic.down,
        c.KEY_LEFT_ALT1: logic.left,
        c.KEY_RIGHT_ALT1: logic.right,
        c.KEY_UP_ALT2: logic.up,
```

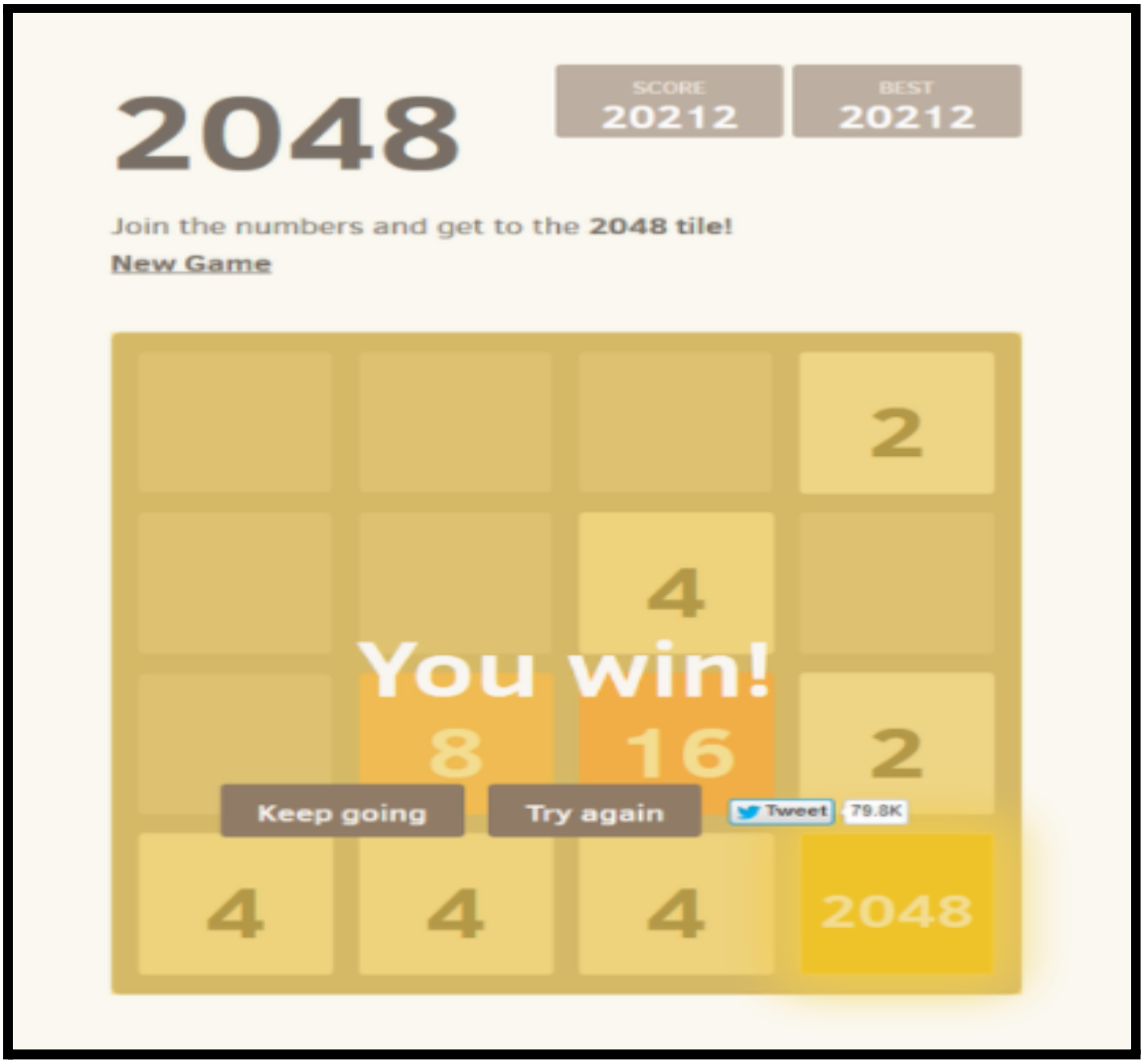
```
        c.KEY_DOWN_ALT2: logic.down,
        c.KEY_LEFT_ALT2: logic.left,
        c.KEY_RIGHT_ALT2: logic.right,
        }

        self.grid_cells = []
        self.init_grid()
        self.matrix = logic.new_game(c.GRID_LEN)
        self.history_matrixs = []
    print(self.history_matrixs)
        self.update_grid_cells()
        self.mainloop()

        def init_grid(self):
        background = Frame(self, bg=c.BACKGROUND_COLOR_GAME, width=c.SIZE, height=c.SIZE)
        background.grid()

        for i in range(c.GRID_LEN):
        grid_row = []
        for j in range(c.GRID_LEN):
        cell = Frame(
                    background,
                bg=c.BACKGROUND_COLOR_CELL_EMPTY,
                    width=c.SIZE / c.GRID_LEN,
                    height=c.SIZE / c.GRID_LEN
        )
        cell.grid(
                    row=i,
                    column=j,
```

```
            padx=c.GRID_PADDING,
             pady=c.GRID_PADDING
)
t = Label(
            master=cell,
            text="",
    bg=c.BACKGROUND_COLOR_CELL_EMPTY,
            justify=CENTER,
            font=c.FONT,
             width=5,
            height=2)
t.grid()
grid_row.append(t)
self.grid_cells.append(grid_row)

def update_grid_cells(self):
for i in range(c.GRID_LEN):
for j in range(c.GRID_LEN):
new_number = self.matrix[i][j]
if new_number == 0:
    self.grid_cells[i][j].configure(text="",bg=c.BACKGROUND_COLOR_CELL_EMPTY)
else:
    self.grid_cells[i][j].configure(
            text=str(new_number),
            bg=c.BACKGROUND_COLOR_DICT[new_number],
       fg=c.CELL_COLOR_DICT[new_number]
            )
self.update_idletasks()
```

```
        def key_down(self, event):
        key = event.keysym
        print(event)
        if key == c.KEY_QUIT: exit()
          if key == c.KEY_BACK and len(self.history_matrixs) > 1:
        self.matrix = self.history_matrixs.pop()
        self.update_grid_cells()
        print('back on step total step:', len(self.history_matrixs))
        elif key in self.commands:
        self.matrix, done = self.commands[key](self.matrix)
        if done:
        self.matrix = logic.add_two(self.matrix)
         self.history_matrixs.append(self.matrix)
        self.update_grid_cells()
         if logic.game_state(self.matrix) == 'win':
            self.grid_cells[1][1].configure(text="You", bg=c.BACKGROUND_COLOR_CELL_EMPTY)
            self.grid_cells[1][2].configure(text="Win!", bg=c.BACKGROUND_COLOR_CELL_EMPTY)
         if logic.game_state(self.matrix) == 'lose':
            self.grid_cells[1][1].configure(text="You", bg=c.BACKGROUND_COLOR_CELL_EMPTY)
            self.grid_cells[1][2].configure(text="Lose!", bg=c.BACKGROUND_COLOR_CELL_EMPTY)


        def generate_next(self):
        index = (gen(), gen())
        while self.matrix[index[0]][index[1]] != 0:
        index = (gen(), gen())
        self.matrix[index[0]][index[1]] = 2

game_grid = GameGrid()
```

# RESULTS :


2048.org
SCORE
3088
BEST
3088
Join numbers to get to the 2048 tile!
New Game
2
64
2
16
64
256
4
2
Game over!
4
64
16
4
Try again
64
8
4
2


2048
SCORE
20212
BEST
20212
Join the numbers and get to the 2048 tile!
New Game
2
4
You win!
8
16
2
Keep going
Try again
Tweet
79.8K
4
4
4
2048