**Computer Modelling 2**

**Practical 4a**

**Content:**

* Adding listeners (‘lect3’)
* Simple drawing application using listeners and the canvas drawing API (‘lect3’ and ‘lect5’)

**Question 1:**

Create two listeners – one for ‘mousedown’ and the other for ‘mouseup’ so that when:

* The user presses the mouse on the canvas, the message ‘mousedown’ should be output to the console.
* The user lifts the mouse off the canvas, the message ‘mouseup’ should be output to the console.

**Question 2:**

Write the JavaScript code to create a simple drawing application as follows:

The user presses the mouse anywhere on the canvas to start drawing. They then move the mouse over the canvas to continue drawing. When they wish to finish drawing they lift the mouse off the canvas.

**Details:**

* Add event listeners for ‘mousedown’ and ‘mouseup’.
* Define ‘onMouseMove’ – the handler that triggers the drawing when the mouse is moved over the canvas.

**Listening for ‘mousedown’:**

The mousedown event handler gets called whenever the user presses a mouse button on the canvas element. This is when the user wants to start drawing a line at the current mouse cursor position. Note that this handler can be defined inline. See ‘lect3’.

In the handler:

* A ***new path*** is added, and the initial position is set using the initial mouse coordinates.
* A listener for a mousemove event is added. Name the handler associated with this listener ‘onMouseMove’. This means that every time that the user moves the mouse over the canvas, the ‘onMouseMove’ handler is called. What does ‘onMouseMove’ do? (See below).

**OnMouseMove:**

Draws a line to the current mouse location.

**Listening for ‘mouseup’:**

This handler ***removes*** the mousemove handler so that nothing else is drawn to the canvas when the mouse is lifted. This handler can also be defined inline.

Note that you can add some simple interactive controls – for example to allow the user change the colour and width variables.

**Question 3:**

Create a simple drawing application using context.quadraticCurveTo (see note below). Use the mouse position for the ***control point***, and hard-code the beginning and ending points. Notice that the curve never reaches the control point, but gets about half-way there.

Note that we only need to add one listener to this script as the curve is drawn as soon as the mouse is moved over the canvas. The ‘pressing down’ and ‘lifting up’ mouse events are not relevant in this case. The mouse is either moving over the canvas or it is not.

**A note on drawing curves with quadraticCurveTo:**

The canvas drawing API provides a number of methods for drawing curves on a path. For example the method context.quadraticCurveTo connects two points with a curve using a ***control point***. The concept is the same as for context.LineTo, in that it begins its drawing at the point where the last drawing ended, or at the point where the last context.MoveTo command moved the pen to. It also uses line styles in the same way as context.LineTo. The only difference is in the shape of the line drawn.

context.quadraticCurveTo(cpx, cpy, x, y) takes two points as arguments – the first is a control point the affects the shape of the curve, and the second is the ending point of the curve. The shape is determined by a standard algorithm known as ***quadratic Bezier curve***. This function calculates a curve from one point to another, curving toward – but never touching – the control point. It’s more like the curve is attracted to it.

**A note on Drawing Paths:**

Remember that in order to draw a line using JavaScript, we use the context.lineTo() command that takes a single point; the ending point of the line. The canvas drawing API stores this information in a path.

The canvas context always has a current path, and there is only one current path. A path has 0 or more sub-paths, each consisting of a list of points connected by straight or curved lines. A closed path is one where its last point is connected to its first point by a straight line. To indicate that you’re starting a new drawing path, call context.beginPath(). Because a path is just a list of positions for a line, to render it to the canvas, you call need to context.Stroke()to make it visible.

You move the path’s position around on the canvas and draw lines to it. We indicate our starting point using context.moveTo().

For example the code below draws a line from the top left corner of the canvas to the point (100, 100):

context.beginPath();

context.moveTo(0, 0);

context.lineTo(100, 100);

context.stroke();

After you draw a line, the ending point for that line is the starting point for the next one. Or, you can use the method context.moveTo() to specify a new starting point for the next line.