Edinburgh Napier University
School of Computing, Engineering, & the Built Environment (SCEBE)

Offer Holders Day Lab for Computing, Computer Science, Data Science, Games Development, & Software Engineering

---

Things to do in the browser when you're bored: *A tutorial introduction to using your web browser as a development environment for fun, 80s style, day-to-day hacking*

Dr Simon Wells

1 Aims

This lab session is meant to give you just a flavour of the kinds of things that we might do during a typical lab. However there are some caveats, namely that you've not arrived yet and we haven't had a chance to learn anything to underpin the practical work. So really we're just going to have some fun, write a bit of code, and build a (simple) game. Our core aim therefore is to build this:

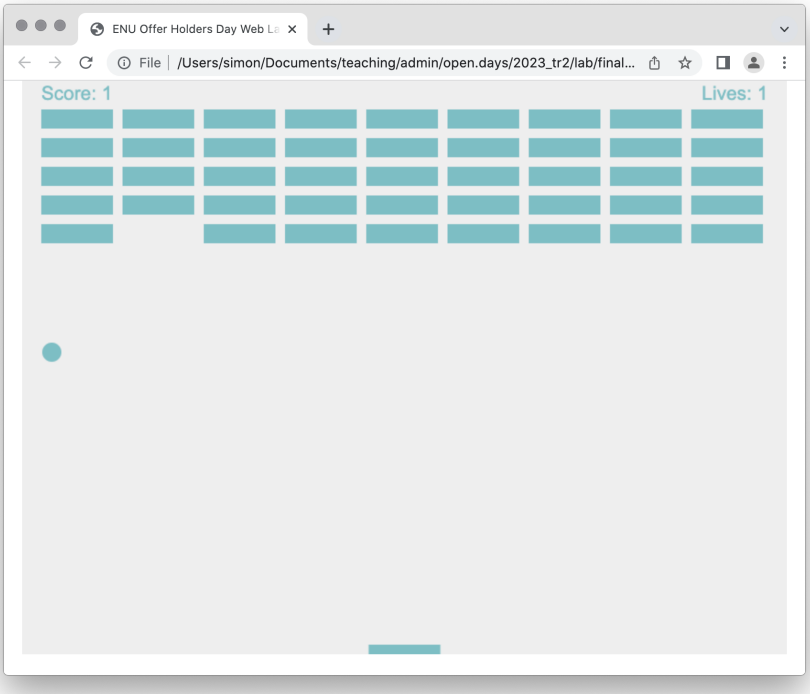

Figure 1: The venerable "breakout" game.

We're going to build this using some HTML, some CSS, and some Javascript, and the resulting game will run entirely within the Chrome web browser.

This will look like a lot of work right now, but the game itself is less than 200 lines of code. Most of the sub-activities of this lab involve breaking down that 200 lines into smaller logical sections and then studying what they do in isolation. A lot of the sub-activities also re-show the code over and over as it grows into the final project so much of the bulk of this lab sheet is actually repetition. It is this repetition that bulks out this document, but it helps to see the changes in context as we add new code to code we've written in previous sub-sections.

If you want to get the game working first, then feel free to skip to Section 2.6. You might also introduce typos along the way, which strictly speaking means bugs that need to be fixed. To help, the complete source code listing for the game, and each sub-section are available from this Github repository: `https://github.com/siwells/offer-holder-day`. Ultimately the aim of this lab is to give you a flavour of the kind of stuff that you can look forward to, as well as giving you a starting place for further exploration. Section 2.7 might inspire you with some suggestions for next steps. If you do want to do further explorartion then you'll likely find that digging into what you can do with HTML, CSS, and JS is good starting place as this lab sheet only scratches the surface.

2 Activities

The following sub-sections each build on each other to construct the complete breakout game.

2.1 Preliminaries

In order to build our game we're going to need three things.

- Access to the university system
- The Chrome Web Browser
- Notepad

First things first we need to log in to the university system. You should have a username:password combination that you can use to log in today. If you don't have one then please ask one of the members of staff and they'll help you out. When you've logged in you will find a standard Windows desktop.

We're going to test and run our game within the Chrome Web browser. So launch Chrome in the usual way.

We're going to need to write some code. Most code is written in an editor before it is executed or otherwise used. There are exceptions to this but for now let's go with it. Our editor for today will be Notepad, Window's built in plain text editor. There are many editors out there and you're encouraged, in the longer term, to find one that you like and which provides the kinds of features that you need in your career as a software developer. However Notepad is simple and available on any version of Windows going back to Windows 1.0 so it's a useful common denominator.

2.2 Skeleton

Start notepad and create a new file. Save this file as *game.html* using the "save as" option from the save menu. If you don't do this then your file will be saved as a text file instead of as an html file. Now type the following code into your game.html file and save it.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>ENU Offer Holders Day Web Lab</title>
6      <style>
7          * {
8              padding: 20px;
9              margin: 0;
10         }
11     </style>
12 </head>
13 <body>
14     Any text you put here will turn up on screen... That's how easy it is to create
           a web page. We can make headings using the h1 tags like so:
15
16     <h1>An example heading</h1>
17     <p>and create paragraphs like this using the p tags. Note that headings can
           range in size from h1 through to h6, getting progressively smaller. This
           gives you a way to organise the text content of your page into sections.</p
           >
18
19     <script>
20
21     </script>
22 </body>
23 </html>
```

This is just a simple web page. It doesn't yet have any Javascript. We'll add that soon in the section between the opening <script> and closing </script> tags in line 20. Notice that you can pick out a closing tag because it contain a forward slash "/". As a rule, most HTML tags come in pairs, the opening and closing tag for whichever HTML *keyword* you're dealing with.

Line 1 tells the browser that this file is an HTML file. Lines 2 and 23 are the opening and closing tags for the HTML document. An HTML document always has two main sections, the head section and the body seciotn. The opening and closing head tags are in lines 3 and 12, and the opening and closing body tags are in lines 13 and 22. Line 4 specifies how the contents of this HTML document are encoded. Line 5 is used to create the document's title, the thing that is displayed in the tab text when the file is opened in a Web browser. Lines 6 and 11 are the opening and closing tags for the style section, which is used to influence how the page is visually depicted. Within the style section, in lines 7 to 10, there is a small bit of CSS which is used to give us a slightly nice user interface once our page is loaded into the browser. In the body section we have some text and some example text-oriented HTML tags, which is just there to give us something to see when we load this into the browser. We'll delete lines 14 to 18 late and replace them with something else. This does show though how straightforward it is to create a web page. The Web was actually designed initially to be something that anyone could publish to with a minimal of programming skills. To a large degree this is actually still the case.

Open your game.html file in Chrome. You can do this by starting Chrome and using the File → Open File menu option, or by dragging your game.html file from the windows explorer onto your Chrome Window. You might even be able to double click you game.html file and have it loaded into Chrome. Note that how this works all depends on how your specific machine is set up so just see what works and if you get stuck, seek assistance from a demonstrator, member of staff, or even the person sitting next to you.

Your page should now look something like this:

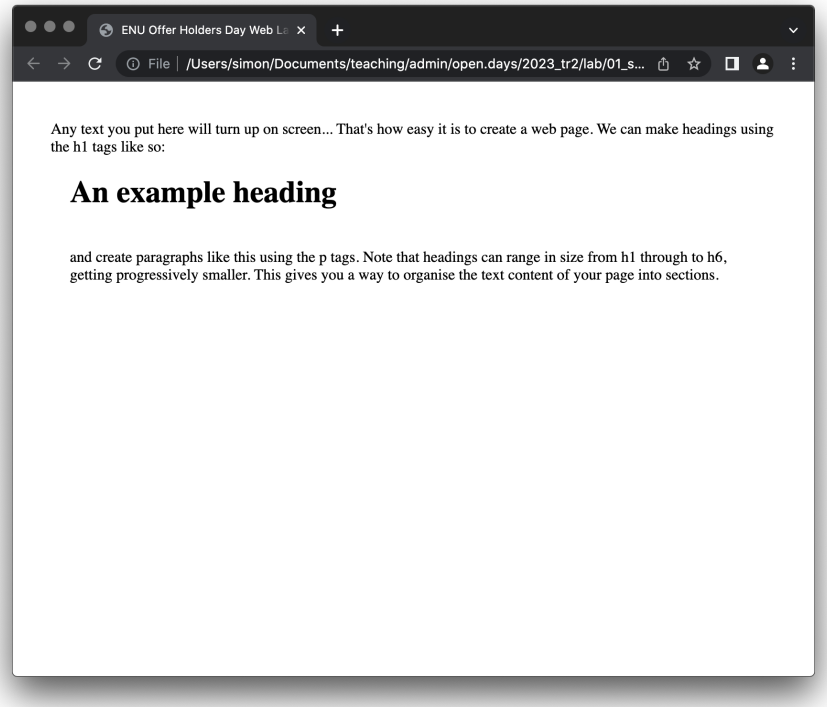

Figure 2: The basic skeleton for our game.

2.3 Drawing

The part of our game that faces the user, the interface, is the first thing we'll create. Take a look at the screenshot in Figure [**?**] and notice that it is built from four elements:

- The Ball
- The Paddle
- The Blocks

- The Score & Lives text

We'll create each, one at a time, as separate *functions*. That is, sections of code that we can execute and reuse as necessary in our game. This is because it turns out that most games, in fact most programmes, are actually built from pieces of code that are grouped together to encapsulate a specific function, and then these are run repeatedly. For example, in our game the ball will need to be redrawn very frequently to reflect it's moving around the screen and colliding with the walls, blocks, and paddle.

So let's draw the ball. Edit your game.html file as follows:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>ENU Offer Holders Day Web Lab</title>
6      <style>
7          * {
8              padding: 0;
9              margin: 0;
10         }
11         canvas {
12             padding: 0;
13             margin: auto;
14             background: #eee;
15             display: block;
16         }
17     </style>
18 </head>
19 <body>
20     <canvas id="game_canvas" width="800" height="600"></canvas>
21
22     <script>
23         var canvas = document.getElementById("game_canvas");
24         var ctx = canvas.getContext("2d");
25         var ball_radius = 10;
26         var x = canvas.width-Math.floor(Math.random()*750);
27         var y = canvas.height-(Math.floor(Math.random()*200)+50);
28         var colour = get_random_colour();
29
30
31         function get_random_colour() {
32             return '#'+Math.floor(Math.random()*16777215).toString(16);
33         }
34
35         function draw_ball() {
36             ctx.beginPath();
37             ctx.arc(x, y, ball_radius, 0, Math.PI*2);
38             ctx.fillStyle = colour;
39             ctx.fill();
40             ctx.closePath();
41         }
42         draw_ball();
43     </script>
44 </body>
45 </html>
```

Much of the code is the same as before except that we've deleted the original text between the <body> and <script> tags then made some additions. The additions are as follows:

Line 20 We created an HTML canvas element. A canvas is a place for drawing. We've basically allocated an area of the web page, 600 pixels high and 800 pixels wide, in which we will draw the elements of our breakout game.

Line 23 We created a JS variable to enable us to interact programmatically with the canvas drawing area on the webpage.

Line 24 We indicate to the browser that we want to draw in 2 dimensions ("2d") instead of 3D. Perhaps bear this in mind in case you someday want to build a 3D breakout game in the future?

Line 25 A variable for the radius of our ball

Line 26 A variable for the x coordinate of the location of our ball

Line 27 A variable for the y coordinate of the location of our ball

Line 28 A variable holding the colour value in which to draw the game. This actually calls a function that we've created in lines 31–33

We could have used fixed values for the x and y coodinates in lines 26 & 27 but that means that the game always starts with the ball in the same place. Instead we've used a maths function to return a bounded random value so that the starting location ofthe ball always lies somewhere between the paddle and the blocks.

We've also go two functions, one function to set a random colour, which will change everyt time the page is refreshed, and one function to draw the ball.

Lines 31–33 The get_random_colour() function. This just returns a hexadecimal number to the place that called it. The number itself is randomly generated from the space of hexadecimal encoded colours so that we can get a different valid colour code whenever we want.

Lines 35–41 The draw_ball() function includes some code to draw a ball onto the canvas, using the x & y coordinates we created earlier. The balls size is set by the radius we set in line 25, and the colour is the one that was created by our call to get_random_colour() in line 28. It's useful to think of drawing in JS as being like drawing with a pencil, we apply the pencil in a particular place to make a mark, and then raise it to stop making a mark. If we apply the pencil and move it then we get a line. A lot of drawing in code can be broken down into a sequence of applying the "pencil", making a mark, and then raising the "pencil" when we're done.

Line 42 Finally, having created our draw_ball() function, we need to "call" it to get the code to run.

Your page should now look something like this:

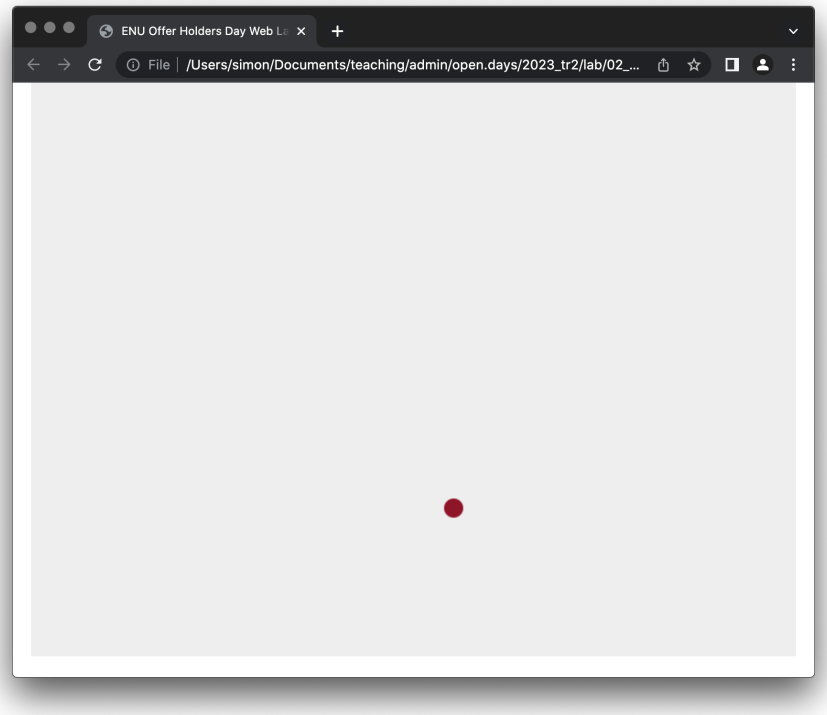

Figure 3: Drawing the ball.

Now we'll draw the paddle. This is just a rectangle that will appear towards the bottom of our campus. The full code listing is as follows which now also contains some code to draw the paddle.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>ENU Offer Holders Day Web Lab</title>
6      <style>
7          * {
8              padding: 0;
9              margin: 0;
10         }
11         canvas {
12             padding: 0;
13             margin: auto;
14             background: #eee;
15             display: block;
16         }
17     </style>
18 </head>
19 <body>
20     <canvas id="game_canvas" width="800" height="600"></canvas>
21
22     <script>
23         var canvas = document.getElementById("game_canvas");
24         var ctx = canvas.getContext("2d");
25         var ball_radius = 10;
26         var x = canvas.width-Math.floor(Math.random()*750);
27         var y = canvas.height-(Math.floor(Math.random()*200)+50);
28         var paddle_height = 10;
29         var paddle_width = 75;
30         var paddle_position = (canvas.width-paddle_width)/2;
31
32         var colour = get_random_colour();
33
34
35         function get_random_colour() {
36             return '#'+Math.floor(Math.random()*16777215).toString(16);
37         }
38
39          function draw_ball() {
40             ctx.beginPath();
41             ctx.arc(x, y, ball_radius, 0, Math.PI*2);
42             ctx.fillStyle = colour;
43             ctx.fill();
44             ctx.closePath();
45         }
46
47         function draw_paddle() {
48             ctx.beginPath();
49             ctx.rect(paddle_position, canvas.height-paddle_height, paddle_width,
                   paddle_height);
50             ctx.fillStyle = colour;
51             ctx.fill();
52             ctx.closePath();
53         }
54
55
56         draw_ball();
57         draw_paddle();
58     </script>
59 </body>
60 </html>
```

The main changes this time are some variables to store some parameters for our paddle, a function to draw the paddle, and a call to execute the paddle drawing function.

Lines 28–30 Create variables to store the height and width of the paddle along with the initial location in which to draw it.

Lines 47–53 Here we create the draw_paddle() function which will cause the paddle to be drawn onto the canvas when it is called. Compare the code in this function to what we have in the draw_ball() function. Both draw a path (remember that pencil metaphor from earlier).

However instead of drawing an arc, this time we're drawing a rectangle by specifing four things, the x coordinate of the upper-left corner of the rectangle, the y-coorindate of the upper-left corner of the rectangle, the width of the rectangle, the height of the rectangle in pixels and the width of the rectangle in pixels.

Line 57 And finally, we draw the paddle by calling the draw_paddle() function.

Open the breakout.html file in Chrome and see how it looks. Your page should now look something like this:

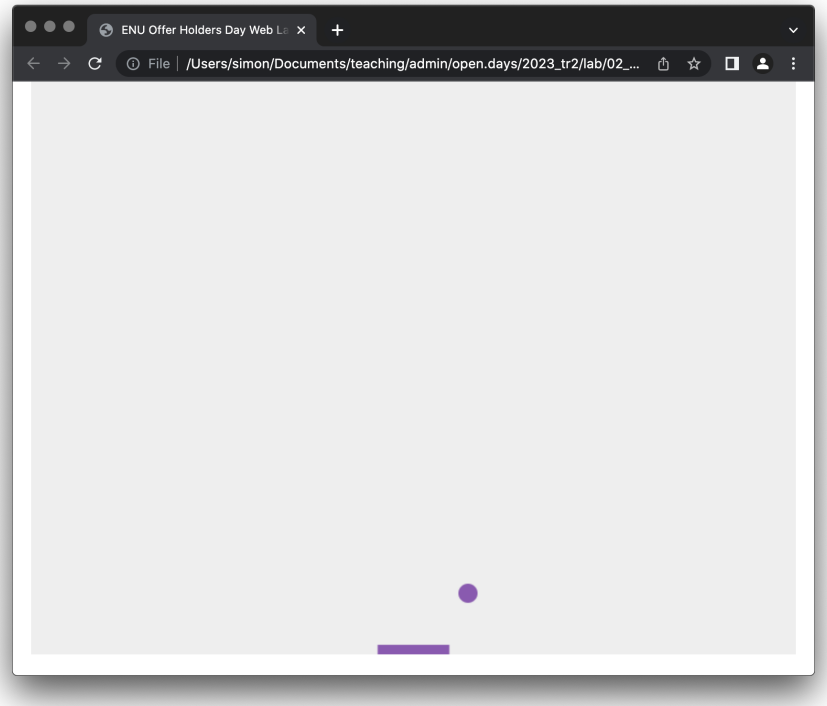

Figure 4: Drawing the paddle.

Hopefuly you should be noticing some patterns now in how we're building things. Having created the initial skeleton we're basically adding new variables and new functions, and then calling them.

Now let's draw the blocks:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>ENU Offer Holders Day Web Lab</title>
    <style>
        * {
            padding: 0;
            margin: 0;
        }
        canvas {
            padding: 0;
            margin: auto;
            background: #eee;
            display: block;
        }
   </style>
</head>
<body>
    <canvas id="game_canvas" width="800" height="600"></canvas>

    <script>
        var canvas = document.getElementById("game_canvas");
        var ctx = canvas.getContext("2d");
```

```
25          var ball_radius = 10;
26          var x = canvas.width-Math.floor(Math.random()*750);
27          var y = canvas.height-(Math.floor(Math.random()*200)+50);
28          var paddle_height = 10;
29          var paddle_width = 75;
30          var paddle_position = (canvas.width-paddle_width)/2;
31          var block_column_count = 9;
32          var block_row_count = 5;
33          var block_height = 20;
34          var block_width = 75;
35          var gutter = 10;
36          var padding = 30;
37          var top_margin = 20;
38          var inside_margin = 20;
39
40          var blocks = [];
41          var colour = get_random_colour();
42
43          function get_random_colour() {
44              return '#'+Math.floor(Math.random()*16777215).toString(16);
45          }
46
47          function build_wall() {
48              for(var c=0; c<block_row_count; c++) {
49                  blocks[c] = [];
50                  for(var r=0; r<block_column_count; r++) {
51                      blocks[c][r] = { x: 0, y: 0, status: 1 };
52                  }
53              }
54          }
55          build_wall();
56
57          function draw_ball() {
58              ctx.beginPath();
59              ctx.arc(x, y, ball_radius, 0, Math.PI*2);
60              ctx.fillStyle = colour;
61              ctx.fill();
62              ctx.closePath();
63          }
64
65          function draw_paddle() {
66              ctx.beginPath();
67              ctx.rect(paddle_position, canvas.height-paddle_height, paddle_width,
                    paddle_height);
68              ctx.fillStyle = colour;
69              ctx.fill();
70              ctx.closePath();
71          }
72
73          function draw_blocks() {
74              for(var c=0; c<block_row_count; c++) {
75                  for(var r=0; r<block_column_count; r++) {
76                      if(blocks[c][r].status == 1) {
77                          var blockX = (r*(block_width+gutter))+inside_margin;
78                          var blockY = (c*(block_height+gutter))+padding;
79                          blocks[c][r].x = blockX;
80                          blocks[c][r].y = blockY;
81                          ctx.beginPath();
82                          ctx.rect(blockX, blockY, block_width, block_height);
83                          ctx.fillStyle = colour;
84                          ctx.fill();
85                          ctx.closePath();
86                      }
87                  }
88              }
89          }
90
91          draw_blocks();
92          draw_ball();
93          draw_paddle();
94      </script>
95  </body>
96  </html>
```

We've followed a similar pattern this time, but in addition to our draw function, draw_blocks(), we've got an additional build_wall() function. This is mainly because the wall is a little more complex than just drawing a single block, as we did for the paddle. Also, when we play a game later, we want to track which blocks have been destroyed by the ball and which haven't. So we're going to create a place to store information about our blocks, then we're going to initialise the values for the blocks, then we're going to drawn them.

Lines 31–40 Here we create a number of variables to govern the dimensions of our blocks (blocl_height and block_width), how many blocks to create (block_row_count and block_column_count). the horizontal and vertical spacing between the blocks (gutter and padding), and the space between the wall and the borders of our canvas (top_margin and inside_margin). We also create a variable in line 40 that stores an array of blocks. This basically means that we want to store infomation about the entire collection of blocks that make up our wall.

Lines 47–54 This is our function to initalise the blocks array with new blocks. For each block we store it's location as a pair of x and y coordinates, and the status of the block.

Line 55 Having created our build_wall() function we now need to call it. We'll only need to do this once at the beginning of the game. As our game progresses the status of the various blocks will be altered to reflect that they've been hit and destroyed by the ball.

Lines 73–89 Our drawing function for the wall, draw_blocks(). This is very similar to our previous draw_paddle function but this time we need to draw many rectangles so we have to loop through the blocks array, drawing each block into it's location. Note that there is also a check on line 76 to determine the status of the block so that destroyed blocks aren't drawn as the game progresses.

Open the breakout.html file in Chrome and see how it looks. Your page should now look something like this:

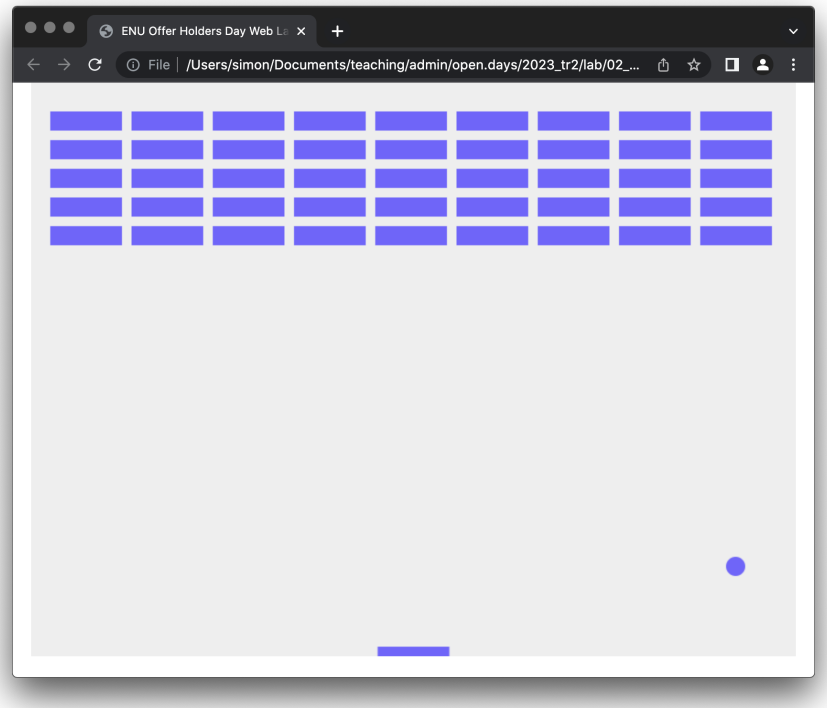

Figure 5: Drawing the blocks.

Finally, let's draw the text for the score and number of lives.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
```

```
5       <title>ENU Offer Holders Day Web Lab</title>
6       <style>
7           * {
8               padding: 0;
9               margin: 0;
10          }
11          canvas {
12              padding: 0;
13              margin: auto;
14              background: #eee;
15              display: block;
16          }
17      </style>
18  </head>
19  <body>
20      <canvas id="game_canvas" width="800" height="600"></canvas>
21
22      <script>
23          var canvas = document.getElementById("game_canvas");
24          var ctx = canvas.getContext("2d");
25          var ball_radius = 10;
26          var x = canvas.width-Math.floor(Math.random()*750);
27          var y = canvas.height-(Math.floor(Math.random()*200)+50);
28          var paddle_height = 10;
29          var paddle_width = 75;
30          var paddle_position = (canvas.width-paddle_width)/2;
31          var block_column_count = 9;
32          var block_row_count = 5;
33          var block_height = 20;
34          var block_width = 75;
35          var gutter = 10;
36          var padding = 30;
37          var top_margin = 20;
38          var inside_margin = 20;
39
40          var score = 0;
41          var lives = 1;
42          var blocks = [];
43          var colour = get_random_colour();
44
45          function get_random_colour() {
46              return '#'+Math.floor(Math.random()*16777215).toString(16);
47          }
48
49          function build_wall() {
50              for(var c=0; c<block_row_count; c++) {
51                  blocks[c] = [];
52                  for(var r=0; r<block_column_count; r++) {
53                      blocks[c][r] = { x: 0, y: 0, status: 1 };
54                  }
55              }
56          }
57          build_wall();
58
59          function draw_ball() {
60              ctx.beginPath();
61              ctx.arc(x, y, ball_radius, 0, Math.PI*2);
62              ctx.fillStyle = colour;
63              ctx.fill();
64              ctx.closePath();
65          }
66
67          function draw_paddle() {
68              ctx.beginPath();
69              ctx.rect(paddle_position, canvas.height-paddle_height, paddle_width,
                    paddle_height);
70              ctx.fillStyle = colour;
71              ctx.fill();
72              ctx.closePath();
73          }
74
75          function draw_blocks() {
76              for(var c=0; c<block_row_count; c++) {
77                  for(var r=0; r<block_column_count; r++) {
78                      if(blocks[c][r].status == 1) {
79                          var blockX = (r*(block_width+gutter))+inside_margin;
```

```
80                          var blockY = (c*(block_height+gutter))+padding;
81                          blocks[c][r].x = blockX;
82                          blocks[c][r].y = blockY;
83                          ctx.beginPath();
84                          ctx.rect(blockX, blockY, block_width, block_height);
85                          ctx.fillStyle = colour;
86                          ctx.fill();
87                          ctx.closePath();
88                      }
89                  }
90              }
91          }
92
93          function draw_score_text() {
94              ctx.font = "20px Arial";
95              ctx.fillStyle = colour;
96              txt  = "Score: "+score;
97              ctx.fillText(txt, inside_margin, top_margin);
98          }
99
100         function draw_lives_text() {
101             ctx.font = "20px Arial";
102             ctx.fillStyle = colour;
103             txt = "Lives: "+lives;
104             ctx.fillText(txt, canvas.width-(ctx.measureText(txt).width +
                    inside_margin), top_margin );
105         }
106
107         draw_blocks();
108         draw_ball();
109         draw_paddle();
110         draw_score_text();
111         draw_lives_text();
112     </script>
113 </body>
114 </html>
```

Similar to before, a couple of variables, a couple of functions, and a call to execute those functions.

Lines 40–41 Two variables to hold the players current score and the number of remaining lives.

Lines 93–98 A function to draw the text for the score, draw_score_text(). Note that this uses a different canvas drawing function, the filltext() function to draw text into our canvas in a specified location.

Lines 100–105 Similar to the last function this one draw_lives_text() is used to display the number of lives our player has. Becuase this text is right-aligned there is a slightly more complex calculation to determing where to start drawing the text as we have to calculate how big the text will be, using he measureText() function, and then adjust the starting place for drawing the text into the canvas as a result.

Lines 110-111 Actually call our functions to draw the score and number of lives.

And that's it. All the code we need for drawing the user interface for our game. The rest of the code is about getting these drawing functions to act like a game by moving things around, checking for collisions, and checking whether the game is finished. However we don't really have a game yet. All we have is something that draws how the game looks but we can't interact with it, and nothing moves.

Open the breakout.html file in Chrome and see how it looks. Your page should now look something like this:

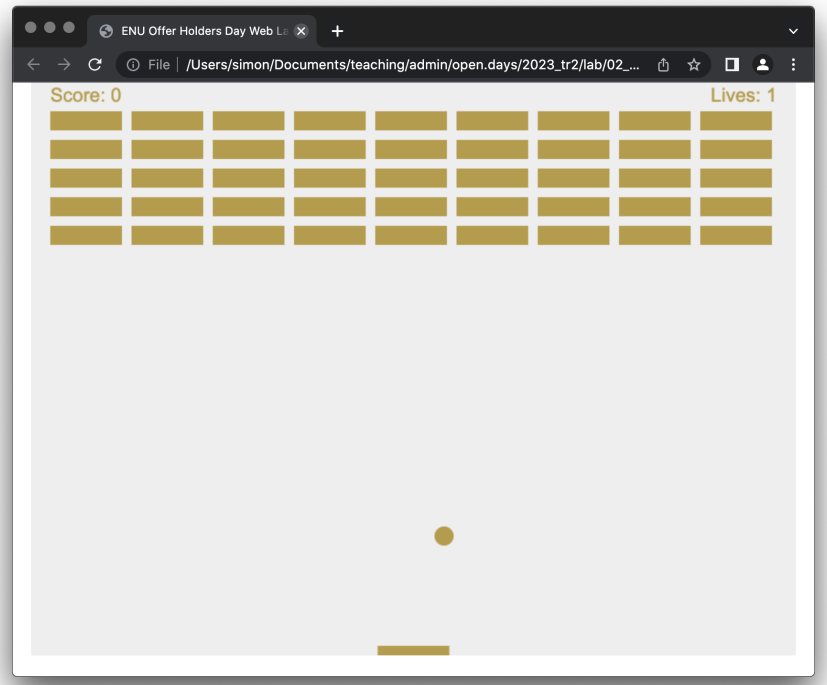

Figure 6: Drawing the text.

2.4 Gameloop

The game loop is something that happens in all games. It encapsulates the idea that we set up our game world, interact with the game world, update the game world as a result of the interaction, check for an end state, and then loop around to checking the interaction again. This loop continues until the game is finished.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>ENU Offer Holders Day Web Lab</title>
    <style>
        * {
            padding: 0;
            margin: 0;
        }
        canvas {
            padding: 0;
            margin: auto;
            background: #eee;
            display: block;
        }
   </style>
</head>
<body>
    <canvas id="game_canvas" width="800" height="600"></canvas>

    <script>
        var canvas = document.getElementById("game_canvas");
        var ctx = canvas.getContext("2d");
        var ball_radius = 10;
        var x = canvas.width-Math.floor(Math.random()*750);
        var y = canvas.height-(Math.floor(Math.random()*200)+50);
        var paddle_height = 10;
        var paddle_width = 75;
        var paddle_position = (canvas.width-paddle_width)/2;
        var block_column_count = 9;
        var block_row_count = 5;
        var block_height = 20;
        var block_width = 75;
```

```
35          var gutter = 10;
36          var padding = 30;
37          var top_margin = 20;
38          var inside_margin = 20;
39          var velocity = 4;
40          var dx = Math.random() < 0.5 ? velocity : -velocity;
41          var dy = -velocity;
42          var score = 0;
43          var lives = 1;
44          var blocks = [];
45          var colour = get_random_colour();
46
47          function get_random_colour() {
48              return '#'+Math.floor(Math.random()*16777215).toString(16);
49          }
50
51          function build_wall() {
52              for(var c=0; c<block_row_count; c++) {
53                  blocks[c] = [];
54                  for(var r=0; r<block_column_count; r++) {
55                      blocks[c][r] = { x: 0, y: 0, status: 1 };
56                  }
57              }
58          }
59          build_wall();
60
61          function draw_ball() {
62              ctx.beginPath();
63              ctx.arc(x, y, ball_radius, 0, Math.PI*2);
64              ctx.fillStyle = colour;
65              ctx.fill();
66              ctx.closePath();
67          }
68
69          function draw_paddle() {
70              ctx.beginPath();
71              ctx.rect(paddle_position, canvas.height-paddle_height, paddle_width,
                    paddle_height);
72              ctx.fillStyle = colour;
73              ctx.fill();
74              ctx.closePath();
75          }
76
77          function draw_blocks() {
78              for(var c=0; c<block_row_count; c++) {
79                  for(var r=0; r<block_column_count; r++) {
80                      if(blocks[c][r].status == 1) {
81                          var blockX = (r*(block_width+gutter))+inside_margin;
82                          var blockY = (c*(block_height+gutter))+padding;
83                          blocks[c][r].x = blockX;
84                          blocks[c][r].y = blockY;
85                          ctx.beginPath();
86                          ctx.rect(blockX, blockY, block_width, block_height);
87                          ctx.fillStyle = colour;
88                          ctx.fill();
89                          ctx.closePath();
90                      }
91                  }
92              }
93          }
94
95          function draw_score_text() {
96              ctx.font = "20px Arial";
97              ctx.fillStyle = colour;
98              txt  = "Score: "+score;
99              ctx.fillText(txt, inside_margin, top_margin);
100         }
101
102         function draw_lives_text() {
103             ctx.font = "20px Arial";
104             ctx.fillStyle = colour;
105             txt = "Lives: "+lives;
106             ctx.fillText(txt, canvas.width-(ctx.measureText(txt).width +
                    inside_margin), top_margin );
107         }
108
```

```
109         function game_loop() {
110             ctx.clearRect(0, 0, canvas.width, canvas.height);
111             draw_blocks();
112             draw_ball();
113             draw_paddle();
114             draw_score_text();
115             draw_lives_text();
116             requestAnimationFrame(game_loop);
117         }
118 
119         game_loop();
120     </script>
121 </body>
122 </html>
```

To turn our collection of drawing functions into a rudimentary game we only need to do one thing:

Lines 109–117 This is where we get our code to loop over itself, to draw the world, then move the ball, then redraw the world, *ad infinitum*. We've basically take all of our earierl calls to our draw functions and wrappe them in a function called game_loop(). At the end of the game loop we call our move() function (line 146) then in line 147 we call requestAnimationFrame(), a canvas function that causes our code in the game_loop function to be repeatedly called. At each cycle through the game loop we get a new frame of the game drawn to the screen. All of these frames give the illusion of continuous animation. This is the basis of all arcade style game interactions, basically draw the world to the screen sufficiently frequently, incorporating any updates or changes, so as to give the illusion of continuous change.

Line 119 Here we call the game loop function. Once we've called it the first time it will call itself again, from within itself which will cause the drawing and move functions to be called over and over again.

We still have a few things to do get our game complete but first, open the breakout.html file in Chrome and see how it looks. Your page should now look something like this (which isn't hugely different from how it looked before because the gameloop doesn't add anything visually impactful to out game. For this reason, we'll not bother with any more screenshots now until the final game):

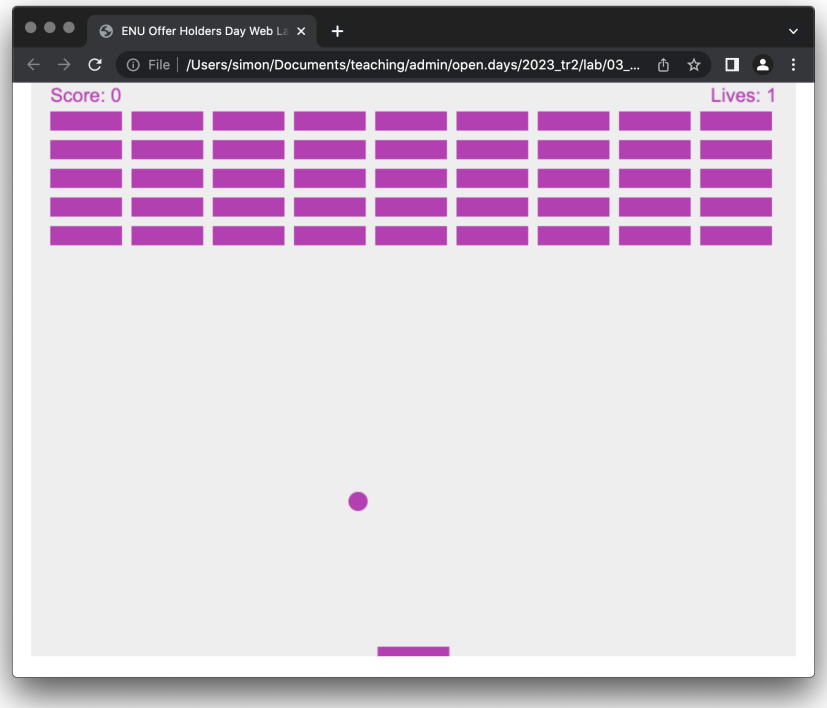

Figure 7: With a game loop to draw new frames as the "game world" changes.

2.5 Interaction

There are few forms of interaction within a game. One is the interaction of game objects between themselves, e.g. the ball interacting with the walls and bouncing off them. The other interaction is between the user and the game. We'll look at the ball's interaction first and then handle the user.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>ENU Offer Holders Day Web Lab</title>
    <style>
        * {
            padding: 0;
            margin: 0;
        }
        canvas {
            padding: 0;
            margin: auto;
            background: #eee;
            display: block;
        }
   </style>
</head>
<body>
    <canvas id="game_canvas" width="800" height="600"></canvas>

    <script>
        var canvas = document.getElementById("game_canvas");
        var ctx = canvas.getContext("2d");
        var ball_radius = 10;
        var x = canvas.width-Math.floor(Math.random()*750);
        var y = canvas.height-(Math.floor(Math.random()*200)+50);
        var paddle_height = 10;
        var paddle_width = 75;
        var paddle_position = (canvas.width-paddle_width)/2;
        var block_column_count = 9;
        var block_row_count = 5;
        var block_height = 20;
        var block_width = 75;
        var gutter = 10;
        var padding = 30;
        var top_margin = 20;
        var inside_margin = 20;
        var velocity = 4;
        var dx = Math.random() < 0.5 ? velocity : -velocity;
        var dy = -velocity;
        var score = 0;
        var lives = 1;
        var blocks = [];
        var colour = get_random_colour();

        function get_random_colour() {
            return '#'+Math.floor(Math.random()*16777215).toString(16);
        }

        function build_wall() {
            for(var c=0; c<block_row_count; c++) {
                blocks[c] = [];
                for(var r=0; r<block_column_count; r++) {
                    blocks[c][r] = { x: 0, y: 0, status: 1 };
                }
            }
        }
        build_wall();

        function move() {
            if(x + dx > canvas.width-ball_radius || x + dx < ball_radius) {
                dx = -dx;
            }
            if(y + dy <= ball_radius) {
                dy = -dy;
            }
            else if(y + dy > canvas.height-ball_radius) {
                if(x > paddle_position && x < paddle_position + paddle_width) {
```

```
                    dy = -dy;
                }
                else {
                    lives--;
                    if(!lives) {
                        alert("GAME OVER");
                        document.location.reload();
                    }
                    else {
                        x = canvas.width-Math.floor(Math.random()*750+1);
                        y = canvas.height-Math.floor(Math.random()*250+1);
                        dx = Math.random() < 0.5 ? velocity : -velocity;
                        dy = -velocity;
                        paddle_position = (canvas.width-paddle_width)/2;
                    }
                }
            }
            x += dx;
            y += dy;
        }

        function draw_ball() {
            ctx.beginPath();
            ctx.arc(x, y, ball_radius, 0, Math.PI*2);
            ctx.fillStyle = colour;
            ctx.fill();
            ctx.closePath();
        }

        function draw_paddle() {
            ctx.beginPath();
            ctx.rect(paddle_position, canvas.height-paddle_height, paddle_width,
                paddle_height);
            ctx.fillStyle = colour;
            ctx.fill();
            ctx.closePath();
        }

        function draw_blocks() {
            for(var c=0; c<block_row_count; c++) {
                for(var r=0; r<block_column_count; r++) {
                    if(blocks[c][r].status == 1) {
                        var blockX = (r*(block_width+gutter))+inside_margin;
                        var blockY = (c*(block_height+gutter))+padding;
                        blocks[c][r].x = blockX;
                        blocks[c][r].y = blockY;
                        ctx.beginPath();
                        ctx.rect(blockX, blockY, block_width, block_height);
                        ctx.fillStyle = colour;
                        ctx.fill();
                        ctx.closePath();
                    }
                }
            }
        }

        function draw_score_text() {
            ctx.font = "20px Arial";
            ctx.fillStyle = colour;
            txt  = "Score: "+score;
            ctx.fillText(txt, inside_margin, top_margin);
        }

        function draw_lives_text() {
            ctx.font = "20px Arial";
            ctx.fillStyle = colour;
            txt = "Lives: "+lives;
            ctx.fillText(txt, canvas.width-(ctx.measureText(txt).width +
                inside_margin), top_margin );
        }


        draw_blocks();
        draw_ball();
        draw_paddle();
        draw_score_text();
```

```
144         draw_lives_text();
145
146         function game_loop() {
147             ctx.clearRect(0, 0, canvas.width, canvas.height);
148             draw_blocks();
149             draw_ball();
150             draw_paddle();
151             draw_score_text();
152             draw_lives_text();
153             move();
154             requestAnimationFrame(game_loop);
155         }
156
157         game_loop();
158     </script>
159 </body>
160 </html>
```

Lines 61–89 This is the move() function. It will move the ball each time we want to redraw the screen. In lines 62–64 it checks the location of the ball in relation to the left and right edges of the canvas and alters the ball location as a result if the ball hasn't gone out of bounds. In lines 65–67 we do a similar thing for the top edge of the canvas. In both cases, if the ball has reached the edge then we need to reflect the ball back in the opposite direction as if it were bouncing off the side or top walls. Line 68 checks whether the ball has reached the lower bound of the canvas. Lines 69–71 check whether the ball is colliding with the paddle and bounce it off if so. If the ball has reached the lower bound of the canvas but not hit the paddle then the player loses a life. This is handled by lines 72–86. Lines 73–77 deal with the case where the paddle misses the ball and the player has no more lives. In which case we use the alert() function to tell the user that the game is over. When the alert is cleared by clicking the OK button then the page is reloaded to give us a fresh game. Lines 78–84 however handle the case where the player lost a life but still has remaining lives, in which case we restart by "serving" the ball towards the wall again. Finally lines 87–88 handle the actual movement of the ball by updating the x and y coordinates once all those previous checks and updates have been completed. This is one of our more complex functions and has a number of moving parts but is integral to the game as we have to check each different context in which the ball can interact with the game's "world".

Now let's look at the user interaction. which is basically moving the paddle around to try to bounce the ball back towards the wall.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>ENU Offer Holders Day Web Lab</title>
6      <style>
7          * {
8              padding: 0;
9              margin: 0;
10         }
11         canvas {
12             padding: 0;
13             margin: auto;
14             background: #eee;
15             display: block;
16         }
17     </style>
18 </head>
19 <body>
20     <canvas id="game_canvas" width="800" height="600"></canvas>
21
22     <script>
23         var canvas = document.getElementById("game_canvas");
24         var ctx = canvas.getContext("2d");
25         var ball_radius = 10;
26         var x = canvas.width-Math.floor(Math.random()*750);
27         var y = canvas.height-(Math.floor(Math.random()*200)+50);
28         var paddle_height = 10;
```

```
29          var paddle_width = 75;
30          var paddle_position = (canvas.width-paddle_width)/2;
31          var block_column_count = 9;
32          var block_row_count = 5;
33          var block_height = 20;
34          var block_width = 75;
35          var gutter = 10;
36          var padding = 30;
37          var top_margin = 20;
38          var inside_margin = 20;
39          var velocity = 4;
40          var dx = Math.random() < 0.5 ? velocity : -velocity;
41          var dy = -velocity;
42          var score = 0;
43          var lives = 1;
44          var blocks = [];
45          var colour = get_random_colour();
46
47          document.addEventListener("mousemove", mouse_move_handler, false);
48
49          function get_random_colour() {
50              return '#'+Math.floor(Math.random()*16777215).toString(16);
51          }
52
53          function build_wall() {
54              for(var c=0; c<block_row_count; c++) {
55                  blocks[c] = [];
56                  for(var r=0; r<block_column_count; r++) {
57                      blocks[c][r] = { x: 0, y: 0, status: 1 };
58                  }
59              }
60          }
61          build_wall();
62
63          function mouse_move_handler(e) {
64              var relative_x = e.clientX - canvas.offsetLeft;
65              if(relative_x > 0 && relative_x < canvas.width) {
66                  paddle_position = relative_x - paddle_width/2;
67              }
68          }
69
70          function move() {
71              if(x + dx > canvas.width-ball_radius || x + dx < ball_radius) {
72                  dx = -dx;
73              }
74              if(y + dy <= ball_radius) {
75                  dy = -dy;
76              }
77              else if(y + dy > canvas.height-ball_radius) {
78                  if(x > paddle_position && x < paddle_position + paddle_width) {
79                      dy = -dy;
80                  }
81                  else {
82                      lives--;
83                      if(!lives) {
84                          alert("GAME OVER");
85                          document.location.reload();
86                      }
87                      else {
88                          x = canvas.width-Math.floor(Math.random()*750+1);
89                          y = canvas.height-Math.floor(Math.random()*250+1);
90                          dx = Math.random() < 0.5 ? velocity : -velocity;
91                          dy = -velocity;
92                          paddle_position = (canvas.width-paddle_width)/2;
93                      }
94                  }
95              }
96              x += dx;
97              y += dy;
98          }
99
100         function draw_ball() {
101             ctx.beginPath();
102             ctx.arc(x, y, ball_radius, 0, Math.PI*2);
103             ctx.fillStyle = colour;
104             ctx.fill();
```

```
105             ctx.closePath();
106         }
107
108         function draw_paddle() {
109             ctx.beginPath();
110             ctx.rect(paddle_position, canvas.height-paddle_height, paddle_width,
                    paddle_height);
111             ctx.fillStyle = colour;
112             ctx.fill();
113             ctx.closePath();
114         }
115
116         function draw_blocks() {
117             for(var c=0; c<block_row_count; c++) {
118                 for(var r=0; r<block_column_count; r++) {
119                     if(blocks[c][r].status == 1) {
120                         var blockX = (r*(block_width+gutter))+inside_margin;
121                         var blockY = (c*(block_height+gutter))+padding;
122                         blocks[c][r].x = blockX;
123                         blocks[c][r].y = blockY;
124                         ctx.beginPath();
125                         ctx.rect(blockX, blockY, block_width, block_height);
126                         ctx.fillStyle = colour;
127                         ctx.fill();
128                         ctx.closePath();
129                     }
130                 }
131             }
132         }
133
134         function draw_score_text() {
135             ctx.font = "20px Arial";
136             ctx.fillStyle = colour;
137             txt  = "Score: "+score;
138             ctx.fillText(txt, inside_margin, top_margin);
139         }
140
141         function draw_lives_text() {
142             ctx.font = "20px Arial";
143             ctx.fillStyle = colour;
144             txt = "Lives: "+lives;
145             ctx.fillText(txt, canvas.width-(ctx.measureText(txt).width +
                    inside_margin), top_margin );
146         }
147
148         function game_loop() {
149             ctx.clearRect(0, 0, canvas.width, canvas.height);
150             draw_blocks();
151             draw_ball();
152             draw_paddle();
153             draw_score_text();
154             draw_lives_text();
155             move();
156             requestAnimationFrame(game_loop);
157         }
158
159         game_loop();
160     </script>
161 </body>
162 </html>
```

Line 47 Here we add an "event handler" to our HTML document. User interactions are events in which something has happened as a result of the user performing some action. In this case, our event handler is designed to fire off events when the user moves the mouse. When a mouse move event is detected then the handler automatically calls our mouse_move_handler() function which we'll look at next.

Lines 63–68 This is our mouse_move_handler() function which is passed some variables when it is called. These variables store coordinates associated with the mouse move and we can use those to update the position of the paddle. First, line 64 we convert the incoming coordinates which refer to the browser window into coordinates associated with our canvas. We then check

that the mouse coordinates are within the bounds of the canvas (line 65) and if they are, we update the position of the paddle to reflect whether the mouse has moved left or right. In essence, we track the horizontal movement of the mouse pointer and move the paddle left or right as a result.

Our mouse moving the paddle, in combination with the move() function from earlier means that we can now interact with our game. So we can bounce the ball off the paddle, back towards the wall. Of course we haven't yet added code to handle interactions between the ball and the blocks that make up the wall yet, but that'll be our next, and final, task.

2.6 Collisions

In this final sub-section, we'll add the final function needed for our complete game. So let's get on with it. As before, here's the full code listing (which is actually the full and final code listing for the lab):

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>ENU Offer Holders Day Web Lab</title>
6      <style>
7          * {
8              padding: 0;
9              margin: 0;
10         }
11         canvas {
12             padding: 0;
13             margin: auto;
14             background: #eee;
15             display: block;
16         }
17     </style>
18 </head>
19 <body>
20     <canvas id="game_canvas" width="800" height="600"></canvas>
21
22     <script>
23         var canvas = document.getElementById("game_canvas");
24         var ctx = canvas.getContext("2d");
25         var ball_radius = 10;
26         var x = canvas.width-Math.floor(Math.random()*750);
27         var y = canvas.height-(Math.floor(Math.random()*200)+50);
28         var paddle_height = 10;
29         var paddle_width = 75;
30         var paddle_position = (canvas.width-paddle_width)/2;
31         var block_column_count = 9;
32         var block_row_count = 5;
33         var block_height = 20;
34         var block_width = 75;
35         var gutter = 10;
36         var padding = 30;
37         var top_margin = 20;
38         var inside_margin = 20;
39         var velocity = 4;
40         var dx = Math.random() < 0.5 ? velocity : -velocity;
41         var dy = -velocity;
42         var score = 0;
43         var lives = 1;
44         var blocks = [];
45         var colour = get_random_colour();
46
47         document.addEventListener("mousemove", mouse_move_handler, false);
48
49         function get_random_colour() {
50             return '#'+Math.floor(Math.random()*16777215).toString(16);
51         }
52
53         function build_wall() {
54             for(var c=0; c<block_row_count; c++) {
55                 blocks[c] = [];
56                 for(var r=0; r<block_column_count; r++) {
57                     blocks[c][r] = { x: 0, y: 0, status: 1 };
```

```
                }
            }
        }
        build_wall();

        function mouse_move_handler(e) {
            var relative_x = e.clientX - canvas.offsetLeft;
            if(relative_x > 0 && relative_x < canvas.width) {
                paddle_position = relative_x - paddle_width/2;
            }
        }

        function collision_detection() {
            for(var c=0; c<block_row_count; c++) {
                for(var r=0; r<block_column_count; r++) {
                    var block = blocks[c][r];
                    if(block.status == 1) {
                        if(x > block.x && x < block.x+block_width && y > block.y &&
                             y < block.y+block_height) {
                            dy = -dy;
                            block.status = 0;
                            score++;
                            if(score == block_column_count*block_row_count) {
                                alert("WINNER! WINNER!");
                                document.location.reload();
                            }
                        }
                    }
                }
            }
        }

        function move() {
            if(x + dx > canvas.width-ball_radius || x + dx < ball_radius) {
                dx = -dx;
            }
            if(y + dy <= ball_radius) {
                dy = -dy;
            }
            else if(y + dy > canvas.height-ball_radius) {
                if(x > paddle_position && x < paddle_position + paddle_width) {
                    dy = -dy;
                }
                else {
                    lives--;
                    if(!lives) {
                        alert("GAME OVER");
                        document.location.reload();
                    }
                    else {
                        x = canvas.width-Math.floor(Math.random()*750+1);
                        y = canvas.height-Math.floor(Math.random()*250+1);
                        dx = Math.random() < 0.5 ? velocity : -velocity;
                        dy = -velocity;
                        paddle_position = (canvas.width-paddle_width)/2;
                    }
                }
            }
            x += dx;
            y += dy;
        }

        function draw_ball() {
            ctx.beginPath();
            ctx.arc(x, y, ball_radius, 0, Math.PI*2);
            ctx.fillStyle = colour;
            ctx.fill();
            ctx.closePath();
        }

        function draw_paddle() {
            ctx.beginPath();
            ctx.rect(paddle_position, canvas.height-paddle_height, paddle_width,
                paddle_height);
            ctx.fillStyle = colour;
            ctx.fill();
```

```
132             ctx.closePath();
133         }
134
135         function draw_blocks() {
136             for(var c=0; c<block_row_count; c++) {
137                 for(var r=0; r<block_column_count; r++) {
138                     if(blocks[c][r].status == 1) {
139                         var blockX = (r*(block_width+gutter))+inside_margin;
140                         var blockY = (c*(block_height+gutter))+padding;
141                         blocks[c][r].x = blockX;
142                         blocks[c][r].y = blockY;
143                         ctx.beginPath();
144                         ctx.rect(blockX, blockY, block_width, block_height);
145                         ctx.fillStyle = colour;
146                         ctx.fill();
147                         ctx.closePath();
148                     }
149                 }
150             }
151         }
152
153         function draw_score_text() {
154             ctx.font = "20px Arial";
155             ctx.fillStyle = colour;
156             txt  = "Score: "+score;
157             ctx.fillText(txt, inside_margin, top_margin);
158         }
159
160         function draw_lives_text() {
161             ctx.font = "20px Arial";
162             ctx.fillStyle = colour;
163             txt = "Lives: "+lives;
164             ctx.fillText(txt, canvas.width-(ctx.measureText(txt).width +
                    inside_margin), top_margin );
165         }
166
167         function game_loop() {
168             ctx.clearRect(0, 0, canvas.width, canvas.height);
169             draw_blocks();
170             draw_ball();
171             draw_paddle();
172             draw_score_text();
173             draw_lives_text();
174             collision_detection();
175             move();
176             requestAnimationFrame(game_loop);
177         }
178
179         game_loop();
180     </script>
181 </body>
182 </html>
```

The main thing to note this time is the addition of the collision_detection() function

Lines 70–87 This is our collision_detection() function. All we're doing in this function is looping through our blocks array, using a pair of nested *for* loops to visit every block in the wall. For each block we check whether it has already been hit by the ball and if so then we ignore it. If a block hasn't yet been hit then we check to see whether the ball is going to hit it using the coordinates of the ball and the coordinates of the block. If the ball will hit the block then we change the status of the block so that it is removed from the wall in the next iteration of the game loop, then we reverse the direction of the ball as though it bounced from the block when it destroyed it. Finally we do a quick check to determine if the block that was hit was the final block and if it was then the game is won because all of the blocks have been destroyed. At this point, as for losing the game in the move() function, we use an alert to tell the player that they won and then reload the document to start a new game once the player clears the alert.

Lines 174 Finally, we call collision_detection() from within the game loop.

If you open your breakout.html file in Chrome it should look something similar to this:

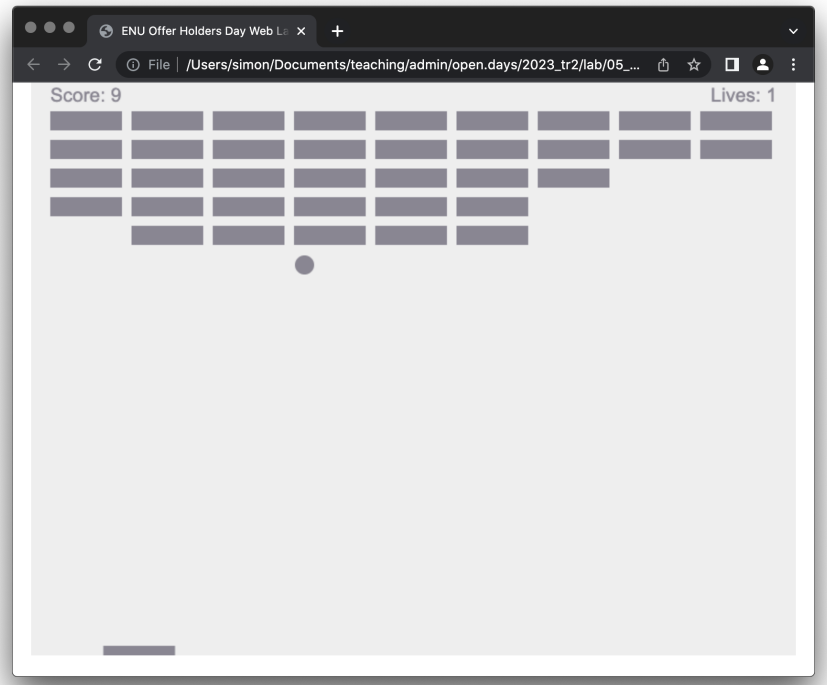

Figure 8: The final game, with collisions removing blocks from the wall.

And that's it folks. You now have a simple, playable, browser-hosted Javascript game in fewer than 200 lines of code. Obviously there's way more to learn about coding in HTML, CSS, & Javascript than we've covered here but I'm hoping it's whetted your apetite for more.

2.7 Challenge

I'm interested to see where you can take this in the time between today and when you arrive for your first trimester. Here are some ideas:

- Work out how to alter the number of lives available.
- Work out how to alter the size of the ball.
- Work out how to alter the size of the paddle.
- Work out how to alter the speed at which the ball moves.
- What would you need to do to have more than one "level"? What would other levels include? Perhaps a faster ball? Or a different pattern of blocks? What if the blocks needed more than one strike to destroy them?
- What about special blocks that give power ups? Perhaps multiple balls for a short duration? Or a superball that blasts through the blocks until it rebounds off a wall? Or a larger paddle for a given duration?
- Visually you could alter things as well. What about colouring the blocks and other aspects of the game in different ways? If you are just using random colours this could be visually cacophonous so consider how you might ensure that the colour palette is cohesive.
- The physics model that we have for things like the ball rebounding is very rudimentary. It might be nice if, for example, the angel of rebound of the ball changed when hit by a rapidly moving paddle instead of a static paddle, a bit like how a tennis ball responds differently depending upon how it is struck.
- Also consider things like sound effects. Even a single sound that plays at certain events such as losing a ball, destroying a block, or rebounding from the wall can greatly change the users experience.

- The classic breakout game isn't too far away from other classic games like Space Invaders. If the blocks could move, and perhaps fired back, and the paddle fired multiple balls one after the other, which didn't rebound, then you'd have most of a space invaders game. Perhaps some of our breakout code would work to achieve that?

Feel free to contact me to let me know how you get on and what you manage to achieve...