Software Systems Qwirkle project verslag.

Naam: Bas van den Borg.

Student nr: 1533266

**Introductie**.

Dit is het verslag over het Qwirkle programma van project groep P33. Het project begon in week 6 van Module 2 en eindigde op 27 januari 2016. Het was de bedoeling dat het project door 2 studenten gemaakt zou worden. Groep P33 bestond voor een tijd niet uit meer dan 1 persoon, maar tijdens week 7 is er toch nog een andere student aan toegevoegd. Deze student staat nu niet bij de namen van dit verslag, omdat hij tijdens week 9 groep P33 verlaten heeft met besproken redenen niet relevant voor dit verslag. Het project was een Qwirkle spel implementatie in java dat aan eisen vermeld in de module handleiding moet voldoen. Niet alle eisen zijn geimplementeerd vanwege het tijdgebrek dat ontstaan was na het verlaten van m’n project partner, ik zal op sommige problemen of gebreken nog commentaar geven.

**MVC patroon in het geschreven programma.**

Het programma is voor het grootste deel MVC, alleen ontbreek het Observer patroon. De packages zijn vernoemt naar de functies die ze hebben in het MVS patroon.

MODEL bestaat uit de klassen: Board, Coord, Move, Tile, TileBag, Game, RetardedStrategy, ComputerPlayer, HumanPlayer en Strategy. Deze klassen worden gebruikt om informatie op te slaan.

VIEW bestaat uit de enkele klasse Client. Deze klassen worden gebruikt om de informatie te weergeven aan de Qwirkle spelers.

CONTROLLER bestaat uit de klasse PlayerHandler en Server. Deze klassen worden gebruikt om de informatie aan te passen in de MODEL en de interactie van de Qwirkle spelers met het programma op te vangen.

Class Diagram

**Functionele eisen van de Server Applicatie en de implementatie daarvan.**

Eis: De Server moet tijdens het starten vragen om een portnummer waar de server vervolgens naar zou luisteren.

Implementatie: Hoewel de server niet naar naar directe user input vraagt om het te beluisteren poortnummer in te stellen, kan het poortnummer wel ingesteld worden in de server klasse. Er was wegens het gebrek aan tijd geen tijd gegeven aan het implementeren van een naar de System.in luisterende methode die het poort nummer zou inlezen.

Eis: Als het verkeerde poortnummer ingevoerd word dan moet de thrown exception opgegevangen worden, waarbij vervolgens een goede error message word weergeven. Nadat de error message weergeven is kan er een ander poort nummer ingevoerd worden.

Implementatie: Zoals in de vorige eis al gegeven is, is er geen directe interactie met de server. Deze eis had in dezelfde eis verwerkt kunnen worden door het toekennen van het poortnummer in een try catch block te zetten. Het catch blok zou dan de thrown exception vangen, een error message printen en dan opnieuw de methode aanroepen waarin het poortnummer word ingelezen vanuit de System.in.

Eis: De server moet meerdere instanties van het spel dat gespeelt word door verschillende clients moeten kunnen ondersteunen. Anders gezegd de server moet multithreaded zijn.

Implementatie: De geimplementeerde server is multithreaded. De Server maakt nieuwe sockets aan wanneer er nog niet evenveel spelers zijn als de GAMESIZE (de GAMESIZE waarde zou ook mooi meegegeven kunnen worden in de methode van de eerdere eisen). Wanneer er een Socket aangemaakt word deze gelijkgesteld aan serverSocket.accept() hierbij word een nieuwe thread gestard met een PlayerHandler erin, zodat de server meerdere players tegelijk kan hanteren met meerdere PlayerHandlers.

Eis: De server moet een TUI hebben waarin alle communiactie berichten naar System.out schrijft.

Implementatie: Voor het grootste deel word de communicatie naar System.out geschreven, de communicatie is alleen niet netjes naar System.out geschreven in een mooie bericht vorm.

Eis: De server moet de tijdens het project ontworpen protocol hanteren.

Implementatie: Dit word besproken in ‘*Met welke structuur word de informatie verstuurd tussen de spelers’*.

**De Klassen.**

PlayerHandler: De PlayerHandler moet de communicatie tussen de server en een speler regelen en behoort tot het CONTROLLER onderdeel van het MVC patroon. Elke PlayerHandler heeft een eigen Socket waarmee hij met de Client van een speler communiceert en een eigen Player waarin hij de informatie van de Player kan opslaan. Verder gebruikt de PlayerHandler hetzlefde Board object als de server zodat er niet met verschillende Boards gespeelt word in hetzelfde spel.

PlayerHandler leest met een BufferedReader uit zijn socket de berichten van de verbonden Client, deze berichten moeten aangekondigd worden met in de Protocol gespecificeerde commando’s. Zodra een commando herkent word moet de structuur van het bericht overeenkomen met de input van het command en de argumenten van het commando geldig zijn. Mocht het commando herkent zijn maar de rest van het bericht is niet volgens de structuur of bevat ongeldige argumenten, dan (dit was wegens tijd gebrek nog niet helemaal geimplementeerd) zou de speler uit het spel gezet moeten worden met de kickplayerFromLobby methode van de Server en een exceptie gegooid moeten worden die beschrijft wat de reden voor de kick is.

De methodes readSwap en readMove in PlayerHandler bevatten te veel nesting, dit maakt de code moeilijker om te lezen. Om deze nesting te verminderen en de methodes leesbaarder te maken, zullen deze methodes in meerdere kleine methodes opgesplits moeten worden met vanzelfsprekende namen. Veel van deze methodes zullen korte if checks zijn, deze methodes hoeven niks te return maar kunnen wel een exceptie gooien als de if statement FALSE is. Deze methodes zullen dan in een try catch blok gezet worden in readSwap en readMove in de volgorde van prioriteit van exceptie.

Server: De Server behoort tot het CONTROLLER onderdeel omdat het met de instructies van de PlayerHandlers de models aan moet passen. Het heeft een Game object die bijhoud hoe het spel verloopt voor alle spelers en deelt dit dan ook via de PlayerHandlers met alle Clients. De skipped Methode in de server word aangeroepen als een PlayerHandler een lege Move stuurt en daarmee aangeeft dat hij z’n beurt wil overslaan, als de skipCount die verhoogt word in de skipped methode gelijk is aan het aantal spelers dan herkent de server het als een sein om het spel te beindigen en de winnaar te BroadCasten met de broadCast methode. Terwijl ik dit schijf bedenkt ik me dat het onlogisch om het spel te eindigen als iedereen maar 1 keer geskipped heeft. Een speler die voor staat zou dan namelijk (als hij de laatste is om te skippen) kunnen bepalen of hij de game eindigd met z’n score en kan dan makkelijk winnen.

De Server vangt geen excepties op voor players die hun verbinding met de PlayerHandler verbreken, dat zou wel moeten anders wordt er op een beurt op een speler gewacht die niet meer mee speelt, als er dan ook geen Aitime ingesteld is (de Aitime timer is overigens niet geimplementeerd) dan zou het spel dus vast lopen.

Board: Het Board behoort tot het MODEL onderdeel omdat het alleen maar informatie bevat en alleen hier zelf aanpassingen op mag en kan maken. Het Board bevat een Tile[][] die het speelveld moet simuleren en heeft hier functies voor om te checken of bepaalde Moves volgens de spelregels op het veld geplaatst mogen worden. Board houd ook bij welke Moves op het veld gedaan zijn met de methode getUsedSpaces. Verder is er nog de methode boardAddMove om een Move daadwerkelijk een Tile op het veld te leggen en boardRemove om plaatsen op het veld leeg te maken.

De validMove methode in Board heeft hetzelfde probleem en dezelfde oplossing als de readMove en readSwap methode in PlayerHandler.

Player: De Player is deel van het MODEL onderdeel. Het heeft een Set<Tile> genaamd hand waar de Tiles in zitten die de player in bezit heeft, een Board die hij zelf gebruikt om de toestand van het veld te kunnen opslaan, dit board word door de Client geupdate. De Player Klasse heeft ook nog een ArrayList<Move> waarin tijdelijk de Moves opgeslagen worden tijdens een beurt zodat bij een undo in de Client de laatst toegevoegde Move terug gedraaid kan worden van het Board van de player. Player had beter een Interface kunnen, omdat er meerdere versies van een Player zijn maar tijdens het maken van het project is dat niet zo gemaakt, op het moment zijn er ook maar 2 verschillende Players dus maakt het niet zo uit.

ComputerPlayer: De ComputerPlayer is een extensie van Player en dus onderdeel van het MODEL onderdeel. Het is niet compleet geimplementeerd, maar het was de bedoeling dat het een player zou zijn die determineMove uit de meegegeven Strategy zou gebruiken om te bepalen waar op het veld hij een tile wil plaatsten. Door de ComputerPlayer verschillende Strategys mee te geven zou je verschillende AI met ander gedrag kunnen laten spelen.

Move: Move is deel van het MODEL onderdeel in het MVC patroon. Move bestaat uit een Tile en een Coord. Het bevat de informatie die nodig is om een Tile op het veld te leggen.

Coord: Coord is deel van het MODEL onderdeel. Coord bestaat uit twee integers, een x en een y die de coordinaten op het veld voorstellen. Verder bevat Coord nog een methode waarmee alle coordinaten om een coordinaat te vragen in de vorm van een Coord[].

Tile: Tile is deel van het MODEL onderdeel. Het bestaat uit twee enums, een Shape en een Color die samen een ‘ tile’ zouden simuleren. Het checken van een verzameling Tiles op een Tile was niet mogelijk met de .contains methode, (dit resulteerde altijd in een FALSE ook al zou .contains() altijd moeten checken of er een element gelijk is aan het obeject waarmee het vergeleken word en niet checken of er een element is dat hetzelfde object is als het object dat vergeleken word) daarom is er no een tileInHand methode geschreven om te checken of een Tile object in een Set<Tile> zit. Het was jammer dat we in de toString methode geen gekleurde characters konden maken, dat had namelijk veel aantrekkelijker gestaan (gekleurd en 1 character die de Tile representeerd).

Strategy: Strategy is deel van het MODEL onderdeel. Het is een inteface die een determineMove methode bevat.

RetardedStrategy: RetardedStrategy is een instantie van Strategy en dus een deel van het MODEL. Het is de enige implementatie van Strategy. De RetardedStrategy kijkt alleen of hij een van de stenen in een hand kan plaatsen naast een van de usedMoves van een Board. Het Board en de hand zouden meegegeven moeten worden tijdens de creatie van een ComputerPlayer.

Game: Game is deel van het MODEL onderdeel. Het houd bij wie er aan de beurt is, welke spelers er aan het spel meedoen, en heeft een TileBag voor het spel. Game heeft methodes om player nummers van players te sturen, een player uit het spel te kicken de beurt te verzetten. Verder heeft Game nog eenvoudige getters en setters.

Client: Client is de complete VIEW. Het luisterd naar input van de speler en verstuurd deze naar de PlayerHandler van de Server waar het spel gespeelt word. Client weergeeft in een textuele vorm wat er binnen komt vanaf de server en wat er van de Speler als input verwacht wordt. De Client heeft altijd een server nodig, door het gebrek aan tijd kon er geen offline Client meer gemaakt worden.

De Client vangt de exceptions die gegooid worden als de verbinding met de server verbroken word niet op, eigenlijk zou er een betere catch gedefineerd worden voor het try block in run, die een message met bijvoorbeeld ‘ The Connection to the Server was lost’. Het gooien en vangen van excepties had bij veel van de input lezende methodes ook geimplementeerd moeten worden. Op het moment worden er veel if statements gebruikt die checken of de invoer klopt, als de invoer niet klopt dan word het op het scherm geprint, soms wordt het simpel weg genegeerd. Op deze plaatsen moeten dus excepties gegooid worden die in de niet bestaande try catch blokken opgevangen zouden worden.

**Met welke structuur word de informatie verstuurd tussen de spelers?**

De spelers ontvangen de protocol commando’s met structuur:

* WELCOME Playername Playernumber

Dit is een speler specifiek commando, hiermee geeft de server aan dat de speler succesvol aan het spel toegevoegd is, daarnaast ontvangt de speler ook z’n Playernumber, zodat hij weet wanneer hij aan de beurt is.

* NAMES [Playername Playernumber] AITime

Dit commando word naar alle spelers gestuurd zodra het spel begint, hiermee krijgen de spelers de informatie over de andere spelers en de Aitime die zal bepalen hoe lang een speler er over mag doen om een Move te sturen (het is niet gespecificeerd in de protocol wat er gebeurt op de server als de Aitime verlopen is (kick of MOVE empty?)).

* NEXT Playernumber

Dit commando word naar alle spelers gestuurd om aan te geven welk playernumber er aan de beurt is.

* NEW [Tile] / NEW empty

Dit commando is speler specifiek, het geeft een aantal Tile.toString() mee met tiles die de speler aan z’n hand moet toevoegen. Dit commando word gestuurd aan het begin van het spel waneer iedereen een hand van de server moet ontvangen, na een beurt waarbij de player Moves heeft gemaakt en na het ruilen van stenen. NEW empty word gestuurd nadat een beurt gedaan is en de TileBag leeg is.

* TURN Playernumber [Tile Row Column]/empty

Dit commando word naar alle spelers gestuurd. Het bevat een verzameling Move.toString() die tijdens een beurt van een andere speler gedaan zijn zodat de andere players hun boards kunnen aanpassen. TURN empty word gestuurd als de vorige speler zijn beurt heeft overgeslagen.

* KICK Playernumber Tilesbacktostack Reason

Dit commando word naar alle spelers gestuurd als een speler van het spel gekickt is, het geeft aan welke Tiles er terug de TileBag ingegaan is zodat sommige AI hierop kunnen speculeren en een rede voor het kicken.

* WINNER Playernumber

Dit commando wordt naar alle spelers gestuurd nadat alle spelers een keer een beurt over geslagen hebben en het spel dus als beeindigt word beschouwd. De Playernumber van de winnende speler word mee gestuurd.

De implementatie van de Server kicked iemand wanneer er een ongeldig commando ontvangen word.

**Verslag van de unit testklassen van de hoogst WMC scorende klassen.**

Wegens het gebrek van tijd zijn er geen test klassen gemaakt, dus hier is weinig over te zeggen.

**Metric verslag.**

De belangrijke onderdelen van metrics die ik deze module geleerd heb zijn:

Cyclomatic complexity: gebruikt de complexiteit van het programma aan te geven. Het is een kwantitatieve maat van het aantal lineair onafhankelijke paden door een programma broncode. Met een control flow graph van een programma, een gestuurd schema met basis blokken van het programma, waarbij elk pijl een beslissing in het programma voorsteld van staat 1 naar 2.

De complexiteit M word gedefineerd als M = E-N+2P, waar E=aantal pijlen, N= aantal staten en P= verbonden component.

Weighted Methods per Class: Het WMC metric is de som van de complexiteit van alle methoden van de klasse. Het geeft aan hoe veel moeite nodig voor het ontwikkelen en onderhouden van een bepaalde klasse.

Lack of cohesion of methods: Deze Metric meet de correlatie tussen de methoden en de lokale instantie variabelen van een klasse. Hoge cohesie geeft goede klasse indeling. Gebrek aan samenhang of lage cohesie verhoogt complexiteit. Klassen met lage cohesie kan waarschijnlijk worden onderverdeeld in twee of meer subklassen met verhoogde cohesie.

Efferent Coupling: Het aantal klassen gerefereerd in een klasse.

Afferent Coupling: Het aantal klassen die refereren naar een klasse.

**Reflectie op de planning en self-management tijdens het project.**

Aangezien dit verslag laat ingeleverd word en het project niet helemaal af is zou het moet blijken dat de planning dus niet voldoende was. De planning was niet beinvloed door de planning van het design project en de bijhorende acadamic skills activiteiten. Er was verwacht dat de programmeer activiteiten wel af zouden komen tijdens de geplande tijdvakken, alleen was in het begin van het project een beetje verwarring wat precies gemaakt moest worden wat het moeilijk maakte om te beginnen. De start van het project was dus niet als gewenst. Tijdens de laatste anderhalve week is de planning compleet uit de hand gelopen, er was een achterstand onstaan en de project groep was opgesplitst (ik zal het daar over hebben in de *commentaar*) wat zorgde voor veel meer werk per persoon. Om de achterstand te proberen in te halen is er hulp gezocht bij jaar genoten, deze jaargenoten hebben geen code geschreven, maar wel geholpen met het sneller identificeren van fouten en nodige methodes. Jammer genoeg was de hulp niet genoeg om de achterstand in te halen en het extra werk af te maken, met dit oncomplete project als getuigen.

Tijdens het volgende project zal er waarschijnlijk niet meer zo veel of geen verwarring meer ontstaan waarmee begonnen word, zodat er vanaf het begin al goed gewerkt kan worden. Ik zal ook eerder actie ondernemen als ik twijfels heb over een project parter om te verkomen dat ik zelf zonder voorbereiding meer werk zal moeten doen.

Als ik een SA was voor het volgende eerste jaar dan zou ik tijdens week 6 zeggen dat ze een stub moeten maken tijdens de vakantie, dan hebben ze een veel beter idee wat er precies allemaal gemaakt moet worden en dan is het makkelijker om in te schatten hoeveel tijd dingen gaan kosten, daarnaast zal ik ook zeggen dat ze in de planning genoeg uitloop tijd moeten inplannen zodat ze dingen waar ze eerder niet aan gedacht hadden nog kunnen implementeren met een comfortabele werk tijd.

**Commentaar.**

Tot week 7 heb ik geen project partner gehad, tot dat ik op de slack van de module (een soort van groepschat applicatie) iemand (ik weet de achter naam niet zeker) vond die een project partner aan het zoeken was, ik stelde me voor en we besloten samen als project groep door te gaan. De eerste project week was er niet veel progressie gemaakt, maar het voelde alsof we nog wel alles af konden krijgen. We maakte het design en de structuur voor het programma en wat stub code. De volgende project week merkte ik dat m’n project partner niet goed kan programmeren, herhaaldelijk heb ik hemn geholpen met het schrijven van code, ik heb ongeveer alles zelf geimplementeerd dat tot toen geimplementeerd was. Vaak begon hij met het maken van methodes die onnodig waren of al op een andere manier bestonden, omdat hij deze zag staan in andermans project. Pas een week voor de deadline van het project kwam ik er achter dat hij de meerderheid van de programmeer opgaves heeft afgekeken en de programmeer toets slecht gemaakt heeft, hij kan dus niet programmeren. Voordat ik hier achter kwam heb ik steeds verwacht dat hij iets zou maken en wel zou kunnen programeren sinds hij tot en met week 7 afgetekend had, maar het bleek dat hij deze oefen opgaves waarmee men moet leren programeren allemaal afgekeken had. Ik ben me zorgen gaan maken en de dag nadat ik erachter kwam dat hij niet kan programeren dit probleem besproken met de module coordinator, waarbij ik besloten heb om de project groep op te splitsen. Op dat moment was er al een grote achterstand en bleek het werk dat nog gedaan moest worden veel meer dan gedaan kon worden omdat ik het alleen moest doen.