How it Works: Painting the Canvas

In this lesson, we will discuss the workings of the exercise on the canvas tag from the previous exercise. Let's begin!

WE'LL COVER THE FOLLOWING

HOW IT WORKS

In **step four**, you defined the `<canvas>` tag and set it up with the `“myCanvas”` identifier. The page loads the **drawing.js** file in its `<head>` section, so the `<script>` placed right after the canvas knows the `initDraw()` function, and invokes it with the id of the canvas passed. The `initDraw()` function does all initialization, and it invokes the `draw()` method to paint the scenery:

drawing.js

```
function initDraw(elemId) {
   ctx = document.getElementById(elemId).getContext('2d');
   ballX = BALL;
   ballY = HEIGHT - BALL - SOIL - GRASS;
   draw();
 }
```

Line two is the most important, as it sets up the context object (`ctx`), which can be used to draw shapes on the canvas. First using `document.getElementById()`, it retrieves the object representing the `“myCanvas”` element on the page. By invoking getContext(‘2d’), it obtains a context object that allows drawing to a two-dimensional surface and stores it in `ctx`.

> 📜**NOTE:** You’re probably asking yourself whether there is a three-dimensional drawing context based on the fact that a two-dimensional

> drawing context is used (“2d” in the code). Not yet, but the creators of HTML5 have left space for one in the future. There are frameworks using canvas with three-dimensional rendering, such as (Babylon.js) [http://www.babylonjs.com/].

The `<canvas>` element uses a coordinate system that designates its top-left corner as the point $(0, 0)$. The canvas used in index.html has a width of $500 pixels$, and a height of $250 pixels$, so its bottom-right corner is the point $(499, 249)$.

Using this coordinate mapping, **Lines three and four** initialize the position of the ball, and then **Line five** calls the `draw()` function.

It is now time to examine `draw()`, since it tells a lot about the canvas API:

drawing.js

```
function draw() {
    drawArea();

    // Draw ball
    ctx.fillStyle = BALLCOLOR;
    ctx.beginPath();
    ctx.arc(ballX, ballY, BALL, 0, Math.PI * 2);
    ctx.closePath();
    ctx.fill();
 }
```

The canvas API offers low-level and low-overhead operations.

For example, to draw the ball, you set up a fill style (**line five**), start a path (**line six**), draw a circle (**line seven**), close the path (**line eight**), and finally instruct the engine to fill the path you’ve just declared. **Line seven** defines the circle as an arc with a center point of (`ballX`, `ballY`), a radius of `BALL`, and the starting point of the arc at 3 o’clock on a dial (0), and by clockwise movement it goes back to 3 o’clock (`Math.PI*2`).

> 📜**NOTE:** Angles in the canvas API use radians (rad). So, 0 rad (0°) is at 3 o’clock, PI/2 rad (90°) at 6 o’clock, PI rad (180°) at 9 o’clock, 3*PI/2 rad (270°) at 12 o’clock, and finally, 2*PI rad (360°) is at 3 o’clock, again.

The `drawArea()` function is used to draw the sky, soil, and grass. It uses a similar approach to paint rectangular areas, as `draw()` does to paint the ball.

The following code snippet uses the `rect()` operation to create the soil:

drawing.js

```
ctx.fillStyle = SOILCOLOR;
ctx.beginPath();
ctx.rect(0, HEIGHT - SOIL, WIDTH, SOIL);
ctx.fill();
```

Here `rect` uses the (`0, HEIGHT-SOIL`) point as the top-left corner of the rectangle, the third and fourth arguments define its width and height, respectively.

Complete live demo at your service!

The complete implementation of the exercise from the previous lesson is given below for you to play around and experiment with.

Learn and enjoy! :)

```
// --- Constants
const HEIGHT = 250;
const WIDTH = 500;
const BALL = 12;
const SOIL = 17;
const GRASS = 3;
const BALLCOLOR = '#CC333F';
const SKYCOLOR = '#9CC4E4';
const SOILCOLOR = '#6A4A3C';
const GRASSCOLOR = '#93A42A';

// --- Drawing context
var ctx;
var ballX;
var ballY;

function initDraw(elemId) {
  ctx = document.getElementById(elemId).getContext('2d');
  ballX = BALL;
  ballY = HEIGHT - BALL - SOIL - GRASS;
  draw();
}

function drawArea() {
  // Draw sky
  ctx.fillStyle = SKYCOLOR;
  ctx.beginPath();
  ctx.rect(0, 0, WIDTH, HEIGHT - SOIL - GRASS);
  ctx.fill();
```

```
  // Draw soil
  ctx.fillStyle = SOILCOLOR;
  ctx.beginPath();
  ctx.rect(0, HEIGHT - SOIL, WIDTH, SOIL);
  ctx.fill();

  // Draw grass
  ctx.fillStyle = GRASSCOLOR;
  ctx.beginPath();
  ctx.rect(0, HEIGHT - SOIL - GRASS, WIDTH, GRASS);
  ctx.fill();
}

function draw() {
  drawArea();

  // Draw ball
  ctx.fillStyle = BALLCOLOR;
  ctx.beginPath();
  ctx.arc(ballX, ballY, BALL, 0, Math.PI * 2);
  ctx.closePath();
  ctx.fill();
}
```

As Exercise 5-10 demonstrated, it's pretty easy to create a static canvas. When you create real apps (games) you use a canvas that displays animations.

Thanks to the low-level graphic primitives of the canvas API, animations are simple to create, as you will learn in the next lesson's exercise.