Model-view-controller patroon
Welke klassen implementeren de view, welke de model en welke de controller.

Functional Requirements

In dit hoofdstuk volgend de functionele eisen en de plek waar een functionele eis is geïmplementeerd.

Aan de server kant:
Eis 1: Als de server start, moet er gevraagd worden naar een port nummer en na het invullen moet ernaar geluisterd worden: Package Server, klasse Server. Dit is geïmplementeerd door bij het uitvoeren eerst te vragen naar het poortnummer. Dan wordt gekeken of het geldig is, als het geldig is, runt de server, is dat niet zo, dan krijgt de gebruiker een bericht te zien.
Eis 2: Als een port nummer al gebruikt wordt, moet de klasse een duidelijk bericht geven en een nieuw poortnummer moet ingevuld kunnen worden. Package Server, klasse Server. Dit is inderdaad geïmplementeerd door de check te kijken of er al een ServerSocket is met hetzelfde poort nummer.
Eis 3: Een server moet meerdere instanties van de game kunnen runnen door verschillende clients. Package Server, klasse ClientHandler/Package Client, klasse ClientHandler. Beide zijn met threads geïmplementeerd en runnen tegelijk, zonder elkaar te kunnen overschrijven.
Eis 4: De server heeft een TUI dat ervoor zorgt dat alle communciatie verloopt via System.out. Dit gebeurt in QwirkleTUIView, waar geen enkele andere soort message komt dan System.out.
Eis 5: De server zou rekening moeten houden met het protocol gegeven in week 7 en moet kunnen communiceren met andere clients. Dit heeft al gewerkt in de wedstrijd met andere clients. Onze server deed het op dat gebied erg goed en gaf geen errors.

Aan de client kant:
Eis 1: De client moet een gebruiksvriendelijke TUI hebben, die opties biedt om een game te joinen. Dit staat in de klasse Client. Als de client fouten maakt in de verbinding, zal het direct een bericht terug geven wat hij of zij fout heeft gedaan. Echter, als een client foute input geeft bij het spelen van het spel, komt er geen reactie. Hij of zij kan wel weer opnieuw een beurt intypen.
Eis 2: De client moet menselijke en computerspelers ondersteunen.

Klasse omschrijvingen

In het volgende hoofdstuk staat klasse per klasse beschreven hoe de structuur er uit ziet, welke functionele eisen de klasse vervult, welke rol de klasse vervult in het model-view-controller-observer patroon en tenslotte wordt nagegaan welke klassen er gebruikt worden in de klasse die omschreven wordt met bijbehorende argumentatie. Alle methode

beschrijvingen staan in de javadoc. Mocht u geïnteresseerd zijn naar de methode omschrijvingen, dan verzoek ik u dus de javadoc te raadplegen.

| *Controller*

In de package controller staan alle klassen die het spel 'aanzetten'. Deze klassen krijgen de spelers al mee en starten het spel op. Daarna is de verantwoordelijkheid voor alle klassen die het spel daadwerkelijk bijhouden en de zetten verwerken. De package controller bevat een interface Controller en de klassen client en localcontroller.

Controller

Bij het maken van het spel wordt ook een controller meegegeven. Om dit toepasbaar te maken voor een LocalController en ServerGameThread, is de interface controller gemaakt.

Client

Client extends Thread en implements Command. Bij het runnen van de klasse wordt een client aangemaakt die verbinding maakt met een server. De client moet echter zelf het spel bijhouden, waardoor er gekozen is de klasse in controller te zetten. De client runt dus zelf ook een spel, maar dit is puur om alle zetten te verwerken en te laten zien hoe het bord eruit ziet. De verantwoordelijkheid voor het spel ligt bij de server.

LocalController

LocalController implements Command en zorgt ervoor dat er een lokaal spel aangemaakt wordt met de gegeven spelers in de main methode. Het echte spel wordt bijgehouden in game, LocalController start alles alleen op.

| *Exceptions*

De package exception bevat logischerwijs alle exceptions die methoden kunnen gooien tijdens het uitvoeren van de methode. Alle excepties zijn op een eenduidige manier geïmplementeerd, dus worden de klassen niet per klasse besproken.
Alle klassen extenden Exception. Bij het aanroepen van de exception, is het mogelijk een toepasselijke message te printen (wanneer dit in de code gevraagd wordt).

|| *Protocol*

De package protocol in package exceptions bevat allerlei exceptions die gegooid kunnen worden bij het uitvoeren van een client-server verbinding. Een deel van de error messages zijn ook al geïmplementeerd in het protocool. De klassen InvalidSocketInputException, ProtocolException en WrongServerCommandException gaan op de eenduidige manier bij

package exceptions beschreven. De andere twee klassen wijken iets af en worden apart besproken.

CommandException
CommandException handelt in het protocol gegeven errors af in de vorm van het kunnen opvragen van de error en het bijbehorende bericht. Het opvragen van de error wijkt hiermee iets af van de andere exceptions.

FirstPositionNotOriginException
Deze klasse extend ProtocolException, waarvan de klasse het bericht opvraagt en deze aanvult met een eigen bericht. Verder is de vorm exact hetzelfde als bij de package exceptions.

| *Model*
De volgende packages vallen allemaal onder het model package. In het model package staan klassen die zorgen voor het opslaan van waarden en de klassen veranderen de waarden wanneer dit “gevraagd” wordt. In de package model staan de packages components, game en players en de klassen block en board.

|| *Components*
In deze package zijn, normaal fysieke, materialen opgenomen om het spel te spelen. Het gaat hier om de stenen, de tas waarin de stenen zitten en het bord. In het vervolg van het verslag krijgen deze de engelse namen block, bag en board.

Block
Deze klasse maakt alle blokken aan die gebruikt worden voor het spel. Het geeft ieder blok een kleur en een type. Andere klassen kunnen de kleur, vorm, een uniek getal van een block, een lange string van een block en een korte string van een block opvragen. Bij de lange string is het blok helemaal uitgeschreven, de korte string bestaat uit twee karakters. Het bord en de speler gebruiken een block om het spel te kunnen spelen.

Board
Bij het aanmaken van de klasse board, maakt één coördinaat aan van het bord: 0,0. Dit is het begin van het bord. De klasse board gebruikt block, aangezien een block wordt meegegeven aan het bord. Verder checkt board of er een perfect vierkant ontstaan is, die niet mag voorkomen.
Klasse Board heeft ook een klasse Position. Board gebruikt position om ieder punt in een board een coördinaat mee te geven, een position. Position wordt dus gebruikt door klassen bij een verandering op het board, of een navraag naar iets op het board.
Klasse Board heeft ook nog een klasse Row. Hierbij is het mogelijk om een Play, richting(Orientation), Position en Play.Entry mee te geven, om zoDaarbij kijkt row ook of de row valide is, gericht op de lengte van een rij, het type blokje (die niet hetzelfde mag zijn) en de kleur óf vorm van de blokjes in de rij, die hetzelfde moeten zijn.

||| *bag*

Bag bestaat uit 3 klassen: De interface bag, en de klassen Realbag en Virtualbag die bag implementeren. In de package bag staat alles over de bag en worden alle waarden van de bag opgeslagen die de game nodig heeft.

Bag

Interface bag levert een aantal methoden die Realbag en Virtualbag allebei nodig hebben. Doordat het type bag is, kan een methode die het type bag heeft, zowel voor de Realbag als Virtualbag functioneren. Mede hierdoor zijn de klassen die bag gebruiken geschikt voor zowel client als server, aangezien beide klassen van het type Bag zijn. Bag maakt gebruik van de klasse block, aangezien de bag gevuld is met een lijst van blocks.

Realbag

Realbag is voor de server kant van het spel of voor het maken van een lokaal spel en implementeert bag. De realbag maakt een ´tas´ aan met drie stenen van alle mogelijke stenen in willekeurige volgorde. De bag is de klasse waaruit spelers stenen halen als ze willen ruilen of als een zet zetten op het bord en de stenen aanvullen.

Virtualbag

De Virtualbag is voor de client kant en implementeert bag. De virtualbag houdt zich bezig met het bijhouden van het spel en houdt bij hoeveel stenen er in de bag van de server zullen zitten.

||| *Move*

Package move heeft de drie klassen die de zetten regelen bij het spelen van het spel, namelijk de abstracte klasse Move en de klassen Play en Trade, die Move implementeren waar nodig. In de package Move zorgen de klassen ervoor dat een zet valide is, en mocht de zet valide zijn, dan zal de klasse ervoor zorgen dat player, board en bag ingelicht worden en deze klassen actie ondernemen.

Move

Abstracte klasse move ´staat boven´ de klassen trade en play. Er is gekozen voor een abstracte klasse, doordat er zo al een aantal standaard methoden, zoals getName, geïmplementeerd konden worden. Daarnaast zijn play en trade beiden types van zetten. Als zich een situatie voordoet waarbij een move gevraagd wordt, en het maakt niet uit of het een zet op het bord is of een ruilzet, dan kan een methode dat met een type Move vragen als return type. Move gebruikt de klassen game en player, maar verandert er nog niet echt iets mee voor andere klassen.

Play
Play is de klasse die een zet op het bord kan valideren en, mocht de zet valide zijn, dan kan de klasse de zet ook uit laten voeren. Play extends move en gebruikt board, block, position, row, game en player. De klasse kan veranderingen aanbrengen in board en block, kan de game verder laten lopen en de bag verkleinen, maar alleen als de zet valide is. Daarbij checkt de klasse twee spelregels: de eerste zet moet de groost mogelijke zet zijn en de speler mag alleen een rij stenen horizontaal óf verticaal neerleggen. Verder berekent de klasse ook de score die de speler behaalt bij zijn zet.

Trade
Trade is de klasse die een ruilzet kan valideren en, mocht de ruilzet valide zijn, dan kan de klasse de zet ook uit laten voeren. Trade extend move en gebruikt de klassen block, game en player. Via game kan de klasse veranderingen uitvoeren op bag. De klasse checkt validiteit op één spelregel: de ruilzet mag niet als eerste de zet zijn.

|| *Game*
Package game bestaat uit 5 klassen: abstracte klassen game en hostgame en klassen LocalGame, ServerGame en ClientGame. Abstracte klasse Game staat boven alle 5 klassen. ClientGame en HostGame extenden Game en LocalGame en ServerGame extenden HostGame. Zo zijn alle game types dus van het type Game, waardoor klassen als player met een type Game alle andere klassen ook kan ontvangen als type in bijvoorbeeld een parameter.

Game
Game bevat alle methoden die sowieso nodig zijn voor het spelen van een spel, of het nou als client, server of local game is. Game maakt bijvoorbeeld een board aan en heeft allerlei instantie variabelen om het spel bij te kunnen houden. Game gebruikt de klassen board, bag, player en controller. Game is dus de basis voor elk type spel, het de instantie variabelen maakt, methoden bevat die elk type game nodig heeft en vooral getters bevat.

ClientGame
ClientGame extends Game en houdt de staat van het spel bij. Het heeft dan ook zelf de components (een Virtualbag, block en board), zodat andere klassen ClientGame kunnen gebruiken. Zelf verandert ClientGame niets, en implementeert dus ook geen spelregels. Klassen die ClientGame gebruikt, zijn Controller, Virtualbag, Play, Player, Localplayer en Command.

HostGame
Abstracte klasse Hostgame extends Game en breidt de basis van Game uit om een eigen game te kunnen runnen, als server of lokaal. Bijvoorbeeld een methode als getStartingPlayer, die bepaalt welke speler als eerst mag beginnen, heeft ClientGame niet nodig. Er is gekozen om HostGame een abstracte klasse te maken, zodat overeenkomende methoden in LocalGame en ServerGame niet dubbel geïmplementeerd hoeven te worden. HostGame implementeert één

spelregel: De eerste speler moet de speler zijn die de meeste stenen op tafel kwijt kan. HostGame gebruikt Bag, Realbag en Controller en alle klassen die Game gebruikt.

LocalGame
Klasse Localgame extend Hostgame en zorgt er mede voor dat er lokaal een spel gespeeld kan worden. De klasse is vooral gericht op het doen en bijhouden van zetten. Localgame vraagt allerlei commando's op van verschillende klassen om tekst te genereren of bijvoorbeeld het board bij te werken. Daardoor gebruikt de klasse ook erg veel andere klassen, zoals Controller, Move, Player, LocalPlayer en alle klassen die HostGame en Game gebruiken. De klasse implementeert geen spelregels.

ServerGame
Klasse Servergame extends Hostgame en zorgt er mede voor dat de server kan runnen. ServerGame lijkt heel erg op LocalGame in de zin van functie, alleen vraagt het niet met methodes als determineMove naar zetten van spelers, maar communiceert het meer met tekst en commando's met de clients. De klasse gebruikt Controller, Block, Move, Play, Trade, Player, SocketPlayer en allerlei Server Commands.

|| *Players*
In de package players staan alle klasses die te maken hebben met de spelerskant van het spel. Daarbij gaat het bijvoorbeeld om het doen van zetten als speler van een lokaal spel, als client en als computerspeler op een lokaal en client spel. De package players bestaat uit twee packages, distant en local, de abstracte klasse Player.

Player
Abstracte klasse Player maakt een player aan en implementeert de basis methoden. Player is een abstracte klasse van zowel computer als humanplayer. Player is verantwoordelijk voor de score en de hand met blokken van de speler. Verder heeft Player methoden waarbij de maximale zet uit de hand kan worden opgevraagd. Daarmee implementeert Player deels de spelregel waarbij geldt dat de eerste speler de speler moet zijn met de grootst mogelijke zet. De klasse Player gebruikt block, position, move, play en game.

||| *Local*
Package local heeft alle klassen die verantwoordelijk zijn voor lokale players, zowel human als computer. Onder lokale players verstaan we spelers die niet via een client verbinding maken. De computerplayer uit deze package kan dus wél opgeroepen worden bij een client-server verbinding, alleen als de server verbinding aan de kant is van de code.

LocalPlayer
LocalPlayer extend Player en is de klasse waarvan HumanPlayer en ComputerPlayer erven. LocalPlayer heeft niet meer functies dan player en is vooral gemaakt om het type LocalPlayer te krijgen. Hierdoor kan specifieker een speler gevraagd worden en kan geen verwarring ontstaan door het (niet bedoelde) gebruik van bijvoorbeeld een SocketPlayer.

Computer
In de package computer staan alle klassen die te maken hebben met de computerPlayer, inclusief strategieën. Deze strategieën staan bij elkaar in de package Strategy. Daarnaast heeft de package computer de klasse ComputerPlayer.

ComputerPlayer
ComputerPlayer extends LocalPlayer en maakt een computer speler aan met de meegegeven strategie. Die strategie wordt uitgevoerd in de determineMove van ComputerPlayer. Verder lijkt ComputerPlayer heel erg op LocalPlayer.

Strategy
In de package strategy staan de drie klassen die te maken hebben met de strategie van de computerplayer. De klassen zijn interface strategy en de twee klassen stupidstrategy en shorttermstrategy.

Strategy
Interface Strategy is de klasse waarin de basis wordt gelegd voor de andere strategieën. Strategy bevat twee al geïmplementeerde static methoden, die bruikbaar zijn voor zowel de smart als de stupid strategy.

StupidStrategy
StupidStrategy implements Strategy die verantwoordelijk is voor een minder slimme strategy. De strategie bekijkt alle valide zetten op het bord, is er een zet valide, dan doet de ComputerPlayer deze zet.

ShortTermStrategy
ShortTermStrategy implements Strategy en is verantwoordelijk voor een slimme strategie. De strategie bekijkt alle valide zetten op het bord en kijkt welke zet de meeste punten op zal leveren. Deze zet geeft de strategie terug.

Human
En klasse HumanPlayer
Human heeft één klasse, namelijk de klasse HumanPlayer. HumanPlayer is de klasse die verantwoordelijk is voor het aanmaken en gebruik van en voor een menselijke speler. HumanPlayer neemt alles over van LocalPlayer en vult alleen de determineMove aan. HumanPlayer implementeert door de determineMove geen nieuwe spelregel. De klasse gebruikt verder precies dezelfde klassen als LocalPlayer en Player.

Distant
Distant heeft twee klassen om over een verbinding als client met een andere server te kunnen spelen. De klassen zijn serverplayer en socketplayer. Daarbij heeft de klasse mede de verantwoordelijkheid om de verbinding met de server goed te laten verlopen.

ServerPlayer
ServerPlayer extends player en houdt enkel en alleen de staat van de player bij. ServerPlayer is dan ook precies hetzelfde als player. Toch is een nieuwe klasse gemaakt, om het duidelijk te maken wat voor de server is.

SocketPlayer
SocketPlayer extends player en regelt de verbinding aan de player kant met de server. Er zijn nieuwe methoden gemaakt en methoden aangepast om de communicatie tussen client en server goed te laten verlopen. Verder is de klasse hetzelfde als player.

| *Network*
Package network is verantwoordelijk voor alle commands die nodig zijn aan de server en client kant. Daarnaast is een lezer en schrijver van de commands geïmplementeerd in deze package. De package Network bevat de packages Commands en IO en de klasse IProtocol. Het protocol wordt niet besproken, aangezien de makers van het digitale spel Qwirkle deze niet geïmplementeerd hebben.

|| *Commands*
De package commands bevat alle commands, aan client en server kant. De klassen opzet is zo gemaakt, dat via een simpele instanceof aanroep duidelijk wordt of de string een commando is. Commands bevat twee packages, client en server, en twee klassen, Command en GameCommand.

GameCommand
Interface gamecommand heeft methoden die nodig zijn om commando's te verwerken met mogelijk een directe invloed op een klasse van het type game.

Command
Abstracte klasse command identificeert de inkomende string en kijkt of het een commando is. Als dat zo is, dan wordt deze commando direct doorgevoerd naar de bijbehorende klasse die het commando afhandelt.

||| *Client*
In package client staan alle commands die de client kan krijgen. Er staat ook weer een klasse die de commando's kan lezen en doorvoeren naar de juiste klasse van het commando. Mocht de client extra toepassingen bevatten, zoals de chat, dan wordt dat in het package features afgehandeld.

ClientIdentifyCommand
Deze klasse extend command en is verantwoordelijk voor het identificeren van de client, vooral gericht op de features. De normale commands zijn namelijk al geïmplementeerd in command.

ClientMovePutCommand
Deze klasse extend command en implementeert gamecommand. ClientMovePutCommand herkent de vorm van een zet op het bord en vertaalt het zo, dat het leesbaar is voor de methoden die de zet valideren en uitvoeren.

ClientMoveTradeCommand
Deze klasse extend command en implementeert gamecommand. De klasse herkent de vorm van een trade zet en vertaalt het zo, dat het leesbaar is voor de methoden die de trade valideren en uitvoeren.

ClientQueueCommand
Deze klasse extend command en geeft een vorm terug aan andere klassen van queue, waaraan andere klassen herkennen hoe groot de queue is waarin de client zich wil bevinden.

ClientQuitCommand
Deze klasse extend Command en zorgt ervoor dat bij het aanroepen van de klasse, de klasse een leesbare quit string terug kan geven.

||| *Features*
Features heeft een aantal klassen die een string kunnen teruggeven van de feature die zij ondersteunen. Aangezien al deze klassen hetzelfde type implementatie hebben, staat hier een stukje overkoepelend over alle klassen. Alle klassen geven het bijbehorende protocol terug.

|| *Server*
In de package server staan alle klassen die te maken hebben met commando's die bij de server binnen komen en afgehandeld moeten worden. De klassen hebben weer dezelfde soort rol, maar passen dit allen toe op andere variabelen, waardoor een andere implementatie nodig is. De rol van alle klassen is het vertalen van een string in een bruikbare variabele voor klassen als game of move en het mogelijk maken om een string terug te geven van de command.

ServerCommand
Abstracte klasse servercommand extend command en voegt één methode toe die voor afhandeling van alle commando's aan de server kant bruikbaar is.

|| *IO*
Package IO bevat de klassen commandreader en commandwriter, die een lijn tekst, geschreven door een client of server, kan omgezet worden naar een commando, zodat van daar uit weer iets kan gebeuren in bijvoorbeeld game.

| *Server*
Server bevat alle klassen waar de communicatie die binnenkomt gelezen kan worden, al dan niet met hulp van andere klassen, en zo nodig doorgestuurd wordt naar de juiste command.

Ook regelen deze klasse de échte verbinding tussen de clients en server. Server bevat de volgende vier klassen:

ClientHandler
ClientHandler extends Thread en leest de regels die binnenkomen van de client. Deze stuurt de clienthandler door als het commands zijn naar de command klassen. Ook kan de clientHandler de verbinding met de client verbreken.

GameCreator
GameCreator extends Thread en kan een game opzetten als er genoeg mensen in de queue zitten. De klasse kan mensen aan de door de speler aangegeven queue toevoegen en een spel maken als de queue vol zit. Mocht een speler de verbinding verbreken, dan verdwijnt hij ook uit de queue, dat verdwijnen uit de queue regelt deze klasse ook.

Server
De klasse server zet de server op met een poortnummer als input. De server maakt bij het verbinding maken van clients een nieuwe clientHandler aan. Ook kan de Server onder andere de verbinding met een client verbreken, zo nodig.

ServerThread
ServerGameThread extends Thread en implements controller en maakt een spel aan voor de server. Dit is een thread, zodat meerdere games tegelijkertijd kunnen draaien. Doordat het de controller implementeert, kan het commando's van de controller uitvoeren en dus een game starten en laten draaien.

| *View*
In de package view staan alle klassen die verantwoordelijk zijn van het sturen van berichten naar de client of speler. Bij het opvragen van methoden van de klassen, geeft het een string terug met de staat van het opgevraagde item, bijvoorbeeld van het bord. De package heeft een interface en een klasse.

QwirkleView
De interface QwirqleView implement observer en maakt methoden aan om op te kunnen vragen wat de staat is van items. Dit is gedaan voor een eventuele GUI, waar de makers niet aan toe zijn gekomen vanwege tijdnood.

QwirkleTUIView
De klasse extend QwirqleView en implementeert alle methoden. De rol is om de staat van alle items te weergeven in een string en deze te printen. De strings voor de communicatie van de server/localgame naar de speler, worden dus in deze klasse gemaakt. Alle prints over het spel staan overzichtelijk bij elkaar en kan, zo nodig, makkelijk gevonden en aangepast worden.