Module 1 Nedap University.

Code review

Over het algemeen ben ik redelijk tevreden met het eindresultaat wat ik in de afgelopen 2.5 week heb kunnen neerzetten. Ondanks het feit dat het een zeer tijdrovende opdracht was heb ik mij erg vermaakt met het bouwen van de go client-server applicatie. Het project heeft mij duidelijk gemaakt dat het ontwerpen van een dergelijke applicatie meer vaardigheden vereist dan het schrijven van wat simpele algoritmes. De afgelopen weken hebben mij meer duidelijkheid verschaft in de valkuilen en moeilijkheden van het ontwerpen van een simpele applicatie.
De eerste les die ik heb geleerd is *"niet te snel tevreden zijn met iets dat werkt"*.

Tijdens mijn vorige studie was het af en toe noodzakelijk een simpel scriptje te schrijven voor een relatief eenvoudig probleem. De uitvoering, netheid en effectiviteit van dit soort scriptjes was voor ondergeschikt aan de uiteindelijke uitkomst. Het is mij de afgelopen periode snel duidelijk geworden dat dit niet absoluut niet geldt voor het schrijven van grotere applicaties. Alles wat je hebt geschreven zal je in een later stadium van het project weer nodig hebben. Dit maakt dat je geschreven code niet alleen zijn functie moet uitvoeren, maar dat dit ook effectief gebeurt, dat het begrijpelijk is geschreven en dat het makkelijk in te bouwen is in andere stukken code. Dit is iets waar is snel tegenaan ben gelopen. Het schrijven van te lange functies die volledig aan elkaar gebreid waren resulteerde in een basis waar niet meer mee verder gewerkt kon worden.

Daarnaast was het ontbreken van een strijdplan voorafgaand aan het beginnen met schrijven van de code een valkuil. Hierdoor loop je aan het eind tegen dingen aan die je vooraf het kunnen voorkomen. Dit zijn lessen die ik kan gebruiken tijdens verdere opdrachten.

In het begin zijn we begonnen met het opstellen van het protocol. Hierin heb ik mij redelijk afzijdig gehouden aangezien een paar jongens het voortouw namen. Achteraf is dit denk ik een goede keuze geweest en hebben we snel een protocol tot stand gebracht (Achteraf was iedereen het wel eens met het feit dat als we nu het protocol opnieuw moesten maken dat het er anders uit had gezien).

Zelf ben ik begonnen met het ontwerpen van het Go spel zelf. Na wat drastische veranderingen bestaat het spel uit 3 Classes.

- Intersection
- Group
- Board

De intersectie klasse is de meest fundamentele klasse van de drie en representeert een intersectie op het go bord. Bij de constructie van een intersectie wordt een positie meegegeven doormiddel van een int Row en int Col. Daarnaast wordt meegegeven bij welk Board de intersectie hoort. Als laatste kan een intersectie een onderdeel zijn van een Group. Bij de constructie van een intersectie is Group standaard null.

De board klasse representeert het bord waar Go op gespeeld kan worden. Het Board (NxN) is variabel in formaat en bestaat een multidimensional arrayList met intersecties. Op het bord kunnen stenen (Zwart wit) gelegd worden waardoor een intersectie een Group vormt of toegevoegd wordt aan een Group.

Een Group is een verzameling aan intersecties (Set) en een verzameling aan vrijheden (Set) .

Deze 3 groepen samen vormen het speelbord van Go. Over het algemeen ben ik tevreden met dit ontwerp. Een Keuze waar ik later last van heb gehad is de keuze om een lege intersectie een null pointer als groep mee te geven. Achteraf had ik beter kunnen kiezen voor een design waar lege intersecties samen ook groepen vormen. Hier had ik helaas geen tijd meer voor tijdens het project.

Voor het hanteren van de regels van het spel heb ik drie klasses gemaakt die static methoden bevatten. Deze 3 klasses zijn.

- Movevalidator
  Deze klasse bevat methoden die kijken of een set valide is. Dit wordt gebaseerd op 3 regels. Een vak moet leeg zijn, een vak moet zich op het bord bevinden en een set mag niet resulteren in een voorgaande bord situatie. Deze functie heeft 4 attributen nodig: Het bord, de gewenste set, bord geschiedenis en kleur van de steen.

- EnforceRule
  Deze klasse checkt of er stenen van het bord verwijdert moeten worden nadat er een set is gedaan. Dit wordt gedaan door het kijken naar de vrijheden van groepen.

- Score
  Deze klasse berekent de uiteindelijke score voor beide spelers.

Zelf ben ik tevreden met de keuze om deze functionaliteit te verweken in static methods en het verdelen van de functionaliteiten in deze 3 klasses.

De 3 bovenstaande klasses, Board, Intersection en Group samen vormen de Basis functionaliteiten om een Spel Go te spelen. Het organiseren van de gameflow behoord tot de verantwoordelijkheden van de server. De opbouw hiervan wordt hieronder beschreven.

De server is een Threaded class. Wanneer de server wordt gestart wordt er aan de gebruiker een poort gevraagd. Wanneer een geldige poort is gegeven wordt er een serversocket gemaakt en word de thread gestart. Wanneer een client probeert te verbinden met de server wordt er een Socket gecreëerd en een ClientHandler gecreëerd. Deze ClientHandler wordt gekoppeld aan een Game.

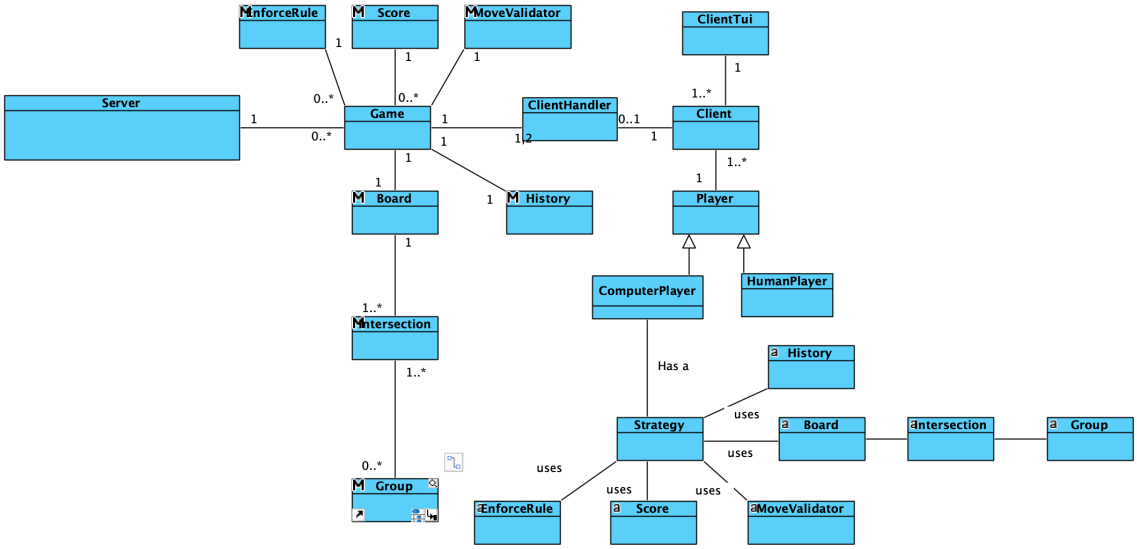
Deze Game klasse is het hart van de applicatie. Via deze klasse worden alle berichten ontvangen, verwerkt en verstuurd. Daarnaast bevat deze klasse alle game logica en zorgt deze klasse voor een goede gameflow.

De Threaded client klasse is in staat om te communiceren met een server. Deze maakt gebruik van de ClientTUI klasse die afhankelijk van de gegeven output en gekregen input een menu toont die gebruikt kan worden om de gewenste commando's te versturen. Afhankelijk van de wens van de gebruiken wordt een ComputerPlayer of HumanPlayer gecreëerd. Deze beide klassen zijn subklassen van de abstracte klasse Player. Een Humanplayer vraagt de gebruiker input voor een gewenste move. Een Computer player heeft een strategy. Voor de strategy is een interface gemaakt. De strategy is zelf instaat om een zet te bepalen met behulp van Board, Intersection, Group, EnforceRule, MoveValidator, Score en History.

In figuur 1 is een overzicht te zien van alle klassen en hoe deze met elkaar verbonden zijn. (Ik ben me ervan bewust dat het een zeer onvolledige en deels onjuiste klassen diagram is, maar wegens

tijdnood heb ik er weinig aan kunnen veranderen en ook geen andere UML diagrammen kunnen maken).

Een grove tekortkoming in mijn project is het ontbreken van vele tests. In principe zijn alleen Board, Intersection, Group, MoveValidator, Score en EnforceRule gestest. Dit had uitgebreider en beter gemoeten. Geen tijd voor gehad/andere prioriteiten gesteld.



Figuur 1

Verbeterpunten applicatie.

- Meer testen
- Minder omslachtige puntentelling
- Beter omgaan met het weg gaan van een speler
- Duidelijkere algehele structuur
- Betere beschrijvingen methoden (te gehaast)
- Betere exception handeling
- Netter werken

Conclusie

Over het algeheel ben ik niet geheel ontevreden over het verloop van het project, maar ook zeker niet volledig tevreden. Ik heb mezelf iets teveel de nadruk gelegd op het afronden van alle eisen die gesteld werden aan onze applicatie, waardoor ik iets te gehaast heb gewerkt. Dit resulteert is een applicatie die alle benodigde functionaliteiten bevat, maar rommelig, slecht getest en instabiel is. De volgende keer moet ik meer de nadruk leggen op kwaliteit boven kwantiteit en netter werken.