A

PROGRAMMING PROJECT REPORT ON



“A 2D PLATFORMER GAME”

SUBMITTED IN THE REQUIREMENTS OF

B.A. (PROG.)-FINAL YEAR

**P.G.D.A.V COLLEGE**

**NEHRU NAGAR, NEW DELHI**

**SUBMITTED BY**

AYUSH JHA

EXAMINATIONS ROLL NO: 21053501233

SESSION: 2021 - 2024

JAYATI TREHAN

EXAMINATIONS ROLL NO: 21053501150

SESSION: 2021 - 2024

SAMYAK JAIN

EXAMINATIONS ROLL NO: 21053501313

SESSION: 2021 - 2024

TABLE OF CONTENTS

 CERTIFICATE 4

ACKNOWLEDGEMENT 5

 6

INTRODUCTION

 7

CODE OF GAME

* assets.py
* settings.py
* button.py
* attack.py
* objects.py
* entities.py
* game.py



OUTPUT SCREEN



BIBLIOGRAPHY

**TO WHOM SO EVER IT MAY CONCERN**

This is to certify that “Ayush Jha”, “Jayati Trehan” and “Samyak Jain” students of B.A. (PROG.) final year (2021-2024) at P.G.D.A.V. COLLEGE have completed the final programming project report of “PokeMario a 2D platformer game” under the guidance of the undersigned in the fulfillment of B.A. (PROG.) course.

**DR. GEETA GUPTA**

**PGDAV**

**NEW DELHI**

**ACKNOWLEDGEMENTS**

We are grateful for the opportunity to express our appreciation to those who played a role in the successful completion of this project.

First and foremost, we extend our sincere thanks to our professor, Dr. Geeta Gupta, whose guidance and support throughout this journey were invaluable. Their insights and feedback were instrumental in shaping the direction of our project and ensuring its quality.

We are incredibly fortunate to have worked alongside such a dedicated and talented group of peers. To our fellow group members, we express our deepest gratitude. Each of you brought unique strengths and perspectives to the table, fostering a truly collaborative environment. Your hard work, positive attitudes, and willingness to go the extra mile were essential in achieving our goals.

**Introduction**

Pokemario is a 2D platform game, inspired by the old “super mario” games, and based on the popular anime series “Pokemon”.

There are two stages in the game, and to clear each stage, all present enemies must be defeated by the player.

We have used python modules **pygame** and **time** to create our game.

pygame is a python module which can be used to make simple 2D games, and time module is used for various time related functions, like getting the exact current time etc.

The game has been completed as per our intentions, and with very minor and rare bugs, almost nothing is left undone.

We would like to mention that sprite animations of “Raichu” remain incomplete, also various sound effects could not have been implemented in the given time.

**CODE OF GAME**

We have created many python files to make it easier to read and navigate through.

Then, all the files have been imported to the main “game.py” file and that’s how the game works.

Showcasing each file one by one:

**assets.py**

import pygame as pg

from pygame import mixer

from settings import \*

from os import listdir

from os.path import join,isfile

mixer.init()

pg.init()

win\_logo = pg.image.load(join('assets','window\_assets','pika.png')) #Loading window logo

main\_menu\_bg = pg.transform.scale(pg.image.load(join('assets','main\_menu\_assets','mario-game.jpg')),(1600,900)) #main menu background

lost\_screen\_bg = pg.image.load(join('assets','lostscreen\_assets','lost2.png'))

win\_screen\_bg = pg.image.load(join('assets','winscreen\_assets','won3.png'))

# button images

on = pg.transform.scale(pg.image.load(join('assets', 'main\_menu\_assets', 's\_on.png')),(250,250))

off = pg.transform.scale(pg.image.load(join('assets', 'main\_menu\_assets', 's\_off.png')),(250,250))

play\_img = pg.transform.scale(pg.image.load(join('assets', 'main\_menu\_assets', 'play\_btn.png')),(200,108))

instructions\_img = pg.transform.scale(pg.image.load(join('assets', 'main\_menu\_assets', 'ins\_btn.png')),(500,100))

credits\_img = pg.transform.scale(pg.image.load(join('assets', 'main\_menu\_assets', 'credits\_btn.png')),(300,108))

ques\_btn = pg.transform.scale(pg.image.load(join('assets','main\_menu\_assets','ques\_button1.png')),(64,64))

play\_again\_img = pg.transform.scale(pg.image.load(join('assets', 'main\_menu\_assets', 'play\_again\_btn.png')),(375,75))

return\_img = pg.transform.scale(pg.image.load(join('assets', 'main\_menu\_assets', 'return\_btn.png')),(337.5,75))

quit\_img = pg.transform.scale(pg.image.load(join('assets', 'main\_menu\_assets', 'quit\_btn.png')),(150,75))

back\_img = pg.transform.scale(pg.image.load(join('assets', 'instructions\_assets', 'back.png')),(128,64))

# title logo

title\_img = pg.image.load(join('assets', 'main\_menu\_assets', 'pokemario.png'))

instructions\_title = pg.image.load(join('assets', 'instructions\_assets', 'INSTRUCTIONS-button.png'))

credits\_title = pg.image.load(join('assets', 'credits\_assets', 'credits.png'))

credits\_title = pg.transform.scale(credits\_title,(850,100)) # rescaling

won\_img = pg.image.load(join('assets', 'winscreen\_assets', 'won.png'))

won\_img = pg.transform.scale(won\_img,(600,400))

lost\_img = pg.image.load(join('assets', 'lostscreen\_assets', 'lose1.png'))

lost\_img = pg.transform.scale(lost\_img,(600,400))

# levels

lvl\_1 = pg.image.load(join('assets', 'BACKGROUNDS', 'evening.jpg'))

lvl\_2 = pg.image.load(join('assets', 'BACKGROUNDS','night.png'))

# objects

platform = pg.image.load(join('assets', 'tileset','GRASSLAND', 'ONE BIG TILE.png'))

# door

door\_open = pg.transform.scale(pg.image.load(join('assets', 'door\_assets', 'Door open.png')),door\_dims)

door\_closed = pg.transform.scale(pg.image.load(join('assets', 'door\_assets', 'Door closed.png')),door\_dims)

# music

mixer.music.load(join('assets','bg\_music','bg.mp3'))

mixer.music.set\_volume(0.5)

# txt files

file1 = open("Docs/INSTRUCTIONS.txt",'r')

file2 = open("Docs/CREDITS.txt",'r')

instructions\_txt = file1.read()

credits\_txt = file2.read()

file1.close()

file2.close()

# fonts

credits\_font = pg.font.Font(join('assets','retro\_computer\_personal\_use.ttf'), 27)

ins\_font = pg.font.Font(join('assets','retro\_computer\_personal\_use.ttf'), 19)

ingame\_font = pg.font.Font(join('assets','retro\_computer\_personal\_use.ttf'), 35)

lastscreen\_font = pg.font.Font(join('assets','retro\_computer\_personal\_use.ttf'), 40)

# player

pikachu\_face = pg.transform.scale(pg.image.load(join('assets','pikachu face.png')),(64,64))

Pikachu = {'idle\_left':[pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','idle pikachu','1.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','idle pikachu','2.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','idle pikachu','3.png')),pikachu\_dims),True,False)],'idle\_right':[pg.transform.scale(pg.image.load(join('assets','Pikachu','idle pikachu','1.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','idle pikachu','2.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','idle pikachu','3.png')),pikachu\_dims)],'run\_left' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','running sprite','1.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','running sprite','2.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','running sprite','3.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','running sprite','4.png')),pikachu\_dims),True,False)],'run\_right' : [pg.transform.scale(pg.image.load(join('assets','Pikachu','running sprite','1.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','running sprite','2.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','running sprite','3.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','running sprite','4.png')),pikachu\_dims)],'jump\_left':[pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Jump','Jump.png')),pikachu\_dims),True,False)],'jump\_right':[pg.transform.scale(pg.image.load(join('assets','Pikachu','Jump','Jump.png')),pikachu\_dims)],'fall\_left':[pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Jump','Jump.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Jump','Jump.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Jump','Jump.png')),pikachu\_dims),True,False)],'fall\_right':[pg.transform.scale(pg.image.load(join('assets','Pikachu','Jump','Jump.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','Jump','Jump.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','Jump','Jump.png')),pikachu\_dims)],'attack\_left' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','1.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','2.png')),pikachu\_dims),True,False)], 'attack\_right' : [pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','1.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','2.png')),pikachu\_dims)]}

pikachu\_bolt\_right = pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','lightning bolt.png')),pikabolt\_dims)

pikachu\_bolt\_left = pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','lightning bolt.png')),pikabolt\_dims),True,False)

raichu = {'run\_left' : [pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','1.png')),raichu\_dims),pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','2.png')),raichu\_dims),pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','3.png')),raichu\_dims),pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','4.png')),raichu\_dims),pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','5.png')),raichu\_dims)], 'run\_right' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','1.png')),raichu\_dims),True,False),pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','2.png')),raichu\_dims),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','3.png')),raichu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','4.png')),raichu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','5.png')),raichu\_dims),True,False)], 'fall\_right' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','1.png')),raichu\_dims),True,False)], 'fall\_left' : [pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','1.png')),raichu\_dims)],'jump\_right' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','1.png')),raichu\_dims),True,False)], 'jump\_left' : [pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','1.png')),raichu\_dims)],'idle\_right' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','1.png')),raichu\_dims),True,False)], 'idle\_left' : [pg.transform.scale(pg.image.load(join('assets','Raichu','Raichu running','1.png')),raichu\_dims)],'attack\_left' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','1.png')),pikachu\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','2.png')),pikachu\_dims),True,False)], 'attack\_right' : [pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','1.png')),pikachu\_dims),pg.transform.scale(pg.image.load(join('assets','Pikachu','Attack','2.png')),pikachu\_dims)]}

# enemy

pidgeott = {'run\_left' : [pg.transform.scale(pg.image.load(join('assets','Pidgeott','1.png')),pidgeott\_dims),pg.transform.scale(pg.image.load(join('assets','Pidgeott','2.png')),pidgeott\_dims),pg.transform.scale(pg.image.load(join('assets','Pidgeott','3.png')),pidgeott\_dims)],'run\_right' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pidgeott','1.png')),pidgeott\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pidgeott','2.png')),pidgeott\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Pidgeott','3.png')),pidgeott\_dims),True,False)]}

golbat = {'run\_left' : [pg.transform.scale(pg.image.load(join('assets','golbat','1.png')),golbat\_dims),pg.transform.scale(pg.image.load(join('assets','golbat','2.png')),golbat\_dims),pg.transform.scale(pg.image.load(join('assets','golbat','3.png')),golbat\_dims),pg.transform.scale(pg.image.load(join('assets','golbat','4.png')),golbat\_dims)],'run\_right' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','golbat','1.png')),golbat\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','golbat','2.png')),golbat\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','golbat','3.png')),golbat\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','golbat','4.png')),golbat\_dims),True,False)]}

haunter = {'run\_left' : [pg.transform.scale(pg.image.load(join('assets','Haunterr','front.png')),haunter\_dims)],'run\_right' : [pg.transform.scale(pg.image.load(join('assets','Haunterr','back.png')),haunter\_dims)]}

bulbasaur = {'run\_right' : [pg.transform.scale(pg.image.load(join('assets','Bulbasaur','3.png')),pidgeott\_dims),pg.transform.scale(pg.image.load(join('assets','Bulbasaur','4.png')),pidgeott\_dims),pg.transform.scale(pg.image.load(join('assets','Bulbasaur','5.png')),pidgeott\_dims)],'run\_left' : [pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Bulbasaur','3.png')),pidgeott\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Bulbasaur','4.png')),pidgeott\_dims),True,False),pg.transform.flip(pg.transform.scale(pg.image.load(join('assets','Bulbasaur','5.png')),pidgeott\_dims),True,False)]}

bulb\_attack = pg.image.load(join('assets','Bulbasaur','Projectile','projectile 1.png'))

enemy\_attack = pg.image.load(join('assets', 'player\_assets', 'bullet.png'))

# traps

charmander\_trap = [pg.transform.scale(pg.image.load(join('assets','Charmander','1.png')),charmander\_dims),pg.transform.scale(pg.image.load(join('assets','Charmander','2.png')),charmander\_dims),pg.transform.scale(pg.image.load(join('assets','Charmander','3.png')),charmander\_dims),pg.transform.scale(pg.image.load(join('assets','Charmander','4.png')),charmander\_dims)]

dugtrio\_trap = [pg.transform.scale(pg.image.load(join('assets','traps\_assets','Dugtrio\_(Sprite\_1).png')),dugtrio\_dims)]

# text functioning

def print\_text\_decor(func):

def inner(text,font,x,y,screen,text\_color = (255,255,255)):

if isinstance(text,list): # to handle list inputs differently

i = 0

for line in text:

img = font.render(line,True,text\_color)

screen.blit(img,(x,y+i))

i += 50 # line spacing

else:

func(text,font,x,y,screen,text\_color = (255,255,255)) # if not list then just blit normally

return inner

@print\_text\_decor

def print\_text(text,font,x,y,screen,text\_color = (255,255,255)):

img = font.render(text, True, text\_color)

screen.blit(img,(x,y))

**settings.py**

win\_x, win\_y = 1600,900 # width,height

title = "Pokémario" # window Title

fps = 45 # sets frame rate to 60

state\_list = ['main','ins','credits','L','W','lvl1','lvl2'] # different game states

state = state\_list[0] # current game state (initial is 'main')

sound\_state = False # sounds off by default

scroll = 0 #background scroll variable

running = True # game loop run

doorBool = False # if door is open or not

door\_dims = (96,96) # dimensions of the door

timer\_running = False # if we want to show the timer

start\_timer = False # to record the start time

stop\_timer = False

seconds = mins = hours = 0

start\_time = 0

start\_game = False

enemy\_projectile\_speed = 10

charmander\_dims = (64,32)

pidgeott\_dims = (64,64)

golbat\_dims = (150,150)

haunter\_dims = (96,96)

dugtrio\_dims = (76,64)

pikachu\_dims = (64,64)

pikabolt\_dims = (60,50)

raichu\_dims = (96,96)

player\_speed = 5

trigger\_list = []

**button.py**

import pygame as pg

from assets import \*

from settings import \*

class Button:

def \_\_init\_\_(self,x,y,image,scale = 1):

self.width = image.get\_width()

self.height = image.get\_height()

# rescaling the button to the desired size

self.image = pg.transform.scale(image, (int(self.width \* scale),int(self.height \* scale)))

self.rect = self.image.get\_rect()

self.rect.x, self.rect.y = x, y

self.clicked = False # is the button clicked?

def draw(self,surface):

action = False # is there a need to do the action?

pos = pg.mouse.get\_pos() # getting mouse position

if self.rect.collidepoint(pos): # collides with mouse pointer

# left mouse button pressed while the button wasn't already pressed

if pg.mouse.get\_pressed()[0] == 1 and self.clicked == False:

self.clicked == True # the button is pressed

action = True # perform the action

if pg.mouse.get\_pressed()[0] == 0: # if the left mouse button gets unpressed

self.clicked == False # button no longer pressed

surface.blit(self.image,(self.rect.x,self.rect.y))

return action # performs action if button clicked else does nothing

# creating buttons

sfx\_on = Button(1530,10,on,0.25)

sfx\_off = Button(1530,10,off,0.25)

play\_btn = Button(600,474,play\_img,0.8)

instructions\_btn = Button(600,600,instructions\_img,0.8)

credits\_btn = Button(600,724,credits\_img,0.8)

back\_btn = Button(15,10,back\_img)

play\_again\_btn = Button(144,97,play\_again\_img)

main\_menu\_btn = Button(144,392,return\_img)

quit\_btn = Button(144,685,quit\_img)

**attack.py**

import pygame as pg

from settings import \*

class Attack(pg.sprite.Sprite):

enemy\_attacks = [] # list of all Enemy Attack class objects

player\_attack = [] # list of all Player Attack class objects

def \_\_init\_\_(self, x, y, direction, image, range, speed, player = False):

super().\_\_init\_\_()

self.x = self.initial\_x = x

self.y = y

self.direction = direction

self.velocity = speed # speed of the projectile

self.image = image

self.rect = self.image.get\_rect()

self.rect.topleft = (self.x,self.y)

self.range = range # number of blocks the projectile travels

self.player = player # if the projectile is of our player or the enemy

self.active = False

self.rem = True

if self.player == False:

Attack.enemy\_attacks.append(self)

else:

Attack.player\_attack.append(self)

def shoot(self,screen):

# blits the object on the screen and deals damage

if self.direction == 'left':

if self.rect.x >= (self.initial\_x - self.range):

self.rem = True

self.active = True

screen.blit(self.image,(self.rect.x,self.rect.y))

self.rect.x -= self.velocity

else:

self.active = False

if self.rem:

if not(self.player):

Attack.enemy\_attacks.remove(self)

else:

Attack.player\_attack.remove(self)

self.rem = False

elif self.direction == 'right':

if self.rect.x <= self.initial\_x + self.range:

self.rem = True

self.active = True

screen.blit(self.image,(self.rect.x,self.rect.y))

self.rect.x += self.velocity

else:

self.active = False

if self.rem:

if not(self.player):

Attack.enemy\_attacks.remove(self)

else:

Attack.player\_attack.remove(self)

self.rem = False

**objects.py**

import pygame as pg

from settings import \*

from assets import \*

class Object(pg.sprite.Sprite):

def \_\_init\_\_(self, x, y, width, height):

super().\_\_init\_\_()

self.x = x

self.y = y

self.width = width

self.height = height

self.active = False

def draw(self, screen):

# draws the objects on the screen

self.active = True

screen.blit(self.image, (self.rect.x,self.rect.y))

class Block(Object): # inheriting Obejct class

objects\_list = []

def \_\_init\_\_(self, x, y, width, height, path):

super().\_\_init\_\_(x, y, width, height)

self.image = pg.image.load(path)

self.image = pg.transform.scale(self.image,(width,height))

self.rect = self.image.get\_rect()

self.rect.topleft = (self.x,self.y)

self.mask = pg.mask.from\_surface(self.image)

Block.objects\_list.append(self)

class Trap(Object): # inheriting Obejct class

objects\_list = []

def \_\_init\_\_(self, x, y, width, height, sprite):

super().\_\_init\_\_(x, y, width, height)

Trap.objects\_list.append(self)

self.velocity = -1

self.tick = 0

self.spritesheet = sprite

self.sprite\_index = 0

self.image = self.spritesheet[self.sprite\_index]

self.rect = self.image.get\_rect()

self.rect.topleft = (self.x,self.y)

self.mask = pg.mask.from\_surface(self.image)

self.initial\_y = self.rect.y

self.up = True

self.down = False

def update\_sprite(self):

if self.tick % 18 == 0:

try:

self.sprite\_index += 1

self.image = self.spritesheet[self.sprite\_index]

except IndexError:

self.sprite\_index = 0

self.image = self.spritesheet[self.sprite\_index]

def move(self):

if self.up:

if self.rect.y >= self.initial\_y - self.height + 10:

self.rect.y += self.velocity

else:

self.up = False

self.down = True

if self.down:

if self.rect.y <= self.initial\_y:

self.rect.y -= self.velocity

else:

self.up = True

self.down = False

def draw(self, screen):

# draws the objects on the screen

self.active = True

self.tick += 1

self.update\_sprite()

screen.blit(self.image, (self.rect.x,self.rect.y))

class Evolve(Object):

objects\_list = []

def \_\_init\_\_(self, x, y, width, height, path):

super().\_\_init\_\_(x, y, width, height)

self.image = pg.image.load(path)

self.image = pg.transform.scale(self.image,(width,height))

self.rect = self.image.get\_rect()

self.rect.topleft = (self.x,self.y)

self.mask = pg.mask.from\_surface(self.image)

Evolve.objects\_list.append(self)

self.display = True

def draw(self,screen):

if self.display:

self.active = True

screen.blit(self.image,(self.rect.x,self.rect.y))

# creating objects

# level 1

lvl1\_block\_size = 200

lvl1\_floor = [Block(i,win\_y - lvl1\_block\_size,lvl1\_block\_size,lvl1\_block\_size,join('assets', 'tileset','GRASSLAND', 'TILE1.png')) for i in range(0,200,lvl1\_block\_size)] # list of block objects in line, which look like a floor

lvl1\_floor\_end = Block(lvl1\_block\_size, win\_y - lvl1\_block\_size+1,lvl1\_block\_size/2,lvl1\_block\_size,join('assets', 'tileset','GRASSLAND', 'TILE2.png'))

lvl1\_floating = Block(450,win\_y - (2\*lvl1\_block\_size) + 20,lvl1\_block\_size\*1.5,lvl1\_block\_size,join('assets', 'tileset','GRASSLAND', 'FLOATING TILE.png'))

lvl1\_bigtile = Block(850,win\_y - (3\*lvl1\_block\_size) + 20,lvl1\_block\_size\*2,lvl1\_block\_size\*4,join('assets', 'tileset','GRASSLAND', 'ONE BIG TILE 2.png'))

lvl1\_endtile = Block(win\_x-lvl1\_block\_size,win\_y - lvl1\_block\_size,lvl1\_block\_size\*1.3,lvl1\_block\_size,join('assets', 'tileset','GRASSLAND', 'TILE1.png'))

lvl1\_dugtrio = Trap(win\_x-lvl1\_block\_size,win\_y - lvl1\_block\_size,dugtrio\_dims[0],dugtrio\_dims[1],dugtrio\_trap)

lvl1\_thunderstone = Evolve(100,600,32,32,join('assets','player\_assets','thunderstone.png'))

# level 2

starting\_block\_1 = Block(0,win\_y-lvl1\_block\_size,lvl1\_block\_size,75,join('assets','DARK AREA BLOCKS','Block 1.png'))

starting\_block\_2 = Block(lvl1\_block\_size,win\_y-lvl1\_block\_size,lvl1\_block\_size,75,join('assets','DARK AREA BLOCKS','middle block.png'))

starting\_block\_3 = Block(lvl1\_block\_size\*2,win\_y-lvl1\_block\_size,lvl1\_block\_size,75,join('assets','DARK AREA BLOCKS','BLOCK 1 REVERSED.png'))

bridge = Block(lvl1\_block\_size\*3,win\_y-lvl1\_block\_size,lvl1\_block\_size,50,join('assets','DARK AREA BLOCKS','bRIDGE.png'))

starting\_block\_4 = Block(lvl1\_block\_size\*4,win\_y-lvl1\_block\_size,lvl1\_block\_size,75,join('assets','DARK AREA BLOCKS','middle block.png'))

floating\_1 = Block(win\_x-96,win\_y-(0.65\*lvl1\_block\_size),96,96,join('assets','DARK AREA BLOCKS','SMALL BLOCK 2.png'))

floating\_2 = Block(win\_x-lvl1\_block\_size,win\_y-(3\*lvl1\_block\_size),lvl1\_block\_size,75,join('assets','DARK AREA BLOCKS','middle block.png'))

floating\_3 = Block(lvl1\_block\_size\*5.75,win\_y-(1.5\*lvl1\_block\_size),lvl1\_block\_size,75,join('assets','DARK AREA BLOCKS','middle block.png'))

floating\_4 = Block(lvl1\_block\_size\*4.5,win\_y-(2.5\*lvl1\_block\_size),lvl1\_block\_size,75,join('assets','DARK AREA BLOCKS','middle block.png'))

floating\_5 = Block(win\_x-lvl1\_block\_size-95,win\_y-(3\*lvl1\_block\_size),100,75,join('assets','DARK AREA BLOCKS','SMALL BLOCK 2.png'))

lvl2\_charmander = Trap(lvl1\_block\_size\*3 - charmander\_dims[0],win\_y - lvl1\_block\_size - charmander\_dims[1],charmander\_dims[0],charmander\_dims[1],charmander\_trap)

lvl2\_thunderstone = Evolve(win\_x-96 + 50,win\_y-(0.65\*lvl1\_block\_size) - 35,32,32,join('assets','player\_assets','thunderstone.png'))

def draw\_floor\_list(list\_obj, screen):

# draws each individual block in the floor list on the screen

for obj in list\_obj:

obj.draw(screen)

**entities.py**

import pygame as pg

import time

from settings import \*

from assets import \*

from objects import \*

from attack import \*

screen = pg.display.set\_mode((win\_x,win\_y))

player\_starting\_x, player\_starting\_y = 0, win\_y - (lvl1\_block\_size + pikachu\_dims[1] + 10)

all\_enemy\_objects = set()

class Player(pg.sprite.Sprite):

lives = 2

gravity = 1 # speed by which player falls/comes down

SPRITES = Pikachu

animation\_delay = 3 # to change sprites

def \_\_init\_\_(self,x,y,width,height):

super().\_\_init\_\_()

self.test = False

self.width = width

self.height = height

self.rect = pg.Rect(x,y,width,height)

# setting player movement speeds

self.x\_vel = 0

self.y\_vel = 0

self.mask = None

# for sprites animation image

self.direction = 'right'

self.animation\_count = 0

# for fallings or gravity

self.fall\_count = 0 # number of frames we have been falling for

self.jumpBool = False # if the player is jumping or not

self.attackBool = True # if the player is ready to attack or not

self.hit = False # if the player is taking damage or not

self.evolve = False

self.start\_time = time.time()

def move(self,dx,dy):

# moves player

self.rect.x += dx

self.rect.y += dy

def move\_left(self,vel):

# moves left

if not (self.rect.x <= 0):

self.x\_vel = -vel

if self.direction != 'left': # updates direction to correctly show sprites animation

self.direction = 'left'

self.animation\_count = 0

def move\_right(self,vel):

# moves right

if not (self.rect.x + self.width >= win\_x):

self.x\_vel = vel

if self.direction != 'right': # updates direction to correctly show sprites animation

self.direction = 'right'

self.animation\_count = 0

def jump(self):

# to jump the player

self.jumpBool = True

self.y\_vel = -12 # decreasing from Y makes the player jump up

self.animation\_count = 0

def loselife(self):

# to make the player lose life in case they are hit

if not self.hit: # player must not be hit already

self.lives -= 1

self.hit = True # set to true until the player stops colliding with any damage giver

def landed(self):

# to enable player land on blocks or platforms

self.y\_vel = 0

def hit\_head(self):

# player hits head and falls down if the block is hit from below

self.y\_vel \*= -1

def powerup(self):

# evolves pikachu into raichu for some seconds, providing invincibility

if self.evolve:

self.current\_time = time.time()

if self.current\_time - self.start\_time <= 5:

print\_text(f'{abs(5 - int(self.current\_time - self.start\_time))}',ingame\_font,self.rect.x,self.rect.y-25,screen)

self.SPRITES = raichu

else:

self.evolve = False

self.SPRITES = Pikachu

def loop(self,fps):

# what the player should be doing every loop iteration

self.y\_vel += min(1,(self.fall\_count/fps) \* Player.gravity) # to bring the player down

if self.rect.y > win\_y: # if player goes below the screen or falls in a pit

if not(self.evolve):

self.lives -= 1

self.rect.x, self.rect.y = player\_starting\_x, player\_starting\_y

if self.attackBool:

self.move(self.x\_vel,self.y\_vel)

self.fall\_count += 1

self.update\_sprite()

self.powerup()

if not(self.attackBool):

self.attack.shoot(screen)

if self.attack.active == False:

self.attackBool = True

def update\_sprite(self):

# updates the player image according to the action being performed

sprite\_sheet = 'idle'

if self.y\_vel < 0 :

if self.jumpBool:

sprite\_sheet = 'jump'

elif self.y\_vel > Player.gravity \* 2 :

sprite\_sheet = 'fall'

elif self.x\_vel != 0:

sprite\_sheet = 'run'

elif not self.attackBool:

sprite\_sheet = 'attack'

sprite\_sheet\_name = sprite\_sheet + "\_" + self.direction

sprites = self.SPRITES[sprite\_sheet\_name]

sprite\_index = (int(self.animation\_count) // self.animation\_delay) % len(sprites)

self.sprite = sprites[sprite\_index]

self.animation\_count += 0.3

self.update()

def update(self):

# updating rect according to the sprite

self.rect = self.sprite.get\_rect(topleft = (self.rect.x, self.rect.y))

self.mask = pg.mask.from\_surface(self.sprite) # masking our sprite image for collision

def draw(self,screen):

# draws player on the screen

screen.blit(self.sprite,(self.rect.x,self.rect.y))

class Enemy(pg.sprite.Sprite):

objects\_list = [] # stores all Enemy instances to help detecting collision

def \_\_init\_\_(self, x, y, dx, direction, rangeBool, speed, spritesheets):

super().\_\_init\_\_()

self.initial\_x = x

self.spritesheets = spritesheets

self.sprite\_index = 0

self.image = spritesheets[f'run\_{direction}'][0]

self.rect = self.image.get\_rect()

self.rect.topleft = (x,y)

self.dx = dx

self.x\_vel = speed # enemy speed

self.active = False # only active when being drawn on the screen

self.rem = True

if rangeBool:

self.ranged = True # if the enemy is a ranged type

else:

self.ranged = False

self.left = self.right = False # initial direction of movement

if direction.lower() == 'left':

self.left = True

elif direction.lower() == 'right':

self.right = True

self.current\_direction = direction # to help with the ranged projectile's direction

Enemy.objects\_list.append(self)

self.tick = 0

def activate(self):

# activates the enemy

if self.tick == 1:

self.active = True

def update\_sprite(self):

# changes sprite sheets to look like animation

if self.tick % 15 == 0:

try:

self.image = self.spritesheets[f'run\_{self.current\_direction}'][self.sprite\_index]

self.sprite\_index += 1

except:

self.sprite\_index = 0

self.image = self.spritesheets[f'run\_{self.current\_direction}'][self.sprite\_index]

def move(self):

# moves enemy according to the parameters provided

screen.blit(self.image,(self.rect.x,self.rect.y))

if self.dx > 0:

if self.left:

if self.rect.x >= (self.initial\_x - self.dx):

self.rect.x -= self.x\_vel

else:

self.left = False

self.right = True

if self.right:

if self.rect.x <= (self.initial\_x + self.dx):

self.rect.x += self.x\_vel

else:

self.right = False

self.left = True

def attack(self):

# shoots if enemy is ranged type

if (self.tick == 1 or self.atk.rem == False) and self.active: # shoot the first tick, then every 100 ticks

self.atk = Attack(self.rect.x, self.rect.y,self.current\_direction,bulb\_attack,500,enemy\_projectile\_speed) # an attack object is created every attack

if self.active:

self.atk.shoot(screen)

def loop(self):

# performs this every iteration

self.tick += 1

self.update\_sprite()

self.activate()

if self.active:

self.move()

elif self.rem:

self.rem = False

if self.ranged:

Attack.enemy\_attacks.remove(self.atk)

Enemy.objects\_list.remove(self)

all\_enemy\_objects.add(self)

self.current\_direction = 'left' if self.left else 'right' if self.right else None

if self.ranged:

self.attack()

# creating player and enemy objects

player = Player(player\_starting\_x, player\_starting\_y,pikachu\_dims[0],pikachu\_dims[1])

# level 1

lvl1\_pidgeott = Enemy(375,win\_y - (2.5\*lvl1\_block\_size) + 20 - 64,300,'right',False, 7,pidgeott)

lvl1\_bulbasaur = Enemy(1000,win\_y - (3\*lvl1\_block\_size) + 20 - pidgeott\_dims[1],150,'right',True,3,bulbasaur)

# level 2

lvl2\_golbat = Enemy(lvl1\_block\_size\*5.75,win\_y-(1.5\*lvl1\_block\_size) - golbat\_dims[1] + 25,300,'right',False,5,golbat)

lvl2\_haunter = Enemy(lvl1\_block\_size\*4.5,win\_y-(2.5\*lvl1\_block\_size)-haunter\_dims[1],250,'left',False,9,haunter)

def handle\_move(player,objects):

# handles player movement and collision

keys = pg.key.get\_pressed()

player.x\_vel = 0

if keys[pg.K\_a]:

player.move\_left(player\_speed)

if keys[pg.K\_d]:

player.move\_right(player\_speed)

handle\_collision\_player(player,objects,player.y\_vel) # checks collision with different objects

def handle\_collision\_player(player,objects,dy):

collided\_objects = []

indices = player.rect.collidelistall(objects) # index of all objects being collided with

BoolList = [] # will store if we collide with a platform or not

for index in indices:

if isinstance(objects[index],Block):

BoolList.append(True) # list full of Trues if we only collide with Block

else:

BoolList.append(False)

if not (False in BoolList): # if there are no Falses, i.e. if we do not collide with anything else

player.hit = False

for obj in objects:

# if player is colliding with an active object

if pg.sprite.collide\_mask(player,obj) and obj.active:

if isinstance(obj, Block): # if the object is a block object or platform

player.fall\_count = 0

if dy > 0 and player.rect.y <= obj.rect.y: # if we are colliding from above

player.rect.bottom = obj.rect.top

player.landed()

player.jumpBool = False

elif dy < 0 : # if we ar colliding from below

player.rect.top = obj.rect.bottom

player.hit\_head()

elif isinstance(obj, Enemy) and not(player.evolve): # if it is an enemy

player.loselife()

elif isinstance(obj, Attack) and obj.player == False and not(player.evolve): # if it is an enemy attack object

player.loselife()

elif isinstance(obj, Trap) and not(player.evolve): # if it is a trap object

player.loselife()

elif isinstance(obj,Evolve):

obj.active = False

obj.display = False

player.evolve = True

player.start\_time = time.time()

collided\_objects.append(obj)

return collided\_objects

def handle\_collision\_enemy(enemy\_list,objects\_list):

for enemy in enemy\_list:

for obj in objects\_list:

if pg.sprite.collide\_mask(enemy,obj):

if enemy.active:

enemy.active = False

**game.py**

import pygame as pg, time

from assets import \*

from button import \*

from objects import \*

from entities import \*

from settings import \*

pg.init()

clock = pg.time.Clock() # to set frame rate

# creating window

srface = pg.Surface((win\_x,win\_y),pg.SRCALPHA)

pg.display.set\_caption(title)

pg.display.set\_icon(win\_logo)

all\_collidable\_p = Block.objects\_list + Enemy.objects\_list + Trap.objects\_list + Attack.enemy\_attacks + Evolve.objects\_list # list of all objects collidable with the player

# animating background for the game

bg\_width = main\_menu\_bg.get\_width()

# sound button functioning

def sfx\_state():

if sound\_state: # SFX toggle

if state not in [state\_list[-1],state\_list[-2]]:

sfx\_on.draw(screen)

else:

mixer.music.play(-1)

if state not in [state\_list[-1],state\_list[-2]]:

sfx\_off.draw(screen)

# screens

def moving\_background(img): # to create a moving background

global scroll, win\_x

scroll -= 5 # sets the amount by which the bg scrolls

screen.blit(img,(-win\_x + scroll,0))

screen.blit(img,(0 + scroll,0)) # prints the bg at a slighlty different position each loop

screen.blit(img,(win\_x + scroll,0))

if scroll == -win\_x : # reset the images and scroll again

scroll = win\_x

def transparent\_bg(rgba):

# this is creating the background-foreground differentiation

global srface,win\_x,win\_y

screen.blit(srface,(0,0))

pg.draw.rect(srface,rgba,(0,0,win\_x,win\_y))

def door(door\_x,door\_y,trig\_list): # level clearing door

global state, state\_list,doorBool

if not(False in trig\_list):

doorBool = True

if doorBool: # if its open and conditions are met

screen.blit(door\_open,(door\_x,door\_y))

door\_open\_rect = pg.Rect(door\_x,door\_y,door\_dims[0],door\_dims[1])

if player.rect.colliderect(door\_open\_rect):

level\_reset()

if state == state\_list[-1]: # if its the last level

state = state\_list[4] # show victory screen

else:

state = state\_list[state\_list.index(state) + 1] # else go to the next level

doorBool = False

else:

screen.blit(door\_closed,(door\_x,door\_y))

def timer():

global seconds, mins, hours, timer\_running, start\_timer, start\_time, stop\_timer

if timer\_running:

if start\_timer:

start\_time = time.time()

start\_timer = False

current\_time = time.time()

seconds = int(current\_time - start\_time)

if seconds == 60 :

seconds = 0

mins += 1

if mins == 60:

mins = 0

hours += 1

print\_text(f'{hours}:{mins}:{seconds}',ingame\_font,win\_x/2,20,screen)

def level\_reset():

# to reset all active objects so that the player doesnt collide with invisible objects

global state\_list,state\_list

if state != state\_list[-1]:

for obj in Block.objects\_list:

obj.active = False

player.rect.x, player.rect.y = player\_starting\_x, player\_starting\_y # reset player pos

player.SPRITES = Pikachu

for obj in Trap.objects\_list:

obj.active = False

for obj in Evolve.objects\_list:

obj.active = False

for obj in all\_enemy\_objects:

obj.tick = 0

Enemy.objects\_list.append(obj)

obj.rem = True

player.evolve = False

def reset\_game():

global doorBool

player.rect.x, player.rect.y = player\_starting\_x, player\_starting\_y # reset player pos

for obj in all\_enemy\_objects:

obj.tick = 0

Enemy.objects\_list.append(obj)

obj.rem = True

for obj in Evolve.objects\_list:

obj.active = True

obj.display = True

for obj in Block.objects\_list:

obj.active = False

doorBool = False

player.evolve = False

def main\_menu():

global state,start\_timer,start\_game,stop\_timer

# background

moving\_background(main\_menu\_bg)

transparent\_bg((0,0,0,85))

screen.blit(title\_img,(419,50))

#buttons

screen.blit(ques\_btn,(520,480))

screen.blit(ques\_btn,(520,610))

screen.blit(ques\_btn,(520,740))

if play\_btn.draw(screen):

reset\_game()

player.lives = 2

start\_game = True

stop\_timer = False

start\_timer = True

state = state\_list[5] # changing state to change screens

if instructions\_btn.draw(screen):

state = state\_list[1]

if credits\_btn.draw(screen):

state = state\_list[2]

def lvl1():

global timer\_running,trigger\_list

trigger\_list = []

timer\_running = True

screen.blit(lvl\_1,(0,0))

transparent\_bg((0,0,0,75))

screen.blit(pikachu\_face,(20,20))

print\_text(f' X {player.lives}',ingame\_font,84,30,screen) # lives count

player.draw(screen)

lvl1\_dugtrio.draw(screen)

lvl1\_dugtrio.move()

draw\_floor\_list(lvl1\_floor,screen) # drawing platform

lvl1\_floor\_end.draw(screen)

lvl1\_floating.draw(screen)

lvl1\_bigtile.draw(screen)

lvl1\_endtile.draw(screen)

# drawing player and enemies

lvl1\_pidgeott.loop() # loops enemy running and shooting

lvl1\_bulbasaur.loop()

lvl1\_thunderstone.draw(screen)

trigger\_list.append(not(lvl1\_pidgeott.active))

trigger\_list.append(not(lvl1\_bulbasaur.active))

# door

door(win\_x-door\_dims[0] - 5,win\_y - lvl1\_block\_size - door\_dims[1] + 5,trigger\_list)

def lvl2():

trigger\_list = []

screen.blit(lvl\_2,(0,0))

transparent\_bg((0,0,0,50))

player.draw(screen)

starting\_block\_1.draw(screen)

starting\_block\_2.draw(screen)

starting\_block\_3.draw(screen)

starting\_block\_4.draw(screen)

bridge.draw(screen)

floating\_1.draw(screen)

floating\_2.draw(screen)

floating\_3.draw(screen)

floating\_4.draw(screen)

floating\_5.draw(screen)

lvl2\_charmander.draw(screen)

lvl2\_golbat.loop()

lvl2\_haunter.loop()

trigger\_list.append(not(lvl2\_golbat.active))

trigger\_list.append(not(lvl2\_haunter.active))

lvl2\_thunderstone.draw(screen)

door(win\_x-door\_dims[0],win\_y-(3\*lvl1\_block\_size) - door\_dims[1],trigger\_list)

screen.blit(pikachu\_face,(20,20))

print\_text(f' X {player.lives}',ingame\_font,84,30,screen) # lives count

def instructions():

global state,screen

#background

moving\_background(main\_menu\_bg)

transparent\_bg((0,0,0,125))

screen.blit(instructions\_title,(332.5,50))

print\_text(instructions\_txt,ins\_font,375,200,screen)

if back\_btn.draw(screen):

state = state\_list[0] # sending back to main menu if back button is pressed

def credits():

global state,screen

#background

moving\_background(main\_menu\_bg)

transparent\_bg((0,0,0,125))

screen.blit(credits\_title,(375,50))

print\_text(credits\_txt,credits\_font,425,200,screen)

if back\_btn.draw(screen):

state = state\_list[0] # sending back to main menu if back button is pressed

def L():

global state,running, timer\_running, start\_timer,stop\_timer

reset\_game()

# background

timer\_running = False

stop\_timer = True

screen.blit(lost\_screen\_bg,(0,0))

screen.blit(ques\_btn,(50,100))

screen.blit(ques\_btn,(50,400))

screen.blit(ques\_btn,(50,700))

# losing screen image

screen.blit(lost\_img,(700,50))

print\_text(f"It took you:\n {hours} Hours {mins} Minutes and {seconds} Seconds.", lastscreen\_font,450,750,screen)

# buttons

if play\_again\_btn.draw(screen):

player.lives = 2

stop\_timer = False

start\_timer = True

state = state\_list[5]

if main\_menu\_btn.draw(screen):

state = state\_list[0]

if quit\_btn.draw(screen):

running = False

def W():

global state,running,timer\_running,start\_timer,stop\_timer

reset\_game()

# background

stop\_timer = True

timer\_running = False

screen.blit(win\_screen\_bg,(0,0))

screen.blit(ques\_btn,(50,100))

screen.blit(ques\_btn,(50,400))

screen.blit(ques\_btn,(50,700))

# winning screen image

screen.blit(won\_img,(625,50))

print\_text(f"It took you:\n {hours} Hours {mins} Minutes and {seconds} Seconds.", lastscreen\_font,450,750,screen)

# buttons

if play\_again\_btn.draw(screen):

player.lives = 3

start\_timer = True

stop\_timer = False

state = state\_list[5]

if main\_menu\_btn.draw(screen):

state = state\_list[0]

if quit\_btn.draw(screen):

running = False

# game loop to run the window

while running:

mouse\_pos = pg.mouse.get\_pos() # Get mouse position

# change screens

if state == state\_list[0]:

main\_menu()

elif state == state\_list[1]:

instructions()

elif state == state\_list[2]:

credits()

elif player.lives < 1:

L()

elif state == state\_list[4]:

W()

elif state == state\_list[5]:

lvl1()

elif state == state\_list[6]:

lvl2()

# event handling

for event in pg.event.get():

if event.type == pg.QUIT: # close button functioning

running = False

if event.type == pg.MOUSEBUTTONDOWN:

if (sfx\_off.rect.collidepoint(mouse\_pos) or sfx\_on.rect.collidepoint(mouse\_pos)):

sound\_state = not sound\_state # switching between sfx on and off

if event.type == pg.KEYDOWN:

if state in state\_list[1:5] and event.key == pg.K\_ESCAPE:

state = state\_list[0] # pressing ESC key on some screens sends back to the main menu

if event.key == pg.K\_w and not(player.jumpBool) :

player.jump()

if event.key == pg.K\_j and player.attackBool:

if player.direction == 'right':

if not player.evolve:

player.attack = Attack(player.rect.x + pikachu\_dims[0],player.rect.y + (pikachu\_dims[1]\*(0.25)),player.direction,pikachu\_bolt\_right,10,1,True)

else:

player.attack = Attack(player.rect.x + raichu\_dims[0],player.rect.y + (raichu\_dims[1]\*(0.25)),player.direction,pikachu\_bolt\_right,10,1,True)

else:

if not player.evolve:

player.attack = Attack(player.rect.x - pikachu\_dims[0],player.rect.y + (pikachu\_dims[1]\*(0.25)),player.direction,pikachu\_bolt\_left,10,1,True)

else:

player.attack = Attack(player.rect.x - raichu\_dims[0],player.rect.y + (raichu\_dims[1]\*(0.25)),player.direction,pikachu\_bolt\_left,10,1,True)

player.attackBool = False

all\_collidable\_p = Block.objects\_list + Enemy.objects\_list + Trap.objects\_list + Attack.enemy\_attacks + Evolve.objects\_list # list of all objects collidable with the player

if start\_game:

player.loop(fps) # loops the player's functions

handle\_move(player,all\_collidable\_p) # handles player movements and collision

handle\_collision\_enemy(Enemy.objects\_list,Attack.player\_attack)

if not(stop\_timer):

timer()

sfx\_state()

clock.tick(fps)

pg.display.update() # keeps updating the screen