AJAX AND CANVAS

INTRODUCTION

AJAX is short for Asynchronous JavaScript And XML. The acronym is now a little dated as 'AJAX' calls are now most often associated with JSON. JSON is Javascript Object Notation and is an extremely common data format.

As with many other tasks in Javascript, jQuery has made the use of AJAX very straightforward. In this lab we are going to extend the `<canvas>` game from the previous lab and add AJAX calls in order to retrieve high score data.

ADDING A START BUTTON

Open the file *snake.html* and the related Javascript file *js/snake.js*. Amend the HTML file to reference *js/snake.js*. As we intend to use jQuery for AJAX you should also reference a copy of jQuery ie:

```
<script src="js/jquery-1.11.3.min.js"></script>
```

Notice that the game starts as soon as the page is loaded. Add an event to the `<button>` such that the game will only start when the button is pressed. Using an anonymous function the event handler would need to include the code from:

```
var newFoodCoords = createFood();
```

to:

```
for(var i = snakeLength-1; i>=0; i--){
            snakeArray.push({x: i, y:0});
}
```

Using the 'vanilla' Javascript the event could appear as:

```
document.getElementById('start').addEventListener('click', function(ev){
      …
}
```

As we will be using jQuery for AJAX you could also choose to use the jQuery click event ie:

```
$('#start').on('click', function(){
      …
});
```

BUILDING A LIST OF HIGH SCORES

Our game is going to store user high scores. In order to do so we will need a database table. Open your copy of phpMyAdmin by visiting *homepages.shu.ac.uk/mysql*.

Create a table called *snakeGame* with the following structure:

☐	1	**ID**	int(11)	No	*None*	AUTO_INCREMENT
☐	2	**gameScore**	int(11)	No	*None*	
☐	3	**gamePlayer**	varchar(255) utf8_unicode_ci	No	*None*	

To populate the table with some test data use the import feature to load the records in the *data/testData.csv* file.

QUERYING THE DATA

Open the file *api/listScores.php*. This file will query the table created above in order to list the high score in the *snake.html* page. Change the include path to ensure that your *conn.inc.php* file is referenced correctly.

This file will generate an output of JSON. As such add the following to the top of the file:

```
header('Content-Type: application/json');
```

This adds the 'Content-Type: application/json' to the HTTP headers telling the receiving browser to expect JSON content.

The PHP code in *api/listScores.php* places the values for *gameScore* and *gamePlayer* into an associate array using the associate keys *dbplayer* and *dbscore*.

Add the following code to test the data is populating the array:

```
print_r($returnAr);
```

Once you are happy you have data remove the `print_r()` method and then add the following:

```
echo json_encode($returnAr);
```

The above uses the PHP `json_encode()` method to encode the PHP array as JSON. It will appear as follows:

```
[{"dbPlayer":"Jane","dbScore":"8"},{"dbPlayer":"Steve","dbScore":"6"},{"dbP
layer":"Bob","dbScore":"2"}]
```

DISPLAYING THE DATA

To display the data in the *snake.html* file we need to make an AJAX call. To do this we will use the jQuery method `$.get()`.

The jQuery `$.get()` method makes a HTTP GET call to a given URL. The request can optionally include data, a callback function and a datatype the method should expect to receive in return. If not set jQuery will make an 'intelligent guess'.

```
jQuery.get( url [, data ] [, success ] [, dataType ] )
```

We will reference *api/listScore.php* as the URL and have the data returned to an anonymous function ie:

```
$.get('api/listScores.php',function(myData){
      console.dir(myData);
});
```

Using `console.dir` we can see the data returned by the AJAX call consists of an array of objects. As such we can use jQuery's `$.each()` method as this is designed to iterate over arrays and objects. This can be used to extract property names and values.

```
$.each(myData, function(propertyName, propertyValue){
      …
}
```

Add the following to see the values returned.

```
$.each(myData, function(name, val){
      console.info(name);
      console.info(val);
}
```

As we are dealing with an array rather than an object the name value returns the index of the array. The value returned will be a Javascript object ie:

```
{dbPlayer: "Jane", dbScore: "8"}
```

To extract the *dbPlayer* and *dbScore* values use dot notation ie:

```
val.dbScore
```

We would like the value added to `<ul class="highscore">` in the HTML:

```
<ul class="highScores">

</ul>
```

With jQuery we could generate a variable called `tempHTML` and populate it via the loop to include a `<li>` for each player.

```
var tempHTML = "";
$.each(myData, function(name, val){
      tempHTML +='<li>';
      tempHTML +='<span>';
      tempHTML += val.dbScore;
      tempHTML +='</span>'
      tempHTML +='<span>'
      tempHTML += val.dbPlayer;
      tempHTML +='</span>'
      tempHTML +='</li>'
});
$('.highScores').html(tempHTML);
```

RESTRUCTURING THE HIGH SCORE CODE

In order to easily reference the code used to create the high score list, we could restructure the code so that it no longer uses an anonymous function. Change the code as follows:

```
$.get('api/listScores.php', scoreTableBuilder);
```

… and create a `scoreTableBuilder()` function.

```
function scoreTableBuilder(myData){
      $('.highScores').html("");
      var tempHTML = "";
      $.each(myData, function(name, val){
            tempHTML +='<li>';
            tempHTML +='<span>';
            tempHTML += val.dbScore;
            tempHTML +='</span>'
            tempHTML +='<span>'
            tempHTML += val.dbPlayer;
            tempHTML +='</span>'
            tempHTML +='</li>'
      });
      $('.highScores').html(tempHTML);
}
```

CHECKING FOR A NEW HIGH SCORE

In the Javascript file locate the code that finishes a game on collision with either the canvas wall or the tail of the snake. This is flagged as `// ajax to database`.

To check whether the user has achieved a new high score we need to run a query against the database via the file *api/checkScore.php*. Add the following SQL to the query.

```
SELECT MAX(gameScore) AS highScore FROM snakeGame
```

The PHP then returns a JSON object containing the current high score ie:

```
{"highScore":"5"}
```

We can then take this value and compare it to the user's score. Using jQuery's `html()` and `show()` methods chained together we can respond to the user's score.

```
$.get('api/checkScore.php', function(data){
	if(data.highScore < score){
		$('#msg').html("New High Score").show();
		$('#newHighScore').show();
	}else{
		$('#msg').html("Game Over").show();
	}
});
```

The `<div id="highscore">` is displayed if the user beats the highest score in the database. This element contains a HTML form to allow the user to add their name to the 'hall of fame'.

ADDING A NEW HIGH SCORE TO THE DATABASE

Add a hidden field to upload the score to the database:

```
<input type="hidden" name="playerScore" id="playerScore">
```

Amend the conditional logic that checks the highscore to update the value in this new hidden element. We can use jQuery's `val()` method to do this:

```
$('#playerScore').val(score);
```

When the user submits the form, the database will be updated using AJAX. To do so we'll use the `$.post()` method. As we are using AJAX we don't want the form to be submitted in the traditional/native way. Adding `ev.preventDefault()` will prevent the form submitting.

```
$('#playerUpdate').on('click', function(ev){
		ev.preventDefault();
});
```

For the values in the form to be sent to the database they need to be in the format of name:value pairs. Fortunately jQuery provides the `serialize()` method that creates a text string in standard URL-encoded notation. So the following:

```
var formVars = $('#playerInfo').serialize();
```

… will produce a text string based on the name:value pairs in this form:

```
playerScore=0&playerEmail=bob%40bob.com
```

This value can then be sent to the database via the jQuery `$.post()` method.

```
$('#playerUpdate').on('click', function(ev){
            ev.preventDefault();
            var formVars = $('#playerInfo').serialize();
            $.post('api/addNewHighScore.php', formVars, function(){
      });
});
```

"UPSERTING" THE NEW HIGH SCORE

When the new high score is sent to the database, we need to check if this is a new user.

```
if($numRows == 1){
      $stmt = $mysqli->prepare("UPDATE snakeGame SET gameScore = ? WHERE ID
= ?");
      $stmt->bind_param('ii', $playerScore, $ID);
      $stmt->execute();
}else{
      $stmt = $mysqli->prepare("INSERT INTO snakeGame(gameScore,
gamePlayer) VALUES (?, ?)");
      $stmt->bind_param('is', $playerScore, $playerEmail);
      $stmt->execute();
}
```

The conditional logic checks to see if the user is a new user. If they are new a SQL INSERT is prepared, if they are a returning player then a SQL UPDATE is prepared.

Once the database has been updated we want to update the HTML. Use the SQL query logic from the *api/listScore.php* file, to output the scores in *api/addNewHighScore.php*.

The JSON returned can then be used in the Javascript ie:

```
$('#playerUpdate').on('click', function(ev){
            ev.preventDefault();
            var formVars = $('#playerInfo').serialize();
            $.post('api/addNewHighScore.php', formVars, function(data){
                        $('#newHighScore').hide();
                        scoreTableBuilder(data)
      });
});
```