Pygame draw circle on surface

Pygame physics simulation In the previous tutorial we created a blank white screen; Now we want to display some graphics on it. In this tutorial, we will:Draw a circle in our windowDefine class particlesTo create and display a particle objectThese program builds on the code from the previous tutorial. Line numbers show where the code should be added, provided that the program is written exactly like the program written in the previous tutorial (including blank lines). In this tutorial and all the following tutorials, the final code is available by clicking on the code on the Github link at the beginning of the article. In Pygame there are various features for drawing simple shapes. We will display our particles as circles, so use the 'pygame.draw.circle()' feature. This function has several parameters:Surface on which the circle will be drawn (here is our screen)Color (x, y) coordinatesSuch radius (optional)For example: pygame.draw.circle(screen, (0.0 255), (150, 50), 15, 1) This one draws blue ((0.0 255) is blue in RGB notation) in the middle (150, 50) with a radius of 15 and a thickness of 1 pixel. Note that this feature must be called after the screen.fill(background_colour) command and before the flip() command. If you run the program now you should see something like this (I also changed the name of the window, but it is not important). If you're not familiar with how computer screens work, you might expect the circle to be located at the bottom of the screen because the particle coordinates are 50 and the screen is 200 units high. However, the computer display has the origin (0,0) at the top, to the left of the screen, with the x-axis increasing from left to right and the y-axis increasing from top to bottom; the circle is therefore ejected 50 pixels down from the top of the screen. In our final simulation we will want several particles, each of which will have the same type of attributes. Therefore, it makes sense to define the particle class. When we create each particle object with __init__ function (note the double underscores that indicate the built-in method), we give it x.y. The color and thickness of the line will have default values. We will add more attributes as we move on. Particle class: def __init__ (self, (x, y), size): self.x = x self.y = y self.size = size self.color = (0, 0, 255) self.thickness = 1 Next, we add the 'display()' method to our particle object that will draw (like a circle) on the screen. def display (self): pygame.draw.circle (screen, self.colour, (self.x, self.y), self.size, self.thickness) Now we can remove the 'pygame.draw.circle()' call and replace it with a code that creates a particle object (defining its x, y coordinates and size), then calling it the 'display()' method. If you are not familiar with object-oriented programming, it may seem more long winded, but it will build on our program easier later. my_first_particle = Particles((150, 50), 15), my_first_particle.display() If you start the program now, it should look exactly the same as before. However, in the next tutorial I will show you how the code can now be easily changed to display many more circles. I will also introduce a random module that is very handy in creating simulations. Pygame physics simulation As well as a mixer module, drawing an API is quite simple with a few examples. Therefore, instead of re-iterating the documentation as part of this tutorial, I will instead show you some simple (and not-so-simple) examples of what it can do with the draw module in PyGame and a few pitfalls to be aware of. At the end of this tutorial /demo you will see a massive code dump sample application. Just run this script and you will be presented with the application PyGame, which is a sequence to draw the module demos. While it starts, press space bar to continue previewing. Nothing spectacular about it: pygame.draw.rect (surface, color, pyhra. Rect (left, top, width, height)) Also nothing spectacular about it: pygame.draw.circle (surface, color, (x, y), radius) This is the first caveat you should be aware of. PyGame method for creating thicker outlines for circles is to draw more 1-pixel contours. Theoretically, it sounds fine until you see the result: The circle has visible pixel spaces in it. Even more embarrassing is the rectangle that uses 4 line-draw calls to the desired thickness. This creates strange corners. The way to do this for most drawing API calls is to pass an optional last parameter that is thickness. # Draw a rectangle pygame.draw.rect (surface, color, pyhra. Rect(10, 10, 100, 100), 10)# draw a circle pygame.draw.circle (surface, color, (300, 60), 50, 10) Moral story: when you draw a polygon, rectangle, circle, etc., draw it filled or with a thickness of 1 pixel. Everything else is not very well implemented. If you have to draw a rectangle that has a 10-pixel-thick border, then it's best that you re-implement the logic yourself with either 10 1-pixel-thick rectangle calls, or 4 10-pixel-thick rectangle calls for each page. For an example, see do_nice_outlines below. This API is pretty simple. The list of points is a list of x-y coordinates for a polygon. pygame.draw.polygon (surface, color, point_list) Lines are also straight-forward: pygame.draw.line (surface, color, (startX, starty), (endX, endy), width) So I decided to go a little crazy and wrote a 3D spinning wireframe cube using line methods and a lot of math. import pygameimport mathimport time # Ignore these 3 functions. Scroll down for the appropriate code. space to display the next demo') background = create_background (width, height) clock = pygame.time.Clock() dems = [do_rectangle_demo, do_circle_demo, do_horrible_outlines, do_nice_outlines, do_polygon_demo, do_line_demo] the_world_is_a_happy_place = 0 while true: the_world_is_a_happy_place += 1 for an event in pygame.event.get(): if is_trying_to_quit(event): return if event.type == pygame. KEYDOWN and event.key == pyhra. K_SPACE: Previews = Demos[1:] screen.blit(background, (0, 0)) if only (previews) == 0: return demos[0](screen, the_world_is_a_happy_place) pygame.display.flip() clock.tick fps)# Everything above this line is irrelevant to this tutorial. def do_rectangle_demo(surface, counter): left = (counter // 2) % surface.get_width() up = (counter // 3) % surface.get_height() width = 30 height = 30 color = (128, 0, 128) # purple # Draw rectangle pygame.draw.rect(surface, color, pyhra. Rect(left, top, width, height))def do_circle_demo(surface, counter): x = surface.get_width() 2 y = surface.get_height() 2 max_radius = min(x, y) * 4 // 5 radius = abs(int(math.sin(counter * * 3.14159 * 2 / 200) * max_radius)) + 1 color = (0,140, 255) # aquamarine # Draw a circle pygame.draw.circle (surface, color, (x, y), radius)def do_horrible_outlines(surface, counter): color = (255, 0, 0) # red # draw rectangle pygame.draw.rect (surface, color, pyhra. Rect(10, 10, 100, 100), 10) # draw circle pygame.draw.circle (surface, color, (300, 60), 50, 10) def do_nice_outlines (surface, counter): color = (0, 128, 0) # green # draw rectangle draw a circle
center_x = 300 center_y = 60 radius = 45 iterations = 150 for i in range(iterations): ang = i * 3.14159 * 2 / iterations dx = int(math.cos(ang) * radius) dy = int(math.sin(ang) * radius) x = center_x + dx y = center_y + dy pygame.draw.circle(surface, color, (x, y), 5)def do_polygon_demo(surface, counter): color = (255, 255, 0) # yellow
num_points = 8 point_list = [] center_x = surface.get_width() // 2 center_y = surface.get_height() // 2 for i in range(num_points * 2): radius = 100 if i % 2 == 0: radius = radius // 2 ang = i * 3.14159 / num_points + counter * 3.14159 / 60 x = center_x + int(math.cos(ang) * radius) y = center_y + int(math.sin(ang) * radius) point_list.append((x, y)) pygame.draw.polygon(surface, color, point_list)def rotate_3d_points(points , angle_x, angle_y, angle_z): new_points = [] pre bod v bodoch: x = bod[0] y = bod[1] z = bod[2] new_y = y * math.cos(angle_x) - z * math.sin (angle_x) new_z = y * math.sin (angle_x) + z * math.cos (angle_x) y = new_y # nie je matematická zábava, deti? z = new_z new_x = x * math.cos(angle_y) - z * math.sin(angle_y) new_z = x * math.sin (angle_y) + z * math.cos(angle_y) x = new_x z = new_z new_x = x * matematika .cos(angle_z) - y * math.sin(angle_z) new_y = x * math.sin(angle_z) + y * math.cos(angle_z) x = new_x y = new_y new_points.append([x, y, z]) new_pointsdef do_line_demo (povrch, počitadlo): farba = (0, 0, 0) # čierna this angle is 1 rotation per second # rotate about x axis every 2 seconds # rotate about y axis every 4 seconds # rotate axis from every 6 seconds points = rotate_3d_points (cube_points, t/2, t/4, t / 6) flattened_points = [] for the point in points: flattened_points.mount((point[0] * (1 + 1.0 / (point(2) + 3)), point (1) * (1 + 1.0 / (point[2] + 3))) for the con in the joinings: p1 = flattened_points[con]] p 2 = flattened_points <>[con[1]] x1 = p1[0] * 60 + 200 y1 = p1[1] * 60 + 150 x2 = p2[0] * 60 + 200 y2 = p2[1] * 60 + 150 # This is the only line, on which really depends pygame.draw.line (surface, color, (x1, y1), (x2, y2), 4) run_demos(400, 300, 60) Next up : Fonts and Text Hey, There, Python People. I hope you enjoyed the post. I just wanted to give a quick shout-out for the weekly Python code golf, which I recently started over at StringLabs.io. If you like Python puzzles, please come take a look! Out!