

Opdracht 2 - TypeScript en JavaScript

INTRODUCTIE

In deze opdracht ontwerpt en implementeert u de functionaliteit van het spelletje Snake.

De pagina voor Snake, die u in de eerste opdracht heeft ontwikkeld, vormt u zo om tot een interactief werkende webapplicatie.

U gaat daarbij als volgt te werk:

- U stelt de functionaliteit op in JavaScript.
- U test de functionaliteit met QUnit.
- U bedenkt hoe u de code in wilt delen in modules.
- U werkt die modules uit in TypeScript.
- U test de gegenereerde JavaScript met QUnit.
- U genereert documentatie met jsDoc.

1 De regels van Snake



FIGUUR 1.1 De user interface van Snake

Wanneer een gebruiker op Start drukt wordt er een nieuw spel gestart. In het midden van het speelveld wordt een slang getekend, bestaande uit twee segmenten (de slang in Figuur 1.1 heeft vier segmenten). Verder worden in het speelveld op willekeurige plaatsen voedseldeeltjes geplaatst (groen in de figuur).

De slang begint direct te bewegen zodra hij is gecreëerd. Hij beweegt in een vast tempo. Een handig tempo is één beweging per 500 milliseconden.

De richting waarin de slang beweegt bij de start van het spel is omhoog. De gebruiker kan vanaf dat moment met de vier pijltjestoetsen bepalen welke kant de slang op beweegt.

De kop van de slang heeft een andere kleur (oranje in de figuur) dan de rest van de slang (donkerrood in de figuur), zodat gemakkelijker te zien is wat de voorkant van de slang is.

Als de kop van de slang met een voedsелеlement samenvalt, wordt het voedsелеlement verwijderd en groeit de slang aan de achterkant met één segment. In figuur 1.1 heeft de slang dus al twee voedsелеlementen gegeten.

Zodra de kop van de slang botst op één van de overige segmenten van de slang, heeft de gebruiker verloren. De bewegingen van de slang worden gestopt en in de console komt te staan: 'VERLOREN!!!'. Aan de gebruiker duidelijk maken dat het spel verloren is mag natuurlijk ook op een mooiere manier dan een melding in de console.

Wanneer alle voedselsegmenten zijn opgegeten heeft de gebruiker gewonnen. De beweging van de slang wordt gestopt en in de console komt te staan: 'GEWONNEN!!!'. Ook hier geldt dat de mededeling aan de gebruiker natuurlijk ook op een mooiere manier mag gebeuren dan een melding in de console.

Als de gebruiker tijdens het spel Stop aanklikt, wordt het spel gestopt en wordt het speelveld leeggemaakt.

2 De uitgangscode

De HTML-pagina van het spelletje (`snake.html`) die de user interface vormt, heeft u al opgesteld. Bij deze opdracht krijgt u een bestand `uitgangscode.js` waarin een kleine aanzet voor de code is gegeven, vooral ook ok te laten zien hoe het tekenen op het canvas werkt.

Denk er aan dat u alles aan de code mag veranderen. We geven de code alleen om u een beetje op weg te helpen.

In de HTML bij de uitgangscode wordt zowel jQuery als jCanvas aangeroepen. Het is nodig om dat over te nemen in de eigen HTML.

U mag *geen* gebruik maken van andere libraries dan jQuery en jCanvas.

2.1 DATASTRUCTUUR

2.1.1 Constanten

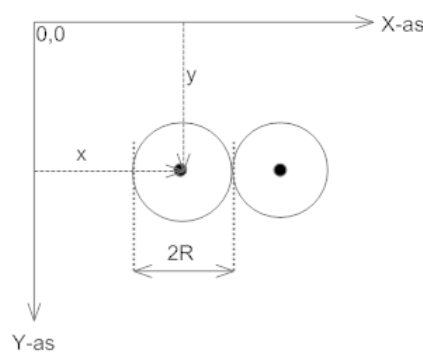
We geven hier eerst de constanten die we gebruiken. Daar onder volgt de uitleg.

```
const R      = 10,           // straal van een element
      STEP   = 2*R,         // stapgrootte
      LEFT   = "left",      // bewegingsrichtingen
      RIGHT  = "right",
      UP     = "up",
      DOWN   = "down",
      NUMFOODS = 15,        // aantal voedsелеlementen
      XMIN   = R,           // minimale x waarde
      YMIN   = R,           // minimale y waarde
```

```
SLEEPTIME = 500,      // aantal milliseconde voor de timer
SNAKE = "DarkRed" ,   // kleur van een slangsegment
FOOD = "Olive",       // kleur van voedsel
HEAD = "DarkOrange"   // kleur van de kop van de slang
```

De voedsелеlementen en de segmenten van de slang zijn cirkels met een straal van R . R is een constante die we als waarde 10 geven (10 pixels). Het middelpunt heeft een x - en een y -coördinaat (zie figuur 1.2).

In de figuur is te zien dat het punt $(0, 0)$ in het canvas *linksboven* ligt. Naar rechts en naar beneden tellen de coördinaten dus als positief, in tegenstelling tot hoe u gewend bent om bijvoorbeeld grafieken te tekenen met een x - en y -as.



FIGUUR 1.2 Voedsелеlementen en slangsegmenten

Een segment dat een stap neemt, gaat $STEP$ in een bepaalde richting. Voor $STEP$ nemen we tweemaal de straal: $2 \cdot R$. Daardoor schuift elk segment een volle plaats op.

We definiëren de vier bewegingsrichtingen $LEFT$, $RIGHT$, UP , en $DOWN$.

Ook het aantal voedsелеlementen waarmee het spel begint definiëren we als constante ($NUMFOODS$).

Om er voor te zorgen dat voedsелеlementen binnen het speelveld worden geplaatst en segmenten niet buiten het speelveld kunnen bewegen, definiëren we twee constanten als minimumwaarden voor de x - en y coördinaten van het middelpunt van een voedsелеlement of segment: $XMIN(R)$, $YMIN(R)$.

De maximumwaarden zullen we als variabelen definiëren, omdat we daarvoor de afmetingen van het canvas-element nodig hebben.

We definiëren constanten voor de kleuren ($SNAKE$, $HEAD$ en $FOOD$). U kunt, afhankelijk van uw implementatie, natuurlijk allerlei waarden veranderen, zoals de kleuren.

2.1.2 Globale variabelen

```
var snake,
    foods = [],      // voedsel voor de slang
    width,           // breedte van het tekenveld
    height;          // hoogte van het tekenveld
    xMax              // maximale waarde van x = width - R
    yMax,             // maximale waarde van y = height - R
    direction = UP;   // de richting waarin de slang beweegt
```

De `width` en `height` haalt u uit de DOM.

Voor de middelpunten van de voedsелеlementen en de segmenten geldt dus:

```
XMIN <= x <= xMax en  
YMIN <= y <= yMax
```

2.1.3 De slang en het voedsel

De segmenten van de slang en de voedsелеlementen zijn twee vormen van elementen. Een element heeft een middelpunt met `x`- en een `y`-coördinaat, een straal en een kleur. In de code die u als uitgangspunt heeft, ziet u de volgende (nog te implementeren) constructor:

```
function Element(radius, x, y, color) {  
    /* in te vullen */  
}
```

We hebben twee hulpfuncties geïmplementeerd. Eén om een voedsel-element te creëren en om één slangsegment te creëren:

```
function createSegment(x, y) {  
    return new Element(R, x, y, SNAKE);  
}  
function createFood(x, y) {  
    return new Element(R, x, y, FOOD);  
}
```

De slang bestaat uit een array van segmenten, waarbij het *laatste* segment de kop is. Dat laatste segment heeft dus een andere kleur dan de andere segmenten. Voor de slang geven we de volgende (nog te implementeren) constructor mee:

```
/**  
 * @constructor Snake  
 * @param segments een array met aaneengesloten slangsegmenten  
 *           Het laatste element van segments wordt de kop van  
 *           de slang  
 */  
function Snake(segments) {  
    /* in te vullen */  
}
```

2.1.4 Commando's

init De slang en het voedsel worden geïnitieerd in de (te implementeren) functie `init`:

```
/**  
 * @function init() -> void  
 * @desc Haal eventueel bestaand voedsel en een bestaande slang  
 *       weg, creëer een slang, genereer voedsel, en teken alles  
 */  
function init() {  
    /* in te vullen */  
}
```

De hulpfunctie `createStartSnake` geven we als volgt:

```
/**
```

```
@function createStartSnake() -> Snake
@desc Slang creëren, bestaande uit twee segmenten,
      in het midden van het veld
@return: slang volgens specificaties
*/
function createStartSnake() {
    var segments = [createSegment(R + width/2, R + height/2),
                    createSegment(R + width/2, height/2 - R)];
    snake = new Snake(segments);
}
```

De hulpfunctie `createFoods` creëert het voedsel en zorgt dat voedsel-elementen niet met elkaar en niet met slangsegmenten samenvallen:

```
/**
 * @function createFoods() -> array met food
 * @desc [Element] array van random verdeelde voedselpartikelen
 * @return [Element] array met food
 */
function createFoods() {
    var i = 0,
        food;
    /* we gebruiken een while omdat we, om een arraymethode
     * te gebruiken, eerst een nieuw array zouden moeten
     * opstellen (met NUMFOODS elementen)
     * Dit is natuurlijk vatbaar voor verbetering!
     */
    while (i < NUMFOODS ) {
        food = createFood(XMIN + getRandomInt(0, xMax),
                          YMIN + getRandomInt(0, yMax));
        if (!food.collidesWithOneOf(snake.segments) &&
            !food.collidesWithOneOf(foods)) {
            foods.push(food);
            i++;
        }
    }
}
```

De methode `collidesWithOneOf` geven we niet. De hulpfunctie `getRandomInt` krijgt u wel mee:

```
/**
 * @function getRandomInt(min: number, max: number) -> number
 * @desc Opleveren van random geheel getal tussen [min, max]
 * @param min een geheel getal als onderste grenswaarde
 * @param max een geheel getal als bovenste grenswaarde (max > min)
 * @return een random geheel getal x waarvoor geldt: min <= x <= max
 */
function getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

stop Het spel kunnen we stoppen met de (te implementeren) functie `stop`:

```
/**
 * @function stop() -> void
 * @desc Laat slang en voedsel verdwijnen, en teken leeg veld
 */
function stop() {
    /* in te vullen */
}
```

draw Van de functie `draw` geven we een beginnetje. Die functie tekent de slang en de voedsелеlementen in het canvas. We gebruiken hier jQuery in plaats van `$`, maar u kunt ook `$` gebruiken. In de functie `draw` kunt u gebruik maken van een hulpfunctie `drawElement`, die we geheel meegeven.

Voor het tekenen maken we gebruik van een plugin voor jQuery: `jCanvas`. U hoeft u niet te verdiepen in de werking van het tekenen.

```
/**
 * @function draw() -> void
 * @desc Teken de slang en het voedsel
 */
function draw() {
    var canvas = jQuery("#mySnakeCanvas").clearCanvas();
    /* in te vullen */
}

/**
 * @function drawElement(element, canvas) -> void
 * @desc Een element tekenen
 * @param element een Element object
 * @param canvas het tekenveld
 */
function drawElement(element, canvas) {
    canvas.drawArc({
        draggable : false,
        fillStyle : element.color,
        x : element.x,
        y : element.y,
        radius : element.radius
    });
}
```

move Ten slotte is de functie `move` gegeven. Die functie heeft een parameter `direction`, die de bewegingsrichting (`LEFT`, `UP`, `RIGHT`, `DOWN`) representeert. De bewegingsrichting geldt vanuit het oogpunt van de speler (dus niet vanuit het perspectief van de slang). De implementatie is als volgt:

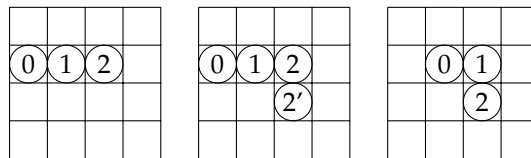
```
/**
 * @function move(direction) -> void
 * @desc Beweeg slang in aangegeven richting
 *       tenzij slang uit canvas zou verdwijnen
 * @param direction de richting (een van de constanten UP, DOWN,
 *       LEFT of RIGHT)
 */
function move(direction) {
    if (snake.canMove(direction)) {
        snake.doMove(direction);
        draw();
    }
    else {
        console.log("snake cannot move " + direction);
    }
}
```

Hier ziet u twee methoden (`canMove` en `doMove`) die u zelf gaat implementeren.

2.2 DE SLANG AUTOMATISCH LATEN BEWEGEN

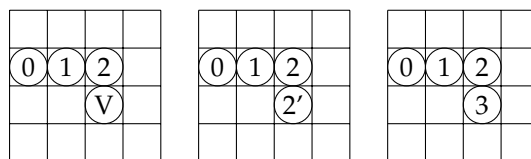
Richting Reageren op de pijltjestoetsen kunt u dan als volgt (we gebruiken hier weer jQuery in plaats van \$):

```
jQuery(document).keydown(function (e) {
  switch (e.which) {
    case 37: // left
      direction = LEFT;
      break;
    case 38: // up
      direction = UP;
      break;
    case 39: // right
      direction = RIGHT;
      break;
    case 40: // down
      direction = DOWN;
      break;
  }
});
```



FIGUUR 1.3 Het bewegen van de slang zonder groei

Bewegen Figuur 1.3 maakt duidelijk hoe we de slang eenvoudig DOWN kunnen laten bewegen. Er wordt een extra segment toegevoegd aan het einde, en het eerste segment verdwijnt. Bovendien krijgt het nieuwe segment de kleur van de kop, terwijl de originele kop verandert in een 'gewoon' segment.



FIGUUR 1.4 Het bewegen van de slang met groei

Figuur 1.4 laat zien wat er gebeurt wanneer de slang DOWN beweegt en voedsel tegenkomt.

Timer Om de slang elke 500 milliseconden automatisch te laten bewegen heeft u een timer nodig. U zet een timer aan met:

```
snakeTimer = setInterval(function() {
  move(); }, SLEEPTIME);
```

U zet hem als volgt weer uit:

```
clearInterval(snakeTimer);
```

Aandachtspunt Bedenk goed wanneer en in welke functie u de timer aan en uitzet.

2.3 WINNEN EN VERLIEZEN

Controleer na elke beweging van de slang of de gebruiker heeft verloren of gewonnen:

- wanneer de nieuwe kop van de slang samenvalt met een van de segmenten van de rest van de slang, heeft de gebruiker verloren,
- wanneer er geen voedsel meer is, heeft de gebruiker gewonnen.

Bij winst of verlies:

- zetten we de slang stil
- geven we een mededeling in de console (of op een mooiere manier in de DOM).

Zet de constante `NUMFOODS` op bijvoorbeeld 5 om gemakkelijker te kunnen testen.

Aandachtspunt Houd dat getal 5 aan, ook wanneer u de uitwerking inlevert (zo kunnen wij ook gemakkelijker testen).

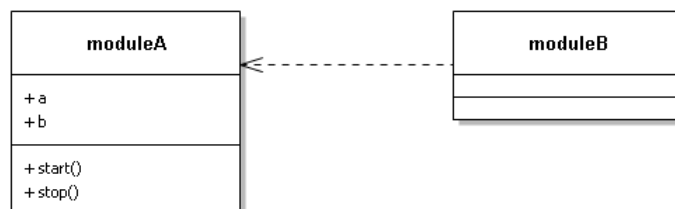
3 Modulen en architectuur

Over het algemeen werkt het het handigst om eerst de applicatie werkend te krijgen, en dan na te gaan denken over modulen. Wanneer u een ervaren programmeur bent, kunt u natuurlijk ook direct alles in modulen onderbrengen.

De architectuur van een applicatie houdt niet alleen in dat u gebruik maakt van modulen, maar ook dat u vastlegt welke module met welke mag communiceren. Probeer er daarbij voor te zorgen dat het eenrichtingverkeer is: de ene module maakt gebruik van de andere, maar niet andersom.

De afhankelijke module moet in de HTML geladen worden *na* de module waarvan de public API nodig is.

Het kan handig zijn om daar een diagram voor te tekenen.



FIGUUR 1.5 Module B is afhankelijk van module A

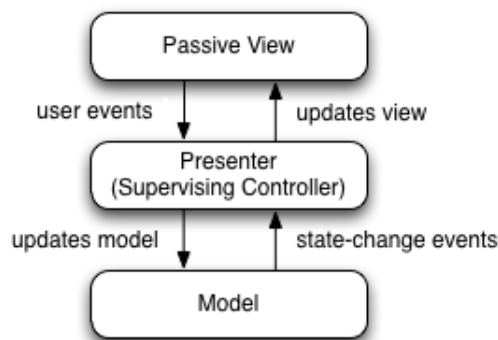
In figuur 1.5 heeft module B (die zelf geen public API heeft) iets nodig van module A (of het om één attribuut of functie gaat of om alles maakt niet uit). Module B moet daarom in de HTML geladen worden *na* module A.

Het kan zijn dat uw code functies bevat die verantwoordelijkheden hebben die bij meerdere modules thuishoren. In zo'n geval moet u zo'n functie splitsen: een functie hoort tenslotte ook slechts één verantwoordelijkheid te hebben.

Minimaliseer, terwijl u alles omvormt naar modules, het aantal globale variabelen. Het liefst zien we dat u geen enkele globale variabele meer nodig heeft.

3.1 MODEL, VIEW, PRESENTER

In applicaties met een user interface wordt vaak een indeling aangehouden op basis van wat model, view en presenter wordt genoemd, zie Figuur 1.6.



FIGUUR 1.6 Model, View, Presenter

Het *model* is het domein. Het is de bedoeling dat het domein geheel *onafhankelijk* is. Het model hoeft dus de rest van de applicatie niet te kennen. Het model kan, indien nodig, events gebruiken om naar de buitenwereld te communiceren. Het model roept dus *niet* direct functies aan van de andere lagen.

De *view* is het zichtbare deel van de applicatie. In het geval van een webapplicatie bestaat de view uit alle code die er voor zorgt dat de DOM wordt aangepast. In dit geval is dat voornamelijk het canvas. De view is *passief*. De view wordt door de presenter gevraagd om de DOM te updaten, en krijgt daarvoor de gegevens mee. De view geeft events van de gebruiker door aan de presenter zonder er zelf iets mee te doen.

De *presenter* zit tussen het domein en de view in. De presenter bekijkt de events die het via de view krijgt, en brengt op basis daarvan veranderingen in het model aan (door het aanroepen van methoden).

Met behulp van de terugkeerwaarden van die methoden *en* (eventueel) met behulp van events die het model afvuurt, besluit de presenter hoe de view moet worden ge-update.

Aanwijzing Het is het handigst om eerst alle functionaliteit aan te brengen, en – zodra het werkt – alles te testen. Pas daarna brengt u de code dan in modules onder: doordat de modules een zo klein mogelijke API hebben, is het veel lastiger om dan nog iets te testen, of om

bugs op te sporen wanneer die zich voordoen.

3.2 EVENTS

Het kan zijn dat u tegen het probleem oploopt dat een module geen gebruik mag maken van methoden van een andere module, volgens de architectuur die u heeft opgesteld, terwijl u dat eigenlijk wel nodig heeft.

U kunt dan communiceren via events. Wanneer module A niet mag communiceren met module B, maar B mag wel communiceren met A, en het blijkt dat A toch iets nodig heeft van B, kunt u in A een nieuw Event-object creëren en direct versturen, op de volgende manier:

```
jQuery(document).trigger(new jQuery.Event("moduleAEventType"));
```

Het type event wordt dus gekarakteriseerd door een string. Uiteraard kunt u elk type event creëren dat u wilt, op die manier.

In module B kunt u op zo'n zelf gecreëerd type event reageren:

```
jQuery(document).on("moduleAEventType", doeIets);
```

Op die manier kunt u vanuit A een methode (`doeIets`) van B aanroepen zonder dat A hoeft te weten welke methode het van B aanroept. Daardoor blijft u uw architectuur trouw.

Aandachtspunt Loop, wanneer u de modules werkend heeft, de code nog eens na op alle ontwerpprincipes.

3.3 TYPESCRIPT

Wanneer u bedacht heeft hoe u de modules wilt inrichten en de verantwoordelijkheden wilt verdelen, kunt u de modules gaan schrijven in TypeScript.

4 De opdracht

4.1 UITGANGSPUNT

U krijgt bij deze opdracht een zipbestand met daarin:

- `uitgangscode.js` met alle gegeven functies en de gegeven documentatie

Verder werkt u met de HTML en de CSS die u samen bent overeengekomen.

4.2 INLEVEREN

Lever, via yOUlearn, een zipbestand in met daarin de volgende directorystructuur:

- de TypeScriptbestanden en `tsconfig.json` in de hoofddirectory,

- een subdirectory `node_modules` (om gemakkelijk TypeScript te kunnen compileren),
- een directory `output` met daarin de gegenereerde JavaScriptbestanden en de HTML en CSS,
- een subdirectory `test` (van de `output-directory`) met de testbestanden
- in de directory `output` ook een HTML-bestand voor het testen,
- een subdirectory `out` van de `output-directory` met de (gegenereerde) documentatie,
- een tekst- of Wordbestand waarin u:
 1. aangeeft wat de verantwoordelijkheden van de modules zijn,
 2. aangeeft hoe de afhankelijkheden van de modules zijn,
 3. uitlegt waarom u voor deze indeling in modules en verantwoordelijkheden heeft gekozen.

Zorg er voor dat `NUMFOODS` op 5 staat, zodat we gemakkelijker kunnen testen.

BEOORDELING

U krijgt voor deze opdracht een gezamenlijk cijfer, dat voor tweederde meetelt bij de beoordeling van deze cursus.

Om te beoordelen hanteren we de rubric die u op de cursuswebsite vindt. U kunt dus uw eigen `snake.html` en `stijl.css` gebruiken (en aanvullen).