**SPRAWOZDANIE**

Zajęcia: Grafika komputerowa

Prowadzący: mgr Mikołaj Grygiel

**Laboratorium 4**

24.03.2024

**Temat:** Modelowanie hierarchiczne w grafice 2D

**Wariant**: 1

Radosław Skrzypczyński

Informatyka I stopień,

stacjonarne,

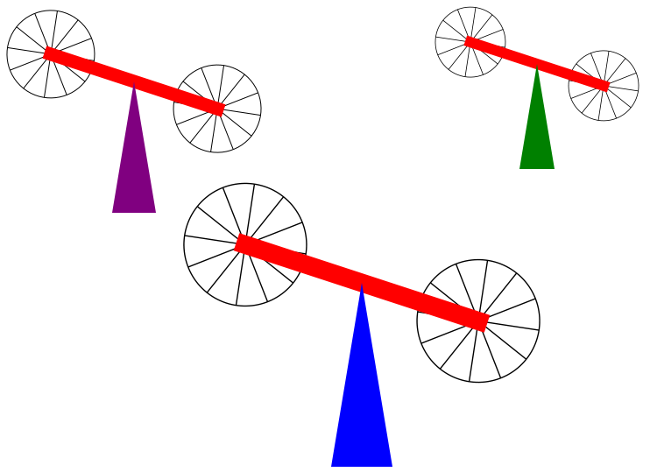
4 semestr,

Gr. 1B

1. **Polecenie:**

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, na dwa sposoby:

1. używając hierarchiję funkcje (sposób subroutinowy)
2. tworząc graf sceny (sposób obiektowy). W tym celu proponuję do pobrania odpowiedni pliki



**2. Wprowadzane dane:**

Przycisk uruchamiający i zatrzymujący animację.

**3. Wykorzystane komendy:**

a) hierarchia funkcji

*<!DOCTYPE html>*

*<html>*

*<head>*

*<meta charset="UTF-8">*

*<title>Subroutine Hieararchy</title>*

*<script>*

*var canvas; // The canvas that is used as the drawing surface*

*var graphics; // The 2D graphics context for drawing on the canvas.*

*var X\_LEFT = -4; // The xy limits for the coordinate system.*

*var X\_RIGHT = 4;*

*var Y\_BOTTOM = -3;*

*var Y\_TOP = 3;*

*var BACKGROUND = "white"; // The display is filled with this color before the scene is drawn.*

*var pixelSize; // The size of one pixel, in the transformed coordinates.*

*var frameNumber = 0; // Current frame number. goes up by one in each frame.*

*// TODO: Define any other necessary state variables.*

*/\*\**

*\* Responsible for drawing the entire scene. The display is filled with the background*

*\* color before this function is called.*

*\*/*

*function drawWorld() {*

*// TODO: Draw the content of the scene.*

*//rotatingRect(); // (DELETE THIS EXAMPLE)*

*staticElements();*

*rotatingPolygons();*

*}*

*/\*\**

*\* This method is called just before each frame is drawn. It updates the modeling*

*\* transformations of the objects in the scene that are animated.*

*\*/*

*function updateFrame() {*

*frameNumber++;*

*// TODO: If other updates are needed for the next frame, do them here.*

*}*

*// TODO: Define methods for drawing the objects in the scene.*

*function staticElements() {*

*filledSpike(0, -1, 1, "blue"); //tipX, tipY, scale, color*

*filledSpike(-1.8, 0, 0.9, "purple"); //tipX, tipY, scale, color*

*filledSpike(2, 0.5, 0.8, "green"); //tipX, tipY, scale, color*

*//filledPolygon(1, 1, 0.8, 7, "orange"); // Dodanie siedmiokąta*

*//filledPolygon(-2, 2, 0.5, 7, "yellow"); // Dodanie siedmiokąta*

*//filledPolygon(-1.1, -2, 0.7, 7, "pink");*

*//filledPolygon(-1.3, -2, 0.9, 7, "pink2");*

*//filledPolygon(-1.4, -2, 0.7, 7, "pink3");*

*//filledPolygon(-1.7, -2, 0.7, 7, "pink4"); // Dodanie siedmiokąta*

*}*

*function rotatingPolygons(){*

*//graphics.save();*

*//graphics.fillStyle = "red";*

*//graphics.rotate( (frameNumber\*0.75) \* Math.PI/180 );*

*//graphics.scale( 0.5, 0.5 );*

*rotatingBar(0, -1, 1.5, 0.2, 45, "red"); //centerX, centerY, width, height, angle, color*

*rotatingBar(-1.8, 0, 1.5, 0.2, 45, "red"); //centerX, centerY, width, height, angle, color*

*rotatingBar(2, 0.5, 1.5, 0.2, 45, "red"); //centerX, centerY, width, height, angle, color*

*//rotatingPolygon(-0.75, -1, 1, 7, "orange"); // Obracanie siedmiokąta*

*//rotatingPolygon(-1.8, 0, 1, 7, "yellow"); // Obracanie siedmiokąta*

*//rotatingPolygon(2, 0.5, 1, 7, "pink");*

*//filledPolygon(-1, -2, 1, 7, "pink2");*

*//filledPolygon(-1, -2, 1, 7, "pink3");*

*//filledPolygon(-1, -2, 1, 7, "pink4"); // Obracanie siedmiokąta*

*//graphics.restore();*

*}*

*function rotatingPolygon(centerX, centerY, scale, sides, color){*

*graphics.save();*

*//graphics.fillStyle = "red";*

*graphics.translate(centerX, centerY);*

*graphics.rotate( (frameNumber\*0.75) \* Math.PI/180 );*

*graphics.scale( 0.5, 0.5 );*

*filledPolygon(0, 0, scale, sides, color);*

*graphics.restore();*

*}*

*function filledSpike(tipX, tipY, scale, color){*

*let width = 0.4 \* scale;*

*let height = 1.2 \* scale;*

*graphics.beginPath();*

*graphics.fillStyle = color;*

*graphics.moveTo(tipX - width/2, tipY - height);*

*graphics.lineTo(tipX + width/2, tipY - height);*

*graphics.lineTo(tipX, tipY);*

*graphics.closePath();*

*graphics.fill();*

*}*

*function rotatingBar(centerX, centerY, width, height, angle, color){*

*//graphics.beginPath();*

*graphics.save();*

*graphics.fillStyle = color;*

*graphics.translate(centerX, centerY);*

*graphics.rotate( (frameNumber\*0.75-45) \* Math.PI/180 );*

*graphics.fillRect(-width/2,-height/4, width, height/2);*

*filledPolygon(-width/2, 0, 0.4, 5, "black");*

*filledPolygon(width/2, 0, 0.4, 5, "black");*

*graphics.restore();*

*}*

*function filledPolygon(centerX, centerY, scale, sides, color){*

*graphics.save();*

*graphics.strokeStyle = color; // Używamy strokeStyle zamiast fillStyle*

*graphics.lineWidth = 0.05;*

*graphics.translate(centerX, centerY);*

*graphics.scale(scale, scale);*

*graphics.beginPath();*

*var angle = (Math.PI \* 2) / sides;*

*var vertices = [];*

*for (var i = 0; i < sides; i++) {*

*var x = Math.cos(i \* angle);*

*var y = Math.sin(i \* angle);*

*vertices.push({x: x, y: y});*

*if (i === 0) {*

*graphics.moveTo(x, y);*

*} else {*

*graphics.lineTo(x, y);*

*}*

*}*

*graphics.closePath();*

*graphics.stroke();*

*// Rysowanie linii od środka do wierzchołków*

*graphics.beginPath();*

*for (var i = 0; i < sides; i++) {*

*graphics.moveTo(0, 0); // Środek wielokąta*

*graphics.lineTo(vertices[i].x, vertices[i].y);*

*}*

*graphics.stroke();*

*graphics.restore();*

*}*

*// ------------------------------- graphics support functions --------------------------*

*/\*\**

*\* Draw one frame of the animation. Probably doesn't need to be changed,*

*\* except maybe to change the setting of preserveAspect in applyLimits().*

*\*/*

*function draw() {*

*graphics.save(); // to make sure changes don't carry over from one call to the next*

*graphics.fillStyle = BACKGROUND*

*; // background color*

*graphics.fillRect(0,0,canvas.width,canvas.height);*

*graphics.fillStyle = "black";*

*applyLimits(graphics,X\_LEFT,X\_RIGHT,Y\_TOP,Y\_BOTTOM,false);*

*graphics.lineWidth = pixelSize; // Use 1 pixel as the default line width*

*drawWorld();*

*graphics.restore();*

*}*

*/\*\**

*\* Applies a coordinate transformation to the graphics context, to map*

*\* xleft,xright,ytop,ybottom to the edges of the canvas. This is called*

*\* by draw(). This does not need to be changed.*

*\*/*

*function applyLimits(g, xleft, xright, ytop, ybottom, preserveAspect) {*

*var width = canvas.width; // The width of this drawing area, in pixels.*

*var height = canvas.height; // The height of this drawing area, in pixels.*

*if (preserveAspect) {*

*// Adjust the limits to match the aspect ratio of the drawing area.*

*var displayAspect = Math.abs(height / width);*

*var requestedAspect = Math.abs(( ybottom-ytop ) / ( xright-xleft ));*

*var excess;*

*if (displayAspect > requestedAspect) {*

*excess = (ybottom-ytop) \* (displayAspect/requestedAspect - 1);*

*ybottom += excess/2;*

*ytop -= excess/2;*

*}*

*else if (displayAspect < requestedAspect) {*

*excess = (xright-xleft) \* (requestedAspect/displayAspect - 1);*

*xright += excess/2;*

*xleft -= excess/2;*

*}*

*}*

*var pixelWidth = Math.abs(( xright - xleft ) / width);*

*var pixelHeight = Math.abs(( ybottom - ytop ) / height);*

*pixelSize = Math.min(pixelWidth,pixelHeight);*

*g.scale( width / (xright-xleft), height / (ybottom-ytop) );*

*g.translate( -xleft, -ytop );*

*}*

*//------------------ Animation framework ------------------------------*

*var running = false; // This is set to true when animation is running*

*function frame() {*

*if (running) {*

*// Draw one frame of the animation, and schedule the next frame.*

*updateFrame();*

*draw();*

*requestAnimationFrame(frame);*

*}*

*}*

*function doAnimationCheckbox() {*

*var shouldRun = document.getElementById("animateCheck").checked;*

*if ( shouldRun != running ) {*

*running = shouldRun;*

*if (running)*

*requestAnimationFrame(frame);*

*}*

*}*

*//----------------------- initialization -------------------------------*

*function init() {*

*canvas = document.getElementById("thecanvas");*

*if (!canvas.getContext) {*

*document.getElementById("message").innerHTML = "ERROR: Canvas not supported";*

*return;*

*}*

*graphics = canvas.getContext("2d");*

*document.getElementById("animateCheck").checked = false;*

*document.getElementById("animateCheck").onchange = doAnimationCheckbox;*

*draw();*

*}*

*</script>*

*</head>*

*<body onload="init()" style="background-color:#EEEEEE">*

*<h3>Subroutine Hierarchy</h3>*

*<noscript>*

*<p><b style="color:red">Error: This page requires JavaScript, but it is not available.</b></p>*

*</noscript>*

*<p id="message"><label><input type="checkbox" id="animateCheck"><b>Run the Animation</b></label></p>*

*<div style="float:left; border: 2px solid black">*

*<canvas id="thecanvas" width="800" height="600" style="display:block"></canvas>*

*</div>*

*</body>*

*</html>*

b) graf sceny

*<!DOCTYPE html>*

*<html>*

*<head>*

*<meta charset="UTF-8">*

*<title>Scene Graph 2D</title>*

*<script>*

*var canvas; // The canvas that is used as the drawing surface*

*var graphics; // The 2D graphics context for drawing on the canvas.*

*var X\_LEFT = -4; // The xy limits for the coordinate system.*

*var X\_RIGHT = 4;*

*var Y\_BOTTOM = -3;*

*var Y\_TOP = 3;*

*var BACKGROUND = "white"; // The display is filled with this color before the scene is drawn.*

*var pixelSize; // The size of one pixel, in the transformed coordinates.*

*var frameNumber = 0; // Current frame number. goes up by one in each frame.*

*var world; // A SceneGraphNode representing the entire scene.*

*var rotatingBar1;*

*var rotatingBar2;*

*var rotatingBar3;*

*var rotatingPolygon1;*

*var rotatingPolygon2;*

*var rotatingPolygon3;*

*var polygonSides = 5;*

*var barWitdh = 1.5;*

*var spike1x = -2.4;*

*var spike1y = 0.3;*

*var spike2x = 2.2;*

*var spike2y = 0.4;*

*var spike3x = 0;*

*var spike3y = -0.2;*

*/\*\**

*\* Builds the data structure that represents the entire picture.*

*\*/*

*function createWorld() {*

*world = new CompoundObject(); // Root node for the scene graph.*

*let filledSpike1 = new TransformedObject(filledSpike);*

*filledSpike1.setColor("purple").setTranslation(spike1x,spike1y).setScale(1.2,1.2);*

*let filledSpike2 = new TransformedObject(filledSpike);*

*filledSpike2.setColor("green").setTranslation(spike2x,spike2y).setScale(1,1);*

*let filledSpike3 = new TransformedObject(filledSpike);*

*filledSpike3.setColor("blue").setTranslation(spike3x,spike3y).setScale(1,1);*

*rotatingBar1 = new TransformedObject(filledBar);*

*rotatingBar1.setColor("red").setTranslation(spike1x,spike1y).setScale(barWitdh,barWitdh);*

*rotatingBar2 = new TransformedObject(filledBar);*

*rotatingBar2.setColor("red").setTranslation(spike2x,spike2y).setScale(barWitdh,barWitdh);;*

*rotatingBar3 = new TransformedObject(filledBar);*

*rotatingBar3.setColor("red").setTranslation(spike3x,spike3y).setScale(barWitdh,barWitdh);*

*//rotatingPolygon1 = new TransformedObject(filledPentagon);*

*//rotatingPolygon1.setTranslation(spike1x-barWitdh/2, spike1y).setScale(0.5, 0.5);*

*world.add(filledSpike1);*

*world.add(rotatingBar1);*

*//world.add(rotatingPolygon1);*

*world.add(filledSpike2);*

*world.add(rotatingBar2);*

*world.add(filledSpike3);*

*world.add(rotatingBar3);*

*//filledfilledSpike1 = new*

*// TODO: Create objects and add them to the scene graph.*

*//rotatingRect = new TransformedObject(filledSpike); // (DELETE THIS EXAMPLE)*

*//rotatingRect.setScale(2,2).setColor("red");*

*//world.add(rotatingRect);*

*}*

*/\*\**

*\* This method is called just before each frame is drawn. It updates the modeling*

*\* transformations of the objects in the scene that are animated.*

*\*/*

*function updateFrame() {*

*frameNumber++;*

*// TODO: Update state in preparation for drawing the next frame.*

*//rotatingRect.setRotation(frameNumber\*0.75); // (DELETE THIS EXAMPLE)*

*rotatingBar1.setRotation(360-frameNumber\*0.75);*

*rotatingBar2.setRotation(360-frameNumber\*0.75);*

*rotatingBar3.setRotation(360-frameNumber\*0.75);*

*//rotatingPolygon1.setRotation(360-frameNumber\*1.5);*

*}*

*//------------------- A Simple Scene Object-Oriented Scene Graph API ----------------*

*/\*\**

*\* The (abstract) base class for all nodes in the scene graph data structure.*

*\*/*

*function SceneGraphNode() {*

*this.fillColor = null; // If non-null, the default fillStyle for this node.*

*this.strokeColor = null; // If non-null, the default strokeStyle for this node.*

*}*

*SceneGraphNode.prototype.doDraw = function(g) {*

*// This method is meant to be abstract and must be OVERRIDDEN in an actual*

*// object. It is not meant to be called; it is called by draw().*

*throw "doDraw not implemented in SceneGraphNode"*

*}*

*SceneGraphNode.prototype.draw = function(g) {*

*// This method should be CALLED to draw the object It should NOT*

*// ordinarily be overridden in subclasses.*

*graphics.save();*

*if (this.fillColor) {*

*g.fillStyle = this.fillColor;*

*}*

*if (this.strokeColor) {*

*g.strokeStyle = this.strokeColor;*

*}*

*this.doDraw(g);*

*graphics.restore();*

*}*

*SceneGraphNode.prototype.setFillColor = function(color) {*

*// Sets fillColor for this node to color.*

*// Color should be a legal CSS color string, or null.*

*this.fillColor = color;*

*return this;*

*}*

*SceneGraphNode.prototype.setStrokeColor = function(color) {*

*// Sets strokeColor for this node to color.*

*// Color should be a legal CSS color string, or null.*

*this.strokeColor = color;*

*return this;*

*}*

*SceneGraphNode.prototype.setColor = function(color) {*

*// Sets both the fillColor and strokeColor to color.*

*// Color should be a legal CSS color string, or null.*

*this.fillColor = color;*

*this.strokeColor = color;*

*return this;*

*}*

*/\*\**

*\* Defines a subclass, CompoundObject, of SceneGraphNode to represent*

*\* an object that is made up of sub-objects. Initially, there are no*

*\* sub-objects. Objects are added with the add() method.*

*\*/*

*function CompoundObject() {*

*SceneGraphNode.call(this); // do superclass initialization*

*this.subobjects = []; // the list of sub-objects of this object*

*}*

*CompoundObject.prototype = new SceneGraphNode(); // (makes it a subclass!)*

*CompoundObject.prototype.add = function(node) {*

*this.subobjects.push(node);*

*return this;*

*}*

*CompoundObject.prototype.doDraw = function(g) {*

*// Just call the sub-objects' draw() methods.*

*for (var i = 0; i < this.subobjects.length; i++)*

*this.subobjects[i].draw(g);*

*}*

*/\*\**

*\* Define a subclass, TransformedObject, of SceneGraphNode that*

*\* represents an object along with a modeling transformation to*

*\* be applied to that object. The object must be specified in*

*\* the constructor. The transformation is specified by calling*

*\* the setScale(), setRotate() and setTranslate() methods. Note that*

*\* each of these methods returns a reference to the TransformedObject*

*\* as its return value, to allow for chaining of method calls.*

*\* The modeling transformations are always applied to the object*

*\* in the order scale, then rotate, then translate.*

*\*/*

*function TransformedObject(object) {*

*SceneGraphNode.call(this); // do superclass initialization*

*this.object = object;*

*this.rotationInDegrees = 0;*

*this.scaleX = 1;*

*this.scaleY = 1;*

*this.translateX = 0;*

*this.translateY = 0;*

*}*

*TransformedObject.prototype = new SceneGraphNode(); // (makes it a subclass!)*

*TransformedObject.prototype.setRotation = function(angle) {*

*// Set the angle of rotation, measured in DEGREES. The rotation*

*// is always about the origin.*

*this.rotationInDegrees = angle;*

*return this;*

*}*

*TransformedObject.prototype.setScale = function(sx, sy) {*

*// Sets scaling factors.*

*this.scaleX = sx;*

*this.scaleY = sy;*

*return this;*

*}*

*TransformedObject.prototype.setTranslation = function(dx,dy) {*

*// Set translation mounts.*

*this.translateX = dx;*

*this.translateY = dy;*

*return this;*

*}*

*TransformedObject.prototype.doDraw = function(g) {*

*// Draws the object, with its modeling transformation.*

*g.save();*

*if (this.translateX != 0 || this.translateY != 0) {*

*g.translate(this.translateX, this.translateY);*

*}*

*if (this.rotationInDegrees != 0) {*

*g.rotate(this.rotationInDegrees/180\*Math.PI);*

*}*

*if (this.scaleX != 1 || this.scaleY != 1) {*

*g.scale(this.scaleX, this.scaleY);*

*}*

*this.object.draw(g);*

*g.restore();*

*}*

*var filledSpike = new SceneGraphNode();*

*filledSpike.doDraw = function(g){*

*let width = 0.4;*

*let height = 1.6;*

*graphics.beginPath();*

*graphics.moveTo(0 - width/2, 0 - height);*

*graphics.lineTo(0 + width/2, 0 - height);*

*graphics.lineTo(0, 0);*

*graphics.closePath();*

*graphics.fill();*

*}*

*var filledBar = new SceneGraphNode(); // Filled square, size = 1, center = (0,0)*

*filledBar.doDraw = function(g) {*

*g.fillRect(-0.5,-0.03, 1, 0.06);*

*DrawPolygon(g,-0.5,0,polygonSides,0.75,"black");*

*DrawPolygon(g,0.5,0,polygonSides,0.75,"black");*

*}*

*function DrawPolygon(g, centerX, centerY, sides, scale, color){*

*//g.fillColor = color;*

*g.strokeStyle = color;*

*scale = scale \* 0.3;*

*g.beginPath();*

*g.moveTo(centerX, centerY);*

*var angle = (Math.PI \* 2) / sides;*

*for (var i = 0; i <= sides; i++) {*

*var x = Math.cos(i \* angle);*

*var y = Math.sin(i \* angle);*

*if (i === 0) {*

*g.lineTo(centerX+x\*scale, centerY + y\*scale);*

*//g.moveTo(centerX+x\*scale, centerY + y\*scale);*

*} else {*

*g.lineTo(centerX+x\*scale, centerY + y\*scale);*

*g.lineTo(centerX, centerY);*

*g.moveTo(centerX+x\*scale, centerY + y\*scale);*

*}*

*}*

*g.closePath();*

*g.stroke();*

*}*

*// ------------------------------- graphics support functions --------------------------*

*/\*\**

*\* Draw one frame of the animation. Probably doesn't need to be changed,*

*\* except maybe to change the setting of preserveAspect in applyLimits().*

*\*/*

*function draw() {*

*graphics.save(); // to make sure changes don't carry over from one call to the next*

*graphics.fillStyle = BACKGROUND; // background color*

*graphics.fillRect(0,0,canvas.width,canvas.height);*

*graphics.fillStyle = "black";*

*applyLimits(graphics,X\_LEFT,X\_RIGHT,Y\_TOP,Y\_BOTTOM,false);*

*graphics.lineWidth = pixelSize; // Use 1 pixel as the default line width*

*world.draw(graphics);*

*graphics.restore();*

*}*

*/\*\**

*\* Applies a coordinate transformation to the graphics context, to map*

*\* xleft,xright,ytop,ybottom to the edges of the canvas. This is called*

*\* by draw(). This does not need to be changed.*

*\*/*

*function applyLimits(g, xleft, xright, ytop, ybottom, preserveAspect) {*

*var width = canvas.width; // The width of this drawing area, in pixels.*

*var height = canvas.height; // The height of this drawing area, in pixels.*

*if (preserveAspect) {*

*// Adjust the limits to match the aspect ratio of the drawing area.*

*var displayAspect = Math.abs(height / width);*

*var requestedAspect = Math.abs(( ybottom-ytop ) / ( xright-xleft ));*

*var excess;*

*if (displayAspect > requestedAspect) {*

*excess = (ybottom-ytop) \* (displayAspect/requestedAspect - 1);*

*ybottom += excess/2;*

*ytop -= excess/2;*

*}*

*else if (displayAspect < requestedAspect) {*

*excess = (xright-xleft) \* (requestedAspect/displayAspect - 1);*

*xright += excess/2;*

*xleft -= excess/2;*

*}*

*}*

*var pixelWidth = Math.abs(( xright - xleft ) / width);*

*var pixelHeight = Math.abs(( ybottom - ytop ) / height);*

*pixelSize = Math.min(pixelWidth,pixelHeight);*

*g.scale( width / (xright-xleft), height / (ybottom-ytop) );*

*g.translate( -xleft, -ytop );*

*}*

*//------------------ Animation framework ------------------------------*

*var running = false; // This is set to true when animation is running*

*function frame() {*

*if (running) {*

*// Draw one frame of the animation, and schedule the next frame.*

*updateFrame();*

*draw();*

*requestAnimationFrame(frame);*

*}*

*}*

*function doAnimationCheckbox() {*

*var shouldRun = document.getElementById("animateCheck").checked;*

*if ( shouldRun != running ) {*

*running = shouldRun;*

*if (running)*

*requestAnimationFrame(frame);*

*}*

*}*

*//----------------------- initialization -------------------------------*

*function init() {*

*canvas = document.getElementById("thecanvas");*

*if (!canvas.getContext) {*

*document.getElementById("message").innerHTML = "ERROR: Canvas not supported";*

*return;*

*}*

*graphics = canvas.getContext("2d");*

*document.getElementById("animateCheck").checked = false;*

*document.getElementById("animateCheck").onchange = doAnimationCheckbox;*

*createWorld();*

*draw();*

*}*

*</script>*

*</head>*

*<body onload="init()" style="background-color:#EEEEEE">*

*<h3>Scene Graph 2D</h3>*

*<noscript>*

*<p><b style="color:red">Error: This page requires JavaScript, but it is not available.</b></p>*

*</noscript>*

*<p id="message"><label><input type="checkbox" id="animateCheck"><b>Run the Animation</b></label></p>*

*<div style="float:left; border: 2px solid black">*

*<canvas id="thecanvas" width="800" height="600" style="display:block"></canvas>*

*</div>*

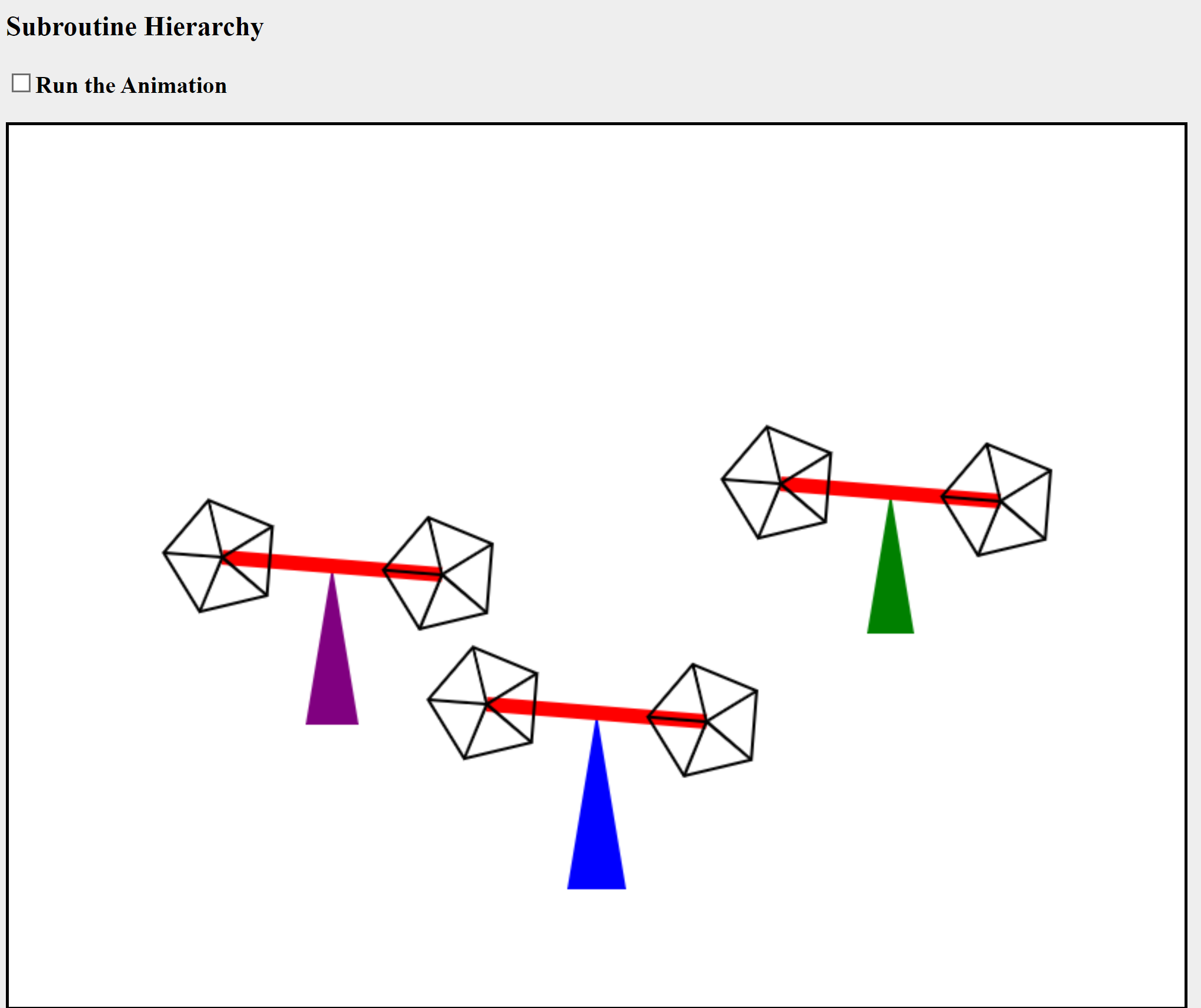
*</body>*

*</html>*

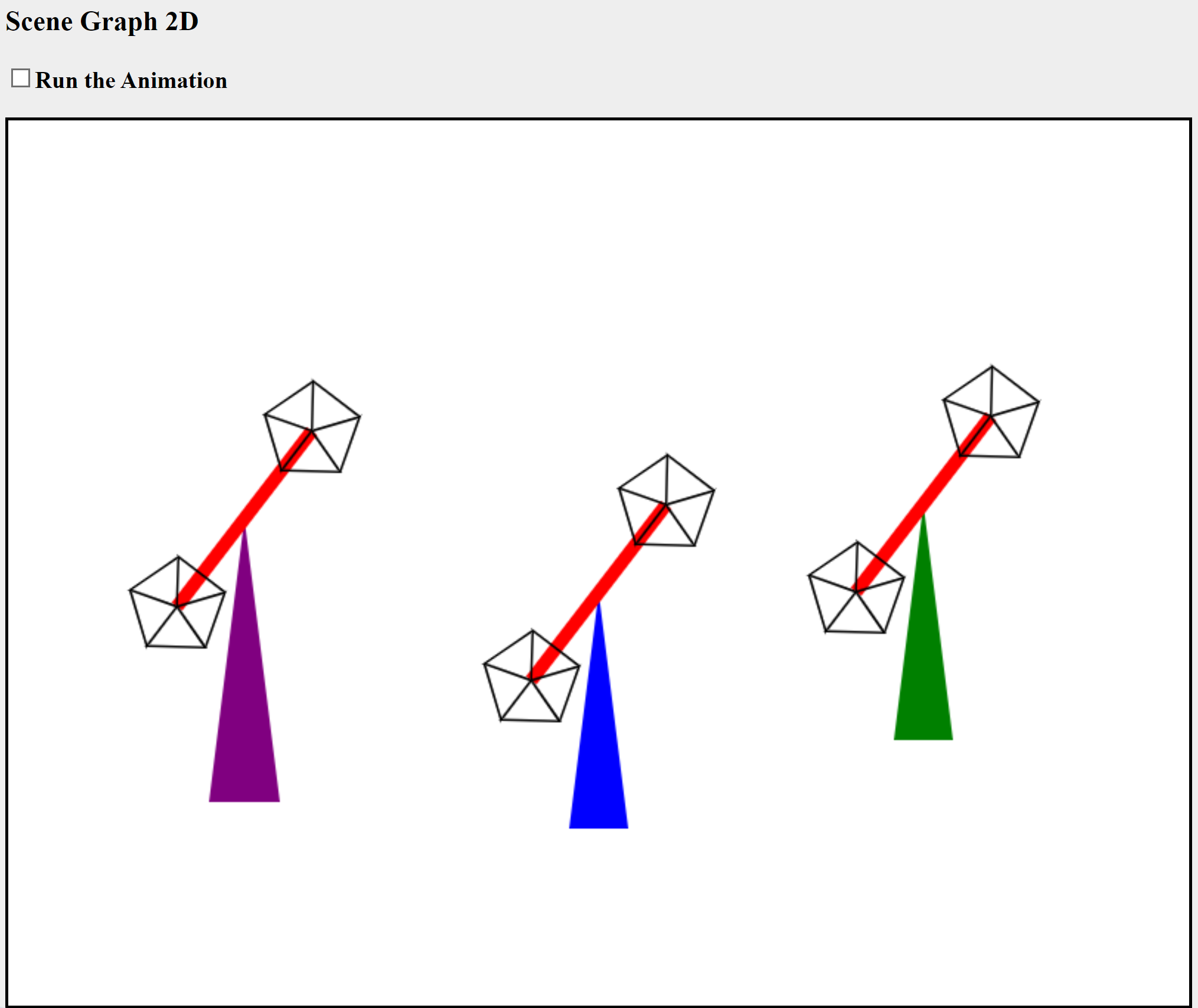
Link do zdalnego repozytorium: https://github.com/Slayzerus/UBB\_GrafikaKomputerowa/tree/main/Lab%204

**4. Wynik działania:**

a) hierarchia funcji



b) graf sceny



**5. Wnioski:**

Modelowanie hierarchiczne w grafice 2D pozwala na organizowanie złożonych scen poprzez strukturę hierarchii, co ułatwia zarządzanie i manipulację obiektami. Stosując podejście hierarchiczne, można skupić się na relacjach pomiędzy obiektami, co jest szczególnie użyteczne przy tworzeniu animacji i złożonych scen graficznych. Zadanie polega na opracowaniu sceny hierarchicznej z użyciem wielokątów zamiast kół, które będą się obracać zgodnie z ustalonym wariantem. Implementacja powinna być wykonana w języku Java lub JavaScript na dwa sposoby: poprzez hierarchię funkcji (sposób subroutinowy) oraz poprzez tworzenie grafu sceny (sposób obiektowy).

Zastosowanie modelowania hierarchicznego umożliwia efektywne przekształcenia geometryczne, co jest fundamentem tworzenia animacji w grafice 2D. Podejście subroutinowe polega na tworzeniu funkcji, które wywołują inne funkcje, tworząc strukturę przypominającą drzewo. W ten sposób łatwo można kontrolować położenie, skalowanie i obrót poszczególnych elementów sceny. Z kolei sposób obiektowy opiera się na tworzeniu grafu sceny, gdzie każdy węzeł grafu reprezentuje obiekt graficzny z jego własnymi atrybutami i transformacjami. To podejście pozwala na jeszcze większą elastyczność i modularność, co jest korzystne przy projektowaniu skomplikowanych scen.