SPRAWOZDANIE

Zajęcia: Grafika komputerowa

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium: 2

Data: 02.03.2022

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

Wariant: 9

Przemysław Holisz

Informatyka I stopień, stacjonarne,

4 semestr, Gr.4/2b

1. Polecenie:

Narysowa´c obraz zgodnie z wariantem zadania (patrz Fig. 1) (u˙zywaja¸c zar´owno standardowe jak i niestandardowe funkcje rysowania).

2. Wprowadzane dane:

```
// Romb

        graphics.beginPath();
        graphics.fillStyle = "#0015ff";
        graphics.fillPoly(250,110,175,210,250,310,325,210);
        graphics.stroke();
        graphics.closePath();

        // Oczy
        graphics.beginPath();
        graphics.fillStyle = "white";
        graphics.arc(220,190,7,0,Math.PI *2);
        graphics.fill();
        graphics.stroke();

        graphics.beginPath();
        graphics.fillStyle = "white";
        graphics.arc(280,190,7,0,Math.PI *2);
        graphics.fill();
        graphics.stroke();
        graphics.closePath();

        graphics.beginPath();
        graphics.fillStyle = "black";
        graphics.arc(220,190,3,0,Math.PI *2);
        graphics.fill();
        graphics.stroke();

        graphics.beginPath();
        graphics.fillStyle = "black";
        graphics.arc(280,190,3,0,Math.PI *2);
        graphics.fill();
        graphics.stroke();

        graphics.beginPath();
        graphics.fillStyle = "white";
        graphics.arc(278,188,1,0,Math.PI *2);
        graphics.fill();

        graphics.beginPath();
```

```
        graphics.fillStyle = "white";
        graphics.arc(218,188,1,0,Math.PI *2);
        graphics.fill();
        graphics.closePath();

        // Usta
        graphics.beginPath();
        graphics.fillStyle = "black";
        graphics.bezierCurveTo(220, 210, 245, 270, 278, 210);
        graphics.fill();
        graphics.stroke();

        graphics.beginPath();
        graphics.fillStyle = "#0015ff";
        graphics.ellipse(249, 210, 15, 30, Math.PI / 2, 0, 2 * Math.PI);
        graphics.fill();

        graphics.beginPath();
        graphics.fillStyle = "black";
        graphics.arc(273,219,3,0,Math.PI *2);
        graphics.fill();
        graphics.stroke();

        graphics.beginPath();
        graphics.fillStyle = "#0015ff";
        graphics.arc(274,217.5,4,0,Math.PI *2);
        graphics.fill();

        graphics.beginPath();
        graphics.fillStyle = "black";
        graphics.arc(224,217.5,3,0,Math.PI *2);
        graphics.fill();
        graphics.stroke();

        graphics.beginPath();
        graphics.fillStyle = "#0015ff";
        graphics.arc(223,215.5,4,0,Math.PI *2);
        graphics.fill();

      // Zeby
      graphics.beginPath();
      graphics.fillStyle = "white";
      graphics.fillRect(245,225, 4,5);
      graphics.fill();

      graphics.beginPath();
```

```
        graphics.fillStyle = "white";
        graphics.fillRect(250,225, 4,5);
        graphics.fill();
```

3. Wykorzystane komendy:

```
<!DOCTYPE html>
<html>
<head>
<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.
```

```
28.
29.         var canvas;    // The canvas element on which we will draw.
30.         var graphics;  // A 2D graphics context for drawing on the
    canvas.
31.         var pixelSize; // The size of a pixel in the coordinate
    system; set up by
32.                        //    applyWindowToViewportTransform function
    when it is called.
33.
34.         /**
35.          *  The draw() function is called by init() after the page
    loads,
36.          *  to draw the content of the canvas.  At the start, clear
    the canvas
37.          *  and save a copy of the state; restore the state at the
    end.  (These
38.          *  actions are not necessary in this program, since the
    function will
39.          *  only be called once.)
40.          */
41.         function draw() {
42.
43.             graphics.clearRect(0,0,600,600);
44.
45.             // TODO: insert code to draw the image for Exercise 1
46.
47.             // Romb
48.             graphics.beginPath();
49.             graphics.fillStyle = "#0015ff";
50.             graphics.fillPoly(250,110,175,210,250,310,325,210);
51.             graphics.stroke();
52.             graphics.closePath();
53.
54.             // Oczy
```

```
55.             graphics.beginPath();
56.             graphics.fillStyle = "white";
57.             graphics.arc(220,190,7,0,Math.PI *2);
58.             graphics.fill();
59.             graphics.stroke();
60.
61.             graphics.beginPath();
62.             graphics.fillStyle = "white";
63.             graphics.arc(280,190,7,0,Math.PI *2);
64.             graphics.fill();
65.             graphics.stroke();
66.             graphics.closePath();
67.
68.             graphics.beginPath();
69.             graphics.fillStyle = "black";
70.             graphics.arc(220,190,3,0,Math.PI *2);
71.             graphics.fill();
72.             graphics.stroke();
73.
74.             graphics.beginPath();
75.             graphics.fillStyle = "black";
76.             graphics.arc(280,190,3,0,Math.PI *2);
77.             graphics.fill();
78.             graphics.stroke();
79.
80.             graphics.beginPath();
81.             graphics.fillStyle = "white";
82.             graphics.arc(278,188,1,0,Math.PI *2);
83.             graphics.fill();
84.
85.             graphics.beginPath();
86.             graphics.fillStyle = "white";
```

```
87.             graphics.arc(218,188,1,0,Math.PI *2);
88.             graphics.fill();
89.             graphics.closePath();
90.
91.             // Usta
92.             graphics.beginPath();
93.             graphics.fillStyle = "black";
94.             graphics.bezierCurveTo(220, 210, 245, 270, 278, 210);
95.             graphics.fill();
96.             graphics.stroke();
97.
98.             graphics.beginPath();
99.             graphics.fillStyle = "#0015ff";
100.            graphics.ellipse(249, 210, 15, 30, Math.PI / 2, 0, 2 *
    Math.PI);
101.            graphics.fill();
102.
103.            graphics.beginPath();
104.            graphics.fillStyle = "black";
105.            graphics.arc(273,219,3,0,Math.PI *2);
106.            graphics.fill();
107.            graphics.stroke();
108.
109.            graphics.beginPath();
110.            graphics.fillStyle = "#0015ff";
111.            graphics.arc(274,217.5,4,0,Math.PI *2);
112.            graphics.fill();
113.
114.            graphics.beginPath();
115.            graphics.fillStyle = "black";
116.            graphics.arc(224,217.5,3,0,Math.PI *2);
117.            graphics.fill();
```

```
118.           graphics.stroke();
119.
120.           graphics.beginPath();
121.           graphics.fillStyle = "#0015ff";
122.           graphics.arc(223,215.5,4,0,Math.PI *2);
123.           graphics.fill();
124.
125.         // Zeby
126.         graphics.beginPath();
127.         graphics.fillStyle = "white";
128.         graphics.fillRect(245,225, 4,5);
129.         graphics.fill();
130.
131.         graphics.beginPath();
132.         graphics.fillStyle = "white";
133.         graphics.fillRect(250,225, 4,5);
134.         graphics.fill();
135.
136.
137.
138.
139.
140.
141.
142.
143.
144.
145.
146.       } // end of draw()
147.
148.
149.       /**
```

```
150.         * Sets up a transformation in the graphics context so that
     the canvas will
151.         * show x-values in the range from left to right, and y-values
     in the range
152.         * from bottom to top.  If preserveAspect is true, then one of
     the ranges
153.         * will be increased, if necessary, to account for the aspect
     ratio of the
154.         * canvas.  This function sets the global variable pixelsize
     to be the
155.         * size of a pixel in the new coordinate system.  (If
     preseverAspect is
156.         * true, pixelSize is the maximum of its horizontal and
     vertical sizes.)
157.         */
158.        function
     applyWindowToViewportTransformation(left,right,bottom,top,preserveAspect
     ) {
159.            var displayAspect, windowAspect;
160.            var excess;
161.            var pixelwidth, pixelheight;
162.            if (preserveAspect) {
163.                // Adjust the limits to match the aspect ratio of the
     drawing area.
164.                displayAspect = Math.abs(canvas.height /
     canvas.width);
165.                windowAspect = Math.abs(( top-bottom ) / ( right-left
     ));
166.                if (displayAspect > windowAspect) {
167.                    // Expand the viewport vertically.
168.                    excess = (top-bottom) *
     (displayAspect/windowAspect - 1);
169.                    top = top + excess/2;
170.                    bottom = bottom - excess/2;
171.                }
172.                else if (displayAspect < windowAspect) {
```

```
173.                    // Expand the viewport vertically.
174.                    excess = (right-left) *
    (windowAspect/displayAspect - 1);
175.                    right = right + excess/2;
176.                    left = left - excess/2;
177.                }
178.            }
179.            graphics.scale( canvas.width / (right-left), canvas.height
    / (bottom-top) );
180.            graphics.translate( -left, -top );
181.            pixelwidth =  Math.abs(( right - left ) / canvas.width);
182.            pixelheight = Math.abs(( bottom - top ) / canvas.height);
183.            pixelSize = Math.max(pixelwidth,pixelheight);
184.        }  // end of applyWindowToViewportTransformation()
185.
186.        /**
187.         * This function can be called to add a collection of extra
    drawing function to
188.         * a graphics context, to make it easier to draw basic shapes
    with that context.
189.         * The parameter, graphics, must be a canvas 2d graphics
    context.
190.         *
191.         * The following new functions are added to the graphics
    context:
192.         *
193.         *    graphics.strokeLine(x1,y1,x2,y2) -- stroke the line from
    (x1,y1) to (x2,y2).
194.         *    graphics.fillCircle(x,y,r) -- fill the circle with
    center (x,y) and radius r.
195.         *    graphics.strokeCircle(x,y,r) -- stroke the circle.
196.         *    graphics.fillOval(x,y,r1,r2) -- fill oval with center
    (x,y) and radii r1 and r2.
197.         *    graphics.stokeOval(x,y,r1,r2) -- stroke the oval
```

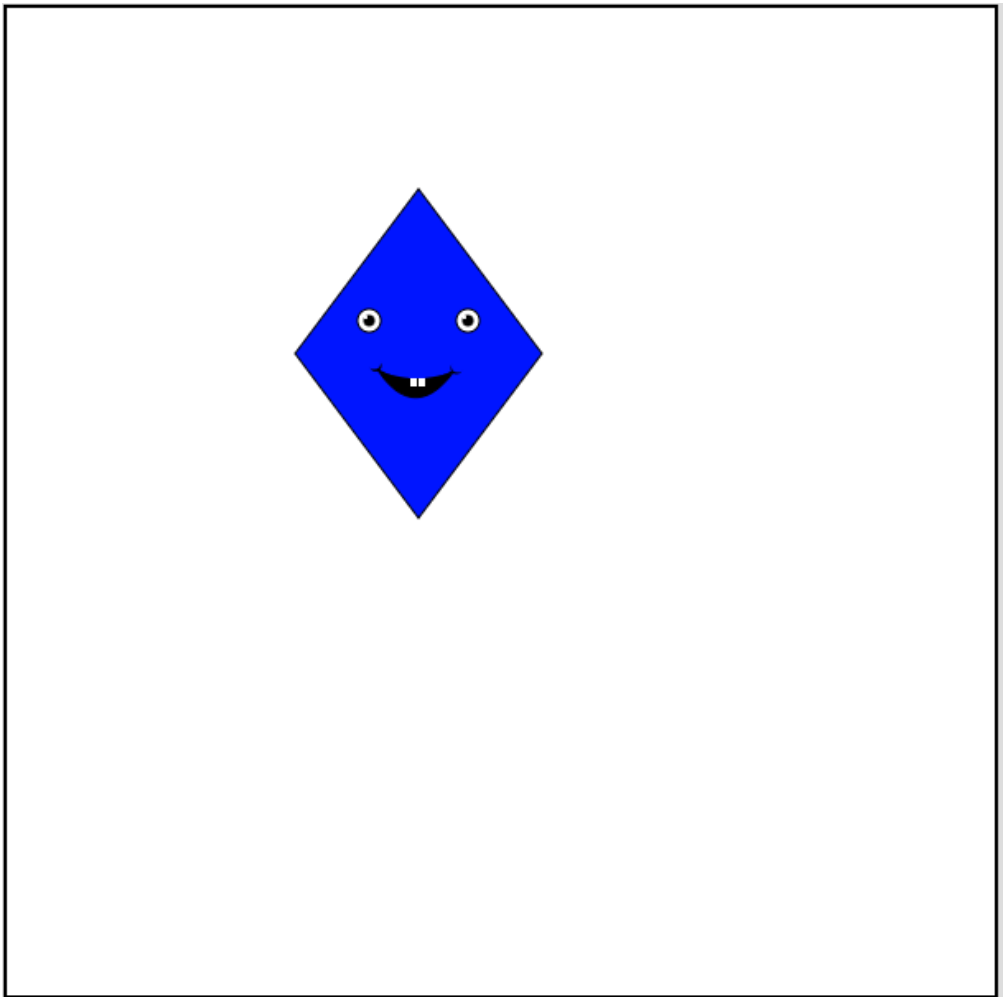
```
198.         *    graphics.fillPoly(x1,y1,x2,y2,...) -- fill polygon with
     vertices (x1,y1), (x2,y2), ...
199.         *    graphics.strokePoly(x1,y1,x2,y2,...) -- stroke the
     polygon.
200.         *    graphics.getRGB(x,y) -- returns the color components of
     pixel at (x,y) as an array of
201.         *         four integers in the range 0 to 255, in the order
     red, green, blue, alpha.
202.         *
203.         * (Note that "this" in a function that is called as a member
     of an object refers to that
204.         * object.  Here, this will refer to the graphics context.)
205.         */
206.        function addGraphicsContextExtras(graphics) {
207.            graphics.strokeLine = function(x1,y1,x2,y2) {
208.                this.beginPath();
209.                this.moveTo(x1,y1);
210.                this.lineTo(x2,y2);
211.                this.stroke();
212.            }
213.            graphics.fillCircle = function(x,y,r) {
214.                this.beginPath();
215.                this.arc(x,y,r,0,2*Math.PI,false);
216.                this.fill();
217.            }
218.            graphics.strokeCircle = function(x,y,radius) {
219.                this.beginPath();
220.                this.arc(x,y,radius,0,2*Math.PI,false);
221.                this.stroke();
222.            }
223.            graphics.fillPoly = function() {
224.                if (arguments.length < 6)
225.                    return;
```

```
226.                 this.beginPath();
227.                 this.moveTo(arguments[0],arguments[1]);
228.                 for (var i = 2; i+1 < arguments.length; i = i + 2) {
229.                    this.lineTo(arguments[i],arguments[i+1]);
230.                 }
231.                 this.closePath();
232.                 this.fill();
233.             }
234.             graphics.strokePoly = function() {
235.                 if (arguments.length < 4)
236.                    return;
237.                 this.beginPath();
238.                 this.moveTo(arguments[0],arguments[1]);
239.                 for (var i = 2; i+1 < arguments.length; i = i + 2) {
240.                    this.lineTo(arguments[i],arguments[i+1]);
241.                 }
242.                 this.closePath();
243.                 this.stroke();
244.             }
245.             graphics.fillOval =
    function(x,y,horizontalRadius,verticalRadius) {
246.                 this.save();
247.                 this.translate(x,y);
248.                 this.scale(horizontalRadius,verticalRadius);
249.                 this.beginPath();
250.                 this.arc(0,0,1,0,2*Math.PI,false);
251.                 this.restore();
252.                 this.fill();
253.             }
254.             graphics.strokeOval =
    function(x,y,horizontalRadius,verticalRadius) {
255.                 this.save();
```

```
256.              this.translate(x,y);
257.              this.scale(horizontalRadius,verticalRadius);
258.              this.beginPath();
259.              this.arc(0,0,1,0,2*Math.PI,false);
260.              this.restore();
261.              this.stroke();
262.          }
263.          graphics.getRGB = function(x,y) {
264.              var color = this.getImageData(x,y,1,1);
265.              return color.data;
266.          }
267.      }    // end of addGraphicsContextExtras()
268.
269.      /**
270.       * The init() funciton is called after the page has been
271.       * loaded.  It initializes the canvas and graphics variables.
272.       * It calles addGraphicsContextExtras(graphics) to add the
      extra
273.       * drawing functions to the graphics context, and it calls
      draw()
274.       * to draw on the canvas.
275.       */
276.      function init() {
277.          try {
278.              canvas = document.getElementById("canvas");
279.              graphics = canvas.getContext("2d");
280.          } catch(e) {
281.              document.getElementById("canvasholder").innerHTML =
282.                 "Canvas graphics is not supported.<br>" +
283.                 "An error occurred while initializing graphics.";
284.          }
285.          addGraphicsContextExtras(graphics);
```

```
286.            draw();  // Call draw() to draw on the canvas.
287.        }
288.
289.    </script>
290.    </head>
291.    <body onload="init()"> <!-- the onload attribute here is what
    calls the init() function -->
292.
293.    <h2>CS 424, Lab 2, Exercise 1</h2>
294.
295.    <noscript>
296.        <!-- This message will be shown in the page if JavaScript is
    not available. -->
297.    <p>JavaScript is required to use this page.</p>
298.    </noscript>
299.
300.    <div id="canvasholder">
301.    <canvas id="canvas" width="600" height="600">
302.        <!-- This message is shown on the page if the browser doesn't
    support the canvas element. -->
303.    Canvas not supported.
304.    </canvas>
305.    </div>
306.
307.    </body>
308.    </html>
309.
```

310. Wynik działania:



311. Polecenie:

W pliku Lab2Ex2.html program domy´slnie rysuje szereg kwadrat´ow. Stworzy´c narze¸dzia pozwalaja¸ce na wykonywanie czynno´sci • "czyszczenie" canvasu - Clear button: Clear (Hint! Przy inicializacji musi by´c document.getElementById("clearButton").onclick = doClear;) • dodanie jednego nowego koloru do elementu

312. Wprowadzane dane:

```
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 = "yellow";
        }

        if (figureChoice == 0) {
            graphics.fillRect(x-20,y-20,40,40);
            graphics.strokeRect(x-20,y-20,40,40);
        }
        else if (figureChoice == 1)
        {
            // Figura romb
             graphics.strokePoly(x,y-25,x+20,y,x,y+25,x-20,y);
             graphics.fillPoly(x,y-25,x+20,y,x,y+25,x-20,y);
        }
```

313. Wykorzystane komendy:

```
<!DOCTYPE html>
<html>
<!--
   This web page does the minimal setup for using mouse events along
   with 2D canvas graphics.
-->
<head>
<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 + ")";
    }

    /**
     * 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 figureChoice;

        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);
            }


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

             // TODO: Anything else to do when mouse is first pressed?
            colorChoice =
Number(document.getElementById("colorChoice").value);

        }

           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 = "yellow";
            }
```

```
            if (figureChoice == 0) {
                graphics.fillRect(x-20,y-20,40,40);
                graphics.strokeRect(x-20,y-20,40,40);
            }
            else if (figureChoice == 1)
            {
                // Figura romb
                 graphics.strokePoly(x,y-25,x+20,y,x,y+25,x-20,y);
                 graphics.fillPoly(x,y-25,x+20,y,x,y+25,x-20,y);
            }

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

            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);
    }

       function Clear(){
            graphics.clearRect(0,0,canvas.width,canvas.height);
        }

</script>
</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>Figura:</b>
    <select id="figureChoice">
        <option value="0">Square</option>
        <option value="1">Romb</option>
    </select>
<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>
        <option value="4">Yellow</option>
    </select>
    <button onclick="Clear()">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>

</body>
</html>
```

314. Wynik działania: