Sprawozdanie

Zajęcia: Grafika Komputerowa

Prowadzący: mgr inż. Mikołaj Grygiel

Laboratorium Grafiki Komputerowej

24.03.2024

Temat: „Modelowanie hierarchiczne w grafice 2D”

Wariant: 7

Albert Więcaszek

053012

Informatyka I stopień

Zaoczne

4 semestr

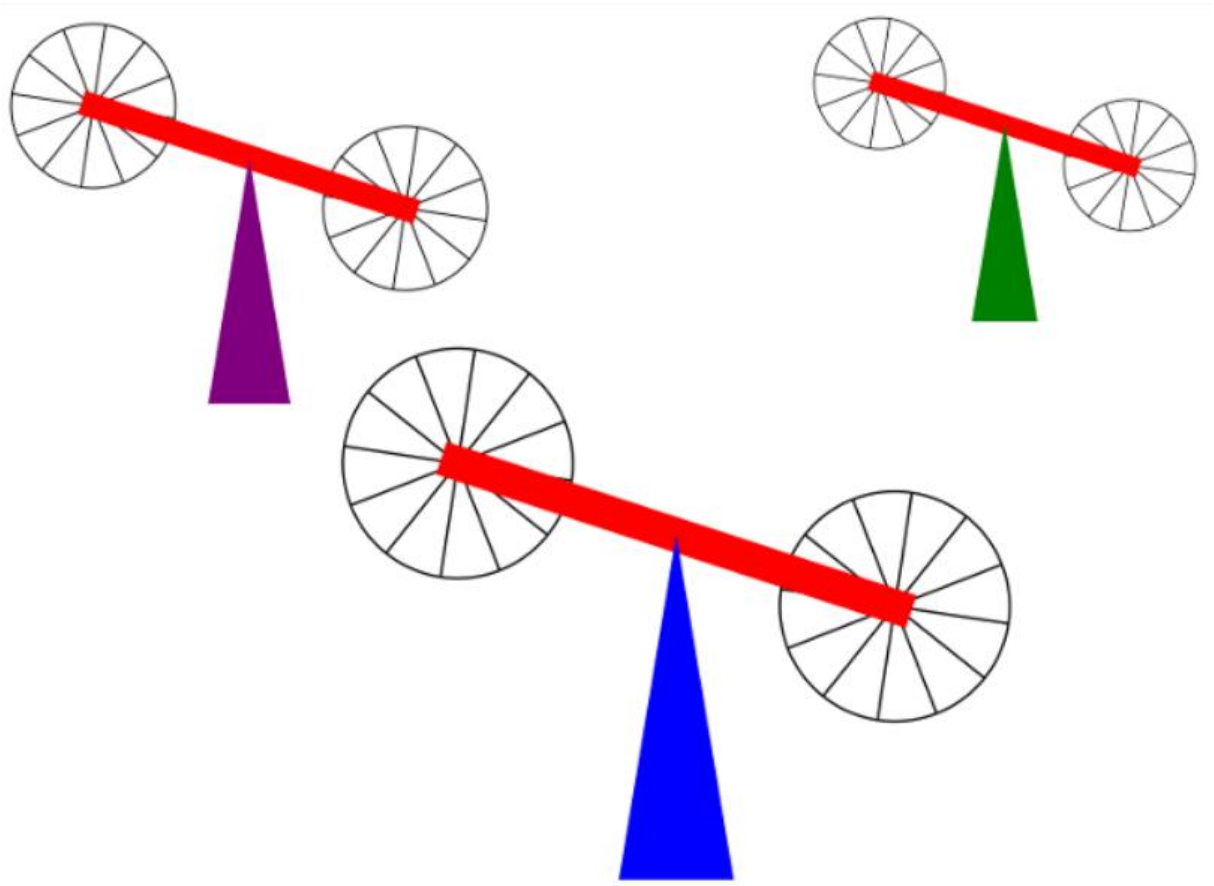
Gr. 2B

1. Polecenia

Opracować scenę hierarchiczną zgodnie z obrazem używając zamiast kół wielokąty obracające się (animacja!) według wariantu. Opracowanie powinno być w jednym z języków: Java lub JavaScript.

a) Używając hierarchiję funkcje (sposób subroutinowy)
b) Tworząc graf sceny (sposób obiektowy)

Efektem powinien być układ widoczny na Zdjęcie 1. Zamiast okręgów należy narysować wielokąt o liczbie krawędzi odpowiadającej numerowi zadania (w tym przykładzie 7-bok foremny).



Zdjęcie 1 Docelowy kształt do uzyskania

2. Wprowadzane dane

Do wykonania zadania sposobem „a" najpierw przygotowano metody pozwalające w sparametryzowany sposób rysować linie, trójkąty oraz siedmiokąty.

```
private static void line(Graphics2D g2,Color color,float width,double x1,
double y1, double x2, double y2) { // Draws a line from (-0.5,0) to (0.5,0)
   g2.setColor(color);
   g2.setStroke(new BasicStroke(width));
   g2.draw( new Line2D.Double( x1,y1,x2,y2) );
}

private static void filledTriangle(Graphics2D g2,Color color,double
x1,double y1,double x2, double y2, double x3, double y3) { // width = 1,
```

```
height = 1, center of base is at (0,0);
   Path2D path = new Path2D.Double();
   path.moveTo(x1,y1);
   path.lineTo(x2,y2);
   path.lineTo(x3,y3);
   path.closePath();
   g2.setColor(color);
   g2.fill(path);
}

private static void siedmiokat(Graphics2D g2, Color color,double xCentre,
double yCentre,double radius){
   int n= 8;
   double[] x = new double[n];
   double[] y = new double[n];

   double theta = Math.PI - 2.24285714286;
   g2.setColor(color);
   g2.setStroke(new BasicStroke(0.02f));

   for (int i=0;i<8;i++){
      x[i] = xCentre + radius * Math.cos(i*theta+frameNumber*0.25);
      y[i] = yCentre + radius * Math.sin(i*theta+frameNumber*0.25);
      if (i!=0){
         g2.draw( new Line2D.Double( x[i-1],y[i-1],x[i],y[i]) );
         g2.draw( new Line2D.Double( x[i],y[i],xCentre,yCentre) );
      }
   }
}
```

Następnie wewnątrz metody *drawWorld()* wywołano zdefiniowane wcześniej funkcje w celu narysowania wymaganych obiektów.

```
siedmiokat(g2,Color.black,-2,1.25,0.3);
siedmiokat(g2,Color.black,-1,0.75,0.3);
siedmiokat(g2,Color.black,1,1.25,0.3);
siedmiokat(g2,Color.black,2,0.75,0.3);
siedmiokat(g2,Color.black,-1.25,-0.75,0.5);
siedmiokat(g2,Color.black,1.25,-1.25,0.5);

line(g2,Color.red,0.1f,-2,1.25,-1,0.75);
line(g2,Color.red,0.1f,1,1.25,2,0.75);
line(g2,Color.red,0.15f,-1.25,-0.75,1.25,-1.25);

filledTriangle(g2,Color.magenta,-1.75,0,-1.25,0,-1.5,1);
filledTriangle(g2,Color.green,1.75,0,1.25,0,1.5,1);
filledTriangle(g2,Color.blue,-0.5,-3,0.5,-3,0,-1);
```

W celu wykonania zadania metodą „b” przygotowano listy przechowujące odpowiednie obiekty (linie, trójkąty oraz siedmiokąty).

```
private static ArrayList<TransformedObject> siedmiobokLines = new
ArrayList<>();
private static ArrayList<SceneGraphNode> triangles = new ArrayList<>();
private static ArrayList<SceneGraphNode> redLines = new ArrayList<>();
```

Następnie utworzono listy zawierające współrzędne punktów wymaganych do narysowania wymaganych kształtów (wierzchołków trójkąta, brzegów linii oraz środków siedmiokątów). Dodano również listę zawierającą mapę kolorów.

```
double[] triangXY = new double[]{-1.75,0,-1.25,0,-
1.5,1,1.75,0,1.25,0,1.5,1,-0.5,-3,0.5,-3,0,-1};
double[] linesXY = new double[]{-2,1.25,-1,0.75,1,1.25,2,0.75,-1.25,-
0.75,1.25,-1.25};
double[] centres = new double[]{-2,1.25,-1,0.75,1,1.25,2,0.75,-1.25,-
0.75,1.25,-1.25};
Color[] cMap = new
Color[]{Color.magenta,Color.green,Color.blue,Color.red,Color.black};
```

Kolejnym krokiem było utworzenie obiektów opisywanych przez zdefiniowane wcześniej punkty. W tym celu użyto pętli *for* aby oszczędzić liczbę wymaganych linii kodu.

```
for (int i=0; i<3;i++){
   int finalI = i;
   triangles.add(new SceneGraphNode() {
      @Override
      void doDraw(Graphics2D g) {
         Path2D path = new Path2D.Double();
         path.moveTo(triangXY[6* finalI],triangXY[6* finalI+1]);
         path.lineTo(triangXY[6* finalI+2],triangXY[6* finalI+3]);
         path.lineTo(triangXY[6* finalI+4],triangXY[6* finalI+5]);
         path.closePath();
         g.fill(path);
      }
   });
   redLines.add(new SceneGraphNode() {
      @Override
      void doDraw(Graphics2D g) {
         g.setStroke(new BasicStroke(0.1f));
         g.draw( new Line2D.Double( linesXY[4* finalI],linesXY[4*
finalI+1],
               linesXY[4* finalI+2],linesXY[4* finalI+3]) );
      }
   });
}

for (int i=0; i<centres.length/2;i++) {
   int finalI = i;
   siedmiobokLines.add(new TransformedObject(new SceneGraphNode() {
      @Override
      void doDraw(Graphics2D g) {
         int n= 8;
         double[] x = new double[n];
         double[] y = new double[n];

         double theta = Math.PI - 2.24285714286;
         g.setColor(color);
         g.setStroke(new BasicStroke(0.02f));

         for (int i=0;i<8;i++){
            x[i] = centres[2*finalI] + 0.3 *
Math.cos(i*theta+frameNumber*0.25);
            y[i] = centres[2*finalI+1] + 0.3 *
Math.sin(i*theta+frameNumber*0.25);
            if (i!=0){
               g.draw( new Line2D.Double( x[i-1],y[i-1],x[i],y[i]) );
```

```
                g.draw( new Line2D.Double(
x[i],y[i],centres[2*finalI],centres[2*finalI+1]) );
            }
        }

    }
   }));
}
```

Finalnie przeliterowano listy zawierające obiekty, nadano im kolory oraz dodano do sceny o nazwie *world*.

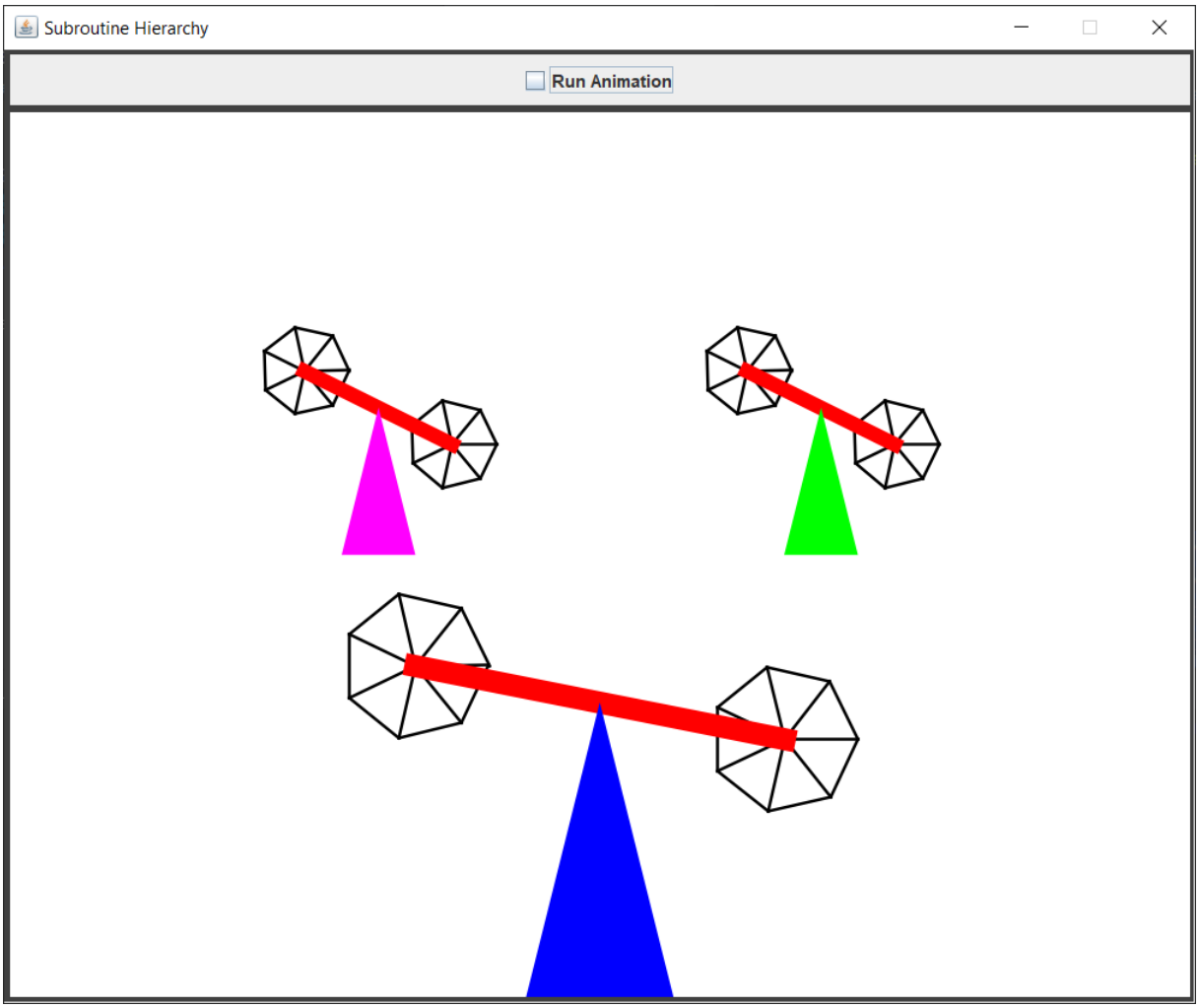
```
for (SceneGraphNode siedmiobokLine : siedmiobokLines) {
   siedmiobokLine.setColor(cMap[4]);
   world.add(siedmiobokLine);
}

for (SceneGraphNode line : redLines) {
   line.setColor(cMap[3]);
   world.add(line);
}


for (SceneGraphNode triangle : triangles) {
   triangle.setColor(cMap[triangles.indexOf(triangle)]);
   world.add(triangle);
}
```

3. Wykorzystane komendy

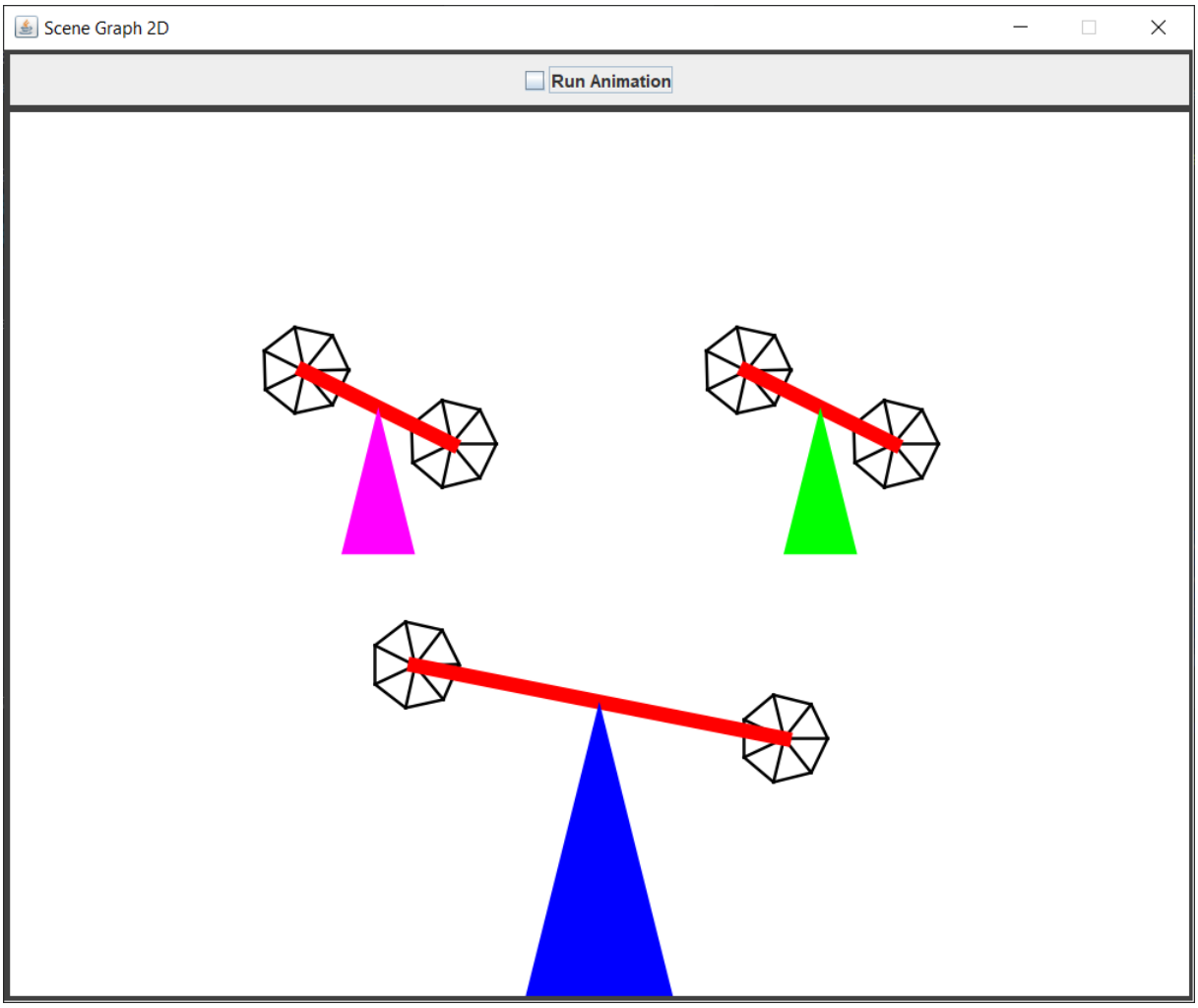
Kod źródłowy programu 1 i 2 ze względu na długość i przejrzystość kodu został umieszczony jako osobny plik w załączniku do sprawozdania.

4. Wynik działania

Wyniki wykonania programu obiema metodami zostały przedstawione na Zdjęcie 2 oraz Zdjęcie 3. Statyczne obrazy wklejone do sprawozdania nie pozwalają zaobserwować iż po zaznaczeniu CheckBox'a *Run Animation* siedmiokąty zaczynają się kręcić. Obroty te można zaobserwować wykonując kody źródłowe dołączone w załącznikach do sprawozdania.



Zdjęcie 2 Wynik wykonania programu metodą "a"



Zdjęcie 3 Wynik wykonania programu metodą "b"

5. Wnioski

- Kolejność w jakiej obiekty są dodawane do Canvy decyduje o tym które z nich będą wyświetlane pod spodem (dodane jako pierwsze), a które na wierzchu (dodane jako ostatnie).