Table of Contents

Introduction

Our computer science project, called "**Arena**", is a game compilation consisting of two games with additional features. The first game is the **Maze**. In this game the player has to collect five gems each having fifteen points. Collecting these five gems unlocks the exit gate of the first game. On successful completion of the first game within the one minute time frame, the player is directed to the second game which is our interpretation of the game "**Flappy Bird**". The second game is not bound by any time constraints. Here the player's aim is to collect as many points as he/she can without flying into obstacles as the speed of the game goes on increasing as the player's points increase. The game also has a home page from which the following can be accessed:-

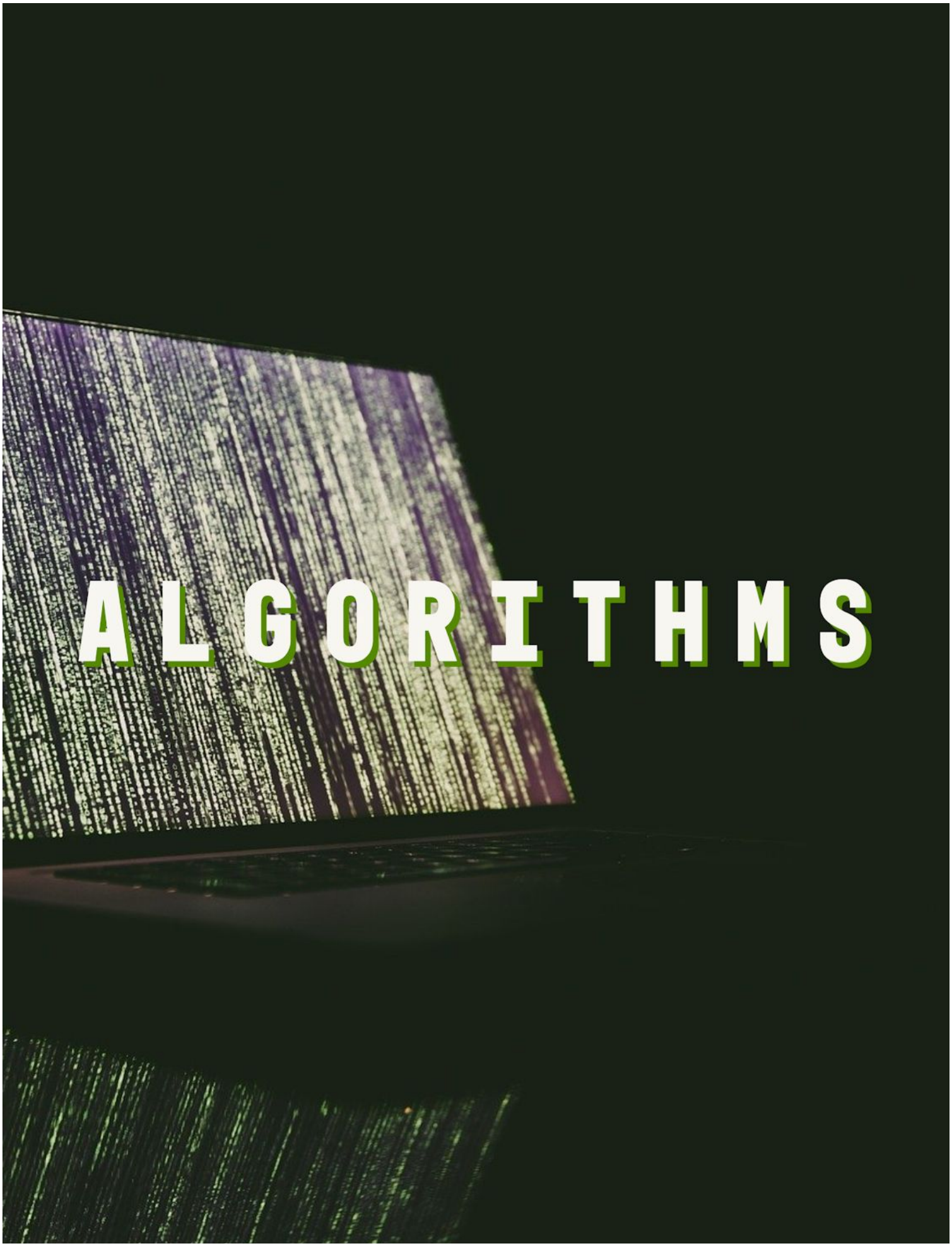
1. A leaderboard where the top ten scores can be viewed
2. The first game (which continues to the second)
3. An information board which gives a brief description of our project
4. A choice window where the player can choose a level
5. A button to quit the program


SYSTEM
REQUIREMENTS

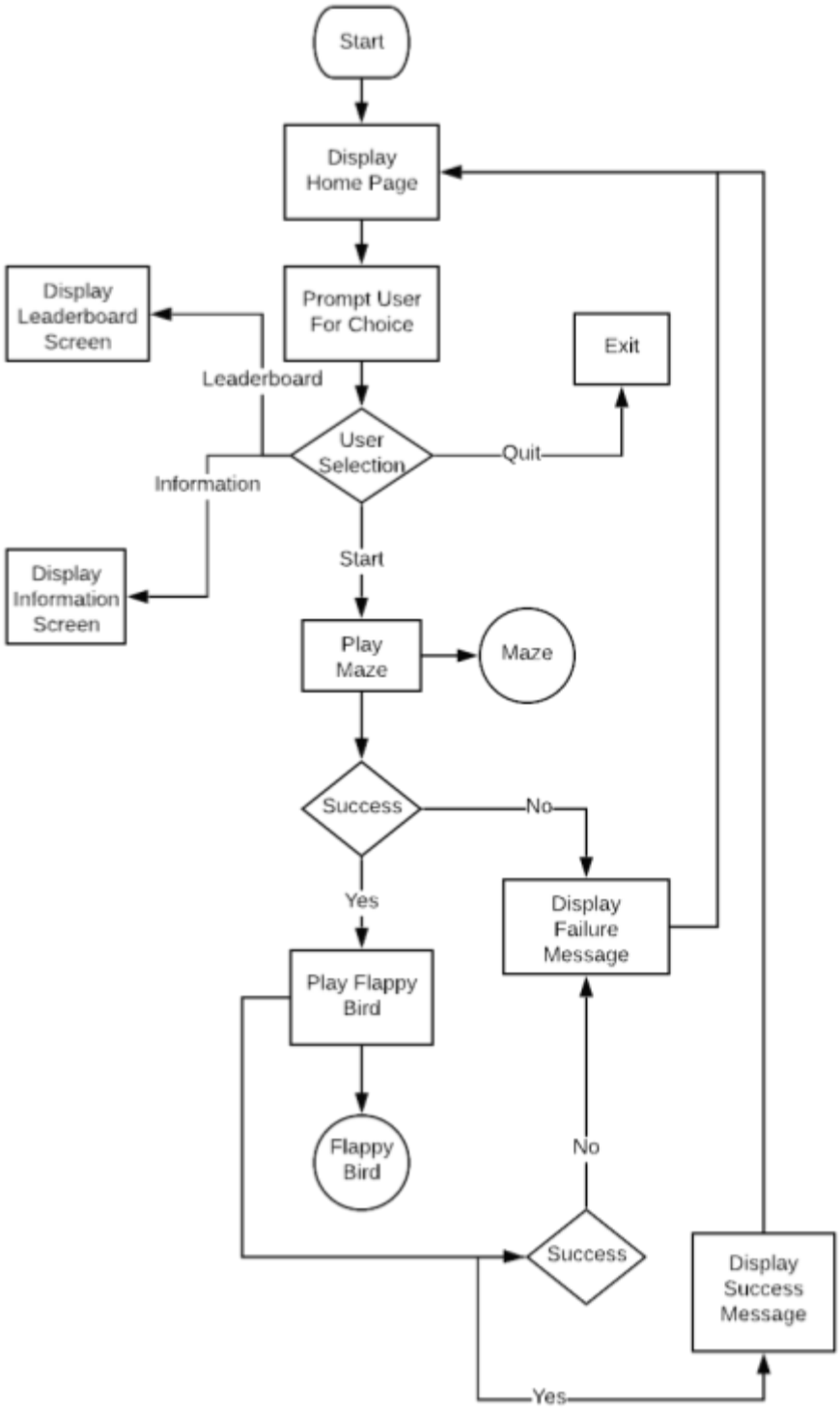
System Requirements

The system requirements for our project are:-

1. IBM PC Compatible
2. 8 GB ram
3. Windows 10
4. Python 3.6
5. Pygame
6. Python-MySQL connector (mysql.connector)
7. MySQL


ALGORITHMS

Algorithms



Flow chart for the working of the entire program

Home Page

On running the program, the first screen that appears is the homepage. In this page there are four buttons - Leaderboard, Start, Quit, Choose a Game and Info - which lead to four subpages.

The Start Button

The start button leads to the Maze game running program. On successful completion of the first game (described below), the program of the second game is run.

The Leaderboard Button

This button leads to the subpage which is responsible for displaying the top ten scores stored in the MySQL database ‘score’.

The Choose a Game Button

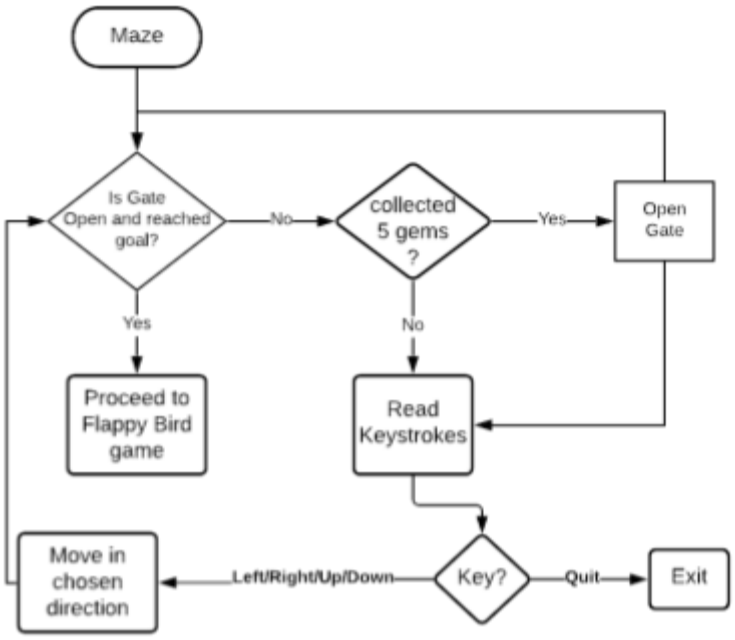
This button leads you to a page where you can pick a level to play.

The Info Button

This button leads to the page which contains a basic description of our project.

The Quit Button

This button ends the program.



Flow chart for the working of the first game - Maze

First Game - Maze

When the start button is pressed this is the first program that runs. In this game the player has to cross a maze while collecting five gems. Once the gems are collected, the exit gate is unlocked. After this the player has to reach the unlocked gate within a one minute time period to successfully cross the level and begin the second one.

If the player is unable to unlock and reach the end gate within one minute, a “Your time is up!” message will pop up followed by a new screen displaying your score. From here the player returns to the home page again.

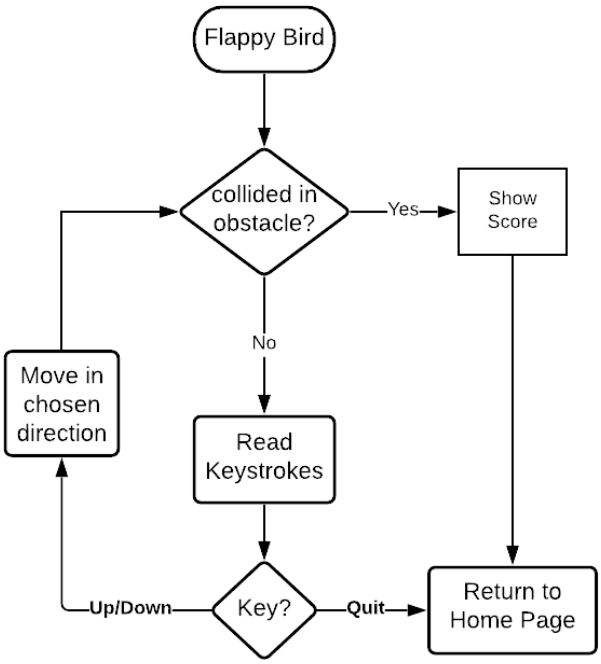
On playing this level, the player is given one of two maze layouts at random. Also, in this game the player can earn scores in two ways - 5 points for each gem and 2 points for each second the player saves to reach the end.

Second Game - Flappy Bird

When the Maze is successfully completed, this game begins. In this game the player has to keep the gaming running as long as possible without running into obstacles. The player controls only the up and down movements of the player icon to avoid pillars(using the up and down arrows) while the forward speed of the game is controlled by the game. As the time increases the number of frames per second increases thereby increasing the forward speed. Points are awarded for each obstacle avoided (+1) and for collecting additional gems(+5) scattered. The points from the first game are carried on to this game too. Once the player collides with an obstacle, the game ends, followed by a page displaying a score. After this the home page is displayed again.

During the entire game the score is being displayed on the top right side of the screen.

We also have sound effects to mimic the car being run, collecting the diamond, passing the pillar or crashing it.



User-Defined Functions and Classes

Functions involved in database

getConnection (Function)

This function creates a connection to the database ‘score’ which stores all the scores in the table ‘scoreboard’ and returns the connection object.

scoreboard (Function)

This function adds a score to the table ‘scoreboard’ after each series of games ends. It is called by the main program. It takes two arguments - name of the user, score obtained.

fetchscore (Function)

This function fetches the first ten scores from the table ‘scoreboard’ when arranged in descending order in the form of a nested list. If the connection is not successfully established or no tuples are found in the ‘scoreboard’ table, None is returned. This condition is checked for by the function calling this function.

Secondary Environments Supporting the Game

LeadEnvironment (Class)

This class creates the entire interface of the leaderboard window including the button and text.

run (Function)

This function runs the leaderboard page when it is called from the main program. It draws all the pieces of the page by calling the draw function and then keeps checking if the mouse cursor intersects with the button.

draw (Function)

This function draws the text, the button and the background of the leaderboard page when called by the run function.

InfoEnvironment (Class)

This class creates the entire interface of the leaderboard window including the button and text.

run (Function)

This function runs the Information page when it is called from the main program. It draws all the pieces of the page by calling the draw function and then keeps checking if the mouse cursor intersects with the button.

draw (Function)

This function draws the text, the button and the background of the Information page when called by the run function.

ScoreEnvironment (Class)

This class creates the entire interface of the leaderboard window including the button and text.

run (Function)

This function runs the scoreboard page when it is called from the main program. It draws all the pieces of the page by calling the draw function and then keeps checking if the mouse cursor intersects with the button.

draw (Function)

This function draws the text, the button and the background of the scoreboard page when called by the run function.

ChoiceEnvironment (Class)

This class creates the entire interface of the 'choose a game' window including buttons and text.

run (Function)

This function runs the 'Choose a game' page when it is called from the main program. It draws all the pieces of the page by calling the draw function and then keeps checking if the mouse cursor intersects with the button.

draw (Function)

This function draws the text, the button and the background of this page when called by the run function.

Classes and Functions involved in the Maze game

Ball (Class)

This class defines the ball object which is used by the player to move around the maze.

moveLeft (Function)

This function changes the x coordinate of the ball so that it moves to the left. Thus, it reduces the `self.x` value by `self.speed` amount.

moveRight(Function)

This function changes the x coordinate of the ball so that it moves to the right. Thus, it increases the `self.x` value by `self.speed` amount.

moveUp (Function)

This function changes the y coordinate of the ball so that it moves up. Thus, it reduces the `self.y` value by `self.speed` amount.

moveDown (Function)

This function changes the y coordinate of the ball so that it moves down. Thus, it increases the `self.y` value by `self.speed` amount.

Gem (Class)

This class makes gem objects which are points in the game. It has the following attributes:-

draw (Function)

It draws the image of the gem.

Wall (Class)

This class makes a wall object. Each wall object is a tile in the maze layout. It has the following attributes:-

draw (Function)

It draws the rectangle of the wall and the image to be drawn on it.

Finish (Class)

This class makes a finish line object. It has the following attributes:-

draw (Function)

It draws the rectangle of the finish line at the specified coordinates on the screen.

Finishwall (Class)

This class makes a finish line object. It has the following attributes:-

draw (Function)

It draws the rectangle of the finish wall at the specified coordinates on the screen.

Button (Class)

This class is used to create button objects. All the operational buttons in the interface are objects of this class.
It has the following attributes:-

draw (Function)

This function is responsible to draw the text and the rectangle depicting the button when the draw function of the main program is called.

intersect (Function)

This function returns a True value if the mouse cursor is within the area of the button and false otherwise.

Rectangle (Class)

This class is used for comparing coordinates of two rectangles for checking intersections and also creating rectangle coordinates out of circles.

fromCircle (Function)

This is a function that takes in the x, y coordinates and the radius of a circle and then returns a Rectangle class object of the same coordinates, width and height as the initial circle.

intersect (Function)

This function takes two Rectangle objects (one passed in the argument and the other is self) and returns True if the two objects intersect.

Environment (Class)

This class controls all the parts of the maze game.

drawTime (Function)

This function does all the calculations for and is used to display the run time of the maze game at the bottom right corner of the screen.

beep (Function)

It plays `self.m1` sound.

unlocksound (Function)

It plays `self.m2` sound.

run (Function)

It does the following:-

- It spawns all the gems.
- It draws all the elements on the screen by calling the draw function.
- It moves the ball object after checking for collisions of any kind.
- It determines when the winning or losing text should be displayed

handleGems (Function)

It checks for the intersection of the ball with the gem object and modifies the score accordingly. It also checks whether all the gems are collected and then changes the `self.unlock` value.

draw (Function)

It does the following:-

- It initialises the screen
- It draws the background and all the walls
- It draws the gems and the ball object
- It draws the finish line and the finish wall blocking it. It also checks the value of `self.unlock` to remove the finish wall.
- It checks whether the win or lose text is not None so that it can display it.

addWall (Function)

It adds wall objects to the list `self.walls`.

addFinish (Function)

It initialises the finish line object.

addFinishwall (Function)

It initialises the finishing wall object.

doesIntersect (Function)

It checks if the x coordinate, y coordinate, width and height passed intersects with the rectangle formed by the wall by making an object from the Rectangle class.

doesIntersectWithFinwall (Function)

It checks if the x coordinate, y coordinate, width and height passed intersects with the rectangle formed by the finish wall by making an object from the Rectangle class.

doesIntersectWithGem (Function)

It checks if the x coordinate, y coordinate, width and height passed intersects with the rectangle formed by the gem by making an object from the Rectangle class.

removeGem (Function)

It removes the gem object passed in the function from the list containing gems.

addGem (Function)

It adds a gem to the list containing gems and also assigns it it's score.

spawnGems (Function)

It makes sure that only the specified number of gems are displayed. It increments the count of gems every time a gem is added to the list of gems.

findPosition (Function)

It finds a position for a gem object such that the gem is not overlapping with walls.

finishGame (Function)

It returns True if the ball intersects with the finish line. Otherwise it returns False.

expireGame (Function)

It returns True if the time elapsed since the program began is greater than sixty seconds. This is used to check whether to end the program.

GameMap (Class)

This class creates the maze structure with the given background images, tile images, maze map etc. Once it has done so, it takes the help of the Environment class to run the maze game.

run (Function)

This function adds the wall, finish line and finish wall objects. It then uses the run function of self.env to run the game. The function returns the score.

runGame (Function)

This function runs the maze and flappy bird game in order. It makes sure that the score is carried on. It chooses one of two designs for the maze level using a random module. Once both the levels end, it returns the total score.

runMaze1 (Function)

This function runs only the first maze layout. It is used when a player tries to play from the 'Choose a Game' section of the program. It returns the score obtained when done.

runMaze2 (Function)

This function runs the second maze layout. The rest of its workings are identical to `runMaze1()`.

runFlappy (Function)

This function runs the Flappy Bird game. It returns the score once the game is over.

StartEnvironment (Class)

This class brings all the sub programs like the games, the various sub pages etc. together.

run (Function)

This function runs the home page. It calls the draw function to draw all the parts of the page. It checks for any intersections made with any of the buttons so that it can direct the player to the respective subpages by calling their functions.

draw (Function)

This function draws all the buttons and the background of the window.

Functions involved in the Flappy Bird game

startScreen (Function)

To display the first screen after passing level1.

mainGame (Function)

Handles actual game play and display.

hitObstacle (Function)

Checks if the car collides with any pillars.

rewardCollectec (Function)

Checks when diamond is collected.

getRandomObstacle (Function)

To get an arbitrary pillar.

runcargame (Function)

Used to keep track of speed and score of car.

Data Members and Variables

Variables in gameworking.py

Global

- `FPS`: Frames per second, speed with which the game is repainted
- `SCREENWIDTH`: It is the width of the screen
- `SCREENHEIGHT`: It is the height of the screen
- `GROUND_Y`: It is the y coordinate of the image which represents the ground.
- `IMAGES`: dictionary of images used in the game. It has the images of the ground, the background, the player, the reward image, the numbers and the obstacle.
- `SOUND`: dictionary of the sounds used in the game. It has the sounds to be played when the player dies, the player hits the obstacle, the player gains a point, the player is flapping their wings etc.
- `PLAYER`: the image representing the player
- `BACKGROUND`: the image for the background of the game
- `OBSTACLE`: the image representing the obstacle
- `REWARD`: The image for the reward.
- `SCREEN`: It is the surface on which everything is drawn
- `FPSCLOCK`: This is the clock object which defines the time of a running program.

startScreen (Function)

- `playerx`: It is the x coordinate of the player image
- `playery`: It is the y coordinate of the player image
- `messagex`: It is the x coordinate of the message image on the start screen.
- `messagey`: It is the y coordinate of the message image on the start screen.
- `basex`: It is the x coordinate of the image for the ground.

mainGame (Function)

- It has an argument running which has a boolean value which tells whether the program within this function should run or not.
- `playerx`: It is the x coordinate of the player image
- `playery`: It is the y coordinate of the player image
- `basex`: It is the x coordinate of the image for the ground.
- `newObstacle1`: It is a list of two dictionaries each giving an obstacle - one from the roof and the other from the ceiling.

- `newObstacle2`: Same as newObstacle1 but of different random values.
- `upperObstacles`: It has a list of two dictionaries again containing two obstacles meant to come from the ceiling. But, these two obstacles take their y values from newObstacle1 and newObstacle2 and have different x values.
- `lowerObstacles`: It has a list of two dictionaries again containing two obstacles meant to come from the floor. These two obstacles have the same x values as thor respective `upperObstacles` and are just below their `upperObstacles`.

runcargame (Function)

This function has two arguments: `incomingscore` and `running`. They by default are 0 and True respectively. The incoming score is the score that has to be carried over from the maze game and the running variable, also carried over from the maze game, tells whether the second game should run or not. This is used when the first game expires when the player does not complete it on time.
`score`: It has the value of `incomingscore`. This variable is returned by this function.
`icScore`: It has the running score.
`IMAGES`: Here the images get loaded into a dictionary
`SOUND`: Here the sounds get loaded into a dictionary and used

getConnection (Function)

`connection`: This is the variable returned by the `getConnection()` function

scoreboard (Function)

- `connection`: This is the variable returned by the `getConnection()` function
- `cursor`: This is the cursor object
- `n`: Stores the name from the argument.
- `sc`: Stores the score from the argument.
- `sql`: This stores the query that is used to add a tuple into the database containing name of user and score.

fetchscore() (Function)

- `connection`: This is the variable returned by the `getConnection()` function.
- `cursor`: This is the cursor object.
- `scores`: Stores a list of the first ten tuples obtained from the table.
- `e`: This variable stores the exception that has occurred (if it occurs).

runGame (Function)

- `map1`: The list that stores the first maze layout in the form of strings inside a list.
- `map2`: The list that stores the second maze layout in the form of strings inside a list.
- `gm1`: The first GameMap object having the first maze layout.
- `gm2`: The second GameMap object having the second maze layout.
- `chosenmap`: It is a random list chosen among gm1 and gm2 using the random module.
- `firstscore`: It is the score the player collects playing the first game.
- `totalscore`: It is the score the player collects in total, i.e., first and second game.

runMaze1 (Function)

- `map1`: The list that stores the first maze layout in the form of strings inside a list.
- `gm1`: The first GameMap object having the first maze layout.
- `score`: It is the score the player collects on completing this game.

runMaze2 (Function)

- `map2`: The list that stores the second maze layout in the form of strings inside a list.
- `gm2`: The second GameMap object having the first maze layout.
- `score`: It is the score the player collects on completing this game.

runFlappy (Function)

`score`: It is the score the player collects on completing this game.


SOURCE
CODE

Source Code

```
#startpg.py
import pygame
from start.button import Button
from maze import mainrun
import pygame, sys, random
from start.leaderboard import LeadEnvironment
from start.choiceboard import ChoiceEnvironment
from start.infoboard import InfoEnvironment
from maze.scorepg import ScoreEnvironment
from start.CompProjMySQL import scoreboard as sb
import getpass
pygame.init()

class StartEnvironment():  #This is a class which makes the start page and all elements on
it when the program is executed
    def __init__(self, background):
        pygame.font.init() # you have to call this at the start,
                           # if you want to use this module.
        self.headsize = 100
        self.h = 700 #screen height
        self.w = 700 #screen width
        self.maxxy = 20
        self.fact = self.h//self.maxxy #this variable helps to easily transform size of the
screen if required. It just needs to be multiplied to whatever is to be changed
        self.screen = pygame.display.set_mode((self.w, self.h))
        self.startbimg1 = "start/startbtnonclick.jpg"               #following are all the
images that are used in this program
        self.startbimg2 = "start/startbtnnoclick.jpg"
        self.headbimg1 = self.headbimg2 = "start/mazebtnbg.jpg"
        self.btnimg1 = "start/genbtnonclick.jpg"
        self.btnimg2 = "start/genbtn.jpg"
        self.leaderbimg1 = "start/leaderboard.jfif"
        self.leaderbimg2 = "start/leaderboardonclick.jfif"
        self.infobimg = "start/info.png"
        #the following 5 variables initialise the buttons on the screen. The buttons are
created by the button class.
        self.headb = Button(6*self.fact,4*self.fact,10*self.fact,3*self.fact,self.headbimg1,
self.headbimg2,(255, 255, 255),self.screen,'Arena',self.headsize)
        self.leadb =
Button(7*self.fact,8*self.fact,8*self.fact,2*self.fact,self.leaderbimg1,
self.leaderbimg2,(0,0,0),self.screen,'Leaderboard')
        self.startb =
Button(8*self.fact,11*self.fact,6*self.fact,2*self.fact,self.startbimg1,
self.startbimg2,(0,0,0),self.screen,'Start')
        self.quitb = Button(8*self.fact,14*self.fact,6*self.fact,2*self.fact,self.btnimg1,
self.btnimg2,(0,0,0),self.screen,'Quit')
        self.infob = Button(18*self.fact, 16*self.fact, 1*self.fact, 1*self.fact,
self.infobimg, self.infobimg, (0,0,0), self.screen, ' ')
        self.choiceb =
Button(6*self.fact,17*self.fact,10*self.fact,2*self.fact,self.leaderbimg1,
self.leaderbimg2,(0,0,0),self.screen,'Choose a Game')
```

```
        self.infoenv = InfoEnvironment() #This is the enivronment of the info page - a
subpage of the start page.
        self.running = True
        self.bgimg = pygame.image.load(background)
        self.bgimg = pygame.transform.scale(self.bgimg,(self.w,self.h))

    def run(self): #This function runs all individual parts of the start page.
        running=True
        self.draw() #the draw function draws all the required images, texts, buttons
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                coord = pygame.mouse.get_pos()
                if self.quitb.intersect(coord): #checking if the mouse pointer is within the
area of the quit button
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        running=False
                elif self.startb.intersect(coord): #checking if the mouse pointer is within
the area of the start button
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        #running=False
                        self.score = mainrun.runGame()
                        if self.score ==None:
                            self.score=0
                        sb(getpass.getuser(), self.score)
                        self.scoreenv = ScoreEnvironment(self.score)
                        self.scoreenv.run()
                        self.screen = pygame.display.set_mode((self.w, self.h))
                elif self.leadb.intersect(coord): #checking if the mouse pointer is within
the area of the leaderboard button
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        self.leadenv = LeadEnvironment()
                        self.leadenv.run()
                        pygame.init()
                elif self.choiceb.intersect(coord): #checking if the mouse pointer is within
the area of the leaderboard button
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        self.chenv = ChoiceEnvironment()
                        self.chenv.run()
                        pygame.init()
                elif self.infob.intersect(coord): #checking if the mouse pointer is within
the area of the info button
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        self.infoenv.run()
                        pygame.init()
            self.draw()
        pygame.quit()

    def draw(self):
        #self.screen.fill((0, 200, 150))
        self.screen.blit(self.bgimg,[0,0])
        self.headb.draw()
        self.leadb.draw()
        self.startb.draw()
        self.quitb.draw()
```

```
        self.infob.draw()
        self.choiceb.draw()
        pygame.display.update()

env = StartEnvironment("start/startbg.jpg")
env.run()

                                    #leaderboard.py
import pygame
from start.button import Button
from start.CompProjMySQL import fetchscore as fs
pygame.init()

class LeadEnvironment():
    def __init__(self):
        pygame.font.init() # you have to call this at the start,
                           # if you want to use this module.
        self.headtxt = 'Top 10'
        self.headrow = "{:10s}     {:4s}".format('User Name', 'Score')
        self.headtxtsize = 70
        self.headrowtxtsize = 40
        self.txtsize =30
        self.headfont = pygame.font.SysFont('Impact', self.headtxtsize)
        self.headrowfont = pygame.font.SysFont('Impact', self.headrowtxtsize)
        self.myfont = pygame.font.SysFont('Comic Sans MS', self.txtsize)
        self.scores= fs()
        self.headsurface = self.headfont.render(self.headtxt, True, (0, 0, 0))
        self.headrowsurface = self.headrowfont.render(self.headrow, True, (44, 0, 105))
        self.textsurfaces=[]
        if self.scores!=None:
            for i in self.scores:
                self.textsurfaces.append(self.myfont.render("{:10s}".format(i[0]), True,
(44, 0, 105)))
                self.textsurfaces.append(self.myfont.render("{:>4d}".format(i[1]), True,
(44, 0, 105)))
        self.headsize = 90
        self.h = 700
        self.w = 700
        self.maxxy = 20
        self.fact = self.h//self.maxxy
        self.screen = pygame.display.set_mode((self.w, self.h))
        self.btnimg1 = "start/genbtnonclick.jpg"
        self.btnimg2 = "start/genbtn.jpg"
        self.returnb =
Button(6*self.fact,16*self.fact,10*self.fact,2*self.fact,self.btnimg1,
self.btnimg2,(0,0,0),self.screen,'Return to Start')
        self.running = True
        self.bgimg = pygame.image.load("start/leaderbg.jfif")
        self.bgimg = pygame.transform.scale(self.bgimg,(self.w,self.h))

    def run(self):
        running=True
        self.draw()
        flag=None
        while running:
            for event in pygame.event.get():
```

```
                if event.type == pygame.QUIT:
                    running = False
                coord = pygame.mouse.get_pos()
                if self.returnb.intersect(coord):
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        print('True')
                        running=False
                        flag='end'
            if flag!= 'end':
                self.draw()


    def draw(self):
        #self.screen.fill((0, 200, 150))
        self.screen.blit(self.bgimg,[0,0])
        self.returnb.draw()
        y=10
        self.screen.blit(self.headsurface, (10, y))
        y+=self.headtxtsize+50
        self.screen.blit(self.headrowsurface, (10, y))
        y+=self.headrowtxtsize+10
        xleft =10; xright= 300
        cnt=0
        for i in self.textsurfaces:
            if cnt%2==0:
                self.screen.blit(i,(xleft, y))
            else:
                self.screen.blit(i,(xright, y))
                y+=self.txtsize
            cnt+=1
        pygame.display.update()
```

#infoboard.py

```
import pygame
from start.button import Button
pygame.init()

class InfoEnvironment(): #This class displays and runs everything on the information page
    def __init__(self):
        pygame.font.init() # you have to call this at the start,
                           # if you want to use this module.
        f=open("start/about.txt", 'r')
        self.textlines = f.readlines()
        self.textsurfaces=[]
        f.close()
        self.txtsize = 20
        self.myfont = pygame.font.SysFont('Comic Sans MS', self.txtsize)
        for i in self.textlines:
            self.textsurfaces.append(self.myfont.render(i.rstrip('\n'), True, (44, 0, 105)))
        self.h = 700
        self.w = 700
        self.maxxy = 20
        self.fact = self.h//self.maxxy
        self.screen = pygame.display.set_mode((self.w, self.h))
        self.btnimg1 = "start/genbtnonclick.jpg"
        self.btnimg2 = "start/genbtn.jpg"
```

```
        self.returnb =
Button(6*self.fact,16*self.fact,10*self.fact,2*self.fact,self.btnimg1,
self.btnimg2,(0,0,0),self.screen,'Return to Start')
        self.running = True
        self.bgimg = pygame.image.load("start/leaderbg.jfif")
        self.bgimg = pygame.transform.scale(self.bgimg,(self.w,self.h))

    def run(self):
        running=True
        self.draw()
        flag=None
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                coord = pygame.mouse.get_pos()
                if self.returnb.intersect(coord):
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        print('True')
                        running=False
                        flag='end'
            if flag!= 'end':
                self.draw()


    def draw(self):
        #self.screen.fill((0, 200, 150))
        self.screen.blit(self.bgimg,[0,0])
        self.returnb.draw()
        y=10
        for i in self.textsurfaces:
            self.screen.blit(i,(10, y))
            y+=self.txtsize
        pygame.display.update()
```

#choiceboard.py

```
import pygame
from start.button import Button
from start.CompProjMySQL import fetchscore as fs
from maze import mainrun
from start.CompProjMySQL import scoreboard as sb
import getpass
from maze.scorepg import ScoreEnvironment
pygame.init()

class ChoiceEnvironment():
    def __init__(self):
        pygame.font.init() # you have to call this at the start,
                        # if you want to use this module.
        self.headtxt = 'Free Play: Choose a level'
        self.headtxtsize = 60
        self.headfont = pygame.font.SysFont('Impact', self.headtxtsize)
        self.scores= fs()
        self.headsurface = self.headfont.render(self.headtxt, True, (0, 0, 0))
        self.headsize = 90
        self.h = 700
```

```
        self.w = 700
        self.maxxy = 20
        self.fact = self.h//self.maxxy
        self.screen = pygame.display.set_mode((self.w, self.h))
        self.btnimg1 = "start/genbtnonclick.jpg"
        self.btnimg2 = "start/genbtn.jpg"
        self.maze1img = "start/maze1.png"
        self.maze2img = "start/maze2.png"
        self.flappyimg = "start/flappy.png"
        self.maze1b = Button(1*self.fact,5*self.fact,5*self.fact,5*self.fact,self.maze1img,
self.maze1img,(0,0,0),self.screen,'1')
        self.maze2b = Button(7*self.fact,5*self.fact,5*self.fact,5*self.fact,self.maze2img,
self.maze2img,(0,0,0),self.screen,'2')
        self.flappybirdb =
Button(13*self.fact,5*self.fact,5*self.fact,5*self.fact,self.flappyimg,
self.flappyimg,(0,0,0),self.screen,'3')
        self.returnb =
Button(6*self.fact,16*self.fact,10*self.fact,2*self.fact,self.btnimg1,
self.btnimg2,(0,0,0),self.screen,'Return to Start')
        self.running = True
        self.bgimg = pygame.image.load("start/leaderbg.jfif")
        self.bgimg = pygame.transform.scale(self.bgimg,(self.w,self.h))

    def run(self):
        running=True
        self.draw()
        flag=None
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                coord = pygame.mouse.get_pos()
                if self.returnb.intersect(coord):
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        print('True')
                        running=False
                        flag='end'
                if self.maze1b.intersect(coord):
                    if event.type == pygame.MOUSEBUTTONDOWN:
                            #running=False
                            self.score = mainrun.runMaze1()
                            if self.score == None:
                                self.score=0
                            sb(getpass.getuser(), self.score)
                            self.scoreenv = ScoreEnvironment(self.score,'Return')
                            self.scoreenv.run()
                            self.screen = pygame.display.set_mode((self.w, self.h))
                if self.maze2b.intersect(coord):
                    if event.type == pygame.MOUSEBUTTONDOWN:
                            #running=False
                            self.score = mainrun.runMaze2()
                            if self.score == None:
                                self.score=0
                            sb(getpass.getuser(), self.score)
                            self.scoreenv = ScoreEnvironment(self.score, 'Return')
                            self.scoreenv.run()
```

```
                            self.screen = pygame.display.set_mode((self.w, self.h))
                if self.flappybirdb.intersect(coord):
                    if event.type == pygame.MOUSEBUTTONDOWN:
                            #running=False
                            self.score = mainrun.runFlappy()
                            if self.score == None:
                                self.score=0
                            sb(getpass.getuser(), self.score)
                            self.scoreenv = ScoreEnvironment(self.score,'Return')
                            self.scoreenv.run()
                            self.screen = pygame.display.set_mode((self.w, self.h))
            if flag!= 'end':
                self.draw()


    def draw(self):
        #self.screen.fill((0, 200, 150))
        self.screen.blit(self.bgimg,[0,0])
        self.returnb.draw()
        self.maze1b.draw()
        self.maze2b.draw()
        self.flappybirdb.draw()
        self.screen.blit(self.headsurface, (10, 10))
        pygame.display.update()
```

#scorepg.py

```
import pygame
from start.button import Button
pygame.init()

class ScoreEnvironment(): #This class is responsible to display and run all elements on the
end score screen
    def __init__(self, score, btntxt='Return to Start'):
        pygame.font.init()
        self.h = 700
        self.w = 700
        self.maxxy = 20
        self.fact = self.h//self.maxxy
        self.score=score
        self.screen = pygame.display.set_mode((self.w, self.h))
        self.lstoftext =['Game Complete!', 'Your Score: {}'.format(self.score)] #This is a
list containing all lines in the screen
        self.txtsize =70
        self.myfont1 = pygame.font.SysFont('Impact', self.txtsize)
        self.myfont2 = pygame.font.SysFont('Comic Sans MS', self.txtsize)
        self.textsurface1=self.myfont1.render(self.lstoftext[0], True, (255,255,255))
        self.textsurface2=self.myfont2.render(self.lstoftext[1], True, (64, 224, 208))
        self.btnimg1 = "maze/scorebtn.jpg"
        self.btnimg2 = "maze/scorebtn1.jpg"
        self.returnb =
Button(7*self.fact,16*self.fact,10*self.fact,2*self.fact,self.btnimg1,
self.btnimg2,(0,0,0),self.screen,btntxt)
        self.running = True
        self.bgimg = pygame.image.load("maze/scorebg.jpg")
        self.bgimg = pygame.transform.scale(self.bgimg,(self.w,self.h))
```

```
    def run(self): #All elements of the score page are run from here
        running=True
        self.draw() #the draw fundtion(which draws everything on the screen) is called
        while running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                coord = pygame.mouse.get_pos()
                if self.returnb.intersect(coord):
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        print('True')
                        running=False
            self.draw()

    def draw(self):
        self.screen.blit(self.bgimg,[0,0])
        self.returnb.draw()
        y=10
        self.screen.blit(self.textsurface1,(50, y))
        self.screen.blit(self.textsurface2,(50, y+self.txtsize))

        pygame.display.update()
```

```
                                    #button.py
import pygame
pygame.font.init()
pygame.init()

class Button(): #This class is used to create buttons everywhere in the program
    def __init__(self, x,y,w,h,bimg1, bimg2,txtcol,screen,txt, txtsize=50):
        self.x =x
        self.y = y
        self.w = w
        self.h = h
        self.txtcol =txtcol
        self.txt=txt
        self.textx = self.x; self.texty = self.y
        self.btnimg1 = pygame.image.load(bimg1)
        self.btnimg1 = pygame.transform.scale(self.btnimg1,(self.w,self.h))
        self.btnimg2 = pygame.image.load(bimg2)
        self.btnimg2 = pygame.transform.scale(self.btnimg2,(self.w,self.h))
        self.btnimg = self.btnimg2
        self.screen = screen
        self.txtsize = txtsize
        self.myfont = pygame.font.SysFont('Impact', self.txtsize)
        self.adjusted = False
        self.textsurface = self.myfont.render(self.txt, True, self.txtcol) #this is the text
on the button
        self.rect = list(self.textsurface.get_rect())
        self.text_width, self.text_height = self.myfont.size(txt)
        self.wgap = (self.w - self.text_width)/2
        self.hgap = (self.h - self.text_height)/2

    def draw(self): #this will draw all the elements of the button on the screen given
        if self.adjusted==True:
            pass
```

```
        else:
            if self.wgap>0:
                self.textx += self.wgap
            elif self.wgap==0:
                self.w +=10
                self.textx+=1

            if self.hgap>0:
                self.texty += self.hgap
            elif self.hgap==0:
                self.y -=10
                self.h +=20

            self.adjusted=True
        #self.screen.draw.text(self.txt, (self.x, self.y), owidth=1.5, ocolor=self.btncol,
color=self.txtcol)

        #pygame.draw.rect(self.screen, self.btncol, (self.x, self.y, self.w, self.h))
        self.screen.blit(self.btnimg,[self.x,self.y])
        self.screen.blit(self.textsurface,(self.textx, self.texty))

    def intersect(self,coord): #it will check if the given coordinate will come within the
area of the button or not
        x,y=coord
        if self.x <= x <= self.x + self.w and self.y <= y <= self.y + self.h:
            self.btnimg =self.btnimg1
            return True
        else:
            self.btnimg = self.btnimg2
            return False
```

#CompProjMySQL.py

```
import mysql.connector

#the database and table have already been created
def getConnection():
    connection = mysql.connector.connect(host = 'localhost', user = 'root', passwd =
'pa123', database = 'score')
    return connection

def scoreboard(name, score):
    #adding a name and the score and displaying in desending order
    connection = getConnection()
    cursor = connection.cursor()
    cursor.execute('use score') #score is the database

    if connection.is_connected():
        n = name
        sc = score
        sql = 'insert into scoreboard (name, score) values (%s, %s)'
        row = [(n, sc)]
        cursor.executemany(sql, row) #name varchar(30) primary key, score int(10)
        connection.commit()
        cursor.close()
        connection.close()
    else:
```

```
        print('cannot connect to the database. please try connecting again.')

def fetchscore():
    connection = getConnection()
    cursor = connection.cursor()
    cursor.execute('use score') #score is the database
    if connection.is_connected():
        try:
            cursor.execute('select * from scoreboard order by score desc')
            scores = cursor.fetchmany(10)
            try:
                cursor.fetchall()
            except:
                pass
            cursor.close()
            connection.close()
            return scores
        except Exception as e:
            print(e)
            cursor.close()
            connection.close()
            return None

    else:
        print('cannot connect to the database. please try connecting again.')
        return None

                                    #mainrun.py
'''
This is the main program executor for the maze project.
\@Author Praakhya Avasthi
\@Roll No 42
'''

import pygame, random
from maze.gamemap import GameMap #a class which draws any maze map
from arushi.gameworking import runcargame as rcg #importing the run code of the second level

pygame.init()

def runGame(): #this function is called whenever the two game levels have to be run

    map1=[                                          #this is the first game map. each letter
W is for 'wall'. if a w is read, a wall will be painted. This operation happens in another
module
        "WWWWWWWWWWWWWWWWWWWW",
        "WEEEEEEEEEEEEEEEEEEW",
        "WEWWWWWWWWWWEEWWEWEW",
        "WEEEEEWEEEEEWWEEWEWEW",
        "WEEEEEWEWWWEEWEEEEWEW",
        "WEEEEEWEWEWWEWWWWWWEW",
        "WEEEEEWEWEEEEEEEEEEEW",
        "WWWEEEWEWWWWWWWWWEWWW",
        "WEWEEEWEEEWEEEEEWEEEW",
        "WEWWWEWEEEWEWWWWWEWEW",
        "WEWEEEWEEEWEWEEEEEWEW",
```

```
        "WEWEWWWWEEWEWEWWWWWW",
        "WEWEWEEEEEWEEEEWEEEW",
        "WEWEWEWWWWWEWWEWEWWW",
        "EEWEWEEEEEEEEWEEWEWEEW",
        "WEWEWWWEWWWWWEEWEEEEE",
        "WEEEWEWEEEEEEEEEEWEEE",
        "WWWWWWWWWWWWWWWWWWWW"
        ]

    map2=[
        'W'*21,
        'WEEEEEEEEEWEEEEEEEEW',
        'WEWWWWWWWEWEWWWEWWWW',
        'WEWEEEEEWEWEWEWEEEEW',
        'WEWEEEEEWEWEWEWWEEEW',
        'WEWWWWWEWEEEWEEEEEEW',
        'WEWEEEWEWWWWWWWWWEEW',
        'WEWWWEWEWEEEWEEEEEEW',
        'WEEEWEWEEEWEWEWWWWWW',
        'WEWWWEWEWWWEWEWEEEEW',
        'WEEEEEEEWEEEWEWEWEWW',
        'WWWWWEWWWWWWWEWEWEEW',
        'WEEEEEWEEEEEEEWEWWEW',
        'WEWEWWWEWWWWWWWEWEEW',
        'EEWEWEEEWEEEWEEEWEEW',
        'WEWWWEWWWEWEWEWEWEEE',
        'WEWEEEEEEEWEEEWEWEEE',
        'W'*21
    ]

    gm1 = GameMap(map1,"maze/nightbg.jpg","maze/neontile.png", "maze/neongem.png",(255, 110,
199)) #the GameMap class will draw the maze
    gm2 = GameMap(map2,"maze/winterlandscape.png","maze/brick1x1.jpg", "maze/gems.png",
(20,20,20))

    chosenmap=random.choice([gm1, gm2]) #at every run, any one of the two mazes could be
printed for the player to play
    firstscore = chosenmap.run()
    totalscore = rcg(firstscore, chosenmap.env.futurerunning)
    return totalscore #this is the total score from bothe games


def runMaze1(): #this function is called whenever the first maze level has to be run


    map1=[
        'W'*21,
        'WEEEEEEEEEWEEEEEEEEW',
        'WEWWWWWWWEWEWWWEWWWW',
        'WEWEEEEEWEWEWEWEEEEW',
        'WEWEEEEEWEWEWEWWEEEW',
        'WEWWWWWEWEEEWEEEEEEW',
        'WEWEEEWEWWWWWWWWWEEW',
        'WEWWWEWEWEEEWEEEEEEW',
        'WEEEWEWEEEWEWEWWWWWW',
        'WEWWWEWEWWWEWEWEEEEW',
```

```
        'WEEEEEEEWEEEWEWEWEWWW',
        'WWWWWEWWWWWWEWEWEEEW',
        'WEEEEEWEEEEEEEWEWWEEW',
        'WEWEWWWEWWWWWWWEWEEEW',
        'EEWEWEEEWEEEWEEEWEEEW',
        'WEWWWEWWWEWEWEWEWEEEE',
        'WEWEEEEEEEWEEEWEWEEEE',
        'W'*21
    ]

    gm1 = GameMap(map1,"maze/winterlandscape.png","maze/brick1x1.jpg", "maze/gems.png",
(20,20,20)) #the GameMap class will draw the maze
    score = gm1.run()
    return score #this is the score from the maze map chosen

def runMaze2(): #this function is called whenever the second maze level has to be run

    map2=[
        "WWWWWWWWWWWWWWWWWWWWW",
        "WEEEEEEEEEEEEEEEEEEW",
        "WEWWWWWWWWWWEEWWEWEW",
        "WEEEEEWEEEEEWWEEWEWEW",
        "WEEEEEWEWWWEEWEEEEWEW",
        "WEEEEEWEWEWWEWWWWWWEW",
        "WEEEEEWEWEEEEEEEEEEEW",
        "WWWEEEWEWWWWWWWWWEWWW",
        "WEWEEEWEEEWEEEEEWEEEW",
        "WEWWWEWEEEWEWWWWWEWEW",
        "WEWEEEWEEEWEWEEEEEWEW",
        "WEWEWWWWEEWEWEWWWWWWW",
        "WEWEWEEEEEWEEEEWEEEEW",
        "WEWEWEWWWWWEWWEWEWWWW",
        "EEWEWEEEEEEEWEEWEWEEW",
        "WEWEWWWEWWWWEEWEEEEE",
        "WEEEWEWEEEEEEEEEEWEEE",
        "WWWWWWWWWWWWWWWWWWWWW"
        ]

    gm2 = GameMap(map2,"maze/nightbg.jpg","maze/neontile.png", "maze/neongem.png",(255, 110,
199)) #the GameMap class will draw the maze
    score = gm2.run()
    return score #this is the score from the maze map chosen

def runFlappy(): #this function is called whenever the Flappy bird level has to be run
    score = rcg(0, True)
    return score
```

#gamemap.py

```
import pygame, sys, random
from pygame.locals import *
from maze.rectangle import Rectangle
import time
from maze.environment import Environment
from maze.wall import Wall
from maze.gem import Gem
from maze.finish import Finish, Finishwall
```

```
pygame.init()

class GameMap():
    def __init__(self, map, bgimg, obstacleimg, pointimg, ballcol, score=0):
        self.cnvh = 700
        self.cnvw = 700
        self.maxx = 21
        self.maxy=18
        self.bgimg = bgimg
        self.obstacleimg = obstacleimg
        self.pointimg = pointimg
        self.ballcol = ballcol
        self.fact = min(self.cnvw//self.maxx, self.cnvh//self.maxy)
        self.env = Environment(self.cnvw, self.cnvh, self.bgimg, self.obstacleimg,
self.pointimg, self.fact, self.ballcol, score)
        self.fin = Finish(int(20*self.fact), int(15*self.fact), 1*self.fact, 2*self.fact,
self.env)
        self.finwall = Finishwall(int(20*self.fact), int(15*self.fact), 2*self.fact,
self.obstacleimg, self.env)
        #walls=[]
        self.map = map

    def run(self):
        for i in range(len(self.map)):
            for j in range(len(self.map[i])):
                if self.map[i][j]=='W':
                    self.env.addWall(Wall(j*self.fact, i*self.fact, 1*self.fact,
1*self.fact, self.env))
        print('done')
        self.env.addFinish(self.fin)
        self.env.addFinishwall(self.finwall)
        self.env.run()
        return self.env.score


                                        #ball.py
import pygame


class Ball():
    def __init__(self, x, y, radius, color, env, speed):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.env = env
        self.speed = speed

    def moveLeft(self):
        self.x = self.x-self.speed
    def moveRight(self):
        self.x = self.x+self.speed
    def moveUp(self):
        self.y = self.y-self.speed
    def moveDown(self):
        self.y = self.y+self.speed
```

```
    def draw(self):
        pygame.draw.circle(self.env.screen, self.color, (self.x, self.y), self.radius)
```

#rectangle.py

```
class Rectangle():
    def __init__(self,x1,y1,w,h):
        self.x1 =x1
        self.y1=y1
        self.x2 = x1+w
        self.y2 = y1+h

    @classmethod
    def fromCircle(self, x,y,r):
        x = x-r
        y = y-r
        w = 2*r
        h = 2*r
        return Rectangle(x,y,w,h)

    def intersect(self, rect):
        r1 = self
        r2 = rect
        if rect.x1 < self.x1:
            r1,r2=r2,r1
        return r1.x1<r2.x2 and r1.x2>r2.x1 and r1.y1<r2.y2 and r1.y2>r2.y1
```

#wall.py

```
import pygame


class Wall():
    def __init__(self, x, y, w, h,env):
        self.x =x
        self.y = y
        self.w = w
        self.h = h
        self.env = env
    def draw(self):
        tileimg = pygame.transform.scale(self.env.tileimg, (self.w, self.h))
        self.env.screen.blit(tileimg,(self.x,self.y))
```

#gem.py

```
import pygame
class Gem():
    def __init__(self, x, y, env):
        self.x = x
        self.y = y
        self.env = env
        self.w = env.gemw
        self.h = env.gemh
        self.score = 5
    def draw(self):
        self.env.screen.blit(self.env.gemimg, (self.x, self.y))
```

#finish.py

```
import pygame
from maze.rectangle import Rectangle
class Finish():
    def __init__(self, x,y,w,h,env):
        self.x =x
        self.y = y
        self.w = w
        self.h = h
        self.finimg = pygame.image.load("maze/finish.jpg")
        self.finimg = pygame.transform.scale(self.finimg,(self.w,self.h))
        self.env = env
        #self.col = (200,50,30)
        self.env = env
    def draw(self):
        #pygame.draw.rect(self.env.screen, self.col, (self.x, self.y, self.w, self.h))
        self.env.screen.blit(self.finimg,[self.x,self.y])

class Finishwall():
    def __init__(self, finx, finy, finh, wallimg, env):
        self.x = finx-10
        self.y = finy
        self.w = 10
        self.h = finh
        self.finwallimg = pygame.image.load(wallimg)
        self.finwallimg = pygame.transform.scale(self.finwallimg,(self.w,self.h))
        self.env =env
    def draw(self):
        #pygame.draw.rect(self.env.screen, self.col, (self.x, self.y, self.w, self.h))
        self.env.screen.blit(self.finwallimg,[self.x,self.y])
```

#environment.py

```
import pygame, sys, random
from pygame.locals import *
from maze.rectangle import Rectangle
import time
from maze.ball import Ball
from maze.gem import Gem
from maze.finish import Finish, Finishwall
from maze.wall import Wall
from pygame import mixer
from start.button import Button
class Environment():
    def __init__(self, w, h, background, tile, gemimg, fact, ballcol, score=0):
        pygame.font.init() # you have to call this at the start,
                        # if you want to use this module.
        self.myfont = pygame.font.SysFont('Impact', 100)
        self.w = w
        self.h = h
        self.fact = fact
        self.ballcol = ballcol
        self.b = Ball(int(0.5*self.fact), int(14.5*self.fact), 0.25*self.fact,
self.ballcol,self, 2)
        self.running = True
        self.score=score
```

```
        #self.clock = pygame.time.Clock()
        self.walls = []
        self.gems=[]
        self.gemPoint = None
        self.fin = None
        self.finwall=None
        self.gemw = 30
        self.gemh = 30
        self.wintext=None
        self.losetext=None
        self.unlock=False
        self.noGems=5
        self.gemCount = 0
        self.bgimg = pygame.image.load(background)
        self.bgimg = pygame.transform.scale(self.bgimg,(self.w,self.h))
        self.tileimg = pygame.image.load(tile)
        #self.tilevimg = pygame.image.load(tilev)
        self.gemimg = pygame.image.load(gemimg)
        self.gemimg = pygame.transform.scale(self.gemimg,(self.gemw,self.gemh))
        self.startTime = time.process_time()
        self.futurerunning = True
        mixer.init()
        self.m1 = pygame.mixer.Sound('maze/beep.wav')
        self.m2 = pygame.mixer.Sound('maze/Latch_01.wav')
        self.m1.set_volume(0.7)
        self.btnimg1 = "maze/turqoise.png"
        self.btnimg2 ="maze/btnimg2.jpg"

    def drawTime(self):
        self.elapsedTime = time.process_time() - self.startTime
        et = self.elapsedTime
        elapsedSeconds = int(et % 60)
        et = et // 60
        elapsedMinutes = int(et % 60)
        et = et // 60
        elapsedHours = int(et)
        timeString = "{:02d}:{:02d}:{:02d}".format(elapsedHours, elapsedMinutes,
elapsedSeconds)
        textsurface = self.myfont.render(timeString, False, (0, 0, 0))
        self.screen.blit(textsurface,(self.w - 400,self.h-100))

    def beep(self):
        self.m1.play(1,200)

    def unlocksound(self):
        self.m2.play(1,200)


    def run(self):
        self.spawnGems()
        self.draw()
        while self.running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                elif event.type == pygame.KEYUP and event.key == pygame.K_q:
```

```
                    self.running = False
            if self.expireGame():
                    self.losetext = 'Your Time is Up!'
                    self.running = False
                    self.futurerunning = False
                    self.beep()
            keys = pygame.key.get_pressed()
            if keys[pygame.K_RIGHT]:
                futureball = Rectangle.fromCircle(self.b.x+self.b.speed, self.b.y,
self.b.radius)
                if self.finishGame(futureball):
                    self.wintext = 'Level Success!'
                    self.running = False
                    self.beep()
                self.handleGems(futureball)
                di1, w = self.doesIntersect(futureball)
                di2 = self.doesIntersectWithFinwall(futureball)
                if  not (di1 or di2):
                    self.b.moveRight()
                else:
                    self.beep()
            elif keys[pygame.K_LEFT]:
                futureball = Rectangle.fromCircle(self.b.x-self.b.speed, self.b.y,
self.b.radius)
                self.handleGems(futureball)
                di1, w = self.doesIntersect(futureball)
                di2 = self.doesIntersectWithFinwall(futureball)
                if  not (di1 or di2):
                    self.b.moveLeft()
                else:
                    self.beep()
            elif keys[pygame.K_UP]:
                futureball = Rectangle.fromCircle(self.b.x, self.b.y-self.b.speed,
self.b.radius)
                self.handleGems(futureball)
                di1, w = self.doesIntersect(futureball)
                di2 = self.doesIntersectWithFinwall(futureball)
                if  not (di1 or di2):
                    self.b.moveUp()
                else:
                    self.beep()
            elif keys[pygame.K_DOWN]:
                futureball = Rectangle.fromCircle(self.b.x, self.b.y+self.b.speed,
self.b.radius)
                self.handleGems(futureball)
                di1, w = self.doesIntersect(futureball)
                di2 = self.doesIntersectWithFinwall(futureball)
                if  not (di1 or di2):
                    self.b.moveDown()
                else:
                    self.beep()
            self.draw()
        #self.clock.tick(60)
        #pygame.quit()

    def handleGems(self,futureball):
```

```
        gi, g = self.doesIntersectWithGem(futureball)
        if gi:
            self.score+=g.score
            self.removeGem(g)
            if len(self.gems) == 0:
                self.unlock=True
                self.unlocksound()



    def draw(self):
        self.screen = pygame.display.set_mode((self.w,self.h))
        self.screen.blit(self.bgimg,[0,0])
        for j in self.gems:
            j.draw()
        self.b.draw()
        self.fin.draw()
        if self.unlock == True:
            self.finwall.w = self.finwall.h = 0
        else:
            self.finwall.draw()
        for i in self.walls:
            i.draw()
        textsurface = self.myfont.render(str(self.score), False, (0, 0, 0))
        self.screen.blit(textsurface,(10,self.h-100))
        self.drawTime()
        if self.wintext!=None:
            '''
            pygame.draw.rect(self.screen,(0, 204, 204 ) , (self.w//2 -300, self.h//2 -100,
700, 120))
            textsurface = self.myfont.render(self.wintext, False, (0, 0, 0))
            self.screen.blit(textsurface,(self.w//2-100,self.h//2-100))
            '''
            self.windisplay =
Button(4*self.fact,self.h//2,10*self.fact,2*self.fact,self.btnimg1,
self.btnimg1,(0,0,0),self.screen,self.wintext)
            self.windisplay.draw()
            pygame.display.update()
            pygame.time.delay(1200)
        if self.losetext!=None:
            '''
            pygame.draw.rect(self.screen,(200, 0, 0 ) , (self.w//2 -300, self.h//2 -100,
700, 120))
            textsurface = self.myfont.render(self.losetext, False, (0, 0, 0))
            self.screen.blit(textsurface,(self.w//2-275,self.h//2-100))
            '''
            self.losedisplay =
Button(4*self.fact,self.h//2,10*self.fact,2*self.fact,self.btnimg2,
self.btnimg2,(0,0,0),self.screen,self.losetext)
            self.losedisplay.draw()
            pygame.display.update()
            pygame.time.delay(1200)
        pygame.display.update()

    def addWall(self, wl):
        self.walls.append(wl)
```

```
def addFinish(self, fin):
    self.fin = fin

def addFinishwall(self, finwall):
    self.finwall = finwall

def doesIntersect(self, r):
    for i in self.walls:
        rw = Rectangle(i.x, i.y, i.w, i.h)
        if rw.intersect(r):
            return True,i
    return False,None

def doesIntersectWithFinwall(self, r):
    rw = Rectangle(self.finwall.x, self.finwall.y, self.finwall.w, self.finwall.h)
    if rw.intersect(r):
        return True
    return False

def doesIntersectWithGem(self, r):
    for g in self.gems:
        rw = Rectangle(g.x, g.y, g.w, g.h)
        if rw.intersect(r):
            return True,g
    return False,None

def removeGem(self, gm):
    self.gems = [g for g in self.gems if g.x != gm.x and g.y != gm.y]

def addGem(self, gm):
    self.gemPoint=gm.score
    self.gems.append(gm)

def spawnGems(self):
    #num = random.randint(0,20)
    #if num==0: #and self.gemCount<self.maxGems:
    while self.gemCount<self.noGems:
        x,y, posfound = self.findPosition()
        if posfound:
            self.addGem(Gem(x,y,self))
            self.gemCount+=1


def findPosition(self):
    for i in range(10000):
        x= random.randint(0,15*self.fact)
        y= random.randint(0,15*self.fact)
        r = Rectangle(x,y,self.gemw, self.gemh)
        iw, w = self.doesIntersect(r)
        ig, g = self.doesIntersectWithGem(r)
        if not (iw or ig):
            return x,y,True
    return -1,-1,False

def finishGame(self, r):
```

```
        rw = Rectangle(self.fin.x, self.fin.y, self.fin.w, self.fin.h)
        if rw.intersect(r):
            self.score+=(60 - int(self.elapsedTime))*2
            return True
        return False

    def expireGame(self):
        if self.elapsedTime>=60:
            return True
        return False
```

#gameworking.py

```
#LEVEL 2
# Creating an interactive game where user has to avoid obstacles, either by going over them
or by ducking under them and collect gems along the way

import sys # Imported to access system commands
import pygame
from pygame.locals import * # Basic pygame imports
import random # For generating random numbers


# Global Variables for the game
FPS = 32 #Frames per second
SCREENWIDTH = 700
SCREENHEIGHT = 700
GROUND_Y = SCREENHEIGHT * 0.75
IMAGES = {}
SOUND = {}
REWARD_POINTS = 5
SCREENCAPTION = 'Game 2'
score = 0
icScore = 0
#Images
PLAYER = 'arushi/gallery/car.png'
BACKGROUND = 'arushi/gallery/background.jpg'
OBSTACLE = 'arushi/gallery/obstacle.png'
REWARD = 'arushi/gallery/diamond.png'


#Initial blank screen
SCREEN = pygame.display.set_mode((SCREENWIDTH, SCREENHEIGHT))

def startScreen():
    #instructions and game2 image
    #Position of car-middle of screen
    global FPSCLOCK
    global FPS
    playerX = int((SCREENWIDTH - IMAGES['player'].get_width())/2)
    playerY = int((SCREENHEIGHT - IMAGES['player'].get_height())/2)
    messageX = int((SCREENWIDTH - IMAGES['message'].get_width())/2)
    messageY = int(SCREENHEIGHT*0.15)

    basex = 0
    SCREEN.blit(IMAGES['background'], (0, 0))
    SCREEN.blit(IMAGES['player'], (playerX, playerY))
```

```
    SCREEN.blit(IMAGES['message'], (messageX,messageY ))
    SCREEN.blit(IMAGES['base'], (basex, GROUND_Y))
    pygame.display.update()
    FPSCLOCK.tick(FPS)

    SCREEN.blit(IMAGES['background'], (0, 0))
    SCREEN.blit(IMAGES['player'], (playerX, playerY))
    SCREEN.blit(IMAGES['message'], (messageX,messageY ))
    SCREEN.blit(IMAGES['base'], (basex, GROUND_Y))
    pygame.display.update()
    FPSCLOCK.tick(FPS)
    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): #Esc
or crossed
                print("GAME QUIT")
                return False

            # 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):
                print("Starting Game")
                return True

            elif event.type == KEYDOWN and event.key == K_i:
                print("Instructions")
                return True


            else:
                pass

def mainGame(running):
    global FPS
    global score
    playerX = int(SCREENWIDTH/5)
    playerY = int(SCREENWIDTH/2)
    basex = 0

    # Create 2 obstacles for blitting on the screen
    newObstacle1 = getRandomObstacle()
    newObstacle2 = getRandomObstacle()

    # List of upper obstacles
    upperObstacles = [
        {'x': SCREENWIDTH+200, 'y':newObstacle1[0]['y']},
        {'x': SCREENWIDTH+200+(SCREENWIDTH/2), 'y':newObstacle2[0]['y']},
    ]
    # my List of lower obstacles
    lowerObstacles = [
        {'x': SCREENWIDTH+200, 'y':newObstacle1[1]['y']},
        {'x': SCREENWIDTH+200+(SCREENWIDTH/2), 'y':newObstacle2[1]['y']},
    ]

    # List of rewards
    rewards = [
```

```
        {'x': SCREENWIDTH+random.randint(0, 50), 'y': random.randint(0, SCREENHEIGHT-30)}, #
Rewards comes at extreme right
        ]

    obstacleVelX = -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 running:
        for event in pygame.event.get():
            if event.type == QUIT or (event.type == KEYDOWN and event.key == K_ESCAPE):
                return
            if event.type == KEYDOWN and (event.key == K_SPACE or event.key == K_UP):
                if playerY > 0:
                    playerVelY = playerFlapAccv
                    playerFlapped = True
                    SOUND['wing'].play()


        if hitObstacle(playerX, playerY, upperObstacles, lowerObstacles):  # True if player
collided with any obstacle
            # Crash happened. Display the score and wait for user input
            print("Game Over")
            while True:
                for event in pygame.event.get():
                    if event.type == QUIT or (event.type == KEYDOWN and event.key ==
K_ESCAPE):
                        return
                    if event.type == KEYDOWN and (event.key == K_RETURN):
                        return

        # Update score
        playerMidPos = playerX + IMAGES['player'].get_width()/2
        for obstacle in upperObstacles:
            obstacleMidPos = obstacle['x'] + IMAGES['obstacle'][0].get_width()/2
            if obstacleMidPos <= playerMidPos < obstacleMidPos +4:  # Crossed the obstacle
                score +=1
                print(f"Your score is {score}")
                SOUND['point'].play()

        # Check if reward collected
        if rewardCollected(playerX, playerY, rewards):
            print("reward collected")
            score += REWARD_POINTS
            SOUND['point'].play()
            # Remove this reward as already claimed. Add a new reward
            rewards = [
                {'x': SCREENWIDTH+random.randint(0, 50), 'y': random.randint(0,
SCREENHEIGHT-30)}, # Rewards enter at extreme right
```

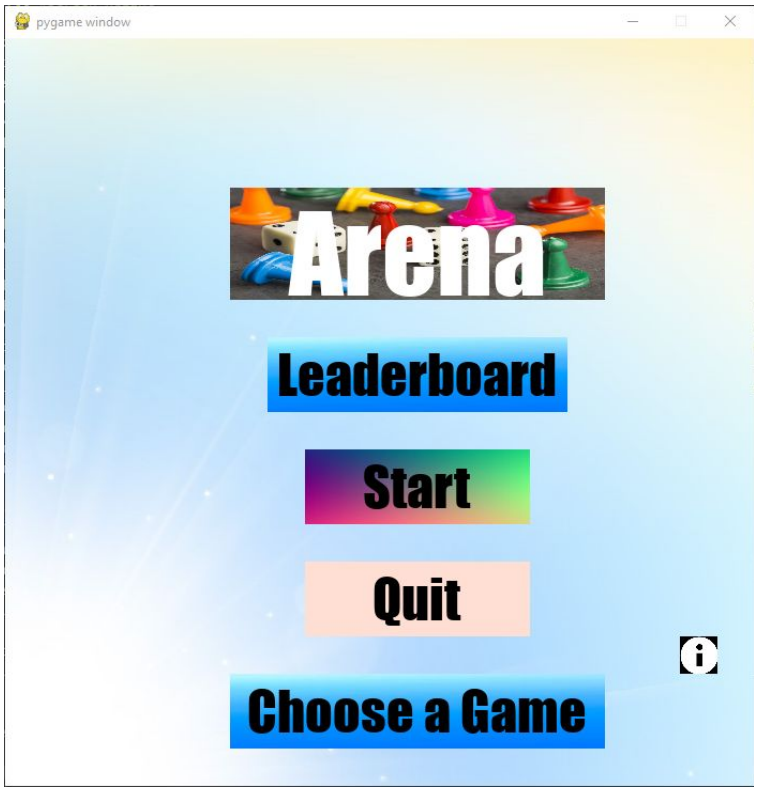
```
                ]

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

        if playerFlapped:
            playerFlapped = False
        playerHeight = IMAGES['player'].get_height()
        playerY = playerY + min(playerVelY, GROUND_Y - playerY - playerHeight)

        # move obstacles to the left
        for upperObstacle , lowerObstacle in zip(upperObstacles, lowerObstacles):
            upperObstacle['x'] += obstacleVelX
            lowerObstacle['x'] += obstacleVelX

        for reward in rewards:
            reward['x'] += obstacleVelX


        # Add a new obstacle when the first is about to cross the leftmost part of the
screen
        if 0<upperObstacles[0]['x']<5:
            newobstacle = getRandomObstacle()
            upperObstacles.append(newobstacle[0])
            lowerObstacles.append(newobstacle[1])

        # if the obstacle is out of the screen, remove it
        if upperObstacles[0]['x'] < -IMAGES['obstacle'][0].get_width():
            upperObstacles.pop(0)
            lowerObstacles.pop(0)

        # if the reward is out of the screen, add a new one
        if rewards[0]['x'] < -IMAGES['rewardImage'].get_width():
            rewards = [
                {'x': SCREENWIDTH+random.randint(0, 50), 'y': random.randint(0,
SCREENHEIGHT-30)}, # Rewards enter at extreme right
                ]

        # Update the screen now
        SCREEN.blit(IMAGES['background'], (0, 0))
        for upperObstacle, lowerObstacle, in zip(upperObstacles, lowerObstacles):
            SCREEN.blit(IMAGES['obstacle'][0], (upperObstacle['x'], upperObstacle['y']))
            SCREEN.blit(IMAGES['obstacle'][1], (lowerObstacle['x'], lowerObstacle['y']))

        for reward in rewards:
            SCREEN.blit(IMAGES['rewardImage'], (reward['x'], reward['y']))

        SCREEN.blit(IMAGES['base'], (basex, GROUND_Y))
        SCREEN.blit(IMAGES['player'], (playerX, playerY))
        myDigits = [int(x) for x in list(str(score))]
        width = 0
        for digit in myDigits:
            width += IMAGES['numbers'][digit].get_width()
        Xoffset = (SCREENWIDTH - width - 10)

        for digit in myDigits:
```

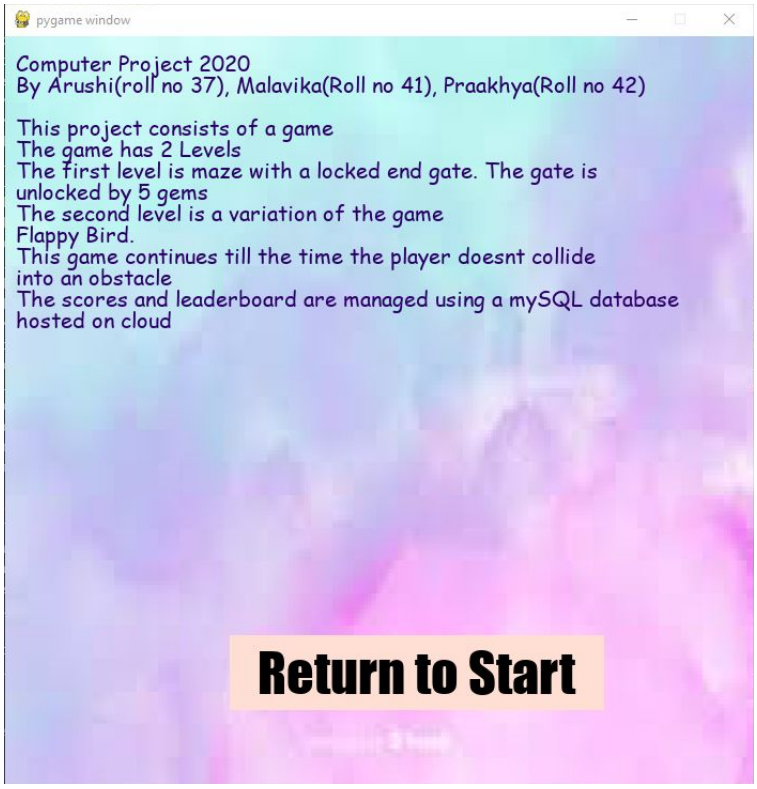
```
            SCREEN.blit(IMAGES['numbers'][digit], (Xoffset, SCREENHEIGHT*0.9))
            Xoffset += IMAGES['numbers'][digit].get_width()
        pygame.display.update()
        FPSCLOCK.tick(FPS+(score - icScore)/2)

def hitObstacle(playerX, playerY, upperObstacles, lowerObstacles):
    if playerY > (GROUND_Y - 25)  or playerY < 0:
        SOUND['hit'].play()
        return True

    for obstacle in upperObstacles:
        obstacleHeight = IMAGES['obstacle'][0].get_height()
        if(playerY < obstacleHeight + obstacle['y'] and abs(playerX - obstacle['x']) <
IMAGES['obstacle'][0].get_width()):
            SOUND['hit'].play()
            return True

    for obstacle in lowerObstacles:
        if (playerY + IMAGES['player'].get_height() > obstacle['y']) and abs(playerX -
obstacle['x']) < IMAGES['obstacle'][0].get_width():
            SOUND['hit'].play()
            return True

    return False

def rewardCollected(playerX, playerY, rewards):

    for reward in rewards:
        obstacleHeight = IMAGES['obstacle'][0].get_height()
        if(playerY > reward['y'] and (playerY <
reward['y']+IMAGES['rewardImage'].get_height()) and abs(playerX - reward['x']) <
IMAGES['rewardImage'].get_width()):
            return True
    return False

def getRandomObstacle():
    """
    Generate positions of two obstacles(one bottom straight and one top rotated ) for
blitting on the screen
    """
    obstacleHeight = IMAGES['obstacle'][0].get_height()
    offset = SCREENHEIGHT/3
    y2 = offset + random.randrange(0, int(SCREENHEIGHT - IMAGES['base'].get_height()  - 1.2
*offset))
    obstacleX = SCREENWIDTH + 10
    y1 = obstacleHeight - y2 + offset
    obstacle = [
        {'x': obstacleX, 'y': -y1}, #upper Obstacle
        {'x': obstacleX, 'y': y2} #lower Obstacle
    ]
    return obstacle

def runcargame(incomingscore=0, running=True):
    global SCREEN
    global FPSCLOCK
    global FPS
```

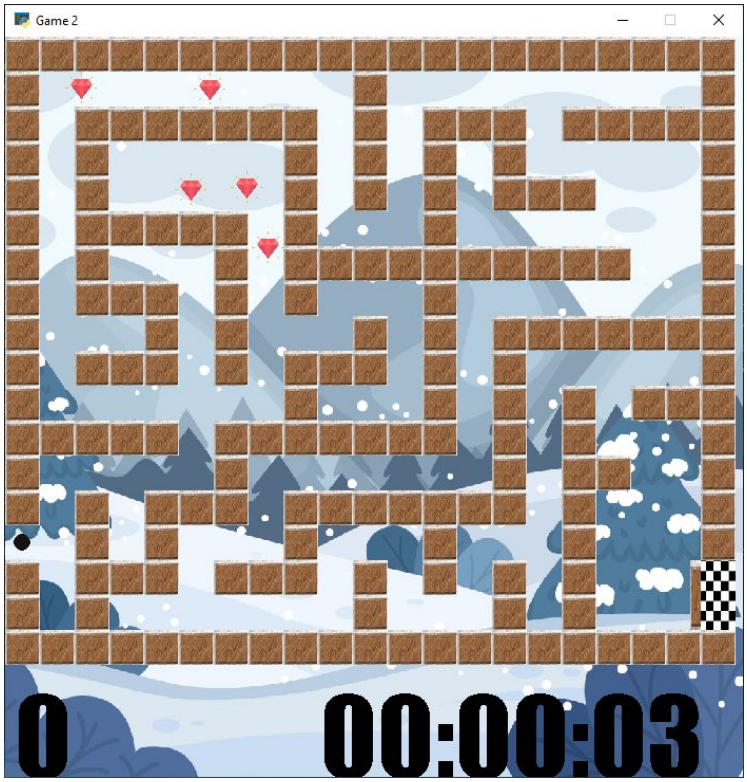
```
    global score
    global icScore
    score = incomingscore
    icScore = incomingscore
    if not running:
        return score
    SCREEN= pygame.display.set_mode((SCREENWIDTH, SCREENHEIGHT))
    # This will be the main point from where our game will start
    pygame.init() # Initialize all pygame's modules
    FPSCLOCK = pygame.time.Clock()
    pygame.display.set_caption(SCREENCAPTION)
    IMAGES['numbers'] = (
        pygame.image.load('arushi/gallery/images/0.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/1.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/2.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/3.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/4.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/5.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/6.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/7.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/8.png').convert_alpha(),
        pygame.image.load('arushi/gallery/images/9.png').convert_alpha(),
    )

    IMAGES['message'] =pygame.image.load('arushi/gallery/firstScreen.png').convert_alpha()
    IMAGES['base']
=pygame.transform.scale(pygame.image.load('arushi/gallery/base.png').convert_alpha(),(SCREEN
WIDTH,300))
    IMAGES['background'] = pygame.image.load(BACKGROUND).convert()
    IMAGES['player'] = pygame.image.load(PLAYER).convert_alpha()
    IMAGES['rewardImage'] = pygame.image.load(REWARD).convert_alpha()
    IMAGES['obstacle'] =(pygame.transform.rotate(pygame.image.load(
OBSTACLE).convert_alpha(), 180),
    pygame.image.load(OBSTACLE).convert_alpha()
    )

    # Game sounds
    SOUND['die'] = pygame.mixer.Sound('arushi/gallery/audio/die.wav')
    SOUND['hit'] = pygame.mixer.Sound('arushi/gallery/audio/hit.wav')
    SOUND['point'] = pygame.mixer.Sound('arushi/gallery/audio/point.wav')
    SOUND['swoosh'] = pygame.mixer.Sound('arushi/gallery/audio/swoosh.wav')
    SOUND['wing'] = pygame.mixer.Sound('arushi/gallery/audio/wing.wav')


    #The program is run
    # Shows welcome screen to the user until a button is pressed
    # This is the main game function
    mainGame(startScreen())
    return score


if __name__ == "__main__":
    runcargame()
```

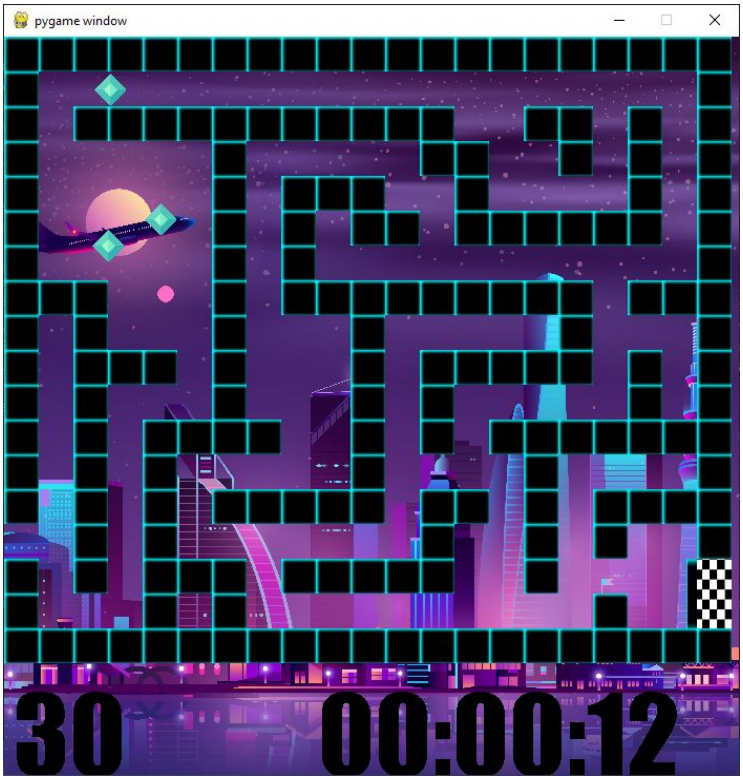

Screen
shots

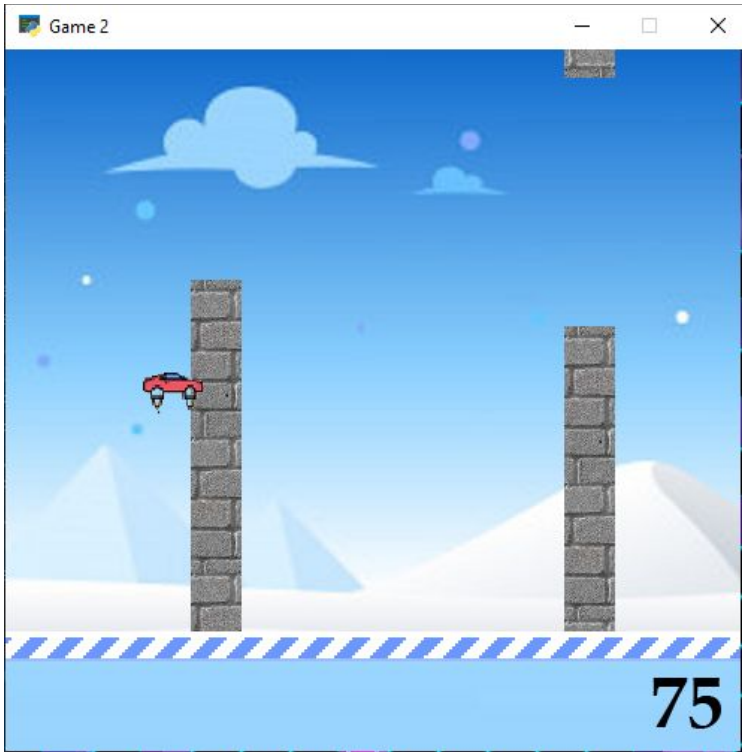
Screen Shots

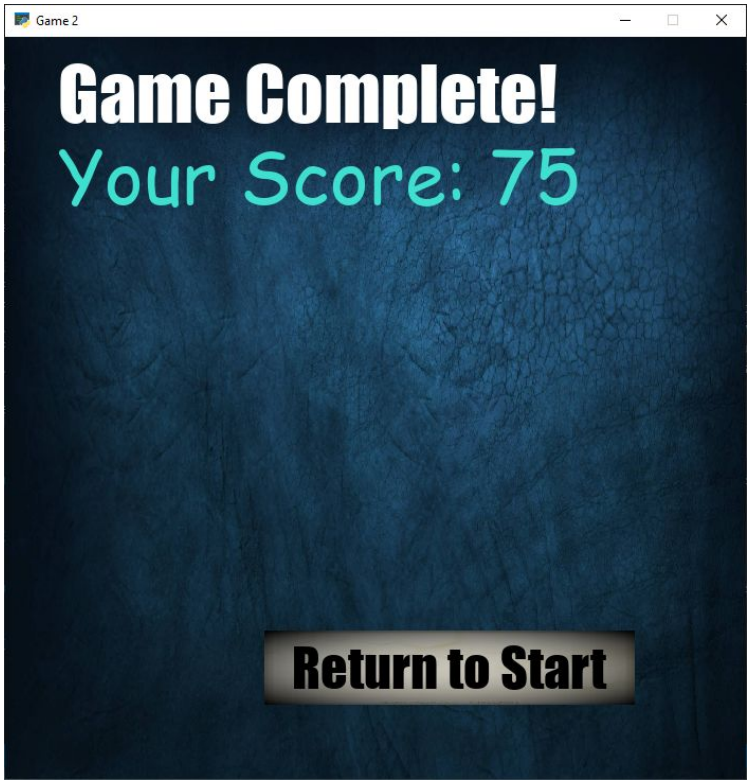





Game 2
0
00:00:03


pygame window
30
00:00:12


Game 2
75


Game 2
Game Complete!
Your Score: 75
Return to Start

Bibliography

Dharmkar, Rajendra. “How to insert date object in MySQL using Python?” 8 January 2018, https://www.tutorialspoint.com/How-to-insert-date-object-in-MySQL-using-Python#:~:text=To%20insert%20a%20date%20in,datetime%20module's%20strftime%20formatting%20function.

“Project 2: Coding Flappy Bird Game (With Source Code) | Python Tutorials For Absolute Beginners #122.” June 2019, https://youtu.be/itB6VsP5UnA.

“Pygame Documentation.” https://www.pygame.org/docs/.

“Pygame Tutorial for Beginners - Python Game Development Course.” October 2019, https://youtu.be/FfWpgLFMI7w.