Sprawozdanie

Zajęcia: Grafika Komputerowa

Prowadzący: mgr inż. Mikołaj Grygiel

Laboratorium Grafiki Komputerowej

25.02.2024

Temat: „Grafika 2D z użyciem HTML Canvas”

Wariant: 3.7

Albert Więcaszek

053012

Informatyka I stopień

Zaoczne

4 semestr

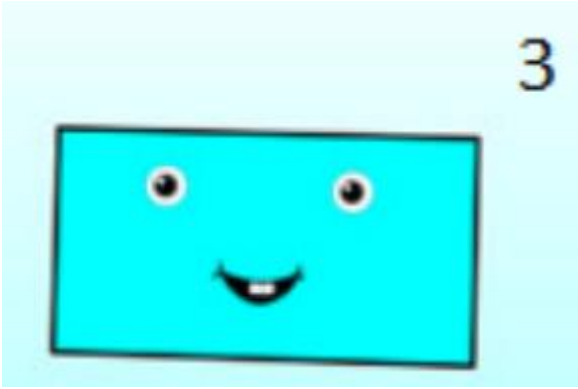
Gr. 2B

1. Polecenia

a) Narysować obraz zgodnie z wariantem zadania (patrz Fig. 1) (używając zarówno standardowe jak i niestandardowe funkcje rysowania).
b) W podanym pliku dodać opcję czyszczenia canvy za pomocą przycisku, dodać dodatkowy kolor do listy wyboru oraz dodać opcję rysowania za pomocą kształtu przedstawiającego siedmiokąt foremny

2. Wprowadzane dane

a) W przypadku pierwszego polecenia zmodyfikowano jedynie metodą „draw()" w taki sposób, aby otrzymać figurę widoczną na Zdjęcie 1



Zdjęcie 1 Kształt który należy odwzorować w zad a

Aby uzyskać efekt pochylenia na początku całą canvę obrócono o 0.05 RAD (około 3 stopnie) za pomocą komendy „rotate()". Następnie za pomocą funkcji „fillRect()" oraz „strokeRect()" utworzono jasnoniebieski prostokąt. Następnie dodano 2 pary oczu składające się z 3 łuków utworzonych za pomocą komendy „arc()". W kolejnym kroku dwoma prostymi wykonano kąciki ust za pomocą komend „moveTo()" oraz „lineTo()". Następnie wypełniono uśmiech tworząc dwie krzywe Beziera komendą „bezierCurveTo()". Na końcu dodano biały prostokąt odzwierciedlający zęby oraz pionową linię jako oddzielenie dwóch zębów.

b) Zadanie drugie rozbite było na 3 podpunkty. Pierwszym było dodanie przycisku pozwalającego wyczyścić canvę. Rozwiązano to dodając przycisk o id „clearButton" i podpinając do niego metodę rysujące pustą canvę za pomocą funkcji „clearRect()".

Drugim etapem było dodanie dodatkowego koloru do listy wyboru. W tym celu po pierwsze w kodzie HTML dodano dodatkową opcję w pojemniku <*select*> o id „colorChoice". Następnie w skrypcie uzupełniono metodę „doMouseMove()" o dodatkowy wariant w instrukcji warunkowej *if else.*

Ostatnim etapem była zmiana kształtu pędzla. W tym celu najpierw utworzono dodatkową zmienną globalną „shapeChoice" przechowywującą informacje na temat obecnie wybranego kształtu. Następnie dodano w kodzie HTML kontener <*select*> o id „shapeChoice" w którym dodano dwie opcje, domyślą „None", która nadal pozwalała rysować za pomocą kwadratów oraz nową opcję „Siedmiobok". Na końcu zmodyfikowano metodę „doMouseMove()" wprowadzając do niej dodatkową instrukcję warunkową *if else,* odpowiedzialną za rysowanie odpowiedniego kształtu w

zależności od wartości zmiennej „shapeChoice”. Aby całość działała poprawnie powiązano również w metodzie „doMouseDown()” indeks opcji „shapeChoice” z wartością zmiennej „shapeChoice”.

3. Wykorzystane komendy

Kod źródłowy zadania „a” przedstawiono poniżej.

```
<!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);

        graphics.rotate(0.05)
        graphics.fillStyle = "rgb(0,247,255)"
        graphics.fillRect(20, 20, 120, 70)
        graphics.strokeStyle = "#000000"
        graphics.strokeRect(20, 20, 120, 70)

        graphics.fillStyle = "#FFFFFF"
        graphics.beginPath()
        graphics.arc(50, 50, 5, 0, 2 * Math.PI)
        graphics.fill()
        graphics.beginPath()
        graphics.arc(100, 50, 5, 0, 2 * Math.PI)
        graphics.fill()

        graphics.fillStyle = "#000000"
```

```
        graphics.beginPath()
        graphics.arc(50, 50, 3, 0, 2 * Math.PI)
        graphics.fill()
        graphics.beginPath()
        graphics.arc(100, 50, 3, 0, 2 * Math.PI)
        graphics.fill()

        graphics.fillStyle = "#FFFFFF"
        graphics.beginPath()
        graphics.arc(49, 49, 1, 0, 2 * Math.PI)
        graphics.fill()
        graphics.beginPath()
        graphics.arc(99, 49, 1, 0, 2 * Math.PI)
        graphics.fill()


        graphics.strokeStyle = "#000000"
        graphics.beginPath()
        graphics.moveTo(60, 70)
        graphics.lineTo(65, 65)
        graphics.moveTo(90, 70)
        graphics.lineTo(85, 65)
        graphics.stroke()

        graphics.fillStyle = "#000000"
        graphics.beginPath()
        graphics.moveTo(63, 67)
        graphics.bezierCurveTo(63 + 12, 67 + 12, 88 - 12, 67 + 12, 88, 67)
        graphics.moveTo(63, 67)
        graphics.bezierCurveTo(63 + 6, 67 + 6, 88 - 6, 67 + 6, 88, 67)
        graphics.fill()

        graphics.fillStyle = "#FFFFFF"
        graphics.fillRect(63 + 9, 67, 8, 5)

        graphics.strokeStyle = "#000000"
        graphics.beginPath()
        graphics.moveTo(63+13, 67)
        graphics.lineTo(63+13, 67+5)
        graphics.stroke()



    } // 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>
<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>
```

Kod źródłowy zadania b przedstawiono poniżej:

```
<!DOCTYPE html>
<html><!--
   This web page does the minimal setup for using mouse events along
   with 2D canvas graphics.
--><head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<meta charset="UTF-8">
<title>CS424, Lab 2, Exercise 2</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.  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 + ")";
    }

    function doClear(){
        graphics.clearRect(0,0,1000,1000)
    }

    /**
     * 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.
        var shapeChoice;
        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);
            shapeChoice = Number(document.getElementById("shapeChoice").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";
            }
            else if (colorChoice == 4) {
                graphics.fillStyle = "cyan";
            }

            if (shapeChoice == 0){
                graphics.fillRect(x-20,y-20,40,40);
                graphics.strokeRect(x-20,y-20,40,40);
            } else if (shapeChoice == 1){
                var theta =Math.PI-2.24285714286; //2.242 RAD = 128 & 4/7 for 7-
sided figure
                                                  //PI-angle because I want rotation
in opposite direction
                var s = Math.sin(theta)
                var c = Math.cos(theta)
                var r = 40
                graphics.beginPath()
                graphics.moveTo(x,y)


                for (let i=0; i< 7 ; i++){
                    x+=r*Math.cos(i*theta)
                    y+=r*Math.sin(i*theta)
                    graphics.lineTo(x,y)
                }

                graphics.stroke()
                graphics.fill()
```

```
            }


            /*-------------------------------------------------------------*/

            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();
```

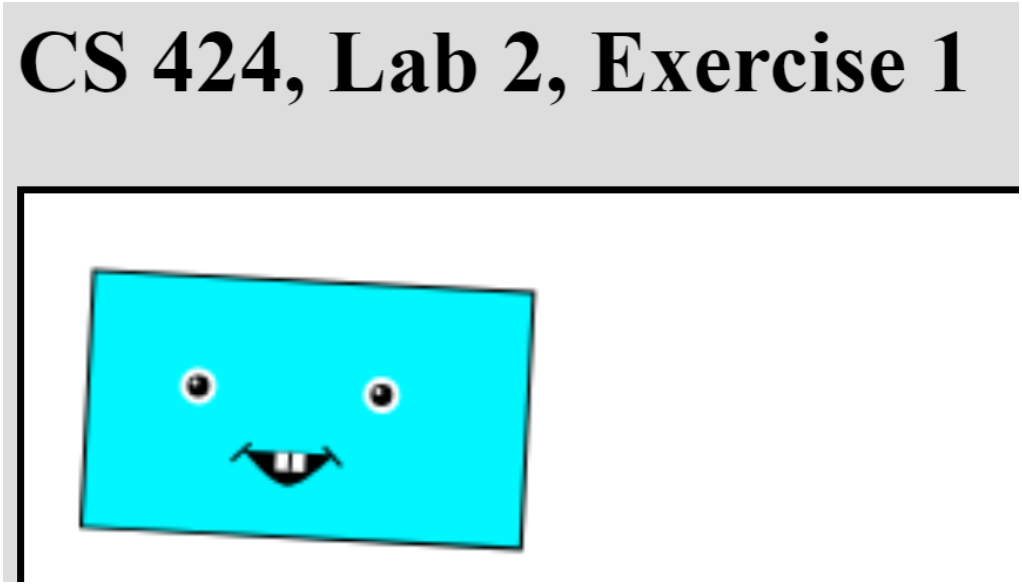
```
            this.arc(x,y,radius,0,2*Math.PI,false);
            this.stroke();
        }
        graphics.fillPoly = function() {
            if (arguments.length < 6)
               return;
            this.beginPath();
            this.moveTo(arguments[0],arguments[1]);
            for (var i = 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);
        document.getElementById("clearButton").onclick = doClear;
    }

</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>Lab 2, Exercise 2: Mousing</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>

<p><b>Color:</b>
    <select id="colorChoice">
        <option value="0" selected="selected">Random</option>
        <option value="1">Red</option>
        <option value="2">Green</option>
        <option value="3">Blue</option>
        <option value="4">Cyan</option>
    </select>
    <select id="shapeChoice">
        <option value="0" selected="selected">None</option>
        <option value="1">Siedmiobok</option>
    </select>
    <button id="clearButton" >Clear</button>
</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>


<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: '0.9739079369679529', 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>
```

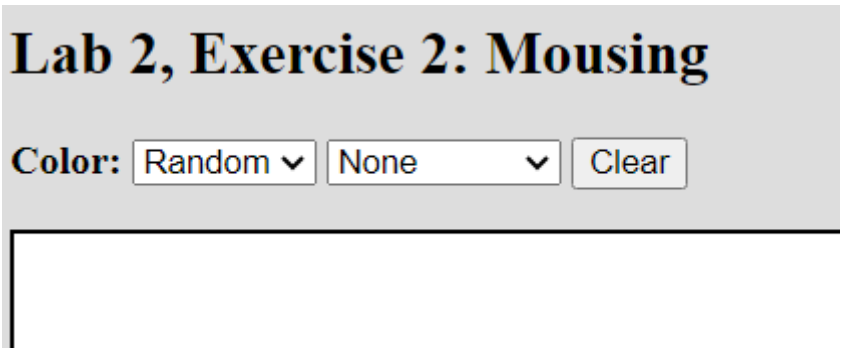
4. Wynik działania

Wynikiem wykonania pierwszego programu jest widoczna na Zdjęcie 2 grafika.

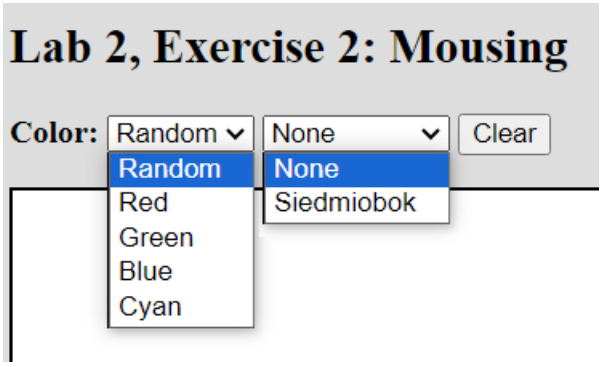


Zdjęcie 2 Wynik wykonania kodu numer 1

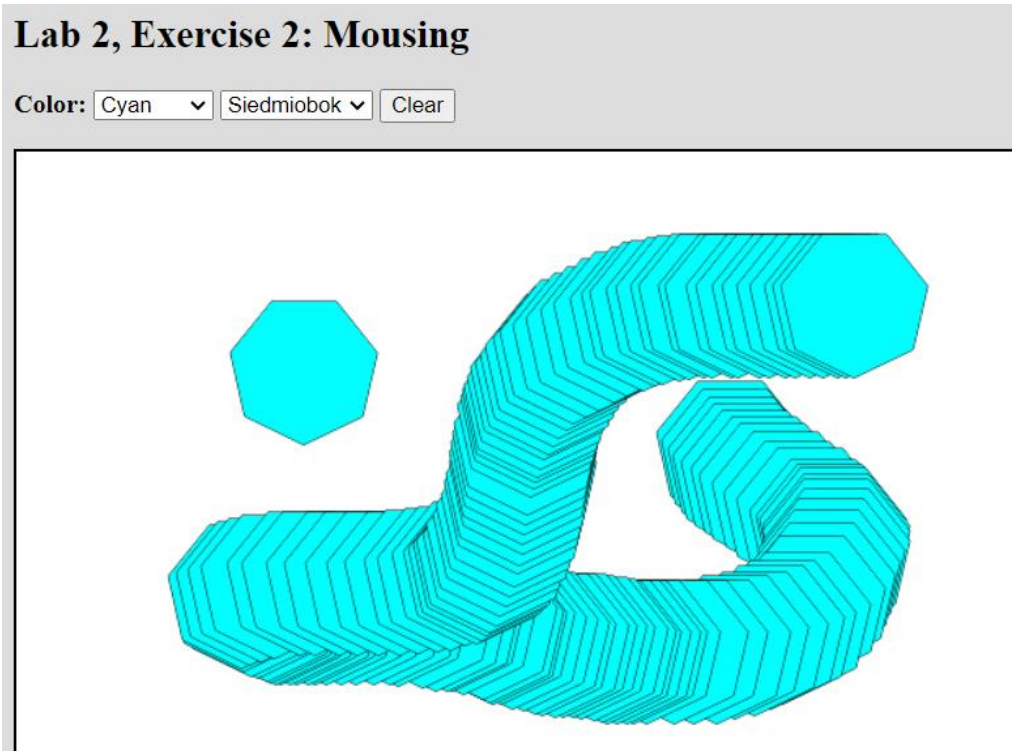
W wyniku wykonania kodu z polecenia drugiego widoczny jest interfejs przedstawiony na Zdjęcie 3. Następnie z rozwijanej listy kolorów oraz typów pędzla można wybrać pożądany kształt co przedstawia Zdjęcie 4. Dodatkowym dodanym kolorem jest *cyan*. Narysowane nim kształty typu „siedmiobok” widoczne są na Zdjęcie 5. Działanie funkcji „clear()” jest oczywiste.



Zdjęcie 3 Okno po otwarcia pliku z zad 2



Zdjęcie 4 Listy typu <select>



Zdjęcie 5 Wynik rysowania zaimplementowanym pędzlem oraz kolorem

5. Wnioski

- HTML Canvas to stosunkowo łatwe do opanowania narzędzie pozwalające na tworzenie różnego rodzaju prostych rysunków
- Zastosowanie javascript'u pozwala dodać logikę aplikacji do istniejącego kodu HTML