Readme

Inhoud

Uitgangspunten voor het programmeren van Snake

We programmeren volgens een functionele stijl daarbij houden we wel rekening met het feit dat JavaScript functionele mogelijkheden heeft, maar niet functioneel is. Hoewel we via een library als immutable en ramda zouden kunnen gebruiken, zonder gebruik te maken van statisch typeren is JavaScript nooit puur functioneel. Dat zou betekenen dat we moeten programmeren in een combinatie van typescript/ramda en dat is een stap te ver.

Wat betekent dan programmeren in de functionele stijl voor ons?

- Functies zijn zelfstandige entiteiten.
- Alles is eigenlijk een functie, een first class citizen.
- Hoewel JS geen abstracte datatypes kent, gebruiken we object literals als ADT. Met het voorgaande in gedachte hebben onze object literals geen gedrag. Oftewel geen methodes.
- Hoewel we geen immutable datastructuren gebruiken behandelen we de gebruikte arrays wel als immutable, door bv slice en concat te gebruiken i.p.v. shift en push.
- We streven totaliteit na in het schrijven van de functies en strooien niet met errors. bv als we een functie schrijven zevenGedeeldDoor dan kan hij niet het functietype int -> int hebben, maar iets als int -> (belofte van een int)
- Er zijn dan ook geen decoratiefuncties als createSegment. Onze voorkeur gaat uit naar één functie waarbij een eventueel verschil als parameter wordt meegeven. BV createElement(r, x, y, color)
- Waar mogelijk maken gebruik van chaining om nieuwe functie te schrijven
- We gebruiken lambda's i.p.v. anonieme functies lambda's hebben geen this en ook geen this conflicten dus.
- We gebruiken recursie i.p.v. iteratoren voor alle functies waarvoor geldt: maximale recursiediepte <10000 (limiet JavaScript/node)

Als een logische conclusie van het bovenstaande hebben wij jullie functie createFoods danig verbouwd. Enkele andere functies hebben wij laten vallen.

Daarmee is Snake 15% gereduceerd in function count, waarvan wij zeggen dat dit ten goede komt aan de onderhoudbaarheid.

Uitgangspunten voor het spel Snake

Het spel Snake is in essentie niet meer dan het manipuleren van twee lijsten van het type Element.

Elementen onderscheiden zich alleen maar via kleur of het elementen van lijst food zijn of van lijst snake.

De lijsten onderscheiden zich alleen maar door dat de lijst snake elementen bevat die aaneengesloten zijn

Een lijst kan ook leeg zijn.

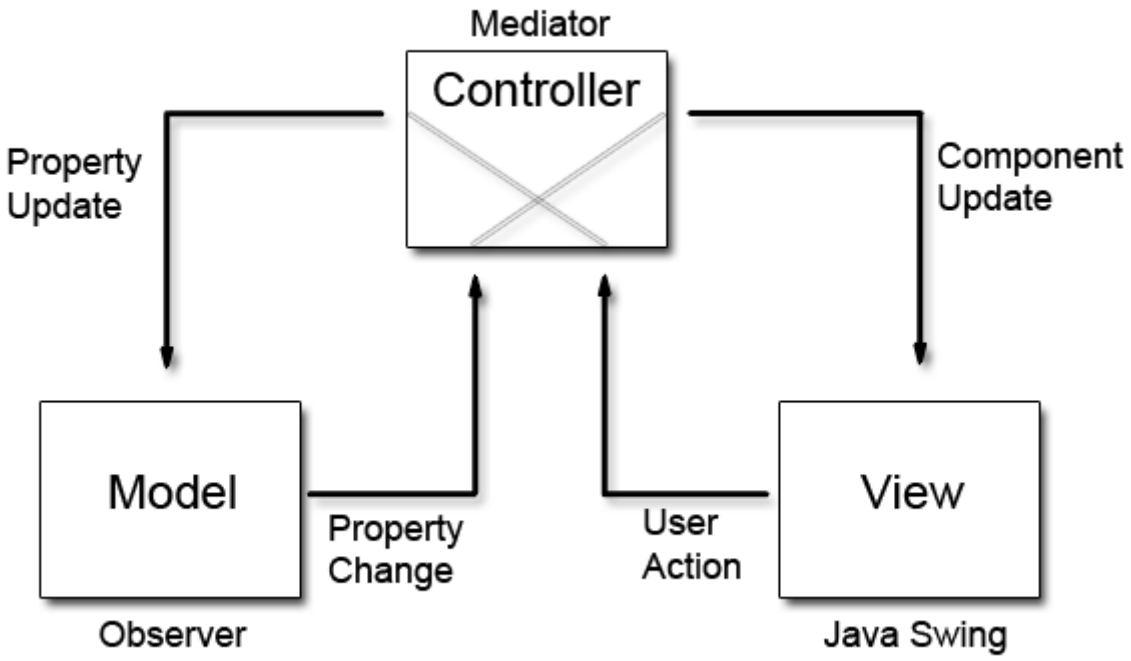
De architectuur

Omdat we programmeren in een functionele stijl maar niet puur functioneel houden we MVC aan. Wel zijn en blijven wij van mening dat het toepassen van het MVC patroon meer gedaan wordt vanuit traditie dan eigen rationele overwegingen. Tevens zijn wij van mening dat OO patronen niet zomaar klakkeloos in een functioneel geschreven programma thuishoren.

We hebben onderzoek naar het SAM patroon gedaan. Het gebruik hiervan als te ingrijpend afgewezen. Wat we wel meenemen is het idee van een view als een pure functie View = Model(Controller(action)) Een concept dat ook in REACT wordt uitgewerkt.

Er is wat discussie mogelijk over wat MVC nu eigenlijk is, een eenduidige definitie (recept) ontbreekt wat ons betreft. Wij gaan uit van Wikipedia (het model update de view, doet dit niet rechtstreeks)

Omdat er redelijk wat discussie mogelijk is over wat MVC nou precies is, gebruiken wij onderstaand model



Er is geen koppeling tussen het model en de view.

Het opslaan van de spel en de spel statistieken zijn door ons ondergebracht in een aparte game module. A priori is Snake maar één van de spellen. Het opslaan van de spellen met dezelfde module bevorderd het hergebruik/generaliteit van de code. Het zou niet moeten uitmaken of we een Snake spel of een schaakspel opslaan.

We hebben dus de volgende opzet in MVC:

Model = model.js

View = snake.html

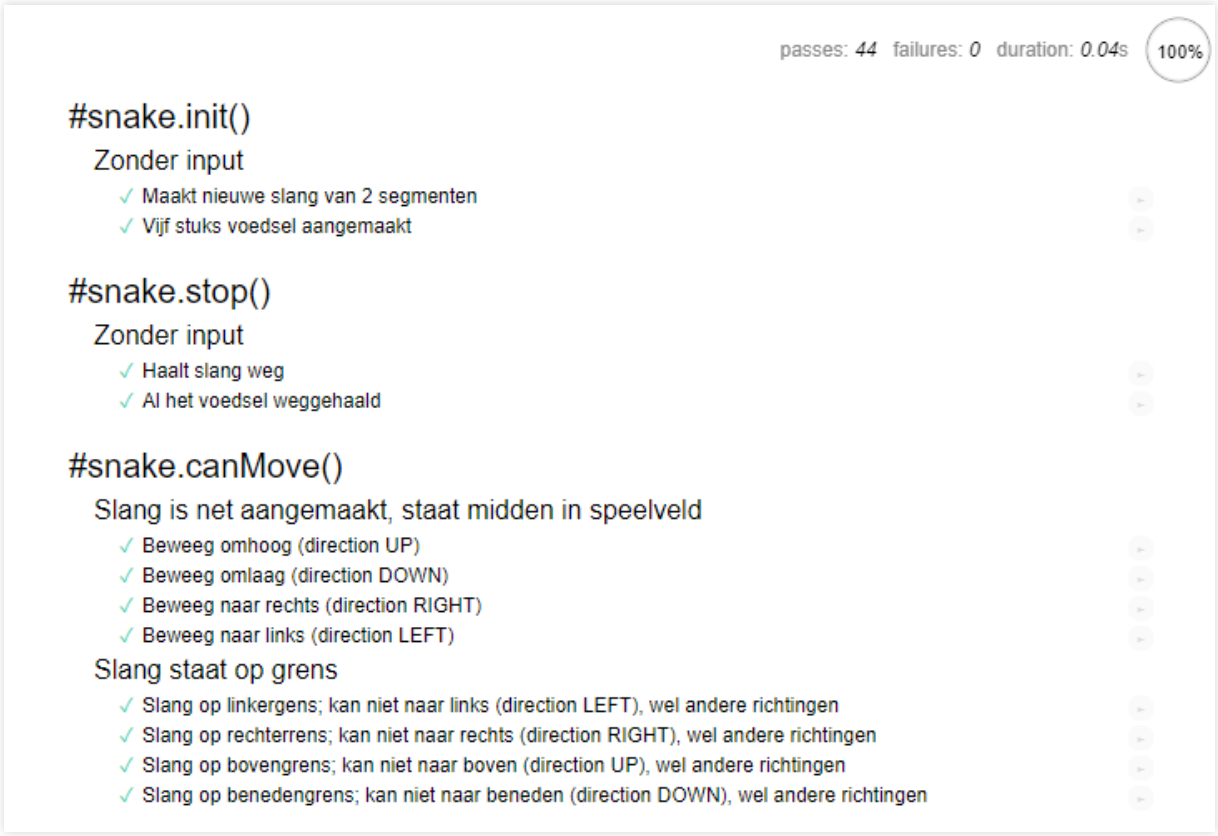
Controller = snake.js

Lost hiervan hebben we nog een game.js – Deze module is “global” zodat we deze opnieuw kunnen gebruiken als we meer games naast snake zouden bouwen.

Testen

Qua tests hebben we geprobeerd zoveel mogelijk de “happyflow” en de edge-cases te bereiken. Helaas hebben we hiervoor een concessie moeten doen: snake.js heeft een aantal meer publieke API’s dan we oorspronkelijk wilden hebben. Dit omdat de tests in snake.test.js anders niet goed konden testen of snake.js zijn werk doet zoals deze het zou moeten doen.

Omdat we besloten hebben chai en mocha te gebruiken heeft de testpagina zijn eigen uiterlijk (zie ook test.html voor deze pagina):



Figuur 1: Screenshot van (een aantal van) de tests

De tests zijn ingedeeld op functionaliteit en maken gebruik van verschillende modules door elkaar heen. Dit omdat het functionele tests zijn, die kijken of inderdaad elke functie het juiste effect heeft en er geen neveneffecten zijn te vinden.

Om te voorkomen dat tests invloed hebben op elkaar begint (nagenoeg) elke test met een init() van de snake-module, waardoor we met “een schone lei” beginnen aan het testen.

Extra onderdelen

Om het spel iets “completer” te maken hebben we een aantal extra onderdelen ingebouwd:

Werkende save-/load-functionaliteit

Naast het kunnen wegschrijven van data naar een fictieve server hebben we besloten ook functionaliteit in te bouwen voor het écht wegschrijven en ophalen van data. Dit gebeurt door middel van localStorage – Hier wordt de data opgeslagen in de browser zelf, waar alleen de gebruiker van die browser op die pagina daadwerkelijk de data kan ophalen en wegschrijven.

Door hiervan gebruik te maken kunnen we de “save game”- en “load game”-knoppen de functionaliteit geven die ze horen te hebben en is het spel dus ook verder te spelen op een later moment.

Veranderen snelheid van spel

Omdat we tijdens het testen opmerkten dat een snelheid van één stap per halve seconde aardig langzaam blijkt hebben we besloten een slider toe te voegen die de snelheid van de slang (het aantal stappen per seconde) aan kan passen.

Het werkt doordat we de waarde van de slider ophalen bij elke arrowKeyMove-call. Hierdoor wordt de snelheid van de slang dus geüpdatet bij de eerstvolgende wijziging van richting van de slang.

Punten en melding van winst/verlies

Na initiële testen merkten we op dat de speler niet genoeg feedback krijgt over het winnen/verliezen van de game. Dit wordt wel in de console log getoond, maar deze staat natuurlijk niet bij eenieder open. Om toch visuele feedback te geven hebben we besloten een deel van de pagina te reserveren voor tekst omtrent winst/verlies van de game.

Confetti bij winst

Om het winnen van een potje snake nog iets vrolijker te maken hebben we besloten om confetti naar beneden te laten dwarrelen. Dit gebeurt slechts voor een paar seconden (kan ingesteld worden) maar maakt het winnen (en spelen) van snake nét dat beetje leuker naar onze mening.

Bijlagen

Een aantal bijlagen zijn meegeleverd. We hebben Google Drive gebruikt voor het onderhouden van een document waarin we onze taken verdeelden, en hebben gekozen voor het gebruik van Github voor het beheren van (versies van) onze code. Beide zaken willen we meeleveren en dus is er gekozen voor het openstellen van de Github-repository en het meeleveren van een extractie van Google Drive.

Ze zijn te vinden onder de namen:

- Github: https://github.com/lausandt/webproject/tree/master/final_ou
- Google Drive: google_drive_extractie.pdf