**SPRAWOZDANIE**

Zajęcia: Grafika komputerowa

Prowadzący: prof. dr hab. Vasyl Martsenyuk,

**Laboratorium 2**

10.03.2024

**Temat:** Przeksztalcenia 2D w bibliotece pygame

**Wariant 11**

Krzysztof Kłoda

Informatyka,

Niestacjonarne

IV sem.

Gr 2a / 3

1. **Polecenie:**

1. Pokazany jest obraz shuttle.jpg w panelu. Narysowac zamiast obrazu

wielokat wedlug wariantu (liczba n). Okno ma wymiary 600 na 600 pikseli,

a wielokat ma promien 150 pikseli. Kolejne zadanie polega na stosowaniu

odpowiednich przekszta lce´n do wielokata (lub bedziesz potrzebowal kombinacji przeksztalcen) po nacisnieciu na klawisze od 1 do 9 (patrz Fig. 1).

2. Narysowac figure okreslona wariantem (patrz Fig. 2). Dostepne sa trzy podstawowe ksztalty: kolo, kwadrat, trojkat. Podstawowe przeksztalcenia dostepne sa przez pygame.transform

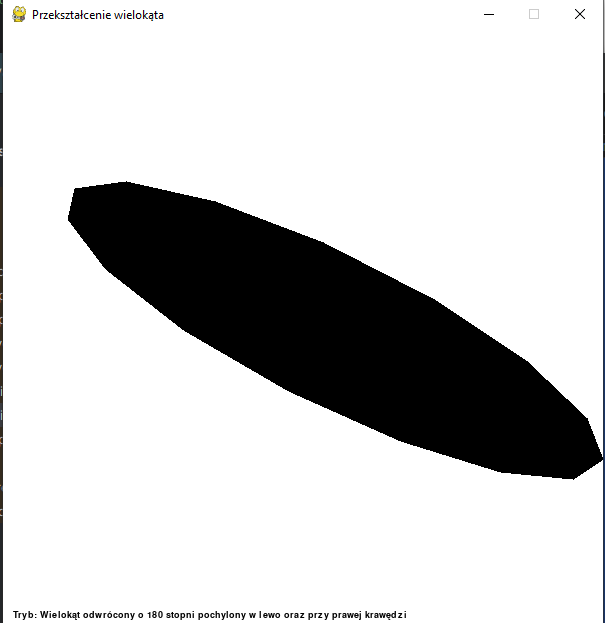
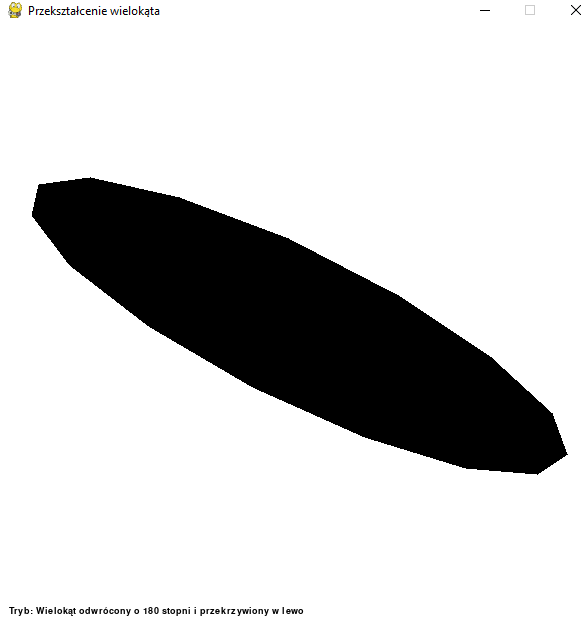
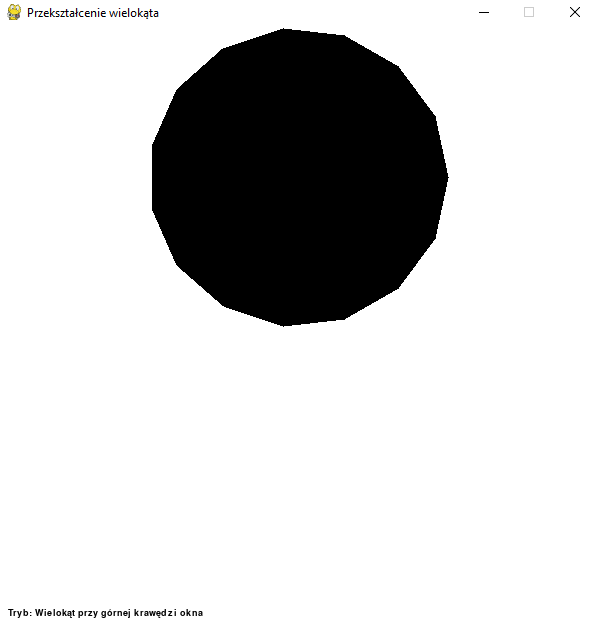
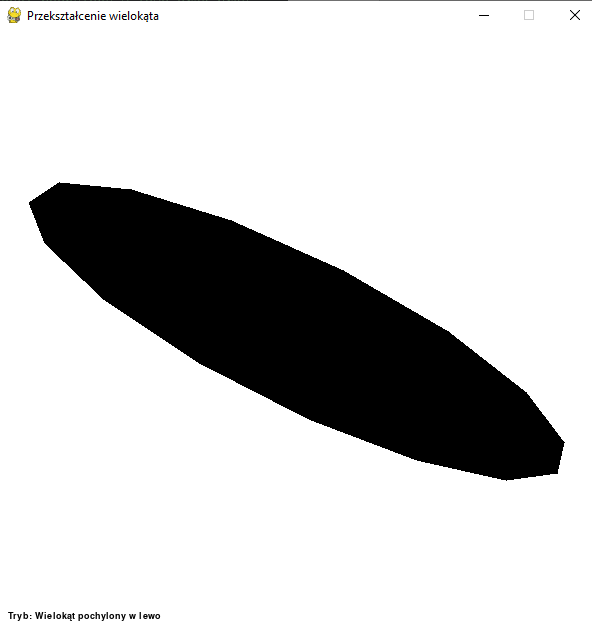
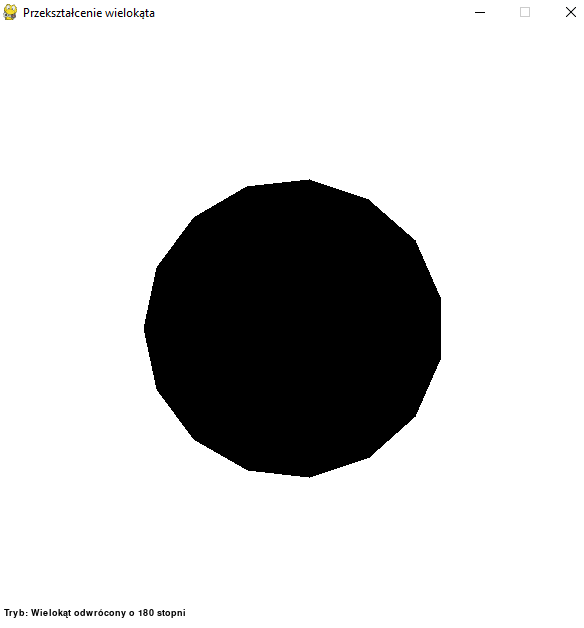
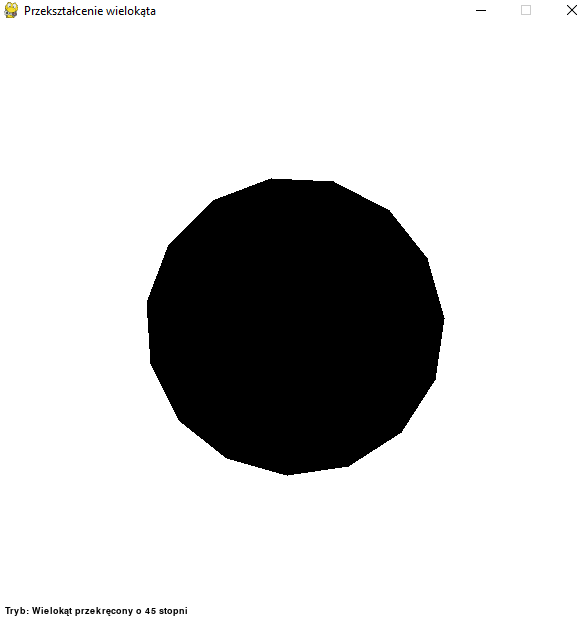
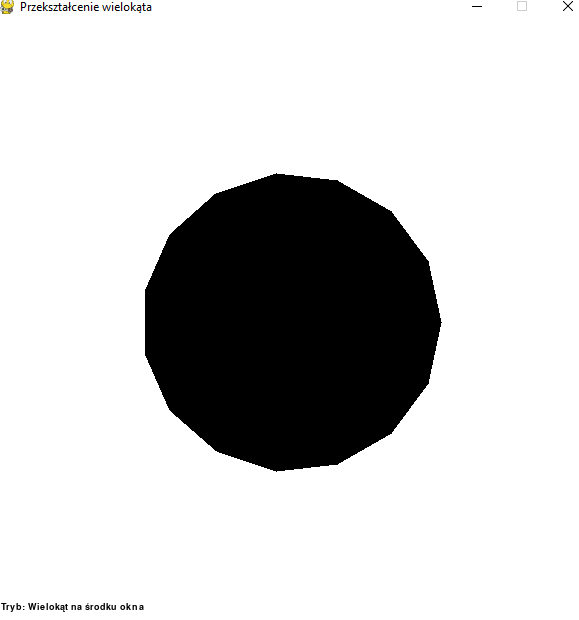
1. **Wprowadzane dane:**
2. **Wykorzystane dane:**

Program : pycharm community edition 2024

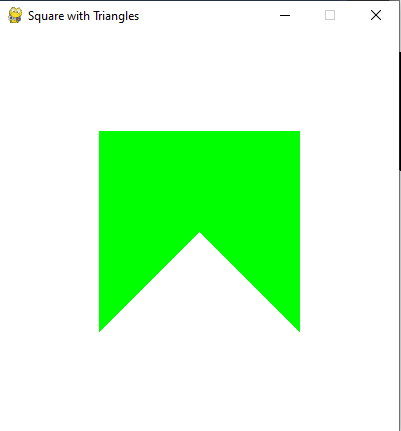
import pygame  
import math  
  
pygame.init()  
  
window\_width = 600  
window\_height = 600  
window = pygame.display.set\_mode((window\_width, window\_height))  
pygame.display.set\_caption("Przekształcenie wielokąta")  
  
WHITE = (255, 255, 255)  
BLACK = (0, 0, 0)  
  
class Polygon:  
 def \_\_init\_\_(self, num\_sides, radius=150):  
 self.num\_sides = num\_sides  
 self.radius = radius  
 self.center = (window\_width // 2, window\_height // 2)  
 self.points = self.calculate\_points()  
  
 def calculate\_points(self):  
 angle\_increment = (2 \* math.pi) / self.num\_sides  
 points = [  
 (  
 self.center[0] + int(self.radius \* math.cos(i \* angle\_increment)),  
 self.center[1] + int(self.radius \* math.sin(i \* angle\_increment))  
 )  
 for i in range(self.num\_sides)  
 ]  
 return points  
  
 def rotate(self, angle):  
 angle\_radians = math.radians(angle)  
 rotated\_points = [  
 (  
 self.center[0] + math.cos(angle\_radians) \* (x - self.center[0]) - math.sin(angle\_radians) \* (y - self.center[1]),  
 self.center[1] + math.sin(angle\_radians) \* (x - self.center[0]) + math.cos(angle\_radians) \* (y - self.center[1])  
 )  
 for x, y in self.points  
 ]  
 self.points = rotated\_points  
  
 def skew(self, factor):  
 skewed\_points = [  
 (x + factor \* (y - self.center[1]), y)  
 for x, y in self.points  
 ]  
 self.points = skewed\_points  
  
 def mirror(self, axis):  
 if axis == "vertical":  
 center\_x = sum(x for x, \_ in self.points) / len(self.points)  
 mirrored\_points = [(2 \* center\_x - x, y) for x, y in self.points]  
 elif axis == "horizontal":  
 center\_y = sum(y for \_, y in self.points) / len(self.points)  
 mirrored\_points = [(x, 2 \* center\_y - y) for x, y in self.points]  
 self.points = mirrored\_points  
  
 def align\_top(self):  
 min\_y = min(y for \_, y in self.points)  
 self.points = [(x, y - min\_y) for x, y in self.points]  
  
 def align\_bottom(self):  
 max\_y = max(y for \_, y in self.points)  
 self.points = [(x, y - max\_y + window\_height) for x, y in self.points]  
  
 def align\_right(self):  
 max\_x = max(x for x, \_ in self.points)  
 self.points = [(x - max\_x + window\_width, y) for x, y in self.points]  
  
 def widen(self, factor):  
 center\_y = sum(y for \_, y in self.points) / len(self.points)  
 self.points = [(x, center\_y + factor \* (y - center\_y)) for x, y in self.points]  
  
def transform\_polygon(polygon, option):  
 if option == 2:  
 polygon.rotate(45)  
 elif option == 3:  
 polygon.rotate(180)  
 elif option == 4:  
 polygon.skew(1.5)  
 elif option == 5:  
 polygon.align\_top()  
 elif option == 6:  
 polygon.skew(1.5)  
 polygon.rotate(180)  
 elif option == 7:  
 polygon.rotate(180)  
 polygon.mirror("vertical")  
 elif option == 8:  
 polygon.rotate(45)  
 polygon.align\_bottom()  
 polygon.widen(1.5)  
 elif option == 9:  
 polygon.rotate(180)  
 polygon.skew(1.5)  
 polygon.align\_right()  
  
def main():  
 running = True  
 option = 1  
 num\_sides = 15 # Changed from 11 to 15  
 polygon = Polygon(num\_sides)  
 font = pygame.font.Font(None, 14)  
  
 while running:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
 elif event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_1:  
 option = 1  
 elif event.key == pygame.K\_2:  
 option = 2  
 elif event.key == pygame.K\_3:  
 option = 3  
 elif event.key == pygame.K\_4:  
 option = 4  
 elif event.key == pygame.K\_5:  
 option = 5  
 elif event.key == pygame.K\_6:  
 option = 6  
 elif event.key == pygame.K\_7:  
 option = 7  
 elif event.key == pygame.K\_8:  
 option = 8  
 elif event.key == pygame.K\_9:  
 option = 9  
  
 window.fill(WHITE)  
 polygon.points = polygon.calculate\_points() # Reset points to original before each transformation  
 transform\_polygon(polygon, option)  
 pygame.draw.polygon(window, BLACK, polygon.points)  
  
 mode\_text = [  
 "Wielokąt na środku okna",  
 "Wielokąt przekręcony o 45 stopni",  
 "Wielokąt odwrócony o 180 stopni",  
 "Wielokąt pochylony w lewo",  
 "Wielokąt przy górnej krawędzi okna",  
 "Wielokąt odwrócony o 180 stopni i przekrzywiony w lewo",  
 "Wielokąt odwrócony o 180 stopni i odwrócenie lustrzane",  
 "Wielokąt odwrócony o 45 stopni oraz przy dolnej krawędzi",  
 "Wielokąt odwrócony o 180 stopni pochylony w lewo oraz przy prawej krawędzi"  
 ][option - 1]  
  
 mode\_text\_surface = font.render("Tryb: " + mode\_text, True, BLACK)  
 window.blit(mode\_text\_surface, (10, 580))  
  
 pygame.display.update()  
  
 pygame.quit()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**4. Wynik działania:**

**1.**



**2.**

****

**5. Wnioski: Na podstawie otrzymanego wyniku można stwierdzić, że**

Przekształcenia 2D są kluczowym elementem grafiki komputerowej, umożliwiając manipulację obiektami w dwuwymiarowej przestrzeni. Wykorzystanie biblioteki Pygame znacząco ułatwia implementację różnych przekształceń dzięki szerokiemu zakresowi dostępnych funkcji. Zrozumienie matematyki leżącej u podstaw przekształceń 2D jest niezbędne dla skutecznego wykorzystania tych technik w praktyce. Przesunięcie, skalowanie i obrót to podstawowe rodzaje przekształceń, które można łatwo zaimplementować w Pygame, co pozwala na szeroki zakres możliwości w projektowaniu gier i aplikacji. Efekty wizualne uzyskane dzięki przekształceniom 2D mogą znacząco poprawić atrakcyjność i funkcjonalność projektów graficznych, zachęcając użytkowników do interakcji. Pomimo bogactwa funkcji oferowanych przez Pygame, umiejętne stosowanie przekształceń wymaga praktyki i doświadczenia programistycznego, aby uzyskać optymalne rezultaty. Stosowanie przekształceń 2D w Pygame otwiera przed programistami szerokie możliwości tworzenia interaktywnych środowisk wizualnych, zarówno w celach edukacyjnych, jak i rozrywkowych.