NEW HORIZON COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade. Accredited by NBA

nirf TOP ENGINEERING COLLEGE OF INDIA as per NATIONAL INSTITUTIONAL RANKING FRAMEWORK, 2018
Ministry of Human Resource Development, Government of India

A

MINI PROJECT REPORT

ON

"THE MARIO GAME"

Submitted in the partial fulfilment of the requirements in the 4th semester of

BACHELOR OF ENGINEERING

IN

INFORMATION SCIENCE AND ENGINEERING

BY

JNANA P J - [1NH18IS041]

FOR

COURSE NAME: MINI PROJECT

COURSE CODE :19ISE49

Under the guidance of

Prof. Mrs. A Rafega Beham
Senior Assistant Professor,
Dept. of ISE, NHCE

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

NEW HORIZON COLLEGE OF ENGINEERING

(Autonomous College Permanently Affiliated to VTU, Approved by AICTE, Accredited by NAAC with 'A' Grade & NBA) Ring Road, Bellandur Post, Near Marathahalli, Bengaluru-560103, INDIA.

CERTIFICATE

Certified that the project work entitled 'The Mario Game' carried out by Ms. JNANA P J, USN 1NH18IS041, a bonafide student of IV semester in partial fulfilment for the award of Bachelor of Engineering in Information Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2019-20. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Signature of the Guide	**Signature of the HOD**	**Signature of the Principal**
[Mrs. A.Rafega Beham]	[Dr. R.J Anandhi]	[Dr.Manjunatha]

Examiners:

	Name	**Signature**
1.	……	……
2.	……	……

ABSTRACT

Video games have gained a lot of importance in today's world. Mario game is one such video game. In this game the player (Mario) has to dodge the fireballs coming out from the dragon. In addition to this the user has to keep in mind not to allow Mario touch the top and the bottom surface of the window screen. The top surface contains the cactus wall and the bottom surface contains the fire. As the level increases the area becomes smaller and smaller and the intensity of the game increases. The user can move Mario using UP and DOWN control keys. The main objective of this game is to safeguard Mario from the dragon as the level increases. Other than this playing the game also provides entertainment for the people and also increases the concentration level of the player.

This project is implemented using Python language. Pygame library is used to build this game. A simple and clean GUI (Graphical User Interface) is provided for the better gameplay.

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and corporation of intellectual, elders and friends. A number of personalities, in their own capacities have helped me in carrying out this mini project. I would like to take this opportunity to thank them all.

I thank the management **Dr. Mohan Manghnani**, Chairman, New Horizon Educational Institutions for providing necessary infrastructure and creating good environment.

I also record here the constant encouragement and facilities extended to us by **Dr.Manjunatha**, Principal, New Horizon College of Engineering.

I extend sincere gratitude for constant encouragement and facilities provided to us by **Dr.R.J.Anandhi**, Head of the Department, Department of Information Science and Engineering.

I sincerely acknowledge the encouragement, timely help and guidance to me by

Mrs. A. Rafega Beham, Sr.Assistant Professor, Department of Information Science and Engineering to complete the mini project within stipulated time successfully.

Finally, a note of thanks to the teaching and non-teaching staff of Information Science and Engineering department for their corporation and my friends, who helped me directly or indirectly in the successful completion of the project.

JNANA P J

1NH18IS041

TABLE OF CONTENTS

LIST OF FIGURES

Fig no. **Page no.**

CHAPTER 1

INTRODUCTION

Mario is a fictional character in the Mario video game franchise which is owned by Ninetendo and is created by Japanese video game developer Shigeru Miyamoto. The character has been appeared in over 200 video games. There is over 600 million units being sold worldwide that makes Mario franchise the best-selling video game franchise of all the time. There are series of game developed by the franchise that includes the Mario Kart racing series, Mario tennis, Mario golf and many more. The project Mario game includes the character Mario and his dreadful journey in saving himself against the fireballs dodged by the dragon.

1.1 Objective

The main objective of this game is to safeguard Mario from the dragon. In addition to this the player has to keep in mind not to let Mario touch top and the bottom surface of the window, as the top surface contains the cactus wall and the bottom surface contains fire. One major objective of playing this game is that it makes player learn some skills such as problem-solving, strategy, trust, calculated risk-taking and how to adapt to unforeseen issues. This game will also become a great source of entertainment to many people.

1.2 Methodology to be followed

- The game begins with the pygame window asking the player to press any key to start. If the player presses the ESC key then the game will end and the pygame window gets closed.
- The player has to move Mario using KEY_UP and KEY_DOWN keys in the keyboard.
- The player must try to score maximum points by safeguarding Mario from the fireballs thrown by the dragon. In addition to this the player must not let Mario to touch the top and the bottom surface of the screen as the top surface contains the cactus wall and the bottom surface contains the firewall.
- As the level increase the intensity of the game also increases because the area becomes smaller and smaller. During each level the cactus and the fire walls on either side of the screen gets increased by one thus reducing the area.

- Once Mario gets hit by the fireballs or if the Mario touch the top or bottom surface the screen, the game ends.
- Top score gets displayed on the screen and the player is asked whether to start the game or to quit.
- The player can start a new game to score the maximum points possible.
- If the player presses ESC key, the game ends and the pygame window gets closed.

1.3 System requirements

Software requirements

Operating system - Windows 10

Programming language - Python

Software version - 3.7.4

Text Editor - IDLE script

Hardware requirements

Processor: Intel core i5

RAM: 8GB or 4GB

Hard disk:120 GB

CHAPTER 2

ANALYSIS AND DESIGN

2.1 Flowchart

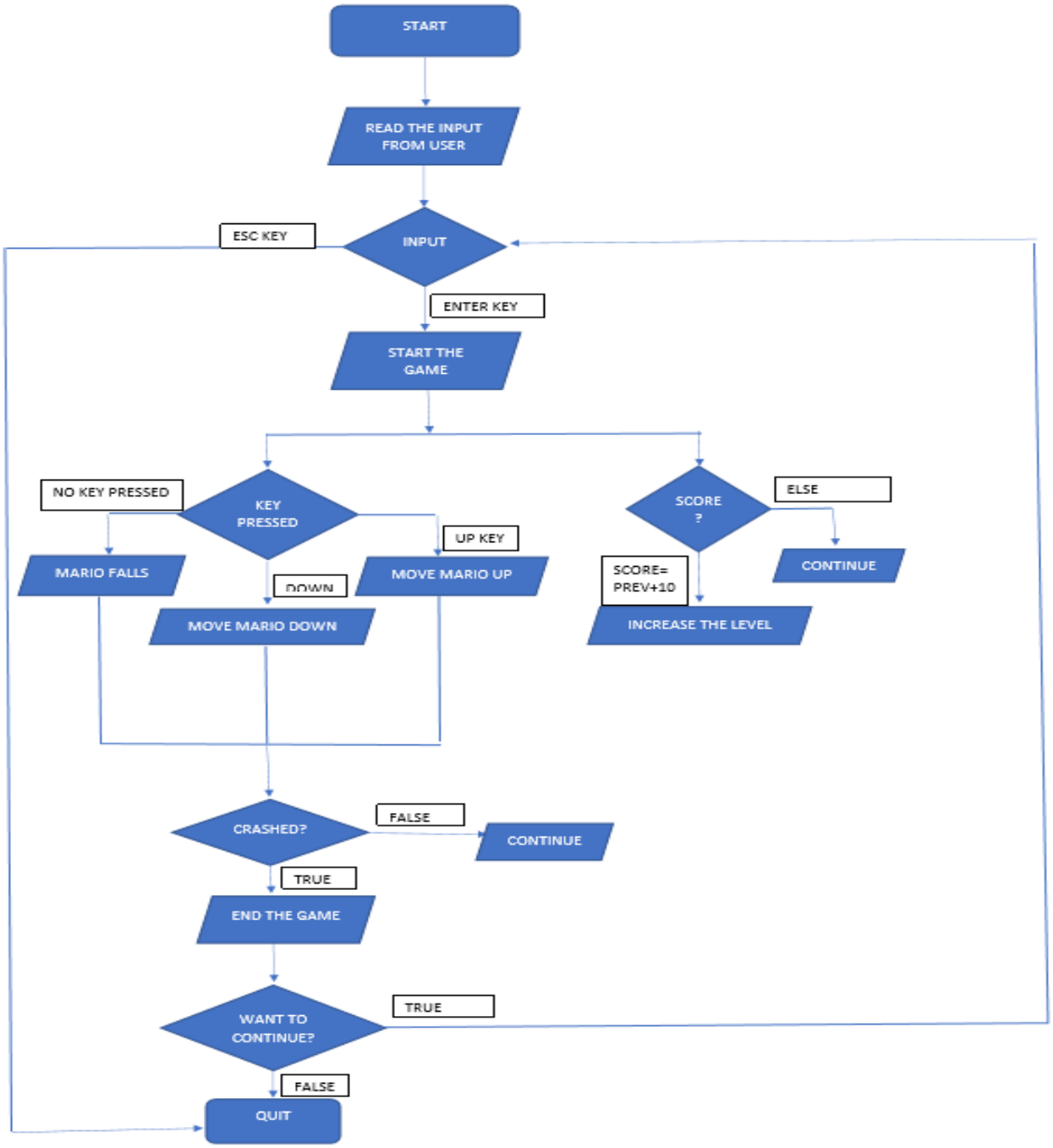

Figure 2.1 Flowchart

2.2 Algorithm

1. Import Python modules such as Pygame, sys.
2. Declare and initialize the variable for the resolution of the window such as window_height, window_width.
3. Initialize the value of the colour that has to be displayed in the background based on RGB values.
4. Create three classes such as Dragon, Flames and Mario.
5. Create the methods inside these classes that are responsible for the activities to be performed by the characters Mario, dragon and fireballs.
6. Create some user defined functions that performs the tasks such as to end the program if the user wishes to do so, to check if the flame has hit Mario or not, to display text on the screen, to increase the game level after certain period.
7. Define a while loop. The game loop or main loop enters the user defined functions and will be constantly running to check whether the events are occurring or not.
8. Create a scoring system to save and display the score every time the player clears a level.
9. Set up the pygame window and call the main function to begin the game.

CHAPTER 3

IMPLEMENTATION

3.1 Modules

A module is a file that contains some definitions and statements of Python. The file name is the module name with the suffix.py appended with it.

There are three ways of importing a module in Python.

- The 'import' statement.

  Syntax: import module1[, module2[, ... moduleN]

  Whenever import statement is used only the modules named will be imported.

- The from import Statement.

  Syntax: from modname import name1[, name2[, ... nameN]

  The from import statement will import only those attributes from the modules that are specified.

- The from import * Statement.

  Syntax: from modname import *

  From this method it imports all the attributes of the modules. The main advantage of this method is that it allows to access the attributes of the module without referring to the specific module name.

3.1.1 Pygame

The Pygame library is an open-source module that is used to build games and other multimedia applications. It is built on top of the highly portable SDL (Simple DirectMedia Layer) development library. Pygame is portable on many platforms and operating systems. Pygame module is imported using 'import' keyword. Pygame modules has additional modules such as pygame.images and pygame.mixer.music and it is not necessary to import these modules using additional import keyword. Once the pygame module is imported in the program the pygame constructor pygame.init() has to be called in order to initialise the modules before calling any other pygame function.

- Setting Up the Display Surface using Pygame.

  The game display surface is set up using pygame.display.set_mode() to initialize the window screen for display .The arguments taken by this function is in the form of tuples. Tuples includes two integer values that represents the width and height of the screen.

- Event handling using Pygame.

  Whenever user presses or releases the key Pygame modules handles the event. pygame.event.get() is used to get the events being performed by the user.

- Setting up background colour using Pygame.

  In this project different colours are given to the background. In Pygame the colours are represented using tuples of three integer values called as RGB values because the three primary colours of light that is red, green and blue are required to form any other colour.

Color	RGB Value		
	R	*G*	*B*
Black	0	0	0
Blue	0	0	255
Red	255	0	0
Yellow	255	255	0
Amber	150	70	0
White	255	255	255

Figure 3.1 Table of RGB values of some colours.

3.1.2 Sys

The sys module provides some functions and variables that are used to change different parts of Python runtime environment. It provides the access for some system specific parameters. Like any other module in Python, even sys module is imported using 'import' keyword.

The following are some of the functions used in sys module.

- **sys.modules**

  The function sys.module is used retrieve the name of the modules that are imported from Python.

- **sys.argv**

  The return type of this function is a list of command line arguments passed to Python script. The name of the script is stored in index 0. The rest of the arguments are stored at the subsequent indices.

- **sys.exit**

  This function is used to exit back to the python console or the command prompt. This is generally used to exit from the program in case there is any exception generated.

- **sys.maxsize**

  Returns the integer, a variable can take.

- **sys.path**

  The function shows the PYTHONPATH set in current system. It is a variable that is used to search path for all Python modules.

- **sys.version**

  The attribute displays a string containing the version number of the current Python interpreter.

3.2 Game Loop

The game loop is the most important part of any game. It includes a continuous while loop which is called as infinite loop that handles events, checks for the inputs, move and draw the objects. In the game loop each iteration is called as a frame. It is important to note that each frame in the game loop should be designed in such a way that it takes the same time in any other machine.

- **While loop**

  The loop mechanism starts with the test condition, if the condition is true then the control enters into the loop otherwise the control moves to the next statement below the while loop.

  The syntax of while loop is given below,

  While <condition>

  {

  Statement 1;

  Statement n;

  }

  Statement after_loop;

The functions performed by the game loop are shown below,

Figure 3.2 Functions of the game loop

The game state updates whenever the mouse gets clicked or the key gets pressed. The game state is a set of values for all the variables in the game. Pygame module has a module called event. This module includes an object called as Event. In order to record the events the object

pygame.event.Event is used. In addition to this in order to get the events that has occurred previously pygame.event.get() function is used. There are two main types of event that is pygame.KEYDOWN and pygame.KEYUP. The function of pygame.KEYDOWN is to analyse whether the keys or the mouse are being pressed. The function of pygame.KEYUP is to check if the pressed keys or mouse are released. In addition to this there is another type of event called pygame.QUIT. If the type of the event is pygame.QUIT, then pygame.quit() function is called in order to deactivate the python library and to close the window. This whole mechanism is called as Event handling.

3.3 Classes and Objects

- **Class**

A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members such as class variables, object variables and methods that are accessed via dot notation. The classes are created using the keyword called 'class'. Attributes may be data or functions. As soon as a class is defined, a new class object is created with the same name. This class object is used to access the different attributes as well as to instantiate new objects of that class.

The syntax of class definition is as follows,

Class className:

<statement 1>
.
.
<statement N>

Class body

- **Object**

A unique instance of a class. An object contains both data members (class variables and instance variables) and methods. An object obj that belongs to a class Triangle, for example, is an instance of the class Triangle. Object is created using className followed by () notation that takes arguments within it.

The syntax for creating an object is shown below,

objectName = className(arguments)

Here the arguments passed during the creation of the object is used to assign values to the object variables in __init__() which is called as constructor.

- **Constructor**

Constructor is used to initialize the variables of the class. By default, the python interpreter calls the constructor internally. There are two types of constructors called parameterized and non-parameterized constructors. The __init__ method can take any number of arguments however the first argument is self. Self is a default variable which refers to the object that is currently being created.

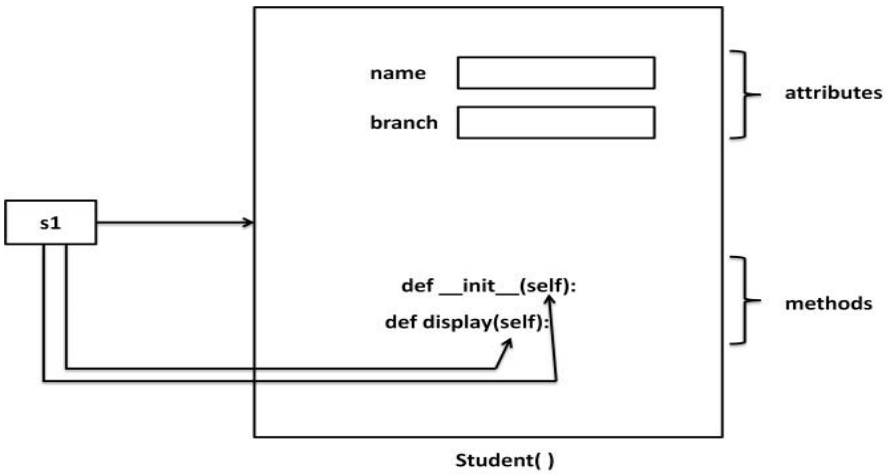

Figure 3.3 Sample representation of classes and objects.

3.4 Functions

Function is a block of code that is used to perform some particular task. The main advantage of using functions is that it reduces the length of the program. A single function can be called multiple times in the program to perform some particular task. All the programming languages have some in-built functions that helps to increase program readability. Other than built-in functions users are also allowed to define their own functions.

There are three components of a function

- Function Declaration
- Function Definition
- Function call

The syntax of defining a function is as shown below,

```
def functionName(parameters):
    //body of function
    return expression
```

The functions in Python are usually defined using 'def' keyword followed by () notation. The arguments/parameters are passed within the parenthesis. The functions may or may not return the values. A function that doesn't return any value are called as void function. The rest of the functions that returns a value are called as fruitful functions. The function defined in the program cannot be executed until and unless the user makes a function call. The functions are called using the function name followed by the parenthesis. The user can also specify the arguments that are required to perform the task within a function.

In the project Mario game, there are some user defined functions that are necessary for the game to execute. They are,

- start_game():

  The execution of the program starts by calling start_game() function. The main objective of this function is to create the home screen of the game. The event handling is also performed by this function where it allows user to get into the game only if the user

press any other key other than the ESC key. If the ESC key is pressed by the user then the function calls pygame.quit() and ends the game.

- Check_level():

  The main objective of this function is to increase the level based on the score. The function checks the current score and then increases the level of the game only if the difference of the previous and the current score is 10. During each increase of the level, walls on either side of the screen increases.

- game_over():

  This function gets executed once the Mario hits the fireball or touches the wall. The function displays the current score achieved by the user. This function also performs event handling operations. It allows the player to play the game again or to quit. Depending on the choice of the player the particular task gets performed.

- game_loop():

  The main function in the project is gameloop() function. This function includes updating the game state and event handling tasks. It includes an infinite while loop where the functions and classes are called in order to perform tasks. This loop is repeated until and unless the pygame.quit() function is called that deactivates the pygame library and closes the window..

3.5 Project Code

```
import pygame

import sys

pygame.init()

WINDOW_WIDTH = 1200

WINDOW_HEIGHT = 600

FPS = 20

BLACK = (0, 0, 0)

GREEN = (0, 255, 0)

ADD_NEW_FLAME_RATE = 25

cactus_img = pygame.image.load('cactus_bricks.png')

cactus_img_rect = cactus_img.get_rect()

cactus_img_rect.left = 0

fire_img = pygame.image.load('fire_bricks.png')

fire_img_rect = fire_img.get_rect()

fire_img_rect.left = 0

CLOCK = pygame.time.Clock()

font = pygame.font.SysFont('forte', 20)

canvas = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))

pygame.display.set_caption('Mario')

class Topscore:

    def __init__(self):

        self.high_score = 0
```

```
    def top_score(self, score):
        if score > self.high_score:
            self.high_score = score
        return self.high_score
topscore = Topscore()
class Dragon:
    dragon_velocity = 10
    def __init__(self):
        self.dragon_img = pygame.image.load('dragon.png')
        self.dragon_img_rect = self.dragon_img.get_rect()
        self.dragon_img_rect.width -= 10
        self.dragon_img_rect.height -= 10
        self.dragon_img_rect.top = WINDOW_HEIGHT/2
        self.dragon_img_rect.right = WINDOW_WIDTH
        self.up = True
        self.down = False
    def update(self):
        canvas.blit(self.dragon_img, self.dragon_img_rect)
        if self.dragon_img_rect.top <= cactus_img_rect.bottom:
            self.up = False
            self.down = True
        elif self.dragon_img_rect.bottom >= fire_img_rect.top:
            self.up = True
```

```
            self.down = False
        if self.up:
            self.dragon_img_rect.top -= self.dragon_velocity
        elif self.down:
            self.dragon_img_rect.top += self.dragon_velocity
class Flames:
    flames_velocity = 20
    def __init__(self):
        self.flames = pygame.image.load('fireball.png')
        self.flames_img = pygame.transform.scale(self.flames, (20, 20))
        self.flames_img_rect = self.flames_img.get_rect()
        self.flames_img_rect.right = dragon.dragon_img_rect.left
        self.flames_img_rect.top = dragon.dragon_img_rect.top + 30
    def update(self):
        canvas.blit(self.flames_img, self.flames_img_rect)
        if self.flames_img_rect.left > 0:
            self.flames_img_rect.left -= self.flames_velocity
class Mario:
    velocity = 10
    def __init__(self):
        self.mario_img = pygame.image.load('maryo.png')
        self.mario_img_rect = self.mario_img.get_rect()
        self.mario_img_rect.left = 20
```

```
        self.mario_img_rect.top = WINDOW_HEIGHT/2 - 100
        self.down = True
        self.up = False
    def update(self):
        canvas.blit(self.mario_img, self.mario_img_rect)
        if self.mario_img_rect.top <= cactus_img_rect.bottom:
            game_over()
            if SCORE > self.mario_score:
                self.mario_score = SCORE
        if self.mario_img_rect.bottom >= fire_img_rect.top:
            game_over()
            if SCORE > self.mario_score:
                self.mario_score = SCORE
        if self.up:
            self.mario_img_rect.top -= 10
        if self.down:
            self.mario_img_rect.bottom += 10
def game_over():
    pygame.mixer.music.stop()
    music = pygame.mixer.Sound('mario_dies.wav')
    music.play()
    topscore.top_score(SCORE)
    game_over_img = pygame.image.load('end.png')
```

```
    game_over_img_rect = game_over_img.get_rect()
    game_over_img_rect.center = (WINDOW_WIDTH/2, WINDOW_HEIGHT/2)
    canvas.blit(game_over_img, game_over_img_rect)
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    pygame.quit()
                    sys.exit()
                music.stop()
                game_loop()
        pygame.display.update()
def start_game():
    canvas.fill(BLACK)
    start_img = pygame.image.load('start.png')
    start_img_rect = start_img.get_rect()
    start_img_rect.center = (WINDOW_WIDTH/2, WINDOW_HEIGHT/2)
    canvas.blit(start_img, start_img_rect)
    while True:
        for event in pygame.event.get():
```

```
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    pygame.quit()
                    sys.exit()
                game_loop()
        pygame.display.update()
def check_level(SCORE):
    global LEVEL
    if SCORE in range(0, 10):
        cactus_img_rect.bottom = 50
        fire_img_rect.top = WINDOW_HEIGHT - 50
        LEVEL = 1
    elif SCORE in range(10, 20):
        cactus_img_rect.bottom = 100
        fire_img_rect.top = WINDOW_HEIGHT - 100
        LEVEL = 2
    elif SCORE in range(20, 30):
        cactus_img_rect.bottom = 150
        fire_img_rect.top = WINDOW_HEIGHT - 150
        LEVEL = 3
```

```
    elif SCORE > 30:
        cactus_img_rect.bottom = 200
        fire_img_rect.top = WINDOW_HEIGHT - 200
        LEVEL = 4
    elif SCORE >40:
        cactus_img_rect.bottom = 250
        fire_img_rect.top = WINDOW_HEIGHT - 250
        LEVEL = 5
def game_loop():
    while True:
        global dragon
        dragon = Dragon()
        flames = Flames()
        mario = Mario()
        add_new_flame_counter = 0
        global SCORE
        SCORE = 0
        global  HIGH_SCORE
        flames_list = []
        pygame.mixer.music.load('mario_theme.wav')
        pygame.mixer.music.play(-1, 0.0)
        while True:
            canvas.fill(BLACK)
```

```
        check_level(SCORE)
        dragon.update()
        add_new_flame_counter += 1
        if add_new_flame_counter == ADD_NEW_FLAME_RATE:
            add_new_flame_counter = 0
            new_flame = Flames()
            flames_list.append(new_flame)
        for f in flames_list:
            if f.flames_img_rect.left <= 0:
                flames_list.remove(f)
                SCORE += 1
            f.update()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    mario.up = True
                    mario.down = False
                elif event.key == pygame.K_DOWN:
                    mario.down = True
                    mario.up = False
```

```
            if event.type == pygame.KEYUP:
                if event.key == pygame.K_UP:
                    mario.up = False
                    mario.down = True
                elif event.key == pygame.K_DOWN:
                    mario.down = True
                    mario.up = False
        score_font = font.render('Score:'+str(SCORE), True, GREEN)
        score_font_rect = score_font.get_rect()
        score_font_rect.center = (200, cactus_img_rect.bottom + score_font_rect.height/2)
        canvas.blit(score_font, score_font_rect)
        level_font = font.render('Level:'+str(LEVEL), True, GREEN)
        level_font_rect = level_font.get_rect()
        level_font_rect.center = (500, cactus_img_rect.bottom + score_font_rect.height/2)
        canvas.blit(level_font, level_font_rect)
        top_score_font = font.render('Top Score:'+str(topscore.high_score),True,GREEN)
        top_score_font_rect = top_score_font.get_rect()
        top_score_font_rect.center = (800, cactus_img_rect.bottom + score_font_rect.height/2)
        canvas.blit(top_score_font, top_score_font_rect)
        canvas.blit(cactus_img, cactus_img_rect)
        canvas.blit(fire_img, fire_img_rect)
        mario.update()
        for f in flames_list:
```

```
            if f.flames_img_rect.colliderect(mario.mario_img_rect):
                game_over()
                if SCORE > mario.mario_score:
                    mario.mario_score = SCORE
        pygame.display.update()
        CLOCK.tick(FPS)
start_game()
```

CHAPTER 4

SAMPLE OUTPUT

- **Python IDLE output screen**

Figure 4.1

- **Home page of Mario game**

  The pygame window opens and the user is asked to enter any key in order to enter the game.

Figure 4.2

- **Overview of the game**

  The player has to move Mario UP and DOWN in order to escape from the fireballs thrown by the dragon.

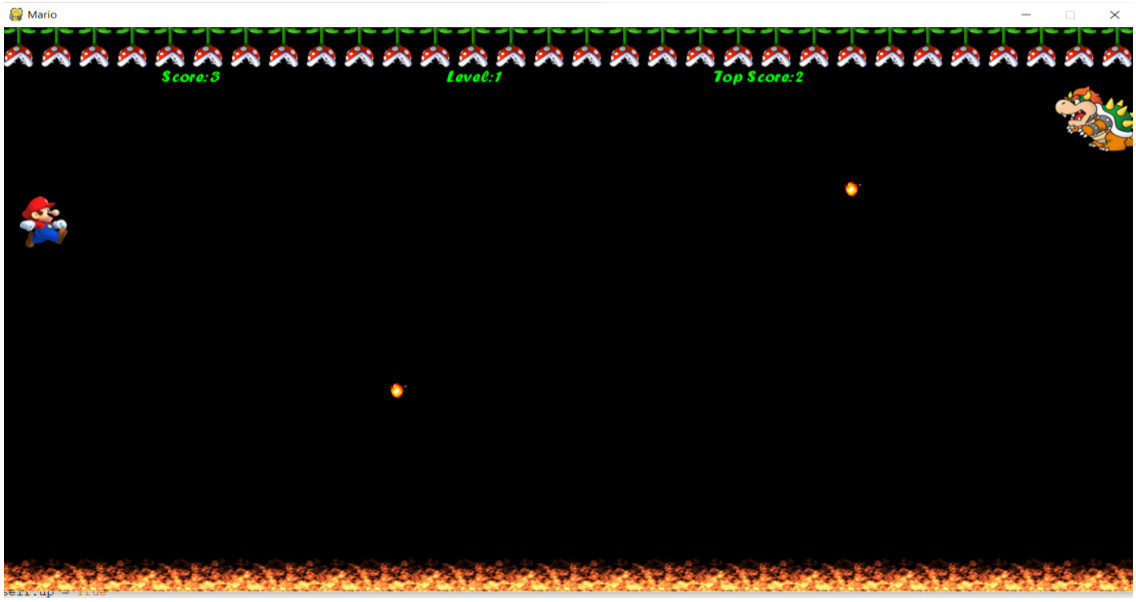

Figure 4.3

- **Increase in the level**

  As the level increases the intensity of the game also increases. The walls on either side of the window increases, thus reducing the area of the game.

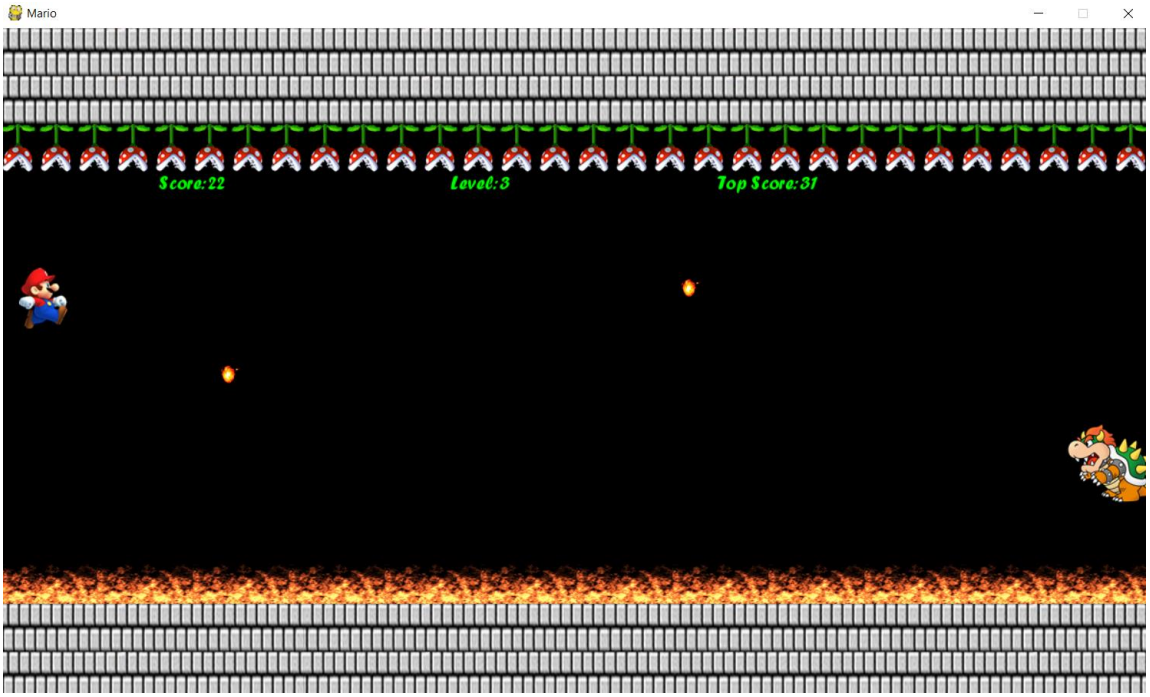

Figure 4.4

- **Losing the game**

  Once the Mario hits the fireballs or touches the wall on either side of the screen, game ends and the player is asked whether he wants to continue the game or quit the game.

Figure 4.5

- **Quitting the game**

  If the player wishes to quit the game by pressing ESC key, the pygame window closes and the game ends.

CHAPTER 5

CONCLUSION

Developing video games is a worldwide trend nowadays. The Mario game is an adventurous game that requires a lot of focus and concentration. Since the player hast to be very attentive to escape from the fireballs, this game requires hand-eye coordination. Studies have shown that playing games increases the grey matter and efficiency of our brain. This adds feather to the cap of this game. Games not only provide entertainment but also makes player learn about problem-solving, strategy, trust and calculated risk taking. It relaxes the mind and creates sportiveness among the players.

BIBLIOGRAPHY

[1]. https://inventwithpython.com

[2]. https://opensource.com

[3]. https://www.pythonforbeginners.com

[4]. https://www.geeksforgeeks.com

[5]. Core Python Programming textbook by Dr R Nageshwara Rao.