Software-Engineering Projekt

„Uno”

Hochschule für angewandte Wissenschaft und Kunst Göttingen

vorgelegt von: Felix Bauer
Matrikelnummer: 695033

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

2 Ziel der Arbeit

Das Ziel der Arbeit ist die Programmierung des Gesellschaftsspiels *UNO*. Das Spiel soll mit mehreren Spielen über ein Netzwerk spielbar sein. Optional soll ein sogenannter *bot* programmiert werden, der einen virtuellen Spieler darstellt. Ein besonderer Fokus bei der Programmierung liegt auf der Organisation und Planung im Team. Es soll gelernt werden, wie ein Software-Projekt strukturiert bearbeitet und fertiggestellt wird.

3 Bedingungen

Damit das Projekt in der vorgegebenen Zeit für eine Einzelperson umsetzbar ist, ist es notwendig vorab Prioritäten zu definieren. An aller erster Stelle steht die Funktion der Software. Die Optischen Eigenschaften stehen an letzter Stelle.
Folgende Hauptanforderungen sind für die Umsetzung des Projektes definiert:

- Spielbar mit vier Spielern über ein Netzwerk
- Grundlegende Spiellogik
- Grafische Benutzeroberfläche / Spielfläche
- Objektorientierte Programmierung
- Strukturierte Vorgehensweise

Neben den Hauptanforderungen sind folgende Nebenanforderungen definiert:

- Intuitive Menüführung
- Farbige Karten

4 UNO

Dieses Kapitel gibt einen Überblick über das Gesellschaftsspiel *UNO*, welches in dieser Arbeit programmiert wird. Es wird ausschließlich auf die Punkte eingegangen, die benötigt werden um die Funktionen in der programmierten Software zu verstehen. *UNO* ist ein Kartenspiel, geeignet für 2 - 10 Spieler ab einem Alter von 7 Jahren. Ziel des Spiels ist es, als erste Person alle Handkarten abgelegt zu haben.

4.1 Karten

Das standard-UNO-Kartendeck besteht aus 108 Karten, wie in Abbildung 4.1 zu sehen ist. Für die weitere Betrachtung werden die Karten in zwei Kategorien (Einfache Karten und Spezialkarten) eingeteilt. Zu den einfachen Karten gehören die nummerierten, die restlichen zu den Spezialkarten. Die einfachen Karten existieren in vier verschiedenen Farben. Jede Farb-

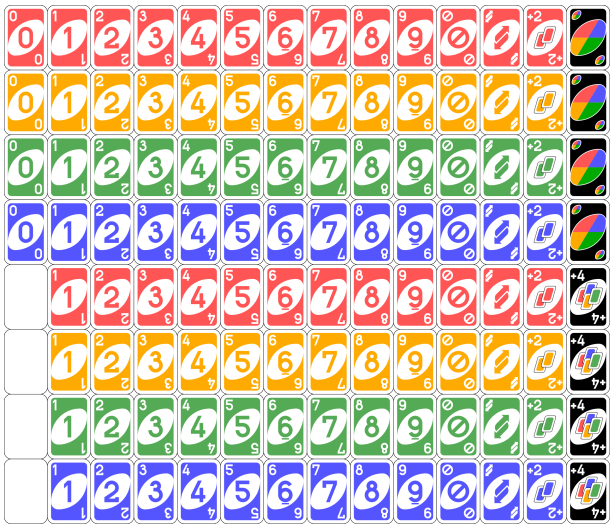

Abbildung 4.1
UNO Karten
Quelle: https://shopping.mattel.com/de-de/products/uno-kartenspiel-w2087-de-de

Zahl-Kombination existiert zweimal, wobei die Karten mit einer null nur einmal existieren. Im Rahmen dieser Arbeit sind vorrangig nur die einfachen Karten von Bedeutung. Spezialkarten verändern das Spielgeschehen, indem zum Beispiel der nächste Spieler für eine Runde aussetzen muss, oder zwei Karten ziehen muss.
Zu Beginn des Spiels erhält jeder Spieler 7 Karten. In der Tischmitte wird eine Karte aufgedeckt

platziert, auf dem die Spieler nach der Reihe Karten ablegen, oder ziehen können. Im folgenden wird auf die Spielregeln eingegangen.

4.2 Spielregeln

Ziel einer Runde ist es, als erster Spieler alle Handkarten auf den in der Tischmitte platzierten Kartenstapel abzulegen. Daraufhin werden die Wertungen der Karten der Gegenspieler addiert und dem Gewinner der Runde als Punkte gutgeschrieben. Der erste Spieler, der 500 Punkte erreicht, gewinnt das Spiel. Die Spieler sind nacheinander im Uhrzeigersinn an der Reihe. Eine Karte kann abgelegt werden, wenn entweder die Zahl oder die Farbe der abzulegenden Karte mit der Zahl oder Farbe der in der Tischmitte liegenden Karte übereinstimmt. Hat der Spieler keine passende Karte auf der Hand, so muss er eine neue Karte vom Stapel ziehen. Der Zug ist damit beendet. Hat ein Spieler nur noch eine Karte auf der Hand, so muss er dies mit dem Wort *UNO* den anderen Spielern mitteilen. Wird die letzte Karte ohne diese Aussage abgelegt, so muss die Karte wieder aufgenommen und eine weitere Strafkarte vom Stapel gezogen werden. Daraufhin ist der Zug beendet. Die Regeln bezüglich der Spezialkarten sind im Rahmen dieser Arbeit nicht von Bedeutung.

5 Software

Der Programmcode kann schnell eine sehr große Größe annehmen. Im Rahmen dieser Arbeit wird nur auf die essentiellen Punkte eingegangen und diese erläutert. Der vollständige Programmcode und jegliche Projektdateien befinden sich im Anhang. Die Software ist mit der Entwicklungsumgebung *Microsoft Visual Studio 2022* erstellt. Die gewählte Programmiersprache ist *C++*. Für die visuelle Darstellung der Spiel- und Benutzeroberfläche wird die Windows presentation foundation (WPF) verwendet. Mit der *WPF* können mithilfe einer sogenannten *Markup Language* (XAML) einfache grafische Benutzeroberflächen programmiert werden.

5.1 Grundlegender Aufbau

In diesem Kapitel wird auf den Aufbau der gesamten Software eingegangen und soll einen Überblick über das Projekt geben. Abbildung 5.1 zeigt die Menüführung der Software als Blockdiagramm. Wird die Software gestartet, so wird die Startseite aufgerufen. Hier kann der Spieler

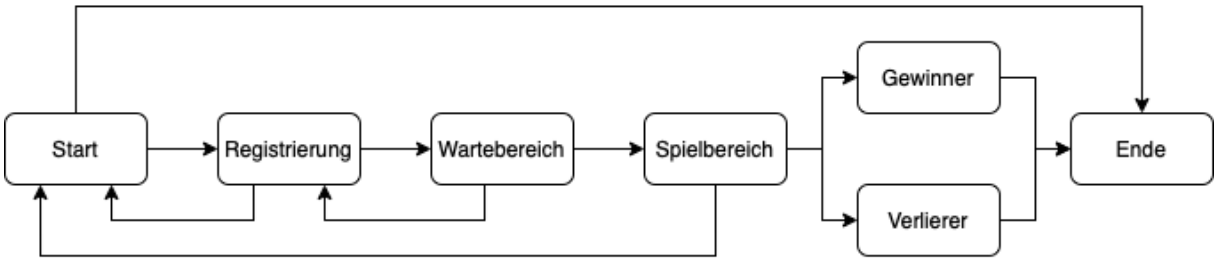

Abbildung 5.1
Blockdiagramm Software-Aufbau

das Spiel beenden, starten oder die Spielregeln einsehen. Wird das Spiel gestartet öffnet sich die Registrierungsseite. Hier wird der Spieler dazu aufgefordert seinen Namen einzugeben. Der Name wird dazu verwendet, um später im Spiel zu identifizieren welcher Spieler gerade am Zug ist. Für die Eingabe des Namens steht ein Textfeld zur Verfügung, welches auf maximal 20 einzugebende Zeichen begrenzt ist. Startet der Spieler das Spiel, so wird standardmäßig versucht eine Verbindung zu einem bestehenden Server aufzubauen. Wird jedoch ein Häkchen auf der Registrierungsseite gesetzt, so kann ein neuer Server gestartet werden. Nach der Registrierung wird der Spieler in einen Wartebereich geleitet. Dort wird auf die anderen drei Spieler gewartet. Entsprechende Benutzerrückmeldungen zeigen den aktuellen Status der Netzwerkverbindung und neu verbundene oder getrennte Spieler. Der Wartebereich wird automatisch verlassen, wenn vier Spieler erfolgreich mit dem Server verbunden sind. Ist dies der Fall, öffnet sich der Spielbereich.

In diesem Bereich wird das Spiel gespielt. Je nach dem ob der Spieler die Runde gewinnt oder verliert, öffnet sich eine entsprechende Seite, auf der das Ergebnis gezeigt wird. Bei den Spielern, die verloren haben, wird der Name des Spielers gezeigt der die Runde gewonnen hat. Die Software kann an diesem Punkt nur noch beendet werden. Um eine neue Runde zu starten muss auch die Software neu gestartet werden.

5.2 Klassen

Wie im Kapitel 3 in den Hauptbedingungen definiert, ist die objektorientierte Programmierung eine Voraussetzung für das Projekt. Durch diese Programmier-Art können komplexe, reale Sachverhalte verhältnismäßig einfach in Programmcode umgesetzt werden. In den folgenden Unterkapitel wird auf die verwendeten Klassen eingegangen und Funktionsweisen erläutert.

5.2.1 Player

Die Klasse *Player* bildet einen Spieler ab. Der Spieler im echten Leben hat einen Namen, einen Sitzplatz am Tisch und einige Karten in der Hand. Nach diesem Prinzip ist auch die Klasse aufgebaut. Codeausschnitt 5.1 zeigt einen Ausschnitt aus der Klasse. Die Variable *Name* beinhaltet den Namen des Spielers, die Variable *ip_port* beinhaltet den virtuellen Sitzplatz, also die IP-Adresse inklusive Port und die Variable *CardStack* beinhaltet die Handkarten des Spielers. Auf die Klasse *CardStack* wird später im Kapitel 5.2.2 näher eingegangen. Die Klasse besitzt nur zwei Konstruktoren und keine Methoden. Die beiden Variablen *NameLabel* und *NumberLabel* werden im weiteren Verlauf nicht verwendet, können jedoch dazu genutzt werden, um ein entsprechendes WPF-Label einem Spieler fest zuzuordnen.

Code 5.1
Codeauschnitt Klasse *Player*

```
1  public class Player
2  {
3    public Player(string name, string ip_port)
4    public Player() : this("?", "?") { }
5
6
7    public string Name { get; set; }
8    public string ip_port { get; set; }
9    public CardStack CardStack;
10   public Label NameLabel;
11   public Label NumberLabel;
12 }
```

Natürlich hat ein Spieler auch die Möglichkeit eine Karte zu legen oder zu ziehen. Diese Funktionen befinden sich im Hauptprogramm. Um den Code noch besser zu gestalten, wäre es sinnvoll diese als Methoden der Klasse hinzuzufügen.

5.2.2 Card / CardStack

Die Spielkarten spielen eine essentielle Rolle. Aus diesem Grund werden zwei Klassen für den Umgang mit den einzelnen Karten (*Card*) und einem Kartenstapel (*CardStack*) verwendet. Zunächst wird die Klasse *Card* für eine einzelne Karte betrachtet. Wie in Kapitel 3 beschrieben, werden nur einfache Karten programmiert. Sie bestehen aus einer Farbe, repräsentiert durch einen ganzzahligen Zahlenwert zwischen null und drei und einer Zahl zwischen null und neun. Codeausschnitt 5.2 zeigt einen Ausschnitt der Klasse *Card*. Weitere Attribute werden für eine einfache Karte nicht benötigt. Die Klasse kann jedoch erweitert werden um Spezialkarten repräsentieren zu können.

Code 5.2

Codeausschnitt Klasse *Card*

```
1  public class Card
2  {
3    public int number;
4    public int color;
5
6    public Card(int number, int color)
7    {
8      this.number = number;
9      this.color = color;
10   }
11 }
```

Codeausschnitt 5.3 zeigt den grundsätzlichen Aufbau der *CardStack*-Klasse. Ein Karten-Stapel (CardStack) besteht aus einer Liste (*Cards*) mit Elementen des Typs *Card*. Zu beginn eines Spiels werden mithilfe der Methode *createAllCards()* alle möglichen Spielkarten, wie in Kapitel 4.1 beschrieben zu der Liste *Cards* hinzugefügt. Um eine Karte zu ziehen, wird die Methode *getRandomCard()* verwendet. Die Methode gibt eine zufällig gewählte Karte zurück und entfernt sie aus dem Stapel. Mithilfe der Methode *returnCard(int index)* kann eine Karte an einer spezifischen Stelle des Stapels erhalten werden. Ein Karte kann dem Stapel mit der Methode *AddCard(Card add)* hinzugefügt und mit *RemoveCard(Card rem)* entfernt werden. Eine Karte kann nur entfernt werden, wenn sie in der Liste vorhanden ist. Ist die Karte nicht in der Liste vorhanden, so ist der Rückgabewert der Methode *null*. Die Anzahl der Karten im Stapel wird mit der Methode *getCounter()* zurückgegeben.

Code 5.3

Codeausschnitt Klasse *CardStack*

```
1  public class CardStack
2  {
3    public List<Card> Cards { get; set; }
4    public CardStack()
5    {
6      this.Cards = new List<Card>();
7    }
8
9    public void createAllCards();
10   public Card getRandomCard();
11   public Card returnCard(int index);
12   public void AddCard(Card add);
13   public Card RemoveCard(Card rem);
14   public int getCounter();
15 }
```

5.2.3 GameServer

Der *GameServer* ist das Herzstück der Anwendung. Er verarbeitet jegliche Spiellogik und Anfragen von Clients aller Spieler. Im folgenden wird genauer auf die Klasse eingegangen.
Für die Server-Client-Verbindung wird ein sogenanntes *NuGet-Paket* verwendet, welches compilierten Code enthält, der in externen Projekten verwendet werden kann [**NuGet**]. Im Rahmen dieser Arbeit wird das *SuperSimpleTCP*-Paket verwendet. Es enthält alle Klassen und Methoden, um einfache Server-Client-Verbindungen über das TCP-Protokoll herzustellen.
Codeausschnitt 5.4 zeigt einen Ausschnitt der *GameServer*-Klasse. Es handelt sich dabei um eine statische Klasse, da der Server zu jedem Zeitpunkt verfügbar sein muss und nur eine einzige Instanz der Klasse benötigt wird. Der Klassen-Member *server* beinhaltet alle Server-Funktionalitäten. Mit der Methode *StartServer()* wird ein neuer Server gestartet und die Event-Methoden in Zeile 11 - 13 an die entsprechenden Server-Events angefügt. Wird nun ein Datenpaket empfangen, so wird ein neuer Thread gestartet, indem das eingehende Datenpaket entsprechend seines Inhalts verarbeitet wird. Die private Klasse *RxMsg* dient zur Verarbeitung der Nachricht. Mit der Methode *Stop()* werden aktive Verbindungen getrennt und der Server gestoppt. Wird eine neue Verbindung eines Clients zum Server festgestellt, wird der Client dazu aufgefordert den auf der Registierungsseite eingegebenen Namen zum Server zu schicken. Es wird geprüft, ob der Name von einem anderen schon verbundenen Spieler genutzt wird. Ist dies der Fall, so wird an den Namen ein Ausrufezeichen angehängt und an den Client zurückgeschickt. Nach der Namensprüfung wird der Spieler zu der Liste *AllPlayers* hinzugefügt. Wird eine bestehende Verbindung getrennt, wird der entsprechende Spieler wieder von der Liste entfernt. Bei jeder neuen Verbindung wird geprüft, ob die Zielanzahl von vier Spielern erreicht ist, also ob die Liste *AllPlayers* vier Elemente enthält. Ist dies der Fall wird eine Nachricht mithilfe der

Methode *serverBroadcast()* an alle verbundenen Clients gesendet, die den Start des Spiels signalisiert. Daraufhin wird serverseitig das Spiel gestartet, d.h. alle Spielkarten werden generiert und in der Liste *AllCards* gespeichert, eine zufällige Karte aus *AllCrards* auf den in der Tischmitte liegenden Stapel (*MiddleStack*) verschoben und jeweils sieben Karten an jeden Client gesendet. Die Variable *activePlayer* steht für den Spieler, der gerade am Zug ist.

Code 5.4
Codeausschnitt Klasse *GameServer*

```
1  static class GameServer
2  {
3    static private SimpleTcpServer server;
4    static private CardStack AllCards;
5    static private CardStack MiddleStack;
6    static private List<Player> AllPlayers = new List<Player>();
7    static private int activePlayer = 0;
8
9    static public bool StartServer();
10
11   private static void Events_DataReceived(object? sender,
         DataReceivedEventArgs e);
12   private static void Events_ClientDisconnected(object? sender,
         ConnectionEventArgs e);
13   private static void Events_ClientConnected(object? sender,
         ConnectionEventArgs e);
14
15   public static void serverBroadcast(string msg);
16   public static void Stop();
17   public static void StartGame();
18   private static void removePlayer(string IpPort);
19   public static bool isActive();
20
21   private class RxMsg;
22   {
23     public string addPlayer(string name, string IpPort);
24     public void removePlayer(string IpPort);
25     private string CheckDuplicateNames(string name);
26     private bool checkMovePossibility(int number, int color);
27   }
28 }
```

5.2.4 Client

Jeder Spieler ist automatisch ein Client. Lediglich der Spieler der das Spiel als Gastgeber (Host) startet, startet einen Server und verbindet sich dann als Client mit dem eigenen Server. Die Klasse *GameClient* wird für die Kommunikation mit dem Server verwendet. Codeausschnitt 5.5

zeigt einen Ausschnitt der Klasse. Die Klasse verarbeitet eingehende Datenpakete vom Server und stellt Anfragen an ihn. Die Klasse ist überwiegend Event-gesteuert. Die Klassen-Member *myName* und *myCards* entsprechen den Membern Name und *CardStack* der Klasse Player. Wegen einer großen Umstrukturierung des Programmcodes, sind diese Member doppelt vorhanden. Sinnvoller wäre es einen Member der Klasse *Player* zu inkludieren. Der Member *client* beinhaltet, ähnlich wie bei der Klasse *GameServer*, alle nötigen Methoden für die Verbindung mit einem Server und die Kommunikation mit diesem.

Code 5.5

Codeausschnitt Klasse GameClient

```
1  static class GameClient
2  {
3    public static string myName;
4    public static CardStack myCards = new CardStack();
5    public static SimpleTcpClient client = new SimpleTcpClient(Globals.ipport);
6
7    public static bool find_server();
8    public static void Stop();
9    public static void RequestServer(string data);
10
11   private static void Events_Disconnected(object? sender, ConnectionEventArgs
         e);
12   private static void Events_Connected(object? sender, ConnectionEventArgs e);
13   private static void Events_DataReceived_Client(object? sender,
         DataReceivedEventArgs e)
14   {
15     string msg = Encoding.UTF8.GetString(e.Data);
16     if(msg.Contains("!counter!"))
17     ...
18     else if(msg.Contains("!card!"))
19     {
20       msg = msg.Remove(0, 6);
21       Card c = new Card(msg[0] - 48, msg[1] - 48);
22       Events.CardReceivedEvent(e.IpPort, c, false, false);
23     }
24     ...
25   }
26 public static class Events
27 {...}
28 ...
29 }
```

Eingehende Nachrichten vom Server werden mit der Methode *Events_DataReceived_Client(...)* verarbeitet. Insgesamt können zehn verschiedene Nachrichten ausgewertet werden. Sieben davon lösen ein Event im Hauptprogramm aus. Codeausschnitt 5.6 zeigt die Klasse *Events*, welche sieben *events*, beinhaltet, an die im Hauptprogramm Methoden gebunden werden können. Die

angebundenen Methoden werden durch einen einfach Aufruf der Event-Methode innerhalb der Klasse im Hauptprogramm aufgerufen. Dadurch wird keine permanente Abfrage von Parametern benötigt. Die Methode im Hauptprogramm wird nur aufgerufen, wenn der bestimmte Fall eingetreten ist. Das sorgt für weniger Rechenaufwand während der Laufzeit des Programms. Jeder Event-Methode werden dem entsprechenden Fall, Parameter übergeben, welche im Hauptprogramm genutzt werden können. Wird beispielsweise eine Karte vom Server empfangen, so kann die Karte im Hauptprogramm an entsprechender Stelle angezeigt werden.

Code 5.6

Codeauschnitt Client Events

```
1  public static class Events
2  {
3    public static event EventHandler<CardReceivedEventArgs> CardReceived;
4    public static event EventHandler<PlayerReceivedEventArgs> PlayerReceived;
5    public static event EventHandler<StatusChangedEventArgs> StatusChanged;
6    public static event EventHandler<ConnectionCounterChangedEventArgs>
         ConnectionCounterChanged;
7    public static event EventHandler<EnemyNameReceivedEventArgs>
         EnemyPlayerNameReceived;
8    public static event EventHandler<MoveEventArgs> MoveReceived;
9    public static event EventHandler<WinEventArgs> WinnerReceived;
10 }
```

5.2.5 Installationsanweisung

Die Software muss nicht installiert werden, sondern kann direkt durch das Öffnen der sich im Anhang befindlichen, ausführbaren *.exe*-Datei gestartet werden. Um ein Spiel spielen zu können, ist es notwendig, dass die Software insgesamt vier Mal im gleichen Netzwerk gestartet wird. Eine Instanz muss dabei als Server gestartet werden. Dazu muss auf der Registrierungsseite das Häkchen bei *Server* gesetzt werden, welches in der Abbildung **??** markiert ist.

Abbildung 5.2
Registrierungsseite - Spiel mit neuem Server starten

6 Zusammenfassung

7 Ausblick