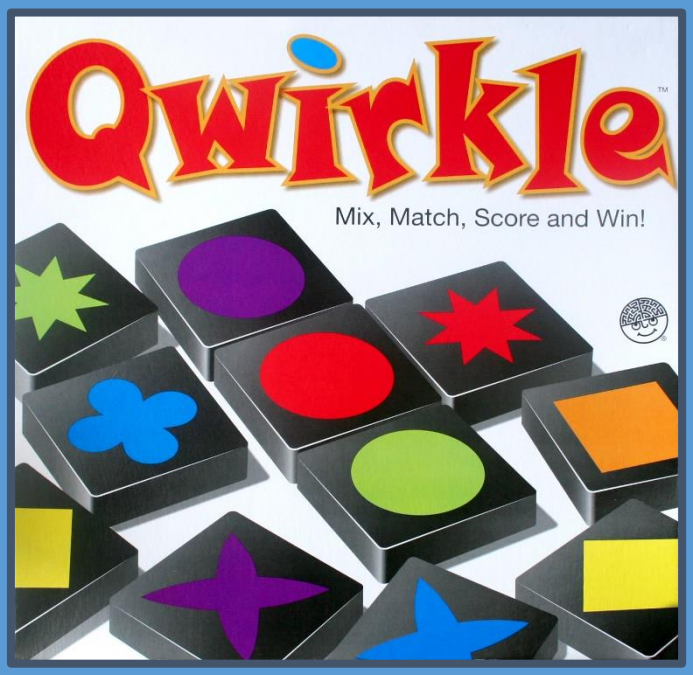
QWIRKLE



Wouter Bolhuis en Cas Putman

MODULE 2, BIT Universiteit Twente

Inhoud

Discussie wat betreft Design:

Observer

Wegens gebrek aan tijd, wat resulteerde uit het verkeerd inschatten van de grootte van het project, hebben we helaas het Observer pattern niet kunnen implementeren.

Model-View-Controller

In de implementatie van Qwirkle hebben we ons best gedaan om het Model-View-Controller pattern op een eenvoudige en natuurlijke manier te implementeren. Het MVC is op deze manier tijdens het programmeren geïmplementeerd en geëvolueerd tot wat het nu is. Als voorbeeld hebben we aan het begin als uitgangspunt genomen dat de Client en Server classes zouden functioneren als Controllers. Uiteindelijk is dit iets anders gelopen; ze dienen ook deels als Views vanwege de vele in- en outputs die zij afhandelen en weergeven aan de gebruiker.

Aan het begin van het project zijn we van start gegaan met het onderverdelen van de verschillende klassen onder Model, View en Controller. Op deze manier zagen we al snel dat de basisklassen van het spel, namelijk Board, Server en Player, Models zouden worden vanwege de visuele verwerking en het manipulerende karakter wat deze klassen bezitten. De Client zal vanzelfsprekend een View en Controller i een worden, vanwege het feit dat deze klasse het bord presenteert aan de gebruiker en aan het hoofd staat van de interactie met de Models.

Game zou je kunnen zien als een controller aangezien het dient als de bindende factor tussen de overige extensie klassen die de applicatie vormen zoals die nu is. Game vertrouwt op het functioneren van klassen als Bag, Tile, Board en Player om te functioneren. Game roept ook de meeste externe functies aan van alle klassen bij elkaar.

Wat betreft de andere klassen: deze zijn over het algemeen ontworpen om te dienen als Controllers. Uiteraard zijn in sommige klassen wat kenmerken van Models en Views terug te vinden. Echter wilden we in onze uiteindelijke implementatie uitgaan van de ideeën die wij als eerste bedacht hadden. Dit is ons aardig gelukt, en het MVC-pattern is naar ons idee en ontwerp tot stand gekomen.

Class Diagram

Het Class Diagram is ietwat groot uitgevallen, dit is bewust gedaan zodat alles overzichtelijk blijft en de lijnen niet over het hele scherm lopen. Linksbovenin bevindt zich de Core package, welke alle basisonderdelen van het spel bevat. De verschillende klassen binnen de Core package benaderen over het algemeen weinig klassen van buiten het eigen package. Als enige uitzondering hier op is Client, welke 4 associaties heeft met de Player klasse.

Rechtsbovenin bevindt zich de Strategy package, welke een enkele associatie heeft met ComputerPlayer. Dit spreekt voor zich: enkel de ComputerPlayer maakt gebruik van strategieën.

Wanneer we kijken naar de package van Player zien we dat deze veel associaties heeft met klassen uit andere packages. Dit is logisch; Player wordt in bijna elke andere klassen aangeroepen en is een cruciaal onderdeel van het spel. Zo bevat Server maximaal 4 Players, en bevat Client ook 4 Player objecten om een spel te kunnen starten. Een veel gebruikte klasse dus, dit verklaart ook de Emma coverage resultaten die later aan bod zullen komen.

Verder bevinden zich in het midden nog de Server en Protocol klassen. Deze hebben beide relatief weinig associaties met andere klassen, daar deze implementatie nog niet voltooid is. Zoals duidelijk te zien is, is ClientHandler de schakel tussen Client en de Server, en maken beide gebruik van het de Protocol-interface.

Functionele eisen aan de applicatie

Hieronder volgt een korte opsomming van de verschillende functionele eisen, en in welke klassen deze geimplementeerd zijn. Erna volgt een korte toelichting:

Server:

1. Server starten aan de hand van een poort.
2. Server returned een IOException wanneer de poort reeds in gebruik is.
3. Server ondersteunt meerdere speler verbindingen, spelen is geen mogelijkheid.
4. Server vangt berichten af, en indien nodig geeft hij deze weer met System.err of System.out. Afhankelijk van de urgentie van het bericht.
5. Server implementeert het protocol wanneer mogelijk, spelen is geen mogelijkheid.

Client:

6. Client heeft de beschikking tot een TUI die ingebouwt is binnen in de Client klasse. De TUI stelt verschillende vragen en laat de gebruiker deze beantwoorden doormiddel van een inputstream.
7. Client vraagt, wanneer de applicatie wordt gestart, of de gebruiker offline wilt spelen. Indien dit het geval is kan de gebruiker spelen tegen maximaal 3 artificiële spelers.
8. Client heeft de beschikking over 2 verschillende moeilijkheidsgradaties artificiële spelers. Deze Client vraagt de gebruiker bij het starten om een keuze.
9. Client heeft geen beschikking over hint functionaliteit, dit is niet geïmplementeerd.
10. Client heeft de beschikking over een herstart optie, dit is echter nooit getest.
11. Client wordt geïnformeerd wanneer de server de verbinding verbreekt en kan kiezen om naar het menu te gaan of om af te sluiten. Online speelfunctionaliteit is niet mogelijk.
12. Client zou moeten kunnen communiceren met andere servers, het protocol is geïmplementeerd. Dit is echter niet getest.

Verder ondersteunen de volgende klassen de implementatie van bovenstaande eisen:

1. Server
2. Server
3. Server, Client
4. Server, Client, ClientHandler
5. Server, Client, ClientHandler + manier van implementatie in elke klasse.
6. Server, Client, ClientHandler, Board, Game, Bag, Tile

7. Client, Player, ComputerPlayer, HumanPlayer
8. Client, Player, ComputerPlayer
9. N.v.t.
10. Server, Client, ClientHandler
11. Server, Client, ClientHandler
12. Server, Client, ClientHandler + manier van implementatie in elke klasse.

Data Opslag en Communicatie Protocollen

Alle data in dit project wordt opgeslagen in de fields van de klassen Game, Client, Bag en Player klassen. Dit leek ons het meeste logische om te doen; ons protocol maakt namelijk geen gebruik van opslag aan de Serverzijde. Borden werden lokaal bijgehouden en kunnen op geen enkele manier opgevraagd worden vanuit de Client, dit geldt ook voor de score van de spelers. Dit lijkt niet logisch, echter is dit besloten samen met de gehele projectgroep. Als resultaat hiervan zouden bord en score eenvoudig te manipuleren zijn, en is het zeer goed mogelijk dat er synchronisatiefouten optreden als gevolg van een verschillende implementatie.

Om het probleem van hoofdletters en kleine letters te ontwijken, hebben wij gekozen om over het algemeen deze te negeren bij het gebruik van commando's. Dit maakt het voor de speler makkelijker om het spel te spelen; er hoeft immers niet op correct hoofdletter gebruik gelet te worden. Commando's zijn dus hoofdletter onafhankelijk.

Toevalligerwijs zijn wij eveneens de groep die het protocol bij houdt en veranderd wanneer hier om gevraagd werd. Dit betekent echter niet dat wij compleet achter ons protocol staan, integendeel, wij denken dat er enkele enorme ontwerpfouten in het protocol zitten. Hiermee doelen wij op de problemen die eerder genoemd zijn. Helaas is hier niks aan te veranderen en hebben wij gebruik moeten maken van het protocol zoals deze in Week 7 is opgesteld. Spelen met andere groepen was anders onmogelijk geweest. (Nu helaas ook, dit ligt echter aan onze gebrekkige implementatie.)

Tijdens het verloop van het programmeerproject hebben wij meerdere vragen gehad omtrent het protocol, en hebben een en ander moeten wijzigen. Dit wijten wij aan de lage opkomst tijdens de protocolsessie.

Het protocol bestand kan online gevonden worden, en is tevens bijgesloten in het ZIP bestand.

JML en JavaDoc

Met behulp van Metrics hebben wij berekend wat, aan de hand van de WMC, de meest complexe klassen van ons project zijn. Hier is uitgekomen dat het gaat om de volgende 3 klassen:

- Client
- Game
- Board

Bij deze klassen zijn bij de bijbehorende methoden bijpassende JML specificaties en JavaDoc geschreven.

Discussie per klasse

Package Qwirkle.Core

Bag

Bag is zoals het woord al zegt, de zak waarin alle tegels zich bevinden. Deze tegels worden aan het begin van het spel uitgedeeld, doordat deze opgevraagd worden door Game. Bag zorgt ervoor dat het spel eerlijk verloopt, en deelt eveneens tegels uit wanneer men deze wilt ruilen. Indien er een tegel uit de zak gehaald wordt, wordt het aantal tegels in de zak verminderd. Hierdoor is het onmogelijk om meer tegels in het spel te hebben dan volgens de regels toegestaan is, en verloopt het spel als het even kan zo eerlijk mogelijk.

Bag is afhankelijk van de klasse Tile voor zijn werking.

Tile

Tile is verzorgd in zijn geheel implementatie van de verschillende tegeltjes die het spel Qwirkle rijk is. Tile is een klasse die bestaat uit 2 Enum's, namelijk die van kleur en vorm. Tile definieert deze twee verschillende kleuren en stelt ze samen zodat ze 1 Tile vormen. Indien er om gevraagd wordt kan er eventueel een textuele representatie van een tegel opgeleverd worden.

Tile is niet afhankelijk van andere klassen voor zijn werking.

Board–

Board is verantwoordelijk voor het bijhouden en representeren van het bord in een manier dat deze voor de gebruiker te begrijpen is. Aangezien het bord dynamisch is, dient het bord ten allen tijden de zetten in de gaten te houden en te verwerken op het scherm van de client. Bord is een zeer belangrijke klasse in de applicatie, en is dan ook afhankelijk van relatief veel andere klassen.

Board is afhankelijk van de klassen: Game, Tile en Bag.

Game

Game is verantwoordelijk voor het aanmaken van een spelinstantie waarin alles samen komt. Als fields zijn dan ook Bag, Player en Rules gedefinieerd. Game controleert of een zet geldig is, wie er aan de beurt is, zorgt ervoor dat er een bord is waar op gespeeld kan worden en een zak waar tegels uit te halen zijn, deelt tegels uit aan spelers, berekent de score en geeft zetten door aan het bord vanuit

de Client of de AI. Wanneer een spel gestart wordt zal als een van de eerste stappen de constructor van Game aangeroepen worden, waarna het spel spoedig aan zal vangen.

Game is afhankelijk van de klassen: Board, Player, Rules, Tile en Bag.

Rules

Rules definieert de verschillende spelregels waar Qwirkle aan onderhevig is. Rules controleert of een zet toegestaan is; dit betekent of er de juiste tegel naast, onder of boven is geplaatst en of er geen zet wordt gedaan op een plek waar al een tegel ligt. Bovendien controleert Rules aan de hand van de score of er een winnaar is, en wie deze winnaar dan wel niet mag zijn.

Rules is niet afhankelijk van andere klassen voor zijn werking.

Client

Client is het commandocentrum van het spel. Client functioneert tevens als TUI voor het spel, en handelt daarmee de input van de gebruiker af. Wanneer een speler een spel wilt spelen roept hij Client aan en beantwoord hij enkele vragen die de TUI aan hem stelt. Wanneer deze vragen zijn beantwoord zal de Client een instantie van Game opvragen die het spel verder afhandelt. De TUI zal verder het bord weergeven, dat hij doorgestuurd krijgt vanuit de spel- en het bord instantie. Client is indirect afhankelijk van elke klasse die het spel bezit.

Client is direct afhankelijk van de klassen: Game, Player en Board.

Package Qwirkle.Player

Player

Player is de interface van de klassen ComputerPlayer en HumanPlayer. Player definieert de standaard methoden en fields die ComputerPlayer en HumanPlayer erven. Een Player instantie is een speler van het spel. Een Player heeft een naam, en verder niks.

Player is niet afhankelijk van andere klassen.

ComputerPlayer

ComputerPlayer definieert een artificiële speler. Deze speler krijgt een strategie mee van, SmartStrategy danwel StupidStrategy, die de speler zetten laat doen. Deze zetten worden doorgegeven aan de methode determineMove.

ComputerPlayer is afhankelijk van de klassen Player, Strategy, StupidStrategy en SmartStrategy.

HumanPlayer

HumanPlayer definieert een menselijke speler. Deze speler kan zijn zetten doen door middel van determineMove, dat aangeroepen wordt door de klasse Game.

HumanPlayer is afhankelijk van de klasse Player.

Package Qwirkle.Strategy

Strategy

Strategy is de interface van de klassen SmartStrategy en StupidStrategy. Strategy definieert de standaard methoden en fields die SmartStrategy en StupidStrategy erven. Een Strategy is een strategie die zetten bepaald die een ComputerPlayer kan gebruiken.

Strategy is niet afhankelijk van andere klassen.

SmartStrategy

SmartStrategy is een slimme strategie die een zet op een slimme beredeneerde positie kan doen. Dit doet hij met behulp van de eerder besproken methode “determineMove”.

SmartStrategy is afhankelijk van de klasse Strategy.

StupidStrategy

StupidStrategy is een simpele strategie die een zet op een willekeurige plek zet zolang de zet toegestaan is. Dit doet hij met behulp van de eerder besproken methode “determineMove”.

StupidStrategy is afhankelijk van de klasse Strategy.

Package Qwirkle.Server

Server

Server zorgt er voor dat het mogelijk is om een spel te spelen over een internetverbinding. Server opent een poort waarmee de Client verbinding kan maken, waardoor er desgewenst een spel gestart kan worden. Indien de Client het verzoek stuurt een spel te starten, zal de Server een nieuwe Game instantie aanmaken met maximaal 4 spelers die zich op dat moment in de lobby verkeren. Het spel zal verder verlopen op dezelfde manier als offline, daar het protocol ook offline geïmplementeerd is. De commando’s blijven dus hetzelfde.

Server is afhankelijk van de klassen ClientHandler, Player, Game en Protocol.

ClientHandler

Handelt, vanzelfsprekend, al het verkeer tussen de Client en de Server af. De Server gebruikt de ClientHandler als referentie naar een Player, verstuurt berichten op deze manier en vice versa. Communicatie tussen de Client en de Server, laat staan het spelen van een spel, zou onmogelijk zijn zonder deze klasse.

ClientHandler is afhankelijk van de klassen Server en Client.

Notitie:

Client en Server functioneren niet naar behoren. Voor het implementeren van een online spelopzet was simpelweg geen tijd. In plaats van dat het spel opgestart wordt aan beide Client zijden, wordt er een spel gestart in de console van de Server.

Er zijn geen speciale gevallen in de contracten van de verschillende klassen, eveneens als enige voorzorgsmaatregelen wat betreft precondities in de contracten van de Server-klassen.

Test Report

Gebruikmakend van de WMC met behulp van de Metrics plugin, hadden wij de volgende 3 klassen moeten testen:

Client, Game en Board

Vanwege een ernstig gebrek aan tijd, zijn we er tot dusver nog niet aan toe gekomen om Junit test classes te schrijven. Er is simpelweg te veel tijd gaan zitten in het proberen werkend te krijgen van een online spel dat het schrijven van testklassen helaas niet gelukt is. Bovendien was dit in combinatie met de opgeleverde code nog zeer lastig geworden, daar er dan met AI tegen AI getest had moeten worden en dit een dusdanig resultaat oplevert dat wij er niet wijzer van zouden worden.

Uiteraard is er wel visueel getest, veelvuldig zelfs, om er voor te zorgen dat de applicatie functioneert zoals hij nu doet. Honderden keren is het spel opnieuw gestart en is er met behulp van verschillende debug mogelijkheid geprobeerd het spel verder te ontwikkelen. Op deze manier zijn er al veel bugs aan het licht gekomen, waardoor het spel op dit moment wanneer hij gespeeld wordt met AI tegen speler redelijk bugvrij is. De score wordt berekend, zetten worden gecontroleerd op geldigheid en tegels kunnen geruild worden tegen andere die nog aanwezig zijn in de zak. Het resultaat bij een eventueel einde is onbekend; de AI is te langzaam om een einde binnen een afzienbare tijd af te dwingen.

Indien wij meer tijd hadden gehad, zou het geen enkel probleem zijn geweest om testklassen op te leveren.

Wanneer wij het spel (onvoltooid) uitvoeren in de vorm AI tegen Speler, krijgen wij de volgende coverage statistieken van Emma:

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Qwirkle	48,7 %	2.663	2.802	5.465
src	48,7 %	2.663	2.802	5.465
qwirkle.server	0,0 %	0	1.131	1.131
qwirkle.core	67,4 %	2.001	969	2.970
qwirkle.strategy	17,6 %	125	585	710
qwirkle.player	90,1 %	537	59	596
qwirkle.test	0,0 %	0	58	58

De coverage van het totale project is nog geen 50%, dit is logisch aangezien Server op geen enkele manier gebruikt wordt maar wel zo'n 15% van de totale code bevat. Als we kijken naar de coverage van de individuele pakketten zien we een veel rooskleuriger beeld. Het pakket Core covert zo'n 67%, en dat terwijl het spel nog niet eens afgelopen was op het moment van meten. Wanneer we dit in acht nemen kunnen we dit een zeer prima score noemen. Wanneer we kijken naar de Player klasse zien we een coverage van wel 90%. Dit is logisch aangezien er gebruik werd gemaakt van een AI en een HumanPlayer, wat de totale inhoud van Player is.

Strategy toont een lage coverage, dit is wederom logisch daar een groot deel van de code deel uit maakt van de SmartStrategy. In deze test werd er gebruik gemaakt van een StupidStrategy.

Metrics

Tijdens de design practica zijn de volgende Metrics besproken, met daar achter onze eigen waardes.

- Lines of code 1812
- Cyclomatic Complexity 3,3
- Coupling between Classes (AF/EF) 3,6/1,9
- Lack of Cohesion in Methods 0,326

Lines of code is slechts interessant voor de CodeJunkies, qua complexiteit voegt het niks toe. Cyclomatic Complexity, anderzijds, dient volgens McCabe onder de 10 te blijven. Dit om het onderhoud, testen en bugfixing te vergemakkelijken. Dit is bij ons dik in orde: met een waarde van 3,3 zit de Cyclomatic Complexity ruim onder de aangeraden waarde van 10.

Wat betreft Coupling Between classes is er volgens McCabe geen richtlijn over wat aangeraden is. Echter indiceert een hogere waarde wederom een lastig te onderhouden en te testen systeem. Wij kunnen op dit moment niet zeggen of een gemiddelde van 3,6 respectievelijk 1,9 hoog is. Uitgaande van de limietwaarde zoals gegeven bij de Cyclomatic Complexity kunnen we er redelijkerwijs vanuit gaan dat het ook met deze Metric wel goed zit in onze code.

Bij een hoge waarde voor de LCOM heeft men vaak te maken met lastige, complexe klassen die fout gevoelig zijn. Met een waarde van 0,326 ligt onze waarde voor de LCOM vrij laag maar is verbetering mogelijk. Dit zouden we mogelijk kunnen oplossen door een aparte TUI te maken, en deze uit Client te verwijderen. Eventueel zouden de methoden die verschillende commando's in de Server afvangen in een aparte klasse kunnen onderbrengen.

Reflectie op planning

Hoe was je planning beïnvloed door je ervaringen met plannen en tijdmanagement tijdens het designproject?

Het designproject heeft onze planning op het programmeerproject totaal niet beïnvloed. Het designproject omvatte achteraf gezien veel minder werk dan het programmeerproject, en met het designproject zijn we dan ook totaal niet in de problemen gekomen. Het designproject hebben wij op tijd afgerond, in tegenstelling tot het programmeerproject wat nog steeds niet afgerond is.

Tot op welke hoogte kwam je planning overeen met het daadwerkelijke werk?

De planning kwam totaal niet overeen. Het programmeerproject bleek zo'n 2 tot 3 keer zoveel werk als ingeschat. De eerste week na de protocolsessie-week leek alles nog vrij volgens planning te gaan, echter in de weken die erop volgen waren we dagelijks vele uren bezig met het programmeren van het project. Vele kleine fouten waar wij uren bij stilgestaan hebben, en de in dit jaar vele malen ingewikkeldere regels die aan het spel gebonden waren dan vorig jaar maakte dat we overal veel meer tijd aan kwijt waren dan ingeschat. De resterende 3 weken bleken niet voldoende om dit werk in te halen, ondanks dat we elke dag bezig zijn geweest.

Welke maatregelen hebben jullie genomen om de verloren tijd in te halen?

We hebben geprobeerd er extra tijd in te steken, dit wilden we bewerkstelligen door bepaalde sociale activiteiten te laten schieten en er overdag meer werk in te steken. Dit resulteerde in het feit dat we de laatste 2 weken bijna non-stop van 11.00 uur tot 18.00 uur bezig zijn geweest met programmeren. Helaas bleek dit niet voldoende; we komen gewoon een week tekort. De kwaliteit had vele malen hoger kunnen zijn wanneer we meer tijd hadden gehad.

Wat heb je geleerd van de planning van het programmeerproject?

Schat nooit de grootte van een project in op basis van andermans advies of ervaring van jezelf uit het verleden. Calculeer altijd veel meer tijd in dan nodig, dan kan je niet voor verassingen komen te staan. Wanneer we hadden geweten dat het project van dit jaar zo'n 2 tot 3 keer zo complex was als die van vorig jaar, waren we al voor de protocolsessie begonnen.

Stel je bent studentassistent, wat zou jij de studenten bij willen brengen wat betreft planning?

1. Begin al voor de protocolsessie met het singleplayer gedeelte.
2. Vertrouw nooit op ervaringen van studenten van vorig jaar; het project kan enorm verschillen.
3. Blijf niet oneindig doorprutsen, laat onderdelen vallen wanneer nodig.
4. Begin al snel met het maken van tests, dan heb je iets te presenteren.