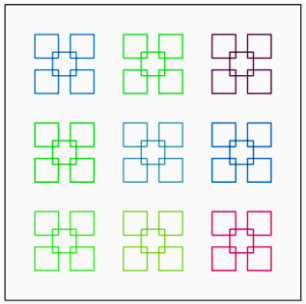
The University Of Sheffield.

COM1004: Web and Internet Technology

Lecture 15: Web graphics: Part 2



Dr. Steve Maddock
s.maddock@sheffield.ac.uk

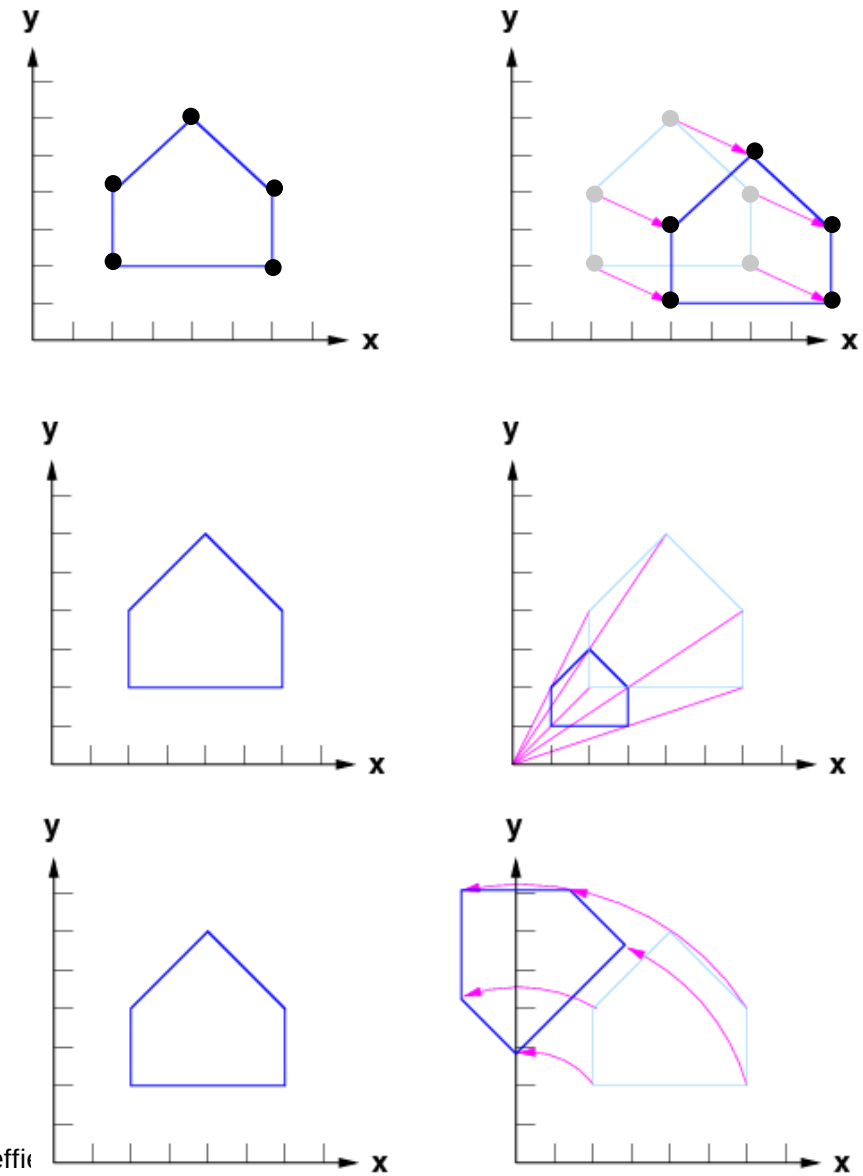
1. Introduction

- Last week we introduced the idea of raster graphics and looked at drawing on the canvas
  - Screen image is made of a rectangular grid of pixels – bitmap

- This week:
  - Transformations
  - Interaction

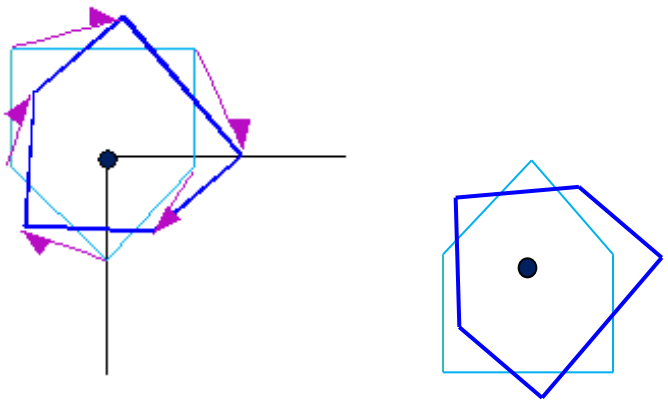
2. Two-dimensional transformations

- We'd like to be able to 'move' objects around space in a more structured way
- This can be achieved using affine transformations
  - Translation; Scaling; Rotation; Shear; Reflection;

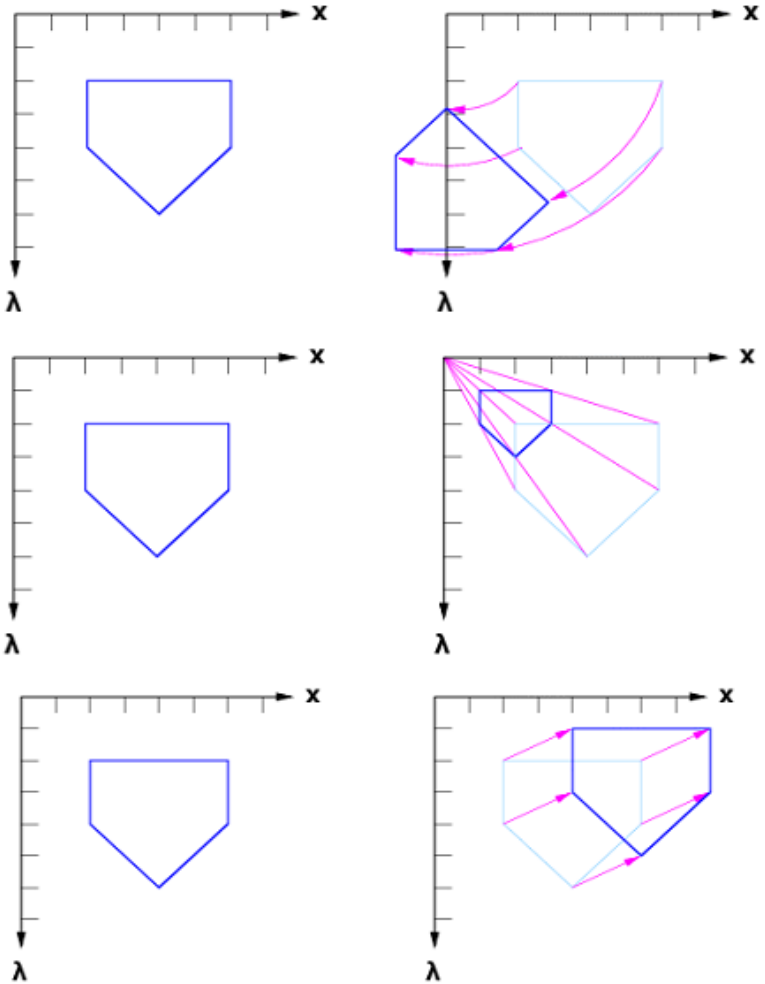


2. 2D transformations

- Rotation is around the origin of the coordinate system
  - The position (0,0) does not move



  - Rotation about an arbitrary point in space involves a combination of transformations
- Scaling is towards or away from the origin



Note: The canvas origin (0,0) is the top left corner

2.1 Rotation about an arbitrary point

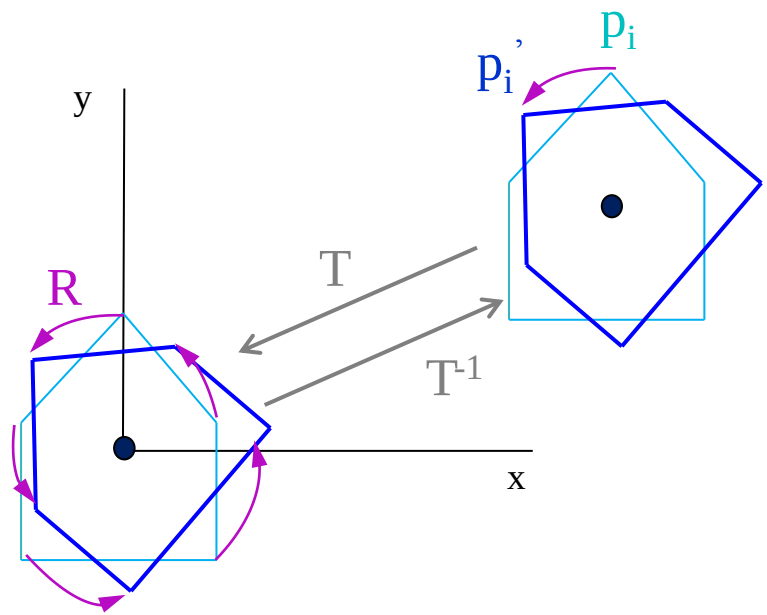
- *General plan*: Translate to world origin (T), rotate (R), and translate back again (T^{-1})

$$p_i' = T^{-1} R T \, p_i$$

- Combining these:

$$M = T^{-1} R T = T^{-1} (R T)$$

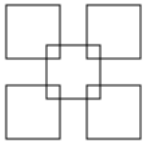
$$p_i' = M \, p_i$$



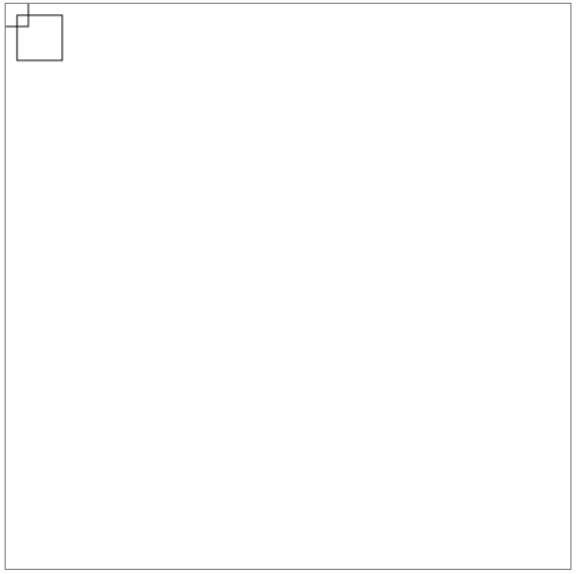
2.2 Example: drawing an object

```
function drawThing() {
  context.strokeStyle = "rgb(0,0,0)";
  context.lineWidth = "1";
  context.strokeRect(-50,-50,40,40);
  context.strokeRect(10,-50,40,40);
  context.strokeRect(-20,-20,40,40);
  context.strokeRect(-50,10,40,40);
  context.strokeRect(10,10,40,40);
}

function draw() {

  drawThing();

}
```

- What I expect:



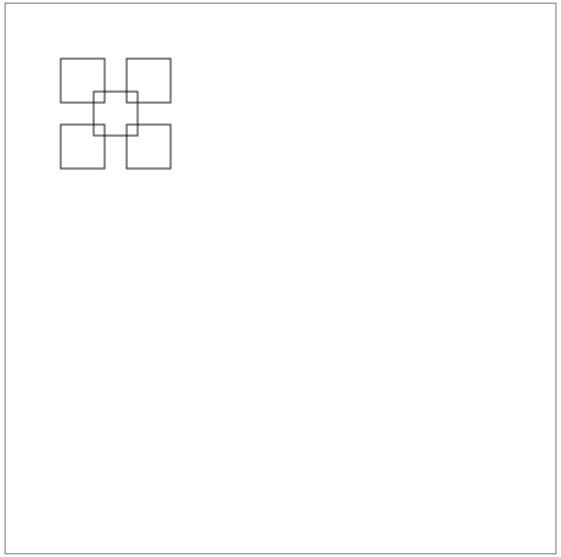
- What I get:



2.2 Example: drawing an object

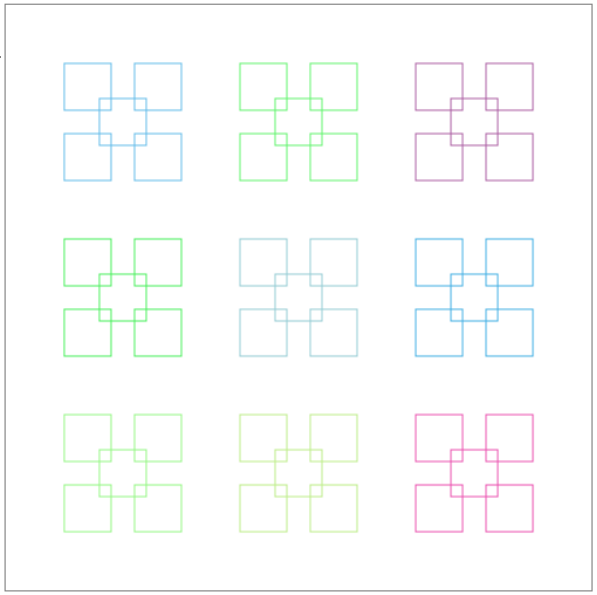
```
function drawThing() {
  context.strokeStyle = "rgb(0,0,0)";
  context.lineWidth = "1";
  context.strokeRect(-50,-50,40,40);
  context.strokeRect(10,-50,40,40);
  context.strokeRect(-20,-20,40,40);
  context.strokeRect(-50,10,40,40);
  context.strokeRect(10,10,40,40);
}

function draw() {

  context.translate(100, 100);
  drawThing();

}
```

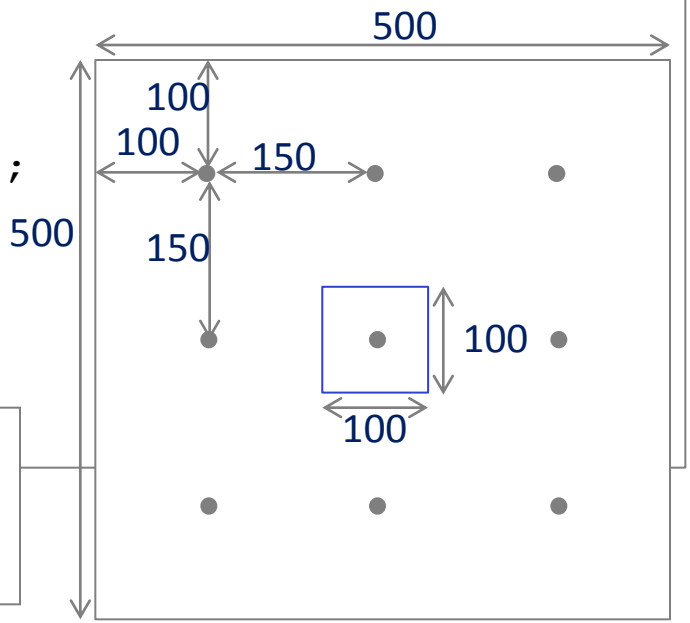
- Including a translation:



2.3 Example: drawing lots of objects

```
function drawThing() {
  context.strokeStyle = randomColour();
  context.lineWidth = "1";
  context.strokeRect(-50,-50,40,40);
  context.strokeRect(10,-50,40,40);
  context.strokeRect(-20,-20,40,40);
  context.strokeRect(-50,10,40,40);
  context.strokeRect(10,10,40,40);
}

function draw() {
  for (var i=0; i<3; i++) {
    for (var j=0; j<3; j++) {
      context.save();
      context.translate(100+j*150,100+i*150);
      drawThing();
      context.restore();
    }
  }
}
```

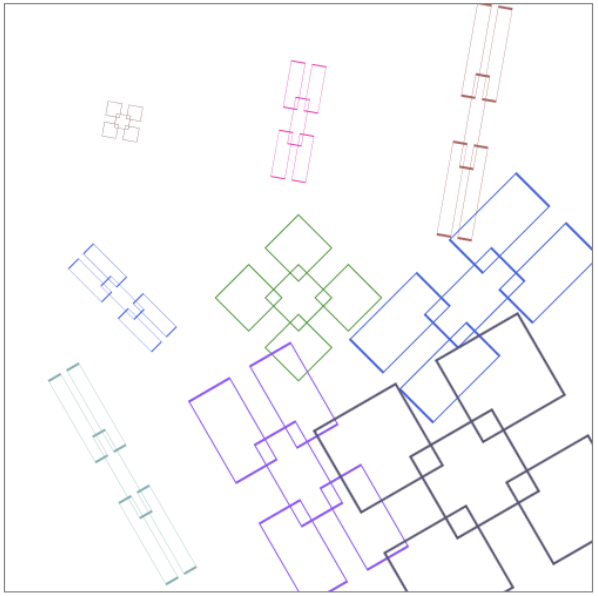


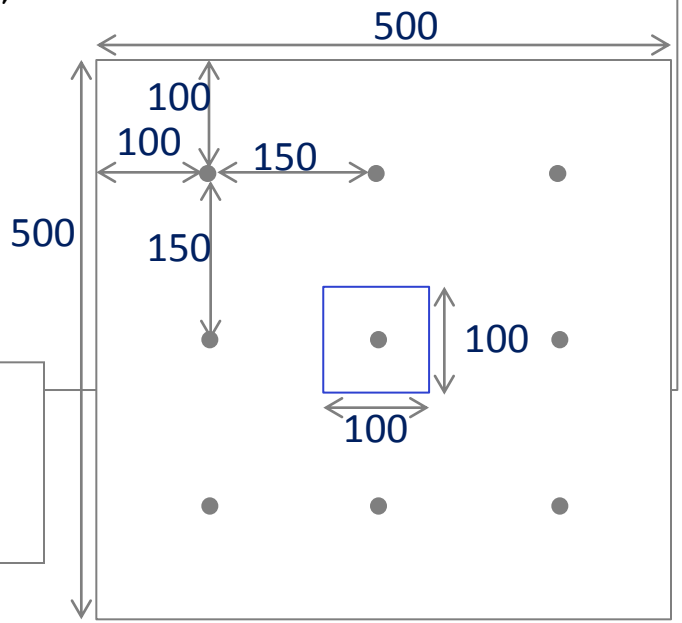


- Matrix transformations are cumulative in their effect, so need to use save and restore to control the accumulation

2.4 Example: multiple transformations

```
var sx = [0.3, 1, 2];
var sy = [0.3, 1, 2];
var r = [10*Math.PI/180,
         45*Math.PI/180,
         60*Math.PI/180];

function draw() {
  for (var i=0; i<3; i++) {
    for (var j=0; j<3; j++) {
      context.save();
      context.translate(100+j*150,100+i*150);
      context.rotate(r[i]);
      context.scale(sx[i],sy[j]);
      drawThing();
      context.restore();
    }
  }
}
```





- The transformations are applied in the reverse order to which they are specified since they are premultiplying the drawing object

2.5 The transformation matrix

- It is also possible to directly modify the transformation matrix

```
transform(m11, m12, m21, m22, dx, dy)
```

$$\begin{pmatrix} m11 & m21 & dx \\ m12 & m22 & dy \\ 0 & 0 & 1 \end{pmatrix}$$

- Q = MP
- Q = new position, P = old position
- Matrix M is premultiplying the position P

$$M = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

scale

$$M = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

rotate (anti-clockwise)

$$M = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

translate

3. Interaction on the canvas

We'll look at three examples:

- Using a button to control something on a canvas
  - Example event: click
  - We've handled input buttons previously when using forms
- Using mouse input on a canvas
  - Events: mousemove, mousedown, mouseup
- Using keyboard input on a canvas
  - Events: keydown, keypress, keyup

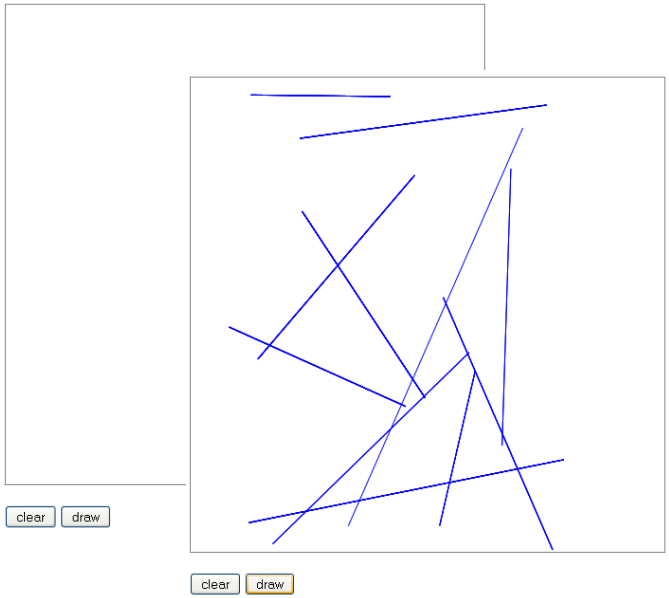
- We'll use jQuery to deal with cross-browser issues

- There are also lots of other events for HTML5 elements
  - E.g. mouse events: click, dbclick, drag, dragend, dragenter. dragleave, dragover, dragstart, drop, mouseout, mouseover, mousewheel, scroll

4. Responding to a button

- Create two buttons that are part of a form

- When the draw button is clicked/pressed, a random line is drawn on the canvas
- Need to add onclick event handlers to each button

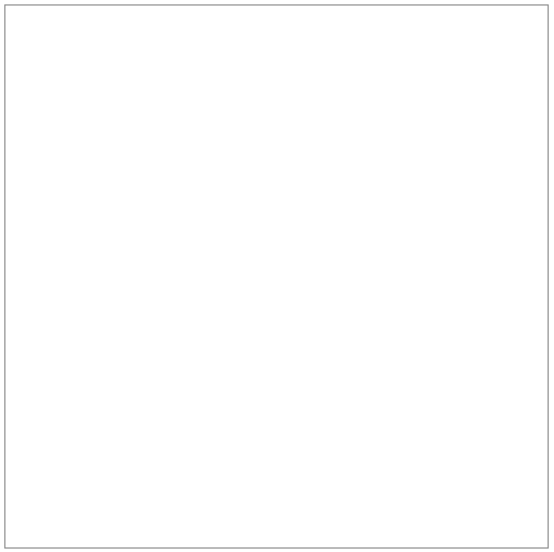



```
<canvas id="example" width="500" height="500"></canvas>

<p>
  <button name="clearbutton" id="clearbutton">clear</button>
  <button name="drawbutton" id="drawbutton">draw</button>
</p>
```


4. Responding to a button

```
function init() {
  document.getElementById("clearbutton").addEventListener(
                                          "click", clearCanvas);
  document.getElementById("drawbutton").addEventListener(
                                          "click", drawRandomLine);

  mycanvas = document.getElementById('example');
  if (!mycanvas || !mycanvas.getContext) return;
  context = mycanvas.getContext('2d');
  WIDTH = mycanvas.width;
  HEIGHT = mycanvas.height;
}
```

- Add the onclick event handler to the buttons
- When the button labelled draw is clicked the function **drawRandomLine** is called

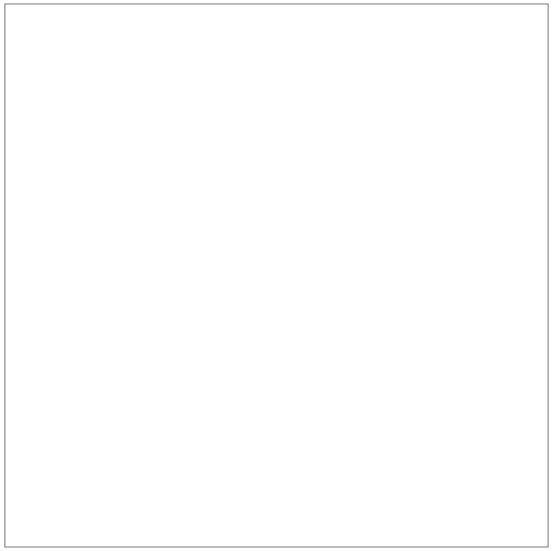


clear draw

4. Responding to a button

```
function init() {
  $('#clearbutton').click(clearCanvas);
  $('#drawbutton').click(drawRandomLine);
  context = $('#example')[0].getContext("2d");
  WIDTH = $('#example').width();
  HEIGHT = $('#example').height();
}
```

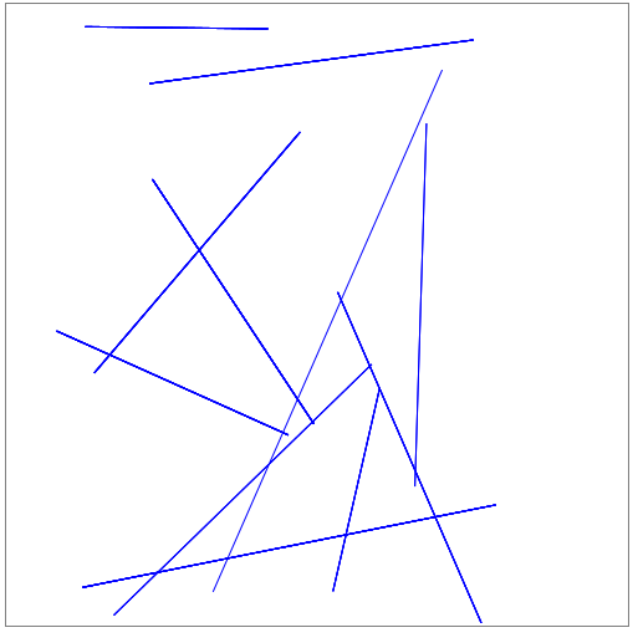
- Using jQuery's click method shortens the code and also deals with any cross-browser issues



clear draw

4. Responding to a button

```
function drawRandomLine() {
  context.strokeStyle = "rgb(0,0,255)";
  var x1 = Math.random()*WIDTH;
  var y1 = Math.random()*HEIGHT;
  var x2 = Math.random()*WIDTH;
  var y2 = Math.random()*HEIGHT;
  context.beginPath();
    context.moveTo(x1,y1);
    context.lineTo(x2,y2);
  context.closePath();
  context.stroke();
}
```

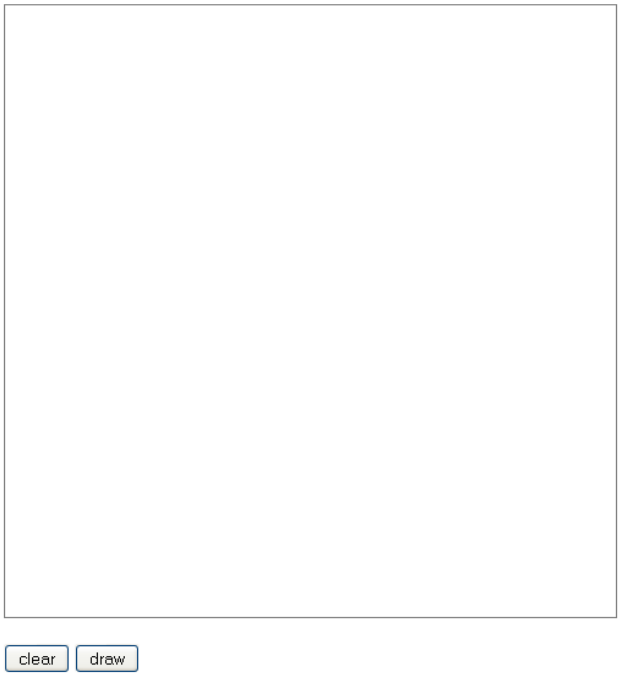


- When the button labelled draw is clicked the function **drawRandomLine** is called
- A blue line is drawn between two random positions on the canvas

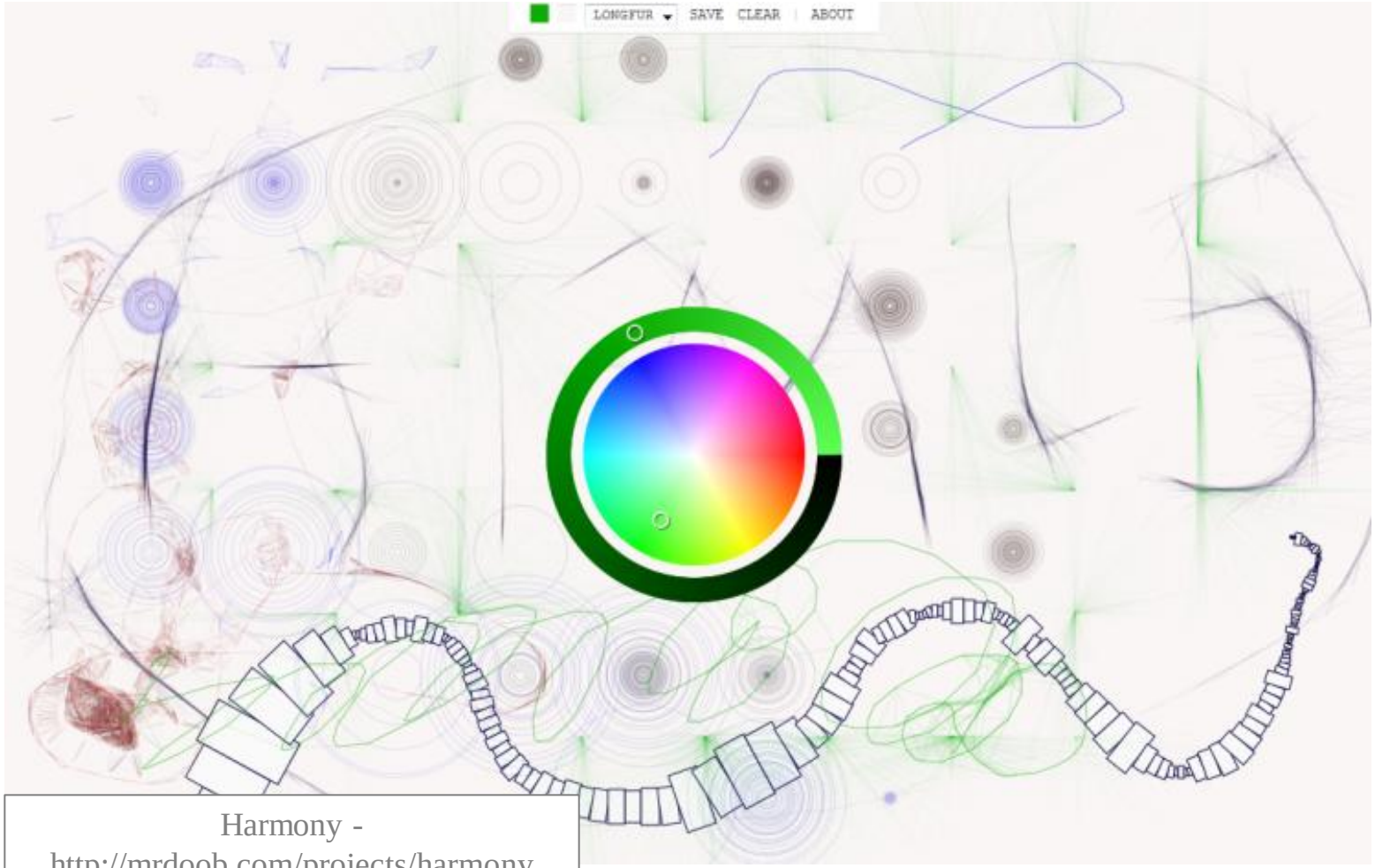
4. Responding to a button

```
function clearCanvas() {
  context.clearRect(0, 0, WIDTH, HEIGHT);
}
```

- When the button labelled clear is clicked the function **clearCanvas** called



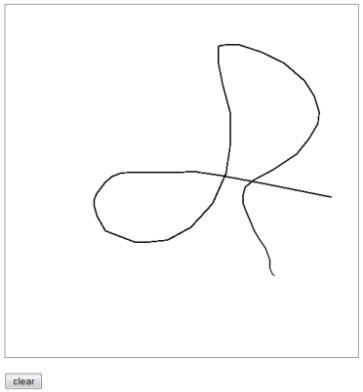
Example



Harmony - http://mrdoob.com/projects/harmony

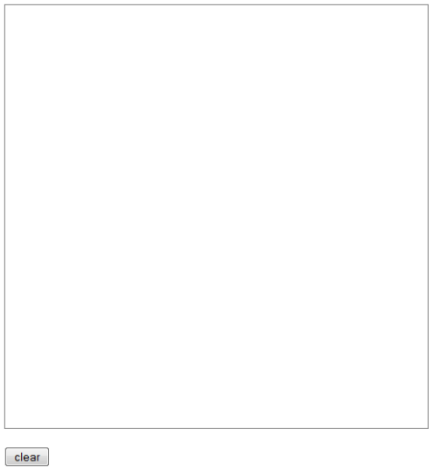
5. Mouse events

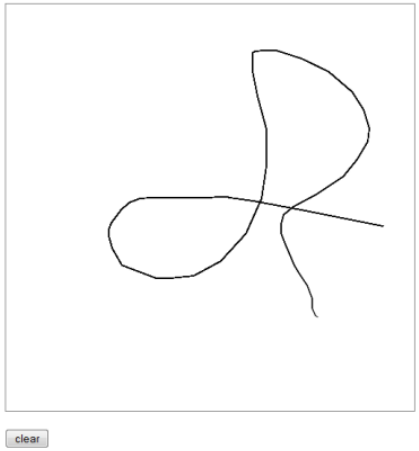
- Using mouse input on a canvas
  - Lots of mouse events: click, dbclick, drag, dragend, dragenter. dragleave, dragover, dragstart, drop, mousemove, mousedown, mouseup, mouseout, mouseover, mousewheel, scroll
- **The canvas will 'handle' the events** that occur
- We'll look at two examples:
  - Example 1: mousemove – a single scribble
  - Example 2: mousemove, mousedown, mouseup – multiple scribbles



5.1 Example 1: mousemove

- Create a canvas
- One button clear
  - Used to clear the canvas
  - An onclick event handler is added to the button, as in the earlier example
- Tell the canvas to responds to mouse events
  - As the mouse moves it leaves a trail behind it
  - Draw a line from the last mouse position to the current mouse position





5.1 Example 1: mousemove

```
<body>
  <canvas id="example" width="500" height="500"></canvas>
  <p>
    <button name="clearbutton" id="clearbutton">clear</button>
  </p>
  <script src="http://code.jquery.com/jquery-2.1.3.min.js">
  </script>
  <script src="./js/i_scribble1.js"></script>
</body>
```

- Create a canvas and a button

clear

5.1 Example 1: mousemove


```
function init() {
  context = $('#example')[0].getContext("2d");
  WIDTH = $('#example').width();
  HEIGHT = $('#example').height();
  canvasMinX = $('#example').offset().left;
  canvasMinY = $('#example').offset().top;
  // compensate for width and height of border
  canvasMinX += ( ($('#example').outerWidth()
                   - $('#example').width())/2 );
  canvasMinY += ( ($('#example').outerHeight()
                   - $('#example').height())/2 );

  $('#clearbutton').click(clearCanvas);
  $('#example').mousemove(onMouseMove);
  clearCanvas();
}
```

- Add the onclick event handler to the button using jQuery's **click** method
- the event handler clearCanvas will be called when the button is clicked

5.1 Example 1: mousemove

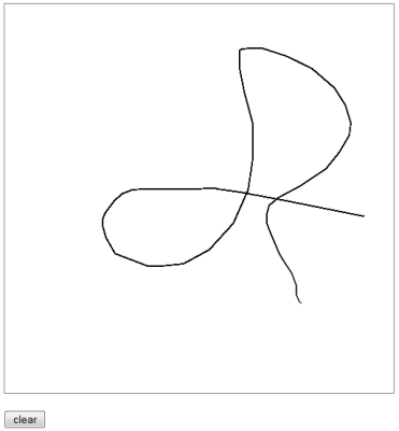

```
function init() {
  context = $('#example')[0].getContext("2d");
  WIDTH = $('#example').width();
  HEIGHT = $('#example').height();
  canvasMinX = $('#example').offset().left;
  canvasMinY = $('#example').offset().top;
  // compensate for width and height of border
  canvasMinX += ( ($('#example').outerWidth()
                   - $('#example').width())/2 );
  canvasMinY += ( ($('#example').outerHeight()
                   - $('#example').height())/2 );

  $('#clearbutton').click(clearCanvas);
  $('#example').mousemove(onMouseMove);
  clearCanvas();
}
```

- Add the onmousemove event handler to the canvas using jQuery's **mousemove** method
- the event handler onMouseMove will be called for every mouse movement on the canvas

5.1 Example 1: mousemove

```
var startedDrawing = false;
var oldx=0, oldy=0, cx=0, cy=0;

function onMouseMove(ev) {
  cx = ev.pageX - canvasMinX;
  cy = ev.pageY - canvasMinY;
  if (!startedDrawing) {
    oldx = cx;
    oldy = cy;
    startedDrawing = true;
  }
  else {
    context.beginPath();
      context.moveTo(oldx,oldy);
      context.lineTo(cx,cy);
    context.closePath();
    context.stroke();
    oldx = cx;
    oldy = cy;
  }
}
```

- IMPORTANT: System **automatically** delivers an **event** object as a parameter when the event handler is called



- The **event** object contains a set of properties such as the mouse position on the Web page when the event handler was called
- Need to subtract offset of top left of canvas area to get mouse position in canvas space – cross-browser issues resolved by jQuery

5.2 Example 2: mousemove, mousedown, mouseup

```
function init() {
  // set up context, WIDTH, HEIGHT, canvasMinX, canvasMinY
  $('#clearbutton').click(clearCanvas);
  $('#example').mousemove(onMouseMove);
  $('#example').mousedown(onMouseDown);
  $('#example').mouseup(onMouseUp);
  $('#example').css('cursor','pointer');
  clearCanvas();
}
```

clear

```
function onMouseMove(ev) {
  var p = getMouseXY(ev);
  if (startedDrawing) {
    cx = p[0];
    cy = p[1];
    drawLine(oldx,oldy,cx,cy);
    oldx = cx;
    oldy = cy;
  }
}
```

5.2 Example 2: mousemove, mousedown, mouseup

```
function onMouseDown(ev) {
  var p = getMouseXY(ev);
  cx = p[0];
  cy = p[1];
  oldx = cx;
  oldy = cy;
  startedDrawing = true;
}
```

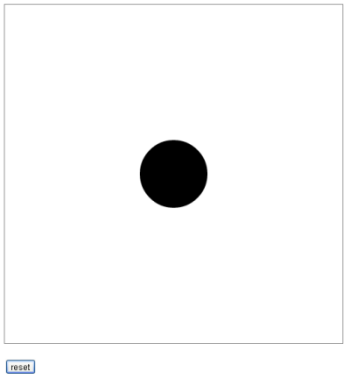
```
function onMouseUp(ev) {
  var p = getMouseXY(ev);
  if (startedDrawing) {
    cx = p[0];
    cy = p[1];
    drawLine(oldx,oldy,cx,cy);
    oldx = cx;
    oldy = cy;
    startedDrawing = false;
  }
}
```

clear

6. Keyboard events

- Three kinds of event
  - keydown event – any key is pressed; gives key code
  - keypress event – a key is pressed; triggers after keydown; gives char code
  - keyup event – generated when key is released
- Complications: http://unixpapa.com/js/key.html
- jquery.com: “The ***event*.which** property normalizes *event*.keyCode and *event*.charCode”
- Examples: *event*.which
  - arrow left on keyboard ← 37
  - arrow up on keyboard ↑ 38
  - arrow right on keyboard → 39
  - arrow down on keyboard ↓ 40

6.1 Example: Keyboard events

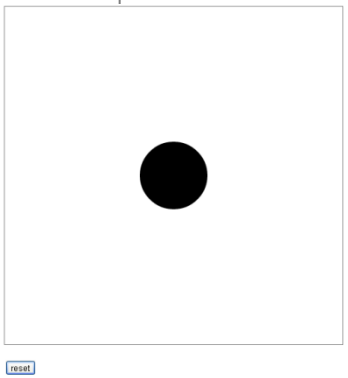


- Press arrow keys to move an object around the canvas
- Object starts in the middle of the canvas

```
<body>
  <canvas id="example" width="500" height="500"></canvas>
  <p>
    <button name="resetbutton" id="resetbutton">reset
    </button>
  </p>
  <script src="http://code.jquery.com/jquery-2.1.3.min.js">
  </script>
  <script src="./js/i_keyboard2.js"></script>
</body>
```

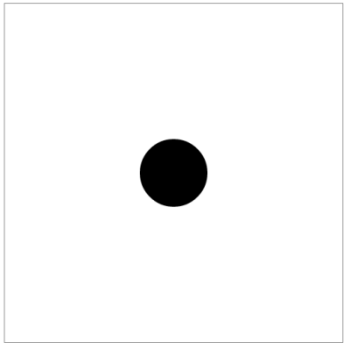

6.1 Example: Keyboard events

```
function init() {
  context = $('#example')[0].getContext("2d");
  WIDTH = $('#example').width();
  HEIGHT = $('#example').height();
  $('#resetbutton').click(resetCanvas);
  $(document).keydown(onKeyDown);
  resetCanvas();
}
```



- A keyboard event, e.g. **keydown**, has to handled by the **document** object
- It is not handled by the canvas

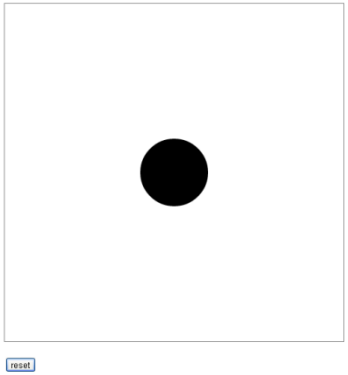
6.1 Example: Keyboard events




```
var currentX, currentY;
var dx = 2; dy = 2;

function resetCurrentPoint() {
  currentX = WIDTH/2;
  currentY = HEIGHT/2;
  context.moveTo(currentX, currentY);
}

function drawObject() {
  context.strokeStyle = "rgb(0,0,255)";
  context.beginPath();
  context.arc(currentX, currentY, RADIUS, 0, Math.PI*2, true);
  context.closePath();
  context.fill();
}
```

- The object is initially drawn at the centre of the canvas

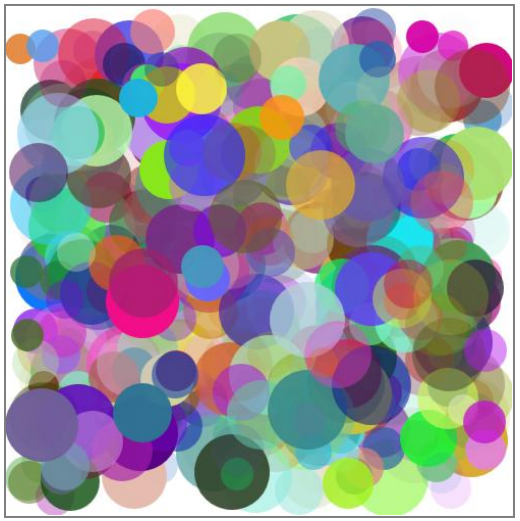
6.1 Example: Keyboard events




```
function onKeyDown(ev) {
  switch (ev.which) {
    case 37:  /* Left arrow was pressed */
      if (currentX - dx > 0) { currentX -= dx; }
      break;
    case 39:  /* Right arrow was pressed */
      if (currentX + dx < WIDTH){ currentX += dx; }
      break;
    case 38:  /* Up arrow was pressed */
      if (currentY - dy > 0) { currentY -= dy; }
      break;
    case 40:  /* Down arrow was pressed */
      if (currentY + dy < HEIGHT) { currentY += dy; }
      break;
  }
  clearCanvas();
  drawObject();
}
```

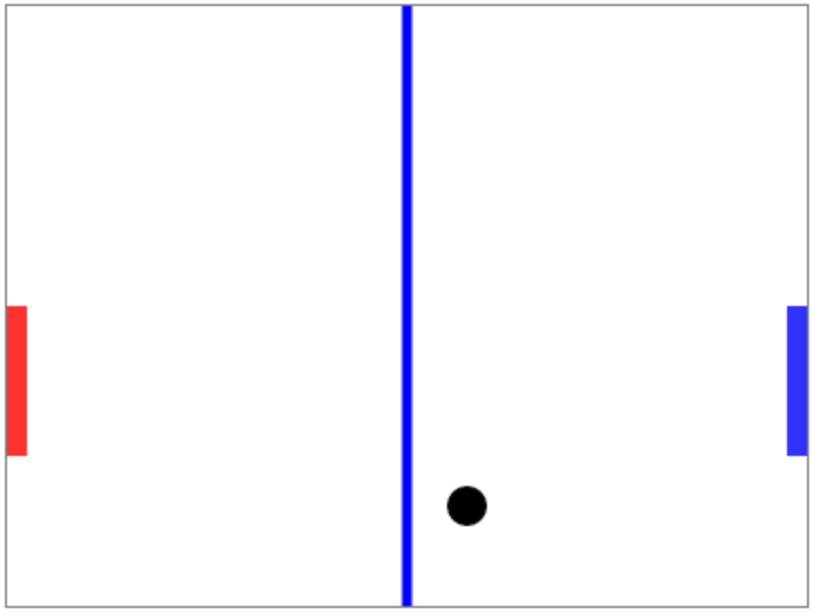
- dx and dx added to or subtracted from the current position when relevant key is pressed

Exercise sheet



Computer 0 : 0 Human

Who will win?

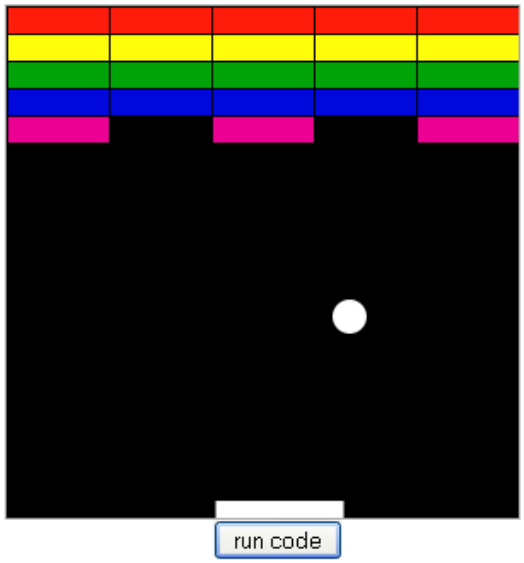


Message area

7. Further information

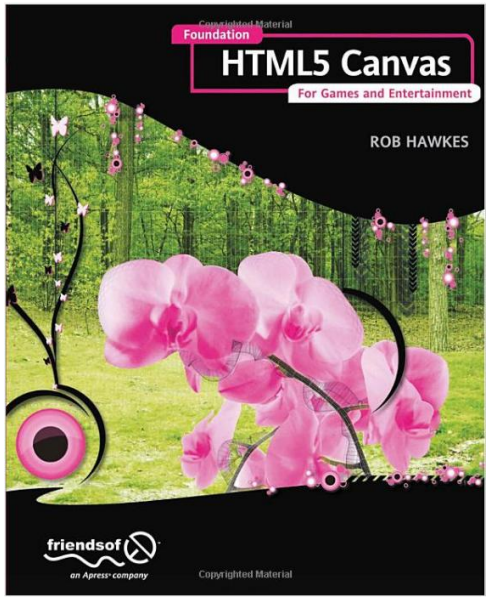
Canvas interaction tutorial:

- Bill Mill, Breakout clone
- http://billmill.org/static/canvastutorial/



Book:

- Rob Hawkes, “Foundation HTML5 Canvas: For Games and Entertainment”, Friends of ED (now part of Apress), 2011
- http://rawkes.com/foundationcanvas



8. Summary

- Transformations can be used to translate, scale and rotate objects
  - Involves matrix multiplication
- Interaction
  - Can use JavaScript, buttons, keyboard events and mouse events to make canvas interactive
  - jQuery is used to deal with cross-browser issues
- Further information:
  - Canvas “cheat sheet”: https://simon.html5.org/dump/html5-canvas-cheat-sheet.html
  - https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Canvas_tutorial
  - jquery.com
- *Next lecture*: SVG

Appendix A. Clicking an image

```
function init() {
  context = $('#example')[0].getContext("2d");
  WIDTH = $('#example').width();
  HEIGHT = $('#example').height();
  canvasMinX = $('#example').offset().left;
  canvasMinY = $('#example').offset().top;
  // compensate for width and height of border
  canvasMinX += ( ($('#example').outerWidth()
                  - $('#example').width())/2 );
  canvasMinY += ( ($('#example').outerHeight()
                  - $('#example').height())/2 );
  console.log("minX: "+canvasMinX+"; minY: "+canvasMinY);
  $('#example').click(checkClick);
  initImage();
}
```

- Set up the global variables to ensure we know size of canvas and top left offset coordinates

Appendix A. Clicking an image

```
var context = null;
var WIDTH, HEIGHT;
var canvasMinX = 0, canvasMinY = 0;
var img = new Image();
var xImg = 50, yImg = 50;

// main program body
init();

//functions
function initImage() {
  img.onload = function() {
    context.drawImage(img,xImg,yImg);
    console.log("image width, height: "+img.width+", "+img.height);
  }
  img.src = "images/image1.jpg";
}
```

- Load and display the image

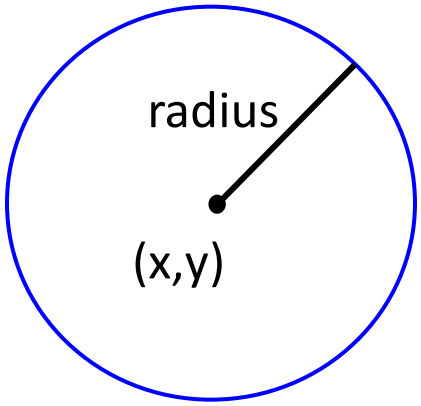
Appendix A. Clicking an image

```
function checkClick(ev) {
  var x = ev.pageX-canvasMinX;
  var y = ev.pageY-canvasMinY;
  console.log("click: "+x+", "+y);
  if ( (x>=xImg) && (x<(xImg+img.width))
      && (y>=yImg) && (y<(yImg+img.height)) ) {
    console.log("inside");
  }
  else {
    console.log("outside");
  }
}
```

- If user clicks on the canvas then state whether or not the image was clicked on
- Could then change the program so that something happens when the image is clicked on

Appendix B. 9. A ‘class’ for a Ball object

- We wish to create a picture containing lots of moving balls
- Start with an object that represents a ball, which will have an (x,y) position for its centre, a radius and a colour
- Properties: x, y, r, colour
- Methods: setX, getX,… draw,…

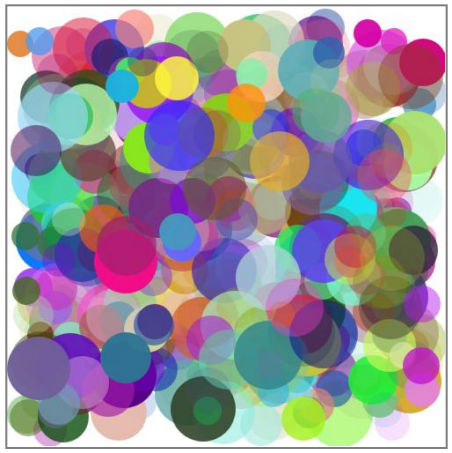


9. A ‘class’ for a Ball object

- This follows a similar pattern to the Rectangle object
- A constructor function is defined, followed by a series of prototype methods

```
function Ball(x,y,r,c) {
  this.x = x;
  this.y = y;
  this.r = r;
  this.c = c;
}

Ball.prototype.setX = function(x) {
  this.x = x;
}
```


10. A collection of Balls



- We need to maintain a list of balls.
- Solution: A BallList ‘class’
- Properties: numBalls, array of Balls
- Methods: getNumBalls, etc…

```
BallList.MAX_NUMBALLS = 500;

function BallList() {
  this.balls = new Array(BallList.MAX_NUMBALLS);
  this.numBalls = 0;
}

BallList.prototype.getNumBalls = function() {
  return this.numBalls;
}
```


10.1 Using the collection

- The following loop creates a collection of Balls

```
var balls = new BallList();

for (var i=0; i < NUM_BALLS; ++i) {
  var r = ???  // some radius
  var sx = ??? // some width
  var sy = ??? // some height
  var c = ???  // some random colour

  var ball = new Ball(sx,sy,r,c);
  balls.add(ball);
}
```


10.2 Adding and getting individual Balls

- We need methods to add a Ball to the list and to get a specific Ball

```
BallList.prototype.add = function(b) {
  if (this.numBalls >= BallList.MAX_NUMBALLS) {
    console.log("too many balls");
    return;
  }
  this.balls[this.numBalls] = b;
  this.numBalls++;
}

BallList.prototype.get = function(i) {
  if (i >= this.numBalls) return 0;
  return this.balls[i];
}
```

Basic error checking

Assume i is positive

10.3 Further properties and methods

- Properties: direction and speed of travel
  - Per frame of animation: x+=dx; y+=dy;
  - Rebound off walls
- Methods: draw a Ball, draw a list of Balls
- See next week's lab sheet