Webapplicaties: De clientkant

Laurens Sandt & Michiel van der Winden

Notes

Meeting 10-12-2019

- Door opdracht 2 heen gelopen
- Besproken wie wat gaat doen:
    - **Laurens**:
        - Snake -- Maken van Snake constructor
        - stop -- Maken van methode stop
    - **Michiel**:
        - Element -- Maken van Element constructor
        - init -- Maken van methode init

- Github van Laurens gebruiken -- Michiel moet nog toegang krijgen.
- Laurens neemt deel aan vergadering 18-12-2019
- Volgende meeting: Komen we op een later moment op terug.

Meeting 22-12-2019

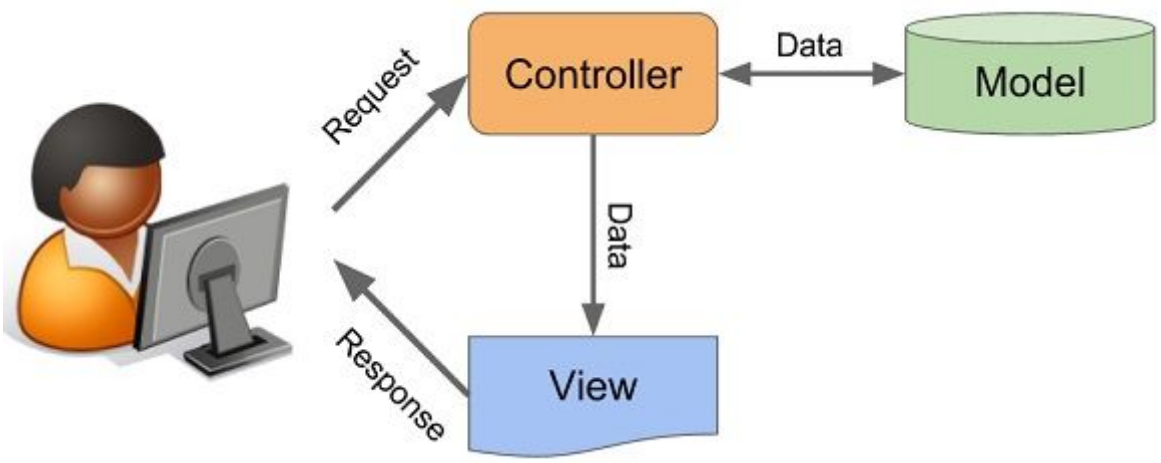
- Rest van de opdracht verdeeld:
    - **Laurens:**
        - doMove -- Maken van de methode doMove
    - **Michiel:**
        - test.html -- Maken van test.html met de JS tests van alle functionaliteit
    - **Allebei:**
        - Documentatie van methods/constructors aanvullen en uiteindelijk JSDoc eruit geneneren

- Gekozen voor het gebruiken van collidesWithOneOf van Laurens i.v.m. gebruik map en some (uit het boek)
- Single line van canMove gemaakt om alle checks in één keer te doen i.p.v. een switch-statement
- Page.js aangemaakt om alle UI-specifieke dingen naar een aparte JS-file te brengen
- Volgende meeting: 29-12-2019

Meeting 12-01-2020

- Eerst de functionaliteit inbouwen, dan pas na volgende meeting refactoren naar MVC
- Opdracht verdeeld:
    - **Laurens:**
        - Automatisch bewegen en sturen van slang via pijltjesknoppen bouwen
        - Zelf snelheid van slang kunnen aanpassen met een slider boven het spel
        - Documentatie bijwerken
    - **Michiel:**
        - State-checks bij gewonnen/verloren inbouwen -- Tegen slang aanbotsen en alle voedseldeeltjes opgegeten
        - Inbouwen save/load-functionaliteit aan de hand van cookies naast de “serverside opslag” die we moeten mocken
        - Tests aanpassen naar nieuwe functionaliteit

- MVC-model:



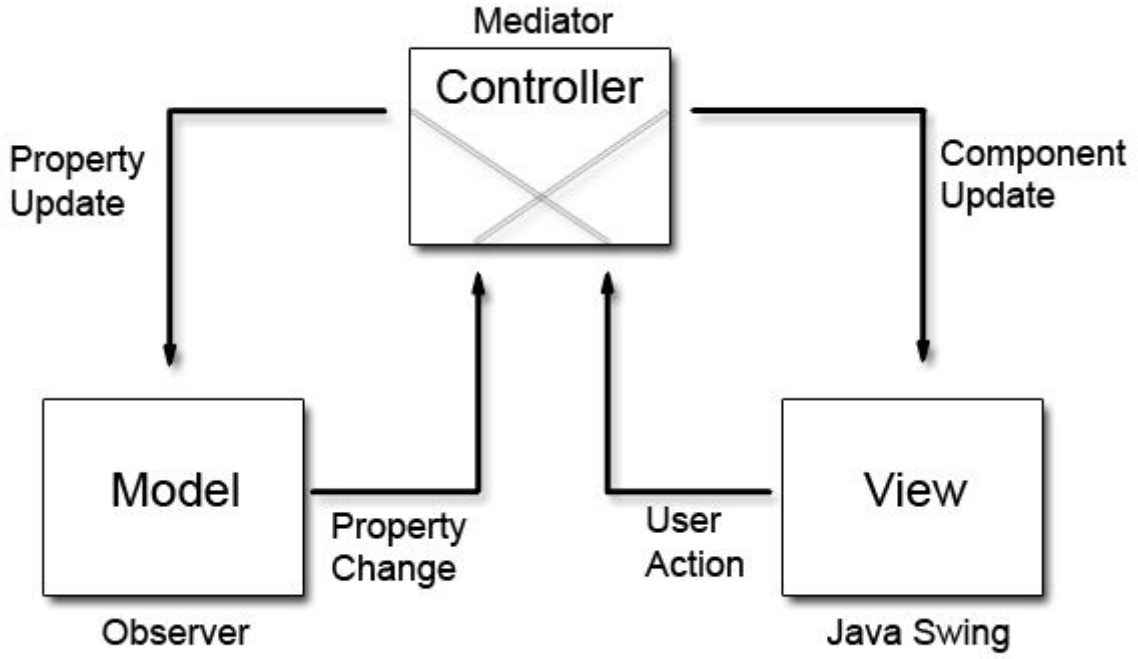
- Nieuwe meeting: **19-01-2020 10:00**

Meeting 19-01-2020

- We gaan verder met MVC-methode zoals in de opdracht afgesproken
- Wel willen we een aantal zaken (o.a. **collidesWithOneOf**) functioneel maken om een goede mix van OO en functioneel programmeren te hebben
- Constanten in aparte file -- Dit om deze eerst in te laden en netjes gescheiden te hebben (**properties.js** met getter/setter voor width/height en andere functies die niet per sé een constante moeten zijn)
- Readme -- Hierin moet staan waarom we bepaalde keuzes hebben gemaakt:
    - Waarom hebben we dingen uit de hulpfuncties aangepast? (we hebben b.v. zaken van constanten naar getters gehaald, en de **collidesWithOneOf** aangepast: Hier moeten we zowel uitleg als motivatie voor schrijven)
    - Waarom hebben we gekozen voor deze opzet?
    - Hoe hebben we het MVC-model toegepast?
    - Hoe hebben we onze publieke APIs opgezet?
- Tests nalopen op dat ze nog steeds werken
- Documentatie oppoetsen -- Opnieuw JSDoc laten generen
- Werkverdeling:
    - **Laurens:**
        - Functionele code samenvoegen met bestaande basis
        - Tests herschrijven om daaraan te voldoen
    - **Michiel:**
        - Properties-file voor constanten e.d.
        - Tests herschrijven om daaraan te voldoen
    - **Allebei:**
        - Mail bespreken MVC-model (**Uiterlijk 22-01-2020**)
        - Door documentatie eigen delen lopen en nalopen of dit nog klopt
        - Aanpak bedenken voor game-object naar functionele methoden om te bouwen
        - MVC-model opbouwen -- Uiterlijk vanaf woensdag 22-01-2020
- Volgende meeting: 26-01-2020 10:00

Meeting 26-01-2020

- MVC-model besproken:
    - Model is niets anders dan de ADT's zoals Snake of de GWL voor game-logica
    - Controller communiceert tussen view en model
    - View is niets meer dan de HTML
- Verdeling werk:
    - **Laurens:**
        - Ombouwen code naar MVC volgens bovenstaande punten uit overleg
        - Visuele feedback win/loss van game
    - **Michiel:**
        - Aanpassen tests aan de hand van MVC
        - Confetti bij winst
- Volgende meeting: 29-01-2020 19:00
-



Programmeeraanwijzingen OU:

Stappen plan OU schrijven JavaScript Applicatie:

1. Ontwerp de publieke interface. Schrijf de signaturen van de functie en documenteer deze.

2. Schrijf testen voor de publieke functies

3. Ontwerp en ontwikkel de code die dit intern waarmaakt, met losse variabelen, objecten en functies in de globale name-space

4. Test de applicatie, specifiek de methoden die niet via de testomgeving te benaderen zijn.

5. Refactor de applicatie met de publieke interface die u bedacht heeft.

7 Modulen en objecten

1. Gebruik de dot operator, tenzij je een variabele als propertynaam wil gebruiken

2. Constructoren met een hoofdletter

3. Gebruik constructoren wanneer u meerdere objecten nodig hebt, bij eenmalig (singleton) gebruik maak een literal

4. Geef in een object geen waarde aan een property die ook een waarde van en prototype property is.

5. Gebruik geen lambda's in een constructor.

6. Neem in het prototype alleen constanten voor deze klasse op. Neem andere properties, ook properties met een default-waarde op in de constructor. (Default eigenschappen in de constructor.)

7. Implementeer methoden in het prototype niet de constructor.

8. Geef een literal object als waarde aan het prototype wanneer u meer dan één property of methode aan het prototype wilt toevoegen.

9. Gebruik een functie om een object mee te geven als prototype om daarvandaan een object te creëren als je iets aan het prototype wilt toevoegen of een object zonder parameters wil creëren.

8 JQuery en de DOM

Gebruik object chaining ipv losse statements. BV $ (“#tasklist”).children().filter(“.important”)