**SPRAWOZDANIE**

Zajęcia: Grafika komputerowa

Prowadzący: prof. dr hab. Vasyl Martsenyuk

# **Laboratorium**

Data: 26.02.2024

**Temat:** "Grafika 2D z użyciem HTML Canvas"

# **Wariant:**

# **Zadanie nr. 1:** figura nr. 11

# **Zadanie nr. 2:** Jedenastokąt

Mateusz Żelazo

Informatyka I stopnia

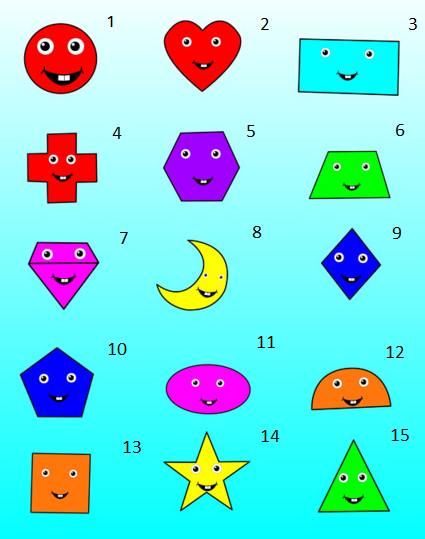
stacjonarne, 4 semestr

Gr.3a

### **Zadanie nr. 1**

1. **Polecenie:**

Plik Lab2Ex1.html proponuje rozszerzenia do standardowych funkcji rysowania HTML Canvas. Narysować obraz zgodnie z wariantem zadania (używając zarówno standardowe jak i niestandardowe funkcje rysowania).



1. **Wprowadzane dane:**
   1. **Rysowanie grafiki, kod został umieszczony w funkcji draw()**

**// Ciało**

**graphics.fillStyle = "#ff00fe";**

**graphics.lineWidth = 1;**

**graphics.strokeStyle = "black";**

**graphics.fillOval(300, 300, 100, 60)**

**graphics.stroke();**

**// Oko 1**

**let x = 260, y = 280;**

**graphics.fillStyle = "#ffffff";**

**graphics.fillCircle(x, y, 10)**

**graphics.fillStyle = "#000000";**

**graphics.fillCircle(x, y, 5)**

**graphics.fillStyle = "#ffffff";**

**graphics.fillCircle(x - 2, y - 2, 1)**

**// Oko 2**

**graphics.fillStyle = "#ffffff";**

**graphics.fillCircle(x + 75, y, 10)**

**graphics.fillStyle = "#000000";**

**graphics.fillCircle(x + 75, y, 5)**

**graphics.fillStyle = "#ffffff";**

**graphics.fillCircle(x + 73, y - 2, 1)**

**// buzia**

**let x2 = 275, y2 = 310;**

**graphics.fillStyle = "#000000";**

**graphics.beginPath();**

**graphics.moveTo(x2, y2);**

**graphics.bezierCurveTo(x2 + 25, y2 + 20, x2 + 25, y2 + 20, x2 + 50, y2);**

**graphics.bezierCurveTo(x2 + 25, y2 + 5, x2 + 25, y2 + 5, x2, y2);**

**graphics.fill();**

**graphics.fillStyle = "#ffffff";**

**graphics.lineWidth = 1;**

**graphics.strokeStyle = "black";**

**graphics.fillRect(x2 + 20, y2 + 3, 5, 5);**

**graphics.fillRect(x2 + 26, y2 + 3, 5, 5);**

**graphics.stroke();**

**graphics.beginPath();**

**graphics.lineWidth = 1;**

**graphics.moveTo(x2, y2 - 5);**

**graphics.bezierCurveTo(x2, y2, x2, y2, x2 - 3, y2 + 5);**

**graphics.stroke();**

**graphics.beginPath();**

**graphics.lineWidth = 1;**

**graphics.moveTo(x2 + 50, y2 - 5);**

**graphics.bezierCurveTo(x2 + 50, y2, x2 + 50, y2, x2 + 53, y2 + 5);**

**graphics.stroke();**

1. **Wykorzystane komendy:**
   1. **kod źródłowy**

**<!DOCTYPE html>**

**<html>**

**<head>**

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

**<style>**

**/\* This style section is here to make the canvas more obvious on the**

**page. It is white on a light gray page background, with a thin**

**black border. \*/**

**body {**

**background-color: #DDDDDD;**

**}**

**canvas {**

**background-color: white;**

**display: block;**

**}**

**#canvasholder {**

**border: 2px solid black;**

**float: left;**

**/\* This makes the border exactly fit the canvas. \*/**

**}**

**</style>**

**<script>**

**"use strict"; // gives improved error-checking in scripts.**

**var canvas; // The canvas element on which we will draw.**

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

**var pixelSize; // The size of a pixel in the coordinate system; set up by**

**// applyWindowToViewportTransform function when it is called.**

**/\*\***

**\* The draw() function is called by init() after the page loads,**

**\* to draw the content of the canvas. At the start, clear the canvas**

**\* and save a copy of the state; restore the state at the end. (These**

**\* actions are not necessary in this program, since the function will**

**\* only be called once.)**

**\*/**

**function draw() {**

**graphics.clearRect(0, 0, 600, 600);**

**// TODO: insert code to draw the image for Exercise 1!**

**graphics.fillRect(100, 100, 200, 100);**

**graphics.strokeRect(100, 100, 300, 200);**

**graphics.fillText("hello", 50, 50);**

**graphics.fillOval()**

**} // end of draw()**

**/\*\***

**\* Sets up a transformation in the graphics context so that the canvas will**

**\* show x-values in the range from left to right, and y-values in the range**

**\* from bottom to top. If preserveAspect is true, then one of the ranges**

**\* will be increased, if necessary, to account for the aspect ratio of the**

**\* canvas. This function sets the global variable pixelsize to be the**

**\* size of a pixel in the new coordinate system. (If preseverAspect is**

**\* true, pixelSize is the maximum of its horizontal and vertical sizes.)**

**\*/**

**function applyWindowToViewportTransformation(left, right, bottom, top, preserveAspect) {**

**var displayAspect, windowAspect;**

**var excess;**

**var pixelwidth, pixelheight;**

**if (preserveAspect) {**

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

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

**windowAspect = Math.abs((top - bottom) / (right - left));**

**if (displayAspect > windowAspect) {**

**// Expand the viewport vertically.**

**excess = (top - bottom) \* (displayAspect / windowAspect - 1);**

**top = top + excess / 2;**

**bottom = bottom - excess / 2;**

**}**

**else if (displayAspect < windowAspect) {**

**// Expand the viewport vertically.**

**excess = (right - left) \* (windowAspect / displayAspect - 1);**

**right = right + excess / 2;**

**left = left - excess / 2;**

**}**

**}**

**graphics.scale(canvas.width / (right - left), canvas.height / (bottom - top));**

**graphics.translate(-left, -top);**

**pixelwidth = Math.abs((right - left) / canvas.width);**

**pixelheight = Math.abs((bottom - top) / canvas.height);**

**pixelSize = Math.max(pixelwidth, pixelheight);**

**} // end of applyWindowToViewportTransformation()**

**/\*\***

**\* This function can be called to add a collection of extra drawing function to**

**\* a graphics context, to make it easier to draw basic shapes with that context.**

**\* The parameter, graphics, must be a canvas 2d graphics context.**

**\***

**\* The following new functions are added to the graphics context:**

**\***

**\* graphics.strokeLine(x1,y1,x2,y2) -- stroke the line from (x1,y1) to (x2,y2).**

**\* graphics.fillCircle(x,y,r) -- fill the circle with center (x,y) and radius r.**

**\* graphics.strokeCircle(x,y,r) -- stroke the circle.**

**\* graphics.fillOval(x,y,r1,r2) -- fill oval with center (x,y) and radii r1 and r2.**

**\* graphics.stokeOval(x,y,r1,r2) -- stroke the oval**

**\* graphics.fillPoly(x1,y1,x2,y2,...) -- fill polygon with vertices (x1,y1), (x2,y2), ...**

**\* graphics.strokePoly(x1,y1,x2,y2,...) -- stroke the polygon.**

**\* graphics.getRGB(x,y) -- returns the color components of pixel at (x,y) as an array of**

**\* four integers in the range 0 to 255, in the order red, green, blue, alpha.**

**\***

**\* (Note that "this" in a function that is called as a member of an object refers to that**

**\* object. Here, this will refer to the graphics context.)**

**\*/**

**function addGraphicsContextExtras(graphics) {**

**graphics.strokeLine = function (x1, y1, x2, y2) {**

**this.beginPath();**

**this.moveTo(x1, y1);**

**this.lineTo(x2, y2);**

**this.stroke();**

**}**

**graphics.fillCircle = function (x, y, r) {**

**this.beginPath();**

**this.arc(x, y, r, 0, 2 \* Math.PI, false);**

**this.fill();**

**}**

**graphics.strokeCircle = function (x, y, radius) {**

**this.beginPath();**

**this.arc(x, y, radius, 0, 2 \* Math.PI, false);**

**this.stroke();**

**}**

**graphics.fillPoly = function () {**

**if (arguments.length < 6)**

**return;**

**this.beginPath();**

**this.moveTo(arguments[0], arguments[1]);**

**for (var i = 2; i + 1 < arguments.length; i = i + 2) {**

**this.lineTo(arguments[i], arguments[i + 1]);**

**}**

**this.closePath();**

**this.fill();**

**}**

**graphics.strokePoly = function () {**

**if (arguments.length < 4)**

**return;**

**this.beginPath();**

**this.moveTo(arguments[0], arguments[1]);**

**for (var i = 2; i + 1 < arguments.length; i = i + 2) {**

**this.lineTo(arguments[i], arguments[i + 1]);**

**}**

**this.closePath();**

**this.stroke();**

**}**

**graphics.fillOval = function (x, y, horizontalRadius, verticalRadius) {**

**this.save();**

**this.translate(x, y);**

**this.scale(horizontalRadius, verticalRadius);**

**this.beginPath();**

**this.arc(0, 0, 1, 0, 2 \* Math.PI, false);**

**this.restore();**

**this.fill();**

**}**

**graphics.strokeOval = function (x, y, horizontalRadius, verticalRadius) {**

**this.save();**

**this.translate(x, y);**

**this.scale(horizontalRadius, verticalRadius);**

**this.beginPath();**

**this.arc(0, 0, 1, 0, 2 \* Math.PI, false);**

**this.restore();**

**this.stroke();**

**}**

**graphics.getRGB = function (x, y) {**

**var color = this.getImageData(x, y, 1, 1);**

**return color.data;**

**}**

**} // end of addGraphicsContextExtras()**

**/\*\***

**\* The init() funciton is called after the page has been**

**\* loaded. It initializes the canvas and graphics variables.**

**\* It calles addGraphicsContextExtras(graphics) to add the extra**

**\* drawing functions to the graphics context, and it calls draw()**

**\* to draw on the canvas.**

**\*/**

**function init() {**

**try {**

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

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

**} catch (e) {**

**document.getElementById("canvasholder").innerHTML =**

**"Canvas graphics is not supported.<br>" +**

**"An error occurred while initializing graphics.";**

**}**

**addGraphicsContextExtras(graphics);**

**draw(); // Call draw() to draw on the canvas.**

**}**

**</script>**

**</head>**

**<body onload="init()">**

**<!-- the onload attribute here is what calls the init() function -->**

**<noscript>**

**<!-- This message will be shown in the page if JavaScript is not available. -->**

**<p>JavaScript is required to use this page.</p>**

**</noscript>**

**<div id="canvasholder">**

**<canvas id="canvas" width="600" height="600">**

**<!-- This message is shown on the page if the browser doesn't support the canvas element. -->**

**Canvas not supported.**

**</canvas>**

**</div>**

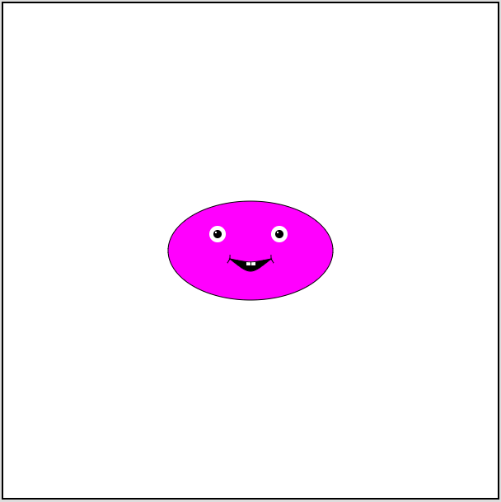
**</body>**

**</html>**

1. **Link do zdalnego repozytorium:**

* https://github.com/Terminalk/GKLab

1. **Wynik działania:** 
   1. **narysowany obraz**

****

1. **Wnioski:**

Wykorzystując płótno Canvas oraz funkcję rysowania w nim jesteśmy w stanie narysować dowolną figurę 2d

### 

### 

### 

### 

### **Zadanie nr. 2**

1. **Polecenie:**

W Plik Lab2Ex2.html program domyślnie rysuje szereg kwadratów. Stworzyć narzędzia pozwalające na wykonywanie czynności

- "czyszczenie" canvasu - Clear button:

- dodanie jednego nowego koloru do elementu <select>. Implementować nowy kolor przez funkcję doMouseMove.

- opracowanie nowego narzędzia - rysowania szeregu wielokątów (zgodnie z wariantem zadania). Opcja ma być dostępna przez nowy element <select>

1. **Wprowadzane dane:**
   1. **Czyszczenie elementu Canvas:** 
      1. **HTML**

**<button id="clearButton">Clear</button>**

* + 1. **JavaScript**

**var clearButton = document.getElementById("clearButton");**

**clearButton.addEventListener("click", function () {**

**graphics.clearRect(0, 0, 800, 600);**

**});**

* 1. **Implementacja nowego koloru**
     1. **HTML**

**<select id="colorChoice">**

**<option value="0">Random</option>**

**<option value="1">Red</option>**

**<option value="2">Green</option>**

**<option value="3">Blue</option>**

**<option value="4">Orange</option>**

**</select>**

* + 1. **JavaScript**

**if (colorChoice == 0) {**

**graphics.fillStyle = randomColorString();**

**}**

**else if (colorChoice == 1) {**

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

**}**

**else if (colorChoice == 2) {**

**graphics.fillStyle = "green";**

**}**

**else if (colorChoice == 3) {**

**graphics.fillStyle = "blue";**

**}**

**else if (colorChoice == 4) {**

**graphics.fillStyle = "orange";**

**}**

* 1. **narzędzie rysowanie wielokątów**
     1. **HTML**

**<select id="figureChoice">**

**<option value="0">Kwadrat</option>**

**<option value="1">Jedenastokąt</option>**

**</select>**

* + 1. **JavaScript**

**// Deklaracja zmiennej dla wyboru figury**

**var figureChoice;**

**// Przypisanie wartości do zmiennej, dla wyboru figury, zdefiniowana w funkcji doMouseDown()**

**figureChoice = Number(document.getElementById("figureChoice").value);**

**// Wybór figury, zdefiniowana w funkcji doMouseMove()**

**if (figureChoice == 0) {**

**graphics.fillRect(x - 20, y - 20, 40, 40);**

**graphics.strokeRect(x - 20, y - 20, 40, 40);**

**} else if (figureChoice == 1) {**

**graphics.fillPoly(**

**(x - 20) + 92.06267664155905, (y - 20) + 77.03204087277987,**

**(x - 20) + 70.77075065009433, (y - 20) + 95.48159976772592,**

**(x - 20) + 42.88425808633574, (y - 20) + 99.49107209404664,**

**(x - 20) + 17.25696330273575, (y - 20) + 87.78747871771293,**

**(x - 20) + 2.0253513192751385, (y - 20) + 64.08662784207151,**

**(x - 20) + 2.0253513192751313, (y - 20) + 35.91337215792851,**

**(x - 20) + 17.25696330273574, (y - 20) + 12.212521282287092,**

**(x - 20) + 42.88425808633574, (y - 20) + 0.5089279059533638,**

**(x - 20) + 70.7707506500943, (y - 20) + 4.518400232274075,**

**(x - 20) + 92.06267664155904, (y - 20) + 22.967959127220087,**

**(x - 20) + 100, (y - 20) + 49.999999999999986**

**);**

**graphics.stroke();**

**}**

1. **Wykorzystane komendy:**
   1. **kod źródłowy**

**<!DOCTYPE html>**

**<html>**

**<!--**

**This web page does the minimal setup for using mouse events along**

**with 2D canvas graphics.**

**-->**

**<head>**

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

**<style>**

**/\* This style section is here to make the canvas more obvious on the**

**page. It is white on a light gray page background, with a thin**

**black border. Also, turn off text selection to avoid having**

**selection interfere with mouse action. \*/**

**body {**

**background-color: #DDDDDD;**

**-webkit-user-select: none; /\* turn off text selection / Webkit \*/**

**-moz-user-select: none; /\* Firefox \*/**

**-ms-user-select: none; /\* IE 10 \*/**

**-o-user-select: none; /\* Opera \*/**

**user-select: none;**

**}**

**canvas {**

**background-color: white;**

**display: block;**

**}**

**#canvasholder {**

**border:2px solid black;**

**float: left; /\* This makes the border exactly fit the canvas. \*/**

**}**

**</style>**

**<script>**

**"use strict"; // gives improved error-checking in scripts.**

**var canvas; // The canvas element on which we will draw.**

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

**/\*\***

**\* This function returns a string representing a random RGB color.**

**\* The returned string can be assigned as the value of graphics.fillStyle**

**\* or graphics.strokeStyle.**

**\*/**

**function randomColorString() {**

**var r = Math.floor(256\*Math.random());**

**var g = Math.floor(256\*Math.random());**

**var b = Math.floor(256\*Math.random());**

**return "rgb(" + r + "," + g + "," + b + ")";**

**}**

**/\*\***

**\* This function is called in init() to set up mouse event handling**

**\* on the canvas. You can modify the nested functions doMouseDown,**

**\* doMouseDrag, and possibly doMouseUp to change the reponse to**

**\* mouse events. As an example, this program does some simple drawing.**

**\*/**

**function installMouseHandler() {**

**var dragging = false; // set to true when a drag action is in progress.**

**var startX, startY; // coordinates of mouse at start of drag.**

**var prevX, prevY; // previous mouse position during a drag.**

**var colorChoice; // Integer code for the selected color in the "colorChoide"**

**// popup menu. The value is assigned in doMouseDown.**

**function doMouseDown(evt) {**

**// This function is called when the user presses a button on the mouse.**

**// Only the main mouse button will start a drag.**

**if (dragging) {**

**return; // if a drag is in progress, don't start another.**

**}**

**if (evt.button != 0) {**

**return; // don't respond unless the button is the main (left) mouse button.**

**}**

**var x,y; // mouse position in canvas coordinates**

**var r = canvas.getBoundingClientRect();**

**x = Math.round(evt.clientX - r.left); // translate mouse position from screen coords to canvas coords.**

**y = Math.round(evt.clientY - r.top); // round to integer values; some browsers would give non-integers.**

**dragging = true; // (this won't be the case for all mousedowns in all programs)**

**if (dragging) {**

**startX = prevX = x;**

**startY = prevY = y;**

**document.addEventListener("mousemove", doMouseMove, false);**

**document.addEventListener("mouseup", doMouseUp, false);**

**}**

**colorChoice = Number(document.getElementById("colorChoice").value);**

**// TODO: Anything else to do when mouse is first pressed?**

**}**

**function doMouseMove(evt) {**

**// This function is called when the user moves the mouse during a drag.**

**if (!dragging) {**

**return; // (shouldn't be possible)**

**}**

**var x,y; // mouse position in canvas coordinates**

**var r = canvas.getBoundingClientRect();**

**x = Math.round(evt.clientX - r.left);**

**y = Math.round(evt.clientY - r.top);**

**/\*------------------------------------------------------------\*/**

**/\* TODO: Add support for more drawing tools. \*/**

**if ( Math.abs(x-prevX) + Math.abs(y-prevY) < 3 ) {**

**return; // don't draw squares too close together**

**}**

**if (colorChoice == 0) {**

**graphics.fillStyle = randomColorString();**

**}**

**else if (colorChoice == 1) {**

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

**}**

**else if (colorChoice == 2) {**

**graphics.fillStyle = "green";**

**}**

**else if (colorChoice == 3) {**

**graphics.fillStyle = "blue";**

**}**

**graphics.fillRect(x-20,y-20,40,40);**

**graphics.strokeRect(x-20,y-20,40,40);**

**/\*------------------------------------------------------------\*/**

**prevX = x; // update prevX,prevY to prepare for next call to doMouseMove**

**prevY = y;**

**}**

**function doMouseUp(evt) {**

**// This function is called when the user releases a mouse button during a drag.**

**if (!dragging) {**

**return; // (shouldn't be possible)**

**}**

**dragging = false;**

**document.removeEventListener("mousemove", doMouseMove, false);**

**document.removeEventListener("mouseup", doMouseMove, false);**

**}**

**canvas.addEventListener("mousedown", doMouseDown, false);**

**} // end installMouseHandler**

**/\*\***

**\* This function can be called to add a collection of extra drawing function to**

**\* a graphics context, to make it easier to draw basic shapes with that context.**

**\* The parameter, graphics, must be a canvas 2d graphics context.**

**\***

**\* The following new functions are added to the graphics context:**

**\***

**\* graphics.strokeLine(x1,y1,x2,y2) -- stroke the line from (x1,y1) to (x2,y2).**

**\* graphics.fillCircle(x,y,r) -- fill the circle with center (x,y) and radius r.**

**\* graphics.strokeCircle(x,y,r) -- stroke the circle.**

**\* graphics.fillOval(x,y,r1,r2) -- fill oval with center (x,y) and radii r1 and r2.**

**\* graphics.stokeOval(x,y,r1,r2) -- stroke the oval**

**\* graphics.fillPoly(x1,y1,x2,y2,...) -- fill polygon with vertices (x1,y1), (x2,y2), ...**

**\* graphics.strokePoly(x1,y1,x2,y2,...) -- stroke the polygon.**

**\* graphics.getRGB(x,y) -- returns the color components of pixel at (x,y) as an array of**

**\* four integers in the range 0 to 255, in the order red, green, blue, alpha.**

**\***

**\* (Note that "this" in a function that is called as a member of an object refers to that**

**\* object. Here, this will refer to the graphics context.)**

**\*/**

**function addGraphicsContextExtras(graphics) {**

**graphics.strokeLine = function(x1,y1,x2,y2) {**

**this.beginPath();**

**this.moveTo(x1,y1);**

**this.lineTo(x2,y2);**

**this.stroke();**

**}**

**graphics.fillCircle = function(x,y,r) {**

**this.beginPath();**

**this.arc(x,y,r,0,2\*Math.PI,false);**

**this.fill();**

**}**

**graphics.strokeCircle = function(x,y,radius) {**

**this.beginPath();**

**this.arc(x,y,radius,0,2\*Math.PI,false);**

**this.stroke();**

**}**

**graphics.fillPoly = function() {**

**if (arguments.length < 6)**

**return;**

**this.beginPath();**

**this.moveTo(arguments[0],arguments[1]);**

**for (var i = 2; i+1 < arguments.length; i = i + 2) {**

**this.lineTo(arguments[i],arguments[i+1]);**

**}**

**this.closePath();**

**this.fill();**

**}**

**graphics.strokePoly = function() {**

**if (arguments.length < 4)**

**return;**

**this.beginPath();**

**this.moveTo(arguments[0],arguments[1]);**

**for (var i = 2; i+1 < arguments.length; i = i + 2) {**

**this.lineTo(arguments[i],arguments[i+1]);**

**}**

**this.closePath();**

**this.stroke();**

**}**

**graphics.fillOval = function(x,y,horizontalRadius,verticalRadius) {**

**this.save();**

**this.translate(x,y);**

**this.scale(horizontalRadius,verticalRadius);**

**this.beginPath();**

**this.arc(0,0,1,0,2\*Math.PI,false);**

**this.restore();**

**this.fill();**

**}**

**graphics.strokeOval = function(x,y,horizontalRadius,verticalRadius) {**

**this.save();**

**this.translate(x,y);**

**this.scale(horizontalRadius,verticalRadius);**

**this.beginPath();**

**this.arc(0,0,1,0,2\*Math.PI,false);**

**this.restore();**

**this.stroke();**

**}**

**graphics.getRGB = function(x,y) {**

**var color = this.getImageData(x,y,1,1);**

**return color.data;**

**}**

**} // end of addGraphicsContextExtras()**

**/\*\***

**\* The init() function is called after the page has been**

**\* loaded. It initializes the canvas and graphics variables,**

**\* and it installs mouse and key listeners. If an error**

**\* occurs, a message is displayed in place of the canvas.**

**\*/**

**function init() {**

**try {**

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

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

**} catch(e) {**

**document.getElementById("canvasholder").innerHTML =**

**"<p>Canvas graphics is not supported.<br>" +**

**"An error occurred while initializing graphics.</p>";**

**return;**

**}**

**addGraphicsContextExtras(graphics);**

**installMouseHandler();**

**graphics.fillStyle = "white";**

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

**}**

**</script>**

**</head>**

**<body onload="init()"> <!-- the onload attribute here is what calls the init() function -->**

**<noscript>**

**<!-- This message will be shown in the page if JavaScript is not available. -->**

**<p>JavaScript is required to use this page.</p>**

**</noscript>**

**<p><b>Color:</b>**

**<select id="colorChoice">**

**<option value="0">Random</option>**

**<option value="1">Red</option>**

**<option value="2">Green</option>**

**<option value="3">Blue</option>**

**</select>**

**</p>**

**<div id="canvasholder">**

**<canvas id="canvas" width="800" height="600">**

**<!-- This message is shown on the page if the browser doesn't support the canvas element. -->**

**Canvas not supported.**

**</canvas>**

**</div>**

**</body>**

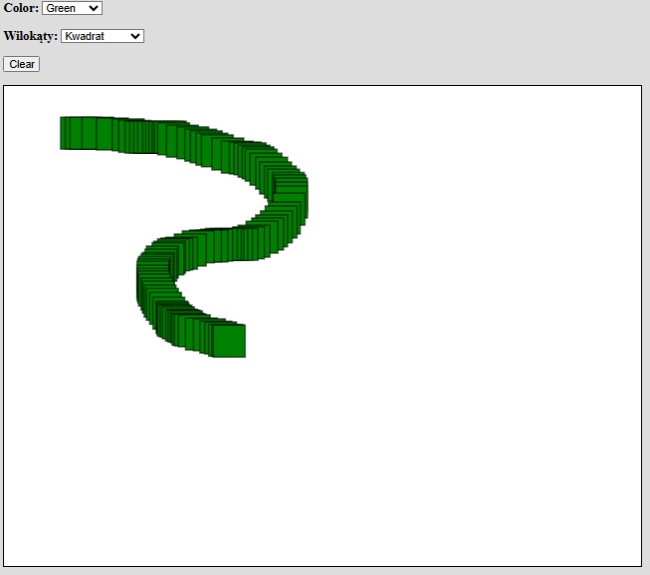
**</html>**

1. **Link do zdalnego repozytorium**

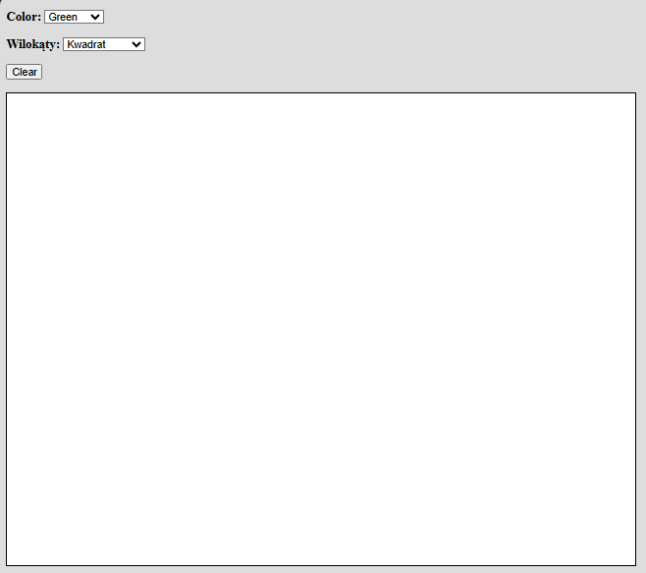
* https://github.com/Terminalk/GKLab

1. **Wynik działania:**
   1. **Wynik działania programu:** 
      1. **Czyszczenie Canvasu**

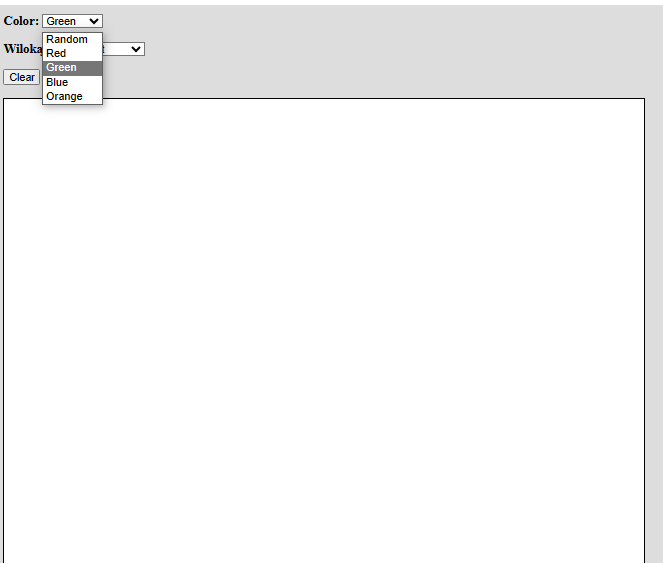
**Przed kliknięciem przycisku “wyczyść”**

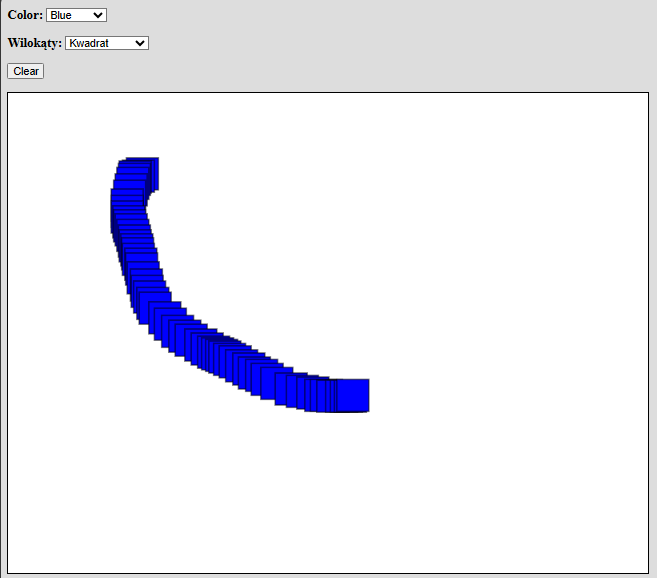
****

**Po kliknięciu przycisku “wyczyść”**

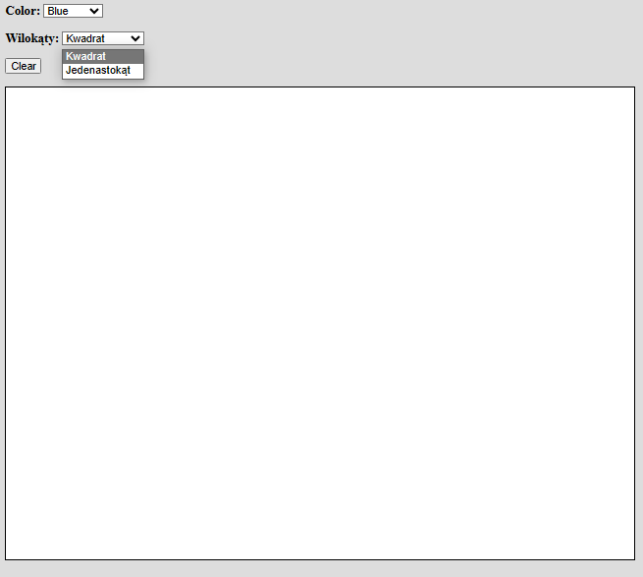
****

* + 1. **Wybór koloru**

****

****

* + 1. **Wybór wielokąta**

****



1. **Wnioski:**

Na obiekt Canvas jesteśmy w stanie wpływać za pomocą elementów HTML poprzez edycje zmiennych po stronie języka JavaScript