"""  
File: skeet.py  
Original Author: Br. Burton  
Designed to be completed by others  
This program implements an awesome version of skeet.  
"""  
from abc import ABC  
from abc import abstractmethod  
  
import arcade  
import math  
import random  
  
# These are Global constants to use throughout the game  
SCREEN\_WIDTH = 600  
SCREEN\_HEIGHT = 500  
  
RIFLE\_WIDTH = 100  
RIFLE\_HEIGHT = 20  
RIFLE\_COLOR = arcade.color.DARK\_RED  
  
BULLET\_RADIUS = 3  
BULLET\_COLOR = arcade.color.BLACK\_OLIVE  
BULLET\_SPEED = 10  
  
TARGET\_RADIUS = 20  
TARGET\_COLOR = arcade.color.BLUE  
TARGET\_SAFE\_COLOR = arcade.color.AIR\_FORCE\_BLUE  
TARGET\_SAFE\_RADIUS = 15  
  
  
class Point():  
 def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
  
class Velocity():  
 def \_\_init\_\_(self, x, y):  
 self.velocity\_x = x  
 self.velocity\_y = y  
  
  
class Rifle:  
 """  
 The rifle is a rectangle that tracks the mouse.  
 """  
  
 def \_\_init\_\_(self):  
 self.center = Point(0, 0)  
 self.angle = 45  
  
 def draw(self):  
 arcade.draw\_rectangle\_filled(self.center.x, self.center.y, RIFLE\_WIDTH, RIFLE\_HEIGHT, RIFLE\_COLOR, self.angle)  
  
  
class FlyingObject(ABC):  
 """  
 Base class for all flying objects in the game.  
 If it flies, it is related somehow to this class.  
 """  
 def \_\_init\_\_(self):  
 self.center = Point(0, 0)  
 self.velocity = Velocity(0, 0)  
 self.radius = 0  
 self.alive = True  
  
 def advance(self):  
 self.center.x += self.velocity.velocity\_x  
 self.center.y += self.velocity.velocity\_y  
  
 '''  
 Any flying object has to leave the screen at some point in time.  
 '''  
 def is\_off\_screen(self, screen\_width, screen\_height):  
 if self.center.x > screen\_width + self.radius or self.center.y > screen\_height + self.radius or self.center.y < 0 - self.radius:  
 return True  
 else:  
 return False  
  
  
class Bullet(FlyingObject):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.radius = BULLET\_RADIUS  
 self.angle = 0.0  
  
 def draw(self):  
 arcade.draw\_circle\_filled(self.center.x, self.center.y, self.radius, BULLET\_COLOR)  
  
 '''  
 This function just provides the correct direction to the bullet.  
 '''  
 def fire(self, angle):  
 self.angle = angle  
 self.velocity.velocity\_x = math.cos(math.radians(self.angle))  
 self.velocity.velocity\_y = math.sin(math.radians(self.angle))  
  
 def advance(self):  
 self.center.x += self.velocity.velocity\_x \* BULLET\_SPEED  
 self.center.y += self.velocity.velocity\_y \* BULLET\_SPEED  
  
  
class Target(FlyingObject):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.center = Point(15, random.uniform(SCREEN\_HEIGHT / 2, SCREEN\_HEIGHT - TARGET\_RADIUS))  
 self.velocity = Velocity(random.uniform(1, 5), random.uniform(-2, 5))  
 self.color = TARGET\_COLOR  
 self.radius = TARGET\_RADIUS  
 self.lives = 1  
  
 '''  
 All targets bounce on the upper side of the screen.  
 They all bounce at +1 point to avoid a bug that makes  
 the target get stuck when spawned right at the   
 SCREEN\_HEIGHT - self.radius point.  
 '''  
 def bounce\_roof(self):  
 if self.center.y > SCREEN\_HEIGHT - self.radius + 1:  
 self.velocity.velocity\_y \*= -1  
  
 @abstractmethod  
 def draw(self):  
 pass  
  
 @abstractmethod  
 def hit(self):  
 pass  
  
  
class StandardTarget(Target):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 def draw(self):  
 arcade.draw\_circle\_filled(self.center.x, self.center.y, self.radius, self.color)  
  
 def hit(self):  
 self.alive = False  
 return 1  
  
  
class StrongTarget(Target):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.lives = 3  
 self.life\_text = '3'  
 self.velocity\_x = random.uniform(1, 3)  
 self.velocity\_y = random.uniform(-2, 3)  
  
 '''  
 A text was drawn above the strong target using an arcade  
 method for it. It changes a class variable named life\_text  
 that changes every time the target is hit. Take a look at  
 hit() method in this class.  
 '''  
 def draw(self):  
 arcade.draw\_circle\_outline(self.center.x, self.center.y, self.radius, self.color, 1.0, 128)  
 arcade.draw\_text(self.life\_text, start\_x=self.center.x - (self.radius/2), start\_y=self.center.y - self.radius, font\_size=30, color=arcade.color.COOL\_BLACK)  
  
 def hit(self):  
 if self.lives > 1:  
 self.lives -= 1  
 self.life\_text = str(self.lives)  
 return 1  
 else:  
 self.alive = False  
 return 5  
  
  
class SafeTarget(Target):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.radius = TARGET\_RADIUS \* 2  
  
 def draw(self):  
 arcade.draw\_rectangle\_filled(self.center.x, self.center.y, self.radius, self.radius, self.color, 90)  
  
 def hit(self):  
 self.lives -= 1  
 self.alive = False  
 return -10  
  
  
class Game(arcade.Window):  
 """  
 This class handles all the game callbacks and interaction  
 It assumes the following classes exist:  
 Rifle  
 Target (and it's sub-classes)  
 Point  
 Velocity  
 Bullet  
 This class will then call the appropriate functions of  
 each of the above classes.  
 You are welcome to modify anything in this class, but mostly  
 you shouldn't have to. There are a few sections that you  
 must add code to.  
 """  
  
 def \_\_init\_\_(self, width=SCREEN\_WIDTH, height=SCREEN\_HEIGHT):  
 """  
 Sets up the initial conditions of the game  
 :param width: Screen width  
 :param height: Screen height  
 """  
 super().\_\_init\_\_(width, height)  
  
 self.rifle = Rifle()  
 self.score = 0  
  
 self.bullets = []  
  
 # TODO: Create a list for your targets (similar to the above bullets)  
 self.targets = []  
  
 arcade.set\_background\_color(arcade.color.WHITE)  
  
 def on\_draw(self):  
 """  
 Called automatically by the arcade framework.  
 Handles the responsibility of drawing all elements.  
 """  
  
 # clear the screen to begin drawing  
 arcade.start\_render()  
  
 # draw each object  
 self.rifle.draw()  
  
 for bullet in self.bullets:  
 bullet.draw()  
  
 # TODO: iterate through your targets and draw them...  
 for target in self.targets:  
 target.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  
 """  
 # TODO:  
 self.check\_collisions()  
 self.check\_off\_screen()  
  
 # decide if we should start a target  
 if random.randint(1, 50) == 1:  
 self.create\_target()  
  
 for bullet in self.bullets:  
 bullet.advance()  
  
 # TODO: Iterate through your targets and tell them to advance  
 for target in self.targets:  
 target.advance()  
 target.bounce\_roof()  
  
 def create\_target(self):  
 """  
 Creates a new target of a random type and adds it to the list.  
 :return: void  
 """  
 random\_target = random.randint(1, 3)  
 # TODO: Decide what type of target to create and append it to the list  
 if random\_target == 1:  
 self.targets.append(StandardTarget())  
 elif random\_target == 2:  
 self.targets.append(StrongTarget())  
 else:  
 self.targets.append(SafeTarget())  
  
 def check\_collisions(self):  
 """  
 Checks to see if bullets have hit targets.  
 Updates scores and removes dead items.  
 :return:  
 """  
  
 # NOTE: This assumes you named your targets list "targets"  
 for bullet in self.bullets:  
 for target in self.targets:  
  
 # Make sure they are both alive before checking for a collision  
 if bullet.alive and target.alive:  
 too\_close = bullet.radius + target.radius  
  
 if (abs(bullet.center.x - target.center.x) < too\_close and  
 abs(bullet.center.y - target.center.y) < too\_close):  
 # its a hit!  
 bullet.alive = False  
 self.score += target.hit()  
  
 # We will wait to remove the dead objects until after we  
 # finish going through the list  
  
 # Now, check for anything that is dead, and remove it  
 self.cleanup\_zombies()  
  
 def cleanup\_zombies(self):  
 """  
 Removes any dead bullets or targets from the list.  
 :return:  
 """  
 for bullet in self.bullets:  
 if not bullet.alive:  
 self.bullets.remove(bullet)  
  
 for target in self.targets:  
 if not target.alive:  
 self.targets.remove(target)  
  
 def check\_off\_screen(self):  
 """  
 Checks to see if bullets or targets have left the screen  
 and if so, removes them from their lists.  
 :return:  
 """  
 for bullet in self.bullets:  
 if bullet.is\_off\_screen(SCREEN\_WIDTH, SCREEN\_HEIGHT):  
 self.bullets.remove(bullet)  
  
 for target in self.targets:  
 if target.is\_off\_screen(SCREEN\_WIDTH, SCREEN\_HEIGHT):  
 self.targets.remove(target)  
  
 def on\_mouse\_motion(self, x: float, y: float, dx: float, dy: float):  
 # set the rifle angle in degrees  
 self.rifle.angle = self.\_get\_angle\_degrees(x, y)  
  
 def on\_mouse\_press(self, x: float, y: float, button: int, modifiers: int):  
 # Fire!  
 angle = self.\_get\_angle\_degrees(x, y)  
  
 bullet = Bullet()  
 bullet.fire(angle)  
  
 self.bullets.append(bullet)  
  
 def \_get\_angle\_degrees(self, x, y):  
 """  
 Gets the value of an angle (in degrees) defined  
 by the provided x and y.  
 Note: This could be a static method, but we haven't  
 discussed them yet...  
 """  
 # get the angle in radians  
 angle\_radians = math.atan2(y, x)  
  
 # convert to degrees  
 angle\_degrees = math.degrees(angle\_radians)  
  
 return angle\_degrees  
  
  
# Creates the game and starts it going  
window = Game(SCREEN\_WIDTH, SCREEN\_HEIGHT)  
arcade.run()