The game framework skeleton

INTRODUCTION

Let us introduce the skeleton of a small game framework, based onthe Black Box Driven Development in JavaScript. In other words: a game framework skeleton is a simple object-based model that uses encapsulation to expose only useful methods and properties.

We will make this framework evolve throughout the lessons in this course, and cut it in different files once it becomes too large to fit in one single file.

Here is the starting point:

```
var GF = function(){
  var mainLoop = function(time){
    //Main function, called each frame
    requestAnimationFrame(mainLoop);
  };
  var start = function(){
    requestAnimationFrame(mainLoop);
  };
  // Our GameFramework returns a public API visible from
outside its scope
  // Here we only expose the start method, under the "start"
property name.
  return {
    start: start
  };
};
```

With this skeleton, it's very easy to create a new game instance:

```
var game = new GF();
```

```
    // Launch the game, start the animation loop, etc.
    game.start();
```

LET'S PUT SOMETHING INTO THE `MAINLOOP`FUNCTION, AND CHECK IF IT WORKS

Try this online example at JSBin, with a new `mainloop`: (check the JavaScript and output tabs). It should display at 60 frames/s a random number right in the body of the document. We are far from a real game yet, but we're improving our game engine :-)

Source code extract:

```
var mainLoop = function(time){
  // main function, called each frame
  document.body.innerHTML = Math.random();
  // call the animation loop every 1/60th of second
  requestAnimationFrame(mainLoop);
};
```

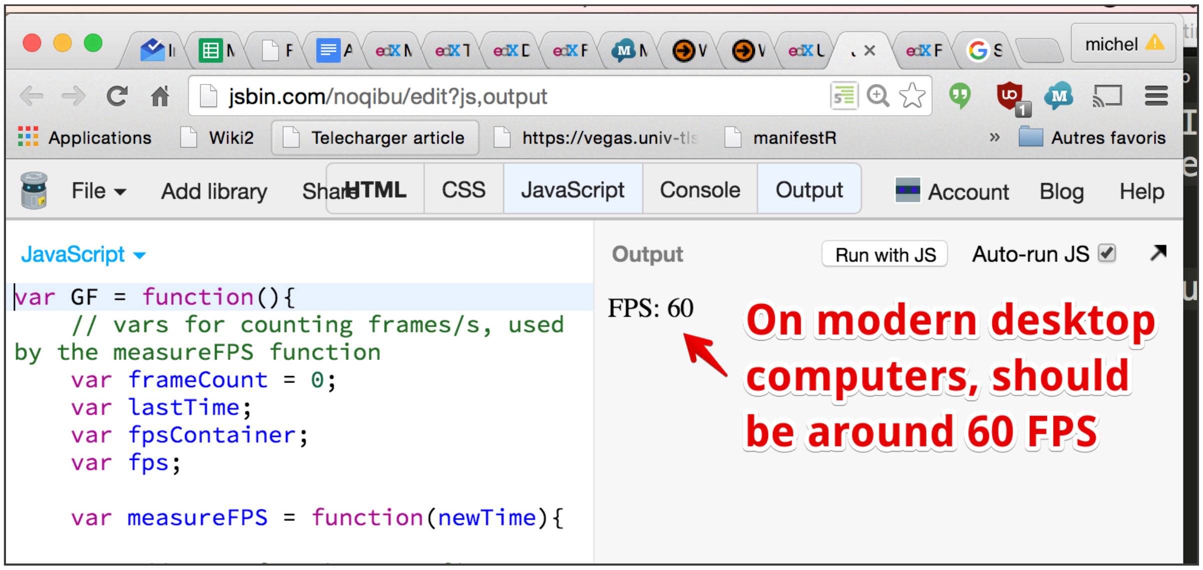
LET'S COUNT THE NUMBER OF FRAMES PER SECONDS IN THAT ANIMATION

Every game needs to have a FPS measuring function.

The principle is simple:

1. Count the time elapsed by adding deltas in the `mainloop`.
2. If the sum of the deltas is greater or equal to 1000, then 1s elapsed.
3. If at the same time, we count the number of frames that have been drawn, then we have the number of frames per second. Remember it should be around 60 frames/second!

Here is a screenshot of an example and the code we added to our game engine, for measuring FPS (try it online at JSBin):



```
var GF = function(){
    // vars for counting frames/s, used
by the measureFPS function
    var frameCount = 0;
    var lastTime;
    var fpsContainer;
    var fps;

    var measureFPS = function(newTime){
```

```
FPS: 60
```

On modern desktop computers, should be around 60 FPS

Source code extract:

```
    // vars for counting frames/s, used by the measureFPS
 function
  var frameCount = 0;
  var lastTime;
  var fpsContainer;
  var fps;
  var measureFPS = function(newTime){
    // test for the very first invocation
    if(lastTime === undefined) {
      lastTime = newTime;
      return;
    }
    // calculate the delta between last & current frame
    var diffTime = newTime - lastTime;
    if (diffTime >= 1000) {
      fps = frameCount;
      frameCount = 0;
      lastTime = newTime;
    }
    // and display it in an element we appended to the
    // document in the start() function
    fpsContainer.innerHTML = 'FPS: ' + fps;
    frameCount++;
  };
```

And we will call the `measureFPS` function from inside the animation loop, passing it the current time, given by the high resolution timer that comes with the `requestAnimationFrame` API:

```
var mainLoop = function(time){
  // compute FPS, called each frame, uses the high resolution
time parameter
  // given by the browser that implements the
requestAnimationFrame API
  measureFPS(time);
  // call the animation loop every 1/60th of second
  requestAnimationFrame(mainLoop);
};
```

And the `<div>` element used to display FPS on the screen is created in this example by the `start()` function:

```
var start = function(){
  // adds a div for displaying the fps value
  fpsContainer =document.createElement('div');
  document.body.appendChild(fpsContainer);
  requestAnimationFrame(mainLoop);
};
```

HACK: ACHIEVING MORE THAN 60 FRAMES/S ? IT'S POSSIBLE BUT TO AVOID EXCEPT IN PRIVATE HACKERS' CIRCLES!

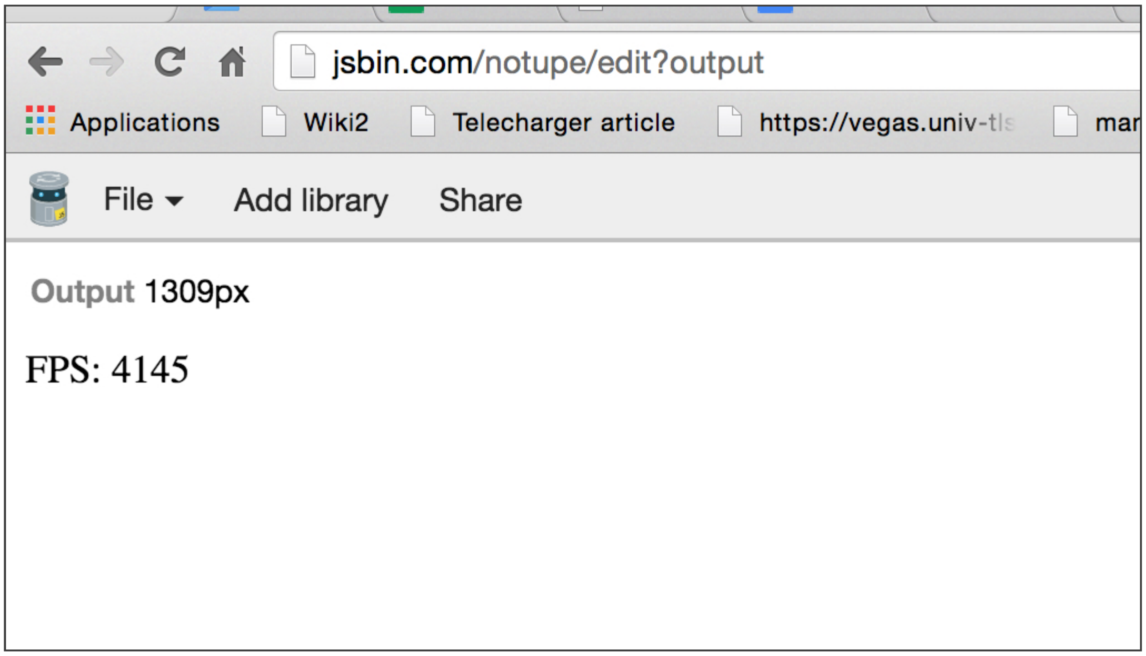
We also know methods of implementing loops in JavaScript and achieving even more than 60fps (this is the limit using`requestAnimationFrame`).

My favorite hack uses the `onerror` callback on an `<img>` element like this:

```
function mainloop(){
  var img = new Image;
  img.onerror = mainloop;
  img.src = 'data:image/png,' +Math.random();
}
```

What we are doing here, is creating a new image on each frame and providing invalid data as a source of the image. The image cannot be displayed properly, so the browser calls the `onerror` event handler that is the `mainloop` function itself, and so on.

Funny right? Please try this and check the number of FPS displayed with this JSBin example.



```
FPS: 4145
```

Source code extract of this example:

```
var mainLoop = function(){
  // main function, called each frame
  measureFPS(+(new Date()));
  // call the animation loop every LOTS of seconds using
previous hack method
  var img = new Image();
  img.onerror = mainLoop;
```

```
    img.src = 'data:image/png,' +Math.random();
};
```

KNOWLEDGE CHECK 2.3.1

How do we measure the time between two consecutive animations?

- ○ We use the parameter passed by the browser to the animation loop. This is a requestAnimationFrame API feature.
- ○ We use the Date() JavaScript object, that returns the current time.