6.3 Testverslag

## Model

Het model is als geheel getest. Daarnaast is de klasse *‘Board’* ook afzonderlijk getest.

In de klasse *‘BoardTest’* wordt *‘Board’* getest, door het doen van een paar zetten.

De veel omvangrijkere test van het hele model, te vinden in de klasse *‘GameTest’*, test het model als geheel. Hiertoe worden drie games aangemaakt. Dit omdat er 3 verschillende mogelijkheden zijn: een spelletje met twee spelers, drie spelers of een spelletje met 4 spelers.

Vervolgens worden de drie games door de AI gespeeld. Het geheel is terug te vinden in de logfiles *‘2playerGame.txt’*, *‘3playerGame.txt’* of *‘4playerGame.txt’*. In deze tekstbestanden is een visuele representatie van het bord gelogd, na elke zet.

Op deze manier wordt het model als geheel getest. Het model bevat de volgende klassen: *‘Game’*, *‘Gamer’*, *‘Board’* en *‘Slot’*.

## Client en Server

De client en de server zijn getest op de commando’s die ze over het netwerk moeten versturen. Hiertoe is een klasse *‘ServerTester’* geschreven, die een client simuleert. Deze klasse verstuurt één voor één alle commando’s die de client kan versturen en de server dus moet kunnen afhandelen, zoals vastgesteld in het protocol.

Als de client een commando heeft verstuurd blijft hij wachten op antwoord van de server. Mocht hij dit niet krijgen, blijft hij in een loop zitten. In dit geval dient de gebruiker de test dus af te breken en te controleren waar het fout ging.

Verder is er een mogelijkheid tot handmatig testen van de commando’s, dit gebeurt door middel van de klassen *‘Client’* en *‘ClientGui’*, de interface *‘MessageUI’* en door een server te starten (via zijn *‘Main’* klasse).

Dit zijn klassen die niet voor onze Rolit-applicatie zijn geschreven, maar prima gebruikt kunnen worden om de letterlijke commando’s te sturen. De server stuurt dan in reactie hierop een antwoordt. De gebruiker dient hierbij zelf te controleren of dit aan zijn verwachtingen voldoet. Dit is de reden waarom de testklassen alleen in een vroeg stadium van het ontwerpen zijn gebruikt.

Bij beide manieren, zoals hierboven beschreven, worden alle klasse van de server als geheel getest.

## AI tests

Er is ook een test geschreven om onze slimme en minder slimme AI’s tegen elkaar te laten spelen.

Tijdens deze test wordt een spelletje op een server gespeeld door beide AI’s, deze klasse is gebruikt om verbeteringen in onze AI te testen. Deze test wordt gestart door de klasse *‘AITest’* te runnen. Deze klasse creëert 2 instanties van de *‘AITestApplicationController’*, een met een slimme AI en een met een simpele AI.

Impliciet wordt ook de server getest, omdat de AI’s verbinden met de server. Daarbij maken ze voor een groot deel gebruik van de bestaande client (*‘AITestApplicationController’* extends immers *‘ApplicationController’* in *‘rolit.client.controllers’*), wat betekent dat ook de *‘NetworkController’* van de client impliciet wordt getest. Deze tests voor de server en client zijn echter heel oppervlakkig.

## Systeemtesten

Bij het testen van het systeem als geheel, hebben we gelet op kleine foutjes, deze waren voornamelijk in de GUI te vinden. Andere fouten zouden zich moeten voordoen in onjuiste afhandeling van het protocol of een fout in het model betekenen.

Het testen van de GUI is voornamelijk gebeurd toen het systeem zover af was, dat er gespeeld kon worden. Onder andere resizen en het invullen van verschillende vormen van tekst en cijfers in de invul velden maakten hiervan deel uit. Bugs zoals het accepteren van een naam bestaande uit enkel spaties of een leeg tekstveld, hebben we zo kunnen verbeteren.