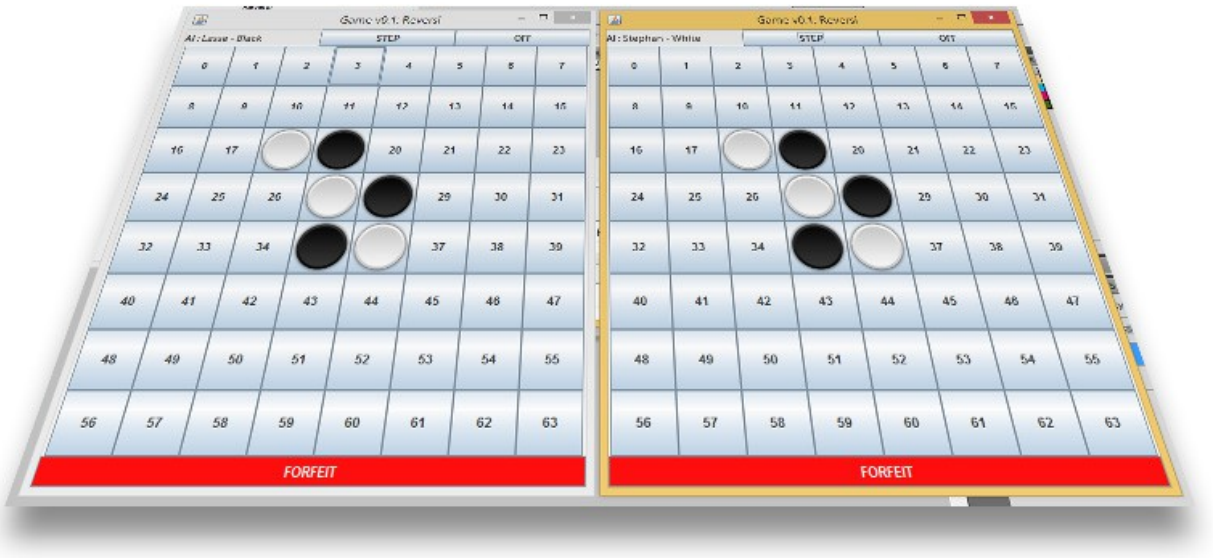
Eindverslag

Two-Player Game Framework



Thema:	2.3 Software Engineering
Opdracht:	Leertaak 6 (Two-Player game Framework)
Docent:	L.W. Bialek
School:	Hanze Hogeschool Groningen
Auteur:	Stephan E.G. Veenstra
Datum:	17-04-14
Leden:	Henderikus Harms (378471) Lasse Benninga (379496) Michel Jansma (388689) Stephan Veenstra (379241)
Versie:	1.0

Contactinformatie

GVB-verzekeringen (opdrachtgever)

Contactpersoon:

Dhr L.W. Bialek
Zernikeplein 11
9747 AS, GRONINGEN
l.w.bialek@pl.hanze.nl

GoFrag Yourself (uitvoerder)

Project Liaison Officer:

Michel Jansma
Prof. Boermastraat 53
9781 JR BEDUM
m.jansma@st.hanze.nl

Projectlead:

Stephan E.G. Veenstra
Wibenaheerd 282
9736 PT, GRONINGEN
s.e.g.veenstra@st.hanze.nl

AI specialist:

Henderikus B. Harms
Hoofdweg 30
9333 PB LANGELO
h.b.harms@st.hanze.nl

Documentation Engineer:

Lasse Benninga
Grote Leliestraat 1
9712 SM GRONINGEN
l.s.w.benninga@st.hanze.nl

Versiebeheer

Versie	Datum	Auteur	Omschrijving
0.1	04-04-14	Stephan E.G. Veenstra	– Opzet + invullen contactgegevens
0.1.1	05-04-14	Stephan E.G. Veenstra	– Verder opzetten van de structuur
0.2	14-04-14	Stephan E.G. Veenstra	– invulling AI – aanvulling Grafische User-Interface – invulling Conclusie met aanbevelingen
0.3	15-04-14	Stephan E.G. Veenstra	– Invulling Use Cases
0.4	16-04-14	Stephan E.G. Veenstra	– Verwerken van onderzoeken/research – Verwerken en samenvatten van eerdere documentatie
1.0	17-04-14	Stephan E.G. Veenstra	– Laatste details en aanpassingen

Inhoudsopgave

1 Inleiding

GVB-VERZEKERINGEN heeft GoFrag Yourself gevraagd een framework te ontwikkelen om zijn klanten spellen tegen elkaar te laten spelen. Dit voor de nieuwe campagne van het bedrijf.

GoFrag Yourself heeft deze opdracht aangenomen en heeft zich in de gedurende periode van vier week bezig gehouden met het ontwikkelen van een proof-of-concept voor dit framework.

Als oplevering is een werkende proof-of-concept van het framework afgesproken die beschikt over de spellen Tic-Tac-Toe en Othello.

Dit project verslag is samen opgeleverd met het opgeleverde product.

In dit document worden allereerst in hoofdstuk twee de eisen besproken waaraan het framework moet voldoen.

In hoofdstuk drie wordt de architectuur besproken van het framework. Hierin worden de onderdelen van het framework behandeld met daarbij de benodigde UML-diagrammen.

Vervolgens wordt het ontwerp nader toegelicht in hoofdstuk vier. Daarbij komen de mock ups en use cases aan bod.

Dan in hoofdstuk vijf bespreken we de implementatie met voorbeelden van cruciale code die nodig is om het framework te realiseren.

In hoofdstuk zes bespreken we hoe de applicatie is getest met de nodige bevindingen.

Omdat nog niet alles te realiseren was en er tijdens de ontwikkeling van het framework nog nieuwe dingen naar boven kwamen, hebben we in hoofdstuk zeven een conclusie met eventuele aanbevelingen voor het systeem beschreven. Deze zouden in de toekomst toegepast kunnen worden.

Tot slot volgt er een evaluatie van de samenwerking in hoofdstuk acht.

2 Eisen

Door GVB-VERZEKERINGEN zijn een aantal eisen opgesteld waaraan het framework moet voldoen. Eisen zijn te onderscheiden in functionele en niet-functionele eisen. De functionele eisen hebben betrekking tot de werking van het framework, de niet-functionele eisen hebben betrekking tot de interactie tussen de gebruiker en het framework.

De eisen hebben een identificatie-code en een prioriteitsmarkering. Deze prioriteitsmarkeringen zijn:

- M(ust Have), deze eis is cruciaal voor het slagen van het project.
- S(hould Have), deze eis heeft een hoge prioriteit maar is niet cruciaal voor het project.
- C(could Have), deze eis kan worden geïmplementeerd wanneer er tijd over is.
- W(on't Have), deze eis wordt in deze versie niet geïmplementeerd.

2.1 Functionele eisen

Uitbreiden van het framework	Code:	F1
Het moet mogelijk zijn om het framework uit te breiden zonder dat de interne werking te veranderen.		
	Prioriteit:	M

Nieuwe spellen toevoegen	Code:	F2
Spellen die gemaakt worden voor het framework, moeten voldoen aan het al bestaande protocol van de StrategicGameServer.		
	Prioriteit:	M

Tic Tac Toe (boter, kaas en eieren)	Code:	F3
Voor het proof-of-concept moet het spel tic tac toe (boter, kaas en eieren) worden gespeeld.		
	Prioriteit:	M

Reversi (othello)	Code:	F4
Voor het proof-of-concept moet het spel reversi (othello) worden gespeeld.		
	Prioriteit:	M

Kiezen wie er begint	Code:	F5
De speler moet kunnen kiezen wie er begint.		
	Prioriteit:	W

Symbool kiezen	Code:	F6
De speler moet kunnen kiezen met welk symbool hij/zij speelt.		
	Prioriteit:	S

Speler vs Speler	Code:	F7.1
Twee spelers moeten tegen elkaar kunnen spelen.		
	Prioriteit:	M

Speler vs Computer	Code:	F7.2
Een spelers moet tegen een computer kunnen spelen		
	Prioriteit:	M

Computer vs Computer	Code:	F7.3
Twee computers moeten tegen elkaar kunnen spelen.		
	Prioriteit:	M

2.2 Niet-functionele eisen

GVB-VERZEKERINGEN heeft geen niet-functionele eisen gedefinieerd.

3 Architectuur

In dit hoofdstuk wordt de architectuur van het framework beschreven. Er wordt omschreven hoe het framework is opgebouwd met daarbij verduidelijkingen in de vorm van diagrammen.

3.1 Packages/modules

Hier worden de verschillende packages/modules beschreven die zich in het framework bevinden.

3.1.1 Framework

De framework-package bevat alle klassen die nodig zijn voor de client om te werken. Het bevat onder andere klassen die de verschillende schermen tekenen, communiceert met de StrategicGameServer en zorgen voor de werking van de AI.

De mappen- /folderstructuur van het framework is als volgt:

- src (sourcecode van het framework)
  - framework.ai
  - framework.controller
  - framework.event
  - framework.gui
  - framework.model
  - framework.utils
    - framework.utils.dto
  - framework.view
- gamemodules (de losse AI-modules)
- game_data (de door de gebruiker aangemaakte borden)
- game_resources (afbeeldingen voor in het bord)

3.1.2 AI-modules

De AI-modules zijn losse onderdelen die in het framework geplugd kunnen worden. Deze modules bevatten de AI-regels voor bepaalde spellen. Het framework gebruikt deze AI om:

1. Te controleren of de door de gebruiker opgegeven zet valide is, om zo te voorkomen dat ongeoorloofde zetten worden doorgestuurd naar de server.
2. De beste zet te bepalen voor de automatische zet wanneer de AI 'aan' staat.

De modules extenden een abstracte klas uit het framework zodat iedere AI, die voldoet aan de abstracte klasse, door het framework gebruikt kan worden.

Een AI module bestaat uit één enkele file /.jar die in runtime wordt ingeladen. Later zou deze kunnen worden uitgebreid met wellicht een eigen view. Voor nu worden de views gemaakt door het framework.

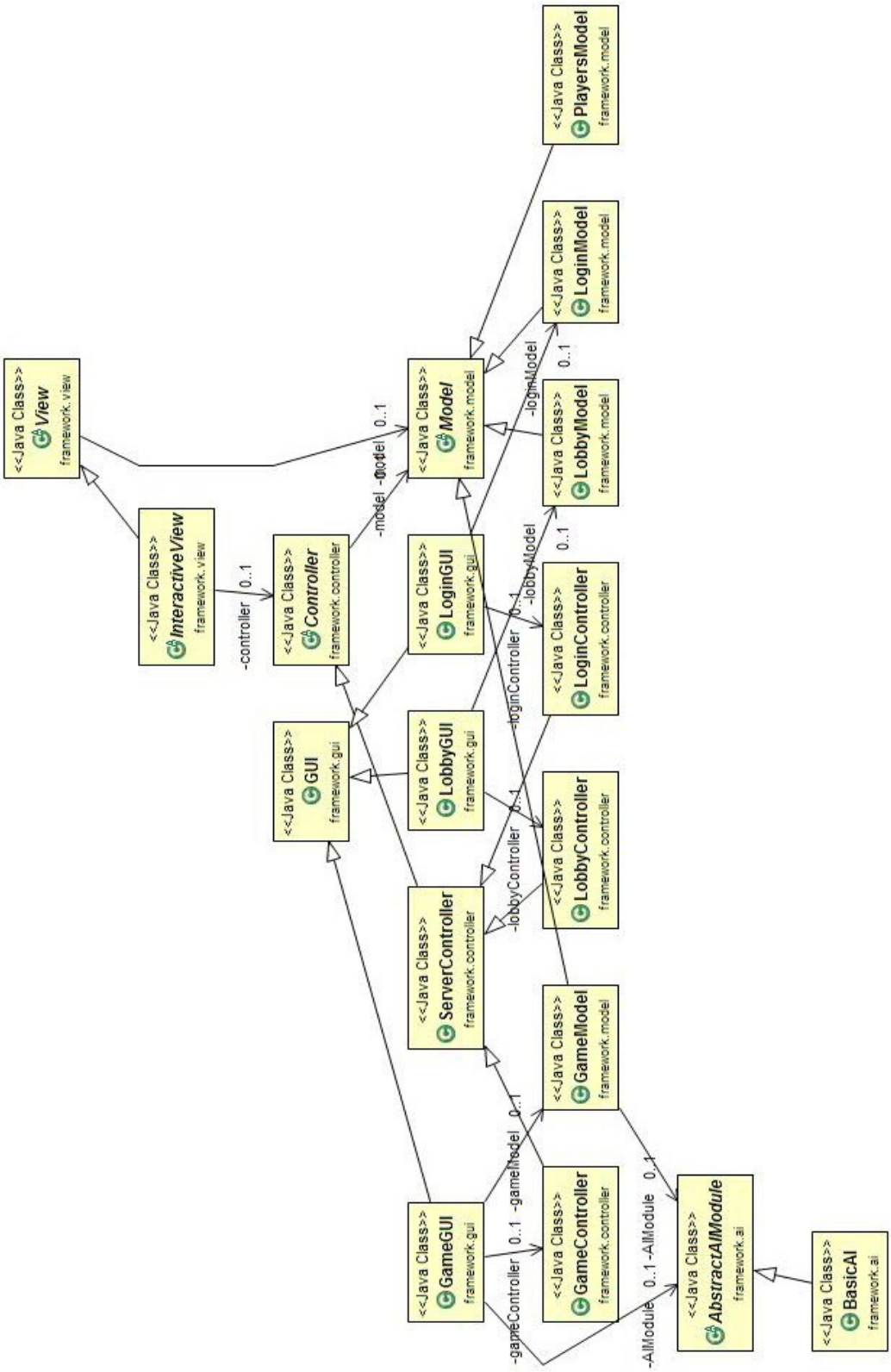
3.1.3 Gamemodules

Gamemodules zijn modules die worden ingeladen door de server en gebruikt worden om

spellen te kunnen spelen. De architectuur van deze modules hoeft niet meer gemaakt te worden omdat deze al bestaat en gebruikt wordt door de server.

3.2 Klassendiagram

Om de structuur / architectuur te verduidelijken volgen hieronder het klassendiagram van het framework. De AI-modules bestaan uit één klasse die de klasse 'AbstractAIModule' extenden.



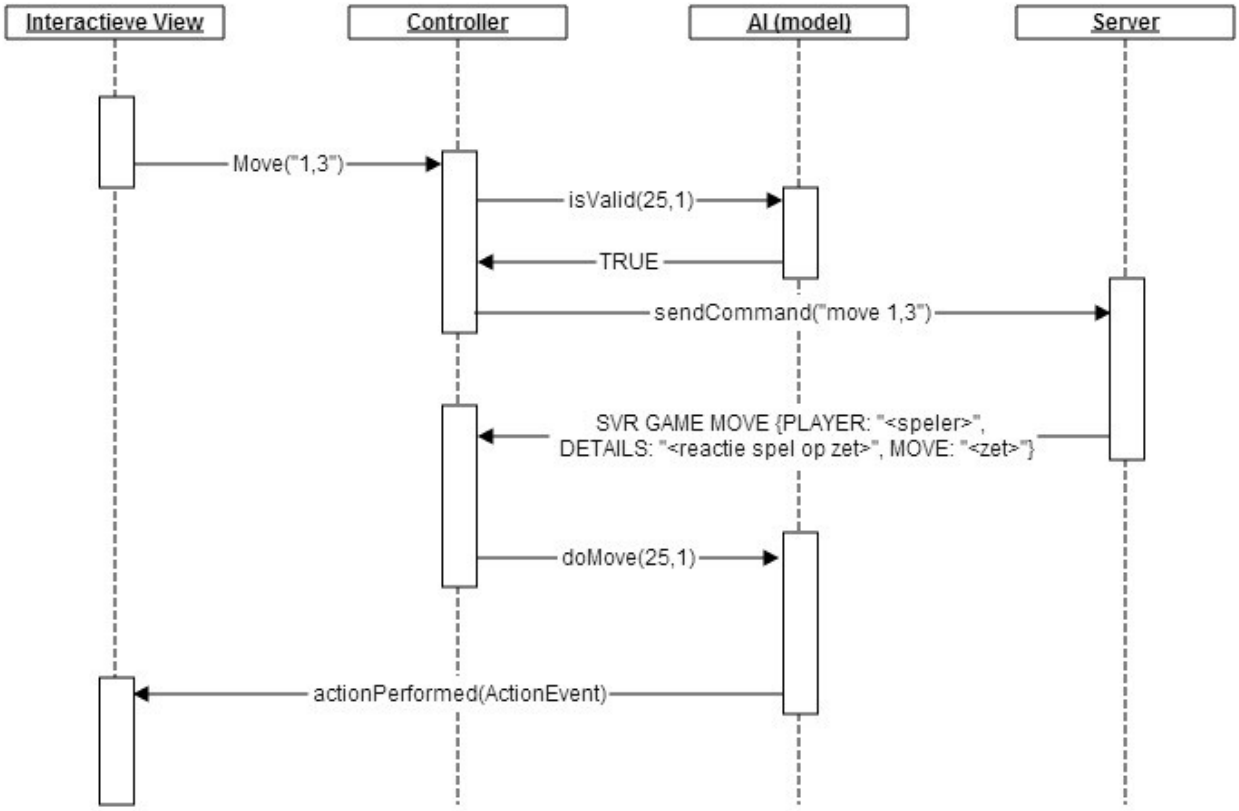
Afbeelding 1: Klassendiagram, framework

3.3 Sequencediagram met toelichting

In de volgende sequencediagram worden de relaties tussen klassen aangegeven tijdens het uitvoeren van een bepaalde aan actie. Er is een voorbeeld gegeven van één actie. Niet alle acties zijn opgenomen omdat deze vrijwel niet tot weinig verschillen.

3.3.1 Versturen van het 'move' -commando

Het versturen van een commando begint met een (subklasse van een) interactieve view. Deze view roept een functie aan van de gekoppelde controller. De controller controleert op zijn beurt aan de hand van de ingeladen AI of de zet gedaan mag worden of niet. Wanneer de zet is goed gekeurd, wordt het commando aangemaakt en doorgestuurd naar de server. Wanneer de server een opdracht heeft ontvangen gaat die deze verwerken en verstuurd een response. De response komt binnen bij de controller, die dan vervolgens aan de hand van dat commando acties gaat afhandelen op de model. De model notificeert vervolgens de geregistreerde views met een ActionPerformed call.



4 Ontwerp

In dit hoofdstuk wordt het ontwerp behandeld van het framework. Hierin wordt de Grafische User-Interface (GUI) nader toegelicht, wordt het client-server-protocol beschreven en het idee achter de AI uitgelegd.

Het uiteindelijke resultaat van het grafisch ontwerp en hoe de flow van het programma is geworden, is te vinden in de handleiding (bijlage H).

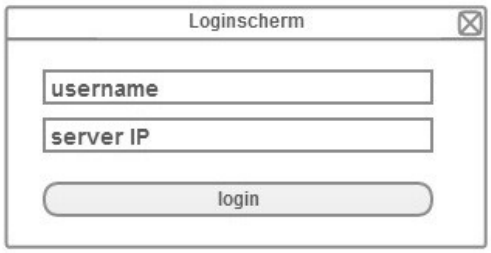
4.1 Grafische User-Interface

Voor de GUI (Grafische User-Interface) is er een onderscheid gemaakt tussen drie hoofdschermen. De reden hiervoor is dat een bepaald aantal functies niet langer relevant zijn op het moment dat een bepaalde fase in gang is gezet. Er is onderscheid gemaakt tussen: de login, lobby en game. Hieronder worden deze verschillende schermen/fasen nader toegelicht.

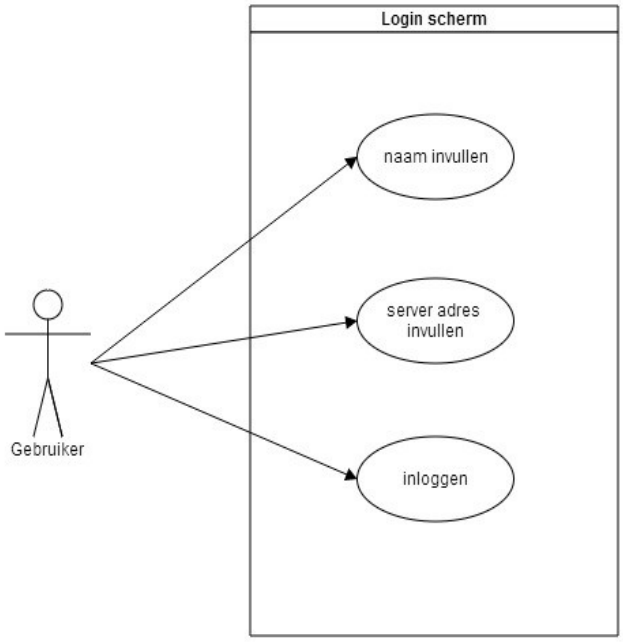
De use-cases voor de GUI zijn te vinden in de bijlagen (A: use cases).

4.1.1 Login

Wanneer de applicatie wordt gestart verschijnt dit scherm. In dit scherm kan de gebruiker enkel proberen in te loggen. Omdat de gebruiker verder eerst nog niets anders 'mag' doen met de applicatie dan proberen in te loggen, kunnen we een eenvoudig klein scherm tonen die intuïtief is voor de gebruiker.



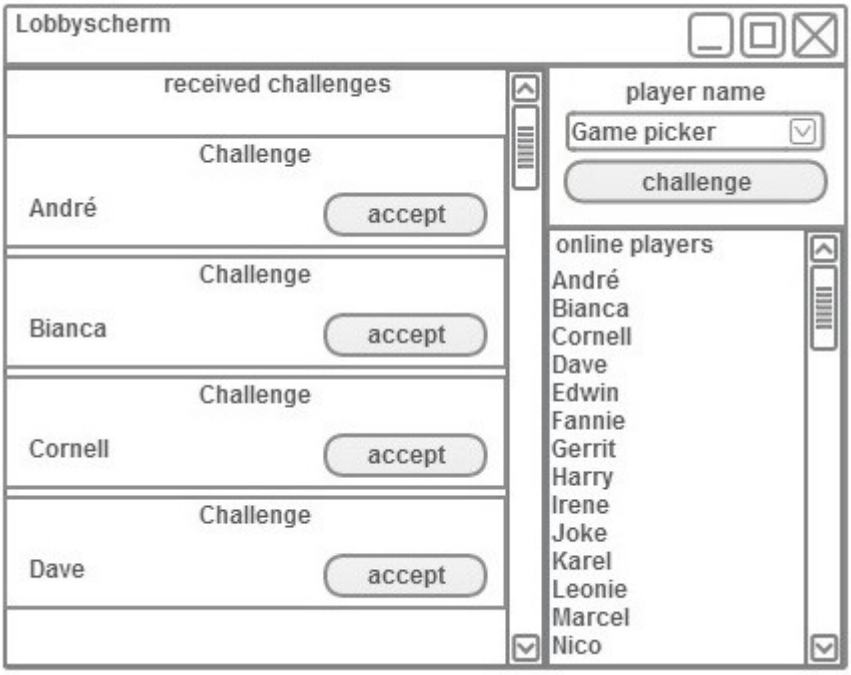
Afbeelding 2: Mock up, login scherm



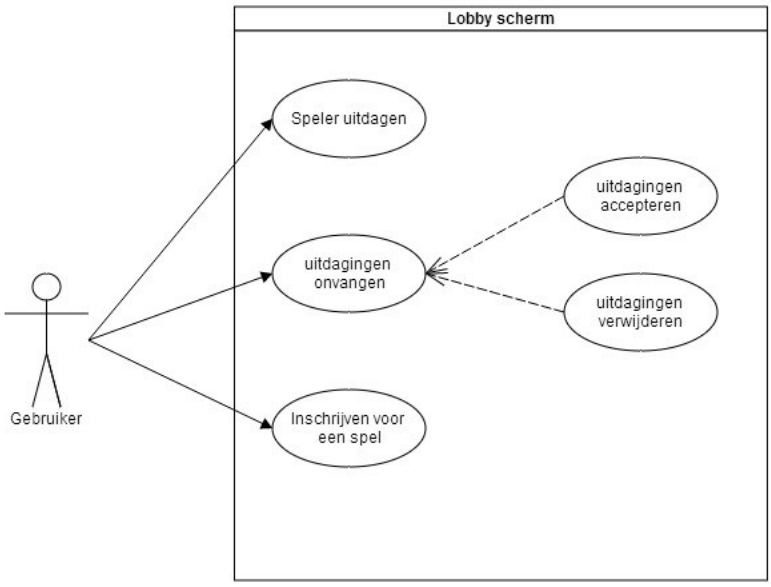
Afbeelding 3: use case diagram, login scherm

4.1.2 Lobby

Zodra de gebruiker is ingelogd zal het loginscherm verdwijnen en komt de lobby tevoorschijn. De gebruiker kan nu andere gebruikers uitdagen voor een game naar keuze. Daarnaast kan de gebruiker ook uitdagingen ontvangen. De uitdagingen kunnen verwijderd of geaccepteerd worden.



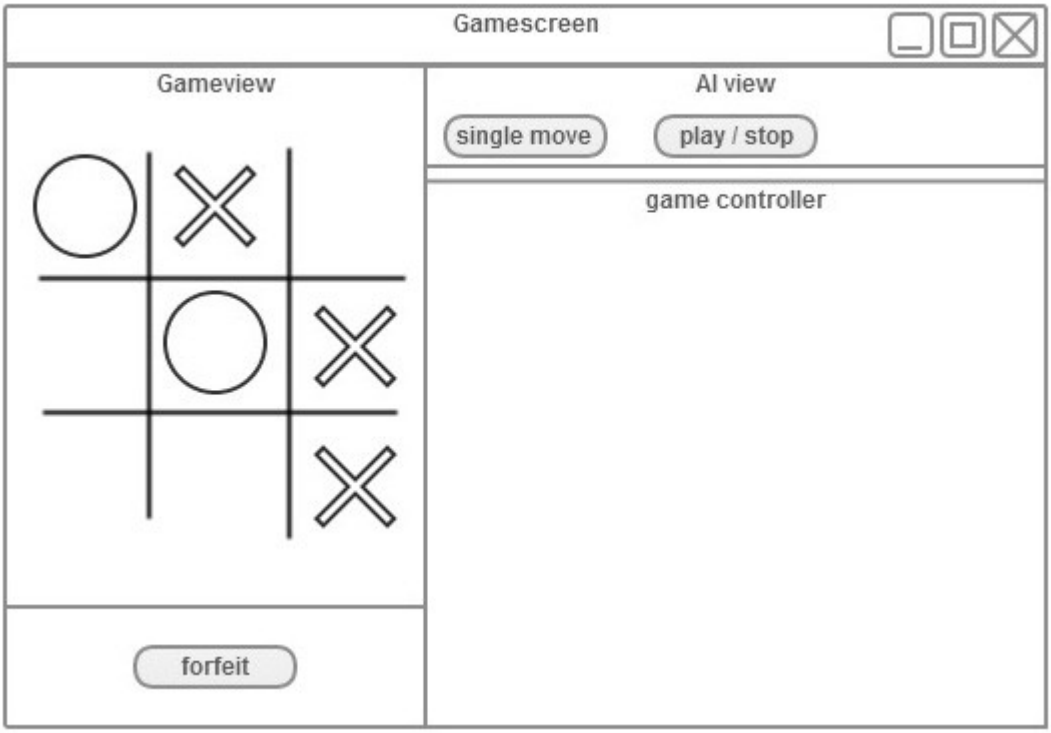
Afbeelding 4: Mock up, lobby scherm



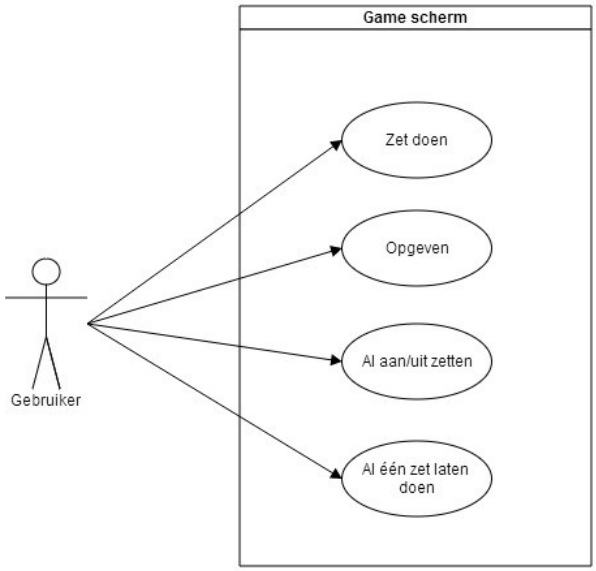
Afbeelding 5: Use Case, lobby scherm

4.1.3 Game

Wanneer de gebruiker een game accepteert, of een andere gebruiker accepteert een uitdaging van de gebruiker, dan zal de lobby verdwijnen en de Gamescherm geladen worden. In dit scherm komen de functies beschikbaar die betrekking hebben tot het spelen van een spel, zoals een zet doen of opgeven.



Afbeelding 6: Mock up, Game scherm



Afbeelding 7: Use Case Diagram, Game scherm

4.2 Client-server protocol

Voor de communicatie tussen het framework en de server is er gebruik gemaakt van het al bestaande protocol van de StrategicGameServer. Dit protocol bevat een aantal commando's van het inloggen in de server tot het doen van een zet. Het protocol van de server is omschreven in de documentatie van de server[Strategic Game Server].

4.3 AI

In het framework kunnen losse modules geplugd worden. Deze losse modules bevatten spel specifieke regels. Het framework gebruikt deze regels om onder andere de 'best move' (beste zet) en 'valid move' (toegestane zet) te bepalen. De AI-modules extenden de abstracte AI-klasse zodat het framework iedere module die aan die klasse voldoet kangebruiken.

De AI wordt voor twee doeleinden gebruikt. Namelijk voor a) de zet van de gebruiker te valideren zodat er voorkomen wordt dat er foutieve invoer wordt doorgestuurd naar de server en b) door de beste zetten te versturen op het moment dat de gebruiker heeft aangegeven dat het framework zelf de zetten mag gaan doen.

Het kan ook voorkomen dat er een spel wordt gestart waarvoor aan de client-side geen AI-module beschikbaar is. Als dat het geval is wordt er een 'basis AI' geladen worden die minimale checks bevat voor de functies uit de abstracte AI-klasse.

De twee belangrijkste methoden uit de AI-modules (getBestMove & isValidMove) worden als volgt ingevuld:

1. **getBestMove**
   Geeft de eerst mogelijke zet (valid move) die gevonden wordt terug. Omdat er geen uitgebreide regels bekent zijn betekent dit niet dat het daadwerkelijk de beste move is.
2. **isValidMove**
   Kijkt alleen of de gewenste zet valt binnen het veld. Er wordt verder geen rekening gehouden met andere spelregels omdat deze simpelweg niet bekent zijn.

Wanneer er geen AI-module beschikbaar is voor het spel dat gespeeld gaat worden moet de gebruiker zelf rekening houden met de spelregels. Het framework kan niet valideren of de zet van de gebruiker wel mag/kan.

Voor de oplevering zijn er twee AI geschreven. Deze AI zijn voor de spellen 'Tic-Tac-Toe' en 'Othello'. Voor beide spellen is er onderzoek gedaan naar de spelregels. Dit onderzoek is vastgelegd en te vinden in de bijlagen (E & F).

5 Implementatie

In dit hoofdstuk wordt de implementatie van het framework behandelt. Ten eerste wordt uitgelegd welk design-patern er is gekozen. Daarbij wordt toegelicht hoe deze is geïmplementeerd in dit project.

Vervolgens komen er cruciale stukken code aan bod met een omschrijving van wat ze doen.

5.1 Design patern: MVC

Als design-patern is er gekozen voor MVC (Model-View-Controller). MVC past perfect bij dit project omdat er veel verschillende *views* zijn die naar een model moeten 'luisteren'. In MVC heeft een model geen weet van de *views* en omdat dat zo is kan het framework eenvoudig worden uitgebreid met nieuwe/betere *views.*

Daarnaast wordt er meestal één *controller* gebruikt om de model aan te passen. De *controllers* van het framework kunnen door meerdere 'sources' (gebruiker/server/AI) worden aangeroepen.

MVC zorgt in dit geval voor een betere uitbreidbaar framework die in de toekomst tamelijk eenvoudig kan worden bijgewerkt.

Hieronder volgt wat de *models, views* en *controllers* zijn in het framework.

5.1.1 Models

De *models* in het framework zijn voornamelijk de AI-modules. Deze modules bevatten de huidige staat van het spel, het bord, de spelers en wie er aan de beurt is. De modules worden niet rechtstreeks beïnvloed/veranderd maar dit gebeurt via een *controller*.

De *views* die te maken hebben met het spel worden geregistreerd bij de module en worden bij een wijziging genotificeerd dat er een verandering heeft plaatsgevonden.

Daarnaast hebben ook de login- en lobbyscherm een eigen *model* om tijdelijke data in op te slaan tijdens de levens duur van dat scherm.

5.1.2 Views

De *views* zijn SWING klassen die de data van de *models* tonen op het scherm. Zij kunnen een *model* direct uitlezen om informatie over het *model* te tonen.

Daarnaast wordt er onderscheid gemaakt tussen twee soorten *views*:

1. **Gewone views**
   Doen niets anders dan data tonen van de *models* waarmee ze geassocieerd zijn.
2. **Interactieve views**
   Kunnen naast het tonen van data ook methodes aanroepen van een *controller*. Een voorbeeld van een interactieve view is het speelveld waarbij de 'buttons' methodes aanroepen van de ServerController.

5.1.3 Controllers

De *controllers* in het framework verwerken input vanaf de server en de gebruiker. Aan de hand van deze input passen zij de geassocieerde *models* aan. De GameController maakt bij het verwerken van Game-gerelateerde acties gebruik van de ingeladen AI-Module.

5.2 Code snippets

Hieronder volgen enkele stukken code met toelichting over de belangrijkste componenten van het framework.

5.2.1 Client-Server communicatie

De client en de server moeten met elkaar communiceren. Omdat niet duidelijk is wanneer welk van de twee gaat 'praten' moeten ze allebei voortdurend 'luisteren' naar binnenkomende commando's.

In het framework is er één thread die voortdurend bezig is met het luisteren naar de inkomende opdrachten. Deze worden opgevangen en afgehandeld naast de main-thread.

De inkomende commando's worden omgezet naar aanroepen van controller-acties.

GameServerListener klasse in framework.utils:

```
public class GameServerListener implements Runnable {

	public GameServerListener() {};

	/**
	 * Run while socket isConnected
	 * @method run
	 */
	@Override
	public void run() {
		while(ServerController.SOCKET.isConnected()) {
			this.readCommand();
		}
	};

	/**
	 * Read data from GameServer
	 * @method readCommand
	 */
	public void readCommand() {
		try {
			String line = ServerController.INPUT_READER.readLine();
			if(line != null) {
				//Print line
				System.out.println(line);
				//SERVER REQUEST RESPONSE
				ServerController._STACK.add(new
					ActionCommand(ServerController.CURRENT_LISTENER, line));
			}
		} catch(Exception e) {
			e.printStackTrace();
			System.out.println(this.getClass().getName() + " -> readCommand() -> " +
e.getMessage());
		};
	};

}
```

5.2.2 AI-modules

De losse modules die in het framework geplugd moeten worden bevatten de spelregels voor het framework. Zonder deze modules weet het framework niet wat de beste zet is of dat een gewenste zet uberhaupt mogelijk is.

Omdat het framework moet weten welke zetten mogelijk zijn en welke de beste zetten zijn moeten de modules een abstracte klasse extenden. Deze klasse heet de *AbstractAIModule*.

Het framework gebruikt de implementatie van de abstracte methoden *isValidMove()* en *getBestMove()* om te controleren of een zet mag voordat deze wordt verstuurd naar de server en wat de beste zet is in een bepaalde situatie.

Hieronder volgt de abstracte klasse *AbstractAIModule*.

```
public abstract class AbstractAIModule{

	protected int[][] board;

	private String gameType;
	private int width;
	private int height;

	protected final int EMPTY = 0;
	protected final int P1 = 1;
	protected final int P2 = 2;

	protected int currentPlayer = P1;

	protected final int UNKNOWN = 0;
	protected final int LOSE = 1;
	protected final int WIN = 2;
	protected final int DRAW = 3;

	public AbstractAIModule(int width, int height, String gameType) {
			this.width = width;
			this.height = height;
			this.gameType = gameType;
			board = new int[width][height];
	}

	public AbstractAIModule(String gameType){
			this.gameType = gameType;
	}

	public void setCurrentPlayer(int player) {
			this.currentPlayer = player;
	}

	public void togglePlayer() {
			this.currentPlayer = this.getOpponent();
	}

	public void setBoard(int width, int height){
			this.width = width;
			this.height = height;
			board = new int[width][height];
	}

	public int getWidth(){
			return this.width;
	}

	public int getHeight(){
			return this.height;
	}

	public String getGameType(){
			return this.gameType;
	}

	public int[][] getBoard() {
			return this.board;
	};

	public int getOpponent(){
			return getCurrentPlayer() == 1 ? 2 : 1;
	}
```

```
	/**
	 * Do a move
	 * @param player
	 * @param move
	 */
	abstract public void doMove(int player, int move);

	/**
	 * Get the current player
	 * @return
	 */
	public int getCurrentPlayer(){
		return this.currentPlayer;
	}

	/**
	 * Get the best move
	 * @param player
	 * @return
	 */
	public abstract Integer getBestMove(int player);

	/**
	 * Check if the given move is valid
	 * @param move
	 * @return
	 */
	public abstract boolean isValidMove(int player, int move);

	/**
	 * Get the state of the board
	 * @return
	 */
	protected abstract int getState(int player);
}
```

6 Testen

In dit hoofdstuk wordt er besproken wanneer en op welke manier het project is getest.

6.1 Aanpak van het testen

Het testen is verdeeld in verschillende fasen. Zo is er in de eerste fase het aangeleverde ServerProject getest met het daarbij geleverde protocol.

Vervolgens zijn er tijdens de implementatie op verschillende momenten test uitgevoerd op de huidige stand van zaken.

6.1.1 Pre-test: server

Om een beeld te krijgen van de werking van de server is er aan de hand van het server protocol getest door een aantal keer tegen elkaar te spelen.

Tijdens het testen is er geprobeerd:

1. In te loggen op de server.
2. Subscribe-en voor een game.
3. Een challenge versturen naar andere gebruikers.
4. Een challenge ontvangen.
5. Een ontvangen challenge accepteren.
6. Een ontvangen challenge verwijderen/wijgeren.
7. Een toernooi te starten.
8. Een juiste zet te doen.
9. Een onjuiste zet te doen.
10. Een spel forfeit-en/opgeven.

6.1.2 Tussentijdse tests

Op verschillende momenten tijdens het project is het framework getest. Deze test gingen voornamelijk over de manier waarop de onderdelen geïmplementeerd werden. De punten waarop is getest zijn:

1 Is de voorbedachte MVC structuur toepasbaar voor ons project?

2 Kunnen wij vanaf JAVA verbinden met de server?

- 2.1 Kunnen er commando's verstuurd worden naar de server?
- 2.2 Kunnen er commando's van de server worden ontvangen?

3 Is de GUI overzichtelijk/duidelijk/gebruiksvriendelijk?

4 Kan er aan de hand van de ontvangen data van de server een game gestart worden?

Naast deze test is er, afhankelijk van de voorkeuren van de programmeurs gekozen voor unittests om bepaalde algoritmes/methodes te testen of is er gekozen om vooral output te printen om methoden te testen.

6.1.3 Eindtest

Dit is de laatste test die vlak voor de oplevering om de status van het framework/project te kunnen bepalen. Tijdens deze test kwamen de vooraf afgesproken eisen aan bod. Aan de hand van deze test kan de kwaliteit bepaald worden.

6.2 Test resultaten

Hieronder volgen de test resultaten van de verschillende tests, te beginnen met de pre-test. Dan volgt de tussentijdse test en tot slot de eindtest.

6.2.1 Conclusie: Pre-test

Bijna alle verwachte handelingen kunnen correct worden uitgevoerd. Alleen het weigeren van een challenge is niet mogelijk door de speler zelf. De server kan wel challenges weigeren wanneer de speler al bezig is.

Zie bijlage B voor de complete test.

6.2.2 Conclusie: Tussentijdse Test

Tijdens de implementatie waren in grote lijnen de tests succesvol. Deze tests gingen niet in op de details, die werden door de programmeurs zelf gedaan d.m.v. Eerder uitgelegde manieren.

Aan de hand van de resultaten van deze tests werden de details van het project verder ingevuld.

Zie bijlage C voor de complete test.

6.2.3 Conclusie: Eind Test

Helaas zijn niet alle eisen gehaald voor de deadline van het project. De eisen die niet gehaald zijn hebben betrekking tot het spelen van Othello tegen de computer en het dynamisch inladen van de AI-modules.

Zie bijlage D voor de complete test.

7 Conclusie met aanbevelingen

Het is gelukt een werkende proof-of-concept te ontwikkelen voor een Multiplayer Game Framework binnen de afgesproken vier weken. Desondanks dat niet alles gerealiseerd kon worden voldoet het project aan bijna alle eisen. Tijdens de ontwikkeling van het framework kwamen er enkel punten naar boven die verder ontwikkeld kunnen worden mocht het framework verder ontwikkeld worden.

De voornamelijke struikelblokken tijdens het project waren:

1. **De beperkingen van de server en de beperkte documentatie die bij de applicatie geleverd was.** Hierdoor was het vooral veel proberen hoe de server zou reageren op bepaalde acties en problemen.
2. **De manier van afhandelen van binnenkomende data (vanaf de server).** Er ontstond op een gegeven moment een probleem waarbij ontvangen opdrachten door verkeerde klassen werden afgehandeld.

7.1 Overzicht van aanbevelingen

Tijdens de ontwikkeling van het framework zijn er een aantal uitbreidingen/ aanpassingen naar voren gekomen die mogelijk in de toekomst kunnen worden geïmplementeerd. Deze uitbreidingen en aanpassingen hebben te maken met onder ander het dynamischer maken van het systeem, het bevorderen van de verdere ontwikkeling van het framework en beleving van de gebruikers.

7.1.1 Verbeteren van de dynamiek van de gamemodules. (SUG01)

In het huidige systeem kunnen enkel spellen worden toegevoegd die voldoen aan een bepaald patroon (twee spelers, turnbased, etc..) Hierdoor wordt de mogelijkheid om nieuwe spellen toe te voegen enigszins beperkt. Daarnaast worden de modules voor de server en client los van elkaar ontwikkeld waardoor de spelregels dubbel bepaald worden. Het gevaar hierin is dat deze regels niet overeenkomen en er fouten ontstaan.

Oplossing
De server en de client gebruik laten maken van dezelfde module pakketten. Op deze manier voorkom je verschil in regels.

7.1.2 Synchroniseren van de regels (SUG02)

Op dit moment is er geen mogelijkheid om te bepalen of client en server de zelfde regels bevatten. Er zit totaal geen controle op en dit kan problemen veroorzaken door dat de client zetten gaat doorsturen die niet geaccepteerd worden door de server.

Oplossing
De server de 'regels' laten bepalen en de clients deze laten ophalen vanaf de server (patcher/ updater). Zo worden voor er gespeeld kan worden de regels gesynchroniseerd en kunnen daarover geen moeilijkheden ontstaan. In combinatie met suggestie SUG01 zou dit betekenen dat de module van de server lokaal kan worden opgeslagen voor gebruik in het framework.

7.1.3 Verbeteren van de vormgeving (SUG03)

Voor veel mensen/gebruikers is de visuele aantrekkingskracht van een applicatie vaak erg belangrijk. Op dit moment zijn er alleen programmeurs bezig geweest met het project waardoor het er nog was technisch en kaal uitziet.

Oplossing
Een persoon of instantie inschakelen die verstand heeft van vormgeving en applicaties om zo de applicatie een beter uiterlijk te geven.

7.1.4 Scheiden van server en client (SUG04)

Zoals de huidige staat van het framework vereist moet er een verbinding zijn met de server. Om de verschillende mogelijkheden (hum vs hum [F7.1], hum vs comp [F7.2] en comp vs comp [F7.3]) te kunnen spelen moeten er twee (of meerdere) clients gestart worden die via de server tegen elkaar kunnen spelen. Het is dus niet mogelijk om lokaal een spelletje tegen de computer te kunnen spelen zonder dat er een server gestart is. Dit zou echter wel een grote meerwaarde geven aan het framework.

Oplossing
De client ook lokaal functioneel maken door singleplayer en multiplayer van elkaar te onderscheiden. Dit kan door bijvoorbeeld de gebruiker pas te laten inloggen op het moment dat er daadwerkelijk multiplayer gespeeld gaat worden.

8 Evaluatie

Het samenwerken ging uitstekend. Vanaf het begin af aan was er duidelijk wie hoofdverantwoordelijk zou zijn bepaalde taken. Doordat de taken goed van elkaar gescheiden waren kon iedereen zelfstandig bezig. De afspraak was dat we elkaar zouden ondersteunen als een persoon ergens niet uit kwam. Dit resulteerde in goed overleg voordat bepaalde afspraken zouden worden gemaakt en implementaties zouden worden toegepast.

Er zou tijdens het project gewerkt worden op school. De werktijden waren vast gesteld op 9:30 tot 16:00. Op te laat komen stond een straf (boete van € 5,-). De uiteindelijk pot zal worden gebruikt voor een groepsactiviteit om het thema goed af te sluiten.

Bijlage G zijn 360 graden formulieren die door de individuele projectleden zijn ingevuld waarin zij elkaar hebben beoordeeld.

Literatuurlijst

Strategic Game Server: , Strategic Game Server - Protocol Documentatie,