Final Report Group 22 End project Captain’s Mistress 28-1-2015

Peter Beers s1201549

Stephan Braams s1359185

Table of Contents

[Discussion of the Overall Design 3](#_Toc408928729)

[Server Class Diagram and explanation 4](#_Toc408928730)

[Client Class Diagram and explanation 6](#_Toc408928731)

[Model implementation for FourUp Class diagram and explanation 7](#_Toc408928732)

[Discussion per Class 8](#_Toc408928733)

[Server 8](#_Toc408928734)

[ServerController 8](#_Toc408928735)

[GUI 8](#_Toc408928736)

[GameController 8](#_Toc408928737)

[Model 9](#_Toc408928738)

[ConnectionHandler 9](#_Toc408928739)

[ServerSocketListener 9](#_Toc408928740)

[Error 9](#_Toc408928741)

[Client 11](#_Toc408928742)

[Classname 11](#_Toc408928743)

[Model 12](#_Toc408928744)

[Game 12](#_Toc408928745)

[Board 12](#_Toc408928746)

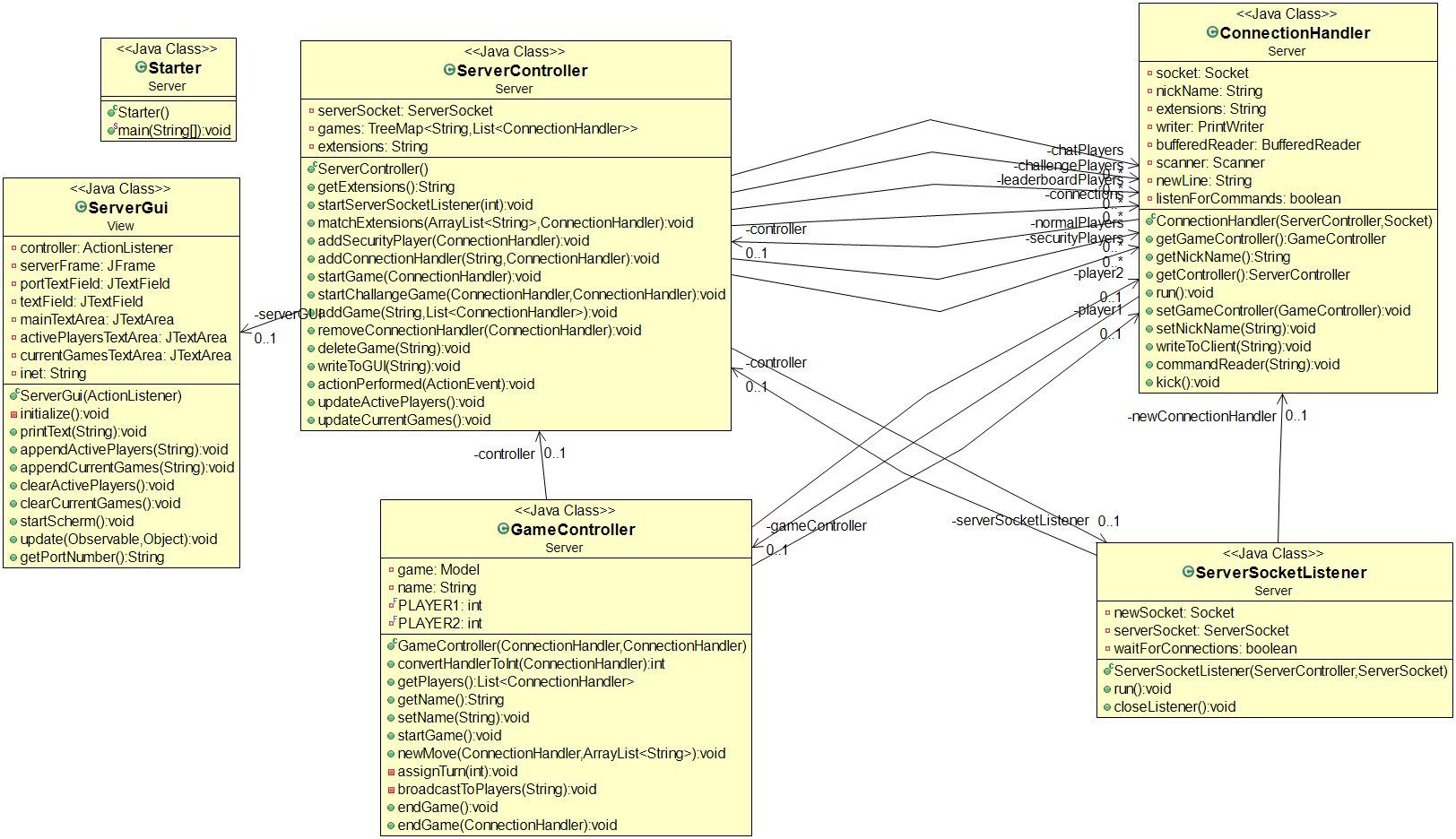
[Field 12](#_Toc408928747)

[Security 13](#_Toc408928748)

[Process 14](#_Toc408928749)

# Discussion of the Overall Design

## Server Class Diagram and explanation



The Model Class models the FourUp game and is responsible for assigning turns, checking if moves are correct and notifying the GameController if the game has ended. As the name suggests it fulfills the Model role in the Model-View-Controller pattern. It was decided to use an interface Class for the model for two reasons, on the one hand to add a level of abstraction to the system and create the hypothetical possibility to swap the game type and more important to the developers, ensure that different parts of the system can more easily built independent from each other.

The View is implemented through the GUI Class. We decided to let the ServerController handle the logic for the GUI through action handlers to ensure a degree of separation between Controller and View. As with the Model we use an interface Class to streamline our development process and to add interchangeability to the application.

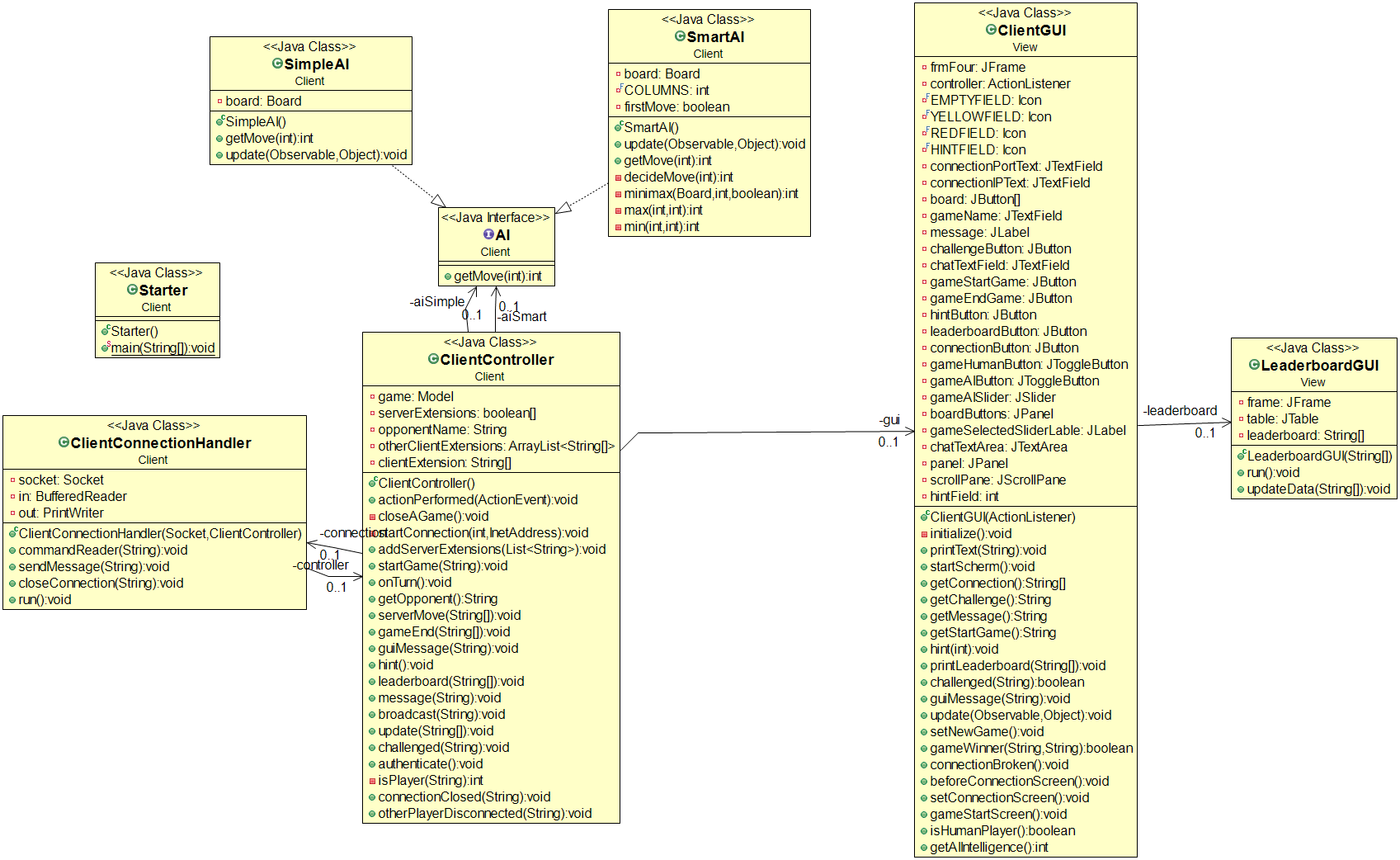
The main Class in the Server architecture is the ServerController Class, taking the controller role in the MVC pattern. It initializes all other Classes and hence facilitates the creation of games, houses the logic for the GUI, creates the ServerSocketListener for the serversocket and will implement the functionalities of any of the facultative extensions.

For exeption handling we use an independent error class.

The GUI and ServerController Classes are coupled in the observer pattern, with the ServerController being the sole observer and the GUI being the observable. The Model and GameController Classes fit a similar pattern with the Model being the observable and GameController being the observer.

The ConnectionHandler Class plays an important part in the implementation of the AMULET tcp protocol by receiving the commands and sending them to the other relative parts of the system. We decide to leave as much of the logic regarding the commands in the respective Classes instead of building a massive switch which would call methods on the other objects. In practice, this means that the ConnectionHandler sends the commands through to the ServerController and GameController with which it doesn’t do anything itself.

## Client Class Diagram and explanation



The Client class diagram exists out of six classes and two interfaces. The Model interface is here the exact same as given in the Model Class diagram, the same goes for the the AI interface which is used for the multiple types of intelligent AI player that can play a game. The ClientController is the starter class of the client who starts the Model, GUI, AI and ClientConnectionHandler. The ClientConnectionHandler is used to start communication with a server and sends and receives messages.

The AI interface is an important part of the implementation of the artificial intelligence requirement the hint requirement is also dependent on the AI. The human player requirement is implemented by the GUI.

The SimpleAI class is an implementation of the AI interface and has as only functionality to give a legal random on the board. This AI class isn’t sophisticated.

The SmartAI class on the other hand is a smart implementation of the AI interface which uses minimax and a set of heuristic values to determine the best move to make.

The GUI and ClientController classes are coupled in the observer pattern, with the ClientController being the sole observer and the GUI being the observable. The Model and GUI classes fit a similar pattern with the Model being the observable and GUI being the observer.

The communication protocol is used and implemented by the ClientConnectionHandler. The other data including the model with it’s board and the AI are kept only at runtime and remade when the game is started again.

## Model implementation for FourUp Class diagram and explanation

## C:\Users\peter\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Model.png

For the specific rules and dimensions of the FourUp game the model is implemented by a structure of three Classes. The Game Class takes the similar role to controller and the Board Class to Model.

The Board consists of 42 Field classes who keep track of their value in an Observable-Observer pattern. The Board Class handles the checking of the legality of a move and ascertains if there is a winning player and if the game has ended.

The Game Class initializes the Board, assigns the turns, and cleanly ends the game if the game has ended.

We also chose here for the use of an Model interface which made it easier to split the functionality from the Server & Client from the Model.

## Systematic overview of which parts of the requirements are implemented in which Classes

|  |  |
| --- | --- |
| Server requirements |  |
| 1. When the server is started, a port number should be entered that the server will listen to. |  |
| 1. If the port number already is in use, an appropriate error message is returned, and a new port number can be entered. |  |
| 1. A server should be able to support multiple instances of the game that are played simultaneously by different clients. |  |
| 1. The TUI ensures that all communication messages are written to System.out |  |
| 1. The server should respect the protocol as defined for the tutorial group during the project session in Week 7, i.e., the server should be able to communicate with all other clients from the tutorial group. |  |

|  |  |
| --- | --- |
| Client requirements |  |
| 1. The client should have a user friendly TUI, which provides several options to the user to request a game at the server | ClientGUI |
| 1. The client should support human players, and computer players with (some) artificially intelligent behaviour. | ClientGUI, ClientController, AI, SimpleAI, SmartAI |
| 1. The thinking time of the computer player should be a parameter that can be changed via the client TUI. | ClientGUI, ClientController |
| 1. The client provides a hint functionality. This shows a humand player a possible move, as indicated by the computer player. The move may only be proposed, the human player should have the possibility to decide whether to play this move, or make a different one. | SimpleAI, ClientGUI, ClientController |
| 1. After the game is finished, the player should be able to start a new game. | ClientGUI |
| 1. If a player quits the game before it has finished, closes the UI, or the client crashes, the other players should be informed. In this case, the other players should be allowed to register again with the server to play the game. | ClientConnectionHandler, ClientGUI, ClientController |
| 1. A server might at all times disconnect. The clients should react to this in a decent way, closing all open connections etc. | ClientConnectionHandler, ClientGUI, ClientController |
| 1. The client should respect the protocol as defined for the tutorial group during the project session in Week 7, i.e., the client should be able to communicate with all other servers from the turorial group. | ClientConnectionHandler, ClientController |

## Formats for data storage and communication protocols

All the data contained in the system is only kept at runtime. Next to that there was no usage of a database or a file to which data could be written to. So the whole system would start blank the next time it started and is the same every time it is started with the same values. During an earlier meeting in Week 7 the protocol was determined and the protocol that is used in this system is the latest version of the protocol of the Bit 2 group that can be found online on blackboard.

# Discussion per Class

## Server

### ServerController

#### Role

The main Class in the Server architecture is the ServerController Class, taking the controller role in the MVC pattern.

#### Responsibilities

This Class initializes all other Classes and hence facilitates the creation of games, houses the logic for the GUI, creates the ServerSocketListener for the serversocket and will implement the functionalities of any of the facultative extensions.

#### Useage by other classes

This Class is directly used by all other classes in the server except for Model.

### ServerGui

#### Role

The View is implemented through the ServerGui Class. We decided to let the ServerController handle the logic for the ServerGui through it being an actionlistener to ensure a degree of separation between Controller and View.

#### Responsibilities

Housing the Graphical User Interface and passing through information to the ServerController

#### Useage by other classes

This Class is only used by the ServerController to get cues for what the user wants and to print information to

### GameController

#### Role

The GameController Class governs a single game between two clients.

#### Responsibilities

The GameController Class is responsible for game related communication between two clients, ending the game and sending relevant information to the ServerController

#### Useage by other classes

This class is used by the ServerController to start games and receive information for the leaderboard and client statuses. The ConnectionHandler sends all its game related commands to this class and the model uses this class to notify if there is a winner in a game

### ConnectionHandler

#### Role

The ConnectionHandler Class plays an important part in the implementation of the AMULET tcp protocol by receiving the commands and sending them to the other relative parts of the system. We decide to leave as much of the logic regarding the commands in the respective Classes instead of building a massive switch which would call methods on the other objects.

#### Responsibilities

The ConnectionHandler handles some of the logic for client to client-user communication and sends the commands through to the ServerController and GameController with which it doesn’t do anything itself.

#### Useage by other classes

This Class is used by the GameController and ServerController to send information to the client-user.

### ServerSocketListener

#### Role

The ServerSocketListener is one of the classes to be created first and handles the TCP communication.

#### Responsibilities

The ServerSocketListener immeaditly starts waiting for new connections and when they arise immediately assigns them a ConnectionHandler. The ServerSocketListener is also responsible for appropriately closing of all connections.

#### Useage by other classes

The ServerSocketListener is used by the ServerController to close all connections.

### Starter

#### Role

The starter creates a new servercontroller. It has the public static void method.

### Error

#### Role

The Error class is used for centralized exception and error handling.

#### Responsibilities

The Error class is responsible for catching excepetions and errors and sending error messages to the client or Server user.

#### Useage by other classes

This Class is used by all classes except for the Model class.

## Client

### ClientController

#### Role

The ClientController class is used to control and connect the functionality of all classes

#### Responsibilities

Makes the Model, AI, GUI and ClientConnectionHandler and handles a huge part of the communication between these classes.

#### Useage by other classes

ClientController is used by the GUI to pass on messages to the ClientConnectionHandler. The ClientController is also used by the ClientConnectionHandler to pass on messages to the GUI and Model of the Client.

### ClientConnectionHandler

#### Role

Opens a connection with a server and keeps track of the communication between the server and the client.

#### Responsibilities

Makes the connection is kept up between the server and client and handles problems accordingly. Also is responsible for receiving a command from the server and passing it on to the ClientController

#### Useage by other classes

The ClientConnectionHandler is only known by the ClientController and the usage of this class exists out of passing messages through to or from the ClientController.

#### Special cases & Precautions

### AI

#### Role

Offers an interface for multiple AI implementations and is used for the hint function

#### Responsibilities

Gives a possible move back to the ClientController which is valid and gives a gateway to using different kinds of AI’s.

#### Useage by other classes

Is used by the ClientController to determine possible moves

### SimpleAI

#### Role

Offers a simple implementation of AI and is used for the hint function

#### Responsibilities

Gives a possible move back to the ClientController which is valid and gives a gateway to using different kinds of AI’s.

#### Useage by other classes

Is used by the ClientController to determine possible moves and also the hint function of the ClientController

### SmartAI

#### Role

Offers a smart implementation of AI that uses minimax to determine a move.

#### Responsibilities

Gives a possible move back to the ClientController when a SmartAI player has been selected on the ClientGui

#### Useage by other classes

Is used by the ClientController to determine possible moves and play a game.

### ClientGUI

#### Role

Starts a graphical user interface of the Client on which a game can be played

#### Responsibilities

Has the responsibility of sending all the information the user fills into the gui to pass this onto the controller when it is wanted. Also has functionality for making a connection and starting a game with a human or artificial intelligence player.

#### Useage by other classes

Is used by the controller to print messages from the server to.

### LeaderboardGUI

#### Role

Offers a leaderboard class on which the leaderboard given by the server can be shown.

#### Responsibilities

Is responsible for a visual representation of the leaderboard.

#### Useage by other classes

Is used by the ClientController to print the leaderboard on.

## Model

### Game

#### Role

Keeps track of the game rules and creates a board for the game to be played on.

#### Responsibilities

The Game class is responsible for creating the Board and for updating the board. Also keeps track if there is a winner in the game.

#### Useage by other classes

Is only used through the Model interface.

### Board

#### Role

Keeps track of a board filled with fields

#### Responsibilities

Creates and updates a board and fills it with empty fields.

#### Useage by other classes

Is used by the Game class to make changes to the board and check the status

### Field

#### Role

Keeping track of the status of a field on the board

#### Responsibilities

Keeping track of the status on the board and updating it if called for.

#### Useage by other classes

Used by the Board class to keep track of a field

### Model

#### Role

The Model Class models the FourUp game as the name suggests it fulfills the Model role in the Model-View-Controller pattern.

#### Responsibilities

The Model Class is responsible for assigning turns, checking if moves are correct and notifying the GameController if the game has ended.

#### Useage by other classes

The Model class is only used by the GameController class to check if moves are correct and to be notified if there is a natural end to the game

# Starting and Ending a game in the Server

Here I will describe all the actions that take place from starting the application, 2 players joining and the game ending in one of three ways.

1. The Server/Server/Starter.java class is run. This Class creates a ServerController.
2. In the Constructor of the ServerController Class a new ServerGui is created. The Instance variables of the ServerController class are initialized.
3. The View/ServerGui.java Class is initialized by the ServerController. It initializes its graphical elements and displays the current ip-address of the system.

[SERVERUSER] The user enters a port number in the JTextField next to the label which says “Port: “ or does not change the default value 2220 and pushes the “Start” button.

1. The ServerController acts as an ActionListener and receives and ActionPerformed ActionEvent. It checks if the port number is indeed a number and if it is between 1025 and 65536. Then it calls the startServerSocketListener() method.
2. startServerSocketListener creates a new ServerSocket and a new ServerSocketListener. It also starts the ServerSocketListener Thread.
3. The ServerSocketListener initialized its instance variables and assigns the flag for the run method to true.

[CLIENT1] A client tries to make a TCP connection on the specified port of this server.

1. The ServerSocketListener run method tries to accept the connection and create a socket for it. It starts the ConnectionHandler thread. The loop has ended, and it starts waiting for a new connection.
2. The ConnectionHandler constructor initializes its instance variables and sets its flag for the run method to true. It tries to create a BufferedReader and a PrintWriter. The Client is waiting for the AMULET EXTENSIONS command, so the ConnectionHandler sends it.

[CLIENT1] Sends back the AMULET EXTENSIONS response

1. The run method in ConnectionHandler uses the readLine() method to wait for new lines send over the connection. After the bufferedReader detects a new line the line is passed to the commandReader.
2. The commandReader uses a scanner to separate the AMULET command and it arguments and stores them in appropriate local variables.
3. A switch handles the different AMULET client responses. If there is any deviation from the established patterns an error is thrown. For the EXTENSIONS case the ConnectionHandler calls the matchExtensions method in ServerController
4. The matchExtensions method puts the ConnectionHandler reference in different Lists so they can be used to support extensions. If there is any deviation from the established patterns an error is thrown. The system now waits for an AMULET JOINREQ command.

[CLIENT1] Sends JOINREQ <<player1name>>

1. As before this line finds its way to the commandReader. For the JOINREQ case in the switch the addConnectionHandler method is called in ServerController. Alse the “Player\_” prefix gets checked and if it isn’t part of the argument after JOINREQ the player is sorted into the securityPlayers list through the ServerController addSecurityPlayer method.
2. The addConnectionHandler method adds the nickname of client1 and the reference to its ConnectionHandler object in the connections map. To update the ActiveClients list in the gui the updateActivePlayers method is called. Unless this is a Challenge player a game should be automatically started if there is at least one other client logged in to the server. [meanwhile for this explanation we assume another player logs into the server, and we’ll call him CLIENT2]
   1. updateActivePlayers updateActivePlayers updates the ActivePlayers list in the ServerGui. It retrieves the keySet from the connectionsmap and it clears the TextArea. It loops over the entries in the keySet and appends each of them to the TextArea. It also lists the Players in the mainTextArea.
   2. startGame selects a random player from the ActivePlayers map. It creates a new GameController with the two selected Clients as parameters. It sets the gameController attributes for the Clients. It calls addGame to handle putting the game into the CurrentGames map and removing them from the ActivePlayers map.
      1. GameController initializes its instance variables and calls the startGame method.
      2. addGame addGame removes the players of a new game from the ActivePlayers map and puts a new entry into the games map using the removeConnectionHandler method which updates the ServerGui.
3. GameController.java/startGame() creates a new Game object. It sends the AMULET GAME commands and the TURN command to the player that is first in its constructor using the ConnectionHandler writeToClient method.
   1. writeToClient uses the native printWriter of the ConnectionHandler object to send a line to its client. It also displays the message on the ServerGui
4. The GameController now waits for the AMULET MOVE command. There could be added a check to disallow a too long waiting time.

[CLIENT1] Sends MOVE 3

1. The MOVE command finds its way into the commandReader of the ClientHandler of CLIENT1. The case for MOVE checks if gameController is set and calls the GameController newMove method for the move. If no GameController reference is found an error is thrown.
2. The newMove method throws an Error which should be handled in GameController. It pulls the integer of the column of the move from the AMULET arguments list it is handed by the ConnectionHandler that calls the method and parses it to a String. It uses the convertHandlertoInt method to convert the ConnectionHandler reference into an integer representation of the player. This is needed because the Game Model uses integers 1 & 2 to keep track of players and not ConnectionHandler references.
   1. convertHandlerToInt knows which player is player1 because the ConnectionHandler reference is stored in an instance variable called “player1”.
3. First the newMove Checks if the player is on turn using the game.onTurn method. If the player is not on turn an error is thrown and the endgame method is called. If it is a legal move the game continues until either
   1. There is a winner
   2. There is a draw
   3. The game is ended (either because a player is kicked or a player leaves)

# Reflection on planning

We planned the project dynamically with the use of Trello. We used a very structural approach. At the beginning of the project period we categorized and mapped all the requirements and plotted any deadlines on a timeline. Subsequently we assigned tasks and set deadlines per task. This was all possible in trello.

We used this detailed planning method because we both have had some bad experiences with these kinds of projects in the past, and wanted to prevent stressful situations where possible. In the end we did miscalculate the time that we both had to spend on extracurricular activities which forced us to adapt our planning.

One of the tasks that we hadn’t calculated any time for was resolving merge conflicts. This is, as many know, a very tedious and frustrating process and we might have eased it by agreeing on some points before hand. There was also some miscommunication and because one partner primarly used the Egit and Bash implementations of git and the other primarily the GitHub GUI & Shell, there was some extra time lost.

We might also have atomized our tasks some more because at one point there was a card in Trello which said “Implement Client”.

Do’s:

* Use a dynamic planning environment such as Trello
* Atomize and S.M.A.R.T.en up your tasks
* Estimate how much time a task will take up

Don’ts

* Use git without good communication and base agreements
* Put your personal deadline to close on the official one. You want enough time for bug fixing.

# Security

### What risks is our system exposed to?

A system implemented fully according to the AMULET standards could be vulnerable to (SQL)injection?

Because an open connection is used clients are vulnerable to impersonation and Man-in-the-middle attacks.

Also Java has a history of know exploits and memory leaks.

### What measures did we take to minimize those

**SQLinjection**

This would normally be hedged against by sanitizing input. In our system all input is checked. We don’t have any direct connection to a database so SQLinjection is not relevant for us.

**Man-in-the-middle**

If the AMULET security protocol was implemented this was protected against because of the secure SSL connection

**Impersonation**

Registering players would prevent this together with the protection against man-in-the-middle.

**Java workarounds?**

No protection possible

# Process Description of the Developers

#### Building the foundation first

We developed the class diagrams for the Server Client and Model first in our development process, after putting all the requirements in a planning roster. Subsequently we documented the class diagram explanations and per-class discussions to create a solid base for our application. With the “create code” function of Visual Paradigm and our usage of interfaces we had a good and inter-independent construct to fill in. We relied predominantly on this report for the Javadoc and agreed that while filling in the bodies of our methods to create Javadoc, JML and test first and add logic later because both of us have had bad experiences in the past with trying to come up with an application structure while coding at the same time. For the tests we created 3 separate JUnits Suites, one for the model, one for the client and one for the server. To implement a class we would typically start with the method stubs generated from the class diagrams, add Javadoc and where relevant JML. Only then we would write the method bodies. During the development process we still found flaws in our design and the diagram, Javadoc, JML and tests still needed to be edited but the overall development method helped us add a lot of structure to our process. We can confidently say that this method, although a little unrewarding in the setup phase, introduced a lot of peace and clarity to the development process and in our experience enabled us to create a system of higher quality and elegance.

#### Tools used

We used a couple of tools to smooth the development process for us. First of all Github and for integration with eclipse EGit. The advantages of git are widely known and we won’t bore the reader with repeating them here.

For planning and task overview purposes we used the online service Trello, which uses cards to create an interactive S.M.A.R.T task assignment, tracking and planning environment.

For GUI Design we used the google-developed WindowbuilderPro which supports a visual drag-and-drop interface for GUI design.