Tutorial – Introduction to Game Programming

In this first tutorial, we're going to start of slowly so you get a chance to really understand the fundamentals of building a game with JavaScript and HTML5.

In this tutorial we're going to make a web page that contains the canvas element that we want to draw to. We'll also create a script file where we'll write our JavaScript game. And will link the two together so that we're able to draw to the canvas from our program.

After we get some simple drawing happening we'll take a look at how to debug your program using the Chrome browser. Knowing how to debug your programs is essential so that you can find any errors or mistakes you've made when writing your game.

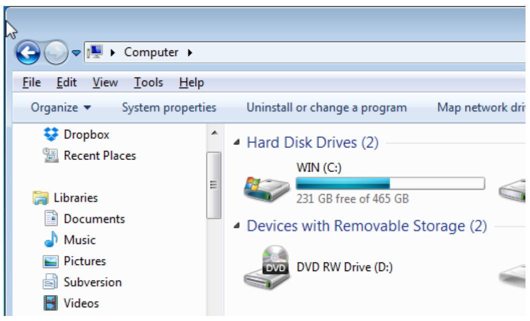
Creating the Web Page:

The first thing we need for our JavaScript game is the web page. Web pages are written in a 'language' called HyperTextMarkupLanguage, or HTML for short. So from here on out I'll be referring to the web page as the HTML page. (Luckily HTML is also the file extension for the file, so it shouldn't get confusing).

1. Create a new folder on your computer for you to store your work. I recommend that you keep each week's tutorial in a separate folder, so you can refer back to it easily if needed. Each week's tutorial will be based on the tutorial from the week before, so make sure you're able to find your work easily.
2. In your new folder create a text file and give it the name 'index.html'. You may get a warning about changing the file extension, that's OK.

   If you're using a Mac then everything should be fine and you can ignore the rest of this step.

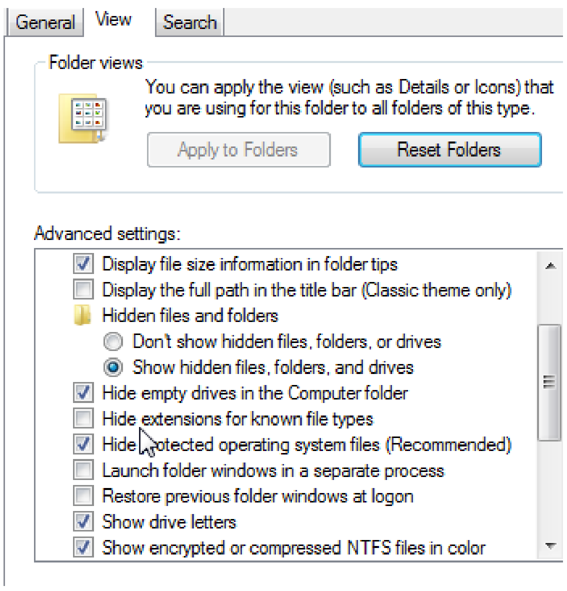
   However, if you're using a Window's machine you may not be able to tell what the file extension is because it could be hidden. To unhide it, follow these steps:
   1. In Windows Explorer, press the 'alt' key so that the menu appears



Press the 'ALT' key in Windows Explorer to display the menu

   2. Select Tools -> Folder Options
   3. Select the View tab

4. Make sure the 'Hide extensions for known file types' option is NOT selected



You can apply the view (such as Details or Icons) that you are using for this folder to all folders of this type.

Advanced settings:

- [✓] Display file size information in folder tips
- [ ] Display the full path in the title bar (Classic theme only)
- Hidden files and folders
  - ○ Don't show hidden files, folders, or drives
  - ● Show hidden files, folders, and drives
- [✓] Hide empty drives in the Computer folder
- [ ] Hide extensions for known file types
- [✓] Hide protected operating system files (Recommended)
- [ ] Launch folder windows in a separate process
- [ ] Restore previous folder windows at logon
- [✓] Show drive letters
- [✓] Show encrypted or compressed NTFS files in color

Make sure 'Hide extensions' is NOT selected

3. Now that your file has the '.html' extension, the file icon should change. It may look different depending on which web browser you've set as your default

index.html

4. Open index.html your text editor of choice and enter the following code:

```
<html>
<body>
<canvas id="gameCanvas" width="640" height="480">
        This text is displayed if your browser does not support HTML5.
</canvas>
<script src="main.js"></script>
</body>
</html>
```

That's the HTML page done! We'll be using the same code for the HTML page until we start doing the second assignment. So from now on the only place you'll write code is inside main.js (which we'll get to in a second.

Let's briefly go over what's happening in this HTML file (the lecture slides have some more information too).

Everything between the <body> tags will be drawn to the page when you open it in your browser.

In this page we're creating a canvas that is 640 pixels wide by 480 pixels high. This is where we'll be drawing our game.

If you ever open this page and you see the text 'This text is displayed if your browser does not support HTML5', then … you guessed it, your browser doesn't support HTML5. You will need to update your web browser (and possibly your operating system)

Finally the <script> tag loads the file main.js and tells the web page that this file contains a script program. This file will contain the code for our game.

Once we start assignment two we'll have more <script> tags that load more files, but for now main.js is all we're going to need.

Creating the JavaScriptfile:

1. In the same folder as your index.html file, create a new text file with the name 'main.js'

   The .js file extension is important because it lets us know we're making a JavaScript code file.

2. Get ready for awesome.

The first thing we need to do is get the canvas from the HTML page. This is done through the document object.

Once we have the canvas object, we can get a 2D drawing context.

Enter the following code inside main.js:

```
// Get the 2D context from the canvas in
// our HTML page
var canvas = document.getElementById("gameCanvas");
var context = canvas.getContext("2d");
```

Get the 2D drawing context from the canvas

If you're paying attention you'll notice that the "gameCanvas" is the same id we gave the <canvas> element in our HTML page.

Now that we have the drawing context we're ready for business.

Enter the following code under what you've already written:

```
// Draw a rectangle
context.rect(20,20,150,100);
context.stroke();
```

Draw a rectangle

Finally we'll have a go at creating some variables and drawing some text. As mentioned before, we'll keep things fairly simple this week so that you can get a good understanding of the foundations. You can start preparing now to be blown away by next week's tutorial where we'll do all the awesome things.

The following code will make a variable to hold some text, do a simple operation on that variable, and then display it by drawing it to the canvas.

```
varscreenWidth = canvas.width/2;
varscreenHeight = canvas.height/2;
var message = "Hello";

message = message + " world!";

context.font="18px Arial";
context.fillText(message, screenWidth, screenHeight);
```

You might notice that we're telling the text to draw in the centre of the screen, but if you look at the program running the text actually starts drawing from the centre of the screen – it's not actually centred.

If you're a bit OCD like me, and you want your text exactly in the centre of the screen, then you can update the code like this

```
context.font="18px Arial";
vartextMeasure = context.measureText(message);
~~context.fillText(message, screenWidth, screenHeight);~~
context.fillText(message, screenWidth - (textMeasure.width/2), screenHeight);
```

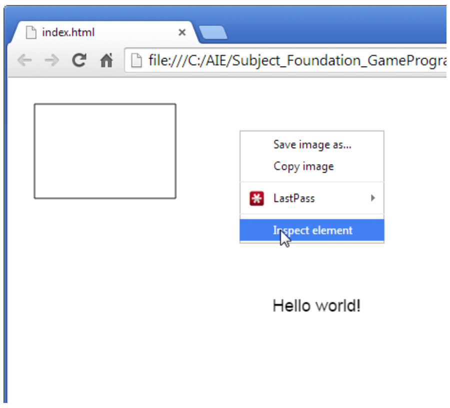
Unfortunately we can only measure how wide the text is, not how high. Your OCD will just have to live with it.

Debugging your Program:

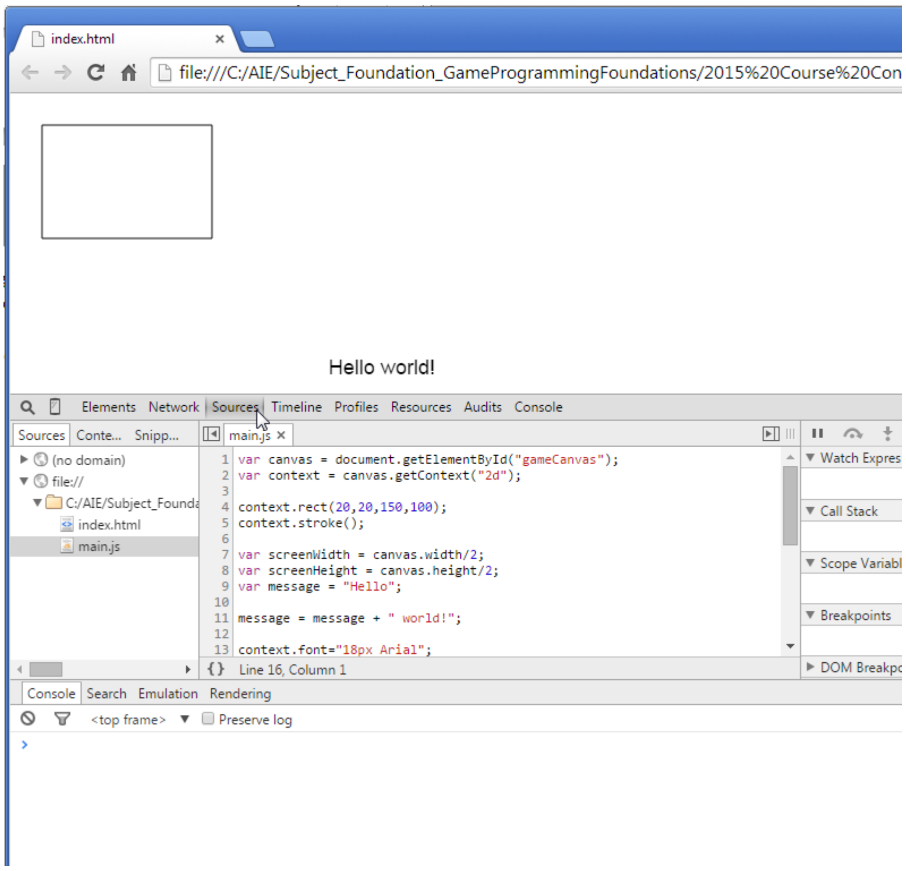
By this stage everything should be working and you should see a rectangle and the 'Hello World' message being drawn to the screen.

While everything's working normally we'll take a look at some of the debugging features of the Google Chrome browser. This will prepare you with the skills you'll need to debug your own programs when things start to go wrong.

1. Open your program in the Chrome browser by either dragging and dropping the index.html file into the browser, or by double-clicking the file (provided Chrome is set as your default browser).
2. When the page opens, right-click anywhere on the page and select 'Inspect Element'



3. This will open the debugging window in Chrome. From the tabs at the top of the debugging window, select 'Sources'

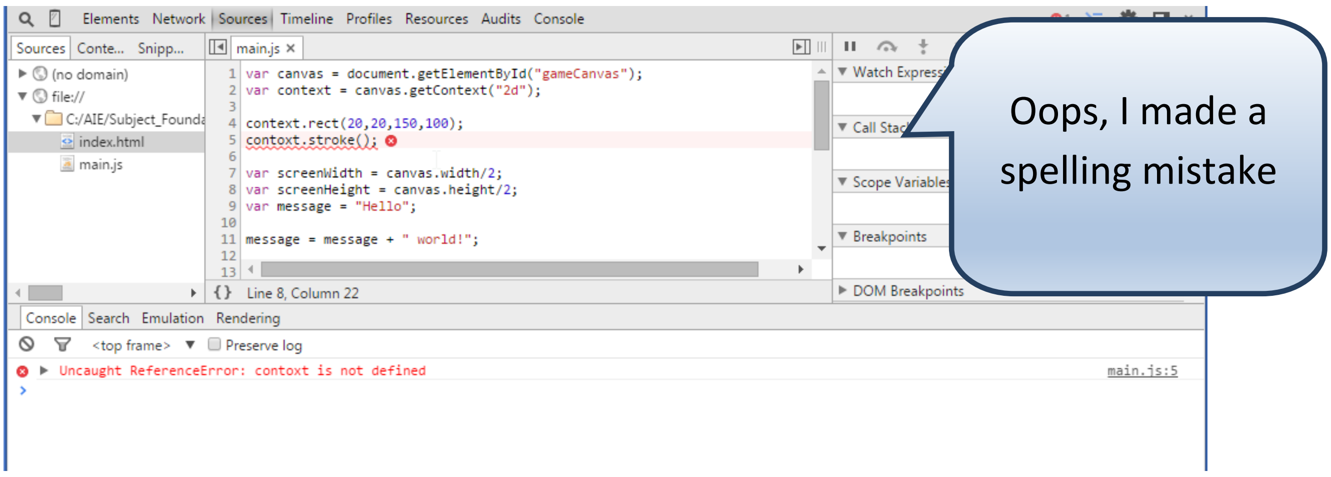

index.html
file:///C:/AIE/Subject_Foundation_GameProgrammingFoundations/2015%20Course%20Con
Hello world!
Elements Network Sources Timeline Profiles Resources Audits Console
Sources Conte... Snipp...
(no domain)
file://
C:/AIE/Subject_Founda
index.html
main.js
Watch Expres
Call Stack
Scope Variabl
Breakpoints
DOM Breakpo
Line 16, Column 1
Console Search Emulation Rendering
<top frame>
Preserve log

```
var canvas = document.getElementById("gameCanvas");
var context = canvas.getContext("2d");

context.rect(20,20,150,100);
context.stroke();

var screenWidth = canvas.width/2;
var screenHeight = canvas.height/2;
var message = "Hello";

message = message + " world!";

context.font="18px Arial";
```

4. In the 'Sources' window on the left, if the file main.js is not already selected then select it. The source code in main.js will then appear in the middle window.

Any time your program is not performing the way you want it to – or if you're not seeing anything at all – you can open this debugging view to help you identify the problem.

If you have any errors in your game (spelling mistakes are a common source of errors) then they will appear in red in the 'Console' window, like this:



```
var canvas = document.getElementById("gameCanvas");
var context = canvas.getContext("2d");

context.rect(20,20,150,100);
contxt.stroke();

var screenWidth = canvas.width/2;
var screenHeight = canvas.height/2;
var message = "Hello";

message = message + " world!";
```

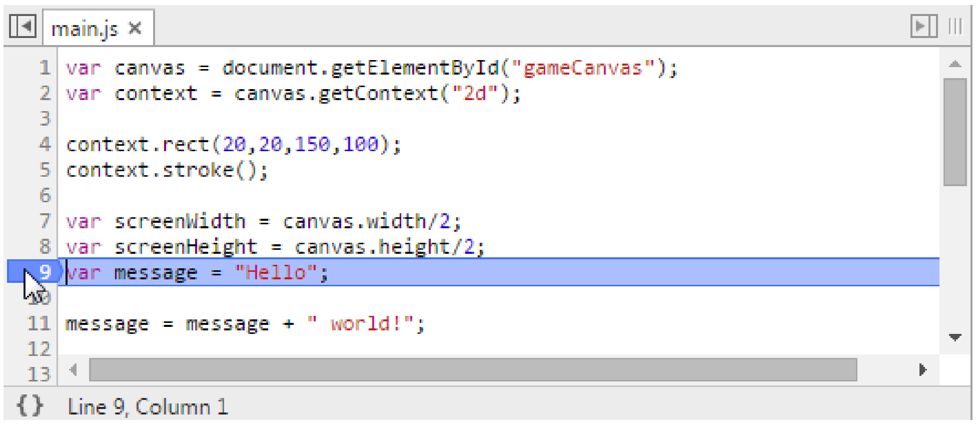
```
Uncaught ReferenceError: contxt is not defined                main.js:5
```

If everything is working normally, then we can continue.

We're going to place a breakpoint in our program. Breakpoints let you stop the execution of your program at a specific place. When the program reaches the breakpoint it will pause until you tell it to resume again.

This is very useful if you ever want to investigate the value of a variable or determine which path of execution your program is taking as it is running.

5. Place a breakpoint on line 9 by clicking in the grey bar where the number 9 appears

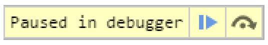


```
1  var canvas = document.getElementById("gameCanvas");
2  var context = canvas.getContext("2d");
3
4  context.rect(20,20,150,100);
5  context.stroke();
6
7  var screenWidth = canvas.width/2;
8  var screenHeight = canvas.height/2;
9  var message = "Hello";
10
11 message = message + " world!";
12
13
```

6. Press F5 to reload the page, or click on the reload button. (Mac uses can press Command+R)

   This will reload the page so that the program can start executing from the start. When the program reaches line 9, it will pause.

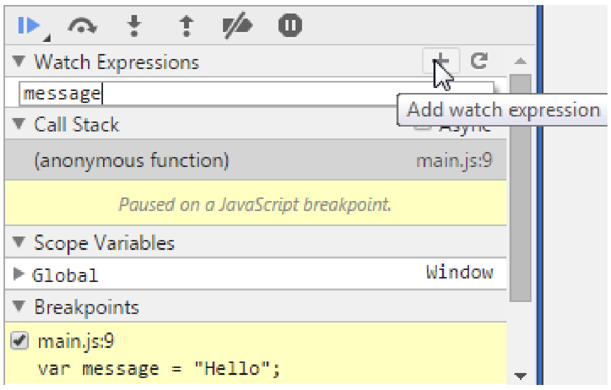
7. Your program should now be paused on line 9. You'll see the 'Paused in debugger' message at the top of the page.



   ▶ This button lets you resume the execution of your program. The program will un-pause and continue until it comes to the end of the program, or another breakpoint.

This button lets the program step to the next like of code. It will execute the next line, then pause again.

8. Add a new watch expression by pressing the '+' button in the 'Watch Expressions' window on the right.



```
▼ Watch Expressions
message
▼ Call Stack
(anonymous function)                main.js:9
Paused on a JavaScript breakpoint.
▼ Scope Variables
▶ Global                            Window
▼ Breakpoints
☑ main.js:9
  var message = "Hello";
```

Type 'message' in the text box.

This will add a watch on the variable message. Whenever the value inside the message variable changes, we'll see it happening in the Watch Expressions window.

At the moment the value of message is undefined because the variable hasn't been created yet.

9. Press the 'step over' button to continue to the next instruction.

   The value of the message variable should change to "Hello"

10. We're now on line 11, where we add the " world" to the "Hello".
    Press the step over button again and watch the value of message change to "Hello world"

11. Press the resume button to continue executing the program. The page should bow be fully drawn in the browser.

Congratulations, you've just debugged your first JavaScript program.

Get familiar with the Chrome debug window because it's a very useful tool to check if there are any errors in your program.

It's a good idea to have this open any time you run your program, just in case any errors are written to the console window.

Exercise:

That concludes this week's tutorial. You should now be able to create a simple HTML page, link it with a JavaScript file, and obtain a reference to the canvas element so that you can draw simple shapes and text. You should also now have a working understanding of how to use the Chrome debugging tools.

As an exercise, take a look at the W3Schools site's reference information for the canvas element: http://www.w3schools.com/tags/ref_canvas.asp

Have a go at drawing some other kinds of shapes yourself – perhaps some lines or arcs. Can you draw a rectangle filled with a colour? Experiment and see what you can come up with.