PROJECT REPORT

On

# “FLAPPY BIRD WITH DYNAMIC OPTION”


***Submitted By***

Pranav P. Bawankule

***Guided By:-***

Mr. Ratnesh K. Choudhary




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**S. B. JAIN INSTITUTE OF TECHNOLOGY**

**MANAGEMENT AND RESEARCH, NAGPUR.**

(An Autonomous Institute, Affiliated to RTMNU, Nagpur)


**2021-2022**

# S.B. JAIN INSTITUTE OF TECHNOLOGY MANAGEMENT AND RESEARCH, NAGPUR

## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SESSION 2021-2022

# CERTIFICATE

This is to certify that the Project titled **"**Flappy Bird With Dynamic Option**"** is a bonafide work of Pranav Bawankule carried out for the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering in **Computer Science & Engineering.**

**Mr. Ratnesh K. Choudhary**
Assistant Professor

**Mr. Animesh Tayal**
Head of Department

# INDEX

# LIST OF FIGURE

# CHAPTER 1

# INTRODUCTION

My project is a new version of Flappy Bird Game. It is a pygame program written in python. Users can play this game either on PiTFT screen or play it in their computer. It contains two play modes, single player and dual player modes.. . In single player mode, each player have 3 lives and their NetIDs and final scores will be written to a score board, which keeps track of top 10 scores. The dual player mode gives users the opportunity to compete with their friends. We also implemented two background settings, daytime and night modes. The users could also choose either one background mode before playing the game. One difficulty for users is that the horizontal shifting speed will increasing as time goes on. We also implemented a hidden trick in the single player mode. When user achieves 10 scores, the flappy bird will evolve into flappy Joe.

## CHAPTER 2

# METHODOLOGY

Here you have to write Methodology of your project.

**Step 1:** In this first step, we have to import libraries.

**Step 2:** After declaring game variables and importing libraries we have to initialize the Pygame

**Step 3:** Initialize the position of the bird and starting the game loop

**Step 4:** Create a function that generates a new pipe of random height

**Step 5:** Now we create a **GameOver()** function which represents whether the bird has hit the pipes or fall into the sea.

**Step 6:** Now we will be creating our main function (**flappygame()**) that will do the following things

# CHAPTER 3

# TOOLS/PLATFORMS

## SOFTWARE REQUIREMENT

**1.IDE**: VS Code

**2. LIBRARIES:** Pygames,Sys,Random

**3. OPERATING SYSTEM:** Windows 7

## VS 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

## PYgames:

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.

## Sys:

sys is a filename extension used in MS-DOS applications and Microsoft Windows operating systems. They are system files that contain device drivers or hardware configurations for the system. Most DOS .sys files are real mode device drivers.

## Random:

Random.org is a website that produces random numbers based on atmospheric noise. In addition to generating random numbers in a specified range and subject to a specified probability distribution.

## Windows 7:

Windows 7 is a major release of the Windows NT operating system developed by Microsoft. It was released to manufacturing on July 22, 2009, and became generally available on October 22, 2009. It is the successor to Windows Vista, released nearly three years earlier.

# CHAPTER 4

# DESIGN & IMPLEMENTATION

## 4.1 ALGORITHM

a) Algorithm to display pictures correctly on monitor At first, how to transfer images to correct coordinate and display on the monitor will be our first challenge. We need an efficient way to convert our image into coordinate in frame buffer.

b) Algorithm for image moving In our game, the image moving is the most important function we need to implement. Besides the background horizontal moving, the vertical jumping combining keyboard input is our second challenge.

c) Game over condition setting In the process of the game, how to detect if the object satisfies the game over condition and the condition setting is our third problem.

d) Adding and fixing some function We will revise our project and improving some details, for example: adding more interesting function and more complicating contents in our game.

## 4.2 FLOWCHART

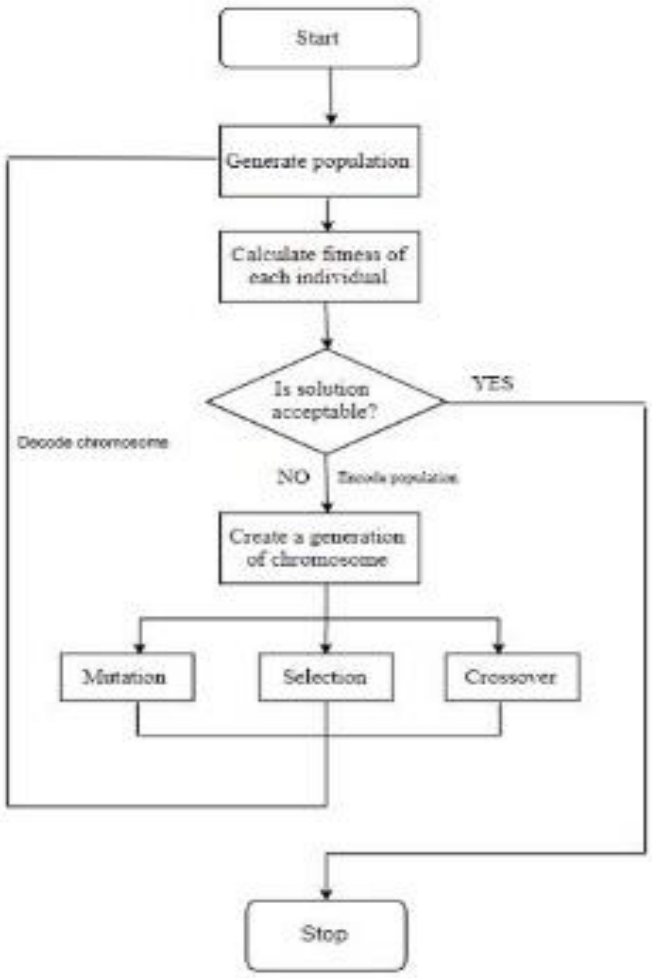


**Fig.4.2.1** Flowchart of Flappy Bird game with dynamic option

## 4.3 SOURCE CODE

```
#! /usr/bin/env python3

"""Flappy Bird, implemented using Pygame."""

import math
import os
from random import randint
from collections import deque

import pygame
from pygame.locals import *


FPS = 60
ANIMATION_SPEED = 0.18  # pixels per millisecond
WIN_WIDTH = 284 * 2     # BG image size: 284x512 px; tiled twice
WIN_HEIGHT = 512


class Bird(pygame.sprite.Sprite):
    """Represents the bird controlled by the player.

    The bird is the 'hero' of this game.  The player can make it climb
    (ascend quickly), otherwise it sinks (descends more slowly).  It must
    pass through the space in between pipes (for every pipe passed, one
    point is scored); if it crashes into a pipe, the game ends.
```

**Attributes:**

**x: The bird's X coordinate.**

**y: The bird's Y coordinate.**

**msec_to_climb: The number of milliseconds left to climb, where a**

**complete climb lasts Bird.CLIMB_DURATION milliseconds.**

**Constants:**

**WIDTH: The width, in pixels, of the bird's image.**

**HEIGHT: The height, in pixels, of the bird's image.**

**SINK_SPEED: With which speed, in pixels per millisecond, the bird**

**descends in one second while not climbing.**

**CLIMB_SPEED: With which speed, in pixels per millisecond, the bird**

**ascends in one second while climbing, on average. See also the**

**Bird.update docstring.**

**CLIMB_DURATION: The number of milliseconds it takes the bird to**

**execute a complete climb.**

**"""**

**WIDTH = HEIGHT = 32**

**SINK_SPEED = 0.18**

**CLIMB_SPEED = 0.3**

**CLIMB_DURATION = 333.3**

**def __init__(self, x, y, msec_to_climb, images):**

**"""Initialise a new Bird instance.**

**Arguments:**

**x: The bird's initial X coordinate.**

```
    y: The bird's initial Y coordinate.
    msec_to_climb: The number of milliseconds left to climb, where a
        complete climb lasts Bird.CLIMB_DURATION milliseconds.  Use
        this if you want the bird to make a (small?) climb at the
        very beginning of the game.
    images: A tuple containing the images used by this bird.  It
        must contain the following images, in the following order:
            0. image of the bird with its wing pointing upward
            1. image of the bird with its wing pointing downward
    """
    super(Bird, self).__init__()
    self.x, self.y = x, y
    self.msec_to_climb = msec_to_climb
    self._img_wingup, self._img_wingdown = images
    self._mask_wingup = pygame.mask.from_surface(self._img_wingup)
    self._mask_wingdown = pygame.mask.from_surface(self._img_wingdown)

def update(self, delta_frames=1):
    """Update the bird's position.

    This function uses the cosine function to achieve a smooth climb:
    In the first and last few frames, the bird climbs very little, in the
    middle of the climb, it climbs a lot.
    One complete climb lasts CLIMB_DURATION milliseconds, during which
    the bird ascends with an average speed of CLIMB_SPEED px/ms.
    This Bird's msec_to_climb attribute will automatically be
    decreased accordingly if it was > 0 when this method was called.
```

```
        Arguments:
        delta_frames: The number of frames elapsed since this method was
            last called.
        """
        if self.msec_to_climb > 0:
            frac_climb_done = 1 - self.msec_to_climb/Bird.CLIMB_DURATION
            self.y -= (Bird.CLIMB_SPEED * frames_to_msec(delta_frames) *
                       (1 - math.cos(frac_climb_done * math.pi)))
            self.msec_to_climb -= frames_to_msec(delta_frames)
        else:
            self.y += Bird.SINK_SPEED * frames_to_msec(delta_frames)

    @property
    def image(self):
        """Get a Surface containing this bird's image.

        This will decide whether to return an image where the bird's
        visible wing is pointing upward or where it is pointing downward
        based on pygame.time.get_ticks().  This will animate the flapping
        bird, even though pygame doesn't support animated GIFs.
        """
        if pygame.time.get_ticks() % 500 >= 250:
            return self._img_wingup
        else:
            return self._img_wingdown

    @property
    def mask(self):
```

```
        """Get a bitmask for use in collision detection.

        The bitmask excludes all pixels in self.image with a
        transparency greater than 127."""
        if pygame.time.get_ticks() % 500 >= 250:
            return self._mask_wingup
        else:
            return self._mask_wingdown

    @property
    def rect(self):
        """Get the bird's position, width, and height, as a pygame.Rect."""
        return Rect(self.x, self.y, Bird.WIDTH, Bird.HEIGHT)


class PipePair(pygame.sprite.Sprite):
    """Represents an obstacle.

    A PipePair has a top and a bottom pipe, and only between them can
    the bird pass -- if it collides with either part, the game is over.

    Attributes:
    x: The PipePair's X position.  This is a float, to make movement
        smoother.  Note that there is no y attribute, as it will only
        ever be 0.
    image: A pygame.Surface which can be blitted to the display surface
        to display the PipePair.
    mask: A bitmask which excludes all pixels in self.image with a
```

**transparency greater than 127. This can be used for collision detection.**

**top_pieces: The number of pieces, including the end piece, in the top pipe.**

**bottom_pieces: The number of pieces, including the end piece, in the bottom pipe.**

**Constants:**

**WIDTH: The width, in pixels, of a pipe piece. Because a pipe is only one piece wide, this is also the width of a PipePair's image.**

**PIECE_HEIGHT: The height, in pixels, of a pipe piece.**

**ADD_INTERVAL: The interval, in milliseconds, in between adding new pipes.**

**"""**

```
WIDTH = 80
PIECE_HEIGHT = 32
ADD_INTERVAL = 3000

def __init__(self, pipe_end_img, pipe_body_img):
    """Initialises a new random PipePair.

    The new PipePair will automatically be assigned an x attribute of
    float(WIN_WIDTH - 1).

    Arguments:
    pipe_end_img: The image to use to represent a pipe's end piece.
```

```
pipe_body_img: The image to use to represent one horizontal slice
    of a pipe's body.
"""
self.x = float(WIN_WIDTH - 1)
self.score_counted = False

self.image = pygame.Surface((PipePair.WIDTH, WIN_HEIGHT), SRCALPHA)
self.image.convert()   # speeds up blitting
self.image.fill((0, 0, 0, 0))
total_pipe_body_pieces = int(
    (WIN_HEIGHT -                 # fill window from top to bottom
     3 * Bird.HEIGHT -            # make room for bird to fit through
     3 * PipePair.PIECE_HEIGHT) /  # 2 end pieces + 1 body piece
    PipePair.PIECE_HEIGHT         # to get number of pipe pieces
)
self.bottom_pieces = randint(1, total_pipe_body_pieces)
self.top_pieces = total_pipe_body_pieces - self.bottom_pieces

# bottom pipe
for i in range(1, self.bottom_pieces + 1):
    piece_pos = (0, WIN_HEIGHT - i*PipePair.PIECE_HEIGHT)
    self.image.blit(pipe_body_img, piece_pos)
bottom_pipe_end_y = WIN_HEIGHT - self.bottom_height_px
bottom_end_piece_pos = (0, bottom_pipe_end_y - PipePair.PIECE_HEIGHT)
self.image.blit(pipe_end_img, bottom_end_piece_pos)

# top pipe
for i in range(self.top_pieces):
```

```
            self.image.blit(pipe_body_img, (0, i * PipePair.PIECE_HEIGHT))
        top_pipe_end_y = self.top_height_px
        self.image.blit(pipe_end_img, (0, top_pipe_end_y))

        # compensate for added end pieces
        self.top_pieces += 1
        self.bottom_pieces += 1

        # for collision detection
        self.mask = pygame.mask.from_surface(self.image)

    @property
    def top_height_px(self):
        """Get the top pipe's height, in pixels."""
        return self.top_pieces * PipePair.PIECE_HEIGHT

    @property
    def bottom_height_px(self):
        """Get the bottom pipe's height, in pixels."""
        return self.bottom_pieces * PipePair.PIECE_HEIGHT

    @property
    def visible(self):
        """Get whether this PipePair on screen, visible to the player."""
        return -PipePair.WIDTH < self.x < WIN_WIDTH

    @property
    def rect(self):
```

```
        """Get the Rect which contains this PipePair."""
        return Rect(self.x, 0, PipePair.WIDTH, PipePair.PIECE_HEIGHT)

    def update(self, delta_frames=1):
        """Update the PipePair's position.

        Arguments:
        delta_frames: The number of frames elapsed since this method was
            last called.
        """
        self.x -= ANIMATION_SPEED * frames_to_msec(delta_frames)

    def collides_with(self, bird):
        """Get whether the bird collides with a pipe in this PipePair.

        Arguments:
        bird: The Bird which should be tested for collision with this
            PipePair.
        """
        return pygame.sprite.collide_mask(self, bird)


def load_images():
    """Load all images required by the game and return a dict of them.

    The returned dict has the following keys:
    background: The game's background image.
    bird-wingup: An image of the bird with its wing pointing upward.
```

```
    Use this and bird-wingdown to create a flapping bird.
bird-wingdown: An image of the bird with its wing pointing downward.
    Use this and bird-wingup to create a flapping bird.
pipe-end: An image of a pipe's end piece (the slightly wider bit).
    Use this and pipe-body to make pipes.
pipe-body: An image of a slice of a pipe's body.  Use this and
    pipe-body to make pipes.
"""

def load_image(img_file_name):
    """Return the loaded pygame image with the specified file name.

    This function looks for images in the game's images folder
    (./images/).  All images are converted before being returned to
    speed up blitting.

    Arguments:
    img_file_name: The file name (including its extension, e.g.
        '.png') of the required image, without a file path.
    """
    file_name = os.path.join('.', 'images', img_file_name)
    img = pygame.image.load(file_name)
    img.convert()
    return img

return {'background': load_image('background.png'),
        'pipe-end': load_image('pipe_end.png'),
        'pipe-body': load_image('pipe_body.png'),
```

```
        # images for animating the flapping bird -- animated GIFs are
        # not supported in pygame
        'bird-wingup': load_image('bird_wing_up.png'),
        'bird-wingdown': load_image('bird_wing_down.png')}


def frames_to_msec(frames, fps=FPS):
    """Convert frames to milliseconds at the specified framerate.

    Arguments:
    frames: How many frames to convert to milliseconds.
    fps: The framerate to use for conversion.  Default: FPS.
    """
    return 1000.0 * frames / fps


def msec_to_frames(milliseconds, fps=FPS):
    """Convert milliseconds to frames at the specified framerate.

    Arguments:
    milliseconds: How many milliseconds to convert to frames.
    fps: The framerate to use for conversion.  Default: FPS.
    """
    return fps * milliseconds / 1000.0


def main():
    """The application's entry point.
```

```
If someone executes this module (instead of importing it, for
example), this function is called.
"""

pygame.init()

display_surface = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT))
pygame.display.set_caption('Pygame Flappy Bird')

clock = pygame.time.Clock()
score_font = pygame.font.SysFont(None, 32, bold=True)  # default font
images = load_images()

# the bird stays in the same x position, so bird.x is a constant
# center bird on screen
bird = Bird(50, int(WIN_HEIGHT/2 - Bird.HEIGHT/2), 2,
            (images['bird-wingup'], images['bird-wingdown']))

pipes = deque()

frame_clock = 0  # this counter is only incremented if the game isn't paused
score = 0
done = paused = False
while not done:
    clock.tick(FPS)

    # Handle this 'manually'.  If we used pygame.time.set_timer(),
```

```
# pipe addition would be messed up when paused.
if not (paused or frame_clock % msec_to_frames(PipePair.ADD_INTERVAL)):
    pp = PipePair(images['pipe-end'], images['pipe-body'])
    pipes.append(pp)

for e in pygame.event.get():
    if e.type == QUIT or (e.type == KEYUP and e.key == K_ESCAPE):
        done = True
        break
    elif e.type == KEYUP and e.key in (K_PAUSE, K_p):
        paused = not paused
    elif e.type == MOUSEBUTTONUP or (e.type == KEYUP and
            e.key in (K_UP, K_RETURN, K_SPACE)):
        bird.msec_to_climb = Bird.CLIMB_DURATION

if paused:
    continue  # don't draw anything

# check for collisions
pipe_collision = any(p.collides_with(bird) for p in pipes)
if pipe_collision or 0 >= bird.y or bird.y >= WIN_HEIGHT - Bird.HEIGHT:
    done = True

for x in (0, WIN_WIDTH / 2):
    display_surface.blit(images['background'], (x, 0))

while pipes and not pipes[0].visible:
    pipes.popleft()
```

```
        for p in pipes:
            p.update()
            display_surface.blit(p.image, p.rect)

        bird.update()
        display_surface.blit(bird.image, bird.rect)

        # update and display score
        for p in pipes:
            if p.x + PipePair.WIDTH < bird.x and not p.score_counted:
                score += 1
                p.score_counted = True

        score_surface = score_font.render(str(score), True, (255, 255, 255))
        score_x = WIN_WIDTH/2 - score_surface.get_width()/2
        display_surface.blit(score_surface, (score_x, PipePair.PIECE_HEIGHT))

        pygame.display.flip()
        frame_clock += 1
    print('Game over! Score: %i' % score)
    pygame.quit()


if __name__ == '__main__':
    # If this module had been imported, __name__ would be 'flappybird'.
    # It was executed (e.g. by double-clicking the file), so call main.
    main()
```

# CHAPTER 5

# RESULT & DISCUSSION

## 5.1 OUTPUT

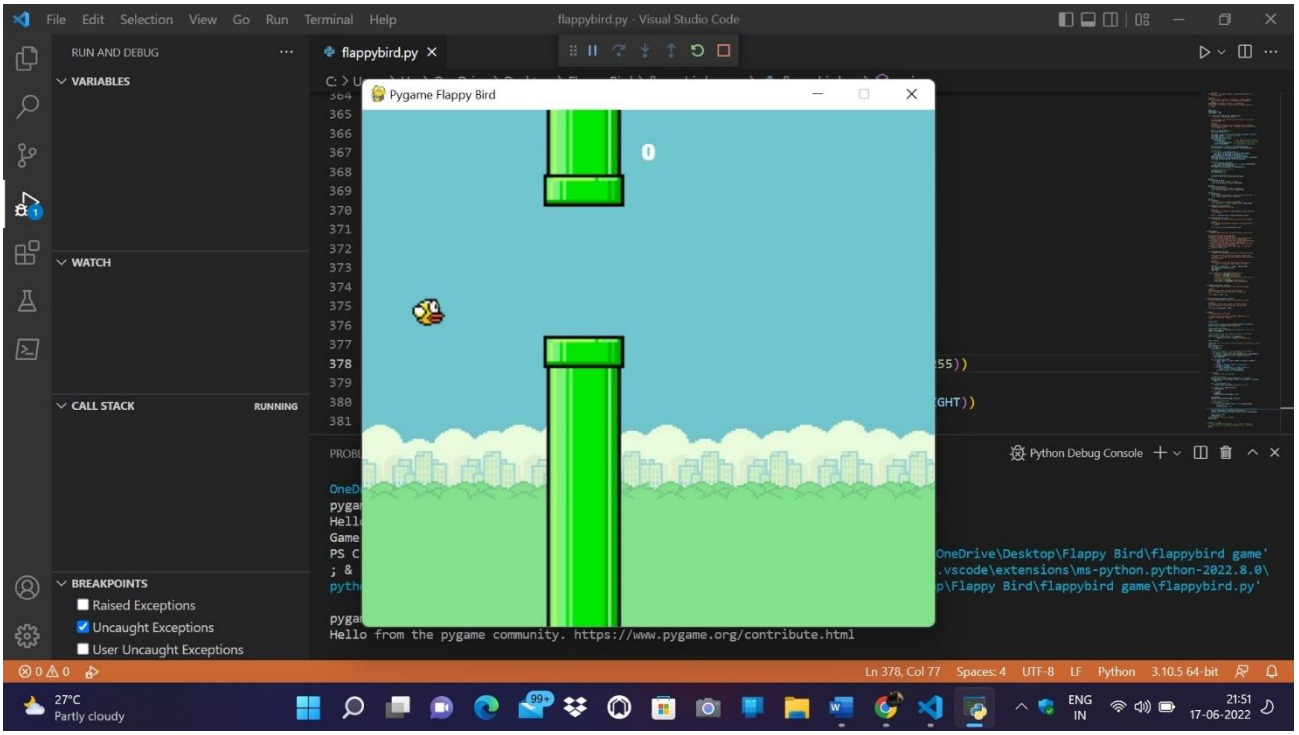


**Fig. 5.1.1** This image show that the how Flappy Bird game start

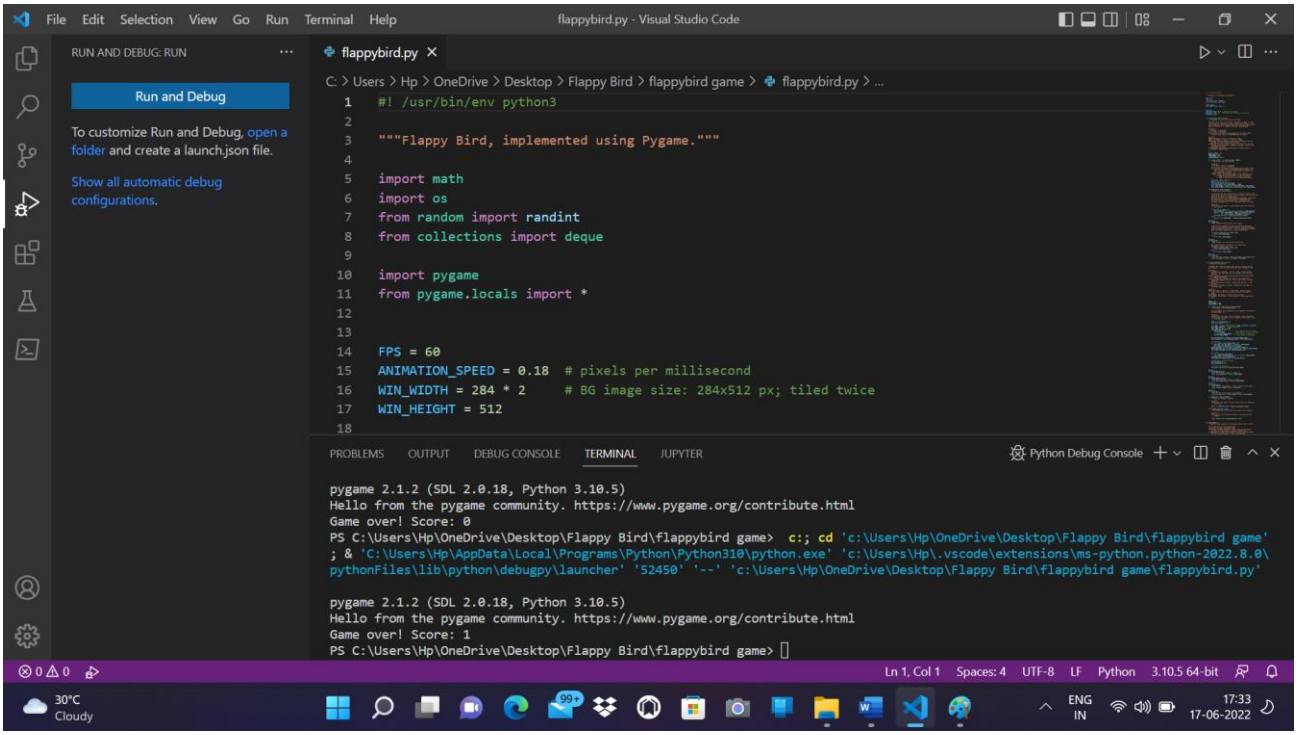


**Fig. 5.1.2** This image show that the score of the Flappy Bird game

## 5.2 DISCUSSION

In this game Flappy bird was originally released as a mobile game where you tap the screen to make the bird fly. If the bird hits the pipes or the edges of the screen, the game ends and the player needs to restart.

## 5.3 APPLICATION

With the help of in this program we can play this game in Easy way We all are familiar with this game. In this game, the main objective of the player is to gain the maximum points by defending the bird from hurdles. Here, we will build our own Flappy Bird game using Python

# CHAPTER 6

# CONCLUSION

My final project was planned, developed and demonstrated as expected. We designed a new version of Flappy Bird Game written in Python, which could be played either on PiTFT screen or personal computer. Firstly, a user-friendly interface was implemented. Secondly, single player mode was realized. In single player mode, each player have 3 lives and their NetIDs and final scores will be writen to a score board, which keeps track of top 10 scores. Then the code of dual player mode was written. It gives users the opportunity to compete with their friends, which will bring a lot of fun. We also implemented two background settings, daytime and night modes. The users could also choose either one background mode before playing the game. One difficulty for users is that the horizontal shifting speed will increasing as time goes on. We also implemented a hidden trick in the single player mode. When user achieves 10 scores, the flappy bird will evolve into flappy Joe.

# REFERENCE

http://www.usatoday.com/story/tech/gaming/2014/03/11/flappy-bird-creator/6302287//

http://frivetenskapligpublicering.blogspot.se/

http://worldwithoutexcuses.blogspot.se/

http://ingaursakter.blogspot.se/

Pygame tutorials

Pygame documents