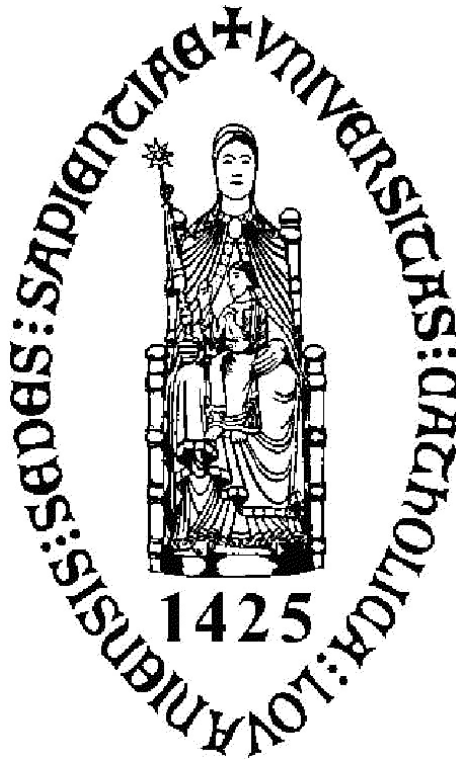


P&O COMPUTERWETENSCHAPPEN
VERSLAG TEAM PLATINUM

Academiejaar 2011–2012

PAC-MAN IN DE ECHTE WERELD,
MET BEHULP VAN LEGO MINDSTORMS



Thomas De Waelheyns
Florian Hendrickx
Michiel Huygen
Ruben Lapauw
Christophe Van Ginneken

Inhoudsopgave

1	Inleiding	1
2	Probleemstelling	2
2.1	Demo 1	2
2.2	Demo 2	2
2.3	Demo 3	3
2.3.1	Verkenning	3
2.3.2	Pac-Man	3
3	Scheidsrechtercommissie	4
3.1	Spelregels	4
3.1.1	Spelwereld	4
3.1.2	Pac-Man	4
3.1.3	Ghosts	5
3.2	Communicatieprotocol	5
3.2.1	Voordelen	5
3.2.2	Nadelen	5
3.2.3	Analyse van het GhostProtocol	6
3.2.4	Aanpassingen Demo 2	7
3.2.5	Aanpassingen Demo 3	7
3.2.6	Finale evaluatie	7
4	Robot	8
4.1	Demo 1	8
4.1.1	Fysiek ontwerp	8
4.1.2	IR-sensor	8
4.2	Demo 2	11
4.3	Demo 3	11
5	Strategie	12
5.1	Achtervolgstrategie	12

5.2	Verkenstrategie	13
5.3	Samenwerking en onzekerheid	15
5.4	Effectiviteit	15
6	Softwaredesign	16
6.1	Robot	16
6.1.1	Driver	18
6.1.2	Grids en Sectoren	18
6.1.3	GhostRobot	21
6.1.4	Performantie	22
6.2	PC	26
6.2.1	Externe componenten	27
6.3	Refactoring	29
6.3.1	Van pure refactoring naar test coverage	30
7	Simulator	31
7.1	Demo 1	31
7.1.1	Aanpassingen	31
7.1.2	Nieuwe Functionaliteit	32
7.1.3	Simulatormodi	32
7.1.4	Mini-Simulator	34
7.2	Demo 2	35
7.3	Demo 3	35
7.4	Simulatie versus realiteit	35
8	Conclusies	36
8.1	Demo 1	36
8.1.1	Feedback	36
8.1.2	Reflectie	37
8.2	Demo 2	37
8.2.1	Feedback	37
8.2.2	Reflectie	37
8.3	Demo 3	38
8.3.1	Reflectie	38
9	Procesbeschrijving	40
9.1	Demo 1	40
9.2	Demo 2	41
9.3	Demo 3	41
9.4	Unit testen	42

10 Werkverdeling	45
10.1 Ruben	45
10.2 Michiel	45
10.3 Thomas	46
10.4 Florian	46
10.5 Christophe	46
11 Kritische analyse	47
A Beoordelingen	50
A.1 Demo 1	50
A.1.1 Verslag	50
A.1.2 Presentatie	51
A.1.3 Positieve punten	51
A.2 Demo 2	51
A.2.1 Verslag	51
A.2.2 Presentatie	54
A.2.3 Positieve punten	54
A.3 Demo 2: Beoordeling Team Rood	54
A.3.1 Verslag	54
A.3.2 Presentatie	57
A.3.3 Positieve punten	57
B Grafische User Interface	58
B.1 Dashboard	58
B.2 Simulator	59
B.3 RASH - Robot Administratie Shell	61
C Klasse Diagrammen	64
D Planning	72

Samenvatting

Tijdens de uitwerking van dit jaar-overschrijdend groepswork, bouwen wij een autonome robot met behulp van Lego Mindstorms. Voor de programmatie wordt beroep gedaan op Lejos, een Open Bron project dat een minimale JAVA virtuele machine heeft gemaakt die de plaats kan innemen van de standaard programmatie omgeving van Lego.

Naast de doelstelling om kennis te maken met het ontwikkelen van een autonome robot, willen we in dit project ook ervaring opdoen i.v.m. het werken in teamverband aan een middelgroot softwareproject. Hierbij zijn organisatie van werk, planning, analyse, architectuur,... belangrijke begrippen.

In het tweede semester wordt hier nog eens de nood tot samenwerking met de andere teams aan toegevoegd. Het traject van de autonome robot blijft behouden, maar de verschillende robots zullen nu moeten samenwerken om een gemeenschappelijk doel te bereiken. Om deze samenwerking in goede banen te leiden werd een scheidsrechtercommissie in het leven geroepen om beslissingen te nemen die voor alle teams van belang zijn. Het doel van dit semester is het insluiten van een Pac-Manrobot bestuurd door het didactisch team met behulp van vier autonome *ghosts*.

In het eerste semester werkten we reeds in een parallel traject aan een simulatieomgeving. Aan de hand van deze omgeving waren we in staat om met meerdere teamleden in parallel software voor de robot te ontwikkelen en te testen zonder nood aan een effectieve fysieke robot. Nu is het ontwikkelen van deze simulator deel geworden van de opdracht en dienen we dus de nodige aanpassingen te doen om het Pac-Manspel in de computer te simuleren. Het gebruik van een simulator is evident, aangezien we niet kunnen verwachten alle fysieke testen uit te voeren met de 4 teams tegelijk.

Hoofdstuk 1

Inleiding

Het project wordt zoals in het eerste semester begeleid door het gebruik van tussentijdse demo's. Het doel is om zo optimaal mogelijk samen te werken met vier teams om de Pac-Man in te sluiten, hoewel we te allen tijden autonoom beslissingen nemen.

Het doel van de eerste demo is een vereenvoudigde vorm van het einddoel waarbij we een stilstaande Pac-Man zoeken binnen het doolhof. De simulator dient reeds beschikbaar te zijn en moet drie virtuele robots de fysieke laten bijstaan. Voor de tweede demo mag de commissie beslissen welke doelstellingen passen in het verdere proces richting het einddoel.

Tijdens het eerste semester zijn we er in geslaagd om zeer herbruikbare code te produceren: de architecturale concepten, zoals *Robot*, *Navigator* en *Model*, zijn duidelijk binnen het team en de verdere uitwerking van deze concepten in het tweede semester werd ervaren als een natuurlijke evolutie. Aangezien we te maken hebben met een grote diversiteit aan kleine taken, gebeurt het grootste deel van de taakverdeling week op week.

Het verslag is anders opgebouwd ten opzichte van het vorige semester: er werd gekozen voor een onderverdeling in subsecties per demo binnen een context van hoofdstukken. De opbouw van deze hoofdstukken is erg gelijkend hoewel natuurlijk de inhoud veranderd is t.o.v. het eerste semester. Nieuw zijn de stukken over de strategie i.v.m. de opgegeven spelomgeving, over de samenwerking en over de finale analyse van het jaarproject.

Hoofdstuk 2

Probleemstelling

De doelstellingen van de demo's worden hier achtereenvolgens beschreven. Zij volgen stapsgewijs een ontwerpproces dat tracht een autonome robot op te leveren die, samen met drie andere *ghosts*, probeert de Pac-Man in te sluiten.

2.1 Demo 1

Het probleem dat we eerst aanpakken is het zo efficiënt mogelijk in kaart brengen van een onbekende omgeving met behulp van de vier *ghosts*. Belangrijk hierbij is natuurlijk dat de gedetecteerde wereld overeenkomt met de realiteit. Tijdens de demo gebruiken we een zgn. hybride simulator: onze eigen robot bestaat in de fysieke wereld, de andere drie zijn virtueel en worden vertegenwoordigd door drie virtuele *ghosts* die in de simulator *leven*.

De *ghosts* hebben geen enkele informatie over het doolhof waarin ze zich bevinden, ze kennen wel hun eigen globale positie, maar dan weer niet hun oriëntatie. Verder staat de Pac-Man stil in het parcours en dient hij enkel gevonden te worden. We dienen ook aan te tonen dat het door de commissie afgesproken communicatieprotocol geïmplementeerd is.

Elk team wordt tijdens de demo afzonderlijk beoordeeld.

2.2 Demo 2

De doelstellingen voor demo 2 werden vastgelegd met de resultaten van de eerste demo in het achterhoofd. Deze waren voor sommige teams niet zo goed. Om de kern van de opdracht niet op de lange baan te schuiven wordt in demo 2 toch getracht een stap in de goede richting te nemen. Om de doelstellingen voor demo 2 haalbaar te houden, werd beslist om de Pac-Man nog steeds stil te laten staan. De tweede demo draait daarom volledig rond het samenwerken met de andere teams.

Dit is duidelijk een van de essentiële aspecten voor de uiteindelijke demo dat niet uit het oog mag verloren worden.

We volgen de raad op om op voorhand samen te zitten met de andere drie groepen. Het behalen van de doelstellingen is een inspanning van de vier teams samen. Om onverwachte resultaten te voorkomen is samen testen dan ook een noodzaak.

Het doel van deze demo ligt dus in het samen verkennen van het doolhof en het samen insluiten van een stilstaande Pac-Man. We doen dit zowel met de virtuele versie van de simulator als met de gedistribueerde. Initieel krijgt ook elk team de kans om eerst nogmaals zijn hybride simulator te tonen en de vooruitgang ten opzichte van de eerste demo aan te duiden.

2.3 Demo 3

2.3.1 Verkenning

De robots rijden in een onbekend doolhof. Alle info die verzameld wordt door de vier ghosts draagt bij tot het creëren van een individuele map die mogelijk inconsistenties bevat. Alle juiste info over het doolhof zal direct bijdragen tot het hoofddoel, namelijk het vangen van de Pac-Man. Elke robot communiceert zijn gevonden informatie en probeert zo veel mogelijk informatie te verzamelen over de voor hem onbekende wereld. Elke fout van één van de robots kan reeds fataal zijn voor het kunnen opbouwen van een beeld van de omgeving. We proberen dus een zo robuust mogelijke robot te maken die zo efficiënt mogelijk zijn eigen info verzamelt en met de nodige voorzichtigheid andere informatie hierin verwerkt.

2.3.2 Pac-Man

Het Pac-Man spel zelf kunnen we winnen door alle vluchtwegen van de Pac-Man af te sluiten. Dit houdt in dat alle omliggende sectoren door een *ghost* bezet of door een muur afgeschermd worden. Men wint enkel als alle robots die meewerken aan de insluiting ook effectief aangeven dat ze gewonnen hebben via hun grafische interface. De probleemstelling vereist dus een afdoende samenwerking, waarbij een consensus moet bereikt worden over de verzamelde informatie en de te volgen strategie.

Hoofdstuk 3

Scheidsrechtercommissie

Deze commissie staat in voor het nemen van beslissingen die betrekking hebben op afspraken tussen de verschillende teams en het didactische team. Hierbij kunnen we deze beslissingen groeperen in:

- De verdere regels waarmee de spelomgeving beperkt wordt.
- Afspraken die noodzakelijk zijn voor de samenwerking van de teams.

3.1 Spelregels

3.1.1 Spelwereld

De spelwereld bestaat uit aangepaste panelen uit het eerste semester. Er is een raster van witte lijnen geïntroduceerd waar muren kunnen staan. Dit komt neer op een mogelijke herschaling van de panelen met een factor vier. Deze nieuwe minimeenheid aan oppervlakte noemen we sectoren. Deze sectoren zijn belangrijk voor de plaatsbepaling. Het doolhof is volledig ommuurd en de absolute afmetingen worden op voorhand opgegeven.

3.1.2 Pac-Man

De Pac-Man wordt zichtbaar gemaakt door een infrarood-beacon dat kan opgemerkt worden met behulp van de voorziene IR-sensor. Het didactisch team kiest waar deze vertrekt en bestuurt hem tijdens de demo's. Hij mag enkel bewegen op sectoren waar zich geen *ghosts* bevinden.

De beweging van Pac-Man is beperkt tot het rechtdoor oversteken van de witte lijnen die de sectoren scheiden.

Insluiting van Pac-Man is gedefinieerd als het niet meer kunnen bewegen van de Pac-Man of het ingesloten zijn in een doodlopend pad.

3.1.3 Ghosts

De ghosts vertrekken op de vier hoekpunten van het doolhof, zodat de verkenning van het doolhof zo snel mogelijk van start kan gaan. De oriëntatie van de robot is onbekend hoewel de hoek t.o.v. het assenstelsel wel een veelvoud is van 90 graden.

3.2 Communicatieprotocol

We verwijzen naar de officiële documentatie van het *Ghost Protocol*. Ten tijde van het schrijven van dit verslag was versie 2.2 beschikbaar. Het *Ghost Protocol* is het resultaat van een werkgroep die aangesteld werd door de scheidsrechtercommissie. In dit deel van het verslag belichten we de positieve en de negatieve punten van het voorgestelde protocol en bespreken we de voor- en nadelen ervan m.b.t. onze eigen ontwikkeling.

3.2.1 Voordelen

Het collaborative diffusion algoritme waarop onze implementatie gebaseerd is, heeft zeer weinig informatie nodig van de andere robots. Het algoritme werkt zelfs met enkel omgeving-gerelateerde informatie en de positie van de andere robots in het doolhof. De minimale subset van commando's die in het protocol moeten aanwezig zijn voor onze implementatie zijn:

- [naam] position [p:coord]
- [naam] discover [p:coord] [w:int]
- [naam] pacman [p:coord] [a:angle]
- [naam] barcode [p:coord]
- [naam] captured

Deze functionaliteit is aanwezig in het huidige protocol, dus is het compatibel met onze verkozen implementatie.

3.2.2 Nadelen

In het algemeen ondervinden we geen nadelen van de beslissingen die genomen zijn. Aangezien we voldoende informatie kunnen verzamelen met een subset, zal het volledige protocol slechts een marginale implementatie-meerkost met zich meebrengen. Volgens ons kan het protocol wel sterk vereenvoudigd worden. Aangezien dit slechts vervelend is en niet als nadeel beschouwd kan worden, voegen we onze opmerkingen toe in de vorm van een analyse.

3.2.3 Analyse van het GhostProtocol

De *JOIN* procedure lijkt overbodig. Wanneer een *ghost* laattijdig *JOIN*'t, zal hij gewoon de volgende berichten horen en zijn wereldbeeld beginnen opbouwen. Eventueel kan een commando toegevoegd worden om reeds gestuurde informatie opnieuw te sturen. In een latere versie van het protocol werd het *SHOWMAP* commando toegevoegd, wat deze functie gedeeltelijk vervult. Dit overlapt gedeeltelijk met het *JOIN* principe. Bovendien eist de procedure dat er exact 4 spoken zijn. Indien een *ghost* uitvalt, of een extra *ghost* op het kanaal komt, zorgt dit voor problemen. Hiervoor werd een 'override' commando toegevoegd, om toch te starten indien er een *ghost* geen *JOIN* stuurt. Het protocol voorziet een beperking van 4 spoken, dit is volgens ons volledig onnodig, net als de *JOIN* procedure.

In plaats van dynamisch namen uit te wisselen kunnen evengoed vaste namen afgesproken worden, dit zou het protocol eenvoudiger maken. Er zijn tal van mogelijkheden: alle teams hebben een naam of de echte namen van de vier spoken kunnen gehanteerd worden. Een conventie lijkt ons hier ver boven een configuratie te verkiezen.

Voor het *DISCOVER* commando lijkt het beter om een *bitfield* te gebruiken voor de aanduiding van de muren. Omdat er zeer veel discover commando's gestuurd kunnen worden, kan dit de overhead op de communicatie verkleinen en zelfs het parsen vereenvoudigen. Met eenvoudige bit-wise shift operaties kan getest worden of een tegel een muur heeft of niet. Dit geldt tevens ook voor de commando's zelf. Ook deze waren beter vervangen door getallen, zodat alle informatie louter numeriek was en optimaler verstuurd zou kunnen worden.

Het *PLAN* commando is naar onze mening onvolledig. Het doorsturen van een pad is pas nuttig indien er ook een *election*-procedure voorzien zou zijn. Verder is zonder een tijdsynchronisatie en informatie over hoe snel de robot zijn pad aflegt al deze informatie nutteloos.

Voor ons algoritme is dit commando tevens volledig overbodig. Onze robot rijdt op enkel informatie van het doolhof, de positie van de andere robots en Pac-Man. De pad informatie zal initieel niet gebruikt worden in onze implementatie.

Nog een algemene opmerking is dat het protocol moeilijk te parsen is door combinatie van strings en numerieke waarden, inconsistenties in de structuur van de commando's, gebruik van verschillende soorten scheidingstekens, enz.

Oorspronkelijk was het idee dat het *Ghost Protocol* letterlijk gebruikt zou worden in het Maze Protocol - het Maze Protocol is een doorslag van het *Ghost Protocol* en wordt gebruikt om doolhoven te beschrijven, bvb. voor de simulator. Op deze manier zou de implementatie van het *Ghost Protocol* kunnen hergebruikt worden. Spijtig genoeg zijn er enkele aanpassingen doorgevoerd waardoor het nu niet meer 100% compatibel is met het *Ghost Protocol*. Deze aanpassingen waren naar onze mening beter doorgevoerd geweest op het *Ghost Protocol* zelf.

3.2.4 Aanpassingen Demo 2

Het *BARCODE* commando wordt vervangen door een *BARCODEAT* commando, dit is een vooruitgang aangezien er nu geen logica meer voorzien moet worden om de barcodes aan een positie te linken.

Er werd eveneens een *UNDOBARCODE* toegevoegd om teams toe te laten eerdere transmissies omtrent barcodes ongedaan te maken.

3.2.5 Aanpassingen Demo 3

De besproken versie betreft versie 2.2 van het *Ghost Protocol*. De belangrijkste aanpassingen voor de laatste demo zijn deze met betrekking tot een *reconnect*-procedure.

Ten eerste werd het scenario uitgewerkt met betrekking tot het ontvangen van een *JOIN* commando, indien we al bezig zijn. Door middel van het *RENAME* commando kunnen de oude robots onderscheiden worden van de robot die na falen opnieuw verbindt met de message queue en zich (opnieuw) aanmeldt met een *JOIN* commando. Deze robot zal uit de *RENAME* antwoorden kunnen opmaken dat hij een bestaande sessie vervoegt. Zelf gebruikt hij wel het *NAME* commando.

Ten tweede werd een commando in het leven geroepen waarmee robots informatie van elkaar kunnen vragen. De robots kunnen, bijvoorbeeld wanneer ze uitvallen, een *SHOWMAP* commando gebruiken, nadat ze terug toegelaten zijn tot het spel. Initieel werd voorgesteld om de bestaande commando's te hergebruiken. Dit zorgde blijkbaar voor problemen, waardoor ze voorzien werden van een *RE*-prefix. Tevens gaat het niet over het delen van de informatie die men als bekend beschouwt, maar om het heruitzenden van een log van alle reeds uitgezonden commando's. Echter deze definitie wordt verder gespecificeerd en het samenballen van informatie, bijvoorbeeld over één zelfde sector, is tevens toegestaan.

3.2.6 Finale evaluatie

Doorheen dit semester worstelde ons team vaak met vragen, opmerkingen of gaten in dit protocol. Tot op het laatste moment werden zaken toegevoegd die tegenstrijdig, onlogisch of ondoordacht waren. Alleen al het ontbreken van een *time-to-live* maakt sommige commando's inherent onbruikbaar. Verder denken we vooral dat er te veel manieren zijn tot interpretatie en dat dit de kans op mislukking vergroot, iets wat de commissie juist moest voorkomen. Gelukkig zal het merendeel van de functionaliteit tijdens de korte tijdspanne dat de demo in beslag neemt niet gebruikt moeten worden, waardoor alle mogelijke problemen zich waarschijnlijk niet zullen voordoen. We vinden het alleen een gemiste kans om te leren hoe een degelijk protocol had kunnen opgesteld worden.

Hoofdstuk 4

Robot

4.1 Demo 1

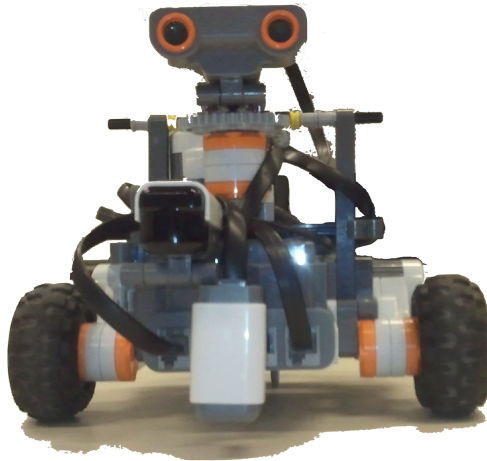
4.1.1 Fysiek ontwerp

Door de gewijzigde specificaties van de fysieke omgeving en de extra infrarood sensor, moesten enkele wijzigingen gemaakt worden aan de fysieke robot. Zo zijn de druksensoren verwijderd om een sensorpoort vrij te maken voor de infraroodsensor. Bovendien is het in dit project noodzakelijk om zo precies mogelijk door het midden van de sectoren te rijden - touch-sensoren zijn minder van toepassing in deze strategie. Aangezien het doolhof minder breed is (sectoren zijn 40 centimeter in plaats van 80) zijn de extra wielen weggehaald om het geheel iets slanker te maken.

Verder is ook de motor, die de sonar laat draaien, iets naar onder en naar het midden verplaatst. Hierdoor hebben we de tandwielen tussen de motor en de sensor kunnen elimineren waardoor de sensor rechtstreeks op de motor zit. Om eenvoudiger de uitvoer op het scherm van de robot te kunnen lezen, is de *NXT Brick* omgedraaid, zodat het scherm zich nu aan de onderzijde van de robot bevindt. Hiermee kunnen we de knoppen van de robot gemakkelijker bedienen. Figuur 4.1 toont het afgeslankte ontwerp voor het tweede semester.

4.1.2 IR-sensor

De infraroodsensor werd vooraan in het midden van de robot aangebracht, net boven de lichtsensor. Het uitlezen van deze sensor zorgde voor een aantal problemen. De maximale gemeten afstand kwam niet overeen met de specificatie van de IR-bal en -sensor. Het bleek dat de Lejos implementatie verouderd was en niet alle mogelijkheden van de sensor ondersteunde. Met behulp van de broncode van Lejos versie 9.0 en enkele aanpassingen, is ons team er in geslaagd de verschillende modi van de sensor te activeren.



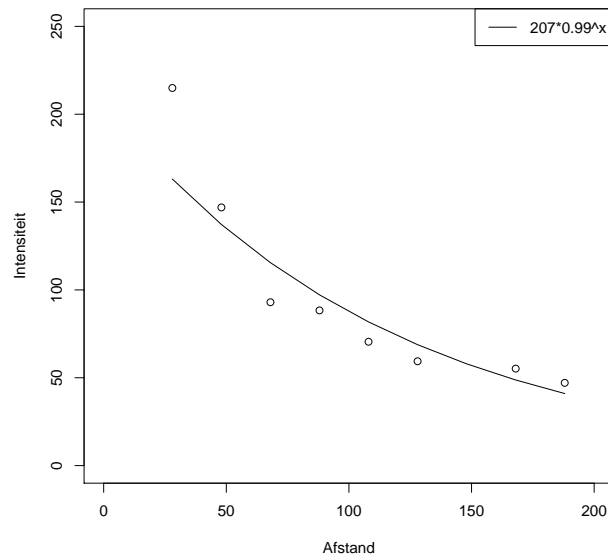
Figuur 4.1: Afgeslankte fysieke ontwerp voor het tweede semester.

Uit testen bleek dat er voor de bal verschillende AC modi en één DC modus is. De sensor heeft één AC-modus en één DC-modus. De AC-modus van de bal die speciaal voor de sensor ontwikkeld was, werkte uiteindelijk het beste. De sensor had dan een reikwijdte van 5 meter in tegenstelling tot ongeveer 80 cm bij andere modi.

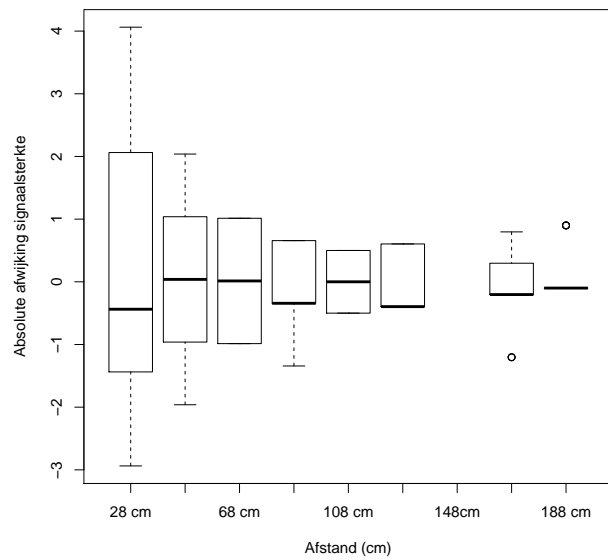
In figuur 4.2 kan men zien dat de gemeten lichtsterkte exponentieel afneemt ten opzichte van de afstand tot de lichtbron. Hieruit zou men een schatting kunnen maken van de afstand tot de bron. In grafiek 4.3 kan men de variatie van de IR-sensor zien voor verschillende afstanden. We zien dat de varianties zeer klein zijn in vergelijking met de lichtsterkte. Een afstandsschatting lijkt dus goed mogelijk.

Uit testen blijkt dat een significante hoeveelheid van het IR licht via weerkaatsingen door de infraroodsensor gedetecteerd wordt. De sterkte is bij weerkaatsing herkenbaar lager dan bij een rechtstreekse detectie (zie grafiek 4.4). Men kan dus eventueel ook aan de hand van de signaalsterkte herkennen of de Pac-Man zich bvb. achter een hoek bevindt. In dit geval kan deze meting zelfs best genegeerd worden. Het reconstrueren van een weerkaatsingspatroon is een zeer moeilijke opgave.

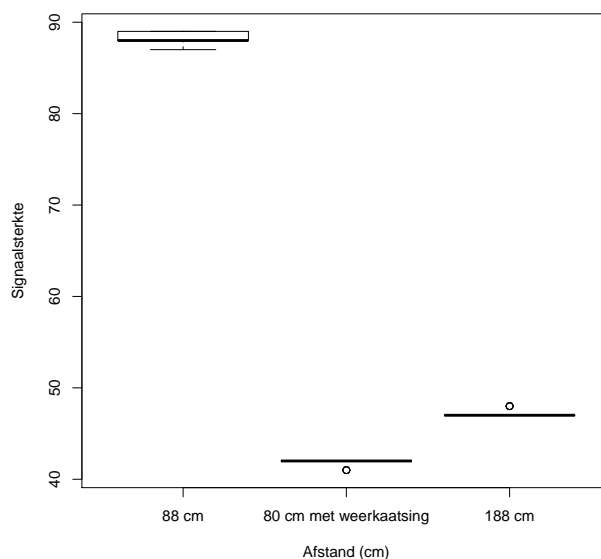
Een nadeel van zo'n filtering is dat deze de afstand waarop geschat kan worden inperkt. Zoals men kan zien op de grafiek 4.4, kunnen we geen afstanden van meer dan 160 cm (4 sectoren) onderscheiden van weerkaatsingen. Voor demo 2 en 3 zal moeten afgewogen worden wat de beste oplossing is: een grotere gemeten afstand maar misschien niet nauwkeurig, of een grotere zekerheid garanderen. Hier zal ook rekening gehouden moeten worden met de andere teams.



Figuur 4.2: De meetwaarden van de IR-sensor volgens een bepaalde afstand



Figuur 4.3: Sensorwaarden van de IR-sensor bij verschillende afstanden



Figuur 4.4: Sensorwaarden van de IR-sensor bij weerkaatsing en verschillende afstanden

4.2 Demo 2

Aangezien in demo 1 bleek dat de lichtsensor niet goed functioneerde, zullen we naar demo 2 toe de lichtsensor opnieuw van dichterbij bekijken. Het is moeilijk om de exacte condities van de demo na te bootsen, maar we kunnen wel de in de databank opgeslagen gegevens analyseren. Tevens zullen andere pistes bewandeld worden, op zoek naar een stabiel resultaat. Een eerste test zal opnieuw het plaatsten van een *kapje* zijn, zoals in het eerste semester getest werd. Deze aanpak werd toen niet gebruikt vanwege de hoogteverschillen in het parcours.

4.3 Demo 3

Aangezien afgesproken werd in de commissie om de robots te voorzien van een vlaggetje waarop hun kleur bekend gemaakt wordt, werd dit aan onze robot bevestigd. Verder werden er geen aanpassingen meer doorgevoerd, aangezien de lichtsensor bij demo 2 werkte naar verwachting.

Hoofdstuk 5

Strategie

Aangezien wij ons slechts sinds enkele maanden begeven op het terrein van autonome robots en artificiële intelligentie, zou het dom zijn te veronderstellen dat er ons nog niemand is voorgegaan. We hebben daarom een literatuurstudie rond de concepten van Pac-Man en autonome *ghosts* gedaan.

We vonden een paper omtrent *Collaborate Diffusion* [1]. Hierin wordt voorgesteld om een vorm van *Hill Climbing* toe te passen om zo verschillende autonome *ghosts* toe te laten samen te werken om een Pac-Man te achtervolgen. Hierbij hebben zij geen nood aan bijkomende onderlinge communicatie naast de informatie over de omgeving.

Ondanks het feit dat we geen overeenstemming met de andere teams konden bereiken om allen dit algoritme te implementeren, blijft het een zeer interessant algoritme om toe te passen voor het bepalen van ons eigen pad.

5.1 Achtervolgstrategie

Wanneer de Pac-Man gezien wordt, stuurt deze een *geur* uit. Deze plant zich langs gekende sectoren voort. Dit wordt gedaan door het gemiddelde van de 4 omliggende sectoren te berekenen, indien er een muur staat is de waarde 0. Zo ontstaat er een *hoogte-kaart* met aan de top een Pac-Man. De *ghosts* kunnen nu aan de hand van een eenvoudig *Hill Climbing* algoritme een optimale weg volgen naar de Pac-Man.

Wanneer een *ghost* de Pac-Man opmerkt, kan deze positie op de kaart een zeer hoge waarde gegeven worden. Deze kaart wordt vaak herberekend en de geur van een Pac-Man zal dus snel verdwijnen. En dit is zoals het in de realiteit ook zal zijn. De Pac-Man beweegt zich doorheen het doolhof en zal niet op één plek blijven wachten. Als een Pac-Man meerdere keren gezien wordt, zal zijn positie ook een constantere *geur* verspreiden, waardoor de *ghosts* van alle kanten op hem kunnen

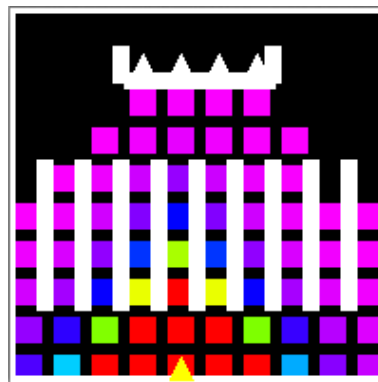
naderen.

Een handige manier om de geur van een Pac-Man te visualiseren is door middel van een kleuren pallet. We hebben hierbij gekozen voor het kleuren pallet zoals dit in de natuur voorkomt. Figuur 5.1 toont dit kleuren pallet met een aanduiding van de overeenkomstige interne numerieke waarde die aan een sector toegekend wordt om de intensiteit van de *geur* voor te stellen.



Figuur 5.1: Kleuren en voorgestelde interne *geur*-waarden

Figuur 5.2 toont een uitgangssituatie met een Pac-Man die een uitdijende *geur* verspreidt.

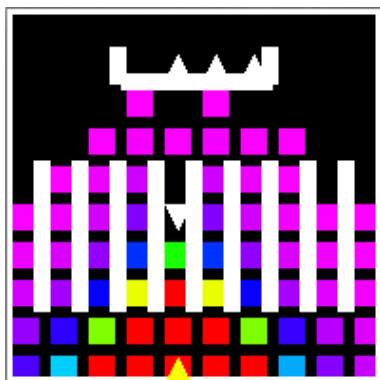


Figuur 5.2: Hill Climbing

Om te voorkomen dat *ghosts* allemaal dezelfde weg volgen en enkel achter Pac-Man lopen slurpt iedere *ghost* de geur op door de waarde van zijn huidige sector op 0 te zetten. Hierdoor ontstaat er een dal rond iedere *ghost* en zullen de anderen een omweg zoeken. Dit zorgt er ook voor dat de *ghosts* elkaar van nature uit ontwijken en niet zullen botsen. Figuur 5.3 illustreert dat, eens een *ghost* een bepaalde *beste* route naar de Pac-Man heeft ingeslagen, het pad achter hem minder interessant wordt voor de overige achtervolgers. In de voorstelling ontstaat er een *deuk* in de *geur* achter de *ghost*. Andere *ghosts* zullen daarom eerder andere paden, parallel aan het reeds ingeslagen pad, kiezen.

5.2 Verkenstrategie

De verkenstrategie is gebaseerd op hetzelfde principe. Hier is er niet één bron die een geur uitstuurt, maar iedere onbekende sector stuurt een geur uit. De



Figuur 5.3: Hill Climbing met afsluiting van pad

robot zal hierdoor altijd naar het dichtstbijzijnde en het meest onverkende stuk van het doolhof willen gaan. Daarnaast zorgt dit principe er ook voor dat een robot zal terugkeren op zijn stappen en automatisch terugkeert naar achtergelaten onbekende sectoren. We kunnen stellen dat het algoritme impliciet *Depth-First-Search* implementeert. Figuur 5.4 toont een robot die een onbekend doolhof aan het verkennen is. De felgroene sectoren zijn onbekende sectoren en hebben een hogere waarde dan de reeds bezochte sectoren.



Figuur 5.4: Hill Climbing met impliciet *Depth-First-Search* zoekgedrag

Het toepassen van hetzelfde algoritme voor beide deelproblemen, levert ons in essentie een enkelvoudige implementatie van beide. Door het toekennen van goed gekozen waarden aan onbekende sectoren en aan sectoren waar Pac-Man werd gezien, stelt het algoritme ons in staat om steeds het optimale doel te kiezen. Wanneer er zich geen pad bevindt naar de Pac-Man zal een *ghost* onbekende sectoren trachten te verkennen. Wanneer er een pad bestaat naar de Pac-Man zal zijn *geur* dit pad nog extra versterken en zal een *ghost* automatisch hierdoor aangetrokken worden.

5.3 Samenwerking en onzekerheid

De informatie die van de verschillende robots verzameld wordt is zeker niet perfect. We moeten hiermee rekening houden, zonder paranoïde te worden en alle informatie te weigeren. We hanteren in de encoding van de informatie over het doolhof een gelaagd principe: enerzijds houden we bij of we ergens een muur hebben waargenomen of niet. Maar daarnaast houden we ook bij of we iets weten over die muur of niet. Zo ontstaan er drie mogelijk statussen waarin een muur zich kan bevinden: *onbekend*, *geen muur* en *wel een muur*.

Aan de hand van deze statussen kunnen we nu overgangen gaan definiëren die we aanvaardbaar vinden. Een *onbekende* muur kan zonder problemen omgezet worden in *geen muur* of *wel een muur*. Wanneer we echter over informatie beschikken over een muur en we krijgen informatie die tegenstrijdig is, dan zullen we de status van de muur terug naar *onbekend* brengen.

De gevolgen voor het *Collaborate Diffusion* algoritme kunnen inhouden dat een sector die volledig gekend was en die dus geen *geur* meer uitstraalde, plots opnieuw een waarde krijgt die zich door het doolhof zal verspreiden. Hierdoor zal de robot geneigd zijn om deze opnieuw te gaan verkennen.

5.4 Effectiviteit

Het *Collaborate Diffusion* algoritme werkt optimaal indien alle achtervolgende robots dezelfde implementatie en parameters gebruiken. Ofschoon dit niet het geval is in onze situatie kan het toch zijn nut bewijzen voor onze robot op zich. Zo is de mogelijkheid om simultaan zowel sectoren te verkennen, als het achtervolgen van Pac-Man, een groot pluspunt voor de robuustheid van de robot.

Het *Collaborate Diffusion* algoritme is in essentie een implementatie van het *Hill Climbing II* algoritme. Dit zorgt ervoor dat de robot ten alle tijde volgens het meest optimale pad naar de Pac-Man tracht te rijden. Mits kennis van dit algoritme is het voor de bestuurder van de Pac-Man zeker mogelijk om hiermee rekening te houden en er voordeel uit te halen. Het algoritme zal bijvoorbeeld nooit een minder optimale weg kiezen om de Pac-Man via een andere weg klem te rijden.

Dit concept is wel inherent ingebouwd indien een andere robot het meest optimale pad kiest en zo de verspreiding van de *geur* tegen gaat. Op dit moment lijkt het alsof de robot een minder optimaal pad zal volgen, doch het is een pad waarlangs de *geur* hem wel bereikt.

Met de mini-simulator hebben we verschillende scenario's uitgetest. Hierbij werd ook gebruik gemaakt van een bewegende Pac-Man. Buiten enkele exotische opstellingen, werd de Pac-Man toch altijd in een redelijke tijdspanne gevangen.

Hoofdstuk 6

Softwaredesign

In het eerste semester hadden we reeds een redelijk uitgebreid raamwerk opgebouwd om robots samen te stellen uit componenten. Dankzij dit werk kunnen we in het tweede semester hier op verder bouwen door de bestaande concepten verder te verfijnen en uit te breiden.

Echter, vooraleer de definitieve beslissing genomen werd om verder te bouwen op de bestaande architectuur, werden enkele andere opties bekeken. Enerzijds werd de bestaande *Pilot* klasse van de Lejos software bekeken en anderzijds werd in literatuur gezocht naar andere mogelijke opstellingen.

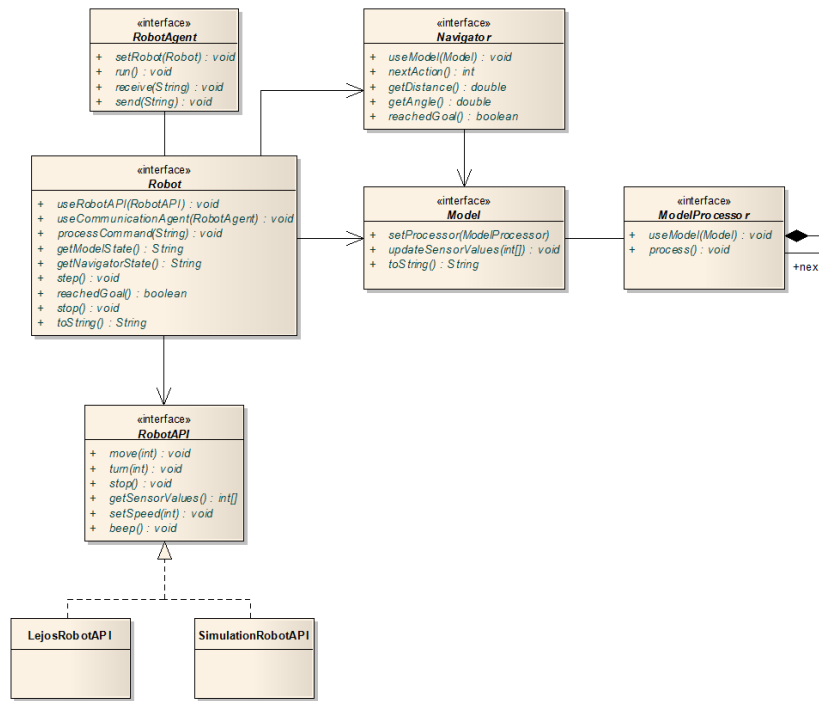
De bestaande *Pilot* klasse van Lejos volgt geen radicaal verschillende aanpak. Wel hebben we het *behaviour* idee geïncorporeerd in ons eigen ontwerp. Bij onze zoektocht in de literatuur moesten we tot het besluit komen dat er waarschijnlijk evenveel mogelijke ontwerpen zijn als er robots zijn. Veel van de bekeken oplossingen begeven zich deels op het vlak van Lejos of belichten een veel hoger niveau. Een concrete, de facto, alternatieve architectuur voor het aansturen van een robot werd niet gevonden. Wel hebben we veel ideeën opgedaan die verwerkt werden tijdens de verschillende bijwerkingen van de software.

De architectuur van het eerste semester heeft zich bewezen. De structuur en concepten zijn gekend, alsook de problemen en tekortkomingen. Alles in beschouwing genomen werd unaniem besloten om verder te werken op het bestaande design en elementen uit de studie van andere oplossingen op te nemen waar nuttig.

6.1 Robot

Het klasse diagram op figuur 6.1 toont de belangrijkste bouwstenen van ons robot-raamwerk, zoals dit tijdens het eerste semester werd ontworpen: een *Robot*-entiteit beschikt over een *RobotAPI* om met de fysieke robot informatie uit te wisselen. Deze informatie heeft betrekking op de gemeten waarden van de verschillende

sensoren als ook het instellen van de motoren. De *RobotAPI* stelt ons in staat om door het vervangen van één enkele klasse, identiek dezelfde *Robot*-entiteit te gebruiken in een echte fysieke robot, alsook in een gesimuleerde wereld.



Figuur 6.1: Design 1^e semester

De *Robot* beschikt verder over een *Model* waarin alle informatie uit de wereld wordt samengebracht. Dit *Model* kan verrijkt of bijgewerkt worden door zgn. *ModelProcessors*. Deze onderzoeken bepaalde delen van het *Model* en trachten hieruit informatie van een hoger niveau te distilleren. Voorbeelden hiervan zijn processors die op basis van metingen door de sonar-sensor bepalen waar er zich muren bevinden.

Tot slot heeft de *Robot* ook een *Navigator* die op basis van de informatie in het *Model* bepaalt waarheen de *Robot* dient te rijden. De navigator geeft zeer basische instructies aan de *Robot*, in de vorm van *beweeg voor- of achterwaarts* en *draai zoveel graden*.

Om met de *Robot* te kunnen communiceren is een *RobotAgent* voorzien, waarlangs berichten kunnen verzonden en ontvangen worden.

Het is duidelijk dat dit raamwerk een sterk herbruikbare basis biedt om verschillende robots te bouwen. Dit heeft al tijdens het eerste semester zijn nut bewezen, maar komt nog sterker naar voor bij aanvang van het tweede semester.

Door het maken van een nieuwe specifieke *Navigator* en bijhorend *Model* met *Modelprocessors*, kunnen we opnieuw een robot samenstellen, waarbij we maximaal gebruik maken van de bestaande software.

6.1.1 Driver

Ofschoon we een nieuwe *Navigator* zouden kunnen maken die perfect zou passen, werd snel duidelijk dat er een verfijning van het raamwerk mogelijk was: daar waar tijdens het eerste semester de robot geen resolutie had bij het rijden, is dit in het tweede semester wel het geval. Door het introduceren van het concept van sectoren is er een duidelijk onderscheid tussen het operationele en strategische aspect van het besturen van de robot. Operationeel dient de robot optimaal van sector naar sector te rijden, zodat op strategisch niveau kan geredeneerd worden in termen van deze sectoren.

De functionaliteit om van sector naar sector te rijden zou typisch terecht gekomen zijn in een *Navigator*-implementatie. Deze wordt in het verfijnde raamwerk ondergebracht in een zgn. *Driver*. Het klasse diagram op figuur 6.2 toont dat de introductie van de *Driver* letterlijk een uitbreiding is van het bestaande raamwerk. Voor het tweede semester is er tevens een eerste implementatie van de *Driver* gemaakt in de vorm van een *ManhattanDriver*.¹

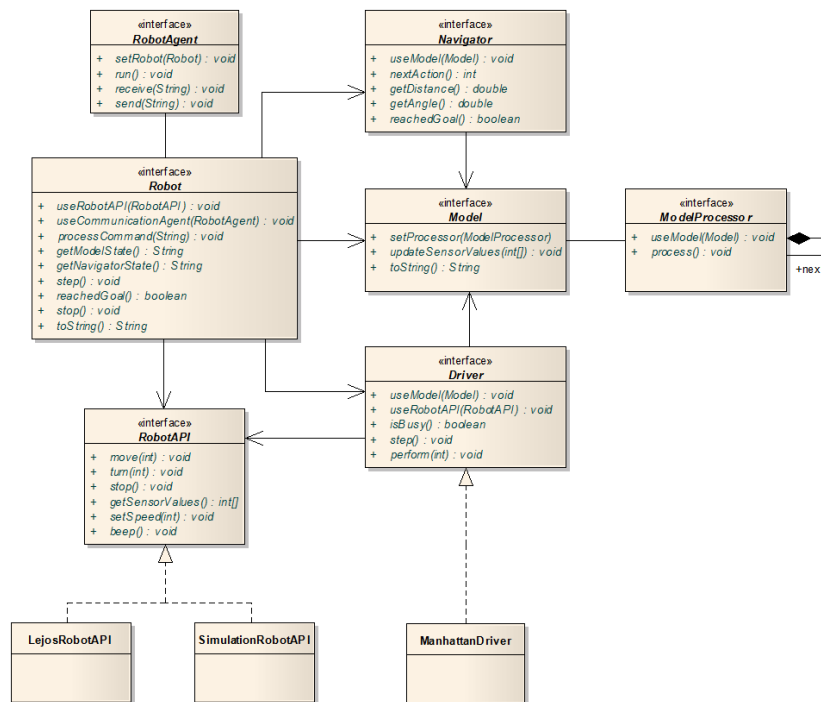
6.1.2 Grids en Sectoren

De *Navigator* kan zich nu specifiek toeleggen op het bepalen van de volgende *Sector*, waarheen de *Driver* moet rijden. Om deze taak te ondersteunen werden de concepten *Grid*, *Sector* en *Agent* in het leven geroepen.

Centraal staat het concept van de *Sector* in een *Grid*. Een *Sector* heeft een coördinaat relatief t.o.v. de *Grid* waarop het zich bevindt, heeft potentieel 4 muren, elk al dan niet gekend, heeft een waarde en kan bezet worden door een *Agent*. De waarde van een *Sector* zal gebruikt worden om de hoogte of de intensiteit van de geur weer te geven. Een *Agent* kan zich doorheen de *Grid* bewegen langs de sectoren. De meeste functionaliteit van een *Agent* is geïmplementeerd in een abstract basis klasse, *MovingAgent*. Mits kleine configuratieaanpassingen, implementeren de *PacmanAgent* en *GhostAgent* zo de agents die op de *Grid* voorkomen. Een *ProxyAgent* werd gebruikt om in een simulator een echte *Agent* voor te stellen die de volledige *Grid* nog niet kent en de *StaticTargetAgent* werd speciaal voorzien voor de voorbereiding van de eerste demo.

Optioneel kan aan een *Grid* ook een *GridView* gegeven worden. Deze voorziet een visualisatie van de *Grid* en kan op verschillende manieren ingevuld worden.

¹Zo genoemd naar analogie met Manhattan Distance en de Taxicab geometrie.



Figuur 6.2: Design 2^e semester

Voor dit project hebben we twee implementaties gemaakt: enerzijds een *Console-GridView* die gedetailleerde informatie over de sectoren weergeeft en een *Swing-GridView* die een grafische voorstelling geeft van de *Grid*. Deze laatste wordt gebruikt voor de visualisatie in o.a. de simulator.

De nieuwe robot zal ook informatie van andere robots ontvangen. Deze worden allemaal opgeslagen in een zelfde *Grid*. Een tweede implementatie van het *Grid* concept, de zgn. *AggregatedGrid*, is in staat om een aantal verschillende grids samen in beschouwing te nemen en de *som* van deze als een geaggregeerd beeld aan te bieden.

Demo 3

In het licht van de derde demo is het grid systeem van de robot opnieuw bekeken. Het ontwerp is aangepast om 2 belangrijke problemen met de vorige implementatie op te lossen:

Geheugengebruik, met name werkgeheugengebruik (oude implementaties maakten al vlug enkele duizenden Point objecten. Deze blijven echter maar 1 robot-frame bestaan, maar geven toch een enorme tijdelijke burst in werkgeheugen, wat evengoed problemen veroorzaakt)

Invoeren van extra abstractielagen: de vorige implementatie bevatte verschillende logische elementen door elkaar, met het zicht op unit testen en stabiliteit was het beter om de functionaliteiten van de grid in onafhankelijke lagen te verdelen.

Geheugengebruik

Met oog op compatibiliteit met de oude code werd de structuur Grid-Sector behouden. Op deze manier kan in de robot logica gewerkt worden met hoog-niveau sectoren en blijft de interne werking van Grid linking, merging en mapping verborgen voor de rest van de software.

In tegenstelling tot de vorige versie, is de logica in de sector klassen verplaatst naar de grid klassen. De sectoren treden op als een *facade*² voor het sector gedeelte van de functies op de Grid interface. Deze sector functions op Grid gebruiken een sector ID om de sectoren te identificeren. Dit heeft als enorme voordeel dat niet alle sector objecten in het geheugen geladen moeten worden. Het aantal objecten in het werkgeheugen neemt op deze manier sterk af, want sectoren kunnen nu met een integer geïdentificeerd worden in plaats van met een Sector object of een Point object. Het resultaat is dat de Grid klassen voor het merendeel van hun bewerkingen geen nieuwe objecten aanmaken eenmaal het programma in een *steady-state* is.

De grids onderling gebruiken geen sector of point objecten, maar maken rechtstreeks gebruik van de Grid interface. Dit zorgt ervoor dat transformatie en aggregatie van grids gebeurt zonder aanmaak van tijdelijke objecten.

Functionaliteiten

LinkedGrid Deze grid implementeert de datastore logica van de grid. De linked grid structuur wordt door deze klasse intact gehouden, alsook de datastorage van de agents. Intern gebruikt deze klasse de *LinkedSector* klassen om informatie in verband met muren op te slaan.

TransformedGrid Deze grid vormt een decorator³ voor een andere Grid, en laat toe op alle functies die uitgevoerd worden op grid een transformatie toe te passen. Dezelfde transformatie is ook van toepassing op de sectoren die opgehaald worden uit deze grid.

ObservableGrid Deze grid is weer een decorator voor een andere *Grid*. Alle changes tot deze decorated grid worden bekeken, en kunnen via een *GridObserver* gedetecteerd worden. Dit laat het uitvoeren van logica bij grid changes toe, zonder dat deze logica gemengd wordt met de overige grid code.

²http://en.wikipedia.org/wiki/Facade_pattern

³http://en.wikipedia.org/wiki/Decorator_pattern

AggregatedGrid Deze grid verzameld een aantal verschillende grids en staat toe deze te activeren gegeven een transformatie. De sectoren, agents en muren van sectoren worden gecombineerd door de aggregated Grid. *AggregatedGrid* implementeert de gewone grid interface. Deze kan dus gebruikt worden zoals elke andere Grid. De *AggregatedGrid* is als een aparte grid-laag ingevoerd zodat het mergen van grids dynamisch kan gebeuren. Op deze manier moet er geen extra gemergde informatie opgeslagen worden in het geheugen van de robot. Dit om alsnog het geheugenverbruik van de robot in te perken.

MultiGhostGrid Intern houdt deze grid laag een aantal sub-grids bij, een voor elke ghost. Met behulp van *ObservableGrids* detecteert de *MultiGhostGrid* veranderingen aan de onderliggende sub-grids. Vervolgens probeert hij aan de hand van barcode informatie de ghost grids te mappen. Indien een correcte mapping gevonden wordt, gaat de *MultiGhostGrid* de betreffende ghost grids mergen door deze correct te activeren op een *AggregatedGrid*.

LinkedSector Deze wordt intern gebruikt door de LinkedGrid om de wall data op te slaan.

FacadeSector De FacadeSector bevat simpelweg een verwijzing naar de grid waartoe hij behoort, plus een sector Id. Alle aanroepen op deze klassen worden doorverwezen naar de Grid.

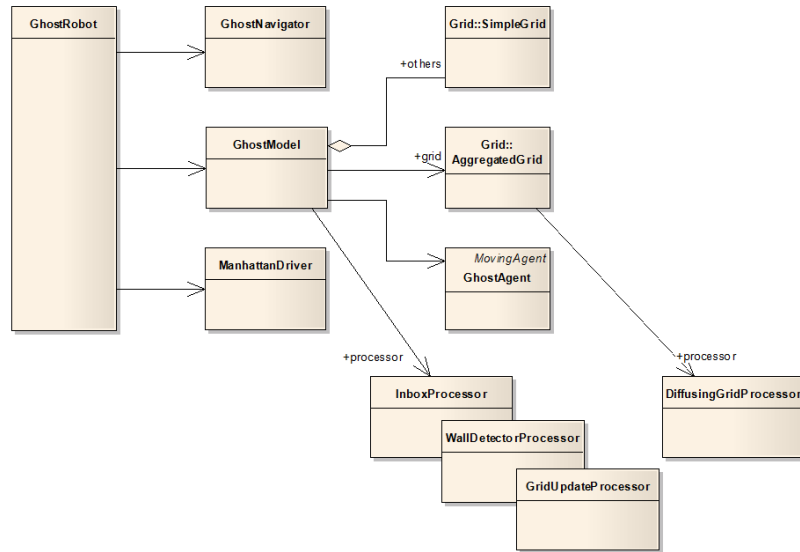
6.1.3 GhostRobot

Het klasse diagram op figuur 6.3 geeft tot slot de volledige compositie van de nieuwe *GhostRobot* weer. De *GhostRobot* beschikt over een *GhostModel*, *GhostNavigator* en *GhostDriver*.

Het *GhostModel* bevat *Grids* voor elk van de andere robots alsook een *AggregatedGrid* voor de eigen *Agent*. Zolang er geen gemeenschappelijk referentiepunt⁴ is gevonden zijn de verschillende *Grids* niet met elkaar in verband te brengen. Wanneer er echter een gemeenschappelijk referentiepunt gevonden is, kan de corresponderende *Grid* van de andere robot toegevoegd worden aan de eigen *AggregatedGrid*, waardoor de informatie van beide *Grids* gecombineerd wordt en een ruimer wereldbeeld beschikbaar wordt.

De eigen *AggregatedGrid* beschikt ook over een *DiffusingGridProcessor* die het algoritme voor het bepalen van de waarde van een *Sector* zal toepassen op de *AggregatedGrid*, waardoor de *GhostNavigator* een redelijk eenvoudige beslissing kan nemen op basis van de waarden van de vier aangrenzende sectoren.

⁴Referentiepunten worden op het parcours voorgesteld door georiënteerde barcodes.



Figuur 6.3: Compositie GhostRobot

Het *GhostModel* is verder voorzien van een aantal *ModelProcessoren* die de inkomende informatie verwerken tot informatie waarmee de *GhostNavigator* beslissingen kan nemen. Enerzijds is er een *InboxProcessor* die de binnenkomende berichten van de andere robots omzet in aanpassingen aan de verschillende *Grids* in het *GhostModel*. Anderzijds is er de *WallDetectorProcessor* die op basis van de gemeten waarden door de sonar-sensor een beeld zal samenstellen van de huidige *Sector*. Tot slot is er de *GridUpdateProcessor* die de nieuwe muurinformatie zal verwerken in de eigen *Grid* en eventueel nieuwe sectoren zal toevoegen.⁵

6.1.4 Performantie

Met de expliciete keuze om alle software op de robot zelf te implementeren, zijn we natuurlijk gebonden aan de beperkingen van de robot. Enerzijds heeft deze geen processor die te vergelijken is met de processors van hedendaagse computers. Dit is echter voor onze architectuur en gekozen zoekstrategie geen probleem. Door gebruik te maken van een zeer eenvoudig *local Hill Climbing* algoritme, gebaseerd op het principe van *Collaborate Diffusion*, vragen we zeer weinig van de processor.

⁵Naast deze drie *ModelProcessoren* zijn er nog andere actief. Het betreft hier bvb. de *LineModelProcessor* of de *WallDetectionModelProcessor*. Deze zorgen voor informatie die door de *Driver* zal gebruikt worden om optimaal naar de volgende sector te rijden. Veel van deze *ModelProcessoren* maakten reeds deel uit van de robots uit het eerste semester.

We slagen er in om een *frame rate* te bekomen die rond de 80 FPS ⁶ ligt, wat betekent dat de frequentie waarmee de robot al zijn taken uitvoert meer dan hoog genoeg is voor de taken die moeten uitgevoerd worden. Proefondervindelijk hebben we vastgesteld dat een frame rate van 60 FPS een minimum is om de verschillende taken naar behoren uit te voeren.

Geheugengebruik

Het aspect geheugen is dan weer wel een sterk beperkende factor. De robot beschikt slechts over een werkgeheugen van 64KB waarvan een deel reeds wordt ingenomen door het geladen programma. Tijdens het eerste semester was de hoeveelheid informatie die opgeslagen werd in het *Model* nagenoeg constant, of had ten minste een gekende bovengrens. In het tweede semester moeten we echter informatie bijhouden over het speelveld waarop we ons bewegen. Aangezien we op voorhand niet weten hoe groot dit speelveld is kunnen we niet op voorhand een bovengrens bepalen voor het geheugen nodig om deze informatie op te slaan. Daarbij komt dat we dit speelveld niet alleen moeten bijhouden voor onze eigen robot, maar ook voor de drie andere robots.

Intern stellen we het speelveld of *Map* voor als een verzameling sectoren, die verbonden zijn met elkaar door bi-directionele verwijzingen. Telkens een robot een nieuwe sector *ontdekt* wordt er intern ook een nieuw *Sector* object aangemaakt. De grootte van zo'n *Sector* object is dus bepalend voor de hoeveelheid sectoren we maximaal kunnen bijhouden in het geheugen van de robot.

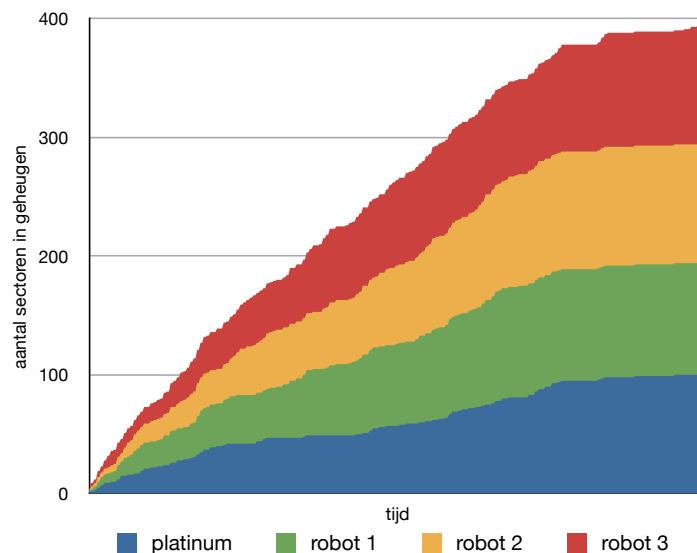
Vlak voor demo 1, was een *Sector* ongeveer 370 bytes groot en konden we ongeveer 145 sectoren opslaan in ons geheugen. In een worst-case scenario betekent dit dat we per robot 36 sectoren konden opslaan, wat overeenkomt met een speelveld van 6 bij 6. We hebben vervolgens twee pistes gestart om dit probleem aan te pakken. Enerzijds hebben we het geheugengebruik van de *Sector* klasse aangepakt en gereduceerd tot 280 bytes. Anderzijds hebben we besloten om binnenkomende informatie van andere robots slechts te bewaren totdat deze informatie op basis van een barcode kon geïmporteerd worden in de *Map* van de eigen robot. Op dat ogenblik gooien we de *Map* van de andere robot weg en voegen de nog volgende binnenkomende informatie rechtstreeks toe aan onze eigen *Map*.

Op deze manier zal er een trapsgewijs verval zijn van het totale aantal sectoren die moeten bijgehouden worden. Immers wanneer alle robots een gemeenschappelijke barcode hebben gevonden, zal er slechts één *Map* overblijven, nl. die van de eigen robot.

⁶Frames Per Second is een term die uit de grafische sector komt en weergeeft hoeveel beelden per seconde gemaakt worden, maar wordt ook gebruikt in andere contexten. Een *frame* is dan het geheel van taken. Voor onze robot bestaat één frame uit het opvragen van de sensorwaarden, het bewerken van het *Model*, het kiezen van een volgende actie en het initiëren van de uitvoering.

Met deze aanpassingen konden we voor demo 1 een maximaal aantal van 200 sectoren opslaan, ofwel 50 per robot, wat overeenkomt met een speelveld van 7 bij 7. Rekening houdend dat deze situatie zich niet kan voordoen, omdat de robots nooit allemaal afzonderlijk het hele speelveld moeten ontdekken en sowieso gemeenschappelijke barcodes zullen tegenkomen, ligt de bovengrens voor de dimensies van het speelveld in realiteit hoger. Simulaties wezen uit dat we met 200 sectoren in staat waren om een (realistisch) speelveld van 10 bij 10 zonder problemen te kunnen verkennen met vier robots.

We hebben deze simulaties gedaan aan de hand van de mini-simulator (zie sectie 7.1.4). Wanneer 4 robots een speelveld van 10 bij 10 verkennen, zonder gebruik te maken van barcodes om informatie uit te wisselen, dan zal het totaal aantal sectoren in het geheugen van onze robot oplopen tot 400. Figuur 6.4 toont een grafiek die het verloop van zo'n sessie weergeeft.

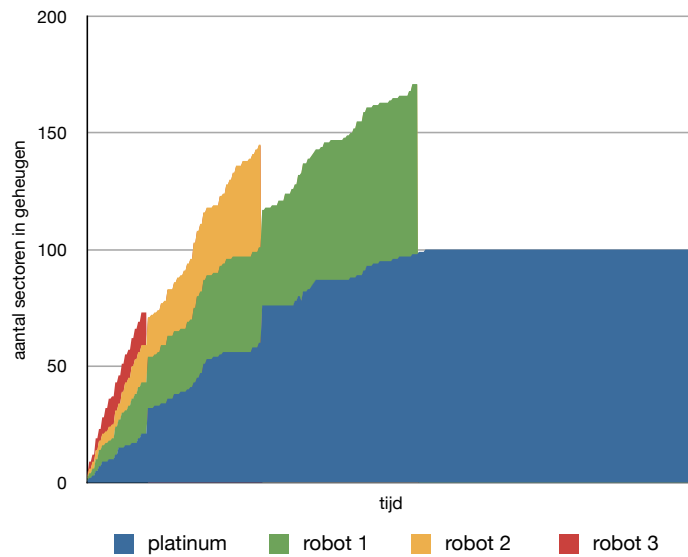


Figuur 6.4: Verloop van het aantal sectoren zonder uitwisseling van informatie.

Figuur 6.5 toont vervolgens het resultaat van dezelfde simulatie, echter nu zal onze robot de door andere robots verkende sectoren importeren in zijn eigen map, de *Map* van de andere robot uit zijn geheugen verwijderen en alle volgende nieuwe informatie rechtstreeks toepassen op zijn eigen *Map*.

We zien hier duidelijk twee zaken naar voor komen: enerzijds beschikt de robot veel sneller over informatie van het hele speelveld en anderzijds wordt het totale geheugengebruik ook drastisch beperkt.

Deze aanpak heeft natuurlijk een groot nadeel. Eens de gegevens van een andere robot geïmporteerd zijn, is het niet mogelijk om deze terug te verwijderen.



Figuur 6.5: Verloop van het aantal sectoren met uitwisseling van informatie.

De architectuur voorziet een *AggregatedGrid* die de gegevens van de echte *Grids* dynamisch aggregaat. Hierbij moeten de *Grids* van de verschillende robots echter wel in het geheugen bewaard blijven.

Met het oog op demo 2 en 3 willen we nog verdere optimalisaties doorvoeren op het vlak van het geheugengebruik van een *Sector*. Anderzijds onderzoeken we een oplossing waarbij er sectoren uit de verschillende mappen van andere robots verwijderd worden in functie van het nog beschikbare geheugen.

Opslagcapaciteit voor het programma

Volgens de specificatie van de Lego NXT Brick komt de maximale opslag neer op 256KB maar de robot blijkt te werken met een 16bit offset bij zijn interne tabellen. Bijgevolg is de theoretische limiet van 64KB, in de praktijk werd deze limiet rond de 60KB opgemerkt.

Reductie werd bekomen door het snijden in de code, zo werd bv. eerst en vooral de MD5 hash verplaatst naar de *Gateway*. Verder werd het *Lejos* gedeelte dat bestond uit *RotationMovement*, *RotatingSonar*, *Main* en *AngieEventLoop* samengebracht in *AngieRobotApi*.

Er werden zoveel mogelijk Strings verwijderd en anders werden deze zo klein mogelijk gemaakt. *toString* methodes werden leeg gemaakt zodat deze geen kostbare geheugenruimte verbruikten. Verdere aanpassingen van grote impact werden reeds vermeld bij de bespreking van *Grid*. Uiteindelijk werd zelfs overgegaan tot

bytecode *shrinking* met behulp van ProGuard ⁷.

Deze tool verkleinde de gebruikte bytecode nog verder aan de hand van volgende optimalisaties⁸:

- Verwijderen van ongebruikte klassen, velden en methoden die de linker mist.
- Evalueren van constante expressies.
- Verwijderen van onnodige methodeoproepen, vergelijkingen, instanceof testen, ongebruikte code blokken, branches, veldtoegangen,
- Samenvoegen van identieke blokken code
- Opkuisen van ongebruikte methode parameters en verplaatsen van dubbele methodes
- Inlinen van constanten, methode parameters, returnwaarden en weinig gebruikte methodes
- Tail recursie versimpelen
- Samenvoegen klassen en interfaces (bvb indien maar 1 keer geïmplementeerd.)
- Methodes en klassen private, static en final maken
- Mini-optimalisaties zoals $. * 2 = . < < 1$.
- Verwijderen van Logging code
- Loop optimalisaties.

6.2 PC

Op het vlak van de PC kunnen we opnieuw verder bouwen op onze bestaande *ServiceAgent*. Naast het logging-kanaal, zal deze nu ook een tweede Bluetooth kanaal volgen, langs het welke de boodschappen van het *Ghost Protocol* kunnen uitgewisseld worden met de *exchange-queue* op de RabbitMQ server.

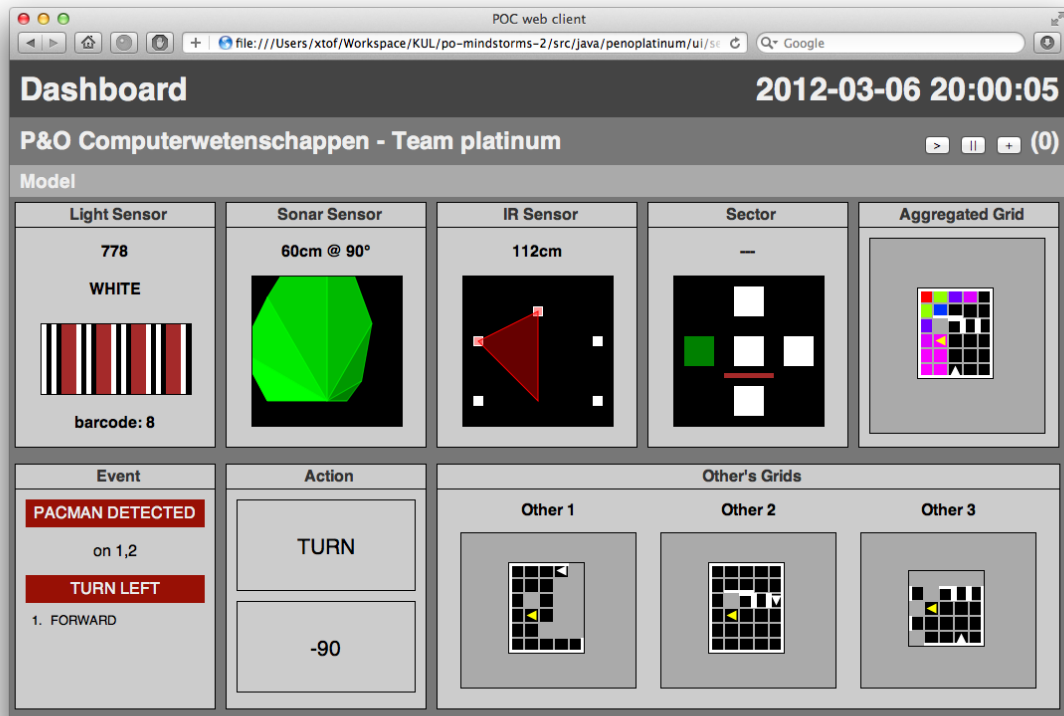
Ook het *Dashboard* wordt uitgebreid met meer informatie over de werking van het zoek-algoritme en de map-verkenning. Hiervoor moet louter bijkomende informatie vanuit het *Model* en de *Navigator* doorgestuurd worden, kunnen we *Log4J*

⁷<http://proguard.sourceforge.net>

⁸We vonden deze tool via de volgende thesis <http://www.vmars.tuwien.ac.at/documents/extern/2601/joptimizer-thesis.pdf>

eenvoudig aanpassen in configuratie en voorzien we de databank van bijkomende kolommen om de nieuwe gegevens op te slaan.

Het Dashboard zelf wordt voorzien van de nodige visualisaties voor deze nieuwe informatie. Figuur 6.6 toont de nieuwe samenstelling.



Figuur 6.6: Nieuw Dashboard

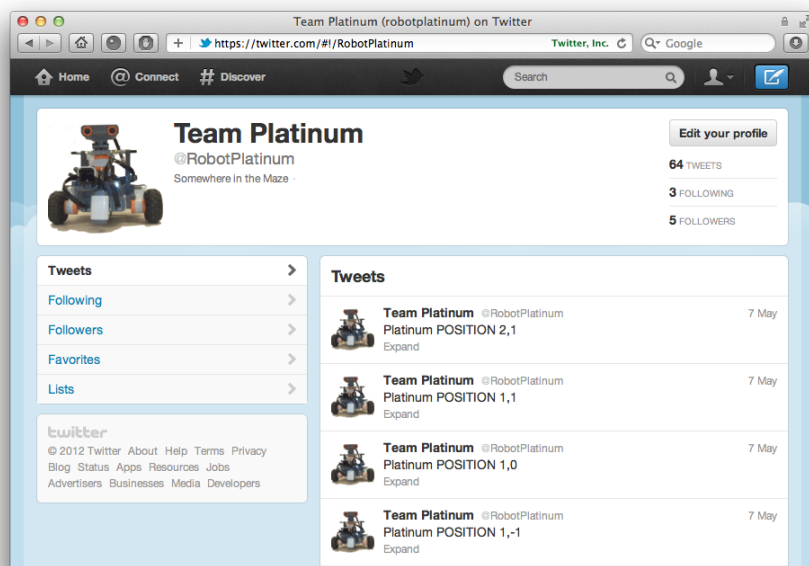
6.2.1 Externe componenten

We trachten bij de ontwikkeling van de software zo veel mogelijk bestaande componenten te gebruiken. Zo worden alle op de PC uitgevoerde applicaties voorzien van opties die kunnen meegegeven worden op de command line. Hierdoor worden de applicaties flexibel en kan veel van de configuratie langs deze weg dynamisch voorzien worden.

Anderzijds trachten we ook het wiel niet opnieuw uit te vinden. Het inzetten van *Log4J* begon als een klassieke keuze voor een industriële standaard, maar groeide gaandeweg uit tot een centrale component. Via de configuratie van *Log4J* zijn we niet alleen in staat om de informatie die van de robot naar de *Gateway*

wordt gezonden, op te slaan in een simpele log file of in de databank. Ook de detail uitsplitsing van deze informatie naar verschillende tabellen kan ondertussen op dit niveau gebeuren. Ook werd een eigen *LogAppender* gemaakt die de informatie in een bestand opslaat. Dit bestand kan vervolgens gebruikt worden door een *Dashboard* zonder dat een connectie naar een server nodig is.

Zonder enige structurele uitbreiding van de code zijn we zelfs in staat om deze informatie te posten naar bijvoorbeeld Twitter, waardoor onze robot zonder twijfel de eerste *twitterende* P&O robot is geworden. Mits een eenvoudige Log4J Appender (op basis van Twitter4J⁹) moeten we louter een sectie toevoegen aan de configuratie en onze robot kan gevolgd worden op Twitter: <https://twitter.com/#!/RobotPlatinum>. Dus *follow @RobotPlatinum*.



Figuur 6.7: RobotPlatinum *tweets* zijn positie door tijdens de finale demo.

Een laatste belangrijke externe component is zonder twijfel *Mockito*¹⁰. Dit *mocking framework* is de hoeksteen van een goed uitgevoerde suite aan unit testen. Van oorsprong was *Mockito* echter een *spying framework*, maar omdat de lijn tussen de twee verantwoordelijkheden zo dun begon te worden en meer en meer mensen gebruik wensten te maken van *Mockito* is de nodige functionaliteit toegevoegd en is het op dit ogenblik één van de beste *mocking frameworks* dat beschikbaar is. De belangrijkste eigenschap van *Mockito* is de laagdrempeligheid. Dit was binnen het

⁹<http://twitter4j.org>

¹⁰<http://code.google.com/p/mockito/>

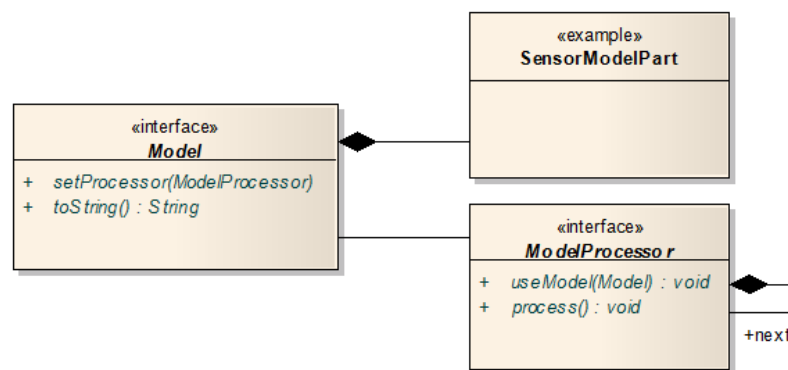
team een belangrijke factor, omdat er weinig tot geen ervaring was hieromtrent bij de meeste van de teamleden. Dankzij deze laagdrempeligheid en de snelle *return on investment* werden alle teamleden snel grote fans van dit framework.

6.3 Refactoring

Een groot deel van de tijd tussen demo 1 en 2 werd besteed aan het refactoren van de code. Aan de hand van een grondige analyse zijn verschillende historisch gegroeide pijnpunten aangeduid. Tot op dit moment was er veelal rond deze problemen gewerkt onder druk van de verschillende demo's. Met een functioneel volledig uitgewerkte oplossing, kunnen we nu zorgen dat de kwaliteit van de hele repository in lijn was met het niveau van de functionaliteit.

De belangrijkste streefdoelen waren het wegwerken van redundantie in de code, het correct benoemen van de verschillende klassen en namespaces, het toevoegen van duidelijke startpunten en het toevoegen van vele kleine extra's die het geheel tot een flexibel en gepolijst resultaat omtoveren.

Een van de grootste acties bestond er in om het sterk uitgegroeide Model op te splitsen in functionele deelmodellen en het verplaatsen van functionaliteit uit deze gegevensdragers naar de ModelProcessors. Figuur 6.8 illustreert deze aanpassing.



Figuur 6.8: Refactoring van Model

Samen met deze refactoring werd ook de redundante code die ontstond bij de introductie van de mini-simulator maximaal weggewerkt. Beide simulatoren gebruiken nu zo veel mogelijk dezelfde code.

Daarnaast wordt er ook bij elke verbetering rekening gehouden met het geheugengebruik. Waar mogelijk worden zo klein mogelijke datastructuren gebruikt om het onnodig reserveren van geheugenruimte te beperken. Dit geldt natuurlijk alleen voor klassen die effectief op de robot gebruikt worden.

6.3.1 Van pure refactoring naar test coverage

Tijdens het refactoren tussen demo 1 en 2 kwamen veel verdoken problemen naar boven en het refactoren zelf verliep zeer moeizaam. De code voelde aan als een groot *Mikadospel*. Wanneer we aan één kant iets aanpasten, viel aan de andere kant een groot deel van het project in duigen. Het was duidelijk dat we een structurele ingreep moesten doen om te voorkomen dat dit probleem verder zou escaleren.

We besloten om het refactoren verder door te drijven en de code sequentieel te doorlopen, alles in orde te brengen en vooral volledig te voorzien van unit testen. Met een Paasvakantie voor de boeg leek ons dit een opportuun moment om de kwaliteit van de code structureel aan te pakken. In sectie 9.4 bespreken we het unit testen in groter detail in het kader van de procesbeschrijving.

Hoofdstuk 7

Simulator

De simulator is zoals reeds vermeld een verplichte component van het tweede semester en bedoeld als belangrijk hulpmiddel voor de ontwikkeling van de ghosts. Aangezien we tijdens het vorige semester reeds een werkende simulator ontwikkeld hadden, zal deze beschrijving zich vooral focussen op de delen die toegevoegd en/of veranderd werden in functie van de nieuwe opdracht.

Als eerste werd de bestaande code verder afgewerkt, waarna we de nodige uitbreidingen hebben toegevoegd ter ondersteuning van de nieuwe functionaliteit. Hierbij werd er voor gezorgd dat alle bestaande code kon behouden blijven.

7.1 Demo 1

7.1.1 Aanpassingen

Aangezien de kleinste eenheid van een parcours veranderd is van een paneel naar een sector, hebben we met behulp van refactoring de simulator aangepast zodat een parcours zowel met panelen als met sectoren kan worden voorgesteld. Hierdoor is het mogelijk zowel het project van het eerste semester als het tweede semester uit te voeren. Deze techniek werd verder toegepast op de rest van het herstructureringsproces.

Naast het aanpassen van de schaal veranderden we ook de wijze waarop de verschillende sensoren voorgesteld werden. Op het einde van de vorige iteratie waren sensoren eigenlijk functies die elke stap werden opgeroepen om de sensorwaarden in het model te updaten. Deze voorstelling is niet ideaal om uit te breiden. Er werd gekozen voor een Sensor interface en voor elke sensor een klasse die deze interface implementeert. Naast een makkelijkere uitbreidbaarheid, heeft dit design ook het voordeel dat men voor elke entiteit zijn eigen sensoren kan aanmaken en dus een custom mapping kan hebben.

7.1.2 Nieuwe Functionaliteit

Voor het simuleren van meerdere robots, zowel lokaal als op afstand, moest de robot afgescheiden worden van de simulator. Tegelijk werd ook de GUI-component van de robot afgescheiden en geabstraheerd zodat er meerdere types robots kunnen bestaan. Door deze abstractie is het mogelijk om verschillende simulatoren te laten samenwerken elk met hun eigen robot. Deze simulatoren communiceren met een speciaal protocol op een eigen RabbitMQ kanaal.

Voor het testen van de verschillende algoritmes, specifiek het heroriëntatie algoritme, is er al de mogelijkheid om op sommige plaatsen meet- en stuurafwijkingen in te voegen in de simulator. De afwijkingen op de gesimuleerde motoren kan men in de API van de robot aanpassen. Er kan zowel een variatie als een gemiddelde fout ingesteld worden, onafhankelijk van het vooruitrijden en het draaien.

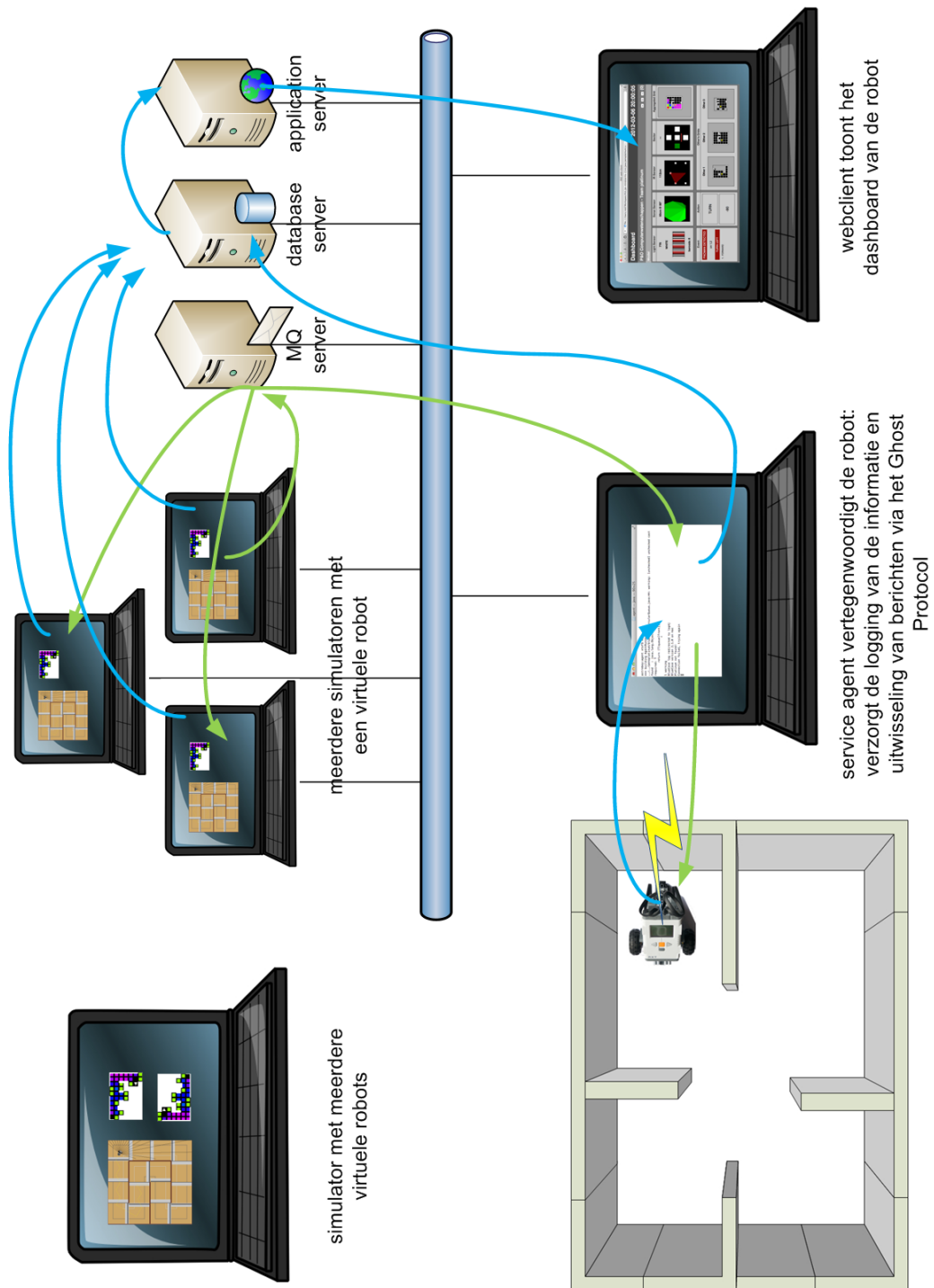
7.1.3 Simulatormodi

De simulator van het eerste semester was een hulpmiddel om met meerdere teamleden in parallel te kunnen werken zonder te moeten strijden om tijd met de echte fysieke robot. In het tweede semester is de simulator een wezenlijk deel geworden van de opdracht en ook de manier waarop de simulator moet kunnen werken is sterk verschillend van onze eigen doelstellingen in het eerste semester.

In het tweede semester moet de simulator op niet minder dan 3 verschillende manieren functioneren:

- Meerdere (virtuele) robots moeten tegelijkertijd binnen dezelfde simulator aangestuurd kunnen worden.
- Meerdere instanties van de simulator moeten robots kunnen laten communiceren met elkaar via het *Ghost Protocol* langs de MQ server.
- Naast onderlinge communicatie tussen de robots in de simulatoren moeten ook fysieke robots kunnen deelnemen, in een zgn. hybride modus.

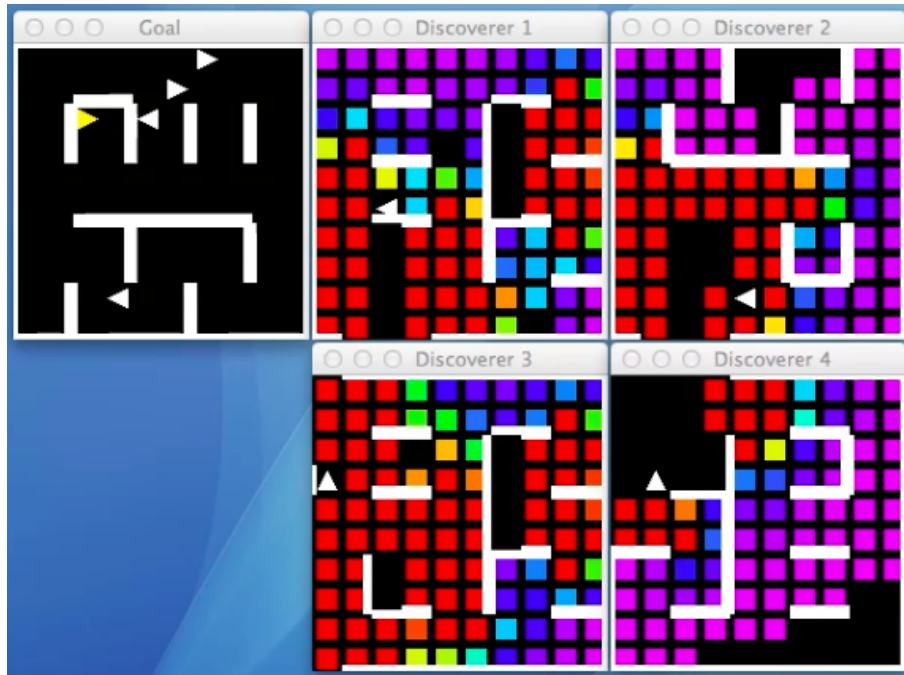
Figuur 7.1 geeft een overzicht van de verschillende mogelijkheden en hoe deze uitgewerkt zullen worden binnen onze oplossing. De groene pijlen tonen hoe berichten tussen de verschillende robots worden uitgewisseld via het *Ghost Protocol* en de MQ-server. De blauwe pijlen geven aan hoe de informatie over de werking van de robot via de *ServiceAgent* en het *Log4J* logging framework opgeslagen worden in de databank, waarna ze consulteerbaar zijn via de webclient.



Figuur 7.1: Werking van de simulator in verschillende opstellingen.

7.1.4 Mini-Simulator

Om het onderzoek naar het algoritme onafhankelijk te laten verlopen van de echte simulator en tevens niet te belasten met details van deze laatste, werd een mini-simulator gemaakt. Deze maakt een abstractie van de wereld en kent alleen een resolutie op niveau van sectoren zoals dit het geval is in de simulator. Figuur 7.2 toont de mini-simulator in actie met vier gesimuleerde robots, elk vertrokken op een andere positie en met een andere oriëntatie.



Figuur 7.2: Mini-simulator in actie

Door de introductie van de *Driver* naast de *Navigator* is er een duidelijke opdeling tussen het effectieve rijden van sector naar sector en de keuze van de volgende sector waarheen gereden moet worden. Deze abstractie is doorgetrokken in de mini-simulator die zich louter met het *Navigator* niveau bezighoudt, waar de (volledige) simulator ook het niveau van de *Driver* in beschouwing neemt.

Zo wordt de mini-simulator ook gebruikt om onderzoek te doen naar het geheugengebruik, nodig om al de navigator-informatie bij te houden. Net zoals bij de (volledige) simulator, is het ook mogelijk om zonder visualisatie te werken, waardoor veel scenario's kunnen getest worden.

7.2 Demo 2

De huidige simulator is zeer geschikt om visuele simulaties uit te voeren met de robot. We willen nu echter ook unit testen uitvoeren. Hiervoor hebben we besloten om visuele representatie optioneel te maken. Dit zorgt ervoor dat we in deze eenvoudigere modus niet langer gebonden zijn aan de ingebouwde vertragingen, die impliciet verbonden zijn met de visualisatie. Dit stelt ons in staat om snel uitgebreide tests uit te voeren en data te verzamelen voor analyse. Deze functionaliteit zit al gedeeltelijk in onze simulator. De enige functionaliteit die we moeten toevoegen zijn functies om onze gesimuleerde robots te ondervragen, zoals hun positie, richting en sensorwaarden.

Overigens werden andere doelstellingen voor demo 2 met betrekking tot de simulator reeds gehaald. De simulator werkte immers al op een continu niveau en ook onzekerheden waren reeds geïmplementeerd.

7.3 Demo 3

Functioneel werden tijdens de voorbereiding van deze demo geen aanpassingen gedaan aan de simulator. Wel werd hij, net zoals alle belangrijke componenten, onderworpen aan een grondige opkuisactie en werd de logica ingebed in een set van unit tests.

7.4 Simulatie versus realiteit

De simulator is gebouwd volgens een principe van continuïteit. Zo worden alle aspecten berekend volgens de fysische grootheden uit de realiteit. Alle sensoren zijn virtueel geïmplementeerd en berekenen op basis van de fysische eigenschappen van de echte robot de juiste verschillen tussen twee opeenvolgende stappen. Zo worden bijvoorbeeld de motoren gesimuleerd volgens het juist aantal rotaties per seconde, wat een effect heeft op de afstand die afgelegd is in relatie tot de echte omtrek van de wielen en de afstand tussen deze wielen.

Op deze manier gedraagt de robot in de simulator zich nagenoeg identiek aan zijn fysieke tegenhanger. Door gebruik te maken van exact dezelfde software, op zowel de fysieke robot als in de simulator, bekomen we een zeer realistisch beeld.

Hoofdstuk 8

Conclusies

8.1 Demo 1

In appendix A werden de verschillende relevante beoordelingsdocumenten toegevoegd.

8.1.1 Feedback

De robot reed consistent en robuust. Zelfs wanneer de robot fouten maakte werden deze hersteld. In het begin las de lichtsensoren foute waarden en reed de robot heen en weer. De introductie van het nieuwe algoritme om deze sensor automatisch te kalibreren lag hier aan de oorsprong. Na enige tijd was de kalibratie duidelijk beter en begon de robot alsnog het speelveld verder te verkennen. Hierbij viel op hoe robuust de implementatie van de *Driver* is.

De feedback van het didactische team was overwegend positief. Er werd duidelijk goed nagedacht over de softwarearchitectuur, zowel op niveau van robuustheid, uitbreidbaarheid als structuur. De gebruikte strategie i.v.m. Collaborative diffusion is goed gevonden en werkt duidelijk goed door zijn eenvoud. Het dashboard werd positief vermeld en de inhoud en opmaak van het verslag waren in lijn met de overige kwaliteit.

Naast opmerkingen over opmaak en verduidelijking, waren er ook tips met zicht op verbetering van de presentatie en het rijgedrag van de robot.

Als directe antwoord op deze opmerkingen kozen we voor het implementeren van een configuratie-klasse om de opstartsnelheid te verhogen, ook het presenteren zelf wordt vanaf nu beter voorbereid. Het rijgedrag wordt vooral geoptimaliseerd door het verwijderen van redundantie.

8.1.2 Reflectie

Hoewel we allemaal zeer tevreden waren met de prestatie die we leverden op de demo, zijn we nog niet tevreden over het niveau dat we willen bereiken met dit project.

We beslisten daarom om een week uit te trekken om een code review te doen. Op basis van de resultaten van die review beslisten we vervolgens wie wat zal doen en hoe grotere delen zullen aangepakt worden. Hieruit volgt dan een planning die de doelstellingen voor de overige demo's definieert. Meer uitleg over dit proces is te vinden in hoofdstuk 9.

In het algemeen hebben we beslist om te blijven bij een volledig autonome robot, hoewel dit beperkingen inhoudt ten gevolge van het beperkte geheugen van de robot. Naast een doorgedreven optimalisatie van het geheugengebruik, hebben we dit ook cijfermatig onderzocht en is dit een van de parameters waar we rekening mee houden tijdens de uitvoering van de logica op de robot. Sectie 6.1.4 gaat dieper in om dit aspect.

De simulator werkte niet zoals verwacht. Barcodes werden niet goed gelezen en dus hebben de robots geen gemeenschappelijke barcodes gevonden. Ook verdween de Pac-Man te snel en werden de robots terug door de nog niet verkende sector waar de Pac-Man op stond aangetrokken. Zo werd de rest van het veld niet meer verkend.

8.2 Demo 2

8.2.1 Feedback

De feedback kwam deze keer van een ander team, namelijk team zilver. Het grootste deel van deze feedback kwam neer op stijlopmerkingen i.v.m. het verslag. Het verslag zelf werd inhoudelijk overzichtelijk en gestructureerd bevonden. Het presentatiegedeelte kreeg echter welverdiende negatieve feedback. Zowel de virtuele simulator als de fysieke simulator faalde immers in hun uitvoering.

Ook merkten zij op dat tijdens de presentatie één van de gesimuleerde robots een fout had gemaakt bij het importeren van gegevens van een andere robot en dat wij dit niet pro-actief zelf aangekaart hadden.

8.2.2 Reflectie

Het rijgedrag van de robot bleef consistent met de vorige demo. De nieuwe licht-sensor implementatie bleek wel beter te werken dan de vorige keer.

De virtuele simulator bleek geen rekening te houden met het gebruik van een intern assenstelsel dat afwijkt van het gebruikte assenstelsel door de andere teams.

Een spijtig voorval dat zijn oorsprong vindt in onze voorsprong uit het eerste semester. Dit werd snel opgelost en getest met de andere teams. Enkele andere interpretatieverschillen met betrekking tot het protocol werden eveneens reeds weggewerkt.

De uitvoering van de strategie was duidelijk niet geslaagd, de implementatie van de grids was dan ook niet vervolledigd, bijgevolg werd de informatie van de teams nog niet correct gecombineerd. De doelstellingen opgesteld na demo 1 werden dus niet gehaald en daarom is het ook logisch dat de demo sterk tegenviel. We slaagden er wel in om het geheugengebruik te reduceren tot een werkbaar niveau, zonder functionaliteit te verwijderen.

8.3 Demo 3

Over het algemeen slaagde er niemand in het behalen van de doelstelling, namelijk Pac-Man vangen. Als we achteraf analyseren waarom niemand in deze opzet slaagde, bleek deze opdracht dan ook vrijwel onmogelijk te kunnen slagen. Het vangen van Pac-Man was immers niet genoeg, alle robots moesten ook aangeven dat er de Pac-Man wel degelijk *captured* was. Als men dan veronderstelt dat er geen enkel team in zijn uitvoering faalt is dit nog steeds een huzarenstukje. Men moet immers *mergen* met alle teams voor men daadwerkelijk de Pac-Man kan beginnen insluiten.

Vervolgens nemen we dan nog in acht dat de opgestelde doolhof vol plekken zat waar minstens vier *ghosts* nodig waren om in te sluiten. De combinatie van deze aspecten, maar ook het ontbreken van een ontwijkingsstrategie met de andere robots en het feit dat de Pac-Man door een mens bestuurd werd, maakten deze opdracht dus gedoemd tot falen. Dit wil natuurlijk niet zeggen dat veel teams met een slecht gevoel naar huis gaan, sterker nog, de meeste hebben de voor zichzelf gestelde doelen behaald. De demo duurde immers maar 10 minuten, voor een gebied van 64 sectoren met slechts enkele barcodes. Als deze van meer barcodes voorzien was geweest had men mogelijk al een interessantere situatie gehad. Een op voorhand afgesproken symmetrie had bijvoorbeeld ook geholpen.

8.3.1 Reflectie

Wij waren dan ook één van de teams die met een goed gevoel naar huis keerden. Hoewel we met een bang hart waren binnengegaan presteerde onze robot even goed of beter als de meeste andere robots. Als we dan in acht nemen dat wij een groot deel van onze tijd gespendeerd hebben aan het realiseren van deze opdracht binnen de beperkingen van een embedded systeem, dan kunnen we dit voor onszelf een meer dan geslaagde opdracht noemen. Zeker wanneer we dit cijfermatig

gaan uitdrukken: on een geheugengebruik van minder dan 64KB ten opzichte van letterlijk GBytes bij andere teams en een processorcapaciteit van 48Mhz t.o.v. van de dual en quad cores in de respectievelijke pc's gebruikt door de andere teams.

Verder vinden wij dan ook dat als er inderdaad iets moet gezegd worden omtrent het niet behalen van de doelstelling, dat dit vooral moet gaan over het samenwerkingsaspect. We vinden dat de laatste demo duidelijk heeft aangetoond dat het ontwikkelen van een protocol in een zijdelings traject een onverantwoord idee is. Als we alleen al kijken naar het minimale gebruik van sommige onderdelen van het protocol in de uiteindelijke uitvoering, dan is dit punt voldoende aangetoond. Beter had men op dag 1 een brainstormgroep gemaakt die dit probleem vanaf het begin in een werkbare oplossing goot en dit dan nog best onder sterke begeleiding van ervaren mensen.

Als laatste willen we nog toevoegen dat de individuele demo beter individueel was gebleven, er ontstond een grote chaos met de veelheid aan teams, waardoor er amper vragen gesteld worden. Wij begonnen ons dan ook al snel vragen te stellen of dit zo bedoeld was, aangezien dit niet zeer respectvol overkwam voor de vele uren die sommigen in dit project gestoken hebben.

Hoofdstuk 9

Procesbeschrijving

9.1 Demo 1

Onze werkwijze tijdens het eerste semester heeft zijn vruchten afgeworpen. We willen deze aanpak ook in het tweede semester verder zetten. Zo volgen we opnieuw twee paden naar ons doel: enerzijds wordt er verder gewerkt aan de simulator en wordt de robot verder verbeterd. Anderzijds starten we ook een traject op dat zich focust op het einddoel.

Omdat de scope van de eerste demo in essentie zo sterkt aanleunt bij het uiteindelijke doel en omdat we met de ontwikkeling van de simulator tijdens het eerste semester een voorsprong hebben t.o.v. de overige teams willen we de scope van de finale demo reeds als doelstelling nemen voor de eerste demo.

Dit gaat hand in hand met de tijd die we ter beschikking hebben tijdens de eerste weken van het semester, wanneer oefenzittingen voor andere vakken nog niet begonnen zijn. We kiezen dus voor een *blitzkrieg* waarbij we trachten om de totale scope in een korte tijd te realiseren, waarna we tijdens de resterende weken dan verder kunnen verfijnen.

De taken van coördinator (Michiel) en secretaris (Christophe) werden opnieuw verdeeld. Ruben is de nieuwe coördinator. Voor de rol van secretaris werd niet één teamlid aangesteld, maar werd besloten om dit een gedeelde verantwoordelijkheid te maken.

Ten gevolge van overlappingsen in het uurrooster en in samenspraak met het didactische team, zal Christophe gedurende twee van de vijf uren die standaard ingepland staan op maandag het team niet vervoegen. Deze uren worden enerzijds vlak voor, alsook 's avonds na deze vaste uren gepresteerd.

9.2 Demo 2

Aangezien we tijdens de demo 1 voorbereiding ons vooral gefocust hebben op de functionele scope, gebruiken we de tijd voor demo 2 om verfijning aan ons model en onze code aan te brengen. Hierbij denken we vooral aan bugfixes, het opkuisen van de code tree en het opkuisen van de klassen zelf.

Om te weten aan wat en hoe we extra aandacht moeten geven aan onze architectuur en code, evenals om iedereen in detail vertrouwd te maken met alle code, besloten we om in de eerste week na demo 1 tijd te nemen om alle code te overlopen. Tijdens dit proces hielden we een gedeeld Google docs bestand bij waarin we opmerkingen, fouten en/of vragen m.b.t. de code konden plaatsen. Hierop konden alle groepsleden commentaar geven of verduidelijkingen toevoegen. Deze lijst werd als basis gebruikt om dan het werk te verdelen over de groepsleden.

Naast deze aanpak gaan we ook proberen om het concept *pair programming* toe te passen. Zo is het de bedoeling dat we met twee paren en één losse speler die overal bijspringt werken. Dit gebeurt zowel tijdens als naast de zittingen op maandag.

9.3 Demo 3

Na de grote teleurstelling in demo 2 beslisten we onze aandacht op de architectuur en de code vol te houden. Onze refactoring aanpak werd wel aangevuld met het creëren van een uitgebreide unit test suite. Een dekking van honderd percent is enerzijds niet realistisch, maar is tevens ook niet nodig. Logischerwijze wensen we een dekking voor de relevante klassen. Bij de creatie van deze suite werd in parallel alle code nagekeken. Zo werd er veel ongebruikte code verwijderd en losten we een heel reeks bugs op. Aangezien we reeds tijdens de voorbereiding van de eerste demo getracht hadden de volledige scope te bereiken, waren we nu tevens in staat om functioneel alles tot in de details uit werken zonder onbekende delen.

Zo werd o.a. de volledige interne voorstelling van de sectoren aangepakt. Met de ervaring die we opgedaan hadden tijdens de voorbereiding van demo 2, waren we nu in staat om deze implementatie sterk te verbeteren met het oog op geheugengebruik en inzetbaarheid.

Verder valt er nog op te merken dat we bij dit proces eerst bepaalde interfaces opnieuw gedefinieerd hebben, zodat we allen konden werken aan afgeronde gehelen. Dit had natuurlijk als nadeel dat de volledige source code onbruikbaar was tot al deze gehelen opnieuw geïntegreerd waren. We geloven er sterk in dat dit proces onze code veel stabielier heeft gemaakt en onze robot hier dus baat bij heeft.

9.4 Unit testen

Na de eerste demo van het tweede semester werden we geconfronteerd met de gevolgen van het negeren van één van de belangrijkste geboden van software ontwikkeling: *Gij zult uw code testbaar maken en testen*. Bijna elke aanpassing aan één deel van de code zorgde voor verschrikkelijke gevolgen aan een ander deel. Elke kleinste wijziging brak andere delen van de oplossing.

Gegeven de manier waarop de software ontwikkeld was tot dan toe, was dit niet geheel onbegrijpelijk. We hadden twee opties: verder gaan op de manier die we gehanteerd hadden en elk probleempje proberen te *fixen* of structureel een ingrijpende verandering doorvoeren in de manier waarop we met de code omgingen.

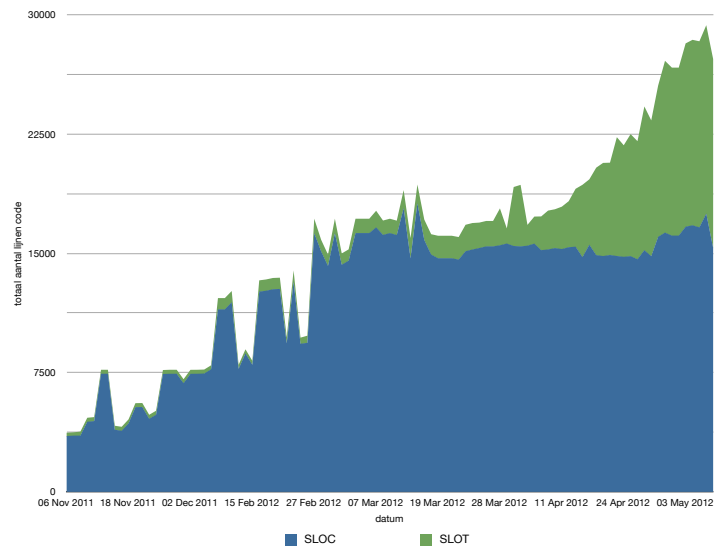
Aangezien één van de doelen van P&O Computerwetenschappen net is om te leren in praktijk hoe een middelgroot softwareproject dient uitgebaat te worden, hebben we deze kans aangegrepen om een stap verder te gaan en zelf aan den lijve te ondervinden of en hoe een andere aanpak effectief meer stabiliteit kan brengen. We begonnen met het invoeren van unit testen voor zoveel mogelijk code.

Aangezien de functionaliteit compleet was, konden we de volledige scope overzien. De architectuur die we gekozen hadden bleek een goed antwoord voor de meeste van de uitdagingen die in de opdracht vervat zaten. Vertrekkende van deze twee pijlers konden we dus sequentieel door de volledige code gaan en elke klasse op zich perfect vormgeven en voorzien van testen.

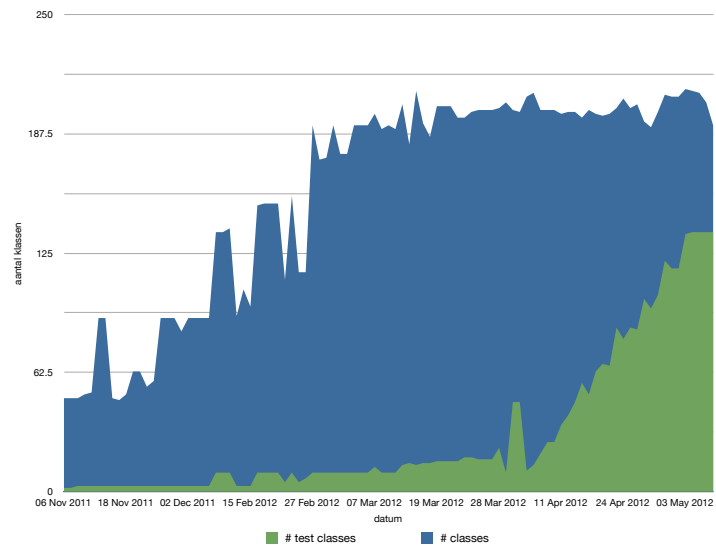
Met een codebase van ongeveer 17.000 lijnen code is dit echter geen sinecure en ging dit veel tijd vragen. Het alternatief was nog veel onzekerder: verder gaan met een onstabiele codebase zou mogelijk een onbetrouwbare oplossing met zich meebrengen en de wetenschap hieromtrent zou net een falen zijn in één van de belangrijkste aspecten van de opdracht.

Met nog vier weken voor de finale deadline begonnen we aan een grote onderneming. Week na week groeide het volume aan testen en begon de *coverage* stilaan nuttige vormen aan te nemen. Toen uitspraken als "*Godzijdank dat we de unit testen hebben.*" bij alle teamleden begonnen te rijzen, wisten we dat we op de goede weg zaten. Figuur 9.1 toont de evolutie van de volledige codebase en het aantal lijnen code (SLOC). Hier is duidelijk te zien dat vanaf begin april er een sterke en gestage toename is van het aantal lijnen code te wijten aan het structureel invoeren van unit testen. Ook toont deze grafiek zeer duidelijk dat ook de groei van de functionele code gestopt is en dat er zelfs een lichte daling is op te tekenen.

Figuur 9.2 toont dan weer het aantal klassen in het project samen met de *coverage* van de unit test klassen. De evolutie is natuurlijk gelijklopend, maar beide grafieken bieden een ander inzicht in de evolutie. Op amper vier weken bereikten we reeds een *class coverage* van meer dan 60%. Ten tijde van de finale demo hadden we zelfs reeds 70% bereikt.



Figuur 9.1: Evolutie van het totale aantal SLOC en het aandeel van de unit testen



Figuur 9.2: Evolutie van het aantal klassen en de coverage van de unit test klassen

Alle voordelen van een codebase met een goede *coverage* begonnen zich op te werpen: bij elke commit kon een *RunAllTests* test suite doorlopen worden en bracht een succesvolle uitvoering een gegronde zekerheid dat een aanpassing geen problemen had teweeggebracht in andere delen van de oplossing. Figuur 9.3 toont deze uitvoer en bevestiging als ook een indicatie van de *class coverage*.

[illegible]

Figuur 9.3: De RunAllTests test suite in actie.

Deze aanpak heeft natuurlijk één groot nadeel. Sequentieel door de volledige codebase lopen, alle code opschonen en elke klasse op zich van de nodige unit testen te voorzien, betekent dat de codebase als een geheel niet langer functioneel is. Dit was een risico dat we moesten nemen om zelf te kunnen ondervinden of deze piste inderdaad een onderbouwde zekerheid kan bieden.

Al het werk heeft uiteindelijk zijn vruchten afgeworpen. Tijdens de laatste voorbereidingen voor de demo hebben de unit testen ons meermaals beschermd tegen ondoordachte keuzes. Met een uiteindelijke *class coverage* van meer dan 70% voelden we de ruggesteun van de testen keer op keer en waren we bij elke aanpassing zeker dat er op andere plaatsen geen problemen ontstonden.

Hoofdstuk 10

Werkverdeling

Aangezien de samenstelling van ons team ons opzadelt met een moeilijke agenda is het gevaarlijk om een op zich staand deel toe te wijzen aan één persoon. Dikwijls moeten we werk van elkaar overnemen om een minimale planning te respecteren. Op die manier moet en mag ook iedereen zich meester maken over alle aspecten van de code.

Natuurlijk zijn er delen die door één persoon meer bekeken worden en we trachten ook expertises optimaal te benutten. Op deze manier kunnen we een ruwe werkverdeling als volgt toekennen aan de verschillende leden:

10.1 Ruben

Ruben was tijdens dit semester de nieuwe coördinator. Daarnaast heeft hij zich hoofdzakelijk bezig gehouden met de technische componenten van de robot zelf. Een tweede belangrijke taak voor hem was het verder op punt stellen van de simulator. Met zijn zeer goede wiskundige en algoritmische achtergrond kon hij complexe problemen dikwijls in zeer weinig code en zeer optimaal oplossen.

10.2 Michiel

Na zijn werk aan de *Bluetooth* laag in het eerste semester, heeft Michiel zijn tanden gezet in het *AggregatedGrid* verhaal. Deze uiterst complexe materie werd nog verder bemoeilijkt door de beperkingen aangaande het geheugen van de robot. Hij heeft zich tevens voor demo 1 ook bezig gehouden met de implementatie van de Driver en de correcties op de robot.

10.3 Thomas

Thomas heeft zich ontpopt tot de *Swing* specialist en heeft tal van verbeteringen aangebracht aan de visuele kant van de simulator. Verder heeft hij de nieuwe *DashboardReporter* en *GhostProtocolHandler* tot in de kleinste details uitgewerkt en getest.

10.4 Florian

Met de afwezigheid van een secretaris-rol tijdens het tweede semester, heeft Florian deze rol dikwijls op zich genomen. Verder heeft hij zich ontfermd over tal van unit testen en heeft hij consequent *pairs* gevormd met iedereen die bijgestaan moest worden. Op het moment dat de opslagcapaciteit voor het programma een probleem werd, heeft Florian de zoektocht getrokken naar een oplossing om onze *class files* te comprimeren.

10.5 Christophe

Tijdens het eerste deel van het semester heeft Christophe onderzoek gedaan naar het *Collaborate Diffusion* algoritme en hiervoor o.a. de mini simulator gebouwd. Daarna heeft hij de basis gelegd voor het latere werk van Michiel omtrent de *AggregatedGrid*. Daarnaast tracht hij vooral de interfaces en top-level applicaties in goede banen te leiden. Hierbij staan het *Dashboard* en bijhorende componenten centraal.

Hoofdstuk 11

Kritische analyse

P&O Computerwetenschappen 2011-2012 bevatte alle elementen van een gefaald project: geen ervaren teamleden, vage vereisten, deadlines voor alles, geen analyse(tijd), een puur technische focus,... In het bedrijfsleven vormt dit het schoolvoorbeeld van hoe een softwareproject niet mag opgezet worden. De leerschool van dit project moet dan ook gezocht worden in de ervaring van zulk een *slecht* voorbeeld en het hopelijk herkennen en vermijden in de toekomst.

Ondanks dit negatieve kader, heeft Team Platinum resoluut gekozen voor de uitdaging om iets te leren. Twee belangrijke pijlers stonden voor ons voorop: software kwaliteit en een consequente aanpak. Het eerste is evident, maar legt een strenge bijkomende eis op. De ervaring binnen het team was duidelijk groot genoeg om te trachten ook op puur software-vlak een mooi design en uitwerking te realiseren.

De consequente aanpak bestond er voor Team Platinum in, om ook in het tweede semester een echt autonome robot op te leveren. Daar waar andere teams ingingen op de mogelijkheid om gebruik te maken van een PC om bepaalde delen van de software op uit te voeren, kozen wij er voor om alle logica op de robot zelf te implementeren.

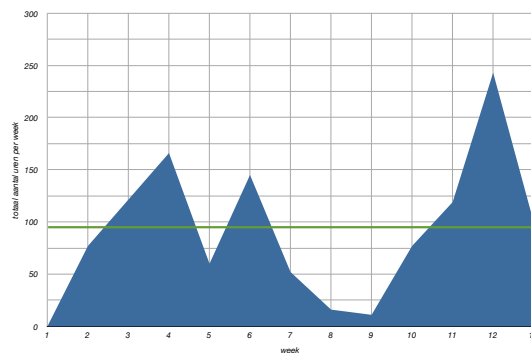
Beide extra focussen legde zonder twijfel een redelijke druk op de schouders van alle team leden. Maar het resultaat mocht er zijn. Tijdens de finale demonstratie presteerden veel robots even goed of zelfs minder goed dan onze robot. Gegeven het grote verschil in mogelijkheden, zijn we trots op het resultaat dat we opgeleverd hebben, zowel op vlak van de robot, als op vlak van de softwarekwaliteit en in extremis ook nog van het niveau van de testen die het geheel inkapselen.

Maar we hebben ook veel fouten gemaakt. De sterke karakters van Team Platinum beten zich niet alleen op een positieve manier vast in een probleem, maar soms ook op negatieve wijze, *te lang*. Door deze volharding werden problemen op bepaalde momenten eerder groter. Dit typische probleem had ondervangen kunnen worden door een gedetailleerdere planning en betere interne communicatie.

We zouden hier kunnen aanbrengen dat we bijna als enige team met vijf totaal andere agenda's dienden rekening te houden en dat we uit totaal andere richtingen kwamen, maar feitelijk hebben we dit zelf nooit als een groot probleem ervaren. Team Platinum is een hecht team dat elkaar gevonden heeft in doorzettingsvermogen en in een drijfveer om kwalitatieve resultaten te boeken.

Verder heeft heel het team aan den lijve ondervonden wat unit testen kunnen betekenen voor een project. Gedurende ruim $2/3$ van het project voelden we op alle mogelijke manieren de afwezigheid ervan en op het einde ervoeren we meermaals de kracht van een degelijke unit test suite. De algemene teneur was unaniem: we hadden vanaf dag één met unit testen moeten beginnen en we hadden nooit zulke problemen gehad zoals bij de tweede demo.

Tot slot traptten we keer op keer in de val van het *time management*. Wanneer we het tijdbestedingspatroon in figuur 11.1 van het gehele team samen bekijken, merken we dat er drie grote piekmomenten zijn geweest, die toevallig samenvielen met de drie demo's.



Figuur 11.1: Gepresteerde uren per week.

Op dezelfde figuur is met een groene lijn het gemiddelde aangeduid dat gepresteerd werd. Het is duidelijk dat mits een betere planning de *zgn.werk-golven* niet opwaarts maar zelfs neerwaarts hadden kunnen verlopen, met veel minder stress tot gevolg. De piekmomenten van de demo's eisten een dalmoment in de weken daarna om opnieuw aandacht aan andere vakken te kunnen geven. Op deze manier ontstond een vicieuze cirkel die we stilaan wel erkenden, maar nog moeilijk konden doorbreken.

De aanleiding van de piekmoment was meestal telkens dezelfde: overschatting van de mogelijkheden van het team en het moeten realiseren van deze initieel te licht ingeschatte geplande aanpassingen. Een degelijke analyse, vooral op functio-

neel vlak, als een meer afgelijnde werkverdeling hadden hier zonder twijfel soelaas kunnen brengen.

Naast de interne evolutie, waren er ook externe factoren waar we beïnvloed door werden. De spelregelcommissie en de beslissingen die door dit orgaan genomen werden werkten zeker niet in ons voordeel. Ook hier hebben we de bal waarschijnlijk mis geslagen op het vlak van communicatie en hebben we te snel de handdoek in de ring geworpen, waardoor de uitkomst van de commissie nog verder ontaard is in een volledig gemiste kans om te leren samenwerken en te leren hoe een commissie echt kan werken.

De slotsom voor elk van de leden van Team Platinum is ronduit positief: we hebben de doelstellingen van het vak bereikt en we hebben onze eigen doelstellingen bereikt. Bovenal heeft elk teamlid het gevoel veel bijgeleerd te hebben, dankzij de robot en vooral van elkaar.

Bijlage A

Beoordelingen

A.1 Demo 1

Team Platinum

A.1.1 Verslag

1.a Grote problemen

- in sectie 1: "iedereen is ondertussen vertrouwd geraakt met de volledige architectuur"
gevaarlijke opmerking

1.b Kleinere opmerkingen

- Figuur 5.1, 5.2 dienen beter uitgelegd te zijn. Wat is wat? Legende van kleuren ontbreekt.
- Figuur 4.1: fit een lijn door de punten om de bewering van exponentiële lijn te ondersteunen
- Referentie naar een wetenschappelijk paper kan beter in Referenties sectie (via bibtex).
- Sectie 4.1.1: wat is het nut van laatste zin? (wat als de robot rijdt??)
- Referenties naar Wikipedia zijn niet nodig, indien nodig beter verwijzen naar een boek
- mini simulator? welk doel heeft dit? is het niet gewoon een weergave van een simulatie?

1.c stilistische opmerkingen

- typo's: ivm \rightarrow i.v.m., jlasse \rightarrow klasse

A.1.2 Presentatie

2.a Presentatie zelf

- presentatie was niet voorbereid, langzame start
- wel een van de betere demo's
- denk nog eens na over optimaliseren van rijgedrag (robuustheid versus snelheid)

2.b antwoorden op vragen

- Geen.

A.1.3 Positieve punten

- software zit goed in elkaar
- zeer aangename compacte schrijfstijl
- analyse van ghost protocol is zeer goed
- Collaborative diffusion is knap gevonden
- dashboard gui is duidelijk

A.2 Demo 2

Team Platinum

A.2.1 Verslag

1.a Grote problemen

- Opgelegde structuur niet nageleefd: strategie en softwareontwerp omgewisseld; beoordelingen in appendix A in plaats van in conclusie.
- Sectie 6: Sequentiediagramma toevoegen van bijvoorbeeld sensorwaardeaanroep. Hoe snel wordt er bemonsterd?

1.b Kleinere opmerkingen

- Namen van de teamleden niet vermeld op voorblad.
- Sectie 2.1: De ghosts, meer bepaald de gesimuleerde robots, hebben wél kennis van het parcours nodig (volledige kaart gegeven voor elke demo), weliswaar niet rechtstreeks. Nuanceren hierover nodig in het verslag.
- Sectie 4.1.1: Op welke manier kan de robot gemakkelijker bestuurd worden door de NXT Brick om te keren?
- Sectie 4.1.1: Foto's van de robot kunnen helpen de beschrijving te ondersteunen.
- Sectie 6: Is er overwogen om gebruik te maken van een PC i.p.v. de robot alles te laten bijhouden en zelf te laten beslissen? Hierdoor zou immers het geheugenprobleem verhoplen kunnen worden.
- Sectie 6.1.4: Waarom is een minimum van 60 FPS noodzakelijk? Na ongeveer 17 ms ($=1000/60$) zijn (waar- schijnlijk) nog niet alle gemeten waarden van de sensoren veranderd. Is dit bijgevolg niet redundant?
- Sectie 6.1.4, deel geheugengebruik: 'mini-simulator': er wordt verwezen naar een verdere sectie in het verslag; dit is ongebruikelijk.
- Meermaals vermeld dat er in het eerste semester reeds een simulator gemaakt is (samenvatting, sectie 7 pag. 26 en 27).

1.c stilistische opmerkingen

- Subjectief taalgebruik, onder meer:
 - "Spijtig genoeg...", pag. 6;
 - "... zou het dom zijn te veronderstellen dat...", pag. 12.
- Inconsistentie, onder meer:
 - Pac-Man, de Pac-Man, Pac-Man-robot;
 - robot, ghost, spook;
 - IR, infrarood (en samenstellingen).
- Schrijf-/taal-/tikfouten:
 - Samenvatting: jaaroverschrijdend → jaaroverschrijdend, Pac-Man-spel → Pac-Manspel, Open Bron project → Open Sourceproject;

- Sectie 2.1: "staat (...) stil op het parcours": in → op;
 - Sectie 2.3.2: Pac-Man spel → Pac-Manspel;
 - Sectie 3.2: "(...) en bespreken we de voor- en nadelen ervan"; mbt. → m.b.t.;
 - Sectie 4.1.2: "(...) IR-bal en -sensor." → "(...) IR-bal en -sensor.", AC modus ! AC-modus (ook van toepassing op analoge situaties);
 - Sectie 6.1.1: Leesteken vergeten na laatste zin;
 - Sectie 6.1.2: configuratie aanpassingen → configuratieaanpassingen;
 - Sectie 6.1.3: muur-informatie → muurinformatie;
 - Sectie 7.1.3: Simulator Modi → Simulatormodi, MQ-server → MQ-server;
 - Sectie 9.2: Laatste 'zin' van deze sectie heeft geen werkwoord;
 - Sectie 10: team lid → teamlid.
- Vreemd taalgebruik:
 - Samenvatting: programmatie, "effectieve fysieke robot";
 - *Map* veelvuldig gebruikt i.p.v. *kaart*, o.m. op pag. 3.
 - Andere opmerkingen:
 - bij voorkeur enkele aanhalingstekens gebruiken i.p.v. dubbele, in het bijzonder bij het benadrukken van één woord; nog beter is gebruik maken van cursivering i.p.v. aanhalingstekens, dit is minder storend bij het lezen;
 - zo weinig mogelijk woorden benadrukken om leesbaarheid te bevorderen, bv. sectie 5.1: "In de voorstelling ontstaat er een "deuk" in de "geur" achter de "ghost".";
 - veel passieve zinnen (wordt, worden, werden,...);
 - gebruik van het woord *men* bij voorkeur te vermijden;
 - L^AT_EX afkortingen zoals i.v.m. en t.o.v. laten eindigen met backslash, zodat een 'normale' spatie gebruikt wordt i.p.v. een grotere zoals op het einde van een zin gebruikelijk is;
 - bij voorkeur gebruik maken van *sections* i.p.v. chapter in de L^AT_EX-broncode; hoofdstukken zijn eerder gebruikelijk bij boeken en cursussen.

A.2.2 Presentatie

2.a Presentatie zelf

- Paar minuten demonstratietijd verloren door te laat aanwezig op demo. Hierdoor is bijvoorbeeld niet gedemonstreerd of de fysieke robot Pac-Man kan signaleren en/of insluiten.
- Niet zelf gezegd op de demo dat de kaarten foutief zijn samengevoegd.

2.b antwoorden op vragen

- Geen duidelijk antwoord op de vraag waarom de kaarten verkeerd toegevoegd zijn.

A.2.3 Positieve punten

- Algemene indruk van het verslag is positief:
 - vlotte schrijfstijl, hoewel soms informeel en indruisend tegen de regels van een wetenschappelijk verslag;
 - sectie 6.1.4 in het bijzonder is erg goed geschreven: aangenaam om te lezen en wetenschappelijk onderbouwd.
- Algemene indruk van de presentatie is positief:
 - de presentatie was kort en bondig, hoofdzaken kwamen aan bod, details zijn achterwege gelaten;
 - correct taalgebruik tijdens de presentatie.

A.3 Demo 2: Beoordeling Team Rood

Team Rood

A.3.1 Verslag

Vorige Beoordeling toegepast? Algemeen gezien is er nog steeds een grote overlap tussen de verschillende hoofdstukken en houdt men zich niet goed aan de onderverdeling.

Voetnoten zouden een verbetering zijn, want deze verbergen details en kunnen bijvoorbeeld verwijzen naar een verder hoofdstuk als iets nog moet uitgelegd worden (bvb *NearestIncompleteSectorFinder*). Dit geldt niet enkel voor voetnoten, maar

ook in het algemeen. Men zou minder moeten herhalen in het verslag en ook moeten vermelden als er iets verder in de tekst wordt uitgelegd.

1.a Grote problemen

- Waarom wordt er geen geaggregeerde voorstelling van de info gemaakt i.p.v. de info te negeren? Een voorstelling van de huidige wereld met behulp van info komende van alle robots zou logischer zijn, er is immers geen geheugen-gebrek. Mogelijks is dit voor later gehouden.
- Waarom wordt alles op de PC gedaan? Misschien kan men informatie toevoegen i.v.m. onderzoek naar de bluetoothverbinding? Alles stond in het eerste semester op de robot dus dient er toch uitgelegd te worden waarom men deze overschakeling doet?
- Er werd volgens ons te weinig aandacht besteed aan het protocol. Wat waren de voor- en nadelen op het vlak van implementatie voor team rood? Wat hadden ze achteraf gezien anders gedaan, wat hebben ze niet nodig, wat willen ze er nog aan toevoegen, etc ?

1.b Kleinere opmerkingen

- sectie 2.1.1: De boodschappen worden in plain text verstuurd en zijn ook begripbaar voor mensen, wat debugging en implementatie eenvoudiger maakt.(Dit maakt implementatie net minder eenvoudig)
- sectie 2.1.2: ... bijzonder sterk is, kan Pac-Man door de muren gezien worden.(men bedoelt waarschijnlijk dat er reflecties zijn, slecht verwoord)
- sectie 2.2.2:...er geen botsingen tussen de ghosts... (wel botsingen?)
- sectie 4.1.3: De eerste sector van dit pad wordt uitgestuurd naar de robot via DataSender.(de robot krijgt enkel richtingen toegestuurd?)
- sectie 4.1.4: ... de binnenkomende informatie genegeerd.(Is dit enkel na mergen? Waarom zelfs niet daarna iets doen met deze info? Als meerdere remotes dezelfde info bevestigen, wordt deze toch geloofwaardig.)
- sectie 4.2.3: Waarom staat de BarcodeScanner op de robot en niet op de computer? Dit is toch logica. Wordt de bluetoothcommunicatie dan overbelast met lichtsensor waarden?
- sectie 5.1.2: Een alternatieve, naïeve... (Dit is een overbodige zin.)

1.c stilistische opmerkingen

- sectie 1.1 : Belangrijk is dat... (zinsconstructie)
- sectie 2.1.1: zodra i.p.v van zodra (meermaals in tekst), beide i.p.v beiden,
- sectie 3.1.2: (hardware) is een overbodige vermelding
- sectie 3.2.2: veel voorkomend (veelvoorkomend)
- sectie 4.1.1: geupdateted (geüpdatet), data is een meervoud
- sectie 4.1.3: receiver- en sendervelden (streepje en velden aan sender, vaak geen streepjes)
- sectie 4.2.2: de zien is (te zien is), achtereenvolgens(s i.p.v d)
- sectie 4.4.1: Er zijn enkele enkele(dubbel)
- sectie 5.1.2: Dit om te vermijden dat (i.p.v. dit om in de te vermijden ...)
- sectie 5.2.1: Een stuk doolhof dat niet verkend kan worden(...)wordt niet verkend. overbodig(meermaals overbodige constructies)
- sectie 6.1.2: dat de robot(tweemaal achtereen)
- sectie 7.1.1: een keer in de mist gegaan(de mist ingegaan)
- sectie 8.1 : Er was geen tijd om te testen ...(in deze zin missen enkele leestekens)
- overige : overbodige komma's ; zinnen zonder werkwoord (bvb sectie 3.1.2: zoals bijvoorbeeld het wit...); samenstellingen schrijven met spatie ertussen is fout; tussen persoonsvormen komma gebruiken; soms enkelvoud gebruikt i.p.v. meervoud bij werkwoorden; foutief gebruik van koppeltekens bij samenstellingen
- consistentie: IRSeeker en IR-Seeker; cursiveren van de klassenamen, methoden, enz. ; RabbitMQ-protocol; Integerwaarde(aan elkaar soms af elkaar)

A.3.2 Presentatie

2.a Presentatie zelf

- De inleiding was niet zo spontaan als gewenst, er werd van een papier afgelezen.
- Er werd benadrukt dat de correcties op het rijden waren verwijderd om de snelheid op te hogen, dit kwam de robuustheid van de robot duidelijk niet ten goede aangezien er drie maal manueel gecorrigeerd moest worden. Nochtans werd er op voorhand vermeldt dat aan het rijgedrag van de robot hard gewerkt is.
- sectie 4.2.2: Het is de verantwoordelijkheid van de computer om ervoor te zorgen dat de robot niet tegen een muur rijdt. Lagen hier de fouten misschien aan, zijn er tests over de vertraging van de communicatie?

2.b antwoorden op vragen

A.3.3 Positieve punten

- Robot redesign werd zeer goed onderbouwd.
- De scenario analyses zijn zeer goed voor de strategie, gevaar is natuurlijk wel dat als dit te veel gedaan wordt dat het verslag zijn bondigheid uit het oog verliest. Deze scenario's kunnen ook de indruk wekken dat er enkel gedachtenexperimenten werden opgezet omtrent zeer specifieke situaties, i.p.v. de algemene strategie uitgebreid te testen. Het blijft wel een zeer aangename manier om de denkstrategie toe te lichten.

Bijlage B

Grafische User Interface

Het project voorziet verschillende gebruikers interfaces: Zo is er een dashboard dat informatie geeft over verschillende parameters van de werking van een robot. Daarnaast geeft een simulator een beeld van de gesimuleerde wereld en van de manier waarop de gesimuleerde robot de wereld ervaart. Tot slot is er een administratie *shell* om een robot van op afstand commando's door te sturen.

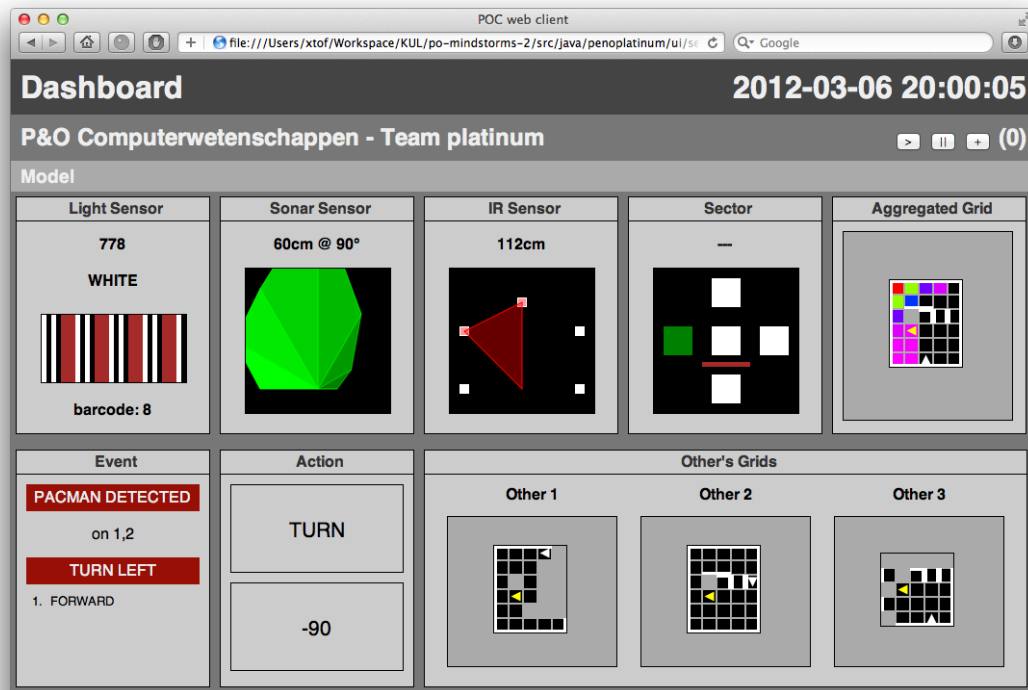
B.1 Dashboard

Het *Dashboard* is een webtoepassing die informatie ontvangt uit de databank waarin de *Gateway* de ontvangen feedback van een robot opslaat. De webtoepassing ontvangt deze informatie via een *JSONP*¹ connectie. Op deze manier kan de webtoepassing toch in real-time een informatiestroom van de robot ontvangen.

Figuur B.1 toont het *Dashboard* met al zijn samenstellende delen. Het principe van het *Dashboard* bestaat uit een reeks van onafhankelijke panelen die elk een conceptueel deel van de informatie waarover de robot beschikt voorstellen. Zo zijn er typisch panelen voor elk van de sensoren. Een paneel voor een sensor toont enerzijds de laatst gelezen waarde, maar tracht ook een nuttige histogram van de informatie aan te bieden. Zo is er bijvoorbeeld voor de lichtsensor een voortschrijdende balk waarop barcodes visueel ook kunnen herkend worden. De sonar wordt weergegeven zoals een klassieke sonar, enz. Daarnaast worden de verschillende *Grids* gevisualiseerd. Tot slot zijn er nog panelen die informatie geven over de gebeurtenissen waarop de robot reageert en de acties dat hij hierdoor onderneemt.

¹JSONP wordt typisch opgebouwd met een lang-lopende connectie waarover de server constant nieuwe Javascript commando's verstuurd. Deze worden op de client uitgevoerd, waardoor updates van bijvoorbeeld data kunnen verwerkt worden. Zie ook <http://en.wikipedia.org/wiki/JSONP>

Dankzij de uitwerking aan de hand van HTML en Javascript kunnen we het *Dashboard* zeer eenvoudig aanpassen en panelen bijmaken of verwijderen.

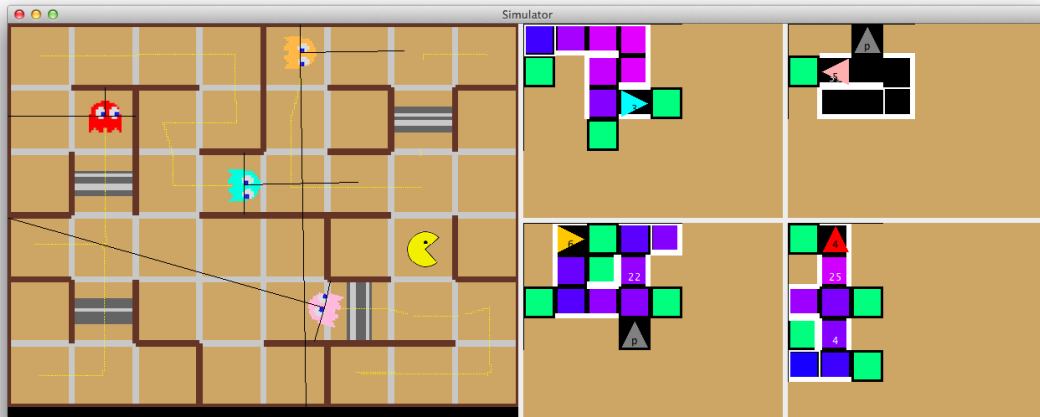


Figuur B.1: Overzicht van de functionaliteit van het Dashboard

B.2 Simulator

De simulator, weergegeven in figuur B.2, bestaat uit twee grote delen: één overzichtsscherm en maximaal vier kleinere deelschermen voor de visualisatie van de *Grid* van elk van de gesimuleerde robots. Deze visualisatie van de *Grid* wordt voorzien door de *GridView*, een optionele component die verbonden kan worden met de effectieve *Grid* van een robot.

De simulator wordt opgestart aan de hand van de *SimulationRunner*. Deze applicatie biedt tal van mogelijkheden om de simulator op te starten. Zo kan elke mogelijke *Driver*, *Navigator*, ... opgegeven worden op de command line, waardoor het een uiterst flexibele tool wordt. Figuur B.3 toont het help scherm met alle opties.



Figuur B.2: De Simulator simuleert vier robots.

```

simulator — bash — 80x25
astroboy:simulator xtof$ make run OPTIONS=--help
usage: SimulationRunner
  -d,--driver <arg>          use driver <classname>.
                             default=penoplatinum.driver.GhostDriver
  -g,--gatewayClient <arg>   use gatewayClient <classname>.
                             default=penoplatinum.simulator.SimulatedGatewayClient
  -h,--help                  show this helpful information.
  -m,--map <arg>             use mapfile. default=../maps/wolfram.txt
  -n,--navigator <arg>       use navigator <classname>.
                             default=penoplatinum.pacman.GhostNavigator
  -p,--reporter <arg>        use reporter <classname>.
                             default=penoplatinum.pacman.DashboardReporter
  -q,--quiet                 don't show a user interface.
  -r,--robot <arg>          use robot <classname>.
                             default=penoplatinum.pacman.GhostRobot
  -s,--start <arg>          use start number <number>. default=0
astroboy:simulator xtof$

```

Figuur B.3: Opties om de simulator op te starten.

B.3 RASH - Robot Administratie Shell

De *Robot Administratie Shell*, kortweg *rash*, werd in het leven geroepen om commando's door te sturen naar de robot. De nood voor een dergelijke tool ontstond uit de noodzaak om bijvoorbeeld een robot te verplichten om het gewone verloop van de start-procedure, zoals gedefinieerd in het *GhostProtocol*, te doorbreken en de robot te verplichten om toch te starten.

Aangezien er in het protocol impliciet ruimte is gelaten om eigen commando's te specificeren, door het verplichten van het negeren van onbekende commando's, en deze via de message queue te sturen, kozen we om een generiek eigen *custom* commando te gebruiken:

```
<name> PENOPLATINUM_CMD <signature> <counter> <command>
```

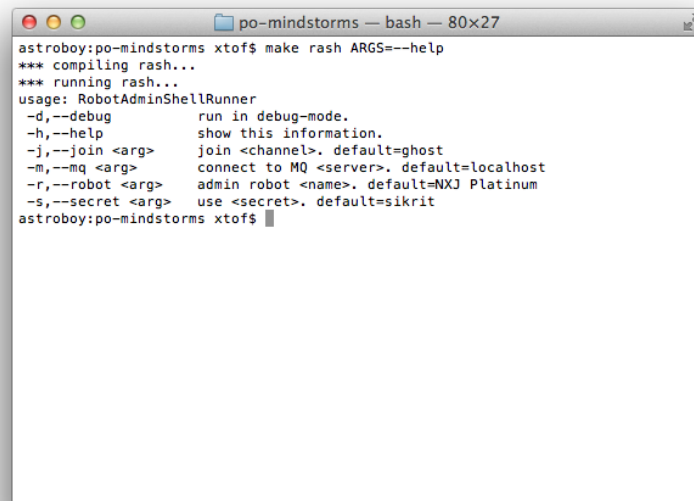
Hierin is *name* de naam van onze robot, *signature* een MD5 hash die dient om het *PENOPLATINUM_CMD* te beveiligen, *counter* een teller om te zorgen dat elk bericht uniek is en niet meerdere malen zou geaccepteerd worden en *command* een commando met eventueel eigen argumenten. Door deze opbouw is het makkelijk om meer eigen commando's toe te voegen. Voorlopig is een *FORCESTART* commando op deze manier geïmplementeerd. Figuur B.4 toont *rash* opgestart met het *help* argument, waardoor een overzicht wordt getoond van de mogelijke command line argumenten.

Figuur B.5 toont *rash* in actie², waarbij het *FORCESTART* commando uitgevoerd wordt.

De opstelling met *rash* bood ons nog een bijkomende opportuniteit. Zo kunnen we *rash* ook gebruiken om een log van een sessie opnieuw af te spelen. Dit maakt het mogelijk om een reeks commando's van andere robots te simuleren en zo een uitgebreide communicatie tussen andere robots en onze eigen robot te testen zonder effectief andere robots nodig te hebben.

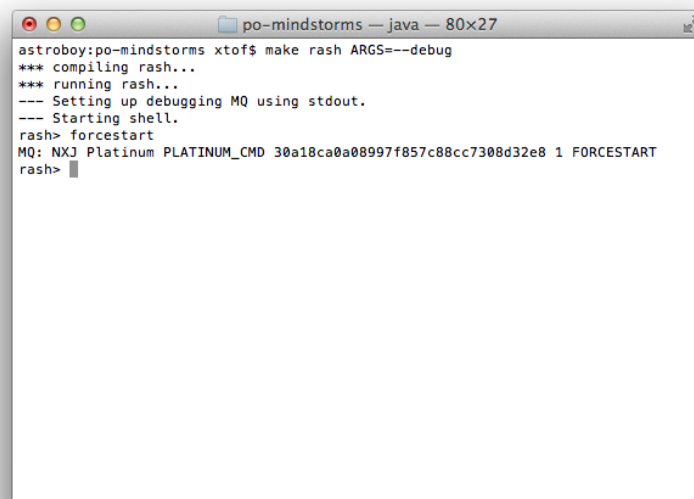
De keuze om van *rash* een aparte applicatie te maken was evident: het beheer van de robot op dit niveau is niet verenigbaar met het eenvoudig volgen van informatie omtrent de robot, zoals dit in het *Dashboard* mogelijk is. De gevoeligheid van de functionaliteit is veel hoger. Anderzijds is het moeten toepassen van deze acties eerder uitzonderlijk. Toch is het design van *rash* zo uitgewerkt dat een integratie met bijvoorbeeld het *Dashboard* mogelijk is. De effectieve logica is ondergebracht in een *RobotAdminClient*. Deze is ingebed in de eigenlijke gebruikersinterface, de *RobotAdminShell*. Mits het toevoegen van een schrijfbaar veld in het *Dashboard* en het opvangen van de inhoud van dit veld in de *Gateway*, kan deze laatste de *RobotAdminClient* zonder aanpassingen gebruiken om ook langs dit kanaal ook dezelfde functionaliteit te implementeren.

²*rash* voorziet een *debug* mode waarbij de boodschappen die naar de MQ server wordt verstuurd op de console getoond worden.



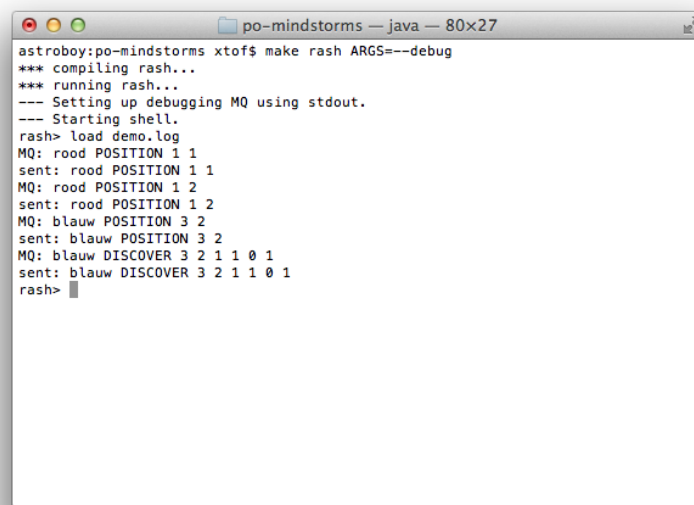
```
astroboy:po-mindstorms xtof$ make rash ARGS=---help
*** compiling rash...
*** running rash...
usage: RobotAdminShellRunner
  -d,--debug          run in debug-mode.
  -h,--help           show this information.
  -j,--join <arg>     join <channel>. default=ghost
  -m,--mq <arg>       connect to MQ <server>. default=localhost
  -r,--robot <arg>    admin robot <name>. default=NXJ Platinum
  -s,--secret <arg>   use <secret>. default=sikrit
astroboy:po-mindstorms xtof$
```

Figuur B.4: Het help overzicht van rash.



```
astroboy:po-mindstorms xtof$ make rash ARGS=---debug
*** compiling rash...
*** running rash...
--- Setting up debugging MQ using stdout.
--- Starting shell.
rash> forcestart
MQ: NXJ Platinum PLATINUM_CMD 30a18ca0a08997f857c88cc7308d32e8 1 FORCESTART
rash>
```

Figuur B.5: Het versturen van het FORCESTART commando met rash.



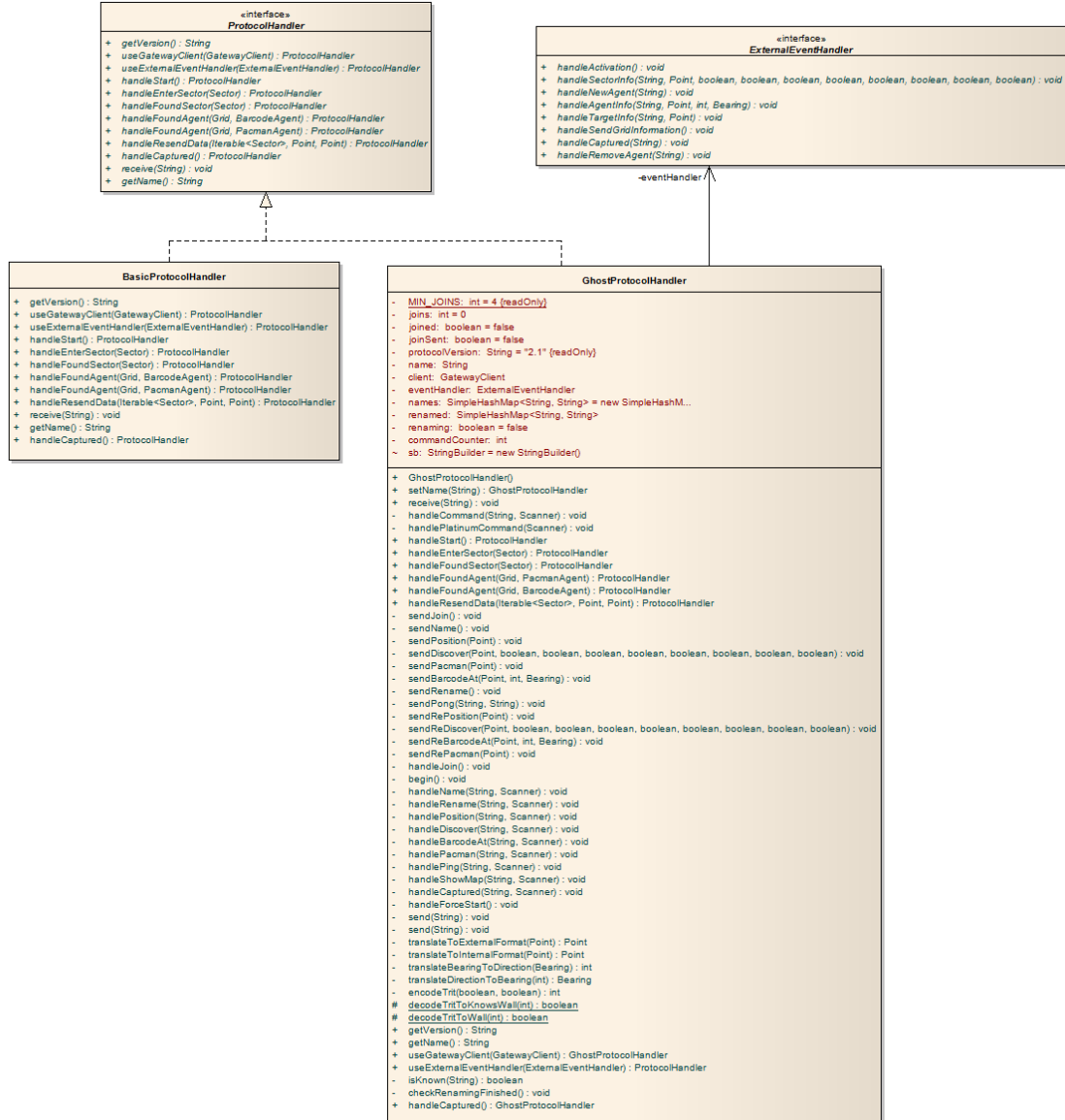
```
astroboy:po-mindstorms xtof$ make rash ARGS=---debug
*** compiling rash...
*** running rash...
--- Setting up debugging MQ using stdout.
--- Starting shell.
rash> load demo.log
MQ: rood POSITION 1 1
sent: rood POSITION 1 1
MQ: rood POSITION 1 2
sent: rood POSITION 1 2
MQ: blauw POSITION 3 2
sent: blauw POSITION 3 2
MQ: blauw DISCOVER 3 2 1 1 0 1
sent: blauw DISCOVER 3 2 1 1 0 1
rash> █
```

Figuur B.6: Met behulp van rash kan een log van commando's verstuurd worden.

Bijlage C

Klasse Diagrammen

De volgende figuren tonen de belangrijkste *packages* uit het project.



Figuur C.2: Protocol



Figuur C.3: Robot

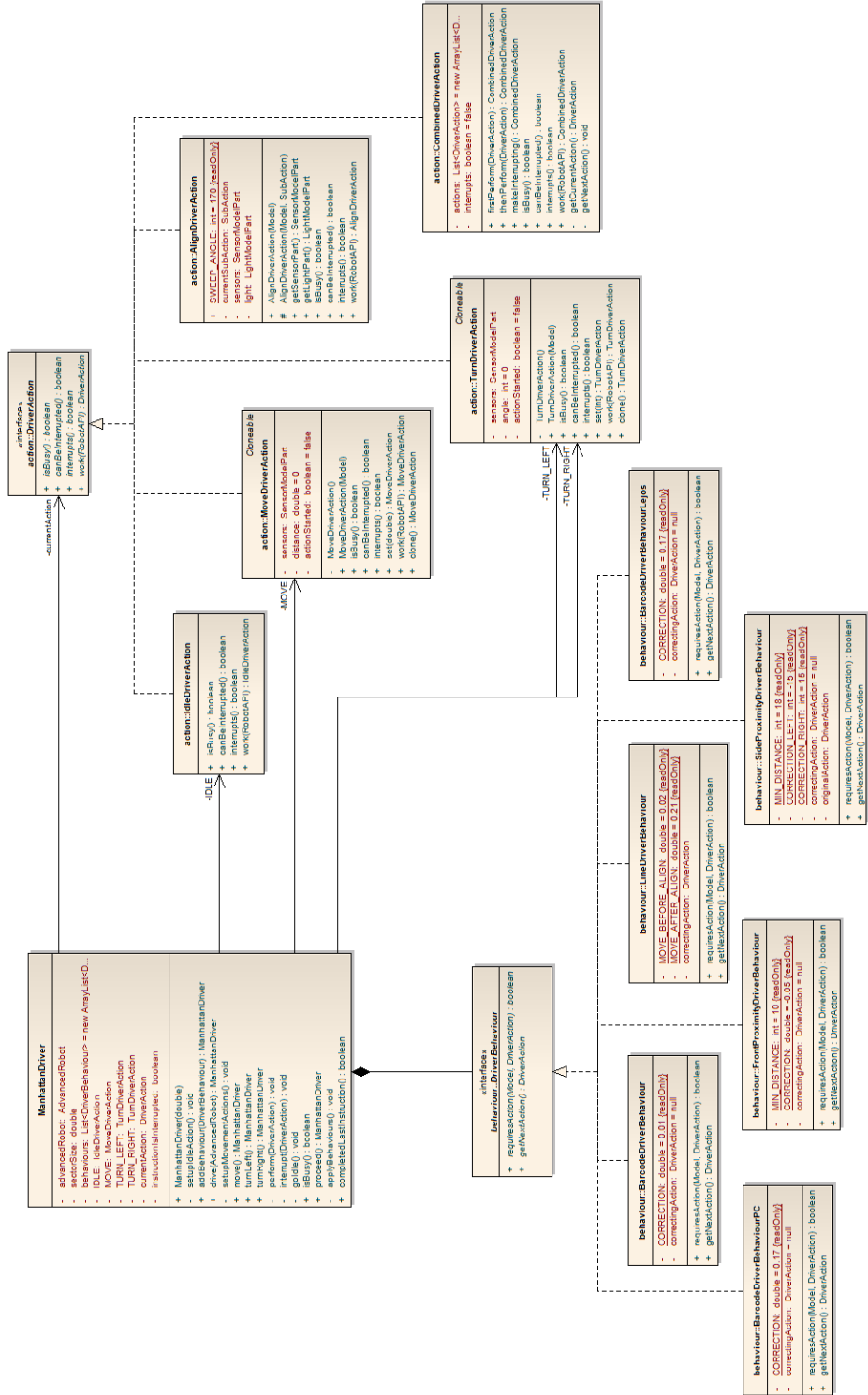
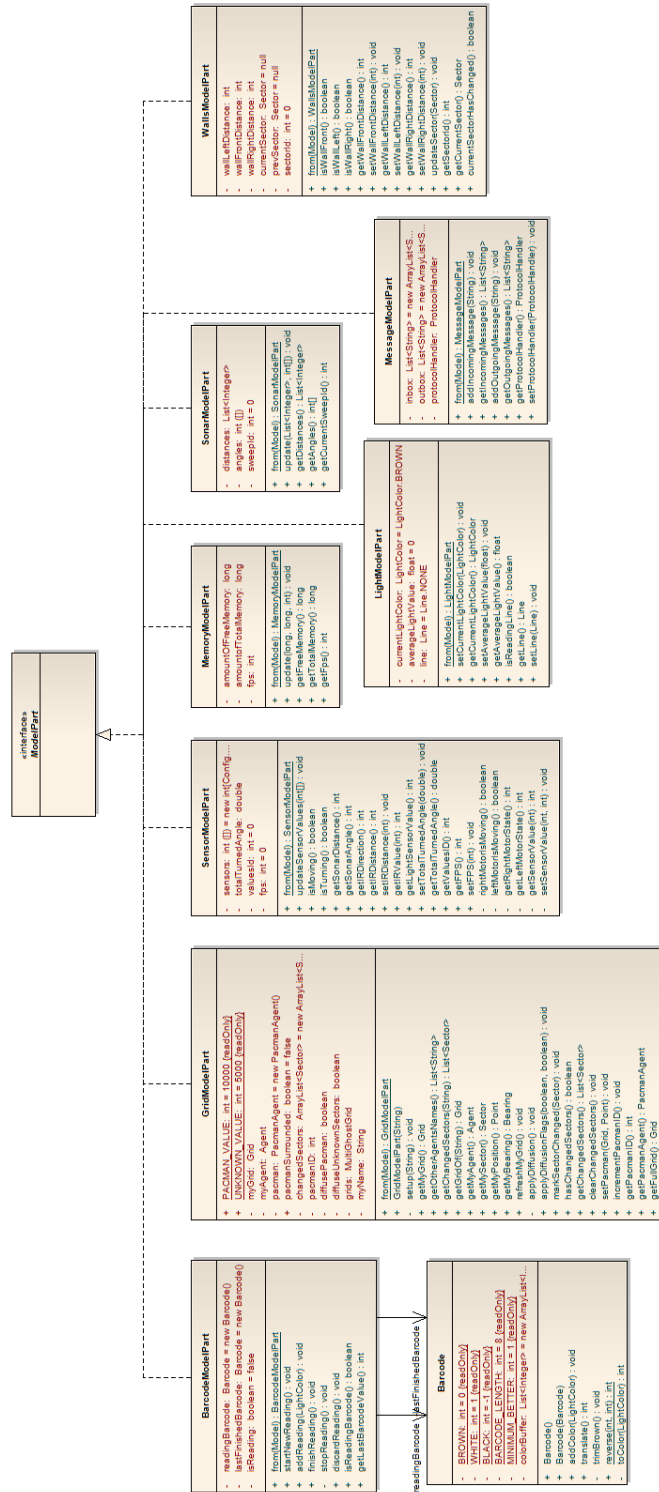


Figure C.4: Driver



Figur C.6: ModelPart

Bijlage D

Planning

Tabel D.1: Overzicht tijdsbesteding

week	Michiel	Florian	Ruben	Thomas	Christophe	Totaal	AVG	Lopend AVG
avg	20,0	18,0	18,1	17,9	20,8		18,95	
sum	260	234	235	233	270	1232		
1	12	9	13	14	16,5	64,5	12,90	12,90
2	16,5	14	14	15	17,5	77	15,40	14,15
3	32,5	15	21	17	29	114,5	22,90	17,07
4	49	24	33	28	32	166	33,20	21,10
5	8	9	11	18	14	60	12,00	19,28
6	26	23	24	36	36	145	29,00	20,90
7	14	8	5	5	20	52	10,40	19,40
8	3	0	0	0	13	16	3,20	17,38
9	0	0	0	0	11	11	2,20	15,69
10	11	24	18	22	2	77	15,40	15,66
11	19	31	19	28	22	119	23,80	16,40
12	50	55	53	38	41	237	47,40	18,98
13	19	22	24	12	16	93	18,60	18,95

Tabel D.2: Michiel tijdsbesteding

Dag	Datum	omschrijving team	omschrijving Michiel	Uren
				260
Mon	13-feb.-2012	P&O sessie, discussie aanpak, research pacman ed,	commissie	5
Wed	15-feb.-2012			5
Fri	17-feb.-2012			2
Mon	20-feb.-2012	P&O sessie	Testen IR, commissie	5
Tue	21-feb.-2012			2
Wed	22-feb.-2012	Testen IR	Testen IR	7,5
Fri	24-feb.-2012	Driving/map voor bouw robot		
Sat	25-feb.-2012	Simulator+behaviour		2
Mon	27-feb.-2012	P&O sessie		8
Tue	28-feb.-2012	verslag		8,5
Wed	29-feb.-2012			7
Thu	1-mrt.-2012		onderzoek naar mogelijke verbeteringen op gebied van rondrijden	7
Fri	2-mrt.-2012			2
Sat	3-mrt.-2012	code bekeken voor verslag		
Sun	4-mrt.-2012	verslag		
Mon	5-mrt.-2012	P&O sessie	commissie, code integration mini-simulator and real si- mulator	5
Tue	6-mrt.-2012	verslag, debugging	lichtsensor herimplementatie	8
Wed	7-mrt.-2012		verslag, ghost protocol	9
Continued on Next Page...				

Tabel D.2 – Continued

Dag	Datum	omschrijving team	omschrijving Michiel	Uren
Thu	8-mrt.-2012		Testen ghost protocol, implementatie RobotBluetooth-hAgent, schrijven scanner implementatie, aanpassen code voor lejos, schrijven ontbrekende code in lejos	9
Fri	9-mrt.-2012	Memory Management	Optimalisatie robot, bespreking memory management, implementatie barcodes in protocol, implementatie grid importing	5
Sun	11-mrt.-2012		refactored unit tests, fixed all build errors, work on grid importing, barcode bugfixing, MQ bugfixing, Dashboard agent implementation	13
Mon	12-mrt.-2012	P&O Sessie	Barcode bugfixing, grid mapping bugfixing, pacman protocol implementatie, reverse mapping implementatie, dashboardagent implementatie	7
Sat	17-mrt.-2012			1
Mon	19-mrt.-2012	P&O Sessie, overlopen resultaten code review en taakverdeling	Vergadering + planning, scheidsrechtercommissie, initial model refactoring	5
Tue	20-mrt.-2012		completed model refactoring, merged minisimulator robot implementation with the actual robot implementation, cleanup of robot project	4
Wed	21-mrt.-2012	Inleveren verslag 2	Added light consistency test, fixed barcode bug, adding barcodes to swinggridview	3
Thu	22-mrt.-2012		Added barcodes to the minisimulator	1
Sat	24-mrt.-2012		implementatie gridmerging	6
Sun	25-mrt.-2012		implementatie gridmerging	7
Mon	26-mrt.-2012	P&O Sessie / Demo 2		5

Continued on Next Page...

Tabel D.2 – Continued

Dag	Datum	omschrijving team	omschrijving Michiel	Uren
Sat	31-mrt.-2012		Unit testen voor grid package	6
Sun	1-apr.-2012		Transformatietesten grid	3
Mon	2-apr.-2012	Paasvakantie	Sector interface	3
Mon	16-apr.-2012	P&O Sessie		5
Sun	22-apr.-2012		Mergen branches + fixen various tests, Implementatie LinkedSector + testen	6
Mon	23-apr.-2012	P&O Sessie	Interface grid + implementatie, compilation errors in test fixen	5
Tue	24-apr.-2012		Implementatie Linkedgrid	2
Wed	25-apr.-2012		Testen Linkedgrid	4
Sun	29-apr.-2012		Implemented Aggregatedgrid + sector, transformedgrid + sector, MultiGhostGrid. Partially tested transformedgrid	8
Mon	30-apr.-2012	P&O Sessie	Fixed all transformedgrid and transformedsector issues + tested. Implemented aggregatedsector test, work on aggregatedgrid test	8
Tue	1/mei/12		Grid optimalisatie	8
Wed	2/mei/12	Inleveren verslag 3	Grid optimalisatie	8
Thu	3/mei/12		Grid optimalisatie	6
Fri	4/mei/12		Grid optimalisatie	8
Sun	6/mei/12		Grid optimalisatie	12
Mon	7/mei/12	Finale Demo	GRID transitief mergen + sector collapse	15
Wed	9/mei/12	Eindverslag		3
Mon	14/mei/12	Finale Presentatie		1

Tabel D.3: Florian tijdsbesteding

Dag	Datum	omschrijving team	omschrijving Florian	Uren
				234
Mon	13-feb.-2012	P&O sessie, discussie aanpak, research pacman ed,	simulator refactoring en nieuwe features	4
Wed	15-feb.-2012		Sector Opdeling	5
Mon	20-feb.-2012	P&O sessie		5
Tue	21-feb.-2012		IR	3
Wed	22-feb.-2012	Testen IR	Testen IR	6
Fri	24-feb.-2012	Driving/map voor bouw robot		
Sat	25-feb.-2012	Simulator+behaviour		
Mon	27-feb.-2012	P&O sessie		5
Tue	28-feb.-2012	verslag	verslag	5
Wed	29-feb.-2012		verslag	5
Thu	1-mrt.-2012		onderzoek naar mogelijke verbeteringen op gebied van rondrijden	
Sat	3-mrt.-2012	code bekeken voor verslag		
Sun	4-mrt.-2012	verslag		
Mon	5-mrt.-2012	P&O sessie	verslag	5
Tue	6-mrt.-2012	verslag, debugging	lichtsensor herimplementatie	5
Wed	7-mrt.-2012		verslag, ghost protocol	7
Thu	8-mrt.-2012		pair programming michiel	5
Fri	9-mrt.-2012	Memory Management		2
Mon	12-mrt.-2012	P&O Sessie		5
Wed	14-mrt.-2012	Review code tree	Bitwiseoperations	4
Continued on Next Page...				

Tabel D.3 – Continued

Dag	Datum	omschrijving team	omschrijving Florian	Uren
Mon	19-mrt.-2012	P&O Sessie, overlopen resultaten code review en taakverdeling	verslag	5
Wed	21-mrt.-2012	Inleveren verslag 2		2
Sat	24-mrt.-2012		verslag rood+colors +dashboardagent+collision	9
Sun	25-mrt.-2012		verslag +collision	7
Mon	26-mrt.-2012	P&O Sessie / Demo 2		8
Mon	16-apr.-2012	P&O Sessie		5
Tue	17-apr.-2012		unit testen	4
Wed	18-apr.-2012		unit testen	5
Thu	19-apr.-2012		unit testen	4
Sat	21-apr.-2012		unit	6
Mon	23-apr.-2012	P&O Sessie	unit	5
Tue	24-apr.-2012		unit	4
Wed	25-apr.-2012		unit	4
Thu	26-apr.-2012		unit	3
Fri	27-apr.-2012		unit	4
Sat	28-apr.-2012		unit	7
Sun	29-apr.-2012		unit	4
Mon	30-apr.-2012	P&O Sessie	verslag	10
Tue	1/mei/12		grid fixes	8
Wed	2/mei/12	Inleveren verslag 3	grid testing	8
Thu	3/mei/12		grid testing	6
Fri	4/mei/12		grid testing	6
Sat	5/mei/12		grid testing	5

Continued on Next Page...

Tabel D.3 – Continued

Dag	Datum	omschrijving team	omschrijving Florian	Uren
Sun	6/mei/12		fixes, unit testen, etc	12
Mon	7/mei/12	Finale Demo	Memory, grid, code shrinking etc	15
Wed	9/mei/12	Eindverslag		6
Mon	14/mei/12	Finale Presentatie		1

Tabel D.4: Ruben tijdsbesteding

Dag	Datum	omschrijving team	omschrijving Ruben	Uren
				235
Mon	13-feb.-2012	P&O sessie, discussie aanpak, research pacman ed,	simulator refactoring en nieuwe features, Projectleider	6
Tue	14-feb.-2012			2
Wed	15-feb.-2012			5
Mon	20-feb.-2012	P&O sessie	Projectleider	5
Tue	21-feb.-2012		IRSeeker	4
Wed	22-feb.-2012	Testen IR	Testen IR	5
Mon	27-feb.-2012	P&O sessie	Projectleider	6
Tue	28-feb.-2012	verslag		7
Wed	29-feb.-2012			3
Thu	1-mrt.-2012		onderzoek naar mogelijke verbeteringen op gebied van rondrijden	5
Mon	5-mrt.-2012	P&O sessie	Projectleider verslag	5

Continued on Next Page...

Tabel D.4 – Continued

Dag	Datum	omschrijving team	omschrijving Ruben	Uren
Tue	6-mrt.-2012	verslag, debugging	barcodes, verslag	6
Wed	7-mrt.-2012		verslag nalezen	7
Thu	8-mrt.-2012		MazeProtocol	4
Fri	9-mrt.-2012	Memory Management	MazeProtocol, IR in simulator, bugfixes	3
Sun	11-mrt.-2012			8
Mon	12-mrt.-2012	P&O Sessie	Testen robot, Framerate verhogen op robot, pacman herkennen	6
Wed	14-mrt.-2012	Review code tree		5
Mon	19-mrt.-2012	P&O Sessie, overlopen resultaten code review en taakverdeling	verslag	5
Tue	20-mrt.-2012		verslag refactoring	3
Wed	21-mrt.-2012	Inleveren verslag 2	refactoring	6
Sat	24-mrt.-2012		barcodegedrag voor robot	5
Sun	25-mrt.-2012		Robot, bluetooth memorytests	5
Mon	26-mrt.-2012	P&O Sessie / Demo 2		5
Mon	16-apr.-2012	P&O Sessie	unit testen map	5
Sat	21-apr.-2012		unit testen simulator	8
Sun	22-apr.-2012		unit testen sensoren	5
Mon	23-apr.-2012	P&O Sessie	unit testen sensoren	6
Tue	24-apr.-2012			2
Sat	28-apr.-2012			6
Sun	29-apr.-2012			5
Mon	30-apr.-2012	P&O Sessie		9
Tue	1/mei/12			9

Continued on Next Page...

Tabel D.4 – Continued

Dag	Datum	omschrijving team	omschrijving Ruben	Uren
Wed	2/mei/12	Inleveren verslag 3		5
Thu	3/mei/12			5
Fri	4/mei/12			5
Sat	5/mei/12			8
Sun	6/mei/12		Robot: barcodes en lijnen	12
Mon	7/mei/12	Finale Demo	Pacman, Memory issues	15
Tue	8/mei/12		verslag	5
Wed	9/mei/12	Eindverslag	verslag	3
Mon	14/mei/12	Finale Presentatie		1

80

Tabel D.5: Thomas tijdsbesteding

Dag	Datum	omschrijving team	omschrijving Thomas	Uren
				233
Mon	13-feb.-2012	P&O sessie, discussie aanpak, research pacman ed,	simulator refactoring en nieuwe features	5
Tue	14-feb.-2012			3
Wed	15-feb.-2012			2
Sat	18-feb.-2012		Sector Opdeling	1
Sun	19-feb.-2012		Sector Opdeling	3
Mon	20-feb.-2012	P&O sessie	Testen IR/opkuisen code	6
Tue	21-feb.-2012		Research/nauwkeurigheid robot	4

Continued on Next Page...

Tabel D.5 – Continued

Dag	Datum	omschrijving team	omschrijving Thomas	Uren
Wed	22-feb.-2012	Testen IR		2
Fri	24-feb.-2012	Driving/map voor bouw robot		1
Sat	25-feb.-2012	Simulator+behaviour		2
Mon	27-feb.-2012	P&O sessie	Sector	6
Tue	28-feb.-2012	verslag		4
Wed	29-feb.-2012			1
Thu	1-mrt.-2012		onderzoek naar mogelijke verbeteringen op gebied van rondrijden	2
Sat	3-mrt.-2012	code bekeken voor verslag	code bekeken voor verslag	2
Sun	4-mrt.-2012	verslag	verslag	2
Mon	5-mrt.-2012	P&O sessie	verslag	6
Tue	6-mrt.-2012	verslag, debugging	barcodes, verslag	5
Wed	7-mrt.-2012		verslag, protocol,...	8
Thu	8-mrt.-2012			3
Fri	9-mrt.-2012	Memory Management	Implementatie, sensors in simulator, IR	3
Sun	11-mrt.-2012			3
Mon	12-mrt.-2012	P&O Sessie	Testen robot, bugfixes,...	5
Tue	13-mrt.-2012		code review: action, bluetooth, driver,...	3
Wed	14-mrt.-2012	Review code tree	code review: map, modelprocessors, sensors	2
Thu	15-mrt.-2012		code review: vizualization + start simulator	4
Fri	16-mrt.-2012		code review: simulator	2
Sat	17-mrt.-2012		code review: simulator	2
Mon	19-mrt.-2012	P&O Sessie, overlopen resultaten code review en taakverdeling	verslag	5

Continued on Next Page...

Tabel D.5 – Continued

Dag	Datum	omschrijving team	omschrijving Thomas	Uren
Tue	20-mrt.-2012		verslag	4
Wed	21-mrt.-2012	Inleveren verslag 2	verslag	3
Thu	22-mrt.-2012		User interface	3
Fri	23-mrt.-2012		User interface	3
Sat	24-mrt.-2012			12
Sun	25-mrt.-2012		Testing, bug fixing	6
Mon	26-mrt.-2012	P&O Sessie / Demo 2	New Barcode representation, merged classes into this single object.	5
Mon	16-apr.-2012	P&O Sessie	Barcode	5
Tue	17-apr.-2012		Barcode	3
Wed	18-apr.-2012			3
Thu	19-apr.-2012			4
Fri	20-apr.-2012			2
Sat	21-apr.-2012			2
Sun	22-apr.-2012			3
Mon	23-apr.-2012	P&O Sessie	Unit testen, Start working on our own command system.	5
Tue	24-apr.-2012		Commands and MD5	2
Wed	25-apr.-2012		Commands and MD5, unit testing	3
Thu	26-apr.-2012		Unit Testing	4
Fri	27-apr.-2012		AdminTool	3
Sat	28-apr.-2012		AdminTool/unit testing	6
Sun	29-apr.-2012		Code cleanup	5
Mon	30-apr.-2012	P&O Sessie	Protocol, Reporter, Unit Testen, Verslag	8
Tue	1/mei/12		Verslag	11

Continued on Next Page...

Tabel D.5 – Continued

Dag	Datum	omschrijving team	omschrijving Thomas	Uren
Wed	2/mei/12	Inleveren verslag 3	Verslag	
Fri	4/mei/12		Protocol, Reporter, Unit Testen	4
Sat	5/mei/12		Reporter, Unit Testen, Bug fixing	6
Sun	6/mei/12		Protocol, Reporter, Unit Testen	9
Mon	7/mei/12	Finale Demo	Protocol, bug fixes, testen	7
Tue	8/mei/12		Verslag	2
Wed	9/mei/12	Eindverslag	Verslag/github terug werkend krijgen	2
Mon	14/mei/12	Finale Presentatie		1

83

Tabel D.6: Christophe tijdsbesteding

Dag	Datum	omschrijving team	omschrijving Christophe	Uren
				277
Mon	13-feb.-2012	P&O sessie, discussie aanpak, research pacman ed,		3
Tue	14-feb.-2012		RabbitMQ research	3,5
Wed	15-feb.-2012		AntiObjects	5
Thu	16-feb.-2012		AntiObjects	3,5
Fri	17-feb.-2012		AntiObjects, RabbitMQ	0,5
Sat	18-feb.-2012			1
Mon	20-feb.-2012	P&O sessie	AntiObjects	4,5
Tue	21-feb.-2012		AntiObjects, verkenningstrategie uitwerken	3

Continued on Next Page...

Tabel D.6 – Continued

Dag	Datum	omschrijving team	omschrijving Christophe	Uren
Wed	22-feb.-2012	Testen IR	verkenningstrategie uitwerken	5
Thu	23-feb.-2012		DFS discovery	5
Mon	27-feb.-2012	P&O sessie	CD discovery	6
Tue	28-feb.-2012	verslag	discovery/hunt	5,5
Wed	29-feb.-2012		integratie disc/hunt/comm	3
Fri	2-mrt.-2012		mini-simulator	3,5
Sat	3-mrt.-2012	code bekeken voor verslag	mini-simulator	6
Sun	4-mrt.-2012	verslag	mini-simulator	5
Mon	5-mrt.-2012	P&O sessie	agents,logging,,grids,...	6,5
Tue	6-mrt.-2012	verslag, debugging	dashboard, verslag	6
Wed	7-mrt.-2012		AggregatedGrid, verslag	4
Thu	8-mrt.-2012		AggregatedGrid	2
Fri	9-mrt.-2012	Memory Management	AggregatedGrid	6
Sat	10-mrt.-2012		ServiceAgent	1
Sun	11-mrt.-2012		ServiceAgent, UI Server, Dashboard	6,5
Mon	12-mrt.-2012	P&O Sessie	MiniSimulator,ServiceAgent, UI Server, Dashboard, me- memory usage improvements	10
Tue	13-mrt.-2012		review code tree	1
Wed	14-mrt.-2012	Review code tree		3
Mon	19-mrt.-2012	P&O Sessie, overlopen resultaten code review en taakverdeling		6
Tue	20-mrt.-2012		verslag (performantie,review.merge)	4
Wed	21-mrt.-2012	Inleveren verslag 2	verslag (merge reviews)	2
Continued on Next Page...				

Tabel D.6 – Continued

Dag	Datum	omschrijving team	omschrijving Christophe	Uren
Thu	22-mrt.-2012		dashboard: multi-robot, local-file source / Agent -i Gateway + UpdatesJSAppender	6
Fri	23-mrt.-2012		SimulationRunner, Gateway(Client)	7
Sat	24-mrt.-2012		SimulatedGatewayClient	4
Sun	25-mrt.-2012		Local Dashboard co	7
Mon	26-mrt.-2012	P&O Sessie / Demo 2	Dashboard, DashboardReporter, GhostProtocol,	10
Tue	27-mrt.-2012		model documentatie + refactoring strategie + unit tests	3
Wed	28-mrt.-2012		unit test voorbeelden	4
Fri	30-mrt.-2012		unit tests	3
Thu	5-apr.-2012	Paasvakantie	unit tests	6
Fri	6-apr.-2012	Paasvakantie	unit tests	3
Sat	7-apr.-2012	Paasvakantie	unit tests	3
Sun	8-apr.-2012	Paasvakantie	unit tests	1
Tue	10-apr.-2012	Paasvakantie	unit tests	2
Wed	11-apr.-2012	Paasvakantie	unit tests	1
Thu	12-apr.-2012	Paasvakantie	unit tests	4
Fri	13-apr.-2012	Paasvakantie	unit tests	4
Mon	16-apr.-2012	P&O Sessie		2
Mon	23-apr.-2012	P&O Sessie		3
Tue	24-apr.-2012		fixed off-by-one bug in CircularQueue	1
Wed	25-apr.-2012		ModelParts -Processors	4
Thu	26-apr.-2012		ModelParts -Processors	3
Fri	27-apr.-2012		ModelParts -Processors	8
Sun	29-apr.-2012		clean up + rash	3

Continued on Next Page...

Tabel D.6 – Continued

Dag	Datum	omschrijving team	omschrijving Christophe	Uren
Mon	30-apr.-2012	P&O Sessie		8
Tue	1/mei/12		GridModelPart + Diffusion	5
Wed	2/mei/12	Inleveren verslag 3	verslag mergen + fixen	6
Thu	3/mei/12		top-level applicaties samenstellen	3
Fri	4/mei/12		fullRobotTest aan de praat krijgen	3
Sat	5/mei/12		poging geheel af te ronden	6
Sun	6/mei/12		loose ends opkuisen	10
Mon	7/mei/12	Finale Demo	more loose ends opkuisen	7
Wed	9/mei/12	Eindverslag	test coverage finalize	4
Thu	10/mei/12		presentatie	1
Fri	11/mei/12		presentatie	1
Sat	12/mei/12		presentatie	1
Sun	13/mei/12		presentatie	1
Mon	14/mei/12	Finale Presentatie		1

Bibliografie

- [1] Alexander Repenning. Collaborative diffusion: programming antiobjects. In *OOPSLA Companion*, pages 574–585, 2006.