

# Using `setTimeout()` instead of `setInterval()`

## INTRODUCTION

One thing you should always remember about using `setInterval`: if we set number of milliseconds at - let's say 20ms - it will call our game loop function EACH 20ms, *even if the previous one is not yet finished*. This may lead to many problems (incomplete rendering, etc.).

That's where we can use another function:

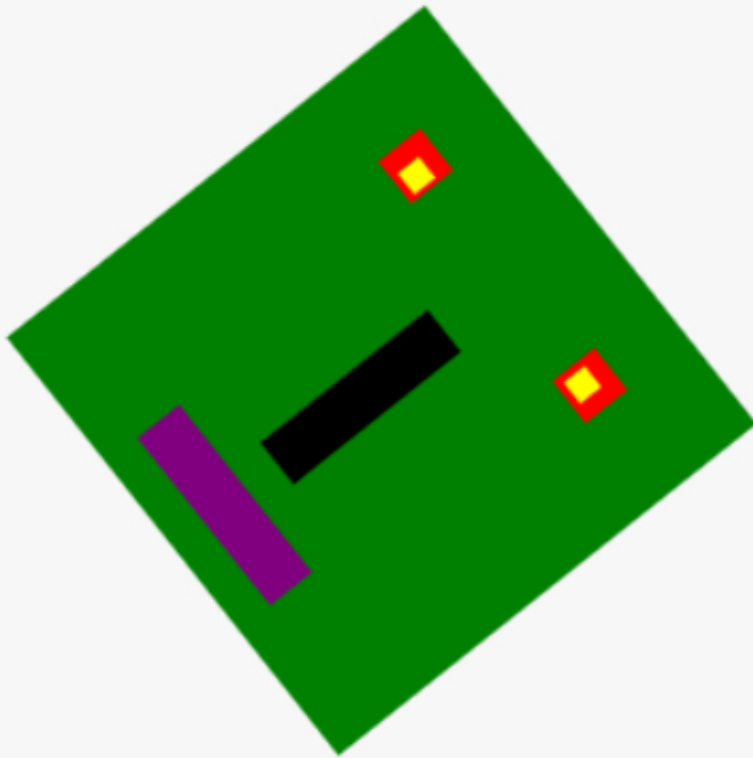
- Syntax: `setTimeout(function, ms);`

This function works like `setInterval(...)` with one difference: it calls your function ONCE and *AFTER a given amount of time*.

## EXAMPLE OF THE MONSTER ANIMATED IN A CANVAS WITH `SETTIMEOUT`

Example of use: <http://jsbin.com/timebu/3/edit> (open the JavaScript, console and output tabs).

## Monster animated in a canvas using the `setTimeout(...)` method



Start animation

Stop animation

This is similar to the previous example except that we called `setTimeout(function, delay)` instead of `setInterval(function, period)`. **As `setTimeout` runs the function passed as the first parameter only once, we also have to call it at the end of the loop.**

Extract from source code:

```
function animationLoop() {  
  // 1 - Clear  
  ctx.clearRect(0, 0, canvas.width, canvas.height);  
  // 2 Draw
```

```

drawMonster(monsterX, monsterY, monsterAngle, 'green', 'yellow');
// 3 Move
monsterX += 10;
10. monsterX %= canvas.width
monsterAngle += 0.01;
// call mainloop again after 20ms
requestId = setTimeout(animationLoop, 20);
}

function start() {
// Start the animation loop, change 20 for bigger
// values
20. requestId = setTimeout(animationLoop, 20);
}
function stop() {
if (requestId) {
clearTimeout(requestId);
}
}

```

This function is certainly more suitable for doing graphic animation, such as for writing an HTML5 game. It will never interrupt an ongoing animation, even if the instructions inside the animation loop take too long.

## PROBLEMS WITH `setInterval()` AND `setTimeout()`

`setTimeout` does not "wait" during the timeout period. It lets the rest of the JavaScript code run. It schedules a new call to the function passed as first parameter with a timer running in the background. This might cause it to take slightly longer than the expected timeout period to start executing.

This problem also occurs with `setInterval`, the timing is not "very" reliable. If you plan to run a function every 20ms, and if you measure precisely the real timing, sometimes you will discover big differences between what is scheduled and what is performed. This is because these methods were designed a long time ago, when high precision timers

and 60 frames per second animation were not an option.

Here comes the [requestAnimationFrame API](#), a very good companion to the canvas API!

**BEST PRACTICE:** AVOID using `setTimeout` for animating in a canvas, except for trivial cases.

**For 60 frames/second animation, use `requestAnimationFrame`!**

---

## KNOWLEDGE CHECK 4.2.4 (NOT GRADED)

```
[  
  setInterval(function, milliseconds);
```

```
  setTimeout(function, milliseconds);  
]
```

---

Regarding the two lines of code above, which statement is true?

- ☐ They will produce the same result: call the function passed as first parameter every interval of time passed as the second parameter (in milliseconds)
  - ☐ `setTimeout` will call the function only once, after a delay corresponding to the value passed as second parameter.
-