

Labo Gebruikersinterfaces

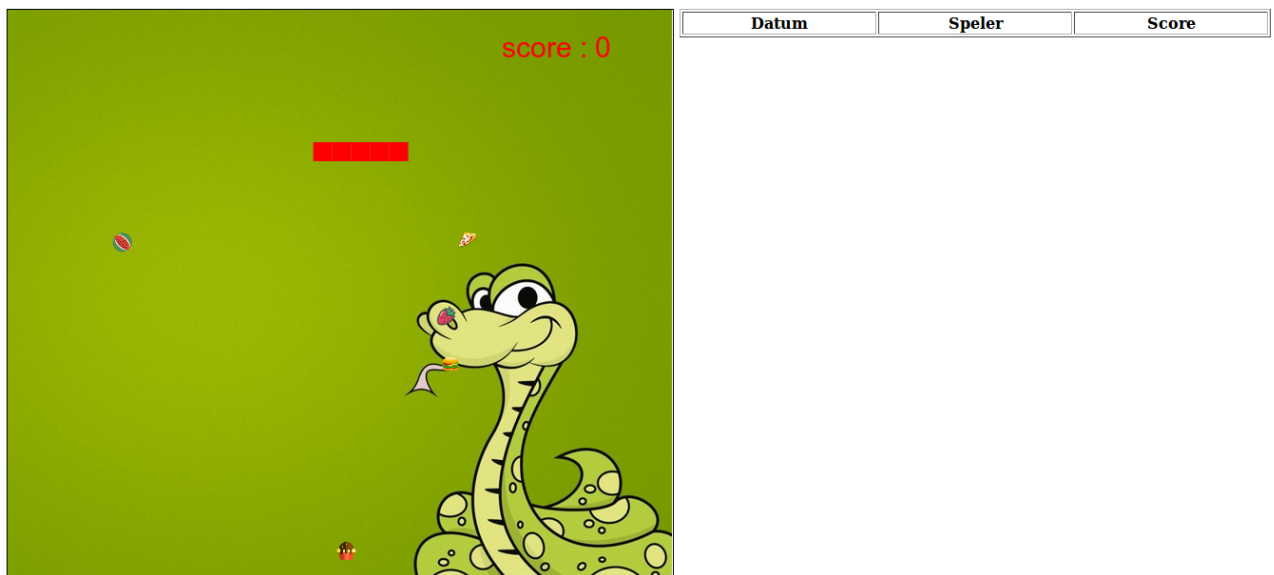
Reeks 3: JavaScript Snake

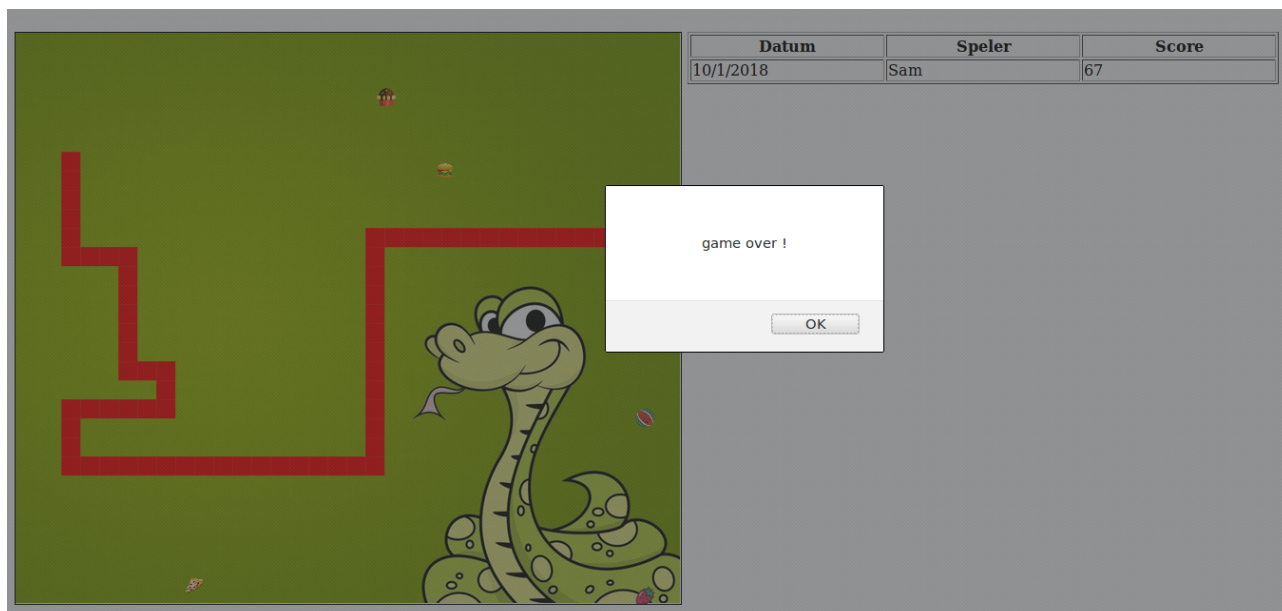
Doel: Een eenvoudig spel implementeren in Javascript (exacter: ES6 of ECMAScript6). Leren werken met het HTML5 canvas element en met de localStorage.

1 Inleiding

HTML5 introduceert een aantal nieuwe elementen en functionaliteiten zoals semantic tags en form validation. Eén van de belangrijkste vernieuwingen is het canvas-element. Dit element laat toe om op een eenvoudige manier te tekenen binnen een webpagina. Een andere nieuwe functionaliteit is de “local storage” die het mogelijk maakt om gegevens lokaal (client side) te bewaren.

Het doel van dit labo is om te leren werken met het HTML5 canvas en met de localStorage en om een complexe programmalogica in JavaScript te implementeren. In deze reeks zullen we een eenvoudige versie van het bekende spel “Snake” maken dat volledig binnen de browser draait.





2 ES6

Bekijk hoofdstuk 1 en 2 uit het boek “Advanced Game Design with HTML5 and Javascript” (zie theorie op Ufora).

Maak een nieuw “Empty project” aan in Webstorm/IntelliJ of een andere editor naar keuze.

3 Het HTML5 canvas

Het canvas biedt de ruimte aan waarbinnen het spel gespeeld zal worden.

- ◆ Maak een HTML5 pagina met een canvas van 700x600 pixels.
- ◆ Voeg de achtergrond met de slang (zie Ufora) toe aan de pagina.
- ◆ Voeg een css-pagina toe aan het project. Hier zorg je ervoor dat het canvas bovenop de achtergrond getoond wordt. Zoek meer informatie via www.w3schools.com/cssref/pr_pos_index.asp.

4 Experimenteren met het canvas

Het tekenen op het canvas gaat via JavaScript. Op internet vind je vele voorbeelden van hoe dit te doen. De officiële API is door W3C gedefinieerd: <http://www.w3.org/TR/2dcontext/>.

- ◆ Maak een nieuwe JavaScript-file aan en voeg de verwijzing ernaar toe in je HTML. We maken gebruik van ES6 modules, en dus wordt de javascript als volgt geïncludeerd: `<script src="script.js" type="module"></script>`.
- ◆ Zoek op hoe je vanuit JavaScript toegang krijgt tot het canvas.
- ◆ Teken een eenvoudige figuur (bvb een rechthoek) op het canvas.

Loopt er iets fout? Bekijk de foutmeldingen in de browser (in Google Chrome: klik rechts op de pagina en vraag “inspecteer”). Werd de pagina volledig geladen en de DOM volledig opgesteld, voor je elementen uit de DOM begon te manipuleren?

5 Een animatie op het canvas

Om een animatie te verkrijgen zal je periodiek de positie van de elementen op het canvas moeten herberekenen en ze op de juiste plaats opnieuw tekenen.

- ◆ Zoek op hoe vanuit je vanuit JavaScript bepaalde code periodiek kan laten uitvoeren.
- ◆ Teken een enkel vierkantje op het canvas dat continu verspringt naar een willekeurige plek op het canvas (elke 100 ms). Bedenk dat het canvas hiervoor telkens hertekend wordt en er eigenlijk telkens een nieuw vierkant getekend wordt. Nog dit: vraag breedte en hoogte van het canvas op in plaats van hard te coderen.
- ◆ Zorg nu dat dit vierkantje op het canvas continu verschuift in plaats van verspringt. Als het aan de rand komt moet het omkeren. De beginpositie initialiseer je met een random waarde maar je zorgt dat je een afstand van minstens 1/4 breedte en 1/4 hoogte van de rand blijft om te starten. De bewegingsrichting initialiseer je ook met een random waarde: ofwel horizontaal starten (random links of rechts), ofwel verticaal starten (random omhoog of omlaag) .

Tip: gebruik de variabelen x en y om de positie van het vierkantje te bewaren, en de variabelen dx en dy voor de bewegingsrichting. Controleer nauwkeurig aan de grenzen: keert het vierkantje op tijd terug?

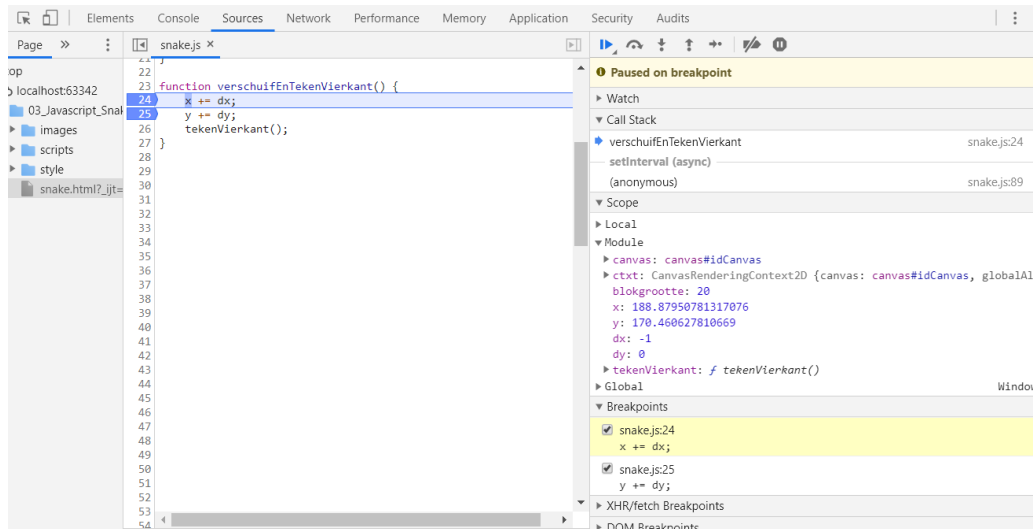
6 Luisteren naar keyevents

Zorg ervoor dat je de bewegingsrichting van het vierkantje kan veranderen met de pijltjestoetsen.

- ◆ Zoek op hoe je kan luisteren naar events van het toetsenbord. Let op, er is een verschil tussen *keypress* en *keydown*.
- ◆ Achterhaal welke codes overeenkomen met de pijltjestoetsen.
- ◆ Implementeer het veranderen van de bewegingsrichting van het vierkantje met de pijltjestoetsen.
- ◆ Voeg ook de mogelijkheid toe om het spel te pauzeren door op “p” te drukken. Het is niet voldoende om hiervoor de bewegingsrichting op (0,0) in te stellen: als het spel in pauze is, mag het ook niet meer reageren op pijltjestoetsen. Het kan enkel uit pauze gehaald worden door weer op “p” te drukken.

Debugtips

1. Gebruik de developer tools in firefox en chrome, vooral het consolevenster en console.log kunnen je helpen.
2. In chrome kan je met de functietoets F12 een venster met debugmogelijkheid openen. Onder het tabblad *Sources* kan je navigeren naar het .js-bestand. Stel breakpoints in door in de kantlijn van de code te klikken. Rechts vind je onder *Scope - Module* de mogelijkheid om de inhoud van de variabelen te bekijken



7 Gameloop

Voor we het volledige spel implementeren, houden we even halt bij de vorige opdrachten.

7.1 gameLoop

Om het spel te animeren (lees: om het blokje te verschuiven) heb je allicht een gameLoop-functie geïmplementeerd. Deze vormt het hart van het spel. Deze functie geef je als parameter mee aan de setInterval-functie van het window-object.

```
function gameLoop() {
    wisScherm();
    verschuifVierkant();
    tekenVierkant();
}
```

```
let timer = window.setInterval(gameLoop, timeOutPeriode);
```

7.2 keyHandler

Om naar het toetsenbord te luisteren, heb je aan het document (niet per se aan een bepaald element binnen het document) een eventListener toegevoegd.

```
let keyHandler = (e) => {
    if (e.key === "ArrowLeft") {
        dx = -1;
        dy = 0;
    } else if (e.key === "ArrowUp") {
        dx = 0;
        dy = -1;
    } [...]
    } else if (e.key === "p") {
        togglePauze();
    }
};
```

```
document.addEventListener("keydown", keyHandler);
```

Let op: `e.keyCode` is verouderd, gebruik ofwel `e.code` ofwel `e.key` (en duik eerst in de online documentatie voor je beslist wat best is). Vergelijk de waarde van `e.key` ook nooit met ASCII-codes, die zijn niet leesbaar.

7.3 Spel pauzeren

Om het spel te pauzeren, heb je een booleaanse variabele nodig die de staat van het spel bijhoudt. Afhankelijk hiervan, zet je de timer uit of terug aan.

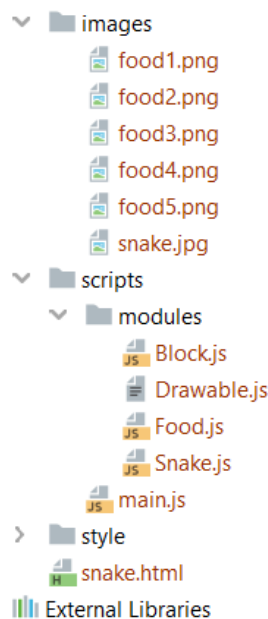
8 Voedsel en score op canvas

Terwijl het blokje (later: de slang) over het scherm beweegt, zijn er ook spelelementen die constant zichtbaar blijven. Teken 5 stuks voedsel op random plaatsen op het canvas. Op Ufora staan een aantal figuren die je kan gebruiken (`foodx.png`). Tips: concentreer je op wat er in de gameloop moet gebeuren. En worden de figuren niet gevonden ondanks dat hun bron juist ingesteld werd? Zorg dat de figuur eerst ingeladen is vooraleer je deze probeert te tekenen <http://www.jefclaes.be/2010/12/html5-drawing-images-to-canvas-gotcha.html>. Merk op dat het zeer onwaarschijnlijk is dat je deze fout zelf tijdens het testen zal tegenkomen, maar het is ‘good practice’ om hierop voorbereid te zijn.

Zet tenslotte in de linkerbovenhoek van het canvas ook de score klaar (voorlopig 0).

9 Modules

Tot nu toe heb je misschien zonder klassen gewerkt. Hier brengen we nu verandering in. Maak in de mappenstructuur ter hoogte van het javascript-bestand met het hoofdprogramma een nieuwe map met naam `modules`. Voor elke hieronder beschreven klasse maak je een nieuw bestand in die map.



9.1 Klasse Drawable

Schrijf de klasse `Drawable` die een object voorstelt dat zijn `x`- en `y`-positie bewaart (linkerbovenhoek van zijn positie relatief ten opzichte van het canvas waarop het getekend zal worden). De afmetingen van het canvas waarop het object getekend zal worden, wordt ook aan de constructor meegegeven. Voorzie een methode `draw(context)`, maar implementeer deze nog niet.

9.2 Klasse Food

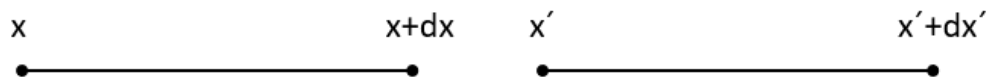
Schrijf de klasse `Food` die afgeleid is van de klasse `Drawable`. Voorzie

```
constructor(worldWidth, worldHeight, imageTitle, score, size)
```

De laatste parameter is de afmeting van de afbeelding (zorg voor een default-waarde van 30), de score is een geheel getal dat later in het spel belangrijk wordt. De `x`- en `y`-positie worden willekeurig toegekend binnen de grenzen van het canvas. (Zorg ervoor dat de afbeelding niet van het canvas valt, als ze helemaal rechts of onderaan gezet wordt.)

Voorzie een methode `reposition()` die random nieuwe `x`- en `y`-waarden bepaalt.

Voorzie een methode `intersects(x,y,dx,dy)` die nagaat of het huidige object op het canvas overlapt met de rechthoek met hoekpunten (x,y) en $(x+dx,y+dy)$. Een beetje wiskunde (lees: logisch denkwerk) kan hierbij helpen.



Bovenstaande lijnstukken overlappen *niet* omdat $\max(x,x')$ groter is dan $\min(x+dx,x'+dx')$.

Tip Indien je hierover liever later nadenkt (en nu doorwerkt aan de rest van de code), kan je de methode `intersects` voorlopig *schetsen*.

```
intersects(x,y,dx,dy){  
    return false;  
}
```

9.3 Klasse Block

Schrijf de klasse `Block` die afgeleid is van de klasse `Drawable`. Voorzie een `constructor(x, y, worldWidth, worldHeight, blockSize)` en stel de `blockSize` default in op 20. Implementeer de functie `draw`.

9.4 Klasse Snake

Schrijf de klasse `Snake`. Ook deze is afgeleid van `Drawable`! De slang bestaat bij de start uit 5 blokjes die allemaal op dezelfde plaats getekend worden. Bij elke beweging gaat de kop van de slang vooruit in de huidige bewegingsrichting. De andere blokjes volgen (lees: het blokje na de kop van de slang neemt nu de plaats van de kop in, etcetera).

Merk op: elk blokje is 20 op 20 pixels groot, en blokjes worden enkel op posities gezet die een veelvoud van 20 zijn. We geven hieronder nog een oplistijng van nodige/nuttige methodes.

```
constructor(worldWidth, worldHeight)
draw(context)
move()
setDirection(dx,dy)
grow(number)
intersectsSelf()
crashesIntoWall()
```

Bepaal zelf waar (in welke klasse, methodes, bestanden) je code schrijft om volgende gedragingen van de slang te implementeren.

- Laat de slang nooit op haar passen terugkeren. Als ze naar rechts beweegt, mag een druk op de pijl naar links geen effect hebben.
- Elk stuk voedsel telt voor een bepaalde score (leg zelf vast hoeveel); als de slang dit voedsel eet groeit ze met evenveel blokjes als de score aangeeft. (Het voedsel zelf verplaatst zich, maar zal nooit (ook niet per ongeluk) op de slang verschijnen.)
- Als de slang in haar eigen lichaam bijt, is het spel gedaan. (Tip: controleer alleen de kop van de slang.)
- Als de slang buiten het canvas wil breken, is het spel gedaan (langs de rand bewegen mag wel).

10 Weergave van high scores

Dankzij de nieuwe local storage functionaliteit kunnen we gegevens lokaal (client side) bewaren. Deze gegevens zijn ook de volgende keer dat de pagina geopend wordt nog beschikbaar.

- ◆ Zoek informatie op over deze local storage api.
- ◆ Er bestaan verschillende mogelijkheden om de gegevens op te slaan, welke zijn dit? Welke is het meest geschikt om high scores in op te slaan?
- ◆ Voeg een tabel toe aan de pagina. In deze tabel zal er een overzicht komen van wie wanneer welke score behaald heeft.
- ◆ Bij het openen van de pagina moet de naam van de gebruiker gevraagd worden (gebruik hiervoor de prompt functionaliteit van JavaScript).
- ◆ Op het einde van het spel moet de score tevoorschijn komen in de tabel, samen met de naam. Dit kan je implementeren door vanuit JavaScript de tabel aan te passen (nieuwe “tr” maken en toevoegen).

11 Uitbreidingen (optioneel)

- ◆ Voorzie verschillende soorten voedsel die verschillende punten opleveren.
- ◆ De verschillende soorten voedsel gedragen zich ook anders, sommige soorten blijven altijd op de zelfde plek, andere bewegen, anderen verdwijnen na verloop van tijd.
- ◆ Laat de gebruiker toe om de parameters van het spel aan te passen, voeg een HTML5 form toe waarin de gebruiker o.a. de snelheid van het spel kan instellen. Maak gebruik van geschikte HTML5 input velden (range, ...).

- ◆ Zorg ervoor dat de gebruiker een snapshot kan nemen van het spel door op 's' te drukken. Sla alle informatie over het spel (positie van de objecten, slang, score, ...) op in de localstorage en voorzie een knop om een bewaard spel opnieuw in te laden.