Robocode

GROEP C1

Jesse Sterkenburgh

Rick Holtman

Jurgen van de Wardt

Jelle Buitenhuis

INHOUD

INLEIDING

Tijdens kwartiel 1.3 hebben wij een project gekregen van Saxion hogescholen. In dit project waren de doelstellingen de student samen te leren werken in een project met onder andere versiebeheer. Hierbij wordt er gebruik gemaakt van de open source software Robocode. De studenten dienen hierbij hun eigen team van robots te ontwikkelen met de door hun bedachte strategie. Elke week worden alle teams van alle projectgroepen tegen elkaar op gezet in het speelveld om de progressie van alle groepen te kunnen volgen. Hierbij kunnen de studenten zien hoe zij hun strategie moeten aanpassen om de robots proberen te laten winnen. Tijdens dit project leren wij ook werken met versiebeheersysteem git. Hierbij kunnen de studenten afzonderlijk aan hetzelfde project werken. Aan het einde van het project wordt er bij Syntaxis op de vierde verdieping een feest georganiseerd genaamd “robocomp”. Dit is de definitieve en bepaalde wedstrijd waarbij ieder projectgroepje gaat zien wie deze wedstrijd gaat winnen.

ROBOCODE

Robocode is een programmeerspel waar het doel is een robot te programmeren die andere robots in de arena zal verslaan. De speler is de programmeur van de robot die geen directe invloed heeft op het spel. De programmeur dient een strategie te schrijven voor de robot waarin alle handelingen op basis van gebeurtenissen in het speelveld worden beschreven en uitgevoerd. De naam robocode is een verkorting van “Robot Code”. Het spel is ontwikkeld om te leren ontwikkelen in Java. Robots zijn geschreven in Java. Robocode kan draaien op elk besturingssysteem dat ondersteund wordt door het Java platform dat draait op alle meest voorkomende besturingssystemen zoals Windows, macOS en Linux. Robocode wedstrijden worden gespeeld op een speelveld waar kleine 6-wielige robots tegen elkaar vechten tot er 1 over blijft. Robocode bevat geen bloed, mensen, politiek of andere grafische beelden die als schokkend ervaren kunnen worden. De enige uitzondering hierop zijn explosies. Wanneer men deze explosies beledigend vindt, kan hij/zij/het deze zelf uitschakelen.

Robocode is ontwikkeld door Matthew A. Nelson, als persoonlijk project. In 2000 werd het een professioneel project toen het werd bij IBM werd ondergebracht in de vorm van een AlphaWorks download, in juli 2001. In 2005 werd Robocode een open source project doordat het op SourceForge werd gepubliceerd vanaf versie 1.0.7. Vanaf dit moment heeft de ontwikkeling van Robocode vrijwel stil gestaan. Wel heeft de community in de tussentijd hun eigen versies van Robocode ontwikkeld om onder andere bugs te fixen.

TECHNISCHE DOCUMENTATIE

BESCHRIJVING VAN DE STRATEGIEËN:

Strategie 1:

Elke robot binnen ons team communiceert naar elke het team altijd de positie van de vijandelijke robot die ze scannen. Onze robot heeft 1 target die die volgt (vijandelijke robot die het meest dichtbij is). Zodra 1 van de vijandelijke robots dood is ga je met een andere robot van het team tegen een enkele vijandelijke vuren totdat er gewonnen is.

VOORDELEN: elke enemy wordt getarget, random bewegingen.

NADELEN: weinig damage, ineffecient bij uitvallen robot.

Strategie 2:

Er is 1 rambot binnen het team (willekeurig bepaald) die zal proberen alle andere vijandelijke robots af te leiden door erg dichtbij te rijden. Ook zal de ramrobot op onverwachte momenten ook de vijandelijke robot rammen. De schietrobots binnen het team zullen willekeurig op hun plek bewegen en de andere robots beschieten.

VOORDELEN: afleiding, rambonus

NADELEN: geen spreiding

Strategie 3:

Een willekeurige robot van het team kiest de vijand die het dichtstbij is. Deze robot communiceert naar de overige robots van het team welke vijandelijke robot de robot binnen het team gekozen heeft. Van de robots die nog over zijn zal de robot die weer het meest dichtbij staat de ram robot worden. Hierdoor ontstaan 2 paartjes. De ramrobot zal nooit in de vuurlijn van de schietrobot mogen zitten en de schietrobot past zich hierop aan door bijvoorbeeld altijd in de “denkbeeldige cirkel” te rijden. Ze zullen wel zoveel mogelijk tegenover elkaar staan en rondom de vijand cirkelen op een random afstand die varieert zodat ze niet beide geraakt kunnen worden. Als de ram robot wegvalt zal het gedrag van de schietrobot niet veranderen. Als de schietrobot wegvalt zal de ram robot deze taak overnemen. Als de vijand dood is zal de vijand die het dichtst bij de ramrobot is de nieuwe target worden als deze nog niet door een andere teamgenoot geclaimd is. De teams kunnen dus nooit hetzelfde target hebben.

VOORDELEN: Rambonus, geen friendly fire, teams onafhankelijk van elkaar

NADELEN: Beweging vallen te voorspellen

AFWEGINGEN

	Voordelen	Nadelen
Strategie 1	- Elke enemy wordt getarget - Random bewegingen	- Weinig damage - Inefficient bij uitvallen robot
Strategie 2	- Afleiding - Rambonus	- Geen spreiding
Strategie 3	- Rambonus - Geen Friendly Fire - Teams onafhankelijk van elkaar	- Bewegingen zijn te voorspellen

GEKOZEN STRATEGIE

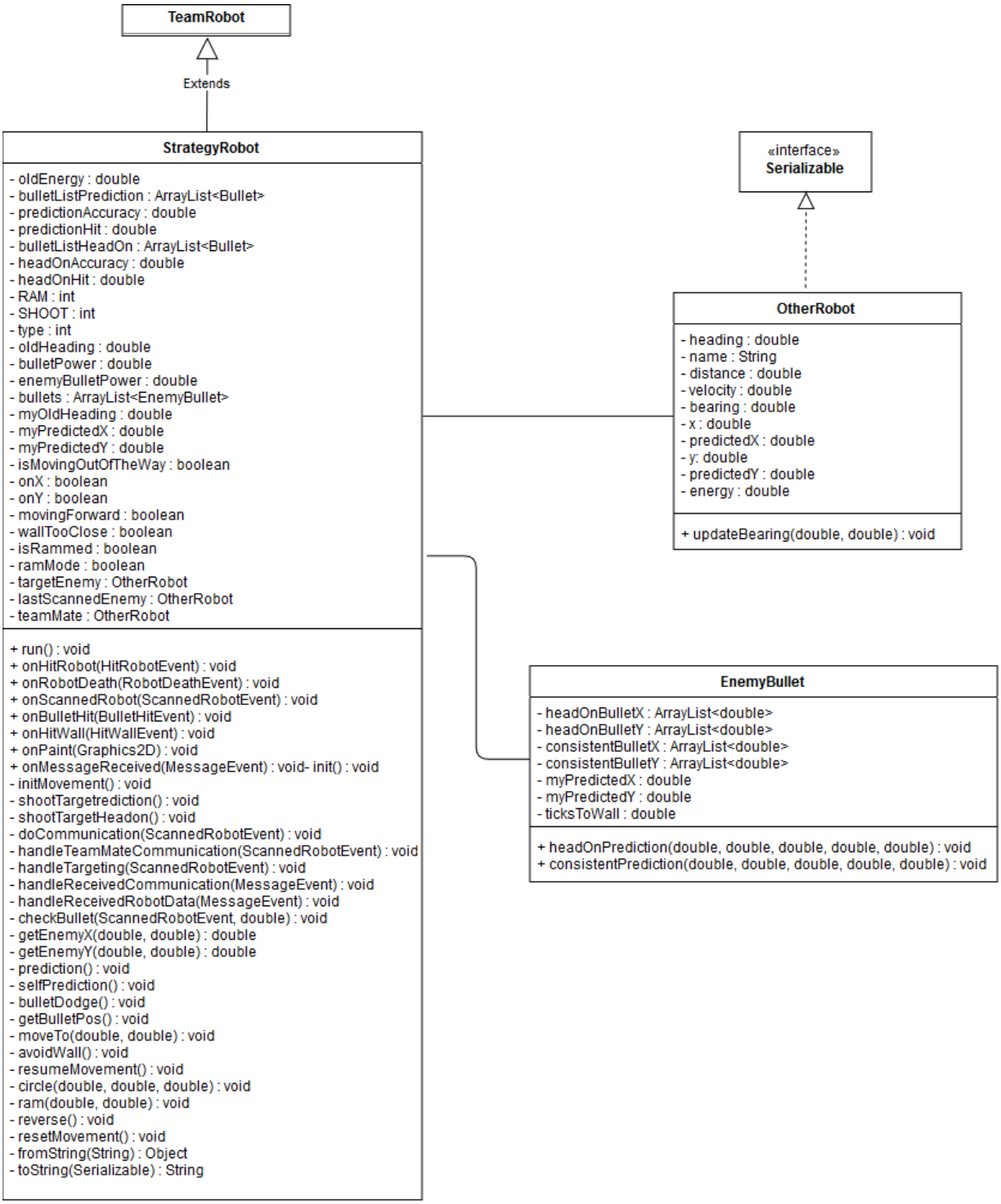
Wij hebben strategie 3 gekozen omdat deze volgens onze beredenering de beste strategie is om te winnen van onze tegenstanders. De reden hierachter is dat wij er van overtuigd zijn dat we zo de beste kans hebben om het vermijden van de strategieën van de andere teams omdat ze waarschijnlijk maar een robot tegen een andere robot zullen laten vechten. Echter deze kan altijd nog aangepast worden op basis van de resultaten van de tussentijdse gevechten.

HOE DE ROBOTS GAAN COMMUNICEREN?

De robots zullen als eerste met elkaar communiceren welke robot het dichtste bij wie staat, daarna zullen ze aan elkaar bekend maken welke robots een team gaan vormen. Nadat de teams zijn gemaakt zullen de 2 robots die samen zijn communiceren over elkaars positie zodat de ‘ram robot' nooit in de vuurlijn van de andere robot kan komen om zo friendly fire te voorkomen. Als de schietrobot neergeschoten wordt dan zal de ramrobot deze taak op zich nemen. Zodra de vijand dood is zal het team zich op de robot richten die het dichtstbij is.

KLASSENDIAGRAM

Hieronder staat het klassendiagram dat opgesteld is aan de hand van de uiteindelijke code die we hebben gemaakt. Onder dit diagram staat een beschrijving per klasse. In het na dit klassendiagram zal er een beschrijving zijn van de belangrijke methodes.



BESCHRIJVING CLASSES

STRATEGYROBOT

De StrategyRobot is onze basis robot met alle inhoudelijke acties die een instantie van onze robot kan uitvoeren. Hieronder is een lijst te vinden van de belangrijke methodes.

Methodenaam	Beschrijving
onHitRobot	Rijdt terug als de robot een vijandelijke robot raakt
OnRobotDeath	Reset de teamgenoot en het doelwit als deze afgeschoten wordt
OnScannedRobot	Controleert of er acties moeten worden uitgevoerd bij het scannen van een robot, communiceert met teamgenoten voor targeting en pairing met een teamgenoot
onBulletHit	Berekend de hits die succesvol zijn afgevuurd
onHitWall	Wordt uitgevoerd als de robot een muur raakt en rijdt vervolgens terug
onMessageRecieved	Handelt het binnengekomen bericht af

OTHERROBOT

In OtherRobot wordt alle data opgeslagen van een andere gescande robot. Denk aan X, Y, heading, velocity etc. Naast de standaard getters en setters is er 1 belangrijke methode.

Methodenaam	Beschrijving
updateBearing	Berekend de bearing vanaf het scanpunt van de scanner robot en aan de hand van de x en y coördinaten van deze gescande robot

ENEMYBULLET

De EnemyBullet klasse wordt gebruikt om informatie op te slaan van de afgevuurde kogels van de tegenstander. De belangrijkste informatie hier zijn alle X en Y posities die de kogel aan kan nemen. Er worden twee kogels berekend. Voor de eerste kogel wordt de aanname gedaan dat die kogel op onze huidige positie wordt afgevuurd. Voor de tweede kogel wordt onze eigen voorspelling omgedraaid, deze wordt gedaan vanuit het perspectief van de tegenstander.

Methodenaam	Beschrijving
headOnPrediction	Maakt een voorspelling door de huidige x en y coördinaten te pakken en de hoeken hier van. Met deze voorspelling kan de rijrichting van de robot worden bepaald.
consistentPrediction	Berekend onze eigen voorspelling omgedraaid, deze wordt gedaan vanuit het perspectief van de tegenstander om zo een nauwkeurigere voorspelling te doen.

PROJECT DOCUMENTATIE

GROEPSAFSPRAKEN

Hierin zijn de groepsafspraken vermeld waar wij ons met ons groepje aan gaan houden.

Indien een student afwezig is bij de lessen dient hij hier vroegtijdig melding van te maken. Eveneens wordt er nog steeds verwacht dat hij zijn werk voor de desbetreffende sprint tijdig ingeleverd heeft.

Het ingeleverde werk dient van hoge kwaliteit te zijn en veelvuldig getest te worden op fouten. Aan het einde van elke sprint zal er een regressietest plaatsvinden om de kwaliteit van het product te waarborgen. Deze regressietest zal inhouden dat we kort voor elk inlevermoment, dus bijvoorbeeld vrijdag, we de losse functionaliteit zullen gaan testen van de onderdelen. Dit zou bijvoorbeeld het testen van een bepaalde methode zijn die zorgt dat we kogels ontwijken. Door deze telkens te testen kunnen we waarborgen dat deze functionaliteit niet kapot is gegaan sinds het vorige inlevermoment. Door deze testen kunnen we fouten vroegtijdig opvangen en kunne we zo het project met minder opstoppingen uitvoeren. Ook zal dit voorkomen dat we steeds verder werken op code die niet helemaal goed functioneert. Dit zal anders resulteren in een code dat niet te hanteren is en waar veel tijd in gaat zitten als men dit wil aanpassen om het werkend te maken.

Verder houden we rekening met elkaars normen en waarden en verwachten we dat iedereen zich voldoende inzet bij dit project.

Mocht het voorkomen dat een groepslid niet aanwezig is en hier ook geen melding van heeft gemaakt, dan zullen we eerst een waarschuwing geven. Mocht het daarna nog vaker voorkomen dan zullen we de docent erbij betrekken en zullen er eventuele consequenties volgen.

DOEL SPRINT 1

Het doel van deze sprint is om de basisfuncties van de gekozen strategie te implementeren. Deze strategie zal de gekozen strategie zijn die te vinden is onder het kopje ***Gekozen strategie***. Voor de uitwerking hiervan hebben wij het in de volgende kleine onderdelen gesplitst:

- Maken van de teams
- Bewegen om het doelwit
- Schieten van de schietbot
- Rammen door de rambot

Tijdens deze sprint zullen wij tevens de opgenomen gevechten met de andere teams bekijken en hieruit proberen om een lijst te maken met dingen die fout gingen. Vervolgens kunnen wij hier de code op aanpassen zoals het implementeren van geavanceerdere beweging door onvoorspelbaar te bewegen. Ook zullen we de technische documentatie verder uitbreiden met een class diagram. Normaliter zouden we al de klassendiagrammen bedacht en gemaakt hebben en hier de code op hebben gemaakt maar we verwachten met dit project erg afhankelijk te zijn van de resultaten van de opgenomen gevechten waardoor we nog erg kunnen verschillen in onze code.

Aan het einde van deze sprint zullen wij onze strategie gaan beoordelen op basis van de effectiviteit die zal blijken uit de opgenomen gevechten. Uit deze beoordeling zal mogelijk een aanpassing van de strategie zelf voortkomen of aanpassingen in de implementatie van de gekozen strategie. Wat je hierbij voor kan stellen is dat we niet meer wisselen van robot type, zoals ramrobot of schietrobot maar we het type niet meer aanpassen nadat het

aangemaakt is omdat deze verandering bijvoorbeeld er voor kan zorgen dat de robot niet meer functioneert door een bug die te lang zou duren om op te lossen.

REFLECTIE SPRINT 1

In de eerste sprint hebben wij de basiselementen die we beschreven hadden kunnen maken en implementeren. Deze zijn als volgt:

- Maken van de teams
- Bewegen om het doelwit
- Schieten van de schietbot
- Rammen door de rambot

Als gevolg hiervan hebben wij besloten om de strategie iets aan te passen, vanaf nu zullen wij de rambot niet constant laten rammen maar zal deze zich als een schietbot gedragen en af en toe rammen voor de bonuspunten. Verder zijn we erachter gekomen dat het samenvoegen van alle individuele componenten en methodes moeilijker is dan gedacht. De reden hiervoor is dat iedereen ook zijn eigen implementatie heeft gespreven in de “OnRobotScanned”, en je dit dus op precies dezelfde wijze moet samenvoegen. Mocht je hiermee iets afwijken dan zou direct de hele robot niet meer werken. De oorzaak hiervan is deels door het niet erg nauw hanteren van het klassendiagram waardoor iedereen een hele hoop methodes had en instantievariabelen die lastig waren te combineren. Hierdoor hebben we het helaas nog niet voor elkaar gekregen om de strategie volledig te kunnen optimaliseren dit is dan ook ons doel voor sprint 2.

De samenwerking is goed verlopen en iedereen heeft netjes het afgesproken deel gemaakt en toegevoegd aan git, verder heeft Jesse de rest van het groepje goed geholpen met git waardoor dit nog super werkt. Ook heeft Jesse zich samen met Jurgen beziggehouden met het bedenken van het klassendiagram. Jelle is vooral bezig geweest met het implementeren van de schietfunctionaliteiten en het ontwijken van de kogels. Rick heeft zich beziggehouden met de implementatie van de bewegingsmethodes, zoals het willekeurig bewegen en het voorkomen dat een robot tegen de muur aan rijdt. Jurgen is als laatst bezig geweest met het samenvoegen van alle code en om te zorgen dat de basis werkte. Dit was zoals net beschreven niet helemaal goed gelukt omdat we onderschat hadden hoe veel werk het was.

DOEL SPRINT 2

Het doel van deze tweede sprint is om de strategie volledig te optimaliseren zodat deze klaar is voor de eindstrijd. Met optimaliseren hebben wij het over de volgende onderdelen:

- Communicatie tussen de methodes verbeteren
- Verwijderen van onnodige code
- De strategie aanpassen om de prestatie van ons team te verbeteren

Met communicatie tussen de methodes verbeteren bedoelen wij het beter verdelen van de verantwoordelijkheden. Op dit moment zijn er nog te veel methodes die bepaalde instantie variabelen aanpassen terwijl deze dan ook als parameters meegegeven kunnen worden. Als we dit niet aan zouden passen dan hebben we bijna geen zicht meer op de huidige staat van de variabelen omdat deze door de methodes telkens veranderen.

Verwijderen van onnodige code is nodig omdat we nu nog veel testcode in het project hebben staan, zoals de robots klassen die aangemaakt zijn om een bepaalde functionaliteit te testen in een git branch.

Verder zullen wij ook de resultaten van de opnames van de gevechten weer goed in de gaten houden en eventueel weer aanpassingen doen in de strategie.

REFLECTIE SPRINT 2

De tweede sprint hebben wij vooral gebruikt om onze huidige robots te verbeteren aan de hand van de resultaten van de verschillende robot battles. Zo hebben we bijvoorbeeld het volgende gedaan:

- Bugs opgelost binnen onze movement code
- Functionaliteit van het ontwijken van kogels laten samenwerken met de movement code
- Onnodige testcode verwijderd
- JavaDoc toegevoegd aan de code

Ondanks dat we misschien de eerste sprint de hoeveelheid werk van het samenvoegen hadden onderschat hebben we dit in deze sprint goed opgelost en alle code goed kunnen samenvoegen.

Wat beter had gekund is misschien de verdelen van de taken. Zo waren de verschillende functionaliteiten die waren opgedeeld (*movement*, *shooting*, *communication* en het *samenvoegen* van deze onderdelen) niet heel erg goed verdeeld. Zo moest er bijvoorbeeld voor de *movement* en *shooting* onderdelen aan het begin en na het samenvoegen veel gebeuren en hadden de andere delen niet heel veel te doen. Nadat er een basis was met deze code kon er, bijvoorbeeld pas aan de *communication* gewerkt worden. Vervolgens kon als laatste pas het *samenvoegen* beginnen waardoor de drukte voor iedereen op een ander punt lang in de sprints.

Wat we hierdoor geleerd hebben is dat er misschien iets meer nagedacht kan worden hoe iedereen gelijkere taken krijgt. Hierdoor kan iedereen even veel werk doen en is iedereen ook meer betrokken waardoor er ook meer kennis is van de huidige code.

In dit project was het geval dat er bij het samenvoegen van de code te veel nagedacht moest worden wat de verschillende methodes deden en waar bepaalde statements goed voor waren. Dit, in combinatie met het klassendiagram dat niet geheel aanwezig was in het uitwerken, maakte het erg verwarrend om de code te zien en de juiste code op de juiste plek te plaatsen.

REFLECTIE PROJECT

Voor de reflectie van dit project hebben wij er voor gekozen om een gezamenlijk verhaal te maken en niet per persoon een apart verhaal te maken. Dit omdat wij gezamenlijk vinden dat het project gewoon goed is verlopen en op de kleine dingen zoals dat de taken niet helemaal goed verdeeld waren niet veel aan hebben te merken. Hier is dan ook niet echt iets aan op te merken omdat dit meer een probleem van de hele groep was. Wel hebben we gezamenlijk het verhaal gemaakt en de feedback aan elkaar gegeven.

Jurgen is voornamelijk bezig geweest met de verslagen en het mergen, waardoor hij weinig tijd voor het programmeren overhield. Wel heeft hij de overige projectleden geholpen wanneer ze tegen problemen aan liepen met hun code. Aan het begin van het project is hij iets minder actief geweest omdat hij vooral bezig was met het samenvoegen, wat pas kon gebeuren net voordat we code in moesten leveren op robocomp.

Rick heeft zijn goed zijn werk gedaan, maar er is altijd ruimte voor groei met betrekking tot zijn programmeer technische vaardigheden. Echter is dat niet heel gek, hij heeft voor de opleiding nog helemaal niet heeft geprogrammeerd. Verder was het een fijne samenwerking en heeft het geen conflicten opgeleverd in het project.

Jelle mag iets minder hard werken binnen het project. Doordat het werk werd verdeeld waren er bepaalde afhankelijkheden in elkaar code. Je wachtte hier niet op en dit resulteerde in duplicaten in de code. Dit was dan uiteindelijk ten koste van het project. Het samenvoegen was nu bijvoorbeeld heel erg lastig, omdat er zo veel methodes waren die aangeroepen werden en die telkens weer andere variabelen ging aanpassen waardoor het dus onoverzichtelijk werd en de code eerst opgeschoond moest worden om het te laten samenwerken. Werkhouding is verder goed en je weet zeker dat hij zijn taken doet. Soms kun je beter een stapje rustiger aan doen. Zo heb je uiteindelijk ook 5000 regels gecommit waarbij de rest rond de 2500 ligt. Verder heeft hij top werk geleverd wat uiteindelijk ook een grote toevoeging is geweest in onze code.

Jesse was ook een fijn persoon om mee samen te werken in het project. Jesse had veel kennis van git, zonder hem had de groep daar een stuk meer tijd in moeten steken, deze kennis liet zich onder andere zien in het maken van release tags en het branchen. Ook van de technische aspecten had hij genoeg kennis. Ook was hij erg actief binnen de groepsapp die we hadden voor overleg en was altijd aanwezig om vragen te beantwoorden. Een verbeterpuntje is dat de code die hij aanleverde pas laat in het project kwam, de planning hierin kan nog verbeterd worden.

Als we uiteindelijk terugkijken op het project kunnen we zeggen dat heel veel goed is gegaan, op een paar kleine dingetjes na. De samenwerking was goed, we hebben Scrum grotendeels volgens de regels gehanteerd, op bijvoorbeeld de stand-ups na. Het was een fijne samenwerking en niemand had echt ergernissen binnen dit project.

BRONNENLIJST (APA)

- Robocode - RoboWiki. (z.d.). Geraadpleegd op 13 april 2019, van http://robowiki.net/wiki/Robocode