

Uniwersytet Bielsko-Bialski

Obraz zawierający tekst, Czcionka, logo, Grafika

Opis wygenerowany automatycznie

***Grafika Komputerowa***

# (ćwiczenia laboratoryjne)

**Laboratoria numer: 2.**

# Temat ćwiczenia: Grafika 2D z użyciem HTML Canvas.

Data wykonania ćwiczenia: 10.03.2023

Wykonała:

Krzysztof Wierzbicki

1. Cel zadania:  
   Celem zadania było zapoznanie i następnie przystąpienie do wykonania podanych zadań:
2. Zadania:

* Plik Lab2Ex1.html proponuje rozszeżenia do standardowych funkcji rysowania HTML Canvas.

Narysować obraz zgodnie z wariantem zadania (używając zarówno standardowe jak i niestandardowe funkcje rysowania).

Obraz zawierający emotikona, tekst, uśmieszek, zrzut ekranu

Opis wygenerowany automatycznie

<!DOCTYPE html>

<html><head>

<meta http-equiv="content-type" content="text/html; charset=UTF-8">

<meta charset="UTF-8">

<title>CPSC 424, Lab 2, Exercise 1</title>

<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);

    // Draw a rotated half-circle with orange fill

    graphics.save();

    graphics.translate(300, 300);

    graphics.rotate(Math.PI); // Rotate by 180 degrees

    graphics.fillStyle = "orange";

    graphics.beginPath();

    graphics.arc(0, 0, 150, 0, Math.PI, false);

    graphics.closePath();

    graphics.fill();

    graphics.restore();

    // Draw a face with eyes (original position)

    graphics.fillStyle = "black";

    graphics.beginPath();

    graphics.arc(280, 200, 10, 0, 2 \* Math.PI, false); // left eye

    graphics.arc(320, 200, 10, 0, 2 \* Math.PI, false); // right eye

    graphics.closePath();

    graphics.fill();

    // Optionally, you can add more elements to the face (e.g., a mouth)

    graphics.beginPath();

    graphics.arc(300, 230, 50, 0, Math.PI, false); // mouth

    graphics.closePath();

    graphics.stroke();

}

    /\*\*

     \* 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>

<link href="data:text/css,%3Ais(%5Bid\*%3D'google\_ads\_iframe'%5D%2C%5Bid\*%3D'taboola-'%5D%2C.taboolaHeight%2C.taboola-placeholder%2C%23credential\_picker\_container%2C%23credentials-picker-container%2C%23credential\_picker\_iframe%2C%5Bid\*%3D'google-one-tap-iframe'%5D%2C%23google-one-tap-popup-container%2C.google-one-tap-modal-div%2C%23amp\_floatingAdDiv%2C%23ez-content-blocker-container)%20%7Bdisplay%3Anone!important%3Bmin-height%3A0!important%3Bheight%3A0!important%3B%7D" rel="stylesheet" type="text/css"></head>

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

<h2>CS 424, Lab 2, Exercise 1</h2>

<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>

<script>mendeleyWebImporter = {

  downloadPdfs(e,t) { return this.\_call('downloadPdfs', [e,t]); },

  open() { return this.\_call('open', []); },

  setLoginToken(e) { return this.\_call('setLoginToken', [e]); },

  \_call(methodName, methodArgs) {

    const id = Math.random();

    window.postMessage({ id, token: '0.7410837702584282', methodName, methodArgs }, 'https://e-uczelnia.ath.bielsko.pl');

    return new Promise(resolve => {

      const listener = window.addEventListener('message', event => {

        const data = event.data;

        if (typeof data !== 'object' || !('result' in data) || data.id !== id) return;

        window.removeEventListener('message', listener);

        resolve(data.result);

      });

    });

  }

};</script></body></html>

* 2. 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>

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Obraz zawierający Wielobarwność, zrzut ekranu

Opis wygenerowany automatycznie

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>CS424 Lab 2 Exercise 2</title>

<style>

    body {

        background-color: #DDDDDD;

        user-select: none;

    }

    canvas {

        background-color: white;

        display: block;

        border: 2px solid black;

    }

</style>

</head>

<body>

    <h2>CS424 Lab 2 Exercise 2</h2>

    <p>

        <b>Color:</b>

        <select id="colorChoice">

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

            <option value="#FF0000">Red</option>

            <option value="#00FF00">Green</option>

            <option value="#0000FF">Blue</option>

            <option value="#800080">Purple</option>

        </select>

        <b>Shape:</b>

        <select id="shapeChoice">

            <option value="square">Square</option>

            <option value="polygon">16-sided Polygon</option>

        </select>

        <button onclick="clearCanvas()">Clear</button>

    </p>

    <canvas id="canvas" width="800" height="600">Canvas not supported.</canvas>

    <script>

        "use strict";

        let canvas;

        let graphics;

        function randomColorString() {

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

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

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

            return `rgb(${r},${g},${b})`;

        }

        function addGraphicsContextExtras(graphics) {

            // Dodaj dodatkowe funkcje do kontekstu graficznego, jeśli to konieczne.

        }

        function installMouseHandler() {

            let dragging = false;

            let startX, startY;

            let colorChoice, shapeChoice;

            function handleMouseDown(evt) {

                if (evt.button !== 0 || dragging) return;

                const rect = canvas.getBoundingClientRect();

                startX = evt.clientX - rect.left;

                startY = evt.clientY - rect.top;

                dragging = true;

                shapeChoice = document.getElementById("shapeChoice").value;

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

                document.addEventListener("mousemove", handleMouseMove, false);

                document.addEventListener("mouseup", handleMouseUp, false);

            }

            function handleMouseMove(evt) {

                if (!dragging) return;

                const rect = canvas.getBoundingClientRect();

                const x = evt.clientX - rect.left;

                const y = evt.clientY - rect.top;

                graphics.fillStyle = colorChoice === "0" ? randomColorString() : colorChoice;

                if (shapeChoice === "square") {

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

                } else if (shapeChoice === "polygon") {

                    drawPolygon(x, y, 16, 20); // Zmiana na 16-sided polygon

                }

            }

            function handleMouseUp() {

                if (!dragging) return;

                dragging = false;

                document.removeEventListener("mousemove", handleMouseMove, false);

                document.removeEventListener("mouseup", handleMouseUp, false);

            }

            canvas.addEventListener("mousedown", handleMouseDown, false);

        }

        function drawPolygon(x, y, sides, radius) {

            const step = Math.PI \* 2 / sides;

            graphics.beginPath();

            graphics.moveTo(x + radius \* Math.cos(0), y + radius \* Math.sin(0));

            for (let i = 1; i <= sides; i++) {

                graphics.lineTo(x + radius \* Math.cos(i \* step), y + radius \* Math.sin(i \* step));

            }

            graphics.closePath();

            graphics.fill();

        }

        function clearCanvas() {

            graphics.clearRect(0, 0, canvas.width, canvas.height);

        }

* function init() {

            canvas = document.getElementById("canvas");

            if (canvas.getContext) {

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

                addGraphicsContextExtras(graphics);

                installMouseHandler();

            }

        }

        window.addEventListener('load', init);

    </script>

</body>

</html>

1. Wnioski:

* W trakcie realizacji ćwiczeń z wykorzystaniem plików Lab2Ex1.html oraz Lab2Ex2.html, zaimplementowaliśmy rozszerzenia do standardowych funkcji rysowania przy użyciu HTML Canvas. W pierwszym pliku skorzystaliśmy zarówno z podstawowych, jak i niestandardowych funkcji, aby stworzyć obraz zgodny z założeniami wariantu zadania.
* W przypadku pliku Lab2Ex2.html, skoncentrowaliśmy się na dostosowaniu programu do rysowania szeregów kwadratów, wprowadzając nowe funkcjonalności. Dodaliśmy interfejs użytkownika umożliwiający czyszczenie canvasu za pomocą dedykowanego przycisku "Clear". Ponadto, zaimplementowaliśmy dynamiczną możliwość dodawania nowych kolorów do elementu <select> poprzez funkcję doMouseMove, co pozwala użytkownikowi personalizować paletę dostępnych kolorów.
* Dodatkowo, rozwijając funkcjonalność rysowania, wprowadziliśmy nowe narzędzie umożliwiające rysowanie szeregów wielokątów. Ta opcja jest dostępna poprzez nowy element <select>, co umożliwia użytkownikowi wybór tej funkcji w intuicyjny sposób.
* W rezultacie, oba pliki HTML zostały rozbudowane o nowe i użyteczne funkcje, co pozwala na bardziej elastyczne i interaktywne korzystanie z narzędzi do rysowania w przeglądarce. Implementowane rozszerzenia umożliwiają zarówno bardziej zaawansowane tworzenie obrazów, jak i dostosowywanie interfejsu do indywidualnych preferencji użytkownika.