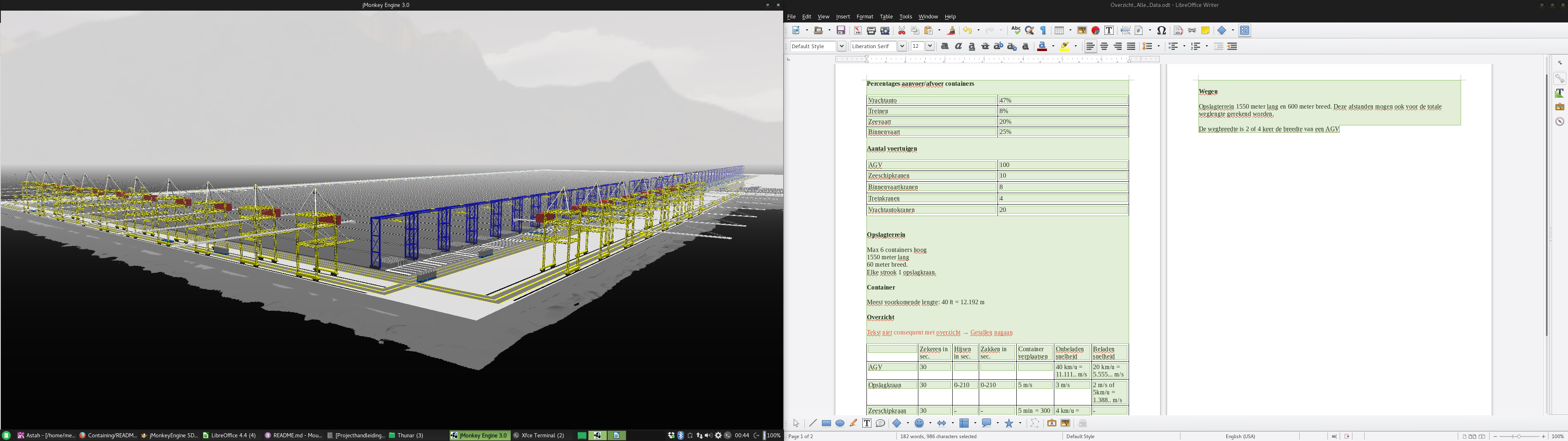
Containing



Door:

Erwin Hager

Melvin Kool

Eelke Fierstra

Jeffery Verbeek

Klaas Bakker

Datum: 05-02-2016

Inhoudsopgave

[Introductie 3](#__RefHeading___Toc1352_1895385619)

[Analyse 4](#__RefHeading___Toc1370_1895385619)

[Gemaakte afspraken 6](#__RefHeading___Toc1354_1895385619)

[Scrum 6](#__RefHeading___Toc1356_1895385619)

[Testen 15](#__RefHeading___Toc1358_1895385619)

[Handleiding 15](#__RefHeading___Toc1360_1895385619)

[Reflectie 15](#__RefHeading___Toc1362_1895385619)

[Samenvatting en evaluatie 16](#__RefHeading___Toc1364_1895385619)

## **Introductie**

Project containing is een project van het tweede jaar Informatica van de NHL. Het doel van project containing is om een simulatie te maken van de Rotterdamse haven. Deze simulatie bestaat uit een app, controller en een 3D weergave. We hebben hier 8 weken de tijd voor gekregen.

In dit verslag zullen we jullie vertellen wat dit project precies inhoud, de techniek en algoritmes die er achter zitten, hoe het project verliep en een korte nabeschouwing. We hopen dat we jullie mee kunnen nemen in ons project.

Ons project kun je vinden op GitHub: <https://github.com/MelvinKool/Containing.git>

## **Samenvatting van de opdracht**

Zoals verteld in de introductie is de opdracht om een simulatie te maken met een 3d weergave, een app en een controller. Deze programma's moeten in Java en C++ geschreven worden. Ook moeten deze programma's getest worden, zo dat de opdrachtgever overtuigd raakt van de kwaliteit.

**Simulator**

De simulator moet gemaakt worden in 3D, waarbij gebruik moet worden gemaakt van de game-engine jMonkey.

**Controller**

De controller moet XML-bestanden verwerken die we aangeleverd krijgen. Deze bestanden bevatten informatie over containers, zoals aankomst- en vertrektijden, inhoud van de container enz. In deze XML-bestanden kunnen fouten zitten, waar wij op moeten controleren, zodat ons programma er geen last van heeft.

**App**

De app moet gemaakt worden voor een mobile platform (Android of iOS). In deze app moeten actuele aantallen containers per categorie getoond worden, dus waar ze zich op dat moment bevinden.

## **Analyse**

Van tevoren hebben we veel analyse gedaan. Als eerste hebben we gekeken naar wat de opdrachtgever precies wil. Dit hebben we omgezet naar getallen en UML. We hebben ook gekeken naar een paar resultaten van dit project van de voorafgaande jaren en wat er beter kan. Vervolgens hebben we dingen overwogen en vastgelegd wat we wel en niet gaan doen. Alle kleine onderdelen van het project hebben we op het scrumboard van trello.com geplaatst.

**Getallen en statistieken**

**Percentages aanvoer/afvoer containers**

|  |  |
| --- | --- |
| Vrachtauto | 47% |
| Treinen | 8% |
| Zeevaart | 20% |
| Binnenvaart | 25% |

**Aantal voertuigen**

|  |  |
| --- | --- |
| AGV | 100 |
| Zeeschipkranen | 10 |
| Binnenvaartkranen | 8 |
| Treinkranen | 4 |
| Vrachtautokranen | 20 |

**Opslagterrein**

Max 6 containers hoog

1550 meter lang

60 meter breed.

Elke strook 1 opslagkraan.

**Container**

Meest voorkomende lengte: 40 ft = 12.192 m

**Overzicht**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Zekeren in sec. | Hijsen in sec. | Zakken in sec. | Container verplaatsen | Onbeladen snelheid | Beladen snelheid |
| AGV | 30 |  |  |  | 40 km/u = 11.111.. m/s | 20 km/u = 5.555... m/s |
| Opslagkraan | 30 | 0-210 | 0-210 | 5 m/s | 3 m/s | 2 m/s of 5km/u = 1.388.. m/s |
| Zeeschipkraan | 30 | - | - | 5 min = 300 sec. | 4 km/u = 1.111.. m/s | - |
| Treinkraan | 30 | 60-120 | 30-60 | 5 m/s | 3 m/s | 2m/s |
| Binnenvaartkraan | 30 | 0-210 | 0-210 | 5 m/s | 8 km/u = 2.222... m/s | 3 km/u = 0.8333... m/s |
| Vrachtautokraan | 30 | 60 | 60 | - | 20km/u = 5.555.. m/s | 15 km/u = 4.1666...m/s |

**Wegen**

Opslagterrein 1550 meter lang en 600 meter breed. Deze afstanden mogen ook voor de totale weglengte gerekend worden.

De wegbreedte is 2 of 4 keer de breedte van een AGV

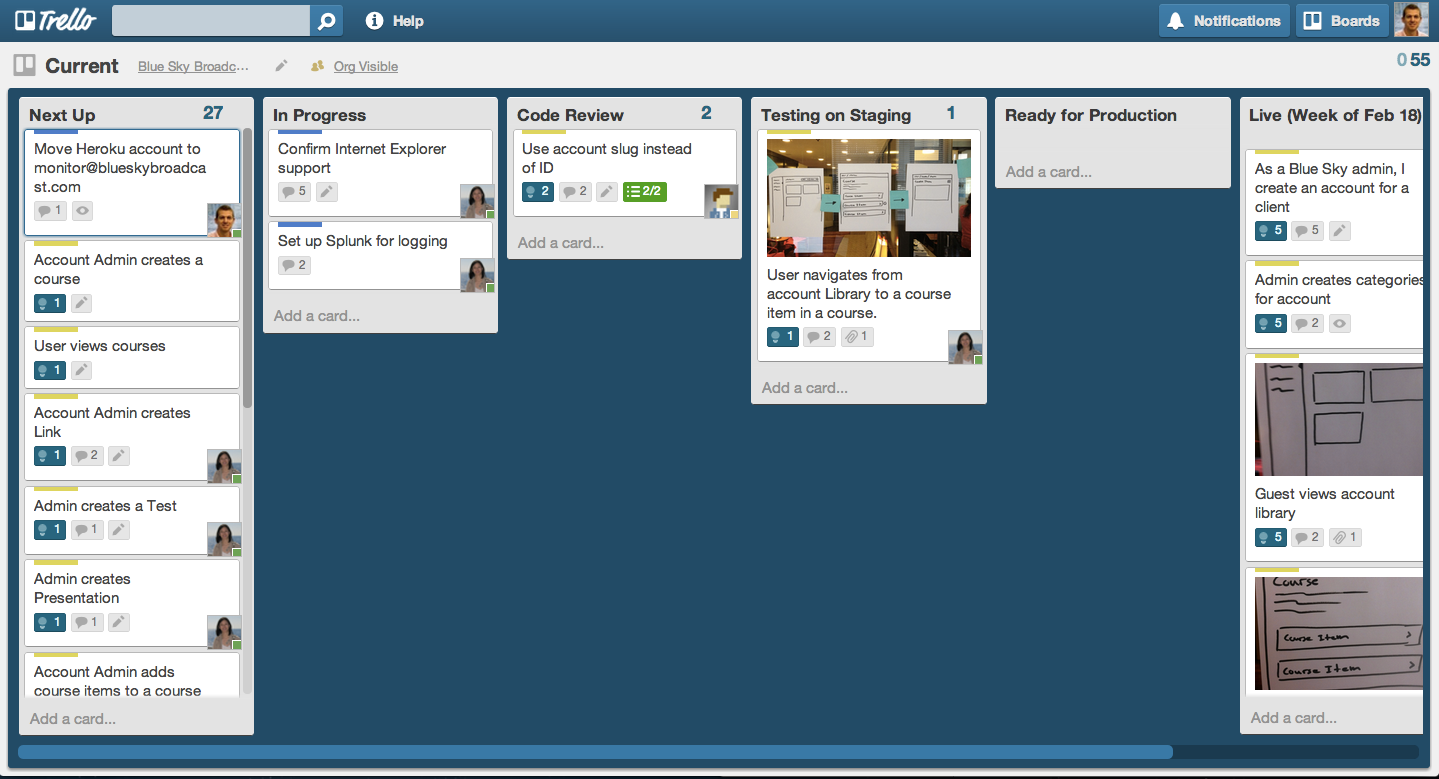
**UML**

Vervolgens hebben we UML van de controller(server), de simulator en de app gemaakt. Helaas is de UML te groot om te laten zien in het verslag, dus zit deze in de bijlage(Docs/UML/).

# Gemaakte afspraken

**Scrum**

Gedurende het project hebben we gebruikt gemaakt van scrum. Dagelijks hebben we een scrum gesprek gehouden waarin we even samenvatten wat we gedaan hadden en de komende dag gaan doen. Via Trello hebben we onze backlog en sprints bijgehouden. De backlog hebben we opgesteld aan de hand van de eisen van de opdrachtgever. In de backlog stonden de eisen met de hoogste prioriteit boven aan en de minst belangrijke zo laag mogelijk. Aan het begin hadden we sprints van twee weken. Later hebben we dat veranderd naar een week aangezien dat fijner werkte. Aan het einde van de sprint hebben we de backlog bijgewerkt en de nieuwe sprint opgesteld. Bij het niet halen van de sprint hebben we overlegt waarom het niet was gelukt en hoe we dit gingen op lossen.



*Voorbeeld van trello*

**Simulator**

De simulator willen wij zo 'dom' mogelijk houden. Dit betekend dat deze alleen de huidige stand van zaken afbeeld. De simulator ontvangt commando's en gegevens via de server(controller). Een simulator kan elk moment verbinden met de server. De simulator is geschreven in java, met als game-engine jMonkey.

**Controller**

In onze controller stoppen we de server. Dit is het makkelijkst, omdat in ons geval alle logica in de controller zit en dan meteen de simulator kan aansturen. De controller is geschreven in c++. Deze bestaat uit verschillende parsers en schrijvers, een webserver, een server voor het versturen en ontvangen van data en vooral de logica die onder andere samenwerkt met onze database.

**App**

De app maken wij in HTML, CSS en JavaScript. Dit is een responsive webapp, die vriendelijk is voor alle platforms en alle devices. Hierdoor maakt het niet uit of je Linux of iOS gebruikt, of je een groot of een klein scherm hebt, en ook bijna niet welke webbrowser je gebruikt. Onze app laat live statistieken zien met behulp van AJAX en ChartJS. De app haalt al deze gegevens op van de webserver die in onze controller verwerkt zit.

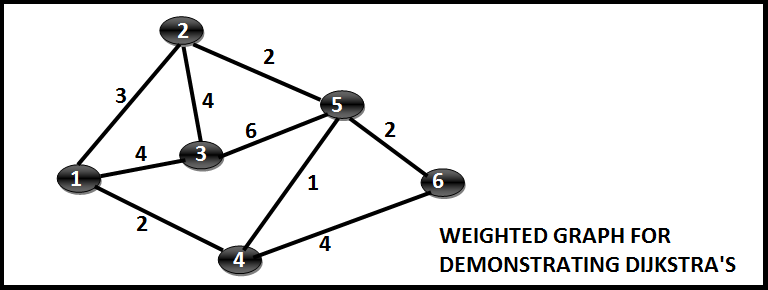
**Kortste pad**

In onze server voeren wij alle kortstepad berekeningen uit. Voor ons kortstepad algoritme gebruiken wij het algoritme van Dijkstra. Met dit algoritme reken je de kortste afstand uit tussen twee punten. Hier hebben we gekozen, omdat ons wegenstelsel vooral bestaat uit knooppunten, waar het algoritme van dijkstra zeer geschikt voor is! Dit levert je dus een snel algoritme op, wat door veel voertuigen tegelijk gebruikt kan worden.

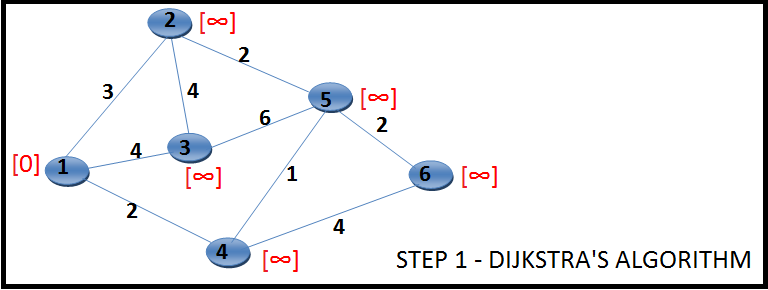
Hier een uitleg hoe het algoritme van dijkstra ongeveer werkt en hoe dit algoritme bij ons gebruikt wordt:

Om het algoritme van Dijkstra schematisch weer te geven zie je hier een voorstelling van een wegenstelsel. Je hebt hier punt 1,2 .. t/m 6. De afstand tussen punt 1 en punt 2 is bijvoorbeeld 3 en de afstand tussen punt 1 en punt 3 is 4.

Laten we als voorbeeld nemen de kortste route tussen punt 1 en punt 6.

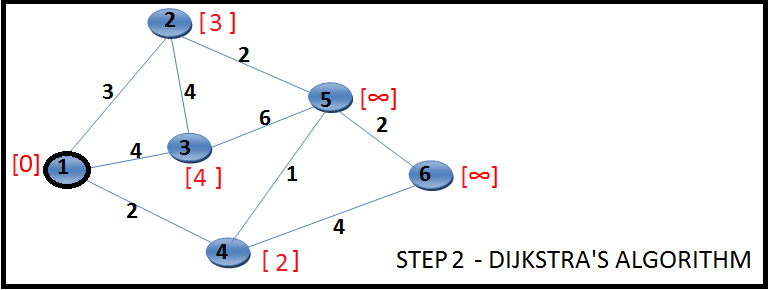
Bron afbeelding: [http://techieme.in/shortest-path-using-dijkstras-algorithm/http://techieme.in/shortest-path-using-dijkstras-algorithm/](http://techieme.in/shortest-path-using-dijkstras-algorithm/)

Als eerste kies je je startpunt, in ons geval punt 1. De afstand naar punt 1 kun je op 0 zetten, aangezien dit ons startpunt is. Punt 1 is dan klaar, want je weet de kortste route naar punt 1. Hier kun je dan bijvoorbeeld een cirkel omheen tekenen, wat betekent dat dit punt klaar is. Vervolgens zet je de afstanden naar alle punten op oneindig, aangezien de route van punt 1 naar punt 2 langer had kunnen zijn dan de route van punt 1 naar punt 2 via punt 3.



Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Vervolgens bereken je de afstand van alle aanliggende punten vanaf het punt met de kleinste afstand (als je begint punt 1 met de afstand van 0). De afstand tussen punt 1 en punt 2 is dan de kleinste afstand naar het huidige punt(punt 1 met afstand 0) 0 + de afstand naar het punt(punt 2) 3 = 3. De voorganger zet je dan op je huidige punt(punt 1) als de afstand naar het punt dat we bekijken(punt 2, afstand = 3) kleiner is vanaf het huidige punt dan de eerder berekende afstand naar het punt(afstand = oneindig), dit is later nodig voor het herleiden van de route vanaf het eindpunt(punt 6). De afstand tussen punt 1 en punt 3 is 4, de voorganger is punt 1. De afstand tussen punt 1 en punt 4 is 2, de voorganger is hier ook weer punt 1.

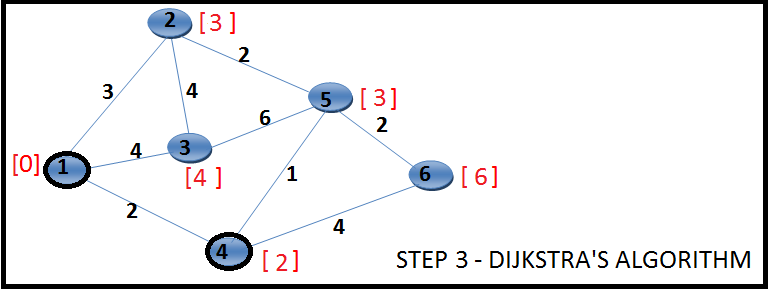
Bron afbeelding: [http://techieme.in/shortest-path-using-dijkstras-algorithm/http://techieme.in/shortest-path-using-dijkstras-algorithm/](http://techieme.in/shortest-path-using-dijkstras-algorithm/)

Vanaf 1

Vanaf 1

Vanaf 1

Vervolgens kijk je welk punt de kleinste totale afstand heeft vanaf het beginpunt en nog niet klaar is, dat is in ons geval punt 4 met een totale afstand van 2. Dit punt is dan klaar, dus zet je er een cirkel omheen. Dit punt wordt dan je huidige punt. Hierna ga je de vorige stappen steeds herhalen. We bekijken eerst weer de afstanden naar de andere punten(punt 5 en punt 6). Deze afstanden zijn kleiner dan oneindig, dus stellen we de voorganger in op het huidige punt(punt 4). Hierna zoeken we het punt met de kleinste totale afstand uit en stellen dit punt in als ons huidige punt. Je ziet nu dat er twee mogelijkheden zijn, punt 2 of punt 5. In ons geval kiezen we punt 2.

Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Vanaf 4

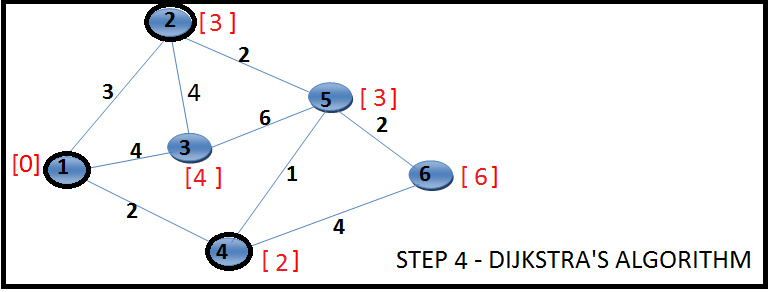
Vanaf 4

Vanaf 1

Vanaf 1

Vanaf 1

Punt 2 is nu klaar. De afstand van punt 2 (3) naar punt 5 (3 + 2 = 5) is groter dan 3, dus stellen we hier geen nieuwe voorganger in. Met punt 3 hetzelfde.



Vanaf 1

Vanaf 1

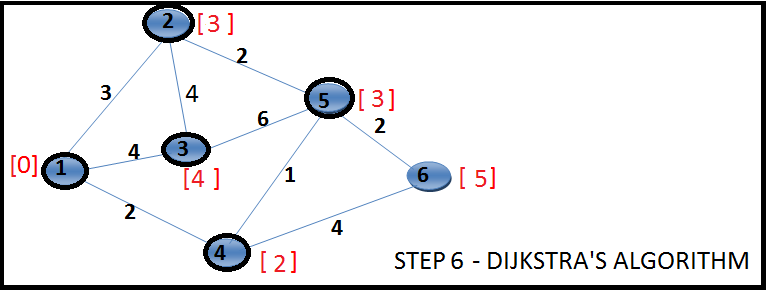
Vanaf 1

Vanaf 4

Vanaf 4

Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/> <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Hierna zoeken we het punt dat nog niet klaar is en de kleinste totale afstand heeft. Dit is punt 5. Punt 5 is dan ook klaar. De afstand van punt 5 naar punt 3(3 + 6 = 9) is langer dan de huidige afstand(4), dus de voorganger veranderen we niet. De afstand van punt 5 naar punt 6(3 + 2 = 5) is korter dan de huidige afstand(6), dus we zetten de voorganger van punt 6 op punt 5 en de afstand op 5.

Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Vanaf 1

Vanaf 1

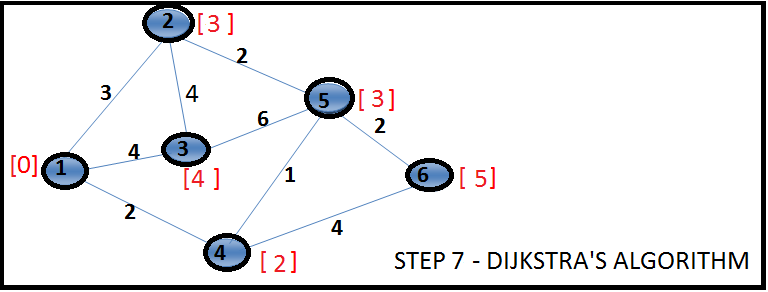
Vanaf 1

Vanaf 4

Vanaf 5

Nu zoeken we weer het punt met de kortste afstand die nog niet klaar is, dat is punt 3. Punt 3 zetten we op de huidige en die is klaar. Punt 3 heeft geen aanliggende punten die nog niet klaar zijn, dus kunnen we verder.

Nu zoeken we het kleinste punt dat nog niet klaar is en de kleinste totale afstand heeft.

Dit is punt 6, dus stellen we punt 6 in als ons huidige punt. Punt 6 is klaar, we hebben ons doel nu bereikt.

Bron afbeelding: <http://techieme.in/shortest-path-using-dijkstras-algorithm/>

Vanaf 1

Vanaf 4

Vanaf 5

Vanaf 1

Vanaf 1

We herleiden nu de kortste route. We zijn naar 6 gekomen vanaf 5 → 5,6. We zijn naar 5 gekomen via 4 → 4,5,6. We zijn naar 4 gekomen vanaf 1 → 1,4,5,6. We zijn nu bij ons beginpunt, dus hebben we de kortste route gevonden: 1,4,5,6.

Hier een voorbeeld in pseudocode hoe je dit zou kunnen programmeren:

|  |  |  |
| --- | --- | --- |
| 1: | **function** Dijkstra(Graph, source): |  |
| 2: | **for each** vertex v in Graph: | // Initialization |
| 3: | dist[v] := infinity | // initial distance from source to vertex v is set to infinite |
| 4: | previous[v] := undefined | // Previous node in optimal path from source |
| 5: | dist[source] := 0 | // Distance from source to source |
| 6: | Q := the set of all nodes in Graph | // all nodes in the graph are unoptimized - thus are in Q |
| 7: | **while** Q **is not** empty: | // main loop |
| 8: | u := node in Q with smallest dist[ ] |  |
| 9: | remove u from Q |  |
| 10: | **for each** neighbor v of u: | // where v has not yet been removed from Q. |
| 11: | alt := dist[u] + dist\_between(u, v) |  |
| 12: | **if** alt < dist[v] | // Relax (u,v) |
| 13: | dist[v] := alt |  |
| 14: | previous[v] := u |  |
| 15: | **return** previous[ ] |  |

Bron: [http://www.gitta.info/Accessibiliti/en/html/Dijkstra\_learningObject1.htmlhttp://www.gitta.info/Accessibiliti/en/html/Dijkstra\_learningObject1.html](http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html)

## Testen

Gedurende het project hebben we heel veel unit testen gedaan. Verder hebben we integratietesten gedaan als dit onderdeel door de test is gekomen. Op git hebben we een branch develop, de master branch en verder alle onderdeel branches waar mensen momenteel op aan het werken zijn. Als een onderdeel door de unit test en integratietest komt, kan dit onderdeel toegevoegt worden aan de develop branch. De develop branch is de meest recent werkende versie. Zo nu en dan voeren we een grotere test uit. Komt de develop branch door deze test heen, dan komt deze in de master branch. De master branch is onze nieuwst stabiele versie. De development branch wordt iedere wekelijkse sprint toegevoegd aan de master branch.

*De belangrijkste testen hebben we opgenomen in ons testverslag die in de bijlage zit.*

## Handleiding

Om onze simulatie te draaien moet als eerste de controller(server) opgestart zijn. Vervolgens kun je een simulator opstarten om de 3d weergave hier van te zien. Verder kun je onze app bekijken door de app te downloaden of door het ip adres van de server in te typen met poort 4000 erachter. Voorbeeld: localhost:4000.

*In README.md zit een gedetailleerde handleiding.*

## Reflectie

Aan het begin van het project hebben we besloten om zoveel mogelijk op de NHL te werken. Naast school hebben we contact gehouden via een groepsapp. Er waren geen problemen in de groep. Soms was de communicatie niet helemaal geweldig. Hierdoor wist niet iedereen precies waar degene mee bezig was wat leidde tot verwarring.

# Samenvatting en evaluatie

Het doel van het project was om een werkende simulatie te maken van een haven. Hierbij moet XML worden ingeladen. Uit deze XML moeten de voertuigen worden aangestuurd. De agv’s moeten de containers bij de schepen, trucks, en treinen weghalen. De containers worden via kranen geladen en gelost. De containers worden opgeslagen op het opslagterrein. Deze moeten op datum worden gesorteerd.

Tijdens Containing liepen we tegen enkele problemen aan. Deze konden opgelost worden door de hulp van een “expert” of medestudent. Ook door het gebruik van Google kon je meestal wel ver komen. Het grootste probleem was echter het managen van het project. Door het niet goed aanpakken van de backlog en deze niet goed te verdelen konden we helaas niet binnen acht weken klaar zijn. Maar door uiteindelijk te luisteren naar onze tutor en alles in veel kleinere stukjes te splitsen is het ons toch nog gelukt een goed eindproduct te leveren na de twee weken die we extra kregen.

Bijna iedere week hebben we een gesprek met onze tutor/opdrachtgever(Wouter) gehad. Tijdens het gesprek besproken we de voortgang, de sprints, de backlog, problemen in de groep en de samenwerking.

Tijdens het project heeft iedereen geleerd om, om te gaan met Git. Daarnaast hebben we natuurlijk ook geleerd qua programmeren. Voor de een is dit meer op het gebied van Java geweest en voor de ander C++. Dit omdat ieder zijn eigen taak had en niet iedereen overal mee bezig was. Projectmanagement is ook een grote leer geweest voor iedereen. Vooral taakverdeling en samenwerken aan een project heeft ons veel bijgebracht.

Al met al zijn we blij dat het project nu voorbij is. Ook zijn we erg blij dat we uiteindelijk toch een semi-werkend eindproduct hebben. Helaas was het project iets te groot voor ons om het binnen de acht weken af te krijgen. In de laatste twee weken hebben we eigenlijk de meeste vooruitsprong geboekt. Verder is het teamwork vrij goed gegaan en hebben we veel plezier gehad ook al werkte JMonkey niet altijd helemaal mee.