**ALIGARH COLLEGE OF**

**ENGINEERING AND TECHNOLOGY**

(Affiliated to AKTU, Lucknow - College Code: 109)

Aligarh, Uttar Pradesh – 202001



**A MINI PROJECT REPORT**

ON

**“K-MAN GAME”**

**Submitted by: Submitted to:**

Name: Kartavya Varshney **Mr. S.K. Sharma**

Roll Number: 1810910033

Sec: A

Semester: 3

**Head of Dept: Dr. Anand Sharma**

**Department of Computer Science & Engineering**

Aligarh College of Engineering and Technology

2019-20

**DECLARATION**

Mr. Kartavya Varshney(1810910033) ,student of B.Tech (CS) 2nd year, studying at Aligarh College of Engineering & Technology, Aligarh, hearby declare that the Mini project report on " Game" submitted to U.P.T.U. (LUCKNOW) in partial fulfilment of degree of Bechlor of Technology in Computer Science &Technology is the original work conducted by me.

The information and data given in the report is authentic to the best of my knowledge. Under any case of discrepancy with the provided details do affect the authentication of the document . I will be responsible for the mistake.

**ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of this project would be complete without the mention of the people who made it possible ,without whose constant guidance and encouragement would have made efforts go in vain. I consider myself privileged to express gratitude and respect towards all those who guided us through the completion of this project.

I convey thanks to my project guide Mr.S.K.Sharma of Computer Science & Engineering Department for providing encouragement, constant support and guidance which was of a great help to complete this project successfully.

Last but not the least, we wish to thank our parents for financing our studies in this college as well as for constantly encouraging us to learn engineering.Their personal sacrifice in providing this opportunity to learn engineering is greatfully acknowledged.

**CERTIFICATION**

This is to certify that the Mini project REPORT entitled "K-MAN Game" submitted by Mr. Kartavya Varshney (1810910033) to the Department of Computer Science & Engineering, Aligarh College of Engineering & Technology, Aligarh, in partial fulfillment of reqirements for the award of degree of Bachelor of Technology in Computer Science Engineering, is a bonafied record of the candidate's own work carried by me under our supervision and guidance.

The matter presented in this report is of innovative nature and has full authenticity.

H.O.D.(CS DEPARTMENT)

Dr. Anand Sharma

Abstract

K-MAN

It is an adventure game in which character name K-MAN. Game is a kind of software with goal to provide entertainment. Anyone can play this game .It is appropriate for everyone and is used to provide entertainment to people . People who gave most time for education and they needs some entertainment for sometime.This game is developed by using python in which pygame library is used . It works on windows 8/10.It can control by using keyboard.

Table of Content

1.Introduction

2.Objective

3.Requirements

4.User Guide

4.1 Controls

5.Design

6.Source Code

7.Important Links

**Introduction**

As we all know in todays world there is lot of tension in people mind.To reduce tension from people mind I have made a game name K-MAN

in which K stands for Kartavya. Anyone can play this Game. This game can also act as a heal of tension. It is very light game i.e. small in size .It is fully offline.

**Objective**

K-MAN is very lite game . It is created in Python and it’s main motive is to reduce tension and engage people . It’s like a refreshing tablet which can refresh people mind when they are in a lot of tension . This project can be a Good time killer too.

**Requirements**

The main requirement to use K-MAN:

1.An PC or Laptop

2.Window 8,10

Other requirements:

Keyboard

**User Guide**

You can start the game by just double click on it.

The keys to control game will be given below:

To move left: Press

To move right: Press

To sit: Press

Jump: Press A

Action: Press S

**Design**

It is a Desktop Based Application made using the Tkinter module.

This project is executed on IDLE 3.7.4

**IDLE:-**

IDLE is Python’s Integrated Development and Learning Environment.

IDLE has the following features:

\* coded in 100% pure Python, using the tkinter GUI toolkit

\* cross-platform: works mostly the same on Windows, Unix, and macOS

\* Python shell window (interactive interpreter) with colorizing of code input, output, and error messages

\* multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features

\* search within any window, replace within editor windows, and search through multiple files (grep)

\* debugger with persistent breakpoints, stepping, and viewing of global and local namespaces

\* configuration, browsers, and other dialogs

**Tkinter:-**

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Use in project:

We have used Tkinter to build the entire GUI i.e. the cars, roads, paragraph display window, text field to enter text etc.

**SOURCE CODE**

Game source file consists of 2 folders and one final compiled python file

1.data

2.resources

3.K-man.py

**1.data**

data folder consists of python files which are uses in executing the game.

1. Constants.py
2. game\_sounds.py
3. main.py
4. setup.py
5. tools.py
6. components
7. states

**2.resources**

This folder consists of graphics file, sound file, music file and fonts

a)constants.py

SCREEN\_HEIGHT = 600

SCREEN\_WIDTH = 800

SCREEN\_SIZE = (SCREEN\_WIDTH,SCREEN\_HEIGHT)

ORIGINAL\_CAPTION = "K-men by kartavya "

## COLORS ##

# R G B

GRAY = (100, 100, 100)

NAVYBLUE = ( 60, 60, 100)

WHITE = (255, 255, 255)

RED = (255, 0, 0)

GREEN = ( 0, 255, 0)

FOREST\_GREEN = ( 31, 162, 35)

BLUE = ( 0, 0, 255)

SKY\_BLUE = ( 39, 145, 251)

YELLOW = (255, 255, 0)

ORANGE = (255, 128, 0)

PURPLE = (255, 0, 255)

CYAN = ( 0, 255, 255)

BLACK = ( 0, 0, 0)

NEAR\_BLACK = ( 19, 15, 48)

COMBLUE = (233, 232, 255)

GOLD = (255, 215, 0)

BGCOLOR = WHITE

SIZE\_MULTIPLIER = 2.5

BRICK\_SIZE\_MULTIPLIER = 2.69

BACKGROUND\_MULTIPLER = 2.679

GROUND\_HEIGHT = SCREEN\_HEIGHT - 62

#kmen FORCES

WALK\_ACCEL = .15

RUN\_ACCEL = 20

SMALL\_TURNAROUND = .35

GRAVITY = 1.01

JUMP\_GRAVITY = .25

JUMP\_VEL = -10

FAST\_JUMP\_VEL = -12.5

MAX\_Y\_VEL = 11

MAX\_RUN\_SPEED = 800

MAX\_WALK\_SPEED = 6

#kmen States

STAND = 'standing'

WALK = 'walk'

JUMP = 'jump'

FALL = 'fall'

SMALL\_TO\_BIG = 'small to big'

BIG\_TO\_FIRE = 'big to fire'

BIG\_TO\_SMALL = 'big to small'

FLAGPOLE = 'flag pole'

WALKING\_TO\_CASTLE = 'walking to castle'

END\_OF\_LEVEL\_FALL = 'end of level fall'

#GOOMBA Stuff

LEFT = 'left'

RIGHT = 'right'

JUMPED\_ON = 'jumped on'

DEATH\_JUMP = 'death jump'

#KOOPA STUFF

SHELL\_SLIDE = 'shell slide'

#BRICK STATES

RESTING = 'resting'

BUMPED = 'bumped'

#COIN STATES

OPENED = 'opened'

#MUSHROOM STATES

REVEAL = 'reveal'

SLIDE = 'slide'

#COIN STATES

SPIN = 'spin'

#STAR STATES

BOUNCE = 'bounce'

#FIRE STATES

FLYING = 'flying'

BOUNCING = 'bouncing'

EXPLODING = 'exploding'

#Brick and coin box contents

MUSHROOM = 'mushroom'

STAR = 'star'

FIREFLOWER = 'fireflower'

SIXCOINS = '6coins'

COIN = 'coin'

LIFE\_MUSHROOM = '1up\_mushroom'

FIREBALL = 'fireball'

#LIST of ENEMIES

GOOMBA = 'goomba'

KOOPA = 'koopa'

#LEVEL STATES

FROZEN = 'frozen'

NOT\_FROZEN = 'not frozen'

IN\_CASTLE = 'in castle'

FLAG\_AND\_FIREWORKS = 'flag and fireworks'

#FLAG STATE

TOP\_OF\_POLE = 'top of pole'

SLIDE\_DOWN = 'slide down'

BOTTOM\_OF\_POLE = 'bottom of pole'

#1UP score

ONEUP = '379'

#MAIN MENU CURSOR STATES

PLAYER1 = '1 player'

#OVERHEAD INFO STATES

MAIN\_MENU = 'main menu'

LOAD\_SCREEN = 'loading screen'

LEVEL = 'level'

GAME\_OVER = 'game over'

FAST\_COUNT\_DOWN = 'fast count down'

END\_OF\_LEVEL = 'end of level'

#GAME INFO DICTIONARY KEYS

COIN\_TOTAL = 'coin total'

SCORE = 'score'

TOP\_SCORE = 'top score'

LIVES = 'lives'

CURRENT\_TIME = 'current time'

LEVEL\_STATE = 'level state'

CAMERA\_START\_X = 'camera start x'

kmen\_DEAD = 'kmen dead'

#STATES FOR ENTIRE GAME

MAIN\_MENU = 'main menu'

LOAD\_SCREEN = 'load screen'

TIME\_OUT = 'time out'

GAME\_OVER = 'game over'

LEVEL1 = 'level1'

#SOUND STATEZ

NORMAL = 'normal'

STAGE\_CLEAR = 'stage clear'

WORLD\_CLEAR = 'world clear'

TIME\_WARNING = 'time warning'

SPED\_UP\_NORMAL = 'sped up normal'

kmen\_INVINCIBLE = 'kmen invincible'

b)game\_sound.py

import pygame as pg

from . import setup

from . import constants as c

class Sound(object):

"""Handles all sound for the game"""

def \_\_init\_\_(self, overhead\_info):

"""Initialize the class"""

self.sfx\_dict = setup.SFX

self.music\_dict = setup.MUSIC

self.overhead\_info = overhead\_info

self.game\_info = overhead\_info.game\_info

self.set\_music\_mixer()

def set\_music\_mixer(self):

"""Sets music for level"""

if self.overhead\_info.state == c.LEVEL:

pg.mixer.music.load(self.music\_dict['main\_theme'])

pg.mixer.music.play()

self.state = c.NORMAL

elif self.overhead\_info.state == c.GAME\_OVER:

pg.mixer.music.load(self.music\_dict['game\_over'])

pg.mixer.music.play()

self.state = c.GAME\_OVER

def update(self, game\_info, kmen):

"""Updates sound object with game info"""

self.game\_info = game\_info

self.kmen = kmen

self.handle\_state()

def handle\_state(self):

"""Handles the state of the soundn object"""

if self.state == c.NORMAL:

if self.kmen.dead:

self.play\_music('death', c.kmen\_DEAD)

elif self.kmen.invincible \

and self.kmen.losing\_invincibility == False:

self.play\_music('invincible', c.kmen\_INVINCIBLE)

elif self.kmen.state == c.FLAGPOLE:

self.play\_music('flagpole', c.FLAGPOLE)

elif self.overhead\_info.time == 100:

self.play\_music('out\_of\_time', c.TIME\_WARNING)

elif self.state == c.FLAGPOLE:

if self.kmen.state == c.WALKING\_TO\_CASTLE:

self.play\_music('stage\_clear', c.STAGE\_CLEAR)

elif self.state == c.STAGE\_CLEAR:

if self.kmen.in\_castle:

self.sfx\_dict['count\_down'].play()

self.state = c.FAST\_COUNT\_DOWN

elif self.state == c.FAST\_COUNT\_DOWN:

if self.overhead\_info.time == 0:

self.sfx\_dict['count\_down'].stop()

self.state = c.WORLD\_CLEAR

elif self.state == c. TIME\_WARNING:

if pg.mixer.music.get\_busy() == 0:

self.play\_music('main\_theme\_sped\_up', c.SPED\_UP\_NORMAL)

elif self.kmen.dead:

self.play\_music('death', c.kmen\_DEAD)

elif self.state == c.SPED\_UP\_NORMAL:

if self.kmen.dead:

self.play\_music('death', c.kmen\_DEAD)

elif self.kmen.state == c.FLAGPOLE:

self.play\_music('flagpole', c.FLAGPOLE)

elif self.state == c.kmen\_INVINCIBLE:

if (self.kmen.current\_time - self.kmen.invincible\_start\_timer) > 11000:

self.play\_music('main\_theme', c.NORMAL)

elif self.kmen.dead:

self.play\_music('death', c.kmen\_DEAD)

elif self.state == c.WORLD\_CLEAR:

pass

elif self.state == c.kmen\_DEAD:

pass

elif self.state == c.GAME\_OVER:

pass

def play\_music(self, key, state):

"""Plays new music"""

pg.mixer.music.load(self.music\_dict[key])

pg.mixer.music.play()

self.state = state

def stop\_music(self):

"""Stops playback"""

pg.mixer.music.stop()

c)main.py

from . import setup,tools

from .states import main\_menu,load\_screen,level1

from . import constants as c

def main():

"""Add states to control here."""

run\_it = tools.Control(setup.ORIGINAL\_CAPTION)

state\_dict = {c.MAIN\_MENU: main\_menu.Menu(),

c.LOAD\_SCREEN: load\_screen.LoadScreen(),

c.TIME\_OUT: load\_screen.TimeOut(),

c.GAME\_OVER: load\_screen.GameOver(),

c.LEVEL1: level1.Level1()}

run\_it.setup\_states(state\_dict, c.MAIN\_MENU)

run\_it.main()

d)setup.py

"""

This module initializes the display and creates dictionaries of resources.

"""

import os

import pygame as pg

from . import tools

from .import constants as c

ORIGINAL\_CAPTION = c.ORIGINAL\_CAPTION

os.environ['SDL\_VIDEO\_CENTERED'] = '1'

pg.init()

pg.event.set\_allowed([pg.KEYDOWN, pg.KEYUP, pg.QUIT])

pg.display.set\_caption(c.ORIGINAL\_CAPTION)

SCREEN = pg.display.set\_mode(c.SCREEN\_SIZE)

SCREEN\_RECT = SCREEN.get\_rect()

FONTS = tools.load\_all\_fonts(os.path.join("resources","fonts"))

MUSIC = tools.load\_all\_music(os.path.join("resources","music"))

GFX = tools.load\_all\_gfx(os.path.join("resources","graphics"))

SFX = tools.load\_all\_sfx(os.path.join("resources","sound"))

e)tools.py

import os

import pygame as pg

keybinding = {

'action':pg.K\_s,

'jump':pg.K\_a,

'left':pg.K\_LEFT,

'right':pg.K\_RIGHT,

'down':pg.K\_DOWN

}

class Control(object):

"""Control class for entire project. Contains the game loop, and contains

the event\_loop which passes events to States as needed. Logic for flipping

states is also found here."""

def \_\_init\_\_(self, caption):

self.screen = pg.display.get\_surface()

self.done = False

self.clock = pg.time.Clock()

self.caption = caption

self.fps = 60

self.show\_fps = False

self.current\_time = 0.0

self.keys = pg.key.get\_pressed()

self.state\_dict = {}

self.state\_name = None

self.state = None

def setup\_states(self, state\_dict, start\_state):

self.state\_dict = state\_dict

self.state\_name = start\_state

self.state = self.state\_dict[self.state\_name]

def update(self):

self.current\_time = pg.time.get\_ticks()

if self.state.quit:

self.done = True

elif self.state.done:

self.flip\_state()

self.state.update(self.screen, self.keys, self.current\_time)

def flip\_state(self):

previous, self.state\_name = self.state\_name, self.state.next

persist = self.state.cleanup()

self.state = self.state\_dict[self.state\_name]

self.state.startup(self.current\_time, persist)

self.state.previous = previous

def event\_loop(self):

for event in pg.event.get():

if event.type == pg.QUIT:

self.done = True

elif event.type == pg.KEYDOWN:

self.keys = pg.key.get\_pressed()

self.toggle\_show\_fps(event.key)

elif event.type == pg.KEYUP:

self.keys = pg.key.get\_pressed()

self.state.get\_event(event)

def toggle\_show\_fps(self, key):

if key == pg.K\_F5:

self.show\_fps = not self.show\_fps

if not self.show\_fps:

pg.display.set\_caption(self.caption)

def main(self):

"""Main loop for entire program"""

while not self.done:

self.event\_loop()

self.update()

pg.display.update()

self.clock.tick(self.fps)

if self.show\_fps:

fps = self.clock.get\_fps()

with\_fps = "{} - {:.2f} FPS".format(self.caption, fps)

pg.display.set\_caption(with\_fps)

class \_State(object):

def \_\_init\_\_(self):

self.start\_time = 0.0

self.current\_time = 0.0

self.done = False

self.quit = False

self.next = None

self.previous = None

self.persist = {}

def get\_event(self, event):

pass

def startup(self, current\_time, persistant):

self.persist = persistant

self.start\_time = current\_time

def cleanup(self):

self.done = False

return self.persist

def update(self, surface, keys, current\_time):

pass

def load\_all\_gfx(directory, colorkey=(255,0,255), accept=('.png', 'jpg', 'bmp')):

graphics = {}

for pic in os.listdir(directory):

name, ext = os.path.splitext(pic)

if ext.lower() in accept:

img = pg.image.load(os.path.join(directory, pic))

if img.get\_alpha():

img = img.convert\_alpha()

else:

img = img.convert()

img.set\_colorkey(colorkey)

graphics[name]=img

return graphics

def load\_all\_music(directory, accept=('.wav', '.mp3', '.ogg', '.mdi')):

songs = {}

for song in os.listdir(directory):

name,ext = os.path.splitext(song)

if ext.lower() in accept:

songs[name] = os.path.join(directory, song)

return songs

def load\_all\_fonts(directory, accept=('.ttf')):

return load\_all\_music(directory, accept)

def load\_all\_sfx(directory, accept=('.wav','.mpe','.ogg','.mdi')):

effects = {}

for fx in os.listdir(directory):

name, ext = os.path.splitext(fx)

if ext.lower() in accept:

effects[name] = pg.mixer.Sound(os.path.join(directory, fx))

return effects

f) components folder

This folders contains component python file which are used in game.

* Bricks.py
* Castle\_flag.py
* Checkpoint.py
* Coin.py
* Coin\_box.py
* Collider.py
* Enemies.py
* Flagpole.py
* Flashing\_coin.py
* Info.py
* Kmen.py
* Powerups.py
* Score.py

g)States folder

This folder contains important python files which are necessary for executing.

i)level1.py

ii)load\_screen.py

iii)main\_menu.py

**i)level1.py**

This python file includes compilation of all resources and data.

The steps behind implementation

1. import pygame as pg

from .. import setup, tools

from .. import constants as c

from .. import game\_sound

from .. components import kmen

from .. components import collider

from .. components import bricks

from .. components import coin\_box

from .. components import enemies

from .. components import checkpoint

from .. components import flagpole

from .. components import info

from .. components import score

from .. components import castle\_flag

2. Called when the State object is created.

3. Sets the background image, rect and scales it to the correct

Proportions.

4. Creates collideable, invisible rectangles over top of the ground

for sprites to walk on.

5. Create collideable rects for all the pipes.

6. Create collideable rects for all the steps.

7. Creates all the breakable bricks for the level. Coin and

powerup groups are created so they can be passed to bricks.

8. Creates all the coin boxes and puts them in a sprite group.

9. Creates the flag pole at the end of the level.

10. Creates all the enemies and stores them in a list of lists.

11. Places kmen at the beginning of the level.

12. Creates invisible checkpoints that when collided will trigger

the creation of enemies from the self.enemy\_group\_list.

13. Sprite groups created for convenience.

14. Updates Entire level using states. Called by the control object.

15. If the level is in a FROZEN state, only kmen will update.

16. Updates kmen in a transition state (like becoming big, small,

or dies). Checks if he leaves the transition state or dies to

change the level state back.

17. If kmen is in a transition state, the level will be in a FREEZE

State.

18. Updates the location of all sprites on the screen.

19. Detect if checkpoint collision occurs, delete checkpoint,

add enemies to self.enemy\_group.

20. Creates the points that appear when kmen touches the

flag pole.

21. Adjusts sprites by their x and y velocities and collisions.

22. Adjusts kmen's position based on his x, y velocities and

potential collisions.

23. Check for collisions after kmen is moved on the x axis.

24. When kmen becomees big, converts all fireflower powerups

to mushroom powerups.

25. When kmen becomes small, converts all mushroom powerups

to fireflower powerups.

26. Puts kmen flush next to the collider after moving on the x axis.

27. Deals with kmen if he hits a shell moving on the x axis.

28. Checks for collisions when kmen moves along the y-axis.

29. Allows collisions only for the item closest to kmens centerx.

30. kmen collisions with coin boxes on the y-axis.

31. kmen collisions with bricks on the y-axis.

32. Kills enemy if on a bumped or broken brick.

33. kmen collisions with pipes on the y-axis.

34. Changes kmen to a FALL state if more than a pixel above a

pipe,ground, step or box.

35. kmen collisions with all enemies on the y-axis.

36. kmen collisions with Koopas in their shells on the y axis.

37. Moves all enemies along the x, y axes and check for collisions.

38. Enemy collisions along the x axis. Removes enemy from

enemy group in order to check against all other enemies then

adds it back.

39. Enemy collisions on the y axis.

40. Moves any koopa in a shell along the x, y axes and checks for collision.

41. Shell collisions along the x axis.

42. Shell collisions along the y axis.

43. Moves mushrooms, stars and fireballs along the x, y axes.

44. Moves mushroom along the x, y axes.

45. Mushroom collisions along the x axis.

46. Mushroom collisions along the y axis.

47. Changes mushroom direction if collision along x axis.

48. Changes mushroom state to SLIDE after hitting ground from

fall.

49. Moves invincible star along x, y axes and checks for collisions.

50. Invincible star collisions along y axis.

51. Allows for a star bounce off the ground and on the bottom of

a box.

52. Moves fireball along the x, y axes and checks for collisions.

53. Simulates fireball bounce off ground.

54. Fireball collisions along x axis.

55. Fireball collisions along y axis.

56. Kills enemy if hit with fireball.

57. Checks if sprite should enter a falling state.

58. Removes enemy from sprite groups if 500 pixels left off the

Screen underneath the bottom of the screen, or right of the

screen if shell.

59. Adjusts kmen's state when the flag is at the bottom.

60. Adds flag score if at top.

61. Restarts the level if kmen is dead.

62. sets the new game values after a player's death.

63. Check if time has run down to 0.

64. Changes the view of the camera.

65. Updates while kmen is in castle at the end of the level.

66. Updates the level for the fireworks and castle flag.

67. End the game.

68. Blit all sprites to the main surface.

**ii)load\_screen.py**

This python file used to load the graphics file and motion of player together.





from .. import setup, tools

from .. import constants as c

from .. import game\_sound

from ..components import info

class LoadScreen(tools.\_State):

def \_\_init\_\_(self):

tools.\_State.\_\_init\_\_(self)

def startup(self, current\_time, persist):

self.start\_time = current\_time

self.persist = persist

self.game\_info = self.persist

self.next = self.set\_next\_state()

info\_state = self.set\_overhead\_info\_state()

self.overhead\_info = info.OverheadInfo(self.game\_info, info\_state)

self.sound\_manager = game\_sound.Sound(self.overhead\_info)

def set\_next\_state(self):

"""Sets the next state"""

return c.LEVEL1

def set\_overhead\_info\_state(self):

"""sets the state to send to the overhead info object"""

return c.LOAD\_SCREEN

def update(self, surface, keys, current\_time):

"""Updates the loading screen"""

if (current\_time - self.start\_time) < 2400:

surface.fill(c.BLACK)

self.overhead\_info.update(self.game\_info)

self.overhead\_info.draw(surface)

elif (current\_time - self.start\_time) < 2600:

surface.fill(c.BLACK)

elif (current\_time - self.start\_time) < 2635:

surface.fill((106, 150, 252))

else:

self.done = True

class GameOver(LoadScreen):

"""A loading screen with Game Over"""

def \_\_init\_\_(self):

super(GameOver, self).\_\_init\_\_()

def set\_next\_state(self):

"""Sets next state"""

return c.MAIN\_MENU

def set\_overhead\_info\_state(self):

"""sets the state to send to the overhead info object"""

return c.GAME\_OVER

def update(self, surface, keys, current\_time):

self.current\_time = current\_time

self.sound\_manager.update(self.persist, None)

if (self.current\_time - self.start\_time) < 7000:

surface.fill(c.BLACK)

self.overhead\_info.update(self.game\_info)

self.overhead\_info.draw(surface)

elif (self.current\_time - self.start\_time) < 7200:

surface.fill(c.BLACK)

elif (self.current\_time - self.start\_time) < 7235:

surface.fill((106, 150, 252))

else:

self.done = True

class TimeOut(LoadScreen):

"""Loading Screen with Time Out"""

def \_\_init\_\_(self):

super(TimeOut, self).\_\_init\_\_()

def set\_next\_state(self):

"""Sets next state"""

if self.persist[c.LIVES] == 0:

return c.GAME\_OVER

else:

return c.LOAD\_SCREEN

def set\_overhead\_info\_state(self):

"""Sets the state to send to the overhead info object"""

return c.TIME\_OUT

def update(self, surface, keys, current\_time):

self.current\_time = current\_time

if (self.current\_time - self.start\_time) < 2400:

surface.fill(c.BLACK)

self.overhead\_info.update(self.game\_info)

self.overhead\_info.draw(surface)

else:

self.done = True

**iii)main\_menu.py**



\_\_author\_\_ = 'kartavya'

import pygame as pg

from .. import setup, tools

from .. import constants as c

from .. components import info, kmen

class Menu(tools.\_State):

def \_\_init\_\_(self):

"""Initializes the state"""

tools.\_State.\_\_init\_\_(self)

persist = {c.COIN\_TOTAL: 0,

c.SCORE: 0,

c.LIVES: 3,

c.TOP\_SCORE: 0,

c.CURRENT\_TIME: 0.0,

c.LEVEL\_STATE: None,

c.CAMERA\_START\_X: 0,

c.kmen\_DEAD: False}

self.startup(0.0, persist)

def startup(self, current\_time, persist):

"""Called every time the game's state becomes this one. Initializes

certain values"""

self.next = c.LOAD\_SCREEN

self.persist = persist

self.game\_info = persist

self.overhead\_info = info.OverheadInfo(self.game\_info, c.MAIN\_MENU)

self.sprite\_sheet = setup.GFX['title\_screen']

self.setup\_background()

self.setup\_kmen()

self.setup\_cursor()

def setup\_cursor(self):

"""Creates the mushroom cursor to select 1 or 2 player game"""

self.cursor = pg.sprite.Sprite()

dest = (220, 358)

self.cursor.image, self.cursor.rect = self.get\_image(

24, 160, 8, 8, dest, setup.GFX['item\_objects'])

self.cursor.state = c.PLAYER1

def setup\_kmen(self):

"""Places kmen at the beginning of the level"""

self.kmen = kmen.kmen()

self.kmen.rect.x = 110

self.kmen.rect.bottom = c.GROUND\_HEIGHT

def setup\_background(self):

"""Setup the background image to blit"""

self.background = setup.GFX['level\_1']

self.background\_rect = self.background.get\_rect()

self.background = pg.transform.scale(self.background,

(int(self.background\_rect.width\*c.BACKGROUND\_MULTIPLER),

int(self.background\_rect.height\*c.BACKGROUND\_MULTIPLER)))

self.viewport = setup.SCREEN.get\_rect(bottom=setup.SCREEN\_RECT.bottom)

self.image\_dict = {}

self.image\_dict['GAME\_NAME\_BOX'] = self.get\_image(

1, 60, 176, 88, (170, 100), setup.GFX['title\_screen'])

def get\_image(self, x, y, width, height, dest, sprite\_sheet):

"""Returns images and rects to blit onto the screen"""

image = pg.Surface([width, height])

rect = image.get\_rect()

image.blit(sprite\_sheet, (0, 0), (x, y, width, height))

if sprite\_sheet == setup.GFX['title\_screen']:

image.set\_colorkey((255, 0, 220))

image = pg.transform.scale(image,

(int(rect.width\*c.SIZE\_MULTIPLIER),

int(rect.height\*c.SIZE\_MULTIPLIER)))

else:

image.set\_colorkey(c.BLACK)

image = pg.transform.scale(image,

(int(rect.width\*3),

int(rect.height\*3)))

rect = image.get\_rect()

rect.x = dest[0]

rect.y = dest[1]

return (image, rect)

def update(self, surface, keys, current\_time):

"""Updates the state every refresh"""

self.current\_time = current\_time

self.game\_info[c.CURRENT\_TIME] = self.current\_time

self.update\_cursor(keys)

self.overhead\_info.update(self.game\_info)

surface.blit(self.background, self.viewport, self.viewport)

surface.blit(self.image\_dict['GAME\_NAME\_BOX'][0],

self.image\_dict['GAME\_NAME\_BOX'][1])

surface.blit(self.kmen.image, self.kmen.rect)

surface.blit(self.cursor.image, self.cursor.rect)

self.overhead\_info.draw(surface)

def update\_cursor(self, keys):

"""Update the position of the cursor"""

input\_list = [pg.K\_RETURN, pg.K\_a, pg.K\_s]

if self.cursor.state == c.PLAYER1:

self.cursor.rect.y = 358

for input in input\_list:

if keys[input]:

self.reset\_game\_info()

self.done = True

elif keys[pg.K\_UP]:

self.cursor.state = c.PLAYER1

def reset\_game\_info(self):

"""Resets the game info in case of a Game Over and restart"""

self.game\_info[c.COIN\_TOTAL] = 0

self.game\_info[c.SCORE] = 0

self.game\_info[c.LIVES] = 3

self.game\_info[c.CURRENT\_TIME] = 0.0

self.game\_info[c.LEVEL\_STATE] = None

self.persist = self.game\_info

These are the important source codes and the rest codes are in CD.

**Important Links:-**

https://drive.google.com/folderview?id=11ZO7ZGIBgIIqv\_VFJt4Pc1Y30kINayzS