

A PROJECT REPORT
ON
Gaming Zone Using Python

Submitted in partial fulfillment for award of the degree of

BACHELOR OF TECHNOLOGY



Under the guidance of
Mr. Kazi Md. Arfin (Assistant Professor)

Submitted By:
Debarathi Sardar (CSE-20/18)
Bittu Dey (CSE-03/18)
Sujit Maity (CSE-05/18)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
BIRBHUM INSTITUTE OF ENGINEERING & TECHNOLOGY, SURI, BIRBHUM



BIRBHUM INSTITUTE OF ENGINEERING & TECHNOLOGY, SURI, BIRBHUM

CERTIFICATE

Certified that the project work entitled Gaming Zone Using Python carried out by Mr. Debarathi Sardar a student of Birbhum Institute of Engineering & Technology, Suri, Birbhum in partial fulfillment for the award of Bachelor of Technology in Computer Science & Engineering of the Maulana Abul Kalam Azad University of Technology, West Bengal during the year 2018-22. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Dr. Debasri Chakraborty
Head Of the Department

Mr. Kazi Md. Arfin
Project Supervisor



BIRBHUM INSTITUTE OF ENGINEERING & TECHNOLOGY, SURI, BIRBHUM

CERTIFICATE

Certified that the project work entitled Gaming Zone Using Python carried out by Mr. Bittu Dey a student of Birbhum Institute of Engineering & Technology, Suri, Birbhum in partial fulfillment for the award of Bachelor of Technology in Computer Science & Engineering of the Maulana Abul Kalam Azad University of Technology, West Bengal during the year 2018-22. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Dr. Debasri Chakraborty
Head Of the Department

Mr. Kazi Md. Arfin
Project Supervisor



BIRBHUM INSTITUTE OF ENGINEERING & TECHNOLOGY, SURI, BIRBHUM

CERTIFICATE

Certified that the project work entitled Gaming Zone Using Python carried out by Mr. Sujit Maity a student of Birbhum Institute of Engineering & Technology, Suri, Birbhum in partial fulfillment for the award of Bachelor of Technology in Computer Science & Engineering of the Maulana Abul Kalam Azad University of Technology, West Bengal during the year 2018-22. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Degree.

Dr. Debasri Chakraborty
Head Of the Department

Mr. Kazi Md. Arfin
Project Supervisor

ACKNOWLEDGEMENT

Our first experience of the project on Gaming Zone Using Python is successfully going on. Thanks for the supports of our friends and respected teachers with gratitude. We wish to acknowledge all of them.

We are especially thankful of our project guide **Mr. Kazi Md. Arfin** under whose guideline we were able to starting our project. We are whole heartedly thankful to him and **Dr. Debasri Chakraborty** (HOD, CSE Department) for giving us her valuable time and attention; and for providing us a systematic way for the continuation of our project.

We are also thankful to our parents who have inspired us to face all the challenges and win all the hurdles in life.

Thank you All.

TABLE OF CONTENTS

<u>TOPIC</u>	<u>PAGE</u>
ABSTRACT -----	1 - 2
INTRODUCTION -----	3
PYTHON GUI GAMING ZONE MENU -----	
GAMES	
GAME: 1 – HUNGRY SNAKE GAME -----	4 - 12
GAME: 2 – FLAPPY BIRD -----	13 - 24
GAME: 3 – PAC MAN -----	25 - 33
GAME: 4 – SLIDING PUZZLE -----	34 - 38
GAME: 5 – TIC-TAC-TOE -----	39 - 44
GAME: 6 – PONG GAME -----	45 - 56
GAME: 7 – ROCK PAPER SCISSOR -----	57 - 64
GAME: 8 – BUBBLE SORT VISUAL -----	65 - 70
HARDWARE DEVICES -----	71 - 72
SOFTWARE -----	73 - 74
MODULE -----	75
FINAL OUTPUT WITH CODE -----	76 - 93
REFERENCES -----	94

ABSTRACT:

A graphics-based operating system interface that uses icons, menus and a mouse to manage interaction with the system. Developed by Xerox, the GUI was popularized by the Apple Macintosh in the 1980s. At the time, Microsoft's operating system, MS-DOS, required the user to type specific commands, but the company's GUI, Microsoft Windows, is now the dominant user interface for personal computers (PCs). A comprehensive GUI environment includes four components: a graphics library, a user interface toolkit, a user interface style guide and consistent applications. The graphics library provides a high-level graphics programming interface. The user interface toolkit, built on top of the graphics library, provides application programs with mechanisms for creating and managing the dialogue elements of the windows, icons, menus, pointers and scroll bars (WIMPS) interface. The user interface style guide specifies how applications should employ the dialogue elements to present a consistent, easy-to-use environment to the user. Application program conformance with a single user interface style is the primary determinant of ease of learning and use, and thus, of application effectiveness and user productivity.

Designing the visual composition and temporal behavior of a GUI is an important part of software application programming in the area of human-computer interaction. Its goal is to enhance the efficiency and ease of use for the underlying logical design of a stored program, a design discipline named usability. The visible graphical interface features of an application are sometimes referred to as chrome or GUI (pronounced gooey).

Typically, users interact with information by manipulating visual widgets that allow for interactions appropriate to the kind of data they hold. The widgets of a well-designed interface are selected to support the actions necessary to achieve the goals of users. A model-view-controller allows flexible structures in which the interface is independent of and indirectly linked to application functions, so the GUI can be customized easily. This allows users to select or design a

different skin at will, and eases the designer's work to change the interface as user needs evolve. Good user interface design relates to users more and to system architecture less. Large widgets, such as windows, usually provide a frame or container for the main presentation content such as a web page, email message, or drawing. Smaller ones usually act as a user-input tool.

A GUI may be designed for the requirements of a vertical market as application-specific graphical user interfaces. Cell phones and handheld game systems also employ application specific touch screen GUIs. Newer automobiles use GUIs in their navigation systems and multimedia centers, or navigation multimedia center combinations.

Python has a lot of GUI frameworks, but Tkinter is the only framework that's built into the Python standard library. Tkinter has several strengths. It's cross-platform, so the same code works on Windows, macOS, and Linux. Visual elements are rendered using native operating system elements, so applications built with Tkinter look like they belong on the platform where they're run. Although Tkinter is considered the de facto Python GUI framework, it's not without criticism. One notable criticism is that GUIs built with Tkinter look outdated. If you want a shiny, modern interface, then Tkinter may not be what you're looking for. However, Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern sheen is unnecessary, and the top priority is to quickly build something that's functional and cross-platform.

INTRODUCTION:

Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk. Tkinter is not the only GuiProgramming toolkit for Python.

Tkinter is the Python interface to Tk, which is the GUI toolkit for Tcl/Tk. Tcl (pronounced as tickle) is a scripting language often used in testing, prototyping, and GUI development. Tk is an open-source, cross-platform widget toolkit used by many different programming languages to build GUI programs.

It is however the most commonly used one. Tkinter is the inbuilt python module that is used to create GUI applications. It is one of the most commonly used modules for creating GUI applications in Python as it is simple and easy to work with. You don't need to worry about the installation of the Tkinter module separately as it comes with Python already. It gives an object-oriented interface to the Tk GUI toolkit.

Some other Python Libraries available for creating our own GUI applications are

- Kivy
- Python Qt
- wxPython

Among all Tkinter is most widely used in this Project.

Python GUI GAMING ZONE Menu

Gaming Zone is a closed space that fills you with excitement and joy as soon as you enter it. Hence, it is essential to make the interior of the room a bit edgy to keep the vibe intact. Right home décor ideas can elevate the idea of a game zone into a practical space that invites attention.

A menu is a set of options presented to the user of a computer application to help the user find information or execute a program function. Menus are common in graphical user interfaces (GUI s) such as Windows or the Mac OS. Menus are also employed in some speech recognition programs.

1 .Hungry Snake Game

The concept of Snake originated from the 1976 arcade game Blockade, developed by a British company called Gremlin Interactive, which shut down in 1984. Blockade was designed as a two-player game in which each would guide their own snakes, leaving a solid line behind them.

Snake is a video game genre where the player maneuvers a growing line that becomes a primary obstacle to itself. The concept originated in the 1976 two-player arcade game Blockade from Gremlin Industries, and the ease of implementation has led to hundreds of versions for many platforms.

In the game of Snake, the player uses the arrow keys to move a "snake" around the board. As the snake finds food, it eats the food, and thereby grows larger. The game ends when the snake either moves off the screen or moves into itself. The goal is to make the snake as large as possible before that happens.

The game continues until the snake "dies".

A snake dies by either,

(1) Running into the edge of the board,

Or

(2) By running into its own tail. The final score is based on the number of apples eaten by the snake.

Code of Hungry Snake Game

```
# import required modules
import turtle
import time
import random

delay = 0.1
score = 0
high_score = 0

# Creating a window screen
wn = turtle.Screen()
wn.title("Snake Game")
wn.bgcolor("green")

# the width and height can be put as user's choice
```

```
wn.setup(width=600, height=600)

wn.tracer(0)


# head of the snake
head = turtle.Turtle()
head.shape("square")
head.color("white")
head.penup()
head.goto(0, 0)
head.direction = "Stop"


# food in the game
food = turtle.Turtle()
colors = random.choice(['red'])
shapes = random.choice(['circle'])
food.speed(0)
food.shape(shapes)
food.color(colors)
food.penup()
food.goto(0, 100)


pen = turtle.Turtle()
pen.speed(0)
pen.shape("square")
```

```
pen.color("white")
pen.penup()
pen.hideturtle()
pen.goto(0, 250)
pen.write("Score : 0 High Score : 0", align="center",
font=("candara", 24, "bold"))

# assigning key directions
def group():
    if head.direction != "down":
        head.direction = "up"

def godown():
    if head.direction != "up":
        head.direction = "down"

def goleft():
    if head.direction != "right":
        head.direction = "left"

def goright():
    if head.direction != "left":
        head.direction = "right"
```

```
def move():
    if head.direction == "up":
        y = head.ycor()
        head.sety(y+20)
    if head.direction == "down":
        y = head.ycor()
        head.sety(y-20)
    if head.direction == "left":
        x = head.xcor()
        head.setx(x-20)
    if head.direction == "right":
        x = head.xcor()
        head.setx(x+20)

    wn.listen()
    wn.onkeypress(group, "w")
    wn.onkeypress(godown, "s")
    wn.onkeypress(goleft, "a")
    wn.onkeypress(goright, "d")

    segments = []

# Main Gameplay
```

```
while True:
    wn.update()

    if head.xcor() > 290 or head.xcor() < -290 or head.ycor() > 290 or
    head.ycor() < -290:
        time.sleep(1)
        head.goto(0, 0)
        head.direction = "Stop"
        colors = random.choice(['red', 'blue', 'green'])
        shapes = random.choice(['square', 'circle'])
        for segment in segments:
            segment.goto(1000, 1000)
        segments.clear()
        score = 0
        delay = 0.1
        pen.clear()
        pen.write("Score : {} High Score : {}".format(
            score, high_score), align="center", font=("candara", 24, "bold"))
        if head.distance(food) < 20:
            x = random.randint(-270, 270)
            y = random.randint(-270, 270)
            food.goto(x, y)

            # Adding segment
            new_segment = turtle.Turtle()
```

```
new_segment.speed(0)
new_segment.shape("square")
new_segment.color("orange") # tail colour
new_segment.penup()
segments.append(new_segment)
delay -= 0.001
score += 10
if score > high_score:
    high_score = score
pen.clear()
pen.write("Score : {} High Score : {}".format(
    score, high_score), align="center", font=("candara", 24, "bold"))
# Checking for head collisions with body segments
for index in range(len(segments)-1, 0, -1):
    x = segments[index-1].xcor()
    y = segments[index-1].ycor()
    segments[index].goto(x, y)
if len(segments) > 0:
    x = head.xcor()
    y = head.ycor()
    segments[0].goto(x, y)
move()
for segment in segments:
    if segment.distance(head) < 20:
```

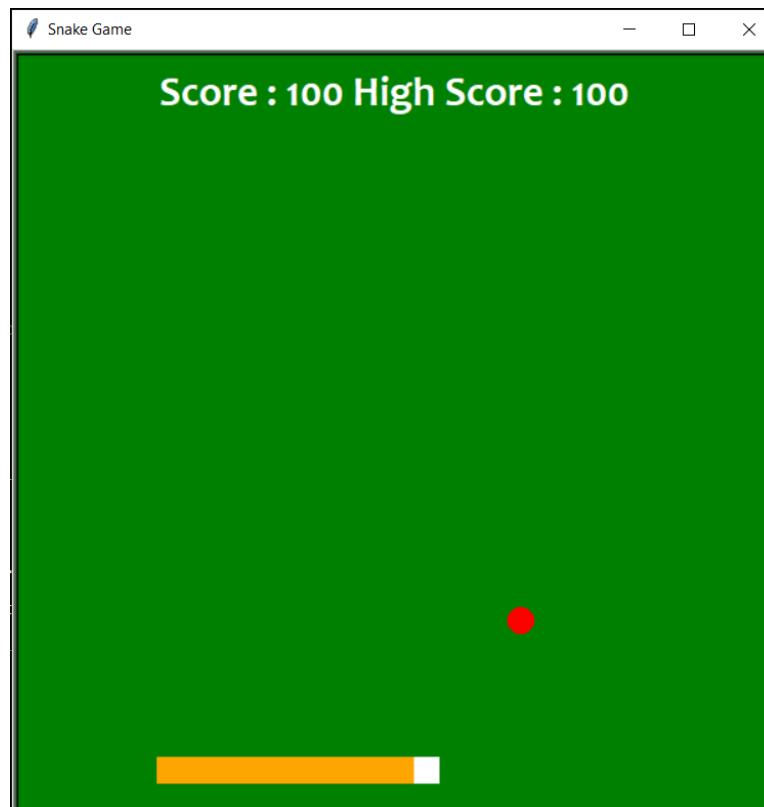
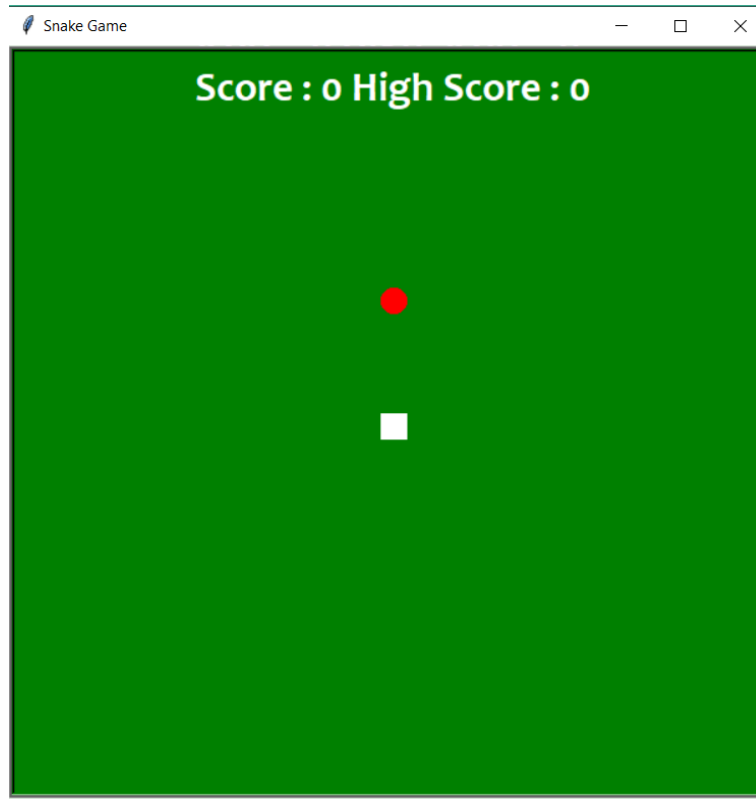


```
time.sleep(1)
head.goto(0, 0)
head.direction = "stop"
colors = random.choice(['red', 'blue', 'green'])
shapes = random.choice(['square', 'circle'])
for segment in segments:
    segment.goto(1000, 1000)
segment.clear()

score = 0
delay = 0.1
pen.clear()
pen.write("Score : {} High Score : {}".format(
score, high_score), align="center", font=("candara", 24, "bold"))
time.sleep(delay)

wn.mainloop()
```

Output of Hungry Snake Game



2 .Flappy Bird Game

Flappy Bird is a mobile game developed by Vietnamese video game artist and programmer Dong Nguyen, under his game development company .Gears. The game is a side-scroller where the player controls a bird, attempting to fly between columns of green pipes without hitting them.

The term/genre is known as an Endless Runner or Endless Flyer. Endless games are defined by just that... They keep going indefinitely until you inevitably lose, either by ramping up difficulty the longer you go like Jetpack Joyride or just being difficult from the start like Flappy Bird and Swing Copters.

Flappy Bird High Score in History over One Million Points 9,999,999 (World Record) No cheats.

Code of Flappy Bird Game

```
import random # For generating random numbers

import sys # We will use sys.exit to exit the program

import pygame

from pygame.locals import * # Basic pygame imports


# Global Variables for the game

FPS = 32

SCREENWIDTH = 289

SCREENHEIGHT = 511
```

```

SCREEN = pygame.display.set_mode((SCREENWIDTH,
SCREENHEIGHT))

GROUNDY = SCREENHEIGHT * 0.8

GAME_SPRITES = {}

GAME_SOUNDS = {}

PLAYER = 'C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\sprites\\bird.png'

BACKGROUND = 'C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\sprites\\background.png'

PIPE = 'C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\sprites\\pipe.png'


def welcomeScreen():
    """
    Shows welcome images on the screen
    """

    playerx = int(SCREENWIDTH/5)
    playery = int((SCREENHEIGHT -
GAME_SPRITES['player'].get_height())/2)
    messagex = int((SCREENWIDTH -
GAME_SPRITES['message'].get_width())/2)
    messagey = int(SCREENHEIGHT*0.13)
    basex = 0
    while True:
        for event in pygame.event.get():

```

```

        # if user clicks on cross button, close the game

        if event.type == QUIT or (event.type==KEYDOWN and
event.key == K_ESCAPE):

            pygame.quit()

            sys.exit()


        # If the user presses space or up key, start the game for them

        elif event.type==KEYDOWN and (event.key==K_SPACE or
event.key == K_UP):

            return

        else:

            SCREEN.blit(GAME_SPRITES['background'], (0, 0))

            SCREEN.blit(GAME_SPRITES['player'], (playerx, playery))

            SCREEN.blit(GAME_SPRITES['message'],
(messagex,messagey ))

            SCREEN.blit(GAME_SPRITES['base'], (basex, GROUNDY))

            pygame.display.update()

            FPSCLOCK.tick(FPS)


def mainGame():

    score = 0

    playerx = int(SCREENWIDTH/5)

    playery = int(SCREENWIDTH/2)

    basex = 0

```

```

# Create 2 pipes for blitting on the screen

newPipe1 = getRandomPipe()
newPipe2 = getRandomPipe()


# my List of upper pipes
upperPipes = [
    {'x': SCREENWIDTH+200, 'y':newPipe1[0]['y']},
    {'x': SCREENWIDTH+200+(SCREENWIDTH/2),
'y':newPipe2[0]['y']},
]

# my List of lower pipes
lowerPipes = [
    {'x': SCREENWIDTH+200, 'y':newPipe1[1]['y']},
    {'x': SCREENWIDTH+200+(SCREENWIDTH/2),
'y':newPipe2[1]['y']},
]


pipeVelX = -4


playerVelY = -9
playerMaxVelY = 10
playerMinVelY = -8
playerAccY = 1


playerFlapAccv = -8 # velocity while flapping

```

```

playerFlapped = False # It is true only when the bird is flapping

while True:

    for event in pygame.event.get():

        if event.type == QUIT or (event.type == KEYDOWN and
event.key == K_ESCAPE):

            pygame.quit()

            sys.exit()

        if event.type == KEYDOWN and (event.key == K_SPACE or
event.key == K_UP):

            if playery > 0:

                playerVelY = playerFlapAccv

                playerFlapped = True

                GAME_SOUNDS['wing'].play()


        crashTest = isCollide(playerx, playery, upperPipes, lowerPipes) #
This function will return true if the player is crashed

        if crashTest:

            return


    #check for score

    playerMidPos = playerx + GAME_SPRITES['player'].get_width()/2

    for pipe in upperPipes:

```

```

        pipeMidPos = pipe['x'] +
GAME_SPRITES['pipe'][0].get_width()/2

        if pipeMidPos<= playerMidPos < pipeMidPos +4:
            score +=1
            print(f"Your score is {score}")
            GAME_SOUNDS['point'].play()

        if playerVelY <playerMaxVelY and not playerFlapped:
            playerVelY += playerAccY

        if playerFlapped:
            playerFlapped = False
            playerHeight = GAME_SPRITES['player'].get_height()
            playery = playery + min(playerVelY, GROUNDY - playery -
playerHeight)

        # move pipes to the left

        for upperPipe , lowerPipe in zip(upperPipes, lowerPipes):
            upperPipe['x'] += pipeVelX
            lowerPipe['x'] += pipeVelX

        # Add a new pipe when the first is about to cross the leftmost
part of the screen

        if 0<upperPipes[0]['x']<5:
            newpipe = getRandomPipe()

```



```

upperPipes.append(newpipe[0])

lowerPipes.append(newpipe[1])


# if the pipe is out of the screen, remove it
if upperPipes[0]['x'] < -GAME_SPRITES['pipe'][0].get_width():
    upperPipes.pop(0)
    lowerPipes.pop(0)


# Lets blit our sprites now
SCREEN.blit(GAME_SPRITES['background'], (0, 0))
for upperPipe, lowerPipe in zip(upperPipes, lowerPipes):
    SCREEN.blit(GAME_SPRITES['pipe'][0], (upperPipe['x'],
upperPipe['y']))
    SCREEN.blit(GAME_SPRITES['pipe'][1], (lowerPipe['x'],
lowerPipe['y']))


SCREEN.blit(GAME_SPRITES['base'], (basex, GROUNDY))
SCREEN.blit(GAME_SPRITES['player'], (playerx, playery))
myDigits = [int(x) for x in list(str(score))]
width = 0
for digit in myDigits:
    width += GAME_SPRITES['numbers'][digit].get_width()
Xoffset = (SCREENWIDTH - width)/2


for digit in myDigits:

```

```

        SCREEN.blit(GAME_SPRITES['numbers'][digit], (Xoffset,
SCREENHEIGHT*0.12))

        Xoffset += GAME_SPRITES['numbers'][digit].get_width()

    pygame.display.update()

    FPSCLOCK.tick(FPS)

def isCollide(playerx, playery, upperPipes, lowerPipes):
    if playery> GROUNDY - 25 or playery<0:
        GAME_SOUNDS['hit'].play()
        return True

    for pipe in upperPipes:
        pipeHeight = GAME_SPRITES['pipe'][0].get_height()
        if(playery < pipeHeight + pipe['y'] and abs(playerx - pipe['x']) <
GAME_SPRITES['pipe'][0].get_width()):
            GAME_SOUNDS['hit'].play()
            return True

    for pipe in lowerPipes:
        if (playery + GAME_SPRITES['player'].get_height() > pipe['y'])
and abs(playerx - pipe['x']) < GAME_SPRITES['pipe'][0].get_width():
            GAME_SOUNDS['hit'].play()
            return True

    return False

```

```

def getRandomPipe():
    """
    Generate positions of two pipes(one bottom straight and one top
    rotated ) for blitting on the screen
    """
    pipeHeight = GAME_SPRITES['pipe'][0].get_height()
    offset = SCREENHEIGHT/3
    y2 = offset + random.randrange(0, int(SCREENHEIGHT -
GAME_SPRITES['base'].get_height() - 1.2 *offset))
    pipeX = SCREENWIDTH + 10
    y1 = pipeHeight - y2 + offset
    pipe = [
        {'x': pipeX, 'y': -y1}, #upper Pipe
        {'x': pipeX, 'y': y2} #lower Pipe
    ]
    return pipe

if __name__ == "__main__":
    # This will be the main point from where our game will start
    pygame.init() # Initialize all pygame's modules
    FPSLOCK = pygame.time.Clock()
    pygame.display.set_caption('Flappy Bird by DEBARATHI')
    GAME_SPRITES['numbers'] = (
        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\0.png').convert_alpha(),

```

```

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\1.png').convert_alpha(),

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\2.png').convert_alpha(),

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\3.png').convert_alpha(),

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\4.png').convert_alpha(),

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\5.png').convert_alpha(),

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\6.png').convert_alpha(),

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\7.png').convert_alpha(),

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\8.png').convert_alpha(),

        pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Flappy Bird\\gallery\\sprites\\9.png').convert_alpha(),

    )

    GAME_SPRITES['message']
=pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\sprites\\message.png').convert_alpha()

    GAME_SPRITES['base']
=pygame.image.load('C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\sprites\\base.png').convert_alpha()

    GAME_SPRITES['pipe']
=(pygame.transform.rotate(pygame.image.load(PIPE).convert_alpha(),
180),

    pygame.image.load(PIPE).convert_alpha()

```

```

)

# Game sounds

GAME_SOUNDS['die'] =
pygame.mixer.Sound('C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\audio\\die.wav')

GAME_SOUNDS['hit'] =
pygame.mixer.Sound('C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\audio\\hit.wav')

GAME_SOUNDS['point'] =
pygame.mixer.Sound('C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\audio\\point.wav')

GAME_SOUNDS['swoosh'] =
pygame.mixer.Sound('C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\audio\\swoosh.wav')

GAME_SOUNDS['wing'] =
pygame.mixer.Sound('C:\\Users\\Acer\\Desktop\\PROJ - CS881 Project -
III\\Flappy Bird\\gallery\\audio\\wing.wav')

GAME_SPRITES['background'] =
pygame.image.load(BACKGROUND).convert()

GAME_SPRITES['player'] =
pygame.image.load(PLAYER).convert_alpha()

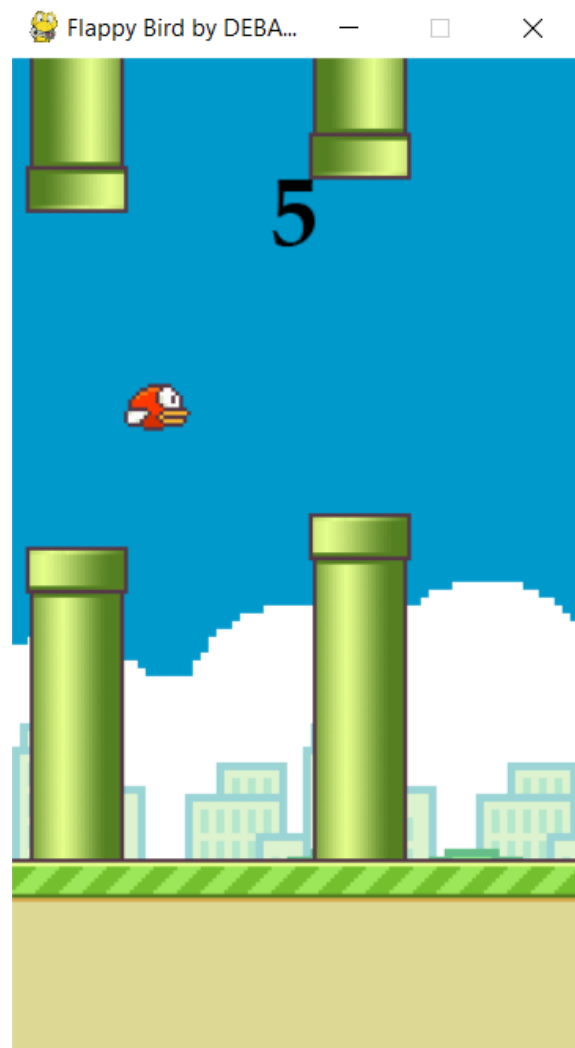
while True:

    welcomeScreen() # Shows welcome screen to the user until he
presses a button

    mainGame() # This is the main game function

```

Output of Flappy Bird Game



3. Pac-Man

Pac-Man is an action maze chase video game; the player controls the eponymous character through an enclosed maze. The objective of the game is to eat all of the dots placed in the maze while avoiding four colored ghosts — Blinky (red), Pinky (pink), Inky (cyan), and Clyde (orange) — that pursue him. When Pac-Man eats all of the dots, the player advances to the next level. If Pac-Man makes contact with a ghost, he will lose a life; the game ends when all lives are lost. Each of the four ghosts have their own unique, distinct artificial intelligence (A.I.), or "personalities"; Blinky gives direct chase to Pac-Man, Pinky and Inky try to position themselves in front of Pac-Man, usually by cornering him, and Clyde will switch between chasing Pac-Man and fleeing from him.

Placed at the four corners of the maze are large flashing "energizers", or "power pellets". Eating these will cause the ghosts to turn blue with a dizzied expression and reverse direction. Pac-Man can eat blue ghosts for bonus points; when eaten, their eyes make their way back to the center box in the maze, where the ghosts "regenerate" and resume their normal activity. Eating multiple blue ghosts in succession increases their point value. After a certain amount of time, blue-colored ghosts will flash white before turning back into their normal, lethal form. Eating a certain number of dots in a level will cause a bonus item - usually in the form of a fruit - to appear underneath the center box, which can be eaten for bonus points.

The game increases in difficulty as the player progresses; the ghosts become faster and the energizers' effect decreases in duration to the point where the ghosts will no longer turn blue and edible. To the sides of the maze are two "warp tunnels", which allow Pac-Man and the ghosts to travel to the opposite side of the screen. Ghosts become slower when entering and exiting these tunnels. Levels are indicated by the fruit icon at the bottom of the screen. In-between levels are short cut scenes featuring Pac-Man and Blinky in humorous, comical situations. The game becomes unplayable at the 256th level due to an integer overflow that affects the game's memory.

Code of Pac-Man Game

```
from random import choice
from turtle import *

from freegames import floor, vector

state = {'score': 0}
path = Turtle(visible=False)
writer = Turtle(visible=False)
aim = vector(5, 0)
pacman = vector(-40, -80)
ghosts = [
    [vector(-180, 160), vector(5, 0)],
    [vector(-180, -160), vector(0, 5)],
    [vector(100, 160), vector(0, -5)],
    [vector(100, -160), vector(-5, 0)],
]
# fmt: off
tiles = [
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
```



```

0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
]
# fmt: on

def square(x, y):
    """Draw square using path at (x, y)."""
    path.up()
    path.goto(x, y)
    path.down()

```

```
path.begin_fill()
```

```
for count in range(4):
```

```
    path.forward(20)
```

```
    path.left(90)
```

```
path.end_fill()
```

```
def offset(point):
```

```
    """Return offset of point in tiles."""
```

```
    x = (floor(point.x, 20) + 200) / 20
```

```
    y = (180 - floor(point.y, 20)) / 20
```

```
    index = int(x + y * 20)
```

```
    return index
```

```
def valid(point):
```

```
    """Return True if point is valid in tiles."""
```

```
    index = offset(point)
```

```
    if tiles[index] == 0:
```

```
        return False
```

```
index = offset(point + 19)

if tiles[index] == 0:
    return False

return point.x % 20 == 0 or point.y % 20 == 0
```

```
def world():
    """Draw world using path."""
    bgcolor('black')
    path.color('blue')

    for index in range(len(tiles)):
        tile = tiles[index]

        if tile > 0:
            x = (index % 20) * 20 - 200
            y = 180 - (index // 20) * 20
            square(x, y)

            if tile == 1:
                path.up()
                path.goto(x + 10, y + 10)
                path.dot(2, 'white')
```

```
def move():  
    """Move pacman and all ghosts."""  
    writer.undo()  
    writer.write(state['score'])  
  
    clear()  
  
    if valid(pacman + aim):  
        pacman.move(aim)  
  
    index = offset(pacman)  
  
    if tiles[index] == 1:  
        tiles[index] = 2  
        state['score'] += 1  
        x = (index % 20) * 20 - 200  
        y = 180 - (index // 20) * 20  
        square(x, y)  
  
    up()  
    goto(pacman.x + 10, pacman.y + 10)  
    dot(20, 'yellow')
```

```
for point, course in ghosts:
    if valid(point + course):
        point.move(course)
    else:
        options = [
            vector(5, 0),
            vector(-5, 0),
            vector(0, 5),
            vector(0, -5),
        ]
        plan = choice(options)
        course.x = plan.x
        course.y = plan.y

up()

goto(point.x + 10, point.y + 10)
dot(20, 'red')

update()

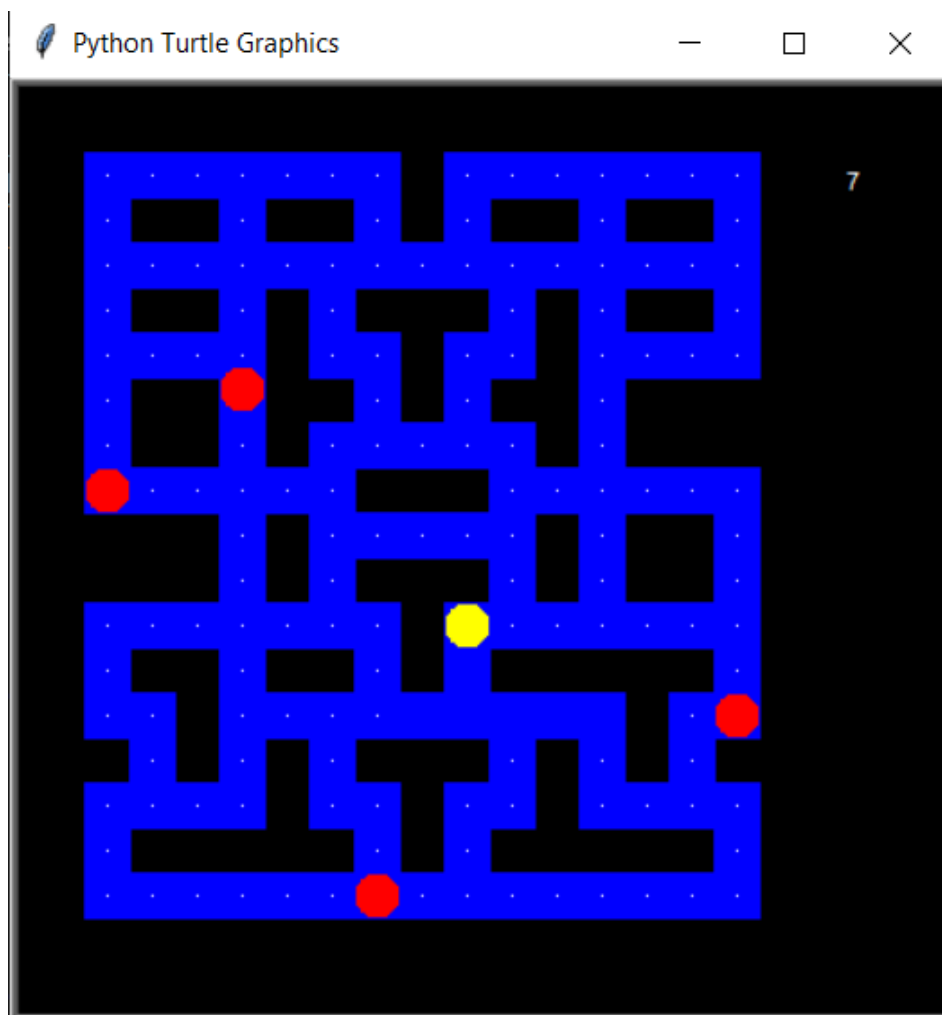
for point, course in ghosts:
    if abs(pacman - point) < 20:
        return
```

```
ontimer(move, 100)

def change(x, y):
    """Change pacman aim if valid."""
    if valid(pacman + vector(x, y)):
        aim.x = x
        aim.y = y

setup(420, 420, 370, 0)
hideturtle()
tracer(False)
writer.goto(160, 160)
writer.color('white')
writer.write(state['score'])
listen()
onkey(lambda: change(5, 0), 'Right')
onkey(lambda: change(-5, 0), 'Left')
onkey(lambda: change(0, 5), 'Up')
onkey(lambda: change(0, -5), 'Down')
world()
move()
done()
```

Output of Pac-Man Game



4. Sliding Puzzle

A sliding puzzle, sliding block puzzle, or sliding tile puzzle is a combination puzzle that challenges a player to slide (frequently flat) pieces along certain routes (usually on a board) to establish a certain end-configuration. The pieces to be moved may consist of simple shapes, or they may be imprinted with colors, patterns, sections of a larger picture (like a jigsaw puzzle), numbers, or letters.

Sliding puzzles are essentially two-dimensional in nature, even if the sliding is facilitated by mechanically interlinked pieces (like partially encaged marbles) or three-dimensional tokens. In manufactured wood and plastic products, the linking and encaging is often achieved in combination, through mortise-and-tenon key channels along the edges of the pieces. In at least one vintage case of the popular Chinese cognate game Huarong Road, a wire screen prevents lifting of the pieces, which remain loose. As the illustration shows, some sliding puzzles are mechanical puzzles. However, the mechanical fixtures are usually not essential to these puzzles; the parts could as well be tokens on a flat board that are moved according to certain rules.

Code of Sliding Puzzle Game

```
from random import *  
from turtle import *  
  
from freegames import floor, vector  
  
tiles = {}
```



```
neighbors = [  
    vector(100, 0),  
    vector(-100, 0),  
    vector(0, 100),  
    vector(0, -100),  
]  
  
def load():  
    """Load tiles and scramble."""  
    count = 1  
  
    for y in range(-200, 200, 100):  
        for x in range(-200, 200, 100):  
            mark = vector(x, y)  
            tiles[mark] = count  
            count += 1  
  
        tiles[mark] = None  
  
    for count in range(1000):  
        neighbor = choice(neighbors)  
        spot = mark + neighbor  
  
        if spot in tiles:
```

```
number = tiles[spot]
tiles[spot] = None
tiles[mark] = number
mark = spot
```

```
def square(mark, number):
```

```
    """Draw white square with black outline and number."""
```

```
    up()
```

```
    goto(mark.x, mark.y)
```

```
    down()
```

```
    color('black', 'white')
```

```
    begin_fill()
```

```
    for count in range(4):
```

```
        forward(99)
```

```
        left(90)
```

```
    end_fill()
```

```
    if number is None:
```

```
        return
```

```
    elif number < 10:
```

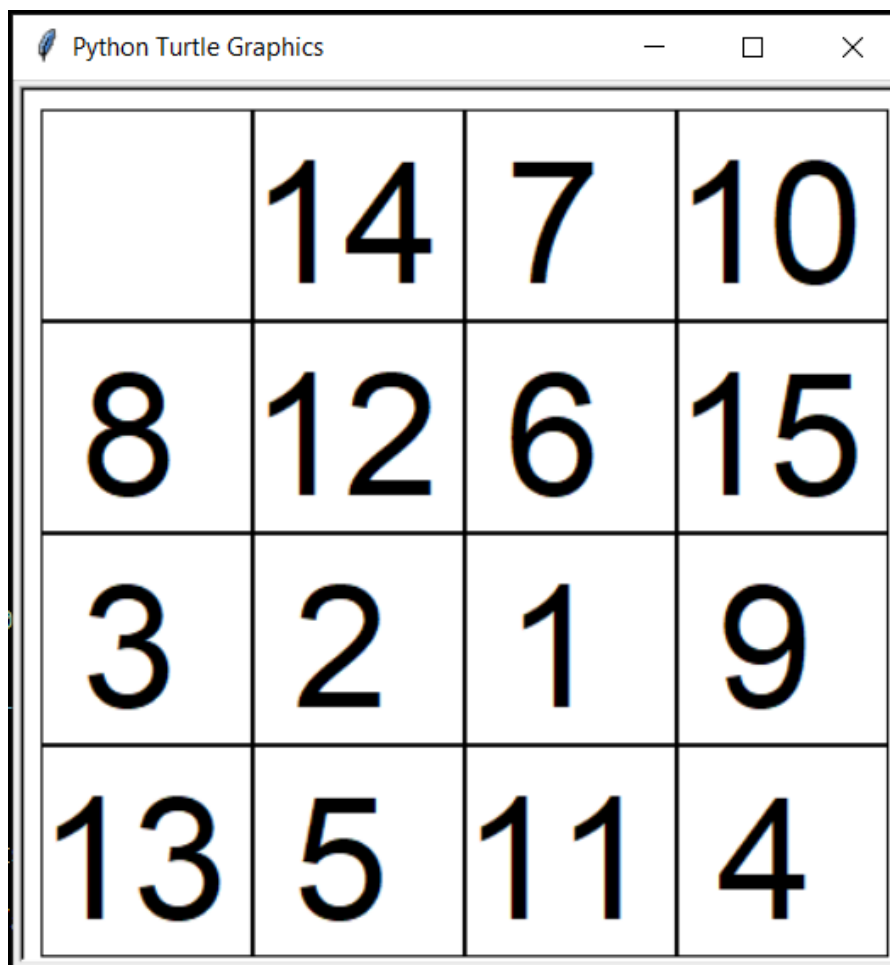
```
        forward(20)
```

```
    write(number, font=('Arial', 60, 'normal'))
```

```
def tap(x, y):  
    """Swap tile and empty square."""  
    x = floor(x, 100)  
    y = floor(y, 100)  
    mark = vector(x, y)  
  
    for neighbor in neighbors:  
        spot = mark + neighbor  
  
        if spot in tiles and tiles[spot] is None:  
            number = tiles[mark]  
            tiles[spot] = number  
            square(spot, number)  
            tiles[mark] = None  
            square(mark, None)  
  
def draw():  
    """Draw all tiles."""  
    for mark in tiles:  
        square(mark, tiles[mark])  
    update()  
setup(420, 420, 370, 0)  
hideturtle()
```

```
tracer(False)
load()
draw()
onscreenclick(tap)
done()
```

Output of Sliding Puzzle Game



The image shows a screenshot of a Python Turtle Graphics window titled "Python Turtle Graphics". Inside the window is a 4x4 grid representing a sliding puzzle. The grid contains the following numbers:

	14	7	10
8	12	6	15
3	2	1	9
13	5	11	4

5. Tic-Tac-Toe

Games played on three-in-a-row boards can be traced back to ancient Egypt, where such game boards have been found on roofing tiles dating from around 1300 BC.

An early variation of tic-tac-toe was played in the Roman Empire, around the first century BC. It was called Terni lapilli (three pebbles at a time) and instead of having any number of pieces, each player had only three; thus, they had to move them around to empty spaces to keep playing. The game's grid markings have been found chalked all over Rome. Another closely related ancient game is three men's morris which is also played on a simple grid and requires three pieces in a row to finish, and Picaria, a game of the Pueblos.

The different names of the game are more recent. The first print reference to "noughts and crosses" (nought being an alternative word for 'zero'), the British name, appeared in 1858, in an issue of Notes and Queries. The first print reference to a game called "tick-tack-toe" occurred in 1884, but referred to "a children's game played on a slate, consisting of trying with the eyes shut to bring the pencil down on one of the numbers of a set, the number hit being scored".[This quote needs a citation] "Tic-tac-toe" may also derive from "tick-tack", the name of an old version of backgammon first described in 1558. The US renaming of "noughts and crosses" to "tic-tac-toe" occurred in the 20th century.

In 1975, tic-tac-toe was also used by MIT students to demonstrate the computational power of Tinker toy elements. The Tinker toy computer, made out of (almost) only Tinker toys, is able to play tic-tac-toe perfectly. It is currently on display at the Museum of Science, Boston.

Code of Tic-Tac-Toe Game

```
#import needed library
from tkinter import *
import random as r

#Function to define a button
def CellButton(frame):
    b=Button(frame,padx=1,bg="#27386B",width=3,text="
",font=('algerian',60,'bold'),relief="sunken",bd=5)
    return b

#changing operator for the next player
def changeLetter():
    global a
    for i in ['O','X']:
        if not(i==a):
            a=i
            break

#Reset the gameboard
def resetBoard():
    global a
    for i in range(3):
        for j in range(3):
            board[i][j]["text"]=" "
            board[i][j]["state"]=NORMAL
```

```

a=r.choice(['O','X'])
#check for winning
def checkWinning():
    for i in range(3):
        if(board[i][0]["text"]==board[i][1]["text"]==board[i][2]["text"]==a
or board[0][i]["text"]==board[1][i]["text"]==board[2][i]["text"]==a):
            if a == 'X':
                lblMesage.config(text="" + firstPlayer.get() + " is Winner")
            else:
                lblMesage.config(text="" + secondPlayer.get() + " is Winner")
            resetBoard()

    if(board[0][0]["text"]==board[1][1]["text"]==board[2][2]["text"]==a or
board[0][2]["text"]==board[1][1]["text"]==board[2][0]["text"]==a):
        if a =='X':
            lblMesage.config(text="" + firstPlayer.get() + " is Winner")
        else:
            lblMesage.config(text="" + secondPlayer.get() + " is Winner")
        resetBoard()

    elif(board[0][0]["state"]==board[0][1]["state"]==board[0][2]["state"]==bo
ard[1][0]["state"]==board[1][1]["state"]==board[1][2]["state"]==board[2][
0]["state"]==board[2][1]["state"]==board[2][2]["state"]==DISABLED):
        lblMesage.config(text="The match is Tied!")
        resetBoard()

def click(row,col):

```

```

board[row][col].config(text=a,state=DISABLED,disabledforeground=colour
[a])

    lblMessage.config(text="")

    checkWinning()

    changeLetter()

    if a=='X':

        label.config(text=firstPlayer.get()+"'s Chance")

    else:

        label.config(text=secondPlayer.get() + "'s Chance")

#Defining window
root=Tk()

#Setting title for window
root.title("Tic-Tac-Toe Game")

#setting width and height for game window
root.geometry("800x490")

#setting background color
root["bg"]=" #03151C"

#declaring variables
firstPlayer=StringVar()

secondPlayer=StringVar()

#label for first player
Label(root,text="First                                     Player
[X]",bg=" #03151C",fg="white",font=("Arial",15,"bold")).place(x=550,y=20)

```



```

#textbox for first player

Entry(root,font=("Arial",15,"bold"),textvariable=firstPlayer).place(x=550,y
=55)

#label for second player

Label(root,text="Second                                Player
[0]",bg="#03151C",fg="white",font=("Arial",15,"bold")).place(x=550,y=90)

#textbox for second player

Entry(root,font=("Arial",15,"bold"),textvariable=secondPlayer).place(x=55
0,y=125)

#label for displaying game result

lblMessage=Label(root,bg="#03151C",fg="yellow",font=("Arial",15,"bold"))
lblMessage.place(x=550,y=185)

#Defining Two operators

a=r.choice(['O','X'])

#setting color for operators

colour={'O':"yellow",'X':"white"}

#variable for creating board

board=[[[],[],[]]]

for i in range(3):
    for j in range(3):
        board[i].append(CellButton(root))
        board[i][j].config(command= lambda row=i,col=j:click(row,col))
        board[i][j].grid(row=i,column=j)

#label for displaying player's turn

```

```

label=Label(text="Welcome",font=('arial',15,'bold'),bg="#03151C",fg="white")

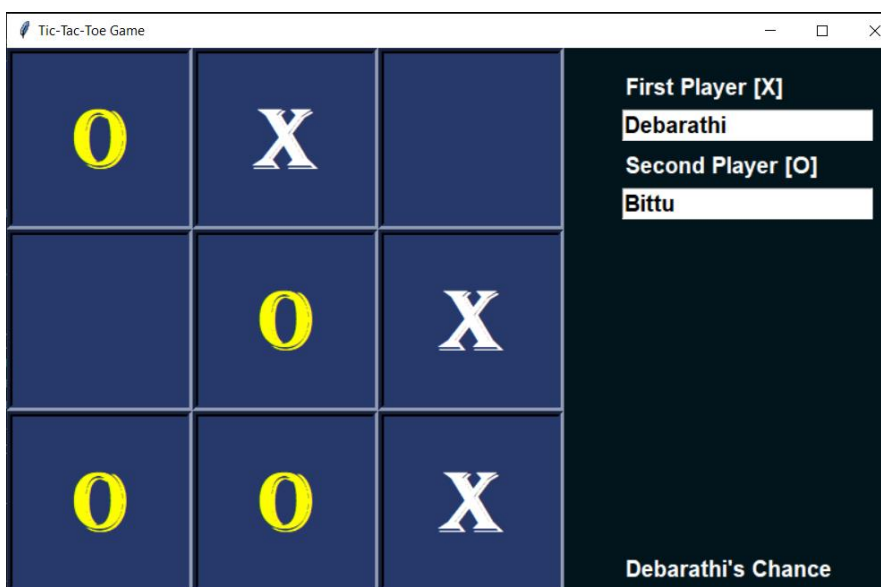
label.place(x=550,y=450)

#run Application

root.mainloop()

```

Output of Tic-Tac-Toe Game



6. Pong Game

Pong is a table tennis–themed twitch arcade sports video game, featuring simple two-dimensional graphics, manufactured by Atari and originally released in 1972. It was one of the earliest arcade video games; it was created by Allan Alcorn as a training exercise assigned to him by Atari co-founder Nolan Bushnell, but Bushnell and Atari co-founder Ted Dabney were surprised by the quality of Alcorn's work and decided to manufacture the game. Bushnell based the game's concept on an electronic ping-pong game included in the Magnavox Odyssey, the first home video game console. In response, Magnavox later sued Atari for patent infringement.

Pong was the first commercially successful video game, and it helped to establish the video game industry along with the Magnavox Odyssey. Soon after its release, several companies began producing games that closely mimicked its gameplay. Eventually, Atari's competitors released new types of video games that deviated from Pong's original format to varying degrees, and this, in turn, led Atari to encourage its staff to move beyond Pong and produce more innovative games themselves.

Atari released several sequels to Pong that built upon the original's gameplay by adding new features. During the 1975 Christmas season, Atari released a home version of Pong exclusively through Sears's retail stores. The home version was also a commercial success and led to numerous clones. The game was remade on numerous home and portable platforms following its release. Pong is part of the permanent collection of the Smithsonian Institution in Washington, D.C., due to its cultural impact.

Code of Pong Game

```
import pygame
import sys
import random
from math import *

pygame.init()

width = 600
height = 400
display = pygame.display.set_mode((width, height))
pygame.display.set_caption(" Pong ")
clock = pygame.time.Clock()

background = (27, 38, 49)
white = (236, 240, 241)
red = (203, 67, 53)
blue = (52, 152, 219)
yellow = (244, 208, 63)

top = white
bottom = white
left = white
```

```
right = white

margin = 4

scoreLeft = 0
scoreRight = 0
maxScore = 20

font = pygame.font.SysFont("Small Fonts", 30)
largeFont = pygame.font.SysFont("Small Fonts", 60)

# Draw the Boundary of Board
def boundary():
    global top, bottom, left, right
    pygame.draw.rect(display, left, (0, 0, margin, height))
    pygame.draw.rect(display, top, (0, 0, width, margin))
    pygame.draw.rect(display, right, (width-margin, 0, margin, height))
    pygame.draw.rect(display, bottom, (0, height - margin, width, margin))

l = 25

pygame.draw.rect(display, white, (width/2-margin/2, 10, margin, l))
pygame.draw.rect(display, white, (width/2-margin/2, 60, margin, l))
pygame.draw.rect(display, white, (width/2-margin/2, 110, margin, l))
```

```
pygame.draw.rect(display, white, (width/2-margin/2, 160, margin, l))
pygame.draw.rect(display, white, (width/2-margin/2, 210, margin, l))
pygame.draw.rect(display, white, (width/2-margin/2, 260, margin, l))
pygame.draw.rect(display, white, (width/2-margin/2, 310, margin, l))
pygame.draw.rect(display, white, (width/2-margin/2, 360, margin, l))
```

Paddle Class

class Paddle:

```
def __init__(self, position):
```

```
    self.w = 10
```

```
    self.h = self.w*8
```

```
    self.paddleSpeed = 6
```

```
    if position == -1:
```

```
        self.x = 1.5*margin
```

```
    else:
```

```
        self.x = width - 1.5*margin - self.w
```

```
    self.y = height/2 - self.h/2
```

Show the Paddle

```
def show(self):
```

```
    pygame.draw.rect(display, white, (self.x, self.y, self.w, self.h))
```

```
# Move the Paddle

def move(self, ydir):

    self.y += self.paddleSpeed*ydir

    if self.y < 0:

        self.y -= self.paddleSpeed*ydir

    elif self.y + self.h> height:

        self.y -= self.paddleSpeed*ydir


leftPaddle = Paddle(-1)
rightPaddle = Paddle(1)


# Ball Class
class Ball:

    def __init__(self, color):

        self.r = 20

        self.x = width/2 - self.r/2

        self.y = height/2 -self.r/2

        self.color = color

        self.angle = random.randint(-75, 75)

        if random.randint(0, 1):

            self.angle += 180

        self.speed = 8
```

```
# Show the Ball

def show(self):

    pygame.draw.ellipse(display, self.color, (self.x, self.y, self.r, self.r))


# Move the Ball

def move(self):

    global scoreLeft, scoreRight

    self.x += self.speed*cos(radians(self.angle))

    self.y += self.speed*sin(radians(self.angle))

    if self.x + self.r > width - margin:

        scoreLeft += 1

        self.angle = 180 - self.angle

    if self.x < margin:

        scoreRight += 1

        self.angle = 180 - self.angle

    if self.y < margin:

        self.angle = - self.angle

    if self.y + self.r >= height - margin:

        self.angle = - self.angle


# Check and Reflect the Ball when it hits the padddle

def checkForPaddle(self):

    if self.x < width/2:

        if leftPaddle.x < self.x < leftPaddle.x + leftPaddle.w:
```



```

        if leftPaddle.y < self.y < leftPaddle.y + 10 or leftPaddle.y < self.y +
self.r < leftPaddle.y + 10:

            self.angle = -45

        if leftPaddle.y + 10 < self.y < leftPaddle.y + 20 or leftPaddle.y + 10 <
self.y + self.r < leftPaddle.y + 20:

            self.angle = -30

        if leftPaddle.y + 20 < self.y < leftPaddle.y + 30 or leftPaddle.y + 20 <
self.y + self.r < leftPaddle.y + 30:

            self.angle = -15

        if leftPaddle.y + 30 < self.y < leftPaddle.y + 40 or leftPaddle.y + 30 <
self.y + self.r < leftPaddle.y + 40:

            self.angle = -10

        if leftPaddle.y + 40 < self.y < leftPaddle.y + 50 or leftPaddle.y + 40 <
self.y + self.r < leftPaddle.y + 50:

            self.angle = 10

        if leftPaddle.y + 50 < self.y < leftPaddle.y + 60 or leftPaddle.y + 50 <
self.y + self.r < leftPaddle.y + 60:

            self.angle = 15

        if leftPaddle.y + 60 < self.y < leftPaddle.y + 70 or leftPaddle.y + 60 <
self.y + self.r < leftPaddle.y + 70:

            self.angle = 30

        if leftPaddle.y + 70 < self.y < leftPaddle.y + 80 or leftPaddle.y + 70 <
self.y + self.r < leftPaddle.y + 80:

            self.angle = 45

    else:

        if rightPaddle.x + rightPaddle.w > self.x + self.r > rightPaddle.x:

```

```
        if rightPaddle.y < self.y < leftPaddle.y + 10 or leftPaddle.y < self.y +
self.r < leftPaddle.y + 10:

            self.angle = -135

        if rightPaddle.y + 10 < self.y < rightPaddle.y + 20 or rightPaddle.y +
10 < self.y + self.r < rightPaddle.y + 20:

            self.angle = -150

        if rightPaddle.y + 20 < self.y < rightPaddle.y + 30 or rightPaddle.y +
20 < self.y + self.r < rightPaddle.y + 30:

            self.angle = -165

        if rightPaddle.y + 30 < self.y < rightPaddle.y + 40 or rightPaddle.y +
30 < self.y + self.r < rightPaddle.y + 40:

            self.angle = 170

        if rightPaddle.y + 40 < self.y < rightPaddle.y + 50 or rightPaddle.y +
40 < self.y + self.r < rightPaddle.y + 50:

            self.angle = 190

        if rightPaddle.y + 50 < self.y < rightPaddle.y + 60 or rightPaddle.y +
50 < self.y + self.r < rightPaddle.y + 60:

            self.angle = 165

        if rightPaddle.y + 60 < self.y < rightPaddle.y + 70 or rightPaddle.y +
60 < self.y + self.r < rightPaddle.y + 70:

            self.angle = 150

        if rightPaddle.y + 70 < self.y < rightPaddle.y + 80 or rightPaddle.y +
70 < self.y + self.r < rightPaddle.y + 80:

            self.angle = 135

# Show the Score

def showScore():
```

```
leftScoreText = font.render("Score : " + str(scoreLeft), True, red)
rightScoreText = font.render("Score : " + str(scoreRight), True, blue)

display.blit(leftScoreText, (3*margin, 3*margin))
display.blit(rightScoreText, (width/2 + 3*margin, 3*margin))

# Game Over
def gameOver():
    if scoreLeft == maxScore or scoreRight == maxScore:
        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    close()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        close()
                    if event.key == pygame.K_r:
                        reset()
            if scoreLeft == maxScore:
                playerWins = largeFont.render("Left Player Wins!", True, red)
            elif scoreRight == maxScore:
                playerWins = largeFont.render("Right Player Wins!", True, blue)

            display.blit(playerWins, (width/2 - 100, height/2))
```

```
pygame.display.update()

def reset():
    global scoreLeft, scoreRight
    scoreLeft = 0
    scoreRight = 0
    board()

def close():
    pygame.quit()
    sys.exit()

def board():
    loop = True
    leftChange = 0
    rightChange = 0
    ball = Ball(yellow)

    while loop:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                close()

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
```

```
    close()

    if event.key == pygame.K_SPACE or event.key == pygame.K_p:
        Pause()

    if event.key == pygame.K_r:
        reset()

    if event.key == pygame.K_w:
        leftChange = -1

    if event.key == pygame.K_s:
        leftChange = 1

    if event.key == pygame.K_UP:
        rightChange = -1

    if event.key == pygame.K_DOWN:
        rightChange = 1

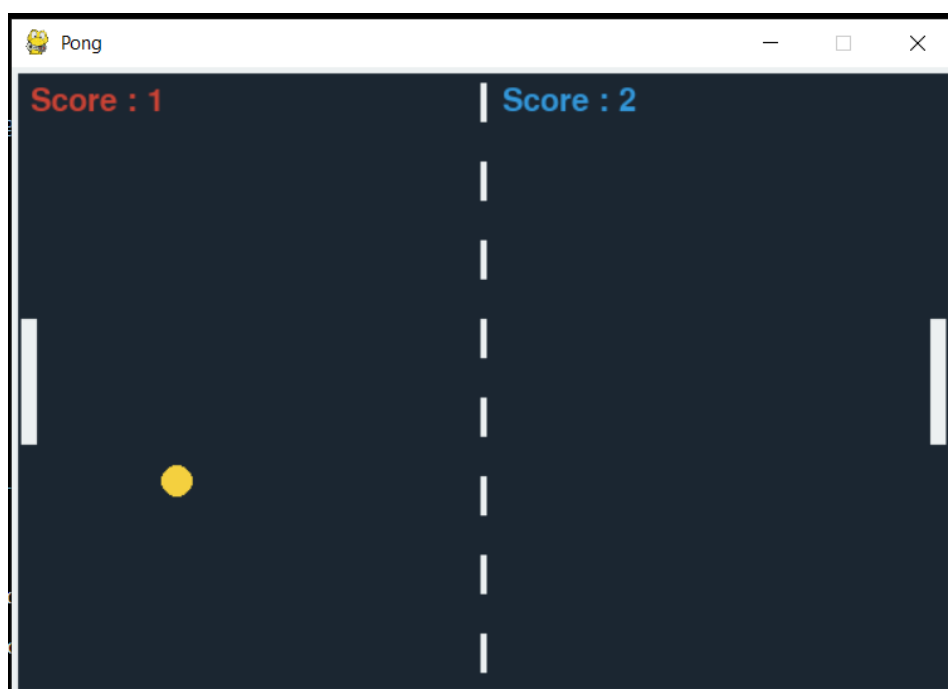
    if event.type == pygame.KEYUP:
        leftChange = 0
        rightChange = 0

    leftPaddle.move(leftChange)
    rightPaddle.move(rightChange)
    ball.move()
    ball.checkForPaddle()

    display.fill(background)
    showScore()
```

```
ball.show()  
leftPaddle.show()  
rightPaddle.show()  
  
boundary()  
  
gameOver()  
  
pygame.display.update()  
clock.tick(60)  
board()
```

Output of Pong Game



7. Rock Paper Scissor

Rock Paper Scissors (also known by other orderings of the three items, with "rock" sometimes being called "stone", or as Rochambeau, roshambo, orro-sham-bo) is a hand game originating from China, usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. These shapes are "rock" (a closed fist), "paper" (a flat hand), and "scissors" (a fist with the index finger and middle finger extended, forming a V). "Scissors" is identical to the two-fingered V sign (also indicating "victory" or "peace") except that it is pointed horizontally instead of being held upright in the air.

A simultaneous, zero-sum game, it has three possible outcomes: a draw, a win or a loss. A player who decides to play rock will beat another player who has chosen scissors ("rock crushes scissors" or "breaks scissors" or sometimes "blunts scissors"[4]), but will lose to one who has played paper ("paper covers rock"); a play of paper will lose to a play of scissors ("scissors cuts paper"). If both players choose the same shape, the game is tied and is usually immediately replayed to break the tie. The type of game originated in China and spread with increased contact with East Asia, while developing different variants in signs over time.

Rock Paper Scissors is often used as a fair choosing method between two people, similar to coin flipping, drawing straws, or throwing dice in order to settle a dispute or make an unbiased group decision. Unlike truly random selection methods, however, rock paper scissors can be played with a degree of skill by recognizing and exploiting non-random behavior in opponents.

Code of Rock Paper Scissor Game

```
# Import Required Library
from tkinter import *
import random

# Create Object
root = Tk()
root.geometry("500x500")
root.title("Rock Paper Scissor Game")
root.configure(bg="orange")
root.resizable(False,False)


# Computer Value
computer_value = {
    "0":"Rock",
    "1":"Paper",
    "2":"Scissor"
}


# Reset The Game
def reset_game():
    b1["state"] = "active"
    b2["state"] = "active"
    b3["state"] = "active"
```



```
l1.config(text = "Player      ")
l3.config(text = "Computer")
l4.config(text = "")

# Disable the Button
def button_disable():
    b1["state"] = "disable"
    b2["state"] = "disable"
    b3["state"] = "disable"

# If player selected rock
def isrock():
    c_v = computer_value[str(random.randint(0,2))]
    if c_v == "Rock":
        match_result = "Match Draw"
    elif c_v=="Scissor":
        match_result = "Player Win"
    else:
        match_result = "Computer Win"
    l4.config(text = match_result)
    l1.config(text = "Rock      ")
    l3.config(text = c_v)
    button_disable()
```

```
# If player selected paper
def ispaper():
    c_v = computer_value[str(random.randint(0, 2))]
    if c_v == "Paper":
        match_result = "Match Draw"
    elif c_v == "Scissor":
        match_result = "Computer Win"
    else:
        match_result = "Player Win"
    l4.config(text = match_result)
    l1.config(text = "Paper    ")
    l3.config(text = c_v)
    button_disable()

# If player selected scissor
def isscissor():
    c_v = computer_value[str(random.randint(0,2))]
    if c_v == "Rock":
        match_result = "Computer Win"
    elif c_v == "Scissor":
        match_result = "Match Draw"
    else:
        match_result = "Player Win"
    l4.config(text = match_result)
```

```

l1.config(text = "Scissor    ")

l3.config(text = c_v)

button_disable()

# Add Labels, Frames and Button

Label(root,text = " * Rock Paper Scissor * ",font = "Cambria 20 bold",fg =
"black",bg="white").pack(pady = 20)

frame = Frame(root)
frame.pack()

l1 = Label(frame,
            text = "PLAYER      ",font = "Cambria 13 bold",bg="orange")

l2 = Label(frame,text = "VS      ",font = "Cambria 13 bold",bg="orange")

l3 = Label(frame, text = "COMPUTER", font = "Cambria 13
bold",bg="orange")

l1.pack(side = LEFT)
l2.pack(side = LEFT)
l3.pack()

l4 = Label(root,
            text = "",

```

```

        font = "normal 20 bold",
        bg = "white",
        width = 15 ,
        borderwidth = 2,
        relief = "solid")
l4.pack(pady = 20)

frame1 = Frame(root)
frame1.pack()
frame1.configure(bg="orange")

b1  =  Button(frame1,  text  =  "ROCK",font  =  "Cambria  13
bold",bg="green",fg="white", width = 7,command = isrock)

canvas1 = Canvas(width=58, height=58, bg="white")
canvas1.pack()
canvas1.place(x=118,y=230)
photo1  =  PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ  -  CS881
Project - III\\rock1.png')
canvas1.create_image(0,0, image=photo1, anchor=NW)

b2  =  Button(frame1,  text  =  "PAPER",font  =  "Cambria  13
bold",bg="navyblue",fg="white", width = 7,command = ispaper)

canvas2 = Canvas(width=58, height=58, bg="white")

```

```
canvas2.pack()

canvas2.place(x=218,y=230)

photo2 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\paper1.png')

canvas2.create_image(0,0, image=photo2, anchor=NW)


b3 = Button(frame1, text = "SCISSOR",font ="Cambria 13
bold",bg="red",fg="white", width = 7,command = isscissor)


canvas3 = Canvas(width=58, height=58, bg="white")
canvas3.pack()

canvas3.place(x=318,y=230)

photo3 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\scissors1.png')

canvas3.create_image(0,0, image=photo3, anchor=NW)


b1.pack(side = LEFT, padx = 10)
b2.pack(side = LEFT, padx = 10)
b3.pack(padx = 10)


btn=Button(root, text = "RESET GAME",font ="Cambria 13 bold", fg =
"white",bg = "maroon", command = reset_game)

btn.pack(pady = 20)

btn.place(x=185,y=320)
```

```

btn      =      Button(root,      text="      EXIT      ",font="Cambria      14
bold",fg="white",bg="grey",command=lambda: root.destroy())

btn.pack(packx=20, pady=20)

btn.place(x=210,y=390)

root.mainloop()

```

Output of Rock Paper Scissor Game



8. Bubble Sort Visual

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.

Example:

First Pass:

(5 1 4 2 8) \rightarrow (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since $5 > 1$.

(1 5 4 2 8) \rightarrow (1 4 5 2 8), Swap since $5 > 4$

(1 4 5 2 8) \rightarrow (1 4 2 5 8), Swap since $5 > 2$

(1 4 2 5 8) \rightarrow (1 4 2 5 8), Now, since these elements are already in order ($8 > 5$), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) \rightarrow (1 4 2 5 8)

(1 4 2 5 8) \rightarrow (1 2 4 5 8), Swap since $4 > 2$

(1 2 4 5 8) \rightarrow (1 2 4 5 8)

1 2 4 5 8) \rightarrow (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:

(1 2 4 5 8) \rightarrow (1 2 4 5 8)

(1 2 4 5 8) \rightarrow (1 2 4 5 8)

(1 2 4 5 8) \rightarrow (1 2 4 5 8)

(1 2 4 5 8) \rightarrow (1 2 4 5 8)

Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

Code of Bubble Sort Visual

```
import pygame

#Initialise pygame window
pygame.init()

array1 = ""
array = []

#Define screen dimensions, declare font for viewing text
screen = pygame.display.set_mode((700,500))
font = pygame.font.SysFont('ubuntu mono', 20)
run = True


#Initially fill the screen black
screen.fill((0,0,0,0))

block = font.render("Visualization Of Bubble Sort by
Project Group - 2", True, (255,0,150))

screen.blit(block, (0,20))

block1 = font.render("Enter Input and press
ENTER to visualize", True, (255,255,150))

screen.blit(block1, (0,40))
```



```

block2 = font.render("                Add comma to separate the
integers and backspace to pop", True, (255,255,150))

screen.blit(block2, (0,60))

pygame.display.update()


#Function to show text
def show_text(array):
    #Create a new screen
    screen.fill((0,0,150,0))

    #Use the font to display the array
    block = font.render(str(array), True, (255,255,150))

    #Display the array
    screen.blit(block, (20,20))


#Function to draw rectangles
def draw_rect():
    for i in range(len(array)):
        #Draw rectangles using array elements

        #To maintain gaps between rectangles, mention the top coordinate
        more than the width

        pygame.draw.rect(screen, (255, 125, 0),((50+i*25,50, 20, array[i]*2)))

    pygame.display.update()


#Function to implement bubble sort Implementer using PyGame
def bubble_sort():

```

```
for i in range(len(array)):
    for j in range(len(array)-1):
        #Compare every element with every other element and switch places
        if array[j] > array[j+1]:
            array[j], array[j+1] = array[j+1], array[j]
    #Display the array
    array1 = [str(i) for i in array]
    array1=",".join(array1)
    show_text(array1)
    #Draw the rectangular boxes
    draw_rect()
    #Keep a delay between changes
    pygame.time.delay(500)
    #Display the changes made
    pygame.display.update()

#Since changes keep happening, a loop is used
while run == True:
    #Detect keyboard press
    for event in pygame.event.get():
        #Keyboard press condition
        if event.type == pygame.KEYDOWN:
            #Spacebar press
            if event.key == pygame.K_RETURN:
```

```
#Start visualising and sorting

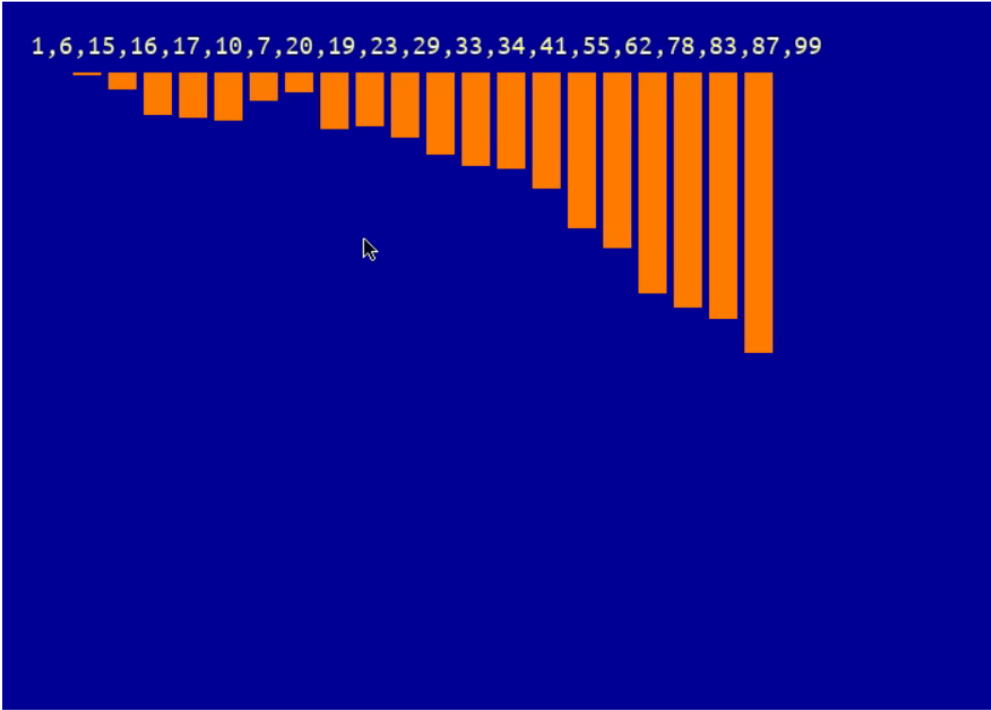
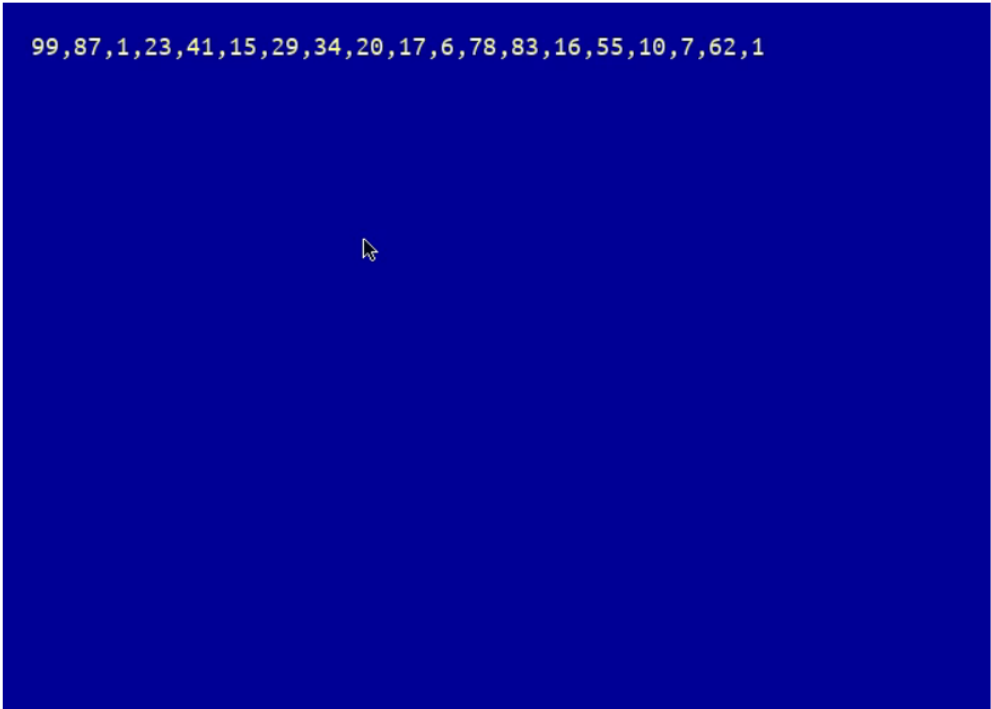
#Convert string to array by splitting the string
array = array1.split(",")
array = [int(i) for i in array]
draw_rect()
pygame.time.delay(3000)
bubble_sort()

elif event.key == pygame.K_BACKSPACE:
    #Remove last element in case of any changes
    array1 = array1[:-1]
else:
    #Check if the keyboard press is a digit
    array1+=event.unicode
    show_text(array1)
    pygame.display.update()

#Check if pygame exit is selected
elif event.type == pygame.QUIT:
    run= False

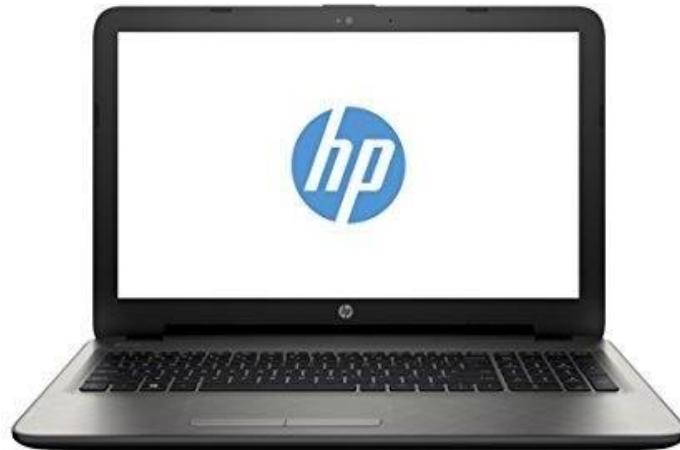
#Quit and close the window
pygame.quit()
```

Output of Bubble Sort Visual



Project Proposal:

Hardware Devices



We will be using a laptop to build our project the specification of that device is given below –

Microprocessor:

Intel® Core™ i3-5005U (2 GHz base frequency, 6 MB cache, 4 cores).

Intel® Core™ i5-10300H (4.5 GHz base frequency, 6 MB cache, 4 cores).



Memory, standard:

8 GB DDR3-1600 SDRAM (2 x 4 GB).

Graphics:

AMD Radeon R5 M430 (2000 MB DDR3 dedicated).



NVIDIA GTX 1650TI with 4GB of dedicated GDDR6 VRAM.



Storage:

1. Hard Drive

1 TB 7200 rpm SATA.

2. SSD

M.2 PCIe 256GB SSD.



Display:

39.62 cm (15.6) HD LED Backlit Widescreen Bright View (1366 x 768).

2 USB 2.0 Gen 1 (Data transfer only); 1 USB 3.1/3.0;

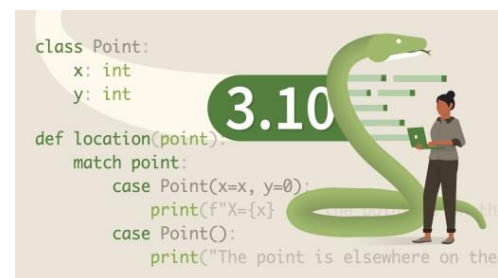
1 HDMI 1.4b; 1 RJ-45; 1 headphone/microphone combo.

Software's & Module's We used to build this Project

Software

1. Python 3.10 :

Python 3.10.0 is the newest major release of the Python programming language, and it contains many new features and optimizations.



2. Visual Studio Code :

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.



3. PyCharm:

PyCharm is an integrated development environment used in computer programming, specifically for the Python programming language. It is developed by the Czech company JetBrains.



4. Sublime Text :

Sublime Text is a shareware cross-platform source code editor. It natively supports many programming languages and markup languages. Users can expand its functionality with plugins, typically community-built and maintained under free-software licenses. To facilitate plugins, Sublime Text features a Python API.



5. Anaconda :

Anaconda is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS.



6. Jupyter Notebook :

The Jupyter Notebook is the original web application for creating and Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Pérez and Brian Granger.



Module

1. Tkinter :

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. Import the Tkinter module. In Python, Tkinter is a standard GUI (graphical user interface) package. Tkinter is Python's default GUI module and also the most common way that is used for GuiProgramming in Python.



2. PyGame :

Pygame is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language. Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language.



3. Free Games :

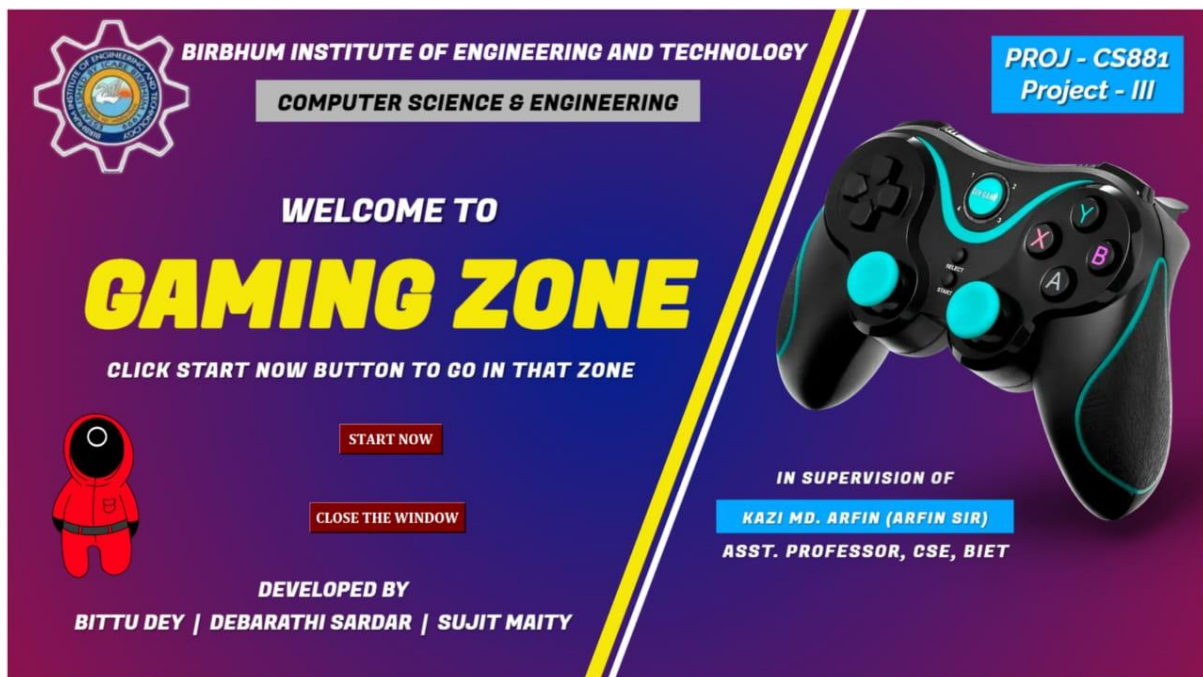
Free Python Games is an Apache2 licensed collection of free Python games intended for education and fun.



OUTPUT

Overview of our final year project:

HOME PAGE



HOME PAGE CODE

```
from tkinter import *
import tkinter

def nextpage():
    root.destroy()
    import Menu_Page
```

```
root = Tk()

canvas = Canvas(width=1550, height=860, bg="black")
canvas.pack()

photo = PhotoImage(file="C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\HomePage.png")

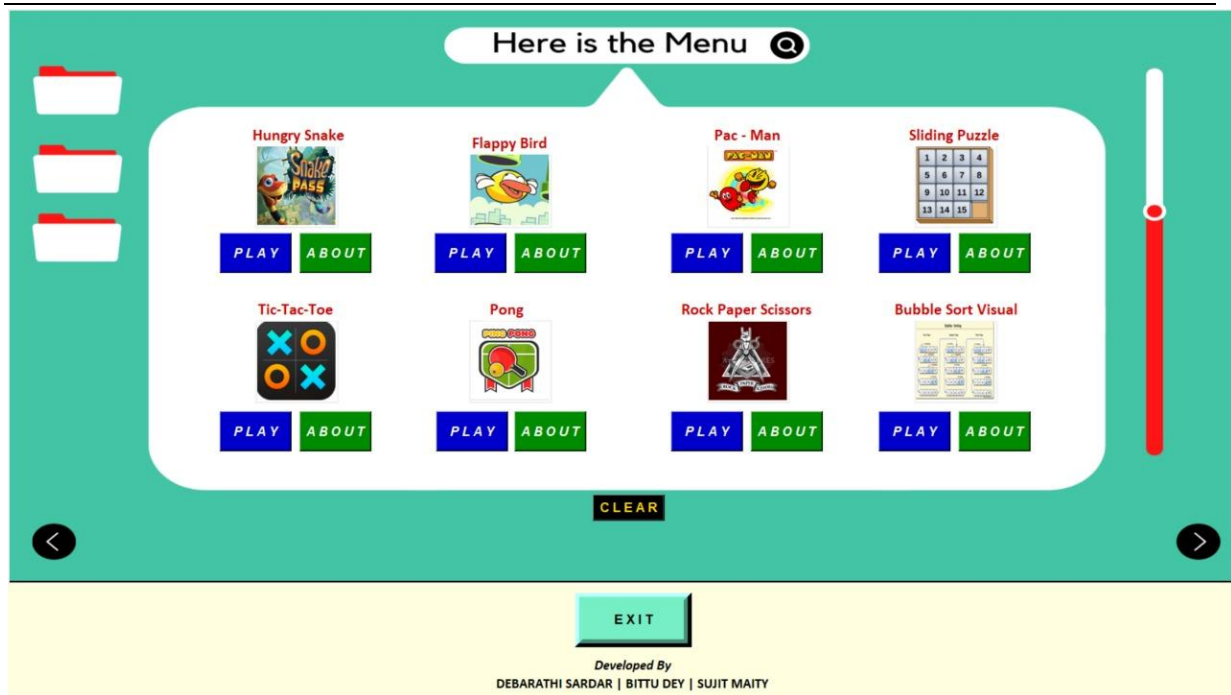
canvas.create_image(0,0, image=photo, anchor=NW)

btn = Button(root, text=" START NOW ",font="Cambria 14
bold",fg="white",bg="maroon",command=nextpage)
btn.pack(padx=25, pady=25)
btn.place(x=419,y=530)

btn = Button(root, text="CLOSE THE WINDOW",font="Cambria 14
bold",fg="white",bg="maroon",command=lambda: root.destroy())
btn.pack(padx=25, pady=25)
btn.place(x=387,y=630)

root.title("Home | Python Game Zone | Welcome")
#root.geometry("1280x720")
root.attributes('-fullscreen' , True)
#root.resizable(False,False)
root.configure(bg="light yellow")
mainloop()
```

MENU PAGE



MENU PAGE CODE

```

from tkinter import *
import tkinter

window = tkinter.Tk()

def snake():
    #window.destroy()
    import snake

```

```
def flappybird():
```

```
    #window.destroy()
```

```
    import flappybird
```

```
def PacMan():
```

```
    #window.destroy()
```

```
    import PacMan
```

```
def RockPaperScissor():
```

```
    #window.destroy()
```

```
    import RockPaperScissor
```

```
def TicTacToe():
```

```
    #window.destroy()
```

```
    import TicTacToe
```

```
def BubbleSortVisual():
```

```
    #window.destroy()
```

```
    import BubbleSortVisual
```

```
def pong():
```

```
    #window.destroy()
```

```
    import pong
```

```
def Tiles():
```

```
    #window.destroy()
```

```
    import Tiles
```

```
def Game1():
```

```
    myLabel = Label(window,text=""Snake is a video game genre where the  
player manoeuvres a growing line that becomes a primary obstacle to itself.
```

```
    The concept originated in the 1976 two-player arcade game Blockade  
from Gremlin Industries, and the ease of implementation has led to  
hundreds of versions for many platforms.""",font="Cambria 12  
bold",bg="white",fg="black")
```

```
    myLabel.pack()
```

```
    myLabel.place(x=105,y=644)
```

```
def Game2():
```

```
    myLabel = Label(window,text=""Flappy Bird is a mobile game developed  
by Vietnamese video game artist and programmer Dong Nguyen, under his  
game development company .Gears. The game is a side-scroller
```

```
    where the player controls a bird, attempting to fly between columns of  
green pipes without hitting them.""",font="Cambria 12  
bold",bg="white",fg="black")
```

```
    myLabel.pack(padx=20, pady=20)
```

```
    myLabel.place(x=105,y=644)
```

```
def Game3():
```

```
    myLabel = Label(window,text=""    Pac-Man is a Japanese video game
franchise created by Toru Iwatani but published,developed
```

```
    and owned by Bandai Namco Entertainment.Entries have been
developed by a wide array of other video game companies, including
Midway Games, Atari and Mass Media, Inc. "",font="Cambria 12
bold",bg="white",fg="black")
```

```
    myLabel.pack(padx=20, pady=20)
```

```
    myLabel.place(x=105,y=644)
```

```
def Game4():
```

```
    myLabel = Label(window,text=""    Tic-tac-toe, noughts and crosses, or Xs
and Os is a paper-and-pencil game for two players who take turns marking
the spaces in a three-by-three grid with X or O. The player who
```

```
    succeeds in placing three of their marks in a horizontal, vertical, or
diagonal row is the winner. "",font="Cambria 12
bold",bg="white",fg="black")
```

```
    myLabel.pack(padx=20, pady=20)
```

```
    myLabel.place(x=105,y=644)
```

```
def Game5():
```

```
    myLabel = Label(window,text=""Pong is a two-dimensional sports game
that simulates table tennis. The player controls an in-game paddle by
moving it vertically across the left or right side of the screen. They can
```

compete against another player controlling a second paddle on the opposing side. Players use the paddles to hit a ball back and forth."",font="Cambria 12 bold",bg="white",fg="black")

```
myLabel.pack(padx=20, pady=20)
```

```
myLabel.place(x=105,y=644)
```

```
def Game6():
```

myLabel = Label(window,text=""Rock Paper Scissors is a hand game originating from China, usually played between two people,in which each player simultaneously forms one of three shapes with an outstretched

hand. These shapes are "rock", "paper", and "scissors". "",font="Cambria 12 bold",bg="white",fg="black")

```
myLabel.pack(padx=20, pady=20)
```

```
myLabel.place(x=105,y=644)
```

```
def Game7():
```

myLabel = Label(window,text=""A sliding puzzle, sliding block puzzle,or sliding tile puzzle is a combination puzzle that challenges a player to slide (frequently flat) pieces along certain routes (usually on a board)

to establish a certain end-configuration. The pieces to be moved may consist of simple shapes. "",font="Cambria 12 bold",bg="white",fg="black")

```
myLabel.pack(padx=20, pady=20)
```

```
myLabel.place(x=105,y=644)
```



```
def Game8():
```

```
    myLabel = Label(window,text=""    Bubble sort, sometimes referred to as
sinking sort, is a simple sorting algorithm that repeatedly steps through the
list, compares adjacent elements and swaps them if they are in
```

```
the wrong order.The pass through the list is repeated until the list is
sorted. """,font="Cambria 12 bold",bg="white",fg="black")
```

```
    myLabel.pack(padx=20, pady=20)
```

```
    myLabel.place(x=105,y=644)
```

```
window.title("Main Menu")
```

```
canvas999= Canvas(width=1550, height=715 , bg ='black' )
```

```
canvas999.pack()
```

```
#canva9992.place(x=75, y=165)
```

```
photo999= PhotoImage(file="C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\MenuPage.png")
```

```
canvas999.create_image(0,0, image=photo999, anchor=NW)
```

```
#lbl = Label(window, text=" * * * LIST OF GAMES * * * " , bg="BLUE",
fg="White",font= ('Cambria 19 bold'))
```

```
#lbl.pack()
```

```
canvas1 = Canvas(width=100, height=100 , bg ='white' )
```

```
canvas1.pack()

canvas1.place(x=310,y=170)

photo1 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\HungrySnake.png')

canvas1.create_image(0,0, image=photo1, anchor=NW)


LL1 = Label(window, text=" Hungry Snake ",font="Calibri 15
bold",fg="red3",bg="white")

LL1.pack()

LL1.place(x=298,y=142)


btn1 = Button(window, text="P L A Y" , command=snake)

btn1.pack(padx=20, pady=20)

btn1.place(x=265, y=280)

btn1.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'blue3' , fg = 'white' )


btn11 = Button(window, text="A B O U T" , command=Game1)

btn11.pack(padx=20, pady=20)

btn11.place(x=365, y=280)

btn11.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'green4' , fg = 'white' )


canvas2 = Canvas(width=100, height=100 , bg ='white' )
```

```
canvas2.pack()

canvas2.place(x=577, y=170)

photo2 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\FlappyBird.png')

canvas2.create_image(0,0, image=photo2, anchor=NW)


LL2 = Label(window, text=" Flappy Bird ",font="Calibri 15
bold",fg="red3",bg="white")

LL2.pack()

LL2.place(x=573,y=152)


btn2 = Button(window, text="P L A Y" , command=flappybird )

btn2.pack(padx=20, pady=20)

btn2.place(x=534, y=280)

btn2.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'blue3' , fg = 'white')


btn22 = Button(window, text="A B O U T" , command=Game2)

btn22.pack(padx=20, pady=20)

btn22.place(x=634, y=280)

btn22.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'green4' , fg = 'white')


canvas3 = Canvas(width=100, height=100 , bg ='white' )
```

```
canvas3.pack()

canvas3.place(x=875, y=170)

photo3 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\PacMan.png')

canvas3.create_image(0,0, image=photo3, anchor=NW)


LL3 = Label(window, text=" Pac - Man ",font="Calibri 15
bold",fg="red3",bg="white")

LL3.pack()

LL3.place(x=876,y=142)


btn3 = Button(window, text="P L A Y" , command=PacMan)

btn3.pack(padx=20, pady=20)

btn3.place(x=830, y=280)

btn3.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'BLUE3' , fg = 'white')


btn33 = Button(window, text="A B O U T", command=Game3)

btn33.pack(padx=20, pady=20)

btn33.place(x=930, y=280)

btn33.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'green4' , fg = 'white')


canvas7 = Canvas(width=100, height=100 , bg ='white' )
```

```
canvas7.pack()

canvas7.place(x=1135, y=170)

photo7 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\SlidingPuzzle.png')

canvas7.create_image(0,0, image=photo7, anchor=NW)


LL7 = Label(window, text=" Sliding Puzzle ",font="Calibri 15
bold",fg="red3",bg="white")

LL7.pack()

LL7.place(x=1120,y=142)


btn7 = Button(window, text="P L A Y" , command=Tiles )

btn7.pack(padx=20, pady=20)

btn7.place(x=1090, y=280)

btn7.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'BLUE3' , fg = 'white')


btn77 = Button(window, text="A B O U T", command=Game7)

btn77.pack(padx=20, pady=20)

btn77.place(x=1190, y=280)

btn77.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'green4' , fg = 'white')


#EXIT
```

```

btn99 = Button(window,height=2 , width =12,text="E X I T", font = 'Aerial
12          bold'          ,bg          =          'aquamarine2',fg          =
'black',command=lambda:window.destroy(), border = '10')

btn99.pack()

btn99.place(x=710,y=730)


#clear


def clear():

    myLabel = Label(window,text="!!! Clear by DS !!! Clear by DS !!! Clear by
DS !!! Clear by DS !!! Clear by DS !!! Clear by DS !!! Clear by DS
!!!Clear by DS !!! Clear by DS !!! Clear by DS !!! @DEBARATHI SARDAR

    !!! Clear by DS !!! Clear by DS !!! Clear by DS !!! Clear by DS !!!Clear by DS
!!!Clear by DS !!!Clear by DS !!! .""",font="Cambria 12
bold",bg="#41c5a7",fg="#41c5a7")

    myLabel.pack(padx=20, pady=20)

    myLabel.place(x=105,y=644)


btn999 = Button(window,height=1 , width =8,text="C L E A R", font =
'Aerial 12 bold' ,bg = 'black',fg = 'gold', command=clear)

btn999.pack()

btn999.place(x=732,y=607)


canvas4 = Canvas(width=100, height=100 , bg ='white' )

```

```
canvas4.pack()

canvas4.place(x=310, y=389)

photo4 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\TicTacToe.png')

canvas4.create_image(0,0, image=photo4, anchor=NW)


LL4 = Label(window, text="  Tic-Tac-Toe      ",font="Calibri 15
bold",fg="red3",bg="white")

LL4.pack()

LL4.place(x=305,y=360)


btn4 = Button(window, text="P L A Y" , command=TicTacToe )

btn4.pack(padx=20, pady=20)

btn4.place(x=265, y=503)

btn4.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'BLUE3', fg = 'white')


btn44 = Button(window, text="A B O U T", command=Game4)

btn44.pack(padx=20, pady=20)

btn44.place(x=365, y=503)

btn44.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'green4' , fg = 'white')


canvas5 = Canvas(width=100, height=100 , bg ='white' )
```

```

canvas5.pack()

canvas5.place(x=577, y=389)

photo5 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\Pong.png')

canvas5.create_image(0,0, image=photo5, anchor=NW)


LL5 = Label(window, text=" Pong ", font="Calibri 15
bold", fg="red3", bg="white")

LL5.pack()

LL5.place(x=595, y=360)


btn5 = Button(window, text="P L A Y", command=pong )

btn5.pack(padx=20, pady=20)

btn5.place(x=536, y=503)

btn5.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'BLUE3', fg = 'white')


btn55 = Button(window, text="A B O U T", command=Game5)

btn55.pack(padx=20, pady=20)

btn55.place(x=634, y=503)

btn55.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'green4' , fg = 'white')


canvas6 = Canvas(width=100, height=100 , bg ='white' )

```



```
canvas6.pack()

canvas6.place(x=875, y=389)

photo6 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\RockPaperScissors.png')

canvas6.create_image(0,0, image=photo6, anchor=NW)


LL6 = Label(window, text=" Rock Paper Scissors ",font="Calibri 15
bold",fg="red3",bg="white")

LL6.pack()

LL6.place(x=835,y=360)


btn6 = Button(window, text="P L A Y" , command=RockPaperScissor )

btn6.pack(padx=20, pady=20)

btn6.place(x=830, y=503)

btn6.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'blue3', fg = 'white')


btn66 = Button(window, text="A B O U T", command=Game6)

btn66.pack(padx=20, pady=20)

btn66.place(x=930, y=503)

btn66.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'green4' , fg = 'white')


canvas8 = Canvas(width=100, height=100 , bg ='white' )
```

```
canvas8.pack()

canvas8.place(x=1135, y=389)

photo8 = PhotoImage(file='C:\\Users\\Acer\\Desktop\\PROJ - CS881
Project - III\\BubbleSortVisual.png')

canvas8.create_image(0,0, image=photo8, anchor=NW)


LL8 = Label(window, text=" Bubble Sort Visual ",font="Calibri 15
bold",fg="red3",bg="white")

LL8.pack()

LL8.place(x=1102,y=360)


btn8 = Button(window, text="P L A Y" , command=BubbleSortVisual)

btn8.pack(padx=20, pady=20)

btn8.place(x=1090, y=503)

btn8.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'BLUE3' , fg = 'white')


btn88 = Button(window, text="A B O U T", command=Game8)

btn88.pack(padx=20, pady=20)

btn88.place(x=1190, y=503)

btn88.configure(height=2 , width =8, font = 'Aerial 12 bold italic' , bg =
'green4' , fg = 'white')


#Developed By
```

```
lbl = Label(window, text="Developed By",font="Calibri 13 bold italic",bg="light yellow",fg="black")

lbl.pack()

lbl.place(x=731,y=807)

lbl = Label(window, text=" DEBARATHI SARDAR | BITTU DEY | SUJIT MAITY ",font="Calibri 13 bold",bg="light yellow",fg="black")

lbl.pack()

lbl.place(x=604,y=830)


window.configure(bg='light yellow')


#window.resizable(False , False)


window.attributes('-fullscreen' , True)


#window.geometry("635x610")


window.mainloop()
```

REFERENCE:

[1] All about Operations of Python and its Functions and Modules from W3 Schools (www.w3schools.com/python/)

[2] Tkinter – Python interface to Tcl/Tk (Notes, Commands, Modules, Uses) – (<https://docs.python.org/3/library/tkinter.html>) in python.org

[3] Python based games using Turtle / Free games modules from Geeks for Geeks (<https://www.geeksforgeeks.org>)

[4] Python GUI's (Graphical User Interface) with Tkinter from (www.codemy.com) And YouTube Videos of Codemy (<https://youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV>)

[5] Python Based Game using pygame module (Game Development Library / Pygame Tutorial) from JavatPoint (<https://www.javatpoint.com/pygame>) and Edureka (<https://www.edureka.co/blog/pygame-tutorial>)

[6] Python Tkinter Tutorial | GUI programming Using Tkinter from Tutorial Point (https://www.tutorialspoint.com/python/python_gui_programming.htm)