Nieuwe gameserver voor project 2.3
Software Architectuur Document

Hanzehogeschool Groningen **Versie 0.2**

Documenthistorie

Datum	Versie	Beschrijving	Auteur
10-11-2020	0.1	Initiële versie	Gertjan Haan
			Remi Reuvekamp
			Timo Strating
			Hilko Janssen
12-11-2020	0.2	Feedback Lech Bialek verwerkt	

Distributie

Naam	0.1	0.2												
Bas Heijne		X												
Lech Bialek	X	X												

Accordering document

Namens Hanzehogeschool Groningen,

Bas Heijne,
Lech Bialek

--- Handtekening ---

Namens Projectgroep 16,

Timo Strating,
Hilko Janssen,
Remi Reuvekamp,
Gertjan Haan

--- Handtekening ---

Inhoudsopgave

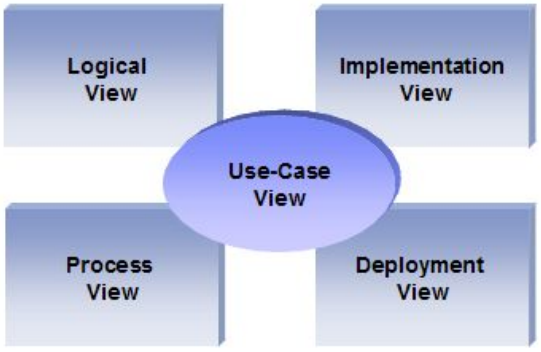
1. Inleiding

Het doel van dit Software Architectuur Document, in het vervolg genoemd SAD, zal worden besproken. Hierna komen de referenties aan bod waar dit document naar verwijst en ten slotte wordt de indeling van dit SAD besproken.

1.1 Doel van dit document

Het SAD bevat een uitgebreide architecturele kijk op het systeem voor de 'Nieuwe gameserver voor project 2.3', ontwikkeld voor Hanzehogeschool Groningen. Het beschrijft een aantal verschillende architecturele views van het systeem om zo verschillende aspecten van het systeem te belichten.

Dit document beschrijft de verschillende RUP views op de software architectuur volgens het 4+1 view model.



Het 4+1 view model stelt de verschillende belanghebbenden in staat om vanuit hun eigen perspectief de invloed van de gekozen architectuur te bepalen. De Process View is, afwijkend van het RUP-op-maat template, ondergebracht in een los hoofdstuk.

1.2 Referenties

In onderstaand overzicht staan de bronnen die gebruikt zijn in dit Software Architectuur Document, alsmede de vindplaats van deze documenten.

Titel	Versie	Auteur	Vindplaats
Acceptatieplan	0.2	Gertjan Haan Remi Reuvekamp	https://github.com/pannekoekmetijsappelmoesensslagroom/projectdocumenten/blob/master/AcceptatieplanGameserver.pdf
Use Case Model & Beschrijvingen	0.1	Hilko Janssen	https://github.com/pannekoekmetijsappelmoesensslagroom/aibuzzwordmachinelearningDSHselfDrivingPotato5Gnetwerk/blob/master/UseCaseDiagram_and_Description.pdf

1.3 Documentoverzicht

Hier volgt een kort overzicht van de komende hoofdstukken, samen met de belanghebbenden van dat hoofdstuk en het hoofddoel van betreffende hoofdstuk.

Hoofdstuk	Belanghebbende	Doel
2. Architecturele eisen	Software Architect	Overzicht van architectureel relevante requirements.
3. Logical View	Programmeur (t.b.v. technisch ontwerp)	Inzicht in de conceptuele structuur van de applicatie.
4. Implementation View	Programmeur (t.b.v. bouw)	Inzicht in de technische structuur van de applicatie.
5. Process View	Programmeur (t.b.v. bouw)	Inzicht in de communicatie tussen verschillende threads binnen de applicatie.
6. Deployment View	Beheerrollen	Inzicht in de manier waarop de applicatie wordt gedeployed en de manier waarop de interne en externe communicatie plaatsvindt.

2. Architecturele eisen

Dit hoofdstuk beschrijft de software-eisen die voor het ontwikkelen van de software-architectuur van belang zijn. Als eerste worden de relevante niet-functionele eisen vanuit het acceptatieplan besproken en ten slotte worden de relevante functionele eisen behandeld aan de hand van de Use Cases in het Use Case Model. De requirements die niet van belang zijn voor de architectuur, staan niet vermeld in dit hoofdstuk.

2.1 Niet-functionele requirements

Onderstaand overzicht verwijst naar de architectureel relevante niet-functionele requirements uit het Acceptatieplan (zie referenties in de inleiding).

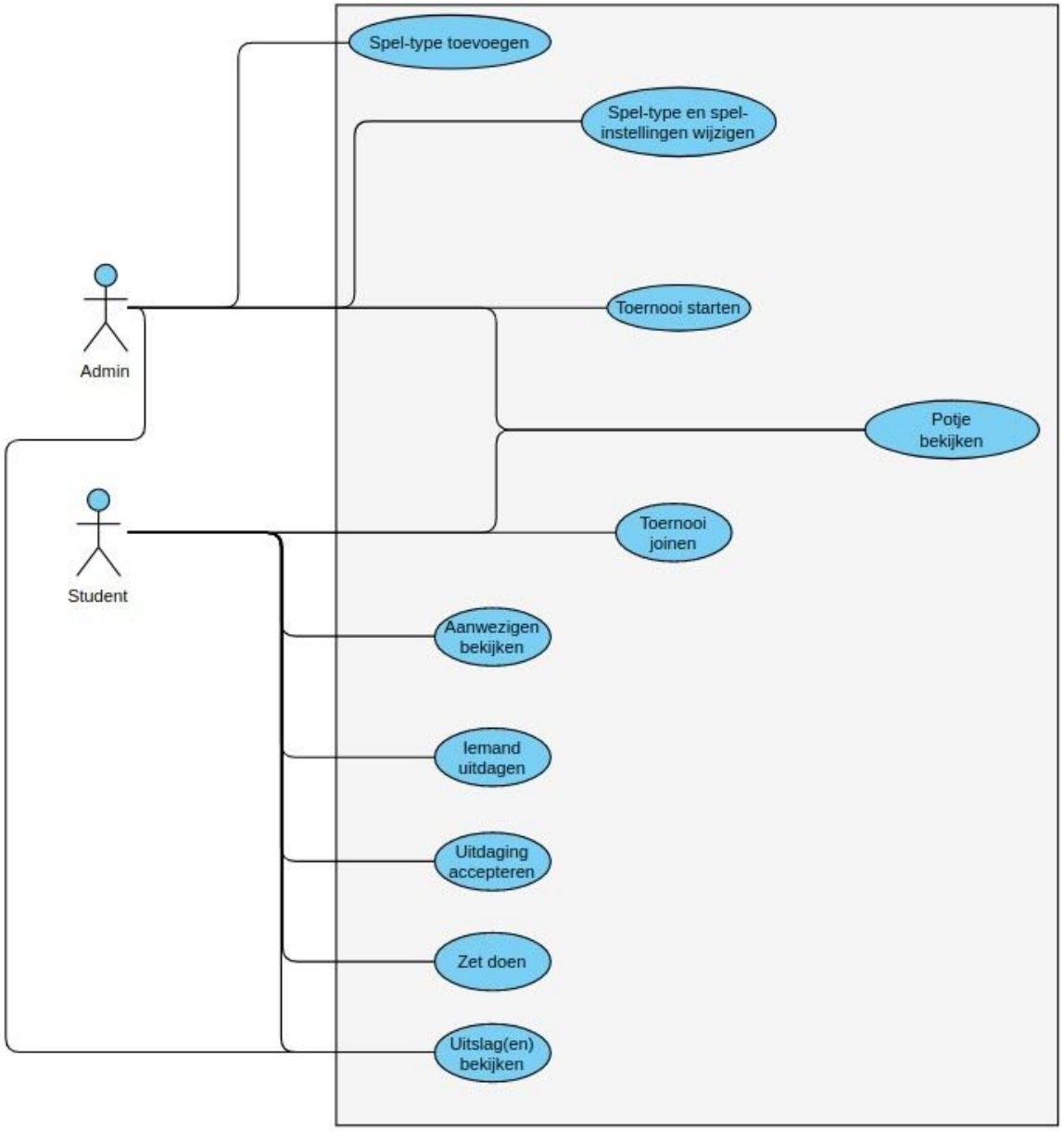
Bron	Naam	Architecturele relevantie
Acceptatieplan 2.1 Performance	Ondersteuning voor 50 spelers	De architectuur dient opgezet te worden met het uitgangspunt dat het minimaal 50 gameclients tegelijkertijd aankan zonder merkbare vertraging.
Acceptatieplan 2.1 Performance	Ondersteuning voor 75 webclients	De architectuur dient opgezet te worden met het uitgangspunt dat het minimaal 75 webclients tegelijkertijd aankan zonder merkbare vertraging.
Acceptatieplan 2.2 Beheerbaarheid	Unittests 80% code coverage	De front- en backend dient gedocumenteerd en getest te worden door middel van unittests.
Acceptatieplan 2.2 Beheerbaarheid	Loggen IP-adressen clients	De backend dient een logfile bij te houden waar naartoe geschreven wordt als een gameclient in- of uitlogt.
Acceptatieplan 2.2 Beheerbaarheid	Adminaccount	De frontend dient een inlogmogelijkheid hebben voor admins. Een admin dient de mogelijkheid te hebben om extra settings aan te passen.
Acceptatieplan 2.3 Betrouwbaarheid	Automatisch herstarten server	Indien de server crasht, dient deze automatisch herstart te worden.
Acceptatieplan 2.4 Beveiliging	Beveiliging DoS-aanvallen	De architectuur moet bestand zijn tegen DoS-aanvallen.
Acceptatieplan 2.4 Beveiliging	Inlogsysteem	Administratie functionaliteit moet slechts te gebruiken zijn door geautoriseerde administrators.

2.2 Use Case View (functionele requirements)

De Use Case View bestaat uit het Use Case Model, de Use Case beschrijvingen en een omschrijving van de architectureel relevantie van de Use Cases uit het Use Case Model. Zie de referentie in de inleiding voor uitgebreide Use Case beschrijvingen.

2.2.1 Use Case Model

Onderstaand overzicht geeft de belangrijkste Use Cases weer, gezien vanuit de actors ‘Admin’ en ‘Student’.



2.2.2 Architecturele relevantie

Onderstaand overzicht verwijst naar de architectureel relevantie Use Cases uit het Use Case Model, met daarbij een omschrijving van de architecturele relevantie.

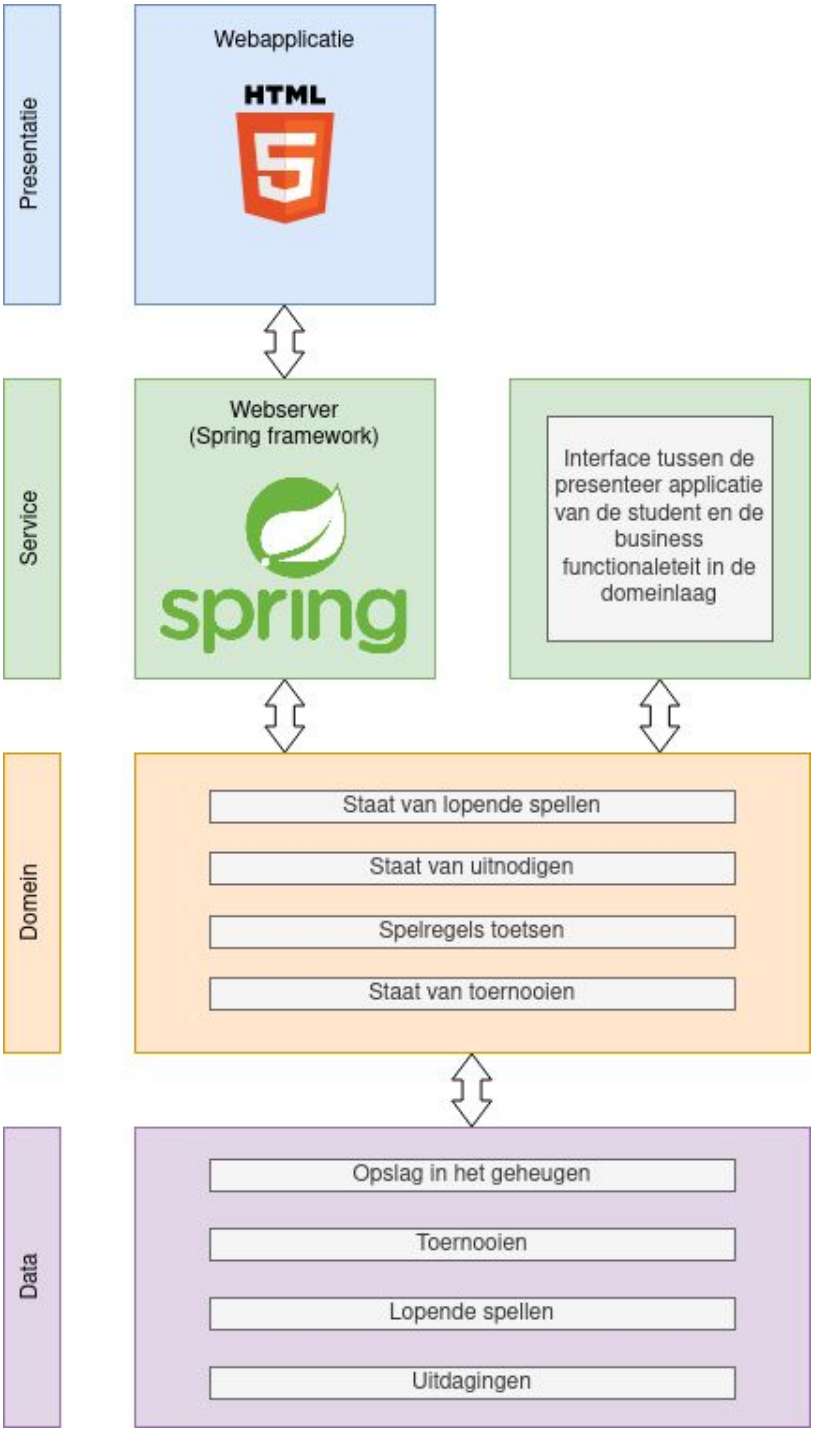
Bron	Naam	Architecturele relevantie
UCM	Speltype toevoegen	De server-architectuur moet modulair genoeg zijn zodat extra speltypes toegevoegd kunnen worden zonder de server-code te wijzigen
UCM	Speltype en spelinstellingen wijzigen	Instellingen moeten niet gehardcoded zijn.
UCM	Toernooi starten	Een toernooi kan op elk moment gestart worden door een admin. Een toernooi wordt dus niet automatisch opgestart door de server.
UCM	Potje bekijken	Potjes moeten in realtime bekeken kunnen worden door meerdere mensen. Om het realtime te maken zijn er (web)sockets nodig in de architectuur.
UCM	Toernooi joinen	Studenten moeten een server en/ of toernooi kunnen joinen vanaf andere computers met hun eigen geschreven clients.
UCM	Iemand uitdagen & Uitdaging accepteren	Studenten moeten elkaar via de server kunnen uitdagen.
UCM	Zet doen	Een client kan een zet naar de server sturen nadat de server heeft aangegeven dat de client aan de beurt is. De server moet deze zet controleren en mogelijk goedkeuren.
UCM	Uitslag(en) bekijken	De uitslagen van een toernooi en/ of potje kunnen in realtime op een frontend bekeken worden.

3. Logical View

De logical view beschrijft de logische opbouw van het systeem. In dit hoofdstuk worden de lagen van het systeem behandeld. Vervolgens worden de deelsystemen en samenhang beschreven en worden een aantal Use Cases uitgewerkt. Ten slotte volgt een activity diagram van Player vs Player.

3.1 Lagen

De logische architectuur is een viertal lagen onderverdeeld. Deze lagen worden weergegeven in onderstaande figuur en hierna verder toegelicht.



3.1.1 Presentatielaag

De presentatielaag is verantwoordelijk voor de communicatie met de gebruiker. De presentatielaag staat in verbinding met haar deel in de servicelaag.

3.1.2 Servicelaag

De servicelaag zorgt voor de koppeling tussen de gebruiker van het systeem en de logica in de domeinlaag. De verantwoordelijkheid van de servicelaag is de invoer van de presentatielaag ontvangen en deze toetsen, alsmede de presentatielaag op de hoogte te stellen van het resultaat van de invoer. De servicelaag staat in directe verbinding met de domeinlaag.

Deze laag bestaat uit twee logisch te onderscheiden delen. Het eerste deel staat garant voor de koppeling met de webapplicatie. Deze webserver is gebaseerd op het Spring framework. Het tweede deel staat garant voor de koppeling met de presentatie-applicatie van de studenten.

De communicatie tussen de servicelaag en de presentatie-applicatie van studenten geschiedt volgens het bestaande protocol over een losse TCP verbinding per presentatie-applicatie. Het protocol is bijgevoegd aan dit architectuur document, zie hiervoor bijlage 1. Voor de communicatie tussen de servicelaag en de webapplicatie in de presentatielaag worden reguliere webtechnologieën gebruikt.

3.1.3 Domeinlaag

De domeinlaag bevat de business logica van het systeem. Hiermee vormt de domeinlaag de kern. Specifiek is de domeinlaag eigenaar van de volgende logica:

- Lopende spellen.
- Lopende toernooien.
- Lopende uitnodigingen voor een spel.

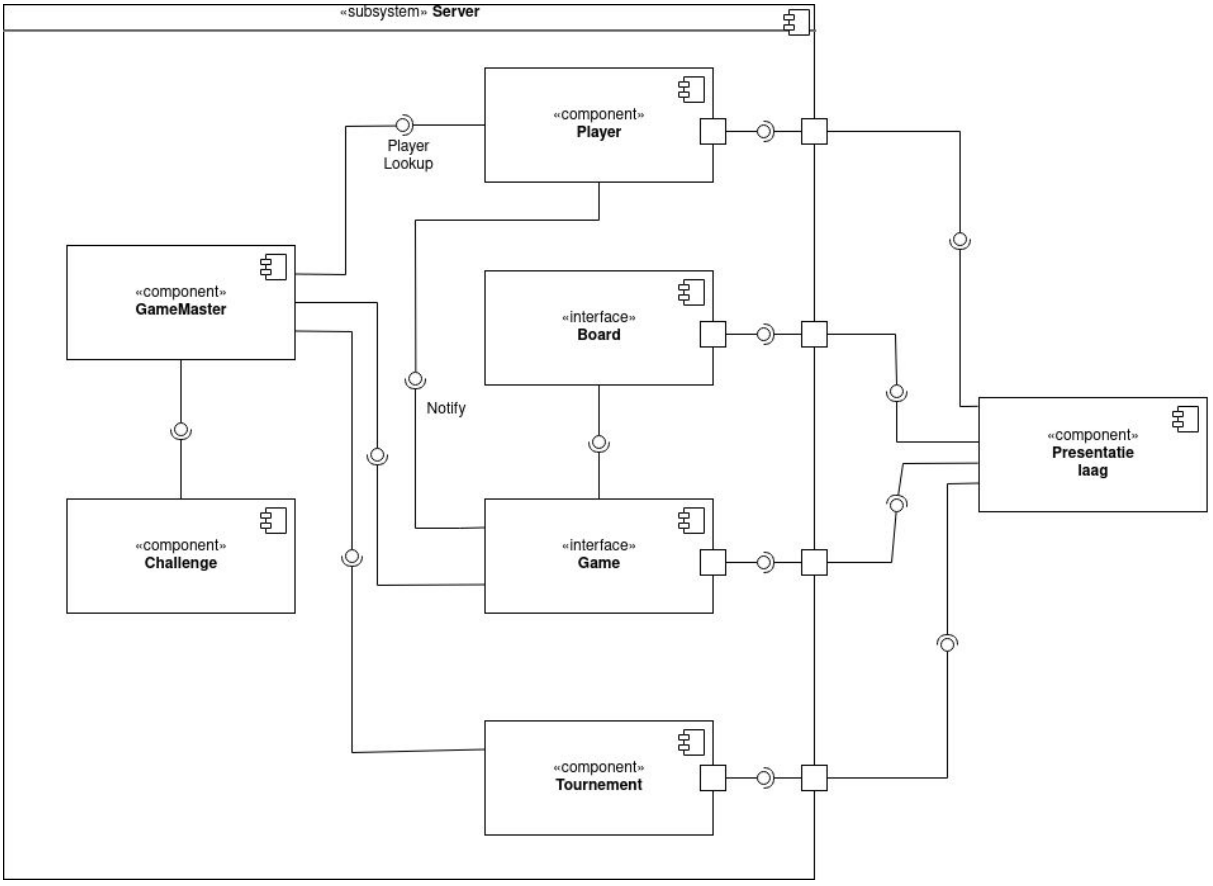
Het is de verantwoordelijkheid van de domeinlaag om opdrachten van de servicelaag uit te voeren en daarbij te toetsen of deze handeling mogelijk is. De domeinlaag communiceert naar de datalaag om wijzigingen aan onderliggende gegevens uit te voeren, als gevolg van handelingen.

3.1.4 Datalaag

De datalaag van het systeem heeft als verantwoordelijkheid het (onder)houden van data objecten van het systeem. De datalaag is geen los product en hoeft de gegevens slechts in het werkgeheugen te houden.

3.2 Deelsystemen

Hier worden de verschillende deelsystemen/ componenten weergegeven die van belang zijn voor de gameserver.

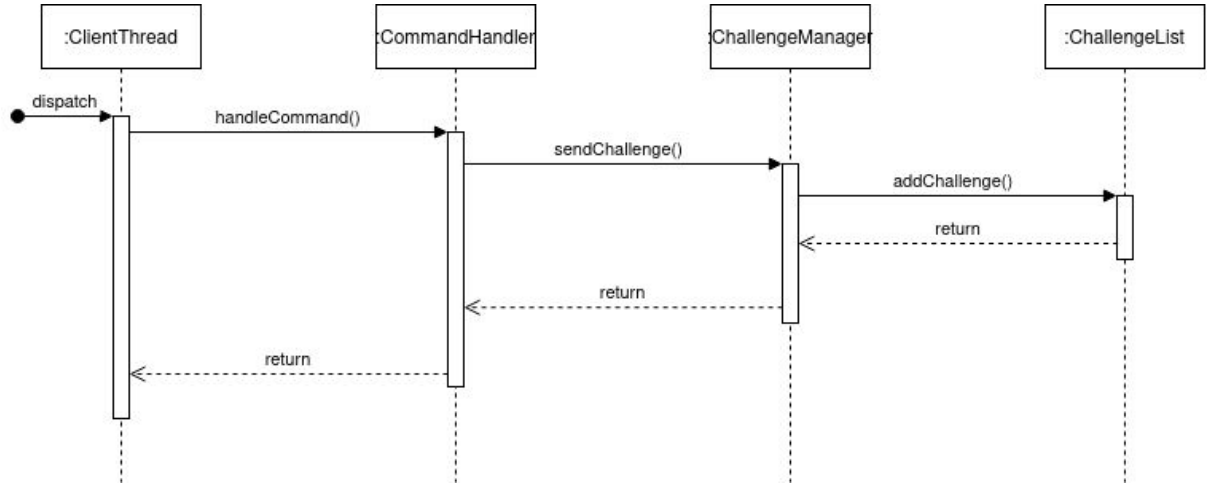


3.3 Use Case Realizations

In dit deelhoofdstuk zijn een aantal Use Cases uitgewerkt in sequence diagrammen. Zoals voorgeschreven zijn niet alle Use Cases uitgewerkt, maar slechts een aantal losstaande met als doel een goed beeld van de benodigde architectuur te geven.

3.3.1 Use Case: 'Iemand Uitdagen'

In onderstaand figuur wordt de Use Case "Iemand Uitdagen" gerealiseerd in een sequence diagram.

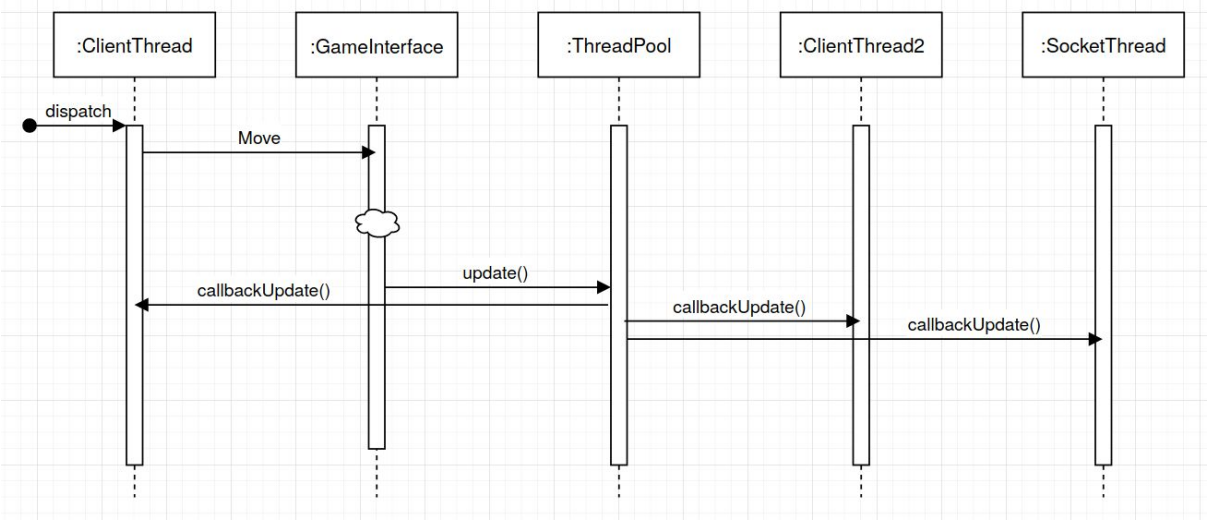


In dit diagram is te zien hoe een aanvraag van een presentatie-applicatie van een student wordt ontvangen door de ClientThread, die deel uitmaakt van de servicelaag. De ruwe aanvraag wordt aan de CommandHandler gegeven, die de aanvraag ontleedt en toets. De ChallengeManager wordt vervolgens op de hoogte gesteld van de nieuwe uitdaging die, nadat blijkt dat de uitdaging mogelijk is, de uitdaging opslaat middels de ChallengeList.

Niet weergegeven is de taak van de ChallengeManager om de uitgedaagde speler in te lichten.

3.3.2 Use Case: 'Een zet doen'

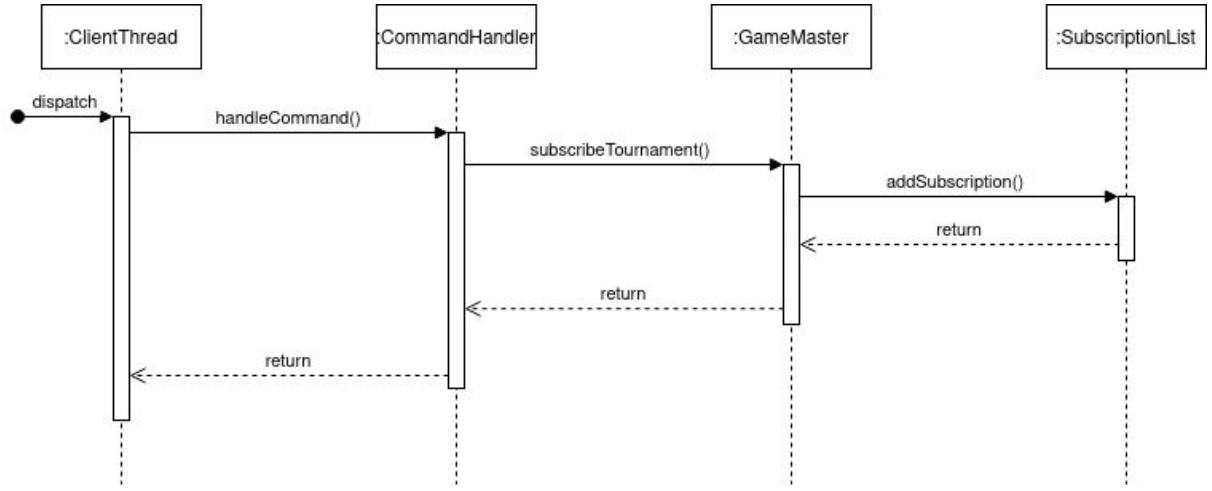
In onderstaande figuur wordt de Use Case "een zet doen" gerealiseerd in een sequence diagram.



Hier is te zien dat een Client een move kan sturen naar de Game-interface. De specifieke game die gespeeld wordt bepaald wat de spelregels zijn. De Game-interface maakt op zijn plaats dus afhankelijk van het spel ook meerdere keren gebruik van het bord. Dit is bij ieder spel anders, daarom staat dit aangegeven met een wolkje in dit diagram. Als de game klaar is stuurt de game naar de Threadpool wat de huidige staat van de game is. Iemand zou bijvoorbeeld kunnen winnen of de volgende speler is aan de beurt etc. Ieder spel zou dit ook weer anders kunnen doen, maar over het algemeen komt het erop neer dat een enkele update naar de threadpool genoeg is om te voldoen aan het protocol en iedereen correct te updaten.

3.3.3 Use Case: “Toernooi joinen”

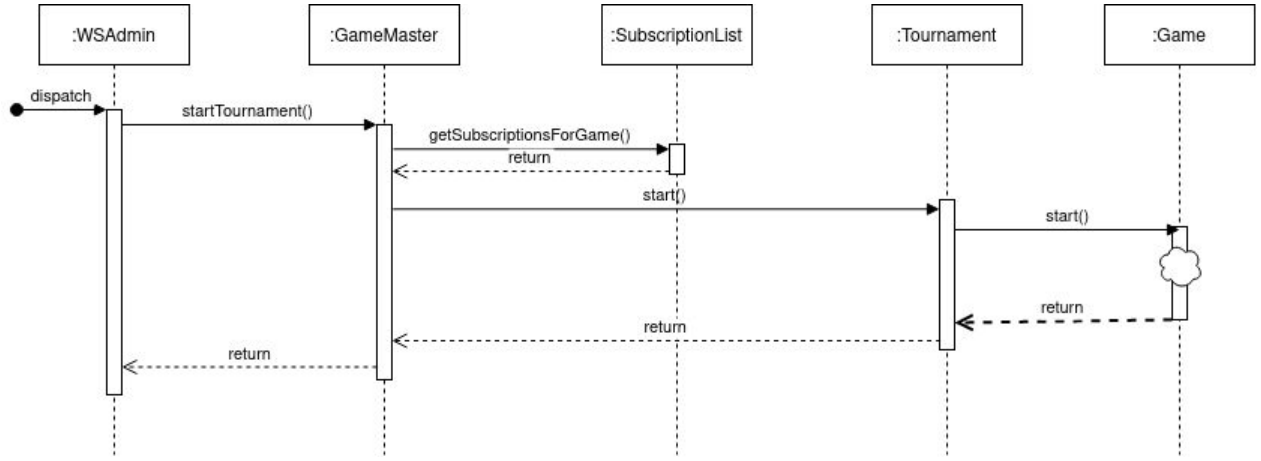
In onderstaand figuur wordt de Use Case “Toernooi joinen” gerealiseerd in een sequence diagram.



In bovenstaand diagram is te zien hoe de aanvraag van de presenteerapplicatie van een student aankomt in de ClientThread. De ruwe text die de aanvraag bevat wordt doorgegeven aan de CommandHandler bijbehorend aan de ClientThread, die dan de aanvraag ontleedt. De GameMaster wordt op de hoogte gebracht van de aanvraag tot inschrijving. Als de aanvraag correct is wordt de inschrijving in het geheugen opgeslagen middels de SubscriptionList.

3.3.4 Use Case: “Toernooi starten”

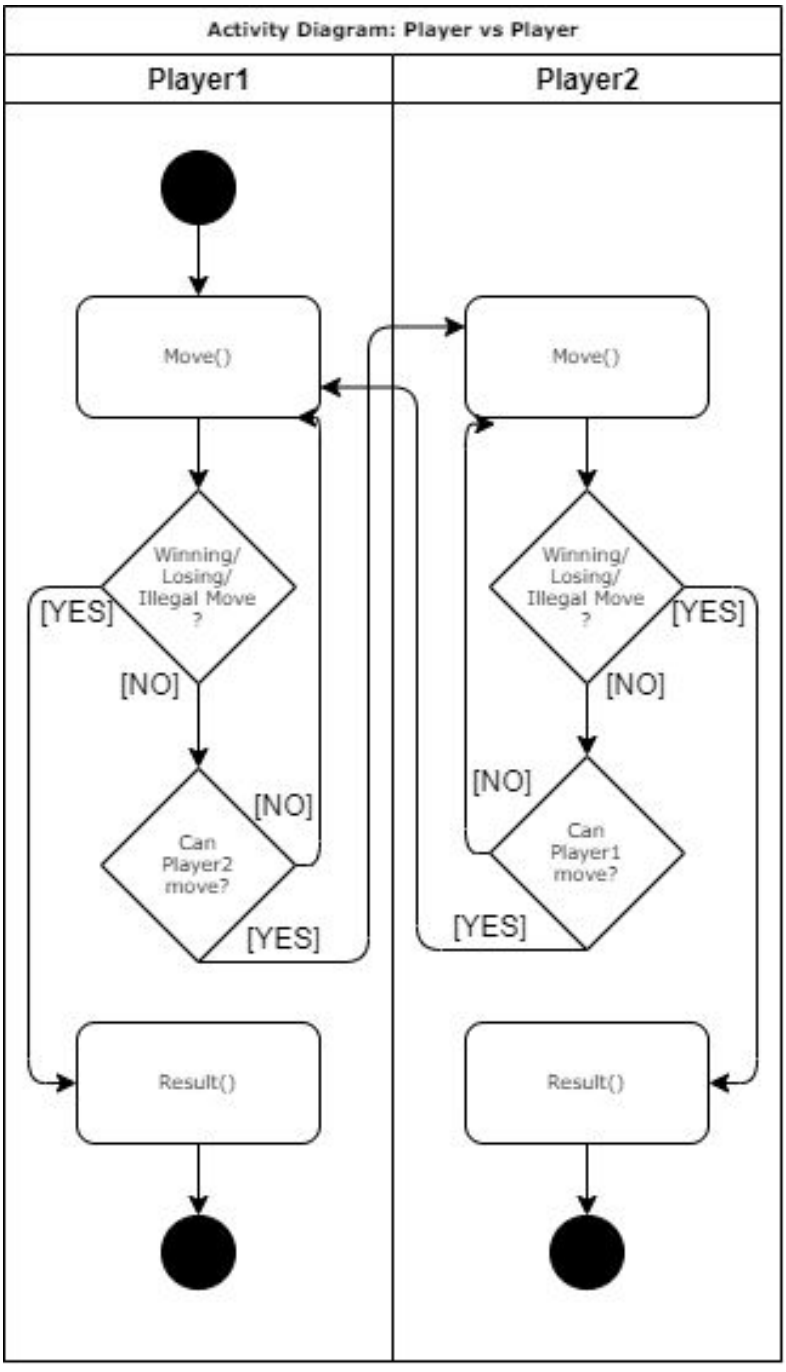
In onderstaand figuur wordt de Use Case “Toernooi starten” gerealiseerd in een sequence diagram.



Op dit diagram is te zien hoe een aanvraag om een toernooi te starten verloopt. Via de webserver komt dit bericht aan bij de WSAdmin, die de aanvraag doorstuurt naar GameMaster. GameMaster haalt als eerste de lijst met spelers op die staan ingeschreven in het toernooi en daarna wordt een tournament opgezet. Dit tournament bepaalt welke game als eerst gespeeld dient te worden, waarna de game wordt opgestart. De specifieke logica van het spel is aangeduid met een wolkje.

3.4 Activity Diagram

In onderstaande activity diagram wordt weergegeven welke activiteiten er zijn zodra een speler tegen een andere speler speelt. Het begint met de speler die aan zet is en het eindigt wanneer iemand een winnende of verliezende zet speelt. Een illegale zet geldt hierbij tevens als verliezende zet.



4. Implementation view

Tijdens dit project werken we met verschillende talen. De implementatie is daarom voor het Java gedeelte anders dan het Vue.js gedeelte. Het komt erop neer dat het wenselijk is net zoals de oude situatie alles in een JAR-bestand te hebben. In dit hoofdstuk staat daarom uitgelegd hoe we ervoor gaan zorgen dat de nieuwe server zo veel mogelijk op de oude server lijkt.

4.1 Unittests

Om te garanderen dat belangrijke onderdelen van de applicatie niet kapot gaan worden er unittests geschreven. Unittests kunnen worden toegevoegd wanneer een programmeur dat nodig dient. Er wordt dus niet gewerkt met Test Driven Development, TDD. Unittests zijn er alleen om eigenschappen en kenmerken van de vorige server af te dwingen in de nieuwe server. Een voorbeeld hiervan is bijvoorbeeld dat de vorige server enkele extra berichten naar clients stuurt die niet in het protocol zijn opgenomen. Als er zo'n verschil ontdekt wordt tussen de voormalige server en de nieuwe server dan zal dit in een unittest worden geplaatst om deze extra requirement te waarborgen.

4.2 Development environment

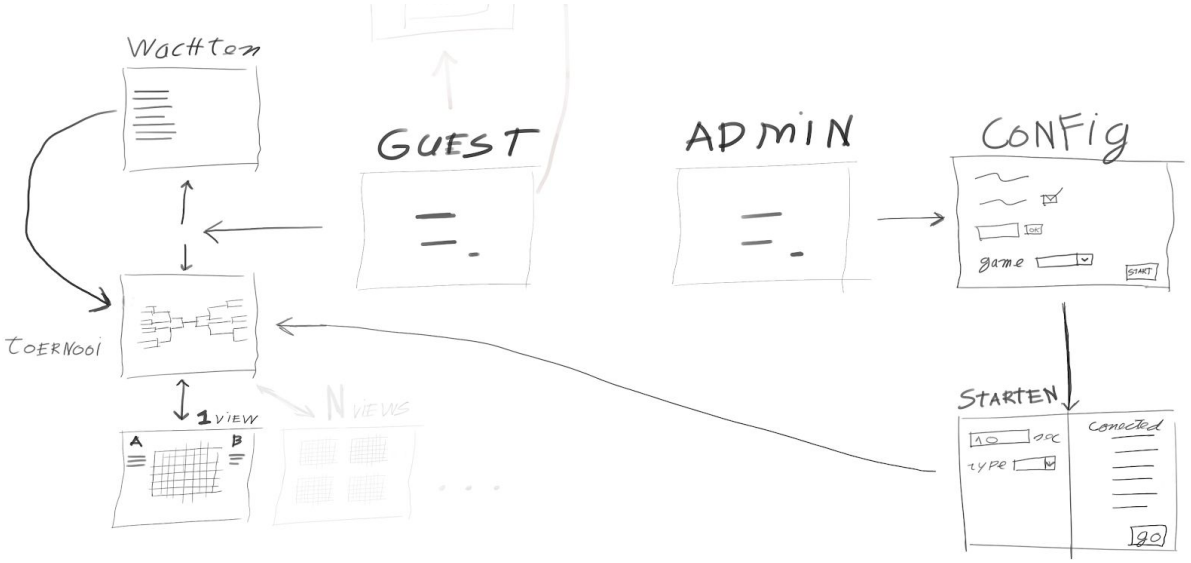
Voor het Java gedeelte van het project gebruiken we Gradle om dependencies te laden. Verder gebruiken we minimaal een Java versie hoger dan 9 zodat we gebruik kunnen maken van de HTTPServer en HTTPClient die in Java 9 geïntroduceerd zijn als beta. Binnen Gradle hebben we daarom voor de eerstvolgende long term support versie gekozen, java 11.

Voor de frontend gebruiken we npm om dependencies in te laden en gebruiken we npm ook om alle resources uiteindelijk te bundelen naar een geoptimaliseerde export.

4.3 Framework / Programming language

In dit project werken we met Java op de backend en gebruiken we Vue.js op de frontend. De Vue.js wordt omgezet naar statische HTML bestanden zodat er vanuit het oogpunt van de eindgebruiker alleen maar een enkele Java JAR file te zien is.

Dit is een ruwe schets van de frontend.

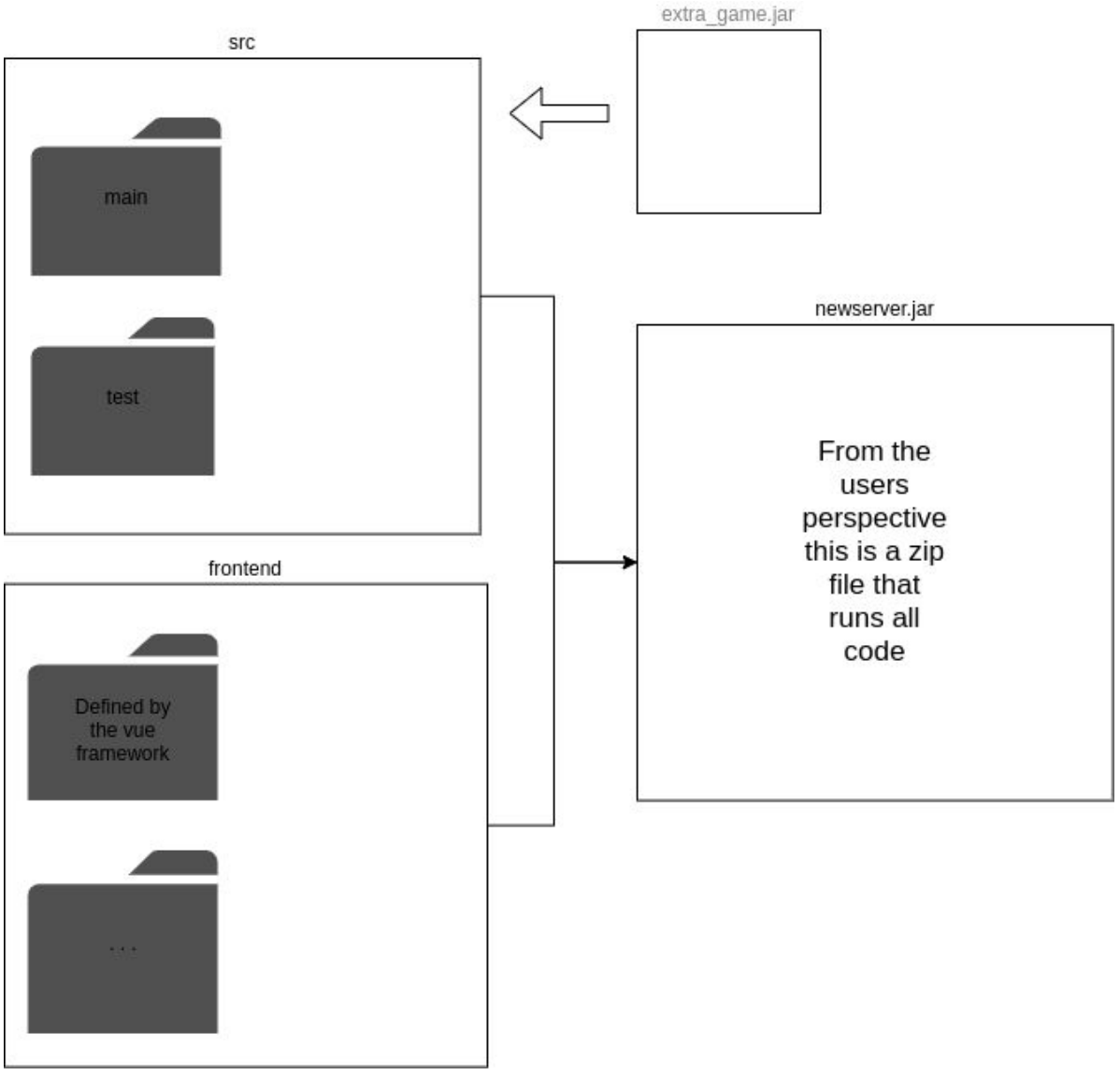


4.4 Package Structuur

De package structuur komt er als volgt uit te zien:

- SRC - (Java servers)
    - Game server
        - (MVP) de verschillende games
    - Frontend/static site server
- frontend - (vue development project)
- (na MVP) games
    - TicTacToe.jar
    - Reversi.jar
    - ....

Dit is ter verduidelijking weergegeven in onderstaand diagram.



4.5 Invulling lagenstructuur

De lagenstructuur is onderverdeeld in vier verschillende lagen. De betreffende lagen worden verder toegelicht, met betrekking op onder meer de gekozen frameworks en programmeertalen

4.5.1 Presentatielaag

De presentatielaag wordt gedaan door het framework Vue. De presentatie laag zal uiteindelijk geëxporteerd worden naar statische bestanden. Deze statische bestanden worden dan in het opgeleverde werk gestopt.

4.5.2 Servicelaag

De gameserver die de mogelijkheid geeft om met het huidige protocol een spel te spelen wordt geïmplementeerd in Java code. Deze Java code zit in de “src/main” folder.

4.5.3 Domeinlaag

De speelbare spellen zijn onderdeel van de domeinlaag. De spellen worden voor de MVP geïmplementeerd in de main folder, maar worden naar wens van de klant mogelijk later verplaatst naar de oude manier van een folder met verschillende JAR-bestanden. Iedere JAR in deze folder implementeert dan een nieuwe spel.

4.5.4 Datalaag

De MVP maakt geen gebruik van een datalaag. Alle belangrijke data wordt in memory opgeslagen. Als er een wens is om data bij te houden kan de standaard out van de server gelogd worden.

De frontend haalt al zijn standaard waarden uit een configuratie bestand of memory. Dit is opnieuwe niet een echte datalaag maar meer een gemakkelijke manier om alles in memory te houden.

4.6 (Her)gebruik van componenten en frameworks

De Javaservers delen hetzelfde geheugen en de code. Tijdens de ontwikkeling is er ook een vue Project. Vue heeft een eigen ontwikkelings server met hot reloading. Deze wordt daarom gebruikt voor de ontwikkeling. Tijdens een oplevering of een demo wordt alle code gecombineerd naar 1 Jar file. Dit wordt gedaan door het Vue project om te zetten naar statische HTML. Deze HTML wordt dan in de Jar geladen. Alleen de Java servers gebruiken dus elkaars code. De frontend is volledig gescheiden en zal enkel via een socket of http connectie met de server praten.

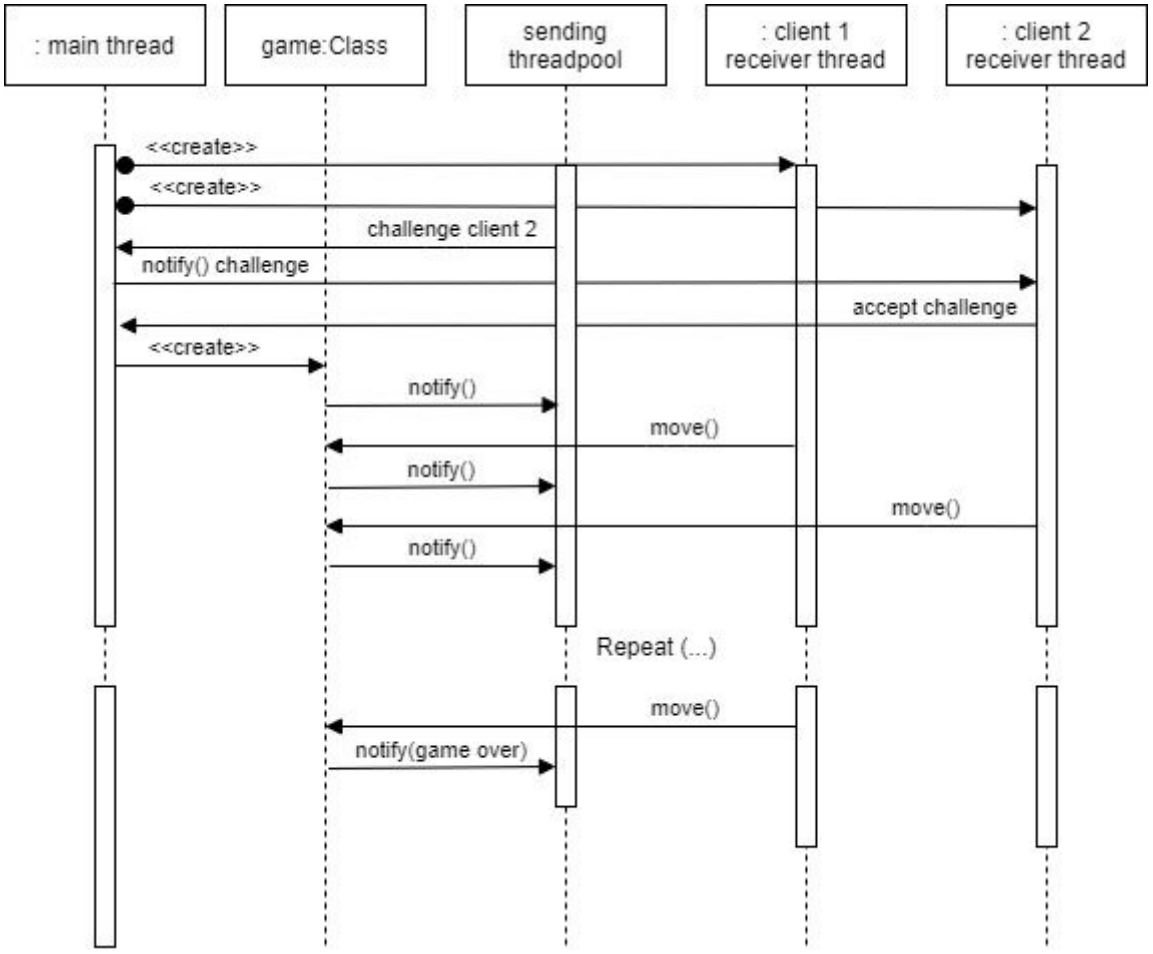
4.7 Errors en warnings

De server schrijft alle gebeurtenissen naar stdout. Indien er gewenst wordt om statistieken te onthouden of om warnings / error te bewaren kan stdout naar een bestand worden geschreven op de productieserver.

Met andere woorden er is geen concrete data laag alles wordt in memory gedaan binnen de MVP.

5. Process view

De process view geeft een overzicht van de belangrijkste threads en de samenwerking tussen deze threads binnen de gameserver. Voor elke cliënt wordt een losse client thread aangemaakt. De threadpool in onderstaand UML diagram bestaat vervolgens uit alle clients die met de gameserver verbonden zijn.

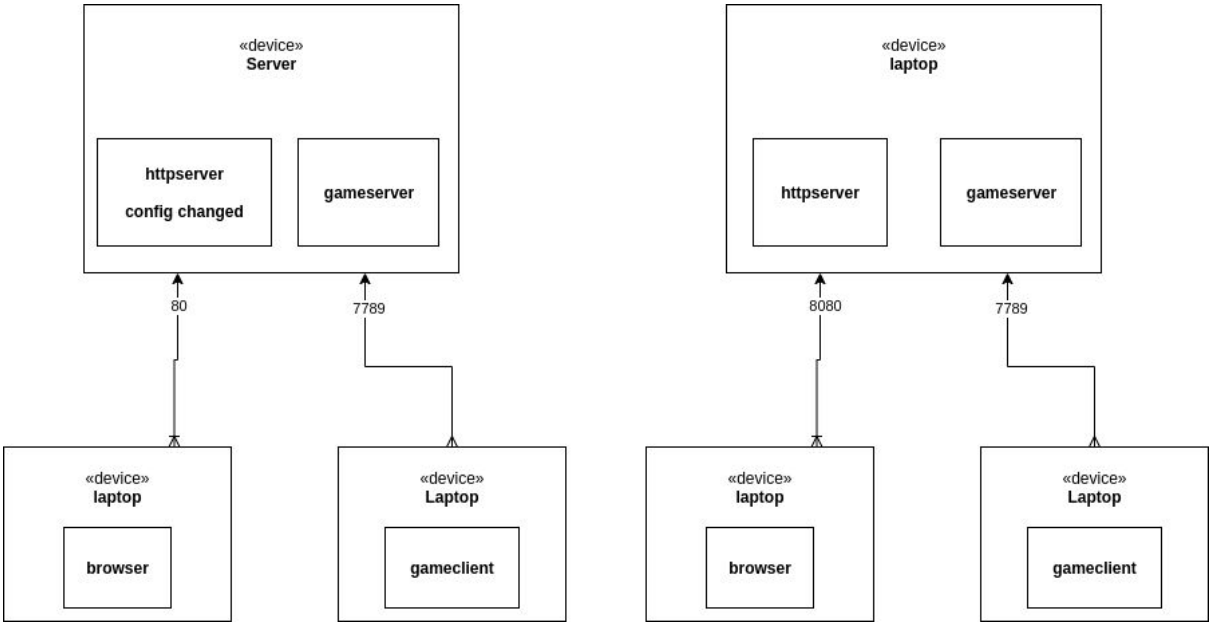


6. Deployment view

De server zal zowel gedeployed worden in productie als op een aantal losse laptops. Op de server zal de server stabiel moeten werken zonder grote aanpassingen te hoeven doen. De deployment op de losse laptops hoeft niet volledig vastgezet te worden. Om de deployment zo makkelijk mogelijk te maken zal de volledige server moeten gaan werken vanuit een enkele JAR-file. Door een enkele JAR-file te gebruiken hoeven we verder weinig rekening te houden op welk besturingssysteem het werk gedeployed wordt.

Naam	Type	Omschrijving
Productieserver		De productieserver is een op windows of linux gebaseerd systeem die al ingesteld staat om bereikbaar te zijn binnen het hanze netwerk. Binnen de config van ons werk zijn alle poorten handmatig in te stellen. Als er op de productieserver ook andere processen/ servers draaien wordt er verwacht dat de beheerder dit zelf naast elkaar weet te laten draaien. Om de server makkelijker te bereiken kan er worden gekozen om in de configuratie poort 8080 te veranderen naar 80.
X aantal studenten laptops		Een laptop met een besturingssysteem dat ondersteuning geeft voor Java is genoeg om de JAR werkent te krijgen.

Als verdere toelichting op de deployment is in onderstaand overzicht weergegeven op welke poorten de httpserver en gameserver te benaderen zijn door de browser danwel gameclient.



Bijlage 1. Protocol

Dit document beschrijft het gameserver protocol tussen de presentatie applicatie van studenten (“client”) en het te ontwikkelen systeem (“server”). Dit protocol is door de opdrachtgever aangeleverd.

Overzicht van server-berichten

OK	Commando geaccepteerd
ERR	Commando afgewezen
SVR [HELP | GAME [MATCH | YOURTURN | MOVE | CHALLENGE | [WIN | LOSS | DRAW]]]	
	Bericht van server
HELP	Bericht met help informatie
GAME	Bericht met betrekking op een spel/match
MATCH	Toewijzing van een match
YOURTURN	Toewijzing van de beurt tijdens de match
MOVE	Een zet gedaan tijdens de match
CHALLENGE	Bericht met betrekking op een uitdaging
WIN	Ontvanger heeft spel gewonnen
LOSS	Ontvanger heeft spel verloren
DRAW	Match is geëindigd in gelijkspel

Overzicht van client-commando's

login	Aanmelden als speler
logout | exit | quit | disconnect | bye	
	Uitloggen en verbinding verbreken
get <gamelist | playerlist>	
	Opvragen van gegevens
gamelist	Opvragen van de lijst met ondersteunde speltypes
playerlist	Opvragen van de lijst met aangemelde spelers
subscribe	Inschrijven voor een speltype
move	Een zet doen tijdens een match
challenge [accept]	Uitdagingen behandelen
accept	Uitdaging accepteren
forfeit	De huidige match opgeven
help [commando]	Help weergeven

Commando's in detail

C = Client
S = Server

Notes bij server antwoorden:
Items tussen vierkante haken ('[' en ']') geven een lijst weer.
Items tussen accolades ('{' en '}') geven een map weer. Zoals bij alle maps, is de volgorde niet bepaald.

Notes bij client commando's:
Alle commando's zijn niet hoofdlettergevoelig.
Alle argumenten zijn niet hoofdlettergevoelig, m.u.v. namen van spelers en speltypes.

Niet ondersteunde commando:
C: <niet ondersteunde commando>
S: ERR <reden>
->Geen gevolg.

Inloggen:
C: login <speler>
S: OK
->Nu ingelogd met spelernaam <speler>.

Uitloggen/Verbinding verbreken:
C: logout | exit | quit | disconnect | bye
S: -
->Verbinding is verbroken.

Lijst opvragen met ondersteunde spellen:
C: get gamelist
S: OK
S: SVR GAMELIST ["<speltype>", ...]
->Lijst met spellen ontvangen.

Lijst opvragen met verbonden spelers:
C: get playerlist
S: OK
S: SVR PLAYERLIST ["<speler>", ...]
->Lijst met spelers ontvangen.

Inschrijven voor een speltype:
C: subscribe <speltype>
S: OK
->Ingeschreven voor speltype <speltype>.

Match aangeboden krijgen, bericht naar beide spelers:
S: SVR GAME MATCH {GAMTYPE: "<speltype>", PLAYERTOMOVE: "<naam speler1>", OPPONENT: "<naam tegenstander>"}
->Nu bezig met een match, de inschrijving voor een speltype is vervallen.

De beurt toegewezen krijgen tijdens match:
S: SVR GAME YOURTURN {TURNMESSAGE: "<bericht voor deze beurt>"}
->Nu mogelijkheid een zet te doen.

Een zet doen na het toegewezen krijgen van een beurt:
C: move <zet>
S: OK
->De zet is geaccepteerd door de server, gevolg voor spel zal volgen

Resultaat van een zet ontvangen, bericht naar beide spelers:
S: SVR GAME MOVE {PLAYER: "<speler>", DETAILS: "<reactie spel op zet>", MOVE: "<zet>"}
->Er is een zet gedaan, dit bericht geeft aan wie deze gezet heeft, wat de reactie van het spel erop is

Resultaat van een match ontvangen, bericht naar beide spelers:
S: SVR GAME <speler resultaat> {PLAYERONESCORE: "<score speler1>", PLAYERTWOSCORE: "<score speler2>", COMMENT: "<commentaar op resultaat>"}
->De match is afgelopen, <speler resultaat> kan de waarde 'WIN', 'LOSS' of 'DRAW' bevatten.

Een match opgeven:
C: forfeit
S: OK
->De speler heeft het spel opgegeven, de server zal het resultaat van de match doorgeven.

Resultaat van een match die opgegeven is door een speler, bericht naar beide spelers:
S: SVR GAME <speler resultaat> {PLAYERONESCORE: "<score speler1>", PLAYERTWOSCORE: "<score speler2>", COMMENT: "Player forfeited match"}
->De match is afgelopen, <speler> heeft de match opgegeven.

Resultaat van een match, speler heeft de verbinding verbroken:
S: SVR GAME <speler resultaat> {PLAYERONESCORE: "<score speler1>", PLAYERTWOSCORE: "<score speler2>", COMMENT: "Client disconnected"}
->De match is afgelopen, <speler> heeft de verbinding verbroken.

Een speler uitdagen voor een spel:
C: challenge "<speler>" "<speltype>"
S: OK
->De speler is nu uitgedaagd voor een spel. Eerder gemaakte uitdagingen zijn komen te vervallen.

Een uitdaging ontvangen:
S: SVR GAME CHALLENGE {CHALLENGER: "Sjors", GAMETYPE: "Guess Game", CHALLENGENUMBER: "1"}
->Nu mogelijkheid de uitdaging te accepteren.

Resultaat van een uitdaging die is komen te vervallen:
S: SVR GAME CHALLENGE CANCELLED {CHALLENGENUMBER: "<uitdaging nummer>"}
->De uitdaging is vervallen. Mogelijke oorzaken: speler heeft een andere uitdaging gestart, speler is een match begonnen, speler heeft de verbinding verbroken.

Een uitdaging accepteren:
C: challenge accept <uitdaging nummer>
S: OK
->De uitdaging is geaccepteerd. De match wordt gestart, bericht volgt.

Help opvragen:
C: help
S: OK
->De client heeft nu help informatie opgevraagd, de server zal antwoorden met help informatie.

Help opvragen van een commando:
C: help <commando>
S: OK
->De client heeft nu help informatie opgevraagd voor een commando, de server zal antwoorden met help informatie.

Help informatie ontvangen:
S: SVR HELP <help informatie>
->Help informatie is ontvangen, kan meerdere achtereenvolgende responses bevatten.