# IMPORTING LIBRARIES

import pygame

import random

# MAIN MENU

def menu(start):

run=True

while run:

start.fill((0,0,0))

tetris\_front(start, 'TETRIS', 80, (204,0,0))

front\_text(start, 'Press Enter To Start', 50, (200,200,200))

pygame.display.update()

for i in pygame.event.get():

if i.type == pygame.KEYDOWN:

main(start)

if i.type == pygame.QUIT:

run = False

pygame.font.init()

# BLOCKS

I = [['XX0XX',

'XX0XX',

'XX0XX',

'XX0XX',

'XXXXX'],

['XXXXX',

'0000X',

'XXXXX',

'XXXXX',

'XXXXX']]

J = [['XXXXX',

'X0XXX',

'X000X',

'XXXXX',

'XXXXX'],

['XXXXX',

'XX00X',

'XX0XX',

'XX0XX',

'XXXXX'],

['XXXXX',

'XXXXX',

'X000X',

'XXX0X',

'XXXXX'],

['XXXXX',

'XX0XX',

'XX0XX',

'X00XX',

'XXXXX']]

L = [['XXXXX',

'XXX0X',

'X000X',

'XXXXX',

'XXXXX'],

['XXXXX',

'XX0XX',

'XX0XX',

'XX00X',

'XXXXX'],

['XXXXX',

'XXXXX',

'X000X',

'X0XXX',

'XXXXX'],

['XXXXX',

'X00XX',

'XX0XX',

'XX0XX',

'XXXXX']]

O = [['XXXXX',

'XXXXX',

'X00XX',

'X00XX',

'XXXXX']]

S = [['XXXXX',

'XXXXX',

'X00XX',

'X00XXX',

'XXXXX'],

['XXXXX',

'XX0XX',

'XX00X',

'XXX0X',

'XXXXX']]

T = [['XXXXX',

'XX0XX',

'X000X',

'XXXXX',

'XXXXX'],

['XXXXX',

'XX0XX',

'XX00X',

'XX0XX',

'XXXXX'],

['XXXXX',

'XXXXX',

'X000X',

'XX0XX',

'XXXXX'],

['XXXXX',

'XX0XX',

'X00XX',

'XX0XX',

'XXXXX']]

Z = [['XXXXX',

'XXXXX',

'X00XX',

'XX00X',

'XXXXX'],

['XXXXX',

'XX0XX',

'X00XX',

'X0XXX',

'XXXXX']]

blocks = [I, J, L, O, S, T, Z]

block\_colors = [(255, 255, 0),(0, 255, 0),(0, 255, 255), (255, 128, 0), (0, 0, 255), (255, 0, 255),(255, 0, 0)]

def GRID(fix\_pos={}):

grid=[[(0, 0, 0) for \_ in range(10)] for \_ in range(20)]

for i in range(len(grid)):

for j in range(len(grid[i])):

if (j, i) in fix\_pos:

c = fix\_pos[(j, i)]

grid[i][j] = c

return grid

class Piece(object):

def \_\_init\_\_(self, a, b, shape):

self.a = a

self.b = b

self.shape = shape

self.color = block\_colors[blocks.index(shape)]

self.rotation = 0

def change\_format(shape):

pos= []

format = shape.shape[shape.rotation % len(shape.shape)]

for i, line in enumerate(format):

row = list(line)

for j, column in enumerate(row):

if column == '0':

pos.append((shape.a + j, shape.b + i))

for i, K in enumerate(pos):

pos[i] = (K[0] - 2, K[1] - 4)

return pos

def valid\_space(shape, grid):

allow\_pos = [[(a, b) for a in range(10) if grid[b][a] == (0,0,0)] for b in range(20)]

allow\_pos = [a for res in allow\_pos for a in res]

designed = change\_format(shape)

for x in designed:

if x not in allow\_pos:

if x[1] > -1:

return False

return True

# CHECKING FOR LOST POSITION

def check\_lost(pos):

for res in pos:

num1, num2 = res

if num2< 1:

return True

return False

# NEXT BLOCK

def next\_shape():

return Piece(5, 0, random.choice(blocks))

# TEXT

def tetris\_front(surface, text, size, color):

font = pygame.font.SysFont("Times New Roman", size, bold=True)

label = font.render(text, 1, color)

surface.blit(label,(tlx + W/2 - (label.get\_width()/2), tly + H/4 - label.get\_height()/2))

def front\_text(surface, text, size, color):

font = pygame.font.SysFont("Times New Roman", size, bold=True)

label = font.render(text, 1, color)

surface.blit(label,(tlx + W/2 - (label.get\_width()/2), tly + H/2 - label.get\_height()/2))

# DESIGNING GRID

def draw\_grid(surface, grid):

sx = tlx

sy = tly

for i in range(len(grid)):

pygame.draw.line(surface, (128,128,128), (sx, sy+ i\*size\_of\_block), (sx+W, sy+ i\*size\_of\_block))

for j in range(len(grid[i])):

pygame.draw.line(surface, (128, 128, 128), (sx + j\*size\_of\_block, sy),(sx + j\*size\_of\_block, sy + H))

# CLEARNING THE ROWS

def remove\_blocks(grid, locked):

flag1 = 0

for \_ in range (len(grid)-1, -1, -1):

hori = grid[\_]

if (0, 0, 0) not in hori:

flag1 += 1

flag2 = \_

for o in range(len(hori)):

try:

del locked[(o, \_)]

except:

continue

if flag1 > 0:

for m in sorted(list(locked), key=lambda x: x[1])[::-1]:

x, y = m

if y < flag2:

new = (x, y + flag1)

locked[new] = locked.pop(m)

return flag1

# SHOWING NEXT BLOCK

def next\_block(shape, surface):

font = pygame.font.SysFont('Comic Sans MS', 30)

label = font.render('Next Block', 1, (255,255,255))

sx = tlx+W+28

sy = tly+H/2-100

format = shape.shape[shape.rotation % len(shape.shape)]

for i, line in enumerate(format):

row = list(line)

for j, column in enumerate(row):

if column == '0':

pygame.draw.rect(surface, shape.color, (sx + j\*size\_of\_block, sy + i\*size\_of\_block, size\_of\_block, size\_of\_block), 0)

surface.blit(label, (sx + 10, sy- 30))

# GAME WINDOW

def window(surface, grid, score=0, last\_score=0):

surface.fill((0, 0, 0))

pygame.font.init()

font = pygame.font.SysFont('Algerian', 60)

label = font.render("Tetris", 1, (255, 255, 255))

surface.blit(label, (tlx+(W/2)-(label.get\_width()/2), 30))

# CURRENT SCORE

font=pygame.font.SysFont('Comic Sans MS', 27)

label=font.render('Your Score: ' + str(score), 1, (255, 255, 255))

sx= tlx+W+20

sy=tly+H/2-100

surface.blit(label, (sx + 20, sy + 160))

# HIGH SCORE

label = font.render('High Score: '+ last\_score, 1, (255, 255, 255))

sx = tlx - 248

sy = tly + 20

surface.blit(label, (sx + 20, sy + 160))

for i in range(len(grid)):

for j in range(len(grid[i])):

pygame.draw.rect(surface, grid[i][j], (tlx + j\*size\_of\_block, tly + i\*size\_of\_block, size\_of\_block, size\_of\_block), 0)

pygame.draw.rect(surface, (255,0,0), (tlx, tly,W,H), 5)

draw\_grid(surface, grid)

# MAXIMUM SCORE

def high\_score():

with open('game score.txt', 'r') as f:

lines = f.readlines()

score = lines[0].strip()

return score

#SCORE UPDATE

def score\_update(score):

new\_score = high\_score()

with open('game score.txt','w') as f:

if (int(score) > int(new\_score)):

f.write(str(score))

else:

f.write(str(new\_score))

# MAIN FUNCTION

def main(start):

fix\_pos= {}

grid = GRID(fix\_pos)

last\_score = high\_score()

change\_piece = False

run = True

current\_piece = next\_shape()

next\_piece = next\_shape()

clock = pygame.time.Clock()

time\_of\_falling = 0

speed\_of\_falling = 0.27

level\_time = 0

score = 0

while run:

grid = GRID(fix\_pos)

time\_of\_falling=time\_of\_falling+clock.get\_rawtime()

level\_time += clock.get\_rawtime()

clock.tick()

if (level\_time/1000 > 5):

level\_time = 0

if (speed\_of\_falling > 0.12):

speed\_of\_falling=speed\_of\_falling-0.005

if (time\_of\_falling/1000 > speed\_of\_falling):

time\_of\_falling = 0

current\_piece.b+=1

if not ((valid\_space(current\_piece, grid)) and current\_piece.b > 0):

current\_piece.b-=1

change\_piece = True

for event in pygame.event.get():

if (event.type==pygame.QUIT):

run=False

pygame.display.quit()

if (event.type==pygame.KEYDOWN):

if (event.key==pygame.K\_LEFT):

current\_piece.a-=1

if not (valid\_space(current\_piece, grid)):

current\_piece.a+=1

if (event.key==pygame.K\_RIGHT):

current\_piece.a+=1

if not (valid\_space(current\_piece, grid)):

current\_piece.a-=1

if (event.key == pygame.K\_UP):

current\_piece.rotation+=1

if not (valid\_space(current\_piece, grid)):

current\_piece.rotation-=1

if (event.key==pygame.K\_DOWN):

current\_piece.b+=1

if not (valid\_space(current\_piece, grid)):

current\_piece.b-=1

shape\_pos = change\_format(current\_piece)

for i in range(len(shape\_pos)):

a, b = shape\_pos[i]

if (b > -1):

grid[b][a] = current\_piece.color

if change\_piece:

for pos in shape\_pos:

p = (pos[0], pos[1])

fix\_pos[p] = current\_piece.color

current\_piece = next\_piece

next\_piece = next\_shape()

change\_piece = False

score += remove\_blocks(grid, fix\_pos) \* 10

window(start, grid, score, last\_score)

next\_block(next\_piece,start)

pygame.display.update()

if check\_lost(fix\_pos):

front\_text(start, "GAME OVER!!", 80, (200,200,200))

pygame.display.update()

pygame.time.delay(1000)

run = False

score\_update(score)

# START

# screen width and height

main\_screen\_width = 830

main\_screen\_height = 700

W = 300

H = 600

# size of blocks

size\_of\_block=30

tlx=(main\_screen\_width - W) // 2

tly=(main\_screen\_height - H)

start= pygame.display.set\_mode((main\_screen\_width, main\_screen\_height))

pygame.display.set\_caption('TETRIS')

menu(start)