



DELHI PUBLIC SCHOOL RUBY PARK KOLKATA



COMPUTER SCIENCE PROJECT SYNOPSIS

(By Adriteyo Das)

CERTIFICATE

This is to certify that Adriteyo Das a bona fide student of Class XI-L has successfully completed the project titled “2048 – An Interactive Entertainment System” in the Computer Lab during the Year 2021-2022 For the A.I.S.S.C.E Computer Science Practical Examination - 2021.

It is further certified that this project is the collective work of 2 students (Adriteyo Das and Shovan Kumar Sasmal).

External Examiner

Internal

DATED: _____

SEAL: _____

ACKNOWLEDGEMENT

I gratefully acknowledge my sincere thanks to our Computer Science Teacher, Miss Tamali Sinha for her remarkable, valuable guidance and supervision throughout the project work. I'm also utmost indebted to all my batch mates for their encouragement, help, suggestion and readily helpful service in the successful completion of the project.

I wish to express my deep gratitude and sincere thanks to the Principal, Mrs. Joyoti Chaudhuri, Delhi Public School, Ruby Park for her encouragement and for all the facilities that she provided for this project work.






-Adriteyo Das

INDEX

SERIAL NO.	CONTENTS	PAGE NUMBER
1	CERTIFICATE	2
2	ACKNOWLEDGEMENT	3
3	INDEX	4
4	OBJECTIVE	5
5	INPUT AND OUTPUT OF THE PROPOSED SYSTEM	6
6	FUNCTIONS OR FEATURES OF PROPOSED SYSTEM	7
7	HARDWARE & SOFTWARE TO BE USED	8
8	PROJECT DESIGN	9
9	MODULE DOCUMENTATION	10
10	SOURCE CODE	11
11	OUTPUT SCREENS	16
12	SCOPE AND LIMITATIONS OF THE PROJECT	20
13	ROLE OF STUDENTS	21
14	BIBLIOGRAPHY AND REFERENCES	22

OBJECTIVE

The objective of this project is to recreate the famous game of “2048” using Python Libraries. 2048 is a single-player sliding tile puzzle video game. As many other games that are seemingly endless, 2048 is challenging your brain to figure out solution to more and more complex situations. The game has brilliantly balanced mechanics thus offering challenge during every moment of the gameplay. There are a number of things that 2048 can do for you. For starters, it’s a game that certainly has brain training capabilities. That means that while you’re playing, you’re actually expanding your capacity and that’s a great thing. A quick list of what 2048 aims to do is:

-  Offers hours of fun so the user can never be bored.
-  Let the user decide how they want to pace it and because games are quick and easy to follow, there is no need to worry about losing game progress.
-  Allow the user to constantly challenge themselves to a better score.
-  Demand a balance amount of logical thinking from the user, such that the game does not become too easy, nor does it become so difficult that only skilled mathematicians can win
-  Teaches the user to estimate, think quickly and strategize by planning their next moves and considering how it will affect the board and which combination of movements will have the desired outcome.

INPUT AND OUTPUT OF THE PROPOSED SYSTEM

Input: -

Input means the raw data to be entered as input or information required for the system. To play the game, all that a user had to do was to swipe number tiles to sum up two 2s or 4s or 8s and so on, till they get to the number 2048. The user will play the game, solely through the use of their computer arrow keys.

Output:-

Output means the useful information generated by the system. With each swipe a new number pops up to add to the challenge and you have to keep clearing the board as the numbers keep growing. The tiles will be displayed on a 4 x 4, for the visual benefit of the user, tiles belonging to different categories of numbers, shall be coloured differently

FUNCTIONS OR FEATURES OF PROPOSED SYSTEM



2048 is played on a plain 4×4 grid, with numbered tiles that slide when a player moves them using the four arrow keys.



Every turn, a new tile randomly appears in an empty spot on the board with a value of either 2 or 4.



Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid.



If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided.



The resulting tile cannot merge with another tile again in the same move.



The highest possible tile is 131,072.



If a move causes three consecutive tiles of the same value to slide together, only the two tiles farthest along the direction of motion will combine.



If all four spaces in a row or column are filled with tiles of the same value, a move parallel to that row/column will combine the first two and last two.



A scoreboard keeps track of the user's score.



The user's score starts at zero, and is increased whenever two tiles combine, by the value of the new tile.



The game is won when a tile with a value of 2048 appears on the board.



Players can continue beyond that to reach higher scores.



When the player has no legal moves (there are no empty spaces and no adjacent tiles with the same value), the game ends.

HARDWARE & SOFTWARE TO BE USED

Hardware Used:-



CPU / Processor - AMD Ryzen 5 2600 Six-Core AM4 Processor Motherboard



Motherboard - MSI B450 Tomahawk AM4 ATX Motherboard



RAM / Memory - 16GB Ballistix Tactical Tracer RGB 3000MHz DDR4 (2 x 8GB)



Storage Drive- 1 x 500GB Samsung 860 EVO SATA III 2.5" SSD



GPU/Graphics Card - MSI Radeon RX 580 ARMOR MK2 8GB OC

Software Used:-



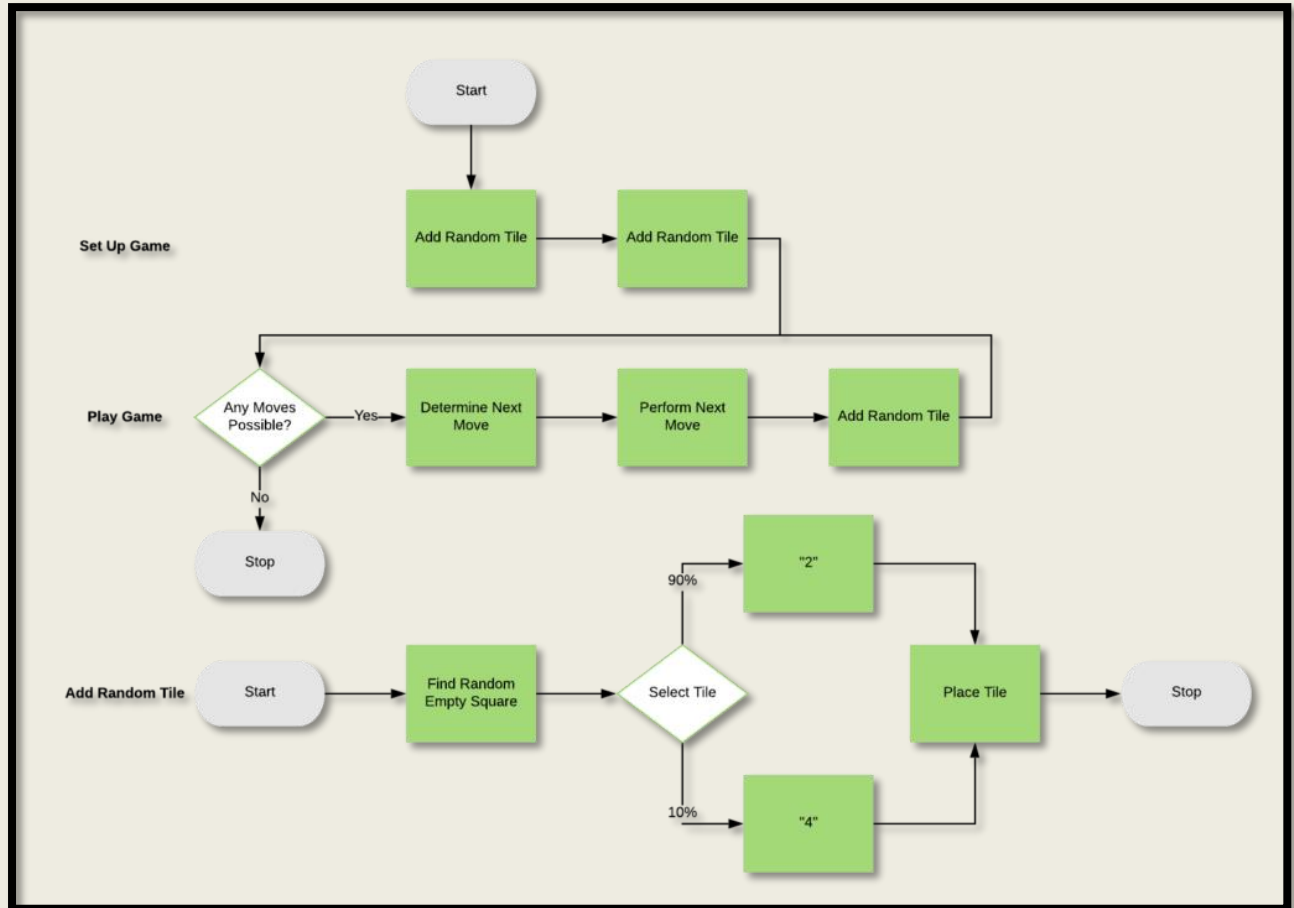
Spyder - Spyder is an open-source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrates with a number of prominent packages in the scientific Python stack, including NumPy, SciPy, Matplotlib, pandas, IPython, SymPy and Cython, as well as other open-source software.



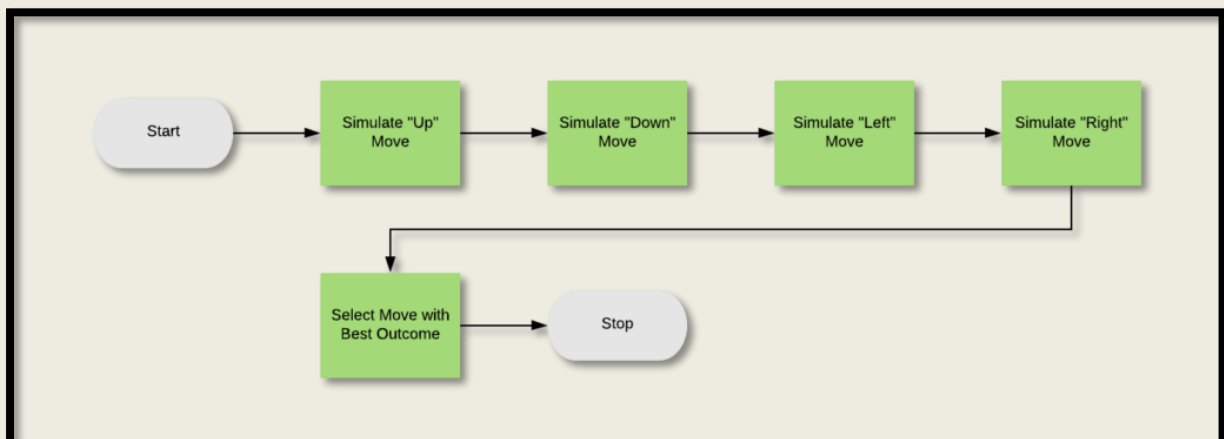
Tkinter - Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python.

PROJECT DESIGN

The general flow of how the gameplay works:



The random aspect of the game is noticed in the “Add Random Tile” process – both in the fact that a random square to add the tile to is searched, and a random value for the tile is selected. This is the algorithm to play the game. The general overflow of this as follows:



MODULE DOCUMENTATION

Tkinter:

Description - Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI.

Functions -

- i) Frame -> Python Tkinter Frame widget is used to organize the group of widgets. It acts like a container which can be used to hold the other widgets. The rectangular areas of the screen are used to organize the widgets to the python application.
- ii) Grid -> The grid() geometry manager organises widgets in a table-like structure in the parent widget. The master widget is split into rows and columns, and each part of the table can hold a widget. It uses column, columnspan, padx, pady, rowspan and sticky.

Random:

Description - Python Random module is an in-built module of Python which is used to generate random numbers. These are pseudo-random numbers means these are not truly random. This module can be used to perform random actions such as generating random numbers, print random a value for a list or string, etc.

Functions -

- i) randint(x, y) -> It returns a random integer in the range of [x, y] i.e. both x and y are included

SOURCE CODE

```

from tkinter import *
from random import *

SCORE = 0
SIZE = 500
GRID_LEN = 4
GRID_PADDING = int(40 / GRID_LEN)
TILES_PER_MOVE = 1

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: "#00ff00", \
                           8192: "#00dd00" }
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: "#f9f6f2", \
                    8192: "#f9f6f2" }
for i in range(14, 2048):
    BACKGROUND_COLOR_DICT[2 ** i] = "#00c000"
    CELL_COLOR_DICT[2 ** i] = "#f9f6f2"

FONT = ("Verdana", int(80 / GRID_LEN), "bold")

KEY_UP_ALT = "\'\\\uf700\'"
KEY_DOWN_ALT = "\'\\\uf701\'"
KEY_LEFT_ALT = "\'\\\uf702\'"
KEY_RIGHT_ALT = "\'\\\uf703\'"
KEY_UNDO_ALT = "\'\\\uf704\'"

KEY_UP = "w"
KEY_DOWN = "s"
KEY_LEFT = "a"
KEY_RIGHT = "d"
KEY_UNDO = "u"
KEY_RESET = "r"

def new_game(n):
    if n == 4:
        return [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
    matrix = []
    for i in range(n):
        matrix.append([0] * n)
    return matrix

def empty_spaces(mat):
    sum = 0
    for i in range(len(mat)):
        sum = sum + mat[i].count(0)
    return sum

def add_two(mat):

```

```

global TILES_PER_MOVE
temp = TILES_PER_MOVE
if empty_spaces(mat) < TILES_PER_MOVE:
    TILES_PER_MOVE = empty_spaces(mat)
for i in range(TILES_PER_MOVE):
    a = randint(0, len(mat) - 1)
    b = randint(0, len(mat) - 1)
    while mat[a][b] != 0:
        a = randint(0, len(mat) - 1)
        b = randint(0, len(mat) - 1)
    n = randint(0, 100)
    if n > 90:
        mat[a][b] = 4
    else:
        mat[a][b] = 2
TILES_PER_MOVE = temp
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)-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 i in range(len(mat)):
        for j in range(len(mat[0])):
            if mat[i][j] == 0:
                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 i in range(GRID_LEN):
        new1 = []
        for j in range(GRID_LEN):
            new1.append(0)
        new.append(new1)
    done = False
    for i in range(GRID_LEN):
        count = 0
        for j in range(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 = False
    global SCORE
    for i in range(GRID_LEN):
        for j in range(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
                SCORE = SCORE + mat[i][j]
    return (mat, done)

def reset(mat):
    global GRID_LEN
    mat = [[0] * GRID_LEN] * GRID_LEN
    done = True
    return mat, done

def up(game):
    game = transpose(game)
    game, done = cover_up(game)
    temp = merge(game)
    game = temp[0]
    done = done or temp[1]
    game = cover_up(game)[0]
    game = transpose(game)
    return (game, done)

def down(game):
    game = reverse(transpose(game))
    game, done = cover_up(game)
    temp = merge(game)
    game = temp[0]
    done = done or temp[1]
    game = cover_up(game)[0]
    game = transpose(reverse(game))
    return (game, done)

```

```

def left(game):
    game, done = cover_up(game)
    temp = merge(game)
    game = temp[0]
    done = done or temp[1]
    game = cover_up(game)[0]
    return (game, done)

def right(game):
    game = reverse(game)
    game, done = cover_up(game)
    temp = merge(game)
    game = temp[0]
    done = done or temp[1]
    game = cover_up(game)[0]
    game = reverse(game)
    return (game, done)

def undo(game):
    game, done = cover_up(game)
    return (game, done)

class GameGrid(Frame):

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

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

        self.commands = { KEY_UP: up, KEY_DOWN: down, KEY_LEFT: left, KEY_RIGHT: right,
KEY_UNDO: undo, KEY_RESET: reset,
                        KEY_UP_ALT: up, KEY_DOWN_ALT: down, KEY_LEFT_ALT: left,
KEY_RIGHT_ALT: right, KEY_UNDO_ALT: undo }

        self.grid_cells = []
        self.init_grid()
        self.init_matrix()
        self.update_grid_cells()

        self.mainloop()

    def init_grid(self):
        background = Frame(self, bg = BACKGROUND_COLOR_GAME, width = SIZE, height =
SIZE)
        background.grid()
        for i in range(GRID_LEN):
            grid_row = []
            for j in range(GRID_LEN):
                cell = Frame(background, bg = BACKGROUND_COLOR_CELL_EMPTY, width = SIZE
/ GRID_LEN, height = SIZE / GRID_LEN)
                cell.grid(row = i, column = j, padx = GRID_PADDING, pady = GRID_PADDING)
                t = Label(master=cell, text="", bg=BACKGROUND_COLOR_CELL_EMPTY,
justify=CENTER, font=FONT, width=8, height=4)
                t.grid()
                grid_row.append(t)

            self.grid_cells.append(grid_row)

```

```

def gen(self):
    return randint(0, GRID_LEN - 1)

def init_matrix(self):
    self.matrix = new_game(GRID_LEN)

    for i in range(2):
        self.matrix=add_two(self.matrix)

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

def key_down(self, event):
    key = repr(event.char)
    if key in self.commands:
        self.matrix,done = self.commands[repr(event.char)](self.matrix)
        if done:
            self.matrix = add_two(self.matrix)
            self.update_grid_cells()
            done = False
            if game_state(self.matrix) == 'win':
                self.grid_cells[1][1].configure(text = "You", bg =
BACKGROUND_COLOR_CELL_EMPTY)
                self.grid_cells[1][2].configure(text = "Win!", bg =
BACKGROUND_COLOR_CELL_EMPTY)
            if game_state(self.matrix) == 'lose':
                self.grid_cells[1][1].configure(text = "You", bg =
BACKGROUND_COLOR_CELL_EMPTY)
                self.grid_cells[1][2].configure(text = "Lose!", bg =
BACKGROUND_COLOR_CELL_EMPTY)

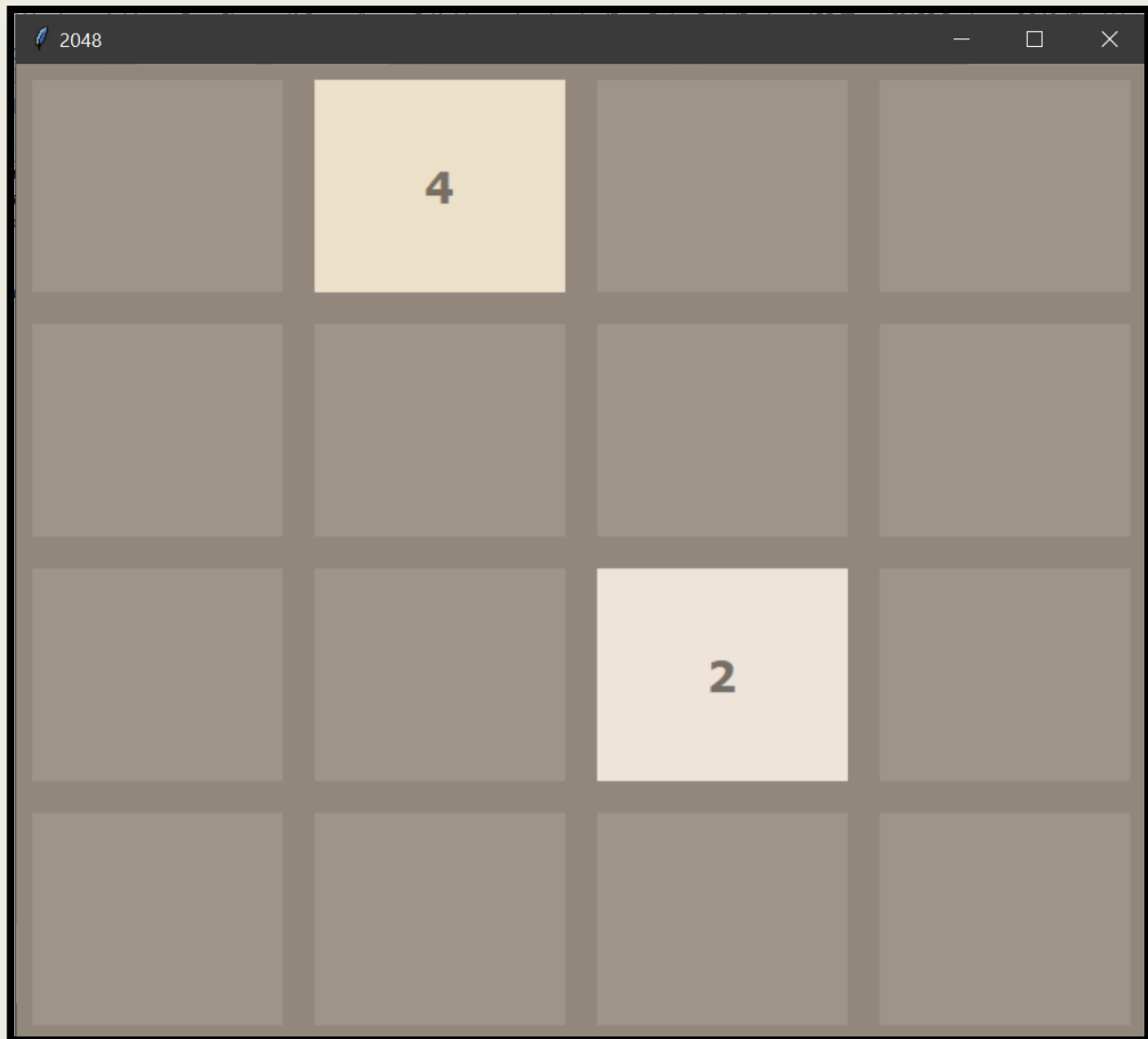
def generate_next(self):
    index = (self.gen(), self.gen())
    while self.matrix[index[0]][index[1]] != 0:
        index = (self.gen(), self.gen())
    self.matrix[index[0]][index[1]] = 2

gamegrid = GameGrid()

```

OUTPUT SCREENS

Note : These are only a few of the possible outputs. Due to the inclusion of the random () module in the program, there are an infinite possibilities and each move is not guaranteed to result in the same outcome every time.



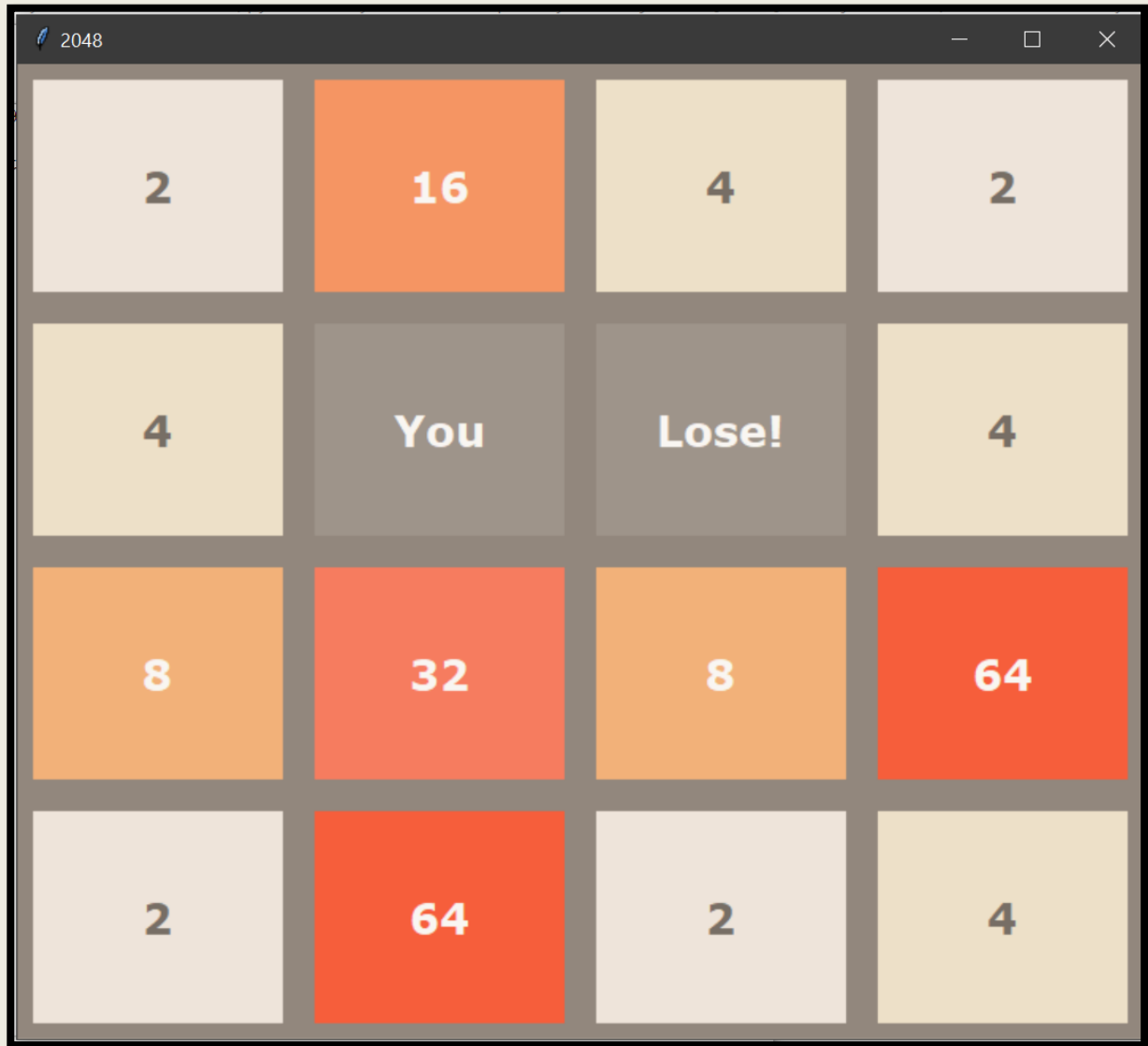
This is the starting screen of the game that is displayed immediately after the player first runs the program



This is the screen displayed after the player has made 3 successive moves (one up, one right and one down)







This is the screen displayed during the intermediate stages of the game. The grid values have gotten exponentially higher.






The screen that signifies the end of the game. The play has run out of space in the grid, no legal moves are left and a 2048 marked grid has failed to appear. The player has unfortunately, failed to win

SCOPE AND LIMITATIONS OF THE PROJECT

Scope:-

-  Possibility of adding improved 3-D visual tiles
-  Possibility of expanding the goal of the game to reach tiles beyond 2048, or even less than that, as a difficulty setting to be selected by the user
-  Possibility of adding an undo feature
-  Possibility of making the game even more challenging by allowing tiles to combine in series of geometric progressions instead of arithmetic ones

Limitations:-

-  Lacks a functional back-end, hence cannot store user's previous score data
-  The game may become too easy or even simple, once the user is able to understand the mathematics or logic behind the game
-  The current version, cannot be ported to a device besides a PC running Windows XP or higher.

ROLE OF STUDENTS

The project has been made possible by the combined works and efforts of two students: -



Adriteyo Das (Class 11-L, Roll Number – 1):

Programming of the game mechanics and non-visual aspects of the project.



Shovan Kumar Sasmal (Class 11-L, Roll Number – 28) : Designing the GUI and overall visual aspects of the projects

BIBLIOGRAPHY AND REFERENCES

Books:

- 1) Computer Science With Python Textbook For Class 11 Examination 2020-2021 by Sumita Arora
- 2) Python GUI Programming with Tkinter: Develop responsive and powerful GUI applications with Tkinter by Alan D. Moore

Websites:

- 1) <https://realpython.com/python-gui-tkinter/>
- 2) <https://towardsdatascience.com/making-simple-games-in-python-f35f3ae6f31a>
- 3) <https://flaviocopes.com/python-tkinter/>
- 4) [https://python-textbok.readthedocs.io/en/1.0/Introduction to GUI Programming.html](https://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html)