


Software Design Description

Created by Auto Mation, last modified by Sven van Beek on Jun 10, 2021

Onderwerp	Software Design Description
Status	DONE
Versie	3.0
Datum	 10 Jun 2021
Docenten	@ Eveline Bouwman @ Rody Middelkoop @ Michel Koolwaaij
Projectgroep	ASD 2020-2021-s2 Projectgroep 1
Eindverantwoordelijke	Architecten en toolbeheerders

Inhoud

- 1. Introductie
 - 1.1. Algemene beschrijving
 - 1.2. Doel van dit document
 - 1.3. Definities, termen en afkortingen
- 2. Architectural Overview
- 3. Detailed Design Description
 - 3.1. Deployment Diagram
 - 3.2. Design Sub-System Action Handling
 - 3.2.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System
 - 3.2.2. Gebruikte interfaces
 - 3.2.3. Algoritme implementatie
 - 3.2.4. Design Class Diagram
 - 3.2.5. Sequence Diagrams
 - 3.2.6. Activity and State Diagrams
 - 3.2.6.1. Move handling
 - 3.2.6.2. Radiation event
 - 3.2.6.3. Attack handling
 - 3.2.7. Design decisions made for the sub-system
 - 3.3. Design Sub-System AI (Parser/Lexer)
 - 3.3.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System
 - 3.3.2. Interface overview
 - 3.3.3. Algoritme Implementatie
 - 3.3.4. Design Class Diagram
 - 3.3.5. Sequence Diagrams
 - 3.3.6. Activity and State Diagrams
 - 3.3.6.1. Activity Diagram Generating
 - 3.3.6.2. Activity Diagram Checking
 - 3.3.7. Design decisions made for the sub-system
 - 3.4. Design Sub-System AI (Monsters/bots/NPC's)
 - 3.4.1. Pathfinder Algoritme
 - 3.4.1.1. Werking A* algoritme
 - 3.4.2. MachineLearning Algoritme
 - 3.4.2.1. Werking neurologisch netwerk
 - 3.4.2.2. Werking Genetisch algoritme
 - 3.4.3. Design Class Diagram
 - 3.4.4. Sequence Diagram
 - 3.4.5. State Diagram
 - 3.4.6. Design decisions made for the sub-system
 - 3.5. Design Sub-System Input Handling
 - 3.5.1. Interfaces:
 - 3.5.2. Functionele en niet-functionele requirements die gedekt worden door het Sub-System
 - 3.5.3. Algoritme Implementatie
 - 3.5.4. Design Class Diagram
 - 3.5.5. Sequence Diagrams
 - 3.5.6. Design decisions made for the sub-system
 - 3.6. Design Sub-System World Generation
 - 3.6.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-Systeem
 - 3.6.2. Design Class Diagram

- 3.6.2.1. Glossary
- 3.6.3. External interfaces
- 3.6.4. Sequence Diagrams
- 3.6.5. Function index
- 3.6.6. Algoritme
- 3.6.7. Design decisions made for the sub-system
- 3.7. Design Sub-System Network Switch
 - 3.7.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System
 - 3.7.2. Gebruikte interfaces
 - 3.7.3. Design Class Diagram
 - 3.7.4. Uitleg Packet, Header en Payload
 - 3.7.5. Sequence Diagrams
 - 3.7.6. Activity and State Diagrams
 - 3.7.7. Design decisions made for the sub-system
- 3.8. Design Sub-System Items
 - 3.8.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System
 - 3.8.2. Gebruikte interfaces
 - 3.8.3. Algoritme implementatie
 - 3.8.4. Design Class Diagram Item
 - 3.8.4.1. Glossary
 - 3.8.5. Sequence Diagram Builder Pattern
 - 3.8.6. Design decisions made for the sub-system
- 3.9. Design Sub-System Database Handling
 - 3.9.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System
 - 3.9.2. Design Class Diagram
 - 3.9.3. Sequence Diagram
 - 3.9.3.1. Initialisatie database verbinding
 - 3.9.4. Design decisions made for the sub-system
- 3.10. Design Sub-System User Interface
 - 3.10.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System
 - 3.10.2. Gebruikte interfaces
 - 3.10.3. Algoritme Implementatie
 - 3.10.4. Design Class Diagram
 - 3.10.4.1. Glossary
 - 3.10.5. Sequence Diagrams
 - 3.10.6. Activity and State Diagrams
 - 3.10.7. Activity diagram logica Editor screen
 - 3.10.8. Design decisions made for the sub-system
- 3.11. Database Design
 - 3.11.1. Design decisions related to the database

1. Introductie

Dit document is de Software Design Description voor het dungeon-spel. Hierin worden de werking en de implementatie van de applicatie beschreven. In het hoofdstuk Detailed Design Description is per onderdeel beschreven hoe deze in elkaar zit. Bijvoorbeeld de lexer/parser en de NPC's/monsters. Elk subteam bepaald zelf of het nuttig is om bijvoorbeeld een activity, sequence of class diagram te maken. Deze diagrammen moeten een verduidelijking geven op de tekstuele uitleg.

1.1. Algemene beschrijving

In het hoofdstuk [Opdrachtoomschrijving](#) van het plan van aanpak is een algemene beschrijving gegeven van de applicatie.

1.2. Doel van dit document

Het doel van dit document is het beschrijven van het design voor het dungeon-spel. Hierin wordt eerst een architectural overview weergeven. Daarna volgen alle verschillende subsystemen met de bijbehorende diagrammen. In dit document staat in ieder geval beschreven:

- Hoe het gebouwd moet worden
- De belangrijkste ontwerpkeuzes

Deze onderdelen worden toegelicht door middel van class diagrams en sequence diagrams en de daarbij horende beschrijvingen. Dit document is zo opgesteld dat het voor een nieuwe ontwikkelaar of groep ontwikkelaars mogelijk is om de hele applicatie te realiseren.

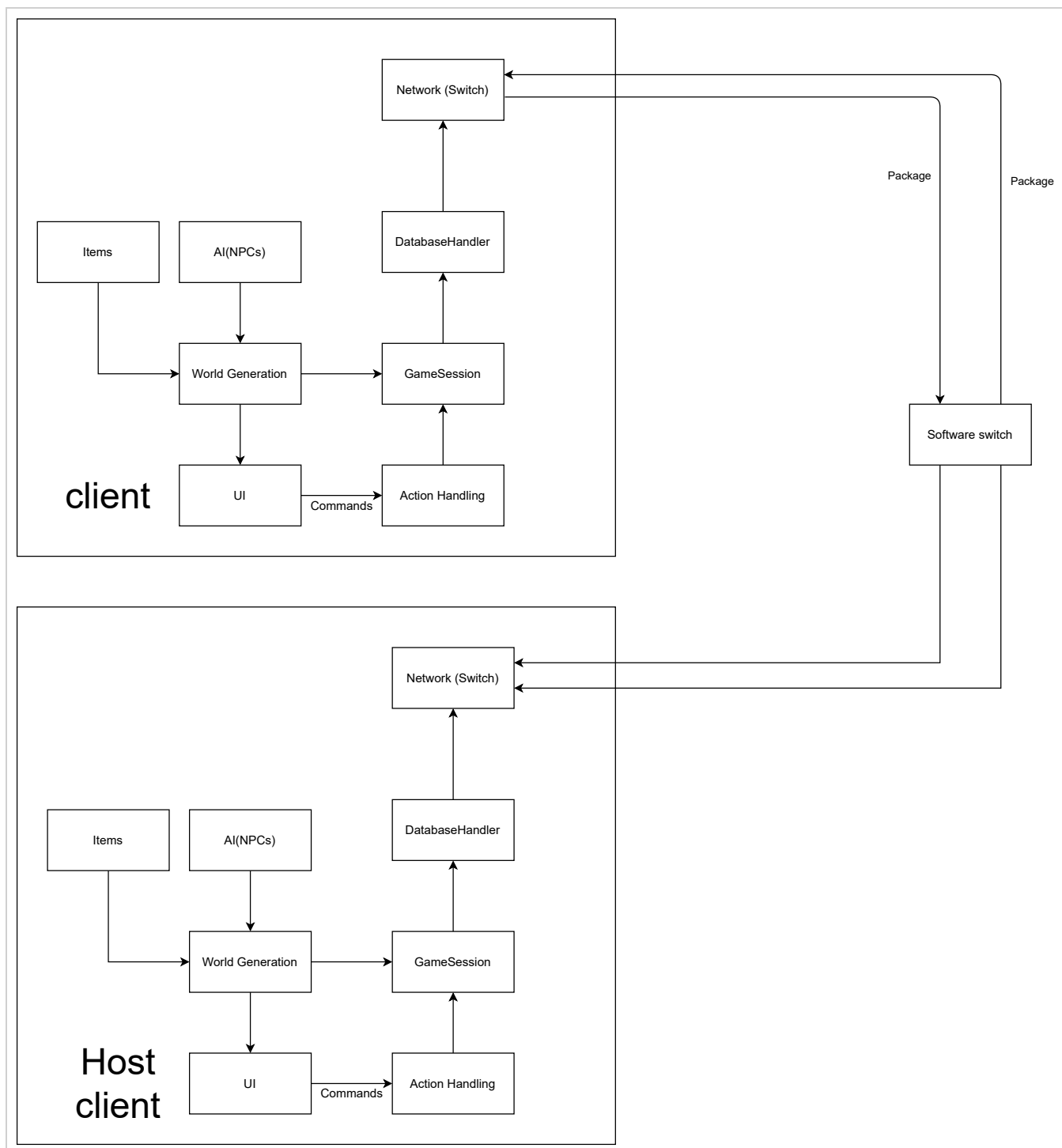
1.3. Definities, termen en afkortingen

Term	Beschrijving
SDD	Software Design Description
NPC	Non-playable character: een karakter in het spel dat niet bespeelbaar is door een gebruiker van het spel.

Term	Beschrijving
UI	User Interface
FR	Functional Requirement
ASR	Architectural Significant Requirement
Parser	Een parser is een computerprogramma , of component van een programma, dat de grammaticale structuur van een invoer volgens een vastgelegde grammatica ontleedt (<i>parse</i>). Een parser converteert de ingevoerde tekst in een datastructuur .

2. Architectural Overview

Hieronder staat de architectural overview van de dungeon game. Binnen het diagram is te zien dat de speler via de UI de commando's kan invoeren en deze door worden gegeven aan de parser. Als de speler het commando voor de agent invoert word het betreffende bestand via de agent geïmporteerd en gecompileerd. Als dit goed gaat word er een dictionary aangemaakt die doorgegeven wordt aan het creature-component. Dit component gebruikt de dictionary om te bepalen hoe de NPC's en agent ingesteld moeten worden. Als de speler beweegt word er tegen het netwerkcomponent gezegd dat er een beweging is gebeurd. Deze beweging word dan gecommuniceerd via de netwerkswitch naar de andere client. Als er in plaats van een beweging een chatbericht wordt gestuurd, dan gaat dit via de chathandler. Die verstuurd het bericht via een package naar alle clients.

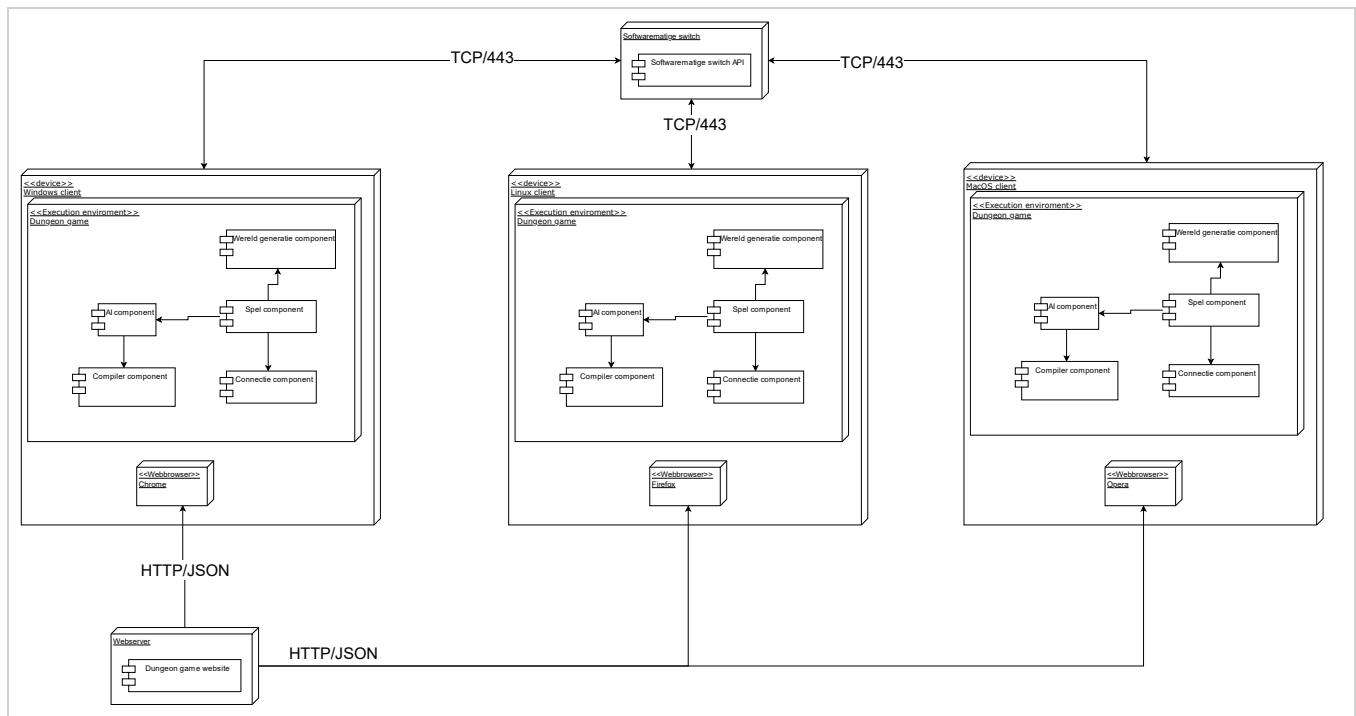


3. Detailed Design Description

Dit onderdeel van het SDD gaat in op de ontwerpen van de verschillende aanwezige **subsystemen** in de op te leveren applicatie.

3.1. Deployment Diagram

Onderstaand deployment diagram, is het physical view uit het SAD. Voor de beslissingen die genomen zijn omtrent dit diagram, zie het [SAD](#). Binnen het deployment diagram is te zien dat via een webserver mensen instaat zijn om het spel te kunnen downloaden. Zodra deze is geïnstalleerd kunnen ze het spel draaien binnen Windows, Linux en macOS. Deze client is dan instaat te communiceren via de netwerk switch met andere clients waarbij hij in de lobby zit.



Figuur 1: Deployment Diagram

3.2. Design Sub-System Action Handling

Dit is het subsysteem dat verantwoordelijk is voor het afhandelen van spelaarsacties.

3.2.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System

Usecase	Naam	Dekking	Waarom niet gedekt?
UC-4	Bewegen van speler	Volledig	-
UC-5	Aanvallen van speler	Volledig	-
UC-6	Inspecteren van Tegel	Volledig	-
UC-12	Versturen van chatbericht	Volledig	-

Functionele requirement	Dekking	Waarom niet gedekt?
FR5	Volledig	-
FR6	Volledig	-
FR20	Volledig	-

Niet functionele Requirement	Dekking	Waarom niet gedekt?
ASR-3	Volledig	-
ASR-4	Volledig	-
ASR-5	Volledig	-
ASR-9	Volledig	-
ASR-21	Volledig	-
ASR-29	Deels	Alle communicatie verloopt via de host, zijn game state is leidend voor alle andere spelers

3.2.2. Gebruikte interfaces

De volgende interfaces worden in dit sub-systeem gebruikt:

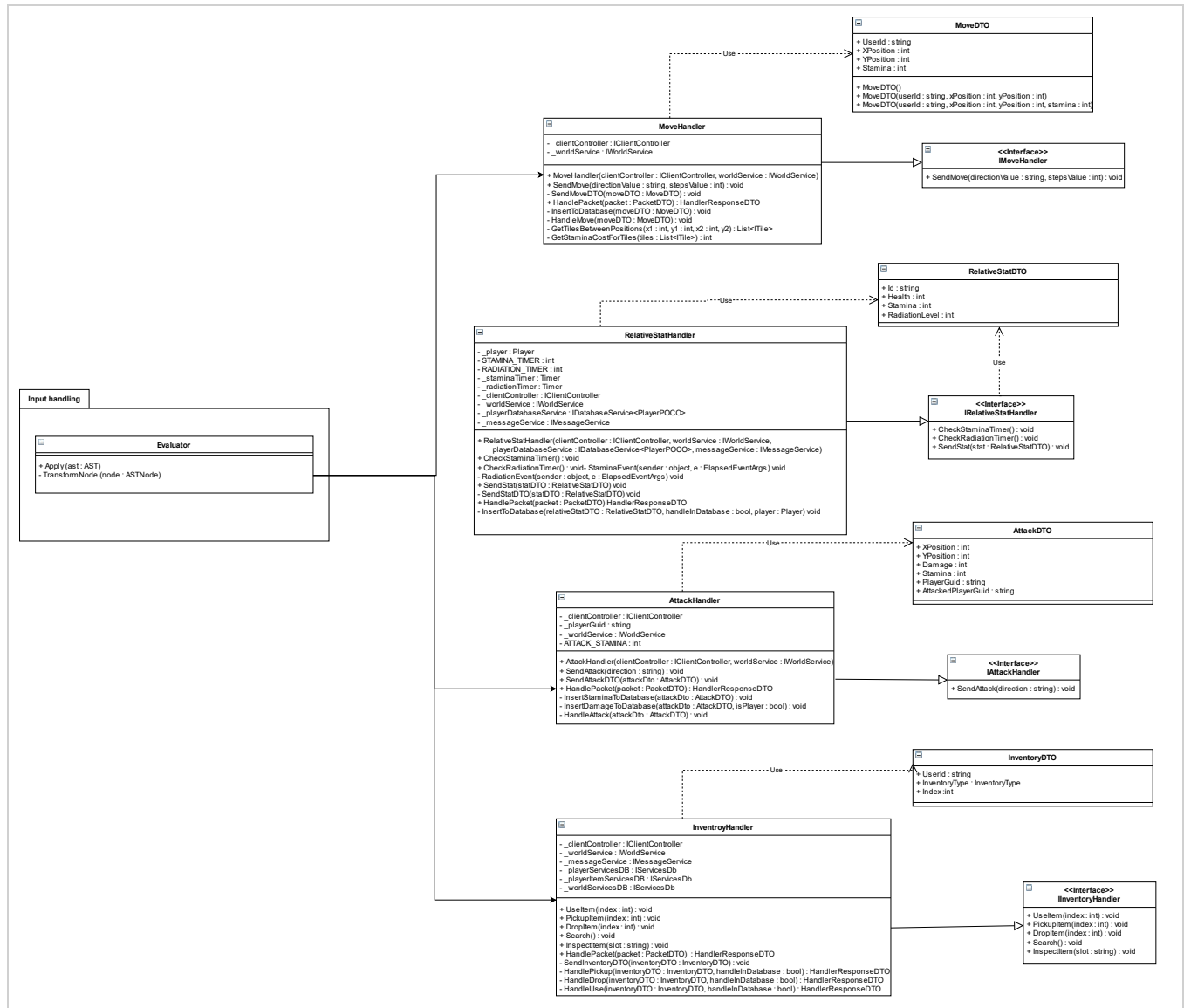
- IMoveHandler
- IRelativeStatHandler
- IAttackHandler
- IInventoryHandler

3.2.3. Algoritme implementatie

Er worden geen speciale algoritmes gebruikt.

3.2.4. Design Class Diagram

Het onderstaande diagram laat zien hoe de structuur van het subsysteem voor het afhandelen van spelersacties eruit ziet.



Figuur 2: Design class diagram sub-system Action Handling

Tabel 1: Glossary

Naam	Beschrijving
MoveDTO	Het DTO voor een beweegactie. Bevat het ID van de speler, de locatie waar de speler naartoe loopt en de stamina van de speler.
MoveHandler	De implementatie van IMoveHandler.
IMoveHandler	De interface voor het uitvoeren en versturen van beweegacties.
RelativeStatDTO	De DTO voor een actie om de statistieken aan te passen. Bevat het ID van de speler en de waarden voor health, stamina en radiation points die bij de speler moeten worden opgeteld.
RelativeStatHandler	De implementatie van IRelativeStatHandler.

Naam	Beschrijving
IRelativeStatHandler	De interface voor het aanpassen en versturen van relatieve veranderingen aan de spelersstatistieken.
AttackDTO	De DTO voor een aanvalsactie. Bevat het ID van de speler, de locatie waar aangevallen wordt, de stamina van de speler, de aangerichte schade en de aangevallen speler.
AttackHandler	De implementatie van IAttackHandler.
IAttackHandler	De interface voor het uitvoeren en versturen van aanvalsacties.
DeadDTO	De DTO waarmee wordt doorgegeven dat een speler dood is.
DeadHandler	De implementatie van IDeadHandler.
IDeadHandler	De interface voor het doorgeven dat een speler dood is.
InventoryDTO	Het DTO voor een actie op het inventory van een player karakter. Bevat het ID van de speler, wat voor soort actie het is en de index voor de actie.
InventoryHandler	De implementatie van IInventoryHandler.
IInventoryHandler	De interface voor het uitvoeren en versturen van acties op het inventory van een spelerkarakter.

3.2.5. Sequence Diagrams

Voor het subsysteem betreft het afhandelen van spelersacties zijn enkele sequence-diagrammen gemaakt die de belangrijkste functionaliteiten verduidelijken.

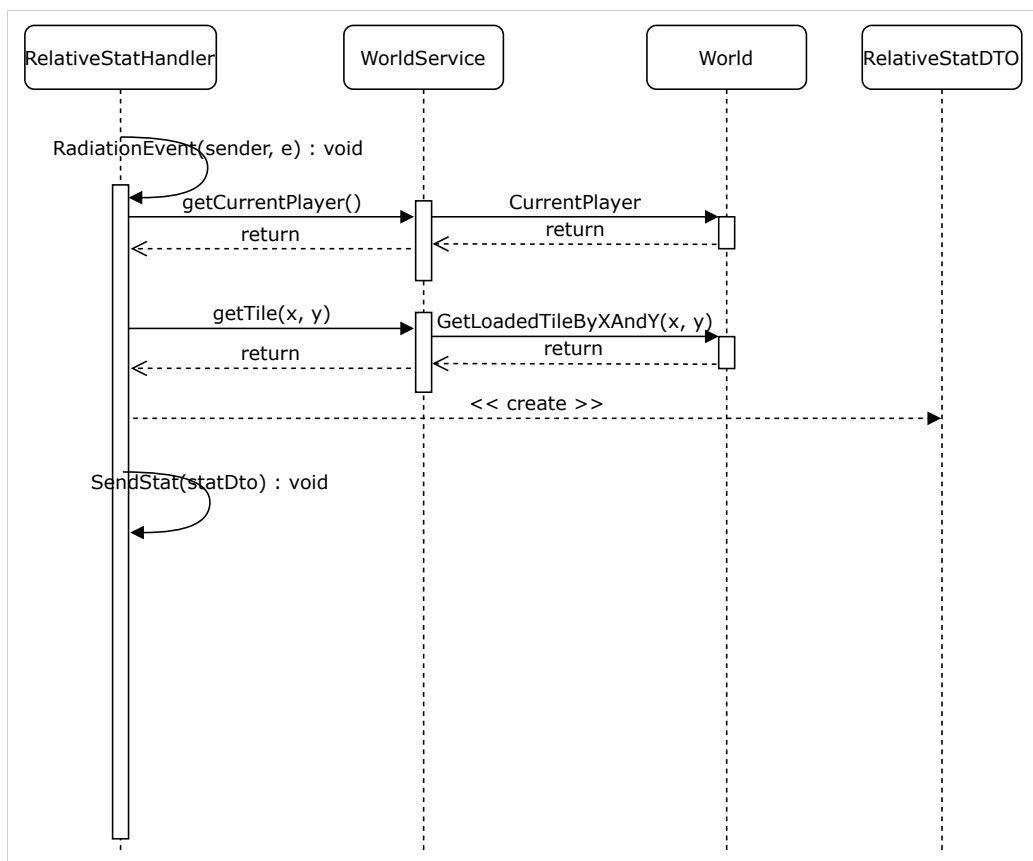


Figure 1:

3.2.6. Activity and State Diagrams

3.2.6.1. Move handling

In het onderstaande diagram is de bewegingsafhandeling te zien bij de host en de clients. De actie begint bij het invoeren van een bewegingscommando met een richting en een aantal stappen. Hierna wordt er eerst gecontroleerd of de speler die de actie uitvoert niet dood is. Vervolgens wordt de tegel berekend waar de speler heen wil bewegen. Hierna wordt de MoveDTO naar de host gestuurd. Deze checkt vervolgens of de tegel al bezet is, berekent de stamina en past de data aan in de database. Als laatst wordt de positie en de stamina aangepast in de speler lijst. De clients volgen hetzelfde proces, ze voegen alleen niks toe aan de database, behalve als hij de back-up-host is.

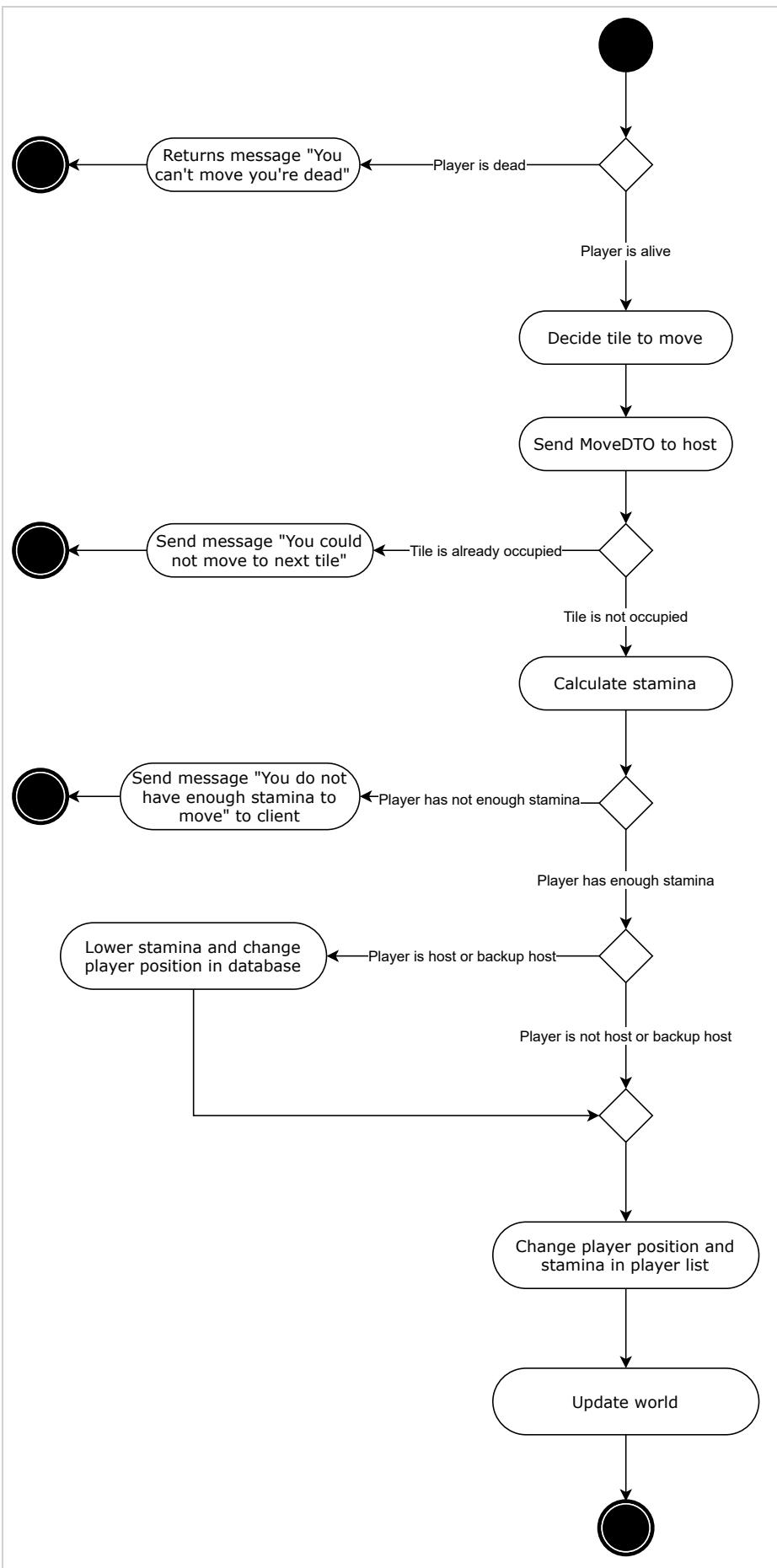


Figure 2: Move handling activity diagram

3.2.6.2. Radiation event

In het onderstaande diagram is beschreven hoe een speler gezondheidspunten en radiatiepunten verliest van gastegels.

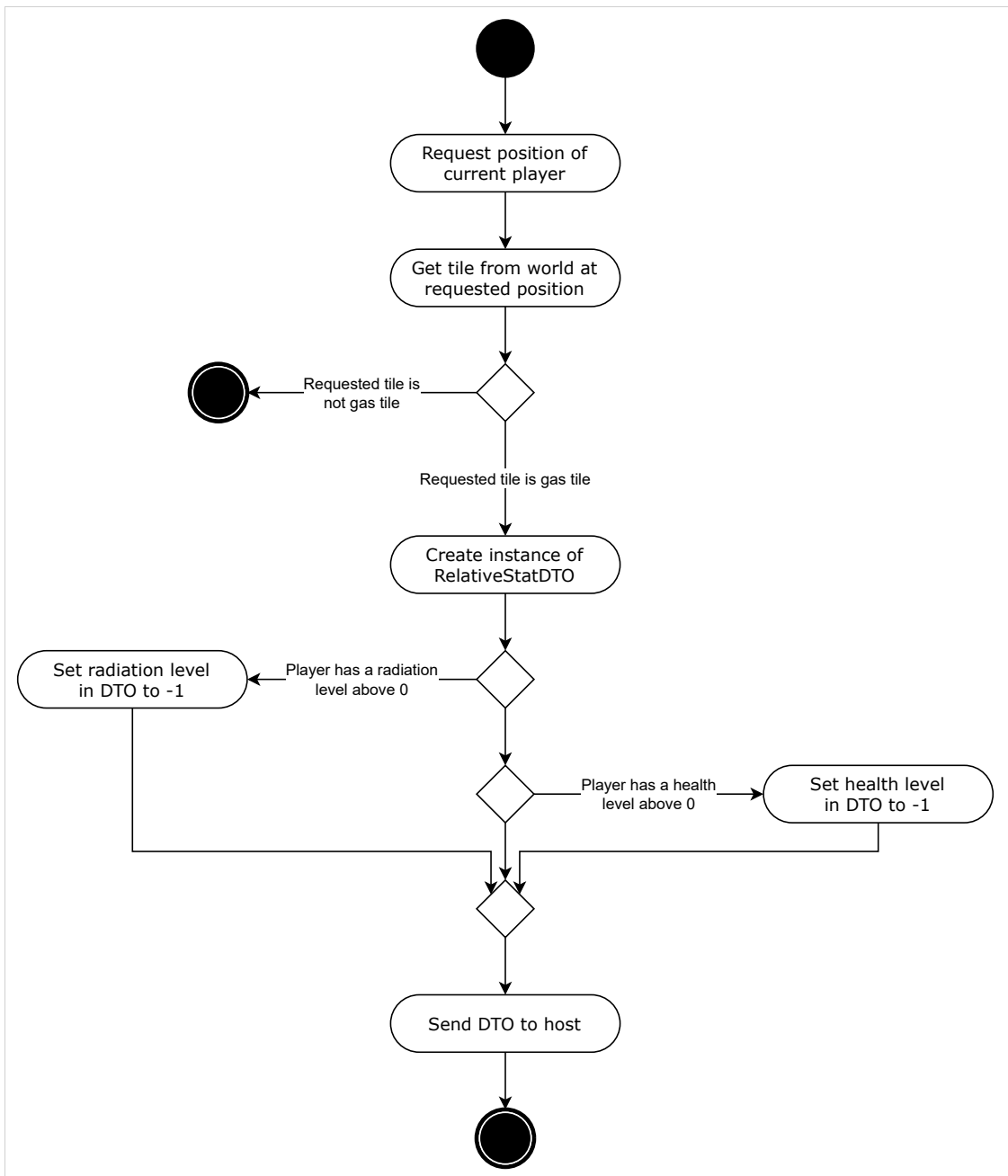


Figure 3:

3.2.6.3. Attack handling

In het onderstaande diagram is de aanvalsfhandeling te zien bij de host. De actie begint bij het invoeren van een attack-commando met een richting. Hierna wordt dit naar de host gestuurd en die checkt vervolgens of de aanval wordt uitgevoerd door een 'levend' persoon. Als dit het geval is dan wordt de range van het wapen opgehaald en de tegel bepaald waar de aanval wordt uitgevoerd. De host checkt vervolgens of er een speler of monster op die tegel staat. Wanneer dit het geval is wordt de stamina van de speler die aanvalt aangepast in de database. Ook wordt de health van de speler die aangevallen wordt verminderd in de database. Wanneer er geen vijand op de tegel staat krijgt de speler een melding en is de aanval afgelopen.

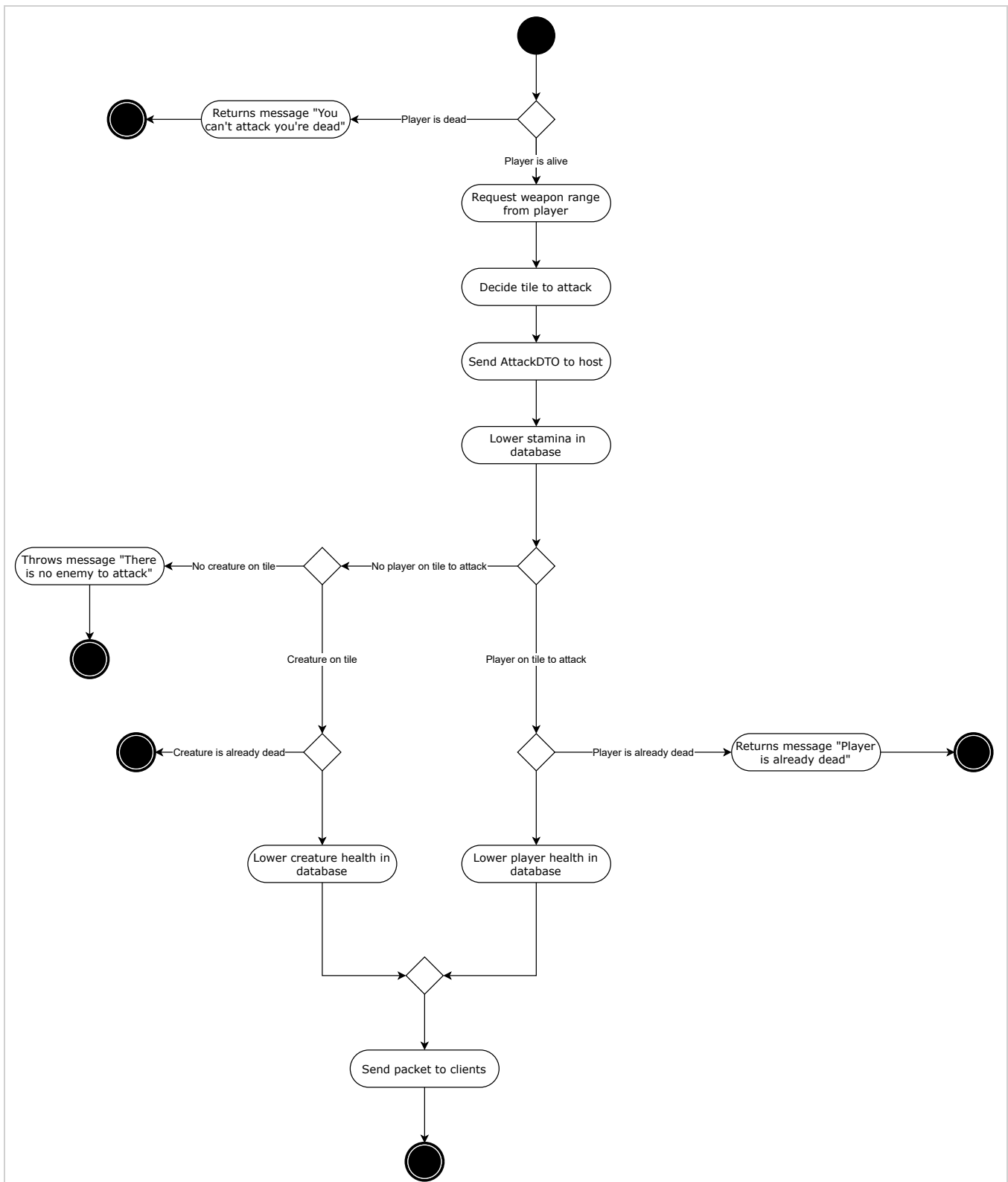


Figure 4: Attack handling activity diagram

In het onderstaande activity diagram is de aanvalsafhandeling te zien bij de client. De actie begint wanneer het verzoek tot een aanvalsactie binnen komt bij alle clients. Iedere client kijkt vervolgens of hij de speler is die aangevallen wordt, als dit het geval is wordt er een bericht getoond in het chatscherf. De speler die de aanval uitvoert krijgt een melding te zien wanneer hij niet genoeg stamina heeft om aan te vallen. Wanneer hij dit wel heeft wordt de stamina van de speler verminderd. Hiernaast verwerken alle spelers de damage op de aangevallen speler. De damage wordt eerst van de helm van de speler afgehaald, hierna van de body armor en als laatste van de health van de speler. Tot slot wordt er gecontroleerd of de speler dood is gegaan, wanneer dit het geval is wordt zijn symbool gewijzigd.

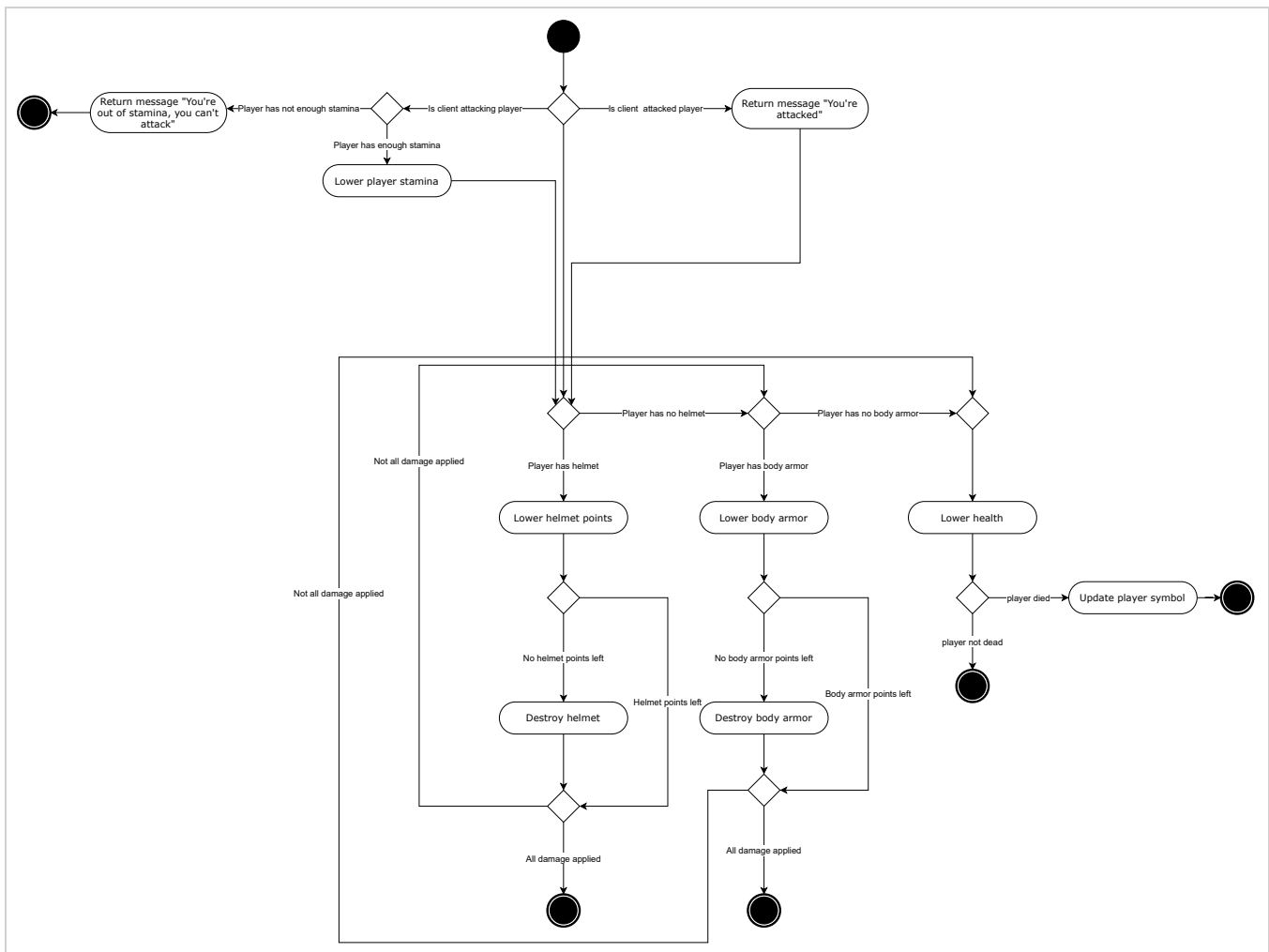


Figure 5: Attack handling cliënt activity diagram

3.2.7. Design decisions made for the sub-system

Hieronder worden beslissingen beschreven die zijn gemaakt voor dit subsysteem.

Tabel 2: Statistieken aanpassen meteen bij client of bij host

Statistieken aanpassen meteen bij client of bij host	Beschrijving
Probleem	Statistieken van speler moeten worden aangepast. Het is mogelijk dat een speler het meteen bij zichzelf aanpast en dan naar de host stuurt, maar is het is ook mogelijk dat de wijziging eerst naar de host wordt verstuurd, dan wordt uitgevoerd en dan teruggestuurd naar de client.
Beslissing	Statistieken worden door de server aangepast.
Alternatieven	Statistieken worden meteen bij de client aangepast.
Argumenten	Er is gekozen om de wijzigingen door de server aan te laten passen. Op deze manier kan de server controleren of de actie wel uitgevoerd kan worden, en dan worden de waardes naar alle spelers, inclusief de huidige speler, gestuurd.

Tabel 3: Waardes van statistieken absoluut of relatief aanpassen

Waardes van statistieken absoluut of relatief aanpassen	Beschrijving
Probleem	De client moet naar de host sturen dat een of meerdere van de statistieken moeten worden aangepast. Dit moet ook goed werken als de statistieken in twee threads tegelijk worden aangepast, omdat er verschillende timers lopen.
Beslissing	Wijzigingen aan de statistieken worden als relatieve waardes meegestuurd. (Als de stamina 50 is en 51 moet worden, dan stuurt de client dat de stamina met één opgehoogd moet worden.)

Waardes van statistieken absoluut of relatief aanpassen	Beschrijving
Alternatieven	Wijzigingen aan de statistieken worden als absolute waardes meegestuurd. (Als de stamina 50 is en 51 moet worden, dan stuurt de client dat de stamina moet worden verandert naar 51.)
Argumenten	Er is gekozen om met relatieve waardes te werken omdat dit minder problemen geeft betreft het tegelijk aanpassen van de statistieken in verschillende threads.

Tabel 4: Schade eerst aan body armor of eerst aan helm

Schade eerst aan body armor of eerst aan helm	Beschrijving
Probleem	Hoe wordt de damage op armor en health verdeeld.
Beslissing	Damage wordt eerst van de helm van de speler gehaald en daarna van de body armor.
Alternatieven	Damage wordt eerst van de body armor van de speler gehaald en daarna van de helm.
Argumenten	Er is hiervoor gekozen omdat de armor-voorwerpen los van elkaar opgepakt kunnen worden en een speler altijd begint met een helm.

3.3. Design Sub-System AI (Parser/Lexer)

Dit is het subsysteem dat verantwoordelijk is voor het compileren van de agent en NPC. Het doet dit op basis van een text file die de gebruiker heeft aangemaakt en compileert dit naar een config file voor een ander subsysteem

3.3.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System

Tabel 5: Sub systeem AI parser lexer - Use cases

Usecase	Naam	Dekking	Waarom niet gedekt?
UC - 1	Configureren van Agent	Volledig	-

Tabel 6: Sub systeem AI parser - Functionele Requirements

Functionele Requirement	Dekking	Waarom niet gedekt?
FR14	Volledig	-

Tabel 7: Sub systeem AI parser - Niet Functionele Requirements

Niet Functionele Requirement	Dekking	Waarom niet gedekt?
ASR-12	Volledig	-
ASR-13	Deel	Onderzoek team 1 geeft aan wat het simpel maakt, toch is dit per persoon verschillend.
ASR-21	Deel	3 checks worden uitgevoerd op zowel of basis elementen bevat, door de parser zelf, en dan nog door een eigen checker die onder ander combinaties checkt. Het kan toch voorkomen dat regels door de checkers heen komen die niet valide zijn.

3.3.2. Interface overview

In dit hoofdstuk zijn alle interfaces beschreven die behoren tot het agent en npc component.

Table 1: Glossary

Naam	Beschrijving
IPipeLine	Deze interface zorgt ervoor dat PipeLine gebruikt kan worden en dat de nodige methodes worden geïmplementeerd.
Node	Het super type voor elke node-soort die gebruikt wordt in de Parse-boom

3.3.3. Algoritme Implementatie

De agent heeft geen direct algoritme die wordt aangesproken, behalve die door 'Antlr' gespecificeerd is en door ons in ingevuld.

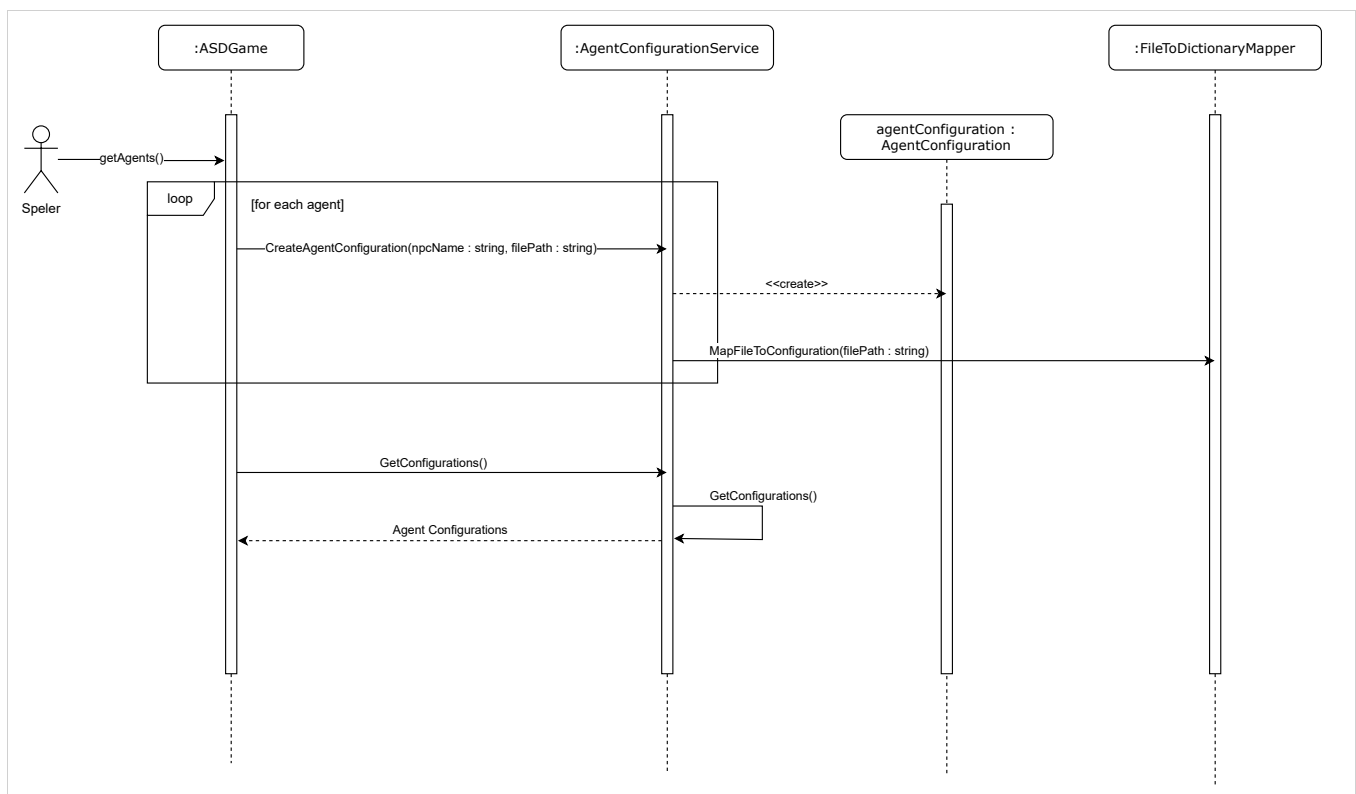
3.3.4. Design Class Diagram

Onderstaande diagram beeld het diagram van de Agent compiler uit. Hierbinnen is te zien hoe de compiler de pipeline, generator en checker eruit zien. Daarnaast is te zien hoe de configuratie service eruit ziet. Deze configuratie maakt verschil tussen de NPC en agent. Om de files van de gebruiker te importeren hebben wij een filehandler class opgesteld. Ook hebben we een FileToDictionaryMapper gemaakt die een dictionary aanmaakt voor het creature component.

Naam	Beschrijving
PipeLine	Deze class regelt de verschillende componenten nodig om een volledig bestand te converteren naar een uitvoerbare code
AST	Deze class waarin de tree opgebouwd wordt.
ASTAgentListener	Deze class maakt de AST tree.
Checking	Deze class checkt de semantiek van de AST tree.
Node	De Super van de AST - nodes waarmee de AST tree opgebouwd wordt.
Configuration (Ast)	Een subclass van Node.
Generating	Deze class zet de AST tree om naar verschillende code lijnen.
BaseConfigurationService	De superklasse van de ConfigurationService klassen. Deze klasse wordt door NpcConfigurationService en AgentConfigurationService geërfd.
NpcConfigurationService	Een service-klasse om NPC configuraties aan te maken en uit te lezen.
AgentConfigurationService	Een service-klasse om Agent configuraties aan te maken en uit te lezen.
Configuration (Models)	Een superklasse waar de configuratiesoorten van overerven. Wordt overgeërfd door NpcConfiguration en AgentConfiguration.
NpcConfiguration	Een klasse om de settings van een NPC soort bij te houden.
AgentConfiguration	Een klasse om de settings van een Agent bij te houden.
FileHandler	Een klasse om (configuratie) bestanden/mappen aan te maken, uit te lezen of wijzigen.
FileToDictionaryMapper	Een klasse met een enkele methode om Agent/Npc configuratie bestanden om te zetten tot een dictionary.
ConsoleRetriever	Een klasse om de console input mee uit te lezen, gemaakt voor de testbaarheid (met name om console input te mocken).

3.3.5. Sequence Diagrams

Dit onderdeel bevat de Sequence Diagrammen van het configureren en ophalen van configuraties van Agents. Omdat het proces van het configureren en ophalen van configuraties voor NPC's vrijwel hetzelfde is, wordt dit niet in dit onderdeel opgenomen en wordt de AgentConfigurationService als uitgangspunt genomen.

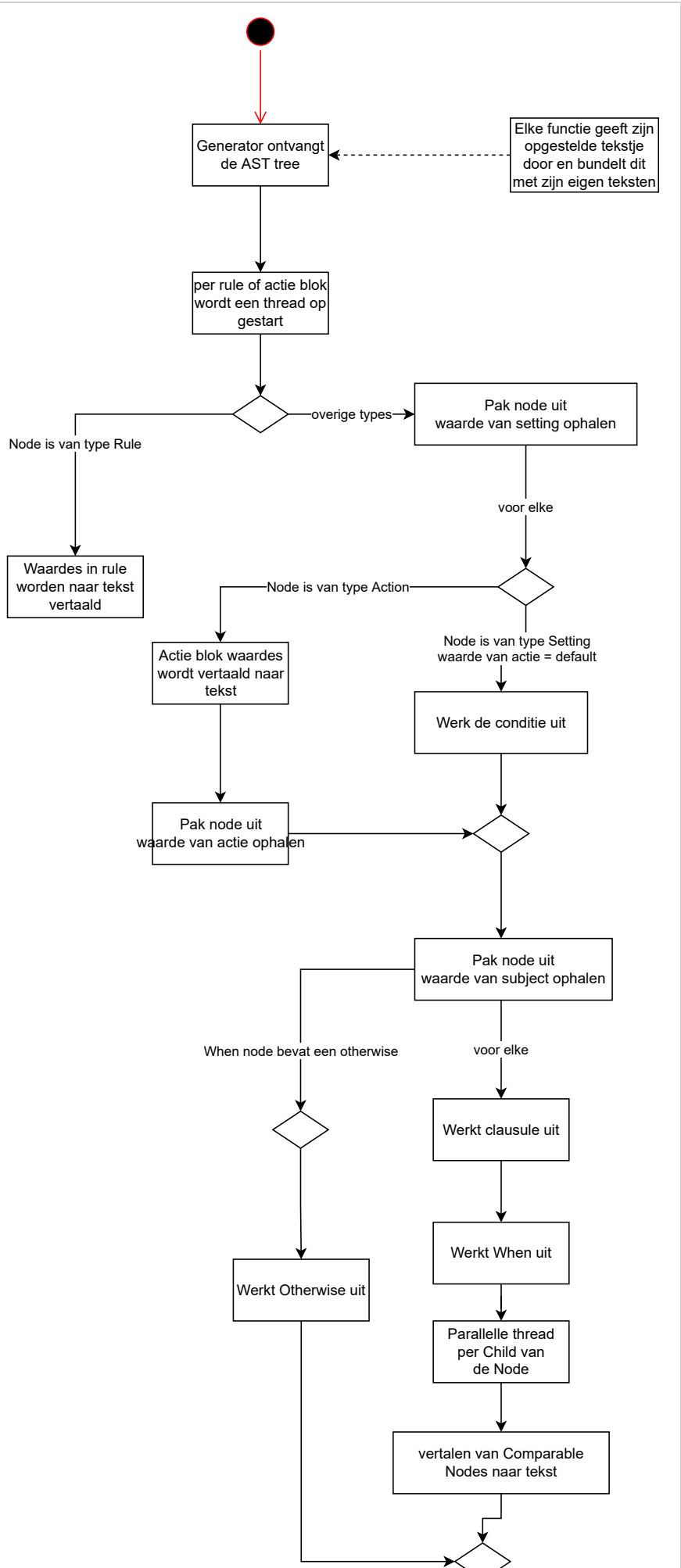


Figuur 4: Sequence diagram voor AgentConfigurationService

3.3.6. Activity and State Diagrams

In dit hoofdstuk staan de activity en state diagrammen. Deze geven een overzicht over de flow van het systeem.

3.3.6.1. Activity Diagram Generating



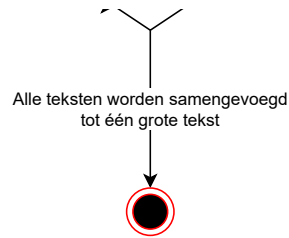
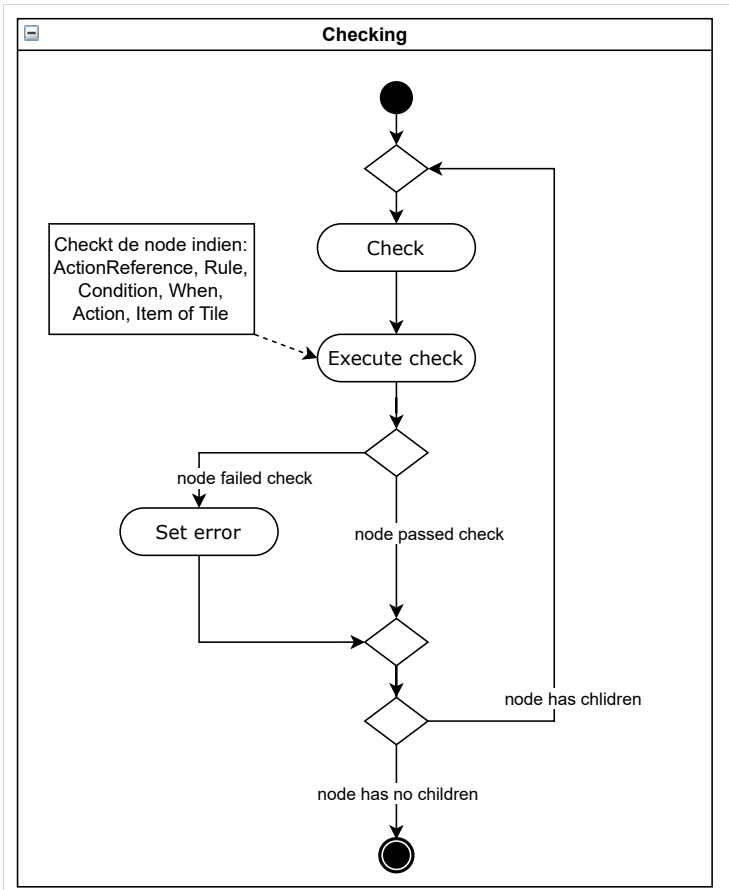


Figure 6: Sequence diagram - Generator

3.3.6.2. Activity Diagram Checking



Figuur 5: Activity Diagram Checking

3.3.7. Design decisions made for the sub-system

Table 2: Design Decisions

Decision	Description
Problem/Issue	Er moet onderscheid gemaakt worden tussen de configuraties van een Agent en NPC's, in de toekomst kunnen de soort configureerbare regels hiervoor verschillen en kan er koppeling ontstaan naar bepaalde NPC klassen.
Decision	Zoals in Figuur 1 (Design Sub-System AI (Parser/Lexer)) is te zien, is er besloten om twee klassen ' <i>AgentConfiguration</i> ' en ' <i>NpcConfiguration</i> ' te maken die van een super klasse ' <i>Configuration</i> ' erven. Ook is er een Service gemaakt voor iedere soort configuratie (' <i>AgentConfigurationService</i> ' en ' <i>NpcConfigurationService</i> '), die van een abstracte klasse ' <i>BaseConfigurationService</i> ' erft.
Alternatives	De logica kan in een enkele klasse geschreven worden, waarbij er onderscheid wordt gemaakt door middel van een ConfigurationType enum klasse.

Decision	Description
Arguments	<ul style="list-style-type: none">Met overerving is het beste onderscheid te maken, het is duidelijker van welk voor soort entiteit (Agent of Npc) een configuratie wordt gebruikt.De logica van het configureren van een Agent en een Npc is verschillend (zie Software Requirements Specification), deze logica wordt in de construction fase nog uiteenlopend. Zo wordt een Agent via een programmeerscherm geconfigureerd en een NPC via een configuratiescherm van een (lopend) spel. Door deze logica nu al te scheiden is er enige voorbereiding getroffen om deze logica op verschillende plaatsen uit te werken.Code duplicatie is voor een groot deel afgevangen door het af te splitsen naar externe klassen (zoals FileHandler en FileToDictionaryMapper).

3.4. Design Sub-System AI (Monsters/bots/NPC's)

In dit hoofdstuk wordt de detailed design rondom de AI van de Monsters, Bots, NPC's en Agents beschreven. De AI maakt gebruik van de [Appccelerate](#) library voor de implementatie van de statemachine. In de onderstaande tabel staat beschreven welke interfaces er gebruikt worden bij de implementatie van AI.

Interface	Doel
IMoveHandler	Wordt gebruikt om AI te laten bewegen.
IWorldService	Wordt gebruikt om alle informatie over de wereld op te halen.
IAttackHandler	Wordt gebruikt om de AI aan te laten vallen.

Tabel 8: Interfaces die gebruikt worden

3.4.1. Pathfinder Algoritme

Het pathfinder systeem voor de AI is gebaseerd op het [A* algoritme](#). We hebben voor dit algoritme gekozen omdat deze als beste optie uit het [onderzoek pathfinding](#) kwam. In dit onderzoek hebben we gekeken naar de volgende alternatieven:

- Dijkstra's algoritme
- A* Search algoritme
- D*

In de tabel hieronder is te zien welke forces we overwogen hebben in het maken van onze keuze.

				Pathfinding algoritme		
				<div>DISCARDED</div>	<div>DECIDED</div>	<div>DISCARDED</div>
				Dijkstra	A* Search	D*
ASR forces						
Code	Description	Concern(s)	Stakeholder	--/+/++	--/+/++	--/+/++
C-10	Pad moet binnen 100 ms berekend zijn	Time behaviour	SH-1	+	++	++
C-10	Dynamisch pad berekenen	Functional appropriateness	SH-1	--	-	++
C-10	Toepasbaarheid in games	Functional appropriateness	SH-2	+	++	?
Other forces						
OF-1	Complexiteit in implementeerbaarheid	Development time	SH-2	++	+	--
Total						

(totaal aantal +) - (totaal aantal -) =				2	4	2
--	--	--	--	---	---	---

Tabel 9: Decision forces pathfining algoritme

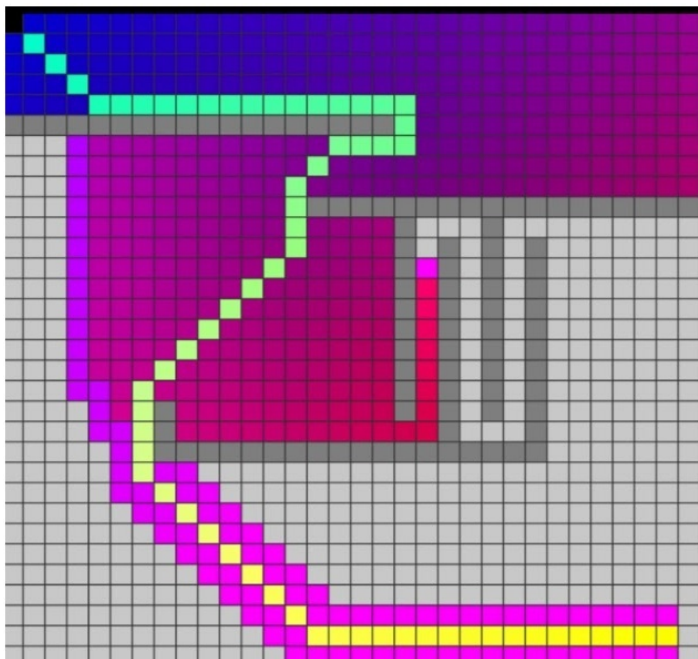
3.4.1.1. Werking A* algoritme

In de afbeelding hieronder is de werking van een A* algoritme te zien.

De groen/gele blokken representeren het pad dat het algoritme als optimaal ziet.

De grijze blokken zijn delen van de map waar het algoritme nooit naar heeft gekeken.

De andere gekleurde blokken zijn de opties die het algoritme heeft overwogen voor het pad maar die minder efficiënt waren.



Figuur 6: Werking A*

3.4.2. MachineLearning Algoritme

Voor de implementatie van machine learning in ons AI systeem is er gebruik gemaakt van een combinatie van twee algoritmes:

- [neurologisch netwerk](#)
- [genetisch algoritme](#)

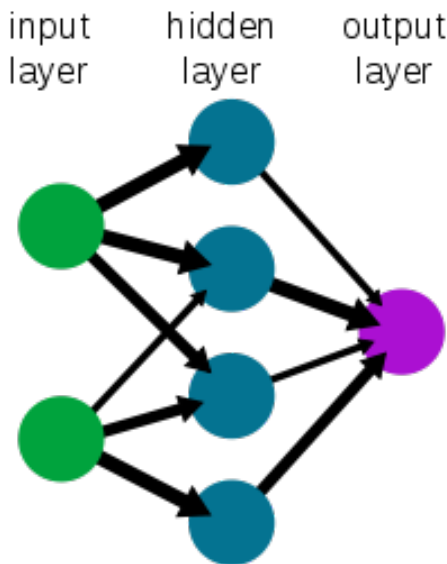
Er is hiervoor gekozen omdat de combinatie van deze twee algoritmes vaak werden gebruikt in spellen. Omdat er geen ervaring was met machine learning algoritmes is er voor gekozen om geen tijd te besteden aan het onderzoeken van de alternatieven. [Onderzoek machinelearning in AI](#)

Het neurologische netwerk zorgt ervoor dat een AI op basis van een set aan input waardes een actie kan uitvoeren en doormiddel van een berekening kijken hoe goed of slecht deze actie was. Deze informatie gebuikt hij dan om connecties te maken tussen de input waarde en een actie om zo te leren welke actie het best past bij een situatie.

Het genetische algoritme word gebruikt om het gedrag van een AI vast te leggen en door te geven. Dit maakt het makkelijker om het geleerde gedrag te gebruiken en aan te passen.

3.4.2.1. Werking neurologisch netwerk

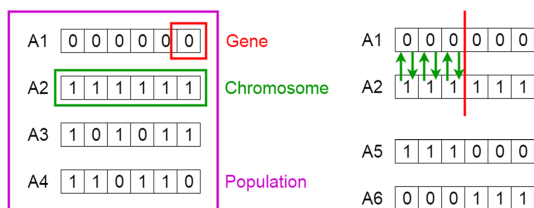
A simple neural network



Figuur 7: Werking neurologisch netwerk

3.4.2.2. Werking Genetisch algoritme

Genetic Algorithms



Figuur 8: Werking genetisch algoritme

3.4.3. Design Class Diagram

In het onderstaande class diagram staat het ontwerp van het character component, waarin de logica achter het gedrag van alle AI character centraal staat.

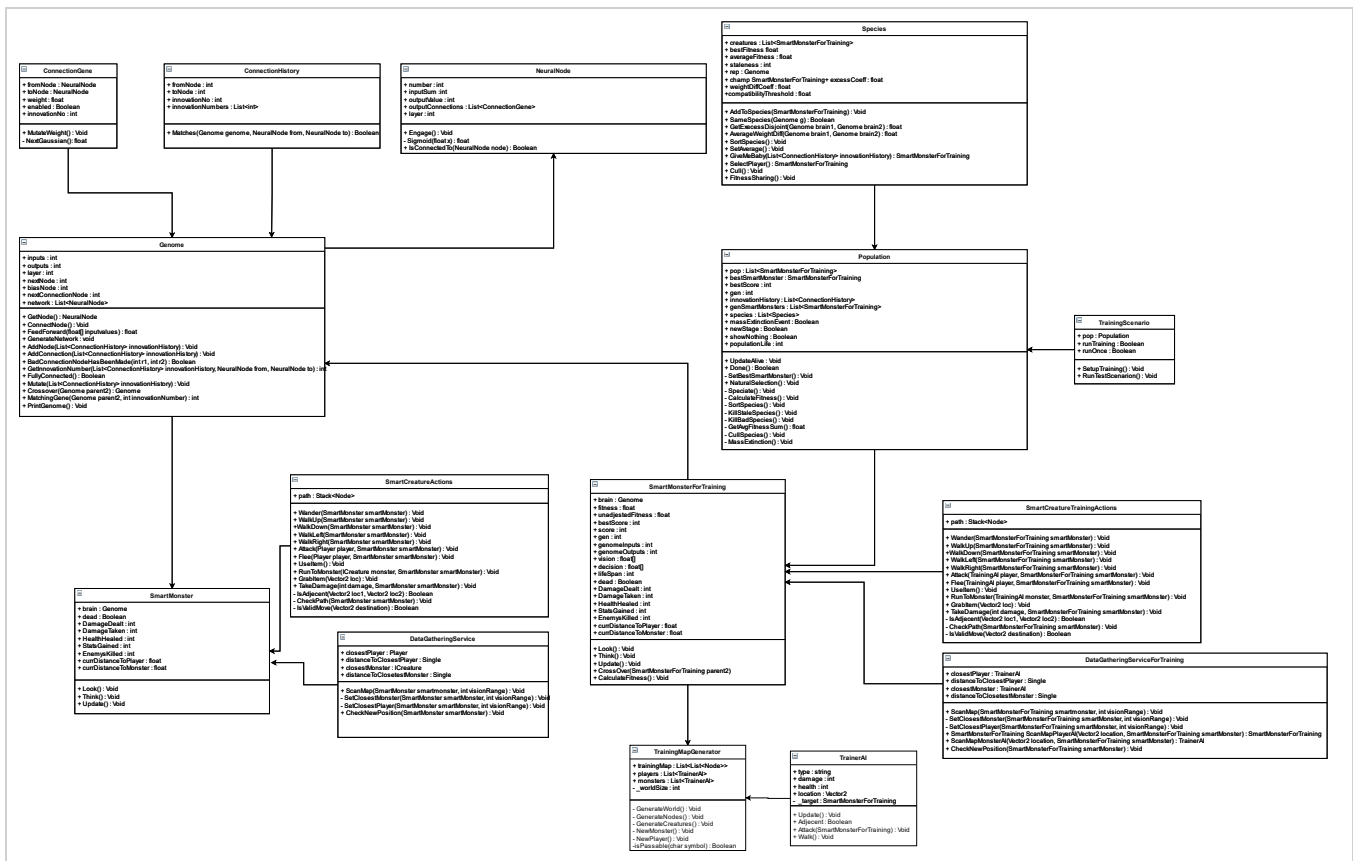
Onder de class character vallen Agent en Monster. Elke AI character staat om pathfinding mogelijk te maken op een Node, waarbij de Pathfinder class een Node grid gebruikt om het A* Search algoritme op toe te passen. De verantwoording voor het gebruik van het A* Search algoritme is te vinden in [Onderzoek Pathfinding](#).

Character gebruikt ook een ICharacterStateMachine. Deze statemachine class gebruikt de Appccelerate library om bij CharacterEvents die "fired" worden de juiste target CharacterState uit te voeren. De verantwoording voor het gebruik van de Appccelerate library is te vinden in hoofdstuk 6.3.3. van het [Onderzoek Wat is AI](#).

ICharacterStateMachine gebruikt ICharacterData, waarin de data van een character zit, zoals een lijst van zelf-instelbare KeyValuePair settings. Deze settings worden opgehaald uit de ConfigurationService class uit het Agent component. De lijst van settings wordt omgezet in een RuleSet lijst door een RuleSetFactory, om het gedrag leesbaarder te maken in de BuilderConfigurator. ICharacterStateMachine gebruikt de BuilderConfigurator om de zelf-ingestelde transities en omstandigheden te bepalen die nodig zijn om een CharacterState uit te voeren.

Verder is de aansluiting met neural networking in het class diagram zichtbaar. De opbouw van het neural network is gebaseerd op [Onderzoek Machine learning in AI](#).

Figuur 9: Design Class Diagram - AI



In onderstaande tabel is een glossary zichtbaar waarin de neural networking classes uit het bovenstaande class diagram toegelicht worden.

Naam	Beschrijving
SmartMonster	Een Monster voor implementatie in het spel wereld die een getrainde Genome overneemt om zijn gedrag te bepalen.
SmartCreateActions	De acties die kunnen worden uitgevoerd door een SmartMonster.
DataGatheringService	Een set aan functies om situatie informatie op te halen voor de data-input van een Smartmonster
Genome	Een collectie van NeuralNodes en ConnectionNodes om de input naar een correcte output om te zetten
ConnectionGene	Een Gene die 2 NeuralNodes met elkaar verbindt om de input waardes door te geven.
ConnectionHistory	Een collectie aan alle connecties die eerder zijn aangemaakt.
NeuralNode	Een informatie punt om input naar uitput te krijgen.
Species	Een of meerdere collecties aan Genes om soorten binnen de AI te scheiden op basis van gedrag.
Population	Een collectie aan SmartMonsters voor het bijhouden en uitvoeren van het leerproces.
TrainingScenario	Een aanroep van een loop om een trainingssessie te starten.
SmartMonsterForTraining	Een Smartmonster voor gebruik in de trainingsscenario's.
TrainingMapGenerator	Een generatie klassen voor het aanmaken van de map en entiteiten voor het trainingsscenario
TrainerAI	Een simpele AI voor het trainen van het neurologische netwerk.
DatagatheringServiceForTraining	Een set aan functies om input data te halen uit een trainingsscenario.
SmartCreatureTrainingAction	Een set aan acties die kunnen worden uitgevoerd door SmartmonstersForTraining.

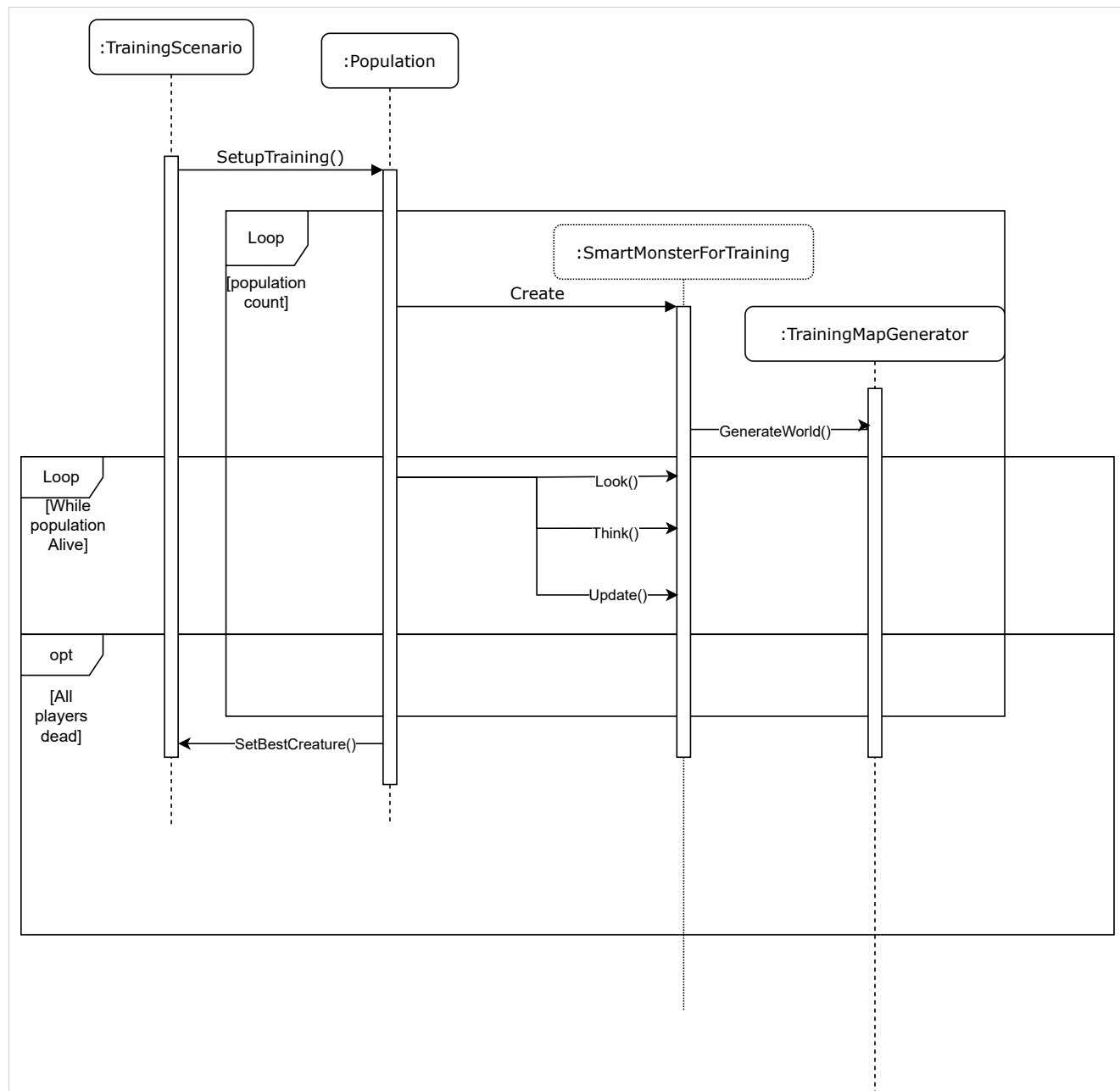
Tabel 11: Klassen glossary NeuralNetwork

3.4.4. Sequence Diagram

In dit sequence diagram word weergegeven hoe een trainingsscenario procesmatig werkt.

Als eerste word er een training opgezet om alle waardes voor de AI characters die getraind moeten worden in te stellen en om de totale populatie(aantal aan AI character) in te stellen. De training scenario blijft doorgaan tot alle AI character in een populatie dood zijn of er geen

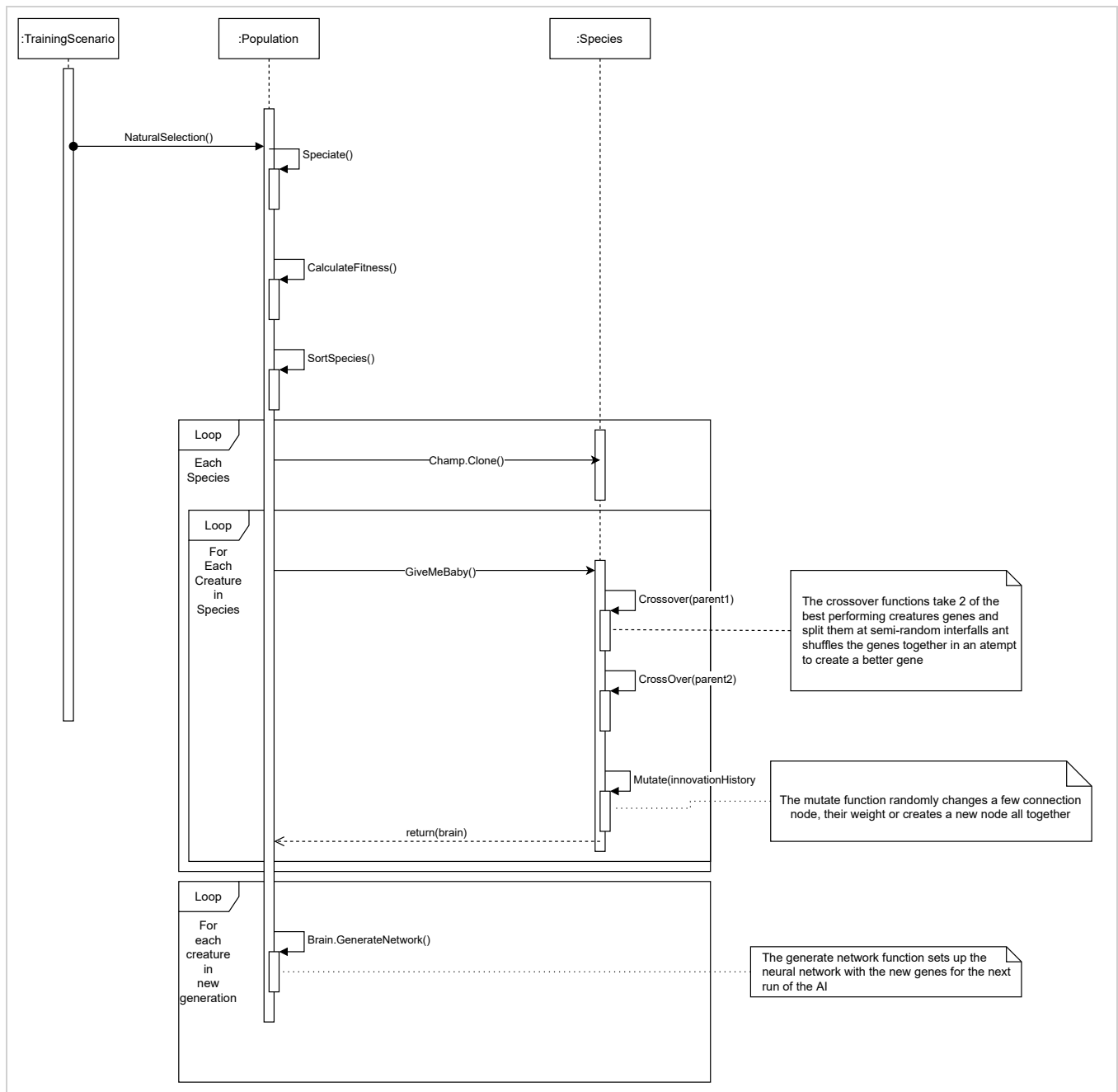
vijanden meer zijn om van te leren. Zodra de populatie dood is word het AI character met de hoogste score bewaard om later door te kunnen geven aan de AI binnen de spelwereld.



Figuur 11: Training Scenario

In dit sequence diagram word weergegeven hoe het automatische verbeteringsproces van het neurologische netwerk werkt.

De species zijn bedoeld voor het onderscheiden van de verschillende speelstijlen binnen een spel zodat er meer mogelijkheden kunnen worden onderzocht door het netwerk om tot hogere speel scores(fitness) te komen.

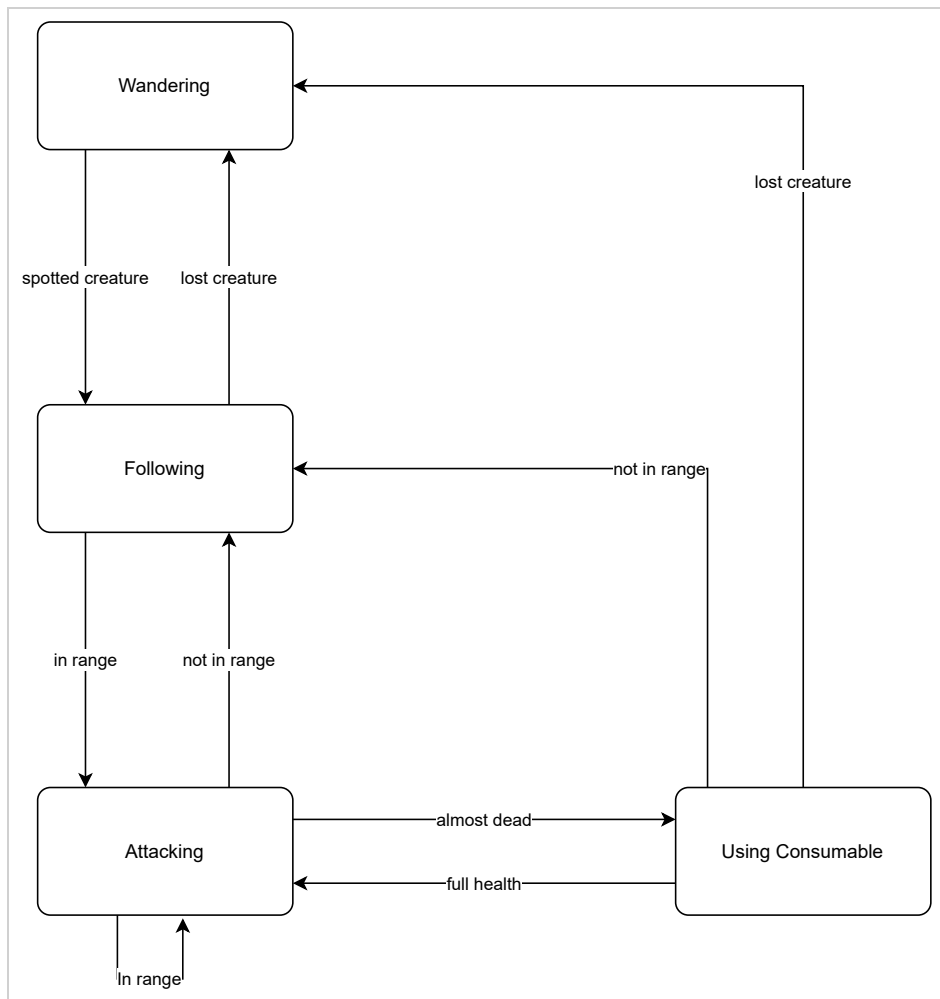


Figuur 12: NaturalSelection process

3.4.5. State Diagram

In dit deelhoofdstuk staan de activity diagrams beschreven van het spel.

In het onderstaande diagram is zichtbaar hoe de transitie van de states verloopt onder bepaalde events.



Figuur 13: State diagram "NPC AI"

3.4.6. Design decisions made for the sub-system

Decision	Description
Problem/Issue	Een AI moet slim kunnen bewegen.
Decision	Er is gekozen voor het A* pathfinding algoritme met de Manhattan distance voor afstandsrekening.
Alternatives	Het D* Lite algoritme.
Arguments	Zoals te zien in hoofdstuk 5.1.4 Deelconclusie van het Onderzoek Pathfinding is er gekozen voor de A* methode omdat dit makkelijker is om te implementeren en omdat het dynamische aspect van het D* algoritme geen meerwaarde voor dit project levert.

Decision	Description
Problem/Issue	De speler moet een AI kunnen instellen.
Decision	Er is gekozen om het gedrag van een AI te regelen via een ruleset.
Alternatives	Dat de speler een statemachine met bijbehorende states levert.
Arguments	Er is gekozen voor de ruleset omdat dit makkelijker te genereren is vanuit een parser.

Decision	Description
Problem/Issue	De statemachine accepteert alleen states en events die 'Comparable' implementeren.
Decision	Er is gekozen om de abstract classes 'CreatureState' en 'CreatureEvent' toe te voegen die 'Comparable' implementeren.

Decision	Description
Alternatives	Iedere state en event 'Comparable' laten implementeren.
Arguments	Het is nu niet nodig voor iedere state en event om 'Comparable' te implementeren.

3.5. Design Sub-System Input Handling

3.5.1. Interfaces:

Het input handling sub-system maakt gebruik van de volgende interfaces:

- IInputHandler
- IPipeline (InputHandler versie)
- ISessionHandler
- IScreenHandler
- IMessageService
- IGameConfigurationHandler

3.5.2. Functionele en niet-functionele requirements die gedekt worden door het Sub-System

De functionele eisen in [Tabel 1 \(Design Sub-System Network Switch\)](#) en [Tabel 2 \(Design Sub-System Network Switch\)](#) komen uit het [Software Requirement Specification](#) document.

Tabel 12: InputHandler usecases

Usecase	Dekking	Waarom niet gedekt?
Usecase: Configureren agent	VOLLEDIG	
Usecase: Aanmaken lobby	VOLLEDIG	
Usecase: Deelnemen lobby	VOLLEDIG	
Usecase: Configureren van een spel door middel van het menu	VOLLEDIG	
Usecase: In-game configuratie aanpassen	VOLLEDIG	

Tabel 13: Tabel InputHandlerl functionele eisen

Eis code	Dekking	Extra opmerkingen
FR25	VOLLEDIG	
FR26	DEELS	
FR27	VOLLEDIG	basis check is in de inputhandler, de precieze checks worden gedaan in de checker van de pipeline

De niet-functionele eisen in [Tabel 3 \(Design Sub-System Network Switch\)](#) komen uit het [Software Architecture Document](#).

Tabel 14: Inputhandler niet-functionele eisen

Eis code	Dekking	Extra opmerkingen
ASR-11	VOLLEDIG	
ASR-12	VOLLEDIG	
ASR-13	VOLLEDIG	

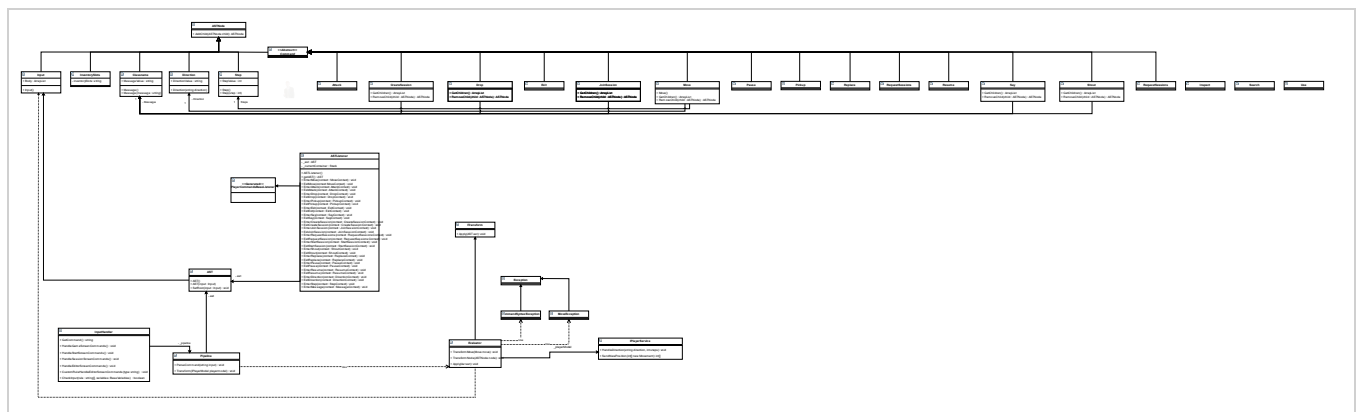
Eis code	Dekking	Extra opmerkingen
ASR-21	DEELS	bij input van tekst bij nummer kan in enkele geval een crash optreden

3.5.3. Algoritme Implementatie

De inpuhandler bevat geen bijzondere algoritmes. Ook hier wordt gebruik gemaakt van 'Antlr' om het gedrag te vertalen naar acties. Het algoritme geïmplementeerd door 'Antlr' is alleen door ons ingevuld.

3.5.4. Design Class Diagram

In onderstaande afbeelding is het design class diagram te zien voor het commando sub-systeem. Een speler geeft aan de inpuhandler aan wat de commando is die de speler wilt uitvoeren. De inpuhandler geeft dit door aan de parser, waarna na een aantal stappen de commando van de speler is verwerkt. In sequence diagram x staat een voorbeeld hoe dit gedaan gaat worden. De voorbeeld is gemaakt met het commando move, de andere commando's werken het zelfde.



Figuur 14: Design Class Diagram Commando

In onderstaande tabel is een glossary te zien van bovenstaande klassendiagram.

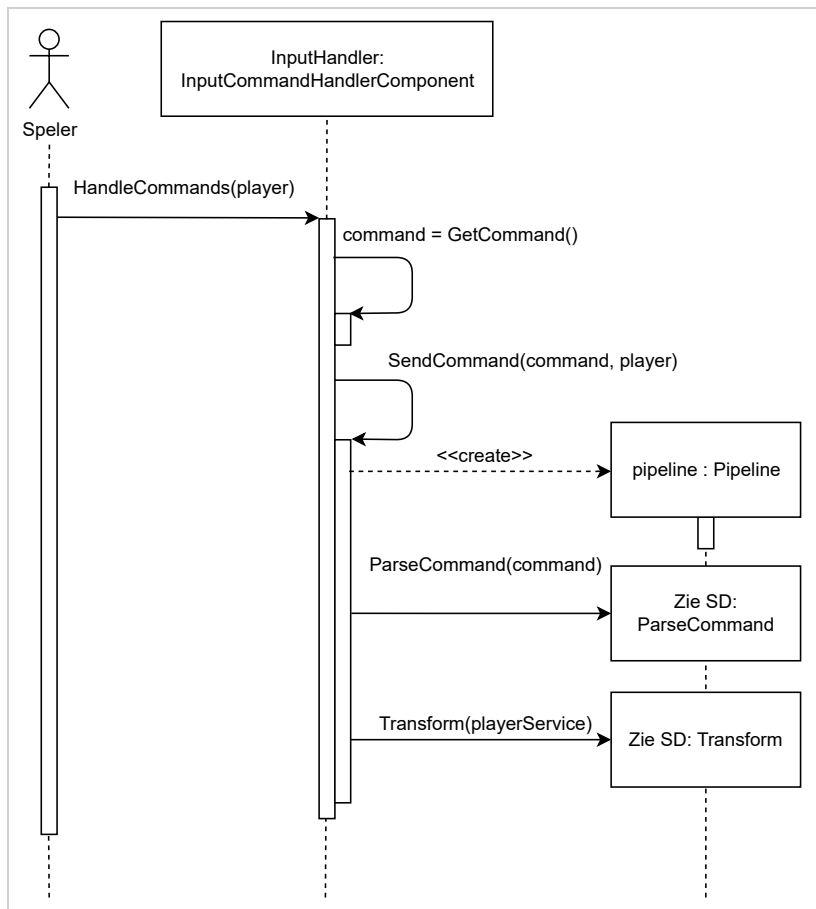
Tabel 15: Design Class Diagram Commando Glossary

Naam	Beschrijving	
AST	De syntaxisboom waar alle nodes in hangen.	
ASTNode	Een onderdeel van de syntaxisboom die opgebouwd wordt.	
Direction	Een node voor aangeven van een richting.	
Input	Een node voor de invoer.	
Message	Een node voor een bericht.	
Step	Node voor het aangeven van hoeveel stappen een speler wilt doen.	
Command	Attack	Commando voor het aanvallen van een speler.
	Drop	Commando voor het droppen van een item.
	Exit	Commando voor het verlaten van een spel.
	Move	Commando voor het bewegen van een speler.
	Pause	Commando voor het pauzeren van het spel.

	Pickup	Commando voor het oppakken van een item.
	Replace	Commando voor het vervangen van een speler door zijn agent en vice versa.
	Resume	Commando voor het hervatten van het spel wanneer deze gepauzeerd is.
	Say	Commando voor het versturen van een tekstbericht naar de teamchat.
	Shout	Commando voor het versturen van een een tekstbericht naar de globale chat.
	Search	Commando voor het opvragen van welke items op de tegel liggen.
	Inspect	Commando voor het opvragen van de statistieken van voorwerpen die een speler in de inventory heeft.
	MonsterDifficulty	Node voor het aangeven wat de moeilijkheidsgraad is van NPC's in de game.
	ItemFrequency	Node voor het aangeven wat de frequentie is van het spawnen van voorwerpen in het spel.
ASTListener	Voor elk ASTNode is een enter en exit methode aangemaakt. Om klassendiagram overzichtelijk te houden zijn maar een aantal enter- en exit-methodes toegevoegd.	
Evaluator	Zorgt ervoor dat de commando's de juiste functies aanroepen.	
ITransform	De interface om commando's te transformeren.	
Pipeline	Krijgt de commando's binnen van InPutHandlerComponent. Maakt hier AST's van.	
Exceptions	CommandSyntaxException	Exceptie voor syntaxisfouten tijdens het typen.
	MoveException	Exceptie voor als je te veel of te weinig stappen zet.
	SlotException	Exceptie voor als je een slot op geeft wat niet valide is.
InputCommandHandlerComponent	Hier komen de commando's binnen en worden ze doorgestuurd naar de pipeline.	

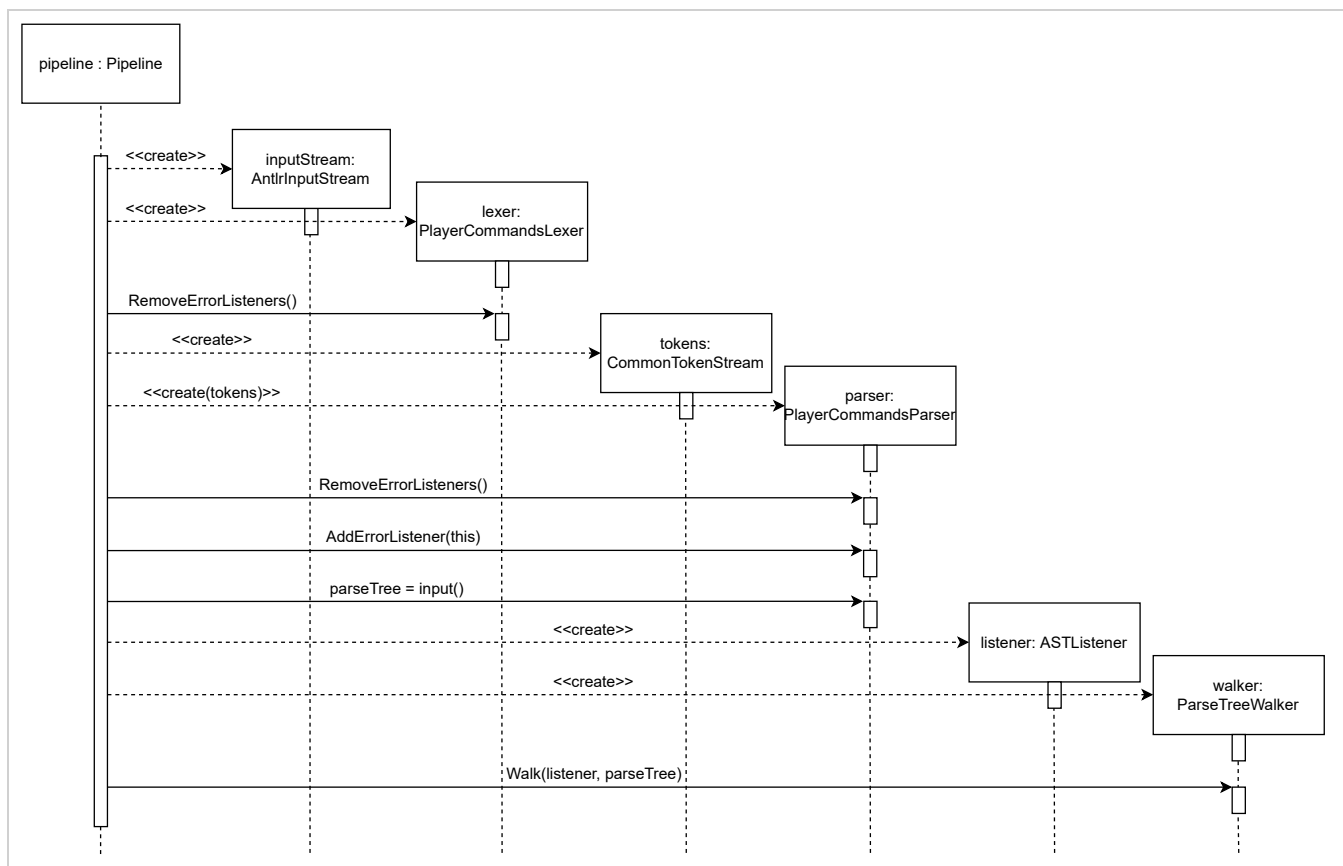
3.5.5. Sequence Diagrams

In dit hoofdstuk staan de sequence-diagrammen die behoren tot de InputCommandHandler.



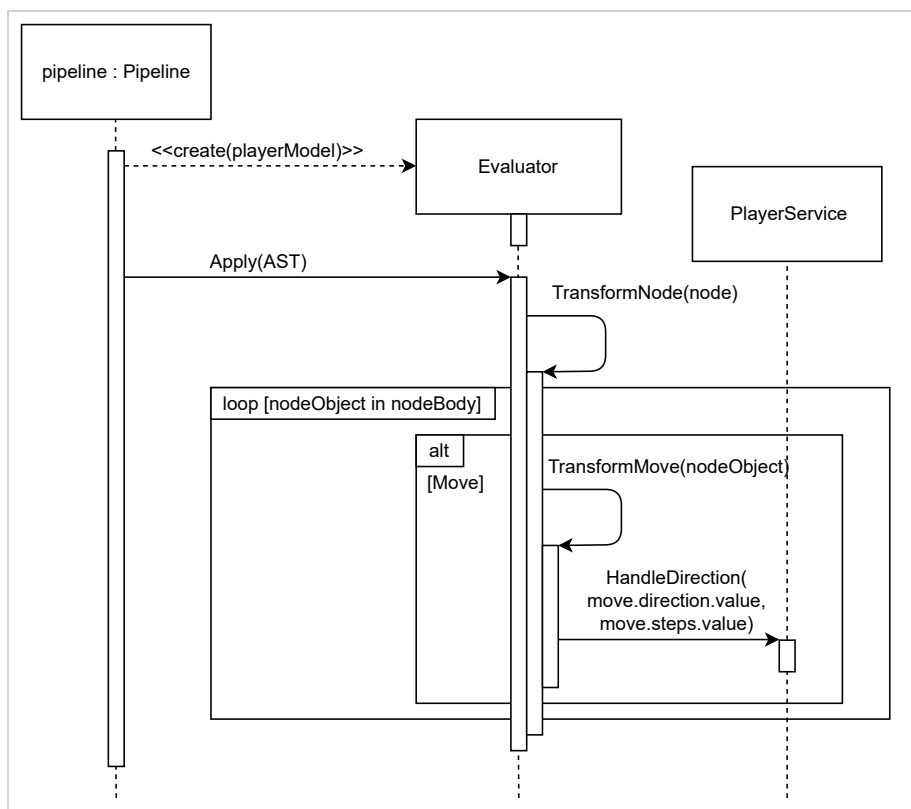
Figuur 15: Sequence diagram HandleCommands

In bovenstaand diagram is te zien wat de `HandleCommands` functie van de `InputCommandHandlerComponent` doet.



Figuur 16: Sequence Diagram ParseCommand

In bovenstaand diagram is te zien wat de ParseCommand functie van de Pipeline doet.



Figuur 17: Sequence Diagram Transform

In bovenstaand diagram is te zien wat de Transform functie van de Pipeline doet. In dit voorbeeld is het commando van 'Move' gebruikt. De alt kan dus ook veranderen, maar om dit diagram niet te uitgebreid te maken zijn alle andere 19 commando's hier niet in opgenomen. Daarnaast is hier niet ingegaan op wat het commando doet binnen de PlayerService, omdat dit niet behoort tot dit sub-system.

3.5.6. Design decisions made for the sub-system

Hier worden de beslissingen die gemaakt zijn voor dit sub-systeem beschreven. Onderstaande decision komt uit de [Decision Detail View](#) van het SAD.

Tabel 16: Decision taal

Decision code	D1
Decision forces	DFV 3 - Parser en Lexer
Decision topic	Welke Parser/Lexer wordt er gebruikt voor de software
Decision	ANTLR
Alternatives	<ul style="list-style-type: none">• GNU Bison• Maken eigen Parser/Lexer• Java-Ast
Arguments	<ul style="list-style-type: none">• Software genereert een begin set aan code waar verder aan gewerkt kan worden.• Software controleert van te voren de taal met de parser.• Parser voor eigen taal kan op één plek worden gewijzigd wat gemakkelijk is voor onderhoud.• Software heeft een goede performance.• Ervaring met ANTLR tijdens de coursefase APP.
Consequences	<ul style="list-style-type: none">• Gebruik maken van een COTS in tegenstelling tot de software zelf schrijven heeft significante tijdswinst.• Afhankelijk van de taal die de COTS ondersteund.
Related requirements	ASR-11

3.6. Design Sub-System World Generation

3.6.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-Systeem

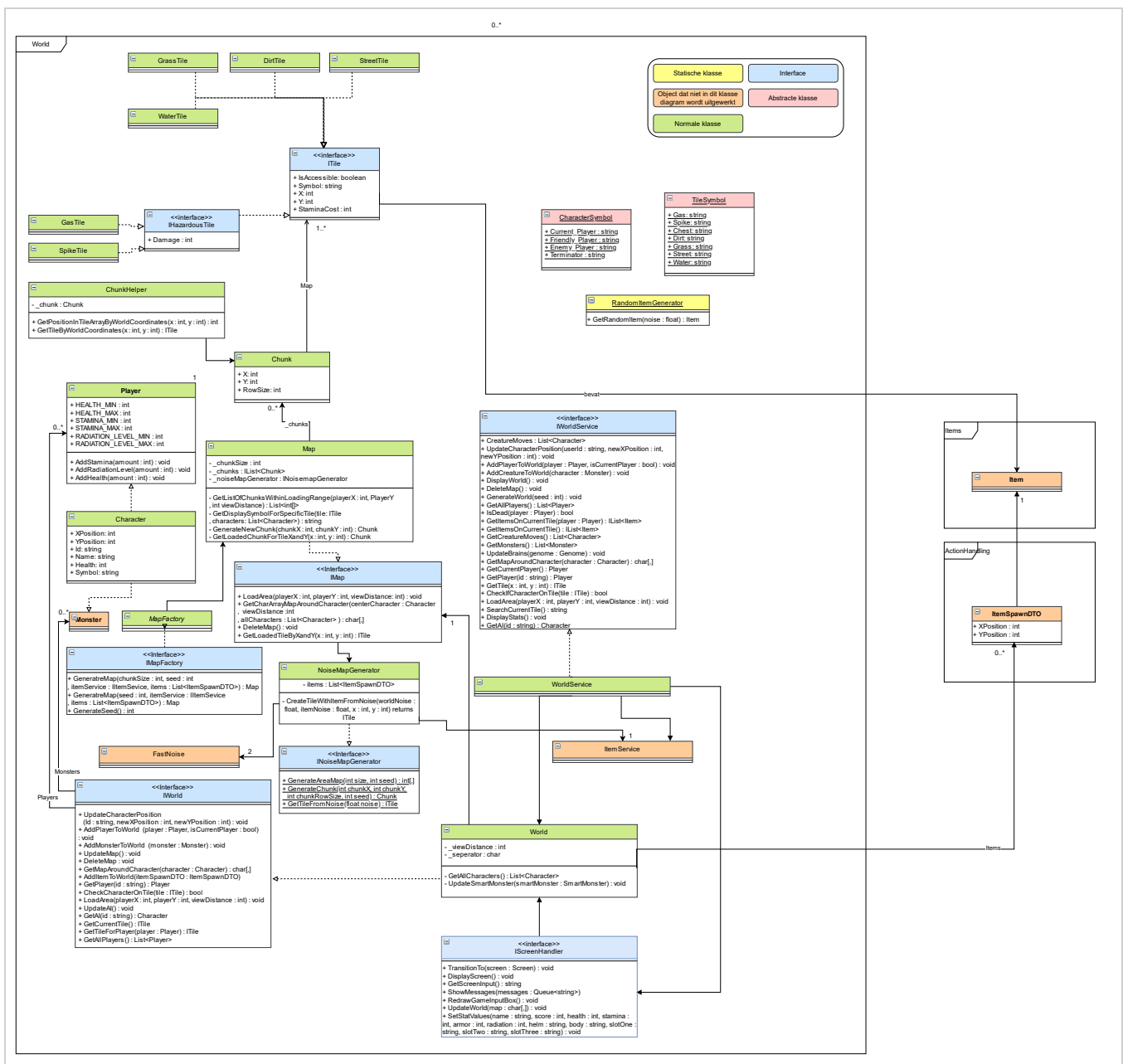
Functionele Requirement	Dekking	Waarom niet gedekt?
FR3	VOLLEDIG	N.V.T.
FR4	VOLLEDIG	N.V.T.
FR5	VOLLEDIG	N.V.T.
FR22	VOLLEDIG	N.V.T.

Niet-functionele Requirement	Dekking	Waarom niet gedekt?
ASR-3	VOLLEDIG	N.V.T.
ASR-4	VOLLEDIG	N.V.T.

Hier wordt het subsysteem van het wereld generatie subsysteem beschreven. Met dit component kan de kaart op basis van een algoritme gegenereerd worden. Deze kaart wordt door andere subsystemen gebruikt om interacties op uit te voeren en om de kaart in het user interface aan de gebruiker te laten zien. Dit subsysteem maakt gebruik van het item subsysteem om items te verdelen op te wereld ook op basis van een algoritme. Er is een onderzoek uitgevoerd naar welke algoritmes mogelijk het best konden werken. In dit onderzoek is de motivatie voor de gemaakte keuze en mogelijke alternatieven terug te vinden, zie het onderzoek [Genereren wereld](#).

3.6.2. Design Class Diagram

Hieronder staat het klassendiagram beschreven. Het klassendiagram omvat alle klassen die gebruikt worden in dit subsysteem.



Figuur 18: Design Class Diagram "Wereld Genereren"

3.6.2.1. Glossary

Naam	Beschrijving
WorldService	Dit is een publieke service die de wereld functies toegankelijk stelt aan andere klassen.
World	De wereld bevat alle spelonderdelen die in de wereld bestaan. Denk hierbij aan spelers en NPC's die rondlopen. De wereld bevat een map object.
MapFactory	De MapFactory verzorgt de onderdelen die het aanroept met de mogelijkheid om een map te 'fabriceren'. Het biedt de mogelijkheid om een nieuwe map te genereren op basis van een seed.
Map	Dit is een map waar de spelers doorheen kunnen bewegen. Hier worden alle chunks opgeslagen.
NoiseMapGenerator	Zorgt voor het genereren van chunks met willekeurige tegels door middel van een noise algoritme.
ChunkHelper	Is verantwoordelijk voor het indexeren van een chunk en op welke plek die staat in de map.
Character	Is een klasse die de basis bevat van elk character in de game. Denk hierbij aan: Spelers, Monsters ect.
Player	Is een character met onder andere een team en stamina. Dit is het character dat de user van de game speelt.
Monster	Is geen speelbaar character.

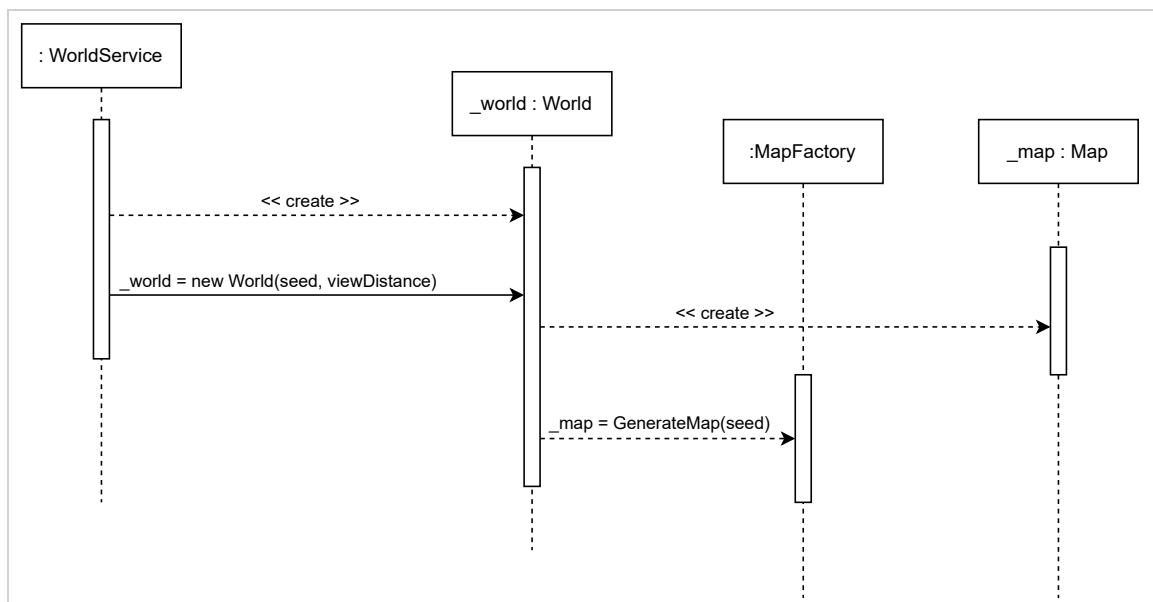
Naam	Beschrijving
Chunk	Een deel van een map. Deze delen zijn allemaal vierkant, even groot en bevatten evenveel tegels.
ITile	Een tegel in de wereld. Een tegel die wordt gebruikt voor verschillende terreinen, zoals water, grass en dirt.
IHazardousTile	Deze soort tegels hebben invloed op de characters die zich bevinden op deze tegels. Denk hierbij aan een gas tile, deze doet damage op characters als ze erop staan.
CharacterSymbol	Bevat de soorten symbolen die horen bij verschillende characters in de game. Dit is de manier waarop characters worden weergegeven in de wereld.
TileSymbol	Bevat de soorten symbolen die horen bij verschillende soorten tiles. Dit is de manier hoe deze tiles worden weergegeven in de wereld.
Item	De items die op een tegel kunnen liggen. Dit kunnen wapens, armor of consumables zijn. Deze is verder beschreven in het sub-system design van items .
IScreenHandler	De screenhandler is de klasse die regelt wat de ggebruikter te zijn krijgt. Deze klasse moet een call krijgen met de nieuwe map als de map wordt ge-update.
RandomItemGenerator	Maakt en returned items op basis van noise.
FastNoise	Een COTS library die een noisemap kan genereren. Voor het besluit om FastNoise te gebruiken, zie het onderzoek Genereren wereld .
ItemService	Een service voor de Item klasse. Deze is verder beschreven in het sub-system design van items .
ItemSpawnDTO	Een data transfer object die gebruikt wordt om de items die zijn spawned op te slaan met de x en y van de tegel waar het item op ligt. Deze is verder beschreven in het sub-system design van items .

3.6.3. External interfaces

De IFastNoise interface is van de externe interface FastNoiseLight. Voor het besluit om FastNoise te gebruiken, zie het onderzoek [Genereren wereld](#).

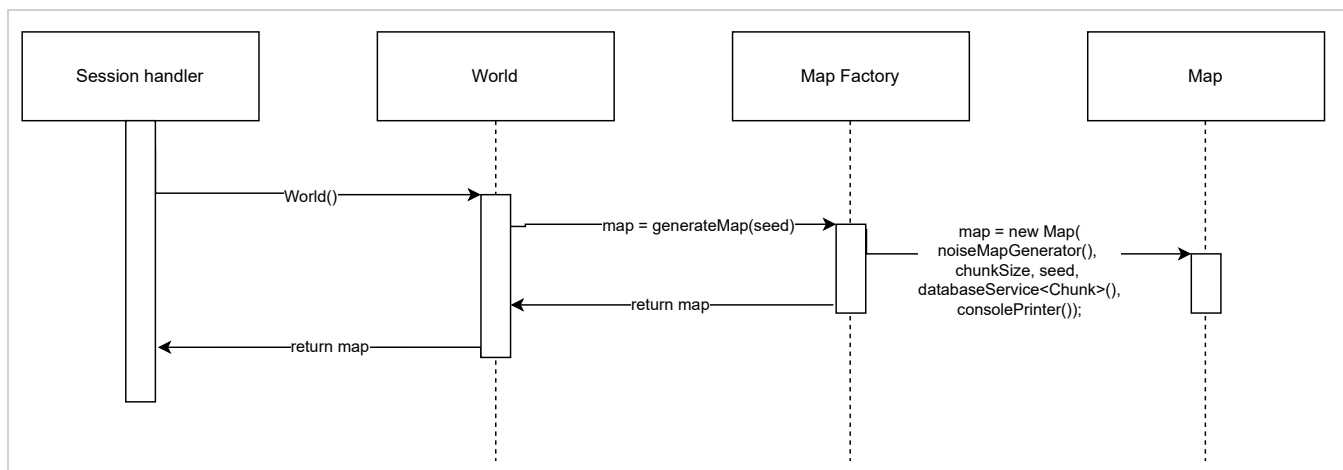
3.6.4. Sequence Diagrams

Voor het onderdeel wereld generatie zijn enkele sequence diagrammen gemaakt die belangrijke functionaliteiten beschrijven.



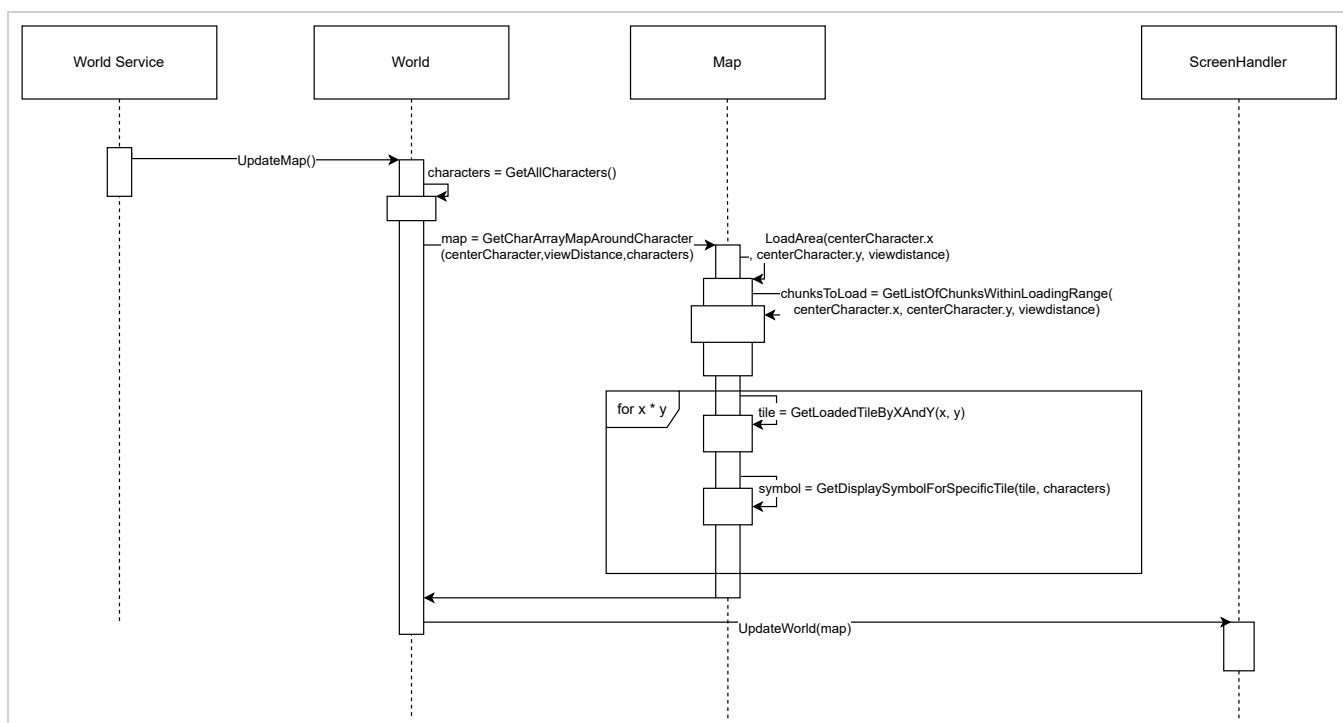
Sequence diagram CreateWorld 1: Sequence diagram CreateWorld

In dit sequence diagram wordt beschreven hoe een nieuwe wereld wordt aangemaakt. De worldService maakt een nieuwe wereld wanneer dit nodig is. Hieraan worden een seed en een viewDistance aan meegegeven. De wereld heeft een instantie van Map nodig, daarom wordt hiervoor een object gemaakt en gevuld met een map die uit de 'GenerateMap' method komt die het seed gebruikt.



SequenceDiagram GetLocalMap 1: SequenceDiagram GetLocalMap

In dit sequence diagram wordt weergegeven hoe een nieuwe map wordt gemaakt. De session handler maakt een nieuwe wereld aan als een session word gestart. Deze maakt in de constructor via een mapfactory een nieuw map object aan, en slaat dit vervolgens op.



SequenceDiagram updateMap 1: SequenceDiagram GetLocalMap

In dit sequence diagram wordt het map ververs comendo verstuurd nadat er een verandering is gebeurd. De Map klasse gaat dan een map opstellen op basis van tegels, items en karakters. Zodra deze lokale map af is wordt deze terug gegeven aan World, die hem weer doorgeeft aan de screen handler om te weergeven.

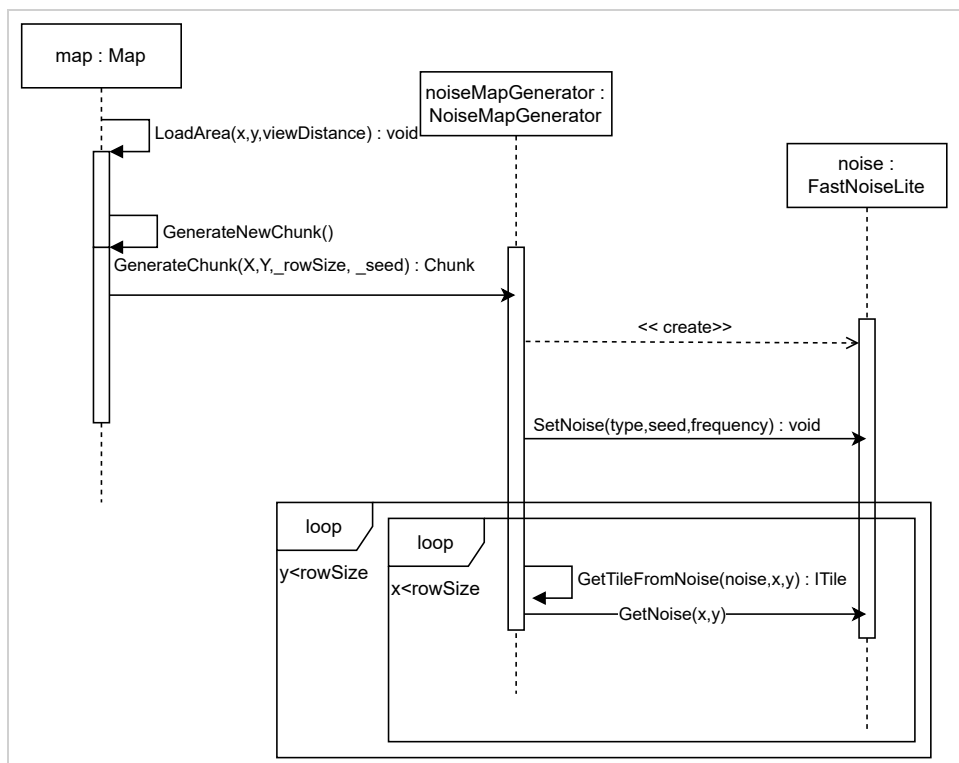


Figure 7: Sequence diagram genereren nieuwe chunk

In dit sequence diagram wordt beschreven hoe nieuwe chunks gegenereerd worden. De 'GenerateNewChunk' method wordt aangeroepen als de wereldkaart gedisplayd moet worden. Hiervoor is een self-call waarin de 'GenerateChunk' method van 'NoiseMapGenerator' wordt aangeroepen. Hieraan worden de x- en y-coördinaten en de grootte van een chunk en de seed van de noisemap meegegeven. Hierna wordt een 'FastNoiseLite' object geïnitieerd en worden de nodige waardes gezet. Binnen de 'GenerateChunk' method wordt een ITile array gemaakt die binnen de twee loops gevuld worden met tiles die overeenkomen met de switch-case statements in de 'GetTileFromNoise' method.

3.6.5. Function index

Klasse	Map
Functie naam	beschrijving
LoadArea	Gebruikt 'GetListOfChunksWithinLoadingRange' om te bepalen welke chunks moeten worden geladen. Daarna kijkt het per chunk of die al geladen is, en zoniet genereert de chunk en slaat de chunk op in de chunk list.
GetListOfChunksWithinLoadingRange	Bepaalt aan de hand van een X, een Y en een laadafstand welke chunks in het gebied zitten dat moet worden geladen (ongeveer het beeldscherm + een extra chunk). Deze lijst van chunks geeft het vervolgens weer terug.
GetCharArrayMapAroundCharacter	<p>Controleert eerst of de viewDistance 0 of hoger is. 0 is acceptabel omdat er 1 wordt toegevoegd aan het view window. Als dit niet zou worden gedaan zou het scherm links van de speler niet even groot zijn als rechts van de speler.</p> <p>Vervolgens worden via de klasse 'LoadArea' de chunks die nodig zijn ingeladen en een 2d char array opgestart. Deze array is "viewDistance * 2 + 1" groot omdat het de viewDistance link en rechts moet hebben + de rij waar de speler op</p>

Klasse	Map
	staat. Hierna wordt in een dubbele for loop elke tegel opgezocht en via 'getDisplaySymbolForSpecificTile' krijgt de resultaten array een extra waarde er in.
getDisplaySymbolForSpecificTile	Deze methode kijkt eerst of er een karakter op een tegel staat. Als dit zo is geeft hij het symbool van het karakter terug. Dan kijkt het of er items op de tegel staan. Als dit zo is geeft hij het symbool van een itemtegel terug. Als deze allebij er niet zijn geeft hij het symbool van de tegel zelf terug.
GenerateNewChunk	Hier wordt de noisemapgenerator aangeroepen om een nieuwe chunk te maken. Deze chunk wordt terug gegeven.
GetLoadedChunkForTileXAndY	Kijkt in welke chunk een tegel met een bepaalde X en Y moet zitten, en geeft deze chunk vervolgens terug als het al bestaat. Anders geeft het null terug.
DeleteMap	Verwijdert alle chunks uit de geladen chunks.
GetLoadedTileByXAndY	Vind door gebruik te maken van 'GetLoadedChunkForTileXAndY' en de chunkHelper functie 'GetTileByWorldCoordinates' de tegel voor een X en Y locatie. Deze tegel wordt terug gegeven.

3.6.6. Algoritme

De World Service maakt gebruik van een noise algoritme, geïmplementeerd als FastNoiseLight. Voor het besluit om Noise te gebruiken, zie het onderzoek [Genereren wereld](#). Dit algoritme krijgt vanuit de session een seed mee, wat het vervolgens gebruikt om voor elke speler dezelfde waardes uit te rekenen bij het bouwen van de wereld. Voor het gebruik van het algoritme zijn twee parameters nodig (meestal een x en y coördinaat). Hier komt altijd een waarde tussen -1 en 1 uit, wat gebruikt wordt als input bij het genereren van de tegels. Voor meer details over het algoritme, zie [Genereren wereld](#).

3.6.7. Design decisions made for the sub-system

In dit hoofdstuk worden de beslissingen die gemaakt zijn voor dit sub-system beschreven.

Tabel 17:

Decision	Description
Problem/Issue	We willen een random seed kunnen meegeven aan een nieuwe map, en er moeten voor unit tests een groot aantal dependencies worden meegegeven aan de map constructor.
Decision	Er wordt een Map Factory gemaakt om alle objecten aan te maken die aan een nieuwe map worden gegeven, en voor een random seed te zorgen.
Alternatives	Map zou zelf een ingewikkelde constructor kunnen krijgen, en elke keer als het wordt aangeroepen moet het heel veel dingen meekrijgen.
Arguments	Met een mapfactory worden alle gerelateerde objecten op een centrale plek aangemaakt in plaats van overal waar je een map zou willen maken.

Tabel 18:

Decision	Description
Problem/Issue	Een groot deel van de functies in MapFactory kunnen niet nuttig worden getest omdat er nieuwe logica in staat.
Decision	De functies die geen logica bevatten worden excluded from code coverage.

Decision	Description
Alternatives	Ze zouden kunnen worden geactiveerd om te kijken of ze een error gooien.
Arguments	Deze functies testen zou ongeveer gelijk staan aan c# zelf gaan testen.

Tabel 19: Tile klasse SOLID principes

Decision	Description
Problem/Issue	De "tile" klasse voldeed niet aan de SOLID principes.
Decision	Er is gekozen om generalisatie, specialisatie en overerving toe te passen.
Alternatives	-
Arguments	Om ervoor te zorgen dat er weinig afhankelijkheden zijn in het systeem, het makkelijk uitbreidbaar is en makkelijk te onderhouden.

Tabel 20: Statische klasse voor TileSymbol

Decision	Description
Problem/Issue	Een "tile" heeft een hard gecodeerde waarde als attribuut.
Decision	Een statische klasse aanmaken dat constanten bevat m.b.t. een "tile" symbol.
Alternatives	De waardes opslaan als string in de "tile" klasse. Er is nagedacht om een Enum te gebruiken maar enums ondersteund alleen integers als waardes.
Arguments	Er is gekozen om een statische klasse te maken voor TileSymbol zodat je voldoet aan o.a. de SOLID principes. Alle symbols kunnen nu op één overzichtelijke plek worden aangepast.

Tabel 21: Gebruik FastNoiseLight

Decision	Description
Problem/Issue	We willen gebruik maken van Noise functionaliteiten maar hebben niet voldoende tijd en kennis om zelf een noise functie te schrijven
Decision	Er wordt gebruik gemaakt van FastNoiseLight, die een functionerend Noise algoritme heeft dat wij kunnen gebruiken.
Alternatives	Zelf een algoritme schrijven, een andere noise implementatie.
Arguments	FastNoiseLight is snel, levert geen verdere afhankelijkheden in de code op, en kan gratis gebruikt worden.

Tabel 22: Statische klasse voor RandomItemGenerator

Decision	Description
Problem/Issue	Er moeten op basis van een noise willekeurige items worden uitgekozen die gespawnd gaan worden.
Decision	Een statische klasse maken met maar één methode die een switch case bevat.
Alternatives	Een normale klasse maken die een methode met een switchcase erin.
Arguments	Bij een normale klasse zou de klasse die deze methode wil gebruiken een object van deze klasse moeten instantiëren. Een statische klasse kan overal worden aangeroepen, want deze hoeft niet geïnstantieerd te zijn.

Tabel 23: ItemSpawnDTO

Decision	Description
Problem/Issue	We wilden alle items in een lijst op kunnen slaan en wilde ook weten welke positie deze items lagen. Voor als spelers items gaan oppakken en deze lijst ge-update moet worden.
Decision	Het maken van een ItemSpawnDTO die de x en y van een tegel meegeeft samen met het item wat erop ligt.

Decision	Description
Alternatives	Hele tegels meesturen als er een item weg wordt gehaald van een tegel.
Arguments	Dit is veel te veel data die je mee stuurt, terwijl je deze niet nodig hebt.

3.7. Design Sub-System Network Switch

3.7.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System

De functionele eisen in [Tabel 1 \(Design Sub-System Network Switch\)](#) en [Tabel 2 \(Design Sub-System Network Switch\)](#) komen uit het [Software Requirement Specification](#) document.

Tabel 24: Use cases die gedekt worden door het Network Switch Sub-System

Usecase	Naam	Dekking	Waarom niet gedekt?
UC4	Speler bewegen	VOLLEDIG	-
UC5	Actie uitvoeren	VOLLEDIG	-
UC6	Versturen chatbericht	VOLLEDIG	-
UC7	Deelnemen lobby	VOLLEDIG	-
UC8	Starten spel	VOLLEDIG	-
UC9	Aanmaken lobby	VOLLEDIG	-

Tabel 25: Andere functionele requirements die gedekt worden door het Network Switch Sub-System

Functionele Requirement	Dekking	Waarom niet gedekt?
FR10	VOLLEDIG	-
FR13	VOLLEDIG	-
FR21	VOLLEDIG	-

De niet-functionele eisen in [Tabel 3 \(Design Sub-System Network Switch\)](#) komen uit het [Software Architecture Document](#).

Tabel 26: Niet-functionele requirement die gedekt worden door het Network Switch Sub-System

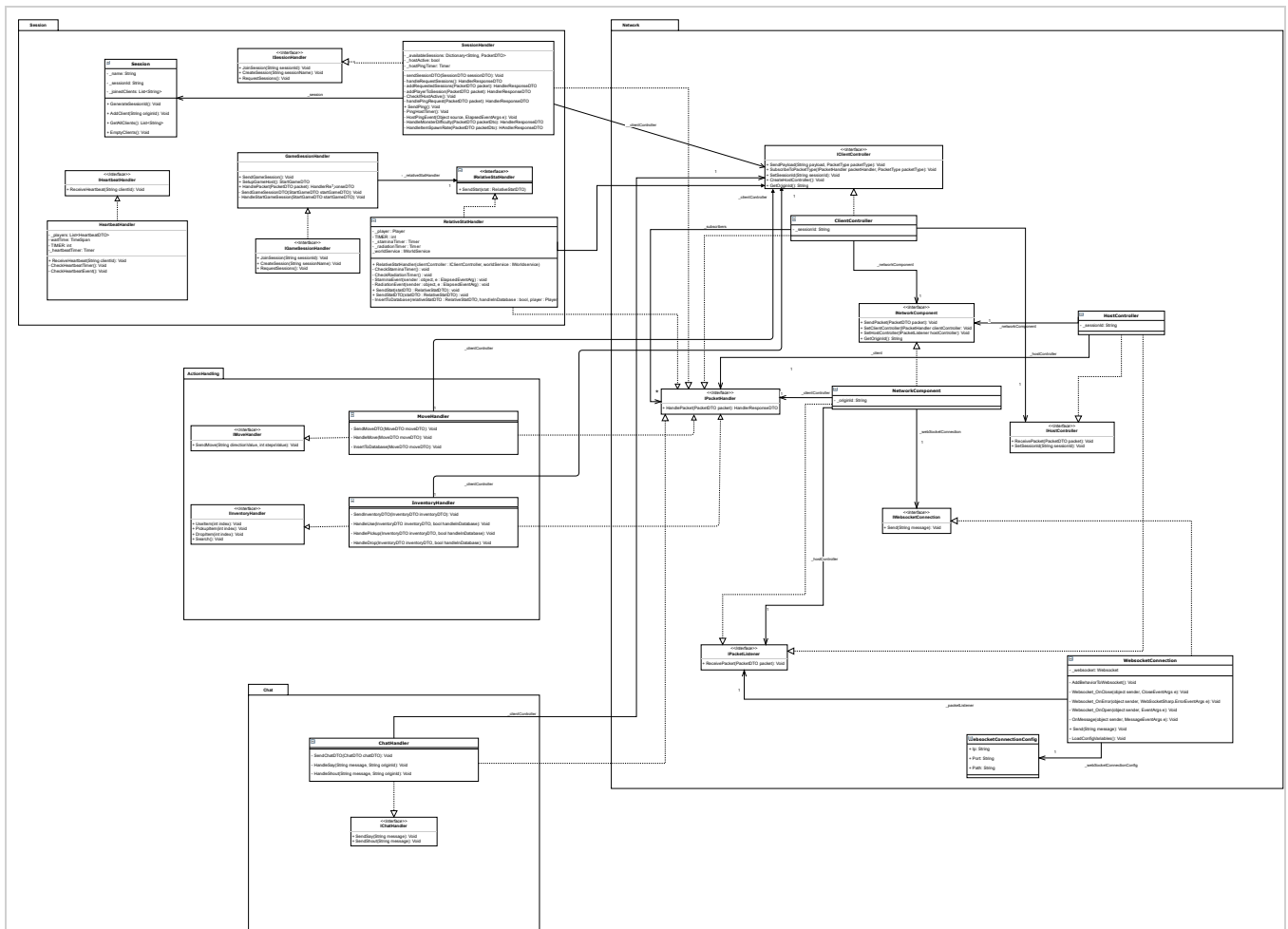
Niet-functionele requirement	Dekking	Waarom niet gedekt?
ASR-4	VOLLEDIG	-
ASR-5	VOLLEDIG	-
ASR-7	VOLLEDIG	-
ASR-11	VOLLEDIG	-
ASR-17	VOLLEDIG	-
ASR-22	VOLLEDIG	-
ASR-25	VOLLEDIG	-
ASR-32	VOLLEDIG	-
ASR-39	VOLLEDIG	-

3.7.2. Gebruikte interfaces

De network component maakt geen gebruik van interfaces uit andere componenten.

3.7.3. Design Class Diagram

Hieronder staat het design class diagram van het network switch sub-systeem. De classes en interfaces uit de Chat en Session solution zijn ook bijgevoegd. Dit omdat er een sterke koppeling met het network switch sub-systeem is.



Figuur 19: Design class diagram network switch

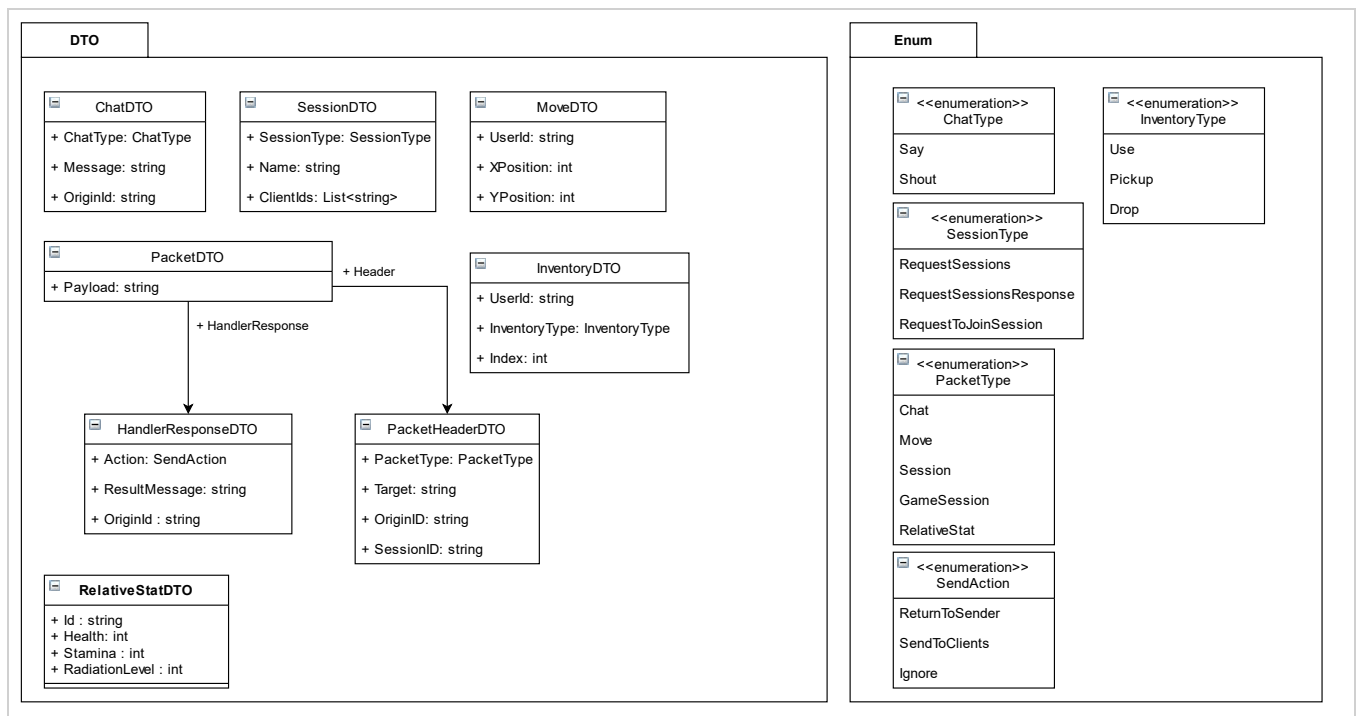
Hieronder staat de glossary die bij het bovenstaande figuur hoort.

Tabel 27: Glossary design class diagram network switch

Class/Interface	Omschrijving
IClientController	Interface voor clients die bepaalde functionaliteiten aanbied
ClientController	De implementatie van de IClientController
IHostController	Interface voor hosts die bepaalde functionaliteiten aanbied
HostController	De implementatie van de IHostController
INetworkComponent	Interface voor het netwerk die bepaalde functionaliteiten aanbied
NetworkComponent	De implementatie van de INetworkComponent
IWebSocketConnection	Interface voor websockets die bepaalde functionaliteiten aanbied
WebSocketConnection	De implementatie van de IWebSocketConnection
WebSocketConnectionConfig	De configuratie van de WebSocketConnection
IPacketHandler	Interface voor het afhandelen van packets die bepaalde functionaliteiten aanbied
IPacketListener	Interface voor het kijken waar welke packet naar toe moet die bepaalde functionaliteiten aanbied
ISessionHandler	Interface voor sessions die bepaalde functionaliteiten aanbied
SessionHandler	De implementatie van de ISessionHandler
Session	Bevat alle informatie en functies die te maken hebben met spelers in een session

Class/Interface	Omschrijving
ICHatHandler	Interface voor de chat die bepaalde functionaliteiten aanbied
ChatHandler	De implementatie van de IChatHandler
IMoveHandler	Interface voor de moves die bepaalde functionaliteiten aanbied
MoveHandler	De implementatie van de IMoveHandler
IInventoryHandler	Interface voor de inventory die bepaalde functionaliteiten aanbied
InventoryHandler	De implementatie van de IInventoryHandler

In onderstaand diagram staan de DTO's en Enum's van de subsystemen network switch, chat en session.



Figuur 20: Design class diagram Enums en DTO's gerelateerd aan de netwerk switch

Hieronder staat de glossary die bij het bovenstaande figuur hoort.

Tabel 28: Glossary design class diagram DTO's en Enum's

Class/Interface	Omschrijving
ChatDTO	Klasse om chat berichten van spelers om te zetten naar JSON.
SessionDTO	Klasse om sessies van host peers om te zetten naar JSON.
MoveDTO	Klasse om moves van spelers om te zetten naar JSON.
InventoryDTO	Klasse om wijzigingen in de inventory van spelers om te zetten naar JSON.
PacketDTO	Bevat een Payload, header en handlerResponse om naar peers te versturen.
HandlerResponseDTO	Bevat het antwoord van een handler. ChatHandler, SessionHandler, MoveHandler etc...
PacketHeaderDTO	Bevat adressering van een packet.
ChatType	Geeft de verschillende soorten subcategorie chat type aan.
InventoryType	Geeft de verschillende soorten subcategorie inventory type aan.
SessionType	Geeft de verschillende soorten subcategorie session type aan.
PacketType	Geeft de verschillende soorten hoofdcategorieën van een packet aan.
SendAction	Geeft in een HandlerResponseDTO aan wat voor actie de host moet ondernemen. Het antwoord aan een specifieke client of alle clients versturen.

3.7.4. Uitleg Packet, Header en Payload

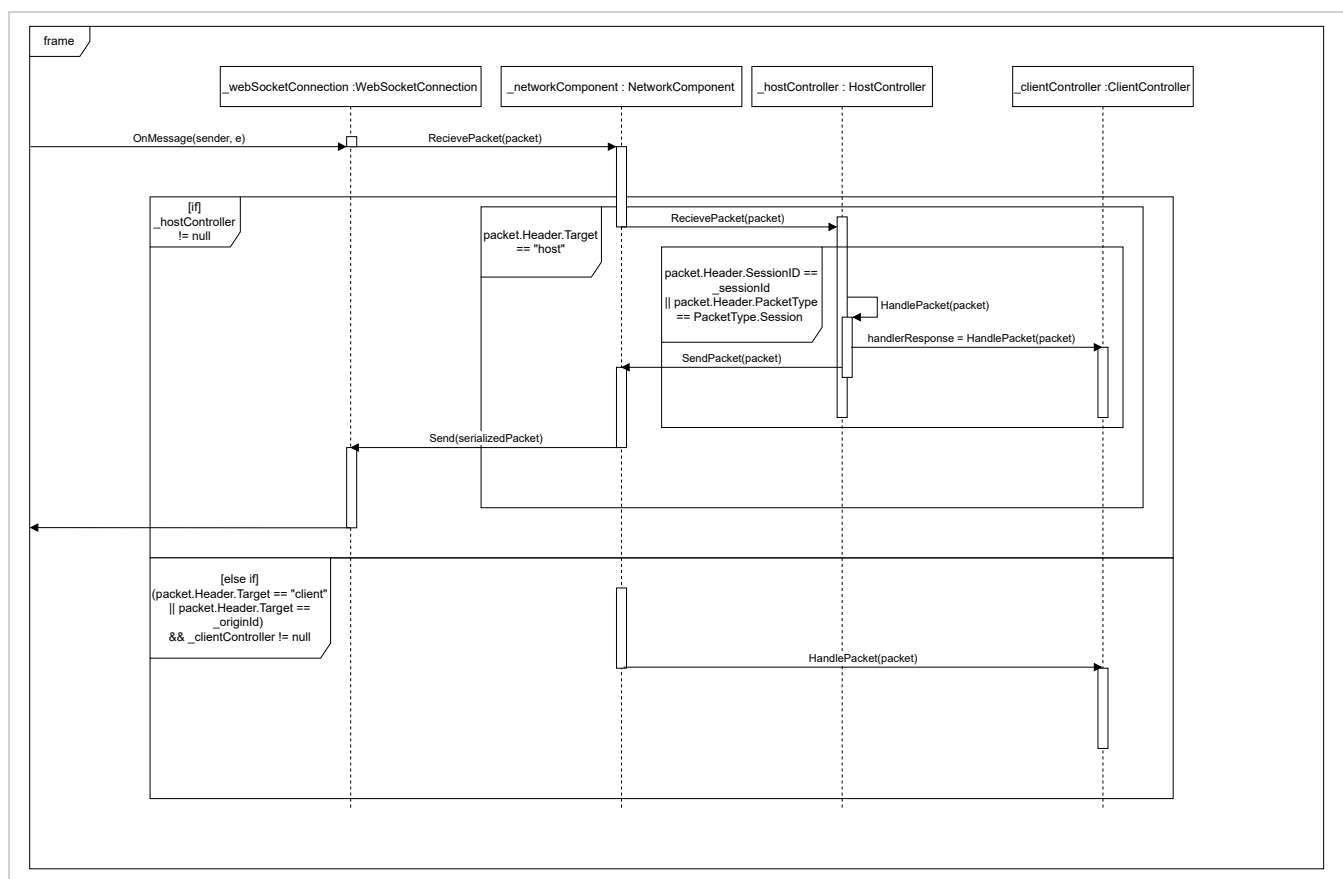
Om de koppeling tussen het netwerk component en de overige componenten laag te houden is er gekozen om gebruik te maken van een payload. Deze payload mag door de componenten zelf gevuld worden. De enige restrictie die wordt opgelegd is dat het een string moet zijn. De Header wordt wellicht door de andere componenten gebruikt maar voornamelijk door het netwerk component om de adressering juist te krijgen. De verantwoordelijkheid voor de netwerk component is te vergelijken met PostNL of de postbode. Er ligt een verantwoordelijkheid om het mogelijk te maken berichten naar andere peers te sturen. Het te versturen bericht kan vergeleken worden met een enveloppe. De enveloppe wordt door een component gevuld met een payload en er wordt een adres op gezet. Deze enveloppe wordt doorgegeven aan het netwerk component. Vanaf dat moment is de versturende component niet meer verantwoordelijk, je levert namelijk niet zelf je post af. Dat doet de postbode.

De componenten hebben wel een verantwoordelijkheid om de enveloppe in en uit te pakken. Serializen en deserializen. Serializen gaat altijd van 'iets' naar een string. En deserializen altijd van een string naar 'iets'. Wat dat 'iets' is mag een component zelf bepalen. Het kan platte tekst zijn, een ANTLR grammatica bestand of een C# klasse. Als het maar naar een string kan worden omgezet en weer terug is het mogelijk om deze naar andere peers te versturen.

3.7.5. Sequence Diagrams

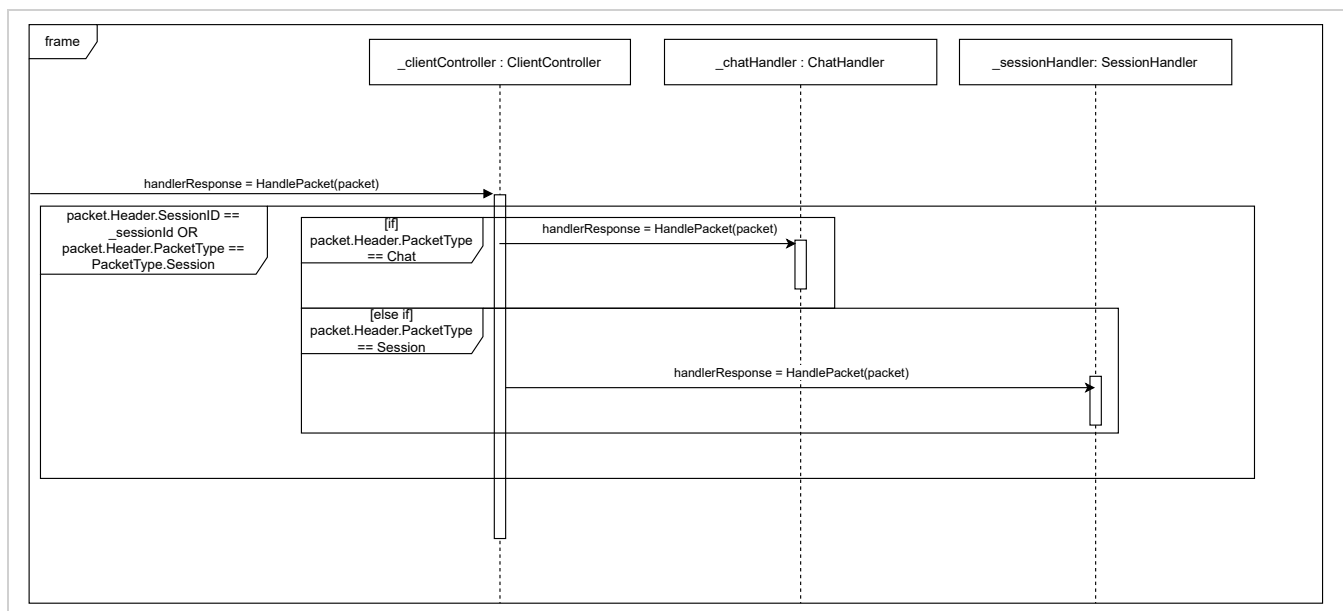
In dit hoofdstuk staan de sequence diagrams die te maken hebben met alle flows in de network switch. De eerste drie diagrammen gaan in op de algemene flow; wat gebeurt er als er een bericht binnenkomt of verstuurd moet worden? De opvolgende diagrammen gaan specifiek in op bepaalde onderdelen die gebruik maken van de algemene flow.

In het onderstaande figuur wordt gevisualiseerd wat er gebeurt als er een packet aankomt.



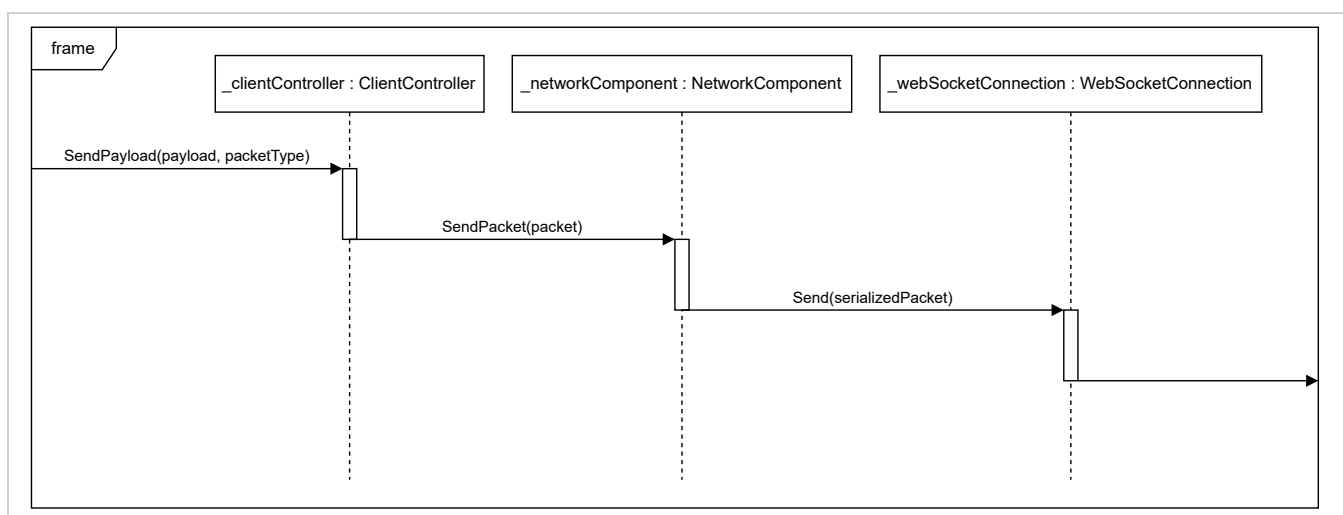
Figuur 21: Sequence diagram: Ontvangen package host en client

In het onderstaande figuur wordt gevisualiseerd wat er gebeurt als er een packet aankomt bij de ClientController.



Figuur 22: Sequence diagram: Client of host handelt packet af

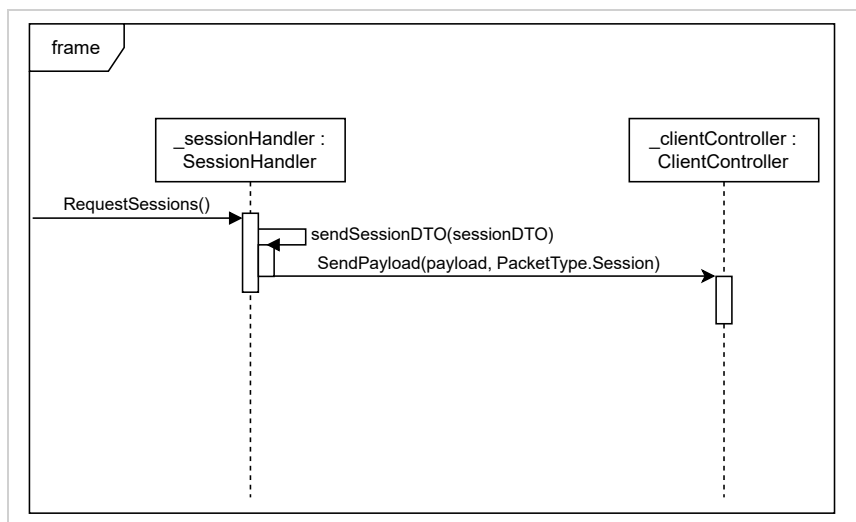
In het onderstaande figuur wordt gevisualiseerd wat er gebeurt als een handler een payload wil versturen.



Figuur 23: Sequence diagram: Versturen packet client

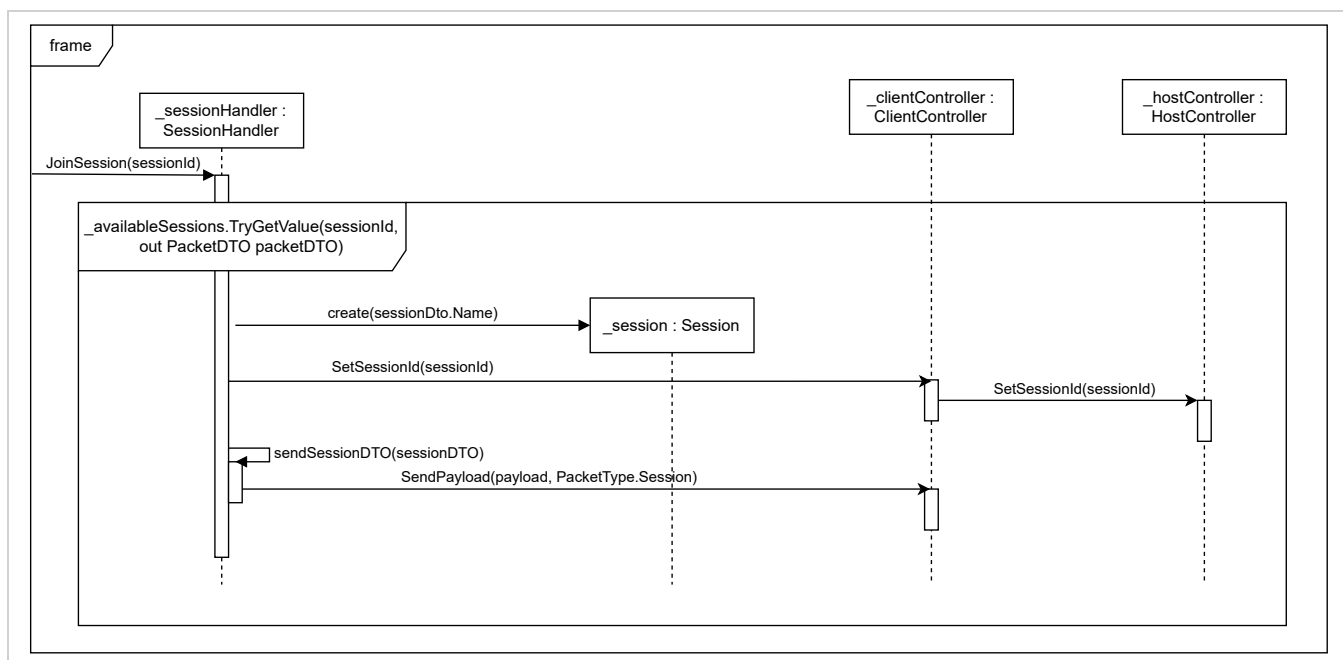
In onderstaande diagrammen wordt verder uitgelegd wat de sessionHandler doet als hij een actie moet uitvoeren of als hij een packet ontvangt.

In het onderstaande figuur wordt gevisualiseerd wat er gebeurt als een speler sessie op wil vragen.



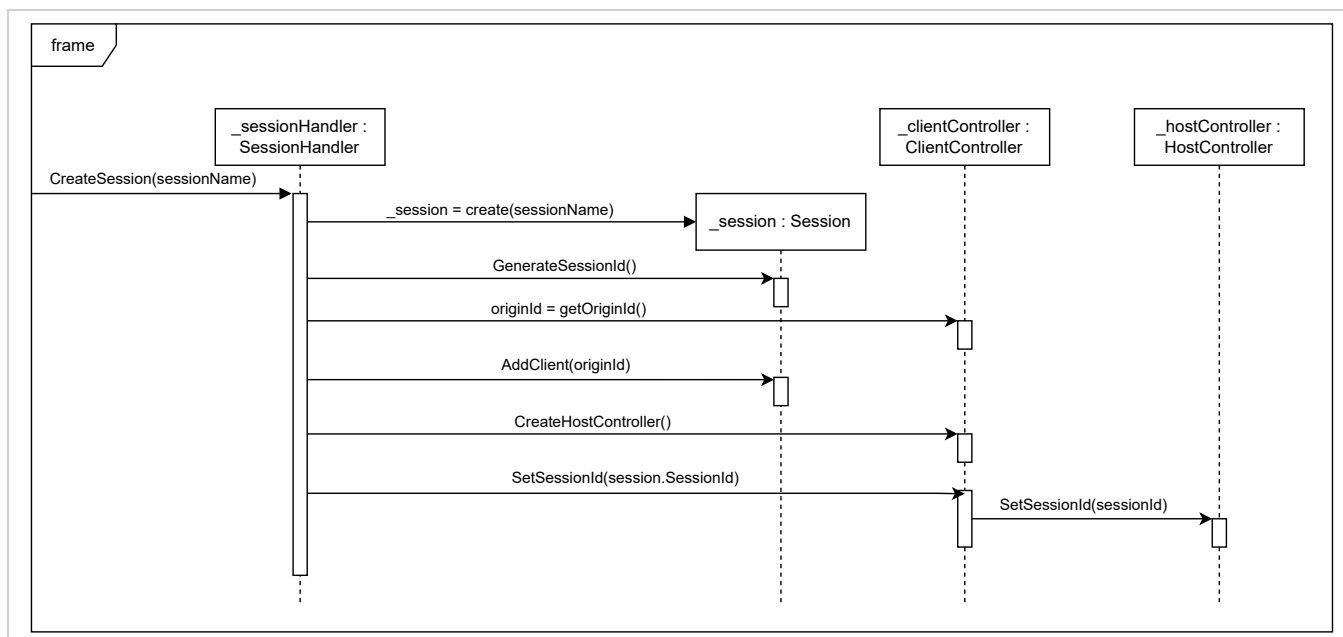
Figuur 24: Sequence diagram: Client vraagt sessies op

In het onderstaande figuur wordt gevisualiseerd wat er gebeurt als een speler aan een sessie wil deelnemen.



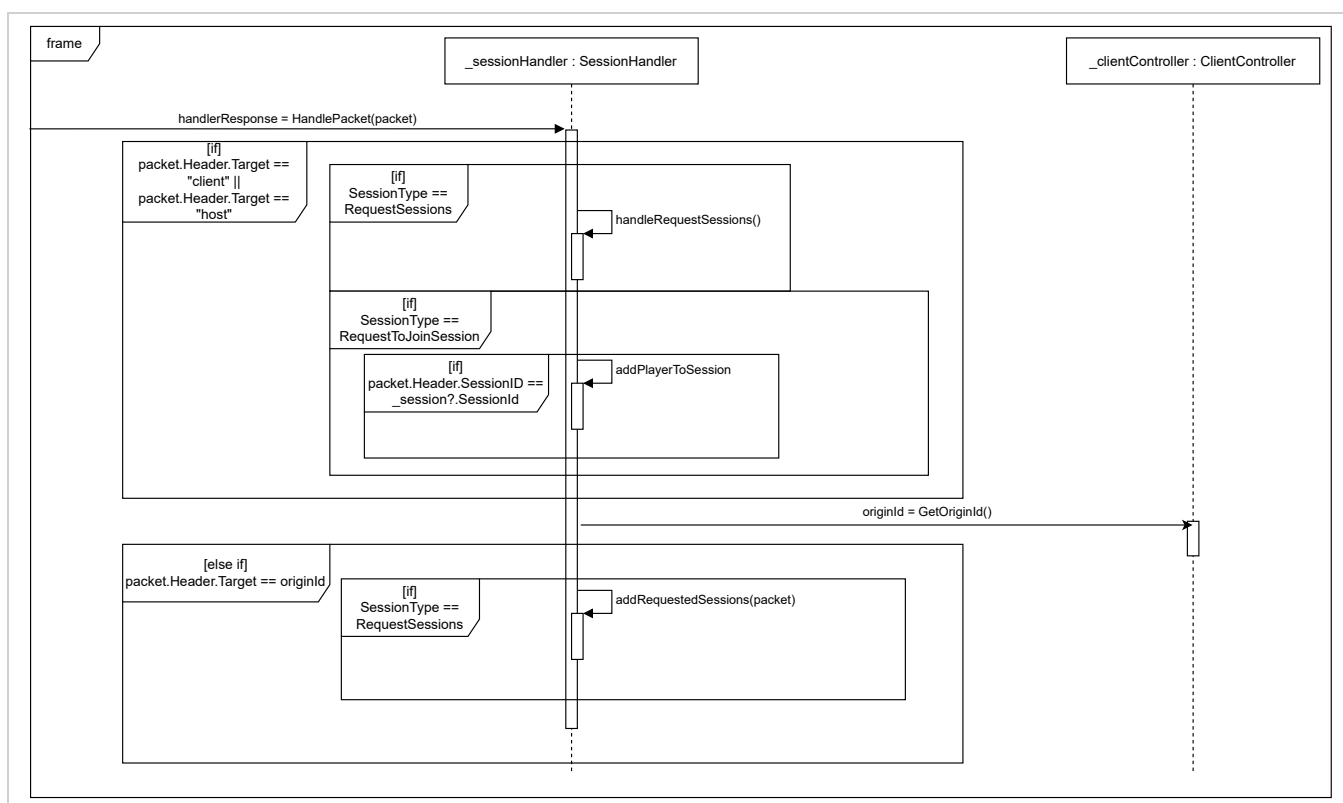
Figuur 25: Sequence diagram: Speler wil aan sessie deelnemen

In het onderstaande figuur wordt gevisualiseerd wat er gebeurt als een speler een sessie wil aanmaken.



Figuur 26: Sequence diagram: Speler wil sessie aanmaken

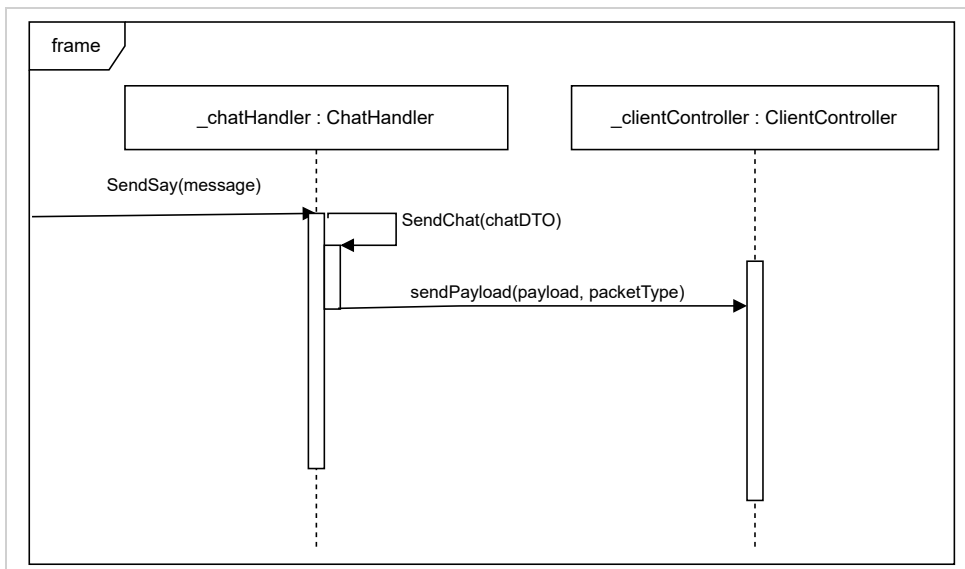
In het onderstaande figuur wordt gevisualiseerd wat er gebeurt als de sessionHandler een packet ontvangt.



Figuur 27: Sequence diagram: Afhandelen packet in SessionHandler

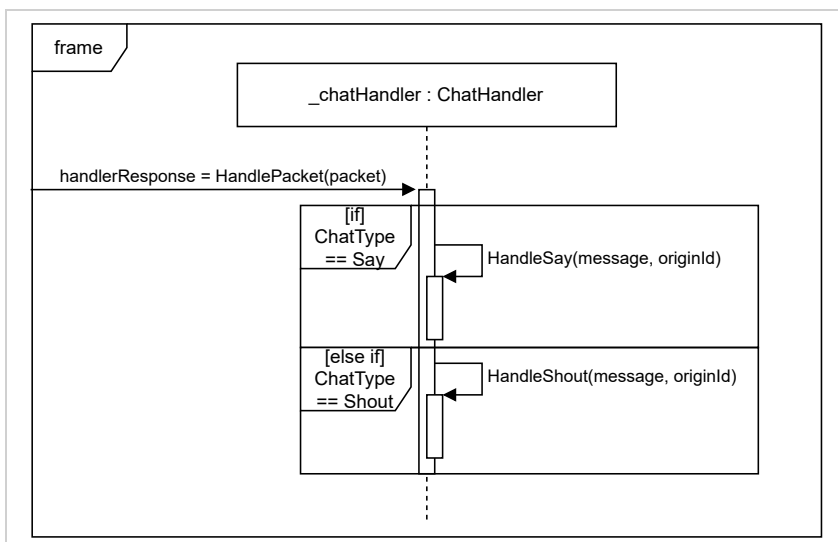
In onderstaande diagrammen wordt verder uitgelegd wat de ChatHandler doet als hij een actie moet uitvoeren of als hij een packet ontvangt.

In het onderstaande figuur wordt gevisualiseerd wat er gebeurt als een speler een chatbericht wil sturen.



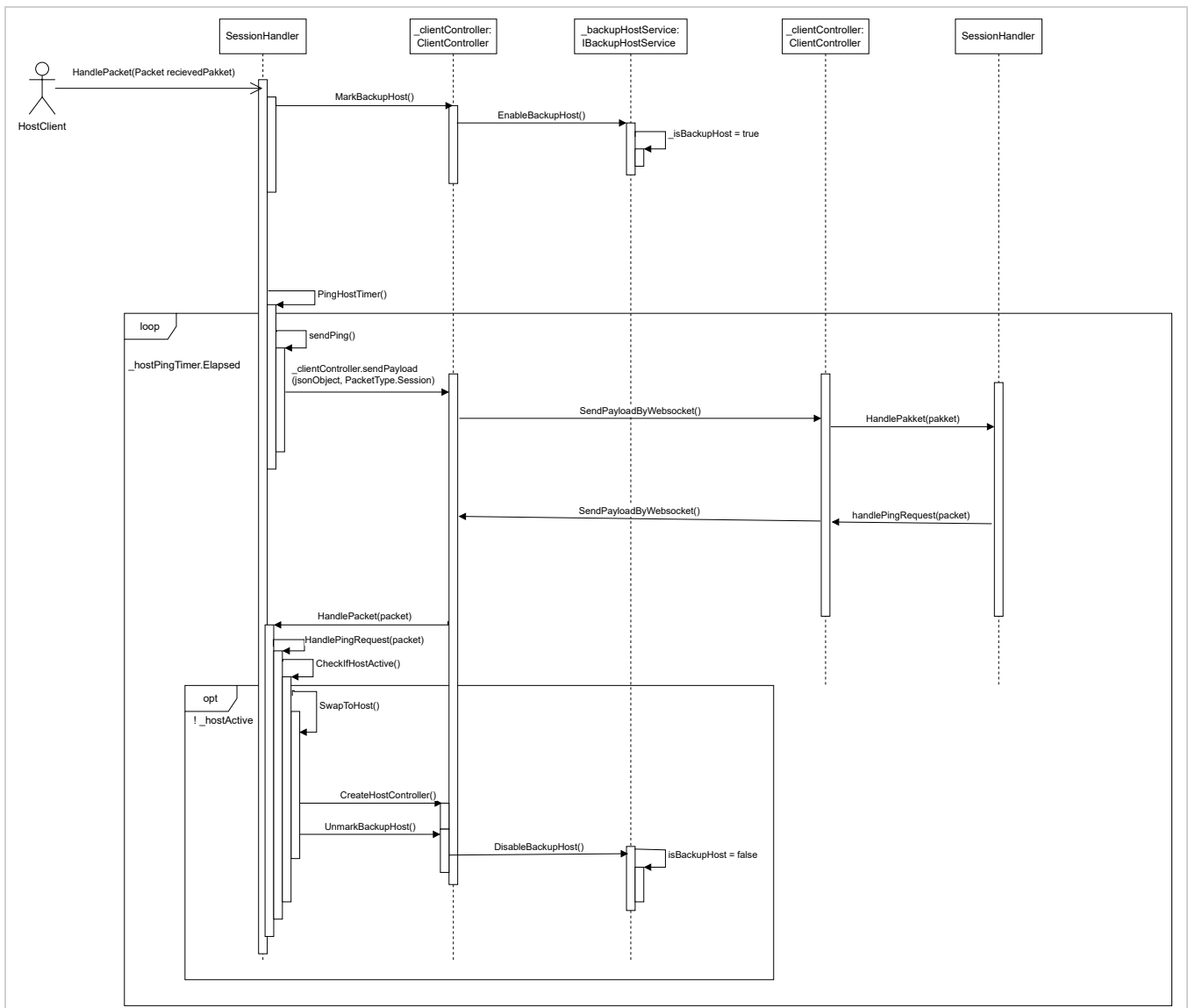
Figuur 28: Sequence diagram: Client wil chatbericht sturen

In het onderstaande figuur wordt gevisualiseerd wat er gebeurt de ChatHandler een packet ontvangt.



Figuur 29: Sequence diagram: Client of host ontvangt chatbericht

Hieronder kunt u het sequence diagram vinden voor de flow van de ping die bepaalt waanneer de backup host de host moet overnemen. Om het diagram overzichtelijker te maken is ervoor gekozen om de functie `sendpayloadbywebsocket` toe te voegen. Deze functie beeld de flow van het sequence diagram van versturen packet client uit.



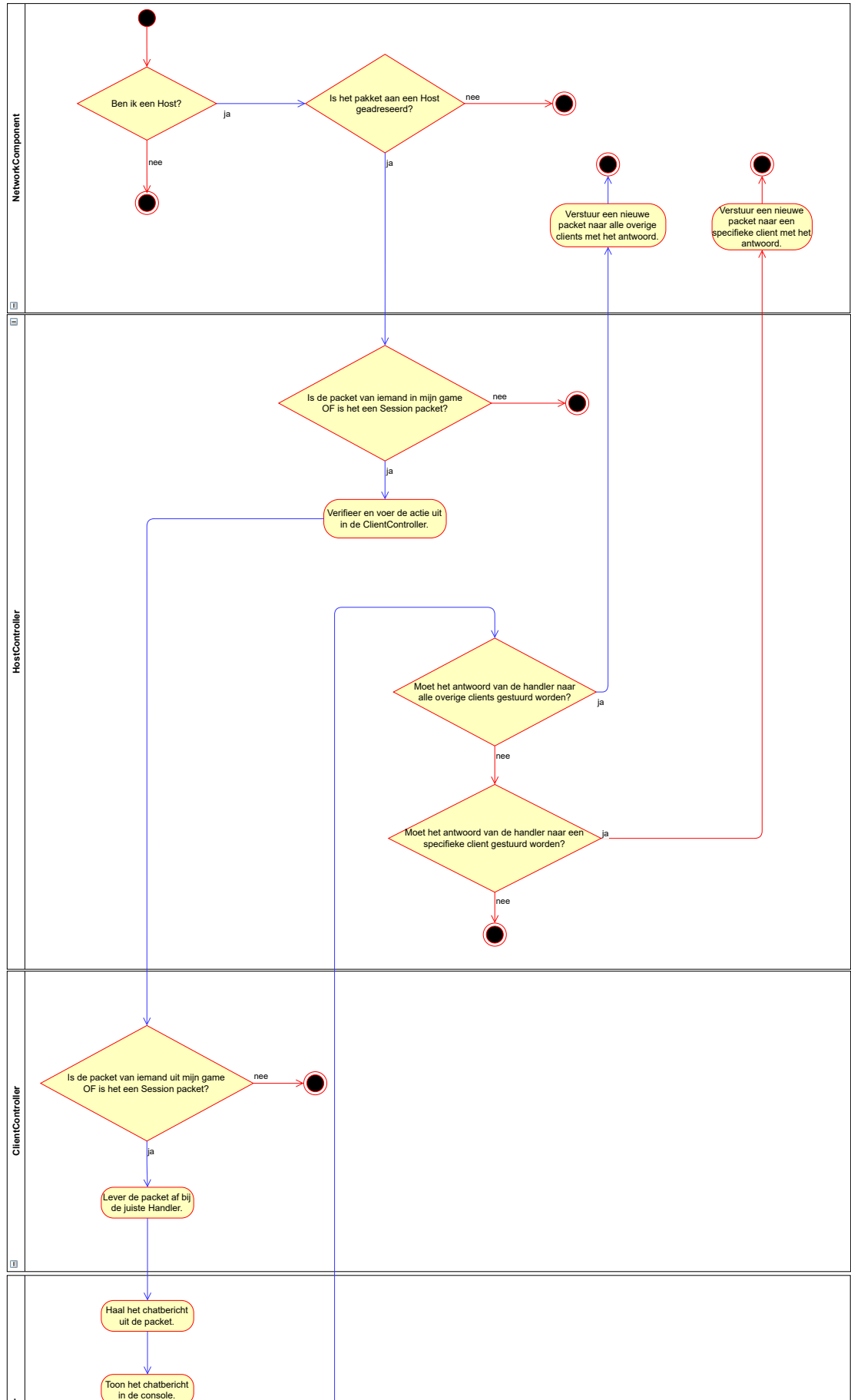
sequence diagram 1: ping

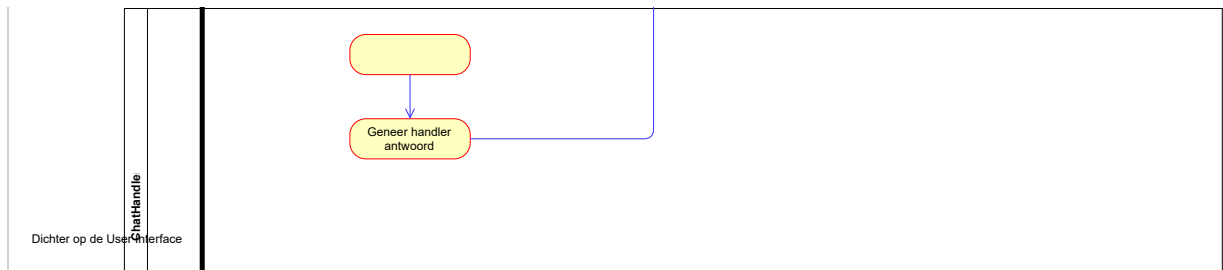
3.7.6. Activity and State Diagrams

In [Figuur 12: Activity diagram: Host ontvangt packet \(Design Sub-System Network Switch\)](#) en [Figuur 13: Activity diagram: Client ontvangt packet \(Design Sub-System Network Switch\)](#) wordt weergegeven hoe een packet binnen de applicatie beweegt. Doormiddel van swimlanes is aangegeven in welke class een actie wordt uitgevoerd of een keuze wordt gemaakt. Aan de bovenkant van beide figuren staat een packet in JSON. De blauwe lijn geeft aan welke weg dat specifieke packet volgt. De bedoeling van deze twee figuren is een simpele weergave. Het bevat zo min mogelijk technische jargon om het begrijpelijk te houden. De packet die als voorbeeld gebruikt wordt is van het type chat.

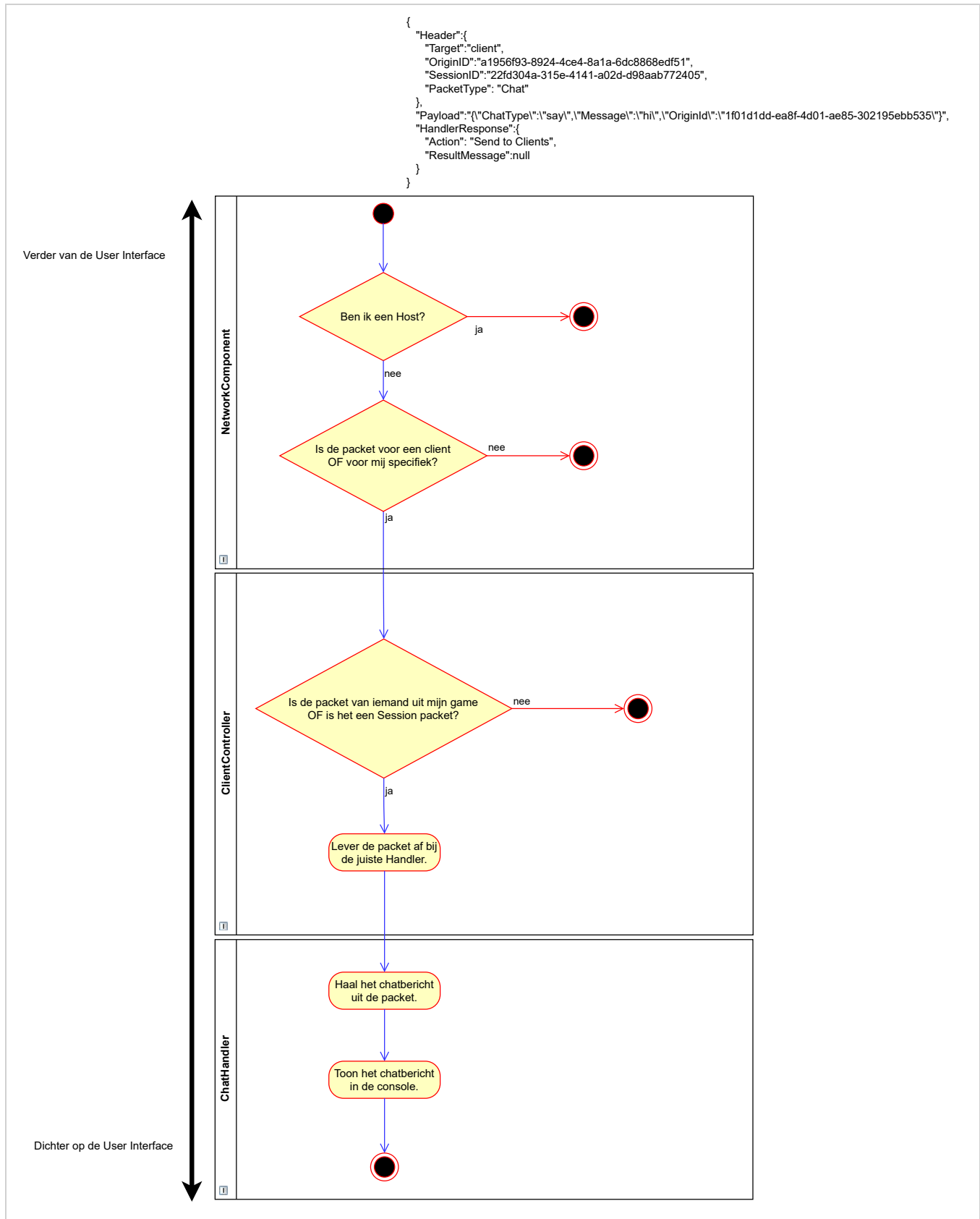
```
{
  "Header": {
    "Target": "host",
    "OriginID": "f101d1dd-ea8f-4d01-ae85-302195ebb535",
    "SessionID": "22fd304a-315e-4141-a02d-d98aab772405",
    "PacketType": "Chat"
  },
  "Payload": {
    "ChatType": "say",
    "Message": "hi",
    "OriginId": "f101d1dd-ea8f-4d01-ae85-302195ebb535"
  },
  "HandlerResponse": null
}
```

Verder van de User Interface





Figuur 30: Activity diagram: Host ontvangt packet



3.7.7. Design decisions made for the sub-system

Decision	Description
Problem/Issue	Keuze geschikte websocket client library.
Decision	WebSocketSharp
Alternatives	Socket.IO-client for .NET
Arguments	Er is gekozen voor de WebSocketSharp library. De netwerk switch maakt ook gebruik WebSocketSharp, dit is een standaard websocket implementatie. Socket.io is een wrapper om een standaard websocket heen en de Socket.IO-client for .NET kan daarom niet gebruikt worden. Zie Onderzoek websocketserver .

3.8. Design Sub-System Items

3.8.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System

Tabel 29: Sub-system Items - Use Cases

Usecase	Naam	Dekking	Waarom niet gedekt?
UC-9	Item inspecteren	Volledig	-
UC-7	Oppakken van voorwerp	Volledig	-
UC-8	Weggooiën van voorwerp	Deels	Item verdwijnt compleet, in plaats van terug op de tegel waar de speler stond
UC-10	Gebruiken van voorwerp	Volledig	-

Tabel 30: Sub-systeem Items - Functionele Requirements

Functionele Requirement	Dekking	Waarom niet gedekt?

Er zijn geen functionele requirement die met dit sub-systeem te maken hebben.

Tabel 31: Sub-systeem Items - Niet functionele Requirements

Niet functionele Requirement	Dekking	Waarom niet gedekt?
ASR-29	Deels	Sub-systeem onderdeel van groter systeem, de data van speler.
ASR-9	Volledig	-

3.8.2. Gebruikte interfaces

De volgende interface zijn in het klasse diagram gespecificeerd:

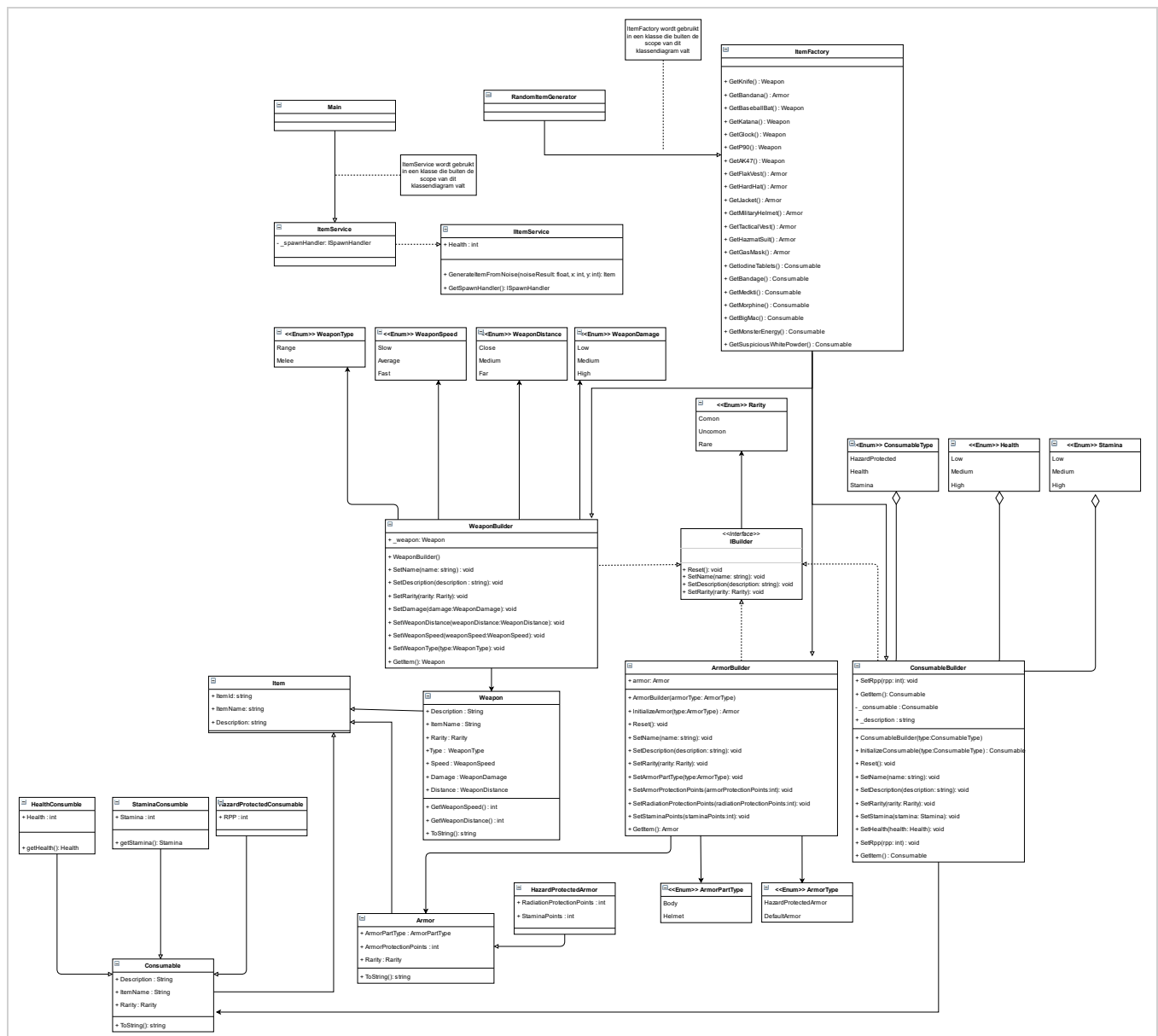
- IBuilder

3.8.3. Algoritme implementatie

Het algoritme dat is gebruikt bij de items is een aangepaste versie 'Abstract factory pattern' omschreven door de 'GoF'. De aanpassingen zijn onder ander dat de 'ConcreteFactory' ontbreekt en is vervangen door een 'Builder Pattern' ook omschreven door de 'GoF'.

3.8.4. Design Class Diagram Item

Hieronder kunt u het klassen diagram van de items zien. Binnen dit diagram word beschreven welke items er zijn en hoe de speler statistieken eruit zien. Daarnaast word er gebruik gemaakt builders om deze statistieken aan te passen. De items zijn voor de speler om meer te interacteren op de ui.



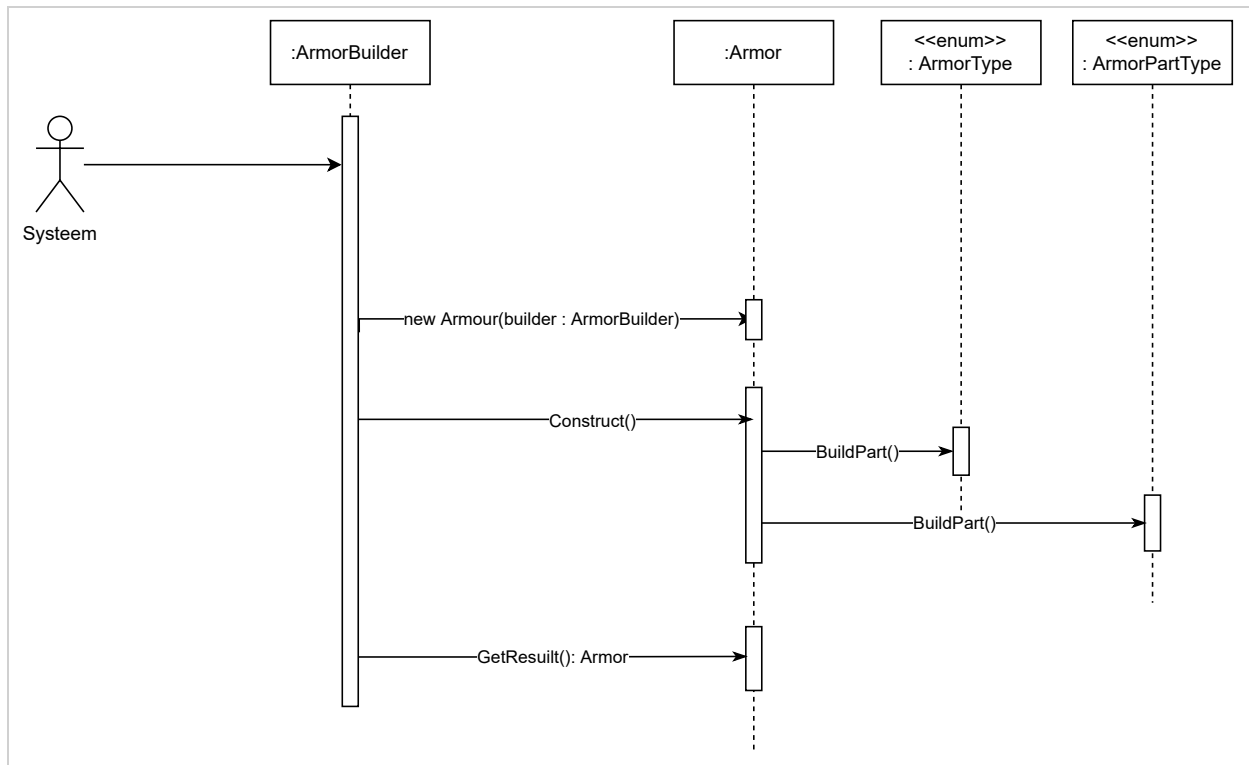
Figuur 32: Design Class Diagram Weapon

3.8.4.1. Glossary

Tabel 32: Glossary

Concept	Description
Item	Base-klasse van alle spullen die in het spel geïmplementeerd worden.
Weapon	Sub-klasse van Item, dit is het wapen waarmee de speler anderen aanvalt, deze klasse wordt gebruikt voor de damage waarden tijdens een attack action.
Consumable	Sub-klasse van Item, representeert de spullen die de speler heeft zoals medicijnen of eten.
HazardProtectedConsumable	Sub-klasse van Consumable voor items die bonus opleveren voor RPP.
HealthConsumble	Sub-klasse van Consumable voor items die bonus opleveren voor HP.
StaminaConsumble	Sub-klasse van Consumable voor items die bonus opleveren voor SP.
Armor	Sub-klasse van Item, dit zijn de beschermende kogel werende kledingstukken die een speler minder vatbaar voor doodgaan maken.
HazardProtectedArmor	Sub-klasse van Armor, dit zijn items die tegen giftige gevaren beschermen in plaats van kogels.

3.8.5. Sequence Diagram Builder Pattern



Figuur 33: Sequence diagram - Armor Builder

Een vergelijkbaar patroon is er voor de verschillende item types: Consumables en Weapons.

3.8.6. Design decisions made for the sub-system

In dit hoofdstuk worden de beslissingen die gemaakt zijn voor dit sub-system beschreven.

Tabel 33: Decision Builder Pattern

Decision	Description
Problem/Issue	Er zijn veel subklassen die de Item class overerven.
Decision	Er is gebruik gemaakt van het builder pattern
Alternatives	-
Arguments	Het Builder pattern suggereert dat je de object construction code uit zijn eigen class extraheert en naar apart objecten die builders worden genoemd, verplaatst.

3.9. Design Sub-System Database Handling

3.9.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System

Tabel 34: Sub-systeem DatabaseHandler - Use cases

Usecase	Naam	Dekking	Waarom niet gedekt?
UC-11	Pauzeren spel sessie	Deels	Minor bugs waardoor het niet gemerged kan worden met develop.
UC-12	Hervatten van een spel sessie	Deels	Minor bugs waardoor het niet gemerged kan worden met develop.
UC-3	Starten spel	Volledig	-
UC-10	Uitvoeren actie	Volledig	-

Tabel 35: Sub-systeem DatabaseHandler - Functionele Requirements

Functionele Requirement	Dekking	Waarom niet gedekt?
FR23	Deels	Minor bug waardoor het niet gemerged kan worden met develop.
FR28	Deels	Minor bug waardoor het niet gemerged kan worden met develop.

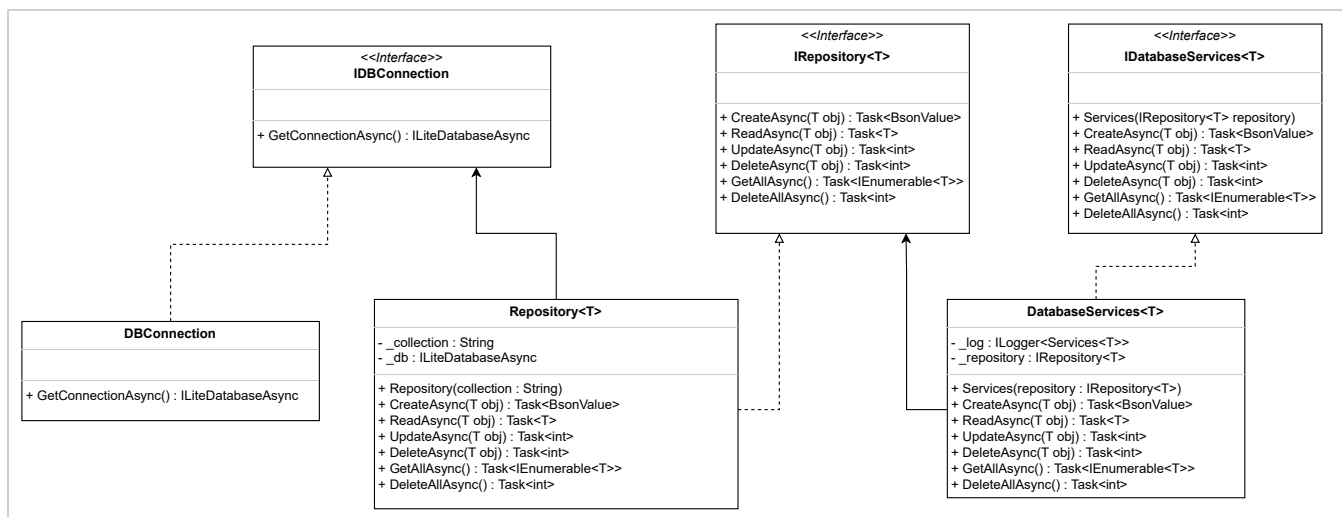
Tabel 36: Sub-systeem DatabaseHandler - Niet functionele requirements

Niet functionele Requirement	Dekking	Waarom niet gedekt?

Er zijn geen non-functionele requirement die met dit sub-systeem te maken hebben.

3.9.2. Design Class Diagram

In onderstaande afbeelding is het design class diagram te zien voor de DatabaseHandler. Om te voorkomen dat elke onderdeel zijn eigen implementatie heeft met betrekking tot het opslaan van data, is er gekozen om een DatabaseHandler sub-systeem te maken. Dit sub-systeem kan elke onderdeel van het project gebruiken voor het opslaan van data. Bij het maken van deze DatabaseHandler is aandacht besteed aan het scheiden van verschillende lagen (Repository-Service design pattern) en dat alle acties asynchroon worden uitgevoerd.



Figuur 34: Design Class Diagram Commando

In onderstaande tabel is een glossary te zien van bovenstaande klassendiagram.

Tabel 37: Design Class Diagram Commando Glossary

Naam	Beschrijving
IDBConnection	Een interface waar methodes worden gedefinieerd die gebruikt worden.
IRepository<T>	Een interface waar methodes worden gedefinieerd die gebruikt worden.
IDatabaseServices<T>	Een interface waar methodes worden gedefinieerd die gebruikt worden.
DBConnection	Een klasse die gebruikt wordt bij het opzetten van een database verbinding d.m.v. GetConnectionAsync() aan te roepen.
Repository<T>	Een klasse die communiceert met de database waarbij CRUD-acties uitgevoerd kan worden.
DatabaseServices<T>	Een klasse die gebruik maakt van Repository<T> om CRUD-acties uit te voeren. Deze klasse wordt ook gebruikt bij het unit testen.

3.9.3. Sequence Diagram

In dit hoofdstuk staat een sequence diagram die behoort tot de DatabaseHandler.

3.9.3.1. Initialisatie database verbinding

In de onderstaande diagram is te zien hoe je de DatabaseHandler moet gebruiken. Als eerst roep je de service aan, de service klasse initialiseert een nieuwe repository klasse aan en in de repository klasse wordt de database verbinding aangemaakt en opgezet voor gebruik.

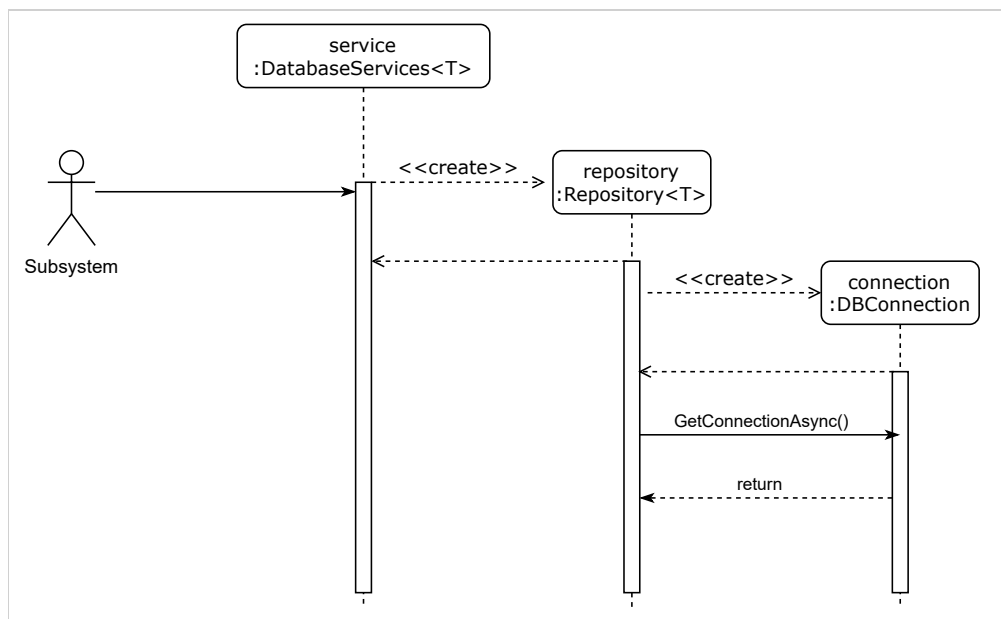


Figure 8: Initialiseren verbinding

3.9.4. Design decisions made for the sub-system

In deze tabel worden de beslissingen die gemaakt zijn voor dit sub-system beschreven.

Tabel 38: Decision data access layer

Decision	Description
Problem/Issue	Het probleem was dat meerdere onderdelen van het project gebruik gaat maken van data opslag. Om ervoor te zorgen dat elke onderdeel van het project niet zijn eigen implementatie heeft van een database moet er een algemene database handler worden geschreven die elke onderdeel van het project kan gebruiken om data op te slaan.
Decision	Gebruik van Repository-Service pattern.
Alternatives	Data Access Object pattern.
Arguments	Er is gekozen voor de Repository-Services design pattern. Door het gebruik van deze design pattern zorg je ervoor dat je de business layer opsplits in meerdere lagen (repository en service layer). Door het gebruik van een service layer bij je repository is het mogelijk om deze laag te unit testen en zorg je ervoor dat de data/business en service logica gecentraliseerd houdt en biedt een flexibele architectuur. Als je data logica of business logica wilt bijwerken dan hoeft je de repository logica niet aan te passen.

3.10. Design Sub-System User Interface

3.10.1. Functionele en niet-functionele requirements die gedekt worden door het Sub-System

Tabel 39: Sub system - User Input Usecases

Usecase	Naam	Dekking	Waarom niet gedekt
UC-1	Configureren van Agent	Volledig	-
UC-13	Aanmaken van lobby	Deel	Alleen UI component
UC-14	Deelnemen aan lobby	Deel	Alleen UI component
UC-20	Laden van gestopt spel	Deel	Alleen UI component

Tabel 40: Sub system - User input - F-Requirements

Functionele	Dekking	Waarom niet gedekt
-------------	---------	--------------------

Requirement		
FR25	Volledig / Deels	Door de custom rules kan de vragen $O(n)$ groot worden. Als de speler geen custom rules wil maken is dit gedekt.
FR26	Volledig	-
FR27	Volledig	-
FR2	Volledig	-
FR14	Deel	Alleen de UI gedeelte, en doorgave aan de pipeline (zie sub-systeem Parser/Lexer Agent)
FR20	Volledig	-

Tabel 41: Sub system - User input - NF Requirements

Niet Functionele Requirement	Dekking	Waarom niet gedekt
ASR-9	Volledig	-
ASR-12	Volledig	-
ASR-13	Volledig	-
ASR-21	Deel	Sommige menu onderdelen zijn niet volledig 'alle input' vriendelijk zoals, spelconfiguatie en configureren van de agent

3.10.2. Gebruikte interfaces

Er zijn een aantal interface gebruikt in dit sub-system:

- IGameSessionHandler
- IGameConfigurationHandler
- IScreenHandlerIScreen
- IGameStatScreenIGameChatScreenIGameWorldScreen

Externe Interfaces:

- Geen

3.10.3. Algoritme Implementatie

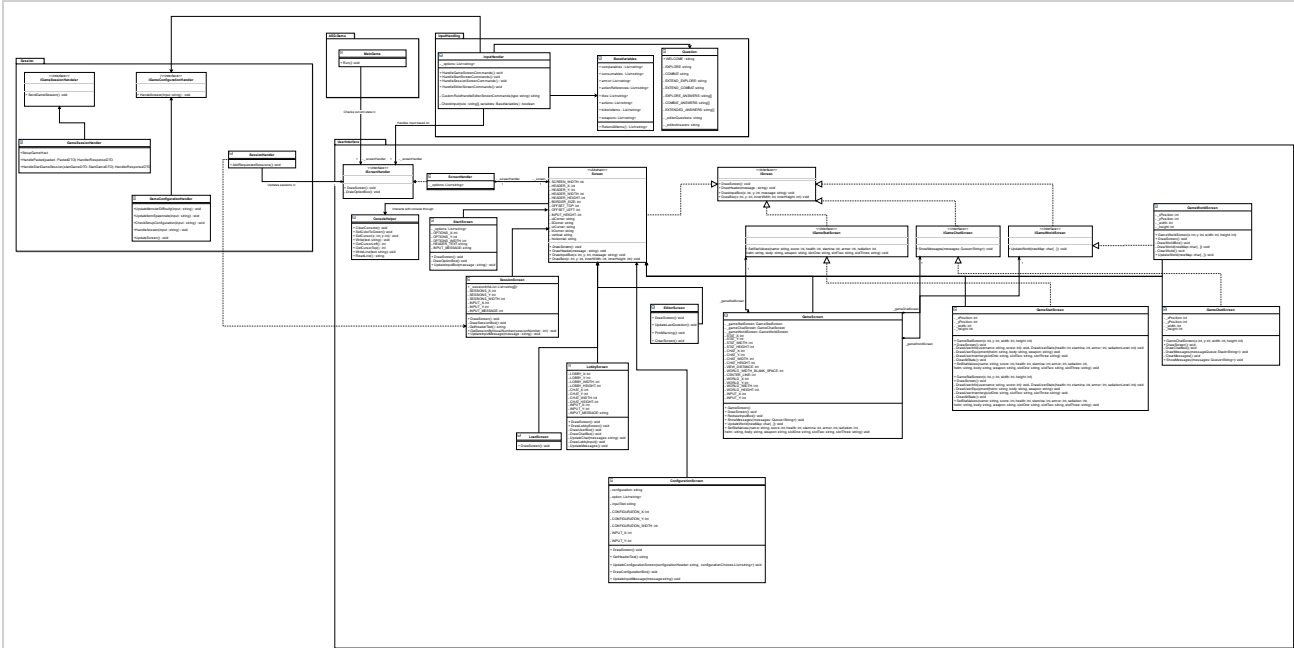
Een vorm van algoritme is menu navigatie dat hier wordt afgehandeld in combinatie met de Input-handler sub-systeem.

Ook de basis checker bij het programmeren van de agent is een simpel algoritme. Verdere checks worden gedaan door het sub-systeem inpuhandler en sub-systeem AI parser lexer

verder zijn er geen algoritmes geïmplementeerd

3.10.4. Design Class Diagram

Het Sub-Systeem User Interface bevat de classes en logica voor het genereren en tekenen van spel output. Door middel van het State Pattern wordt de huidige staat van het spel in de vorm van een scherm bijgehouden. De overgang naar nieuwe schermen wordt gedaan door de state te veranderen en van de nieuwe state het scherm te tekenen.



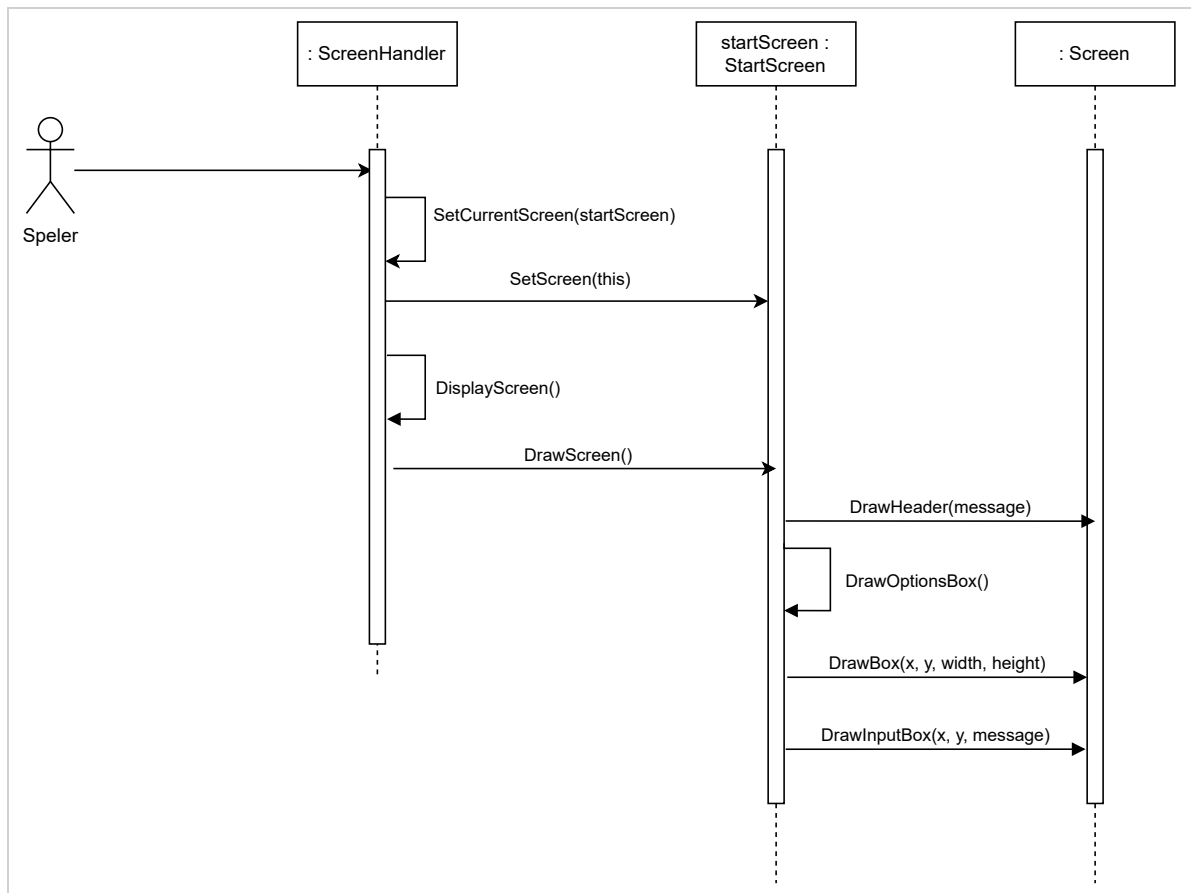
3.10.4.1. Glossary

Naam	Beschrijving
MainGame	De MainGame class is de hoofdklasse waar de huidige staat van het spel wordt bepaald.
InputHandler	De InputHandler class handelt aan de hand van het huidige scherm de input op een bepaalde manier af.
ScreenHandler	De ScreenHandler class handelt alle schermacties af zoals het overgaan naar een ander scherm.
Screen	De Screen class is de superklasse dat de basis functionaliteiten van de schermen bevat.
StartScreen	De StartScreen class is het scherm waar de menu items wordt getekend.
SessionScreen	De SessionScreen class is het scherm waar de lijst van sessies wordt getekend.
LoadScreen	De LoadScreen class is een laadscherm dat wordt weergegeven tussen verschillende sessies.
LobbyScreen	De Lobbyscreen class is het scherm waarop de spelers komen zodra zij een sessie joinen
ConfigurationScreen	De ConfigurationScreen zijn de schermen waarbij de configuratie wordt weergegeven.
EditorScreen	De EditorScreen class is het scherm waar de gebruiker zijn agent kan programmeren, verschillende vragen worden getekend.
GameScreen	De GameScreen class is het scherm waarop de speler het spel kan spelen. Dit scherm is onderverdeeld in 3 aparte schermen genaamd GameStatScreen, GameChatScreen en GameWorldScreen
GameStatScreen	De GameStatScreen class is waar alle statistieken van de speler worden getoond in de console.
GameChatScreen	De GameChatScreen class is waar alle systeem berichten en chat berichten worden getoond in de console.
GameWorldScreen	De GameWorldScreen class is waar de wereld wordt getoond in de console.
Question	Een serie aan variabele die de vragen en juiste antwoorden zijn voor het configureren van de agent.
BaseVariabeles	Een simpelton waar alle goede variabele in staan die de pipeline accepteert.

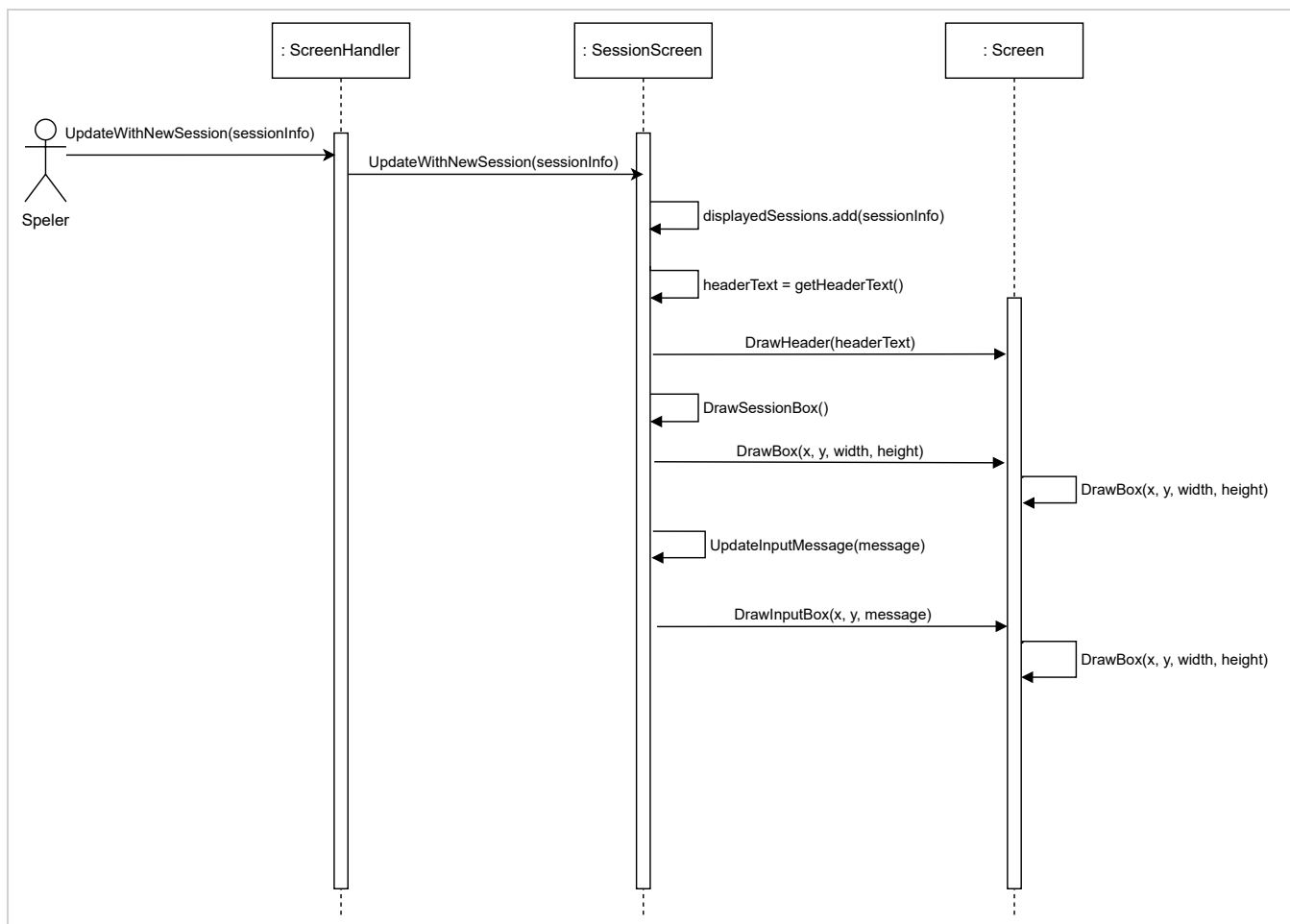
3.10.5. Sequence Diagrams

Om het proces van de schermovergangen te verduidelijken, is er een Sequence Diagram opgesteld voor de overgang en het tekenen van het startscherm (zie [Figuur 1 \(Design Sub-System User Interface\)](#)). Ook wordt er door middel van een Sequence Diagram het proces van een huidig scherm bijwerken met nieuwe data toegelicht. In dit geval wordt dit beschreven voor het bijwerken van het sessie scherm [Figuur 2 \(Design Sub-System User Interface\)](#).

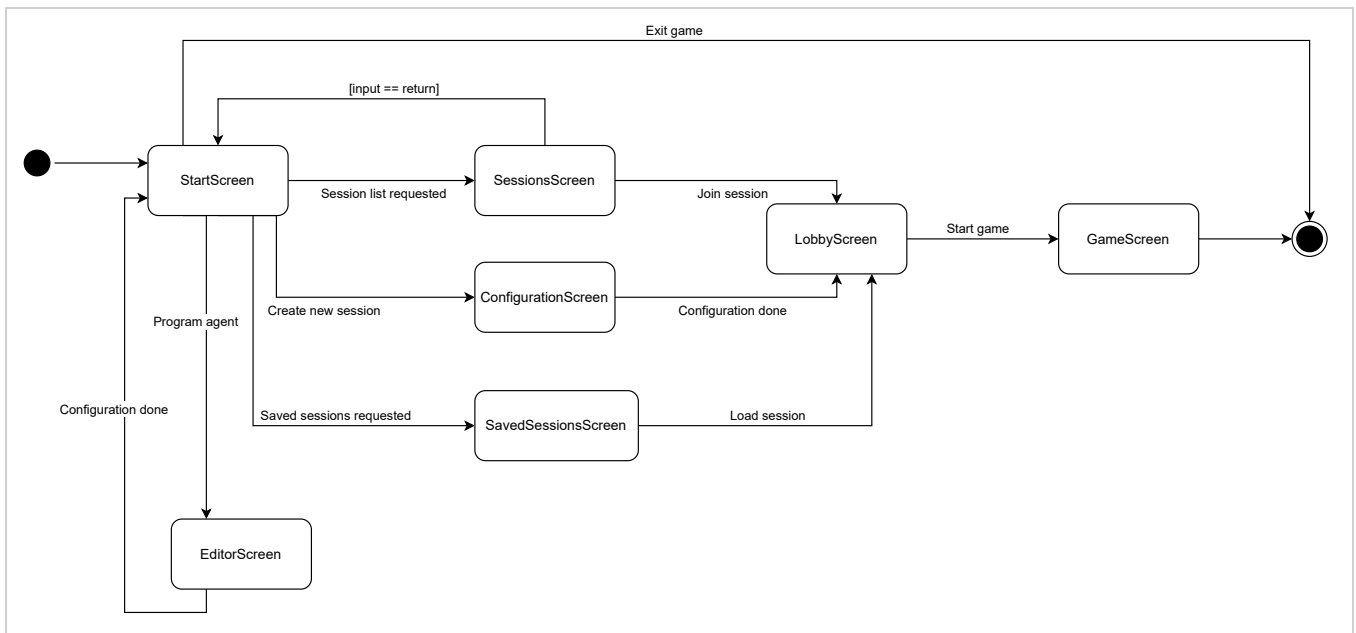
Figuur 35: Sequence Diagram Schermovergang



Figuur 36: Sequence Diagram Bijwerken Scherm inhoud



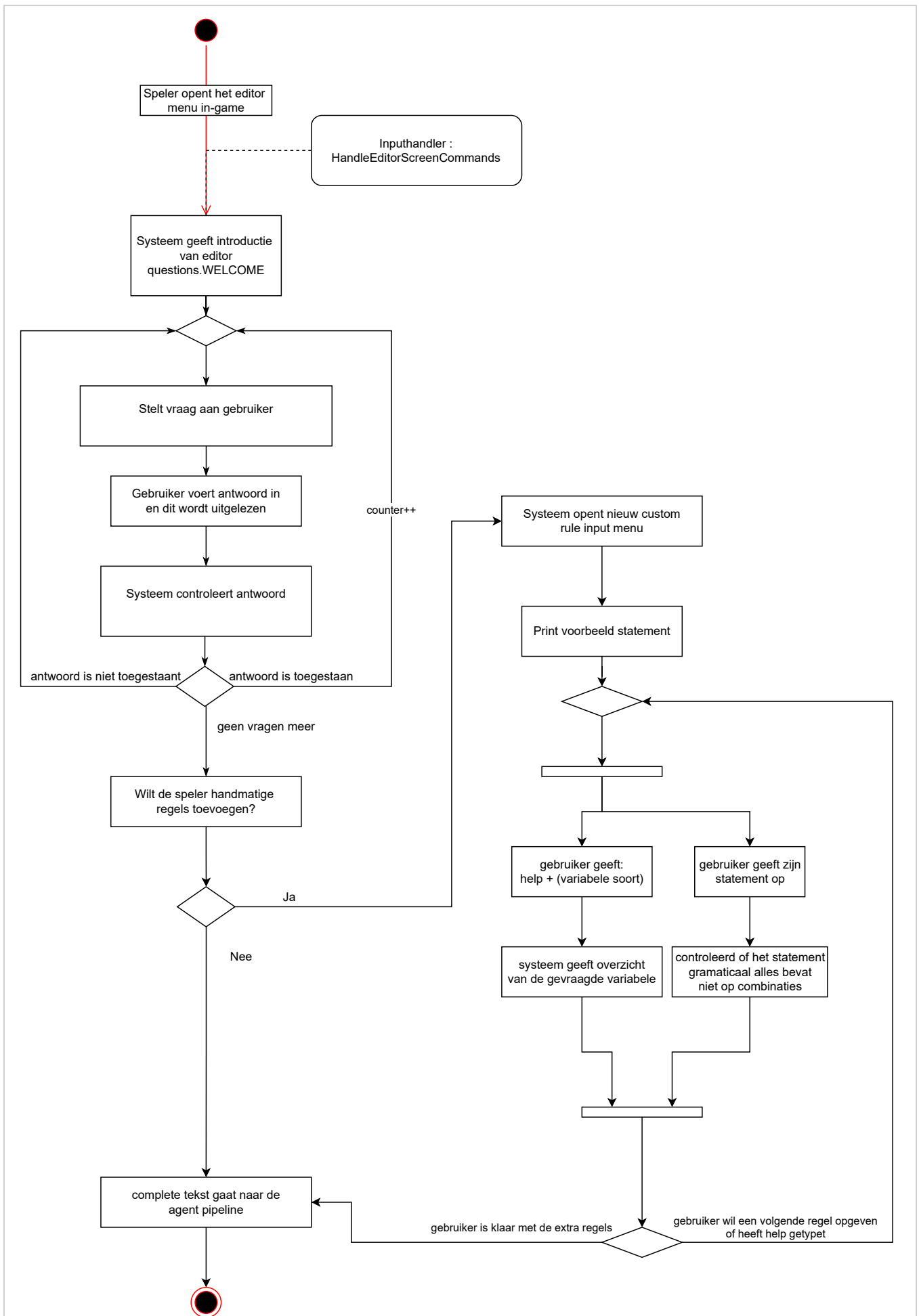
3.10.6. Activity and State Diagrams



Figuur 37: State Diagram User Interface

3.10.7. Activity diagram logica Editor screen

Figuur 38: activity diagram editor



3.10.8. Design decisions made for the sub-system

Bij het realiseren van de User Interface zijn een aantal keuzes gemaakt. Zie de tabellen hieronder.

Tabel 42: Decision State Pattern

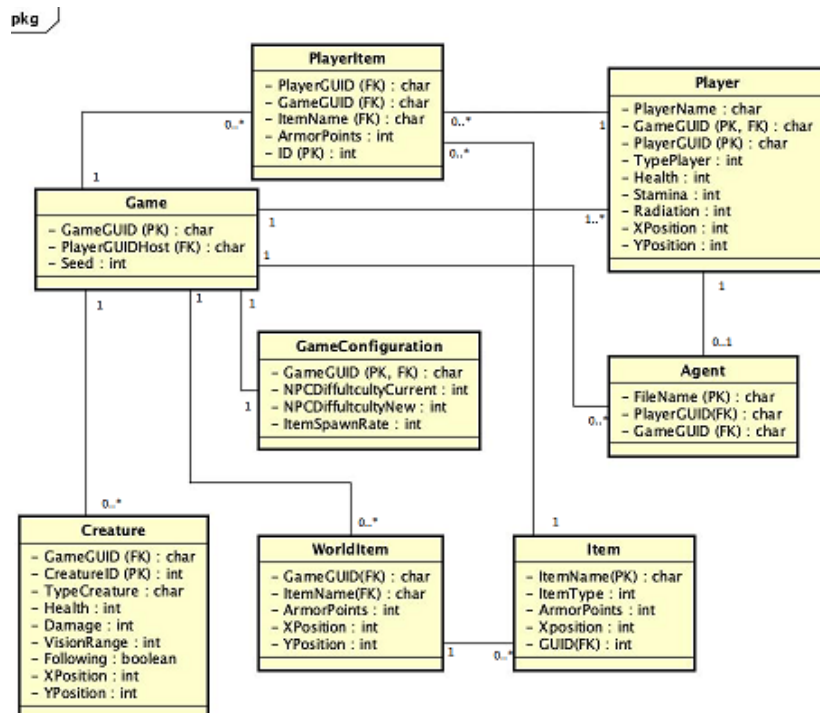
Decision	Description
Problem/Issue	Er moet onderscheid gemaakt worden tussen de verschillende schermen. Ook moet er makkelijk geschakeld worden naar een nieuw scherm.
Decision	Het gebruik van het State Pattern.
Alternatives	Chain-of-responsibility pattern, logica van schermen bij desbetreffende klassen schrijven.
Arguments	Er is gekozen om het tekenen los te koppelen van andere logica, met het State Pattern is het mogelijk om het systeem zich in een bepaalde state te laten bevinden en voor die state specifieke console onderdelen te tekenen. Ook zou het met State Pattern specifieke aanpassingen binnen die state verwerkt kunnen worden zonder dat dit effect heeft op de andere states. Met het State Pattern is het mogelijk om gemakkelijk van schermen wisselen.

Tabel 43: Decision UI Library

Decision	Description
Problem/Issue	Indien mogelijk zou het tekenen makkelijker gemaakt kunnen worden door een library.
Decision	Het tekenen binnen de console wordt zelf uitgevoerd met handgeschreven methoden.
Alternatives	Terminal.Gui of andere console-based UI toolkits.
Arguments	Er is gekozen om zelf de logica voor het tekenen te schrijven binnen de verschillende scherm classes. Dit is gedaan omdat Terminal.Gui en mogelijk ook andere UI toolkits niet binnen de applicatiestructuur te integreren is. Het is niet mogelijk om de input logica en de logica van het tekenen los te koppelen. Zo is het de bedoeling bij Terminal.Gui om de inputafhandeling direct aan een getekend onderdeel te koppelen. Door zelf de onderdelen op een eenvoudige manier te tekenen hoeft de applicatie structuur niet gewijzigd te worden, wat wegens tijdsdruk zwaarder weegt dan het tekenen makkelijker te maken.

3.11. Database Design

In [Figuur 1: Databaseontwerp \(Database Design\)](#) valt het gekozen databaseontwerp te zien. In [Tabel 1: Glossary database \(Database Design\)](#) wordt uitgelegd wat de verschillende tabellen allemaal moet kunnen uitvoeren.



Figuur 39: Databaseontwerp

Tabel 44: Glossary database

Tabel	Uitleg
Game	Slaat het opgestarte spel op en geeft aan dat hij zelf de host is, zodat bij het opnieuw opstarten de juiste gegevens opgehaald kunnen worden.

Tabel	Uitleg
Player	Wordt gebruikt om de spelers die in het spel zitten op te slaan met de bijbehorende gegevens. Wanneer een speler iets aanpast wordt dit in de database aangepast.
Agent	Is geconfigureerd door een speler.
GameConfiguratie	Voordat het spel gespeeld wordt kunnen er een aantal instellingen ingesteld worden, ook tijdens het spelen kunnen nog een aantal instellingen aangepast worden.
Item	Zijn items die op een positie staan.
World item	Slaat op welke items in een spel zitten.
Creature	Houdt alle gegevens bij van een character die in een wereld gespawnt is.
PlayerItem	Houdt bij welke items een speler in een spel allemaal heeft gehad.

3.11.1. Design decisions related to the database

De databasekeuze is gevallen op LiteDB. De onderbouwde keuzes zijn terug te lezen in [hoofdstuk 6.5.11](#) van het SAD.

Wel zaten een aantal extra nadelen aan het gebruik van LiteDB. Zo werkte de foreign keys niet in onze implementatie, daarom is ervoor gekozen om de foreign keys niet letterlijk aan te geven, maar ze zo wel in te voegen in de database. Dus bij een speler wordt altijd nog een gameGUID ingezet zonder de connectie met de ander tabel, hierdoor kan het fout gaan dat een speler in een spel komt wat niet bestaat. Maar door de implementatie goed toe te passen is de kans hierop heel klein. Zo worden de spelers alleen bij de host in de database gezet tijdens het aanmaken van het spel. De spelers hoeven zelf niets in de database te zetten.

De host gaat dan als eerste een spel-ID aanmaken aan de hand van zijn sessie-ID dit zet hij dan in de database. Omdat de sessie-ID altijd hetzelfde is tijdens een sessie kan de host steeds opnieuw de sessie-ID opvragen en gebruiken bij het toevoegen in de database van de spelers. Zo is de kans heel klein dat dit fout gaat.

Door gebruik te maken van C# Linq kan alsnog de tabellen met elkaar gejoint worden aan de hand van de overeenkomende ID's.

Er wordt niet specifiek aangegeven hoe lang een string die in de database gezet moet worden mag mogen. Dit omdat dit over het algemeen al van tevoren gecheckt wordt. Wel kunnen er problemen voorkomen met het instellen van een spelnaam of een spelernaam. Wanneer dit te lang is of rare tekens gaat bevatten kan dit voor problemen zorgen, hier is verder niet over nagedacht of acties mee ondernomen.



Powered by a free **Atlassian Confluence Community License** granted to DDOA. Evaluate
Confluence today.

This Confluence installation runs a Free Gliffy License - Evaluate the Gliffy Confluence Plugin for your
Wiki!

