## ClientTUI.java

### Rol van de klasse in het systeem

ClientTUI is de TUI (Textual User Interface), die de gebruiker te zien krijgt zodra hij het programma opstart. Deze klasse regelt het vragen van de naam, IP-adres en poort. Zodra deze bekend zijn, en er een succesvolle verbinding met de server tot stand is gebracht, wordt geluisterd naar commando’s van System.in. Daarnaast laat het berichten aan de gebruiker zien.

### De verantwoordelijkheden van deze klasse

Deze klasse is verantwoordelijk voor het doorsturen van ingevoerde commando’s naar Client.java, en het tonen van berichten die door Client zijn verstuurd.

### De andere klassen die door deze klasse gebruikt worden om zijn verantwoordelijkheden te bereiken

Client voor het ontvangen van commando’s en het doorsturen van berichten die moeten worden laten zien in de TUI.

### Voorzorgsmaatregelen om de precondities in the server te vervullen

Geen.

## ClientView.java

### Rol van de klasse in het systeem

ClientView is een interface. Deze wordt door ClientTUI geïmplementeerd. Het bevat een aantal methoden die een UI sowieso moet hebben. Mochten we kiezen om een GUI te maken, dan kan die ook deze interface implementeren.

## LeaderboardPair.java

### Rol van de klasse in het systeem

Houdt de score van spelers bij, en heeft een functie die ervoor zorgt, dat twee LeaderboardPairs op een juiste manier vergeleken kunnen worden. Op die manier is het mogelijk om het Leaderboard te sorteren.

### De verantwoordelijkheden van deze klasse

Ervoor zorgen, dat LeaderboardPairs goed gesorteerd kunnen worden en de scores bijhouden.

## MessageUI.java

### Rol van de klasse in het systeem

Deze interface wordt gebruikt als een globale interface voor UI’s. Deze klasse wordt geextend door ClientView en geïmplementeerd door ServerGUI.

## Server.java

### Rol van de klasse in het systeem

Server is de klasse die nieuwe verbindingen/clients accepteert en er een nieuwe ClientHandler thread voor maakt. Ook is server verantwoordelijk voor het versturen van berichten tussen ClientHandlers, het bijhouden en updaten van invites en het bijhouden, updaten en opslaan van het leaderboard.

### De verantwoordelijkheden van deze klasse

Zorgen dat er nieuwe ClientHandlers worden gestart voor elke verbinding en dat alle ClientHandlers worden bijgehouden, zorgen dat berichten bij de juiste ClientHandlers aankomen, zorgen dat de invites goed worden bijgehouden en geüpdatet en ervoor zorgen dat het leaderboard gesorteerd blijft, wordt bijgehouden, geüpdatet wordt en wordt opgeslagen.

### De andere klassen die door deze klasse gebruikt worden om zijn verantwoordelijkheden te bereiken

LeaderboardPair om het leaderboard bij te houden en gesorteerd te houden en ClientHandler om de communicatie met een nieuwe client op te zetten.

### Voorzorgsmaatregelen om de precondities in the server te vervullen

De ClientHandlers controleren de precondities voordat ze de methode in server aanroepen en de server heeft alle lijsten synchronized zodat er niet iets uit de collectie wordt verwijderd terwijl een andere methode door de lijst aan het bladeren is.

## ServerGUI.java

### Rol van de klasse in het systeem

Geeft het IP van de server weer, en laat de gebruiker een poort kiezen waarop de Server gestart moet worden. Laat na een succesvolle start alles zien wat er op de server gebeurt. Mocht de poort al in gebruik zijn door een andere applicatie, dan wordt er een foutmelding gegeven.

### De verantwoordelijkheden van deze klasse

Een server starten op de juiste poort en alle berichten die doorgegeven worden aan deze klasse laten zien in het berichtvak.

### De andere klassen die door deze klasse gebruikt worden om zijn verantwoordelijkheden te bereiken

Server.java

### Voorzorgsmaatregelen om de precondities in the server te vervullen

Als de ingevoerde poort geen integer is, wordt er een foutmelding weergegeven.

## Board.java

### Rol van de klasse in het systeem

Het regelen van een Board object. Dit houdt onder andere in: moves maken en kijken of een speler gewonnen heeft.

### De verantwoordelijkheden van deze klasse

De moves die doorgegeven worden maken op het bord en op een correcte manier kijken of een speler gewonnen heeft.

## BoardTest.java

### Rol van de klasse in het systeem

Een testklasse voor Board.java. Deze klasse test iedere methode van Board waarvoor dit zinvol is.

### De verantwoordelijkheden van deze klasse

Board.java feilloos testen.

### De andere klassen die door deze klasse gebruikt worden om zijn verantwoordelijkheden te bereiken

Board.java

### Voorzorgsmaatregelen om de precondities in the server te vervullen

Er worden geen onmogelijke bordsituaties getest.

## ComputerPlayer.java

### Rol van de klasse in het systeem

Beheert een ComputerPlayer met een Strategy. Wordt vooral gebruikt vanwege zijn determineMove methode om een move te maken of om een hint te vragen.

### De verantwoordelijkheden van deze klasse

Een valide move returnen volgens zijn Strategy.

### De andere klassen die door deze klasse gebruikt worden om zijn verantwoordelijkheden te bereiken

\*Strategy.java.

## Disc.java

### Rol van de klasse in het systeem

Dit is een enum om de verschillende Discs bij te houden. Een Disc kan YELLOW, RED of EMPTY zijn.

## MinMaxStrategy.java

### Rol van de klasse in het systeem

Dit is onze implementatie van de MinMaxStrategy voor een ComputerPlayer. Deze Strategy bepaalt een score voor de move afhankelijk van hoe vaak hij wint of verliest als hij een aantal zetten vooruitdenkt.

### De verantwoordelijkheden van deze klasse

De move returnen moet de hoogste score. Verder moet de ingestelde diepte gerespecteerd worden.

### De andere klassen die door deze klasse gebruikt worden om zijn verantwoordelijkheden te bereiken

Board.java om de move te bepalen en Strategy.java omdat deze geïmplementeerd word.

### Voorzorgsmaatregelen om de precondities in the server te vervullen

Er worden alleen valide moves gereturned door de determineMove methode.

## NaiveStrategy.java

### Rol van de klasse in het systeem

Deze klasse is voor een ComputerPlayer, die iedere keer een compleet random move op een Board doet.

### De verantwoordelijkheden van deze klasse

Een random valide move op het Board returnen.

### De andere klassen die door deze klasse gebruikt worden om zijn verantwoordelijkheden te bereiken

Board.java, Strategy.java.

## SmartStrategy.java

### Rol van de klasse in het systeem

Deze klasse is voor een ComputerPlayer, die iedere keer een zet vooruit denkt, en dan een move op een Board doet.

### De verantwoordelijkheden van deze klasse

Maak een move waarbij een move wordt gedaan op een Board waarbij 1 zet vooruit is gedacht.

### De andere klassen die door deze klasse gebruikt worden om zijn verantwoordelijkheden te bereiken

Board.java.

## Strategy.java

### Rol van de klasse in het systeem

Een interface voor de andere strategies. Alle andere strategies moeten ten minste een getName en een determineMove methode hebben.

# Test report

Hieronder vindt u per klasse de testgegevens over die klasse. Als een kopje niet van toepassing was op die klasse, hebben we hem weggelaten.

## Client.java

### Hoe is deze klasse getest?

Alle tests die we konden doen zonder een socketverbinding op te zetten, zijn gedaan in ClientTest.java. Verder hebben we deze klasse heel vaak gebruikt om spelletjes mee te spelen, het leaderboard op te vragen en noem maar op.

### Welke test techniek is toegepast?

Visuele inspectie van de UI en het toevoegen van een main methode die een aantal methoden van de klasse aanroept.

### Welke testprogramma’s zijn ontwikkeld?

ClientTest.java

### Wat waren de verwachte resultaten en de echte resultaten?

Wij verwachtten, dat de methoden werken zoals we dat wilden, en dat gebeurde in zowel de ClientTest, als in systeemtesten.

### Hoeveel procent van iedere methode wordt gedekt door tests?

Omdat we geen inputstream konden faken, hebben we maar een coverage in ClientTest van 35,5%. Tijdens het spelen van spelletjes zijn alle commando’s vaak genoeg getest om ook een garantie te geven dat die werken zoals wij willen.

## ClientHandler.java

//TODO: Laat Sven dit fixen.

### Hoe is deze klasse getest?

### Welke test techniek is toegepast?

### Welke testprogramma’s zijn ontwikkeld?

### Wat waren de verwachte resultaten en de echte resultaten?

### Hoeveel procent van iedere methode wordt gedekt door tests?

## ClientTUI.java

### Hoe is deze klasse getest?

We hebben deze klasse heel vaak gebruikt om games te spelen op de server. Verder hebben we, voor zover dat kon, een klasse ClientTUITest gemaakt, die een aantal methoden van ClientTUI test.

### Welke test techniek is toegepast?

Visuele inspectie van de UI en het toevoegen van een main methode die een aantal methoden van de klasse aanroept.

### Welke testprogramma’s zijn ontwikkeld?

ClientTUITest.java

### Wat waren de verwachte resultaten en de echte resultaten?

Dat we een spelletje konden spelen op de server, en dat alle commando’s deden wat ze moesten doen. Dat gebeurde ook. Verder verwachtten wij de resultaten zoals beschreven in ClientTUITest, en die kregen wij ook.

### Hoeveel procent van iedere methode wordt gedekt door tests?

EMMA gaf ons een coverage van 62,3% op de tests.

## ClientView.java

### Hoe is deze klasse getest?

Het is een interface. Deze klasse is niet getest.

## LeaderboardPair.java

### Hoe is deze klasse getest?

Door het Leaderboard op te vragen, dan weer een paar spelletjes te spelen, en dan weer het Leaderboard op te vragen, en kijken of het juist gesorteerd is.

### Welke test techniek is toegepast?

Visuele inspectie van de UI.

### Welke testprogramma’s zijn ontwikkeld?

Geen.

### Wat waren de verwachte resultaten en de echte resultaten?

Dat het Leaderboard goed gesorteerd werd, en dat werd het ook.

### Hoeveel procent van iedere methode wordt gedekt door tests?

De gehele klasse moet goed werken om het Leaderboard op een juiste manier te laten zien. Het Leaderboard wordt telkens juist weergegeven, dus de klasse is voldoende getest.

## MessageUI.java

### Hoe is deze klasse getest?

Het is een interface. Deze klasse is niet getest.

## Server.java

//TODO: Laat Sven dit fixen.

### Hoe is deze klasse getest?

### Welke test techniek is toegepast?

### Welke testprogramma’s zijn ontwikkeld?

### Wat waren de verwachte resultaten en de echte resultaten?

### Hoeveel procent van iedere methode wordt gedekt door tests?

## ServerGUI.java

### Hoe is deze klasse getest?

Iedere keer, dat we de Server opstarten, gebruikten we deze klasse. We hebben deze klasse meer dan voldoende gebruikt, om te kunnen zeggen dat hij doet wat hij moet doen.

### Welke test techniek is toegepast?

Visuele inspectie van de UI.

### Welke testprogramma’s zijn ontwikkeld?

Geen.

### Wat waren de verwachte resultaten en de echte resultaten?

Dat je een de berichten van de server kon lezen, en dat je een poort kon kiezen, en dat werkte perfect.

### Hoeveel procent van iedere methode wordt gedekt door tests?

Alle methoden worden vaak genoeg gebruikt om te zeggen dat ze in het geheel worden gedekt. We hebben voor de zekerheid even EMMA laten lopen terwijl we een normale server openden, en we kregen 89% coverage. Op een paar exceptions na, helemaal getest.

## Board.java

### Hoe is deze klasse getest?

Board.java is getest in de klasse BoardTest.java. Voor vrijwel iedere methode is een zinvolle test ontwikkeld. Voor enkele methoden was het niet zinvol om een test te schrijven, bijvoorbeeld omdat deze niets anders deden, dan te checken of 1 van de 2 calls naar 2 andere methoden true waren. Bij deze methoden hebben we in BoardTest.java een stub gemaakt, met commentaar erin. In dat commentaar staat de reden, dat we die methoden niet getest hebben. Verder hebben we bij het spelen van games bepaald, dat deze klasse prima werkt.

### Welke test techniek is toegepast?

De tests zijn ontwikkeld met behulp van JUnit. De documentatie voor de tests staat bij de code in de vorm van Javadoc. Verder hebben we ook visuele inspectie van de UI toegepast.

### Welke testprogramma’s zijn ontwikkeld?

Voor deze klasse zijn geen aparte testprogramma’s ontwikkeld.

### Wat waren de verwachte resultaten en de echte resultaten?

Dit staat in het commentaar bij de desbetreffende code. Na enkele revisies van Board.java kwamen de verwachte resultaten overeen met de echte resultaten. Verder verwachtten we, dat nadat er een move was gedaan, hij op de juiste plaats in het bord kwam.

### Hoeveel procent van iedere methode wordt gedekt door tests?

Emma gaf ons een globaal percentage van 85,1%. Alle methoden zijn óf voor 100% getest, of voor 0%. De methoden, die voor 0% getest zijn, zijn: gameOver(), hasWinner(), isWinner(Disc) en toString().

## BoardTest.java

### Hoe is deze klasse getest?

Deze klasse is getest door te kijken hoeveel van Board hij test.

### Welke test techniek is toegepast?

Hij is getest aan de hand van de coverage die de EMMA plugin gaf.

### Welke testprogramma’s zijn ontwikkeld?

EMMA, maar deze is niet door ons ontwikkeld.

### Wat waren de verwachte resultaten en de echte resultaten?

We gingen voor een percentage boven de 80, en we kregen 85,1%.

### Hoeveel procent van iedere methode wordt gedekt door tests?

EMMA dekt alle methoden van deze klasse.

## ComputerPlayer.java

### Hoe is deze klasse getest?

Deze klasse is gebruikt voor het testen van veel strategieën, en hij is in week 5 bij de gamelogic ook uitgebreid getest, en tussen toen en nu is de klasse vrijwel niet aangepast.

### Welke test techniek is toegepast?

Visuele inspectie van de UI.

### Welke testprogramma’s zijn ontwikkeld?

De Strategies en de game zelf. Deze klasse is talloze keren succesvol gebruikt.

### Wat waren de verwachte resultaten en de echte resultaten?

Dat de ComputerPlayer een zet terug geeft, die van de Strategy komt, en dat deed hij ook.

### Hoeveel procent van iedere methode wordt gedekt door tests?

Er waren maar 2 methoden, getName en determineMove. getName doet niks anders dan de naam van de Strategy returnen, en determineMove is in de praktijk vaak genoeg getest.

## Disc.java

### Hoe is deze klasse getest?

Het is een enumklasse. Deze klasse is tijdens het ontwikkelen meer dan voldoende getest, tijdens het spelen van testgames. Hij is lang niet meer aangepast.

### Welke test techniek is toegepast?

Visuele inspectie van TUI.

### Welke testprogramma’s zijn ontwikkeld?

Geen.

### Wat waren de verwachte resultaten en de echte resultaten?

Verwacht: Laat R zien voor alle rode Discs, Y voor alle gele Discs en niks voor empty Discs. Dit gebeurde ook.

### Hoeveel procent van iedere methode wordt gedekt door tests?

100%.

## MinMaxStrategy.java

### Hoe is deze klasse getest?

Spelletjes spelen tegen deze Strategy.

### Welke test techniek is toegepast?

Visuele inspectie van de TUI.

### Welke testprogramma’s zijn ontwikkeld?

Geen.

### Wat waren de verwachte resultaten en de echte resultaten?

Dat hij zo slim zou zijn, dat hij menselijke spelers zou verslaan. We hebben de strategy op een server van iemand anders laten spelen tegen nog iemand anders, en onze strategy won.

### Hoeveel procent van iedere methode wordt gedekt door tests?

De getName methode is niet getest, maar deze wordt ook niet gebruikt. determineMove is geheel getest.

## NaiveStrategy.java

### Hoe is deze klasse getest?

Spelletjes spelen tegen deze Strategy.

### Welke test techniek is toegepast?

Visuele inspectie van de TUI.

### Welke testprogramma’s zijn ontwikkeld?

Geen.

### Wat waren de verwachte resultaten en de echte resultaten?

Dat hij random zetten zou doen, en dat deed hij ook.

### Hoeveel procent van iedere methode wordt gedekt door tests?

De getName methode is niet getest, maar deze wordt ook niet gebruikt. determineMove is geheel getest.

## SmartStrategy.java

### Hoe is deze klasse getest?

Spelletjes spelen tegen deze Strategy.

### Welke test techniek is toegepast?

Visuele inspectie van de TUI.

### Welke testprogramma’s zijn ontwikkeld?

Geen.

### Wat waren de verwachte resultaten en de echte resultaten?

Dat hij redelijk slimme zetten zou doen, en dat deed hij ook.

### Hoeveel procent van iedere methode wordt gedekt door tests?

De getName methode is niet getest, maar deze wordt ook niet gebruikt. determineMove is geheel getest.

## Strategy.java

### Hoe is deze klasse getest?

Het is een interface. Deze klasse is niet getest.

# Metrics report

## Program size (lines of code)

Onze Method Lines of Code staat of 2201, en onze Total Lines of Code is 3017. Deze nummers zeggen verder niet zoveel.

## Cyclomatic Complexity

Onze Cyclomatic Complexity is volgens Metrics in de klassen Client, ClientHandler en ClientTUI te hoog. Dat heeft te maken met de run/readInput methoden, die de commando’s tussen de Client en de Server accepteren. Wij konden dit niet echt netter neerzetten, dan dat het nu staat. Board’s hasDiagonal methode heeft ook een te hoge Cyclomatic Complexity. Dat heeft ermee te maken, dat als we in 2 methoden gaan checken of er een diagonal is, het 2x zo lang duurt om de berekening uit te voeren. De code ziet er zo misschien minder mooi uit, maar wij vonden dat hier dubbele efficiëntie belangrijker was, dan mooie code. Verder heeft de testEmptyRow methode van BoardTest een te hoge complexity. Als we deze test zouden opsplitsen in aparte methoden, zou het onoverzichtelijk worden, daarom hebben we ervoor gekozen, om dit zo te laten.

## Weighted methods per class

Metrics geeft aan, dat we op dit gebied niets fout doen. Dat betekent dat de som van de complexiteit van onze methoden in de klassen niet te hoog is. De klassen met de meeste weighted methods zijn ClientHandler en Client.

## Lack of cohesion in methods

Met een gemiddelde van 0,317 valt dit ook mee. Dit betekent dat we veel methoden in dezelfde klassen hebben, die elkaar aanroepen.

# Reflectie op de planning

## Hoe was je planning beïnvloed door jouw ervaringen met planning en tijdschrijven in week 4 van de module?

Op geen enkele manier.

## In hoeverre kwam je planning overeen met de werkelijkheid?

We zijn begonnen met de week 5 gamelogic overzetten naar Connect4. Daarna zijn we bezig gegaan met het implementeren van het week 6 Model-View-Controller model. Vervolgens zijn we verder gegaan met het implementeren van het protocol aan de hand van de multi-client chat van week 7. Met dat laatste zijn we ook het langst bezig geweest. We hebben veel te weinig tijd ingepland voor het protocol. Gelukkig hadden we niet heel veel uren ingepland, dus kon er heel makkelijk geschoven worden.

## Welke tegenmaatregelen heb je genomen om afwijking van de originele planning te compenseren? Wat was de impact hiervan op de kwaliteit van ons project?

Meer tijd besteden aan het protocol en extra features toevoegen. Hierdoor ging de kwaliteit van ons project omhoog.

## Wat heb je geleerd van deze ervaring voor je volgende (project)planning?

Als je voorloopt op de planning, heb je geen stress, en komt alles op tijd af. Ook heb je dan nog voldoende tijd om extra features toe te voegen.

## Twee do’s en don’t’s voor studenten die volgend jaar dit project gaan doen

Do: Begin op tijd. Wij zijn 2 weken voor de deadline serieus begonnen, en hebben iedere weekdag van 10 tot 5 besteed aan het werken aan het project. Ervoor hadden we al een klein opzetje met klassen die mogelijk nuttig konden zijn. Verder: Spreek een plek af, waar je iedere dag zit. Wij wonen 30 kilometer van de universiteit af in hetzelfde dorp. De ene dag gaan we naar die persoon’s huis, de andere dag naar de andere persoon’s huis. Dit scheelt reistijd en is duidelijk voor ons allebei.

Don’t: Niet op tijd beginnen. Als je nog niet iets hebt 1 week voor de deadline: good luck. Verder: Niet tot laat ’s avonds doorwerken, en dan uitslapen, en dan weer tot laat doorwerken. Niet goed voor de motivatie en je bent meer tijd kwijt met slapen dan je denkt.

//TODO: dingen toevoegen over extra features