"""  
File: pong.py  
Original Author: Br. Burton  
Designed to be completed by others  
This program implements a simplistic version of the  
classic Pong arcade game.  
"""  
import random  
from zipfile import Path  
  
import arcade  
  
# These are Global constants to use throughout the game  
SCREEN\_WIDTH = 400  
SCREEN\_HEIGHT = 300  
BALL\_RADIUS = 10  
  
PADDLE\_WIDTH = 10  
PADDLE\_HEIGHT = 50  
MOVE\_AMOUNT = 5  
  
SCORE\_HIT = 1  
SCORE\_MISS = 5  
  
class Point:  
 def \_\_init\_\_(self):  
 self.x = 0.0  
 self.y = 2.3  
  
class Velocity:  
 def \_\_init\_\_(self):  
 self.dx = random.uniform(3, 4)  
 self.dy = random.uniform(2, 4)  
  
class Ball:  
 def \_\_init\_\_(self):  
  
 self.center = Point()  
 self.velocity = Velocity()  
 self.center.x = 10  
 self.center.y = random.uniform(10, 290)  
 def draw(self):  
 arcade.draw\_circle\_filled(self.center.x, self.center.y, 10, arcade.color.RED)  
  
 def advance(self):  
 self.center.x += self.velocity.dx  
 self.center.y += self.velocity.dy  
  
 def bounce\_horizontal(self):  
 self.velocity.dx \*= -1  
  
 def bounce\_vertical(self):  
 if self.velocity.dy < 0:  
 self.velocity.dy \*= -1  
 else:  
 self.velocity.dy \*= -1  
  
 def restart(self):  
 self.\_\_init\_\_()  
  
class Paddle():  
 def \_\_init\_\_(self):  
 self.center = Point()  
 self.center.x = SCREEN\_WIDTH - 10  
 self.center.y = PADDLE\_HEIGHT / 2  
  
 def draw(self):  
 arcade.draw\_rectangle\_filled(self.center.x, self.center.y, PADDLE\_WIDTH,PADDLE\_HEIGHT, arcade.color.BLUE)  
  
 def move\_up(self):  
 if self.center.y < SCREEN\_HEIGHT - PADDLE\_HEIGHT/2:  
 self.center.y += MOVE\_AMOUNT  
  
 def move\_down(self):  
 if self.center.y > (PADDLE\_HEIGHT/2):  
 self.center.y -= MOVE\_AMOUNT  
  
class Pong(arcade.Window):  
 """  
 This class handles all the game callbacks and interaction  
 It assumes the following classes exist:  
 Point  
 Velocity  
 Ball  
 Paddle  
 This class will then call the appropriate functions of  
 each of the above classes.  
 You are welcome to modify anything in this class,  
 but should not have to if you don't want to.  
 """  
  
 def \_\_init\_\_(self, width, height):  
 """  
 Sets up the initial conditions of the game  
 :param width: Screen width  
 :param height: Screen height  
 """  
 super().\_\_init\_\_(width, height)  
  
 self.ball = Ball()  
 self.paddle = Paddle()  
 self.score = 0  
  
 # These are used to see if the user is  
 # holding down the arrow keys  
 self.holding\_left = False  
 self.holding\_right = False  
  
 arcade.set\_background\_color(arcade.color.WHITE)  
  
 def on\_draw(self):  
 """  
 Called automatically by the arcade framework.  
 Handles the responsiblity of drawing all elements.  
 """  
  
 # clear the screen to begin drawing  
 arcade.start\_render()  
  
 # draw each object  
 self.ball.draw()  
 self.paddle.draw()  
  
 self.draw\_score()  
  
 def draw\_score(self):  
 """  
 Puts the current score on the screen  
 """  
 score\_text = "Score: {}".format(self.score)  
 start\_x = 10  
 start\_y = SCREEN\_HEIGHT - 20  
 arcade.draw\_text(score\_text, start\_x=start\_x, start\_y=start\_y, font\_size=12, color=arcade.color.NAVY\_BLUE)  
  
 def update(self, delta\_time):  
 """  
 Update each object in the game.  
 :param delta\_time: tells us how much time has actually elapsed  
 """  
  
 # Move the ball forward one element in time  
 self.ball.advance()  
  
 # Check to see if keys are being held, and then  
 # take appropriate action  
 self.check\_keys()  
  
 # check for ball at important places  
 self.check\_miss()  
 self.check\_hit()  
 self.check\_bounce()  
  
 def check\_hit(self):  
 """  
 Checks to see if the ball has hit the paddle  
 and if so, calls its bounce method.  
 :return:  
 """  
 too\_close\_x = (PADDLE\_WIDTH / 2) + BALL\_RADIUS  
 too\_close\_y = (PADDLE\_HEIGHT / 2) + BALL\_RADIUS  
  
 if (abs(self.ball.center.x - self.paddle.center.x) < too\_close\_x and  
 abs(self.ball.center.y - self.paddle.center.y) < too\_close\_y and  
 self.ball.velocity.dx > 0):  
 # we are too close and moving right, this is a hit!  
 self.ball.bounce\_horizontal()  
 self.score += SCORE\_HIT  
  
 def check\_miss(self):  
 """  
 Checks to see if the ball went past the paddle  
 and if so, restarts it.  
 """  
 if self.ball.center.x > SCREEN\_WIDTH:  
 # We missed!  
 self.score -= SCORE\_MISS  
 self.ball.restart()  
  
 def check\_bounce(self):  
 """  
 Checks to see if the ball has hit the borders  
 of the screen and if so, calls its bounce methods.  
 """  
 if self.ball.center.x < 0 and self.ball.velocity.dx < 0:  
 self.ball.bounce\_horizontal()  
  
 if self.ball.center.y < 0 and self.ball.velocity.dy < 0:  
 self.ball.bounce\_vertical()  
  
 if self.ball.center.y > SCREEN\_HEIGHT and self.ball.velocity.dy > 0:  
 self.ball.bounce\_vertical()  
  
 def check\_keys(self):  
 """  
 Checks to see if the user is holding down an  
 arrow key, and if so, takes appropriate action.  
 """  
 if self.holding\_left:  
 self.paddle.move\_down()  
  
 if self.holding\_right:  
 self.paddle.move\_up()  
  
 def on\_key\_press(self, key, key\_modifiers):  
 """  
 Called when a key is pressed. Sets the state of  
 holding an arrow key.  
 :param key: The key that was pressed  
 :param key\_modifiers: Things like shift, ctrl, etc  
 """  
 if key == arcade.key.LEFT or key == arcade.key.DOWN:  
 self.holding\_left = True  
  
 if key == arcade.key.RIGHT or key == arcade.key.UP:  
 self.holding\_right = True  
  
 def on\_key\_release(self, key, key\_modifiers):  
 """  
 Called when a key is released. Sets the state of  
 the arrow key as being not held anymore.  
 :param key: The key that was pressed  
 :param key\_modifiers: Things like shift, ctrl, etc  
 """  
 if key == arcade.key.LEFT or key == arcade.key.DOWN:  
 self.holding\_left = False  
  
 if key == arcade.key.RIGHT or key == arcade.key.UP:  
 self.holding\_right = False  
  
# Creates the game and starts it going  
window = Pong(SCREEN\_WIDTH, SCREEN\_HEIGHT)  
arcade.run()