Javascript – deel 07

Deze les behandelt enkele onderwerpen die nodig zijn voor de “hit an object” opdracht.

Permanente evaluatie

In de volgende les kan je docent je oplossingen van de opdrachten opvragen en laten meetellen voor je permanente evaluatie.

Verslag

In dit deel van de cursus staan verschillende vragen die je moet beantwoorden en opdrachten om iets te maken of uit te proberen. Het is belangrijk dat je alle opdrachten zorgvuldig uitvoert!

Documenteer je werk in een verslag document waarin je

- voor elke uitprobeer opdracht een entry maakt met screenshots ter staving van wat je deed
- je antwoorden op de gestelde vragen neerschrijft

Oplossingen van grotere opdrachten (met veel code) bewaar je apart in een .zip file.

Werken met timers

Je kunt in javascript op twee manieren met timers omgaan

- setInterval : code herhaaldelijk laten uitvoeren om de zoveel milliseconden
- setTimeout : code éénmaal laten uitvoeren na een wachttijd in milliseconden

De setInterval function

Met deze function kun je een van je eigen functies om de zoveel milliseconden laten uitvoeren.

Je kunt aangeven welke functie moet herhaald worden en hoe vaak de herhaling dient te gebeuren :

```
let taskId = setInterval(f, m);
```

Dit zal de functie f om de m milliseconden uitvoeren (of toch proberen).

De 'f' parameter is een verwijzing naar een functie, dit kan de naam van een functie zijn of een functie-expressie. We noemen dit trouwens een callback functie. Op zich is dit niks nieuws, we kwamen dit ook al tegen bij de addEventListener en de array.sort methods.

Merk op dat het niet gegarandeerd is dat de timing wordt aangehouden (best effort only). Enkele factoren die de timing kunnen beinvloeden zijn

- wat je javascript programma nog zoal doet (bv. meerdere timers)
- hoeveel uitvoeringstijd je browser krijgt van het OS (multitasking)

Mocht je moeten weten hoeveel milliseconden er daadwerkelijk verstreken zijn, bv. om een animatie te maken die een accurate timing volgt :

1. vraag de huidige tijd in milliseconden sinds 1/1/1970 op via Date.now()
2. bereken het verschil met de vorige keer dat je functie f werd opgeroepen
3. bewaar de tijd uit stap 1 voor de volgende timer callback

De teruggegeven taskId heb je nodig wanneer je de timer weer wil afzetten :

```
clearInterval( taskId )
```

Voorbeeld

Een programmafragment dat om de 500ms het woordje "hallo" op de console zet en stopt na 10 keer.

```
let count = 0;
let taskId = setInterval(print, 500);

const print = () => {
  if (count<5) {
    console.log("hallo");
    count++;
  } else {
    clearInterval(taskId);
  }
}
```

Je kan dit trouwens makkelijk uitproberen in het Console venster van de Chrome Developer Tools.

De setTimeout function

De setTimeout function kan je eigen functie oproepen na een opgegeven wachttijd in milliseconden.

```
let taskId = setTimeout(f, m);
```

Dit zal de 'f' function éénmaal uitvoeren nadat 'm' milliseconden verstreken zijn.

De wachttijd gaat in net na de oproep van setTimeout, maar ook hier is niet gegarandeerd dat onze function na exact 'm' milliseconden wordt opgeroepen. Het is weerom een *best effort only* verhaal, afhankelijk van wat je computer nog zoal aan het doen is.

De taskId heb je nodig om de geplande oproep te annuleren

```
clearTimeout( taskId );
```

Belangrijke opmerking

Je browser gebruikt maar één enkele thread om je javascript code uit te voeren.

Dit betekent dat er dus niks "tegelijkertijd" gebeurt, de browser kan enkel je timer functionaliteit oproepen indien hij niet met een ander stuk van je programma bezig is. Anders gezegd, de opgegeven callback function kan enkel opgeroepen worden als je programma met 'niets' bezig is.

Enkele gevolgen hiervan zijn

- setInterval : indien de uitvoering van je callback function langer duurt dan de opgegeven periode, zal je programma steeds meer "achterop" raken.

- setTimeout : indien je programma na de setTimeout oproep nog iets doet dat langer duurt dan de opgegeven wachttijd, zal hoedanook langer gewacht worden dan je had voorzien.

Een typisch voorbeeld van de eerste situatie is een complexe animatie die te lang duurt. het kan echter ook in een tragere context optreden zoals een webpagina die haar data automatisch periodiek (opnieuw) binnentrekt van een server. Indien je bv. om de 10sec een request stuurt maar de server antwoordt pas na 15sec, dan raakt je programma dus 5sec achterop. Na verloop van tijd zal dit falen door teveel *pending* (i.e. openstaande) requests.

> *Een mogelijke oplossing hiervoor is niet met setInterval te werken, maar voor setTimeout te kiezen. Je moet dan wel op het einde van de callback function steeds de volgende setTimeout oproep lanceren.*

Random getallen

Je kan een willekeurig getal bekomen met de volgende method

```
let getal = Math.random();
```

De variabele 'getal' zal een kommagetal bevatten in [0,1[

Indien je een getal in [0,15[wil, doe je

```
let getal = Math.random() * 15;
```

Indien je een getal in [10,25[wil, doe je

```
let getal = 10 + Math.random() * 15;
```

Gehele getallen bekomen

Indien je een geheel getal wil bekomen op basis van een kommagetal (dus zonder fractie), kun je de volgende Math methods gebruiken :

Math.floor(kommaGetal) dit
zal naar onderen afronden bv.
Math.floor(7.67) geeft 7
bv. Math.floor(-7.34) geeft -8!

Math.ceil(kommaGetal) dit
zal naar boven afronden bv.
Math.ceil(7.34) geeft 8
bv. Math.ceil(-7.67) geeft -7!

Math.round(kommaGetal)
dit zal afronden
bv. Math.round(7.34) geeft 7
bv. Math.round(7.67) geeft 8
bv. Math.round(7.5) geeft 8
bv. Math.round(-7.5) geeft 7!

Let goed op met waarden waarvan je niet op voorhand weet of ze positief dan wel negatief zijn!

Globale variabelen

We zagen eerder al dat globale variabelen kunnen gebruikt worden om gegevens te onthouden over meerdere functie-oproepen heen.

(terzijde : het is niet omdat twee functies allebei een lokale varibele 'x' hebben, dat je er een globale variabele van moet maken!)

Zoals in elke programmeertaal is het beter om het gebruik van globale variabelen zo veel mogelijk te beperken. De voornaamste redenen hiervoor zijn

- het risico op een naam conflict : indien 2 delen van het programma toevallig dezelfde naam kiezen voor een 'eigen' globale variabele zullen ze elkaars waarden overschrijven.
- de toegang tot globale variabelen is ongecontroleerd : indien er een foutieve waarde

in een globale variabele terechtkomt, is het moeilijk om de verantwoordelijke code te vinden.

Hoe complexer en omvangrijker het programma, hoe relevanter deze redenen worden.

Bij javascript in een browseromgeving is de situatie voor globale variabelen nog precairder; globale variabelen zijn daar nml. properties van het window object. Hierdoor is het risico op een naam conflict nog groter

- het window object zit sowieso al vol met voorgedefinieerde properties
- de namen van 'losse' functies leiden eveneens tot properties van het windows object
- sommige zaken uit de DOM-tree leiden tot properties van het window object
    - nml. alle waarden van id-attributen en sommige name-attributen http://w3c.github.io/html/browsers.html#named-access-on-the-window-object
        - (maak hier trouwens nooit gebruik van!!)

Om de vervuiling van deze global namespace tegen te gaan kan men slim gebruik maken van *closures* en *Immediately Invoked Function Expressions* (IIFE) om variabelen af te schermen. Deze geavanceerde techniek valt echter buiten het bestek van deze cursus.

Een eenvoudige manier om de risico's van globale variabelen enigszins te beperken is ze samen in één enkel object te bundelen en slechts 1 globale variabele te definiëren. Bijvoorbeeld :

```
let global = {
        score : 0,
        players : [],
        PATH_PREFIX : "images\sprite",
        PATH_SUFFIX : ".png"
}
```

Aanspreken van de property score

```
        let score = global.score
```

Dit heeft als bijkomend voordeel dat je ze makkelijk kan terugvinden bij het debuggen, open gewoon een Watch voor deze ene variabele en je kan via diens properties al je globale variabelen bekijken.

Belangrijk : een globale variabele hoort bij (de uitvoering van een javascript programma in) de context van het huidig HTML document. Als de browser naar een andere pagina navigeert of dezelfde pagina herlaadt, gaat deze variabele en diens waarde onherroepelijk verloren.

Opdracht: hit an object

In deze opdracht programmeren we een klein interactief spel. Elke seconde laten we een figuur zien die moet worden aangeklikt. Per aangeklikt object krijgt de speler een punt. Klikt de speler een bom aan, dan is het spel over.

Bekijk: hitAnObject.mp4

Je implementeert een timer die elke seconde een object toont of weg haalt van het scherm. Bij elke klik op een object, wordt een punt toegekend. Is het object een bom en wordt er daar op geklikt, dan is het spel afgelopen. Het aantal juiste "hits" wordt bijgehouden zoals in het filmpje is weergegeven.

Hoe moet ik aan de opdracht beginnen?

Begin vanaf het gegeven lege project en voeg de functionaliteit in stappen toe, bv.

1) Creëer een div met id="playField". Geef de playField een hoogte van 800px en een breedte van 600px. Voeg in dit div-element een <img> toe voor 0.png.
2) voeg aan dat image element een click event listener toe die een alert toont als je erop klikt
3) Definieer de volgende constanten als globale variabelen

```
let global = {
   IMAGE_COUNT: 5,  // aantal figuren
   IMAGE_SIZE: 48, // grootte van de figuur
   IMAGE_PATH_PREFIX: "images/",  // map van de figuren
   IMAGE_PATH_SUFFIX: ".png",  // extensie van de figuren

   MOVE_DELAY: 3000, // aantal milliseconden voor een nieuwe afbeelding verschijnt

   score: 0,    // aantal hits
   timeoutId: 0 // id van de timeout timer, zodat we die kunnen annuleren
};
```

4) verplaats het image naar een willekeurige nieuwe positie als je erop klikt
5) vervang het image door een ander als je erop klikt
6) experimenteer met de setInterval functie, zodat er elke seconde een bericht op de console komt
7) zorg dat de afbeelding elke seconde verspringt
8) voeg ook een knop toe om het spel te starten

Opdracht : Matching game (een mooie uitdaging!!)

Voorbeeld: speel een spel (20 cards) op

https://www.memozor.com/memory-games/for-kids/dora-the-explorer

Maak zelf een memory spel met 12 kaarten (i.e. 6 verschillende prentjes).

- Noem de afbeeldingen kaart1.png, ..., kaart6.png
- Noem de afbeelding voor de achterkant van een kaart achterkant.png
- Definieer de volgende constanten als globale variabelen
  let AANTAL_HORIZONTAAL=4;
  let AANTAL_VERTICAAL=3;
  let AANTAL_KAARTEN=6;
- Stop de namen van de te gebruiken afbeeldingen in een array
- Bouw dynamisch de nodig elementen om de kaarten op het scherm te krijgen
- Indien de gebruiker op een kaart klikt, wordt deze omgedraaid
- Indien 2 omgedraaide kaarten gelijk zijn, worden ze verwijderd
- Zoniet worden ze na een korte wachttijd weer omgedraaid
- Om bij te houden welke kaarten al omgedraaid zijn kun je een css class gebruiken
  je zult die waarschijnlijk sowieso nodig hebben om al dan niet een kaart te tonen
- Wanneer er geen kaarten meer over zijn eindigt het spel

Dit is een grote opdracht, ga best stapsgewijs tewerk :

1. maak eerst een programma dat de kaarten zichtbaar op het scherm brengt
   - je kan hiervoor eerst de html hardcoderen (i.e. de kaarten in je html stoppen)
   - probeer dan de css goed te krijgen zodat de presentatie er goed uitziet
   - verwijder dan de kaarten uit de html en voeg ze programmatorisch via javascript toe
2. eenmaal dit werkt, maak de volgende uitbreidingen
   - toon enkel de achterkanten
   - definieer een click handler die de kaart omdraait
   - voeg logica toe om te beslissen of kaarten verwijderd dan wel teruggedraaid moeten worden (let erop dat de overblijvende kaarten op dezelfde plaats moeten blijven!)
   - voeg een wachttijd toe zodat de gebruiker de gelegenheid krijgt om te zien of het goed of fout was alvorens de foute terug te draaien of de goede te verwijderen.
   - voeg een visuele indicator toe die tijdens de wachttijd aangeeft of het goed of fout was (bv. groene/rode border rond prentje)

Enkele aandachtspunten

- Een klik op een kaart die reeds omgedraaid is, mag geen effect hebben.
- In de finale versie moet de volgorde van de kaarten willekeurig zijn, maar hou tijdens het ontwikkelen een vaste volgorde aan zodat je makkelijker kan testen zonder te moeten zoeken naar de juiste kaarten.
- Zorg ervoor dat als je met wachttijden werkt, de gebruiker niet kan klikken op kaarten want anders draait het wellicht in de soep (en hou je bv. achteraf onverwacht solo kaarten over)
    - Een simpele manier om dit te doen is in je click event listener een voorwaarde checken (bv. globale variabele isBusy op true/false).
    - Je kan hiervoor trouwens ook een hourglass cursor instellen zodat de gebruiker ziet waarom de kliks niet werken, zie https://developer.mozilla.org/en/docs/Web/CSS/cursor
- Om kaarten van het speeldveld te verwijderen kun je natuurlijk de corresponderende HTML elementen verwijderen uit de DOM-tree maar je kunt ook met de zichtbaarheid spelen, zie http://stackoverflow.com/questions/133051/what-is-the-difference-between-visibilityhidden-and-displaynone

Mogelijke uitbreidingen (voor wie vroeger klaar is dan voorzien)

- voorzie geluidseffectjes bij het omdraaien en terugdraaien van de kaarten.
    - zie http://stackoverflow.com/questions/9419263/playing-audio-with-javascript
- een variant waarin de gebruiker op zoek moet gaan naar 3 dezelfde kaarten ipv dubbels.
    - kun je je oplossing zo algemeen maken dat het aantal gelijke kaarten die moet gevonden worden, een simpele wijziging is aan een constant als let AANTAL_GELIJKE_KAARTEN=4 ?
- een variant op basis van geluid ipv afbeeldingen, zodat de gebruiker geen prentjes te zien krijgt maar per geklikte kaart een geluidje hoort en dezelfde geluiden moet herkennen
- een algoritme dat de layout zodanig instelt dat de kaarten steeds een mooie rechthoek vormen (dus alle rijen exact evenveel kaarten en niet een paar overblijvers op de laatste rij) die de beschikbare ruimte zo goed mogelijk invult
    - rekening houden met breedte/hoogte verhouding van de beschikbare ruimte
    - zie http://stackoverflow.com/questions/3513081/create-an-optimal-grid-based-on-n-items-total-area-and-hw-ratio