Game:Pong

Version: 3

Description

Game begins when a 500x400px window opens with a black background. The title ‘Pong’ is displayed. Two white rectangles are present, 10px width, 45px length, one on each side of the screen, 75px away from their respective sides, in the middle for the y dimension. In the corners are 50px font numbers reflective of the score for both sides. A white ball with radius 5px appears in the middle of the screen moving with a larger horizontal velocity than vertical velocity. When the ball hits the front of the paddle or a wall, ~~it wraps, changing the location of the ball,~~ it bounces changing the direction of the velocity. When the ball hits the left wall, the score of the right player increases by one. When the ball hits the right wall, the score of the left player increases by one. When either player’s score = 11, the game stops, but the window remains open until a player clicks the X, **the paddles can’t be moved once the game ends. When the ‘q’ key is pressed, the left paddle moves up on the screen. When the ‘a’ key is pressed, the left paddle moves down on the screen. When the ‘p’ key is pressed, the right paddle moves up on the screen. When the ‘l’ key is pressed, the right paddle moves down on the screen.**

# Pong V3

# This is an example program that contains graphics using

# the pygame module, contains user-defined classes, and

# responds to multiple kinds of events.

# It contains these kinds of statements: expression, assignment,

# import, function definition, while, for, if, return, class

# definition

# It contains these kinds of expressions: identifier, literal,

# attribute reference, function call, binary operator, expression

# list

# It uses these types: str, int, float, bool, NoneType, function,

# module, tuple, pygame.Surface, pygame.Color, pygame.Rect, Game,

# Ball

import pygame, time, uaio

from pygame.locals import \*

# User-defined functions

def main():

surface = create\_window()

game = Game(surface)

game.play()

pygame.quit()

def create\_window():

# Open a window on the display and return its Surface

title = 'Pong (the game)'

size = (500, 400)

pygame.init()

surface = pygame.display.set\_mode(size, 0, 0)

pygame.display.set\_caption(title)

return surface

# User-defined classes

class Ball:

# An object in this class represents a colored Ball.

def \_\_init\_\_(self, center, radius, color, surface, velocity):

# Initialize a Ball.

# - self is the Ball initialize

# - center is a list containing the x and y int

# coords of the center of the Ball

# - radius is the int pixel radius of the Ball

# - color is the pygame.Color of the Ball

# - surface is the window's pygame.Surface object

self.center = center

self.radius = radius

self.color = color

self.surface = surface

self.velocity = velocity

self.right\_score = 0

self.left\_score = 0

def draw(self):

# Draw the Ball.

# - self is the Ball to draw

pygame.draw.circle(self.surface, self.color, self.center, self.radius)

def move(self):

# Move the Ball.

# - self is the Ball to move

surface\_size = self.surface.get\_size()

for coord in range(0, 2):

self.center[coord] = self.center[coord] + self.velocity[coord]

if self.center[coord] < self.radius:

self.velocity[coord] = -self.velocity[coord]

if coord == 0:

self.right\_score += 1

if self.center[coord] > surface\_size[coord] - self.radius:

self.velocity[coord] = -self.velocity[coord]

if coord == 0:

self.left\_score += 1

class Game:

# An object in this class represents a complete game.

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

# Initialize a Game.

# - self is the Game to initialize

# - surface is the window's pygame.Surface object

self.surface = surface

self.bg\_color = pygame.Color('black')

self.pause\_time = 0.009 # smaller is faster game

self.close\_clicked = False

self.continue\_game = True

self.ball = Ball([250, 200], 5, pygame.Color('white'), surface, [3, 1])

self.left\_paddle = pygame.Rect(75, ((self.surface.get\_height()/2) - (0.5\*30)), 10, 45) #parameters are (x, y, length, width)

self.right\_paddle = pygame.Rect((self.surface.get\_width()-75)-10, (self.surface.get\_height()/2)-(0.5\*30), 10, 45)

pygame.key.set\_repeat(20, 20)

def play(self):

# Play the game until the player presses the close box.

# - self is the Game that should be continued or not.

self.draw()

while not self.close\_clicked: # until player clicks close box

# play frame

self.handle\_event()

if self.continue\_game:

self.update()

self.decide\_continue()

self.draw()

time.sleep(self.pause\_time) # set game velocity by pausing

def handle\_event(self):

# Handle each user event by changing the game state

# appropriately.

# - self is the Game whose events will be handled

event = pygame.event.poll()

speed = 6

if event.type == QUIT:

self.close\_clicked = True

if event.type == KEYDOWN and self.continue\_game:

if pygame.key.get\_pressed()[K\_q]:

if self.left\_paddle.top > speed:

self.left\_paddle.top = self.left\_paddle.top - speed

print(type(event.type))

else:

self.left\_paddle.top = 0

if pygame.key.get\_pressed()[K\_a]:

if self.left\_paddle.top + self.left\_paddle.height + speed < self.surface.get\_height():

self.left\_paddle.top = self.left\_paddle.top + speed

else:

self.left\_paddle.top = self.surface.get\_height() - self.left\_paddle.height

if pygame.key.get\_pressed()[K\_p]:

if self.right\_paddle.top > speed:

self.right\_paddle.top = self.right\_paddle.top - speed

else:

self.right\_paddle.top = 0

if pygame.key.get\_pressed()[K\_l]:

if self.right\_paddle.top + self.right\_paddle.height + speed < self.surface.get\_height():

self.right\_paddle.top = self.right\_paddle.top + speed

else:

self.right\_paddle.top = self.surface.get\_height() - self.right\_paddle.height

def draw(self):

# Draw all game objects.

# - self is the Game to draw

self.surface.fill(self.bg\_color)

self.ball.draw()

pygame.draw.rect(self.surface, pygame.Color('white'), self.left\_paddle)

pygame.draw.rect(self.surface, pygame.Color('white'), self.right\_paddle)

self.draw\_score()

pygame.display.update()

def draw\_score(self):

size = 50

xcoord = self.surface.get\_width() - uaio.get\_width(str(self.ball.right\_score), size)

uaio.draw\_string(str(self.ball.left\_score), self.surface, (0,0), size)

uaio.draw\_string(str(self.ball.right\_score), self.surface, (xcoord,0), size)

def update(self):

# Update the game objects.

# - self is the Game to update

self.ball.move()

if self.ball.velocity[0] > 0:

self.collide\_paddle(self.right\_paddle)

else:

self.collide\_paddle(self.left\_paddle)

def decide\_continue(self):

# Check and remember if the game should continue

# - self is the Game to check

if self.ball.right\_score == 11 or self.ball.left\_score == 11:

self.continue\_game = False

def collide\_paddle(self, paddle):

if paddle.collidepoint(self.ball.center):

self.ball.velocity[0] = -self.ball.velocity[0]

main()