Programmeerproject 2: Tweede Zittijd

Titularis: Prof. Elisa Gonzalez Boix
Begeleidende assistenten: Sébastien Hoorens, Janwillem Swalens, Lars Van Holsbeeke
project2ba@dinf.vub.ac.be

Tweede Bachelor Computerwetenschappen, 2016-2017

Deadline: **maandag, 21 augustus 2017 om 8:00**

1 Inleiding

In het programmeerproject van de eerste bachelor hebben jullie een eerste groot programma in Scheme gebouwd. Voor programmeerproject 2 maken jullie individueel een complete softwaretoepassing. Deze bevat een software-architectuur met meerdere componenten, inclusief hardwarecomponenten. Daarenboven zal je uiteindelijke "product" niet enkel bestaan uit een werkend programma maar ook de nodige documentatie moeten bevatten. Dit project zal gequoteerd worden op drie vlakken:

- Het **ontwikkelen** van een programma van grote omvang. Dit moet voldoen aan een reeks gegeven **functionele en kwaliteitseisen**.
- Het **ontwerpen, testen en documenteren** van een softwareproject, dat blijk geeft van een grondige analyse van de gegeven opdracht.
- De schriftelijke en mondelinge **rapportering** van het ontwikkelingstraject. Dit bestaat uit een **handleiding** gericht op de eindgebruiker, een **specificatie** gericht op andere ontwikkelaars die eventuele aanpassingen aan jouw project moeten kunnen uitvoeren, een **demonstratie** van het programma, en een **mondelinge verdediging** van de gemaakte keuzes.

Dit document bevat een algemene omschrijving van de opstelling (sectie 2), de minimale functionele en niet-functionele vereisten (sectie 3), en details over de deliverables (sectie 4). In dit project is er veel ruimte gelaten om je creativiteit op bot te vieren. Je wordt aangemoedigd om de originele opdracht te verrijken met eigen inbreng die in het thema van de opdracht past.

Het project voor tweede zittijd bestaat uit de combinatie van de vereisten voor het eerste en tweede semester van de eerste zittijd, aangevuld met een extra vereiste, nl. de automatische berekening van trajecten.

Let op: tijdens de tweede zittijd vragen we geen voorstudie of tussentijds verslag. Als je tijdens het werk aan je project feedback wil, zal je er dus expliciet om moeten vragen. Aarzel niet om ons te mailen of een afspraak te maken. Vergeet ook niet dat het tijdens de vakantie wat langer kan duren voor je antwoord krijgt.

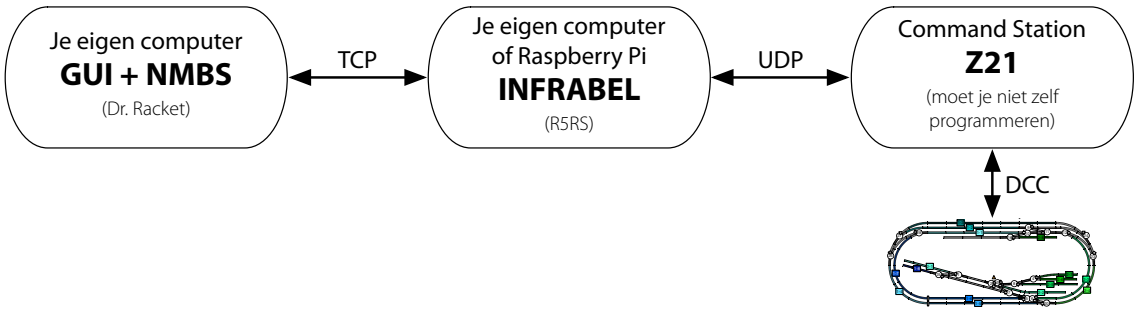
Figuur 1: Een opstelling van Märklin-modeltreinen. (Bron: Wikipedia)

2 Opstelling

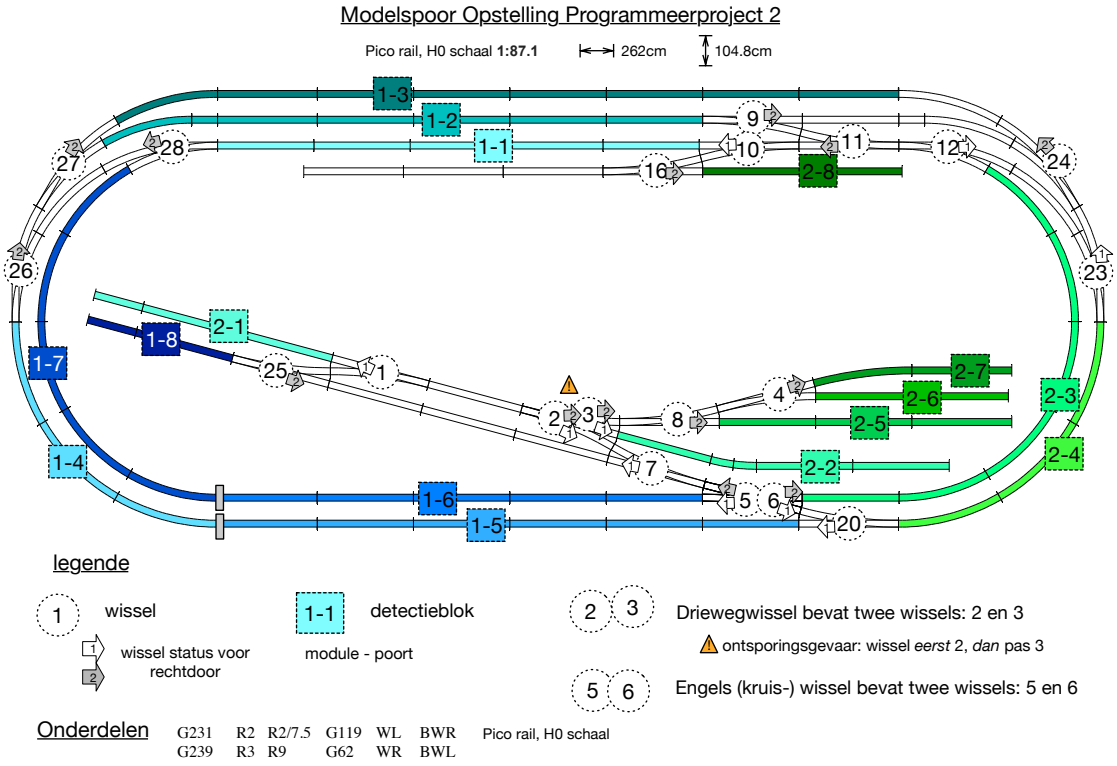
Het onderwerp van het project is de ontwikkeling van een controlesysteem voor een modelspoor (figuur 1). Modeltreinen kunnen tegenwoordig volledig digitaal aangestuurd worden. In oudere systemen werd een locomotief aangestuurd door een simpele draaiknop die de spanning regelde. Nu wordt de trein door een extern programma aangestuurd met digitale commando's. Digitale locomotieven gebruiken het elektrisch circuit op de sporen als een datacommunicatiebus. Niet alleen de snelheid van de locomotieven wordt op deze manier digitaal geregeld, maar ook andere functionaliteiten zoals het veranderen van wissels.

De functionaliteiten die je in dit project moet ontwikkelen, kunnen opgesplitst worden in drie grote componenten (figuur 2):

- Een **Command Station (Z21)** dat de verschillende elementen van het modelspoor aanstuurt. Dit Command Station communiceert met de hardware via het *Digital Command Control (DCC)* protocol. De software hiervoor programmeer je niet zelf. Het Command Station ondersteunt een groot aantal commando's die er via het netwerk naar kunnen gestuurd worden. Deze staan opgelijst in de documentatie van de Z21-bibliotheek, die je op PointCarré kan vinden.
- **INFRABEL** is de component die instaat voor de communicatie tussen de software en de modelbouwhardware (detectie van treinen en aansturing van treinen en wissels), en het doorlopend beheer van de infrastructuur (trajectcontrole en automatisch remsysteem). Dit is deel van de infrastructuur en moet permanent draaien. INFRABEL kan je draaien op je eigen computer of op een Raspberry Pi (een kleine gelimiteerde computer).
- **NMBS** en de **grafische interface (GUI)**. NMBS staat in voor de functionaliteit die geen logisch onderdeel is van de infrastructuur maar er bovenop bouwt, zoals een grafische



Figuur 2: De drie grote componenten in dit project.



Figuur 3: Schematische voorstelling van de opstelling.

interface, het uitstippelen van trajecten, of het uitrekenen van tijdstabellen. Het NMBS-gedeelte draait op je eigen computer en communiceert met INFRABEL.

De fysieke opstelling bevindt zich in lokaal 10F704. Figuur 3 toont hier een schematische voorstelling van. De opstelling bestaat uit:

Spoorsegmenten Dit zijn de stukken spoor (exclusief wissels).

Wissels De cirkels duiden wissels aan, deze maken het mogelijk van spoor te veranderen. Het pijltje duidt de oriëntatie van de wissel aan. Wissels 2–3 en 5–6 zijn speciaal: dit zijn combinaties van twee wissels die ervoor zorgen dat er tussen meer sporen kan gewisseld worden. Voor wissel 2–3 is het belangrijk dat wissel 2 voor wissel 3 ingesteld wordt.

Detectieblokken De gekleurde zones 1-1 t.e.m. 2-8 zijn detectieblokken. Hiermee bepaal je de locatie van de treinen: een detectieblok geeft weer of er zich al dan niet een trein op dat blok bevindt. Je kan de exacte locatie van een trein dus niet bepalen, enkel in welk blok hij zich bevindt. Als een trein zich buiten een detectieblok bevindt (enkele witte segmenten en alle wissels), weet je niet zeker waar hij is.

Omdat het nogal complex kan zijn om met de werkelijke hardware te werken, bieden we ook een simulator aan. Daardoor kan je in het begin van het project proberen eerst een eenvoudigere opstelling werkend te krijgen; later in het project kan je experimenteren met opstellingen die complexer, uitdagender en interessanter zijn. Uiteindelijk moet je project zowel op de simulator als op de echte hardware kunnen draaien.

2.1 Simulator

Het INFRABEL-gedeelte dient commando's door te sturen naar de Z21 Command Station om de verplaatsing van locomotieven over het modelspoor te beheren. Het ontwikkelen en uitvoeren van de code die hiervoor verantwoordelijk is kan vrij moeizaam verlopen: je beschikt (vermoedelijk) niet over een eigen modelspoor thuis, en bovendien moeten de commando's door middel van message passing via het UDP-protocol doorgegeven worden. Het gebruik van een simulator die het gedrag van de Command Station nabootst kan deze moeilijkheden opheffen. Verder kan een simulator nuttig zijn om je project te demonstreren zonder de hardware. Eens je code werkt met de simulator, hoef je met minder zaken rekening te houden wanneer je je programma uitbreidt om het echte modelspoor te kunnen bedienen. Hier zul je het belang ondervinden van modulair werken: hoe minder je code afhankelijk is van de manier waarop de Command Station bediend wordt, hoe gemakkelijker je het zult hebben om je code op zowel hardware als simulator te laten werken. We raden dus aan om met verschillende abstractieniveaus te werken.

Om je op weg te helpen hebben we een eenvoudige simulator geschreven in Racket. De code van de simulator bestaat uit drie Racket-files, waarvan `interface.rkt` de belangrijkste is voor het project. De interface exporteert een aantal functies zoals beschreven in tabel 1 (in de appendix). Je INFRABEL-gedeelte (het NMBS-gedeelte zou de simulator niet nodig moeten hebben) mag enkel deze functies gebruiken. Tijdens het ontwikkelen van je code kan het eventueel handig zijn om *tijdelijk* de andere Racket-files van de simulatie te gebruiken (bv. om de exacte positie van een locomotief te kunnen verkrijgen). Je mag zeker en vast wijzigingen aanbrengen aan de code van de simulator, zolang je maar geen extra functies exporteert in de interface en zolang de signaturen van deze functies dezelfde blijven, deze simuleren immers de functionaliteit van de echte Command Station.

De structuur van een modelspoor wordt gespecificeerd door een text-bestand dat ingeladen wordt door de simulator. Het text-bestand die we voorzien hebben komt echter niet overeen met de structuur van het echte modelspoor waarmee je tijdens het tweede semester zult werken. Het kan interessant zijn om zelf extra text-bestanden volgens zelf-ontworpen modelsporen toe te voegen om specifieke use-cases voor het project uit te testen. Merk op dat de interface niet toestaat om de ingeladen representatie van het modelspoor te raadplegen vanuit INFRABEL. Bovendien hebben we opzettelijk gekozen voor een eenvoudige en niet al te efficiënte datastructuur om het modelspoor in bij te houden. Om een goed project te hebben is het de bedoeling dat je zelf een efficiëntere datastructuur implementeert waarmee je de representatie van het modelspoor kan inladen in INFRABEL (bv. de datastructuur zou het mogelijk moeten maken om op een efficiënte manier de modelspoorcomponenten te vinden die verbonden zijn met een gegeven component).

2.2 Communicatie met hardware

Je vindt de treinen in lokaal 10F704. Om je laptop te verbinden met de treinen, zet je de router aan en maak je verbinding met het Wifi-netwerk. Het wachtwoord van dit netwerk vind je op de onderkant van de router (de code achter PIN). De Raspberry Pi kan je verbinden met de treinen via de Ethernet-kabel. Om te communiceren met de treinen gebruik je de Z21-bibliotheek, die je op PointCarré kan vinden. De communicatie gebeurt via UDP. Je moet eerst een socket aanmaken, dit kan je doen met de functie `(setup)` van de bibliotheek (in `Z21Socket.rkt`). Deze opent een socket, en maakt verbinding met de controller van de treinen via het standaard IP-adres en de poort waarop deze zich bevindt (deze moet je normaal gezien niet aanpassen). De socket wordt teruggegeven.

Figuur 4: De mobiele app voor de Z21. Met de knoppen '⟨' en '⟩' rechtsonder kan je de rijrichting van de locomotief kiezen, met de slider erboven kies je de snelheid. Het tabje aan de rechterkant van het scherm brengt je naar een scherm waarmee je wissels kan verzetten.

Voor de verdere communicatie gebruik je de functies `(send socket message)` om berichten te verzenden, en `(listen socket . reader)` om een thread te starten die naar berichten luistert en voor elk bericht de lambda `reader` aanroept. Er zijn verschillende manieren om berichten te lezen: `(read-messages reader)` leest meerdere berichten, `(get-message socket)` en `(get-message* socket)` lezen één bericht op respectievelijk synchrone en asynchrone manier. Meer informatie over deze functies vind je in de documentatie van de Z21-bibliotheek.

De communicatie met de modeltreinen gebeurt via het *Digital Command Control (DCC)* protocol. Dit specificeert welke berichten je kan sturen naar de controller. De Z21-bibliotheek bevat functies om deze berichten te maken, bv. `make-set-loco-drive-msg` en `make-set-switch-msg` om locomotieven en wissels aan te sturen (uit `Z21MessageDriving.rkt` en `Z21MessageSwitches.rkt`). Je vindt een beschrijving van alle functies in de documentatie van de bibliotheek.

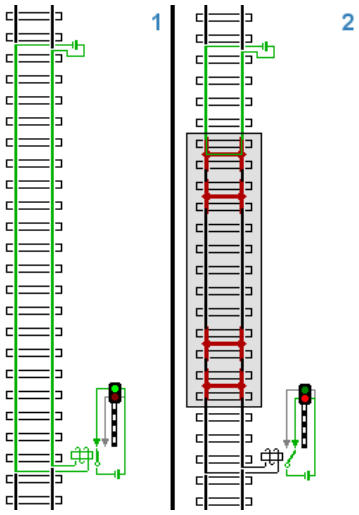
De bibliotheek bevat ook een aantal tests, in de map `APITesting`. Hier kan je zien hoe de interface typisch gebruikt wordt. Probeer deze code eens uit te voeren.

Daarnaast heeft de Z21 ook een mobiele app voor Android en iPhone, die je vindt via `http://www.z21.eu/en/Downloads`. Deze kan handig zijn voor je allereerste experimenten. Figuur 4 toont een screenshot. Normaal gezien werkt deze app automatisch nadat je je telefoon met het Wifi-netwerk verbindt. Je start de app best wel pas ná dat je verbinding met het netwerk hebt gemaakt, anders herkent de app de treinen meestal niet.

2.3 Raspberry Pi

Er zijn twee manieren om je project op de hardware te draaien:

Met Raspberry Pi In dit scenario draait NMBS op je laptop en INFRABEL op een Raspberry Pi. Je laptop communiceert met de Raspberry Pi; deze communiceert op zijn beurt met de modeltreinen. Dit is het scenario zoals uitgelegd in het document van het eerste semester. Hierbij kan je je laptop uitschakelen, de veiligheid blijft gegarandeerd door INFRABEL op de Raspberry Pi.



Figuur 5: Signalisatie: lichten bevinden zich aan het begin van elk detectieblok. Wanneer er een trein gedetecteerd wordt op het segment, wordt het licht rood. (Bron: Wikipedia)

Zonder Raspberry Pi In dit scenario draaien zowel NMBS als INFRABEL op je laptop. Deze communiceert rechtstreeks met de modeltreinen. Hierbij is het belangrijk dat je je laptop niet uitschakelt, en de netwerkverbinding niet verbreekt: INFRABEL staat immers in voor de veiligheid en moet continu actief zijn.

Het gebruik van de Raspberry Pi is *niet verplicht*. We zien dit als een extra dat bonuspunten kan opleveren. Zorg er dus voor dat je project eerst zonder Raspberry Pi werkt voordat je deze extra hardware gebruikt. Indien je een Raspberry Pi wilt gebruiken, kan je er een lenen door ons te mailen.

3 Vereisten

In deze sectie beschrijven we de functionaliteit die je toepassing zeker moet bevatten. Zoals eerder vermeld word je aangemoedigd dit uit te breiden met extra functionaliteit. Let wel op dat de nadruk in dit project nog steeds ligt op de kwaliteit van je code en design. Liever een klein aantal goed uitgewerkte functionaliteiten dan een groot aantal half-werkende extra's. Bespreek je plannen dus eerst met de assistenten.

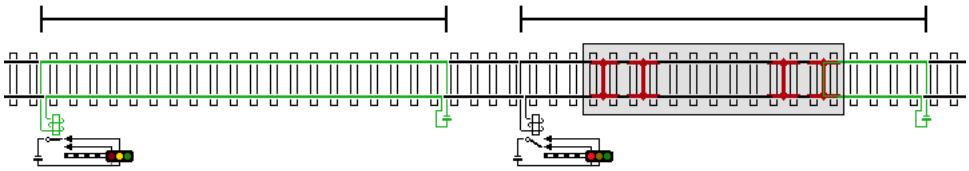
3.1 Controlesysteem

De functionaliteiten van het controlesysteem splitsen we op in twee groepen: de infrastructuur (INFRABEL) en het beheer (NMBS).

INFRABEL staat in voor de communicatie tussen de software en de modelbouwhardware (of de simulator) en het doorlopende beheer van de spoor- en treininfrastructuur.

Ten eerste implementeer je hier **Command & Control**: het aansturen van locomotieven, wissels en signalisatie. Locomotieven moet je kunnen starten en stoppen, en hun rijsnelheid en rijrichting veranderen. Wissels en signalisatie moeten van stand kunnen veranderen.

Ten tweede bestaat het doorlopende beheer uit een **veiligheidsfunctie** die botsingen voorkomt. Dit systeem bouwt op signalisatie (verkeerslichten/seinen) en een systeem voor 'blokdetectie'.



Figuur 6: Op deze afbeelding zie je twee detectieblokken. Wanneer er zich een trein bevindt op het tweede detectieblok, moet het licht van het eerste blok oranje worden. Treinen die een oranje licht passeren moeten vertragen, omdat het volgende licht rood kan zijn. Tegen de tijd dat ze het volgende licht bereiken kan dit wel al terug groen zijn.

De signalisatie in de echte opstelling is puur kosmetisch: er is geen mechanisme in de locomotief ingebouwd om hem aan een rood licht te doen stoppen. Je moet het naleven van de signalisatie dus zelf voorzien. Zowel in de realiteit als in modelspoor is dit systeem gebaseerd op detectieblokken. Een detectieblok is een deel van het spoor dat minstens zo lang is als de langste trein. Per blok is er een mechanisme dat detecteert of er een trein op aanwezig is. Aan het begin van elk blok staat een licht, zoals geïllustreerd in figuur 5. Als een trein een blok binnenrijdt wordt het licht aan het begin van dat blok rood. Wanneer de trein het blok verlaat, wordt het terug groen. Op deze manier kunnen er zich nooit twee treinen in één blok bevinden. Daarenboven zal je ook een oranje licht moeten gebruiken, zoals getoond in figuur 6: dit signaleert dat het volgende licht mogelijk rood is, dus dat de trein moet vertragen zodat een volledige stop aan het volgende licht mogelijk is.

Waar de blokken staan maakt deel uit van de opstelling, zoals getoond werd in figuur 3. Merk op dat in de gegeven opstelling niet elk spoorsegment deel uitmaakt van een blok (de witte segmenten). Het is ook belangrijk dat wissels geen deel van een blok mogen zijn, om impasses of *deadlock* te vermijden. Let dus ook op dat een trein niet bovenop een wissel blijft wachten.

Zoals eerder vermeld, kan je ervoor kiezen om INFRABEL te laten draaien op een **Raspberry Pi**, of op je eigen computer. In ieder geval is het belangrijk een duidelijke scheiding tussen INFRABEL en NMBS te hebben. Het design van de API tussen beide componenten is van cruciaal belang.

Tijdens de verdediging van je project zal je het moeten demonstreren *zonder* de hardware, maar met de simulator. Het moet dus mogelijk zijn te kiezen tussen **de simulator en de hardware**. Hiervoor kan je eventueel een abstractielaag voegen tussen INFRABEL en de simulator of de hardware. Je mag ook aanpassingen maken aan de simulator, om deze meer zoals de hardware te laten functioneren, zolang je deze aanpassingen documenteert in je specificatie.

NMBS beheert alles bovenop de infrastructuur.

Ten eerste moet het mogelijk zijn de **lay-out en toestand van de opstelling** in te geven. Deze informatie moet persistent worden opgeslagen, zodat je deze niet telkens opnieuw moet ingeven.

Ten tweede heb je **trajectcontrole** nodig. Het moet mogelijk zijn een traject uit te stippelen, en een trein dit te laten volgen. Dit betekent dat je NMBS-component continu commando's naar INFRABEL moet sturen om de locomotief te starten en stoppen (of versnellen en vertragen), de signalisatie uit te lezen, en de wissels te verzetten.

Ten derde vragen we voor **automatische trajectberekening**: een gebruiker moet kunnen de eindbestemming van een trein ingeven, waarna het traject ernaartoe automatisch berekend wordt.

Daarnaast kan je deze component nog uitbreiden met **extra functionaliteit**, zoals automatisch routes berekenen voor meerdere treinen die tegelijk rijden, een maximale rijsnelheid opleggen per segment, omgaan met calamiteiten (botsing, brand...), of treinen brandstof laten verbruiken die ze moeten bijtanken in vaste herlaadpunten. Deze zal beloond worden in de uiteindelijke quotering. Bespreek deze wel met de assistenten voor je eraan begint.

Dit zijn dus de minimale vereisten:

Command & Control (INFRABEL) Het aansturen van de locomotieven (start, stop, versnel, vertraag, vooruit en achteruit), wissels (verander richting) en signalisatie (rood, groen).

Hardware en Simulator (INFRABEL) INFRABEL moet zowel de echte hardware als de simulator kunnen gebruiken.

Veiligheid (INFRABEL) Wanneer een trein zich op een detectieblok bevindt wordt het bijhorende licht rood, en treinen stoppen voor rode lichten. Treinen wachten niet buiten detectieblokken.

Invoer en persistentie (NMBS) De lay-out van de opstelling moet kunnen ingevoerd, geopend en opgeslagen worden.

Trajectcontrole (NMBS) Het moet mogelijk zijn een uitgestippeld traject door een trein te laten volgen. Hierbij worden locomotief, wissels en signalisatie continu aangestuurd.

Trajectberekening (NMBS) Wanneer een eindbestemming ingegeven wordt door de gebruiker, moet het traject ernaartoe automatisch berekend kunnen worden.

Communicatie INFRABEL en NMBS moeten duidelijk van elkaar gescheiden zijn, en communiceren via een netwerkverbinding.

Optioneel: Raspberry Pi Je mag INFRABEL draaien op een Raspberry Pi, maar dit is niet verplicht.

3.2 Grafische gebruikersinterface (GUI)

Ten slotte moet een grafische gebruikersinterface (GUI) de eindgebruiker toelaten de toestand van het spoornetwerk te zien, en je functionaliteit te gebruiken. Daarnaast kan het ook een logboek bijhouden van evenementen (bv. trein 3 gestart, licht 2 op rood). Voor deze component kan je de “Racket Graphical Interface Toolkit” gebruiken.

Hierbij moet je de echte functionaliteit van je programma scheiden van de grafische interface, dit komt de modulariteit van het programma ten goede. Een goede vuistregel: stel dat je een andere interface moet toevoegen (bv. één voor de smartphone), hoeveel van je code kan je dan hergebruiken en hoeveel moet je aanpassen?

Met toestemming van de assistenten mag je in de plaats van een Racket GUI een website of webapplicatie bouwen. Hiervoor moet je nog steeds binnen Racket werken, bekijk en bestudeer dus eerst de bibliotheek voor webapplicaties van Racket.

3.3 Niet-functionele vereisten

Bij het ontwikkelen van een groot softwareproject is het belangrijk dat je de kwaliteit van je code niet uit het oog verliest. Bijgevolg zullen we rekening houden met de codekwaliteit bij het evalueren van je project. Een groot project kan snel leiden tot ingewikkelde en moeilijk begrijpbare code. Je project opsplitsen in logische modules, het gebruik van goede ADT’s, en het ontwerp van goede API’s kunnen dit voorkomen. Daarnaast wordt verwacht dat je je programma test door middel van “unit tests”, en dat je rekening houdt met de gegeven kwaliteitseisen en codeconventies.

Tests Unit testing is een techniek waarbij softwaremodules of stukjes broncode ("units", bv. een module of een ADT) afzonderlijk getest worden. Bij unit testing worden voor iedere unit verschillende tests ontwikkeld, die alle functionaliteit van dat unit verifiëren. Deze doorlopen een aantal test cases, waarbij telkens de echte output wordt afgetoetst tegen een verwachte waarde. Het doel van unit testing is om functionele units (modules, ADT's) onafhankelijk van elkaar te kunnen testen op correcte werking. Zo weet je in welk deel van je programma een fout is geslopen.

We verwachten dat je unit tests ontwikkelt voor de cruciale functionaliteit (o.a. veiligheid en trajectberekening). Telkens je iets verandert aan je code, kan je de overeenkomstige unit tests doorlopen om te controleren of alles nog werkt. Een batterij aan tests zal van onschatbare waarde zijn wanneer je je code aanpast naar het einde van het project toe. Je moet deze tests ook toevoegen aan de code die je inlevert.

Je kan de RackUnit bibliotheek[1] gebruiken. Gebruik geen `display` statements: deze tonen wel wat gebeurt, maar geven niet aan wat de verwachte waarde is en of de verkregen waarde hiermee overeenkomt. Het is ook niet nodig elke kleine functie te testen, focus in de plaats op het testen van blokken functionaliteit, zoals trajectberekening of de veiligheidsfunctie.

In het ideale geval is elke test case onafhankelijk van de rest van het systeem. Eventueel worden hiertoe stukken code geschreven die andere stukken van je programma "simuleren", om zo één stuk te kunnen testen zonder afhankelijkheden op een ander stuk. Dit noemt *mocking*, en kan nuttig zijn om bijvoorbeeld tests te schrijven die onafhankelijk zijn van netwerkcommunicatie.

Kwaliteitseisen en codeconventies Algemene informatie omtrent de kwaliteitseisen vind je in het document "Kwaliteitseisen Programeerproject" op PointCarré. Lees dit document grondig door! Hou rekening met elke eis die in dit document staat. Bovendien zal je in de cursus "Algoritmen en Datastructuren 2" (prof. De Meuter) ondervinden hoe goede Scheme-code, met gebruik van modules, eruit ziet.

Enkele tips specifiek voor dit project:

- Kies bij elk ADT de gepaste abstractietechniek en implementatie.
- Splits je programma op in modules: logische onderdelen met elk een eigen specifieke functionaliteit. Een software-ontwerp met goed gekozen modules zal je veel tijd besparen bij het ontwikkelen van je project, en je project robuuster en begrijpelijker maken.
- Denk bij grote ADT's na of ze niet kunnen gesplitst worden in kleinere, lossere onderdelen. Kleine ADT's die bijna altijd samenwerken kan je dan weer samenvoegen. Dit helpt bijvoorbeeld om de grafische interface gescheiden te houden van de rest van je programma. Kleine veranderingen aan je GUI zullen zo weinig tot geen impact hebben op de rest van je programma.
- Denk goed na over de interfaces (API's) tussen de verschillende componenten. Hou rekening met het feit dat deze op verschillende machines zullen draaien.
- Een goede voorstudie zal je al een heel eind op weg brengen.

4 Deliverables

Deze sectie beschrijft wat je moet indienen voor tweede zittijd:

[1] `http://docs.racket-lang.org/rackunit/`

Code Alle broncode van je toepassing, inclusief tests. Zorg ervoor dat deze uitvoerbaar is vanuit Dr. Racket.

Documenteer je code zodat deze gemakkelijk te begrijpen is. Laat geen debugging code (bv. print statements) in de broncode staan. Het maakt je broncode slordig en schaadt de leesbaarheid.

Het is toegestaan om code van derden te gebruiken, maar niet die van medestudenten! Vermeld steeds van waar de code komt en wie de originele auteurs zijn, *in de code én in je specificatiedocument*. In elk geval moet zeer duidelijk zijn welke code van jezelf is en welke overgenomen is. Dit is een individueel project, het overnemen van broncode van medestudenten of vorige studenten is plagiaat en wordt niet geduld!

Handleiding (5 tot 10 pagina's inclusief screenshots) Een handleiding van je toepassing gericht op eindgebruikers. Dit bevat:

- Een algemene omschrijving van je toepassing.
- Hoe deze te starten: welk Racket-bestand moeten we starten, wordt er bepaalde input verwacht, waar wordt het spoornetwerk gedefinieerd...
- Een beschrijving van alle functionaliteit en hoe deze gebruikt kan worden.

Gebruik screenshots waar nodig.

Let er op dat je alle nodige bibliotheken bundelt met je broncode, bv. wanneer je bibliotheken via de "Collection Paths" van Racket hebt gebruikt. Indien nodig, leg ook uit welke extra pakketten of bibliotheken moeten geïnstalleerd worden (bv. via de Package Manager van Racket).

Specificatie (max. 10 pagina's) Een specificatie van je toepassing gericht op ontwikkelaars. Dit bevat:

- Een algemene omschrijving van je toepassing.
- Beschrijf de gehele software-architectuur. Leg uit hoe de verschillende componenten van elkaar gescheiden zijn. Voeg hiervoor een diagram toe en bespreek het.
- Beschrijf elke component apart. Leg uit wat de taken of functionaliteiten van deze component zijn. Je kan hierbij zowel de ADT's van de component oplijsten als de algoritmes die erin gebruikt worden (bv. voor trajecten te berekenen of de veiligheid te garanderen).
- Beschrijf de interfaces of communicatieprotocols tussen de componenten. Gebruik waar nodig een diagram en licht dit toe.

Je specificatie zou allesomvattend moeten zijn: het lezen van dit document zou moeten volstaan om je code te kunnen begrijpen. Stel je bij het schrijven van je document voor dat iemand die nu in derde bachelor zit in de zomervakantie aan de slag moet kunnen met jouw project.

Inleveren Converteer alle documenten naar PDF-formaat. We aanvaarden geen Word, OpenOffice of andere bestanden. Steek al je documenten en code (ook die van derden) in een mapje met je voor- en achternaam (bv. jan.peeters), en maak hiervan een zip-bestand met dezelfde naam. Upload dit via PointCarré vóór de deadline.

Voorbladen voor de documenten zijn niet nodig; een titel, je naam en je rolnummer volstaan. Voeg wel aan elk document een korte inleiding en conclusie toe. Gebruik voor alle documenten de automatische spellingscontrole van je tekstverwerker om schrijffouten te voorkomen.

De deadline voor dit project is **maandag, 21 augustus om 8:00**. Deze deadline staat vast, er wordt geen uitstel gegeven. *Per dag* dat je te laat indient, gaan er *twee punten* van je eindscore af. Als je meer dan 4 dagen te laat indient, wordt je als afwezig gequoteerd.

Je bent verplicht *alle* delen (code, handleiding en specificatie) in te dienen. Ook de mondelinge verdediging is verplicht. Als je één van de onderdelen niet indient en/of afwezig bent op de verdediging, zal je als afwezig gemarkeerd worden voor het hele vak.

Demonstratie en verdediging Tijdens de examenperiode organiseren we een mondeling examen van het project. Elke student krijgt 25 minuten:

- 10 minuten presentatie. Hiervoor raden we aan een presentatie te maken van zo'n vijf tot tien slides. Zorg ervoor dat je zeker het diagram van de software-architectuur uit je specificatie toont. Leg aan de hand hiervan uit welke componenten je programma bevat, wat hun functionaliteit is, en hoe ze met elkaar communiceren.
- 5 minuten demonstratie. We raden je aan om een klein scenario uit te werken. Test dit zeker op voorhand! Tijdens je verdediging kan je enkel demonstreren met de simulator, maar je mag daarnaast ook een filmpje laten zien van je project op de echte hardware. Let op de tijd; je hoeft niet alle functionaliteit te demonstreren.
- 10 minuten vragen. Hierbij zullen we je enkele vragen stellen op basis van je handleiding, specificatie en presentatie.

Tabel 1: Een overzicht van de functies die de interface van de simulator voorziet. Je code voor de INFRABEL- en NMBS-gedeeltes mogen enkel deze functies gebruiken. Indien je wijzigingen aanbrengt aan de simulator, moeten de signaturen van deze functies intact blijven.

Functie	**Signatuur**	**Beschrijving**
`start-simulator`	`( ∅ → ∅ )`	Maakt een thread aan die de verplaatsing van de locomotieven in de achtergrond doorvoert door middel van een loop.
`stop-simulator`	`( ∅ → ∅ )`	Stopt de simulator thread. Deze functie blijft wachten tot de thread effectief gestopt is.
`get-loco-speed`	`( symbol → number )`	Geeft de snelheid terug van de gegeven locomotief. Elke locomotief is geïdentificeerd door een uniek symbool. Initieel staat elke locomotief stil.
`set-loco-speed!`	`( symbol number → ∅ )`	Past de snelheid aan van de gegeven locomotief. Je kan een locomotief achteruit laten rijden door een negatieve waarde mee te geven.
`get-loco-detection-block`	`( symbol → symbol )`	Geeft het detectieblok terug, geïdentificeerd door een uniek symbool, waarop de gegeven locomotief momenteel gedetecteerd is. Indien de locomotief niet op een detectieblok staat, wordt `#f` teruggegeven.
`get-switch-position`	`( symbol → number )`	Geef de stand van de gegeven wissel (geïdentificeerd door een uniek symbool) terug. Een wissel kan in stand 1 (initiële waarde) of 2 staan.
`set-switch-position!`	`( symbol number → ∅ )`	Verander de stand van de gegeven wissel.