Fachbereich 4: Informatik

Entwicklung einer interaktiven Applikation unter Android

Bachelorarbeit

zur Erlangung des Grades einer Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Yessika Legat

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Kevin Keul

Koblenz, im Juni 2015

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

. . . .

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1 Abstract **1**

2 Einleitung **2**

3 Android Grundlagen **4**
3.1 Geschichte 4
3.2 XML 4
3.3 Views and Layouts 6
3.4 Interaktion des Nutzers 6
3.5 Activities 6
3.6 Shared Preferences 8

4 Yaniv Spielgregeln **9**

5 Entwurf **12**
5.1 Model View Controller 12
5.2 Konzept 13

6 Implementierung **16**
6.1 Kartenmodell 17
6.2 Spielstart 17
6.3 Ermittlung der Spielerreihenfolge 18
6.4 Valider Zug 18
6.5 Aufnehmen einer Karte 21
6.6 Computer-Logik 22
6.7 Spielende 25
6.8 Highscore 25

7 Fazit **29**

1 Abstract

Die vorliegende Arbeit befasst sich mit der Entwicklung einer interaktiven Applikation unter Android, welche das Spielen eines Kartenspiels ermöglicht. Exemplarisch wurde das hebräische Spiel **Yaniv** implementiert. Schwerpunkt ist die Herausarbeitung benötigter Hintergrundkomponenten und die dazugehörige Umsetzung in jener Applikation. Benötigte Spielprozesse werden durchleuchtet und eine mögliche Lösungsvariante aufgezeigt.

This thesis deals with the development of an interactive Android card game. As an example, the hebrew game **Yaniv** was implemented. Focus is the elaboration of required background components and the corresponding implementation in that application. Required game processes will be screened and a possible solution will be identified.

2 Einleitung

Die Möglichkeiten der Unterhaltungsmedien erreichen immer größere Dimensionen. Neue Eingabemethoden, wie Sensorik, Sprach- und Gestensteuerung, erlauben ein neues virtuelles Spielerlebnis. Trotz dieser Entwicklungen erfreuen sich klassische Kartenspiele immer noch großer Beliebtheit. Heutzutage gibt es noch regelmäßige Stammtisch-Runden, die den Abend mit Kartenspielen wie Skat verbringen. Allerdings sind damit einige Hürden verbunden. Alle Beteiligten müssen zur selben Zeit am selben Ort sein. Ferner scheint es sinnvoll klassische Kartenspiele auf digitalen Endgeräten anzubieten. Mobile Geräte, wie Smartphones oder Tablet-Computer, ermöglichen das Spielen an denkbar vielen Plätzen, wodurch die Nutzung eines virtuellen Spieles nicht mehr an einen festen Ort gebunden ist. Der gesellschaftliche Aspekt geht bei einem Kartenspiel auf einem mobilen Endgerät allerdings verloren. Daher kann eine Kartenspiel Applikation selten eine echte Spielrunde ersetzen. Es kann vielmehr als ein zusätzliches Angebot gesehen werden, falls einmal nicht alle Mitspieler zur gleichen Zeit verfügbar sind. Die Existenz einer Kartenspiel-Applikation ist vor allem damit zu begründen, dass sie durch den mobilen Charakter der Endgeräte jederzeit spielbar ist. Hervorzuheben ist zudem, dass bei der Betrachtung der App-Stores der Marktführer Apple und Google auffällt, dass in beiden eine eigene Kategorie zum Thema Kartenspiele existiert. Dies unterstreicht die These, dass klassische Kartenspiele ebenfalls bei mobilen Endgeräten beliebt sind [Que15c, Que15b].

Ziel der vorliegenden Arbeit ist die Entwicklung einer spielfähigen und interaktiven Kartenspiel-Applikation, sowie die Ergründung der prototypischen Ablaufprozesse im Hintergrund eben jener Applikation. Es sollen somit die Routinen, die hinter dem Spiel liegen, herausgearbeitet und eine exemplarische Lösungsvariante aufgezeigt werden. Dies soll dem induktiven Transfer der gewonnenen Erkenntnisse auf andere Kartenspiel-Applikationen dienen. Unzählige Game-Engines unterstützen bereits Programmierer, indem vorgefertigte Routinen bereitstehen. Dadurch können Entwickler mehr Zeit in Design und Spielkonzept investieren. Im Vordergrund dieser Arbeit steht das Herausarbeiten jener verwendeten Routinen, daher ist es unabdingbar die Prozesse und Routinen eigenständig zu entwickeln und zu erarbeiten. Aufgrund dessen wird bewusst auf die Verwendung einer Game-Engine oder eines vorgefertigten Frame-Works verzichtet.

Der erste Aspekt in der Entwicklung eines Kartenspiels für ein mobiles Endgerät ist die Festlegung des genutzten Betriebssystems. Hierbei bietet sich das Open-Source Android-Betriebssystem an, da es sich dabei um eines der weltweit verbreitetsten Betriebssysteme für mobile Endgeräte han-

delt [Que15e]. Die Entscheidung für ein bestimmtes Kartenspiel stellt den zweiten Aspekt der Entwicklung dar. Diese Arbeit beschäftigt sich mit dem hebräischen Spiel **Yaniv**. Dieses enthält viele Aspekte eines klassischen Kartenspiels und zeichnet sich zudem durch einige Zusatzregeln aus. Es bietet sich somit als exemplarisches Spiel für eine Android Applikation an.

Der erste Teil der Ausarbeitung beschäftigt sich mit den Grundlagen der Android-Programmierung. Es werden Hintergrundinformationen über die allgemeinen Bestandteile einer Android Applikation sowie die Geschichte des Android-Betriebssystems dargestellt. Im darauf folgenden Kapitel 4 werden die dem Spiel **Yaniv** zugrunde liegenden Spielregeln erläutert. Aus dem festgelegten Regelwerk ergeben sich Anforderungen, die die Applikation erfüllen soll. Neben diesen Anforderungen werden weitere funktionale Anforderungen bestimmt und daraufhin ein Konzept entwickelt. Dieses Konzept stellt den Schwerpunkt des nächsten Abschnittes dar. Anschließend wird in dem Implementationskapitel 6 näher auf den programmspezifischen Teil eingegangen. Abschließend werden die Ergebnisse in Bezug auf die Zielsetzung erläutert und ein ergänzender Ausblick auf mögliche Weiterentwicklungen gegeben.

3 Android Grundlagen

Bei Android handelt es sich um eine Open-Source-Plattform, die hauptsächlich für mobile Geräte konzipiert wurde. Eigentümer ist die Open Handset Alliance, die von Google gegründet wurde. Android ist eine offene Plattform, die die Hardware von der Software, die darauf läuft, trennt. Dadurch bieten sich viele Möglichkeiten der parallelen Ausführung für Anwendungen [Gar11]. Es handelt sich um eine umfassende Plattform für mobile Geräte. Dies bedeutet, dass sie Entwicklern für mobile Anwendungen alle benötigten Funktionen bereitstellt. Das Android-SDK bietet eine gute Unterstützung für Entwickler. SDK steht für Software Development Kit. Es bietet eine Sammlung von Anwendungen inklusive einer Dokumentation. Ein physisches Gerät kann durch einen Emulator ersetzt werden und ist daher nicht von Nöten.
Android Entwicklern stehen alle Funktionen kostenlos zur Verfügung. Die gesamten Low-Level-Linux-Module bis hin zu nativen Bibliotheken sind offen zugänglich. Des Weiteren sind die verfügbaren Bibliotheken lizenzfrei und lassen sich für verschiedene Zwecke frei erweitern [Gar11].

Das folgende Kapitel beschäftigt sich mit den Grundlagen der Android-Programmierung. Es wird eine kurze historische Einleitung gegeben und anschließend werden die einzelnen wichtigen Komponenten einer Android-Applikation beschrieben, welche in dieser Arbeit Verwendung finden.

3.1 Geschichte

Im Jahre 2005 kaufte Google Android auf. Erst 2007 wird die Open Handset Alliance bekannt gegeben und Android ist ein offizielles zugängliches Open-Source Produkt. 2008 wird das erste G1-Handy von HTC hergestellt und von T-Mobile auf den amerikanischen Markt gebracht. Das Android SDK 1.0 ist veröffentlicht. Ein Jahr später sind bereits mehr als 20 Android-basierte Geräte auf dem Markt erwerblich. Sie verfügen über neuere Versionen des Betriebsystems, wie Cupcake (1.5), Donut (1.6) oder aber Eclair (2.0 und 2.1). Bereits 2010 handelt es sich bei Android um eine der meistverkauften Smartphone-Plattformen für mobile Geräte [Gar11]. Seit Januar 2010 stellt Google eigene Android Geräte her, die unter der Produktreihe Nexus verkauft werden. Da die Software direkt von Google erstellt wird, ergeben sich kurze Wartezeiten bei einem Update [Que15d].

3.2 XML

Mittels verschiedener XML-Dateien lässt sich die Darstellung einer Applikation anpassen. Das AndroidManifest, Ressourcen, Views und Layouts werden als XML-Dateien repräsentiert. Eine spezielle XML-Datei ist das

AndroidMaifest.xml. Listing 1 bildet die in dieser Arbeit verwendete Manifest-Datei ab, um beispielhaft die benötigten Deklarationen anzuzeigen. Diese XML-Datei beinhaltet den Namen der Applikation und deklariert die Java-Klasse, welche zu Beginn der Applikation gestartet werden soll. In diesem Beispiel wird in den Zeilen 14 bis 23 die MainActivity als Start-Activitiy festgelegt. Zusätzlich enthält das AndroidManifest Informationen über unterstütze Android Versionen. In den Zeilen 6 bis 8 wird das minimale und maximale API-Level angegeben, auf welchem die Applikation funktionieren soll. Die Definition *android:screenOrientation="portrait"* legt für jede Activity die Orientierung des Bildschirmes fest. In diesem Fall wird die Orientierung *portrait* gewählt.
Jede Applikation benötigt eine AndroidManifest-Datei [BF12].

Listing 1: AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/
      apk/res/android"
3      package="com.yessi.yaniv2"
4      android:versionCode="1"
5      android:versionName="1.0" >
6      <uses-sdk
7          android:minSdkVersion="9"
8          android:targetSdkVersion="21" />
9      <application
10         android:icon="@drawable/logo_yaniv_black"
11         android:label="@string/app_name"
12         android:theme="@style/AppTheme" >
13         <activity
14             android:name=".MainActivity"
15             android:configChanges="orientation|
                   keyboardHidden"
16             android:label="@string/app_name"
17             android:screenOrientation="portrait" >
18             <intent-filter>
19                 <action android:name="android.intent.
                       action.MAIN" />
20                 <category android:name="android.intent
                       .category.LAUNCHER" />
21             </intent-filter>
22         </activity>
23         <activity
24             android:name=".GameActivity"
25             android:configChanges="orientation|
                   keyboardHidden"
```

```
26                  android:label="@string/app_name"
27                  android:screenOrientation="portrait" >
28              </activity>
29              <activity
30                  android:name=".ScoresActivity"
31                  android:configChanges="orientation|
                        keyboardHidden"
32                  android:label="@string/app_name"
33                  android:screenOrientation="portrait" >
34              </activity>
35          </application>
36      </manifest>
```

3.3 Views and Layouts

Android organisiert alle User-Interface Objekte als Views oder Layouts. Alle sichtbaren Elemente, wie Buttons, Labels oder Text-Boxen, werden als View dargestellt. Layouts gruppieren Views und ordnen sie an. Es existieren verschiedene Haupt-Layouts, wie beispielsweise das LinearLayout oder das TableLayout. Das LinearLayout ist eines der einfachsten und demzufolge eine der weit verbreitetsten Layout-Varianten. Es arrangiert seine untergeordneten Objekte vertikal oder horizontal nebeneinander. Das TableLayout ordnet seine unterordneten Views in einzelnen Spalten an und bildet dadurch eine Tabelle.

3.4 Interaktion des Nutzers

Eine von einer View abgeleitete Klasse bietet unter anderem eine vorgefertigte Methode, die nützlich für das Abfragen der Eingabe ist. Dabei handelt es sich um die Methode *onTouchEvent(MotionEvent)*. Diese Methode erhält als Eingabeparameter ein InputEvent. Als Input kann
$ACTION_DOWN$, $ACTION_MOVE$ und $ACTION_UP$ erfolgen.
$ACTION_DOWN$ erkennt, welche Stelle des Bildschirms angewählt wurde. $ACTION_UP$ registriert daraufhin das Loslassen dieser Stelle, während $ACTION_MOVE$ eine Verschiebung des Eingabegerätes erkennt. Mittels dieser Abfragen können die häufigsten Gesten sehr schnell erkannt und dementsprechend auf die präzise Eingabe reagiert werden.

3.5 Activities

Eine Activity repräsentiert in der Regel einen einzigen Screen (Bildschirm), welcher dem Benutzer zu einem Zeitpunkt auf seinem Endgerät dargestellt wird. Eine Anwendung besteht normalerweise aus mehreren Acti-

vities. Der Nutzer kann zwischen ihnen hin und her wechseln. Bei Activities handelt es sich folglich um die sichtbaren Teile einer Anwendung. Meist existiert eine Main-Activity, welche oftmals die zuerst sichtbare Activity nach dem Starten ist [Gar11]. Die Activity kann als Benutzeroberfläche gesehen werden, da sie auf die Eingaben des Nutzers reagiert und eventuelle Ausgaben erzeugt. Die Activity übernimmt dementsprechend einen großen Teil der Anwenderlogik. Um auf Eingaben reagieren zu können, wird jeder Activity eine View zugeordnet, die die jeweiligen Informationen über die Bildinhalte gespeichert hat [Que15d].

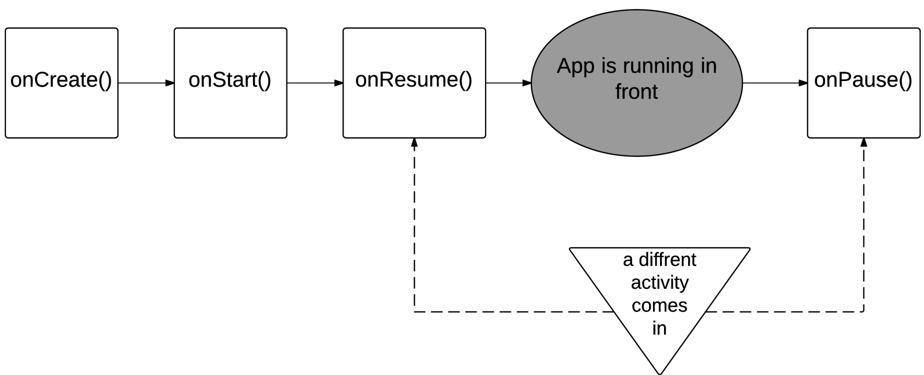


Abbildung 1: Activity-Life-Circle

Das Starten einer Aktivität kann zeitintensiv sein und viele Ressourcen beanspruchen. Unter Umständen kann es zu der Schaffung eines neuen Linux-Prozesses, neuer Zuweisungen aller User-Interface-Objekte oder einer kompletten Neuzeichnung des gesamten Bildschirms führen. Daher existiert für jede Activity ein Activity-Life-Circle , welcher die Lebensdauer einer Activity regelt. Der gesamte Zyklus soll nur einmal beim erstmaligen Starten der Aktivität aufgerufen werden. Wenn die Aktivität nicht komplett geschlossen wird, soll der bereits hergestellte Zustand gespeichert bleiben und dadurch das erneute Starten schneller erfolgen. Eine Verschwendung von Ressourcen wird mittels des Activity-Life-Circle umgangen. Dieser ist für die Erstellung, Zerstörung und die Verwaltung von Aktivitäten zuständig. In Abbildung 1 ist dieser sehr verkürzt dargestellt. Dabei liegt das Augenmerk auf den wichtigsten Methoden [Sim12]. Der komplette Zyklus ist online auf der offiziellen Android-Seite zu finden [Que15a].

Die *onCreate()*-Methode wird aufgerufen, wenn die Activity zum ersten Mal geöffnet wird. Im Regelfall werden hier Layouts und Konfigurationen erstellt. Wenn die Activity auf dem Bildschirm erscheint, wird die *onStart()*-Metode aufgerufen. Wird die Activity durch etwas Unerwartetes, wie bei-

spielsweise einen Telefonanruf, unterbrochen, wird diese durch *onPause()* oder *onResume()* in einen Pause-Zustand versetzt. Ein erneutes Starten der Applikation wird schneller erfolgen, da im Speicher bereits die durch *onCreate()* erzeugten Daten gespeichert sind.

3.6 Shared Preferences

Das Implementieren eines Spieles auf einem mobilen Endgerät kann unter Umständen zu inhärenten Problemen führen, da das Gerät zusätzlich für viele andere Anwendungen verwendet wird. Es ist nicht auszuschließen, dass das Spiel durch einen externen Telefonanruf oder ähnliches unterbrochen wird. In solch einem Fall und auch in dem generellen Fall der Schließung der Applikation sollen mitunter einige Informationen gespeichert werden [Jam12]. Für diese Intention wird von Android die Interface-Klasse Shared Preferences bereitgestellt. Diese kann verschiedene Daten speichern und modifizieren. Die Werte bleiben so lange gespeichert, bis sie überschrieben werden, oder die komplette App deinstalliert wird. Sie sind demzufolge nicht vom Activity-Life-Circle betroffen, sondern sind gesondert zu betrachten. Unter dem Namen *Shared Preferences* ist dabei keinesfalls zu verstehen, dass alle Applikationen auf die gespeicherten Daten zugreifen können. Lediglich diese eine Applikation besitzt Zugriff auf die jeweiligen Informationen. Jede Komponente dieser Applikation besitzt den Zugriff auf jene Daten [Gar11].

Einfache Datentypen, wie Strings oder Integer-Werte können als Programmeinstellung in Shared Preferences gespeichert werden. So kann beispielsweise der Name des Spielers dauerhaft gesichert werden, bis eben jener überschrieben wird. Es ist somit nicht von Nöten, dass der Nutzer bei jedem Start seinen Nutzernamen erneut eingeben muss.
Highscore-Werte werden meist mit Shared Preferences in Android Spielen realisiert. Ergänzend hierzu befindet sich im Kapitel 6.8 die Beschreibung der Highscore-Umsetzung dieser Applikation.

4 Yaniv Spielgregeln

Die Spielregeln des hebräischen Kartenspiels **Yaniv** werden in diesem Kapitel erläutert und zusätzlich anhand eines Diagramms verdeutlicht.

Es wird ein 54er Standard-Kartendeck bestehend aus 52 Spielkarten und zwei Jokern benötigt. Jede Karte des Kartendecks hat einen bestimmten Kartenwert. Das Ass zählt 1, die Zahlen von 2 bis 10 erhalten jeweils ihren eigenen Wert, alle Bilder werden mit 10 gezählt und die Joker werden jeweils mit 0 Punkten gewertet.
Ziel des Spieles ist es in jeder Runde, so wenig Punkte wie möglich zu sammeln. Hat ein Spieler fünf Punkte oder weniger auf der Hand, so darf dieser im nächsten Zug **Yaniv** sagen. Besitzt kein anderer Spieler weniger oder gleich viele Punkte, so gewinnt ersterer und erhält null Punkte auf sein Punkte-Konto.
Ein Flussdiagramm in Abbildung 2 stellt den vereinfachten Spielablauf dar.
Im folgenden werden die einzelnen Komponenten einer **Yaniv**-Runde erläutert:
Jeder Spieler erhält zu Beginn fünf Karten. In der Mitte des Spielbrettes befindet sich jeweils ein Zieh-Stapel und eine offene Karte, welche den Ablage-Stapel symbolisiert. Nacheinander dürfen die Spieler jeweils ihren Zug ausführen, bis ein Spieler **Yaniv** sagt und somit die Runde beendet.

Ein Zug besteht entweder aus zwei einzelnen Schritten oder aus dem Ausruf **Yaniv**. Die Bedingung, um **Yaniv** sagen zu dürfen besteht darin, dass der Spieler fünf Punkte oder weniger auf seiner Hand hält. Wenn kein anderer Spieler **Assaf** sagt, so ist diese Runde beendet und der Spieler gewinnt jene. Der Spieler bekommt ein Smiley, vergleichbar mit null Punkten, zu seinem Punktekonto addiert. Alle anderen erhalten jeweils den Wert ihrer Hand. Die nächste Runde darf der Gewinner beginnen. Wird jedoch **Assaf** gesagt, bedeutet dies, dass ein Spieler weniger oder gar gleich viele Handpunkte besitzt, wie der Spieler, der **Yaniv** sagte. In diesem Fall gewinnt der Spieler, der **Assaf** sagte. Dieser erhält trotzdem seine Handpunkte zu seinem Punktekonto, er darf allerdings die nächste Runde beginnen. Die Person, die **Yaniv** sagte, verliert nicht nur diese Runde, sondern bekommt zusätzlich 30 Strafpunkte und seine eigenen Handpunkte zu seinem Punktekonto addiert.
Jeder Zug ist in zwei Schritte aufgeteilt. Durch geschickte Züge kann ein Spieler fünf oder weniger Punkte sammeln und hat dadurch die Chance die Runde zu gewinnen.
Schritt 1: Abwerfen!
Es muss immer mindestens eine Karte abgeworfen werden. Mehrere Karten können nur abgeworfen werden, wenn sie Pärchen, Drillinge, Vierlinge des gleichen Kartenwertes (z.B. Herz König und Karo König bilden ein

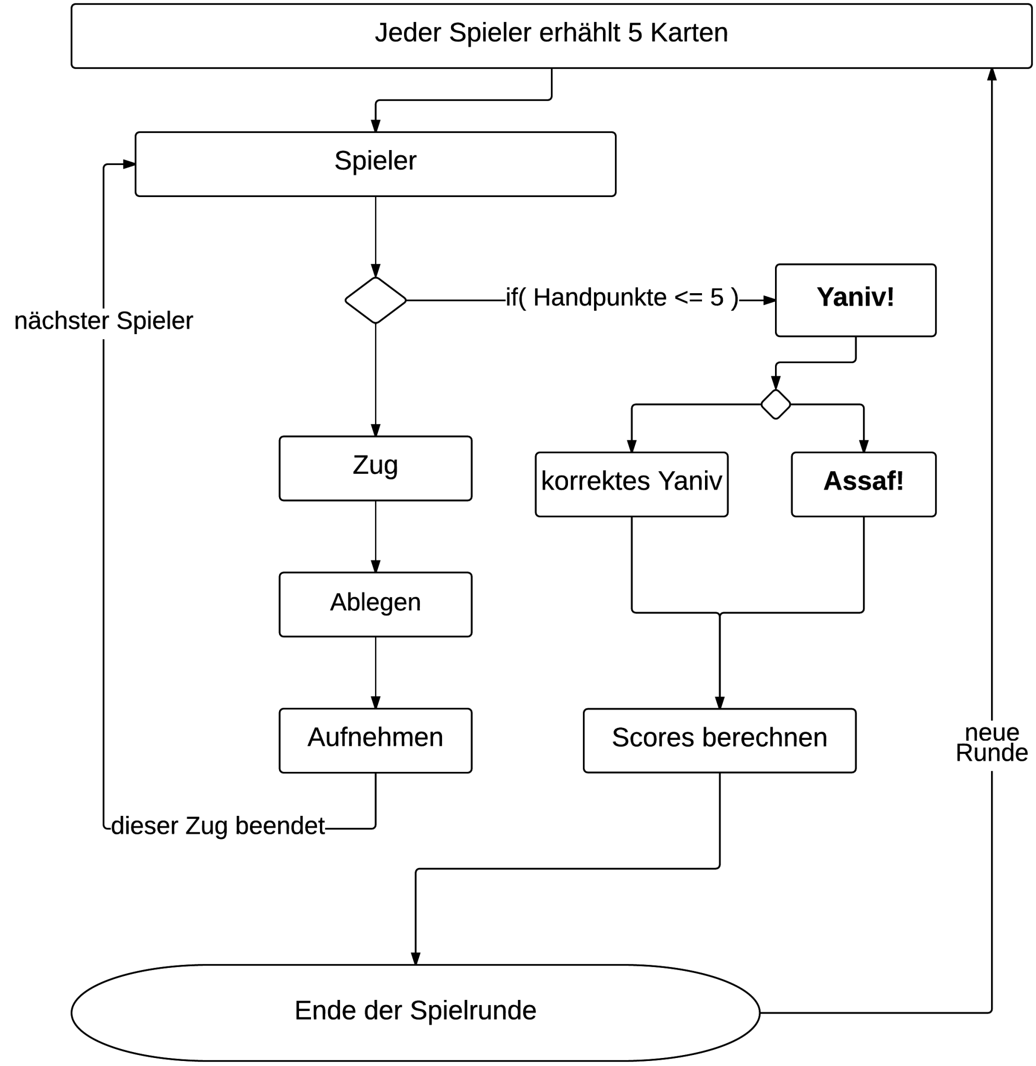


Abbildung 2: Flussdiagramm Spielrunde

Pärchen) oder Straßen bilden. Eine Straße besteht aus mindestens drei aufeinanderfolgenden Karten **einer** Farbe (z.B. Kreuz 3, Kreuz 4, Kreuz 5). Ein Joker kann anstelle einer dieser Karten abgelegt werden (z.B. Kreuz 3, Joker, Kreuz 5). Das Ablegen mehrerer Karten ermöglicht es, die Karten Anzahl der Hand zu verringern.
Schritt 2: Aufnehmen!
Nachdem eine oder mehrere Karten abgelegt wurden, muss immer genau **eine** Karte aufgenommen werden. Dabei hat der Spieler die Wahl zwischen dem regulären Zieh-Stapel oder dem Ablage-Stapel. Wird der Zieh-Stapel

gewählt, so erhält er die oberste umgedrehte Karte von eben jenem Stapel. Entscheidet er sich für den Ablagestapel, so darf immer nur die oberste Karte gewählt werden. Wurden mehrere Karten abgelegt, beispielsweise eine Straße, so darf nur eine der äußeren Karten gezogen werden. (z.B. Kreuz 3, Kreuz 4, Kreuz 5, dann darf nur zwischen Kreuz 3 und Kreuz 5 entschieden werden). Die mittlere Karte muss liegen bleiben.

Nach jeder Runde werden die jeweils erzielten Punkte auf einen totalen Punktestand (Score) addiert. Überschreitet ein Spieler dabei 200 Punkte, so endet das gesamte Spiel und keine weitere Runde wird gespielt. Dieser Spieler ist der Verlierer des gesamten Spieles. Der Spieler, welcher unter dieser Punktzahl liegt, ist der Gewinner. Überschreitet die Spieler-Anzahl die Zahl Zwei, so wird so lange gespielt, bis nur ein Spieler einen Score unter 200 Punkten besitzt. Jene Spieler, die diese Zahl überschritten haben, werden ab diesem Zeitpunkt ausgeschlossen. Die Score-Berechnung verfügt über Sonderregelungen. Erreicht ein Spieler genau 50, 100, 150 oder 200 Punkte, so werden ihm jeweils 50 Punkte von seinem Score abgezogen. Demzufolge ist es sinnvoll, neben dem eigentlichen Spielablauf auf die Scores zu achten.
Es ist möglich, das gesamte Spiel zu gewinnen, ohne eine einzige Runde mit einem korrekten **Yaniv** abgeschlossen zu haben.

5 Entwurf

Das Konzept ist angelehnt an das *Model-View-Contorller* (kurz *MVC*) Design Pattern. Daher wird dieses im folgenden Kapitel kurz erläutert und dargestellt. Anschließend wird das dem prototypischen Spiel zugrunde liegende Konzept abgebildet und in Bezug zu dieser Design-Vorlage gesetzt.

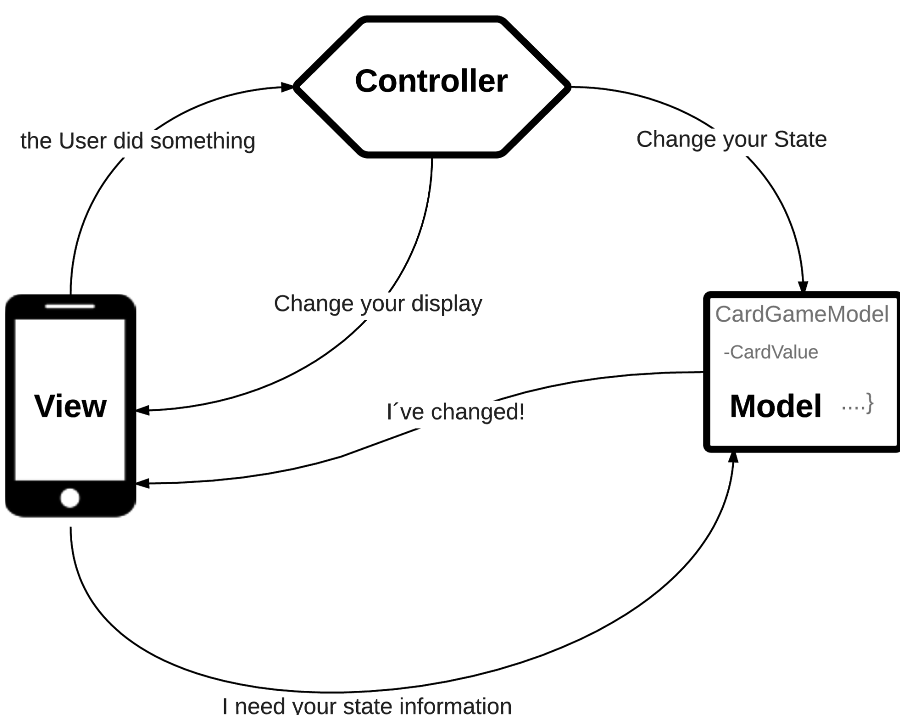


Abbildung 3: Design Pattern Model View Controller

5.1 Model View Controller

Das *Model View Controller* Konzept besteht aus drei großen Hauptbereichen. Das Modell, die Spiellogik und das UserInterface bilden die drei Komponenten. Eine Trennung dieser ist bei der Modellierung sinnvoll, da dadurch eine erleichterte Anpassung an andere Kartenspiele ermöglicht wird. Soll eine der Komponenten verändert oder ausgetauscht werden, so ist dies problemlos möglich. Zusätzlich wird eine bessere Verständlichkeit, Flexibilität und Wartbarkeit des Programmes erzeugt. Dieser Ansatz ermöglicht es, ein *Model* mit verschiedenen Ansichten und Präsentationen zu erzeugen. Es kann eine neue Ansicht des *Models* geschrieben werden, ohne das eigentliche *Model* zu verändern [GHJV94, MDMN13]. Abbildung 3 zeigt

ein Diagramm der Funktionsweise des *MVC* Pattern [FFSB04]. Der *Controller* übernimmt die Funktion der zentralen Steuerung der Applikation. Er registriert Änderungen im *Model* und passt diese der gezeigten *View* an. Das UserInterface stellt die Grafikkomponente dar, welche über eine direkte Verbindung zum *Controller* steht. Sie registriert User Eingaben und leitet die Inputs an den *Controller* mittels des Observer-Patterns weiter. Dabei handelt es sich um ein Entwurfsmuster aus der Softwareentwicklung. Es gibt Änderungen eines Objektes an die von diesem Objekt abhängigen Strukturen weiter. Der *Controller* verarbeitet diese Befehle und passt die Grafikkomponente dementsprechend an. Es kann zusätzlich vorkommen, dass der *Controller* der *View* vorschreibt, wie sie sich zu verändern hat. Ein einfaches Beispiel ist das Einblenden oder Ausblenden eines Buttons, welcher über diese Vorgehensweise gesteuert wird. Die *View* erhält zusätzlich Informationen vom *Model* und kann so auf direkte Änderungen im *Model* reagieren und sich neu zeichnen. Das *Model* enthält alle Anwendungsdaten und die Anwendungslogik. Es ist zwischen den vorherigen Komponenten angesiedelt. Es stellt eine Schnittstelle bereit, welche es erlaubt Zustände abzurufen, Daten zu manipulieren und Benachrichtigungen über Statusveränderungen senden zu können [FFSB04].

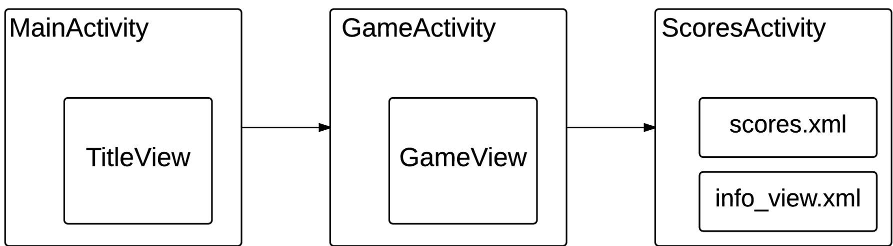


Abbildung 4: Ablauf Konzept Activities

5.2 Konzept

Das protoypische Spiel besteht aus drei Activities. Abbildung 4 verdeutlicht die Reihenfolge der aufgerufenen Activites. Das *AndroidManifest.xml* beinhaltet alle verwendeten Activities und legt die *GameActivity* als Start-Activity fest. Diese beinhaltet als View die *TitleView*. Dabei handelt es sich um das Hauptmenü, welches direkt nach dem Starten der Activity auf dem Endgerät angezeigt wird. Wird der Play Button, der einen auswählbaren Bestandteil des Menüs darstellt, ausgewählt, wird die *GameActivity* aktiviert, welche die *GameView* enthält. Diese stellt den Hauptbildschirm des Spieles dar. Abbildung 7 in Kapitel 6.2 bildet diesen Bildschirm ab. Aus dieser *GameActivity* kann die *ScoreActivity* gestartet werden. Die *ScoreActi-*

Abbildung 5: Konzeption

vity, welche die Highscore Tabelle als Layout anzeigt, kann auf zwei verschiedene Weisen aufgerufen werden. Wird eine Runde beendet, so startet sie automatisch. Andernfalls öffnet ein Score-Button, welcher innerhalb des Spielfeldes platziert ist, diese Ansicht. Die detaillierte Beschreibung der *ScoreActivity* findet sich in Kapitel 6.8 wieder.

Eine grobe Übersicht der architektonischen Verteilung der Bestandteile und deren Klassen bietet Abbildung 5. Das Kartenmodell, welches in der Abbildung oben links angeordnet ist, beherbergt die Klasse des Kartenobjektes, welches zusätzlich über die Enums *CardType* und *CardValue* verfügt. Dieses Konzept sieht keine Veränderung des Kartenmodells vor. Lediglich die Positionen der Kartenobjekte können verändert werden. Das *Model* wird zu Beginn des Spieles einmal komplett erzeugt und anschließend nur verwendet. Eine Änderung ist nicht erlaubt, da keine Karte während des Spielens ihren Wert oder ihre Textur verändern soll.

Die Activities sind unten links in der Grafik abgebildet. Sie regeln den Ablauf der Aufrufe für die sichtbaren Teile der Applikation. Sie starten die für die Darstellung zuständigen Views. Die *GameView* beherbergt zusätzlich

zur Darstellung des Spielfeldes, fast die gesamte Spiellogik. Lediglich der Computerplayer ist ausgelagert. Die Computerlogik und die Logik aus der *GameView* bilden gemeinsam den Controller, der auf Benutzereingaben reagieren und anschließend den Befehl für das Neu-Zeichnen geben kann. Die Auslagerung der gesamten Spiellogik wäre problemlos möglich und dadurch wäre das Prinzip des Model View Controller Design Patterns *MVC* vollends erfüllt. Die so eng verbundene Kombination der Komponenten View und Spiellogik wird hier verwendet, da Android bereits vorgefertigte Funktionen zum Abfangen der Benutzereingabe in Views bietet. So kann die Spiellogik direkt auf die Eingaben reagieren.

Das Spielfeld wird als eine eigene View implementiert, da so das hin und her blättern zwischen Menü und Spielfeld ermöglicht wird.

6 Implementierung

Das Implementieren der Spiellogik eines Kartenspiels beinhaltet häufig dieselben Bestandteile. So besitzen die meisten klassischen Kartenspiele die Aspekte des Spielstarts, der Computerlogik, der Überprüfung eines korrekten Zuges und des Spiel- oder Rundenendes. Zusätzlich kann ein Spiel über eine Scoreanzeige verfügen. Im Folgenden soll beispielhaft an dem Spiel **Yaniv** eine Möglichkeit aufgezeigt werden, wie diese Aspekte der Spielimplementation umgesetzt werden können. Des Weiteren soll die Umsetzung der Erstellung und Repräsentation der Karten, der Festlegung der Spielerreihenfolge und die Ermittlung und Darstellung der Scores erläutert werden.

♦	♥	♠	♣	VALUE
1	14	27	40	1
2	15	28	41	2
3	16	29	42	3
4	17	30	43	4
5	18	31	44	5
6	19	32	45	6
7	20	33	46	7
8	21	34	47	8
9	22	35	48	9
10	23	36	49	10
11	24	37	50	10
12	25	38	51	10
13	26	39	52	10
53				0
54				0

Abbildung 6: Übersicht über die Verteilung der ID´s und den dazugehörigen Karten Values

6.1 Kartenmodell

Jede Karte verfügt über eine Farbe und einen Kartenwert. Das Ass zählt 1, die Zahlen ihren jeweiligen Kartenwert und alle Bilder erhalten den Kartenwert 10. Beide Joker werden mit 0 als Wert belegt. Zusätzlich verfügt jede Karte über ihre eigene ID. Diese wird bei der Instanziierung festgelegt und kann anschließend nicht mehr verändert werden. Grafik 6 visualisiert die Verteilung der Karten ID´s und deren Kartenwerte. Anhand der ID werden Kartenwert und Kartenfarbe ermittelt und anschließend die passende Textur zugewiesen. Die Kartenfarbe wird wie folgt festgestellt: Bei Karten-ID´s zwischen 1 und 13 wird Karo vergeben, bei 14 bis 26 Herz, 27 bis 39 Pik, 40 bis 52 erhalten Kreuz und die Joker sind mit den ID´s 53 und 54 belegt. Der Wert einer Karte wird mittels einer Modulo-Division ermittelt. Beide Zuweisungen sind von großer Bedeutung, da so leichter spätere Aktionen ausgeführt werden können. Die Ermittlung eines validen Zuges basiert auf diesen.

6.2 Spielstart

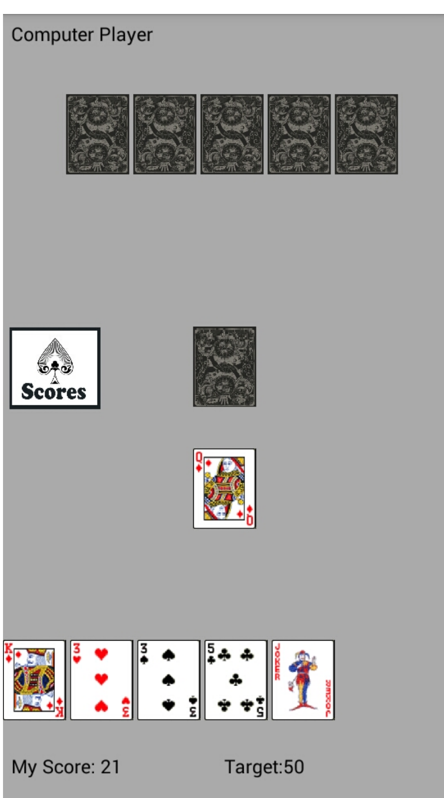


Abbildung 7: Darstellung des Spielbildschirms

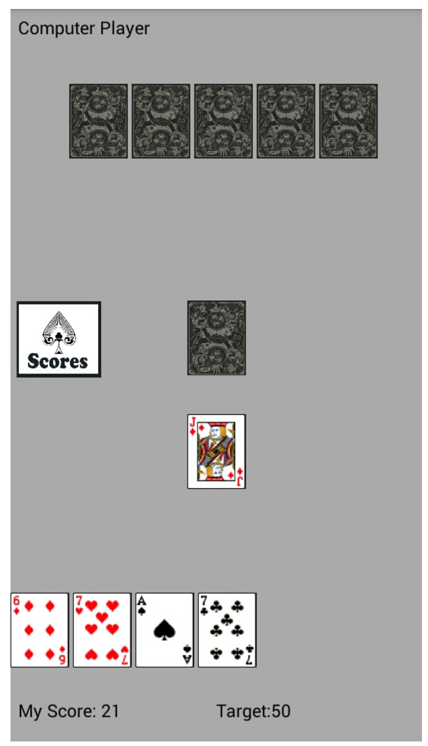
Die Methode *initCards()* erstellt das gesamte Kartendeck mit insgesamt 54 Karten. Jede Karte verfügt über eine eigene ID und bekommt mittels dieser das passende Bitmap als Textur zugewiesen. Anschließend sorgt die Methode *dealCards()* dafür, dass das Kartendeck gemischt wird und jeder Spieler fünf Karten erhält. Die verschiedenen Draw-Methoden zeichnen die verteilten Karten an ihre zugewiesenen Positionen. Diese Funktionen berücksichtigen die Kartenanzahl und zeichnen abhängig von dieser die Karten an die passenden Positionen. Der Zieh-Kartenstapel, so wie die Karten des Gegners werden lediglich mit der Textur der Kartenrückseite belegt. Die eigene Hand wird permanent offen angezeigt.
Abbildung 7 zeigt exemplarisch den Spielbildschirm. Die fünf verdeckten Karten symbolisieren die Handkarten des Computers. Auf der linken Seite, in der Mitte des Spielbildschirms befindet sich ein Score-Button. In der Bildschirmmitte befindet sich verdeckt der Zieh-Kartenstapel und etwas versetzt nach unten, ist der Ablagestapel zu sehen, welcher in diesem Fall eine Karte (Karo Dame) beinhaltet. Unten links sind die fünf Handkarten des Spielers abgebildet. Unter diesen Karten stehen Informationen für den Spieler. *MyScore* errechnet die aktuellen Handpunkte und *Target* ermittelt die Anzahl der Punkte, die für einen Score-Bonus benötigt werden. Siehe hierzu Kapitel 6.8.

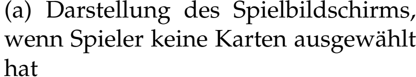
6.3 Ermittlung der Spielerreihenfolge

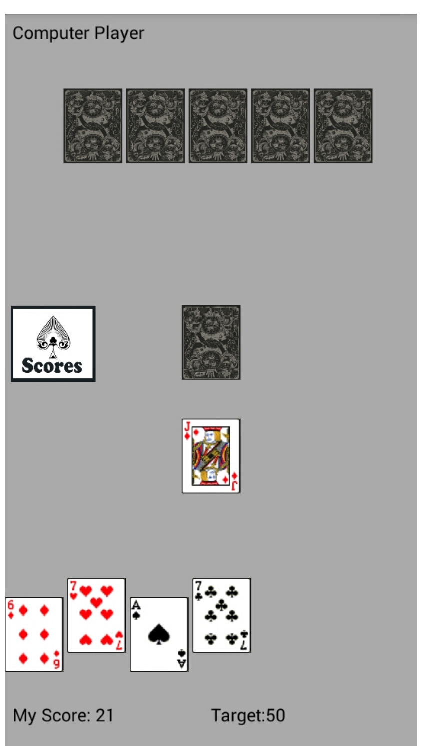
Die prototypische **Yaniv**-Applikation ermöglicht momentan das Spielen eines Spielers (Player) gegen einen Computerspieler. Eine boolesche Variable regelt die Reihenfolge. Sobald ein neues Spiel startet, wird diese Variable zufällig gesetzt. Ist diese true, so beginnt der Spieler, andernfalls darf der Computerspieler seinen ersten Zug ausführen.
Während einzelner Spielrunden entscheidet der Ausgang einer vorherigen Runde über den Starter der neuen Runde. Ist ein korrektes **Yaniv** gespielt worden, so darf der Spieler, der diesen Ausruf tätigte, beginnen. Ist ein korrektes **Yaniv** durch ein **Assaf** verhindert worden, so darf jener Spieler beginnen, welcher **Assaf** spielen konnte.
Ist ein Zug beendet, so wird diese Variable verändert und der nächste Spieler darf seinen Zug ausführen.

6.4 Valider Zug

Bevor ein weiterer Spielzug geschehen kann oder die Veränderung des aktuellen Spielzugs gezeichnet wird, muss überprüft werden, ob es sich um einen validen Zug handelt. Als Erstes wird überprüft, ob der Spieler überhaupt an der Reihe ist. Andernfalls darf dieser keinen Zug ausführen. Ist ein Spieler an der Reihe, muss dieser mindestens eine Karte seiner Hand auswählen, welche er ablegen möchte. Die Auswahl erfolgt über die Me-




(a) Darstellung des Spielbildschirms, wenn Spieler keine Karten ausgewählt hat



(b) Angewählte Karten werden höher gezeichnet, um Auswahl darzustellen

Abbildung 8: Visualisierung der Auswahl der Karten

thode *prepareCardForMovement()*, welche den Status der ausgewählten Karte verändert. Der Status besteht aus einer booleschen Variable. Ist der Status auf true gesetzt, so möchte der Spieler diese Karte auswählen. Ist eine Karte ausgewählt, so wird sie höher gezeichnet, als die restlichen Handkarten. Visualisiert wird dies in Abbildung 8. Durch erneutes Anwählen einer ausgewählten Karte, wird der Status wieder auf false gesetzt und die Karte an ihrer ursprünglichen Position gezeichnet. Andernfalls bleibt diese Variable false und sie wird nicht bei der Überprüfung eines validen Zuges berücksichtigt. Der Spieler kann beliebig lange Karten aus- und abwählen. Er beginnt seinen wirklichen Zug erst mit der Auswahl der Aufnahmekarte. Welche Möglichkeiten ein Spieler bei der Aufnahme besitzt, wird im Folgenden Kapitel 6.5 (Aufnehmen einer Karte) näher beschrieben. Vorerst wird sich darauf beschränkt, dass eine Aufnahmekarte ausgewählt wird. Diese Auswahl startet die Abfragemechanismen, ob ein valider Zug vorliegt.

Die zur Validierung aufgerufene Methode *checkForValidDraw()* überprüft, ob die ausgewählten Karten überhaupt gemeinsam abgelegt werden dürfen. Zuerst wird die Anzahl der ausgewählten Karten über ihren Status ermittelt und anschließend eine Fallunterscheidung durchgeführt. Abbildung 9 visualisiert die Vorgehensweise der Überprüfung. Der Buchstabe *i* repräsentiert in der Grafik die Anzahl der ausgewählten Handkarten. Handelt es sich lediglich um eine ausgewählte Karte, so bestätigt die Methode bereits an dieser Stelle, dass es sich um einen korrekten Zug handelt. Andernfalls müssen weitere Bedingungen überprüft werden. Zulässig sind außer einzelnen Karten Pärchen, Drillinge und Vierlinge. Sind also mehrere Karten gewählt, werden die jeweiligen Karten-Values miteinander verglichen und daraufhin entschieden, ob es sich um einen validen Zug handelt. Das Ablegen einer Straße ist zusätzlich erlaubt. Eine Straße benötigt mindestens drei Karten einer Farbe mit einem ansteigenden Kartenwert (z.B. Kreuz 3, Kreuz 4, Kreuz 5). Daher wird die Auswahl, die mindestens drei Karten beinhaltet, auf eine mögliche Straße untersucht. Falls die Kartenfarbe bei allen Karten übereinstimmt, wird die Auswahl der Größe nach sortiert. Zudem werden die jeweiligen ID´s betrachtet, ob diese lückenlos aufeinander folgen. Ist dies der Fall, können diese Karten abgelegt werden.

Falls die Überprüfung herausfindet, dass keine regelkonforme Auswahl der Ablage-Karten erfolgt ist, wird der Spieler mit einem Toast: „You have an invalid play! “informiert. Dieser wird nur kurze Zeit eingeblendet und überlagert dabei die eigenen Hand-Karten.

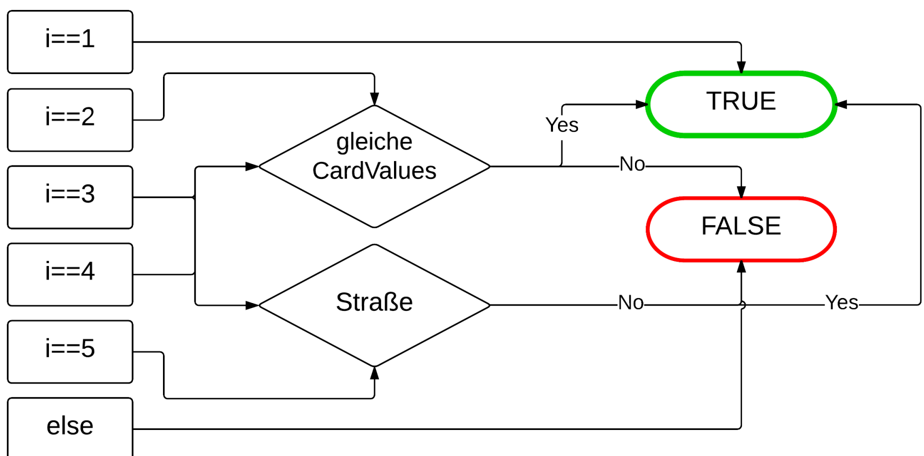


Abbildung 9: Visualisierung der Funktionsweise *checkForValidDraw()*

6.5 Aufnehmen einer Karte

Ist eine Ablagekombination als valide erkannt, muss der Spieler eine Karte ziehen. Hierbei besitzt er zwei Möglichkeiten. Zum einen kann er die oberste verdeckte Karte des Zieh-Stapels wählen, oder er entscheidet sich für den Ablagestapel. Das Ziehen einer Karte des Zieh-Stapels erfordert keine weiteren Abfragen, weswegen die oberste Karte ohne weitere Bedingungen der Spieler-Hand hinzugefügt wird. Der Zug des Spielers ist somit beendet und der Computerspieler ist an der Reihe.
Wählt der Spieler den Ablagestapel, so müssen zusätzlich einige Abfragen erfolgen. Nicht jede Karte des Ablage-Stapels darf aufgenommen werden. Besteht der Ablagestapel aus nur einer einzigen Karte, so ist diese Karte immer wählbar. Liegen dort Pärchen, Drillinge oder Vierlinge, so darf der Spieler sich eine dieser Karten aussuchen. Die Position der Karte und die jeweilige Farbe spielt hierbei keine Rolle. Ist eine Karte zugehörig zu einer Straße, so darf sie nur als Zieh-Karte ausgewählt werden, wenn sie am Rand dieser Straße liegt. Listing 2 zeigt einen Ausschnitt aus der View-Klasse GameView und demonstriert die Abfrage nach einer korrekten Zieh-Karte für den Sonderfall einer Straße. Besteht eine Straße beispielsweise aus vier Karten, so darf nur die erste oder die vierte Karte ausgewählt werden. Die Karten zwei und drei wären demzufolge keine möglichen Optionen. Die Code Zeilen in Listing 2 werden nur durchlaufen, wenn der Ablagestapel mehr als drei Karten beinhaltet. Ist dies der Fall, erfolgt in Zeile 2 eine Abfrage, ob die als Liste *openDiscardPile* repräsentierten Karten des Ablagestapels eine Straße bilden. Handelt es sich um eine Straße, wird in Zeile 3 überprüft, ob die ausgewählte Karte den ersten Index 0 besitzt, oder ob es sich um die letzte Karte dieser Liste handelt. Die letzte Indexstelle wird mittels *openDiscardPile.get (openDiscardPile.size()-1)* ermittelt. Falls die gewählte Karte keinen dieser Indizes besitzt, ist der Zug nicht regelkonform und es erscheint ein Toast auf dem Bildschirm mit folgendem Informationstext: „Choose a card by the edges! “
Ist die Auswahl der Karte des Ablagestapels valide, so wird sie der Hand hinzugefügt und der Computerspieler kann seinen Zug starten.

Listing 2: Auszug aus der Klasse GameView

```
1  // is the pile a street?
2  if (isStreet(openDiscardPile)) {
3   if (temp != openDiscardPile.get(0) && temp != (
       openDiscardPile.get(openDiscardPile.size() - 1)))
       {
4    Toast.makeText(myContext,"Choose_a_card_by_the_edges
        !",
5    Toast.LENGTH_SHORT).show();
6    break;
7   }
8  }
```

6.6 Computer-Logik

Der Computergegner beschränkt sich auf eine simple künstliche Intelligenz. Ist der Computergegner an der Reihe, hat er zwei Möglichkeiten: Sobald seine Handpunkte fünf oder weniger betragen, beendet er die Runde durch **Yaniv**. Andernfalls muss er die Phasen: Ablegen und Aufnehmen durchlaufen. In beiden Phasen werden seine Möglichkeiten auf ein Minimum beschränkt. Diese Entscheidung liegt darin begründet, dass erst eine funktionierende künstliche Intelligenz erschaffen wird, die als erste Version fungieren soll, um das Spielen zu ermöglichen. Weitere optionale Schwierigkeitsstufen können an dieser Stelle eingebaut werden. Das Erschaffen einer umfassenden Intelligenz ist nicht Bestandteil dieser Arbeit und wird demzufolge nicht weiter betrachtet.
Die Ablege-Phase wird folgendermaßen realisiert:
Der Computergegner ermittelt die valide Ablagekombination seiner Karten mit der höchsten Punktzahl. Dabei wird die mögliche Straßenbildung aus seinen Karten außer Acht gelassen. Es wird sich auf die Ablage einer einzelnen Karte oder die Bildung möglicher Pärchen/ Drillinge oder Vierlinge beschränkt. Der Algorithmus ist in Listing 3 abgebildet. Er ermittelt folgendermaßen die höchstmögliche Ablage der Computer Karten:
Die Methode *getHighestNPairs(List<Card>hand)* ermittelt die höchstmögliche Ablagekombination der Computer Karten, welche sich in der Liste *hand* befinden. Sie gibt eine Kartenliste mit den abzulegenden Karten zurück. Zuerst wird in Zeile 3 die Kartenliste *hand* kopiert, damit zunächst nicht die Original Liste verändert wird. Die kopierte Liste wird in der weiteren Beschreibung des Algorithmus *copyHand* genannt. Zusätzlich wird eine neue Kartenliste *highestHand* erstellt und auf *null* gesetzt. Diese Liste wird die spätere Rückgabeliste. Die *while*-Schleife in Zeile 5 wird so lange ausgeführt, bis die *copyHand*-Liste keine Kartenelemente mehr enthält. Der Schleifenkörper erstellt zuerst eine weitere temporäre Kartenliste *temp*

mittels der Funktion *getNPair(List<Card>copyHand)*, welche direkt auf der *copyHand*-Liste arbeitet.

Diese Funktion wird ab Zeile 15 abgebildet. Innerhalb der Methode wird eine Teilliste der gesamten Hand erstellt. Das erste Kartenelement der übergebenen Liste *copyHand* wird einer neuen LinkedList *nPair* hinzugefügt und aus der *copyHand* gelöscht. Anschließend wird über die gesamte *copyHand*-Liste iteriert und jeder Karten-Value mit dem Kartenelement aus *nPair* verglichen. Ist dieser Wert gleich, wird auch dieses Element der Liste *nPair* hinzugefügt. Andernfalls verbleiben diese Elemente in ihrer jetzigen Liste. Abschließend werden alle Elemente, die in der *nPair*-Liste enthalten sind, aus der *copyHand*-Liste entfernt, um das wiederholte Auffinden einer Karte zu vermeiden. Die Liste *nPair* enthält demnach nur Karten des gleichen Karten-Values. Daher kann es sich um eine einzelne Karte, Pärchen, Drillinge oder Vierlinge handeln. Diese gefundene Liste wird zurückgegeben und innerhalb der *while*-Schleife weiter verarbeitet.

Ist die *temp*-Liste nach dem obigen Verfahren erstellt worden, folgt eine Fallunterscheidung. Ist bisher *highestHand* noch nicht belegt worden, so ist dies der erste Schleifendurchlauf und jene Liste wird mit der *temp*-Liste überschrieben. Der Schleifenkopf wird erneut ausgeführt. Andernfalls wird die bereits existierende *highestHand* mit der aktuellen *temp*-Liste verglichen. Dies ist von Bedeutung, da es durchaus vorkommen kann, dass eine einzelne Karte einen höheren Karten-Value besitzt, als ein niedriges Pärchen. So soll der Computerspieler beispielsweise einen einzelnen König als Ablagevariante vorziehen, wenn er zusätzlich noch drei Asse auf der Hand hält. Für diese Abfrage wird die Methode *getCardListValue(List<Card> list)* benötigt. Diese errechnet mittels eines Schleifendurchlaufs in Zeile 29 den Integer-Value-Wert einer Liste. Dadurch lassen sich die einzelnen Teil-Listen der Computer-Hand vergleichen. Die höchste mögliche Ablagekombination wird als Ergebnis-Liste *highestHand* der Funktion *getHighestNPairs(List<Card>hand)* zurückgegeben.

Die Realisierung der Aufnahme-Phase wird folgendermaßen implementiert: Es wird sich zur Vereinfachung darauf beschränkt, dass der Computergegner immer eine verdeckte Karte des Zieh-Stapels aufnimmt. Nachdem beide Phasen abgeschlossen sind, ist der Computer Zug beendet und der Spieler ist wieder an der Reihe.

Listing 3: Auszug aus der Klasse ComputerPlayer

```
1  // ermittelt die die höchste Ablagekombination
2  private List<Card> getHighestNPairs(List<Card> hand) {
3          List<Card> copyHand = new LinkedList<Card>(
              hand);
4          List<Card> highestHand = null;
5  //so lange Elemente in copyHand vorhanden sind:
6          while (!copyHand.isEmpty()) {
7  //befülle Liste mittels Methode getNPair
8                  List<Card> temp = getNPair(copyHand);
9  //handelt es sich um den ersten Schleifendurchlauf:
10                 if (highestHand == null) {
11 //dann setzte temp als Rückgabeliste;
12                         highestHand = temp;
13 //andernfalls vergleiche vorher gefundene Liste mit
      aktueller temporären
14                 } else if (getCardListValue(
                      highestHand) < getCardListValue(
                      temp)) {
15                         highestHand = temp;
16                 }
17         }
18 //gebe Liste mit höchsten Value-Wert zurück
19         return highestHand;
20 }
21 //nehme erstes Element aus der Liste und schaue, ob
      weitere Karten mit gleichem Card-Value in Liste
      existieren
22 //wenn ja speichere alle in einer Liste und remove sie
       von der ursprünglichen Liste
23 private List<Card> getNPair(List<Card> list) {
24         List<Card> nPair = new LinkedList<Card>();
25         Card temp = list.remove(0);
26         nPair.add(temp);
27         for (Card card : list) {
28                 if (card.getValue() == temp.getValue()
                      ) {
29                         nPair.add(card);
30                 }
31         }
32         list.removeAll(nPair);
33         return nPair;
34 }
```

```
35 //summiere die Card-Values aller Karten einer Liste
      auf und gebe diesen Integerwert zurück
36 public static int getCardListValue(List<Card> list) {
37         int value = 0;
38         for (Card card : list) {
39                 value += card.getValue().getValue();
40         }
41         return value;
42 }
```

6.7 Spielende

Besitzt ein Spieler höchstens fünf Punkte, so erscheint in der unteren rechten Ecke des Bildschirms ein **Yaniv** Button. Abbildung 10 zeigt den auswählbaren **Yaniv** Button. Wird dieser angewählt, endet die Runde und die *endHand()*- Methode wird aufgerufen. Diese prüft zunächst, ob es sich um ein korrektes **Yaniv** handelt, oder um **Assaf**. Es wird die Methode *updateScores()* aufgerufen, welche die finalen Runden-Punkte der Spieler ermittelt und zu den bereits gesammelten Punkten addiert. Es wird die Activity *ScoresActivity* gestartet.
Es erscheint eine Text-View, die den Spieler über das Ende und die jeweiligen Punkte aller Spieler informiert. Diese benutzt ein Linear-Layout, um die Elemente vertikal innerhalb eines Dialoges anzuzeigen. Ein Text zur Information über den Spielausgang erscheint. Wird diese Text-View beendet, ist eine Tabelle mit den Punkteständen der Spieler zu sehen. Zusätzlich verfügt diese View über einen Button, der bei Betätigung eine neue Runde einleitet. Dieser Button ruft die Methode *initNewHand()* auf. Diese sorgt dafür, dass alle Karten wieder zum Kartendeck hinzugefügt werden. Anschließend wird neu gemischt und die Karten ausgeteilt. In der *endHand()*-Methode wird außerdem noch über das finale Spielende entschieden. Erreicht ein Spieler 200 Punkte, oder übersteigt er diese, endet das Spiel. Die Text-View wird dahingehend angepasst und der Spieler wird informiert. Daraufhin hat er die Möglichkeit über einen Button ein neues Spiel zu starten.

6.8 Highscore

Ein totaler Punkte-Score der Spieler soll nach Beendigung des Spieles gespeichert bleiben. Da allerdings keine großen Datenmengen benötigt werden, wird auf das Verwenden einer SQL-Datenbank verzichtet. Stattdessen werden Shared Preferences verwendet. Dabei handelt es sich um ein Interface zum Speichern und Modifizieren von Daten, welches Android bereitstellt. Die Werte bleiben so lange gespeichert, bis sie überschrieben werden,

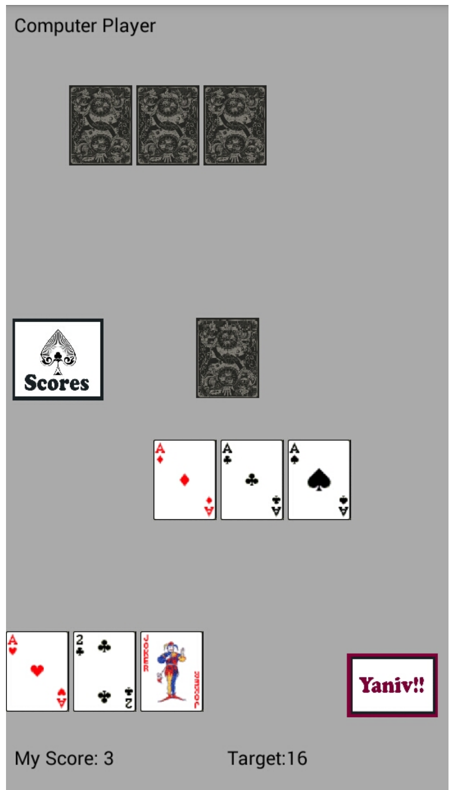
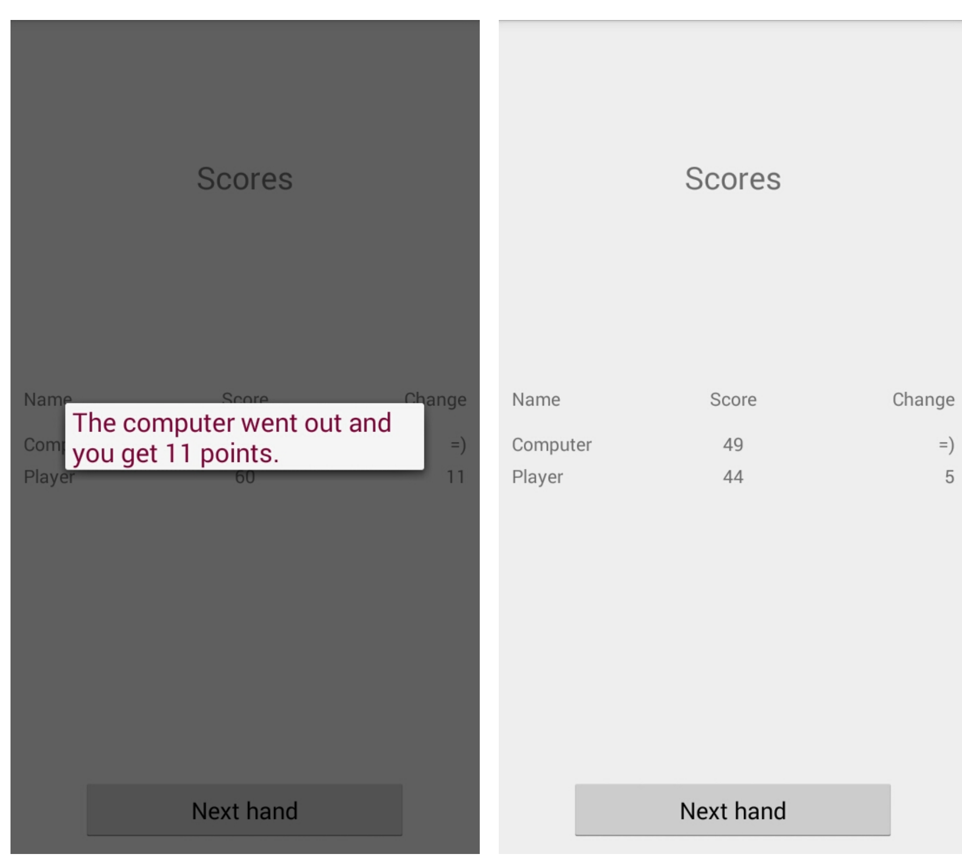


Abbildung 10: Darstellung des Spielbildschirms mit auswählbarem **Yaniv**-Button

oder die komplette Applikation deinstalliert wird. Das Speichern der Shared Preferences benötigt folgende Schritte [ADC13] :

1. Aufrufen einer Instanz einer Shared Preference
2. Erstellen eines Editors, der den Datenwert verändern kann
3. Die gespeicherten Werte befüllen/ abändern
4. Abspeichern der Änderung

Das Anzeigen der Scores wird durch eine eigene Activity umgesetzt. Diese Activity kann auf zwei Arten gestartet werden. Zum einen existiert auf dem Spielbildschirm ein Button mit der Aufschrift Scores, welcher die Scores manuell anzeigen lässt. Zum anderen wird diese Activity automatisch gestartet, wenn die Methode *endHand()* aufgerufen wird. Diese wird gestartet, wenn ein Spieler die Runde durch **Yaniv** beendet. Abhängig von der Art und Weise, wie diese Activity gestartet wird, zeigt sie verschiedene Informationen.
Wird die Activity durch das Auswählen des Score-Buttons aufgerufen, wird



(a) Scoreanzeige bei Beendigung einer Runde mit TextView

(b) Scoreanzeige, nachdem die TextView geschlossen wurde

Abbildung 11: Darstellung der ScoreActivity

die Tabelle mit Spielnamen und den Spielpunkten aller Spieler angezeigt. Abbildung 11 b) zeigt eine vergleichbare Ansicht. Allerdings sind in diesem Fall die Change-Punkte, welche Auskunft über die aktuelle Runde geben, auf 0 gesetzt, damit kein Spieler während des Spielens Auskünfte über die Handpunkte der Gegner erhalten kann. Zudem ist es dem Spieler nicht möglich eine neue Runde zu beginnen, da der entsprechende Button ausgeblendet wird. Der Spieler erhält durch diese Funktion allerdings die Möglichkeit, seine Gesamtpunktzahl und die seiner Gegner einzusehen.
Startet die Activity nach einer beendeten Runde, so wird die obige Tabelle zusätzlich mit einer Text-View überlagert, welche den Spieler über den Ausgang der jeweiligen Runde informiert. Angepasst an diesen Ausgang wird ausgegeben, welcher Spieler die Runde gewonnen hat und wie viele Punkte auf das jeweilige Punktekonto der Spieler gelangen. Wird dieser Text durch Wählen des Zurück-Pfeiles auf dem mobilen Endgerät geschlossen, so erhält der Spieler die komplette Aufsicht auf die Score-Tabelle. Die-

se beinhaltet nun zusätzlich die einzelnen Change-Werte der Spieler. Hat ein Spieler ein korrektes **Yaniv** erzielt, so wird an dieser Stelle des Spielers ein Smiley abgebildet. Andernfalls sind hier die Rundenpunkte der zuletzt gespielten Runde zu finden. Abbildung 11 verdeutlicht beide Ausgänge.

7 Fazit

Im Fokus der vorliegenden Ausarbeitung stand die Entwicklung einer prototypischen Umsetzung einer interaktiven Android Applikation. Zusätzlich sollten an jener Applikation die typischen Hintergrundroutinen eines Spiels exemplarisch an dem hebräischen Kartenspiel **Yaniv** durchleuchtet und eine mögliche Lösungsvariante der Umsetzung aufgezeigt werden.

Entstanden ist eine lauffähige Android Applikation, welche das Spielen des Kartenspiels **Yaniv** ermöglicht. Die gesamten Spielregeln aus Kapitel 4 konnten verwirklicht und als Funktionalität in der Applikation umgesetzt werden. Das Spiel verfügt des Weiteren über eine einfache künstliche Intelligenz, welches das Spielen gegen einen Computergegner realisiert. Obwohl der Computerspieler nicht alle regelkonformen Möglichkeiten ausschöpft, handelt es sich um einen nicht zu unterschätzenden Gegner. Durch diese Tatsache und den Aspekt, dass die Applikation auf mobilen Endgeräten spielbar ist, ist es möglich die interaktive **Yaniv** Applikation unabhängig von einem Ort, einem Mitspieler und der Zeit zu spielen.

Das softwaretechnische Konzept lehnt sich an das *Model View Controller* Design Patern an und schafft eine akzeptable Trennung der Komponenten. Dadurch ist eine leichte Übertragung des Konzepts auf weitere Arbeiten möglich. Mit Hilfe des entworfenen Konzepts und der damit verbundenen Lösungsvariante zur Umsetzung der dieser Arbeit zugrunde liegenden Applikation, lassen sich durch wenige Veränderungen andere klassische Kartenspiele realisieren. Lediglich die jeweilige Spiellogik und der Computergegner müssten den speziellen Spielregeln angepasst werden.
Die in dieser Arbeit aufgezeigten Hintergrundroutinen erleichtern die Umsetzung weiterer Spiele. Die Erstellung der Karten, sowie die Methoden zum Austeilen der Karten und die Ermittlung der Spielerreihenfolge können problemlos übernommen werden. Andere Methoden müssten der jeweiligen Spiellogik angepasst werden. Es wäre eine neue Überprüfung eines validen Zuges notwendig und auch die Endbedingungen einer Runde oder eines gesamten Spieles müssten neu festgelegt werden. Oft existiert in einem klassischen Kartenspiel das Aufnehmen einer verdeckten Spielkarte als Bestandteil eines Zuges, welches bereits als Methode vorgefertigt bereitsteht.

Abschließend lässt sich sagen, dass die gesetzten Ziele erfüllt wurden. Ein spielbarer Prototyp wurde entwickelt und die wichtigsten Hintergrundroutinen eines klassischen Kartenspieles wurden herausgearbeitet und durchleuchtet. Zusätzlich könnte diese Arbeit Entwicklern als Leitfaden dienen, welche ein eigenes Kartenspiel implementieren möchten. Persönlich konnte bei der Umsetzung dieser Ausarbeitung, ohne Engine oder weitere Hilfs-

mittel, ein guter Einblick in die Spielentwicklung unter Android erfolgen und eigene Kenntnisse ausgeweitet und vertieft werden.

Die App-Entwicklung ist ein sich ständig weiterentwickelnder Prozess, der von stetigen Erweiterungen, Verbesserungen und Updates lebt. In der entwickelten Applikation lag der Fokus nicht auf der Grafikkomponente, weswegen diese durchaus ausbaufähig ist. In 2D lassen sich beispielsweise durch eingebaute Licht- und Schatteneffekte Tiefe erzeugen. Dadurch könnte die Präsentation des Spielfeldes aufgewertet werden. Des Weiteren ist es möglich die **Yaniv** Applikation mit mehr Computergegnern zu bestücken. Dadurch könnte die Schwierigkeit und das Spielvergnügen gesteigert werden. Denkbar wäre es, verschiedene Schwierigkeitsstufen der Computergegner einzubauen. Dafür müsste die künstliche Intelligenz überarbeitet werden. Bisher beschränkt diese sich auf wenige Möglichkeiten. Daher bieten sich an dieser Stelle viele Erweiterungsmöglichkeiten an.
Die Option, dass das Spiel mit einem Online-Modus erweitert wird, wäre zudem interessant. Dadurch könnte ein Online-Mehrspieler-Modus erzeugt werden und der Spieler erhält die Möglichkeit, sich mit realen Personen zu messen. Über einen zusätzlich bereitgestellten Spieler-Chat, könnte die gesellschaftliche Komponente zumindest im Ansatz bedient werden. Eine Erstellung eines vorgefertigten Frameworks aus den erarbeiteten Ergebnissen wäre nicht auszuschließen. Dadurch könnten weitere Entwicklungen vereinfacht werden.

Literatur

[ADC13] J. Annuzzi, L. Darcey, and S. Conder. *Introduction to Android Application Development: Android Essentials*. Developer's Library. Pearson Education, 2013.

[BF12] M. Burton and G. Franken. *Android Application Development For Dummies*. –For dummies. Wiley, 2012.

[FFSB04] E. Freeman, E. Freeman, K. Sierra, and B. Bates. *Head First Design Patterns*. Head first series. O'Reilly Media, Incorporated, 2004.

[Gar11] M. Gargenta. *Learning Android*. O'Reilly Media, 2011.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

[Jam12] D. James. *Android Game Programming For Dummies*. –For dummies. Wiley, 2012.

[MDMN13] Z. Mednieks, L. Dornin, G.B. Meike, and M. Nakamura. *Android-Programmierung*. O'Reilly, 2013.

[Que15a] Android hp @ONLINE, May 2015.

[Que15b] Applestore @ONLINE, May 2015.

[Que15c] Googleplaystore @ONLINE, May 2015.

[Que15d] wikibooks @ONLINE, May 2015.

[Que15e] Zdnet statistik @ONLINE, May 2015.

[Sim12] J. Simon. *Head First Android Development*. Head First Series. O'Reilly Media, Incorporated, 2012.