SPRAWOZDANIE

Zajęcia: Grafika komputerowa
Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium 2
01.03.2022
Temat:" Grafika 2D z użyciem HTML Canvas "
Wariant 2

Bartosz Medoń
Informatyka I stopień,
stacjonarne,
4 semestr,
Gr.1a

1. **Polecenie: Wyświetlenie odpowiedniego wariantu obrazka. Dodanie opcji czyszczenia ekranu, nowego koloru, pędzla w odpowiednim kształcie.**
2. **Wykorzystane komendy:**

a)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>CPSC 424, Lab 2, Exercise 1</title>
<script type="text/javascript" src="https://gc.kis.v2.scr.kaspersky-
labs.com/2F4D86ED-AB58-C748-9FD7-45B5EB35903E/main.js"
charset="UTF-8"></script>
<style>
   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);

   //serce
        graphics.beginPath();
        graphics.fillStyle = "#FF0000";
   graphics.moveTo(300, 300);
        graphics.bezierCurveTo(150, 150, 300, 150, 300, 180);
        graphics.bezierCurveTo(300, 150, 450, 150, 300, 300);
        graphics.fill();
        graphics.stroke();


        //oczy
        graphics.beginPath();
        graphics.fillStyle = "#FFFFFF";
        graphics.strokeStyle ="#FFFFFF";
        graphics.fillCircle(270,200,8);
        graphics.fillCircle(335,200,8);
        graphics.fill();

        graphics.fillStyle = "#000000";
        graphics.strokeStyle ="#000000";
        graphics.fillCircle(268,200,4);
        graphics.fillCircle(333,200,4);
        graphics.fill();

        graphics.fillStyle = "#FFFFFF";
        graphics.fillCircle(266.8,199.5,1.5);
        graphics.fillCircle(331.8,199.5,2);
        graphics.fill();
        graphics.stroke();

        //usta
        graphics.beginPath();
   graphics.fillStyle = "#000000";
        graphics.strokeStyle ="#000000";
   graphics.moveTo(280, 250);
        graphics.bezierCurveTo(290, 265, 310, 265, 320, 250);
```

```
    graphics.bezierCurveTo(310, 255, 290, 255, 280, 250);
            graphics.fill();
            graphics.moveTo(279, 245);
      graphics.bezierCurveTo(279.5, 250, 279.5, 250, 277, 252);
      graphics.moveTo(321, 245);
      graphics.bezierCurveTo(319.5, 250, 319.5, 250, 323.5, 252);
            graphics.stroke();
            //

            //zeby
            graphics.beginPath();
            graphics.fillStyle = "#FFFFFF";
            graphics.moveTo(300, 254.5);
            graphics.lineTo(300, 257.5);
      graphics.lineTo(296, 257.5);
      graphics.lineTo(296, 254.5);
      graphics.moveTo(300, 257.5);
      graphics.lineTo(304, 257.5);
      graphics.lineTo(304, 254.5);
      graphics.lineTo(300, 254.5);
      graphics.lineTo(296, 254.5);
        graphics.fill();
            graphics.stroke();
        graphics.closePath();

    }


    /**
     * 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,preserveAspe
ct) {
        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 -->

<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>
</body>
</html>
```

b)

```
<html>
<head>
<meta charset="UTF-8">
<title>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 ClearAll()
    {
        graphics.clearRect(0,0,800,600);
    }
    /**
     * 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"
      var shapeChoice;  // 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?
         shapeChoice =
Number(document.getElementById("shapeChoice").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 = "gray";
        }
                else if (colorChoice == 5) {
           graphics.fillStyle = "lime";
        }

        if (shapeChoice == 0) {
           graphics.fillRect(x-20,y-20,40,40);
           graphics.strokeRect(x-20,y-20,40,40);
        }
        else if (shapeChoice == 1)
        {
```

```
                        graphics.beginPath();
                        for (let i = 0; i < 15; i++) {
                        let px = x + Math.cos(i * (Math.PI * 2) / 15) * 50, py
= y + Math.sin(i * (Math.PI * 2) / 15) * 50;
                        graphics[i === 0 ? 'moveTo' : 'lineTo'](px, py);
                        }
                        graphics.fill();
                        graphics.closePath();
                        graphics.stroke();
          }
          /*-----------------------------------------------------------*/

          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();
```

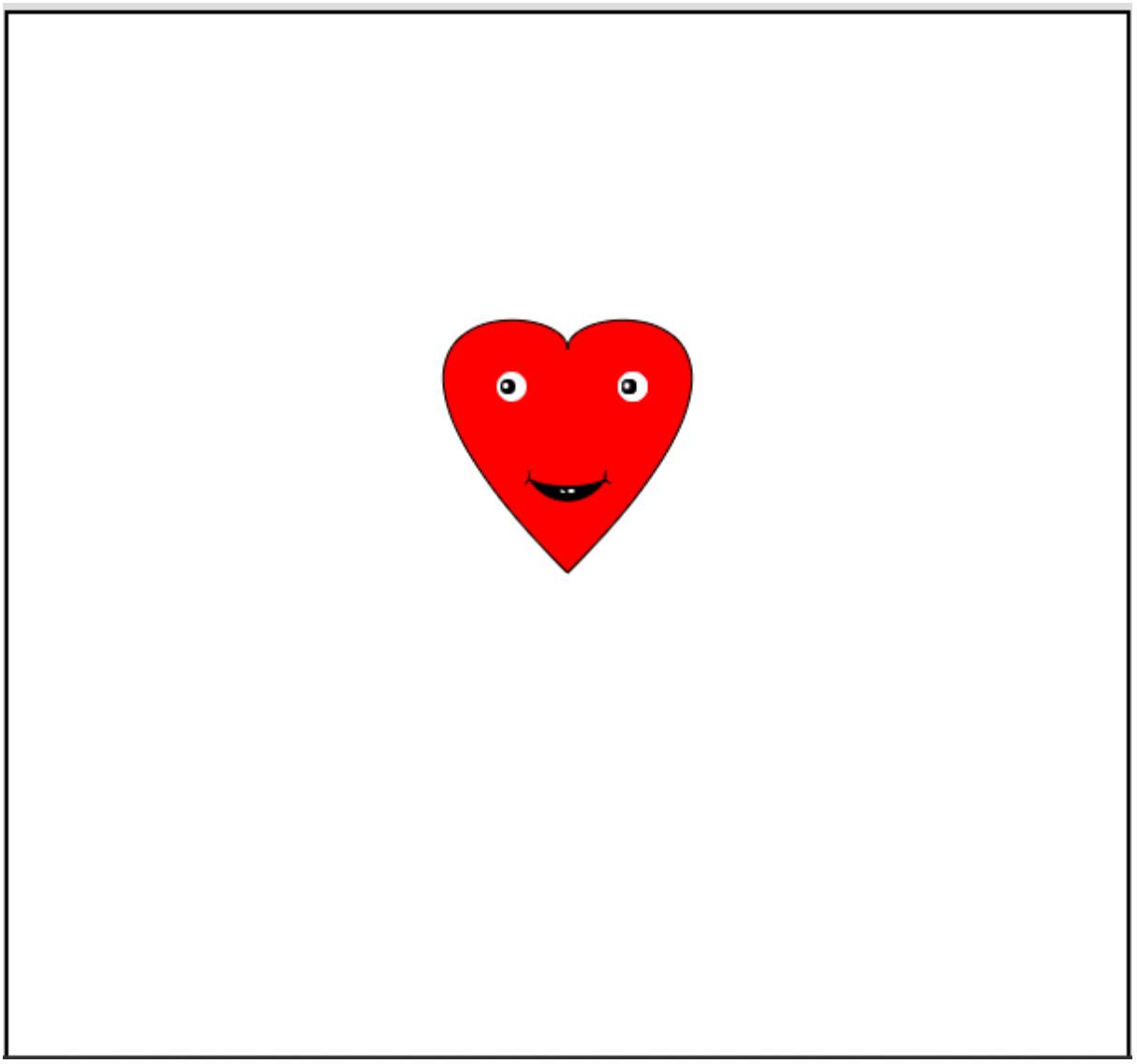
```
    }
    graphics.strokePoly = function() {
        if (arguments.length < 4)
          return;
        this.beginPath();
        this.moveTo(arguments[0],arguments[1]);
        for (var i = 2; i+1 < arguments.length; i = i + 2) {
          this.lineTo(arguments[i],arguments[i+1]);
        }
        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 -->

<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">Random</option>
      <option value="1">Red</option>
      <option value="2">Green</option>
      <option value="3">Blue</option>
      <option value="4">Gray</option>
            <option value="5">Lime</option>
   </select>
   <b>Style:</b>
   <select id="shapeChoice">
      <option value="0">Square</option>
      <option value="1">Poli</option>
   </select>
```

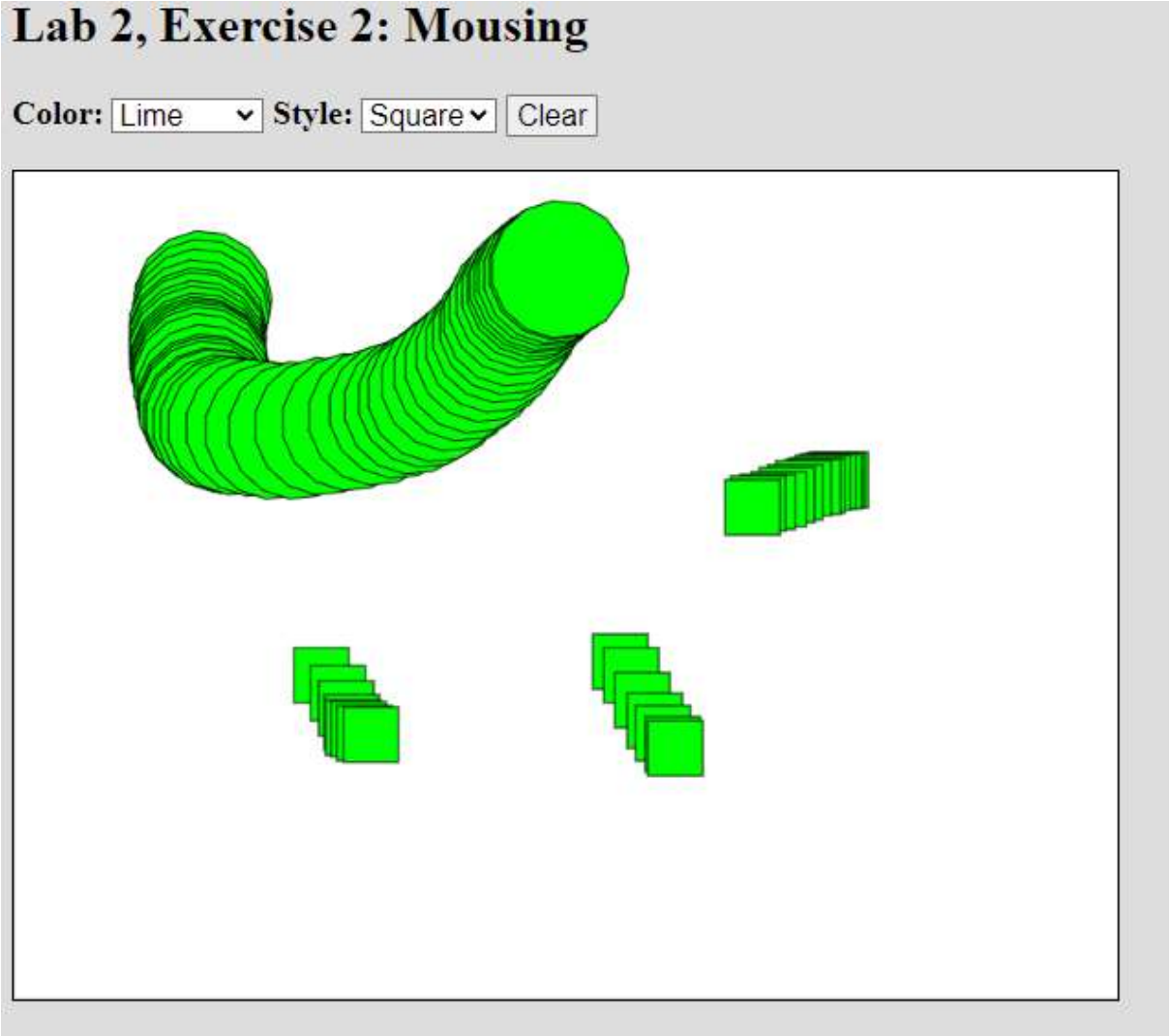
```
    <button onclick="ClearAll()">Clear</button>
</p>

<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>
```

3. Wynik:

a)



b)



4. **Wnioski:** HTML wraz z JavaScript umożliwiają wyświetlanie grafiki w przeglądarce.