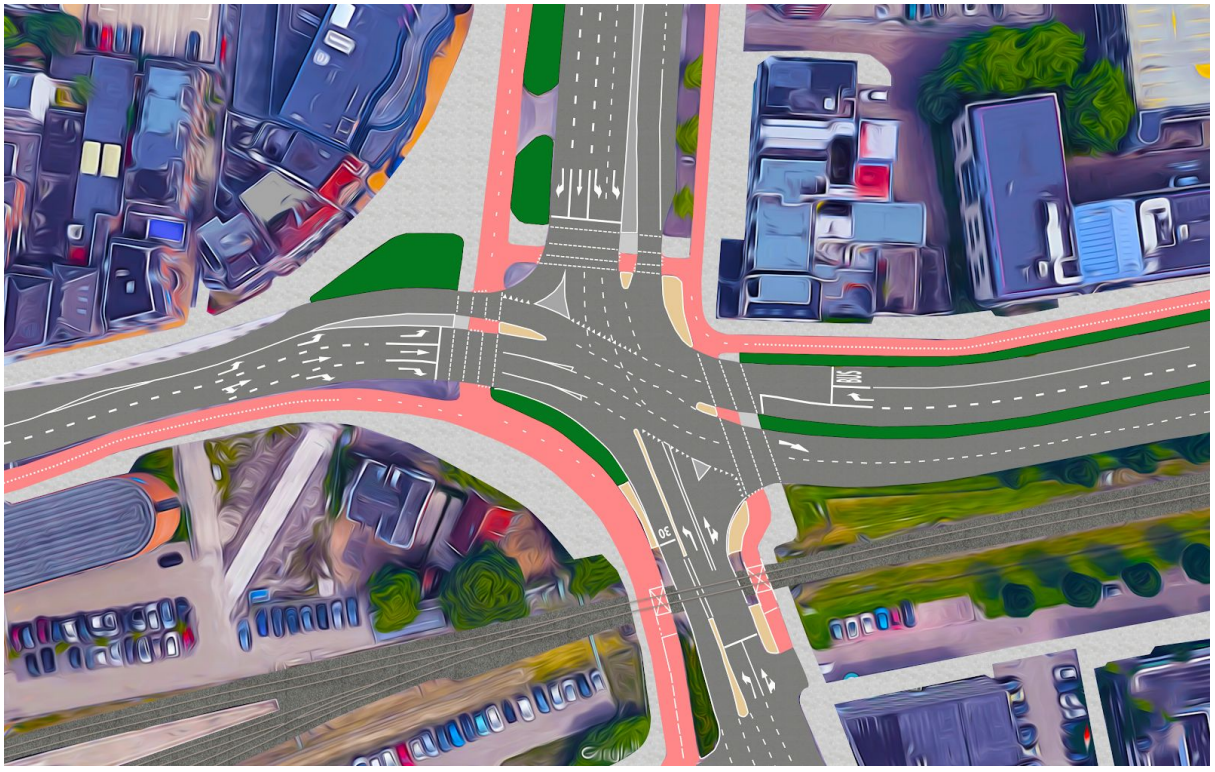
Verslag Team Rood



Vak: Software Development (SDM)

Jaar: 2018/2019

Periode: 1 en 2

Programmeertalen: Python (Controller) en JavaScript (Simulator)

Teamleden:

- Marten Hoekstra Hoekstra (hoek1613)
- Dion Velberg (velb1600)
- Koen Lukkien (lukk1600)

Inhoudsopgave

Inleiding

De opdracht die we kregen was het maken van een verkeerssimulatie per tweetal bestaande uit een slimme controller (server) en een domme simulator (cliënt). De bedoeling is dus de de simulator de verkeerssituatie weergeeft en dat de controller de verkeerssituatie beheert. Deze controller moet er voor zorgen dat er verkeerslichten op groen, oranje en rood worden gezet, er treinen kunnen passeren, voetgangers en fietsers kunnen oversteken, bussen op de busbaan voorrang krijgen en dat alles zonder ongelukken te veroorzaken.
Dit eerdergenoemde was niet het lastige aan deze opdracht, het addertje onder het gras was dat de simulator en tevens de controller van elk tweetal moest kunnen samenwerken met de simulator en controller van alle andere tweetallen. Om deze eis succesvol uit te voeren was een duidelijk en goed doorgesproken protocol vereist. Hier werd in de eerste weken van het semester veel aandacht aan besteed.

Planning

Een echte planning hebben wij als groep dit semester niet aangehouden. We spraken vaak (de dag van te voren) af hoe laat we op school samen komen en wat er die dag gedaan moest worden. Dit had voornamelijk betrekking op SDM (Software Development), soms kon dit tevens betrekking hebben op een ander vak.
We hadden tijdens het ‘project’ van SDM niet een bepaalde systematische werkmethode zoals scrum met daily stand-ups. Dit was meer iets wat we gedurende de dag samen bespraken en konden bijstellen, we werkten daarom altijd samen op school en niet apart van elkaar thuis bijvoorbeeld.
Dit voorkomt veel misverstanden en daar hebben we dan ook weinig tot geen last van gehad als groep. Bij onduidelijkheden werd er gewoon ter plekke overlegt en de onduidelijkheden werden dan ook opgehelderd.

Plan van aanpak

De eis van de opdrachtgever is het maken van een verkeerssimulatie bestaande uit een ‘domme’ simulator en een ‘slimme’ controller, de simulator moet kunnen samenwerken met de controllers van alle andere groepen en tevens moet de controller kunnen samenwerken met de simulatoren van alle andere groepen. Hierbij komt dus een hele hoop overleg bij kijken met als resultaat dat we een duidelijk universeel communicatie protocol hebben waar alle controllers en simulatoren zich aan houden.

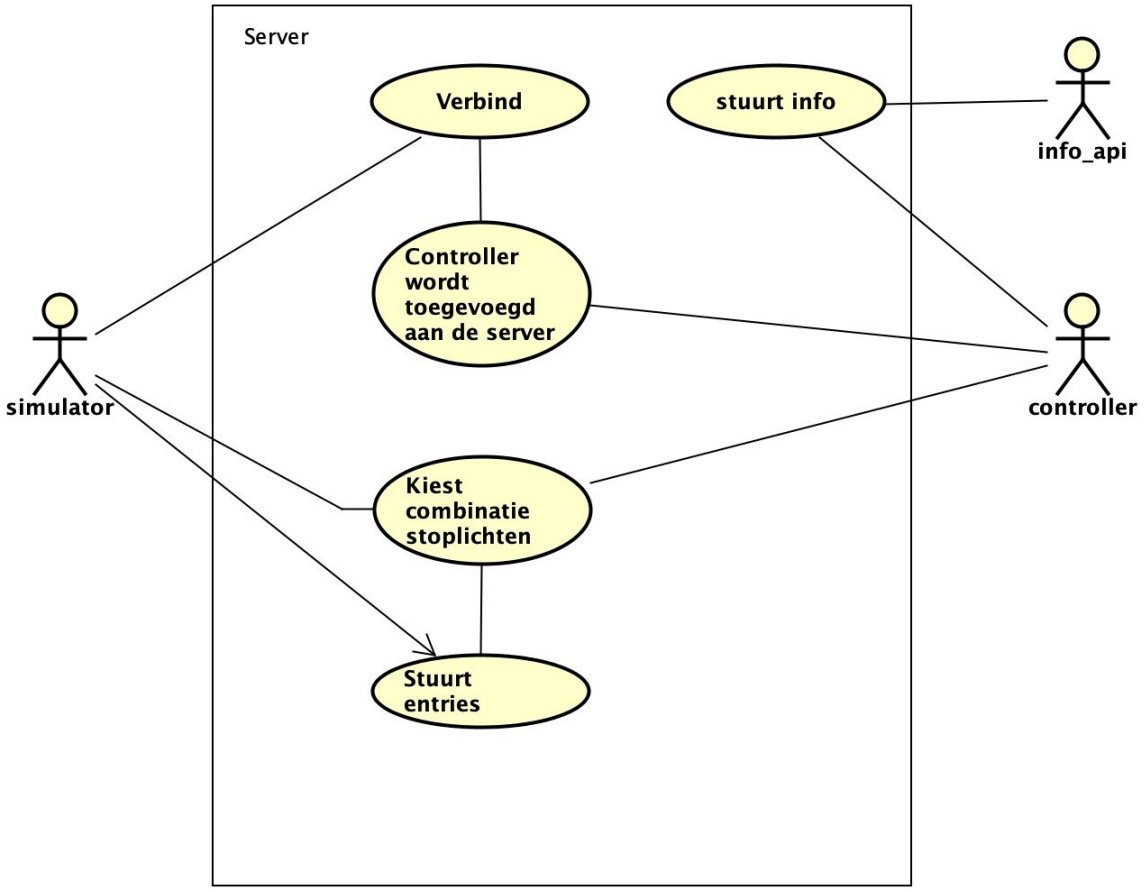
Aangezien Dion later bij onze groep is gekomen is het plan vanaf toen wat aangepast, wat natuurlijk begrijpelijk is. Ons eerste plan was dat we gezamenlijk, in eerste instantie dus Marten en Koen, aan de simulator en controller zouden werken. Dus beide personen aan beide onderdelen. Dit is veranderd toen Dion bij de groep kwam.
Hierop hebben we besloten dat Dion en Koen samen aan de (JavaScript) simulator en Marten aan de (Python) controller gaan/gaat werken.
Voor de simulator in JavaScript wordt gebruik gemaakt van Phaser, een JavaScript framework voor het ontwikkelen van (2D) games voor HTML5 (Canvas en WebGL).
Voor de controller wordt verder gebruik gemaakt van een library voor het ontvangen van Websocket verbindingen.

Om als samenwerkende groepen grip te houden op het project is de afgelopen weken elke week één keer een testsessie gehouden. Zo zijn we samen tegen een aantal onvoorziene bugs aan gelopen die we gezamenlijk tijdig hebben kunnen aankaarten en op kunnen lossen.

UML analyse

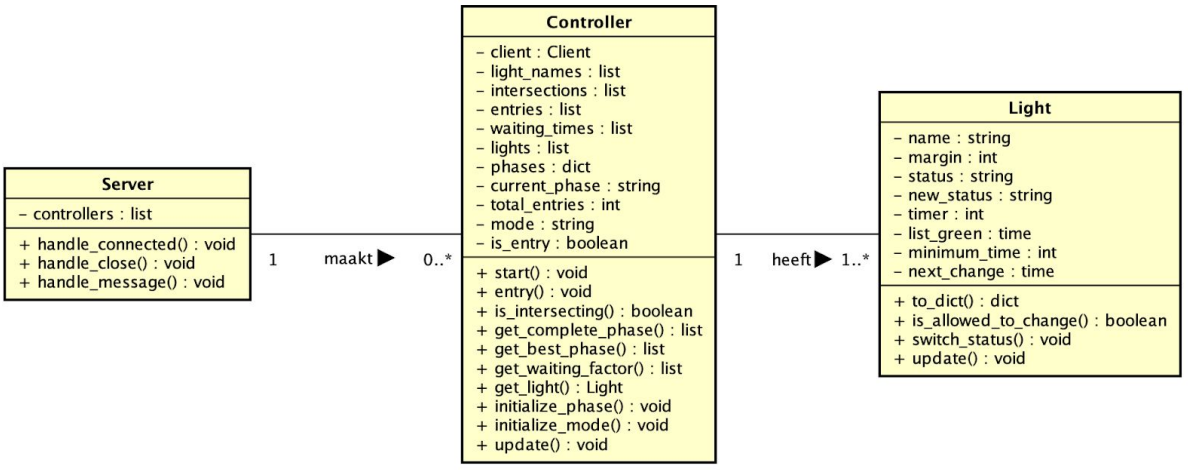
Use case diagram

De use case diagram van de controller laat zien hoe de verschillende onderdelen van de controller ontstaan bij het verbinden van simulatoren en de info server.



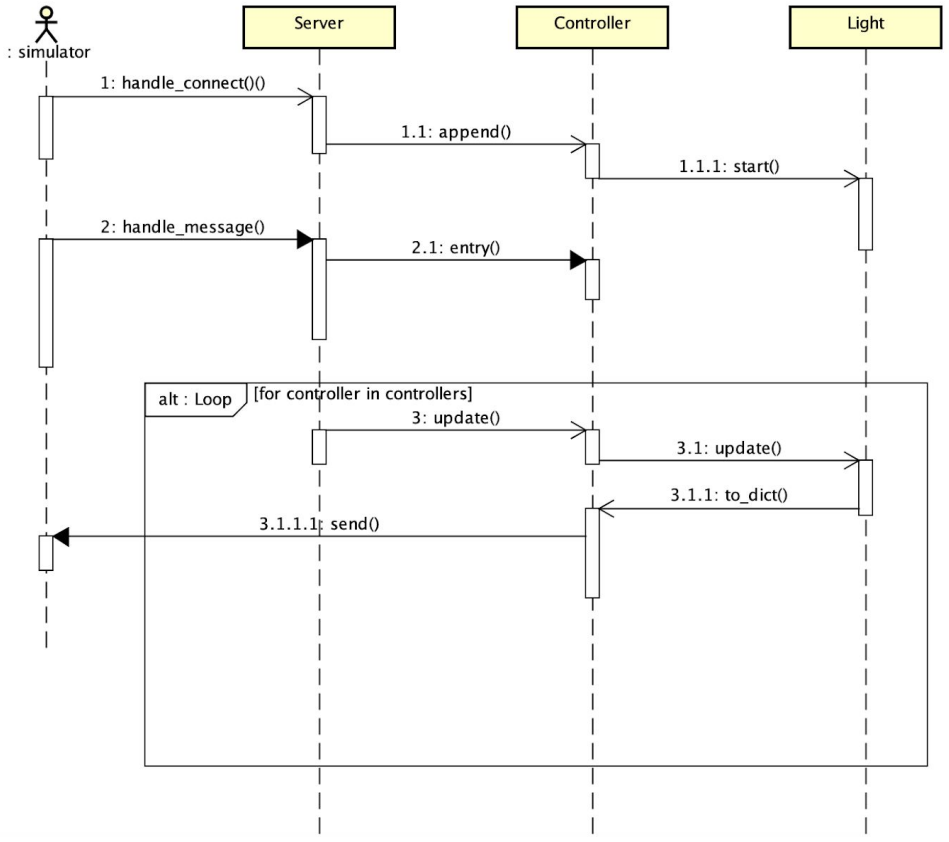
Klassendiagram Python Controller

Overzicht van de python server en controller met bijbehorende properties en methods.

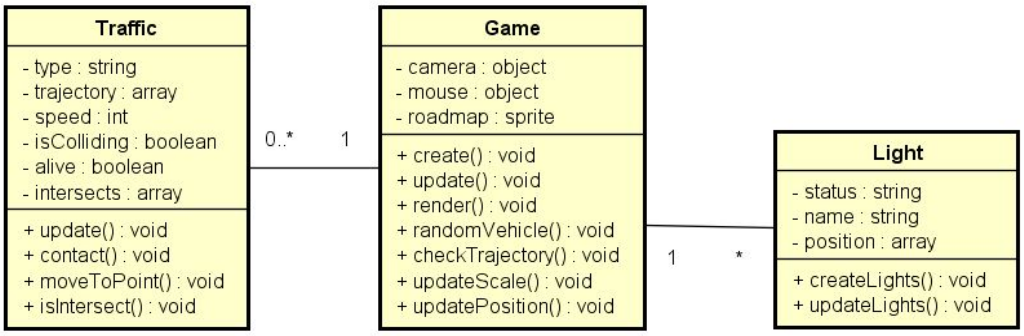


Sequence diagram Controller

De sequence diagram geeft weer in welke volgorde de communicatie weer



Klassendiagram JavaScript simulator



Protocol

Het protocol, een cruciaal onderdeel om deze opdracht als grote groep Software Developers succesvol af te ronden. Dit houd in: de communicatie over de Websocket verbinding van én naar de simulator en controller. Alle deelnemende groepen (tweetallen) moeten dit uiteraard exact hetzelfde doen. Hiervoor is een protocol groep opgericht in de eerste weken van het semester. Deze protocol groep heeft het volgende JSON communicatie protocol opgesteld:

Vanaf de simulator naar de controller

De simulator is uiteraard niet helemaal 'dom', deze moet een klein beetje informatie naar de controller sturen. De controller moet weten waar op het kruispunt verkeersdeelnemers staan te wachten (voor welke verkeerslichten). Dit gebeurd door de zo gedefinieerde 'drukplaten'. Zodra een verkeersdeelnemer voor een verkeerslicht komt te staan, dus op een drukplaat stuurt hij zijn aanmelding naar de controller, de controller kan vervolgens de verkeerslichten prioriteren en deze bepaald dan uiteindelijk ook welke verkeerslichten welke kleur krijgen. De communicatie van de simulator naar de controller wordt gedaan met de volgende JSON:

```
[
    "A1",
    "C3.2",
    "E1"
]
```

Vanaf de controller naar de simulator

De 'slimme' controller regelt de hele verkeerssituatie naar de informatie die hij krijgt van de simulator, de controller gaat vervolgens aan de gang met die informatie en stuurt terug naar de simulator hoe de verkeerssituatie er uit moet komen te zien, dus welke kleuren welke verkeerslichten krijgen. De simulator dient daarop de verkeersdeelnemers aan te passen: verkeersdeelnemers te laten stoppen voor een verkeerslicht.
De communicatie van de controller naar de simulator wordt gedaan met de volgende JSON:

```
[
   {
     "light": "A11",
     "status": "green",
     "timer": 0
   },
   {
     "light": "A12",
     "status": "orange",
     "timer": 0
   },
   {
     "light": "A13",
     "status": "red",
     "timer": 20
   }
]
```

Test plan

Testperiode

We hebben 4 weken in totaal om samen met andere groepen onze producten onderling te gaan testen. Voor elke week hebben we samen met de andere groepen doelen afgesproken.

Week 1

Voor de eerste week is het doel dat je tenminste verbinding kan maken met andere simulators/controller en zelf ook met andere simulators/controllers kan verbinden.

Week 2

Het doel van de tweede week is gericht op het versturen en ontvangen.
Voor je eigen simulatie is de bedoeling dat hij steeds entries verstuurt naar een controller van een andere groep. Als de controller de entries goed heeft ontvangen maakt hij phases en stuurt hij een lijst terug met daarin de verkeerslicht waarden. De simulatie moet hierbij alleen de verkeersdeelnemers laten bewegen als ze groen hebben.
Voor je eigen controller is de bedoeling dat hij de entries van de simulator van een andere groep goed kan ontvangen en lezen. Aan de hand van de entries moet je controller de beste phases creëren en een lijst met daarin verkeerslicht waarden terug sturen naar de simulator.
Bij het testen van je eigen simulatie/controller gaan we ook na of alle waarden tussen de simulaties en controller gelijk zijn. Niet dat de simulatie een entry geeft van A7 maar dat de controller dit leest als A8 of zelfs A8 groen zet.

Week 3

De derde week van testen is gericht op het controleren en verbeteren van de phases.
Het testen van de phases wordt er op gelet dat de phases die zijn gemaakt kloppen. Phases die niet goed zijn kunnen botsingen veroorzaken, onnodig rood of slechte doorstroom.
Om je eigen phases te verbeteren is het handig om een schema te maken en zo bij te houden welke verkeerslichten dus wel of niet tegelijk op groen kunnen gaan.

Week 4

De laatste week van testen is eigenlijk alleen je eigen controller en simulatie compleet testen met andere groepen.

Testresultaat

Week 1

We hebben met meerdere groepen onze simulator en controller kunnen testen. Beide producten kwamen met succes door de testen. De simulator kon verbinden met alle controllers en onze controller verbond met alle simulators.

Week 2

In week 2 konden we helaas maar met 3 groepen testen maar gaf ons wel genoeg feedback. Onze simulator kon goed entries doorsturen en werden ook goed gelezen door de controller. Wanneer de controller phases had gemaakt en een lijst met verkeerslichten terug stuurde reageerde de simulator correct en kregen de juiste verkeerslichten groen of rood.
Onze controller kreeg alle entries binnen van de simulators en kon ze ook goed doorlezen. De controller creëerde phases en stuurde ze succesvol weer terug naar de simulators.
Alle waarden die wij verstuurden of ontvingen naar de andere groepen kwamen allemaal overeen.

Week 3

Bij het testen van onze phases in deze week kwamen we eigenlijk weinig fouten tegen. Alleen bij één test kwamen we tegen dat er een auto verkeerslicht samen op groen ging met een fietsverkeerslicht terwijl dat een botsing zou veroorzaken.

Week 4

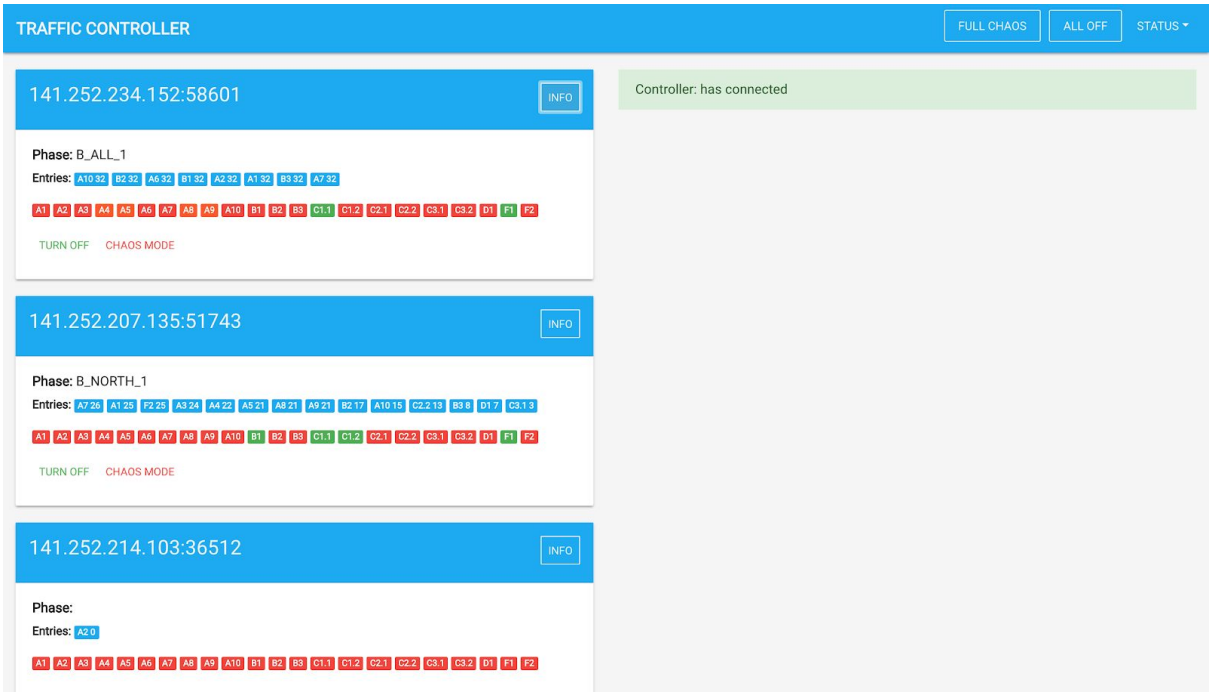
In de laatste week van het testen ging alles soepel en kwamen we niks verkeerds tegen. We alle feedback van de vorige weken goed gebruikt en kwamen op een goed en compleet product.

Controller

Onze controller bestaat uit twee onderdelen: een websocket server die behoud welke simulatoren zijn aangesloten en een info REST API. Bij de REST API hoort een React.js pagina die info laat zien over aangesloten simulatoren.
De server is geprogrammeerd in Python 3. De REST API in Node.js javascript en lopen naast elkaar continu op een dedicated server. Doordat de REST API via lokaal IP verbindt weet de Websocket server dat het gaat om de info server in plaats van een simulator.

Info API

In het overzicht van de info API is te zien welke entries staan te wachten (en hoe lang), de status van de verkeerslichten, ip adres van de simulatoren en de phase (huidige combinatie verkeerslichten). Daarnaast is het mogelijk om een speciale “modus” voor de controller in te stellen.



Afbeelding: Meerde simulatoren zijn verbonden met de server.

Python controller

De Python server houd bij welke simulatoren zijn aangesloten en koppelt die aan een Controller object. In het controller object wordt er op basis van de gestuurde entries bepaald welke Lights aan en uit gaan.

Lights

De controller heeft een constante lijst met verkeerslichten met daarbij behorende marges en minimum tijden. Hiermee wordt een lijst gevuld met Light objecten die afzonderlijk van elkaar worden geupdate.

Entries

De entries worden in de Server ontvangen en dan toegevoegd aan de juiste Controller. Daarna worden deze aan een lijst met entries toegevoegd met de tijd wanneer ze zijn toegevoegd.

Phases

De Phases zijn combinaties van verkeerslicht namen die samen op groen kunnen. Een voorbeeld van een Phase:
"A_NORTH": ["A1", "A2", "A3", "D1"]
De Phases worden bij het opstarten van de controller aangevuld met lichten die in combinatie met de al toegevoegde lichten op groen kunnen. Hierbij wordt gebruikt gemaakt van een tabel met doorkruisende verkeerslichten.
In de update functie van de controller wordt voor elke Phase een aantal punten berekend op basis overeenkomende entries, wachttijd van diezelfde entries en of er een speciaal voertuig staat te wachten. (entries met een trein of bus krijgen meer punten). De Phase met de meeste punten wordt ingezet wanneer alle verkeerslichten in de vorige Phase mogen veranderen.

Modus

De controller kan op verschillende modi worden ingesteld. Standaard staat de modus op “normal”.

Drie modi

- “off”: alle verkeerslichten op rood.
- “chaos”: alle verkeerslichten op groen.
- “modus”: verkeerslichten worden aangestuurd op basis entries.

Simulator

De simulator is een webapplicatie weergeven in 2D met als bovenaanzicht een verkeerskruispunt. Om dit te realiseren hebben wij Javascript (ES6, NPM, WebPack en ESLint) met het 2D JavaScript Framework 'Phaser' gebruikt.

Trajectories

De banen in de simulator hebben verschillende trajecten. Een traject is een array van x en y waardes die samen het pad vormen die de autos volgen. Naast de X en Y waarde van het pad kan er ook een licht naam voorkomen in de array. Op dit punt wordt er op de kaart een verkeerslicht getekend waar autos voor kunnen stoppen. Meerdere verkeerslichten in een traject zijn daardoor ook mogelijk.

Traffic

Het Traffic object is een verkeersdeelnemer die een traject volgt. In het traffic object wordt gecheckt of de auto in de buurt kom van een verkeerslicht. Daarnaast wordt in het Traffic object gecheckt of een auto moet stoppen voor andere auto’s of in het geval van een doorkruising botst met andere verkeersdeelnemers met als gevolg de daarbij behorende explosies.

Game

Het game object zorgt voor de game logica van de simulator. Hier worden de kaart, verkeerslichten en verkeersdeelnemers aan de simulator toegevoegd.

Communicatie

De simulator heeft constante communicatie met de controller en zorgt er voor dat er een goede doorstroming wordt gesimuleerd. De simulator stuurt steeds de entries van de verkeerslichten en de controller antwoord hierop met een lijst van verkeerslichten met hun statussen.

Entries

De entries worden verstuurd zodra een verkeersdeelnemer zich aanmeldt bij de drukplaten die zich bevinden vlak voor de verkeerslichten. Elke entry bevat simpelweg alleen de naam van het verkeerslicht. Bijv. “A1”.