6.1 Toelichting op het ontwerp

In bijgeleverde Visio bestanden zijn 3 tekeningen te vinden waarop onze Rolit applicatie is weergegeven. De tekeningen bevatten:

* Een overzicht van de relaties tussen alle klassen van onze applicatie.
* Een overzicht van alle klassen en hun variabelen en methoden die door de client worden gebruikt (hierin staan geen relaties aangegeven).
* Een overzicht van alle klassen en hun variabelen en methoden die door de server worden gebruikt (hierin staan geen relaties aangegeven).

## Functionaliteit server

De server beschikt over een klasse *‘MainView’* die een GUI verzorgt. Hier kunnen de poort, waarnaar de server moet luisteren, en de log van de server gevonden worden. Ook start de gebruiker de server via de GUI.

De *‘NetworkController’* start de ServerSocket. Als de poort in gebruik is, wordt dat aan de gebruiker getoond in de log. Hierop kan hij opnieuw een poortnummer invullen en de server starten.

De *‘NetworkController’* houdt een list bij van de verschillende Rolit-spelletjes die hij verzorgt.

De communicatieberichten van de server worden getoond in de log van de klasse *‘MainView’*, dit is een *‘java.swing.JTextArea’*.

De *‘NetworkController’* zorgt voor de afhandeling van netwerk berichten. Deze berichten komen binnen via de *‘ConnectionController’* van een bepaalde client. Commands die moeten worden verzonden, worden aan de *‘ConnectionController’* van de desbetreffende client(s) doorgegeven die er vervolgens voor zorgt dat het bericht bij de client(s) aankomt.

## Functionaliteit client

De GUI van de client wordt samengesteld door de klassen *‘ConnectView’*, *‘GameView’* en *‘LobbyView’*. Deze klassen bevatten de code om de GUI in een nieuw frame te laten zien.

De *‘ApplicationController’* zorgt er vervolgens voor dat de events, door de gebruiker gegenereerd, worden afgehandeld en zorgt ervoor dat de juiste view getoond wordt. In de *‘LobbyView’* heeft de gebruiker de mogelijkheid om te kiezen voor een computerspeler of om zelf te spelen. In geval dat de gebruiker kiest om de computer te laten spelen, zal de *‘AIController’* de zetten bepalen.

Door de *‘GameView’* te sluiten wordt de gebruiker uit de game gekickt (aan de kant van de server) en keert hij terug naar de *‘LobbyView’*. De *‘ApplicationController’* vangt het *‘WindowClosingEvent*’ af waarop hij de voorgenoemde acties onderneemt.

De gebruiker geeft via de *‘GameView’* zijn zet door aan de client. De *‘ApplicationController*’ zorgt ervoor dat het event wordt omgezet in een command voor de server en stuurt dit naar de *‘NetworkController’*. Die zorgt er vervolgens voor dat het naar de server wordt verzonden, die vervolgens zorgt dat de andere client(s) de zet door krijgen.

<punt 5>

De gebruiker geeft ook hier via de *‘GameView’* aan dat hij een hint wil. De *‘ApplicationController*’ vraagt in reactie op dit event aan de *‘AIController’* welke zet het beste is. Vervolgens zal de *‘ApplicationController’* dit aan de *‘GameView’* doorgeven, waardoor de gebruiker de hint kan zien.

Als het spel is afgelopen stuurt de server dit naar alle clients. Dit bericht komt binnen bij de *‘NetworkController*’. Die zal dit aan de *‘ApplicationController’* doorgeven, die in reactie hierop de game aan de client kant zal beëindigen en daarom de *‘GameView’* sluiten en *‘LobbyView’* zal tonen. Vanuit de *‘LobbyView’* kan de gebruiker een nieuwe game starten.

Als er een onverwachte gebeurtenis plaatsvindt zoals een gebruiker die plotseling weggaat, zal de server een *‘kick’*-commando versturen aan alle (overige) clients. Als dit eindigt in het einde van het huidige spel, zal de gebruiker op dezelfde manier de *‘LobbyView’* getoond worden.

Mocht de client verbinding verliezen met de server, dan zal hij hier bericht van krijgen en de *‘ConnectView’* wordt tevoorschijn gebracht.

## Overige functionaliteit

De chat-functie en challenge-functie zitten geïntegreerd in de *‘NetworkController’*-klassen van de client en de server. Aan de client kant is deze ook voor een deel te vinden in de *‘ApplicationController’* en uiteraard zorgen de views voor terugkoppeling naar de gebruiker.

## *Observer-* en *Model-View-Controller*-patronen

Uit onze packages wordt model-view-controller al voor een groot deel duidelijk. Het model is hier echter iets lastiger in te vinden. Ons model van Rolit is namelijk te vinden in *‘rolit.sharedModels’* en wordt door de *‘ApplicationController’* van de client respectievelijk de server geobserveerd.

Het model bestaat uit de volgende klassen.

* Game
* Board
* Gamer
* Slot

Door deze klassen wordt een spelletje Rolit gemodelleerd, uiteraard worden ze aangestuurd door de controllers.

Door middel van het Observer-patroon geeft de klasse *‘Game’* aan de *‘ApplicationController’* door dat er iets is veranderd in het model. Dit kan bijvoorbeeld een zet zijn die is gespeeld.

De views worden door de *‘ApplicationController’* aangestuurd. Op aangeven van de *‘NetworkController’* beslist de *‘ApplicationController’* welke view getoond moet worden aan de gebruiker. Vervolgens gaat de input van de gebruiker via de *‘ApplicationController’* naar de *‘NetworkController’*. Deze zorgt er op zijn beurt weer voor dat het naar de server verstuurd wordt.

## Netwerk protocol

Het netwerk protocol dat tijdens het werkcollege is besproken is volledig overgenomen, er zijn geen toevoegingen nodig geweest.