VRIJE UNIVERSITEIT BRUSSEL

SPECIFICATIE PROGRAMMEERPROJECT 2

Fase 3

JANNES DE METS
0549193
JANNES.DE.METS@VUB.BE
BA COMPUTERWETENSCHAPPEN
Juni 2021

Wetenschappen en Bio-ingenieurswetenschappen

1. Inleiding

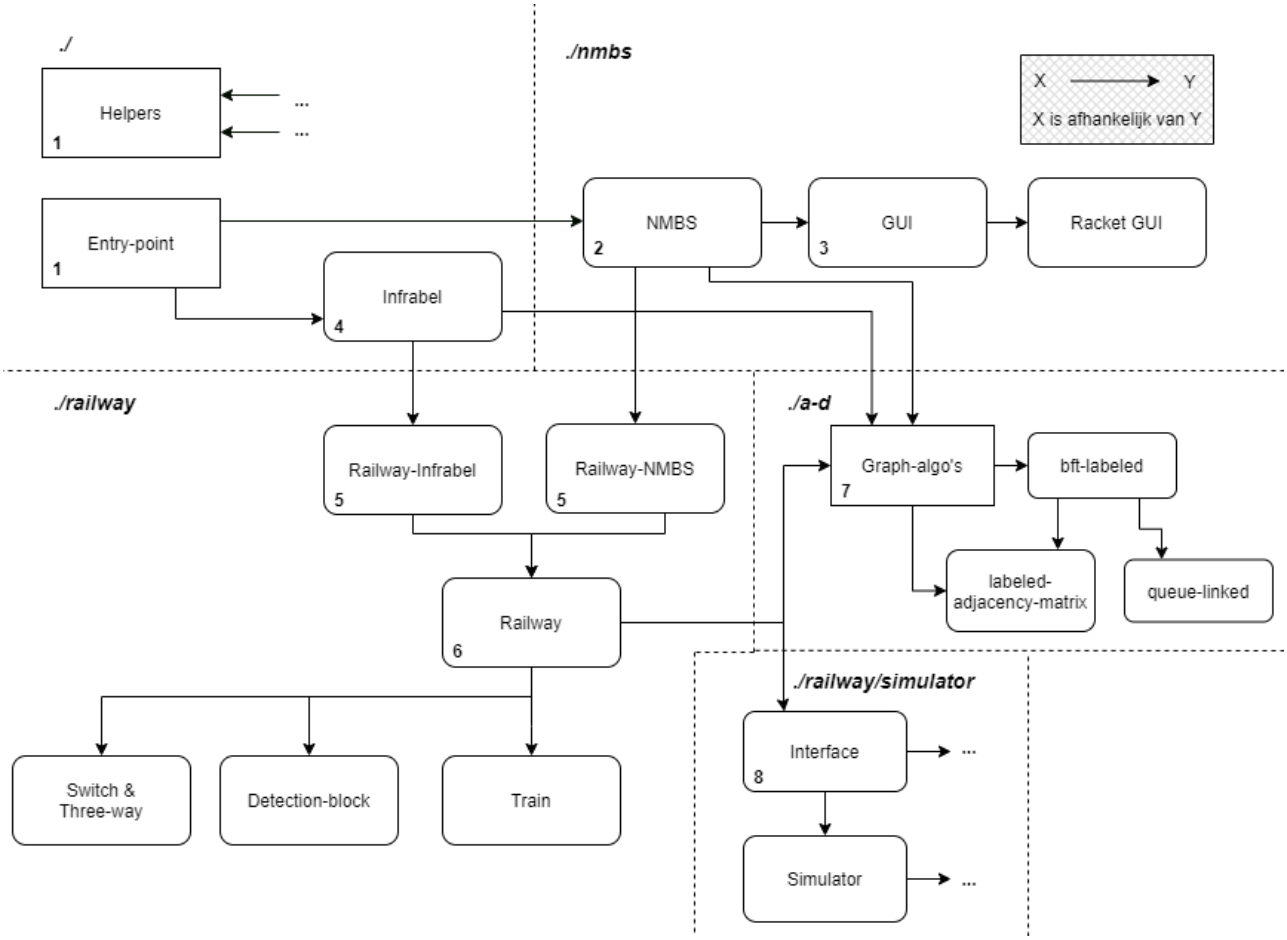
Dit is de specificatie voor het programmeerproject 2. Deze bevat informatie over alle functionaliteiten van het programmeerproject zoals de beschrijving van alle ADTs en de relaties ertussen.

2. Algemene beschrijving

Voor programmeerproject 2 is er een softwaretoepassing gemaakt die een controlesysteem realiseert om modeltreinen te besturen. Dit systeem wordt opgedeeld in 2 lagen. De eerste heet Infrabel, deze staat in voor communicatie tussen de software en de hardware. De zogenaamde *command & control*, de regeling van doorlopend verkeer, is ook de taak van Infrabel. Dit wordt bereikt d.m.v. Racket. De andere component is NMBS samen met de Grafische User Interface (GUI). Dit is de communicatie naar de gebruiker toe en is geen onderdeel van Infrabel, hiervoor wordt ook Racket gebruikt. Er is een duidelijke splitsing, NMBS bouwt boven op de infrastructuur van Infrabel en ze kunnen draaien als aparte processen. De uitgebreide functionele vereisten zijn gefocust op automatisering. Het is mogelijk om een traject te berekenen naar een gekozen locatie, botsingen worden voorkomen aan de hand van een reservatiesysteem en de snelheden van de treinen worden automatisch aangepast op basis van hun traject. De automatische trajectberekening hoort bij NMBS, eenmaal verwerkt stuurt NMBS de juiste commando's door naar Infrabel om het effectieve traject af te leggen. Het voorkomen van botsingen en het automatisch aansturen van de snelheden worden bestuurd door Infrabel.
De 'remove-loco' functie in 'interface.rkt' is aangepast zodat het de 'remove-all-trains' oproept i.p.v. 'remove-train', dit omdat ik problemen ondervond met het laatste.

3. Software-architectuur

3.1. Afhankelijkheidsmodel



Afhankelijkheidsmodel

1. Entry-point is de plaats om het programma op te starten, de uitvoering hiervan zal een Infrabel- en een NMBS-object aanmaken met als argument een poort-nummer om verbinding ertussen tot stand te brengen. Een commando wordt gestuurd naar Infrabel om de spooropstelling in gang te steken, erna naar NMBS om zijn GUI weer te geven. Helpers is een bestand met abstracties en helper-functies (meer uitleg in punt 5: Extra).

2. Het NMBS-component is afhankelijk van de GUI om commando's van de gebruiker te ontvangen, ook is het van Railway-NMBS afhankelijk om gegevens van de spooropstelling beschikbaar te hebben en van Graph-algo's voor het uitvoeren van graaf-operaties.

3. De GUI maakt gebruik van de Racket-GUI-bibliotheek voor het tekenen van de grafische elementen.

4. Infrabel is afhankelijk van Railway-Infrabel voor het verkrijgen van informatie of voor het aansturen van componenten in de fysieke en artificiële spooropstelling. Ook is het afhankelijk van Graph-algo's voor het uitvoeren van graaf-operaties.

5. Railway-NMBS en Railway-Infrabel zijn lagen boven op het spooropstelling-ADT. Deze werken als 'filters' zodat er maar 1 unieke spooropstelling is, dit om codeduplicatie te vermijden en synchronisatie tussen NMBS en Infrabel te behouden. De enige met 'destructieve' methodes (setters) is Railway-Infrabel, de NMBS-versie kan enkel info verkrijgen.

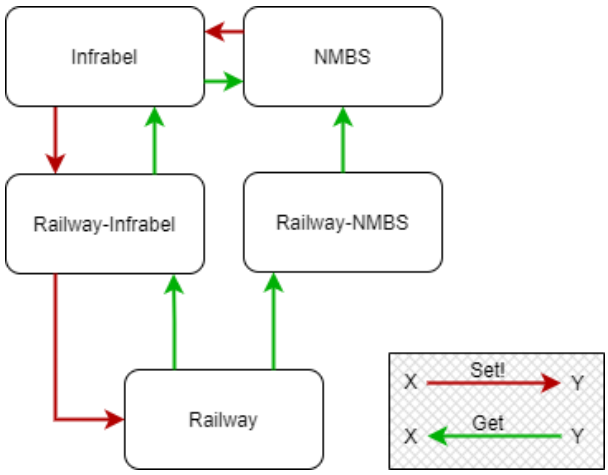
6. De spooropstelling, dit is de eerste laag boven de simulatie of hardware, deze maakt gebruik van spoorelementen zoals trein, wissel en detectieblok maar ook van Graph-algo's voor het uitlezen van informatie in een graaf-structuur.

7. Dit bestand bevat algoritmes die te maken hebben met grafen en de *hard-coded* graaf-representatie van de fysieke opstelling. Het is afhankelijk van het graaf-ADT en graaf-*traversal* algoritme BFT die gebruikmaakt van een *linked-queue* implementatie. (*Deze algortimes zijn gebaseerd op het werk van Wolfgang De Meuter, meer info in 4.7 Graph-algo's.)*

8. De Interface is afhankelijk van de simulator en al zijn componenten.

3.2. API & TCP

Deze afbeelding geeft de communicatie tussen de hoofdonderdelen weer, de groene pijlen zijn om informatie te verkrijgen zonder te moeten schrijven, namelijk lees-operaties, en de rode pijlen duiden destructieve schrijf-operaties aan. Enkel Infrabel kan (rechtstreeks) aanpassingen maken aan de hardware, NMBS zal deze commando's via Infrabel moeten propageren.



Basis API

Berichten die worden uitgewisseld:

NMBS -> Infrabel		**Infrabel -> NMBS**	
exit-railway!	\	update-speed	id, speed
set-speed!	id, speed	update-switch	id, state
flip-direction!	id	update-detection	id, detect-id
switch-state!	id, state	update-traject	id, traject
set-trajectory!	id, traject	destination-reached	id
get-trajectory!	traject	get-trajectory	index, index

NMBS stuurt schrijf-operaties zoals snelheid aanpassen, richting veranderen etc. door naar Infrabel. Infrabel stuurt updates door naar NMBS zodat de GUI up-to-date blijft.
Als Infrabel een nieuw traject wil berekenen, stuurt deze een bericht naar NMBS en zal hij wachten op het antwoord.

TCP
NMBS en Infrabel communiceren over een TCP-verbinding die in stand wordt gebracht bij het instantiëren van beide. Het poortnummer waarop de server werkt wordt hierbij meegegeven en staat gedefinieerd in het 'entry-point'-bestand als 8080. Infrabel werkt als de server en gaat bij het aanmaken een *thread* laten lopen die wacht op een *client*. Als er een gevonden is zullen de 'in'- en 'out'-variabelen op het juiste poortnummer gezet worden om de communicatie tot stand te brengen. NMBS is een *client* en gaat op dezelfde manier wachten tot een server gevonden is via 'localhost' en het meegegeven poortnummer om zijn 'in'- en 'out'-variabelen een nummer te geven. Om informatie te sturen wordt er in beide ADT's gebruikgemaakt van Racket's 'write'-

functie met erna een ‘flush-output’ om de *output-buffer* van de poort effectief weg te schrijven. Geldige berichten zijn altijd een lijst van een Symbol, namelijk de naam van de functie, gevolgd door de argumenten in hun respectievelijke types. Om een input te ontvangen, hebben beide een ‘input-buffer’-methode in een *thread* draaien die wacht tot er communicatie tot stand is gebracht om dan de effectieve ‘input-loop’ op te starten. NMBS en Infrabel hebben beide ook een *hash-table* waar de namen van de bovenstaande functies als een Symbol de *keys* zijn met als waarde de functie zelf. Als de lus input detecteert en het commando wordt gevonden in de *hash-table* wordt ‘apply’ opgeroepen op de waarde ervan met als argumenten de rest van het inkomend bericht. De poorten worden gesloten als de gebruiker de GUI afsluit.

4. ADTs

Een ADT is een Abstract Data Type.
Legende:

Naam ADT		
naam-functie	Argument-type(s) 1, …	Return-type(s)
example-1	Number, Symbol	Pair
example-2	\	{ \ -> Number }
example-3	Pair	{Number, #f }

‘\’ betekent leeg (void) en ‘{ X -> Y }’ is een functie met X als argument-type en Y als return-type. Als in een type-veld meerdere elementen staan zoals in ‘example-3’ is dit een opsomming van de mogelijke types ({ A, B } zonder een pijl zoals bij een methode). Elk ADT heeft een *dispatcher*-methode, wat de teruggegeven functie is bij aanmaak van een nieuwe instantie. Elke *dispatcher* kan omgaan met het commando ‘type’, om de naam van het soort object in een Symbol terug te krijgen. Als een commando niet bekend is, wordt een *error* gegeven. Hierin staat de naam van de *dispatcher* waaruit de fout komt. Niet alle functies staan noodzakelijk in de tabellen opgesomd.

4.1. NMBS

NMBS		
start-ui	\	\
train-callback	Train	\
…- callback	\	\
draw-…	\	\
update-train-speed	Symbol, Number	\
update-switch	Symbol, Number	\
update-detection	Symbol, Symbol	\
update-traject	Symbol, Pair	\
get-trajectory	Number, Number	Pair
destination-reached	Symbol	\

NMBS is de laag tussen de GUI en Infrabel. Bij initialisatie van een NMBS-object wordt een tcp-poortnummer en de afmetingen van de gui meegegeven en erna wordt een Railway-NMBS en een GUI-object aangemaakt. Als NMBS het commando ‘start-ui’ binnenkrijgt worden de draw-functies ‘draw-train’, ‘draw-detection’, ‘draw-switch’ en ‘draw-traject’ opgeroepen, gevolgd met het commando om de GUI weer te geven op het scherm. ‘draw-trains’ zal over alle huidige treinen

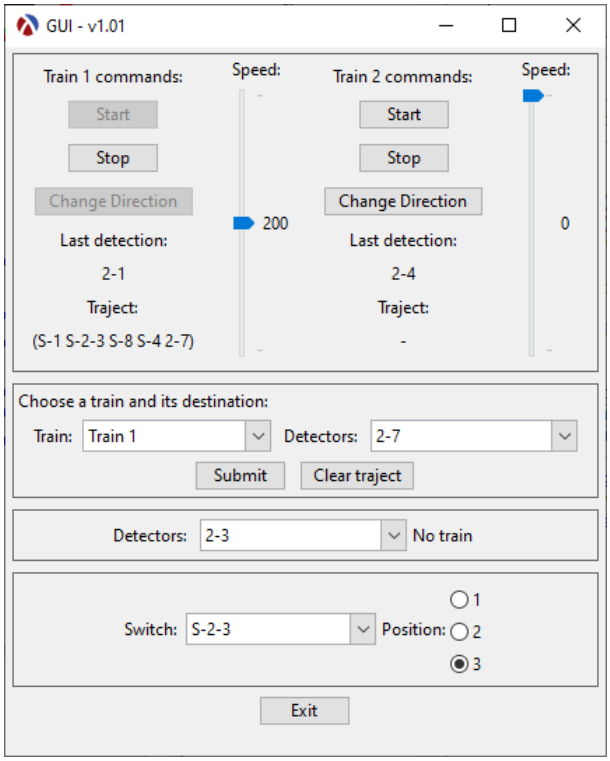
itereren om per trein de UI's 'draw-train' op te roepen, de andere 'draw'-functies delegeren het commando door naar de UI. NMBS heeft *callback*-objecten, die met de GUI's *draw*-functies worden meegegeven, zo kan de GUI de juiste methodes oproepen via de *callback-dispatcher*. In het algemeen zullen *setter*-methodes commando's doorsturen naar Infrabel en *getters* zullen van de spooropstelling informatie vragen. 'train-callback' wordt per trein aangemaakt en meegegeven naar de GUI, deze verwacht een id van een trein als argument om makkelijker per trein commando's te delegeren. Dit object heeft de functies 'start', 'stop', 'change-dir!', 'set-speed-from-slider!' en 'get-prev-detection', binnen de context van een trein zijn de namen vanzelfsprekend, de functies sturen naar Infrabel of de spooropstelling commando's door na eventuele logica te hebben verwerkt. 'detection-callback' kan alle detectie-id's opvragen en heeft een functie 'train-on-track?', die als argument een detectie-id verwacht. Deze functie retourneert *true* als er een trein op dat blok staat. 'switch-callback' heeft de mogelijkheden om een lijst op te vragen van alle wissel-id's en er kan ook de status van een specifieke wissel gelezen of veranderd worden. 'traject-callback' heeft een methode dat gegeven een trein en een bestemming een traject door-communiceert naar Infrabel. Dit traject is het kortste pad van de huidige treinlocatie naar het gegeven detectieblok en wordt door Graph-algos' 'shortest-path' berekend. De 'update-…'-functies zijn methodes die Infrabel aanroept om NMBS (en dus ook de GUI) *up-to-date* te houden, de Symbols zijn ids van de tracks en de Numbers zijn de nieuwe waarden. Als de GUI via de 'exit-callback' het programma wil afsluiten, wordt de 'exit'-Boolean van NMBS gezet, Infrabel krijgt een 'exit-railway' commando, *threads* worden beëindigd en de TCP poorten gesloten.

4.2. GUI

GUI		
open-GUI	{ \ -> \ }	\
close-GUI	\	\
draw-train	{ Train -> \ }	\
draw-…	{ \ -> \ }	\
update-speed	Symbol, Number	\
update-switch	Symbol, Number	\
update-detection-dropbox	{ Symbol -> {Symbol, #f} }, Number	\
update-detection-message	Symbol, Symbol, Number	\
update-traject	Symbol, Pair, Number	\
manual-controls!	Symbol, Boolean, Boolean	\

De Grafische User Interface (GUI) is de communicatie van en naar de gebruiker toe. De GUI is afhankelijk van de Racket/GUI-bibliotheek, deze bevat alle bouwstenen nodig om een UI te maken. De grafische componenten worden aangemaakt met behulp van een *frame*, wat het zichtbaar venster is. *Panels*, of panelen, worden aangemaakt om een venster te verdelen in groepen.
In deze applicatie is het hoofdvenster genaamd 'main-app-frame', dit is verder opgedeeld in 'panel-train', 'panel-detection', 'panel-switch' en 'panel-traject'. Verder is er een *status-line* onderaan het scherm voorzien om eventuele foutmeldingen aan de gebruiker te tonen en een *exit*-knop om de UI af te sluiten. 'open-GUI' en 'close-GUI' zullen respectievelijk het hoofdvenster weergeven of sluiten, 'open-GUI' neemt een functie als argument, namelijk 'exit-callback' van NMBS zodat het stop signaal naar NMBS kan gestuurd worden. Elke 'draw-…'-functie verwacht een functie als argument, dit is een *callback*-object dat in 4.1, NMBS gedefinieerd staat. Per trein is er een knop om te starten, stoppen en één om te veranderen van richting, de snelheid kan aangepast

worden m.b.v. een *slider*. Om te voorkomen dat de slider bij een verandering elke tussenstap als commando doorstuurt, wordt de tijd van het *event* bewaard en vergeleken met de volgende, er moet een bepaalde tijd tussen zitten voordat de *callback*-functie een nieuwe snelheid doorstuurt naar NMBS. Om te voorkomen dat de laatste verandering van de slider niet doorgevoerd wordt, is er een methode genaamd 'buffer' die in een *thread* opgestart wordt bij elke verandering van de slider. Het eerste dat deze doet is een bepaalde tijd wachten om dan de snelheid door te sturen naar NMBS. In het begin van elke *event*-uitvoering wordt deze thread beëindigd, dus enkel als de slider niet meer verandert, zal deze *thread* voorbij zijn *sleep*-instructie geraken. Verder zijn er per trein twee labels, één om de laatste positie van de trein en één om het traject weer te geven. Een detectieblok kan geselecteerd worden uit een *drop down*-menu en zijn status wordt dan zichtbaar in het bijhorende label, eenmaal een wissel geselecteerd is zal zijn status uit te lezen en aan te passen zijn via de set radio-knoppen. Een traject kan berekend worden door een trein te selecteren samen met een bestemming, deze input is beschermd tegen onjuiste waarden. Als de input geldig is zal een commando via de *callback*-functie naar Infrabel gestuurd worden. Waarden zoals de laatste bestemming van de trein, detectie-waardes en switch-standen worden dynamisch geüpdatet op basis van het huidige spoor met behulp van de update-functies. Dit zijn dezelfde argumenten als ze binnenkomen bij NMBS behalve bij de 'update-detection-dropbox'. Daar wordt de functie 'train?' meegegeven aan de GUI om te berekenen of er een trein zich op de huidig geselecteerde detectieblok bevindt. Dit zorgt ervoor dat enkel de juiste component geüpdatet is in plaats van het hele paneel te hertekenen. Ook een nummer dat staat voor de hoeveelheid treinen is nodig om de panelen juist te deconstrueren, bij beide 'update-detection-…' functies en bij 'update-traject'. Om een trein minimaal te beschermen terwijl deze een traject is aan het uitvoeren, zal de gebruiker niet in staat zijn om de trein van richting te doen veranderen of om de trein op te starten. Dit gebeurt a.d.h.v. de methode 'manual-controls!' met als argumenten twee Booleans respectievelijk of de startknop en de 'clear-traject' knoppen ingeschakeld moeten zijn of niet, het symbool wijst op het id van de trein in kwestie.



Screenshot GUI

4.3. Infrabel

Infrabel		
setup-railway!	Boolean	\
exit-railway!	\	\
set-speed!	Symbol, Number	\
flip-direction!	Symbol	\
set-switch-state!	Symbol, Number	{\, #f}
set-trajectory!	Pair, Symbol	\
execute-trajectories	\	\
next-detection-block	Symbol, Symbol	Symbol
calculate-trajectory	Symbol, Symbol	Pair
get-goal-state	Symbol, Symbol	Number

Infrabel zorgt voor de communicatie tussen de software en de hardware, bij initialisatie wordt een tcp-poortnummer meegegeven voor de communicatie en een Railway-Infrabel-object wordt aangemaakt (zie 4.4, Railway-Infrabel). Alle functies die eindigen op een '!' gaan fysiek iets veranderen, bv. de snelheid van een trein aanpassen. Als Infrabel een van deze commando's uitvoert stuurt hij deze door naar zijn Infrabel-spooropstelling en roept de correcte update-functie van NMBS op. Alle nodige *Symbols* zijn ids om de juiste trein of het juiste spoor te veranderen, de *Numbers* zijn nieuwe waarden. 'setup-railway!' krijgt een Boolean 'simulation?' mee die deze doorgeeft aan Railway. Als 'exit-railway' uitgevoerd via NMBS wordt de 'exit'-Boolean van Infrabel gezet, de spoortopstelling krijgt een bericht om af te sluiten, lopende *threads* worden beëindigd en de TCPpoorten gesloten samen met de server.

Automatische trajectberekening
'setup-railway!' gaat ook 'execute-trajectories' opstarten in een *thread* die continu blijft draaien tot de gebruiker Infrabel afsluit. Deze bestaat uit een hoofd-loop en een trein-loop, de hoofdloop zal een bepaalde tijd wachten om dan de trein-loop uit te voeren en opnieuw te beginnen. De 'train-loop' heeft als argument een lijst van alle huidige treinen waarover hij itereert en hun trajecten bijwerkt. Per trein stuurt de loop allereerst een bericht naar Infrabel om het vorige detectieblok op te vragen waar de trein zich bevond/bevindt. Dan controleert hij of er een traject ingesteld staat en zoniet stuurt hij de locatie-update naar NMBS door en gaat naar de volgende trein. Als er wel een traject aanwezig is, zijn er 4 scenario's: de bestemming is bereikt, er zijn geen reservaties voor/door de huidige trein, de volgende detectieblok in het traject is bereikt of het volgende spoorstuk in het traject is een wissel. Als de eindbestemming bereikt is, stopt de trein, wordt zijn traject leeggemaakt en NMBS wordt op de hoogte gebracht. Als er geen reservaties zijn, dan wordt gekeken of het mogelijk is een blok sporen te reserveren. Als de volgende detectieblok in het traject bereikt is, zal indien nodig de trein omgedraaid worden, het traject bijgewerkt worden en NMBS zal een update-bericht ontvangen. Indien het een wissel is, wordt eerst de *goal*-positie van NMBS opgevraagd en de *from*-positie van de switch berekend. Dan onderscheiden we twee gevallen: het normale geval waar de wissel gewoon in de *goal*-stand gezet wordt en een speciaal geval waarbij NMBS een van-naar-instructie heeft gegeven dat de trein niet rechtstreeks kan afleggen. In dit geval vraagt Infrabel een nieuw traject aan van de huidige locatie van de trein naar een volgend detectieblok om op te draaien en voegt deze toe aan het huidige traject. De methodes die hiervoor zorgen noemen 'next-detection-block' en 'calculate-trajectory'. De tweede gaat op basis van het vorige treinspoor en het berekende detectieblok het huidige traject uitbreiden. 'next-detection-block' kijkt naar de richting dat de trein de huidige wissel is opgereden en zal op basis van dit en de *goal-state* het volgende spoor berekenen en zichzelf her-oproepen. De functie stopt als het een spoorstuk vindt dat geen wissel is en geeft het id ervan terug. Indien

beide uitgangen mogelijk zijn (= de *goal-state* van het huidige spoorstuk is 0) dan zal het algoritme kijken of 1 van de 2 uitgangen een detectieblok is, zoniet wordt uitgang 2 gekozen. (*Dit kan geoptimaliseerd worden door beide takken af te gaan en de kortste route te kiezen.)* 'get-goal-state' gaat het *label* op een specifieke *edge* opvragen en de juiste waarde teruggeven als een nummer. In beide gevallen van de wissels wordt erna het traject geüpdatet m.b.v. 'update-traject'. Uiteindelijk wordt de trein-loop opgeroepen met de volgende trein.

Reservaties
Als een trein geen reservaties heeft, betekent dit dat er geen, of enkel zijn huidige locatie, gereserveerd is. De huidige locatie wordt in elke iteratie van de 'train-loop' gereserveerd door 'reserve-current-location' zodat bij manuele controle de huidig bezette stukken gereserveerd blijven. Een reservatie is altijd van een detectieblok naar een detectieblok. Om een reservatie te maken wordt 'make-reservation' opgeroepen waar er over het huidige traject van de trein wordt geïtereerd tot het volgende detectieblok gevonden is of indien een spoor al gereserveerd is. In dat geval zal de trein zijn 'waiting?' Boolean op waar gezet worden en zal hij een bepaalde tijd wachten voor hij opnieuw probeert te reserveren. Als er geen sporen door een andere trein gereserveerd waren, zal de 'succes?' Boolean waar blijven en kunnen de sporen in kwestie effectief gereserveerd worden samen met een update van de reservatielijst van de trein. Als de laatste detectieblok van een reservatie bereikt is, worden de voorgaande sporen vrijgesteld en de reservaties van de trein leeggemaakt a.d.h.v. 'unreserve-all!', wat betekent dat in de volgende iteratie er geprobeerd wordt om het volgende deel van het traject te reserveren. Als er een speciale manoeuvre nodig is, worden de spoorstukken die al afgelegd zijn tot dat punt vrijgesteld en de reservaties van de trein worden leeggemaakt vóór het nieuwe traject ingesteld is.

Dynamische snelheden
Als het traject van een trein geüpdatet wordt, zal op basis van de lengte van het huidige traject en het type van de huidige bestemming de snelheid aangepast worden m.b.v. de functie 'speed-control'.
Als het traject nog maar 1 of 2 sporen bevat zal de trein trager rijden, idem voor de lengte van de reservaties. Als de trein zich bevindt op een lang stuk zal deze snel rijden, op een kort stuk zal hij trager rijden, in andere gevallen mag de trein zich op zijn standaard snelheid voortbewegen.
Treinen worden geleidelijk aan opgestart/gestopt/omgedraaid in 4.5 Railway.
Enkel 'setup-railway' is verkrijgbaar via de *dispatcher*.

4.4. Railway-Infrabel

Railway-Infrabel		
setup-railway!	Boolean	\
exit-railway!	\	\
get-trains	\	Pair
find-train	Symbol	Train
prev-detection	Symbol	Symbol
switch	Symbol	Switch
switch-state	Symbol	Number
switch-state!	Symbol, Number	\
train-speed	Symbol	Number
train- speed!	Symbol, Number	\
flip-direction!	Symbol	\
reserved?	Symbol, Symbol	{#f, Pair}
set-reserve!	Symbol, {#f, Pair}	\

unreserve-all!	Symbol, Symbol	\

Railway-Infrabel is een abstractielaag boven op de algemene spooropstelling, specifiek de versie van Infrabel. Infrabel houdt een instantie hiervan bij en stuurt er commando's naar door, Railway-Infrabel stuurt ze op zijn beurt door naar de algemene Railway die het delegeert naar de effectieve spooropstelling of simulator. Infrabel zal destructieve operaties uitvoeren, de functie-namen spreken voor zich, alle nodige Symbols zijn ids om de juiste operatie uit te voeren en de Numbers zijn nieuwe waarden. 'find-train' en 'get-switch' gaan een instantie teruggeven. Het argument van 'setup-railway' gaat bepalen welke import gebruikt wordt, wat *true* is om de simulatie te kiezen. *(Niet geïmplementeerd, nu is het altijd de simulatie die opgestart wordt.)* Om een spoorstuk te reserveren zal via de 'switch?' en 'detection?' hulpprocedures Railway's correcte 'find-..'-functie opgeroepen worden om de 'reserved' variabele van het juiste spoorstuk te veranderen. Om te kijken of een spoorstuk gereserveerd is, wordt met dezelfde hulpprocedures geïtereerd over de correcte lijst van sporen om de 'reserved' variabele uit te lezen. 'unreserve-all!' zal op basis van een binnenkomende trein-id alle sporen de-reserveren die deze trein in beslag nam, behalve het meegegeven spoor-id. Alle bovenstaande functies zijn verkrijgbaar via de dispatcher.

4.5. Railway-NMBS

Railway-NMBS		
train-ids	\	Pair
detection-ids	\	Pair
switch-ids	\	Pair
prev-detection	Symbol	Symbol
switch-state	Symbol	{Number, #f}
train-speed	Symbol	Number
train-direction	Symbol	Boolean
train-traject	Symbol	Pair

Railway-NMBS is de abstractielaag boven op de algemene spooropstelling, specifiek de versie van NMBS. Als NMBS info nodig heeft over bepaalde spoorelementen zal hij commando's sturen naar zijn Railway-NMBS-instantie, deze worden doorgestuurd naar de algemene Railway die het delegeert naar de effectieve spooropstelling. Alle nodige Symbols zijn ids om de juiste informatie te ontvangen. De Pairs zijn gelinkte-lijsten (zie 4.6 Railway voor meer info). De 'get-train-ids', 'get-switch-ids' en 'get-detection-ids' geven een gelinkte lijst van symbolen terug, namelijk de ids van de respectievelijke types. 'get-switch-state' gaat de lijst van wissel-ids opvragen, de gevraagde wissel zoeken en zijn status teruggeven. Als de wissel niet gevonden is zal deze functie *false* teruggeven. De rest van de functies spreken voor zichzelf, alle bovenstaande functies zijn verkrijgbaar via de *dispatcher*.

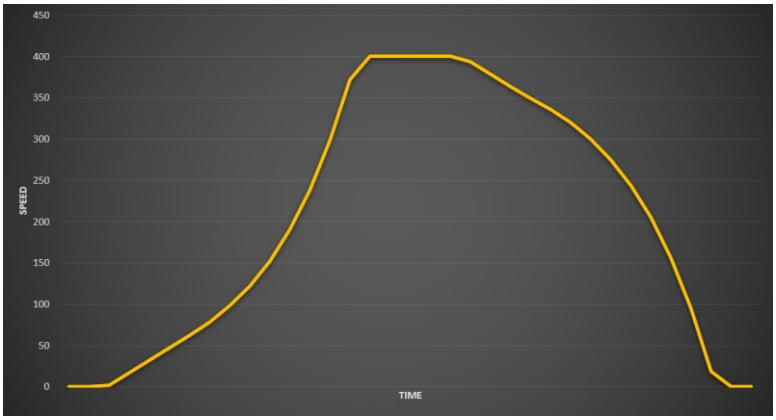
4.6. Railway

Railway		
setup-railway!	Boolean	\
exit-railway!	\	\
get-trains	\	Pair
find-train	Symbol	Train
get-train-direction	Symbol	Boolean
get-prev-detection	Symbol	Symbol

get-train-speed	Symbol	Number
get-train-traject	Symbol	Pair
change-train-speed!	Symbol, Number	\
flip-direction!	Symbol	\
get-detections	\	Pair
get-switches	\	Pair
find-switch	Symbol	Switch
find-detection	Symbol	Detection
get-switch-state	Symbol	{Number, #f }
set-switch-state!	Symbol, {Number, Pair}	\
smooth-speed	Number, Symbol	\

Railway stelt de algemene spooropstelling voor en bouwt boven op de simulator of de hardware een laag op voor NMBS en Infrabel. Via hun eigen Railway-ADTs blijven functionaliteiten gescheiden. Dit ADT bevat de gelinkte lijsten 'trains', 'detections' en 'switches' om fysieke spoorelementen in bij te houden en zijn via hun *get*-methodes beschikbaar. Ook is er een id-*generator*-object geïmporteerd vanuit 'helpers.rkt' om nieuwe treinen een unieke identificatie te geven. 'setup-railway' brengt de fysieke opstelling tot stand, wat sowieso de simulatie is, de keuze tussen de verschillende gesimuleerde opstellingen is *hard-coded*. Erna worden de (globale) lijsten met behulp van 'set-detections!' en 'set-switches!' aangevuld en ten slotte worden 2 treinen toegevoegd. Als Infrabel het commando stuurt om de hardware te stoppen, zal 'exit-railway!' ervoor zorgen dat de lopende *threads* van alle treinen afgesloten worden om erna de hardware te stoppen. 'set-detections!' krijgt vanuit de interface een lijst van ids, per id wordt een nieuwe detectieblok aangemaakt en zijn variabelen met de juiste waarden gezet. In 'set-switches!' gebeurt hetzelfde behalve dat er een controle is op de speciale drie-weg-wissel, deze wordt *hard-coded* ingesteld. De argumenten om een nieuwe 'switch aan te maken, zijn afhankelijk van de geconnecteerde spoorstukken die aan de hand van 'state-connections' opgevraagd worden (zie 4.7 Graph-algo's). Over deze lijst wordt geïtereerd om de posities van de wissel te verbinden met de juiste spoorstukken. Om de status van een wissel te veranderen geef je het id van de wissel mee met 'set-switch-state' die dan de fysieke wissel aanstuurt. Indien het de drieweg is zal hij de twee aparte wissels in de hardware aansturen en zal de meegegeven state een *cons-cell* zijn van de individuele statussen, vandaar de types Number of Pair bij het 2^{e} argument. Een trein toevoegen gebeurt d.m.v. 'add-train!', het argument is een paar met twee opeenvolgende detectieblok-ids om de trein een beginplaats te geven. 'get-prev-detection' vraagt als argument het id van een trein en geeft het id terug van de detectieblok waar de trein zich bevindt of laatst bevonden heeft. 'find-train' en 'find-switch' gaan op basis van een gegeven id de overeenkomstige trein- of switch-instantie teruggeven. Om een trein in de simulator realistischer te doen opstarten, afremmen, stoppen en draaien zal elke oproep van 'change-train-speed!' de trein haar 'speed-controller' updaten, dit is een *thread* die de snelheid van de trein aanpast. Als de trein zich niet in een draaimanoeuvre bevindt, zal de huidige *thread* beëindigd en herstart worden met de functie 'smooth-speed' en als argument de goal-snelheid. Deze methode heeft een loop met als argument een *compare*-functie, wat de kleiner-dan of groter-dan *comparator* is (<, >) afhankelijk of de trein moet versnellen of afremmen. In deze lus wordt berekend hoeveel de nieuwe snelheid voor deze iteratie zal zijn door simpelweg een bepaalde snelheid te delen door vier en dit resultaat eraan toe te voegen. Als de trein moet versnellen, wordt zijn huidige snelheid hieraan meegegeven zodat hij altijd sneller optrekt. In het geval van afremmen zal het verschil tussen de maximumsnelheid en de huidige snelheid worden meegegeven, dit creëert dezelfde waarden als het optrekken maar dan invers. Om een trein te doen omdraaien, wordt de 'change-train-speed' geblokkeerd door de 'flipping?' Boolean zodat de trein tijd heeft om tot stilstand te komen om zijn richting juist te zetten. Vanaf dit punt is de snelheid weer gedeblokkeerd voor de rest van het systeem en wordt de trein heropgestart.

Het Railway-bestand heeft geen *dispatcher*-functie. De scheiding van functies voor Infrabel of NMBS gebeuren respectievelijk via de Railway-Infrabel- en Railway-NMBS-ADTs. Functies die niet benoemd zijn spreken voor zichzelf.



Het optrekken naar de maximumsnelheid en terug tot stilstand komen van een trein.

4.6.1. Train

Train		
id	\	Symbol
get/set: speed	\ - Number	Number - \
get/set: direction	\ - Boolean	Boolean - \
get/set: prev-detect	\ - Symbol	Symbol - \
get/set: prev-track	\ - Symbol	Symbol - \
get/set: traject	\ - Pair	Pair - \
get/set: reservations	\ - Pair	Pair - \
get/set: waiting?	\ - Boolean	Boolean - \
get/set: speed-controller	\ - Thread	Thread - \

Een trein stelt de hardware-versie voor in de software, dit ADT bevat enkel algemene *get*- en *set*-functies. Een trein-instantie heeft een id, richting, vorig detectie-block, vorig-spoorelement, een traject en reservatie-lijst, beide een lijst van ids, een Boolean die het wachten voorstelt en een speed-controller waar een Thread in gedefinieerd wordt die de snelheid van de trein aanpast. De richting wordt voorgesteld met een Boolean waar *true* vooruit betekend en *false* achteruit. Het vorig-spoorelement verschilt met het vorige detectieblok: het kan ook een wissel zijn en het is nodig voor de trajectberekening in Infrabel.

4.6.2. Switch

Switch		
get/set: id	\ - Symbol	Symbol - \
get/set: state	\ - Number	Number - \
get/set: track-id-0	\ - Symbol	Symbol - \
get/set: track-id-1	\ - Symbol	Symbol - \
get/set: track-id-2	\ - Symbol	Symbol - \
get/set: reserved	\ - {#f, Symbol}	{#f, Symbol } - \

Een gewone wissel zorgt dat treinen van spoor kunnen veranderen en heeft twee statussen: 1 en 2. Een nieuwe instantie heeft een id en de ids van de aangesloten stukken rond hem nodig, respectievelijk met welke stand de wissel zich in moet begeven om ermee verbonden te zijn. (0, 1 of 2). De variable 'reserved' is het id van de trein die dit stuk reserveert of *false* indien het vrij is. Voor de kruiswissel is er geen apart ADT voorzien omdat het werkt als 2 gewone wissels. Alle functies zijn beschikbaar via de *dispatcher*.

4.6.3. Three-way

Three-way		
first-switch	Switch	
second-switch	Switch	
get/set: state-1	\ - Number	Number - \
get/set: state-2	\ - Number	Number - \
track-id-3	\	Symbol

Omdat een drieweg inwendig zeer anders werkt, is het geen rechtstreekse uitbreiding van de switch-klasse maar een klasse op zichzelf. Het bevindt zich wel in het zelfde '.rkt'-bestand en maakt gebruik van 2 normale switch-instanties.
Een drieweg is een speciaal type wissel, hij bevat twee 'gewone' wissels en moet in een bepaalde volgorde geactiveerd worden om ontsporingsgevaar te vermijden. De nodige argumenten zijn idem als bij de gewone wissel behalve dat er nog een spoorstuk bijkomt. Bij initiatie worden twee gewone switches aangemaakt en dan verbonden met elkaar, bij de *set* worden de inwendige *states* van de wissels in de juiste volgorde gezet, de *get* is anders dan een gewone wissel omdat het ook een 3 kan teruggeven. De drieweg heeft een unieke en anders gevormde *identifier*, een andere betekenis van statussen 1 en 2 en er is ook een get voor track-id-3 en de aparte statussen van de wissels. Alle functies zijn beschikbaar via de *dispatcher*.

4.6.4. Detection-block

Detection-block		
id	\	Symbol
get/set: status	\ - Boolean	Boolean - \
get/set: reserved	\ - {#f, Symbol}	{#f, Symbol } - \

Een detectie-blok is een stuk spoor waar het mogelijk is om te weten of er zich een trein op bevindt of niet. Een nieuwe instantie heeft zijn eigen id en een variabele 'reserved' die vertelt of een trein dit stuk gereserveerd heeft of niet, de variabele zal de waarde *false* of het gereserveerde trein-id hebben.

4.7. Graph-algo's

Graph Algos		
shortest-path	Number, Number	Pair
get-edge-label	Symbol, Symbol	Symbol
state-connections	Number	Pair

Ik maak gebruik van de code van Wolfgang De Meuter voor labeled graphs met een adjacency-matrix alsook de linked-implementatie voor een queue, de graaf-traversal algo's zijn gebaseerd op het bft-applications.rkt-bestand.
Om een spooropstelling voor te stellen wordt een ongericht, gelabelde graaf gebruikt te verkrijgen onder de naam 'hardware-graph'. Het is ongericht omdat het enkel op specifieke plaatsen handig zou zijn om een richting te hebben, maar het zou te veel code-duplicatie zijn om elders overal bi-directionele pijlen te zetten. In de plaats van het gericht te maken, wordt gebruikgemaakt van een labelsysteem. De *nodes* hebben het label van hun id en de *edges* hebben een label afhankelijk van het spoortype, dit is *hard-coded*. Een label is een symbool bestaande uit twee nummers gescheiden door een streep en ziet er bv. als volgt uit: '1-2. Beide nummers stellen een positie van een wissel voor om van of naar de *node* te geraken, een detectieblok heeft standaard 0 als positie. Elke node heeft een index en op basis van welke de grootste of kleinste is wordt een richting gecreëerd, bv. als van node met index 20 naar node met index 35 de juiste status van de pijl wordt gevraagd zal dat het eerste getal zijn, in de andere richting wordt dat het tweede getal bv. '0-1: van laag naar hoog is 0, van hoog naar laag 1. 'shortest-path' vraagt twee indexen en zal met behulp van BFT het kortste pad teruggeven, dit is een lijst dat opgebouwd wordt in het algoritme. 'get-edge-label' geeft het label terug op een pijl. Een lijst van aangesloten spoorelementen van een bepaalde switch per positie wordt opgebouwd in 'state-connections', dit is afwisselend het nummer van de positie en een Symbol voor het verbonden spoor-id. In het bestand /labeled/adjacency-matrix.rkt is er een 'my-for-each-edge'-functie bijgekomen die een lijst opbouwt en teruggeeft afhankelijk van de meegegeven procedures. De reden dat BFT gebruikt wordt i.p.v. DFT is omdat het beter werkt op de manier dat de 'hardware-graph' is opgebouwd en in een ongerichte graaf moet het DFT algoritme meer aangepast worden om niet in loops vast te geraken.

5. Extra en conclusie

Bugs/optimalisaties/uitbreidingen. Manuele controle is niet 100% gegarandeerd en in specifieke scenario's wordt de richting van de trein verkeerd gezet bij de automatische trajectberekening. Dit heeft te maken met het feit dat de vorige track (niet vorige detectie) van de trein niet correct gezet wordt na een manuele verandering van richting. Een oplossing hiervoor zou zijn om de graaf te deconstrueren en zo deze variabele ook te updaten bij de manuele controle in Infrabel. De speciale gevallen S-16 en 2-8 worden niet opgevangen, de dode eindes zijn wel gemarkeerd als 'x en zouden als controle gebruikt kunnen gworden om de trein te stoppen.

Helpers.rkt is een bestand met abstracties voor waarden (bv. *thread*-timings, trein snelheden …) en hulp-functies (bv. lijstoperaties, debuggen ...) Ook bevindt er zich een vertaling 'label->index' die op basis van een gegeven spoor-id de index van de node in de graaf-opstelling teruggeeft, dit is *hard-coded*.

Motivatie. Ik maak gebruik van symbolen en lijsten doorheen dit project, dit omdat de lijsten zeer beperkt blijven van lengte en in de meeste gevallen sowieso door een hele reeks moet geïtereerd worden, met *car* en *cdr* in O(1) is dit efficiënt en zeer dynamisch. De symbool-operaties zijn ook efficiënt genoeg omdat het gaat over symbolen van max lengte 6.
In commentaar staan er nog een '2do's' maar de onnodige lijnen en debug-statements zijn er niet meer.

Conclusie. Het was een uitdagend project met vele creatieve vereisten en uitbreidingsmogelijkheden. Met iets minder werkdruk en deadlines voor andere vakken, minder stress en mentaal deprimerende situaties (onder andere corona) had ik de losse eindjes beter kunnen vastbinden om dit project naar een beter einde te leiden, een completer einde, maar uiteindelijk ben ik tevreden met het finale resultaat.