Is this really a course about games? Where are the graphics?

INTRODUCTION

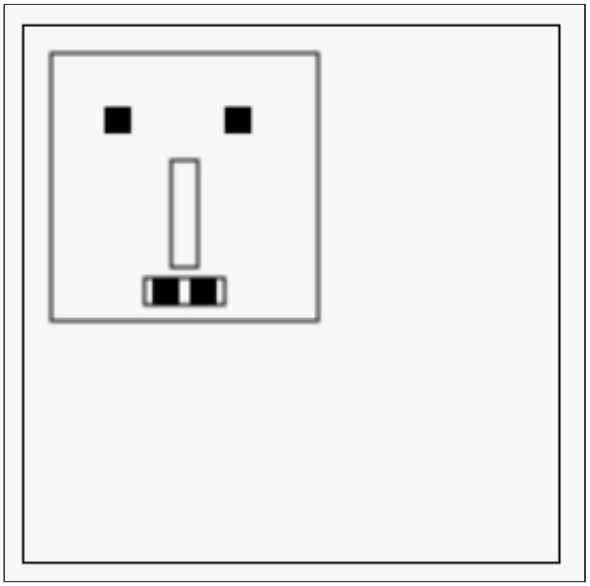
- *Note*: drawing in a canvas has been studied in detail in HTML5 Part 1, week 3.

Good news! We will add graphics to our game engine in this lesson! So far, we have just played with "basic concepts", but naturally you can't wait to draw something, to animate, and to move shapes on the screen :-)

First, a quick reminder of the basic concepts of using the canvas; with the same "monster" we used in many examples during the HTML5 Part 1 course.

HTML5 CANVAS BASIC USAGE: DRAWING A MONSTER

How to draw a monster in a canvas: you can try it online at JSBin.



HTML code (declaration of the canvas):

```
    <!DOCTYPE html>
```

```
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>Draw a monster in a canvas</title>
</head>
<body>
   <canvas id="myCanvas" width="200"height="200"></canvas>
</body>
</html>
```

The canvas declaration is at *line 8*. Use attributes to give it a `width`and a `height`, but unless you add some CSS properties, you will not see it on the screen: it's transparent!

Example of CSS to reveal the canvas, for example, add a 1px black border around it:

```
canvas {
   border: 1px solid black;
}
```

And here is a reminder of best practices when using the canvas, as described in the HTML5 Part 1 course:

1. In a function that is called AFTER the page is fully loaded (and the DOM ready), get a pointer to the canvas node in the DOM.
2. Then, get a 2D graphic context for this canvas (the `context` is an object we will use to draw on the canvas, to set global properties such as color, gradients, patterns and line width).
3. Only then can you can draw something,
4. And do not forget to use global variables for the canvas and context objects. I also recommend keeping the width and height of the canvas somewhere. This might be useful later.
5. For each function that changed the context (color, line width, coordinate system, etc.), start by saving the context, end with restoring the context.

Here is the JavaScript code that corresponds to these best practices:

```
// useful to have them as global variables
var canvas, ctx, w, h;


window.onload = function init() {
  // Called AFTER the page has been loaded
  canvas =document.querySelector("#myCanvas");
  // Often useful
  w = canvas.width;
  h = canvas.height;
  // Important, we will draw with this object
  ctx = canvas.getContext('2d');
  // Ready to go!
  // Try to change the parameter values to move
  // the monster
  drawMyMonster(10, 10);
};

function drawMyMonster(x, y) {
  // Draw a big monster!
  // Head
  // BEST practice: save the context, use 2D transformations
  ctx.save();
  // Translate the coordinate system, draw relative to it
  ctx.translate(x, y);
  // (0, 0) is the top left corner of the monster.
  ctx.strokeRect(0, 0, 100, 100);
  // Eyes
  ctx.fillRect(20, 20, 10, 10);
  ctx.fillRect(65, 20, 10, 10);
  // Nose
  ctx.strokeRect(45, 40, 10, 40);
  // Mouth
  ctx.strokeRect(35, 84, 30, 10);
  // Teeth
  ctx.fillRect(38, 84, 10, 10);
  ctx.fillRect(52, 84, 10, 10);
```

```
    // BEST practice: restore the context
50.   context.restore();
    }
```

In this small example, we used the `context` object to draw a monster using the default color (black) and wireframe and filled modes:

- `ctx.fillRect(x, y, width, height)`: draws a rectangle whose top left corner is at (x, y) and whose size is specified by the `width` and `height` parameters.
- `ctx.strokeRect(x, y, width, height)`: same but in wireframe mode.
- Note that we used (*line 30*) `ctx.translate(x, y)` for moving the monster more easily, and that all the drawing orders are coded as if the monster was in (0, 0), at the top left corner of the canvas (look at *line 33*). We draw the body outline with a rectangle located at (0, 0)). Calling `context.translate` just before will "changesthe coordinate system" and moves the "old (0, 0)" to (x, y).
- *Line 19*: we call the `drawMonster` function with (10, 10) as parameters, that will translate the original coordinate system to (10, 10).
- And if we change the coordinate system (this is what the call to`ctx.translate(...)` does) in a function, it is always good practice to save the previous context at the beginning of the function and restore it at the end of the function (*lines 27 and 50*).

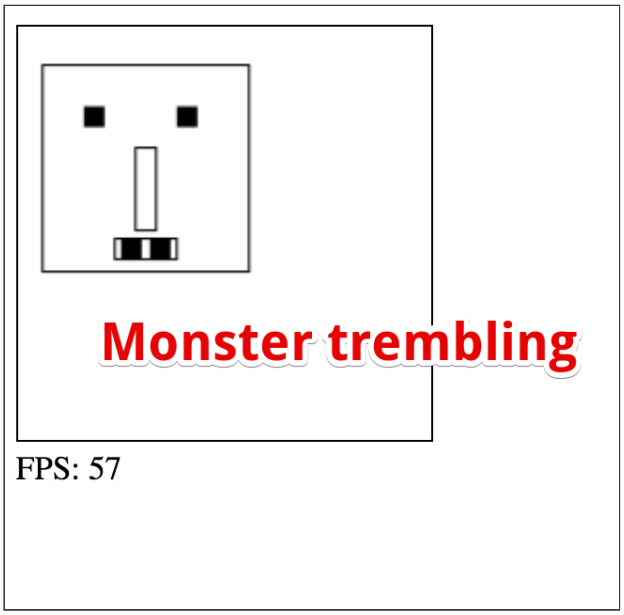
ANIMATE THE MONSTER, INCLUDE IT IN OUR GAME ENGINE

Ok, now that we know how to move the monster, let's integrate it into our game engine:

1. add the canvas to the HTML page,
2. add the content of the `init()` function to the `start()` function of the game engine,
3. add a few global variables `(canvas, ctx`, etc.),
4. call the `drawMonster(...)` function from the mainLoop,
5. add a random displacement to the x, y position of the monster to see it moving,

6. in the main loop, do not forget to clear the canvas before drawing again; this is done using the `ctx.clearRect(x, y, width, height)` function.

You can try this version online at JSBin.



HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>Trembling monster in the Game Framework</title>
</head>
<body>
 <canvas id="myCanvas" width="200"height="200"></canvas>
</body>
</html>
```

JavaScript complete code:

```
// Inits
window.onload = function init() {
  var game = new GF();
```

```
  game.start();
};


// GAME FRAMEWORK STARTS HERE
var GF = function(){
  // Vars relative to the canvas
  var canvas, ctx, w, h;

  ...
  var measureFPS = function(newTime){

    ...
  };
  // Clears the canvas content
  function clearCanvas() {
    ctx.clearRect(0, 0, w, h);
  }
  // Functions for drawing the monster and perhaps other
objects
  function drawMyMonster(x, y) {
    ...
  }
  var mainLoop = function(time){
    // Main function, called each frame
    measureFPS(time);
    // Clear the canvas
    clearCanvas();
    // Draw the monster
    drawMyMonster(10+Math.random()*10,10+Math.random()*10);
    // Call the animation loop every 1/60th of second
    requestAnimationFrame(mainLoop);
  };

  var start = function(){
    ...
    // Canvas, context etc.
    canvas =document.querySelector("#myCanvas");

    // often useful
```

```
    w = canvas.width;
    h = canvas.height;
    // important, we will draw with this object
    ctx = canvas.getContext('2d');

    // Start the animation
    requestAnimationFrame(mainLoop);
};

//our GameFramework returns a public API visible from outside its scope
return {
  start: start
};
};
```

Explanations:

- Note that we now start the game engine in a `window.onload` callback (*line 2*), only once the page has been loaded.
- We also moved 99% of the `init()` method we wrote in the previous example into the `start()` method of the game engine, and added the `canvas, ctx, w, h` variables as global variables to the game framework object.
- Finally we added a call to the `drawMonster()` function in the main loop, with some randomness in the parameters: the monster is drawn with an offset in x,y between [0, 10] at each frame of animation.
- And we clear the canvas content before drawing the current frame content (*line 35*).

If you try the example, you will see a trembling monster. The canvas is cleared + monster drawn at random positions around 60 times per second!

In the next part of this week's course, we'll see how to interact with it using the mouse or the keyboard.