Verslag Connect4 project

Michael Koopman s1401335 en Sven Konings s1534130

Inleiding

In deze module hebben wij ervaring opgedaan in Java. In dit project hebben wij die kennis in de praktijk toegepast. Dit project bestond uit het maken van een Client en een Server voor het Connect4 spel, waarbij aan de hand van een protocol, spelletjes gespeeld kunnen worden met andere mensen van onze practicumgroep met een andere implementatie dan de onze. Dit verslag zal onder andere inzicht geven in ons design, hoe wij de klassen geïmplementeerd hebben, en hoe onze tests werken. Wij vermelden hier nog even, dat wij toestemming hebben van mevrouw Huisman, om niet aan de Checkstyle conventie te voldoen, aangezien 2 studentassistenten het niet aan de praat konden krijgen. Verder hebben we als extra (bonus) features, een chatfeature, een leaderboardfeature en een custom board size feature. Met onze MinMaxStrategy zijn wij 3^{e} geworden op het toernooi, dus helaas net buiten de prijzen.

Discussie van het algehele design

Klassediagrammen

Het volledige klassendiagram voor onze applicatie kunt u vinden in het bestand ClassDiagram.png, die in dezelfde map als dit bestand zit. Dit heeft als reden, dat de kwaliteit nogal laag wordt, als we hem moeten verkleinen in Word.

Uitleg per package

Client

Bevat de klassen Client, ClientTUI en ClientView. ClientView is een interface, die geimplementeerd wordt door ClientTUI, zodat, als we bijvoorbeeld een GUI willen toevoegen, dat heel gemakkelijk gaat. ClientTUI is de TUI die de Client te zien krijgt. Client is de engine van alles wat client-side moet gebeuren. Versturen en ontvangen van commando's gaan via Client.

Game

Bevat de klassen Board en Disc. Deze klassen zijn nodig om het bord bij te houden.

Game.strategy

Bevat de klassen ComputerPlayer, MinMaxStrategy, NaiveStrategy, SmartStrategy en Strategy. ComputerPlayer wordt gebruikt als AI-klasse. Het krijgt een strategie mee, en heeft een determineMove methode waarmee aan de hand van de strategie die hij heeft, een move kan doen. NaiveStrategy doet compleet random moves, SmartStrategy denkt 1 zet vooruit en MinMaxStrategy denkt meerdere zetten vooruit. Strategy is een interface die wordt geïmplementeerd door al deze strategieën.

Server

Bevat de klassen ClientHandler, LeaderboardPair, MessageUI, Server en ServerGUI. MessageUI is een interface geïmplementeerd door ServerGUI, zodat we eventueel ook makkelijk een TUI zouden kunnen maken voor de server. LeaderboardPair wordt gebruikt om het Leaderboard te sorteren door een paar van scores van spelers te vergelijken. Server wordt gebruikt om ClientHandlers aan te maken bij inkomende connecties en ClientHandler wordt gebruikt om de connectie te beheren en dus de commando's op te vangen.

Test

Bevat de klassen BoardTest, ClientTest, ClientTUITest en ServerTest. Deze klassen worden gebruikt om de bijbehorende klassen te testen.

Systematisch overzicht van welk van de eisen is geïmplementeerd in welke klassen

We zijn in deze tabel uitgegaan van de Functional Requirements of the Application zoals op pagina 18 van de handleiding is beschreven.

EIS	**GEÏMPLEMENTEERD IN:**
Server	
Als de server is gestart, moet er een poortnummer ingevoerd worden waarnaar de server zal luisteren.	ServerGUI. Bij het opstarten van de ServerGUI kun je een poort invoeren, en daarna kun je klikken op "Start listening". Na het klikken op "Start listening", kun je de poort niet meer veranderen (tenzij de poort al gebruikt wordt door een ander programma).
Als het poortnummer al gebruikt wordt, wordt een geschikte foutmelding gegeven, en kan een nieuw poortnummer ingevoerd worden.	ServerGUI. Als je een poortnummer invoert, die al in gebruik is, dan krijg je het bericht: "Error listening on port <port>, please select a different one", en worden het invoerveld voor de port en de start listening knop weer actief.
Een Server moet meerdere instanties van het spel ondersteunen, die tegelijkertijd door verschillende Clients gestuurd worden.	ClientHandler. 2 ClientHandlers krijgen hetzelfde bord, en die regelen de Game verder. Als er een move gecheckt wordt, is alleen de ClientHandler van degene die de move wil doen bezig.
De TUI zorgt ervoor alle communicatie naar System.out geschreven.	ServerGUI. Wij hebben onze implementatie gebaseerd op de Multi-Client chat uit week 7, en daarbij hebben we de ServerGUI klasse gebruikt, die daarbij zat. Hierdoor hebben wij geen TUI voor de Server. Alle berichten worden geschreven naar het berichtenvak in de GUI, en je kunt scrollen door dat vak. Hierdoor is de functionaliteit hetzelfde.
De Server respecteert het protocol zoals afgesproken met de tutorialgroep tijdens de projectsessie in week 7, oftewel de Server moet kunnen communiceren met alle andere Clients.	Features, Server en (vooral) ClientHandler. Er zijn constanten gedefinieerd in Server en Features, zoals die in het protocol staan, en die worden door zowel Server als ClientHandler continu gebruikt.
Client	
De Client moet een gebruiksvriendelijke TUI hebben, die verschillende opties aanbiedt aan de	ClientTUI en Client. Zodra er een Client object wordt gemaakt, wordt aan de TUI om een IP-

gebruiker, zoals het invoeren van een IP-adres en poort om een Game van de Server op te vragen.	adres en een poort gevraagd. Als er een verbinding kan worden opgezet, dan vraagt de Client de TUI om een naam. Tijdens het vragen van de naam kan ook aangegeven worden, dat je een ComputerPlayer met een Strategy voor je wilt laten spelen. Kan er geen verbinding gemaakt worden, dan wordt er opnieuw om een IP en een poort gevraagd. Verder werkt onze Tutorialgroep met een lobby en invite systeem om een Game op te vragen.
De Client moet HumanPlayers ondersteunen, en ComputerPlayers met een beetje kunstmatige intelligentie.	ClientTUI en Client. Bij het opgeven van je naam, kun je kiezen of je zelf wilt spelen, of de computer voor je wilt laten spelen. We hebben onze eigen versie van het MinMax-algoritme geïmplementeerd, aan de hand van een uitleg over hoe het algoritme werkt, die we vonden op het internet.
De bedenktijd van de ComputerPlayer moet een parameter zijn, die veranderd kan worden in de ClientTUI.	ClientTUI en Client. Door het commando DIFFICULTY te gebruiken kun je de diepte (het aantal zetten dat het algoritme vooruit denkt) van de MinMaxStrategy aanpassen. Bij de overige Strategies is de bedenktijd niet aanpasbaar omdat deze niet beter of slechter kunnen gaan zoeken.
De Client moet een hintfunctionaliteit hebben, die een HumanPlayer een mogelijke move laat zien, voorgesteld door een ComputerPlayer. De move moet alleen voorgesteld worden, en niet automatisch gedaan worden.	ClientTUI, Client, ComputerPlayer en MinMaxStrategy. Als het HINT commando worst ingevoerd in de ClientTUI, wordt dat aan de Client doorgegeven, en de Client laat een nieuwe ComputerPlayer met een MinMaxStrategy een move bepalen op een kopie van het huidige Board. Deze wordt vervolgens voorgesteld aan de HumanPlayer, maar niet daadwerkelijk gedaan.
Nadat het spel is afgelopen, moet de speler een nieuw spel kunnen beginnen.	Server, ClientHandler en Client. Als het spel is afgelopen, wordt er een game end pakket gestuurd, en gaat de Client weer naar de lobby. Vanuit de lobby kan hij dan weer iemand inviten of een invite van iemand anders accepteren.
Als een speler het spel afsluit voordat het is afgelopen, de UI sluit, of de client crasht, moeten de andere spelers daarover geïnformeerd worden. In dit geval moeten de andere spelers weer kunnen registreren met de Server om een weer te kunnen spelen.	Client en ClientHandler. Als de verbinding wordt verloren sluiten de client en de clientHandler allebei af. Dus op het moment dat de server wordt gesloten sluit de client af en als de client wordt gesloten of het QUIT command wordt ingevoerd sluit de clientHandler ook af. Als een clientHandler afsluit meldt hij dit aan de serveren stuurt hij een game end of een nieuwe lobby package afhankelijk van of de client in game was.
Een Server kan op ieder moment disconnecten. De Client moet hier op een goede manier mee omgaan.	Client. De Client sluit af op het moment dat de server disconnect.

De Client respecteert het protocol zoals afgesproken met de tutorialgroep tijdens de projectsessie in week 7, oftewel de Client moet kunnen communiceren met alle andere Clients.	Client en Features. Er zijn constanten gedefinieerd in Client en Features, zoals die in het protocol staan, en die worden door Client continu gebruikt.
Global	
De Client en de Server moeten altijd in dezelfde game state zijn.	Als de game state van de server verandert, stuurt de server een MOVE OK pakket met player nummers. Daarmee kunnen de Clients bijhouden wie er aan de beurt is, en de move op het Board doen.
De game mag niet gekopieerd zijn van het internet.	Alle klassen zijn door onszelf geschreven, gebaseerd op de Multi-Client chat van week 7, het MVC model van week 6 en de gamelogic van week 5.

Het gebruik van het Observer en het Model-View-Controller patroon.

Observer

Bij de client was het niet zinvol een klasse te observen. Veranderingen in het board opvangen is niet zinvol, aangezien die na iedere move op de console geprint wordt. Het observen van een ComputerPlayer is ook niet nuttig, want die maakt de move op het bord, en nergens hoeven we te weten of het bord is aangepast. Omdat het niet zinvol, en zelfs omslachtig was voor onze implementatie om een observer pattern voor Client te implementeren, hebben we ervoor gekozen om dit niet te doen.

Bij de server wordt het board geobserveerd door de ServerGUI. Door de directe connectie tussen Server en de ServerGUI was het niet nodig om server te observeren aangezien de server het direct aan de view kon doorgeven. Maar het board werd niet bijgehouden door de server, alleen door de ClientHandlers. Dus door het board te observeren kon de ServerGUI makkelijk weergeven of er een board aangepast was.

Model-View-Controller

Bij de client is het model de klassen Board en Client. Board bevat alle logica voor het spel, en de client bevat de logica om invites te versturen en ontvangen, een move te vragen aan een menselijke speler of een computerspeler. De controller is Client. Deze verzorgt alle communicatie tussen de client en de server en het verwerken van commando's. De view is ClientTUI, deze laat alle gebeurtenissen zien aan de speler en stuurt commando's door naar Client.

Bij de server is het model de klassen Board en Server, board bevat alle logica voor de Connect4 game en server bevat de logica om invites, ClientHandlers en het leaderboard bij te houden. De controllers zijn de ClientHandlers en deels de server. De server zorgt ervoor dat ClientHandlers worden aangemaakt. De ClientHandlers vangen de commando's van de clients op en roepen de bijbehorende methodes aan.

Opslag van data en communicatieprotocollen

Ons leaderboard wordt opgeslagen als tekstbestand in gesorteerde volgorde met 1 speler per regel. De regel bestaat uit <naam> <wins> <losses> <games player>. Verder slaan wij lokaal, behalve de java-bestanden natuurlijk, geen data op. Informatie over de communicatie tussen de Client en de Server kunt u vinden in het protocol van onze tutorialgroep, die staat in doc/protocol.

Discussie per klasse

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

Client.java

Rol van de klasse in het systeem

Client.java regelt alles client-side, van het opvangen van servercommando's, tot het sturen van commando's en het bijhouden van het bord.

De verantwoordelijkheden van deze klasse

Deze klasse regelt het opvangen en verwerken van de volgende servercommando's:

- ACCEPT_CONNECT: Client geeft door aan de TUI dat er een succesvolle verbinding is opgezet met de Server, en welke features de Server heeft. Ook wordt de boolean isConnected op true gezet.
- LOBBY: Client geeft de spelers in de lobby door aan ClientTUI.
- INVITE: Client slaat een nieuwe invite op, en geeft aan de TUI door dat de Client geinvite is.
- DECLINE_INVITE: Client verwijdert de invite van de persoon, en geeft aan de TUI door dat de invite gedeclined is.
- GAME_START: Zet myNumber op nummer van de speler, zoals dat op de Server bekend is, en geeft aan de TUI door, dat er een nieuwe game is begonnen tussen jou en de andere speler.
- GAME_END: Zet het board op null, en geeft aan de TUI door, wie heeft gewonnen (of dat het gelijkspel was). Mocht de Server dit niet hebben doorgegeven, dan wordt doorgegeven dat de game is geëindigd vanwege de reden die de Server meegaf.
- REQUEST_MOVE: Checkt of de speler een HumanPlayer is, en zo ja, geeft door aan de TUI dat de speler een move moet doen, en zo niet, laat de ComputerPlayer een move maken.
- MOVE_OK: Probeert de move te doen op het board. Mocht dit niet lukken, dan wordt het bord opgevraagd.
- ERROR: Geeft door aan de TUI dat er een error is gebeurd.
- BOARD: Verandert het Board naar het bord, dat de Server net doorgaf.
- CHAT: Geeft een chatbericht door aan de TUI.
- LEADERBOARD: Zorgt ervoor dat het leaderboard in een goede vorm wordt doorgegeven aan de TUI.
- PONG: Geeft door aan de TUI dat de Server een PONG bericht stuurde.
- Alle andere commando's: Geef door aan de Server: Unknown command.

Verder regelt deze klasse het verwerken van de volgende Clientcommando's:

- HELP: Stuurt door aan de view welke commando's er op dit moment gebruikt kunnen worden, afhankelijk van of de Client ingame is.
- MOVE: Checkt of er om een move gevraagd is. Zo ja, dan stuurt hij de move door naar de Server, zo nee, dan geeft hij een foutmelding.
- INVITE: Voegt de invite toe aan de lijst met invites. Als er een custom board size mee is gegeven, dan wordt deze ook opgeslagen.

- ACCEPT: Stuurt de Server een pakket waarin staat dat de invite van de speler geaccepteerd is.
- DECLINE: Stuurt de Server een pakket waarin staat dat de invite van de speler gedeclined is.
- HINT: Als de Client in-game is, regel dat er een hint wordt doorgegeven aan de Client, anders, geef een foutmelding.
- DIFFICULTY: Verandert de moeilijkheid van de ComputerPlayer (het aantal zetten dat deze vooruit denkt).

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

ClientTUI voor het vragen om input en om berichten aan door te geven, Board om het bord bij te houden en ComputerPlayer voor het maken van hints, en eventueel zetten te doen.

Voorzorgsmaatregelen om de precondities in the server te vervullen

Waar nuttig zijn checks geïmplementeerd, zoals het checken van de lengte van de commando's die je stuurt.

ClientHandler.java

Rol van de klasse in het systeem

ClientHandler.java regelt het opvangen van alle commando's die de client stuurt en de bijbehorende methodes uitvoeren. Ook houdt de ClientHandler de game bij samen met de ClientHandler van de tegenstander.

De verantwoordelijkheden van deze klasse

Deze klasse is verantwoordelijk voor het aanmelden van een client met behulp van het connect command. Dus deze klasse is ook verantwoordelijk voor het controleren van de naam en de features van de client en het opvangen van alle andere commando's van de client en het bijbehorende antwoord geven. Deze klasse is ook verantwoordelijk voor het afmelden van de client als de verbinding verbroken wordt, voor het maken en verwijderen van invites (maar niet voor het bijhouden van de invites) en voor het bijhouden van het leaderboard. Verder is deze klasse nog verantwoordelijk voor het starten van een game, het bijhouden van het bord, controleren of de moves wel kunnen en de beurt van de volgende speler doorgeven of aangeven dat er iemand heeft gewonnen.

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

Deze klasse gebruikt server voor het uitvoeren van de meeste commando's en het communiceren met andere ClientHandlers. Verder wordt LeaderboardPair gebruikt om een gesorteerd leaderboard bij te houden.

Voorzorgsmaatregelen om de precondities in the server te vervullen

Bij elk command van de client checks ingebouwd om te controleren of deze aan het protocol voldoen en ook of deze aan de precondities van de server voldoen

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
Zorgen dat er geen invalide of lege commando's worden gestuurd.

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 wordt gemaakt en geeft de berichten door aan de GUI.

Voorzorgsmaatregelen om de precondities in the server te vervullen
Als de ingevoerde poort niet bestaat, 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 wordt gebruikt om de move van de ComputerPlayer te bepalen.

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

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

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. De coverage resultaten van de systeemtesten van de Client en de Server zijn te vinden in doc/systemtests. Het handigst is, om een van beide te openen, en dan het .sessions.html bestand te openen. Dit geeft een makkelijk doorklikmenu.

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 een systeemtest gedaan, waarbij we verschillende commando's probeerden.

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. Verder wilden we, dat de Client tijdens de systeemtest deed, wat we wilden, dat hij deed.

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%. De systeemtest leverde een coverage van 57%.

ClientHandler.java

Hoe is deze klasse getest?
Deze klasse had vooral private methodes waardoor we hier slecht en individuele test voor konden schrijven. We hebben echter wel uitgebreid getest met een systeemtest door zoveel mogelijk commando's aan te roepen, ook met verkeerde argumenten zodat de ClientHandler ook de errors stuurde. Bij deze tests hadden wij een coverage van 77%, ook werkte de ClientHandler correct.

Welke test techniek is toegepast?
Wij hebben systeemtest toegepast met visuele inspecties van de UI.

Wat waren de verwachte resultaten en de echte resultaten?
De verwachte resultaten kwamen overheen met de echte resultaten. Wanneer we een fout command invoerde kregen we een error terug en wanneer het command goed werd ingevoerd deed de ClientHandler ook wat die moest doen.

Hoeveel procent van iedere methode wordt gedekt door tests?
Zie de coverage in /doc/systemtests/Server/4oprij/src/server/ClientHandler.html

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. Daarnaast is een systeemtest uitgevoerd.

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. In de systeemtest voor de Client kregen we een coverage van 66%.

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

Hoe is deze klasse getest?
Er is een ServerTest.java gebruikt, die een aantal methoden van Server.java aanroept, en daarnaast hebben we deze ook in een systeemtest getest.

Welke test techniek is toegepast?
Een systeemtest en het toevoegen van een main methode die een aantal methoden van de klasse aanroept.

Welke testprogramma's zijn ontwikkeld?
ServerTest.java

Wat waren de verwachte resultaten en de echte resultaten?
Dat de Server op een juiste manier zou reageren als we zijn methoden aanriepen, en dat gebeurde ook. Verder wilden we dat de coverage hoger was dan 50% op zowel de individuele test, als de systeemtest. Dat gebeurde ook.

Hoeveel procent van iedere methode wordt gedekt door tests?

EMMA geeft bij ServerTest.java een coverage van 51,8%. In een systeemtest kregen we een coverage van 72%. Tijdens deze systeemtest gebeurde niets, dat we niet wilden dat gebeurde.

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. Verder hebben we ook een testklasse ontwikkeld voor Server, die tegelijkertijd deze klasse test.

Welke test techniek is toegepast?

Visuele inspectie van de UI en een systeemtest.

Welke testprogramma's zijn ontwikkeld?

ServerTest.java

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. Verder wilden we, dat de Server juist reageerden als we commando's stuurden, en dat gebeurde ook.

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. Verder hebben we in een systeemtest een coverage bereikt van 99%.

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(). In de systeemtest van de Client kregen we een coverage van 71%, in de systeemtest van de Server kregen we een coverage van 59%.

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?

We hebben een vast tijdschema aangehouden van 10 tot 5 en van maandag tot vrijdag. We hebben wel gelet of we goed op schema zaten en omdat dit zo was hebben we dit tijdschema niet hoeven aan te passen.

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 en iets te veel voor het schrijven van computerspelers. 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: test het protocol met mede studenten. In ons protocol waren wat verwarringen doordat er op één plek stond dat er een dubbele \n moest worden gebruikt voor de package end, terwijl er op de andere plek stond dat er een enkele \n moest worden gebruikt. Ook was er niet afgesproken welke nummers er werden doorgegeven voor de spelers, waardoor sommige 0 en 1 hadden en sommige 1 en 2. Door meer met elkaar te testen hadden deze fouten veel eerder opgelost kunnen worden.

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.