Inhaltsverzeichnis

[1. Abstract 2](#_Toc527613485)

[2. Zielsetzung 2](#_Toc527613486)

[3. Vorwort 2](#_Toc527613487)

[4. Theorie 3](#_Toc527613488)

[4.1 TCP/IP 3](#_Toc527613489)

[4.2 Client-Server-Kommunikation 3](#_Toc527613490)

[4.3 Delegations-Eventmodell 4](#_Toc527613491)

[5. Tichu 5](#_Toc527613492)

[5.1 Die Kombinationen 5](#_Toc527613493)

[5.2 Die Sonderkarten 5](#_Toc527613494)

[5.3 Ablauf 7](#_Toc527613495)

[5.4 Ende der Runde 7](#_Toc527613496)

[6. Vorgehen 8](#_Toc527613497)

[7. Design 8](#_Toc527613498)

[8. Implementierung 9](#_Toc527613499)

[8.1 Kommunikation 9](#_Toc527613500)

[8.2 Spiellogik 11](#_Toc527613501)

[8.3 Grafik 11](#_Toc527613502)

[9. Testing 12](#_Toc527613503)

[10. Fazit 13](#_Toc527613504)

[11. Anhang 14](#_Toc527613505)

[11.1 Glossar 14](#_Toc527613506)

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.

Des Weiteren werden in der vorliegenden Arbeit die Regeln des Kartenspiels Tichu erklärt, die Vorgehensweise nach dem Wasserfallmodell, welches in dieser Arbeit umgesetzt wurde, erläutert, der Ablauf der Planung der Software beschrieben, die Implementierung der wichtigsten Programmelemente wie der Kommunikation, der Spiellogik und der Grafik erklärt und die Testszenarien, nach welchen das Programm getestet wurde beschrieben.

2. Zielsetzung

Das Ziel der vorliegenden 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. Vorwort

4. Theorie

4.1 TCP/IP

TCP/IP steht für Transmission Control Protocol/Internet Protocol und ist eine Familie von Netzwerkprotokollen. Wegen ihrer Wichtigkeit für das Internet, wir 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]](#footnote-1) «Kommunikation wird in Rechnernetzen durch Netzwerkprotokolle umgesetzt und in der Praxis in funktionale Schichten (*layer*) unterteilt.»[[2]](#footnote-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. Das TCP/IP-Referenzmodell besteht 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 |

4.2 Client-Server-Kommunikation

Ein Bild, das Text enthält.

Mit sehr hoher Zuverlässigkeit generierte BeschreibungWann immer 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]](#footnote-3)

Abbildung : Beispiel eines Client-Server-Aufbaus

4.3 Delegations-Eventmodell

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. 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, was nur möglich ist, 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.

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]](#footnote-4)

5. 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.

5.1 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. |

5.2 Die Sonderkarten

|  |  |
| --- | --- |
| C:\Users\Nick\Documents\GitHub\Tichu-Maturaarbeit\TichuApp\res\cards\MahJong1 - Kopie.png5.2.1 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 an der Reihe ist, 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. |
| Dog0 - Kopie.png5.2.2 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. |
| Dragon15 - Kopie.png5.2.3 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. |
| Phoenix-1 - Kopie.png5.2.4 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. |

5.3 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.

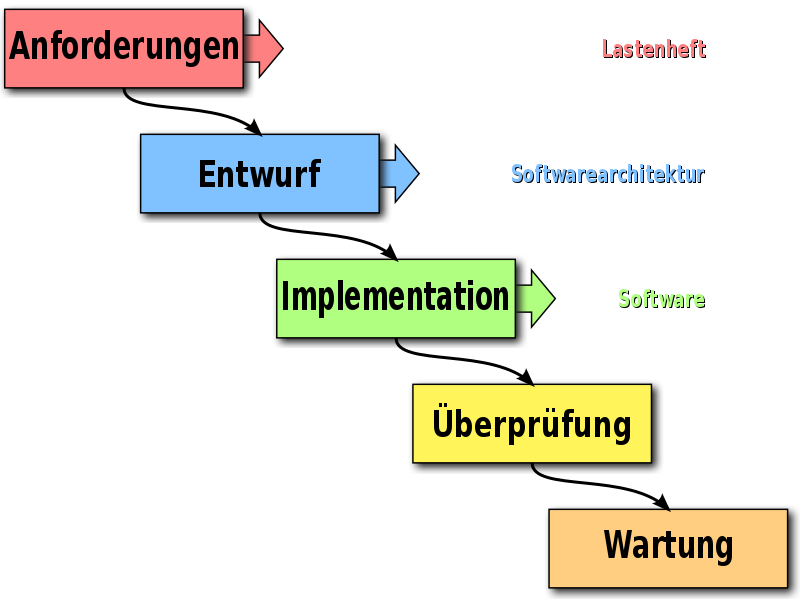
5.4 Ende der Runde

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]](#footnote-5)

6. 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. Ein 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 in Betrieb genommen wird.

Abbildung ???



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

7. 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.

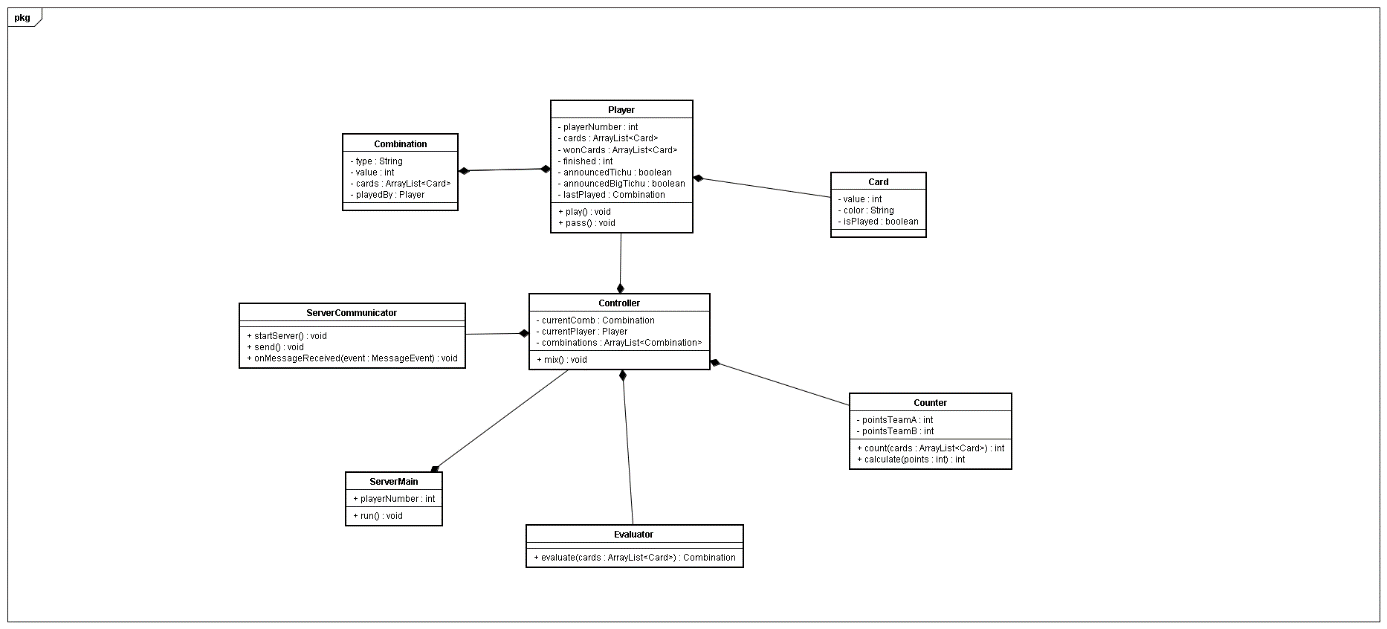


Abbildung : Beispiel eines Klassendiagramms

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.

8. Implementierung

8.1 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 Connectioin-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, sondern nur ihre IDs und die Anweisung, was damit gemacht werden soll, wodurch nicht noch die Objekte serialisiert werden müssen.

8.2 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.

8.3 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.

Beispielcode : 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.

9. Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Testobjekt** | **Testszenario** | **Erwartetes Verhalten** | **Ergebnis** | **Datum** |
| Mah Jong | Die zulezt 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 Felermeldung. |  |  |
| Mah Jong | Die zulezt 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. |  |  |
| 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. |  |  |
| Phönix | Der aktive Spieler möchte eine Einzelkarte zusammen mit dem Phönix (mit selbem Wert) als Paar spielen. | Das Paar wird akzeptiert. |  |  |
| 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. |  |  |
| Spielablauf | Ein Spieler der keine Karten mehr hat kommt an die Reihe. | Er wird übersprungen und der Spieler nach ihm ist an der Reihe. |  |  |
| Spielablauf | Drei Spieler haben keine Karten mehr und sind somit fertig. | Die Runde endet und die Punkte werden zusammengezählt. |  |  |
| Spielablauf | Zwei Spieler haben keine Karten mehr und sind aus demselben Team. | Die Runde endet und das Team, das fertig ist bekommt 200 Punkte. |  |  |
| 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 seien. |  |  |
| 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. |  |  |

10. 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 unterschätzt, wodurch der Terminplan gegen Ende nicht eingehalten werden konnte, was zu unnötigem Stress führte.. Eine weitere Verbesserungsmöglichkeit wäre, den Programmcode von Anfang an und durchgehend zu kommentieren. Dies würde helfen, den Programmcode verständlich zu machen und die 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.

11. Anhang

11.1 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:**

1. Wikipedia-Artikel Transmission Control Protocol/Internet Protocol: <https://de.wikipedia.org/wiki/Transmission_Control_Protocol/Internet_Protocol> [↑](#footnote-ref-1)
2. Wikipedia-Artikel Internetprotokollfamilie: <https://de.wikipedia.org/wiki/Internetprotokollfamilie> [↑](#footnote-ref-2)
3. Jedadokumentation Netzwerk-Kommunikation: <https://jeda.ch/wiki/doc:network> [↑](#footnote-ref-3)
4. Plüss, Ägidius: Java Exemplarisch <https://books.google.ch/books?id=DsTnBQAAQBAJ&pg=PT223&lpg=PT223&dq=Delegations+eventmodell&source=bl&ots=4uR7cGyJU-&sig=0RWeLqhB1XWCm_n5N4yzPkP4oBo&hl=de&sa=X&ved=2ahUKEwjenuuR0NjdAhXEDywKHf7cD-cQ6AEwAHoECAUQAQ#v=onepage&q=Delegations%20eventmodell&f=false> [↑](#footnote-ref-4)
5. Für weitere Inforationen siehe Tichu Anleitung: <http://www.fatamorgana.ch/tichu/ti_regel.pdf> [↑](#footnote-ref-5)