Programmierung Portfolio

Uno spiel

Annika Bender, Florian Kenner, Malena Knoch, Kim Weber

2022

Contents

[1.Einleitung 2](#_Toc104282086)

[2.Prozess 2](#_Toc104282087)

[3.Funktion 4](#_Toc104282088)

# 1.Einleitung

Unsere Gruppe hat sich für unser Portfolio in Fortgeschrittene Programmierung überlegt das berühmte Uno Spiel zu programmieren.

# 2.Prozess

Um mit unserem Projekt zu beginnen, mussten wir uns erst einmal Gedanken machen, was so ein Uno-Spiel beinhaltet. Wir haben uns die verschiedenen Komponenten des Spiels angeschaut und diese in verschiedenen Klassen unterteilt. So haben wir zum Beispiel eine Klasse Karte, Spieler, Spiel und Stapel, den wir wiederum in Unterklassen unterteil haben und zwar in einen Stapel zum Ziehen und einen Stapel zum Ablegen. Erst haben wir unsere Ideen aufgezeichnet und eine Art Brainstorming gestartet, um dann später ein Ablaufdiagramm zu erstellen.

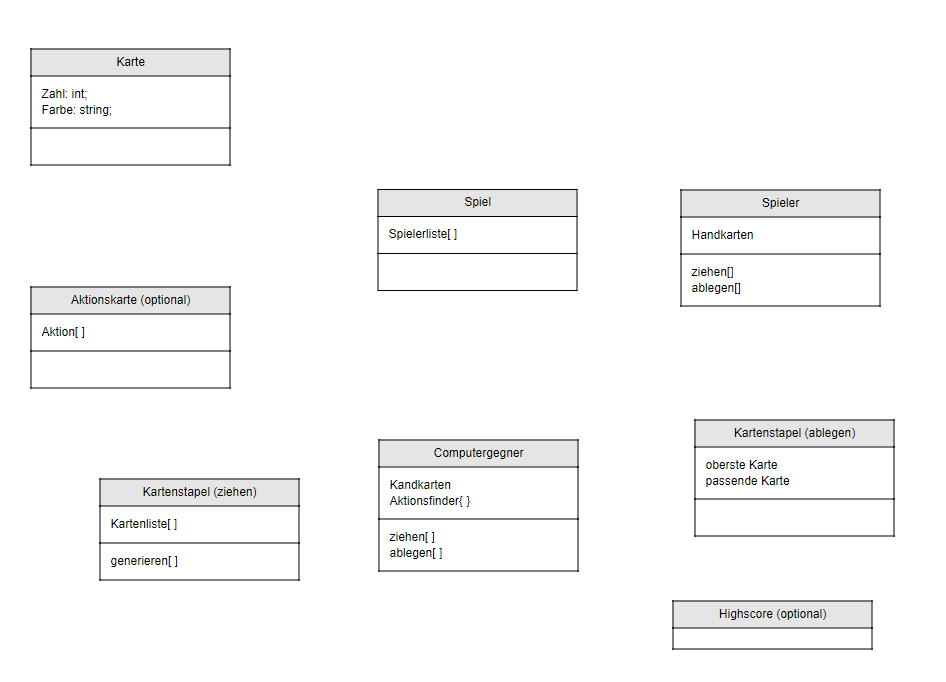


Abbildung 1 Verschiedenen Klassen

So konnten wir uns an der groben Struktur orientieren und waren uns sicher die wichtigsten Aufgaben auf einem Blick zu haben. Das Diagramm zeigte uns grob auf, wie unser Programm am besten funktionieren sollte. Hier haben wir die verschiedenen Szenarien ausgezeigt, die in unserem Spiel auftreten können und mit welchen Methoden wir diese am ehesten behandeln können. Später werden die das Diagramm noch einmal genauer erklären.

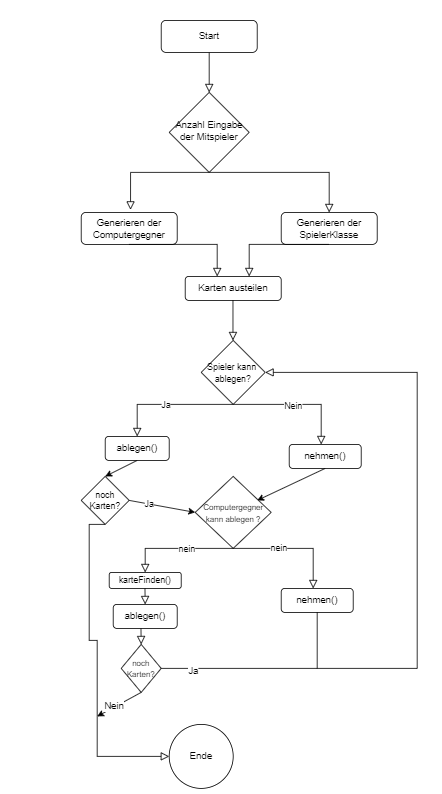


Abbildung 2 Abblaufdiagramm

Dann ging es auch schon an die Aufgabenverteilung. Zu nächst haben wir uns um die verschiedenen Klassen gekümmert, so haben zum Beispiel Kim um die Klasse Karte gekümmert, Annika um die Klasse …., Florian um die Klasse.... und Malena um die Klasse...

Um uns die Zusammenarbeit zu vereinfachen haben wir viel mit git hub gearbeitet. Dort hatte fürs erste jeder seinen eigenen branch, wo jeder sich ausprobieren konnte und für sich auf seinen eigenen Geräten arbeiten konnte. Wir haben uns nach der Uni getroffen, um uns gegenseitig zu updaten oder trafen uns im Discord, um schnelle Fragen zu besprechen. Unsere Treffen nutzten wir dann auch, um unsere Arbeit zu mergen und um unsere Arbeit abzugleichen und ggf. Sachen anpassen, damit die verschiedenen Klassen gut miteinander kooperieren können. Als wir nun die Klassen geschrieben haben, mussten wir die Logik fehlerfrei und effizient hinbekommen, so dass unser Spiel erstmal ohne Benutzeroberfläche gut funktionieren konnte. Hier haben wie uns Dinge überlegt wie zum Beispiel Exception Handeling oder das Benutzen von Sortieralgorithmen, um die Handkarten sortiert anzeigen zu können.

# 3.Funktion

Wie oben schon erwähnt haben wir das Spiel in verschiedene Klassen unterteilt. Wir haben jede Klasse in verschiedenen Packeten angelget, damit wir eine gute Übersicht haben.

Beginnen wir mit der Klasse Karten. Hier haben wir die Attribute Farbe und Zahl vergeben, da diese für das Spiel essenziell notwendig sind. Im Kontruktor haben wir die farben und die zahl noch einmal vergeben und mit einer toString Methode aufgerufen. Für eine korrekte Ausgabe haben die Zahl noch einmal in eine String umgewandelt.

-highscore hashmap strin auf arry jede position von spieler (alle spieler)

Hashmap alle durchschnitt position durchschnitsposition wird sotiert

Spielabspeichern spielerlist wird übergeben

Funktioniert über ine csv datei wird nach der anzahl der karten verglichen

Mit buffer reinschreiben, name und postion wird eingeschrieben ergibt sich aus idex plus 1

Einlese funktion wird eingelesen und wird in der Arryliste Spieler abgespeichert

Jeder steht nur einmal in der liste

Karte haben Hashmaps für die Farben

Farbe und zahl wir übergeben, zusätzlich javafx farbe übergeben, wird als Java fx objekt angelgt karte auf dem Ablegestapel

Überprüfung von farbe und zahl gleich

Spiel

Vier propertys (sind mega geil für ui, updaten den aktuellen stand, ui elemte bin an property und aktualiesieren alles, änderung wird direkt regestriert)

Konstruktor

Stapel wird generriert, spieler, karten werden ausgeteilt update bliblablub probertys werden neu gestezt(aktualisieren), kartenstand karten werden oben angezeigt

Buttons von mesnchlichen spieler, wandelt karten in buttons um und packt die in eine liste, die später in UI verwendet wird

amZug prüfen schaut ob man dran ist, schauen ob man dran ist und ob eine Karte passt

nächster Spieler der nächste spieler kommt dran

propertys werden wirder aktualisiert

generiere spieler werden die spieler + Computer

jemand ist Fertig

Spielen plus thread

Computer karte finden, passende farbe wird gesucht dann zahl mit for schleifen

Wenn das nicht funktioniert dann exception handling

Spieler hat Handkarten array list, wird zum visualisern benutzt

Benutzt hashmap von string zu int

Alle farben werden vorher im konstruktor angeben

Ziehen wird karte übergeben

Ablegen, sucht in der map nach einer richtigen karte

Wird überprüft ob er karten hat und wie viele und packt die Handkarten wieder in eine Array liste

Scene controller verbindet die Elemente,

wenn interface implementiert wird

Zielsetzung umsetzung, funktion

Wer war wie beteiligt, Ergebnis, erweiterungspotential (Aktionskarten, ggf. Anderer menschl. Spieler)