**NU Module I Eindopdracht GoGame**

*Renske Schram*

**Design Go Game**

Voor mijzelf heb de opzet opgebouwd vanuit drie aspecten: de server-kant, de client-kant en de gemene deler tussen beiden. Aan de serverkant wilde ik het accepteren/overzien van clients, aanmaken van nieuwe Games en daarin natuurlijk ook het correct kunnen uitvoeren van een Game en diens regels. Aan andere kant wilde ik een speler die juiste doorgeeft welke set wordt gedaan dan wel zelf een nieuwe set verzint. Om die client-speler ook “slim” te kunnen laten zijn, moet deze en diens strategie ook bewust zijn van het spel en diens status. Daarom heb ik de code zo opgebouwd dat beide “kanten” gebruik maken van de gemene delers zoals de Player die zetten doet, Game met Board voor de status en regels van het spel een versie van de SocketConnection. Zodoende ben ik begonnen met het uitwerken van het onderstaande Go Game design:

A diagram of a computer

Description automatically generated

*Board & Game*

Mijn Board en Game vormen de basis van het spel, waarin het bord de huidige status van de stenen bijhoudt en de Game zorgt dat alles met de spelers daarbij ook volgens de regels verloopt. Achteraf gezien ben ik snel vanuit dit idee aan de slag gemaakt en vanuit daar alles werkend proberen te maken, terwijl ik misschien beter soms even een stap terug had kunnen nemen om het overzicht te krijgen en verbeteringen toe te passen. Zo zie ik achteraf ook in dat ik bijvoorbeeld hele grote classes heb gemaakt met mijn Board, Game en GameServer en was het met het ook op object georiënteerd programmeren beter geweest als ik deze in kleine klassen (en dus objecten) had opgesplitst. Zo geef ik in mijn Board aan veel functies een Stone/Intersection mee die ik had kunnen vangen met een losse class die een locatie en kleur heeft en/of een Group class omdat nu aan veel methoden een groep van stenen meegeef, terwijl ik die had kunnen onderscheiden als losse functionaliteiten van een groep stenen. Ik heb een eerste idee hoe ik dat nu zou aanpakken laten zien door de methodes in deze classes te onderscheiden van elkaar. Zo zou een Board denk ik ook juist Intersections, Stones, Groups en bij de Game onder andere Move, MoveTimer (en misschien wel een samenvoeging van de Rules tot één class) kunnen afsplitsen.

Ik denk wel dat beiden classes logische eigenschappen hebben en daar heb ik denk ik vaak ook goede encapsulation op kunnen toepassen. Bij de methoden/functies is dat wat minder goed gelukt. In mijn enthousiasme aan het begin heb ik ook niet altijd even goed gelet op encapsualtion, waardoor ik veel functies in eerste instantie op public had gezet om alles eerst werkend te krijgen. Omdat ik ook mijn ComputerStrategy gebruik wilde laten maken van de Game en Board en diens functies, waardoor ik nu aan het eind niet meer de tijd had om de structuur daarin te veranderen en verbeteren om functies wel meer af te schermen per class en heb ik eerst op andere dingen gefocust.

Ik heb tijdens de eerste dagen wel gekozen om de spelregels in twee beiden classes af te vangen. Er zijn moves die niet mogen op basis van het bord (heel praktisch omdat deze locatie niet bestaat of dat deze al bezet is) of meer op Game niveau (op basis van regels omdat het bijvoorbeeld niet iemand beurt is of de ko-regel).

De functies in het Board lijken misschien ook wat omslachtig (waarom een lijst teruggeven van intersection/liberties en niet gewoon zeggen of het vrij is of niet of een aantal liberties geven?), dat heb ik nu zo vormgegeven zodat uiteindelijk mijn computerspeler ook zoveel mogelijk informatie uit het Board en de vrije plekken zou kunnen halen als nodig is om zelf een nieuwe set te bepalen. Uiteindelijk niet alle tijd genomen om daar ook gebruik van te kunnen maken bij mijn ComputerStrategy, maar daar komt het idee van die opbouw in ieder geval vandaan.

*Player*

Ik heb mijn Player ingedeeld als een relatief simpele abstract class, waarin mijn spelers een kleur, naam en mogelijk spel hebben. Spelers kunnen zetten doen, dus als vanuit de connectie een zet binnenkomt, geeft de speler met zijn toegewezen kleur door aan het spel welke zet het echt is. Ik heb ervoor gekozen om de communicatie tussen de Game en aan de ServerConnection te laten lopen via deze speler. Een spel heeft spelers en als er een update is voor een speler, dan gaat dat ook via die speler. Op die manier blijft de lijn van communicatie direct (met minder kans op fouten) en hoeft het bijvoorbeeld niet via een server (die ik echt wilde houden voor alleen het connecten/disconnecten en starten van spellen). Ik kan me ook voorstellen dat bij een grotere opzet het prettiger is om wel met listeners e.d. te gaan werken en dan mee te geven naar wie het bericht moet.

*SocketConnection, ServerConnection & ClientConnection*

De SocketConnection en child classes Server/ClientConnection vormen de basis van de communicatie tussen de server en client-kant. Het idee was dat de SocketConnectie puur het leggen van de verbinding en doorgeefluik van String-berichten is. De Server/ClientConnections ontvangen die input en ontcijferen die op basis van het protocol en sturen de Server of Clients aan om de juiste vervolgactie te ondernemen. Dat is grotendeels gelukt. Maar zo staat er nu in de SocketConnection class twee functies om vanuit andermans input een Color en Intersection te krijgen. Dar was in eerste instantie met het idee dat deze connectie de “ruwe” input behandeld die ontvangen is vanuit een speler, dus dan kan ook een functionaliteit zijn om het te vertalen naar de juiste/bruikbare input, maar dit had misschien ook mooier in een Move/Stone object gekund om die uit verschillende input waarden te kunnen construeren. Zoals te zien is in de repository ben ik ook begonnen aan de Move class omdat deze combinatie van een locatie en kleur natuurlijk zo een gemene deler is in heel veel functies en ik daarin ook variabelen en functies had kunnen toevoegen om deze Moves op allerlei mogelijke tot stand te laten komen. Uiteindelijk omwille van de tijd is het me niet gelukt om dat af te maken en heb ik eerst gefocust op andere onderdelen.

Aan de client-kant heeft de ClientConnection nu ook een Game. Dat is eigenlijk niet helemaal logisch, maar nu wel zo tot stand gekomen zodat de Game en Board direct worden bijgewerkt voor de Strategy&GUI op basis van de ontvangen set van de Server-kant.

*Bij meer tijd…. meer en beter testen*

Ik heb de afgelopen weken vooral gefocust op het werkend krijgen van het geheel en ik ben blij dat dat me in ieder geval gelukt is. Zoals al aangegeven zou ik het een volgende keer met alle nieuwe kennis/inzichten uiteraard anders opzetten vanaf het begon, maar, als ik nu met deze opziet iets meer tijd had gehad, dan zou ik nog verder willen hebben verdiepen in het testen en ook het vastlopen gebruiken om in te zien hoe ik volgende keer zo een project anders zou opbouwen.

Vanuit de opdrachten en oefeningen op de UT ben ik begonnen met leren hoe ik tests moet schrijven en die kennis kwam vooral goed van pas tijdens het testen van mijn Board, Game etc. Daarin heb ik soms ook eerst de test geschreven en dan de functie en had ik goed voor ogen hoe ik het aan wilde pakken. Maar het testen rondom de netwerkverbinding vond ik lastig en heb ik daarom met unit-tests in ieder geval niet zover kunnen uitwerken als dat ik zou willen. Uiteindelijk heb ik besloten om dat voornamelijk te doen met system-tests. Deels hoop ik dat dat komt omdat dat ook een lastig aspect is, maar ik denk dat het ook deels komt door mijn opbouw. Game, Board, Player, daar had ik allemaal wel een duidelijk beeld voor ogen welke eigenschappen ze hebben en wat ze kunnen doen. Zo heb ik vandeweek nog geleerd dat ik de testcoverage kan inzien (zo hieronder de figuur) en daarin is ook terug te zien dat de testcoverage van mijn Board en Game >94% is en van alle methoden en heb ik daar zoveel mogelijk op basis van complexity proberen te testen. Maar “de communicatie” is natuurlijk iets wat door alle classes heen loopt en een netwerk vormt om verschillende acties in gang te zetten. Dus als ik rondom het netwerken wilde testen, dan zette ik automatisch andere classes in gang die ik dan moest initiëren. Ik denk dat daarin heel mooi terug te zien is dat ik de code nog iets meer had mogelijk compartimentaliseren omdat makkelijker te maken, maar ook meer kan leren hoe je zulke testen goed kan opbouwen.

A screenshot of a computer screen

Description automatically generated

*TUI’s & GUI*

Opvallende classes met lage unit-test coverage zijn de TUI’s en GUI*.* De GUI implementatie heb ik nu erg simpel gelaten. Wordt een move goedgekeurd door het sever-spel en ontvangen door de client dan wordt deze gezet op het board van de client-speler (die deze kant printen) maar ook direct op de GUI. Deze heb ik vooral getest met system-testing en vergeleken met de geprinte Boards en dat gaat goed en ben ik geen fouten tegengekomen. Ook de StrategyGame is laag qua test-coverage, de toevoeging ten opzichte van de Game class is uiteindelijk alleen het activeren van de GUI, dus die is ook vooral met system-tests getest.

A board game with black and white balls

Description automatically generatedA black and white background with many circles

Description automatically generated

Beiden TUIs hebben redelijk dezelfde opzet, waarbij het opstarten de inputs worden gevraagd om de server op een bepaalde port te starten of de client te verbinden met de server op een bepaald server InetAdress en PORT (en ook meteen het kiezen van een spelersstrategie). Ook heir heb ik vooral system-tests gedaan en eventuele foute inputs van de port en inetadress worden afgevangen met try-catch en exceptions, waarbij een nieuwe poging gedaan kan worden.

A black screen with red text

Description automatically generated

*Strategy*

Verder is ook het testen van mijn Strategies laag. Het idee van de Strategy interface is dat als je met HumanStrategy speelt, een bepaalde actie wordt ondernemen door via de TUI te vragen om een actie waar je als “mens” op kan reageren en als je als computer speelt, dat deze acties allemaal automatisch worden ondernomen. Voor mijn HumanStrategy heb ik daarom vooral weer system-tests uitgevoerd en gecheckt of de berichten juist worden doorgegeven en de strategie automatisch goed wordt gezet. Voor de ComputerStrategy heb ik deze lang op een random valid move generator laten staan, waardoor ik hier niet veel op heb getest en steeds tegen elkaar laten uitspelen om te zien of er geen random/valid moves werden gestuurd tegen het einde van het spel. Ook bestaan veel functies daarvan wel uit combinaties van Board functies die daar wel al zijn getest, waardoor ik dit met oog op de tijd geen prioriteit heb gegeven.

Ik heb nu wel nog toegevoegd dat mijn AI in zijn eigen bord gaat uitproberen of een set een grotere territory oplevert en de set maakt wat het meeste oplevert. Mocht er geen verbeterende set zijn, dan kijkt de computer of hij voor/achter staat. Als deze achter staat, dan wordt een random move gekozen om hopelijk nog iets te kunnen forceren, als deze voor staat dan wordt er gepasst. Als gevolg dat spelletjes met mijn eigen computers heel lang door kunnen gaan wegens het ontbreken van afkeuren van suicide. Als dat mag, dan ik dat misschien dat voor vrijdag nog even aanpassen, dat mijn speler na zoveel random zetten om zijn hachje te redden toch na een bepaalde tijd passt met het gevolg te verliezen.

*Reflectie*

Volgende keer neem ik mee dat ik vaker tussendoor ook weer het overzicht wil nemen en wil herschikken in mijn plan dat ik had. Nu had ik het idee, alles moet eerst werken en dan maak ik het beter. Dat ging grotendeels goed, maar nu achteraf wanneer ik alles beter wil maken blijft er (1) niet genoeg tijd over om te doen wat je wil doen en (2) kost het verbeteren meer tijd dan als ik dat in het begin al goed had aangepakt. Een manier om dat te doen is denk ik toch ook meer toepassen om eerst tests te schrijven en daarna te coderen, op die manier moet ik mijzelf dan meer stimuleren om beter na te denken wat er precies moet gebeuren en onder welke voorwaarden en het op die manier werkend maken.

Naast alle technisch inhoudelijk bevindingen heb ik ook een andere, voor mij in ieder geval nuttige, bevinding gedaan en dat is dat ik mijzelf nog iets meer zou willen laten toelaten om te mogen leren.

Ik merkte dat ik dit project met super veel plezier en enthousiasme heb gedaan en elke dag weer vol energie aan de slag ging, maar richting het einde ineens ook steeds onzekerder werd over veel onderdelen, maar niet de rust/tijd had om daarin grote veranderingen te maken. Toen jij/Björn maandag vroeg of we nog even samen naar de code wilden kijken, had ik het liefste “nee” geroepen, omdat ik het best wel spannend vond. Des te blijer ben ik dat ik het wel heb gedaan, omdat ik wel heel veel had aan alle feedback. Niet omdat ik veel heb kunnen veranderen (daarvoor waren de grote dingen wat te laat), maar wel om mezelf weer even te prikkelen dat het oké is dat ik wel nog dingen moet leren en dat ik met die feedback alweer heel anders naar mijn code kon kijken voor een volgende keer. Uiteindelijk zijn er veel dingen die ik beter had willen doen, maar een maand gelden wist ik überhaupt nog niet precies hoe een netwerk tot stand kan komen, dus ben ik ook zeker heel blij met hoe ik dit alles zo tot een werkend geheel heb kunnen brengen.