*NANOCLIP*

***DONE BY***

***DHANUSH S.R. (12-E)***





(Affiliated to Central Board of Secondary Education, New Delhi)

(Chettinad House, R.A.Puram, Chennai – 600 028)

COMPUTER SCIENCE

Certified to be the Bonafide Record of work done by

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of Std XII Sec \_\_\_\_

in the Computer Science Lab of the CHETTINAD VIDYASHRAM,

CHENNAI, during the year 2020 – 2021.

Date: Teacher-in-charge

REGISTER NO. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted for All India Senior Secondary Practical Examination in

Computer Science held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_at

Chettinad Vidyashram, Chennai – 600 028.

Principal Internal Examiner External Examiner

**ACKNOWLEDGEMENT**

I would like to express my sincere thanks to Meena Aunty, Principal Mrs. S.Amudhalakshmi

for their encouragement and support to work on

this project. I am grateful to my computer science

teacher Mrs. S.Vijayalakshmi and to the computer science department for the constant guidance and support to complete the project.

*INDEX*

|  |  |  |
| --- | --- | --- |
| **SNO** | **TOPIC** | **PAGE NO.** |
| 1 | Overview of Python and Python Objects | 5 |
| 2 | Project Description | 6 |
| 3 | Modules Used | 7 |
| 4 | User defined functions used | 9 |
| 5 | Source Code | 11 |
| 6 | Sample Output | 34 |
| 7 | Conclusion | 40 |
| 8 | Bibliography | 41 |

**OVERVIEW OF PYTHON**

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

* **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
* **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
* **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

**LOOPING STRUCTURE:**

A loop statement allows us to execute a statement or group of statements multiple times.

* **for loop:**

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

* **While loop:**

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

**PROJECT DESCRIPTION**

Our project NANOCLIP is game hub which has three games :

1. 2048
2. TIC TAC TOE
3. ROCK PAPER SCISSOR

The games designed are multiplayer games using single device input mode and puzzle based game.

You need a mouse to play rock paper scissor game

You need a keyboard as well as mouse to play 2048, tic tac toe.

There is redo move option in 2048 which allows the user to play the last done moves again.

The game is built using IDLE 3.9 and PyCharm.

We have used Tkinter and random inbuilt modules to create the game.This games works on all platforms: Windows, MAC, LINUX etc.

**Modules used to Create NANOCLIP**

**1)Tkinter Module:**

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

* Import the *Tkinter* module.
* Create the GUI application main window.
* Add one or more of the above-mentioned widgets to the GUI application.
* Enter the main event loop to take action against each event triggered by the user.

**2)Random Module:**

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

Almost all module functions depend on the basic function [random()](https://docs.python.org/3/library/random.html#random.random), which generates a random float uniformly in the semi-open range [0.0, 1.0). The functions supplied by this module are actually bound methods of a hidden instance of the [random.Random](https://docs.python.org/3/library/random.html" \l "random.Random" \o "random.Random) class. You can instantiate your own instances of [Random](https://docs.python.org/3/library/random.html#random.Random) to get generators that don’t share state.

Class [Random](https://docs.python.org/3/library/random.html#random.Random) can also be subclassed if you want to use a different basic generator of your own devising: in that case, override the random(), seed(), getstate(), and setstate() methods. Optionally, a new generator can supply a getrandbits() method — this allows [randrange()](https://docs.python.org/3/library/random.html" \l "random.randrange" \o "random.randrange) to produce selections over an arbitrarily large range.

**3)** **two\_zero\_four\_eight.constants:**

This is a user designed module for 2048(game). This module is made to construct 2048, the buttons colour of the button, background, text, text colour.

The controls for the game is designed in this module.

**4)** **two\_zero\_four\_eight.logic:**

In this module we have written a code (logic) for the game or to be more specific we have built the brain for 2048. the code tells the computer how to change the matrix after each input by the user . The games works accordingly to this code.

**5)** **two\_zero\_four\_eight.puzzle:**

In this module we have written a code(logic) for the game or to be more specific we have created rules of the games and program games works according to the rules .

**UDFs used to Create NANOCLIP**

**UDF used in ROCK PAPER SCISSOR**

1) def \_\_init\_\_() : to display the frame/window of the rock paper scissor

2) def Rock() :rock function with 3 comparison

3) def Scissor() :Scissor Function with 3 comparison

4) def Paper() :Paper Function with 3 comparison

**UDF used in TIC TAC TOE**

5) def\_\_init\_\_() : to display the frame/window of the tic tac toe

6) def disablebutton() :this function is used because the second player cannot

input on the button which has already taken input

7) def btnClick() :global bclick, flag, player2\_name, player1\_name,

Player b, player a.

8) def checkForWin() : condition for win

**UDF used in 2048**

9) def new\_game() : new game

8) def add\_two() : number two is placed randomly in the matrix

10) def game\_state() : checking for win after each input

11) def reverse() :inversing the matrix

12) def transpose() :flipping the matrix along its diagonal

13) def cover\_up() : erase the before matrix after user input

14)def merge() : adding two individual elements in the matrix

15) def up() :the matrix after up command by the user

16) def down() :the matrix after down command by the user

17) def left () :the matrix after left command by the user

18) def right() :the matrix after right command by the user

19) def \_\_init\_\_() :to display the frame/window of the 2048

20) def init\_grid() :to display the grid/box of 2048 game.

21) def gen() :generating new number after user input for next

input.

22) def init\_matrix() :changing number color after two cells are added.

23) def update\_grid\_cell() :changing the cell color after two cells are added.

24) def key\_down() :its designed to redo move if the user lost the game.

25) def generate\_next() :checking whether the matrix has changed after user

Input.

**UDF used in HOMEPAGE**

26) def \_\_init\_\_() :to display the frame/window of the nanoclip

27) def game\_box() :(button access to each game from homepage)

28) def play\_game() :used to play the game after the button is clicked

**SOURCES CODE**

**SOURCE CODE OF HOMEPAGE**

import two\_zero\_four\_eight.puzzle as g2048

import tic\_tac\_toe.tic\_tac\_toe as ttt

import rock\_paper\_scissor.rps as rps

from tkinter import \*

class Game(Frame):

def \_\_init\_\_(self):

Frame.\_\_init\_\_(self)

self.grid()

self.grid\_cells = []

self.master.title('Nanoclip')

self.game\_box()

self.mainloop()

def game\_box(self):

label = Label(master=self, text="SELECT A GAME TO PLAY ",

font=("Verdana", 24, "bold"), fg='red')

label.grid()

background = Frame(self, bg="#92877d",

width=100, height=400)

background.grid()

games = [["2048", "#ef6c00"], ["Tic-Tac-Toe", "blue"], ["Rock Paper

Scissors", "red"]]

grid\_row = []

for index, game in enumerate(games):

cell = Frame(background, bg=game[1],

width=100,

height=100)

cell.grid(row=1, column=index + 1, padx=10,

pady=10)

t = Label(master=cell, text=game[0],

bg=game[1], fg='white',

justify=CENTER, font=("Verdana", 20, "bold"), width=15,

height=4)

t.grid()

t.bind("<Button-1>", lambda e, ga=game[0]: self.play\_game(ga))

grid\_row.append(t)

self.grid\_cells.append(grid\_row)

def play\_game(self, game):

Frame.destroy(self)

print(game)

if game == "Tic-Tac-Toe":

ttt.TicTacToe()

elif game == "2048":

g2048.GameGrid()

else:

rps.RPS()

Game()

**SOURCE CODE OF 2048**

**SOURCE CODE OF CONSTANTS MODULE IN 2048**

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\_UP\_ALT = "\'\\uf700\'"

KEY\_DOWN\_ALT = "\'\\uf701\'"

KEY\_LEFT\_ALT = "\'\\uf702\'"

KEY\_RIGHT\_ALT = "\'\\uf703\'"

KEY\_UP = "'w'"

KEY\_DOWN = "'s'"

KEY\_LEFT = "'a'"

KEY\_RIGHT = "'d'"

KEY\_BACK = "'b'"

KEY\_J = "'j'"

**SOURCE CODE OF LOGIC MODULE IN 2048**

import random

import two\_zero\_four\_eight.constants as c

def new\_game(n):

matrix = []

for i in range(n):

matrix.append([0] \* n)

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)-1):

# intentionally reduced to check the row on the right and below

# more elegant to use exceptions but most likely this will be their solution

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)): # check for any zero entries

for j in range(len(mat[0])):

if mat[i][j] == 0:

return 'not over'

for k in range(len(mat)-1): # to check the left/right entries on the last row

if mat[len(mat)-1][k] == mat[len(mat)-1][k+1]:

return 'not over'

for j in range(len(mat)-1): # check up/down entries on last column

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 = False

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")

# return matrix after shifting up

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):

print("down")

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):

print("left")

# return matrix after shifting left

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):

print("right")

# return matrix after shifting right

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)

**SOURCE CODE OF PUZZEL MODULE IN 2048**

mport random

from tkinter import \*

import two\_zero\_four\_eight.logic as logic

import two\_zero\_four\_eight.constants as c

class GameGrid(Frame):

def \_\_init\_\_(self):

Frame.\_\_init\_\_(self)

self.grid()

self.master.title('2048')

self.master.bind("<Key>", self.key\_down)

# self.gamelogic = gamelogic

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\_ALT: logic.up, c.KEY\_DOWN\_ALT: logic.down,

c.KEY\_LEFT\_ALT: logic.left, c.KEY\_RIGHT\_ALT: logic.right,

c.KEY\_H: logic.left, c.KEY\_L: logic.right,

c.KEY\_K: logic.up, c.KEY\_J: logic.down}

self.grid\_cells = []

self.init\_grid()

self.init\_matrix()

self.update\_grid\_cells()

self.mainloop()

def init\_grid(self):

label=Label(text="CONTROLS: W(up),S(down),A(left),D(right),B(redo

move)",font=("Verdana",18),fg='Black')

label.grid()

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 gen(self):

return random.randint(0, c.GRID\_LEN - 1)

def init\_matrix(self):

self.matrix = logic.new\_game(c.GRID\_LEN)

self.history\_matrixs = list()

self.matrix = logic.add\_two(self.matrix)

self.matrix = logic.add\_two(self.matrix)

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 = repr(event.char)

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[repr(event.char)](self.matrix)

if done:

self.matrix = logic.add\_two(self.matrix)

# record last move

self.history\_matrixs.append(self.matrix)

self.update\_grid\_cells()

done = False

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 = (self.gen(), self.gen())

while self.matrix[index[0]][index[1]] != 0:

index = (self.gen(), self.gen())

self.matrix[index[0]][index[1]] = 2

**SOURCE CODE OF TIC TAC TOE**

from tkinter import \*

import tkinter.messagebox # because at the end to show the result eg(you win ,its a tie)

class TicTacToe(Frame): # class is used as blueprint to create object

def \_\_init\_\_(self):

Frame.\_\_init\_\_(self)

self.grid()

self.master.title("Tic Tac Toe")

self.pa = StringVar()

self.playerb = StringVar()

self.p1 = StringVar()

self.p2 = StringVar()

tk = Frame(self, bg="black",

width=100, height=500)

tk.grid()

self.player1\_name = Entry(tk, textvariable=self.p1, bd=5)

self.player1\_name.grid(row=1, column=1, columnspan=8)

self.player2\_name = Entry(tk, textvariable=self.p2, bd=5)

self.player2\_name.grid(row=2, column=1, columnspan=8)

self.bclick = True

self.flag = 0

self.buttons = StringVar()

label = Label(tk, text="Player 1:", font='Times 25 bold', bg='black', fg='white', height=1,

width=8) # players 1: name input box

label.grid(row=1, column=0)

label = Label(tk, text="Player 2:", font='Times 25 bold', bg='black', fg='white', height=1,

width=8) # player 2 : name input box

label.grid(row=2, column=0)

self.button1 = Button(tk, text=" ", font='Times 25 bold', bg='white', fg='black', height=4,

width=10, command=lambda: self.btnClick(self.button1))

self.button1.grid(row=3, column=0)

self.button2 = Button(tk, text=' ', font='Times 25 bold', bg='white', fg='black', height=4,

width=10, ommand=lambda: self.btnClick(self.button2))

self.button2.grid(row=3, column=1)

self.button3 = Button(tk, text=' ', font='Times 25 bold', bg='white', fg='black', height=4,

width=10, command=lambda: self.btnClick(self.button3))

self.button3.grid(row=3, column=2)

self.button4 = Button(tk, text=' ', font='Times 25 bold', bg='white', fg='black', height=4,

width=10, command=lambda: self.btnClick(self.button4))

self.button4.grid(row=4, column=0)

self.button5 = Button(tk, text=' ', font='Times 25 bold', bg='white', fg='black', height=4,

width=10,command=lambda: self.btnClick(self.button5))

self.button5.grid(row=4, column=1)

self.button6 = Button(tk, text=' ', font='Times 25 bold', bg='white', fg='black', height=4,

width=10, command=lambda: self.btnClick(self.button6))

self.button6.grid(row=4, column=2)

self.button7 = Button(tk, text=' ', font='Times 25 bold', bg='white', fg='black', height=4,

width=10, command=lambda: self.btnClick(self.button7))

self.button7.grid(row=5, column=0)

self.button8 = Button(tk, text=' ', font='Times 25 bold', bg='white', fg='black', height=4,

width=10, command=lambda: self.btnClick(self.button8))

self.button8.grid(row=5, column=1)

self.button9 = Button(tk, text=' ', font='Times 25 bold', bg='white', fg='black', height=4,

width=10, command=lambda: self.btnClick(self.button9))

self.button9.grid(row=5, column=2)

tk.mainloop()

def disableButton(self):

self.button1.configure(state=DISABLED)

self.button2.configure(state=DISABLED)

self.button3.configure(state=DISABLED)

self.button4.configure(state=DISABLED)

self.button5.configure(state=DISABLED)

self.button6.configure(state=DISABLED)

self.button7.configure(state=DISABLED)

self.button8.configure(state=DISABLED)

self.button9.configure(state=DISABLED)

def btnClick(self, buttons):

# global bclick, flag, player2\_name, player1\_name, playerb, pa

if buttons["text"] == " " and self.bclick == True:

buttons["text"] = "X"

self.bclick = False

self.playerb = self.player2\_name.get() + " Wins!"

self.playera = self.player1\_name.get() + " Wins!"

self.checkForWin()

self.flag += 1

elif buttons["text"] == " " and self.bclick == False:

buttons["text"] = "O"

self.bclick = True

self.checkForWin()

self.flag += 1

else:

tkinter.messagebox.showinfo("Tic-Tac-Toe", "Button already Clicked!")

def checkForWin(self): # conditions for win

if (self.button1['text'] == 'X' and self.button2['text'] == 'X' and self.button3['text'] == 'X'

or

self.button4['text'] == 'X' and self.button5['text'] == 'X' and self.button6['text'] ==

'X' or

self.button7['text'] == 'X' and self.button8['text'] == 'X' and self.button9['text'] ==

'X' or

self.button1['text'] == 'X' and self.button5['text'] == 'X' and self.button9['text'] ==

'X' or

self.button3['text'] == 'X' and self.button5['text'] == 'X' and self.button7['text'] ==

'X' or

self.button3['text'] == 'X' and self.button6['text'] == 'X' and self.button9['text'] ==

'X' or

self.button1['text'] == 'X' and self.button4['text'] == 'X' and self.button7['text'] ==

'X' or

self.button2['text'] == 'X' and self.button5['text'] == 'X' and self.button8['text'] ==

'X' or

self.button7['text'] == 'X' and self.button6['text'] == 'X' and self.button9['text'] == 'X'):

self.disableButton()

tkinter.messagebox.showinfo("Tic-Tac-Toe", self.playera)

elif ((self.button1['text'] == 'O' and self.button2['text'] == 'O' and self.button3['text'] ==

'O') or

(self.button4['text'] == 'O' and self.button5['text'] == 'O' and self.button6['text'] ==

'O') or

(self.button7['text'] == 'O' and self.button8['text'] == 'O' and self.button9['text'] ==

'O') or

(self.button1['text'] == 'O' and self.button5['text'] == 'O' and self.button9['text'] ==

'O') or

(self.button3['text'] == 'O' and self.button5['text'] == 'O' and self.button7['text'] ==

'O') or

(self.button3['text'] == 'O' and self.button6['text'] == 'O' and self.button9['text'] ==

'O') or

(self.button1['text'] == 'O' and self.button4['text'] == 'O' and self.button7['text'] ==

'O') or

(self.button2['text'] == 'O' and self.button5['text'] == 'O' and self.button8['text'] ==

'O') or

(self.button7['text'] == 'O' and self.button6['text'] == 'O' and self.button9['text'] ==

'O')):

self.disableButton()

tkinter.messagebox.showinfo("Tic-Tac-Toe", self.playerb)

elif self.flag == 8:

tkinter.messagebox.showinfo("Tic-Tac-Toe", "It is a Tie")

**SOURCE CODE OF ROCK PAPER SCISSOR**

from tkinter import \*

import random

class RPS(Frame):

def \_\_init\_\_(self):

# Screen Layout

Frame.\_\_init\_\_(self)

self.grid()

self.master.title("Rock | Paper | Scissor")

global userScore, compScore

userScore = 0

compScore = 0

background = Frame(self, bg="white",

width=100, height=400)

background.grid()

# Declaring Buttons

B1 = Frame(background, bg='black', height="10", width="20")

B1.grid(row=1, column=1, padx = 10)

t1 = Label(master=B1, text="ROCK",

bg="black", fg='white',

justify=CENTER, font=("Verdana", 20, "bold"), width=15, height=4)

t1.grid()

t1.bind("<Button-1>", self.Rock)

B2 = Frame(background, bg='black', height="10", width="20")

B2.grid(row=1, column=2, padx = 10)

t2 = Label(master=B2, text="PAPER",

bg="black", fg='white',

justify=CENTER, font=("Verdana", 20, "bold"), width=15, height=4)

t2.grid()

t2.bind("<Button-1>", self.Paper)

B3 = Frame(background, bg='black', height="10", width="20")

B3.grid(row=1, column=3, padx = 10)

t3 = Label(master=B3, text="SCISSOR",

bg="black", fg='white',

justify=CENTER, font=("Verdana", 20, "bold"), width=15, height=4)

t3.grid()

t3.bind("<Button-1>", self.Scissor)

self.text\_area = Label(master=self, text="Start your Game by Choosing any of the

choices", font=("Verdana", 12), fg='red')

# text\_area.pack()

self.text\_area.grid(pady = 50)

self.mainloop()

def Rock(self, rock):

""" Rock Function with 3 comparision"""

global userScore, compScore

comp = random.randint(1, 3)

if comp == 3:

comp = "Scissor"

userScore += 1

self.text\_area['text'] = ("Congratulations!", "You WIN!! \n""Your Choice:Rock" +

"\nComp Choice: " \+ comp + "\nYour Score: " + str(userScore)

+ "\nComputer Score: " +str(compScore))

elif comp == 1:

comp = "Rock"

self.text\_area['text'] = ("Same pinch!!",

"IT'S A TIE!! !!\n" + "Your Choice:Rock" + "\nComp Choice: " +

comp + "\nYour Score: "+ str(userScore) + "\nComputer Score: "

+ str(compScore))

else:

comp = "Paper"

compScore += 1

self.text\_area['text'] = ("Hard Luck!!",

"YOU LOOSE!!\n" + "Your Choice:Rock" + "\nComp Choice: " +

comp + "\nYour Score: "+ str(userScore) + "\nComputer Score: " +

def Scissor(self, scissor):

""" Scissor Function with 3 comparision"""

global userScore, compScore

comp = random.randint(1, 3)

if comp == 2:

comp = "Paper"

userScore += 1

self.text\_area['text'] = ("Congratulation!!",

"YOU WIN!!\nYour Choice: Scissor\n" + "Comp choice: " + comp

+ "\nYour Score: " + str( userScore) + "\nComputer Score: " +

str(compScore))

elif comp == 3:

comp = "Scissor"

self.text\_area['text'] = ("Same pinch!!",

"IT'S A TIE!!\nYour Choice: Scissor\n" + "Comp choice: " + comp +

"\nYour Score: " + str(userScore) + "\nComputer Score: " +

str(compScore))

else:

comp = "Rock"

compScore += 1

self.text\_area['text'] = ("Hard Luck!!",

"YOU LOOSE!!\nYour Choice: Scissor\n" + "Comp choice:" + comp

+ "\nYour Score: " + str(userScore) +"\nComputer Score: " +

str(compScore))

def Paper(self, paper):

""" Paper Function with 3 comparision"""

global userScore, compScore

comp = random.randint(1, 3)

if comp == 1:

comp = "Rock"

userScore += 1

self.text\_area['text'] = ("Congratulation!!",

"YOU WIN!!\nYour Choice: Paper\n" + "Comp choice: " + comp +

"\nYour Score: " + str(userScore) + "\nComputer Score: " +

str(compScore))

elif comp == 2:

comp = "Paper"

self.text\_area['text'] = ("Same pinch!!",

"IT'S A TIE!!\nYour Choice: Paper\n" + "Comp choice: " + comp +

"\nYour Score: " + str(userScore) + "\nComputer Score: " +

str(compScore))

else:

comp = "Scissor"

compScore += 1

self.text\_area['text'] = ("Hard Luck!!",

"YOU LOOSE!!\nYour Choice: Paper\n" + "Comp choice:" + comp

+ "\nYour Score: " + str(userScore) + "\nComputer Score: " +

str(compScore))

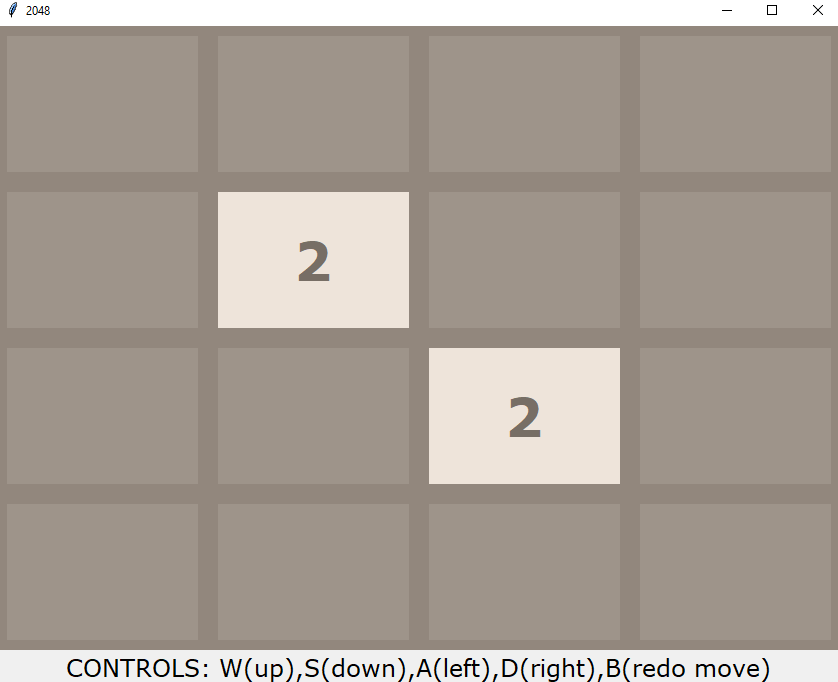
**SAMPLE OUTPUTS**

**HOMEPAGE OUTPUT**



HOMEPAGE OF NANOCLIP

**2048 OUTPUTs**

****

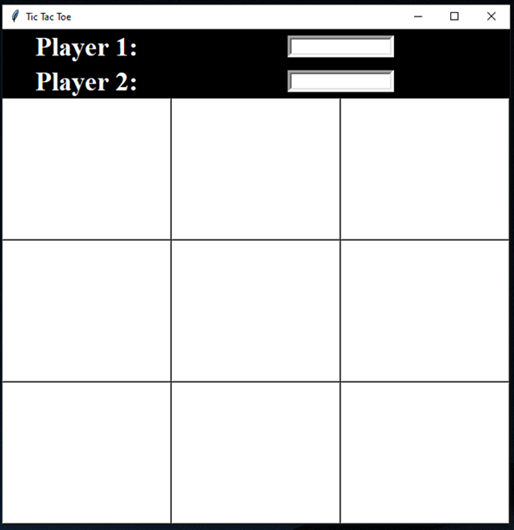
AT THE STARTING



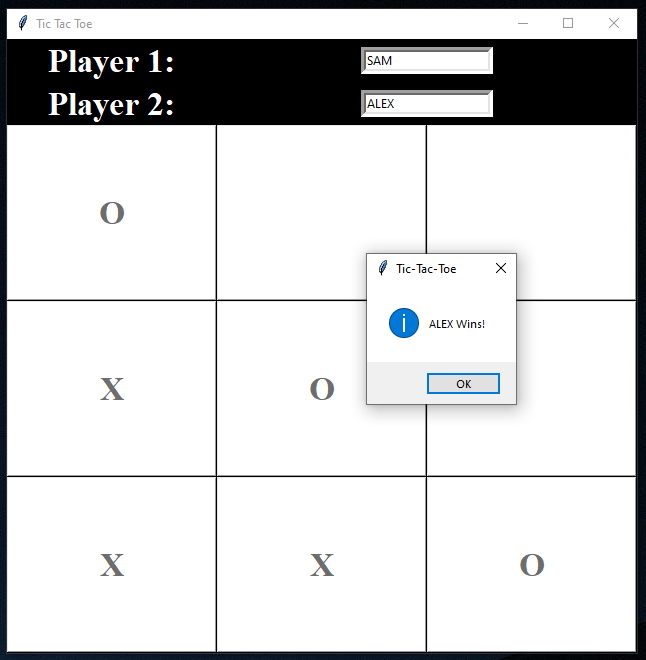
LOSE OUTPUT



WIN OUTPUT IN 2048

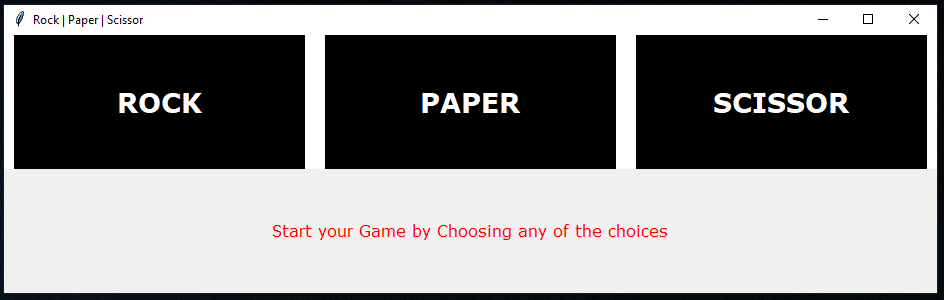
**TIC TAC TOE OUTPUTs**

AT THE START

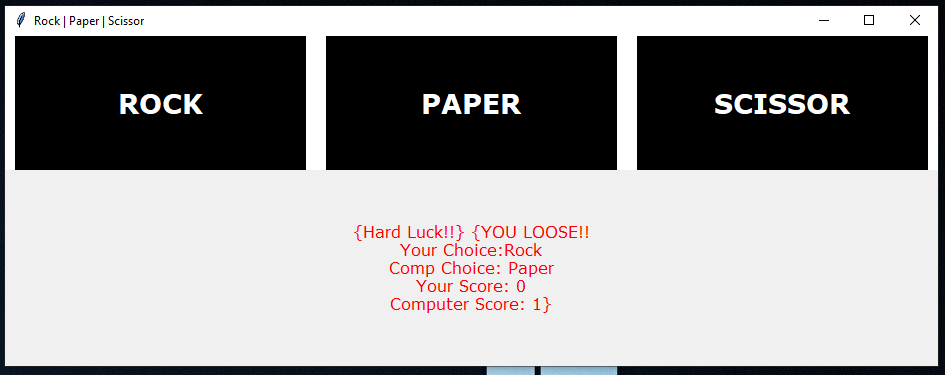


AT THE END

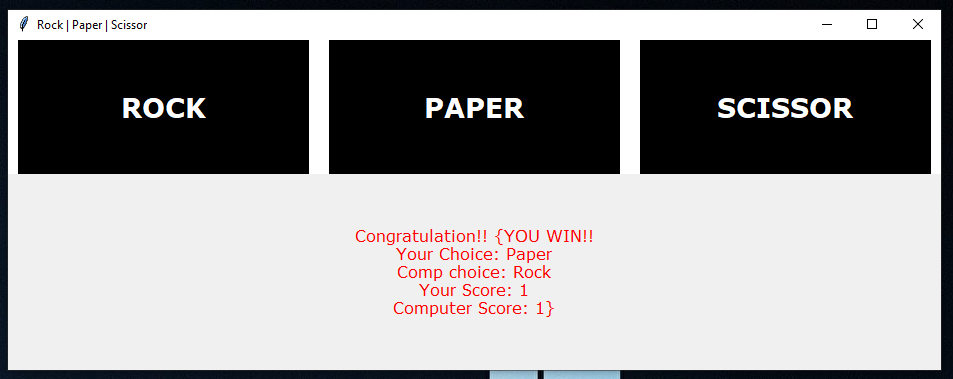
**ROCK PAPER SCISSOR OUTPUTS**



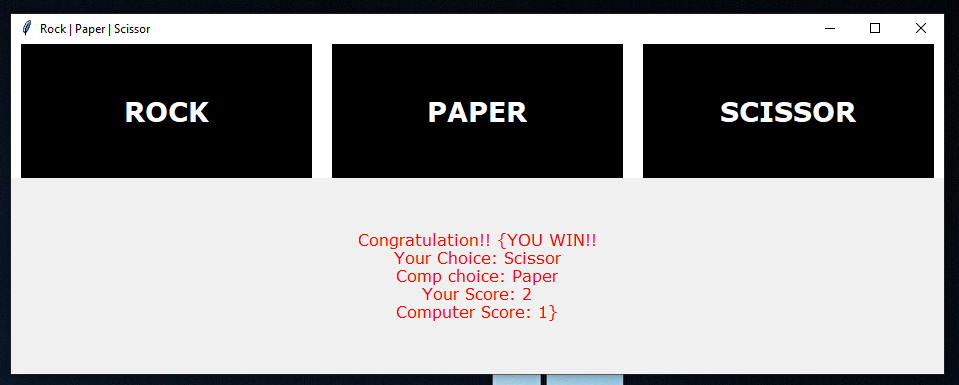
AT THE START



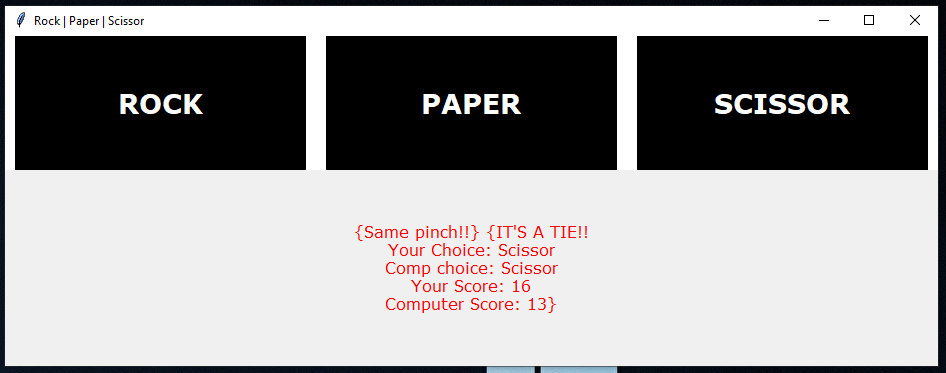
CHOICE=ROCK



CHOICE=PAPER



CHOICE=SCISSOR



AFTER SOMETIME

**CONCLUSION**

* Our project : “NANOCLIP” has three games :

1. 2048
2. Tic Tac Toe
3. Rock Paper Scissor

* All the games that we have created is user friendly, 3+ games.
* We have used python IDLE 3.9 and PyCharm to design the game.
* Tic Tac Toe and Rock Paper Scissor are multiplayer games with single device input mode.
* The odd in favour towards and against the user are equal in 2048.

**BIBLIOGRAPHY**

* <https://www.tutorialspoint.com>
* <https://www.w3schools.com>
* <https://www.quora.com>
* Computer Science with Python [Class 12] by Sumita Arora.