Graphics with Canvas

Overview

In this lab, you'll implement a web page that displays graphics on a `<canvas>` element. If time permits, you'll also implement a web page to draw Bezier curves.

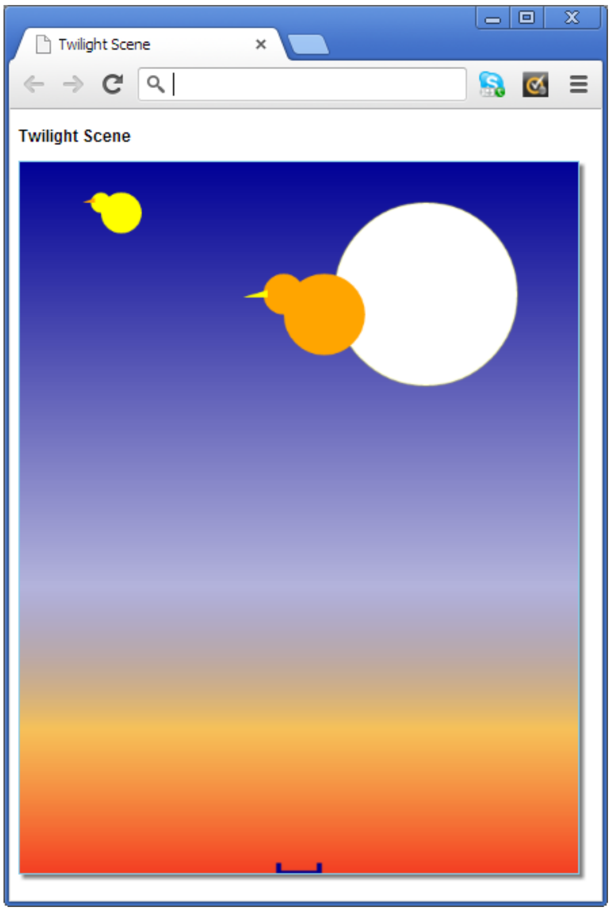
Roadmap

There are 4 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Creating a canvas and getting the context
2. Drawing a gradient background
3. Drawing shapes on the canvas
4. Drawing Bezier curves (if time permits)

Familiarization

Open a browser window and browse to `TwilightScene.html` in the *Solutions* folder. The web page appears as follows:



This might be the background scene for a simple HTML game. Your task in this lab is to draw this scene on the canvas:

- First you'll fill the entire content of the canvas using a gradient colour scheme.
- Next you'll draw the full moon in the top right corner.
- Then you'll draw the two birds (a small yellow bird and a larger orange bird). In the real game, these birds will be flying across the screen.
- Finally you'll draw the bucket at the bottom of the canvas. In a real game, the user would probably be able to move the bucket to catch balls before they hit the ground.

Exercise 1: Creating a Canvas and Getting the Context

In File Explorer, go to the *Start* folder and open `TwilightScene.html` in the text editor. Where indicated by the TODO comment, define a `<canvas>` element. Set its width and height to suitable values (e.g. 550 and 700 respectively).

Now open `TwilightScene.js` in the text editor. This file will contain all the implementation logic for the web page. In the `init()` function, add code to get the 2d context for the `<canvas>` element, and store it in the global `context` variable.

Exercise 2: Drawing a Gradient Background

Locate the `drawSky()` function, and add code to draw the sky as a blue-to-red linear gradient (like a sunset). Here are some hints:

- First, save the current state of the context.
- Create a linear gradient that starts at the top-left corner of the canvas and ends at the bottom-left corner of the canvas. (Note that the `context` object has a `canvas` property, which gives you the `<canvas>` element associated with the context. Once you have the `<canvas>` element, you can then get its height).
- Add several colour stops to define a colour gradient similar to that in the *Solution* web page.
- Set the context's fill style to use the linear gradient.
- Draw a filled rectangle on the context (it will automatically use the current fill style, i.e. the linear gradient).
- Finally, restore the original state of the context (so that your colour changes are not retained).

Open the *Start* web page in the browser. Verify the canvas is painted with the linear gradient.

Exercise 3: Drawing Shapes on the Canvas

Locate the `drawMoon()` function, and add code to draw a big moon in the top-right corner of the canvas (after you've implemented the function, refresh the web page in the browser to ensure it all looks fine). Here are some hints:

- First, save the current state of the context as always.
- Set the context's fill style to white (for the moon filler), and set its stroke style to a faint yellowish colour (for the moon outline). Also set the line width to 2 pixels.
- Call `context.beginPath()` ready to draw the new shape. You always call `beginPath()` before you draw a new shape, to ensure you don't tag on to any previous shapes that might have been drawn.
- Call `context.arc()` to draw the moon as a full circle.
- Call `context.fill()` and `context.stroke()` to draw the filler and outline of the moon.
- Finally, restore the original state of the context as always.

Locate the `drawSmallBird()` function, and add code to draw a small yellow bird at the specified x, y coordinates. Here are some brief hints:

- The bird's head is a small yellow circle.
- The bird's body is a larger yellow circle.
- The beak is an orange triangle – use the `moveTo()` and `lineTo()` functions to draw the triangle.

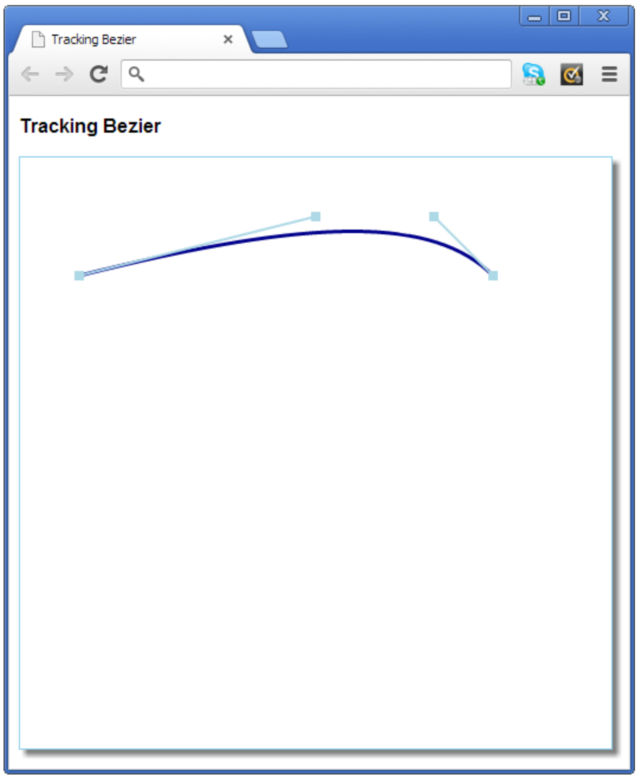
Now locate the `drawLargeBird()` function, and add code to draw a larger orange bird. The code is similar to `drawSmallBird()`, but the maths and colours are different!

Finally, locate the `drawBucket()` function, and add code to draw a bucket at the bottom of the canvas (in the middle). The `bucketWidth` and `bucketHeight` parameters specify the desired size of the bucket. Here are some brief hints:

- The bucket comprises three thick straight lines, representing the left-hand side of the bucket, the bottom, and the right-hand side.
- Ensure the bucket is positioned centrally at the bottom of the canvas – you'll need to do some maths based on the width and height of the canvas.

Exercise 4 (If Time Permits): Drawing Bezier Curves

Open a browser window and browse to `TrackingBezier.html` in the *Solutions* folder. The web page appears as follows:



The web page draws a Bezier curve. The Bezier curve connects the start point (on the left of the canvas) and the end point (on the right of the canvas). As you might recall, a Bezier curve has two control points that influence the shape of the curve. To illustrate this, our web page draws a light-blue line connecting the start point to the first control point, and another light-blue line connecting the end point to the second control point.

You can move the control points around on the screen, to see how they affect the shape of the Bezier curve:

- If you hold down the SHIFT key and move the mouse, it moves the first control point to that location.
- If you hold down the CONTROL key and move the mouse, to moves the second control point to that location.

Try it out, and observe how the shape of the curve changes when you move the control points around the canvas.

Switch back to File Explorer and open `TrackingBezier.html` in the *Start* folder. Notice that the web page already has a `<canvas>` element. All the code for drawing on the canvas is located in `TrackingBezier.js`, so open this file in the text editor and take a look at the existing code and comments.

- The global variables at the start of the file define the initial coordinates for the start point, the end point, and the two control points. Each point is an array – element 0 is the x coordinate, and element 1 is the y coordinate.
- The `init()` function gets the 2d context for the canvas, and sets up an event handler for `mousemove` events on the canvas. It also calls `draw()` to draw the initial appearance of the canvas.
- The `mousemove()` function detects whether the SHIFT or CONTROL key is pressed, and either moves the first control point or the second control point (or neither). It also clears the canvas and redraws it.
- The `draw()` function contains code to draw all the points and straight lines for the canvas, by making use of the prewritten `drawPoint()` and `drawLine()` functions. Take a look at this code and make sure you understand it.

Where indicated by the TODO comment, complete the `draw()` function so that it draws a Bezier curve as dictated by the start point, the end point, and the two control points. Draw the curve in dark blue, with a line thickness of 3 pixels.

Open the *Start* web page in the browser. Verify the Bezier curve appears correctly in the canvas, and that the curve is repainted continuously as you move the mouse around.