Sprawozdanie

Zajęcia: Grafika Komputerowa

Prowadzący: mgr inż. Mikołaj Grygiel

Laboratorium Grafiki Komputerowej

10.03.2024

Temat: „Przekształcenia 2D w bibliotece pygame”

Wariant: 7.3

Albert Więcaszek

053012

Informatyka I stopień

Zaoczne

4 semestr

Gr. 2B

1. Polecenia

1. Pokazany jest obraz shuttle.jpg w panelu. Narysować zamiast obrazu wielokąt według wariantu (liczba n). Okno ma wymiary 600 na 600 pikseli, a wielokąt ma promień 150 pikseli. Kolejne zadanie polega na stosowaniu odpowiednich przekształceń do wielokąta (lub będziesz potrzebował kombinacji przekształceń) po naciśnięciu na klawisze od 1 do 9 (patrz Fig. 1).

2. Narysować figurę określoną wariantem (patrz Fig. 2). Dostępne są trzy podstawowe kształty: koło, kwadrat, trójkąt. Podstawowe przekształcenia dostępne są przez pygame.transform

3. Wprowadzane dane

Aby urozmaicić wykonywane zadanie postanowiono nie korzystać z wbudowanych metod do transformacji, a zaimplementowanie własnej funkcji odpowiedzialnej za skalowanie, rotację, pochylenie oraz symetrię względem osi pionowej i poziomej. Wszystkie zaimplementowane przekształcenia są wykonywane względem środka geometrycznego figury.

W tym celu najpierw przygotowano zmienne odpowiedzialne za sterowanie poszczególnymi przekształceniami:

```
r = 150
scaleX = 1
scaleY = 1
phase = 0
mirrorX = False
mirrorY = False
slash = 0
```

Zmienna *r* odpowiada za bazowy promień rysowania, zmienne *scaleX* oraz *scaleY* odpowiadają za procentową wartość skalowania figury względem tych osi, zmienna *phase* opisuje obrót figury (w radianach), flagi *mirrorX* oraz *mirrorY* stwierdzają czy figura ma zostać symetrycznie odbita względem którejś z osi, natomiast zmienna *slash* nadaje wartość (w pikselach) o ile pochylona ma być figura (wszystkie wierzchołki powyżej geometrycznego środka są przesuwane w kierunku minus X o zadaną wartość).

Kolejnym krokiem było zdefiniowanie metod *getCentroid()* zwracającej obiekty typu *tuple* zawierający współrzędne środka figury oraz metodę *drawFigure()* które rysuje wybrany wielobok (w tym przykładzie siedmiokąt foremny) i nakłada na niego odpowiednie transformacje. Metoda ta również zwraca obecne współrzędne środka ciężkości figury. Kod wspomnianych metod przedstawiono poniżej:

```
def getCentroid(x, y):
    return sum(x) / len(x), sum(y) / len(y)
```

```
def drawFigure():
    surface = pygame.display.set_mode((surfWidth, surfHeight))
    surface.fill((ZOLTY))
    theta = math.pi - 2.24285714286  # 2.242 RAD = 128 & 4 / 7 for 7 -
sided figure
    x = []
    y = []
    x0, y0, x1, y1 = initX, initY, initX + r * math.cos(phase) * scaleX,
initY + r * math.sin(phase) * scaleY
    for i in range(7):
        x.append(x0)
        y.append(y0)
        x0 = x1
        y0 = y1
        x1 += r * math.cos((i + 1) * theta + phase) * scaleX
        y1 += r * math.sin((i + 1) * theta + phase) * scaleY

    x.append(x0)
    y.append(y0)

    if mirrorX:
        for i in range(len(x)):
            if x[i] > initX:
                x[i] = x[i] - 2 * math.fabs(x[i] - initX)
            else:
                x[i] = x[i] + 2 * math.fabs(x[i] - initX)

    if mirrorY:
        for i in range(len(y)):
            if y[i] > initY:
                y[i] = y[i] - 2 * math.fabs(y[i] - initY)
            else:
                y[i] = y[i] + 2 * math.fabs(y[i] - initY)

    if slash != 0:
        for i in range(len(y)):
            if y[i] < 335:
                x[i] -= slash

    for i in range(7):
        if i == 2:
            pygame.draw.line(surface, CZERWONY, (x[i], y[i]), (x[i + 1],
y[i + 1]), 10)
        else:
            pygame.draw.line(surface, CZARNY, (x[i], y[i]), (x[i + 1], y[i
+ 1]), 10)

    return getCentroid(x, y)
```

Następnie w głównej pętli *while run* programu dodano eventy sterujące obecnie wybranym przekształceniem:

```
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_1:
            currentDrawMode = 1
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_2:
            currentDrawMode = 2
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_3:
            currentDrawMode = 3
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_4:
            currentDrawMode = 4
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_5:
            currentDrawMode = 5
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_6:
            currentDrawMode = 6
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_7:
            currentDrawMode = 7
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_8:
            currentDrawMode = 8
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_9:
            currentDrawMode = 9
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_0:
            currentDrawMode = 0
```

Aby oszczędzić zasoby i nie renderować figury od podstaw przy każdym powtórzeniu pętli *while*, nowe rysowanie nastepuje jedynie gdy obecny trym rysowania opisany zmienną *currentDrawMode* różni się od trybu rysowania w poprzedniu przejściu petli opisanego zmienną *lastDrawMode*. Pod koniec każdego przejścia pętli wartość zmiennej *currentDrawMode* jest przypisywana do zmiennej *lastDrawMode*:

```
lastDrawMode = currentDrawMode
pygame.display.update()
```

W momencie gdy użytkownik wybierze z klawiatury przycisk odpowiedzialny za daną transformację wywoływana jest komenda odpowiedzialna za aktualizację parametrów rysowania (nałożenie odpowiednich przekształceń). Najpierw wszystkie parametry są przywracane do wartości domyślnych, a następnie za pomocą instrukcji *match* dopasowywane jest odpowiednie przekształcenie. Na koniec figura jest jednorazowo przerysowywana:

```
if currentDrawMode != lastDrawMode:
    print("Redraw")
    win.fill(ZOLTY)
    phase = 0
    scaleX = 1
    scaleY = 1
    initX = 200
    initY = 200
    mirrorX = False
    mirrorY = False
    r = 150
    slash = 0
```

```
match currentDrawMode.__str__():
        case "1": #symetrycznie przeskaluj całą figurę o 50%
            scaleX = 0.5
            scaleY = 0.5
            initX = initX * scaleX + r
            initY = initY * scaleY + r
        case "0": ##nie rób nic, bazowe parametry są nadawane zawsze
            print("Restore default")
        case "2": #obróć figurę o 15 stopni i zaktualizuj położenie środka
ciężkości tak aby znajdował się tam gdzie początkowo
            xc, yc = drawFigure()
            phase = 15.0 * math.pi / 180.0  # 45 deg to rad
            xc2, yc2 = drawFigure()
            initX += xc - xc2
            initY += yc - yc2
        case "3": #przeskaluj o 50% w osi X oraz odbij symetrycznie
względem osi Y
            scaleX = 0.5
            mirrorY = True
            initX += 75 / 2
            initY += 335
        case "4": #pochyl figurę
            slash = 30
        case "5": #rozszerz figurę o 50% wzdłuz osi X oraz dosuń figurę do
górnej krawędzi ekranu
            initY = 0
            scaleX = 1.5
        case "6": #pochyl figurą oraz obróć ją o 90 stopni
            slash = 30
            xc, yc = drawFigure()
            phase = 90.0 * math.pi / 180.0  # 45 deg to rad
            xc2, yc2 = drawFigure()
            initX += xc - xc2
            initY += yc - yc2
        case "7": #odbij figurę symetrycznie względem osi X oraz Y
            scaleX = 0.5
            mirrorX = True
            mirrorY = True
            initX += 75 * 3 / 2
            initY += 335
        case "8": #obróć figurę, wydłuż ją wzdłuż osi X oraz przesuń w dół
ekranu
            phase = 45.0 * math.pi / 180.0  # 45 deg to rad
            xc = 275
            yc = initY + 335 / 2
            xc2, yc2 = 143, 363
            initX += xc - xc2
            initY += yc - yc2
            scaleX = 1.5
            scaleY = 0.5
            initY += 100
        case "9": #pochyl figurę oraz obróć ją o 180 stopni i dosuń do
prawej krawędi ekranu
            slash = 30
            xc, yc = drawFigure()
            phase = 180.0 * math.pi / 180.0  # 45 deg to rad
            xc2, yc2 = drawFigure()
            initX += xc - xc2+160
            initY += yc - yc2

    drawFigure()
```

W przypadku ćwiczenia 2 należało narysować prostokąt oraz 2 trójkąty. Prostokąt narysowano komendą *pygame.draw.rect()*, natomiast trójkąty za pomocą komendy *pygame.draw.polygon()*.

4. Wykorzystane komendy

Kod źródłowy programu 1:

```
import pygame
import math

pygame.init()
win = pygame.display.set_mode((600, 600))
pygame.display.set_caption("First Game")

# deklarowanie kolorów
CZERWONY = (255, 0, 0)
ZIELONY = (0, 255, 0)
ZOLTY = (255, 255, 0)
FIOLETOWY = (128, 0, 128)
JASNY_NIEBIESKI = (0, 255, 255)
POMARANCZOWY = (255, 165, 0)
NIEBIESKI = (0, 0, 255)
SZARY = (128, 128, 128)
CZARNY = (0, 0, 0)

currentDrawMode = 0
lastDrawMode = 0
initX, initY, surfWidth, surfHeight = 200, 200, 600, 600
surface = pygame.display.set_mode((surfWidth, surfHeight))
surface.fill((ZOLTY))

r = 150
scaleX = 1
scaleY = 1
phase = 0
mirrorX = False
mirrorY = False
slash = 0

def getCentroid(x, y):
    return sum(x) / len(x), sum(y) / len(y)
def drawFigure():
    surface = pygame.display.set_mode((surfWidth, surfHeight))
    surface.fill((ZOLTY))
    theta = math.pi - 2.24285714286  # 2.242 RAD = 128 & 4 / 7 for 7 -
sided figure
    x = []
    y = []
    x0, y0, x1, y1 = initX, initY, initX + r * math.cos(phase) * scaleX,
initY + r * math.sin(phase) * scaleY
    for i in range(7):
        x.append(x0)
        y.append(y0)
        x0 = x1
        y0 = y1
        x1 += r * math.cos((i + 1) * theta + phase) * scaleX
        y1 += r * math.sin((i + 1) * theta + phase) * scaleY

    x.append(x0)
```

```
    y.append(y0)

    if mirrorX:
        for i in range(len(x)):
            if x[i] > initX:
                x[i] = x[i] - 2 * math.fabs(x[i] - initX)
            else:
                x[i] = x[i] + 2 * math.fabs(x[i] - initX)

    if mirrorY:
        for i in range(len(y)):
            if y[i] > initY:
                y[i] = y[i] - 2 * math.fabs(y[i] - initY)
            else:
                y[i] = y[i] + 2 * math.fabs(y[i] - initY)

    if slash != 0:
        for i in range(len(y)):
            if y[i] < 335:
                x[i] -= slash

    for i in range(7):
        if i == 2:
            pygame.draw.line(surface, CZERWONY, (x[i], y[i]), (x[i + 1],
y[i + 1]), 10)
        else:
            pygame.draw.line(surface, CZARNY, (x[i], y[i]), (x[i + 1], y[i
+ 1]), 10)

    return getCentroid(x, y)




drawFigure()

run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_1:
            currentDrawMode = 1
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_2:
            currentDrawMode = 2
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_3:
            currentDrawMode = 3
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_4:
            currentDrawMode = 4
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_5:
            currentDrawMode = 5
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_6:
            currentDrawMode = 6
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_7:
            currentDrawMode = 7
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_8:
            currentDrawMode = 8
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_9:
            currentDrawMode = 9
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_0:
```

```
            currentDrawMode = 0

    # print("Current:Last drawMode:
"+drawMode.__str__()+":"+lastDrawMode.__str__())

    if currentDrawMode != lastDrawMode:
        print("Redraw")
        win.fill(ZOLTY)
        phase = 0
        scaleX = 1
        scaleY = 1
        initX = 200
        initY = 200
        mirrorX = False
        mirrorY = False
        r = 150
        slash = 0
        match currentDrawMode.__str__():
            case "1": #symetrycznie przeskaluj całą figurę o 50%
                scaleX = 0.5
                scaleY = 0.5
                initX = initX * scaleX + r
                initY = initY * scaleY + r
            case "0": ##nie rób nic, bazowe parametry są nadawane zawsze
                print("Restore default")
            case "2": #obróć figurę o 15 stopni i zaktualizuj położenie
środka ciężkości
                        # tak aby znajdował się tam gdzie początkowo
                xc, yc = drawFigure()
                phase = 15.0 * math.pi / 180.0  # 45 deg to rad
                xc2, yc2 = drawFigure()
                initX += xc - xc2
                initY += yc - yc2
            case "3": #przeskaluj o 50% w osi X oraz odbij symetrycznie
względem osi Y
                scaleX = 0.5
                mirrorY = True
                initX += 75 / 2
                initY += 335
            case "4": #pochyl figurę
                slash = 30
            case "5": #rozszerz figurę o 50% wzdłuz osi X oraz dosuń figurę
do górnej krawędzi ekranu
                initY = 0
                scaleX = 1.5
            case "6": #pochyl figurą oraz obróć ją o 90 stopni
                slash = 30
                xc, yc = drawFigure()
                phase = 90.0 * math.pi / 180.0  # 45 deg to rad
                xc2, yc2 = drawFigure()
                initX += xc - xc2
                initY += yc - yc2
            case "7": #odbij figurę symetrycznie względem osi X oraz Y
                scaleX = 0.5
                mirrorX = True
                mirrorY = True
                initX += 75 * 3 / 2
                initY += 335
            case "8": #obróć figurę, wydłuż ją wzdłuż osi X oraz przesuń w
dół ekranu
                phase = 45.0 * math.pi / 180.0  # 45 deg to rad
```

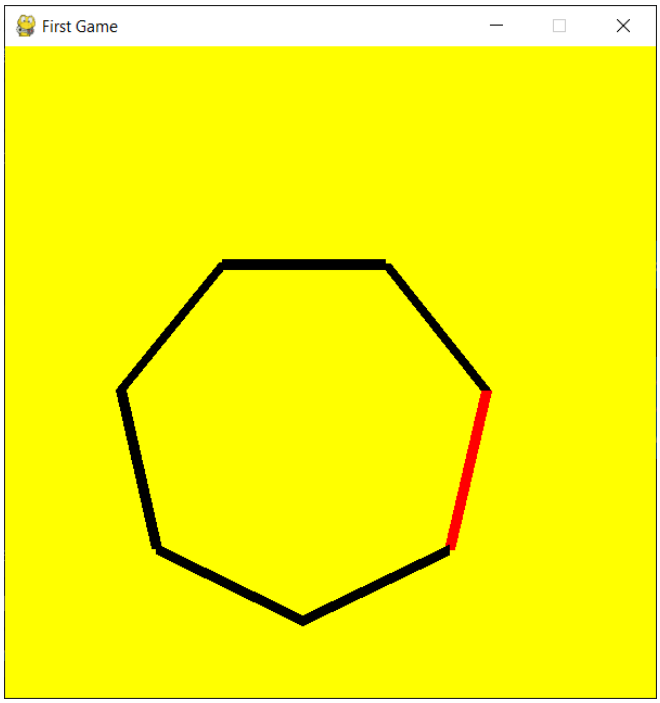
```
                xc = 275
                yc = initY + 335 / 2
                xc2, yc2 = 143, 363
                initX += xc - xc2
                initY += yc - yc2
                scaleX = 1.5
                scaleY = 0.5
                initY += 100
            case "9": #pochyl figurę oraz obróć ją o 180 stopni i dosuń do
prawej krawędi ekranu
                slash = 30
                xc, yc = drawFigure()
                phase = 180.0 * math.pi / 180.0  # 45 deg to rad
                xc2, yc2 = drawFigure()
                initX += xc - xc2+160
                initY += yc - yc2

        drawFigure()

    lastDrawMode = currentDrawMode
    pygame.display.update()
```

Kod źródłowy programu 2:

```
import pygame

pygame.init()
win = pygame.display.set_mode((600, 600))
pygame.display.set_caption("First Game")

# deklarowanie kolorów
CZERWONY = (255, 0, 0)
ZIELONY = (0, 255, 0)
ZOLTY = (255, 255, 0)
FIOLETOWY = (128, 0, 128)
JASNY_NIEBIESKI = (0, 255, 255)
POMARANCZOWY = (255, 165, 0)
NIEBIESKI = (0, 0, 255)
SZARY = (128, 128, 128)
CZARNY = (0, 0, 0)

initX, initY = 100, 200




run = True
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    pygame.draw.rect(win, NIEBIESKI, ( initX, initY, 400, 200))
    pygame.draw.polygon(win, NIEBIESKI, [( initX+200, initY), (initX+100,
initY-150), (initX+300, initY-150)] )
    pygame.draw.polygon(win, NIEBIESKI, [( initX+200, initY+200),
(initX+100, initY+350), (initX+300, initY+350)] )



    pygame.display.update()
```

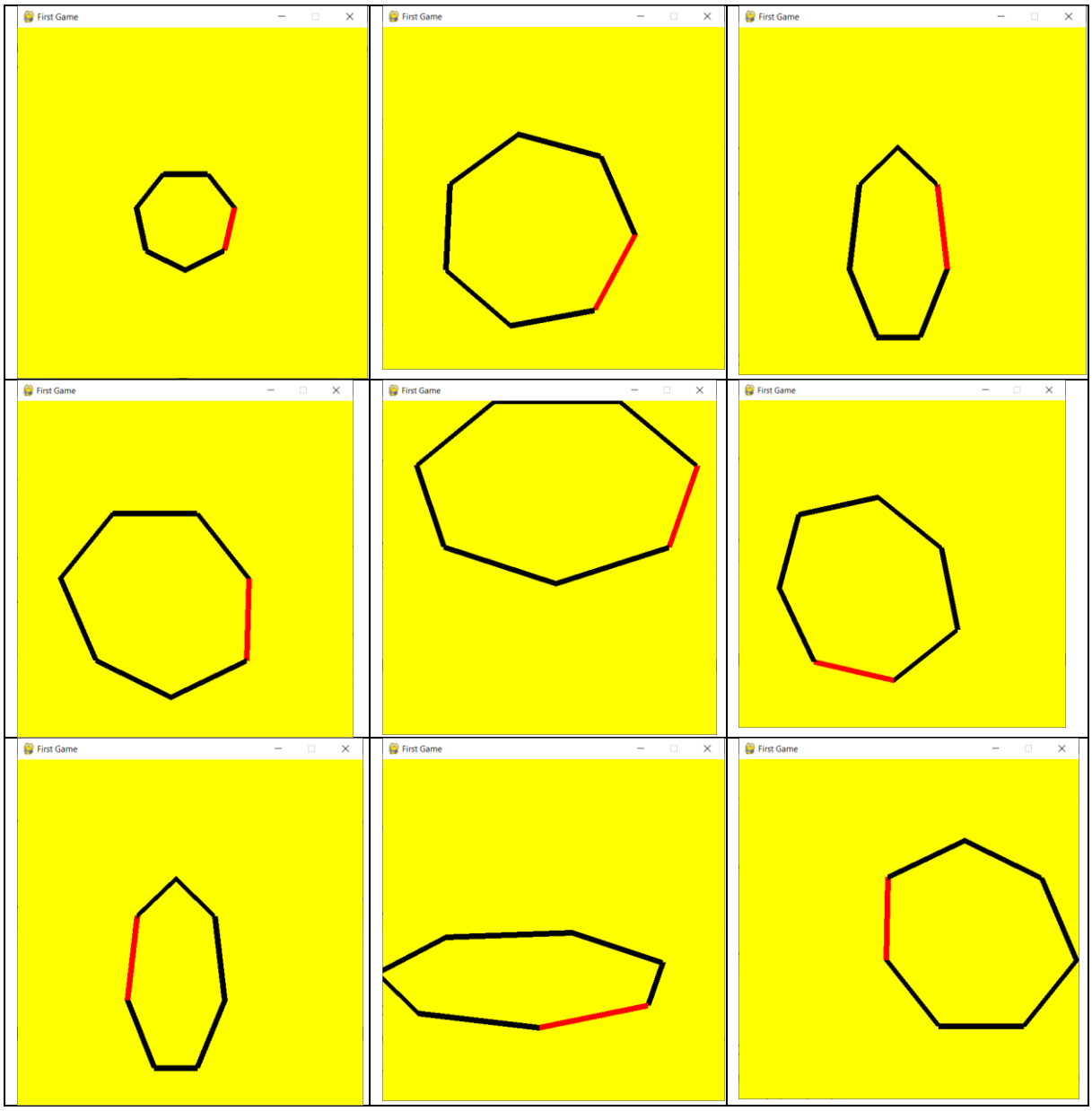
5. Wynik działania

Wyniki wykonania pierwszego programu w zależności od zastosowanego przekształcenia przedstawiono w tabeli poniżej. Dodatkowo jedna krawędź figury została przemalowana na czerwono aby łatwiej było obserwować prawidłowość wykonania przekształceń (głównie obrotów i symetrycznych odbić).

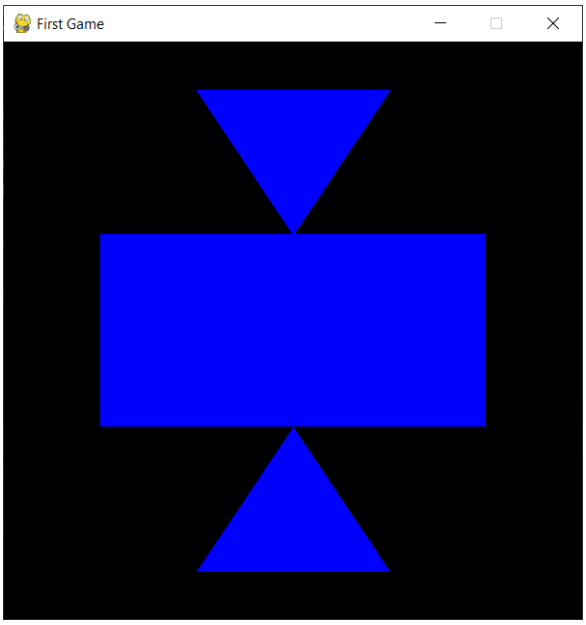


Zdjęcie 1 Bazowa figura w zadaniu 1

Tabela 1 Wyniki operacji po naciśnięciu klawiszy 1-9



Wynik wykonania programu 2 przedstawiono poniżej:



Zdjęcie 2 Wynik wykonania programu 2

6. Wnioski

- Biblioteka pygame to stosunkowo łatwe do opanowania narzędzie pozwalające na tworzenie różnego rodzaju prostych rysunków, które mogą być dynamicznie przekształcane.
- Wszystkie klatki są renderowane kilkadziesiąt razy na sekundę, natomiast jeżeli wiadomo iż pewna część interfejsu musi być odświeżana tylko w szczególnych przypadkach (przez większość czasu zachowuje się jak obiekt statyczny), warto zaimplementować flagę sterującą wykonaniem ponownego przerysowania tego obiektu