COMP3421 Exercise 4 (HTML Canvas)

Name:

Student ID:

1. **Modify the Javascript code below to display the "seconds hand" in the webpage. For a date() object, the three methods to get the hour, minute and second are getHours(), getMinutes(), and getSeconds(), respectively.**

```
<!DOCTYPE html>
<html>
<head>
<title>Clock</title>

<style>
body { background: #dddddd; }

#canvas {
position: absolute;
left: 0px;
top: 0px;
margin: 20px;
background: #ffffff;
border: thin solid #aaaaaa;
}
</style>
</head>

<body>
<canvas id='canvas' width='400' height='400'>
Canvas not supported
</canvas>

<script src='lex1.js'></script>
</body></html>
```

```
var canvas = document.getElementById('canvas'),   context = canvas.getContext('2d'),
FONT_HEIGHT = 15,                                 MARGIN = 35,
HAND_TRUNCATION = canvas.width/25,HOUR_HAND_TRUNCATION = canvas.width/10,
NUMERAL_SPACING = 20,                             RADIUS = canvas.width/2 - MARGIN,
HAND_RADIUS = RADIUS + NUMERAL_SPACING;

function drawCircle() {
  context.beginPath();
  context.arc(canvas.width/2, canvas.height/2, RADIUS, 0, Math.PI*2, true);
  context.stroke();
}
function drawNumerals() {
  var numerals = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ],      angle = 0,  numeralWidth = 0;
  numerals.forEach(function(numeral) {                   angle = Math.PI/6 * (numeral-3);
    numeralWidth = context.measureText(numeral).width;
    context.fillText(numeral,
        canvas.width/2  + Math.cos(angle)*(HAND_RADIUS) - numeralWidth/2,
        canvas.height/2 + Math.sin(angle)*(HAND_RADIUS) + FONT_HEIGHT/3);
  });
}
function drawCenter() {
  context.beginPath();
  context.arc(canvas.width/2, canvas.height/2, 5, 0, Math.PI*2, true);
  context.fill();
}
function drawHand(loc, isHour) {
  var angle = (Math.PI*2) * (loc/60) - Math.PI/2,
  handRadius = isHour ? RADIUS - HAND_TRUNCATION-HOUR_HAND_TRUNCATION
                : RADIUS - HAND_TRUNCATION;
  context.moveTo(canvas.width/2, canvas.height/2);
  context.lineTo(canvas.width/2  + Math.cos(angle)*handRadius,
          canvas.height/2 + Math.sin(angle)*handRadius);
  context.stroke();
}
function drawHands() {
  var date = new Date,
  hour = date.getHours();
  hour = hour > 12 ? hour - 12 : hour;
  drawHand(hour*5 + (date.getMinutes()/60)*5, true);
  drawHand(date.getMinutes(), false);
}
function drawClock() {
  context.clearRect(0,0,canvas.width,canvas.height);
  drawCircle();   drawCenter();   drawHands();   drawNumerals();}
// Initialization................................................
context.font = FONT_HEIGHT + 'px Arial';          loop = setInterval(drawClock, 1000);
```

2. Complete the Javascript code to create 2 rectangles as shown and remove them which a mouse click occurs.

```
<!DOCTYPE html>
<head>
<title>Rectangles</title>
<style>
body {  background: #dddddd; }

#canvas {
        background: #eeeeee;    border: thin solid #aaaaaa;
}
</style>
</head>
<body>
<canvas id='canvas' width='600' height='400'>
Canvas not supported
</canvas>
<script src='lex2.js'></script>
</body>
</html>
```

```
var canvas = document.getElementById('canvas'),
   context = canvas.getContext('2d');

context.lineWidth = 30;
context.font = '24px Helvetica';
context.fillText('Click anywhere to erase', 175, 40);

// MISSING LINES

context.canvas.onmousedown = function (e) {
  // MISSING LINES
};
```

Include CANVAS rect methods here.

3. Complete the Javascript code to enable a slider to zoom in an image. The zoom can be at any starting point of an image but with the right scale. You can download any image you like to work with.

```
<!DOCTYPE html>

<html>
<head>
<title>Scaling images</title>

<style>
body {  background: rgba(100, 145, 250, 0.3);    }

#scaleSlider {
vertical-align: 10px;      width: 100px;   margin-left: 90px;         }

#canvas {
margin: 10px 20px 0px 20px;    border: thin solid #aaaaaa;         cursor: crosshair;           }

#controls {
margin-left: 15px;         padding: 0;       }

#scaleOutput {
position: absolute;        width: 60px;     height: 30px;
margin-left: 10px;         vertical-align: center;    text-align: center;
color: blue;               font: 18px Arial;
text-shadow: 2px 2px 4px rgba(100, 140, 250, 0.8);        }

</style>
</head>
<body>
<div id='controls'>
<output id='scaleOutput'>1.0</output>
<input id='scaleSlider' type='range' min='1' max='3.0' step='0.01' value='1.0'/>
</div>
<canvas id='canvas' width='800' height='520'>
Canvas not supported
</canvas>
<script src='lex3.js'></script>

</body>
</html>
```

```
var canvas = document.getElementById('canvas'),
context = canvas.getContext('2d'),
image = new Image(),
scaleSlider = document.getElementById('scaleSlider'),
scaleOutput = document.getElementById('scaleOutput'),
scale = 1.0,
MINIMUM_SCALE = 1.0,
MAXIMUM_SCALE = 3.0;

function drawImage() {
  var     w = canvas.width,   h = canvas.height,
          sw = w * scale,     sh = h * scale;

          // MISSING LINES(2 steps, clear the canvas, redraw the portion of the image)
}

function drawScaleText(value) {
  var text = parseFloat(value).toFixed(2);
  var percent = parseFloat(value - MINIMUM_SCALE) /
                parseFloat(MAXIMUM_SCALE - MINIMUM_SCALE);

  scaleOutput.innerText = text;
  percent = percent < 0.35 ? 0.35 : percent;
          scaleOutput.style.fontSize = percent*MAXIMUM_SCALE/1.5 + 'em';
}

// Event Handlers..............................................
scaleSlider.onchange = function(e) {
  scale = e.target.value;
  if (scale < MINIMUM_SCALE) scale = MINIMUM_SCALE;
  else if (scale > MAXIMUM_SCALE) scale = MAXIMUM_SCALE;
  drawScaleText(scale);
  drawImage();
}

// Initialization..............................................

context.fillStyle     = 'cornflowerblue';
context.strokeStyle   = 'yellow';
context.shadowColor   = 'rgba(50, 50, 50, 1.0)';
context.shadowOffsetX = 5;
context.shadowOffsetY = 5;
context.shadowBlur    = 10;

image.src = 'heading1.gif';
image.onload = function(e) {
          drawImage();
          drawScaleText(scaleSlider.value);
};
```

4. Modified the Javascript code such that the discs are only moving horizontally in the same direction.

```
<!DOCTYPE html>
<head>
<title>Discsl Moving</title>
<style>
body {  background: #dddddd;   }

#canvas {
        background: #ffffff;       cursor: pointer;
        margin-left: 10px;         margin-top: 10px;
        -webkit-box-shadow: 3px 3px 6px rgba(0,0,0,0.5);
        -moz-box-shadow: 3px 3px 6px rgba(0,0,0,0.5);
        box-shadow: 3px 3px 6px rgba(0,0,0,0.5);
}

#controls {
        margin-top: 10px;          margin-left: 15px;
}
</style>
</head>
<body>
<div id='controls'>
<input id='animateButton' type='button' value='Animate'/>
</div>

<canvas id='canvas' width='750' height='500'>
Canvas not supported
</canvas>

<script src='NextAnimationFrame.js'></script>
<script src='lex4.js'></script>

</body>
</html>
```

```
window.requestNextAnimationFrame =    (function () {
    var originalWebkitRequestAnimationFrame = undefined,
        wrapper = undefined,
        callback = undefined,
        geckoVersion = 0,
        userAgent = navigator.userAgent,
        index = 0,
        self = this;

    // Workaround for Chrome 10 bug where Chrome does not pass the time to the animation function
    if (window.webkitRequestAnimationFrame) {         // Define the wrapper
      wrapper = function (time) {
        if (time === undefined) {
          time = +new Date();
        }
        self.callback(time);
      };

      // Make the switch
      originalWebkitRequestAnimationFrame = window.webkitRequestAnimationFrame;
      window.webkitRequestAnimationFrame = function (callback, element) {
        self.callback = callback;

        // Browser calls the wrapper and wrapper calls the callback
        originalWebkitRequestAnimationFrame(wrapper, element);
      }      }

    // Workaround for Gecko 2.0, which has a bug in mozRequestAnimationFrame() that
    // restricts animations to 30-40 fps.
    if (window.mozRequestAnimationFrame) {
      // Check the Gecko version. Gecko is used by browsers other than Firefox.
      // Gecko 2.0 corresponds to Firefox 4.0.

      index = userAgent.indexOf('rv:');
      if (userAgent.indexOf('Gecko') != -1) {
        geckoVersion = userAgent.substr(index + 3, 3);
        if (geckoVersion === '2.0') {
          // Forces the return statement to fall through to the setTimeout() function.
          window.mozRequestAnimationFrame = undefined;
        }        }      }
   return window.requestAnimationFrame   ||        window.webkitRequestAnimationFrame ||
        window.mozRequestAnimationFrame    ||        window.oRequestAnimationFrame      ||
        window.msRequestAnimationFrame     ||        function (callback, element) {
                                                    var start,            finish;
                                                    window.setTimeout( function () {
          start = +new Date();
          callback(start);
          finish = +new Date();
          self.timeout = 1000 / 60 - (finish - start);
        }, self.timeout);
      };    }   ) ();
```

```
var canvas = document.getElementById('canvas'),
    context = canvas.getContext('2d'),
    paused = true,
    discs = [
      { x: 150,     y: 250,     lastX: 150,     lastY: 250,     velocityX: -3.2,     velocityY: 3.5,
        radius: 25,     innerColor: 'rgba(255,255,0,1)',     middleColor: 'rgba(255,255,0,0.7)',
        outerColor: 'rgba(255,255,0,0.5)',     strokeStyle: 'gray',     },

      { x: 50,     y: 150,     lastX: 50,     lastY: 150,     velocityX: 2.2,     velocityY: 2.5,
        radius: 25,     innerColor: 'rgba(100,145,230,1.0)',     middleColor: 'rgba(100,145,230,0.7)',
        outerColor: 'rgba(100,145,230,0.5)',     strokeStyle: 'blue'     },

      {  x: 150,     y: 75,     lastX: 150,     lastY: 75,     velocityX: 1.2,     velocityY: 1.5,
        radius: 25,     innerColor: 'rgba(255,0,0,1.0)',     middleColor: 'rgba(255,0,0,0.7)',
        outerColor: 'rgba(255,0,0,0.5)',     strokeStyle: 'orange'
      },
    ],
    numDiscs = discs.length,
    animateButton = document.getElementById('animateButton');

// Functions.....................................................
function drawBackground() {
   var STEP_Y = 12,     i = context.canvas.height;
   context.strokeStyle = 'lightgray';
   context.lineWidth = 0.5;
   while(i > STEP_Y*4) {
      context.beginPath();
      context.moveTo(0, i);
      context.lineTo(context.canvas.width, i);
      context.stroke();
      i -= STEP_Y;
   }
   context.save();
   context.strokeStyle = 'rgba(100,0,0,0.3)';
   context.lineWidth = 1;
   context.beginPath();
   context.moveTo(35,0);
   context.lineTo(35,context.canvas.height);
   context.stroke();
   context.restore();
}

function update() {
   var disc = null;
   for(var i=0; i < numDiscs; ++i) {
      disc = discs[i];
      if (disc.x + disc.velocityX + disc.radius > context.canvas.width ||
          disc.x + disc.velocityX - disc.radius < 0)        disc.velocityX = -disc.velocityX;
      if (disc.y + disc.velocityY + disc.radius > context.canvas.height ||
          disc.y + disc.velocityY - disc.radius  < 0)        disc.velocityY= -disc.velocityY;
      disc.x += disc.velocityX;
      disc.y += disc.velocityY;
    }
}

function draw() {
   var disc = discs[i];
   for(var i=0; i < numDiscs; ++i) {
      disc = discs[i];
      gradient = context.createRadialGradient(disc.x, disc.y, 0,disc.x, disc.y, disc.radius);

      gradient.addColorStop(0.3, disc.innerColor);
      gradient.addColorStop(0.5, disc.middleColor);
      gradient.addColorStop(1.0, disc.outerColor);

      context.save();
      context.beginPath();
      context.arc(disc.x, disc.y, disc.radius, 0, Math.PI*2, false);
      context.fillStyle = gradient;
      context.strokeStyle = disc.strokeStyle;
      context.fill();
      context.stroke();
      context.restore();
   }
}

// Animation....................................................
function animate(time) {
   if (!paused) {
      context.clearRect(0,0,canvas.width,canvas.height);
      drawBackground();
      update();
      draw();
      window.requestNextAnimationFrame(animate);
   }
}

// Initialization...............................................

context.font = '48px Helvetica';
animateButton.onclick = function (e) {
   paused = paused ? false : true;
   if (paused) {      animateButton.value = 'Animate';   }
   else {
    window.requestNextAnimationFrame(animate);
      animateButton.value = 'Pause';
   }
};
```

5. Modified the Javascript code to the possible collision between 2 shapes.

```
<!DOCTYPE html>

<html>
<head>
<title>Polygon Collision Detection</title>

<style>
#canvas {
            background: lightskyblue; cursor: pointer;
            -webkit-box-shadow: 4px 4px 8px rgba(0,0,0,0.5);
            -moz-box-shadow: 4px 4px 8px rgba(0,0,0,0.5);
            box-shadow: 4px 4px 8px rgba(0,0,0,0.5);
}
</style>
</head>

<body>
<canvas id='canvas' width='600' height='400'>
Canvas not supported
</canvas>

<script src = 'shapes.js'></script>
<script src = 'lex5.js'></script>
</body>
</html>
```

```
var canvas = document.getElementById('canvas'),
    context = canvas.getContext('2d'),
    shapes = [],
    polygonPoints = [
      [ new Point(250, 150), new Point(250, 250),  new Point(350, 250) ],

      [ new Point(100, 100), new Point(100, 150),  new Point(150, 150), new Point(150, 100) ],

      [ new Point(400, 100), new Point(380, 150),  new Point(500, 150), new Point(520, 100) ]
    ],
    polygonStrokeStyles = [ 'blue', 'yellow', 'red'],
    polygonFillStyles   = [ 'rgba(255,255,0,0.7)',  'rgba(100,140,230,0.6)', 'rgba(255,255,255,0.8)' ],
    mousedown = { x: 0, y: 0 },
    lastdrag = { x: 0, y: 0 },
    shapeBeingDragged = undefined;

// Functions.....................................................

function windowToCanvas(e) {
   var x = e.x || e.clientX,       y = e.y || e.clientY,
   bbox = canvas.getBoundingClientRect();
   return { x: x - bbox.left * (canvas.width  / bbox.width), y: y - bbox.top  * (canvas.height / bbox.height)
          };
};

function drawShapes() {
   shapes.forEach( function (shape) {
      shape.stroke(context);
      shape.fill(context);
   });
}

function detectCollisions() {

             // MISSING LINES
}

// Event handlers...............................................
canvas.onmousedown = function (e) {
   var location = windowToCanvas(e);
   shapes.forEach( function (shape) {
      if (shape.isPointInPath(context, location.x, location.y)) {
         shapeBeingDragged = shape;
         mousedown.x = location.x;
         mousedown.y = location.y;
         lastdrag.x = location.x;
         lastdrag.y = location.y;
      }
   });
}
```

```
canvas.onmousemove = function (e) {
  var location,      dragVector;
  if (shapeBeingDragged !== undefined) {
    location = windowToCanvas(e);
    dragVector = { x: location.x - lastdrag.x,  y: location.y - lastdrag.y   };
    shapeBeingDragged.move(dragVector.x, dragVector.y);
    lastdrag.x = location.x;
    lastdrag.y = location.y;
    context.clearRect(0,0,canvas.width,canvas.height);
    drawShapes();
    detectCollisions();
  }
}

canvas.onmouseup = function (e) {   shapeBeingDragged = undefined;   }

for (var i=0; i < polygonPoints.length; ++i) {
  var polygon = new Polygon(),      points = polygonPoints[i];
  polygon.strokeStyle = polygonStrokeStyles[i];
  polygon.fillStyle = polygonFillStyles[i];
  points.forEach( function (point) {     polygon.addPoint(point.x, point.y);   });
  shapes.push(polygon);
}

// Initialization................................................

context.shadowColor = 'rgba(100,140,255,0.5)';
context.shadowBlur = 4;
context.shadowOffsetX = 2;
context.shadowOffsetY = 2;
context.font = '38px Arial';
drawShapes();
context.save();
context.fillStyle = 'cornflowerblue';
context.font = '24px Arial';
context.fillText('Drag shapes over each other', 10, 25);
context.restore();
```

```
var Point = function (x, y) {   this.x = x;   this.y = y; };

var Shape = function () {
  this.x = undefined;
  this.y = undefined;
  this.strokeStyle = 'rgba(255, 253, 208, 0.9)';
  this.fillStyle = 'rgba(147, 197, 114, 0.8)';
};
Shape.prototype = {
   collidesWith: function (shape) {
         var axes = this.getAxes().concat(shape.getAxes());
         return !this.separationOnAxes(axes, shape);   },

  separationOnAxes: function (axes, shape) {
     for (var i=0; i < axes.length; ++i) {
       axis = axes[i];
       projection1 = shape.project(axis);
       projection2 = this.project(axis);
       if (! projection1.overlaps(projection2)) {
         return true; // don't have to test remaining axes
       }
     }
     return false;
  },

  move: function (dx, dy) {      throw 'move(dx, dy) not implemented';   },
  createPath: function (context) {      throw 'createPath(context) not implemented';   },
  getAxes: function () {      throw 'getAxes() not implemented';   },
  project: function (axis) {      throw 'project(axis) not implemented';    },

  fill: function (context) {
     context.save();
     context.fillStyle = this.fillStyle;
     this.createPath(context);
     context.fill();
     context.restore();
  },
  stroke: function (context) {
     context.save();
     context.strokeStyle = this.strokeStyle;
     this.createPath(context);
     context.stroke();
     context.restore();
  },
  isPointInPath: function (context, x, y) {
     this.createPath(context);
     return context.isPointInPath(x, y);
  },
};
```

```
var Projection = function (min, max) {
  this.min = min;
  this.max = max;
};
Projection.prototype = {
  overlaps: function (projection) {return this.max > projection.min && projection.max > this.min;
  }       };

var Vector = function(x, y) {
    this.x = x;
    this.y = y;
};
Vector.prototype = {
  getMagnitude: function () {      return Math.sqrt(Math.pow(this.x, 2) + Math.pow(this.y, 2));   },
  add: function (vector) {
    var v = new Vector();
    v.x = this.x + vector.x;
    v.y = this.y + vector.y;
    return v;
  },
  subtract: function (vector) {
    var v = new Vector();
    v.x = this.x - vector.x;
    v.y = this.y - vector.y;
    return v;
  },
  dotProduct: function (vector) {
    return this.x * vector.x +
           this.y * vector.y;
  },
  edge: function (vector) {
    return this.subtract(vector);
  },
  perpendicular: function () {
    var v = new Vector();
    v.x = this.y;
    v.y = 0-this.x;
    return v;
  },
  normalize: function () {
    var v = new Vector(),
        m = this.getMagnitude();
    v.x = this.x / m;
    v.y = this.y / m;
    return v;
  },
  normal: function () {
    var p = this.perpendicular();
    return p.normalize();
  }
};
```

```
var Polygon = function () {
  this.points = [];
  this.strokeStyle = 'blue';
  this.fillStyle = 'white';
};

Polygon.prototype = new Shape();
Polygon.prototype.getAxes = function () {
  var v1 = new Vector(),      v2 = new Vector(),      axes = [];
  for (var i=0; i < this.points.length-1; i++) {
    v1.x = this.points[i].x;
    v1.y = this.points[i].y;
    v2.x = this.points[i+1].x;
    v2.y = this.points[i+1].y;
    axes.push(v1.edge(v2).normal());
  }
  v1.x = this.points[this.points.length-1].x;
  v1.y = this.points[this.points.length-1].y;
  v2.x = this.points[0].x;
  v2.y = this.points[0].y;
  axes.push(v1.edge(v2).normal());
  return axes;
};
Polygon.prototype.project = function (axis) {
  var scalars = [], v = new Vector();
  this.points.forEach( function (point) {
    v.x = point.x;
    v.y = point.y;
    scalars.push(v.dotProduct(axis));
  });
  return new Projection(Math.min.apply(Math, scalars),  Math.max.apply(Math, scalars));
};
Polygon.prototype.addPoint = function (x, y) {   this.points.push(new Point(x,y)); };

Polygon.prototype.createPath = function (context) {
  if (this.points.length === 0)      return;
  context.beginPath();
  context.moveTo(this.points[0].x, this.points[0].y);
  for (var i=0; i < this.points.length; ++i) {
    context.lineTo(this.points[i].x,  this.points[i].y);
  }
  context.closePath();
};

Polygon.prototype.move = function (dx, dy) {
  var point, x;
  for(var i=0; i < this.points.length; ++i) {
    point = this.points[i];
    point.x += dx;
    point.y += dy;
  }
```

```
};

Polygon.prototype.move = function (dx, dy) {
  for (var i=0, point; i < this.points.length; ++i) {
    point = this.points[i];
    point.x += dx;
    point.y += dy;
  }
};

var ImageShape = function(imageSource, x, y, w, h) {
  var self = this;
  this.image = new Image();
  this.imageLoaded = false;
  this.points = [ new Point(x,y) ];
  this.x = x;
  this.y = y;
  this.image.src = imageSource;
  this.image.addEventListener('load', function (e) {
    self.setPolygonPoints();
    self.imageLoaded = true;
  }, false);
}

ImageShape.prototype = new Polygon();
ImageShape.prototype.fill = function (context) { };
ImageShape.prototype.setPolygonPoints = function() {
  this.points.push(new Point(this.x + this.image.width, this.y));
  this.points.push(new Point(this.x + this.image.width, this.y + this.image.height));
  this.points.push(new Point(this.x, this.y + this.image.height));
};

ImageShape.prototype.drawImage = function (context) {
  context.drawImage(this.image, this.points[0].x, this.points[0].y);
};
ImageShape.prototype.stroke = function (context) {
  var self = this;
  if (this.imageLoaded) {
    context.drawImage(this.image, this.points[0].x, this.points[0].y);
  }
  else {
    this.image.addEventListener('load', function (e) {
      self.drawImage(context);
    }, false);
  }
};
var SpriteShape = function (sprite, x, y) {
  this.sprite = sprite;
  this.x = x;
  this.y = y;
  sprite.left = x;
  sprite.top = y;
  this.setPolygonPoints();
};

SpriteShape.prototype = new Polygon();
SpriteShape.prototype.move = function (dx, dy) {
  var point, x;
  for(var i=0; i < this.points.length; ++i) {
    point = this.points[i];
    point.x += dx;
    point.y += dy;
  }
  this.sprite.left = this.points[0].x;
  this.sprite.top = this.points[0].y;
};
SpriteShape.prototype.fill = function (context) { };
SpriteShape.prototype.setPolygonPoints = function() {
  this.points.push(new Point(this.x, this.y));
  this.points.push(new Point(this.x + this.sprite.width, this.y));
  this.points.push(new Point(this.x + this.sprite.width, this.y + this.sprite.height));
  this.points.push(new Point(this.x, this.y + this.sprite.height));
};
SpriteShape.prototype.stroke = function (context) {
  this.sprite.paint(context);
};
```