Software-Engineering Projekt

„Uno”

Hochschule für angewandte Wissenschaft und Kunst Göttingen

vorgelegt von: Felix Bauer
Matrikelnummer: 695033

Inhaltsverzeichnis

Abbildungsverzeichnis **II**

1 Ziel der Arbeit **1**

2 Bedingungen **2**

3 UNO **3**

3.1 Karten 3

3.2 Spielregeln 4

4 Software **5**

4.1 Grundlegender Aufbau 5

4.2 Klassen 6

Literatur **9**

Abbildungsverzeichnis

3.1 UNO Karten Quelle: https://shopping.mattel.com/de-de/products/uno-kartenspiel-w2087-de-de 3

4.1 Blockdiagramm Software-Aufbau 5

1 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.

2 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

3 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.

3.1 Karten

Das standard-UNO-Kartendeck besteht aus 108 Karten, wie in Abbildung 3.1 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-

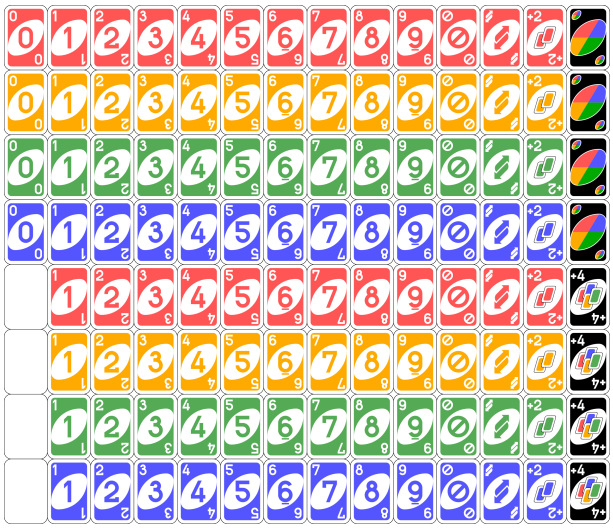

Abbildung 3.1
UNO Karten
Quelle: https://shopping.mattel.com/de-de/products/uno-kartenspiel-w2087-de-de

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.

3.2 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.

4 Software

4.1 Grundlegender Aufbau

In diesem Kapitel wird auf den Aufbau der gesamten Software eingegangen und soll einen Überblick über das Projekt geben. Abbildung 4.1 zeigt die Menüführung der Software als Blockdiagramm. Wird die Software gestartet, so wird die Startseite aufgerufen. Hier kann der Spieler

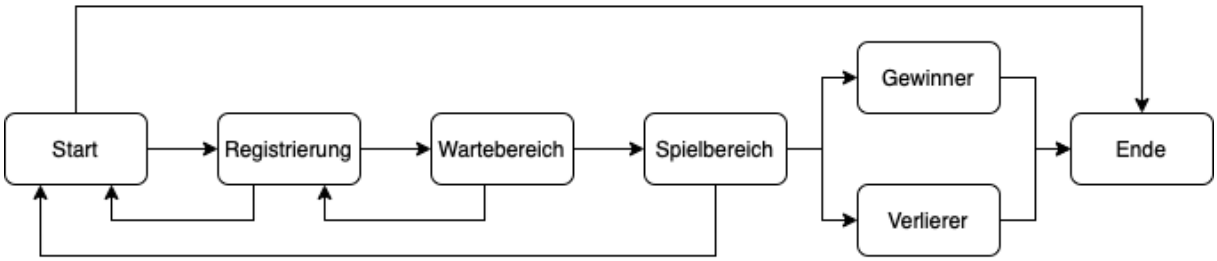

Abbildung 4.1
Blockdiagramm Software-Aufbau

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.

4.2 Klassen

4.2.1 GameServer

Der *GameServer* ist das Herzstück des ganzen Spiels. Er verarbeitet jegliche Spiellogik und Anfragen von Clients. Im folgenden wird genauer auf die Klasse eingegangen.
Für die Server-Client-Verbindungen wird ein sogenanntes *NuGet-Paket* verwendet, welches compilierten Code enthält, der in externen Projekten verwendet werden kann [1]. 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 4.1 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.

Code 4.1
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    }
```

4.2.2 Client

4.2.3 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 2 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 4.2 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 4.2
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 4.3 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 3.1 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 4.3

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 }
```

4.2.4 Player

Literatur

[1] Microsoft. *Eine Einführung in NuGet*. online abgerufen am 16.08.2022. URL: `https://docs.microsoft.com/de-de/nuget/what-is-nuget`.