Inhaltsverzeichnis

1. Abstract

In dieser Maturaarbeit geht es um die Entwicklung eines Computerspiels mit «Java» und der Library «Jeda». Das Kartenspiel Tichu wurde im Rahmen dieser Arbeit als Computerspiel mit Verwendung von Client-Server-Kommunikation und einer grafischen Oberfläche umgesetzt.

Im Theorieteil setzt sich diese Maturaarbeit mit Client-Server-Kommunikation, dem TCP/IP-Modell und dem Delegations-Event-Modell auseinander. Im Praxisteil wird nach der Erklärung der Regeln des Kartenspiels Tichu die Vorgehensweise nach dem Wasserfallmodell erläutert, welches in dieser Arbeit umgesetzt wurde. In den weiteren Kapiteln werden der Ablauf der Planung der Software beschrieben, die Implementierung der wichtigsten Programmelemente (wie Kommunikation, Spiellogik und Grafik) erklärt und die verwendeten Testszenarien erläutert.

2. Zielsetzung

Das Ziel dieser Arbeit war, ein Programm zu entwickeln, welches vier Spielern erlaubt, zusammen das Kartenspiel Tichu zu spielen. Das Programm soll vier Geräte mittels Client-Server-Kommunikation verbinden und eine grafische Oberfläche für das Spiel bieten. Das Programm soll nur spielregelkonforme Spielzüge zulassen und am Ende jeder Spielrunde die Punkte der Teams zusammenzählen.

3. Theorie

3.1 TCP/IP

TCP/IP steht für Transmission Control Protocol/Internet Protocol und ist eine Familie von Netzwerkprotokollen. Wegen ihrer Wichtigkeit für das Internet, wird sie auch häufig als Internetprotokollfamilie bezeichnet. Die Erkennung von Geräten funktioniert beim TCP/IP mittels IP-Adressen. TCP/IP gehört zu den ersten Netzwerkprotokollen, die auf allen gängigen Betriebssystemen laufen, und ist das einzige erfolgreiche dieser Art. Gegen seine früheren Konkurrenten, die jeweils für ein Betriebssystem spezialisiert waren, wie zum Beispiel AppleTalk für Apple oder netBEUI für Windows, setzte TCP/IP sich vor allem wegen seiner Flexibilität durch und auch wegen der Verbreitung des Internets, welches ebenfalls IP-Adressen braucht. [1] «Kommunikation wird in Rechnernetzen durch Netzwerkprotokolle umgesetzt und in der Praxis in funktionale Schichten (*layer*) unterteilt.»[2] Dabei werden das Internet und die Internetprotokollfamilie nach dem TCP/IP-Referenzmodell gegliedert. Das TCP/IP-Referenzmodell besteht aus vier Schichten, die aufeinander aufbauen, und zwar aus der Anwendungsschicht, der Transportschicht, der Internetschicht und der Netzzugangsschicht. In der untenstehenden Tabelle sind die einzelnen Schichten und die dazugehörigen Protokolle, die bei dieser Arbeit benutzt wurden, abgebildet.

TCP/IP-Schicht	**Protokoll**
Anwendung	Tichu-Protokoll
Transport	TCP
Internet	IP
Netzzugang	Ethernet/WLAN

3.2 Client-Server-Kommunikation

So gut wie immer, wenn zwei Geräte miteinander Daten austauschen, spielt die Client-Server-Kommunikation eine Rolle. Wie der Name schon sagt, geht es bei der Client-Server-Kommunikation um die Kommunikation zwischen Client und Server. Der Client ist dabei das Programm, das Anfragen an den Server sendet und mit welchem der Nutzer interagiert. Wogegen der Server das Programm ist, welches auf Anfragen des Clients reagiert und antwortet. Beim Tichuspiel sendet der Client beispielsweise eine Anfrage an den Server, ob die vom Spieler gewählten Karten eine gültige Kombination ergeben, worauf der Server die Anfrage überprüft und die Antwort an den Client zurückschickt. Der Server wartet immer auf einem vordefinierten Port auf eine Anfrage des Clients, wobei allen Clients, die mit dem Server interagieren wollen, der Port bekannt sein muss. Die Portnummern gehen von 0 bis 65535, wovon die Ports 0 bis 1023 für Systemdienste reserviert sind und nicht anders gebraucht werden dürfen. [3]

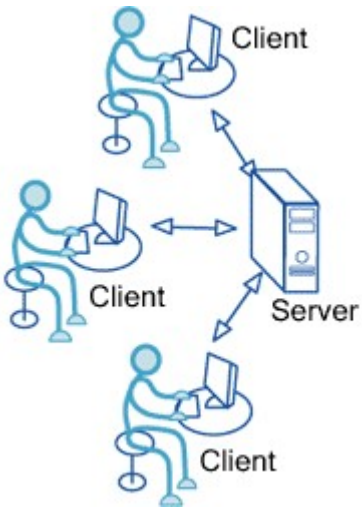


Abbildung 1: Beispiel eines Client-Server-Aufbaus

[1] Wikipedia-Artikel Transmission Control Protocol/Internet Protocol: https://de.wikipedia.org/wiki/Transmission_Control_Protocol/Internet_Protocol
[2] Wikipedia-Artikel Internetprotokollfamilie: https://de.wikipedia.org/wiki/Internetprotokollfamilie
[3] Jedadokumentation Netzwerk-Kommunikation: https://jeda.ch/wiki/doc:network

3.3 Delegations-Eventmodell

Als Eventmodell bezeichnet man das Verfahren, mit welchem dieses Konzept mit Callbackmethoden programmiertechnisch umgesetzt wird. Da das klassische Eventmodell nicht objektorientiert ist und daher nicht wirklich in eine Klassenstruktur passt, entwickelten die Java-Entwickler in der Java Version 1.1 das Delegations-Eventmodell, welches viel besser zum objektorientierten Programmieren passte.[4]

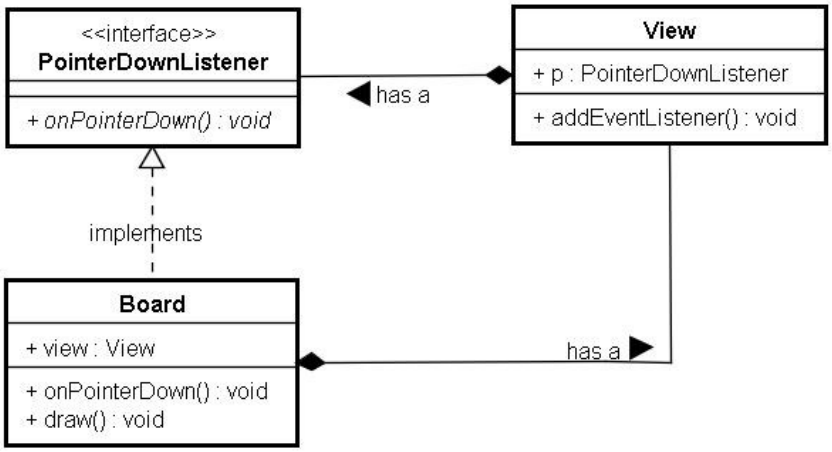


Abbildung 2: Klassendiagramm mit Eventlistener

Sobald ein Programm nicht mehr sequentiell abläuft, da es auf Ereignisse, wie Tastatur- oder Mauseingaben, wartet, wird es als ereignisgesteuertes Programm bezeichnet. Ereignisgesteuerte Programme haben Methoden, die inaktiv sind, bis ein bestimmtes Event sie aktiviert. Solche Methoden nennt man Callbackmethoden. Ein Programm, welches Callbackmethoden verwenden will, muss einen Eventlistener implementieren, wodurch beim Auftreten des erwarteten Events die zugehörige Callbackmethode aufgerufen werden kann. Die Callbackmethode wird in der Klasse deklariert, welche den Eventlistener implementiert, wodurch selbst bestimmt werden kann, was passieren soll, sobald die Callbackmethode aufgerufen wird.[4] Im Tichuprogramm wird zum Beispiel zum Markieren der Karten der PointerDownListener benötigt, da dieser bei einem Mausklick ein Event auslöst, über das auch die Position des Mausklicks abgefragt werden kann. In diesem Beispiel implementiert die Klasse Board, die für die grafische Oberfläche zuständig ist, den PointerDownListener, weshalb sie die Methode onPointerDown() haben muss. Danach muss der Klasse View, welche dem Fenster, in dem die Grafik gezeichnet wird, entspricht, ein PointerDownListener hinzugefügt werden, welcher in diesem Fall das Objekt der Klasse Board ist. Dies ist nur möglich, weil die Klasse Board den EventListener implementiert. Durch das Hinzufügen des Board Objekts als PointerDownListener zur Klasse View, kann die Klasse View nun die Callbackmethode onPointerDown() in der Board-Klasse aufrufen, falls mit der Maus in das Fenster geklickt wird.

[4] Plüss, Ägidius: Java Exemplarisch. S.217-224

4. Tichu

Tichu ist ein Kartenspiel für vier Spieler, wobei jeweils die zwei einander diagonal gegenübersitzenden Spieler ein Team bilden. Gespielt wird Tichu mit 56 Karten: von Zwei bis Ass in vier Farben plus vier Sonderkarten. Ziel des Spiels ist, in den einzelnen Spielrunden möglichst wertvolle Karten in den Stichen einzufangen, um so als erstes Team die zum Voraus vereinbarte Zielpunktzahl (in der Regel 1'000 Punkte) zu erreichen. Eine andere Möglichkeit Punkte zu sammeln, abgesehen von den Stichen, ist die Ansage eines Tichus. Ein Spieler kann ein Tichu ansagen, bevor er seine erste Karte spielt. Mit dem Ansagen eines Tichus, sagt er, dass er als Erster fertig sein wird. Ist diese Annahme richtig und er wird tatsächlich Erster, so erhält er 100 Punkte zusätzlich. Wird er jedoch nicht Erster, so werden ihm 100 Punkte abgezogen. Neben dem «normalen» Tichu gibt es auch die Möglichkeit, ein grosses Tichu anzusagen. Dies muss ein Spieler jedoch vor dem Aufnehmen der neunten Karte ansagen; ist er erfolgreich, so erhält er 200 Punkte.

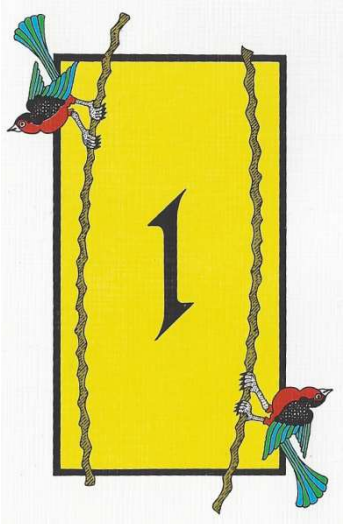
4.1 Die Kombinationen

Beim Tichu gibt es viele verschiedene Kartenkombinationen:

Einzelkarte	Eine einzelne Karte
Paar	Zwei Karten von derselben Höhe
Drilling	Drei Karten von derselben Höhe
Full House	Ein «Drilling» und ein «Paar», wobei die Höhe des «Drillings» die Höhe bzw. den Wert des «Full House» bestimmt
Treppe	Zwei oder mehr direkt auf einander folgende «Paare»
Strasse	Fünf oder mehr direkt auf einander folgende Karten. Nur eine gleichlange «Strasse» mit höheren Karten kann eine andere «Strasse» schlagen
Bombe	Alle vier Karten derselben Höhe
Strassenbombe	Eine «Strasse» in nur einer Farbe. Eine längere «Strassenbombe» schlägt eine kürzere, egal wie hoch sie ist.

4.2 Die Sonderkarten

4.2.1 Der «Mah Jong»



Der «Mah Jong» wird häufig auch «Eins» genannt und hat auch diese Höhe. Der Spieler, der beim effektiven Spielstart den «Mah Jong» auf der Hand hat, muss ausspielen. Dabei ist es egal, ob er den «Mah Jong» oder eine andere Kartenkombination ausspielt. Wird der «Mah Jong» als Einzelkarte gespielt - was neben der «Strasse» die einzige Kombination ist, in welcher der «Mah Jong» gespielt werden darf - so kann der Spieler, der ihn gespielt hat, eine Karte wünschen, die gespielt werden muss. Sonderkarten dürfen nicht gewünscht werden. Falls der nächste Spieler die gewünschte Karte hat, muss er sie spielen. Falls er die gewünschte Karte nicht hat, kann er eine andere Karte spielen und der nächste Spieler, der kann, muss die gewünschte Karte spielen. Zum Teil wird Tichu auch so gespielt, dass jeder, der die gewünschte Karte nicht hat, passen muss, bis jemand die gewünschte Karte spielen kann.

4.2.2 Der «Hund»

Der «Hund» ist die tiefste Karte im Spiel. Er kann nur zum Ausspielen genutzt werden, da er über keine andere Karte gespielt werden kann. Wird der «Hund» ausgespielt, so geht das Ausspielrecht an den Teampartner über. Falls der Teampartner bereits fertig ist, wird das Ausspielrecht nach rechts weitergegeben.

4.2.3 Der «Drache»

Der «Drache» ist die höchste Karte im Spiel. Er kann nur einzeln gespielt werden und ist 25 Punkte Wert. Wird ein Stich mit dem «Drachen» gemacht, so muss der ganze Stich einem der Gegner gegeben werden.

4.2.4 Der «Phönix»

Der «Phönix» funktioniert wie ein Joker und kann folglich als beliebige andere Karte in Kombinationen eingebaut werden. Einzig in «Bomben» darf der «Phönix» nicht eingebaut werden. Wird der «Phönix» einzeln gespielt, so ist er immer einen halben Punkt höher als die zuvor gespielte Karte. Wird er beispielsweise über eine «Zwei» gespielt, so hat er einen Wert von zweieinhalb. Somit kann der «Phönix» auch über ein Ass gespielt werden und ist nach dem «Drachen» die höchste Einzelkarte. Allerdings hat der «Phönix» einen Wert von -25 Punkten, so dass sein Einsatz gut zu überdenken ist.

4.3 Ablauf

Wenn alle Spieler ihre Karten bekommen haben, beginnt das so genannte Schupfen, bei dem jeder Spieler allen anderen je eine Karte gibt. Nach dem Schupfen beginnt der Spieler, der den «Mah Jong» in seinen Karten hat, mit dem Spielen. Dabei ist es egal, ob er als erstes den «Mah Jong» oder eine andere Karte spielt. Auf eine gespielte Kombination kann nur eine höhere gleichartige Kombination oder eine «Bombe» bzw. «Strassenbombe» gespielt werden. Falls der Spieler, der an der Reihe ist, keine höhere Kombination spielen kann oder will, so kann er passen. Wenn gegen Schluss der Spielrunde alle anderen Spieler gepasst haben, geht der Stich an jenen Spieler, der die letzte Kombination gespielt hat. Der Spieler, der den Stich gewonnen hat, erhält das Ausspielrecht. Wenn ein Spieler keine Karten mehr auf der Hand hat, ist er fertig mit der Spielrunde.

4.4 Ende der Runde

Die Spielrunde endet, sobald beide Spieler desselben Teams fertig sind. Falls noch keiner aus dem anderen Team fertig ist, haben sie einen Doppelsieg errungen und erhalten 200 Punkte. Falls schon ein Spieler des gegnerischen Teams fertig ist, gehen die Handkarten des Spielers, der nicht fertig wurde, an das gegnerische Team, welches ganz fertig ist. Seine Stiche gehen an den Spieler, der zuerst fertig war. Die Punkte der Stiche werden zusammengezählt und das Team mit mehr Punkten gewinnt die Runde. Gezählt wird so, dass die Fünfer 5, die Zehner und Könige 10, der Drache 25 und der Phönix -25 Punkte geben. Alle anderen Karten geben keine Punkte, so dass es insgesamt 100 Punkte pro Spielrunde gibt.[5]

[5] Für weitere Inforationen siehe Tichu Anleitung: http://www.fatamorgana.ch/tichu/ti_regel.pdf

5. Vorgehen

Dieses Projekt wurde nach dem Wasserfallmodell entwickelt. Das Wasserfallmodell unterteilt ein Projekt in mehrere Phasen. In der ersten Phase müssen die Anforderungen an das Projekt definiert werden. In der nächsten Phase wird die Softwarearchitektur entworfen. In der dritten Phase wird die Software implementiert. In der darauffolgenden Phase wird die Software getestet. Die letzte Phase des Wasserfallmodells ist das Warten der Software. Auf diese Phase konnte bei diesem Projekt verzichtet werden, da die Anwendung nicht publiziert wird.

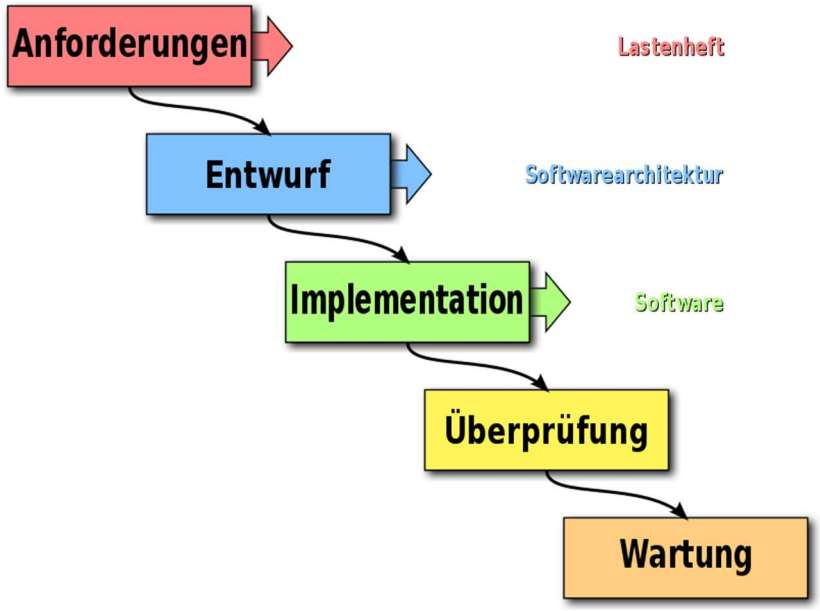


Abbildung 3:Abbildung des Wasserfallmodells

Ausserhalb dieser Phasen wurde auch noch ein Zeitplan erstellt, nach welchem grob gearbeitet wurde.

6. Design

Nachdem Idee und Betreuer gefunden waren, war der nächste Schritt die Planung des Projektes. Zuerst wurde grob überlegt, welche Funktionen das Programm haben muss. Als nächstes wurde eine Programmiersprache gewählt, in der das Programm später implementiert werden sollte. Da der Autor sowohl im Ergänzungsfach Informatik, wie auch früher im Fakultativfach Programmieren, mit «Java» arbeitete, fiel die Wahl auf die Verwendung von «Java», zusammen mit der Library «Jeda», die bereits im Fakultativfach verwendet wurde. Die Library «Jeda» wurde verwendet, da diese Klassen zur grafischen Darstellung und zu Client-Server-Kommunikation beinhaltet. Die nächsten Schritte bestanden aus dem Einlesen in die Client-Server-Kommunikation von «Jeda» und dem Erstellen eines Klassendiagramms.

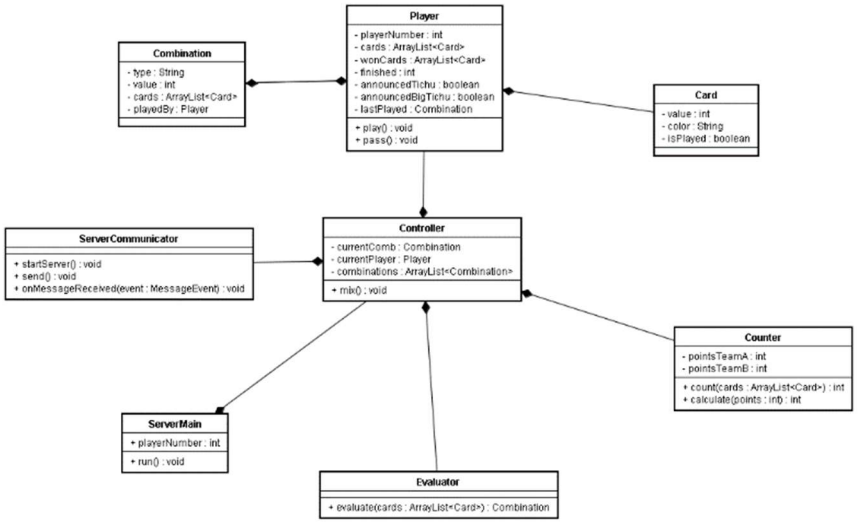


Abbildung 4: Beispiel eines Klassendiagramms

Da beim Erstellen des Klassendiagramms immer wieder neue Dinge, an die vorher noch nicht gedacht wurden, eingebaut werden mussten, dauerte es eine Weile bis es so weit ausgearbeitet war, dass mit der Implementierung begonnen werden konnte. Vorher wurde aber noch eine Skizze der grafischen Oberfläche erstellt, die bei der Implementierung als Orientierung diente.

7. Implementierung

7.1 Kommunikation

Die Client-Server-Kommunikation wurde mit den Klassen «Connection» und «Server» der «Jeda» Library umgesetzt. Als erstes muss im Server-Programmteil ein neues Serverobjekt erstellt und an einem angegebenen Port gestartet werden. Danach kann im Client-Programmteil ein neues Connection-Objekt erstellt werden und mit der «open()-Methode» eine Verbindung zum Server auf dem angegebenen Port hergestellt werden. Wenn eine Verbindung zum Server hergestellt wurde, erhält der Server ein ConnectionEvent, aus dem er ein Connection-Objekt erstellen kann. Dieses Connection-Objekt entspricht der Verbindung zum Client, über welche der Server mit dem Client kommunizieren kann. Client und Server kommunizieren in diesem Fall mithilfe von Strings miteinander.

In der untenstehenden Tabelle sind die Strings, die zur Kommunikation gebraucht werden, und deren Bedeutung abgebildet.

Strings	Übermittelt	Erklärung	Sender
«Cards:k1,k2,…,k14»	Karten des Spielers	Karten-IDs: k1,…,k14	Server
«x:SchupfCards:k1,k2,k3»	Karten, die Spieler schupft	Karten-IDs: k1, k2, k3, x = Spielernummer	Client
«x:Play:k1,…,kn»	Karten, die Spieler spielen will	Karten-IDs: 0<n<15, k1,…,kn:, x = Spielernummer	Client
«x:Pass»	Spieler passt	x = Spielernummer	Client
«Error:Text»	Problem beim Spielen der Karten	Text: Grund für Problem	Server
«Played:k1,…,kn»	Neu gespielte Karten	Karten-IDs: 0<n<15, k1,…,kn:	Server
«SchupfedCards:k1,k2,k3»	Karten, die dem Spieler geschupft werden	KartenIDs: k1,k2,k3:	Server
«YourTurn:true»	Spieler ist an der Reihe		Server
«YourTurn:false»	Spieler ist nicht mehr an der Reihe		Server
«Passed:x»	Spieler x hat gepasst	x = Spielernummer	Server
«Message:Text»	Nachricht	Text: Nachricht	Server
«x:Pass: »	Spieler passt	x = Spielernummer	Client
«x:Won:Round»	Spieler gewinnt Runde	x = Spielernummer	Client
«x:Finnished:now»	Spieler hat keine Karten mehr	x = Spielernummer	Client
«x:Phoenix:n»	Welchen Wert Phönix in Kombination hat	x = Spielernummer, n = Kartenhöhe	Client
«x:MahJong:n»	Gewünschte Kartenhöhe	x = Spielernummer, n = Kartenhöhe	Client
«x:Dragon:p»	Welcher Gegner den Drachen bekommen soll	x = Spielernummer, p = Nummer des Spielers der Drache bekommt	Client
«Finnished:x»	Spieler x hat keine Karten mehr	x = Spielernummer	Server
«Won:x»	Spieler x gewinnt den Stich	x = Spielernummer	Server
«RoundOver:a,b,A,B»	Die Runde ist vorbei	a,b = Punkte der Teams dieser Runde, A,B = Punkte der Teams insgesamt	Server

Da zur Kommunikation Strings verschickt werden, müssen beim Spielen nicht die Kartenobjekte zwischen Client und Server verschoben werden, sondern nur ihre IDs und die Anweisung, was damit gemacht werden soll, wodurch nicht noch die Objekte serialisiert werden müssen.

7.2 Spiellogik

Das wichtigste Element für das Implementieren der Spiellogik ist die Evaluator-Klasse, welche in vier Hauptschritten überprüft, ob die vom Spieler ausgewählten Karten eine gültige Kombination ergeben. Zuerst wird überprüft, wie viele Karten ausgewählt wurden. Danach erfolgt die Überprüfung der für diese Kartenanzahl möglichen Kombinationen. Danach wird geprüft, ob die gespielte Kombination von derselben Art ist wie die zuletzt gespielte. Im letzten Schritt wird kontrolliert, ob die gewählte Kombination höher ist als die vom vorherigen Spieler gespielte Kombination.

7.3 Grafik

Die grafische Oberfläche wurde mithilfe der Klassen «View» und «Canvas» aus der «Jeda Library» implementiert. Nach jedem Spielzug wird die Methode «draw()» aufgerufen, welche das ganze Spielfeld neu zeichnet. Also die eigenen Karten, die zuletzt gespielten Kombinationen der anderen Spieler und die Knöpfe zum Spielen und Passen oder zum Schupfen.

```
public void draw() {
    background.setColor(Color.WHITE);
    background.fillRectangle(0, 0, width, height);
    for(int i=0;i<controller.myCards.size();i++){
        Card c = controller.myCards.get(i);
        if(c.isSelected){
            Image img = new Image(c.imageS);
            background.drawImage(i*cardW, 0, cardW, cardH, img);
        }
        else{
            Image img = new Image(c.image);
            background.drawImage(i*cardW, 0, cardW, cardH, img);
        }

    }
    background.setTextSize(24);
    background.setColor(Color.BLACK);
    if(pass[me]){
        background.drawText(width/2, height/9*2, "Pass");
    }
    else{
        int meSize = controller.playedCards[me].size();
        for(int i = 0; i < meSize; i++){
            Card c =controller.playedCards[me].get(i);
            Image img = new Image(c.image);
```

Beispielcode 1: Die Methode draw()

In der Methode «draw()» wird zuerst das ganze Fenster von einem weissen Rechteck bedeckt, was als Löschvorgang dient, bei dem die alte Grafik von vorher überdeckt wird. Dies ist nötig, da die bereits gespielten Karten sonst teilweise weiterhin angezeigt würden. Nachdem das weisse Rechteck gezeichnet wurde, wird mit einer «for-Schleife» durch die Handkarten des Spielers gewandert, und die Karten werden gezeichnet. Dabei wird bei jeder Karte geprüft, ob sie angewählt ist oder nicht, und je nachdem wird die Grafik mit oder ohne gelbe Markierung gezeichnet. Danach werden die Textgrösse und -farbe gesetzt und bei jedem Spieler getestet, ob er gepasst hat. Gegebenenfalls wird an der richtigen Stelle «Pass» geschrieben, und andernfalls die zuletzt gespielte Kombination gezeichnet.

8. Testing

Testobjekt	Testszenario	Erwartetes Verhalten	Ergebnis	Datum
Mah Jong	Die zuletzt gespielte Karte ist der Mah Jong. Der aktive Spieler möchte eine andere Karte spielen, als gewünscht wurde.	Die Karte wird nicht als gültige Kombination erkannt und der Spieler bekommt eine Fehlermeldung.	✓	21.10.2018
Mah Jong	Die zuletzt gespielte Karte ist der Mah Jong. Der aktive Spieler möchte passen, obwohl er die gewünschte Karte hat.	Das Passen wird nicht akzeptiert und der Spieler bekommt eine Fehlermeldung.	✓	21.10.2018
Phönix	Der Phönix wurde über eine Einzelkarte gespielt. Der nächste Spieler möchte eine Karte spielen, die um eins höher ist, als die Karte vor dem Phönix.	Die Karte kann gespielt werden.	✓	21.10.2018
Phönix	Der aktive Spieler möchte eine Einzelkarte zusammen mit dem Phönix (mit dem selbem Wert) als Paar spielen.	Das Paar wird akzeptiert.	✓	21.10.2018
Spielablauf	Alle Spieler die noch Karten haben ausser demjenigen, der die letzte Kombination gespielt hat, passen.	Der Spieler, der die letzte Kombination gespielt hat, gewinnt den Stich und kann ausspielen.	✓	21.10.2018
Spielablauf	Ein Spieler, der keine Karten mehr hat, kommt an die Reihe.	Er wird übersprungen und der Spieler nach ihm ist an der Reihe.	✓	21.10.2018
Spielablauf	Drei Spieler haben keine Karten mehr und sind somit fertig.	Die Runde endet und die Punkte werden zusammengezählt.	✓	21.10.2018
Spielablauf	Zwei Spieler haben keine Karten mehr und sind aus demselben Team.	Die Runde endet und das Team, das fertig ist, bekommt 200 Punkte.	✓	21.10.2018
Evaluator	Der aktive Spieler will eine Kombination von anderer Art spielen als die zuletzt gespielte Kombination.	Die Kombination wird nicht akzeptiert und der Spieler bekommt eine Fehlermeldung.	✓	21.10.2018
Server	Es sind bereits vier Clients mit dem Server verbunden und ein fünfter will sich verbinden.	Der Client bekommt eine Fehlermeldung, dass bereits vier Spieler im Spiel sind.	✓	21.10.2018

9. Fazit

Rückblickend betrachtet, wäre es besser gewesen, mit dem Schreiben des Berichts früher zu beginnen und vor allem den Teil zur Implementierung parallel zum Implementieren zu schreiben, so wie es vom Betreuer empfohlen wurde. Da die Freude am Programmieren viel grösser war als das Interesse am Schreiben des Berichts, ging letzteres leider etwas unter. Für ein anders Mal wäre es sicherlich auch gut, mehr Zeit für die Implementierung einzuplanen. Der Zeitaufwand wurde etwas unterschätzt, wodurch der Terminplan gegen Ende nicht eingehalten werden konnte, wodurch auch einiges an theoretisch vermeidbarem Stress entstand. Eine weitere Verbesserungsmöglichkeit wäre, den Programmcode von Anfang an und durchgehend zu kommentieren. Dies würde helfen, den Programmcode verständlich zu machen und die Fehlersuche erleichtern.
Erfreulich ist die Tatsache, dass das Programm an und für sich gut gelungen ist und Tichu (wenn auch nicht mit allen möglichen Spielelementen) wie vorgesehen von vier Spielern gespielt werden kann.

10. Anhang

10.1 Glossar

- **Implementierung:** Als Implementierung bezeichnet man in der Informatik den Prozess, bei welchem die Idee des Programms in Programmcode geschrieben wird.
- **Klasse:** Klassen sind die Grundbausteine vieler Programmiersprachen. Klassen haben Attribute und Methoden.
- **Library:** Als Library wird beim Programmieren eine Sammlung von bereits von anderen geschriebenen Klassen bezeichnet.
- **Port:** Ein Port ist eine Art Zuang, über welchen ein Gerät mit einem anderen kommunizieren kann.

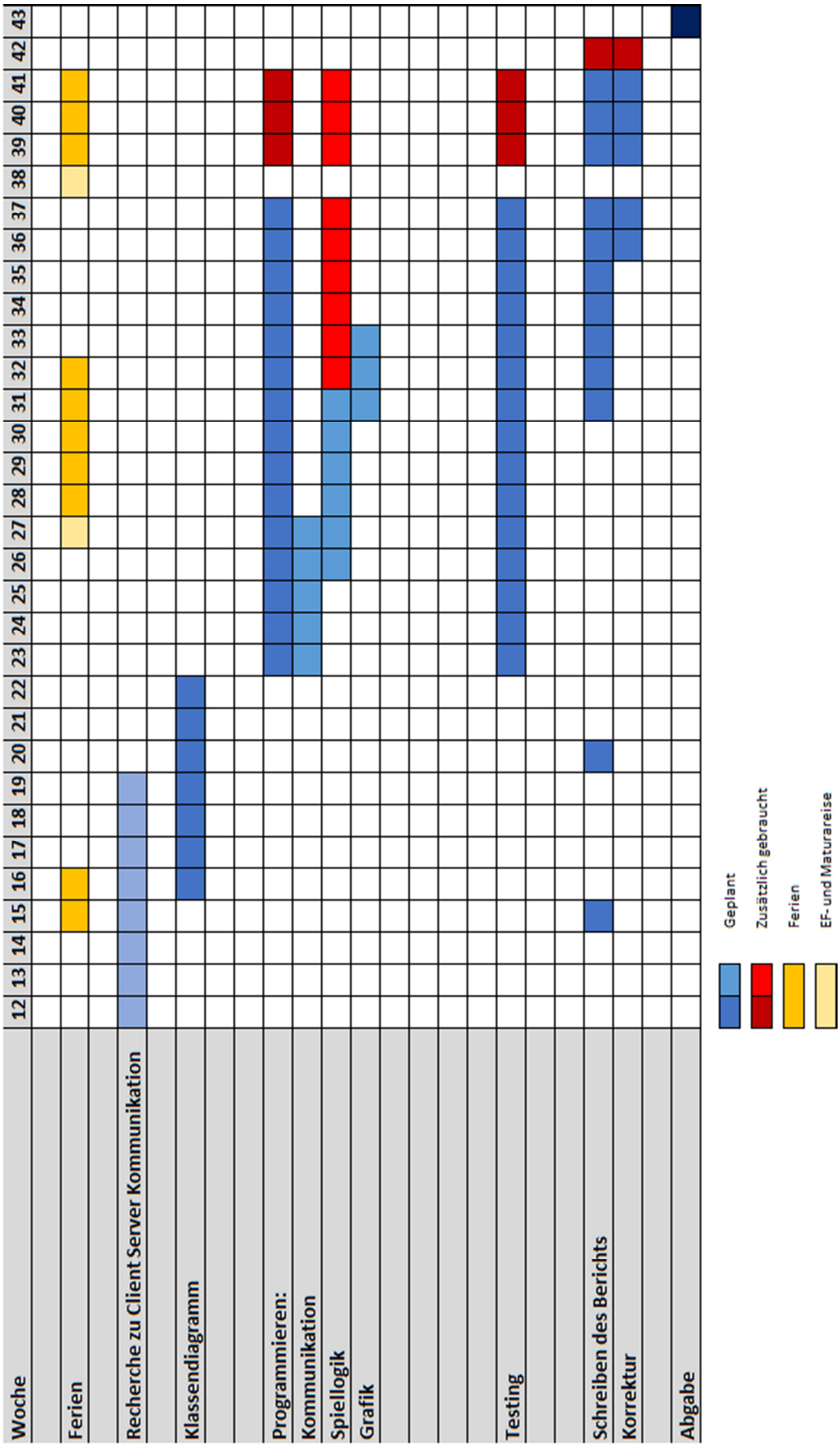
10.2 Literaturverzeichnis

[1] Wikipedia. Transmission Control Protocol/Internet Protocol. URL: https://de.wikipedia.org/wiki/Transmission_Control_Protocol/Internet_Protocol (zuletzt besucht am 20.10.2018)
[2] Wikipedia. Internetprotokollfamilie. URL: https://de.wikipedia.org/wiki/Internetprotokollfamilie (zuletzt besucht am 20.10.2018)
[3] Jedadokumentation Netzwerk-Kommunikation. URL: https://jeda.ch/wiki/doc:network (zuletzt besucht am 20.10.2018)
[4] Plüss, Ägidius: Java Exemplarisch. Learning by doing. Oldenbourg 2004.
[5] Fatamorgana. Tichu Anleitung. URL: http://www.fatamorgana.ch/tichu/ti_regel.pdf (zuletzt besucht am 20.10.2018)

10.3 Abbildungsverzeichnis

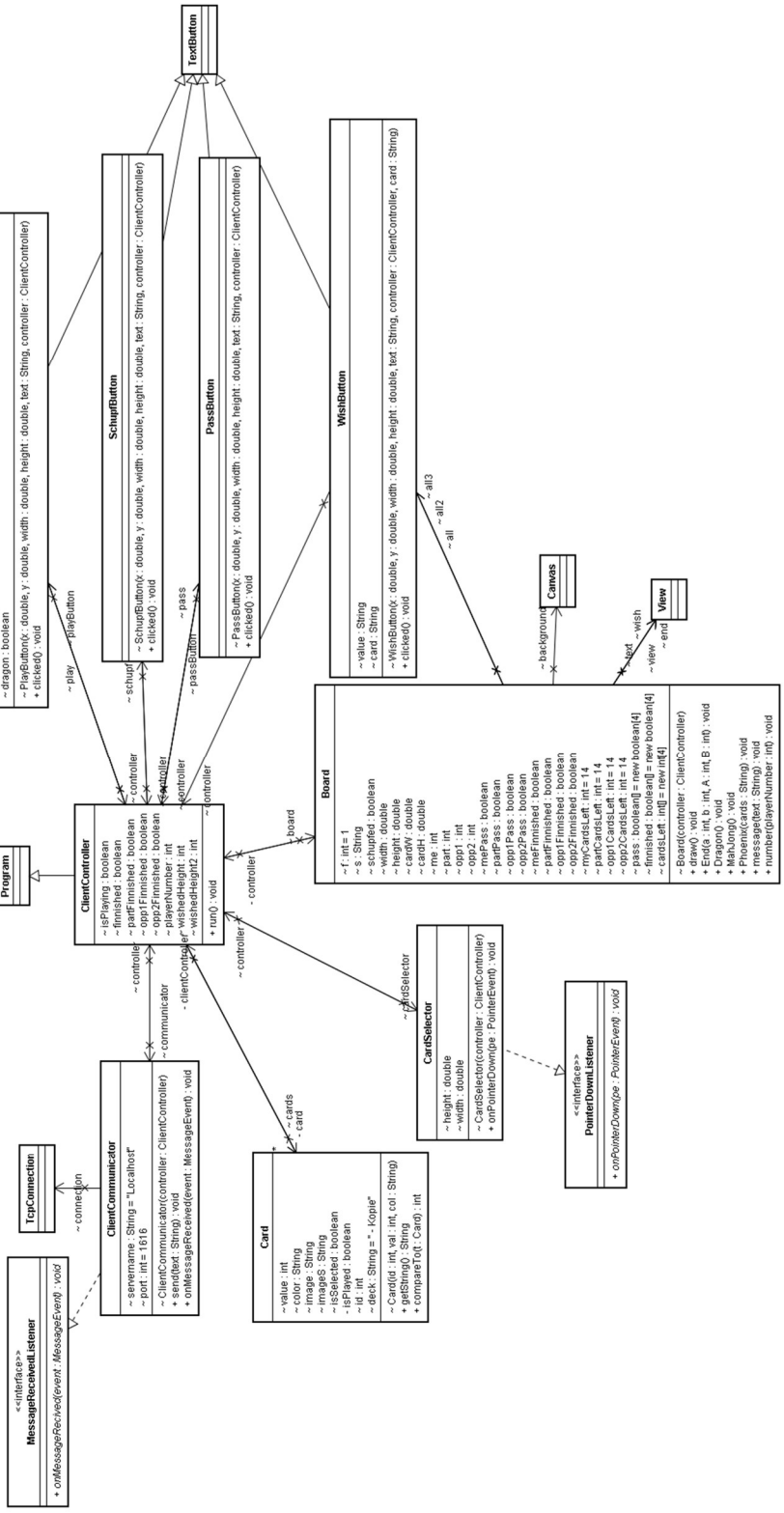
[1] https://www.e-teaching.org/technik/vernetzung/architektur/client-server/client_server
[2] Selbst erstellt
[3] https://de.wikipedia.org/wiki/Wasserfallmodell#/media/File:Waterfall_model-de.svg
[4] Selbst erstellt

10.4 Terminplan

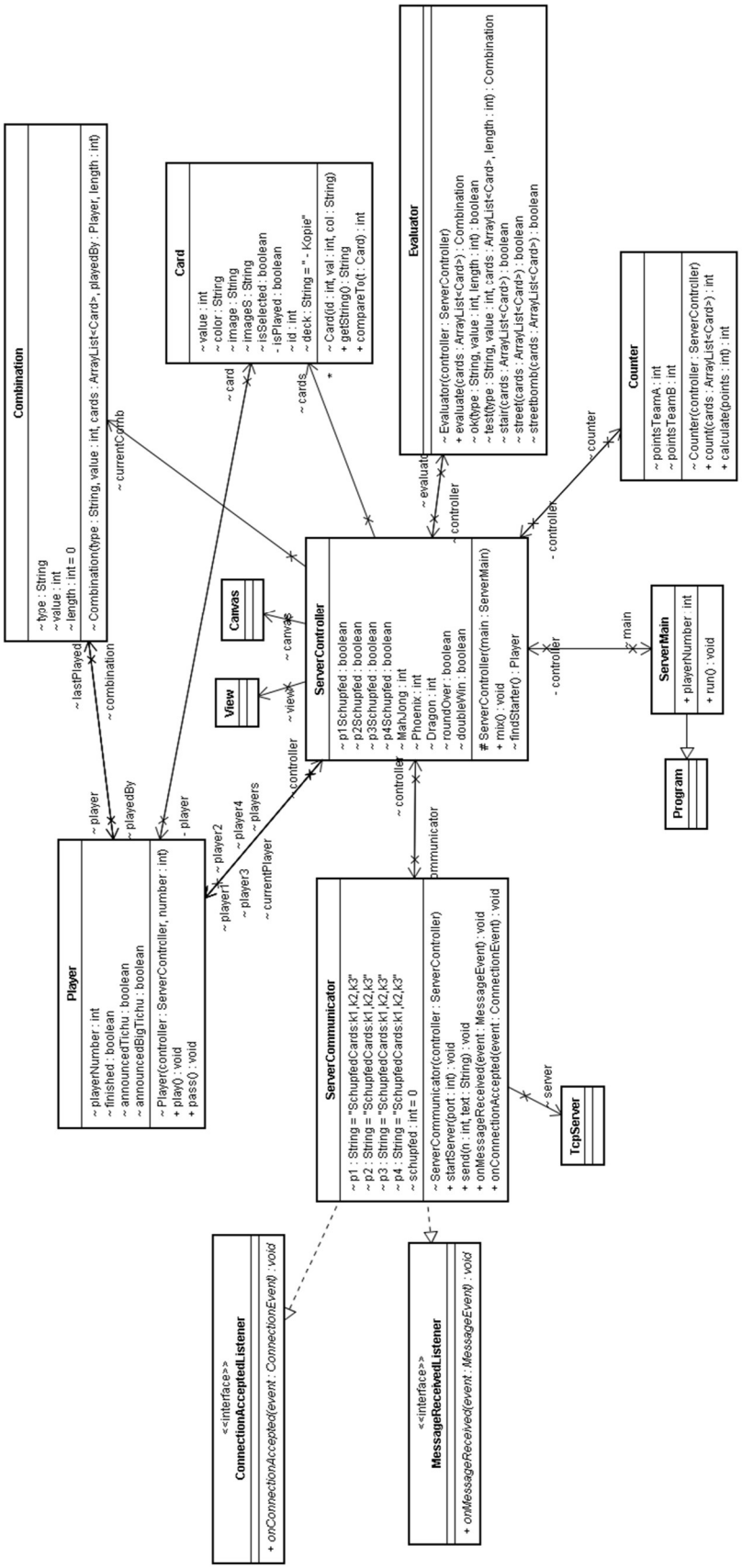


10.5 Klassendiagramm

10.5.1 Client



10.5.2 Server


Combination
~ type : String
~ value : int
~ length : int = 0
~ Combination(type : String, value : int, cards : ArrayList<Card>, playedBy : Player, length : int)
Card
~ value : int
~ color : String
~ image : String
~ imageS : String
~ isSelected : boolean
- isPlayed : boolean
~ id : int
~ deck : String = "- Kopie"
~ Card(id : int, val : int, col : String)
+ getString() : String
+ compareTo(t : Card) : int
Evaluator
~ Evaluator(controller : ServerController)
+ evaluate(cards : ArrayList<Card>) : Combination
+ evaluate(cards : ArrayList<Card>, length : int) : Combination
~ ok(type : String, value : int, length : int) : boolean
~ test(type : String, value : int, cards : ArrayList<Card>, length : int) : Combination
~ stair(cards : ArrayList<Card>) : boolean
~ street(cards : ArrayList<Card>) : boolean
~ streetbomb(cards : ArrayList<Card>) : boolean
Counter
~ pointsTeamA : int
~ pointsTeamB : int
~ Counter(controller : ServerController)
+ count(cards : ArrayList<Card>) : int
+ calculate(points : int) : int
ServerController
~ p1Schupfed : boolean
~ p2Schupfed : boolean
~ p3Schupfed : boolean
~ p4Schupfed : boolean
~ MahJong : int
~ Phoenix : int
~ Dragon : int
~ roundOver : boolean
~ doubleWin : boolean
# ServerController(main : ServerMain)
+ mitX() : void
~ findStarter() : Player
Canvas
View
Player
~ playerNumber : int
~ finished : boolean
~ announcedTichu : boolean
~ announcedBigTichu : boolean
~ Player(controller : ServerController, number : int)
+ play() : void
+ pass() : void
ServerMain
+ playerNumber : int
+ run() : void
Program
ServerCommunicator
~ p1 : String = "SchupfedCards:k1,k2,k3"
~ p2 : String = "SchupfedCards:k1,k2,k3"
~ p3 : String = "SchupfedCards:k1,k2,k3"
~ p4 : String = "SchupfedCards:k1,k2,k3"
~ schupfed : int = 0
~ ServerCommunicator(controller : ServerController)
+ startServer(port : int) : void
+ send(n : int, text : String) : void
+ onMessageReceived(event : MessageEvent) : void
+ onConnectionAccepted(event : ConnectionEvent) : void
TcpServer
<<interface>>
ConnectionAcceptedListener
+ onConnectionAccepted(event : ConnectionEvent) : void
<<interface>>
MessageReceivedListener
+ onMessageReceived(event : MessageEvent) : void
~ card
~ cards
~ currentComb
~ lastPlayed
~ combination
~ player
~ playedBy
~ player1
~ player2
~ player3
~ player4
~ players
~ currentPlayer
~ canvas
~ view
~ controller
~ evaluator
~ counter
~ main
~ server
~ communicator

10.6 Code

10.6.1 Board

```
package ch.jeda.tichu;

import ch.jeda.ui.*;

public class Board {

    double f = 1;               //Faktor der Fenstergrösse, 1 für 1600*900,
    //1.2 für 1920*1080, 2.4 für 3840*2160
    View view;
    View wish;
    View end;
    View text;

    WishButton[] all;
    WishButton[] all2;
    WishButton[] all3;
    String s;

    Canvas background;
    boolean schupfed;

    private ServerMain serverMain;

    private ClientController controller;

    double width;
    double height;
    double cardW;
    double cardH;

    int me;
    int part;
    int opp1;
    int opp2;

    boolean mePass;
    boolean partPass;
    boolean opp1Pass;
    boolean opp2Pass;

    boolean meFinnished;
    boolean partFinnished;
    boolean opp1Finnished;
    boolean opp2Finnished;

    int myCardsLeft = 14;
    int partCardsLeft = 14;
    int opp1CardsLeft = 14;
    int opp2CardsLeft = 14;

    boolean[] pass = new boolean[4];

    boolean[] finnished = new boolean[4];

    int[] cardsLeft = new int[4];

    Board(ClientController controller) {
        this.controller = controller;
        view = new View((int) (1600 / f), (int) (900 / f));
        view.setTitle("Spieler " + Integer.toString(controller.playerNumber));
        background = view.getBackground();
        width = background.getWidth();
        height = background.getHeight();
        cardW = width / 16;
        cardH = height / 6;

    }

    //Zeichnet die Spieloberfläche (genauere Erklärung in Bericht, Kapitel 8.3)
    public void draw() {
        background.setColor(Color.WHITE);
        background.fillRectangle(0, 0, width, height);
        for (int i = 0; i < controller.myCards.size(); i++) {
            Card c = controller.myCards.get(i);
            if (c.isSelected) {
                Image img = new Image(c.imageS);
                background.drawImage(i * cardW, 0, cardW, cardH, img);
            } else {
                Image img = new Image(c.image);
                background.drawImage(i * cardW, 0, cardW, cardH, img);
```

```
            }

        }
        background.setTextSize(24);
        background.setColor(Color.BLACK);
        background.drawText(width / 2, height / 9 * 8, "" + cardsLeft[part]);
        background.drawText(cardW * 15, cardH * 3.74, "" + cardsLeft[opp1]);
        background.drawText(cardW / 2, cardH * 3.74, "" + cardsLeft[opp2]);
        if (pass[me]) {
            background.drawText(width / 2, height / 9 * 2, "Pass");
        } else {
            int meSize = controller.playedCards[me].size();
            for (int i = 0; i < meSize; i++) {
                Card c = controller.playedCards[me].get(i);
                Image img = new Image(c.image);
                background.drawImage(width / 2 - (meSize / 2 * (cardW / 2)) + i * (cardW / 2), height / 9 * 2, cardW
/ 2, cardH / 2, img);
            }
        }
        if (pass[part]) {
            background.drawText(width / 2 - 2, height / 9 * 7, "Pass");
        } else {
            int partSize = controller.playedCards[part].size();
            for (int i = 0; i < partSize; i++) {
                Card c = controller.playedCards[part].get(i);
                Image img = new Image(c.image);
                background.drawImage(width / 2 - (partSize / 2 * (cardW / 2)) + i * (cardW / 2), height / 9 * 7,
cardW / 2, cardH / 2, img);
            }
        }
        if (pass[opp2]) {
            background.drawText(cardW, cardH * 3.74, "Pass");
        } else {
            int opp2Size = controller.playedCards[opp2].size();
            for (int i = 0; i < opp2Size; i++) {
                Card c = controller.playedCards[opp2].get(i);
                Image img = new Image(c.image);
                background.drawImage(cardW + i * (cardW / 2), cardH * 3.5, cardW / 2, cardH / 2, img);
            }
        }
        if (pass[opp1]) {
            background.drawText(cardW * 14, cardH * 3.74, "Pass");
        } else {
            int opp1Size = controller.playedCards[opp1].size();
            for (int i = 0; i < opp1Size; i++) {
                Card c = controller.playedCards[opp1].get(i);
                Image img = new Image(c.image);
                background.drawImage(cardW * 14 - (opp1Size * (cardW / 2)) + i * (cardW / 2), cardH * 3.5, cardW / 2,
cardH / 2, img);
            }
        }
        if (schupfed) {

        }

    }

    //Fenster mit Punkten am Ende der Runde
    public void End(int a, int b, int A, int B) {
        end = new View(700, 500);
        end.setTitle("Rundenende");
        Canvas canvas = end.getBackground();
        canvas.drawText(300, 400, "Spieler 1 & 3");
        canvas.drawText(500, 400, "Spieler 2 & 4");
        canvas.drawText(100, 200, "Diese Runde:");
        canvas.drawText(300, 200, a + " Punkte");
        canvas.drawText(500, 200, b + " Punkte");
//            canvas.drawText(100,200, "Insgesamt:");
//            canvas.drawText(300,200, A+" Punkte");
//            canvas.drawText(500, 200, B+" Punkte");
    }

    //Fenster zum Auswählen des Spielers, der den Drachen bekommt
    public void Dragon() {
        wish = new View(400, 100);
        wish.setTitle("Wähle eine Gegener");
        Canvas canvas = wish.getBackground();
        WishButton eins = new WishButton(5, 40, 195, 20, "Links", controller, "Dragon");
        WishButton zwei = new WishButton(200, 40, 195, 20, "Rechts", controller, "Dragon");
        wish.add(eins);
        wish.add(zwei);
        all3 = new WishButton[2];
        all3[0] = eins;
        all3[1] = zwei;
    }
```

```
//Fenster zum Wünschen der Karte mit dem Mah Jong
public void MahJong() {
    wish = new View(400, 100);
    wish.setTitle("Wähle eine Kartenhöhe");
    Canvas canvas = wish.getBackground();
    WishButton zwei = new WishButton(5, 40, 30, 20, "2", controller, "MahJong");
    WishButton drei = new WishButton(35, 40, 30, 20, "3", controller, "MahJong");
    WishButton vier = new WishButton(65, 40, 30, 20, "4", controller, "MahJong");
    WishButton funf = new WishButton(95, 40, 30, 20, "5", controller, "MahJong");
    WishButton sechs = new WishButton(125, 40, 30, 20, "6", controller, "MahJong");
    WishButton sieben = new WishButton(155, 40, 30, 20, "7", controller, "MahJong");
    WishButton acht = new WishButton(185, 40, 30, 20, "8", controller, "MahJong");
    WishButton neun = new WishButton(215, 40, 30, 20, "9", controller, "MahJong");
    WishButton zehn = new WishButton(245, 40, 30, 20, "10", controller, "MahJong");
    WishButton j = new WishButton(275, 40, 30, 20, "J", controller, "MahJong");
    WishButton q = new WishButton(305, 40, 30, 20, "Q", controller, "MahJong");
    WishButton k = new WishButton(335, 40, 30, 20, "K", controller, "MahJong");
    WishButton a = new WishButton(365, 40, 30, 20, "A", controller, "MahJong");
    wish.add(zwei);
    wish.add(drei);
    wish.add(vier);
    wish.add(funf);
    wish.add(sechs);
    wish.add(sieben);
    wish.add(acht);
    wish.add(neun);
    wish.add(zehn);
    wish.add(j);
    wish.add(q);
    wish.add(k);
    wish.add(a);

    all = new WishButton[13];
    all[0] = zwei;
    all[1] = drei;
    all[2] = vier;
    all[3] = funf;
    all[4] = sechs;
    all[5] = sieben;
    all[6] = acht;
    all[7] = neun;
    all[8] = zehn;
    all[9] = j;
    all[10] = q;
    all[11] = k;
    all[12] = a;
}

//Fenster zum Wählen der Höhe des Phönix
public void Phoenix(String cards) {
    s = cards;
    wish = new View(400, 100);
    wish.setTitle("Wähle eine Kartenhöhe für den Phönix");
    Canvas canvas = wish.getBackground();
    WishButton zwei = new WishButton(5, 40, 30, 20, "2", controller, "Phoenix");
    WishButton drei = new WishButton(35, 40, 30, 20, "3", controller, "Phoenix");
    WishButton vier = new WishButton(65, 40, 30, 20, "4", controller, "Phoenix");
    WishButton funf = new WishButton(95, 40, 30, 20, "5", controller, "Phoenix");
    WishButton sechs = new WishButton(125, 40, 30, 20, "6", controller, "Phoenix");
    WishButton sieben = new WishButton(155, 40, 30, 20, "7", controller, "Phoenix");
    WishButton acht = new WishButton(185, 40, 30, 20, "8", controller, "Phoenix");
    WishButton neun = new WishButton(215, 40, 30, 20, "9", controller, "Phoenix");
    WishButton zehn = new WishButton(245, 40, 30, 20, "10", controller, "Phoenix");
    WishButton j = new WishButton(275, 40, 30, 20, "J", controller, "Phoenix");
    WishButton q = new WishButton(305, 40, 30, 20, "Q", controller, "Phoenix");
    WishButton k = new WishButton(335, 40, 30, 20, "K", controller, "Phoenix");
    WishButton a = new WishButton(365, 40, 30, 20, "A", controller, "Phoenix");
    wish.add(zwei);
    wish.add(drei);
    wish.add(vier);
    wish.add(funf);
    wish.add(sechs);
    wish.add(sieben);
    wish.add(acht);
    wish.add(neun);
    wish.add(zehn);
    wish.add(j);
    wish.add(q);
    wish.add(k);
    wish.add(a);

    all2 = new WishButton[13];
    all2[0] = zwei;
    all2[1] = drei;
    all2[2] = vier;
    all2[3] = funf;
```

```
        all2[4] = sechs;
        all2[5] = sieben;
        all2[6] = acht;
        all2[7] = neun;
        all2[8] = zehn;
        all2[9] = j;
        all2[10] = q;
        all2[11] = k;
        all2[12] = a;
    }

    //Fenster mit Nachrichten und Fehlermeldungen
    public void message(String text) {
        View message = new View(400, 100);
        message.setTitle("Error");
        Canvas canvas = message.getBackground();
        canvas.setTextSize(24);
        canvas.drawText(10, 50, text);
    }

    //Zuordnen der Spielernummern zu Gegener, Partner etc
    public void number(int playerNumber) {
        view.setTitle("Spieler " + Integer.toString(playerNumber));
        switch (playerNumber) {
            case 1:
                me = 1;
                part = 3;
                opp1 = 2;
                opp2 = 4;
                break;
            case 2:
                me = 2;
                part = 4;
                opp1 = 3;
                opp2 = 1;
                break;
            case 3:
                me = 3;
                part = 1;
                opp1 = 4;
                opp2 = 2;
                break;
            case 4:
                me = 4;
                part = 2;
                opp1 = 1;
                opp2 = 3;
                break;

        }
        me--;
        part--;
        opp1--;
        opp2--;

        pass[me] = mePass;
        pass[part] = partPass;
        pass[opp1] = opp1Pass;
        pass[opp2] = opp2Pass;

        finnished[me] = meFinnished;
        finnished[part] = partFinnished;
        finnished[opp1] = opp1Finnished;
        finnished[opp2] = opp2Finnished;

        cardsLeft[me] = myCardsLeft;
        cardsLeft[part] = partCardsLeft;
        cardsLeft[opp1] = opp1CardsLeft;
        cardsLeft[opp2] = opp2CardsLeft;

    }
}
```

10.6.2 Card

```
package ch.jeda.tichu;

public class Card implements Comparable<Card> {

    String deck = " - Kopie";       //Auswahl der Karten Bilder:
                                    //" - Kopie" für Originalkarten,
                                    //"" für selbst gemalte
    int value;

    String color;
```

```
    String image;
    String imageS;
    boolean isSelected;

    private boolean isPlayed;

    private Player player;

    private ClientController clientController;

    int id;

    Card(int id, int val, String col) {
        this.id = id;
        color = col;
        value = val;
        image = "res:cards/" + color + val + deck + ".png";      //Pfade für nicht gewählte
        imageS = "res:cardsSel/" + color + val + deck + ".png";  //und gewählte Karten
    }

    public String getString() {
        String s = color + value;
        return s;
    }

    @Override
    public int compareTo(Card t) {
        if (value < t.value) {
            return -1;
        } else if (value > t.value) {
            return 1;

        } else {
            return 0;
        }
    }

}
```

10.6.3 CardSelector

```
package ch.jeda.tichu;

import ch.jeda.event.*;
import java.util.*;

public class CardSelector implements PointerDownListener {

    ArrayList<Card> selected = new ArrayList<Card>();
    ArrayList<Card> cards;
    double height;
    double width;

    ClientController controller;

    CardSelector(ClientController controller) {
        this.controller = controller;
        height = controller.board.height;
        width = controller.board.width;
        cards = controller.myCards;
        controller.board.view.addEventListener(this);
    }

    //An- und abwählen von Karten
    @Override
    public void onPointerDown(PointerEvent pe) {
        double x = pe.getX();
        double y = pe.getY();
        if (y < height / 6) {
            int i = (int) (x / (width / 16));
            if (i < 14) {
                if (selected.contains(cards.get(i))) {
                    selected.remove(cards.get(i));
                    cards.get(i).isSelected = false;
                    selected.trimToSize();
                } else {
                    selected.add(cards.get(i));
                    cards.get(i).isSelected = true;
                    selected.trimToSize();
                }
            }
        }
        controller.board.draw();
    }
```

```
}
```

10.6.4 ClientCommunicator

```
package ch.jeda.tichu;

import ch.jeda.*;

import ch.jeda.event.*;

import java.util.Collections;

public class ClientCommunicator implements MessageReceivedListener {

    ClientController controller;

    TcpConnection connection;
    String servername = "Localhost";
    int port = 1616;

    ClientCommunicator(ClientController controller) {
        this.controller = controller;
        connection = new TcpConnection();
        connection.open(servername, port);
        Jeda.addEventListener(this);
    }

    //sendet eineNachricht an den Server
    public void send(String text) {
        connection.sendLine(controller.playerNumber + ":" + text);
    }

    @Override
    public void onMessageReceived(MessageEvent event) {

        String[] parts = event.getLine().split(":");
        String message = parts[1];
        String mType = parts[0];
        String player = "";
        if (parts.length > 2) {
            player = parts[1];
            message = parts[2];
        }

        if (mType.equals("Cards")) {
            String[] ids = message.split(",");
            for (String s : ids) {
                int x = Integer.parseInt(s);
                controller.myCards.add(controller.cards[x]);
            }
            Collections.sort(controller.myCards);

            for (Card c : controller.myCards) {
                System.out.println(c.getString());
            }
            controller.board.draw();
        } else if (mType.equals("YourTurn")) {
            if (message.equals("true")) {
                System.out.println("playing");
                controller.isPlaying = true;
                String title = controller.board.view.getTitle();
                controller.board.view.setTitle(title + " playing");
                if (controller.board.pass[controller.board.opp1] || controller.board.finnished[control-
ler.board.opp1]) {
                    if (controller.board.pass[controller.board.opp2] || controller.board.finnished[control-
ler.board.opp2]) {
                        if (controller.board.pass[controller.board.part] || controller.board.finnished[control-
ler.board.part]) {
                            System.out.println("won");
                            send("Won:Round");
                            controller.board.draw();
                        }
                    }
                }

            } else if (message.equals("false")) {
                System.out.println("not playing");
                controller.isPlaying = false;
                String title = controller.board.view.getTitle();
                controller.board.view.setTitle(title.replaceAll(" playing", ""));
            }

        } else if (mType.equals("Finnished")) {
            int p = Integer.parseInt(message);
            controller.board.finnished[p - 1] = true;
```

```
} else if (mType.equals("Won")) {
    controller.board.pass[controller.board.opp1] = false;
    controller.board.pass[controller.board.opp2] = false;
    controller.board.pass[controller.board.part] = false;
    controller.board.pass[controller.board.me] = false;
    for (int i = 0; i < 4; i++) {
        controller.playedCards[i].clear();
        controller.playedCards[i].trimToSize();
    }
    controller.board.draw();
} else if (mType.equals("SchupfedCards")) {
    String[] ids = message.split(",");
    for (String s : ids) {
        int x = Integer.parseInt(s);
        controller.myCards.add(controller.cards[x]);
    }
    Collections.sort(controller.myCards);
    controller.board.draw();
} else if (mType.equals("Error")) {
    controller.board.message(message);
} else if (mType.equals("Passed")) {
    int p = Integer.parseInt(message);
    controller.playedCards[p - 1].clear();
    controller.board.pass[p - 1] = true;
    controller.board.draw();
} else if (mType.equals("Played")) {
    String[] ids = message.split(",");
    System.out.println(player);
    int p = Integer.parseInt(player);
    if (p == controller.playerNumber) {
        if (ids.length == 1) {
            if (ids[0].equals("53")) {
                controller.board.MahJong();
            }
        }
        for (String s : ids) {
            int x = Integer.parseInt(s);
            boolean remove = controller.myCards.remove(controller.cards[x]);
            if (remove == true) {
                System.out.println(controller.cards[x].getString() + " enternt");
            } else {
                System.out.println("fehler beim Entfernen von ");
                System.out.println(controller.cards[x].getString());
            }

        }

        controller.myCards.trimToSize();
        Collections.sort(controller.myCards);
        if (controller.myCards.isEmpty()) {
            controller.board.finnished[controller.board.me] = true;
            send("Finnished:now");
        }
        controller.board.draw();
    }
    controller.playedCards[p - 1].clear();
    for (String s : ids) {
        int x = Integer.parseInt(s);
        controller.playedCards[p - 1].add(controller.cards[x]);
        controller.board.cardsLeft[p - 1]--;
    }
    controller.playedCards1.trimToSize();
    Collections.sort(controller.playedCards1);
    for (int i = 0; i < 4; i++) {
        controller.board.pass[i] = false;
    }

    controller.board.draw();
} else if (mType.equals("RoundOver")) {
    String[] strings = message.split(",");
    controller.board.End(Integer.parseInt(strings[0]),
            Integer.parseInt(strings[1]), Integer.parseInt(strings[2]),
            Integer.parseInt(strings[3]));
} else if (mType.equals("Message")) {
    System.out.println(message);
    if (message.startsWith("Wollkommen Spieler nr.1")) {
        int x = 1;
        controller.playerNumber = x;
        System.out.println("playerNumber = " + x);
    } else if (message.startsWith("Wollkommen Spieler nr.2")) {
        int x = 2;
        controller.playerNumber = x;
        System.out.println("playerNumber = " + x);
    } else if (message.startsWith("Wollkommen Spieler nr.3")) {
        int x = 3;
        controller.playerNumber = x;
```

```
                System.out.println("playerNumber = " + x);
            } else if (message.startsWith("Wollkommen Spieler nr.4")) {
                int x = 4;
                controller.playerNumber = x;
                System.out.println("playerNumber = " + x);
            }
            controller.board.number(controller.playerNumber);
        } else {
            System.out.println("Unerwartete Meldung:" + event.getLine());
        }
    }

}
```

10.6.5 ClientController

```
package ch.jeda.tichu;

import ch.jeda.*;
import java.util.*;

public class ClientController extends Program {

    boolean isPlaying;

    boolean finnished;
    boolean partFinnished;
    boolean opp1Finnished;
    boolean opp2Finnished;

    int playerNumber;

    PlayButton play;
    PassButton pass;
    SchupfButton schupf;

    Card[] cards = new Card[56];

    ArrayList<Card> myCards = new ArrayList<Card>();

    ArrayList<Card> playedCards1 = new ArrayList<Card>();
    ArrayList<Card> playedCards2 = new ArrayList<Card>();
    ArrayList<Card> playedCards3 = new ArrayList<Card>();
    ArrayList<Card> playedCards4 = new ArrayList<Card>();

    ArrayList<Card>[] playedCards = (ArrayList<Card>[]) new ArrayList[4];

    Board board;

    ClientCommunicator communicator;

    CardSelector cardSelector;

    private Card card;

    PlayButton playButton;

    PassButton passButton;

    int wishedHeight;
    int wishedHeight2;

    @Override
    public void run() {

        board = new Board(this);
        cardSelector = new CardSelector(this);
        //Erstellen der Karten
        cards = new Card[56];
        String[] col = {"blue", "black", "red", "green"};
        int id = 0;
        for (String c : col) {
            for (int i = 0; i < 13; i++) {
                cards[id] = new Card(id, i + 2, c);
                id++;
            }
        }
        //Erstellen der Sonderkarten
        cards[id] = new Card(id, 15, "Dragon");
        cards[id + 1] = new Card(id + 1, 1, "MahJong");
        cards[id + 2] = new Card(id + 2, 0, "Dog");
        cards[id + 3] = new Card(id + 3, -1, "Phoenix");

        communicator = new ClientCommunicator(this);
```

```
        play = new PlayButton(board.cardW * 14, board.cardH * 2 / 3, board.cardW * 2, board.cardH / 3, "Spielen",
this);
        pass = new PassButton(board.cardW * 14, 0, board.cardW * 2, board.cardH / 3, "Passen", this);
        schupf = new SchupfButton(board.cardW * 14, board.cardH / 3, board.cardW * 2, board.cardH / 3, "Schupfen",
this);
        board.view.add(schupf);

        playedCards[0] = playedCards1;
        playedCards[1] = playedCards2;
        playedCards[2] = playedCards3;
        playedCards[3] = playedCards4;

    }

}
```

10.6.6 Combination

```
package ch.jeda.tichu;

import java.util.*;

public class Combination {

    String type;

    int value;

    int length = 0;

    ArrayList<Card> cards;

    Player playedBy;

    Player player;

    Combination(String type, int value, ArrayList<Card> cards, Player playedBy, int length) {
        this.cards = cards;
        this.playedBy = playedBy;
        this.type = type;
        this.value = value;
        this.length = length;
    }

}
```

10.6.7 Counter

```
package ch.jeda.tichu;

import java.util.*;

public class Counter {

    int pointsTeamA;

    int pointsTeamB;

    private ServerController controller;

    Counter(ServerController controller) {
        this.controller = controller;
    }

    public int count(ArrayList<Card> cards) {
        int i = 0;
        if (cards.isEmpty()) {
            return i;
        }
        for (Card c : cards) {
            switch (c.value) {
                case -1:
                    i = i - 25;
                    break;
                case 5:
                    i += 5;
                    break;
                case 10:
                    i += 10;
                    break;
                case 13:
                    i += 10;
                    break;
                case 15:
                    i += 25;
```

```
                    break;
                default:
                    break;
            }
        }
        return i;
    }

    public int calculate(int points) {
        int i = 100 - points;
        return i;
    }
}
```

10.6.8 Evaluator

```
package ch.jeda.tichu;

import java.util.*;

public class Evaluator {

    ServerController controller;

    Evaluator(ServerController controller) {
        this.controller = controller;

    }

    public Combination evaluate(ArrayList<Card> cards) {
        int n = cards.size();
        switch (n) {
            case 1:
                if (controller.currentComb != null) {
                    if (controller.currentComb.cards.size() == 1
                            && controller.currentComb.cards.get(0).color.equals("MahJong")) {
                        if (cards.get(0).value == controller.MahJong) {
                            return test("SingleCard", cards.get(0).value, cards, 0);
                        } else {
                            controller.communicator.send(controller.currentPlayer.playerNumber,
                                    "Error:Du musst die gewünschte Karte spielen");
                            return null;
                        }
                    }
                    if(controller.currentComb.cards.size() == 1
                            && controller.currentComb.cards.get(0).color.equals("Phoenix")){
                        Combination comb = test("SingleCard", controller.currentComb.value+1,cards,0);
                        if(comb != null){
                            return new Combination("SingleCard", controller.currentComb.value,
                                    cards, controller.currentPlayer, 0);
                        }
                    }
                } else if (cards.get(0).color.equals("Phoenix")) {
                    if (controller.currentComb == null) {
                        return test("SingleCard", 0, cards, 0);
                    }
                    if (test("SingleCard", controller.currentComb.value + 1, cards, 0) != null) {
                        Combination comb = new Combination("SingleCard", controller.currentComb.value,
                                cards, controller.currentPlayer, 0);
                        return comb;
                    }
                }
                return test("SingleCard", cards.get(0).value, cards, 0);
            case 2:
                if (cards.get(0).value == cards.get(1).value) {
                    return test("Pair", cards.get(0).value, cards, 0);
                }
            case 3:
                if (cards.get(0).value == cards.get(1).value
                        && cards.get(1).value == cards.get(2).value) {
                    return test("Triplet", cards.get(0).value, cards, 0);
                }
            case 4:
                if (cards.get(0).value == cards.get(1).value
                        && cards.get(1).value == cards.get(2).value
                        && cards.get(2).value == cards.get(3).value) {
                    return test("Bomb", cards.get(0).value, cards, 0);
                } else if (cards.get(0).value == cards.get(1).value
                        && cards.get(2).value == cards.get(3).value) {
                    return test("Stair", cards.get(0).value, cards, 2);
                }
            case 5:
                if (cards.get(0).value == cards.get(1).value
                        && cards.get(1).value == cards.get(2).value
                        && cards.get(3).value == cards.get(4).value) {
```

```
                return test("FullHouse", cards.get(0).value, cards, 0);
            } else if (cards.get(0).value == cards.get(1).value
                    && cards.get(2).value == cards.get(3).value
                    && cards.get(3).value == cards.get(4).value) {
                return test("FullHouse", cards.get(2).value, cards, 0);
            }
        default:
            if (n > 4) {
                if ((n % 2) == 0) {
                    if (stair(cards)) {
                        return test("Stair", cards.get(0).value, cards, n);
                    }
                }
                if (street(cards)) {
                    if (streetbomb(cards)) {
                        return test("Streetbomb", cards.get(0).value, cards, n);
                    }
                    return test("Street", cards.get(0).value, cards, n);
                }
            }
    }
    return null;
}

boolean ok(String type, int value, int length) {
    if (controller.currentComb == null) {
        return true;
    } else if (controller.currentComb.type.equals("SingleCard")
            && controller.currentComb.cards.get(0).color.equals("Dog")) {
        return true;
    } else if ("Bomb".equals(type)) {
        if (!controller.currentComb.type.equals("Streetbomb")) {
            return !(controller.currentComb.type.equals(type)
                    && controller.currentComb.value >= value);
        } else {
            return false;
        }
    } else if ("Sreetbomb".equals(type)) {
        if (controller.currentComb.type.equals(type)) {
            if (controller.currentComb.length >= length) {
                return !(controller.currentComb.value >= value);
            }
        }
        return true;
    } else {
        return controller.currentComb.type.equals(type) && controller.currentComb.value < value;
    }
}

Combination test(String type, int value, ArrayList<Card> cards, int length) {
    if (ok(type, value, length)) {
        Combination comb = new Combination(type, value, cards, controller.currentPlayer, length);
        return comb;
    } else {
        return null;
    }
}

boolean stair(ArrayList<Card> cards) {
    for (int n = 0; n < cards.size(); n = n + 2) {
        if (cards.get(n).value != cards.get(n + 1).value) {
            return false;
        }
        if (n > 0) {
            if (cards.get(n).value == cards.get(n - 1).value) {
                return false;
            }
        }
    }
    return true;
}

boolean street(ArrayList<Card> cards) {
    for (int n = 0; n < cards.size() - 1; n++) {
        if (cards.get(n).value != cards.get(n + 1).value - 1) {
            return false;
        }
    }
    return true;
}

boolean streetbomb(ArrayList<Card> cards) {
    for (int n = 0; n < cards.size(); n++) {
        if (cards.get(n).value == cards.get(n + 1).value
                && cards.get(n).color.equals(cards.get(n + 1).color)) {
        } else {
```

```
                return false;
            }
        }
        return true;
    }

}
```

10.6.9 PassButton

```
package ch.jeda.tichu;

import ch.jeda.ui.*;

public class PassButton extends TextButton {

    ClientController controller;

    PassButton(double x, double y, double width, double height, String text, ClientController controller) {
        super(x, y, text, 0);
        this.controller = controller;
        this.setWidth(width);
        this.setHeight(height);

    }

    @Override
    public void clicked() {
        if (controller.isPlaying) {
            controller.communicator.send("Pass: ");
        }
    }

}
```

10.6.10 PlayButton

```
package ch.jeda.tichu;

import ch.jeda.ui.*;
import java.util.*;

public class PlayButton extends TextButton {

    ClientController controller;
    boolean phoenix;
    boolean dragon;

    PlayButton(double x, double y, double width, double height, String text, ClientController controller) {
        super(x, y, text, 0);
        this.controller = controller;
        this.setWidth(width);
        this.setHeight(height);

    }

    @Override
    public void clicked() {
        String s = "Play:";
        if (controller.cardSelector.selected.isEmpty()) {
            controller.board.message("Du musst mind. 1 Karte wählen");
        } else {
            Collections.sort(controller.cardSelector.selected);
            for (Card c : controller.cardSelector.selected) {
                s = s + c.id + ",";
                c.isSelected = false;
                if (c.color.equals("Phoenix")) {
                    phoenix = true;
                } else if (c.color.equals("Dragon")) {
                    dragon = true;
                }
            }
            s = s.substring(0, s.length() - 1);
            if (phoenix) {
                if (controller.cardSelector.selected.size() != 1) {
                    controller.board.Phoenix(s);
                }
                phoenix = false;
            }
            if (dragon) {
                if (controller.cardSelector.selected.size() == 1) {
                    controller.board.Dragon();
                }
                dragon = false;
                controller.communicator.send(s);
```

```
            } else {
                controller.communicator.send(s);
            }
            controller.cardSelector.selected.clear();
            System.out.println(s);
            controller.board.draw();
        }
    }
}
```

10.6.11 Player

```
package ch.jeda.tichu;

import java.util.*;

public class Player {

    int playerNumber;

    ArrayList<Card> cards;

    ArrayList<Card> wonCards;

    boolean finished;

    boolean announcedTichu;

    boolean announcedBigTichu;

    Combination lastPlayed;

    Combination combination;

    Card card;

    ServerController controller;

    Player(ServerController controller, int number) {
        cards = new ArrayList<Card>();
        this.controller = controller;
        playerNumber = number;
        wonCards = new ArrayList<Card>();
    }

    public void play() {

    }

    public void pass() {

    }

}
```

10.6.12 SchupfButton

```
package ch.jeda.tichu;

import ch.jeda.ui.*;

public class SchupfButton extends TextButton {

    ClientController controller;

    SchupfButton(double x, double y, double width, double height, String text, ClientController controller) {
        super(x, y, text, 0);
        this.controller = controller;
        this.setWidth(width);
        this.setHeight(height);

    }

    @Override
    public void clicked() {
        if (controller.cardSelector.selected.size() == 3) {
            String s = "SchupfCards:";
            for (Card c : controller.cardSelector.selected) {
                s = s + c.id + ",";
                controller.myCards.remove(c);
            }
            controller.cardSelector.selected.clear();
            s = s.substring(0, s.length() - 1);
            controller.communicator.send(s);
            controller.board.draw();
```

```
            System.out.println(s);
            controller.board.view.remove(this);
            controller.board.view.add(controller.play);
            controller.board.view.add(controller.pass);
        } else if (controller.cardSelector.selected.size() < 3) {
            controller.board.message("Du musst 3 karten Schupfen");
        } else if (controller.cardSelector.selected.size() > 3) {
            controller.board.message("Du kannst nur 3 karten Schupfen");
        }

    }

}
```

10.6.13 ServerCommunicator

```
package ch.jeda.tichu;

import ch.jeda.*;
import ch.jeda.event.*;
import java.util.ArrayList;
import java.util.Collections;

public class ServerCommunicator implements MessageReceivedListener,
        ConnectionAcceptedListener {

    ServerController controller;

    TcpServer server;
    ArrayList<Connection> connections = new ArrayList<Connection>();

    String p1 = "SchupfedCards:k1,k2,k3";
    String p2 = "SchupfedCards:k1,k2,k3";
    String p3 = "SchupfedCards:k1,k2,k3";
    String p4 = "SchupfedCards:k1,k2,k3";

    int schupfed = 0;

    ServerCommunicator(ServerController controller) {
        this.controller = controller;
        startServer(1616);
        Jeda.addEventListener(this);

    }

    public void startServer(int port) {

        server = new TcpServer();
        if (server.start(port)) {
            System.out.println("Server gestarted");
        }
    }
    //sendet eine Nachricht an einen Client
    public void send(int n, String text) {
        if (text.equals("YourTurn:true")) {
            if (controller.players[n - 1].finished) {
                if (n == 4) {
                    n = 1;
                } else {
                    n++;
                }
                switch (n) {
                    case 1:
                        controller.currentPlayer = controller.player1;
                        break;
                    case 2:
                        controller.currentPlayer = controller.player2;
                        break;
                    case 3:
                        controller.currentPlayer = controller.player3;
                        break;
                    case 4:
                        controller.currentPlayer = controller.player4;
                        break;

                }
                send(n, text);
                return;
            }
        }
        connections.get(n - 1).sendLine(text);
    }

    @Override
    public void onMessageReceived(MessageEvent event) {
```

```
String message = event.getLine().split(":")[2];
String mType = event.getLine().split(":")[1];
String sender = event.getLine().split(":")[0];

if (mType.equals("SchupfCards")) {

    String[] ids = message.split(",");
    int[] iIds = new int[3];
    for (int i = 0; i < 3; i++) {
        iIds[i] = Integer.parseInt(ids[i]);
    }
    switch (Integer.parseInt(sender)) {
        case 1:
            controller.player4.cards.add(controller.cards[iIds[0]]);
            controller.player2.cards.add(controller.cards[iIds[1]]);
            controller.player3.cards.add(controller.cards[iIds[2]]);
            p4 = p4.replaceFirst("k2", ids[0]);
            p2 = p2.replaceFirst("k1", ids[1]);
            p3 = p3.replaceFirst("k3", ids[2]);
            controller.player1.cards.remove(controller.cards[iIds[0]]);
            controller.player1.cards.remove(controller.cards[iIds[1]]);
            controller.player1.cards.remove(controller.cards[iIds[2]]);
            break;
        case 2:
            controller.player1.cards.add(controller.cards[iIds[0]]);
            controller.player3.cards.add(controller.cards[iIds[1]]);
            controller.player4.cards.add(controller.cards[iIds[2]]);
            p1 = p1.replaceFirst("k2", ids[0]);
            p3 = p3.replaceFirst("k1", ids[1]);
            p4 = p4.replaceFirst("k3", ids[2]);
            controller.player2.cards.remove(controller.cards[iIds[0]]);
            controller.player2.cards.remove(controller.cards[iIds[1]]);
            controller.player2.cards.remove(controller.cards[iIds[2]]);
            break;
        case 3:
            controller.player2.cards.add(controller.cards[iIds[0]]);
            controller.player4.cards.add(controller.cards[iIds[1]]);
            controller.player1.cards.add(controller.cards[iIds[2]]);
            p2 = p2.replaceFirst("k2", ids[0]);
            p4 = p4.replaceFirst("k1", ids[1]);
            p1 = p1.replaceFirst("k3", ids[2]);
            controller.player3.cards.remove(controller.cards[iIds[0]]);
            controller.player3.cards.remove(controller.cards[iIds[1]]);
            controller.player3.cards.remove(controller.cards[iIds[2]]);
            break;
        case 4:
            controller.player3.cards.add(controller.cards[iIds[0]]);
            controller.player1.cards.add(controller.cards[iIds[1]]);
            controller.player2.cards.add(controller.cards[iIds[2]]);
            p3 = p3.replaceFirst("k2", ids[0]);
            p1 = p1.replaceFirst("k1", ids[1]);
            p2 = p2.replaceFirst("k3", ids[2]);
            controller.player4.cards.remove(controller.cards[iIds[0]]);
            controller.player4.cards.remove(controller.cards[iIds[1]]);
            controller.player4.cards.remove(controller.cards[iIds[2]]);
            break;
    }
    schupfed++;
    if (schupfed == 4) {
        send(1, p1);
        send(2, p2);
        send(3, p3);
        send(4, p4);
        controller.currentPlayer = controller.findStarter();
        send(controller.currentPlayer.playerNumber, "YourTurn:true");
    }
} else if (mType.equals("Play")) {
    int n = Integer.parseInt(sender);
    if (n == controller.currentPlayer.playerNumber) {
        ArrayList<Card> played = new ArrayList<Card>();
        String[] ids = message.split(",");
        for (String s : ids) {
            int x = Integer.parseInt(s);
            played.add(controller.cards[x]);
        }
        Collections.sort(played);
        if (played.get(0).color.equals("Phoenix")) {
            played.get(0).value = controller.Phoenix;
        }
        Combination comb = controller.evaluator.evaluate(played);
        if (comb == null) {
            send(n, "Error:Falsche oder zu tiefe Kombination");
        } else {
            comb.player = controller.players[n - 1];
            controller.currentComb = comb;
            controller.combinations.add(comb);
```

```
            String cards = "Played:" + n + ":";
            for (Card c : played) {
                cards = cards + c.id + ",";
            }
            cards = cards.substring(0, cards.length() - 1);
            for (int i = 1; i < 5; i++) {
                send(i, cards);
            }
            switch (n) {
                case 1:
                    for (Card c : played) {
                        controller.player1.cards.remove(c);
                    }
                    if (comb.cards.get(0).color.equals("Dog")) {
                        controller.currentPlayer = controller.player3;
                        send(3, "YourTurn:true");
                    } else {
                        controller.currentPlayer = controller.player2;
                        send(2, "YourTurn:true");
                    }
                    send(1, "YourTurn:false");
                    break;

                case 2:
                    for (Card c : played) {
                        controller.player2.cards.remove(c);
                    }
                    if (comb.cards.get(0).color.equals("Dog")) {
                        controller.currentPlayer = controller.player4;
                        send(4, "YourTurn:true");
                    } else {
                        controller.currentPlayer = controller.player3;
                        send(3, "YourTurn:true");
                        send(3, "YourTurn:true");
                    }
                    send(2, "YourTurn:false");
                    break;
                case 3:
                    for (Card c : played) {
                        controller.player3.cards.remove(c);
                    }
                    if (comb.cards.get(0).color.equals("Dog")) {
                        controller.currentPlayer = controller.player1;
                        send(1, "YourTurn:true");
                    } else {
                        controller.currentPlayer = controller.player4;
                        send(4, "YourTurn:true");
                    }
                    send(3, "YourTurn:false");
                    break;
                case 4:
                    for (Card c : played) {
                        controller.player4.cards.remove(c);
                    }
                    if (comb.cards.get(0).color.equals("Dog")) {
                        controller.currentPlayer = controller.player2;
                        send(2, "YourTurn:true");
                    } else {
                        controller.currentPlayer = controller.player1;
                        send(1, "YourTurn:true");
                    }
                    send(4, "YourTurn:false");
                    break;
            }

        }
    } else {
        send(n, "Error:Du bist nicht an der Reihe");
    }
} else if (mType.equals("Pass")) {
    int n = Integer.parseInt(sender);
    boolean pass = true;
    if (n == controller.currentPlayer.playerNumber) {

        if (controller.currentComb.cards.size() == 1
                && controller.currentComb.cards.get(0).color.equals("MahJong")) {
            for (Card c : controller.players[n - 1].cards) {
                if (c.value == controller.MahJong) {
                    pass = false;
                    send(n, "Error:Du musst die gewünschte Karte spielen");
                }
            }
        }
        if (pass) {
            for (int i = 1; i < 5; i++) {
```

```
                send(i, "Passed:" + n);
            }
            switch (n) {
                case 1:
                    controller.currentPlayer = controller.player2;
                    send(2, "YourTurn:true");
                    send(1, "YourTurn:false");
                    break;
                case 2:
                    controller.currentPlayer = controller.player3;
                    send(3, "YourTurn:true");
                    send(2, "YourTurn:false");
                    break;
                case 3:
                    controller.currentPlayer = controller.player4;
                    send(4, "YourTurn:true");
                    send(3, "YourTurn:false");
                    break;
                case 4:
                    controller.currentPlayer = controller.player1;
                    send(1, "YourTurn:true");
                    send(4, "YourTurn:false");
                    break;
            }
        }

    } else {
        send(n, "Error:Du bist nicht an der Reihe");
    }
} else if (mType.equals("Finnished")) {
    int n = Integer.parseInt(sender);
    if (!controller.finnished.isEmpty()) {
        if (controller.finnished.size() == 1) {
            if (n < 3) {
                if (controller.players[n + 1].playerNumber == controller.finnished.get(0).playerNumber) {
                    controller.doubleWin = true;
                }
            } else if (controller.players[n - 3].playerNumber == controller.finnished.get(0).playerNumber) {
                controller.doubleWin = true;
            }
        } else if (controller.finnished.size() == 2) {
            controller.roundOver = true;
        }
    }
    switch (n) {
        case 1:
            controller.player1.finished = true;
            controller.finnished.add(controller.player1);
            break;
        case 2:
            controller.player2.finished = true;
            controller.finnished.add(controller.player2);
            break;
        case 3:
            controller.player3.finished = true;
            controller.finnished.add(controller.player3);
            break;
        case 4:
            controller.player4.finished = true;
            controller.finnished.add(controller.player4);
            break;

    }
    for (int i = 1; i < 5; i++) {
        send(i, "Finnished:" + n);
    }
    if (controller.roundOver) {
        int x = 0;
        for (Player p : controller.players) {
            if (!p.finished) {
                for (Card c : p.wonCards) {
                    controller.finnished.get(0).wonCards.add(c);
                }
                if (p.playerNumber > 2) {
                    for (Card c : p.cards) {
                        controller.players[p.playerNumber - 3].wonCards.add(c);
                    }
                }
                if (p.playerNumber < 3) {
                    for (Card c : p.cards) {
                        controller.players[p.playerNumber + 1].wonCards.add(c);
                    }
                }
            }
            if (p.playerNumber == 1 || p.playerNumber == 3) {
                x = x + controller.counter.count(p.wonCards);
```

```
                    }
                }
                controller.counter.pointsTeamA += x;
                int y = controller.counter.calculate(x);
                controller.counter.pointsTeamB += y;
                for (int i = 1; i < 5; i++) {
                    send(i, "RoundOver:" + x + "," + y + "," + controller.counter.pointsTeamA + "," + control-
ler.counter.pointsTeamB);
                }
            } else if (controller.doubleWin) {
                int x = 0;
                int y = 0;
                if (n == 1 || n == 3) {
                    x = 200;
                } else if (n == 2 || n == 4) {
                    y = 200;
                }
                controller.counter.pointsTeamB += y;
                controller.counter.pointsTeamA += x;
                for (int i = 1; i < 5; i++) {
                    send(i, "RoundOver:" + x + "," + y + "," + controller.counter.pointsTeamA + "," + control-
ler.counter.pointsTeamB);
                }
            }
        } else if (mType.equals("Won")) {
            int n = Integer.parseInt(sender);
            int x = 0;
            int p;
            if (n == 4) {
                x = 1;
            } else {
                x = n + 1;
            }
            if (controller.currentComb.player.playerNumber == x || controller.players[x - 1].finished) {
                System.out.println("test");
            } else {
                if (controller.currentComb.type.equals("SingleCard") && controller.currentComb.cards.get(0).co-
lor.equals("Dragon")) {
                    p = controller.Dragon;
                    System.out.println("Drache wurde Spieler " + p + " gegeben.");
                } else {
                    p = n;
                }
                for (Combination comb : controller.combinations) {
                    for (Card c : comb.cards) {
                        controller.players[p - 1].wonCards.add(controller.cards[c.id]);
                    }
                }
                controller.combinations.clear();
                controller.currentComb = null;
                for (int i = 1; i < 5; i++) {
                    send(i, "Won:" + n);
                }
            }

        } else if (mType.equals("Phoenix")) {
            controller.Phoenix = Integer.parseInt(message);
        } else if (mType.equals("MahJong")) {
            controller.MahJong = Integer.parseInt(message);
            for (int i = 1; i < 5; i++) {
                send(i, "Error:" + controller.MahJong + " wurde gewünscht");
            }
        } else if (mType.equals("Dragon")) {
            controller.Dragon = Integer.parseInt(message);
        } else {
            System.out.println("Unerwartete Meldung:" + event.getLine());
        }
    }

    @Override
    public void onConnectionAccepted(ConnectionEvent event) {
        if (4 <= connections.size()) {
            event.getConnection().sendLine("Message:Es sind bereits vier Spieler in diesem Spiel");
            event.getConnection().close();
        }
        connections.add(event.getConnection());
        int x = connections.size();
        event.getConnection().sendLine("Message:Wollkommen Spieler nr." + x);
        System.out.println("verbunden");
        if (connections.size() == 4) {
            controller.mix();
        }
    }

}
```

10.6.14 ServerController

```
package ch.jeda.tichu;

import ch.jeda.ui.*;
import java.util.*;

public class ServerController {

    Combination currentComb;

    Player currentPlayer;

    ArrayList<Combination> combinations;

    ArrayList<Player> finnished = new ArrayList<Player>();

    Evaluator evaluator;

    Counter counter;

    Canvas canvas;

    ServerMain main;

    Player player1;
    Player player2;
    Player player3;
    Player player4;

    Player[] players = new Player[4];

    boolean p1Schupfed;
    boolean p2Schupfed;
    boolean p3Schupfed;
    boolean p4Schupfed;

    Card[] cards;

    int MahJong;
    int Phoenix;
    int Dragon;

    boolean roundOver;
    boolean doubleWin;

    ServerCommunicator communicator;

    View view;

    protected ServerController(ServerMain main) {
        this.main = main;
        communicator = new ServerCommunicator(this);
        evaluator = new Evaluator(this);
        combinations = new ArrayList<Combination>();
        counter = new Counter(this);

        view = new View(10, 10);
    }

    public void mix() {
        player1 = new Player(this, 1);
        player2 = new Player(this, 2);
        player3 = new Player(this, 3);
        player4 = new Player(this, 4);

        players[0] = player1;
        players[1] = player2;
        players[2] = player3;
        players[3] = player4;
        //Karten werden erstellt
        cards = new Card[56];
        String[] col = {"blue", "black", "red", "green"};
        int id = 0;
        for (String c : col) {
            for (int i = 0; i < 13; i++) {
                cards[id] = new Card(id, i + 2, c);
                id++;
            }
        }
        //Sonderkarten werden erstellt
        cards[id] = new Card(id, 15, "Dragon");
        cards[id + 1] = new Card(id + 1, 1, "MahJong");
        cards[id + 2] = new Card(id + 2, 0, "Dog");
        cards[id + 3] = new Card(id + 3, -1, "Phoenix");
```

```
    //verteilt die Karten zufällig auf die Spieler
    ArrayList<Card> used = new ArrayList<Card>();
    for (int n = 0; n < 14; n++) {
        Card c = cards[(int) (Math.random() * 56)];
        while (used.contains(c)) {
            c = cards[(int) (Math.random() * 56)];
        }
        player1.cards.add(c);
        used.add(c);
    }
    for (int n = 0; n < 14; n++) {
        Card c = cards[(int) (Math.random() * 56)];
        while (used.contains(c)) {
            c = cards[(int) (Math.random() * 56)];
        }
        player2.cards.add(c);
        used.add(c);
    }
    for (int n = 0; n < 14; n++) {
        Card c = cards[(int) (Math.random() * 56)];
        while (used.contains(c)) {
            c = cards[(int) (Math.random() * 56)];
        }
        player3.cards.add(c);
        used.add(c);
    }
    for (Card c : cards) {
        if (used.contains(c)) {

        } else {
            player4.cards.add(c);
            used.add(c);
        }
    }
    //Karten werden den Clients zugesendet
    String p1 = "Cards:";
    String p2 = "Cards:";
    String p3 = "Cards:";
    String p4 = "Cards:";

    for (Card c : player1.cards) {
        p1 = p1 + c.id + ",";
    }
    for (Card c : player2.cards) {
        p2 = p2 + c.id + ",";
    }
    for (Card c : player3.cards) {
        p3 = p3 + c.id + ",";
    }
    for (Card c : player4.cards) {
        p4 = p4 + c.id + ",";
    }
    p1 = p1.substring(0, p1.length() - 1);
    p2 = p2.substring(0, p2.length() - 1);
    p3 = p3.substring(0, p3.length() - 1);
    p4 = p4.substring(0, p4.length() - 1);

    communicator.send(1, p1);
    communicator.send(2, p2);
    communicator.send(3, p3);
    communicator.send(4, p4);

    System.out.println(p1);
    System.out.println(p2);
    System.out.println(p3);
    System.out.println(p4);

}

//findet Spieler, der den Mah Jong auf der Hand hat
Player findStarter() {
    for (Card c : player1.cards) {
        if (c.id == 53) {
            return player1;
        }
    }
    for (Card c : player2.cards) {
        if (c.id == 53) {
            return player2;
        }
    }
    for (Card c : player3.cards) {
        if (c.id == 53) {
            return player3;
        }
```

```
        }
        for (Card c : player4.cards) {
            if (c.id == 53) {
                return player4;
            }
        }
        return null;
    }
}
```

10.6.15 ServerMain

```
package ch.jeda.tichu;

import ch.jeda.*;

public class ServerMain extends Program {

    public int playerNumber;

    private ServerController controller;

    @Override
    public void run() {
        controller = new ServerController(this);
    }
}
```

10.6.16 WishButton

```
package ch.jeda.tichu;

import ch.jeda.ui.*;

public class WishButton extends TextButton {

    ClientController controller;
    String value;
    String card;

    WishButton(double x, double y, double width, double height, String text,
               ClientController controller, String card) {
        super(x, y, text, 0);
        this.controller = controller;
        this.setWidth(width);
        this.setHeight(height);
        value = text;
        this.card = card;

    }

    @Override
    public void clicked() {
        //Falls für Mah Jong Wunsch verwendet wird
        if (card.equals("MahJong")) {
            if (value.equals("J")) {
                controller.wishedHeight = 11;
            } else if (value.equals("Q")) {
                controller.wishedHeight = 12;
            } else if (value.equals("K")) {
                controller.wishedHeight = 13;
            } else if (value.equals("A")) {
                controller.wishedHeight = 14;
            } else if (Integer.parseInt(value) < 11) {
                controller.wishedHeight = Integer.parseInt(value);
            }
            controller.board.wish.remove(controller.board.all);
            controller.communicator.send("MahJong:" + controller.wishedHeight);
        } //Falls für Phönix höhe verwendet wird
        else if (card.equals("Phoenix")) {
            if (value.equals("J")) {
                controller.wishedHeight2 = 11;
            } else if (value.equals("Q")) {
                controller.wishedHeight2 = 12;
            } else if (value.equals("K")) {
                controller.wishedHeight2 = 13;
            } else if (value.equals("A")) {
                controller.wishedHeight2 = 14;
            } else if (Integer.parseInt(value) < 11) {
                controller.wishedHeight2 = Integer.parseInt(value);
            }
            controller.board.wish.remove(controller.board.all2);
            controller.communicator.send("Phoenix:" + controller.wishedHeight2);
            controller.communicator.send(controller.board.s);
        } //Falls für Drachen weggeben vrwendet wird
```

```
        else if (card.equals("Dragon")) {
            int x = 0;
            if (value.equals("Rechts")) {
                x = controller.board.opp1 + 1;
            }
            if (value.equals("Links")) {
                x = controller.board.opp2 + 1;
            }
            controller.communicator.send("Dragon:" + x);
            controller.board.wish.remove(controller.board.all3);
            System.out.println("x = " + x);
        }
        controller.board.wish.setTitle(value + " gewählt");
    }
}
```

10.7 Selbstständigkeitserklärung

gym | NEUFELD

Selbständigkeitserklärung Maturaarbeit

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Das Informationsblatt «Plagiatserkennung» ist mir bekannt und somit auch die Konsequenzen eines Teil- oder Vollplagiats.

Ort und Datum: _____

Unterschrift des Verfassers: _____