from tkinter import \*

import random

import time

class Ball(object):

'''

초기화 함수로 Ball 객체가 생성됐을 때 시작 위치와 방향, 크기를 설정하고 canvas 객체에서

ball 객체를 구별하는 id 생성

Parameters :

canvas : ball 객체가 생성될 Canvas 객체

'''

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

self.canvas = canvas

self.starting\_posX = 240

self.starting\_posY = 300

self.starting\_direction = [-3, -2, -1, 1, 2, 3]

self.sizeX1 = 10

self.sizeY1 = 10

self.sizeX2 = 25

self.sizeY2 = 25

random.shuffle(self.starting\_direction)

# starting\_direciton 리스트에서 뽑은 랜덤한 X방향으로 이동, 각도를 결정

self.posX = self.starting\_direction[0]

self.posY = -3 # 공을 위쪽 방향으로 이동

self.id = self.canvas.create\_oval(self.sizeX1, self.sizeY1, self.sizeX2, self.sizeY2, fill = 'red')

self.canvas.move(self.id, self.starting\_posX, self.starting\_posY) # (x,y)로 공을 이동

'''

Ball 객체의 위치를 조정하는 함수로 x좌표 값을 설정

Parameters :

x : Ball 객체의 조정할 x 좌표 (음수면 왼쪽, 양수면 오른쪽 방향 값을 가짐)

'''

def set\_posX(self, x):

self.posX = x

'''

Ball 객체의 위치를 조정하는 함수로 y좌표 값을 설정

Parameters :

y : Ball 객체의 조정할 y 좌표 (음수면 위쪽, 양수면 아래쪽 방향 값을 가짐)

'''

def set\_posY(self, y):

self.posY = y

'''

현재 Ball 객체의 위치(x, y 좌표)를 canvas 객체에 나타냄

'''

def draw(self):

self.canvas.move(self.id, self.posX, self.posY)

'''

canvas 객체에서 Ball 객체의 id를 반환

'''

def get\_id(self):

return self.id

class Bar(object):

'''

초기화 함수로 Bar 객체가 생성됐을 때 시작 위치와 방향, 크기를 설정하고 canvas 객체에서

Bar 객체를 구별하는 id 생성

Parameters :

canvas : Bar 객체가 생성될 Canvas 객체

'''

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

self.canvas = canvas

self.starting\_posX = 205

self.starting\_posY = 325

self.sizeX1 = 0

self.sizeY1 = 0

self.sizeX2 = 100

self.sizeY2 = 5

self.id = self.canvas.create\_rectangle(self.sizeX1, self.sizeY1, self.sizeX2, self.sizeY2, fill='green')

self.canvas.move(self.id, self.starting\_posX, self.starting\_posY)

self.posX = 0

'''

현재 Bar 객체의 위치(x, y 좌표)를 canvas 객체에 나타냄

'''

def draw(self):

self.canvas.move(self.id, self.posX, 0)

'''

Bar 객체의 위치를 조정하는 함수로 x좌표 값을 설정

Parameters :

x : Bar 객체의 x 좌표 (음수면 왼쪽, 양수면 오른쪽 방향 값을 가짐)

'''

def set\_posX(self, x):

self.posX = x

'''

Left Key 입력에 따른 Bar 객체의 이동방향 설정

Parameters :

evt : Key-Press event 감지

'''

def set\_posX\_left(self, evt):

self.posX = -3

'''

Right Key 입력에 따른 Bar 객체의 이동방향 설정

Parameters :

evt : Key-Press event 감지

'''

def set\_posX\_right(self, evt):

self.posX = 3

'''

canvas 객체에서 Bar 객체의 id를 반환

'''

def get\_id(self):

return self.id

class Brick:

'''

초기화 함수로 Brick 객체가 생성됐을때 시작 위치와 크기를 설정하고 canvas 객체에서 Brick

객체를 구별하는 id 생성

Parameters :

canvas : Bar 객체가 생성될 Canvas 객체

X : Brick 객체의 X좌표 위치

Y : Brick 객체의 Y좌표 위치

'''

def \_\_init\_\_(self, canvas , X , Y ):

self.canvas = canvas

self.posX = X

self.posY = Y

self.sizeX1 = 0

self.sizeY1 = 0

self.sizeX2 = 50

self.sizeY2 = 25

self.id = self.canvas.create\_rectangle(self.sizeX1,self.sizeY1,self.sizeX2,self.sizeY2,fill = 'blue')

self.canvas.move(self.id, self.posX, self.posY)

'''

canvas 객체에서 Brick 객체의 id 반환

'''

def get\_id(self):

return self.id

'''

Controller Class 정의. Controller Class는 Ball, Bar와 Brick 객체들을 제어하는 클래스이다.

'''

class Controller:

'''

초기화 함수로 tkinter의 tk 객체를 받아 canvas 객체를 생성한 뒤 Ball, Bar, Brick 객체들을 생성한다.

생성한 Brick 객체들은 모두 리스트 bricks에 저장한다.

Parameters :

tk : canvas가 생성될 tk 객체

'''

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

self.tk = tk

# 게임 화면을 그린다. 가로, 세로 사이즈를 지정한다.

self.canvas = Canvas(self.tk,width=500,height=400)

self.canvas.pack() # 앞서 설정한 캔버스를 화면에 추가한다.

self.tk.update()

self.canvas\_height = self.canvas.winfo\_height()

self.canvas\_width = self.canvas.winfo\_width()

self.ball = Ball(self.canvas)

self.ball\_id = self.ball.get\_id()

self.bar = Bar(self.canvas)

self.bar\_id = self.bar.get\_id()

self.bricks = []

self.bias = -10

for i in range(0, 4):

for j in range(0, 5):

self.bricks.append(Brick(self.canvas, 55\*j+120, 30\*i))

# 왼쪽 방향키에 bar.set\_posX\_left()를 바인딩

self.canvas.bind\_all('<KeyPress-Left>', self.bar.set\_posX\_left)

# 오른쪽 방향키에 bar.set\_posX\_right()를 바인딩

self.canvas.bind\_all('<KeyPress-Right>', self.bar.set\_posX\_right)

'''

ball 객체와 wall(canvas의 테두리)가 충돌하였을 때 ball의 움직임을 제어하는 함수.

canvas의 크기(width, height)와 ball의 (x1,y1), (x2,y2)좌표를 비교하여 판단

'''

def collision\_ball\_wall(self):

ball\_pos = self.canvas.coords(self.ball\_id)

if ball\_pos[0] <= 0:

self.ball.set\_posX(3)

if ball\_pos[2] >= self.canvas\_width:

self.ball.set\_posX(-3)

if ball\_pos[1] <= 0:

self.ball.set\_posY(3)

if ball\_pos[3] >= self.canvas\_height:

return False

return True

'''

ball 객체와 bar 객체가 충돌하였을때 ball의 움직임을 제어하는 함수.

ball의 (x1,y1), (x2,y2) 좌표와 bar의 (x1,y1), (x2,y2)좌표를 비교하여 판단

'''

def collision\_ball\_bar(self):

ball\_pos = self.canvas.coords(self.ball\_id)

bar\_pos = self.canvas.coords(self.bar\_id)

if ball\_pos[0] >= (bar\_pos[0] + self.bias):

if ball\_pos[2] <= (bar\_pos[2] - self.bias):

if ball\_pos[3] >= bar\_pos[1]:

if ball\_pos[3] <= bar\_pos[3]:

self.ball.set\_posY(-3)

'''

ball 객체와 brick 객체가 충돌하였을때 ball의 움직임과 brick의 상태를 제어하는 함수.

ball의 (x1,y1), (x2,y2) 좌표와 brick의 (x1,y1), (x2,y2)좌표를 비교하여 판단

'''

def collision\_ball\_brick(self):

ball\_pos = self.canvas.coords(self.ball\_id)

for i in self.bricks:

brick\_pos = self.canvas.coords(i.get\_id())

#공이 벽돌의 밑변에 맞았을 때

if ball\_pos[1] <= brick\_pos[3] and ball\_pos[3] >= ball\_pos[3] and

ball\_pos[0] >= brick\_pos[0] + self.bias and ball\_pos[2] <= brick\_pos[2] + self.bias:

self.canvas.delete(i.get\_id())

try:

self.bricks.remove(i)

except ValueError:

pass

self.ball.set\_posY(3)

#공이 벽돌의 윗변에 맞았을 때

if ball\_pos[1] <= brick\_pos[1] and ball\_pos[3] >= brick\_pos[1] and

ball\_pos[0] >= brick\_pos[0] + self.bias and ball\_pos[2] <= brick\_pos[2] + self.bias:

self.canvas.delete(i.get\_id())

try:

self.bricks.remove(i)

except ValueError:

pass

self.ball.set\_posY(-3)

#공이 벽돌의 왼쪽에 맞았을 때

if ball\_pos[0] <= brick\_pos[0] and ball\_pos[2] >= brick\_pos[0] and

ball\_pos[1] >= brick\_pos[1] + self.bias and ball\_pos[3] <= brick\_pos[3] + self.bias:

self.canvas.delete(i.get\_id())

try:

self.bricks.remove(i)

except ValueError:

pass

self.ball.set\_posX(-3)

#공이 벽돌의 오른쪽에 맞았을 때

if ball\_pos[0] <= brick\_pos[2] and ball\_pos[2] >= brick\_pos[2] and

ball\_pos[1] >= brick\_pos[1] + self.bias and ball\_pos[3] <= brick\_pos[3] + self.bias:

self.canvas.delete(i.get\_id())

try:

self.bricks.remove(i)

except ValueError:

pass

self.ball.set\_posX(3)

'''

bar 객체와 wall(canvas의 테두리)가 충돌하였을때 bar의 움직임을 제어하는 함수. bar의 (x1,y1),

(x2,y2)좌표와 canvas의 크기(width, height)를 비교하여 판단

'''

def collision\_bar\_wall(self):

bar\_pos = self.canvas.coords(self.bar\_id)

if bar\_pos[0] <= 0:

self.bar.set\_posX(3)

if bar\_pos[2] >= self.canvas\_width:

self.bar.set\_posX(-3)

'''

brick 객체를 canvas에 그리는 함수

'''

def draw\_brick(self):

for i in self.bricks:

self.canvas.move(i.get\_id, i.posX, i.posY)

'''

ball 객체와 bar 객체, brick객체와 canvas의 테두리의 충돌 여부를 확인하고 객체들을 canvas에

그리는 함수

'''

def draw\_all(self):

#종료조건. 벽돌이 0개 남았거나 ball 객체의 위치가 bar 객체보다 아래에 있을 때

while self.collision\_ball\_wall() and len(self.bricks) != 0:

self.collision\_bar\_wall()

self.collision\_ball\_brick()

self.collision\_ball\_bar()

self.ball.draw()

self.bar.draw()

self.draw\_brick()

self.tk.update()

time.sleep(0.005) #실행 속도 조절

'''

main으로 tk객체를 생성하고 초기화한 뒤 Controller객체를 생성해 draw\_all() 함수를 호출해

게임을 실행시킨다.

'''

if \_\_name\_\_=="\_\_main\_\_":

tk = Tk() # 창을 생성한다.

tk.title("test") # 제목 설정

tk.resizable(0,0)

tk.wm\_attributes("-topmost",1)

cs = Controller(tk)

cs.draw\_all()