6.2 Toelichting per klasse

## Algemeen

De client en server packages zijn vergelijkbaar onderverdeeld. De client bestaat uit de volgende packages:

rolit.client

rolit.client.controllers

rolit.client.models

rolit.client.views

De server packages zien er hetzelfde uit, namelijk:

rolit.server

rolit.server.controllers

rolit.server.models

rolit.server.views

Daarnaast hebben we nog een package *‘rolit.sharedModels’* aangemaakt voor de gedeelde models tussen client en server. Ook is er nog een package *‘rolit.test’* te vinden, hierin zijn de gebruikte testklassen te vinden. De package *‘rolit.test’* bevat sub-packages waaruit duidelijk wordt of de desbetreffende testklassen voor de server, client of het model waren.

De klassen die te vinden zijn in de package en sub-packages *‘rolit.client’* worden onder het kopje [*client klassen*](#_Client_klassen) beschreven. Voor de package en sub-packages *‘rolit.server’* geldt dat de beschrijving te vinden is onder [*server klassen*](#_Server_klassen). [Model klassen](#_Model_klassen) bevat een beschrijving van de klassen die een spel Rolit modeleren, deze klassen zijn in de package *‘rolit.sharedModels’* te vinden. De gebruikte klassen voor het testen zijn te vinden onder [test klassen](#_Test_klassen).

## Client klassen

De klasse *‘Main’* te vinden in *‘rolit.client’*:

1. Deze klasse dient als doel om de Rolit client te starten. Als een gebruiker dus een spelletje wil spelen, zorgt deze klasse ervoor dat de ApplicationController gestart wordt.
2. Het starten van de client. Dit gebeurt door een nieuw object aan te maken van de ApplicationController. De ApplicationController neemt het vanaf daar over.
3. Deze klasse start een instantie van de ApplicationController, maar werkt hier verder niet mee samen.
4. –
5. –

In *‘rolit.client.controllers’* zijn onder andere de volgende twee klassen te vinden, *‘ApplicationController’* en *‘NetworkController’*. Deze twee klassen sturen vormen de spil van de Rolit applicatie aan de kant van de client. Zoals duidelijk wordt uit de naam van de klassen zijn het controller klassen. Deze sturen het model en de views aan op basis van informatie die ze over het netwerk van de server krijgen of via de views van de gebruiker.

De *‘ApplicationController’*:

1. Deze controller klasse zorgt ervoor dat op de events, gegenereerd door de views, actie wordt ondernomen en dat ze worden afgehandeld. Ook handelt de *‘ApplicationController’* de berichten die hij van de *‘NetworkController’* ontvangt af.   
   Als de gebruiker een zet doet of de *‘NetworkController’* een bericht ontvangt van de server dat er een verandering in het spel/op het bord heeft plaats gevonden, stuurt de *‘ApplicationController’* het model aan. Hierdoor komt het model van het Rolit spel aan de kant van de client overeen met dat van de server.
2. Het aansturen van de views in *‘rolit.client.views’* en de *‘NetworkController’*. Verder reageert de *‘ApplicationController’* op berichten en events van de *‘NetworkController’* respectievelijk de verschillende views.
3. De ‘*ApplicationController’* werkt nauw samen met de *‘NetworkController’*. De *‘ApplicationController’* is verder voor alle views de betreffende *‘ActionListener’* en stuurt de AI aan. Daarnaast checkt de *‘ApplicationController’* geldige zetten aan de hand van het model van Rolit (te vinden in *‘rolit.shardedModels’*).
4. –
5. –

De *‘NetworkController*’:

1. Deze klasse creëert de socket die met de server is verbonden en handelt alle netwerkverkeer met de server af.
2. De belangrijkste verantwoordelijkheden van de *‘NetworkController’* zijn het versturen en afhandelen van netwerkberichten van en naar de server. De *‘ApplicationController’* stuurt berichten aan de server door deze aan de *‘NetworkController’* te geven.  
   Als de server een command verstuurt aan de client, handelt de *‘NetworkController’* dit als eerste af. De *‘NetworkController’* zoekt uit welk command het is en roept daarna de desbetreffende methode in de *‘ApplicationController’* aan, zodat de *‘ApplicationController’* het verder kan afhandelen. Mogelijke parameters van het command worden door de *‘NetworkController’* behandelt (door ze bijvoorbeeld in een generieke list te zetten), zodat de *‘ApplicationController’* hier optimaal mee kan werken.
3. Vice versa werkt de *‘NetworkController’* nauw samen met de *‘ApplicationController’*.
4. –
5. –

Ook de *‘AIController’* en de *‘SmarAIController’* bevinden zich in *‘rolit.client.controllers’*:

1. Het doel van deze klassen is het aansturen van de AI als de gebruiker ervoor kiest om een de computer een potje te laten spelen. Ook kan de *‘SmartAIController’* de gebruiker een hint geven als hij zelf aan het spelen is.
2. De klassen moeten een potje Rolit kunnen spelen zonder een foute zet te doen, hiervoor baseert de AI zich op het model van het spel Rolit (te vinden in *‘rolit.shardedModels’*).
3. De klassen die het spel Rolit modeleren (te vinden in *‘rolit.shardedModels’*).
4. –
5. –

Zowel de *‘AIController’* als de *‘SmartAIController’* implementeren de interface *‘AIControllerInterface’* in *‘rolit.client.models’*:

1. Deze interface dwingt af dat er een methode is om de (beste) zet te berekenen.
2. Het afdwingen van een methode om een zet te berekenen.
3. –
4. –
5. –

Vervolgens hebben we de interface *‘LoggingInterface’* in *‘rolit.client.models’*:

1. Deze interface dwingt af dat er een manier is om te output aan de gebruiker te laten zien via een log.
2. Het afdwingen van een methode om te loggen.
3. –
4. –
5. –

Alle klassen uit de package *‘rolit.client.views’* die een bepaalde view modeleren zijn analoog aan elkaar opgebouwd, daarom worden ze in één keer behandeld. Er zijn drie views, namelijk een *‘ConnectView’*, *‘LobbyView’* en een *‘GameView’*. Deze klassen implementeren allemaal de interface *‘AlertableView’*. Deze interface garandeert, net zoals de *‘LogginInterface’*, dat er een manier is om een log bericht aan de gebruiker over te brengen.

1. De viewklassen zorgen voor de GUI waarmee de gebruiker kan verbinden respectievelijk een spelletje Rolit spelen of spelers uitdagen voor een potje en chatten in de lobby.
2. Het weergeven van de benodigde view en de input van de gebruiker doorgeven aan de ‘A*pplicationController’*.
3. De viewklassen krijgen van de *‘ApplicationController’* de benodigde informatie en koppelen vervolgens de input van de gebruiker terug.
4. –
5. –

## Server klassen

Van de server klassen spelen de klassen *‘Main’* in *‘rolit.server’*, *‘ApplicationController’* in *‘rolit.server.controllers’* en *‘LoggingInterface’* in *‘rolit.server.models’* een rol die zeer vergelijkbaar is met de gelijknamige klassen van de client. Daarom volgt hier geen beschrijving van deze klassen, maar kunt u de beschrijving van de client erbij pakken.

Een voetnoot wordt echter nog wel gemaakt bij de *‘ApplicationController’* van de server: Deze hoeft geen verschillende views aan te sturen zoals het geval is bij de client. De enige view waar de server over beschikt, is namelijk de zogenaamde *‘MainView’* te vinden in *‘rolit.server.views’*.

Verder zijn er nog enkele kleine verschillen met de *‘*ApplicationController’ van de client, de voornaamste is dat de *‘Game’* instanties aan de kant van de server worden beheert door de *‘NetworkController’*. Deze verschillen worden duidelijk bij het lezen van de toelichting van de *‘NetworkController’*.

Waar aan de kant van de client de belangrijkste klassen de *‘ApplicationController’* en de *‘NetworkController’* zijn, is er aan de kant van de server nog een belangrijke controller te vinden. Dit is namelijk de *‘ConnectionController’* ook te vinden in *‘rolit.server.controllers’*.

De *‘ConnectionController’*:

1. De *‘ConnectionController’* beheert de afzonderlijke sockets van elke gebruiker die met de server verbonden is. Een *‘ConnectionController’* correspondeert dus met één client.
2. Het beheren van de socket van één afzonderlijke gebruiker.
3. De *‘ConnectionController’* werkt nauw samen met de *‘NetworkController’*, deze werkt alle commands af die de *‘ConnectionController’* binnenkrijgt. Verder werkt de *‘ConnectionController’* samen met de *‘ApplicationController’*, de *‘ConnectionController’* gebruikt deze klasse om logentries te laten zien (in de view die door de *‘ApplicationController’* wordt beheerd).  
   Ook staat de *‘ConnectionController’* met de *‘NetworkController’* van de client in verbinding door middel van een socket.
4. –
5. –

De *‘NetworkController’*:

1. Deze klasse beheert alle *‘ConnectionController’* instanties van de server. De *‘NetworkController’* gaat dus over alle, met de server verbonden, gebruikers, waar de *‘ConnectionController’* de afzonderlijke verbinding beheert.
2. Commands verstuurt van en naar de server worden hier gelezen of doorgegeven aan de desbetreffende *‘ConnectionController’* in geval van verzending. Op de binnengekomen commands wordt actie ondernomen.
3. Ook aan de kant van de server is de samenwerking met de andere controller klassen, de *‘ApplicationController’* en de *‘ConnectionController’*, nauw. Zoals hierboven gezegd gaat communicatie met het model van Rolit (*‘rolit.sharedModels’*) bij de server via de *‘NetworkController’* in plaats van via de *‘ApplicationController’* zoals bij de client het geval is.
4. –
5. –

De ‘MainView’:

1. Deze klasse zorgt ervoor dat bij het starten van de server, door aanroepen van *‘Main’*, er een window zichtbaar wordt. Hierin is onder andere aan te geven op welke poort de server moet luisteren. Ook kan de gebruiker de server starten en als laatst voorziet de view ook nog in een loggedeelte. Dankzij deze log kan de server aan de gebruiker laten zien wat er gebeurt en waar hij mee bezig is.
2. Het weergeven van de benodigde view en de input van de gebruiker doorgeven aan de ‘A*pplicationController’*.
3. De viewklassen krijgen van de *‘ApplicationController’* de benodigde informatie en koppelen vervolgens de input van de gebruiker terug.
4. –
5. –

## Model klassen

Er zijn vier klassen die een spelletje Rolit modeleren. Deze klassen bevinden zich in de package *‘rolit.sharedModels’*. Ze modeleren een spelletje respectievelijk het bord, een plaats op het bord of een speler.

De klasse *‘Game’*:

1. Deze klasse beheert het geheel dat nodig is voor een spelletje Rolit.
2. De klasse *‘Game’* zorgt ervoor dat er speelbord wordt aan gemaakt en beheert de instanties van *‘Gamer’* die meedoen aan het spel. Ook houdt *‘Game’* de punten bij voor elke speler en checkt of een spelletje afgelopen is.
3. *‘Game’* werkt samen met *‘Board’* en *‘Gamer’*.
4. Deze klasse extends *‘Observable’* en wordt door de *‘ApplicatonController’* van de client of de server geobserveerd.
5. –

De klasse *‘Board*:

1. Deze klasse modelleert speelbord van Rolit.
2. De klasse *‘Board’* maakt een speelbord aan en beheert de instanties van *‘Slot’* die de verschillende vakjes op het bord modelleren. Verder checkt *‘Board*’ of een zet geldige is en plaats hem indien *‘Game*’ dit aangeeft.
3. *‘Board’* werkt samen met *‘Slot’* en *‘Game’*.
4. –
5. –

De klasse *‘Slot’*:

1. Deze klasse modelleert een enkel vakje op het speelbord.
2. Een instantie van *‘Slot’* moet zijn kleur bijhouden en zo nodig aanpassen.
3. *‘Slot’* heeft geen weet van de andere klasse en wordt alleen door *‘Board’* aangeroepen.
4. –
5. –

De klasse *‘Gamer’*:

1. Deze klasse modelleert een speler van het spel Rolit.
2. Houdt bij welke kleur deze speler is als hij deelneemt aan een spel. Wil de gebruiker een game joinen, dan houdt deze klasse bij hoeveel tegenstanders de gebruiker wenst. Verder houdt hij nog de naam van de speler bij.
3. Andere klassen houden instanties van *‘Gamer’* bij en geven aan wanneer er een verandering plaatsvindt.
4. –
5. –