import pygame

from pygame.locals import \*

pygame.init()

screen\_width = 600

screen\_height = 600

screen = pygame.display.set\_mode((screen\_width, screen\_height))

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

# define font

font = pygame.font.SysFont('Constantia', 30)

# define colours

bg = (234, 218, 184)

# block colours

block\_purple = (242, 85, 96)

block\_green = (86, 174, 87)

block\_blue = (69, 177, 232)

# paddle colours

paddle\_col = (142, 135, 123)

paddle\_outline = (100, 100, 100)

# text colour

text\_col = (78, 81, 139)

# define game variables

cols = 6

rows = 6

clock = pygame.time.Clock()

fps = 60

live\_ball = False

game\_over = 0

# function for outputting text onto the screen

def draw\_text(text, font, text\_col, x, y):

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

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

# brick wall class

class wall():

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

self.width = screen\_width // cols

self.height = 50

def create\_wall(self):

self.blocks = []

# define an empty list for an individual block

block\_individual = []

for row in range(rows):

# reset the block row list

block\_row = []

# iterate through each column in that row

for col in range(cols):

# generate x and y positions for each block and create a rectangle from that

block\_x = col \* self.width

block\_y = row \* self.height

rect = pygame.Rect(block\_x, block\_y, self.width, self.height)

# assign block strength based on row

if row < 2:

strength = 3

elif row < 4:

strength = 2

elif row < 6:

strength = 1

# create a list at this point to store the rect and colour data

block\_individual = [rect, strength]

# append that individual block to the block row

block\_row.append(block\_individual)

# append the row to the full list of blocks

self.blocks.append(block\_row)

def draw\_wall(self):

for row in self.blocks:

for block in row:

# assign a colour based on block strength

if block[1] == 3:

block\_col = block\_blue

elif block[1] == 2:

block\_col = block\_green

elif block[1] == 1:

block\_col = block\_purple

pygame.draw.rect(screen, block\_col, block[0])

pygame.draw.rect(screen, bg, (block[0]), 2)

# paddle class

class paddle():

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

self.reset()

def move(self):

# reset movement direction

self.direction = 0

key = pygame.key.get\_pressed()

if key[pygame.K\_LEFT] and self.rect.left > 0:

self.rect.x -= self.speed

self.direction = -1

if key[pygame.K\_RIGHT] and self.rect.right < screen\_width:

self.rect.x += self.speed

self.direction = 1

def draw(self):

pygame.draw.rect(screen, paddle\_col, self.rect)

pygame.draw.rect(screen, paddle\_outline, self.rect, 3)

def reset(self):

# define paddle variables

self.height = 20

self.width = int(screen\_width / cols)

self.x = int((screen\_width / 2) - (self.width / 2))

self.y = screen\_height - (self.height \* 2)

self.speed = 10

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

self.direction = 0

# ball class

class game\_ball():

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

self.reset(x, y)

def move(self):

# collision threshold

collision\_thresh = 5

# start off with the assumption that the wall has been destroyed completely

wall\_destroyed = 1

row\_count = 0

for row in wall.blocks:

item\_count = 0

for item in row:

# check collision

if self.rect.colliderect(item[0]):

# check if collision was from above

if abs(self.rect.bottom - item[0].top) < collision\_thresh and self.speed\_y > 0:

self.speed\_y \*= -1

# check if collision was from below

if abs(self.rect.top - item[0].bottom) < collision\_thresh and self.speed\_y < 0:

self.speed\_y \*= -1

# check if collision was from left

if abs(self.rect.right - item[0].left) < collision\_thresh and self.speed\_x > 0:

self.speed\_x \*= -1

# check if collision was from right

if abs(self.rect.left - item[0].right) < collision\_thresh and self.speed\_x < 0:

self.speed\_x \*= -1

# reduce the block's strength by doing damage to it

if wall.blocks[row\_count][item\_count][1] > 1:

wall.blocks[row\_count][item\_count][1] -= 1

else:

wall.blocks[row\_count][item\_count][0] = (0, 0, 0, 0)

# check if block still exists, in whcih case the wall is not destroyed

if wall.blocks[row\_count][item\_count][0] != (0, 0, 0, 0):

wall\_destroyed = 0

# increase item counter

item\_count += 1

# increase row counter

row\_count += 1

# after iterating through all the blocks, check if the wall is destroyed

if wall\_destroyed == 1:

self.game\_over = 1

# check for collision with walls

if self.rect.left < 0 or self.rect.right > screen\_width:

self.speed\_x \*= -1

# check for collision with top and bottom of the screen

if self.rect.top < 0:

self.speed\_y \*= -1

if self.rect.bottom > screen\_height:

self.game\_over = -1

# look for collission with paddle

if self.rect.colliderect(player\_paddle):

# check if colliding from the top

if abs(self.rect.bottom - player\_paddle.rect.top) < collision\_thresh and self.speed\_y > 0:

self.speed\_y \*= -1

self.speed\_x += player\_paddle.direction

if self.speed\_x > self.speed\_max:

self.speed\_x = self.speed\_max

elif self.speed\_x < 0 and self.speed\_x < -self.speed\_max:

self.speed\_x = -self.speed\_max

else:

self.speed\_x \*= -1

self.rect.x += self.speed\_x

self.rect.y += self.speed\_y

return self.game\_over

def draw(self):

pygame.draw.circle(screen, paddle\_col, (self.rect.x + self.ball\_rad, self.rect.y + self.ball\_rad),

self.ball\_rad)

pygame.draw.circle(screen, paddle\_outline, (self.rect.x + self.ball\_rad, self.rect.y + self.ball\_rad),

self.ball\_rad, 3)

def reset(self, x, y):

self.ball\_rad = 10

self.x = x - self.ball\_rad

self.y = y

self.rect = Rect(self.x, self.y, self.ball\_rad \* 2, self.ball\_rad \* 2)

self.speed\_x = 4

self.speed\_y = -4

self.speed\_max = 5

self.game\_over = 0

# create a wall

wall = wall()

wall.create\_wall()

# create paddle

player\_paddle = paddle()

# create ball

ball = game\_ball(player\_paddle.x + (player\_paddle.width // 2), player\_paddle.y - player\_paddle.height)

run = True

while run:

clock.tick(fps)

screen.fill(bg)

# draw all objects

wall.draw\_wall()

player\_paddle.draw()

ball.draw()

if live\_ball:

# draw paddle

player\_paddle.move()

# draw ball

game\_over = ball.move()

if game\_over != 0:

live\_ball = False

# print player instructions

if not live\_ball:

if game\_over == 0:

draw\_text('CLIQUE E COMECE', font, text\_col, 100, screen\_height // 2 + 100)

elif game\_over == 1:

draw\_text('VOCê GANHOU!', font, text\_col, 240, screen\_height // 2 + 50)

draw\_text('CLIQUE E COMECE', font, text\_col, 100, screen\_height // 2 + 100)

elif game\_over == -1:

draw\_text('VOCÊ PERDEU!', font, text\_col, 240, screen\_height // 2 + 50)

draw\_text('CLIQUE E COMECE', font, text\_col, 100, screen\_height // 2 + 100)

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

if event.type == pygame.MOUSEBUTTONDOWN and live\_ball == False:

live\_ball = True

ball.reset(player\_paddle.x + (player\_paddle.width // 2), player\_paddle.y - player\_paddle.height)

player\_paddle.reset()

wall.create\_wall()

pygame.display.update()

pygame.quit()