



EasySet

Dominik Schneider
Oliver Suchan

1 Inhalt

2	INFORMATIONEN ZUM PROJEKT	2
2.1	WAS IST SET?	2
2.2	WAS IST EASYSET?	5
2.3	DIE MATHEMATIK HINTER SET	5
3	INFORMATIONEN ZUM QUELLCODE	6
3.1	INFORMATIONEN ZU QTS SIGNALE & SLOTS	6
4	INFORMATIONEN ZUM NETWORKING	8
4.1	INFORMATIONEN ZUM FIELD_SYNCHRO-PAKET	9
5	INFORMATIONEN ZUM AUSGELIEFERTEN KOMPILAT	10
6	SOFTWARE VON DRITTANBIETERN	11
6.1	DOXYGEN	11
6.2	GITHUB	11
6.3	QT 5.4	12
7	GENESIS	12
8	EVOLUTION	13
9	FINALEM	16
9.1	DIE KLASSEN	16
9.2	WIE SPIELE ICH RICHTIG?	20
9.2.1	LOKALES SPIEL EINRICHTEN	20
9.2.2	ONLINE SPIEL	21
9.3	SPIELEN	23
10	QUELLEN	26
10.1	DOXYGEN	26
10.2	GITHUB	26
10.3	QT 5.4.0	26

2 Informationen zum Projekt

2.1 Was ist Set?

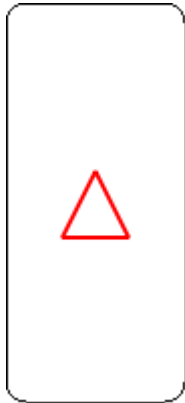
Bei Set handelt es sich um ein Spiel, genauer: ein Kartenspiel. Bei Set gibt es insgesamt 81 Karten mit jeweils verschiedenen Motiven. Diese Zahl 81 kommt daher zustande, dass es 81 unterschiedliche Kombinationen aus den jeweiligen Eigenschaften der Karten gibt. Eine Karte hat jeweils folgende Attribute:

- Farbe
 - Rot
 - Grün
 - Blau
- Form
 - Dreieck
 - Viereck
 - Kreis
- Anzahl
 - Eins
 - Zwei
 - Drei
- Einfärbung
 - Vollfarbig
 - Leer
 - Blass

Das heißt, jedes Attribut hat 3 verschiedene Möglichkeiten. Das Attribut Farbe hat zum Beispiel Rot, Grün und Blau also mögliche Werte. Rechnet man nun die Anzahl der möglichen Werte \wedge die Anzahl der Attribute, kommt man auf die 81 verschiedenen Karten. $3^4 = 81$

Wie sieht so eine Karte denn aus?

Eine Karte kann zum Beispiel folgendermaßen aussehen:



Die Werte dieser Karte lauten:

- Rot
- Dreieck
- Eins
- Leer

Am Anfang des Spiels werden 12 Karten der 81 Karten vom Stapel genommen und offen auf das Spielfeld gelegt. Die Spieler haben dann die Aufgabe, aus diesen Karten sogenannte Sets oder auch Pärchen zu finden.

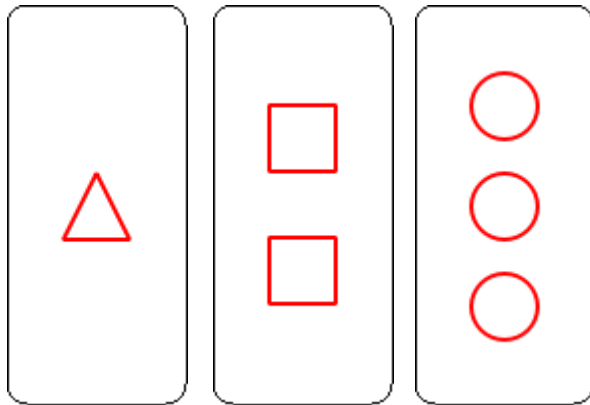
Ein Set besteht aus genau 3 Karten und kann so gebildet werden:

- Alle Karten haben dieselbe Anzahl an Symbolen oder jede hat eine andere Anzahl
- Alle Karten zeigen dasselbe Symbol oder jede zeigt ein anderes Symbol
- Die Symbole einer Karte haben dieselbe Farbe wie die der anderen Karten oder jede Karte hat eine andere Farbe
- Die Symbole einer Karte haben dieselbe Füllung wie die der anderen Karten oder jede Karte hat eine andere Füllung

Zusammengefasst in einem Satz: Drei Karten bilden genau dann ein Set, wenn keine der vier Eigenschaften bei genau zwei der drei Karten die gleiche Ausprägung erfährt.

(Quelle: Wikipedia)

Im Folgenden ein Beispiel für ein Set:



Die Drei Karten haben alle unterschiedliche Symbole, unterschiedliche Anzahlen an Symbolen, gleiche Farbe und die gleiche Füllung. Sprich, die Karten unterscheiden sich in nur 2 Eigenschaften und bilden daher ein Pärchen

Weiter geht's im Regelwerk. Nachdem ein Spieler ein Pärchen gefunden hat, werden 3 Karten nachgelegt. Die Karten die ein Set bilden, darf der Spieler an sich nehmen. Wird nach einer bestimmten Wartezeit kein Set gefunden, werden 3 weitere Karten hinzugelegt, sodass 15 Karten auf dem Spielfeld liegen. Sollte dann noch immer kein Set gefunden werden, werden wieder 3 Karten nachgelegt, usw.

Wird jedoch ein Set gefunden und es sind mehr als 12 Karten auf dem Spielfeld, so werden keine neue Karten nachgelegt, bis wieder 12 Karten auf dem Spielfeld liegen.

Meldet sich ein Spieler um ein Set anzusagen und das Set ist jedoch falsch, dann muss dieser aus seinem Stapel mit den Karten die er gewonnen hat, 3 Karten zurück ins Deck legen.

Das Spiel ist zu Ende, wenn alle Karten im Deck aufgebraucht wurden und auf dem Spielfeld kein Set mehr gefunden wird.

Der Gewinner ist derjenige Spieler, der die meisten Pärchen gefunden hat.

Das Spiel kann mit minimal 1 Spieler und maximal 8 Spielern gespielt werden.

2.2 Was ist EasySet?

EasySet ist der Name unseres Spiels. Das digitale Pendant zu dem Kartenspiel Set. Bei diesem Projekt besitzt der Präfix „*Easy*“ des Namens jedoch keinen tieferen Sinn, sondern galt lediglich der Konvention.

Nun denn, mit unserem Projekt ist es möglich, *Set* quasi auf jedem Gerät zu spielen und das auch noch mit anderen Personen. Das bedeutet, unser Spiel kann sowohl online als auch offline gespielt werden. Des Weiteren kann man mit mehreren Freunden (maximal 8 – vielleicht schon zu viel für einen ITler(?)) an einem Computer spielen. Aber natürlich kann der einsame Informatiker auch alleine spielen, umso selbst besser zu werden.

2.3 Die Mathematik hinter Set

Da unser Programm ja auch irgendwie überprüfen muss, ob die vom Spieler ausgewählten Karten ein Set bilden, haben wir Folgendes überlegt:

Wir stellen jede Karte als 4-dimensionalen Vektor dar, wobei jedes Attribut eine Koordinate repräsentiert. Diese Vektoren werden dann summiert und jede Koordinate wird Modulo 3 (Anzahl der möglichen Werte pro Attribut) gerechnet. Dadurch kommen wir jeweils auf einen Koordinatenwert der sich nur im Intervall $I = [0; 2]$ befindet (0, 1, 2, 0, 1, 2, 0, 1, 2,...).

Summieren wir nun die einzelnen Koordinaten des Ergebnisvektors auf und überprüfen diesen Wert auf 0, dann handelt es sich bei diesen 3 Karten um ein Set, andernfalls halt nicht.

Die verschiedenen Werte der Attribute werden natürlich von 0 bis 2 durchnummeriert. (Rot = 0, Grün = 1, usw.)

$$\begin{pmatrix} \text{Farbe} \\ \text{Form} \\ \text{Anzahl} \\ \text{Füllung} \end{pmatrix} \Rightarrow \begin{pmatrix} \text{Rot} = 0 \\ \text{Viereck} = 1 \\ \text{Eins} = 0 \\ \text{Vollfarbig} = 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$
$$= \begin{pmatrix} (0 + 1 + 2) \bmod 3 \\ (1 + 1 + 1) \bmod 3 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \bmod 3 \\ 3 \bmod 3 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$\Rightarrow 0 + 0 + 0 + 0 = 0 \Rightarrow$ Es handelt sich um ein Set

3 Informationen zum Quellcode

Das Programm ist in der Programmiersprache C++ (genauer: C++0x) programmiert. Anfänglich stand die Programmiersprache Java noch zur Auswahl, da Dominik noch nie davor in C++ programmiert hatte. Nichtsdestotrotz entschlossen wir uns für C++, was, wie sich herausstellte, ebenfalls kein Problem für Dominik darstellte.

3.1 Informationen zu Qts Signale & Slots

In diesem Projekt ist sehr oft eine Eigenheit des verwendeten Qt Frameworks aufzufinden. Die sogenannten Signale & Slots. Bei diesen Signalen & Slots handelt es sich im Prinzip um so genannte Callbacks, nur ein noch ein bisschen ausgeklügelter und des Weiteren wird auch noch Typsicherheit gewährt!

Deklariert man nun in einer Klasse ein Signal, so kann dieses Signal mit sogenannten Slots verbunden werden. Slots können zusätzlich wie normale Funktionen gehandhabt werden. Sprich, man kann sie ganz normal aufrufen.

Angenommen wir haben folgendes Signal und folgenden Slot:

packethandler.hpp:

```
signals:  
  
    void readField(QByteArray p_field);
```

window.hpp:

```
public slots:  
  
    void retrieveField(QByteArray p_field);
```

Dieses Signal kann nun folgendermaßen mit dem jenem Slot verbunden werden:

```
connect(m_packetHandler, &PacketHandler::readField, &Window::getInstance(),  
&Window::retrieveField);
```

Was bedeutet diese Verbindung denn nun?

Ganz einfach, wenn die Klasse *PacketHandler* das Signal *readField* mit dem Schlüsselwort *emit* nun emittiert, dann werden automatisch alle Slots die mit diesem Signal verbunden sind, ausgeführt – unter diesem Fall *retrieveField* in der Klasse *Window*.

```
void Window::clientDisconnected()  
{  
    static const void * sentBy = sender();  
    if(sentBy != sender())  
        return;  
    //Quellcode  
}
```

Trifft man auf einen Slot mit obiger Struktur, dann meint dies, dass ein Slot – der mehrmals mit Signalen verbunden ist – lediglich ein einziges Mal aufgerufen werden kann.

Der Aufruf der Methode *sender()* in einem Slot, liefert also das Objekt, das das Signal und somit den Slot aufgerufen hat.

4 Informationen zum Networking

Da dieses Projekt das erste für uns beide ist, wo wir Sockets verwenden, kann es natürlich sein, dass wir gegen die übliche Konvention verstoßen haben. Dennoch glauben wir, dass das was wir im Bezug aufs Networking zustande bekommen haben, auf jeden Fall akzeptabel ist.

Wie bereits gesagt, besteht die Möglichkeit einem Spiel beizutreten, das weder auf dem Rechner lokal noch im LAN läuft. Das heißt also: man kann sich zu externen Servern verbinden. Dieses Spiel kann jedoch nicht – wie es bei anderen Multiplayer-Spielen üblich ist – einfach nur die Aufgabe eines Servers besitzen. Theoretisch schon, jedoch wird durch den Einrichtungsassistenten auf jedem Computer genau eine GUI pro Anwendungsinstanz erzeugt. Jede Anwendungsinstanz erzeugt also einen Server und einen Client, weswegen der Server als ein Mitspieler gilt. Wir sahen es nicht als notwendig an, den Server-Computer manuell als Client einrichten zu müssen und taten dies somit implizit.

Im Allgemeinen haben unsere implementierten Pakete folgende Struktur

1 Byte 1 Byte x Byte
{Paket-Kopf}{Paket-Größe}{Daten}

Im Folgenden nun eine Tabelle aller von uns implementierten Pakete und deren Aufgabe:

Paketname	Paketkopf	Datenstruktur	Aufgabe
GAME_STATE	0x01	{1Byte} (1 oder 2)	Gibt den Status des Spiels an. Dieser kann „Spielstarten“ (1) oder „Spielbeenden“ (2) sein
DECK	0x02	{1Byte}	Gibt die Anzahl der im Deck befindlichen Karten an
INPUT_STATE	0x03	{1 Byte} (1 oder 2)	Gibt den Status der Eingabe an. 1 für „Entsperrt“ und 2 für „Gesperrt“
PLAYER_TURN	0x04	{0 Byte}	Gibt an, dass ein Spieler eine Eingabe tätigen möchte
SCORES	0x05	{[Spieleranzahl] Byte}	Gibt den Punktestand eines jeden Spielers an
FIELD_SYNCHRO	0x07	{[Spielfeldgröße] Byte}	Gibt das aktuelle Spielfeld an. Jede Karte wird zu einem <i>char</i> konvertiert
CLICK	0x0A	{3 Byte}	Gibt an, welche Karten angeklickt wurden

4.1 Informationen zum FIELD_SYNCHRO-Paket

Das Paket **FIELD_SYNCHRO** = 0x07 ist ein spezielles Paket, denn alle Karten auf dem Spielfeld werden zu jeweils einem Byte zusammengefasst.

Wie das funktioniert ist ganz einfach zu erklären.

Jede Karte besitzt Attribute, die jeweils die Werte $W = \{0, 1, 2\} = \{0b00, 0b01, 0b11\}$ (*0b* kennzeichnet die Zahl als Binärzahl).

Das heißt, jedes Attribut verwendet von den 8 Bit das ihm zur Verfügung steht nur 2 Bit, die restlichen sind immer null. Dies kann man sich zunutze machen, indem man jeweils die letzten beiden Bits aller Attribute durch *Links-Schieben* und *Oder* zu einem Byte formt. Das sieht dann ungefähr wie folgt aus:

```
short m_color = 0b01;
short m_shape = 0b01;
short m_number = 0b01;
short m_opacity = 0b01;
short resultVal = 0b00;

resultVal |= m_color << 6;
resultVal = 0b01000000;

resultVal |= m_shape << 4;
resultVal = 0b01010000;

resultVal |= m_number << 2;
resultVal = 0b01010100;

resultVal |= m_opacity;
resultVal = 0b01010101;
```

Nun stecken in unserer Variable *resultVal* alle 4 Variablen (*m_color*, *m_shape*, *m_number* und *m_opacity*). Das ist auch genau das, was die Funktion *attributesToByte* in unserer Klasse *Card* übernimmt.

```
char Card::attributesToByte()    // Attribute werden als Byte gespeichert
{
    return ((m_color << 6) | (m_shape << 4) | (m_number << 2) | m_opacity);
}
```

5 Informationen zum ausgelieferten Kompilat

Wie aus vorherigen Kapiteln erkennbar, verwenden wir das Framework Qt ([[kju:t](#)]). Doch es mag vielleicht aufgefallen sein, dass das erhaltene Kompilat im Vergleich zu sonstigen Qt-Anwendungen erstaunlich wenige DLLs (Dynamic Link Library) braucht. Dafür ist die Anwendung wesentlich größer als sonst, ganze ~17MB misst die Anwendung. Dies liegt lediglich daran, dass bei diesem Projekt ein sogenannter Qt static build statt einem dynamic build vorgenommen wurde. Bei einem static build wird Qt (da es ja opensource ist) direkt in das

Programm mithinein kompiliert. Nebenbei erwähnt durften wir sicherlich 1 Stunde lang warten, bis Qt endlich kompiliert und bereit für den static build war... Diese Art von Erstellung darf man laut Lizenz jedoch nur vornehmen, wenn das Programm komplett quelloffen zur Verfügung steht. Da dies bei uns der Fall war, konnten wir beruhigt diese Art der Kompilierung verwenden und wurden so die nervigen Bibliotheken, die sonst im Ordner der Anwendung rumgeschwirrt hätten, los.

6 Software von Drittanbietern

6.1 Doxygen

Doxygen ist das C++-Pendant zu *Javadoc*. Sprich, es generiert Webseiten anhand von dokumentiertem Quellcode. Aber auch kann es LaTeX-Dateien generieren, wenn man es denn richtig konfiguriert. Wie sich wohl erschließen lässt, haben wir diese Software verwendet, um unsere Webseiten-Dokumentation zu generieren. Natürlich mussten wir dennoch unseren Quellcode dokumentieren...

Unsere Webseiten-Dokumentation kann über folgenden Pfad aufgerufen werden:

„documentation/html/index.html“

6.2 GitHub

GitHub ist eine kostenlose Versionsverwaltungssoftware. Natürlich kam diese auch bei unserem Projekt zur Verwendung, damit wir unser Projekt synchron halten können und uns nicht in die Quere kommen. Wir wussten also genauso gut über die Arbeit des Partners Bescheid wie die NSA – sofern dieser seine Arbeit denn auch hochgeladen hat. Da wir nur auf die kostenlose Version von GitHub zurückgreifen konnten, blieb uns keine andere Wahl, als eine öffentliches Respository anzulegen, was dennoch kein Problem darstellte.

Unser Projekt ist unter folgender Adresse verfügbar:

<https://github.com/cranktec/EasySet/tree/reincarnation>

Unsere Hauptbranch ist bei diesem Repository „reincarnation“ wie sonst „master“. Dies liegt lediglich daran, dass wir vergessen hatten, den Master-Zweig und den Reincarnation-Zweig zusammenzuführen und gegen Ende schlichtweg keine Lust mehr hatten.

Warum der Zweig den Namen „Wiedergeburt“ trägt, werden Sie im Verlauf erfahren.

6.3 Qt 5.4

Wie bereits gespoilert wurde, verwendet dieses Projekt auch wieder das kostenlose, quelloffene Framework Qt. Dieses Mal jedoch eine neuere Version, nämlich 5.4. (Wobei man diese Änderungen in unserem Fall quasi gar nicht mitbekommt...)

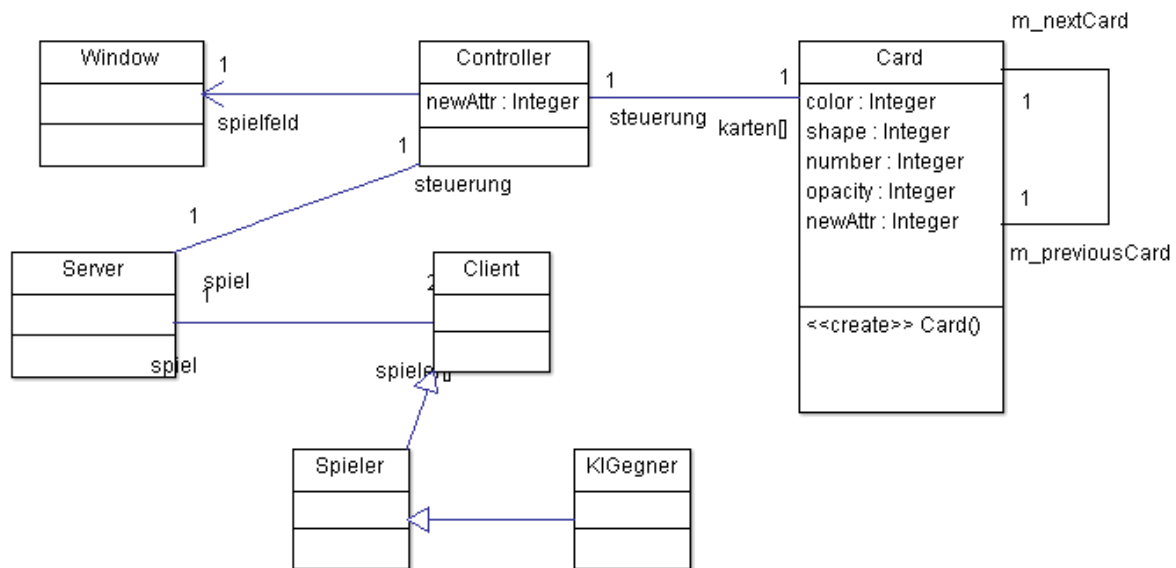
Selbstverständlich handelt es sich bei Qt um ein Framework und blablablabla... Das wissen Sie ja schon alles.

7 Genesis

Gott sprach es werde EasySet und es ward EasySet.

Hier einmal unser aller erstes UML-Diagramm zu EasySet. Zu diesem Zeitpunkt trug es noch den langweiligen Namen „Set“, doch glücklicherweise konnte ich Dominik davon überzeugen, dass es von signifikanter Bedeutung sei, den Präfix „Easy“ anzuhängen. Vielleicht hab ich da auch ein bisschen geflunkert, sagen Sie ihm das aber bloß nicht!

Zurück zum Thema:



Wie man in unserem ersten Klassendiagramm gut sehen kann, ist noch die sogenannte Linked-List-Struktur implementiert (auf die Später eingegangen wird) und außerdem stand eine KI noch zur Debatte. Diese wurde dann verworfen und in der finalen Version wieder implementiert. Erkennbar ist außerdem, dass die Verwendung von Sockets von Anfang an feststand und bis zum Ende mitgeschleift wurde. Das Networking war außerdem das erste Feature das implementiert wurde.

8 Evolution

Legende:

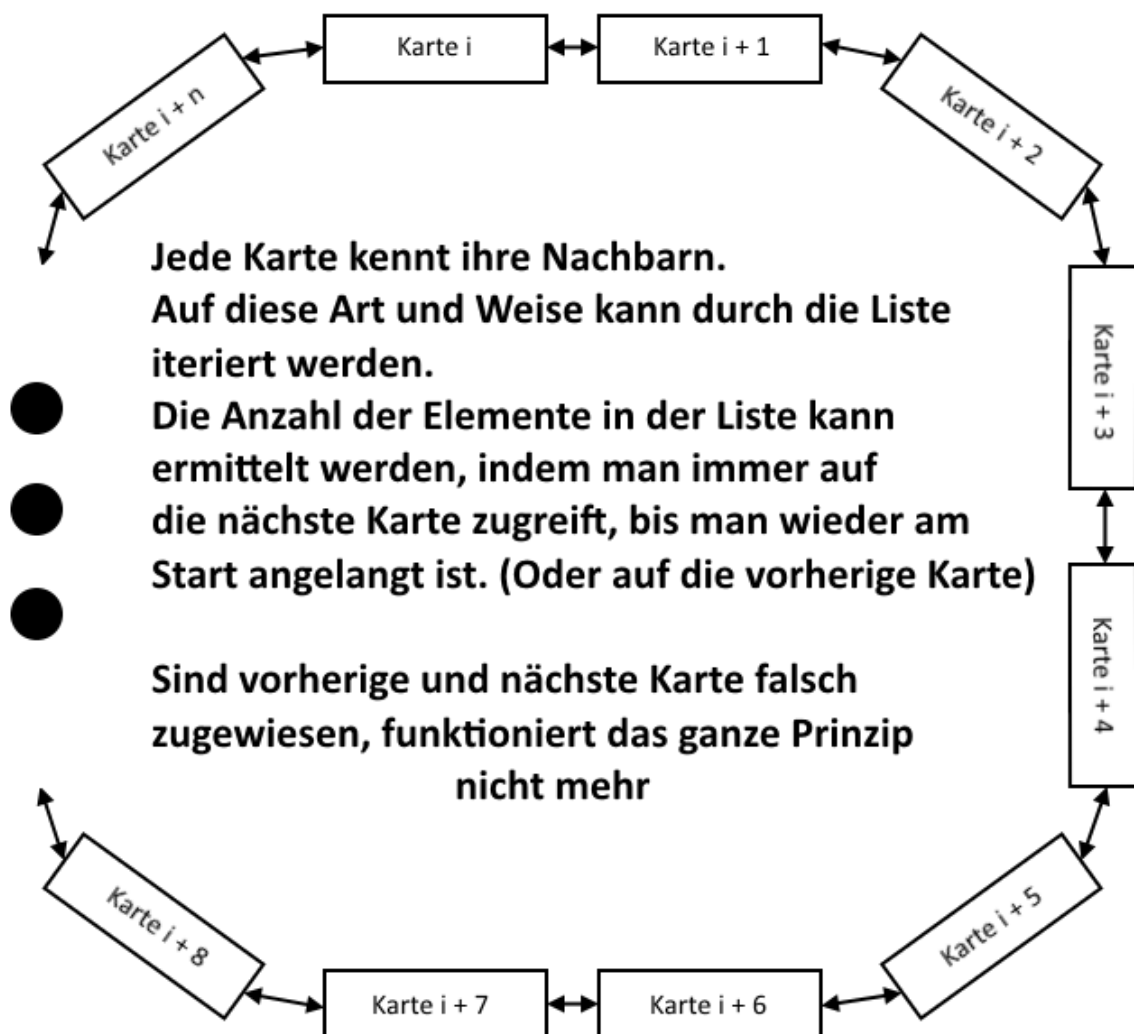
- netzwerkrelevante Klasse
- Normale Klasse
- Singleton-Klasse
- Graphical user interface

Die Klasse *Card* besaß noch das LinkedList-Pattern. Dies wurde jedoch aufgrund der unnötigen Komplexität und Speicherfehler durch eine normale `std::list<Card*>` ersetzt. Da die LinkedList ein doch sehr aufwändiges Verfahren war und es sich lohnt, darüber Bescheid zu wissen, widmen wir dieser Klasse einen Platz in dieser Dokumentation.

Wie man dem vorherigen Klassendiagramm entnehmen kann, besaß die Klasse *Card* 2 Attribute vom Typ *Card**. Diese Attribute hatten den Namen *m_nextCard* und *m_previousCard*. Diese beiden Attribute scheinen also auf die vorherige und auf die nächste Karte zu zeigen, doch was genau bedeutet das?

Man kann sich diese Art von LinkedList wie ein geschlossener Kreis vorstellen, bei dem jeder Knoten auf den nächsten und vorherigen Knoten zeigt.

Vorstellen kann man sich dies wie folgt:



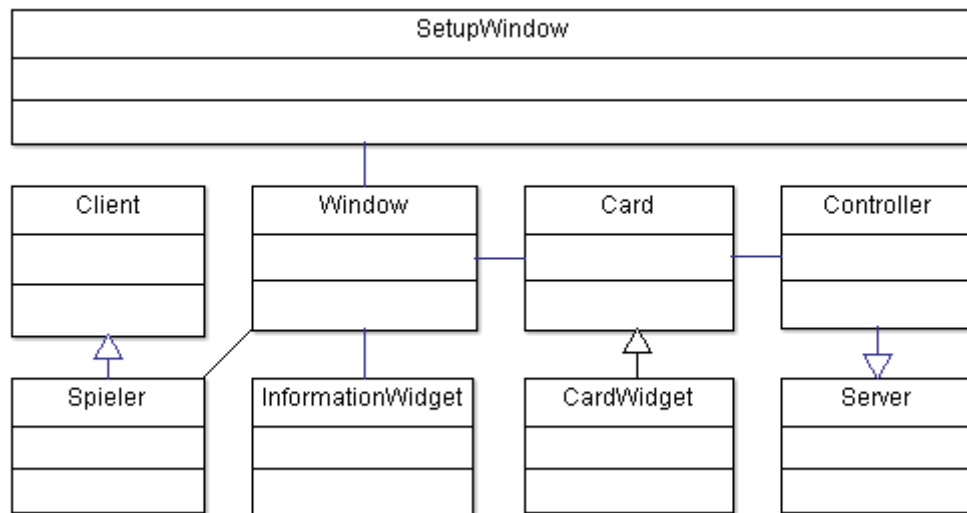
Zu Anfang waren die Variablen *m_nextCard* und *m_previousCard* ganz normale Pointer, doch eines Nachts entschlossen wir uns dazu, statt stinknormalen Pointer sogenannte *unique_ptr* der STL zu verwenden. *unique_ptr* sind sogenannte Smartpointer der Standardbibliothek und übernehmen die Speicherverwaltung für einen Anwender. Sprich sie löschen sich selbstständig usw. Eine weitere Eigenschaft von *unique_ptr* ist, dass auf den belegten

Speicher immer nur von einem zugegriffen werden kann. Bei Zugriffen auf denselben Speicher von unterschiedlichen Instanzen kommt es zu einem sogenannten Segmentation Fault. Schlichtweg ein Speicherzugriffsfehler. Aus genau diesem Grund entschlossen wir uns die Idee mit den Smartpointer wieder über Bord zu werfen (es gibt auch einen *shared_ptr* mit dem unser Problem hätte gelöst werden können, das war uns die Mühe dann jedoch doch nicht wert). Auch entschlossen wir uns das LinkedList-Pattern wieder zu entfernen, da es so eine hohe Fehleranfälligkeit gegeben hätte, wenn wir irgendwo die vorherige, beziehungsweise nächste Karte, nicht korrekt definiert hätten. Des Weiteren ist eine einfache Liste – für die wir uns entschlossen – um Einiges einfacher handzuhaben.

Die GUI *Window* wurde durch eine weitere GUI *SetupWindow* ergänzt, die für die Einrichtung des Spiels zuständig ist.

Des Weiteren wurde die Assoziation zwischen *Server* und *Client* getrennt und *Controller* erbt nun von *Server*. Bei *Client* und *Server* handelt es sich jeweils um abstrakte Klassen. Des Weiteren wurde *Window* zu einer Singleton-Klasse umgeformt und es wurde eine Klasse *CardWidget* hinzugefügt, die von *Card* erbt und für die Darstellung einer Karte sorgt. Hinzu kam auch das *InformationWidget*, das den Spielstatus liefert und als Steuerelement auf dem *Window* erzeugt wird.

Das Klassendiagramm hatte sich also bisher grob gesehen so entwickelt:



9 Finale

9.1 Die Klassen

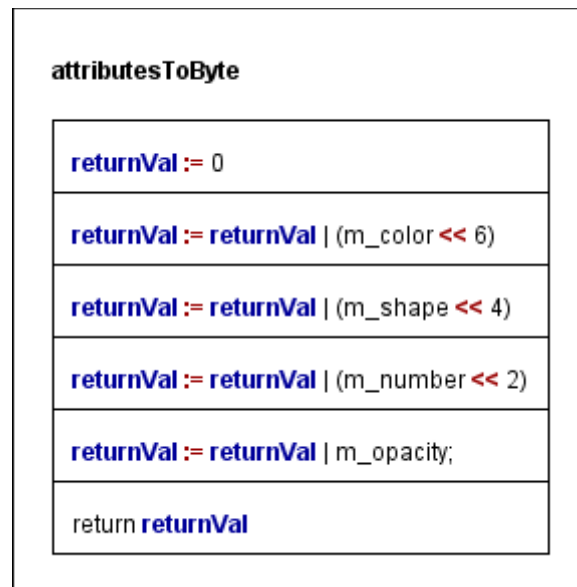
Es gilt dieselbe Legende wie oben, also:

- netzwerkrelevante Klasse
 - Normale Klasse
 - Singleton-Klasse
 - Graphical user interface
-
- **Server:**
Diese abstrakte netzwerkrelevante Klasse ist für die Struktur eines TcpSocket-Servers zuständig.
 - **Controller:**
Diese Klasse erbt von der abstrakten Klasse *Server* und regelt den gesamten Spielablauf.

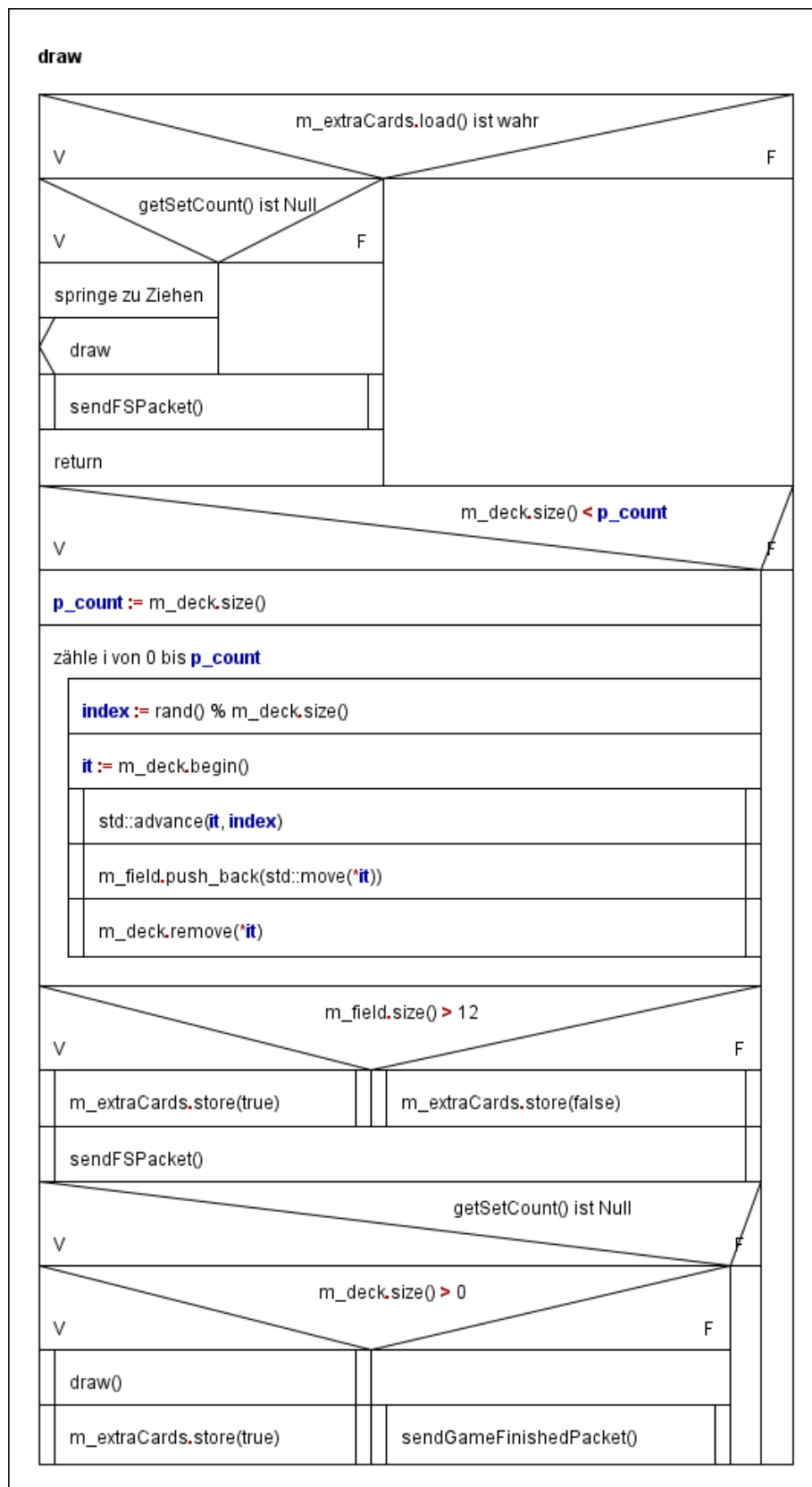
- *Client*:
Diese abstrakte netzwerkrelevante Klasse stellt die Client-TcpSocket-Struktur zur Verfügung.
- *Player*:
Diese Klasse erbt von der abstrakten Klasse *Client* und macht das Spiel spielbar.
- *PacketHandler*:
Die Klasse *PacketHandler* hat nichts direkt mit Sockets zu tun, verarbeitet jedoch die einzelnen eingehenden Pakete und gilt daher als netzwerkrelevant. Sowohl *Server* als auch *Client* besitzen eine Instanz dieser Klasse, an die die eingehenden Pakete übergeben werden. Die Klasse emittiert dann je nach Pakettyp Signale, die dann die verbundenen Slots aufrufen.
- *Card*:
Die Klasse *Card* stellt das digitale Pendant zu den analogen Karten des Spiels Set dar.
- *KI*:
Die *KI* erbt von *Player* und ist somit ein optionaler computergesteuerter Spieler.
- *CardWidget*:
Diese Klasse erbt von *Card* und stellt eine Karte grafisch dar.
- *InformationWidget*:
Bei dieser Klasse handelt es sich um ein Steuerelement, dass dafür zuständig ist, den aktuellen Spielstatus – wie zum Beispiel Punktestand, Spieleranzahl etc. – auszugeben.

- **SetupWindow:**
Die *SetupWindow* Klasse dient als Einrichtungsassistent für sowohl lokales als auch offline Spiel.
- **Window:**
Die Klasse *Window* ist sowohl eine Klasse mit dem Singleton-Designpattern als auch ein graphical user interface. Die Klasse *Window* bringt den gesamten Netzwerkverkehr zu einem Spiel zusammen.

Im Folgenden ein Strukturgramm der Funktion um eine Karte zu einem *Char* zu konvertieren (Prototyp: `char Card::attributesToByte();`)



Strukturgramm zur Funktion *draw()* der Klasse Controller.

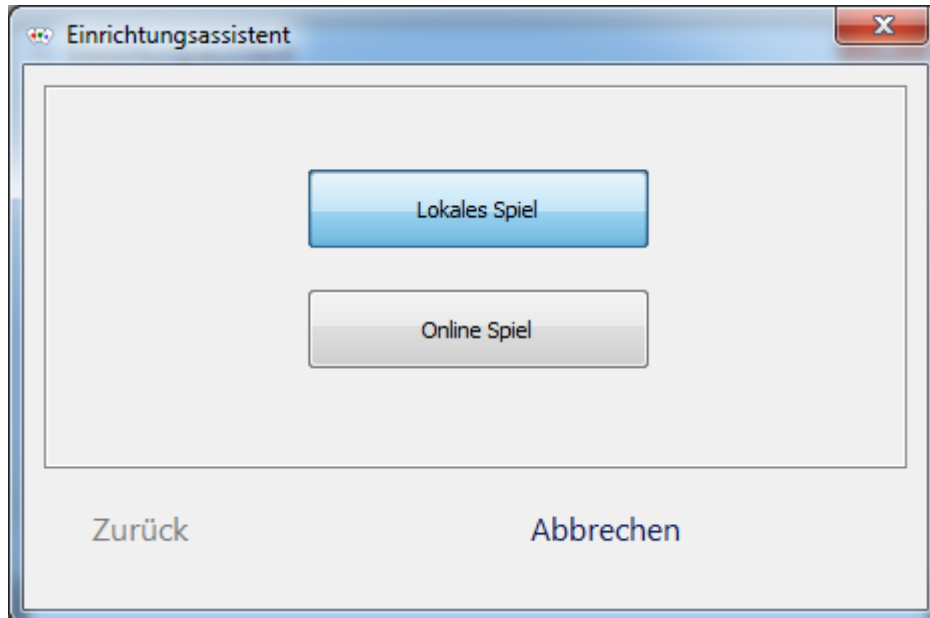


Diese Funktion ist dafür zuständig, dass Karten aufs Spielfeld gelegt werden.

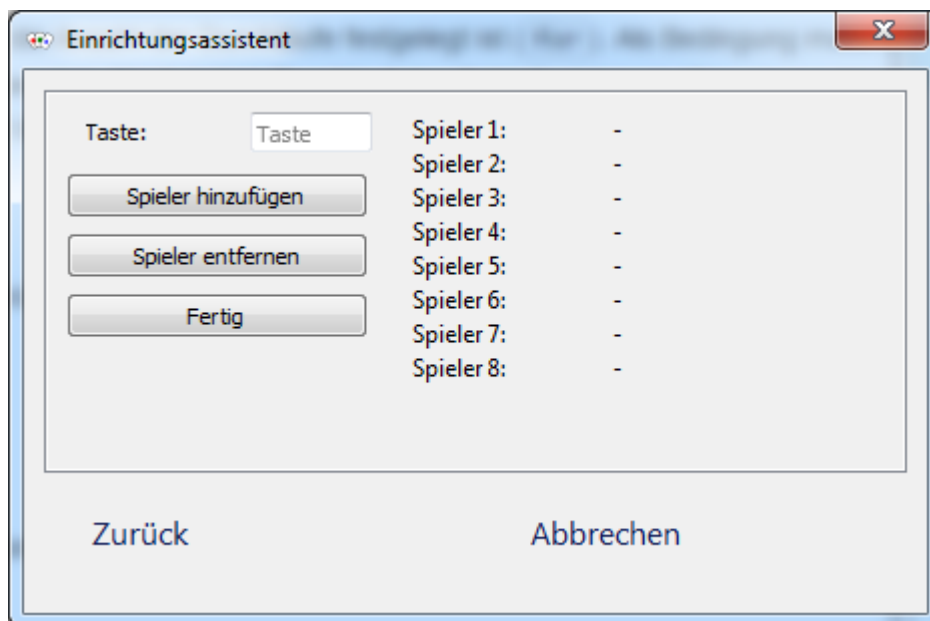
(Prototyp: `void draw(unsigned int p_count = 3);`)

9.2 Wie spiele ich richtig?

9.2.1 Lokales Spiel einrichten



Erscheint dieses Fenster, so muss auf „Lokales Spiel“ gedrückt werden.

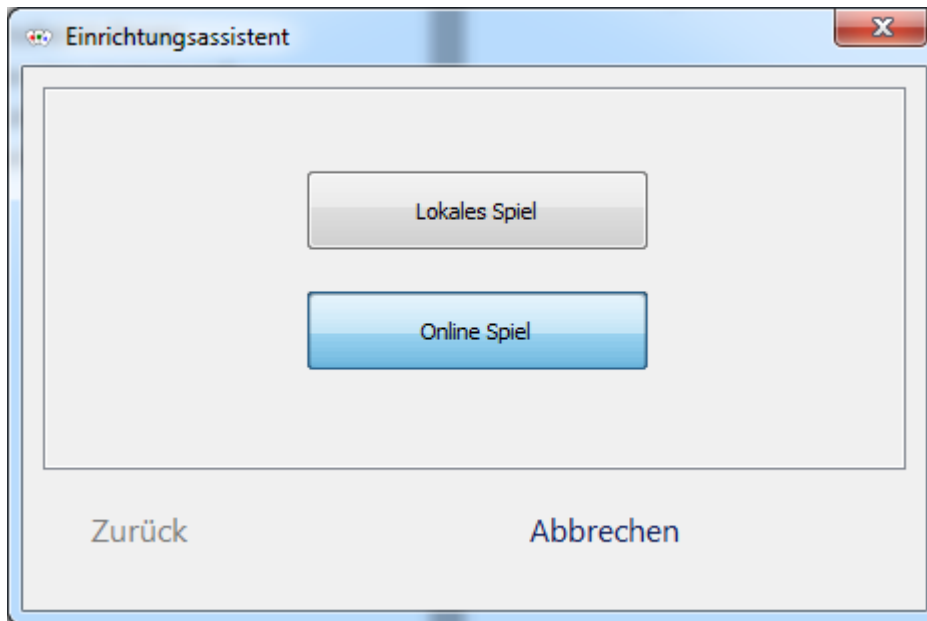


In diesem Fenster können nun Spieler hinzugefügt werden. Dazu muss man einen der 8 möglichen Spieler in der Liste auswählen und dann in dem Eingabefeld eine Taste eingeben. Dabei können nur alphanumerische Eingaben getätigt werden. Hat man die gewünschte Anzahl an Spieler

hinzugefügt, kann man auf „Fertig“ drücken.

Die Spieler benötigen alle eine unterschiedliche Taste, da der Server wissen muss, welchem Spieler die Punkte zugeschrieben oder abgezogen werden. Bei einem Klick auf „Fertig“ öffnet sich das Hauptfenster und das Spiel kann losgehen.

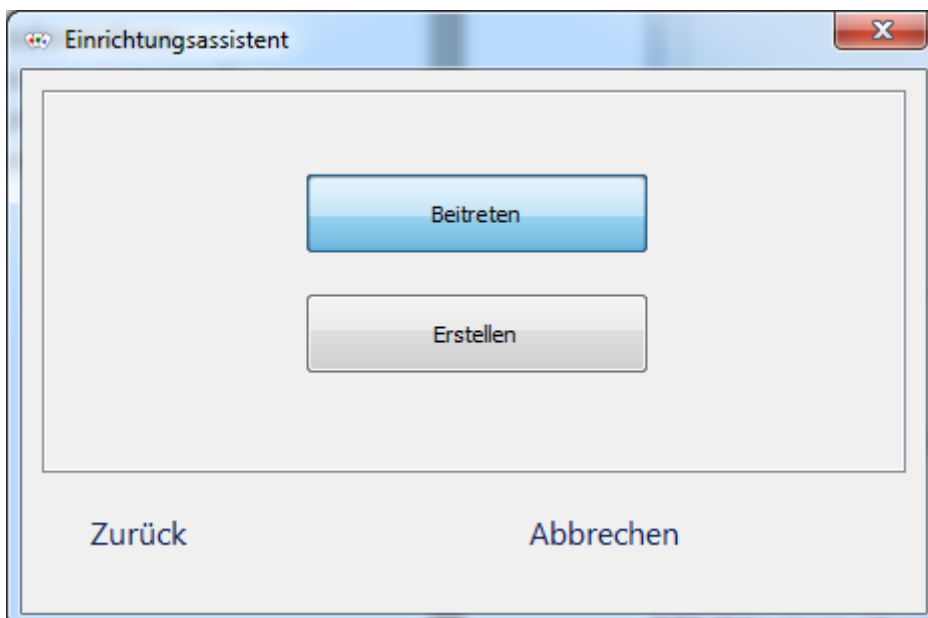
9.2.2 Online Spiel



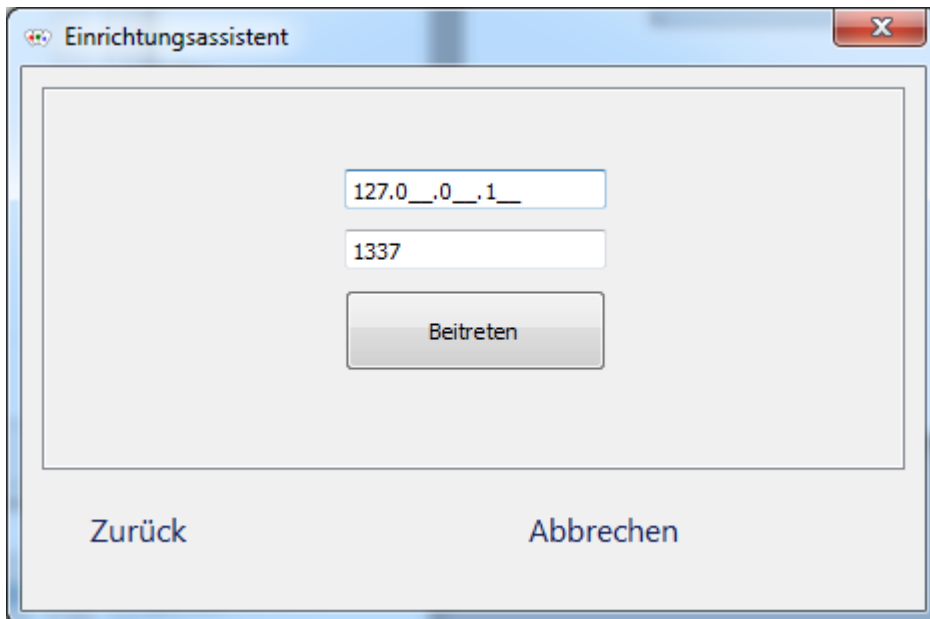
Erscheint dieses Fenster, so muss auf „Online Spiel“ geklickt werden.

Spiel beitreten

Möchte man einem bereits bestehenden Spiel beitreten, so muss im nächsten Fenster die Auswahl „Beitreten“ getroffen werden:



Nach dem diese Auswahl getroffen wurde, öffnet sich folgendes Fenster:

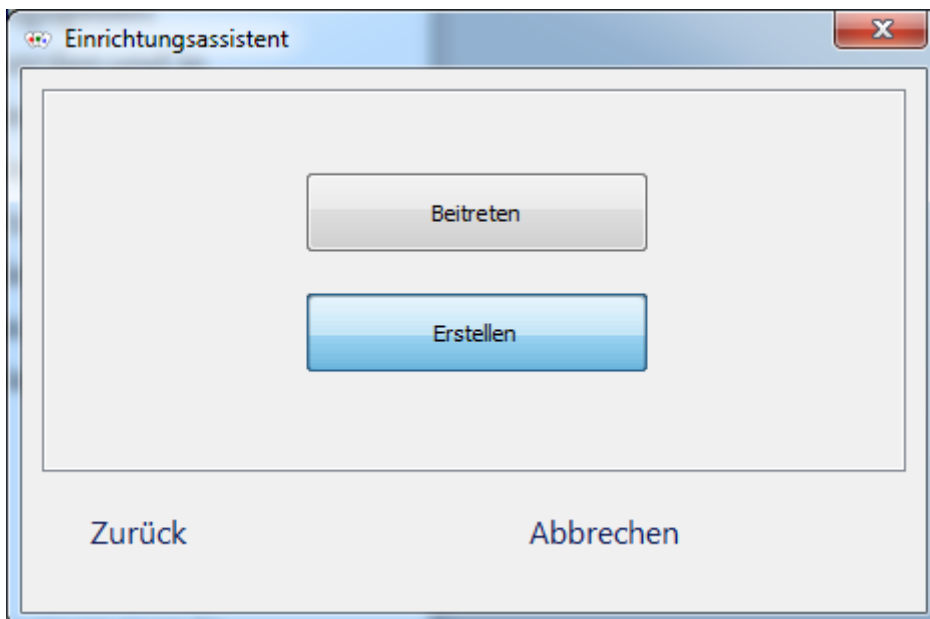


In dem ersten Textfeld muss die IP-Adresse des Servers angegeben werden und in dem zweiten Textfeld den Port über den die Verbindung laufen soll. Danach kann

„Beitreten“ geklickt werden.

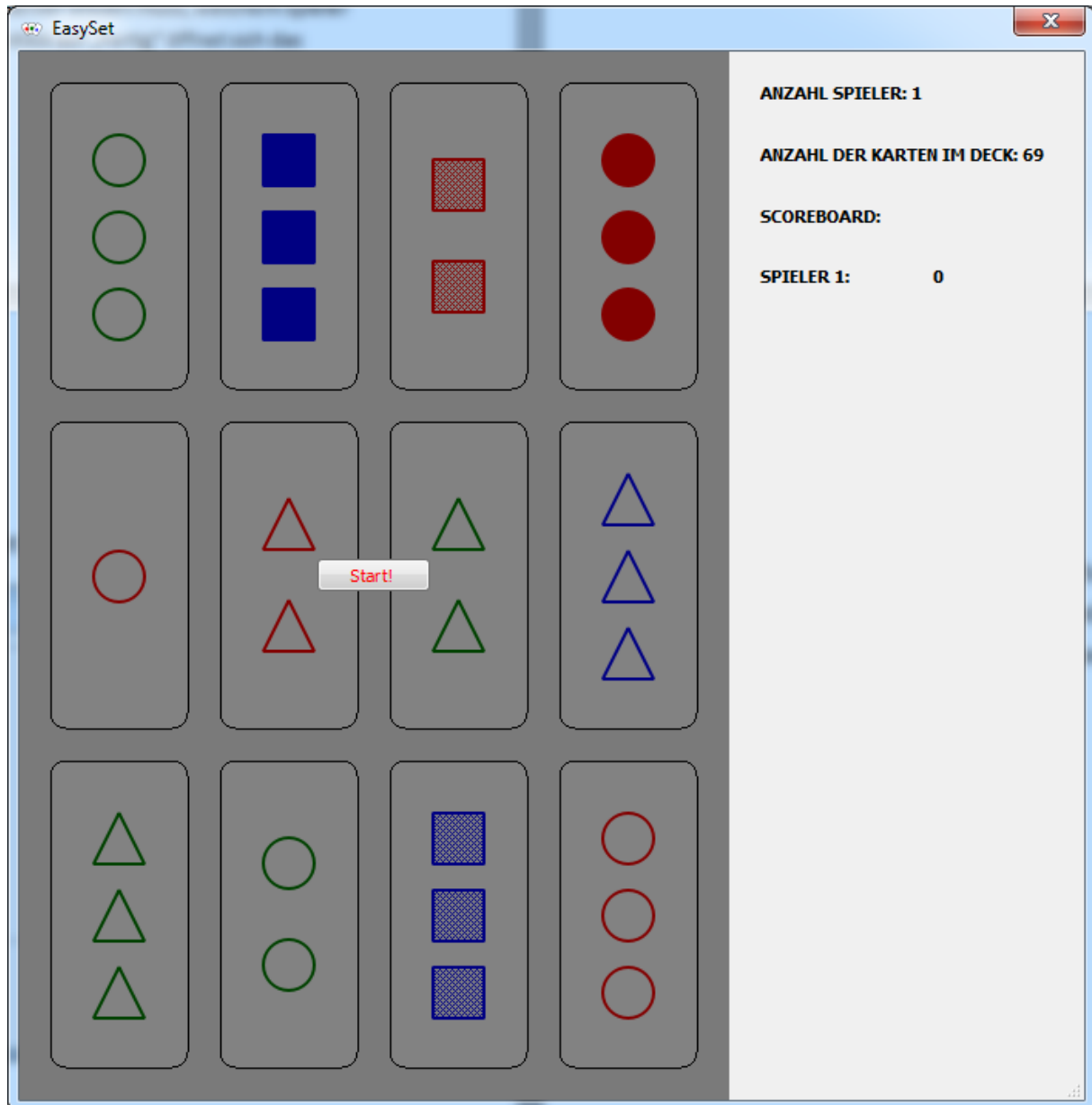
Spiel erstellen

Möchte man ein Spiel erstellen und somit als Server fungieren, so muss man in diesem Fenster lediglich auf „Erstellen“ klicken:



9.3 Spielen

Nachdem einer der vorherigen Schritte gewählt wurde, öffnet sich dieses Fenster, wobei der Start-Button lediglich für den Server sichtbar ist, für alle anderen Teilnehmer ist die Eingabe ganz normal gesperrt (ohne Start-Button):



Nach einem Klick auf den Button „Start“ wird für alle Teilnehmer die Eingabe entsperrt.

Möchte man nun ein Set auswählen, muss man die Taste auf der Tastatur drücken, die zuvor festgelegt wurde (Lokales Spiel). Bei einem Online Spiel

muss für jeden Spieler die Leertaste gedrückt werden, sofern man ein Set anklicken möchte. Nachdem man die Taste auf der Tastatur geklickt hat, hat man 5 Sekunden Zeit, um die 3 Karten anzuklicken, die ein Set bilden.

Außerdem werden die Eingaben der Mitspieler gesperrt während ein anderer Spieler an der Reihe ist. Schafft man dies nicht, so wird dem Spieler der am Zug ist ein Punkt abgezogen, ansonsten 1 Punkt hinzugefügt.

Sollte der Server erkennen, dass kein Set auf dem Spielfeld liegt, legt dieser automatisch 3 Karten nach. Das Spielfeld umfasst somit 15 Karten statt wie gewohnt 12.

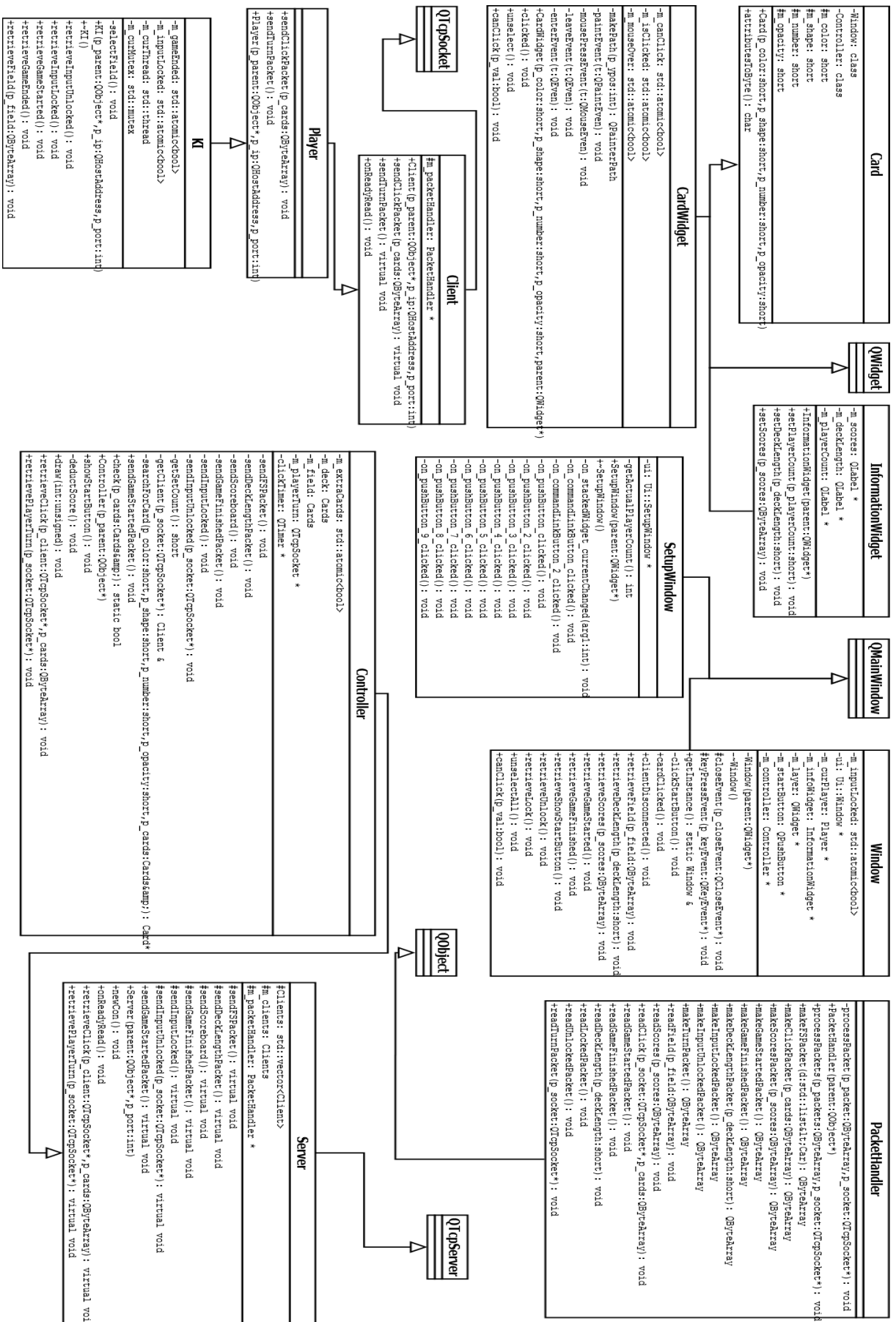
Der Spieler mit der höchsten Punktzahl gewinnt logischerweise.

Hinweis: Eingaben können NUR nach drücken der gewählten Taste auf der Tastatur getätigt werden! Der Server muss wissen, welchem Spieler er die Punkte gutschreibt.

9.4 Klassendiagramm

Zu guter Letzt darf natürlich nicht das finale Klassendiagramm fehlen, darum:

(Bei Bedarf liegt das UML-Diagramm auch noch einmal als Bild dabei)



10 Quellen

10.1 Doxygen

<http://www.stack.nl/~dimitri/doxygen/>

10.2 GitHub

<https://github.com/>

10.3 Qt 5.4.0

<http://www.qt.io/>

10.4 C++ Buch

ISBN 978-3-8266-9195-9