Read me

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.

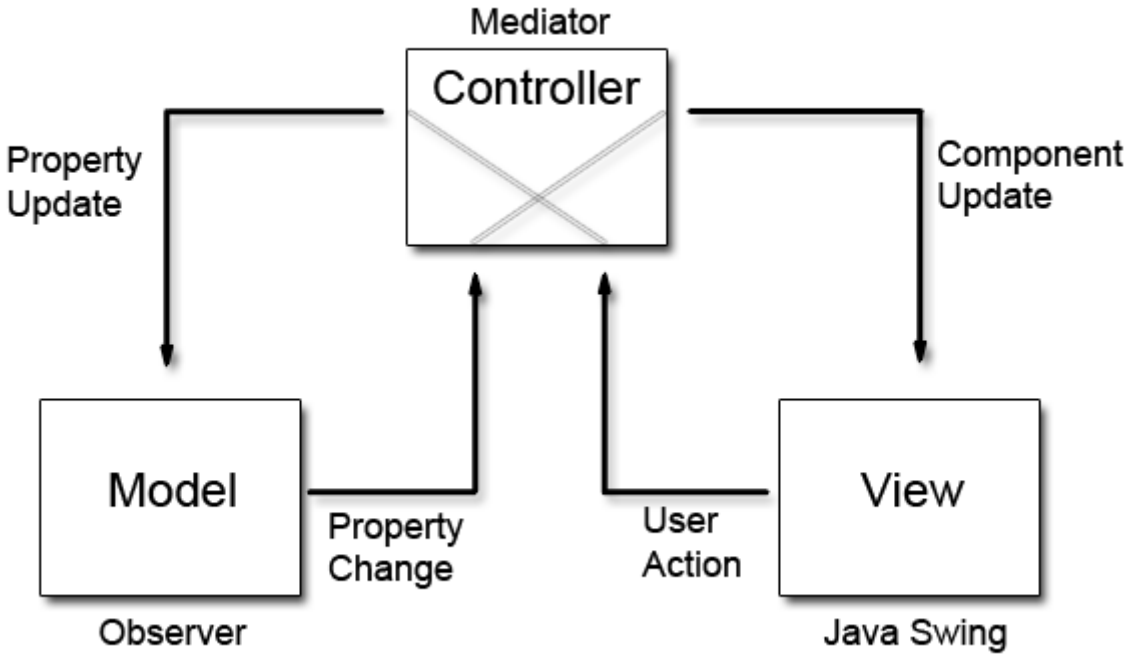
De architectuur:

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 onderstaand model wat wij volgen.



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:

View = snake.html

Controller = snake.js

Model = model.js

En los daarvan nog een game.js

Testen: