VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVÁNÍ KARET V SYSTÉMU ANDROID

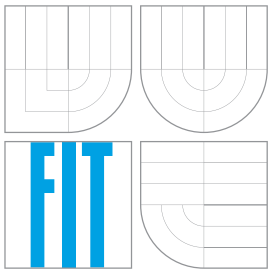
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN KLUSOŇ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVÁNÍ KARET V SYSTÉMU ANDROID

CARDS RECOGNITION USING ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN KLUSOŇ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALENA PAVELKOVÁ

BRNO 2015

Abstrakt

Práce se zabývá návrhem aplikace pro mobilní operační systém Android. Aplikace rozpoznává v obraze z kamery zařízení hrací karty a jejich hodnoty. Ulehčuje tím sčítání bodů ve hře Žolíky. Rozpoznávání je řešeno kombinací tří kaskádových klasifikátorů. Řešení založená na detekci pomocí klíčových bodů a OCR se ukázala jako nevhodná. Výsledná aplikace detekuje hodnoty karet (kromě žolíku), nikoliv jejich barvy.

Abstract

This work deals with design of an application for the mobile operating system Android. Application recognize playing cards and their values from image from camera device. Application helps to count points in game Rummy. Recognition is solved by a combination of three cascaded classifiers. Solutions based on keypoint detection and OCR proved useless. The application detects values of cards (excluding jokers), not their colors.

Klíčová slova

Android, Java, OpenCV, Tess-two, Klasifikátory, Karetní hra, Žolíky, zpracování obrazu.

Keywords

Android, Java, OpenCV, Tess-two, Classifier, Card game, Rummy, image processing.

Citace

Martin Klusoň: Rozpoznávání karet v systému Android, bakalářská práce, Brno, FIT VUT v Brně, 2015

Rozpoznávání karet v systému Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením paní Ing. Aleny Pavelkové a uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Klusoň
21. května 2015

Poděkování

Na tomto místě bych rád poděkoval všem, kteří se podíleli na vzniku této práce. V první řadě kamarádu Bc. Janu Nádvorníkovi za inspiraci k námětu práce. Dále pak paní Ing. Aleně Pavelkové za vedení a odbornou pomoc. Nakonec všem dalším pracovníkům fakulty, kteří mi přispěli svými radami.

Obsah

Kapitola 1

Úvod

Mobilní aplikace postupně pronikají do všech oblastí lidského života. Každý majitel chytrého telefonu nebo tabletu zná aplikace pro komunikaci, sledování sociálních sítí, zobrazování webových stránek, psaní poznámek, přehrávání videa a samozřejmě i hraní her. Ale v katalozích dostupných aplikací jsou stovky dalších, které se snaží přijít s novým nápadem nebo inovativním řešením. Jejich přínos nemusí být všem úplně zřejmý a ocení jej jen hrstka uživatelů. Do této kategorie se pravděpodobně zařadí i aplikace, jež vznikne v rámci této práce. Jejím cílem je vytvořit pomůcku pro karetní hru Žolíky.

Aplikace se nesnaží měnit léty prověřená herní pravidla, radit hráčům s jejich strategií nebo jinak zasahovat do průběhu hry. Místo toho udržuje informaci o stavu bodů a průběžném pořadí hráčů. Aby nebyla jen náhradou tužky a papíru, nebo jakousi specializovanou kalkulačkou zabalenou do líbivého designu, přináší další, z pohledu této práce zásadní, funkcionlitu. Počet bodů, jenž hráči v daném kole získali, se spočítá automaticky. Postačí, aby každý hráč vyfotil svoje zbylé karty. Aplikace pak analýzou obrazu najde jednotlivé karty, určí jejich hodnoty a přičte odpovídající počet bodů.

Stěžejní část práce se zabývá počítačovým viděním. Perspektivním oborem, jehož možnosti uplatnění se s rozmachem chytrých mobilních zařízení znásobily. Chytré mobilní telefony disponují dostatečným výkonem, standardně i fotoaparátem a díky klesající ceně postupně nahrazují obyčejné mobilní telefony [4].

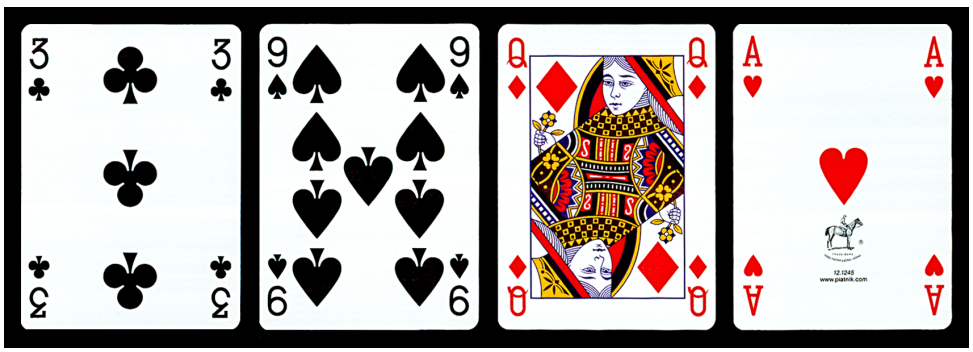
Kapitola 2

Rozpoznávání hracích karet v operačním systému Android

Pro rozpoznávání hracích karet v obrazu z kamery zařízení, za účelem zlepšení karetní hry, je zapotřebí skloubit poznatky z více oblastí. Je třeba orientovat se v operačím systému Android, znát různé algoritmy pro detekci objektů, ale také umět pravidla karetní hry Žolíky.

2.1 Pravidla karetní hry Žolíky

Žolíky jsou v Česku rozšířená karetní hra. Nejčastěji se hraje v rodinném kruhu nebo mezi přáteli. Její pravidla se většinou dědí z generace na generaci. Proto není divu, že se jejich výklad v některých detailech liší rodinu od rodiny. Na internetu můžeme nalézt pravidla na stránkách Jocker clubu ČR[1], výrobce karet Abone[2], případně v angličtině na Rummy.com[3]. I tady se jednotlivá pravidla navzájem liší. Tato práce se řídí pravidly uvedenými v knize Oficiální pravidla karetních her [5].



Obrázek 2.1: Příklad francouzských karet

Hra je určena pro minimálně dva a maximálně šest hráčů. Používají se dva balíčky francouzských karet. Ve hře je tedy každá karta právě dvakrát. Karty jsou rozděleny do

[1]Konkrétní znění pravidel: http://www.clubjoker.cz/pravidlajoker.htm
[2]Konkrétní znění pravidel: http://www.reklamnikarty.cz/pravidla_zoliky.htm
[3]Konkrétní znění pravidel: http://rummy.com/rummyrules.html

čtyř barev: piky, kříže, srdce a káry. V každé barvě se vyskytují karty v hodnotách: 2 až 10, Kluk (J), Dáma (Q), Král (K) a Eso (A). V balíčku jsou ještě dva žolíky. Na obrázku 2.1 je zleva křížová trojka, listová devítka, kárová dáma a červené eso.

Hra se sestává z několika sehrávek, které se opakují. Sehrávka začíná rozdáním karet a končí, když některý z hráčů odhodí poslední kartu z ruky rubem vzhůru na odkládací balíček, nebo-li zavře. Hráč, který sehrávku zavřel, je vítěz a nepřipisuje si žádné trestné body. Ostatní hráči si připíší tolik trestných bodů, kolik odpovídá součtu hodnot karet, které jim zbyly v ruce. V jednodušší variantě pouze jejich počet. Hodnoty jednotlivých karet jsou v tabulce 2.1.

Karta	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**J**	**Q**	**K**	**A**	**Žolík**
Hodnota	2	3	4	5	6	7	8	9	10	10	10	10	11	15
Alternativní hodnota	–	–	–	–	–	–	–	–	–	–	–	–	1	20

Tabulka 2.1: Hodnoty jednotlivých karet

Hra končí po poslední sehrávce. Celkovým vítězem se stane hráč, jenž má nejmenší součet trestných bodů ze všech sehrávek.

2.2 Operační systém Android

Operační systém Android (dále jen Android) je open-source operační systém (dále jen OS) navržený výhradně pro mobilní zařízení. U jeho zrodu stála americká internetová společnost Google, která v roce 2005 koupila společnost Android. První telefon, G1 od HTC, se začal prodávat v roce 2008. Další zařízení založená na Androidu se objevily následující rok [3]. V roce 2015 používá Android osm z deseti prodaných telefonů [4].

Ačkoli se na jeho vývoji největší mírou podílí společnost Google, oficiálně je vlastněn konsorciem Open Handset Alliance. Open Handset Alliance je sdružení 84 firem[4], které podnikají v oblastech internetu, mobilních služeb a technologií.

2.2.1 Architektura operačního systému Android

Architektura OS Android je znázorněna na obrázku 2.2. Dále budou podrobněji představeny všechny vrstvy postupně od nejnižší. Teorie vychází z druhé kapitoly knihy Learning Android [3].

Linuxové jádro se skládá z ovladačů hardware a stará se o jeho abstrakci pro vyšší vrstvy. Linux je bezpečný a stabilní operační systém, jehož nejnižší úroveň je převážně implementována v jazyce C, což ulehčuje nasazení Androidu na různých zařízeních.

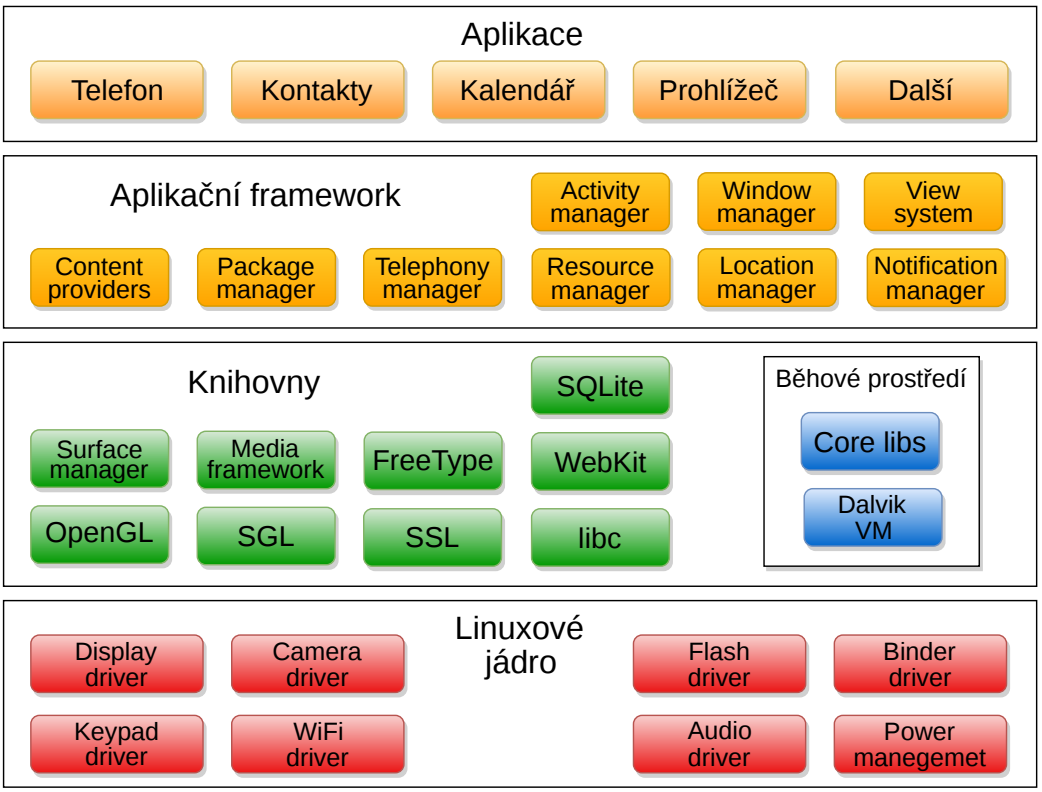
Knihovny poskytují nezbytné služby pro aplikační vrstvu. Slouží k ukládání dat, práci s grafikou a multimediálními soubory, šifrování a zabezpečení dat. Jsou napsány v jazyce C nebo C++ a nejčastěji pocházejí od open-source komunit.

Dalvik je účelový virtuální stroj speciálně navržený pro Android a vyvinutý v Googlu. Vznikl jako náhrada za Java Virtual Machine (Java VM). Je efektivnější, jelikož je určen výhradně pro mobilní zařízení. Dalvik je navíc oproti Java VM licencován pod open-source licencí Apache License 2.0.

Aplikační framework obsahuje Java knihovny speciálně vytvořené pro Android, které umožňují vývojářům pracovat s prvky OS a dalšími aplikacemi.

[4]Více na: http://www.openhandsetalliance.com/oha_members.html

Díky *aplikacím*, jež vytváří vývojáři, mohou uživatelé naplno využívat potenciál svého zařízení. Jsou buď předinstalované, nebo si je uživatel může stáhnout z dostupných obchodů s aplikacemi. Nejznámější z nich, Google Play, provozuje společnost Google. Celá aplikace je uložena v jednom souboru s příponou apk a skládá se ze tří částí. Dalvik executable obsahuje Dalvik bajt kód. V resources se nachází doplňkové soubory jako obrázky, zvuky, videa nebo XML s daty. Poslední částí jsou případně nativní knihovny implementované v jazyku C nebo C++.



Obrázek 2.2: Achitektura operačního sytému Android (podle předlohy z knihy [3])

2.2.2 Vývoj aplikací pro operační systém Android

Pro vývoj aplikací je, kromě Javy a oficiálních nástrojů popsaných níže, možné použít i několik dalších frameworků. Například nástroj PhoneGap[5] umožňuje vytvořit jednoduchou aplikaci pomocí webových technologií a následně ji exportovat na další mobilní OS. Takto vzniklé aplikace mají však nižší výkon a omezené možnosti, proto se jim v této práci nebudeme více věnovat.

Oficiální vývojové nástroje

Na konci roku 2014 se oficiálním vývojovým prostředím stal program Android Studio[6]. Android Studio, které je založené na prostředí InteliJ, tak nahradilo dosud používanou kombinaci vývojového prostředí Eclipse s rozšířením Android Development Tools.

[5] Více na: http://phonegap.com/about
[6] Více na: http://android-developers.blogspot.com/2014/12/android-studio-10.html

Android Software Development Kit (Android SDK) je součástí Android Studia. Odpadá tak potřeba jeho samostatné instalace.

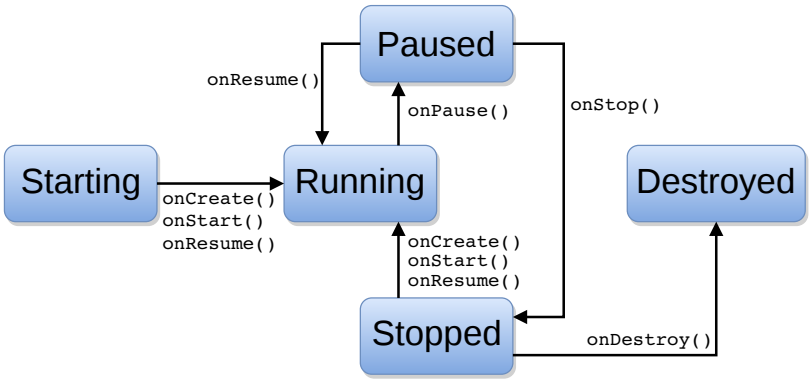
Oficiálně je podporován Java Development Kit pouze od společnosti Oracle[7], nicméně lze použít i open-source variantu OpenJDK[8].

Také je potřeba mít Android zařízení pro testování aplikace. To je buď možné virtualizovat pomocí Android Virtual Devices, nebo připojit skutečné s využitím Android Debug Bridge.

Základní stavební prvky Android aplikace

Následují podstatné komponenty, ze kterých se může aplikace skládat. Teorie vychází ze čtvrté kapitoly knihy Learning Android [3].

- *Aktivity* vytváří uživatelské rozhraní. Jedna aktivita obvykle odpovídá všem objektům zobrazeným na displeji v jeden okamžik. Po spuštění aplikace se zobrazí hlavní aktivita (zpravidla menu). Z ní je pak možné spouštět další aktivity s jiným obsahem. Stavy, ve kterých se může aktivita nacházet, jsou znázorněny na obrázku 2.3.



Obrázek 2.3: Životní cyklus activity (podle předlohy z knihy [3])

- *Služba* je komponenta, která dlouhodobě provádí svou činnost na pozadí bez ohledu na dění na displeji. Typickým příkladem je hudební přehrávač. Oproti aktivitám má služba jen stavy `Starting`, `Running` a `Destroyed`.

- *Poskytovatelé obsahu* umožňují aplikacím mezi sebou sdílet data. Standardně každá aplikace běží v uzavřeném prostředí (tzv. sandboxu) a může přistupovat jen k vlastním datům. Příkladem využití poskytovatele obsahu může být aplikace, jež zobrazuje kontakty. Žádné kontakty neobsahuje, ale zobrazuje údaje, které získala od poskytovatele kontaktů. Díky tomu je možné ji nahradit jinou aplikací se stejnou funkčností.

- *Záměry* jsou zprávy, které jsou posílány mezi hlavními komponentami systému. Záměry mohou vytvářet aplikace nebo OS. Tímto způsobem se systémem například šíří informace o nově příchozí sms zprávě nebo o aktuálním stavu baterie. Aplikace, které chtějí, na záměry pak reagují. Stějně tak aplikace může pomocí záměru požádat o zobrazení webové stránky. Stránku pak zobrazí prohlížeč, který je nastaven jako výchozí.

[7] Více na: http://www.oracle.com/technetwork/java/index.html
[8] Více na: http://openjdk.java.net

Kombinace různých počtů výše zmíněných komponent dohromady tvoří aplikaci. Tyto komponenty mají stejný kontext. Kontext aplikace odkazuje na její prostředí a na proces, ve kterém všechny její komponenty běží. Kontext aplikace se vytvoří při spuštění první komponenty aplikace a trvá po celou dobu běhu. Aktivity a služby jsou podtřídy kontextu.

2.3 Algoritmy pro detekci objektů v obraze

V této části budou představeny algoritmy pro detekci objektů v obraze.

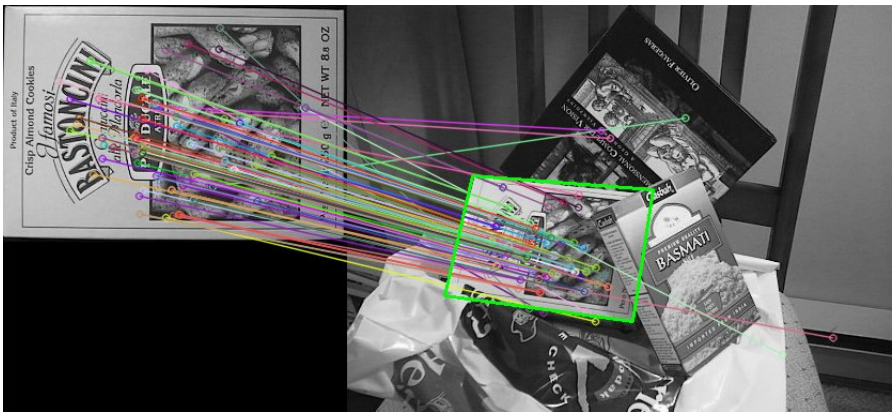
2.3.1 Detekce pomocí klíčových bodů

Následující algoritmy se zabývají hledáním význačných (nebo-li klíčových) bodů v obraze a jejich popisem pomocí *deskriptorů*. Tyto body pak mohou být použity například pro sledování pohybu objektů ve videu, spojování více fotografií do jedné panoramatické a také pro detekci objektů v obraze.

- *Scale-invariant feature transform – SIFT*
- *Speeded-Up Robust Features – SURF*
- *Features from accelerated segment test – FAST*
- *Oriented FAST and Rotated BRIEF – ORB*

Klíčový bod je definován jako zajímavé místo obrazu. Metody, kterými jsou body v obraze nalezeny, se liší dle algoritmu. Nejjednodušššími příklady jsou hrany a rohy.

Obrázek 2.4 ukazuje úspěšnou detekci objektu pomocí algoritmu SURF. Obrázek je převzat ze stránek projektu OpenCV[9].



Obrázek 2.4: Příklad detekce objektu pomocí klíčových bodů

[9]Konkrétně: http://docs.opencv.org/doc/tutorials/features2d/feature_homography/feature_homography.html

2.3.2 Viola-Jones detektor

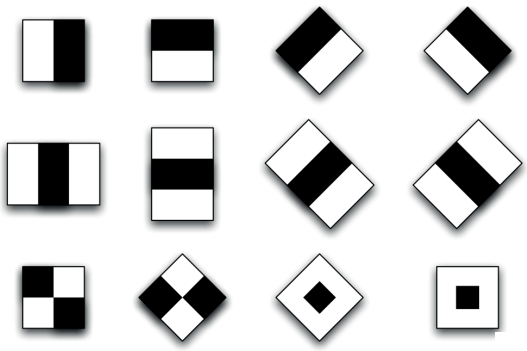
Tento detektor objektů v obraze představili v roce 2001 Paul Viola a Michael Jones. Publikovali ho na příkladu detekce obličeje, ale lze ho používat i na jiné objekty. Vyznačuje se přesností a rychlostí detekce.

Pro natrénování používá algoritmus AdaBoost, který vybírá nejvhodnější příznaky. Jako příznaky slouží Haarovy příznaky. Pro jejich výpočet se využívá integrální obraz. Pro rychlý průchod snímkem slouží kaskáda klasifikátorů. Tato část používá informace z přednášek předmětu IKR[10] a [6].

Haarovy příznaky

Haarovy příznaky jsou jednoduché konvoluční filtry. Můžeme je vidět na obrázku 2.5. Dělí se na hranové, čárové a středové. Jejich aplikací na obraz získáme odezvy.

Příznaky se postupně aplikují na celý obraz. Velikosti příznaků se postupně zvětšují až na velikost vstupního obrazu. Odezva příznaku se vypočítá jako součet hodnot pixelu v bílé části mínus součet pixelu v černé části.



Obrázek 2.5: Haarovy příznaky

Integrální obraz

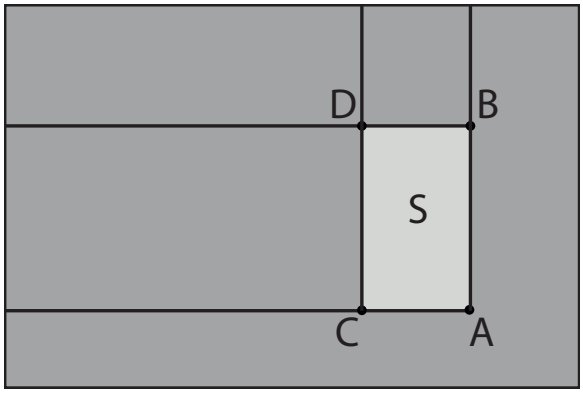
Během výpočtů odezev Haarových příznaků je často potřeba vypočítat sumu hodnot pixelů v daném obdélníku. Pro urychlení této operace slouží integrální obraz.

Integrální obraz je jiná reprezentace obrazu. Každý jeho pixel (se souřadnicemi x a y) má hodnotu rovnou součtu všech pixelů ležících od něj nalevo a nahoru v normálním obraze. Tato definice se dá také zapsat vzorcem (2.1).

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.1}$$

Hlavní výhodou je, že součet všech hodnot v libovolném obdélníku obrazu je možné vypočítat třemi aritmetickými operacemi a čtyřmi přístupy do paměti. Není potřeba iterovat přes všechny body výřezu a doba výpočtu je konstantí pro jakoukoliv velikost výřezu. Viz obrázek 2.6 a na něj navazující vzorec (2.2) pro výpočet plochy obdélníku S.

[10] Stránky předmětu: http://www.fit.vutbr.cz/study/courses/IKR/public



Obrázek 2.6: Integrální obraz

$$S = A + D - C - B \tag{2.2}$$

AdaBoost

Během tréninku se pro podokno o velikosti 24×24 pixelů vypočítá 160 000 Haarových příznaků. To je násobně více, než je počet pixelů, a i když je výpočet příznaků rychlý, tak je výpočet všech příznaků náročný. Bylo dokázáno, že efektivní klasifikátor může být vytvořen i s malým počtem Haarových příznaků.

Pro nalezení těchto příznaků se používá algoritmus AdaBoost. AdaBoost (nebo-li Adaptive Boosting) byl představen v roce 1995 Y. Freundem a R. Schapirem. Jeho princip je vytvářet silný klasifikátor z lineární kombinace několika slabých klasifikátorů. Silný klasifikátor musí být úspěšnější než všechny slabé.

K natrénování potřebujeme dvě sady obrázků. První sada obsahuje objekt pro detekci a nazývá se pozitivní, druhá neobsahuje hledané objekty, a proto je negativní.

Kaskáda klasikátorů

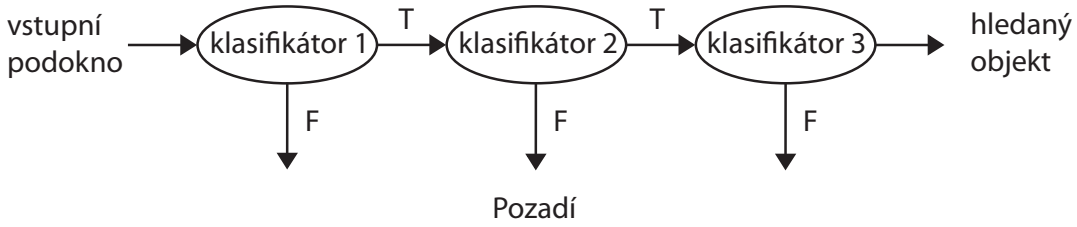
Kaskádový klasifikátor se snaží zkrátit čas potřebný k vyhodnocení každého podokna obrazu. Využívá faktu, že podoken obrazu, která neobsahují hledaný objekt, je víc než těch, které ho obsahují.

Skládá se z více jednoduchých binárních klasifikátorů poskládaných za sebe. V každém z nich se určuje, zda-li podobraz obsahuje hledaný objekt či nikoliv. Zamítnutá podokna se už dále nezpracovávají, zbylá pokračují do dalších stupňů kaskády. Dojde-li podokno až na konec celé kaskády, tak obsahuje hledaný objekt. Schéma kaskádového klasifikátoru je na obrázku 2.7.

2.3.3 Optické rozpoznávání znaků

Optické rozpoznávání znaků se zabývá získáváním textů z obrazu. Vžila se pro něj zkratka OCR (z anglického Optical Character Recognition). Poprvé bylo použito, na začátku devadesátých let minulého století knihovnou ve Velké Británii, pro digitalizaci starých novinových výtisku[11]. Další teorie této části vychází z knihy Optical Character Recognition [2].

[11] http://www.dlib.org/dlib/march09/holley/03holley.html



Obrázek 2.7: Schéma kaskádového klasifikátoru

Základní dělení oblastí rozpoznávání znaků

Základní dělení OCR je na *online* a *offline*. *Online* probíhá v momentě, kdy je znak psán. Jsou tak k dispozici informace o pohybu pera. Naopak *offline* rozpoznávání zpracovává znaky až po jejich vytištění či napsání.

Offline rozpoznávání se specializuje buď na texty napsané *nevázaným písmem*, nebo na texty psané *vázaným písmem*. Liší se tím, zda jsou jednotlivá písmena od sebe oddělena, nebo spojena dohromady.

Rozpoznávání nevázaných znaků

OCR metody jsou založeny na automatickém rozeznávání prvků do tříd. Třídy jsou písmena, číslice a další speciální znaky. Každá třída si na základě různých obdržených vzorů vytvoří prototyp nebo popis svého znaku. Během rozpoznávání je neznámý znak postupně porovnán se všemi třídami. Nejlepší shoda se vzorem se považuje za výsledek rozpoznávání.

Postup rozpoznávání

Průběh rozpoznávání znaků lze rozdělit do následujících částí.

Skenování je pořízení obrazu, ve kterém se nachází text. Zahrnuje taky základní úpravy jako je převedení obrazu do stupňů šedi a prahování.

Umístění a segmetace zahrnuje nalezení pozice, kde se text nachází. Dále se text rozdělí na jednotlivá slova a slova následně na jednotlivé znaky.

Předzpracování se skládá z normalizace a vyhlazení znaků. Normalizace upraví sklon, velikost a natočení znaku. Vyhlazení zaplní případné mezery ve znaku a zúží šířku čar znaku. Příklad předzpracování je na obrázku 2.8.

Obrázek 2.8: Normalizace a vyhlazení znaku

Extrakce příznaků zjišťuje charakteristické vlatnosti znaku tak, aby pro něj byly jedinečné. Používá se několik typů příznaků:

- rozložení bodů každého znaku,
- transformační metody – používají popisu křivek, které opisují okraje znaku,
- strukturální analýza – extrahuje ze znaku jeho charakteristické geometrické rysy.

Klasifikace porovnává každý znak s třídami podle extrahovaných příznaků. Klasifikace také může využívat více metod:

- porovnání – identifikuje výsledný znak podle třídy, se kterou má rozpoznávaný znak nejlepší shodu,
- optimální statistické klasifikátory – vypočítají pro znak pravděpodobnosti, s jakými spadá do všech tříd, následně je znak zařazen do třídy s nejvyšší pravděpodobností,
- neuronové sítě.

Dodatečné zpracování spojuje jednotlivé rozpoznané znaky do slov. Slova pak prochází detekcí chyb podle slovníku nějakého jazyka, aby měla smysl. Rozpoznávání samostatných znaků je z tohoto důvodu obtížné.

2.4 Knihovny pro zpracování obrazu

Výše popsané algoritmy jsou spolu s dalšími implementovány v knihovnách pro práci s obrazem. Dále budou představeny některé z nich.

2.4.1 Knihovna OpenCV

Open Source Computer Vision Library je knihovna, která implementuje několik set algoritmů počítačového vidění. Vývoj projektu začala společnost Intel v roce 1999. Zdrojové kódy knihovny jsou uvolněny pod licencí BSD. Mezi podporovanými operačními systémy je i Android [1].

Použití OpenCV v operačním systému Android

Rozhraní OpenCV pro Android se nazývá *OpenCV4Android* a nabízí dva způsoby použití[12].

První variantou je použít nativní C++ rozhraní s pomocí Android NDK. (Android NDK, nebo-li Native Development Kit, je sada nástrojů, které umožňují, aby část aplikace byla napsána v jazyce C++.) Tímto způsobem je možné vytvořit přenositelný kód, případně použít již vyvinutý program, a zasadit ho do Android aplikace.

Jednodušší možností vývoje je použití Java rozhraní knihovny. To nabízí téměř všechny její možnosti. Daní za jednoduchost je pomalejší běh aplikace než v první variantě.

OpenCV4Android nabízí od verze 2.4.3 aplikaci *OpenCV Manager*. Ta spravuje potřebné binární knihovny. Zajišťuje jejich aktualizace a případně optimalizace pro konkrétní hardware. Všechny aplikace sdílejí jen jednu kopii knihoven. V případě, že *OpenCV Manager* není nainstalován a nějaká aplikace potřebuje k běhu OpenCV knihovny, je uživatel k jeho

[12] Více na: http://opencv.org/platforms/android/opencv4android-usage-models.html

instalaci vyzván před jejím spuštěním. Instalace probíhá buď automaticky pomocí Google Play, nebo ručně z balíčku dostupného na stránkách projektu.

2.4.2 Prostředky pro optické rozpoznávání znaků

Implementací OCR algoritmů se zabývá celá řada knihoven[13]. Použití v mobilních zařízeních s Andoridem umožňují následující:

- *ABBYY Mobile OCR Engine*[14] je komerční OCR nástoroj. Sdružuje více funkcí, například slovníky pro překlad rozpoznaného textu do dalších jazyků.
- *Tesseract*[15] je považována za nejpřesnější open-source OCR nástroj. Vyvinula ji společnost Hewlett Packard v letech 1984–1994. V roce 2005 ji uvolnila pod open-source licencí a její vývoj převzala společnost Google. Vývoj však ustal (poslední aktualizece na stránce projektu je z dubna 2013).
- *Tess-two*[16] je odnož knihovny *Tesseract*, která pokračuje ve vývoji jejího rozhraní pro Android (Tesseract Tools for Android).

[13] http://en.wikipedia.org/wiki/Comparison_of_optical_character_recognition_software
[14] Více na: http://www.abbyy.com/mobile-ocr
[15] Více na: http://code.google.com/p/tesseract-ocr
[16] Více na: http://github.com/rmtheis/tess-two

Kapitola 3

Návrh aplikace

Celou problematiku aplikace je možné abstrahovat na dva téměř nezávislé podúkoly. Prvním je vyhledání hracích karet v obraze z kamery, a také správné rozpoznání jejich hodnot. Dále je třeba navrhnout uživatelské rozhraní, které zobrazí výsledky a umožní správu her. Tato kapitola se zaměří především na první podproblém.

3.1 Rozpoznávání hodnot karet

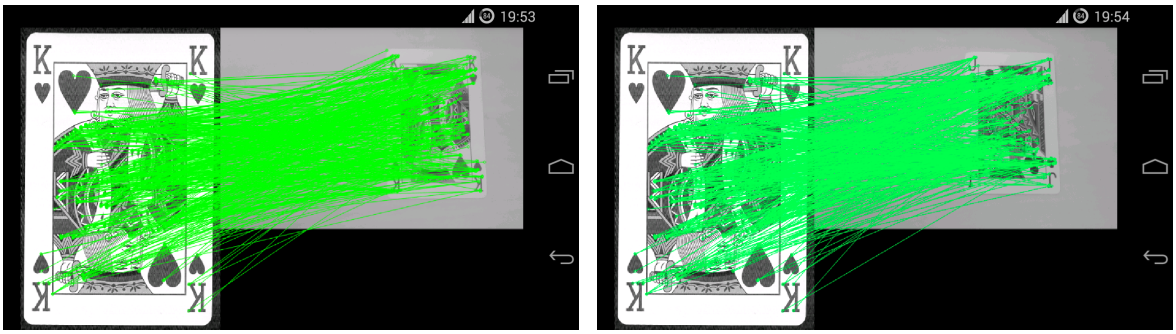
Obecně je možné pro detekci objektů v obraze použít více způsobů. Ovšem ne všechny jsou vhodné pro řešení zadaného problému.

3.1.1 Detekce pomocí klíčových bodů

Problematice detekce založené na klíčových bodech se blíže věnuje část 2.3.1. V prostředí operačního systému Andorid je z knihovny OpenCV dostupná funkce ORB. Její použití je vidět na obrázku 3.1. V levé části obrazovky aplikace je zobrazena předloha (vytvořená naskenováním karty), napravo od ní pak náhled z kamery. V obou částech byly detekovány klíčové body. Ty, které by si měly odpovídat, jsou spojeny zelenými čárami.

Z obrázku je patrné, že algoritmus dokáže najít pozici karty v rámci vstupního obrazu, ale neumí od sebe jednotlivé karty odlišit. Na obrázku 3.1a je znázorněna detekce srdcového krále, ovšem podobná shoda nastane i s dalšími kartami, například s křížovým J na obrázku 3.1b.

Tento způsob detekce tedy není vhodný a bude nahrazen efektivnějšími algoritmy.



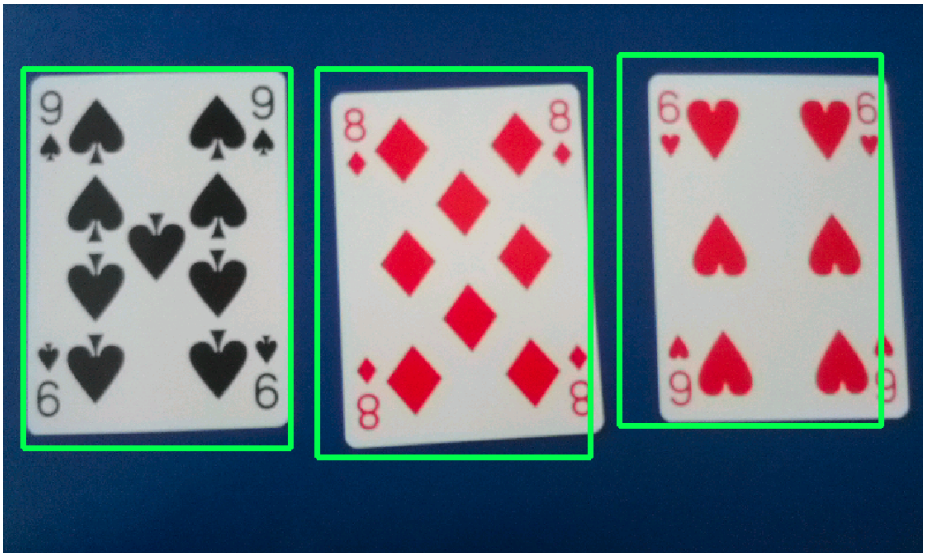
(a) Srdcové K (b) Křížové J

Obrázek 3.1: Detekce karet algoritmem ORB

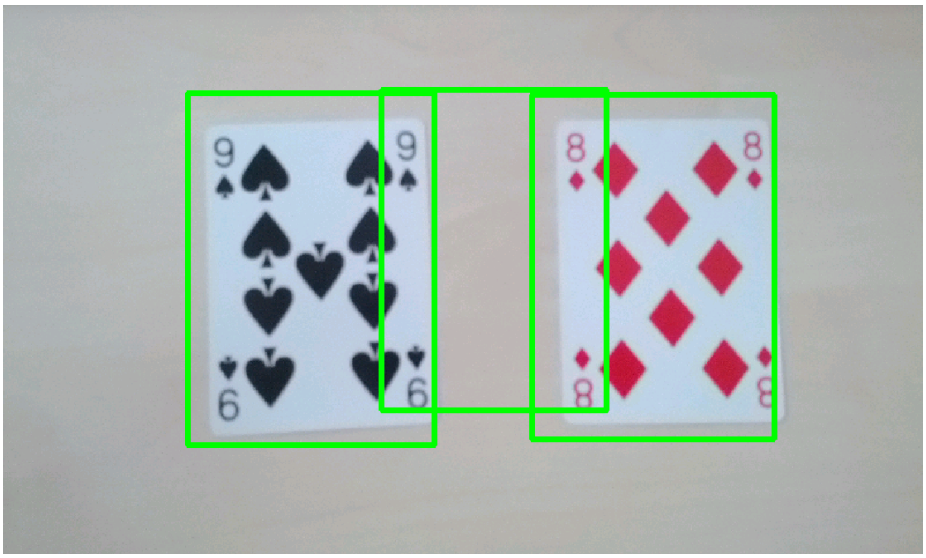
3.1.2 Kaskádový klasifikátor

Kaskádový klasifikátor, který je podrobněji popsaný v části 2.3.2, umožňuje najít v obraze objekty karet řádově rychleji než předchozí způsob. Výsledky detekce jsou vidět na obrázku 3.2. Ve zhruba 15 % rozpoznávání dojde k nesprávné detekci (například mezi okraji sousedních karet). Míra nepřesnosti je přímo úměrná počtu karet, které se v obraze nacházejí. Přesnost ovlivňují i další faktory jako barva pozadí, světelné podmínky, odlesky nebo vlastní vzhled karty.

Dále je potřeba určit bodové hodnocení jednotlivých karet. To se odvíjí pouze od hodnoty karty, nikoliv od její barvy. Proto by určování barev karet bylo plýtvání časem i systémovými prostředky.



(a) Bezchybná detekce



(b) Chybná detekce

Obrázek 3.2: Detekce karet kaskádovým klasifikátorem

Určovaní hodnoty karty podle znaku

První možnost, která se nabízí, je použití znaku v levém horním rohu karty. Protože znak, který určuje hodnotu karty, je buď číslice (dvě v případě desítky), nebo jedno z písmen J, Q, K či A, zdá se jako nejvhodnější použití OCR algoritmů.

Z knihoven pro rozpoznávání textu v obraze dostupných pro Android (2.4.2) jsem vyzkoušel *Tess-two*, protože je možné ji volně používat a je neustále vyvíjená. Knihovna funguje dobře pro celé věty, nebo alespoň jednotlivá slova. Správné určení samostatně stojícího znaku jí dělá problémy. V takovém případě je úspěšnost mizivá. Například rozpoznání slova *JOKER* proběhne bez problémů, ale samostatně stojící *J* se detekuje chybně.

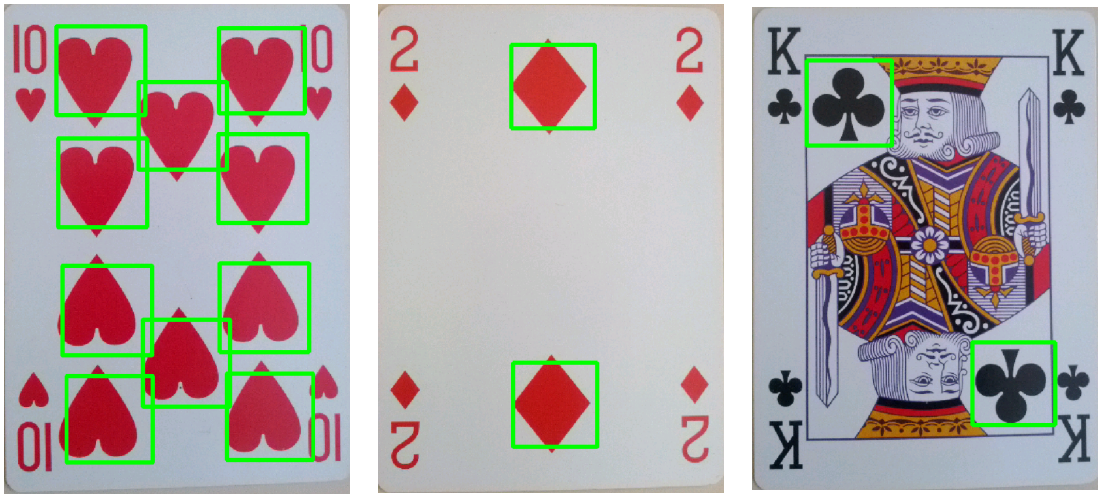
Funkci je možné nastavit několik parametrů, které by měly ovlivnit její fungování. Prvním je nastavení režimu stránky na hodnotu `PSM_SINGLE_CHAR`. Dále je možné parametrem `whitelist` explicitně určit množinu znaků pro rozpoznání. Tyto parametry však nevedou ke zjevnému zlepšení výsledků. Nepomůže ani úprava vstupního obrazu (například prahováním).

Použití OCR knihovny *Tess-two* pro rozpoznávání znaků nevede k řešení problému, je tedy třeba hledat další řešení.

Určení hodnoty podle počtu symbolů

Podíváme-li se blíže na francouzské karty, zjistíme, že počet symbolů na každé z nich koresponduje s jejím bodovým ohodnocení ve hře Žolíky. Samořejmě existuje několik výjimek, o těch bude řeč níže.

Pro hledání symbolů jsem natrénoval další kaskádový klasifikátor. Jako vstupní obraz mu je předán pouze výřez s kartou určený předešlým klasifikátorem. Jeho výsledky jsou uspokojivé a můžeme je vidět na obrázku 3.3



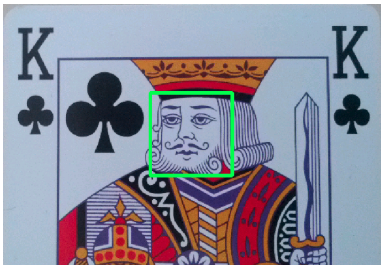
Obrázek 3.3: Symboly rozpoznané kaskádovým klasifikátorem

Aby nedocházelo k detekci i čtveřice symbolů v rozích karty, je minimální velikost hledaného symbolu nastavena na pětinu šířky karty. Pro karty s hodnotou 2 až 10 odpovídá počet rozpoznaných symbolů bodovému ohodnocení karty.

Karty J, Q a K obsahují pouze dva symboly, i když jejich bodová hodnota je 10 bodů. Je potřeba je odlišit od karty s hodnotou 2. Jako první možnost rozlišení se nabízí využít vzájemnou polohu obou symbolů v rámci karty. Zatímco na kartě s hodnotou dva jsou

oba symboly umístěny osově pod sebou, na kartách J, Q, K se nacházejí v protilehlých rozích (obě karty jsou na obrázku 3.3). Porovnáním vodorovných souřadnic obou detekovaných symbolů dojde k rozeznání pouze ve zhruba šesti pokusech z deseti. To je způsobeno zvýšeným výskytem chybně detekovaných symbolů u karet J,Q a K.

Druhým společným rysem karet J, Q, K, který je odlišuje od karty s hodnotou 2, je vyobrazená postava. Nabízí se tak možnost její detekce. Rozhodl jsem se využít detekci obličeje. Knihovna OpenCV standardně nabízí prostředky pro detekci obličejů, ale protože jsou obličeje postav na kartách specifické, raději jsem zvolil vlastní kaskádový klasifikátor. Tento, v pořadí třetí klasifikátor, dostává jako vstup horní polovinu detekované karty. Pokud dojde k detekci obličeje, je bodová hodnota karty změněna na 10 bodů.



Obrázek 3.4: Detekce obličeje na kartě K

Dalším možným řešením rozpoznání J, Q, K od karty dva, je použití čtveřice kaskádových klasifikátorů, které by byly natrénovány zvlášť pro každou barvu (srdce, piky, kříže a káry). V takovém případě by se mohl počet chybně detekovaných symbolů zredukovat na úroveň, aby bylo možné použít první zmíněný způsob rozeznání. Toto řešení jsem neimplemntoval, protože by zbytečně navýšilo počet kaskádových klasifikátorů, a tak prodloužilo čas detekce.

Poslední karty, jež je třeba rozpoznat, jsou esa a žolíky. Na první pohled je řešení jednoduché. Esa obsahují jeden symbol a žolík žádný. Situaci komplikuje pikové eso (obrázek 3.5). To stejně jako žolík neobsahuje žádný standardní symbol. Je tedy potřeba je od sebe odlišit. Jako první jsem zkusil detektoru symbolů nastavit nižší minimální velikost symbolů tak, aby detekoval symboly v rozích esa. Při této velikosti ovšem dojde k chybné detekci i na kartě žolíku. Navíc podoba karty žolíku se liší balíček od balíčku, tudíž na něm není možné cokoliv detekovat.

Obrázek 3.5: Piková esa

Opět se nabízí více řešení. Například prohlásit za podporované pouze balíčky karet, které mají na kartě žolíku nápis *JOKER*. Nápis lze rozeznat pomocí OCR algoritmů. Esa by pak byly karty s žádným nebo jedním symbolem. Tím by se však zásadně zmenšilo uplatnění aplikace a zvětšila se její velikost kvůli přítomnosti knihony *Tess-two* (o nezanedbatelných 10 MB).

Z výše zmíněných důvodů jsem se rozhodl pro následující řešení. Karty žolíků nejsou vůbec zahrnuty do skenování. Jejich počet uživatel zadá pomocí tlačítka v prostředí skenování karet. Během skenování je pak každá karta s maximálně jedním detekovaným symbolem považována za eso a přiřadí se jí odpovídající bodová hodnota.

Výpočet výsledného skóre

Plocha, kterou je možné fotoaparátem snímat, není neomezená. Karty musí v obraze zaujímat dostatečný prostor, aby bylo možné je správně rozpoznávat. Optimální je umístit do náhledu fotoaparátu 1 až 4 karty najednou. V případě, kdy uživateli zbyde po sehrávce karet více, musí provést jejich rozpoznání po etapách. V náhledu fotoaparátu se do obrazu promítají obrysy karet detekované prvním kaskádovým klasifikátorem. Uživatel tak má přehled o tom, které karty právě snímá.

Jako první jsem vyzkoušel variantu, kde uživatel postupně zachytí snímky všech svých karet. Ty se následně zpracují výše popsaným způsobem. Výhodou je malá výpočetní náročnost, protože snímků jsou řádově jednotky. Naopak výraznou nevýhodou je, že v případě chybné detekce jakéhokoliv kaskádového klasifikátoru dojde hned k nesprávnému vyhodnocení.

Proto jsem navrhl systém určení hodnot, který je založený na statistice. Vycházím z předpokladu, že nadpoloviční většina rozpoznání proběhne správně. V momentě, kdy uživatel zahájí snímání, je každý snímek z kamery zpracován výše popsaným způsobem. Výsledné hodnoty každého snímku se pak ukládají do tabulky. Poté, co tabulka obsahuje dostatek dat, jsou z ní vybrány nejpravděpodobnější výsledky.

Tabulka 3.1 ve zkrácené podobě ukazuje příklad výsledků pro rozpoznání karet s hodnotami Q, 8 a 3. Můžeme vidět, že na snímcích číslo 2, 8 a 10 proběhla chybná detekce karet (karet je detekováno více, respektive méně než je jejich skutečný počet). Na dalších snímcích jsou chybně detekovány pouze hodnoty některých karet. Například na snímku číslo jedna bylo Q rozpoznáno jako karta dva.

Po dokončení přidávání hodnot do tabulky se v každém řádku spočítá počet karet. Mezi těmito hodnotami se nalezne ta nejčastější (také označovaná jako modus). Na řádcích, které obsahují jiný než nejčastější počet karet, došlo pravděpodobně k chybné detekci. Ale protože nelze zpětně určit, která z hodnot chybí, nebo naopak přebývá, jsou všechny řádky s rozdílným počtem detekovaných karet z tabulky odstraněny.

Protože není zaručeno stejné pořadí detekovaných karet na všech snímcích, seřadí se hodnoty karet ve všech všech řádcích od nejmenší. Zvýší se tak pravděpodobnost, že rozpoznané hodnoty jedné karty budou ve stejném sloupci. Tabulka 3.2 znázorňuje stav po smazání nestandardních řádků a seřazení hodnot ve zbylých řádcích.

V posledním kroku se z každého sloupce vybere taktéž nejčastější hodnota a ta se považuje za nejpravděpodobnější výsledek rozpoznávání.

pořadí snímku	hodnoty				
1	2	8	3	-	-
2	2	5	8	3	2
3	10	7	4	-	-
4	10	7	3	-	-
5	8	10	3	-	-
6	10	8	3	-	-
7	3	10	8	-	-
8	10	2	-	-	-
9	3	8	2	-	-
10	3	8	3	3	-

Tabulka 3.1: Načtené hodnoty

pořadí snímku	hodnoty		
1	2	3	8
3	4	7	10
4	3	7	10
5	3	8	10
6	3	8	10
7	3	8	10
9	2	3	8

Tabulka 3.2: Promazané hodnoty

3.2 Objekty pro správu her

Aplikace, která by uměla pouze rozpoznávat a přiřazovat bodové ohodnocení karet, by postrádala smysl. Uživatel potřebuje mít přehled o hře. Proto je nutné navrhnout další logické prostředky, které umožní vyhodnocení stavu hry. Také je třeba vytvořit grafické uživatelské prostředí pro interakci s uživatelem.

Uživatel chce primárně hrát hru Žolíky. Tedy mít skupinu hráčů a postupně jim přidávat body. Pro tento účel bude v aplikaci sloužit objekt *hra*. Ta obsahuje množinu hráčů, název a další dodatečné informace, kterými jsou datum jejího vytvoření, počet hráčů, počet odehraných kol (sehrávek) a statistiky.

Hlavní položkou hry je objekt *hráče*, jehož hlavní položky jsou jméno, skóre a počet odehraných kol. V každé hře musí být alespoň dva hráči. Pokud tomu tak není, jsou automaticky doplněni. Horní hranice počtu hráču jedné hry není programově omezena, i když pravidla připouští maximálně 6 hráčů. Od omezení jsem upustil po diskuzi s potencionálními uživateli aplikace.

Aplikace může obsahovat více her. Konkrétně je jejich maximální počet z technických důvodů omezen na 100. Předpokládám, že uživatelé budou ukládat maximálně desítky her a tato hranice jim poskytuje dostatečnou rezervu. Aplikace samozřejmě musí obsahovat nástroje pro vytváření nových her, jejich mazání a výběr aktuální.

Všechny výše popsané objekty se po ukončení ukládají do paměti zařízení ve formátu XML, aby bylo možné po opětovném spuštění obnovit jejich stav.

Tento návrh bylo možné bez komplikací realizovat. Jeho výsledná podoba, včetně uživatelského rozhraní, bude představena v další kapitole.

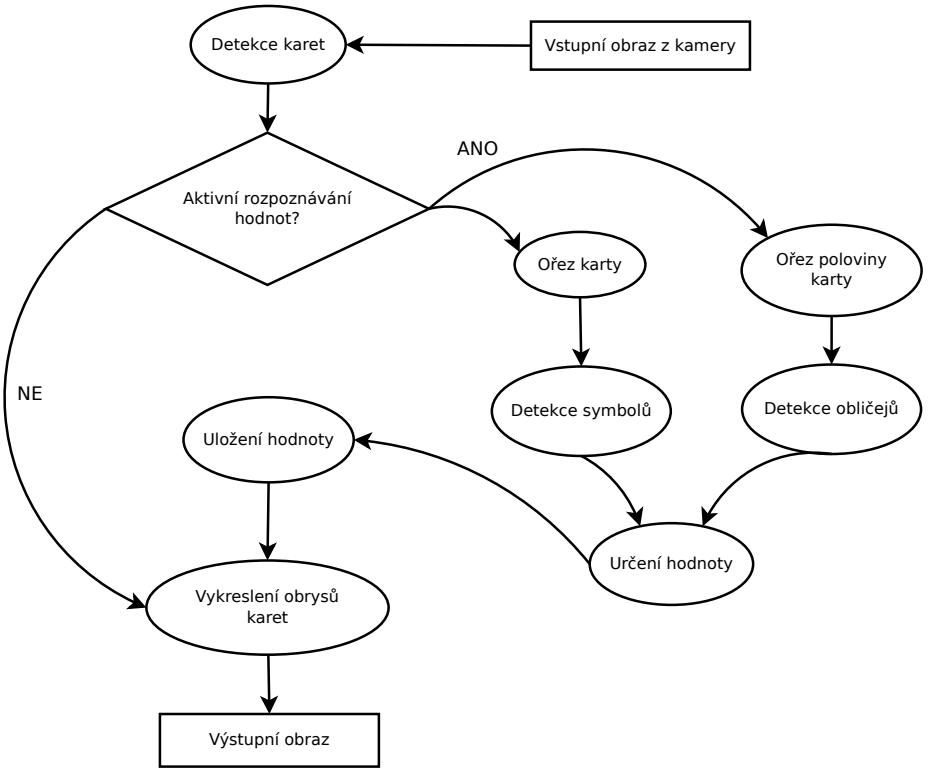
Kapitola 4

Výsledná aplikace

Následující kapitola popisuje výslednou aplikaci. Blíže bude představena implementace postupů navržených v předešlé kapitole. Také se bude věnovat trénování použitých kaskádových klasifikátorů a testování.

4.1 Zpracování snímků

Zpracování snímků, popsané v předešlé kapitole, se provádí pro každý snímek a odehrává se ve funkci `onCameraFrame()` třídy `Camera`. Vlastní zpracování smínku má pak na starost třída `Finder`. Její fungování je znázorněno v diagramu na obrázku 4.1. Bod *uložení dat* v sobě zahrnuje přidání hodnot do tabulky popsané v předcházející kapitole.



Obrázek 4.1: Diagram zpracování snímků

4.2 Uživatelské rozhraní

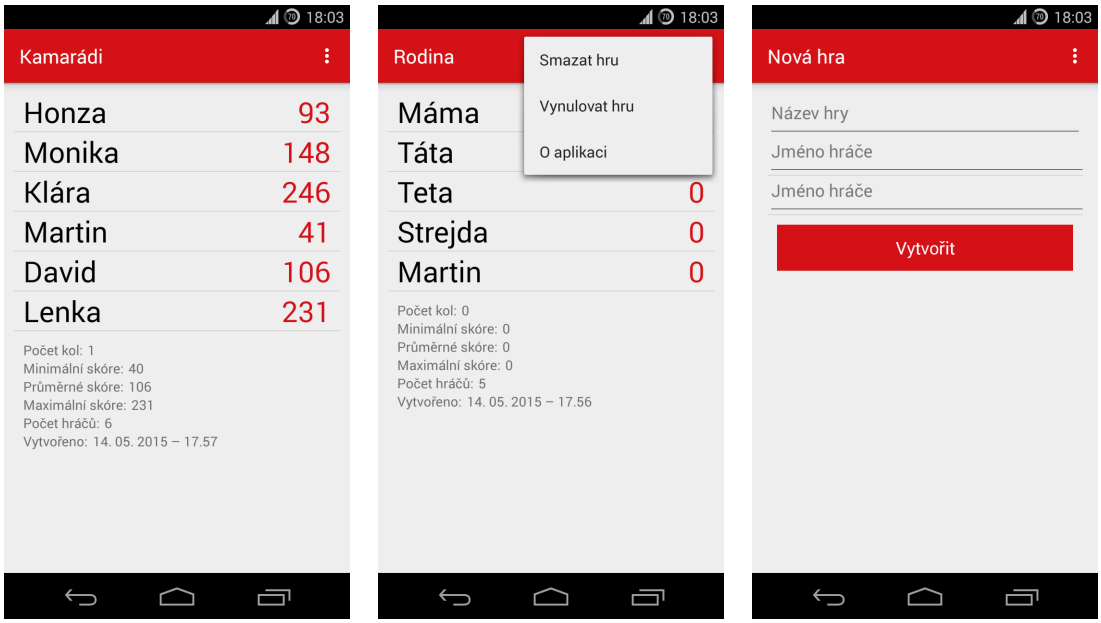
Po spuštění aplikace se zobrazí poslední vytvořená hra. Posouváním obrazovek vlevo se procházejí všechny dříve vytvořené hry. Posunutím vpravo se dostaneme na formulář pro vytvoření nové. Celé toto schéma je implementováno jako jedna aktivita (pomocí `ViewPager adapteru`) a je znázorněno na obrázku 4.2.

Hra (obrázky 4.2a a 4.2b) zobrazuje průběžné pořadí hráčů a jejich skóre. Po každém kole hry (sehrávce) si hráči přičtou body kliknutím na své jméno, čímž spustí aktivitu fotoaparátu (obrázek 4.3). Hráči jsou řazeni podle skóre od nejnižšího. V průběhu přidávání nových výsledků jsou hráči s již navýšeným skóre řazeni před ty zbylé. Aplikace také hlídá, aby si hráč přičetl body po každé sehrávce pouze jednou.

Formulář pro vytvoření nové hry (obrázek 4.2c) tvoří políčko pro zadání názvu hry a políčka pro jména hráčů. Ty jsou na začátku pouze dvě. Další se přidávají automaticky po kliknutí do posledního políčka. V případě že není uložená žádná hra, zobrazí se tento formulář jako první po spuštění aplikace.

Na všech obrazovkách je možné z horní lišty zobrazit menu (k vidění na obrázku 4.2b). To nabízí vynulování a smazání aktuálně zobrazené hry, případně zobrazení základních informací o aplikaci. Na nejpravější obrazovce s formulářem (obrázek 4.2c) jsou položky vynulovat a smazat nahrazeny možností vymazání formuláře.

Barevnou kombinaci bílé, červené a černé jsem zvolil záměrně s odkazem na barvy francouzských karet.



(a) Rozehraná hra (b) Nově vytvořená hra (c) Vytvoření nové hry

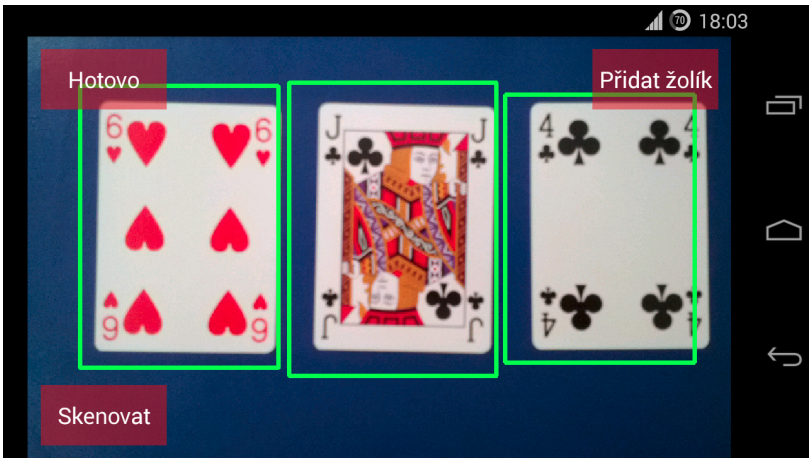
Obrázek 4.2: Uživatelské prostředí aplikace

Poslední důležitou částí uživatelského rozhraní, znázorněnou na obrázku 4.3, je obrazovka pro skenováním karet. Je implementovaná jako samostatná aktivita, která zobrazuje náhled z fotoaparátu doplněný o obrysy detekovaných karet.

Aktivita také obsahuje tři tlačítka pro její ovládání. Tlačítko *Skenovat*, v levém dolním rohu, spustí detekci a zaznamenávání hodnot karet. O jeho průběhu je uživatel informován změnou barvy obrysů detekovaných karet na červenou.

Jelikož aplikace nepodporuje skenování žolíků, je v pravém horním rohu tlačítko pro jejich přičtení. Každým stiskem se přidá jeden žolík. Informace o počtu již přidaných se zobrazí v popisku tlačítka.

Aktivita se ukončuje buď tlačítkem *Hotovo*, nebo systémovým tlačítkem *Zpět*. Obě mají stejnou funkčnost, ukončí aktivitu a vrátí celkové dosažené skóre ze všech skenování.



Obrázek 4.3: Aktivita detekce karet

4.3 Trénování kaskádových klasifikátorů

Všechny tři použité kaskádové klasifikátory byly trénovány stejným postupem. Použil jsem funkce z knihovny OpenCV. Funkce `opencv_createsamples` vytvoří pozitivní vstupní vzorky. Tyto vzorky vzniknou z obrázků, které obsahují alespoň jeden hledaný objekt. Polohu všech objektů je nutné definovat pomocí souřadnic `x` a `y`. Výstup funkce `opencv_createsamples` se předá společně se seznamem negativních vzorků funkci `opencv_traincascade`, která provádí vlastní trénování.

4.3.1 Popis datových sad pro trénování

Hlavním úkolem bylo vytvoření datové sady. Pro klasifikátor karet bylo třeba vyfotit 400 snímků skupin karet na různých površích a v nich označit polohu každé karty. Vzniklo tak přes 1800 pozitivních vzorků. Nejvíce se osvědčilo vytvářet negativní vzorky z těch pozitivních, a to začerněním oblastí, kde se nacházely hledané objekty. Tato úprava probíhala zároveň s označováním poloh objektů.

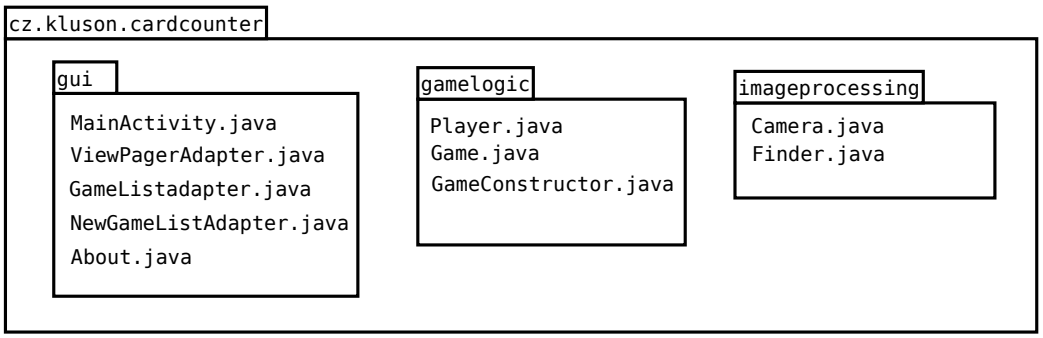
Stejný způsob byl použit i u datové sady pro klasifikátor symbolů. Ze 104 fotografií karet bylo vytvořeno téměř 900 pozitivních vzorků.

Ze stejných fotografií vychází i vzorky pro poslední kaskádový klasifikátor obličejů. V nich bylo označeno všech 24 obličejů (jako vstup slouží jen poloviny karet, jejichž zrcadlový obraz byl odstraněn). Sadu negativních vzorků posledního klasifikátoru tvoří fotografie všech zbylých karet.

4.4 Přehled implementovaných tříd

Zdrojové soubory aplikace jsem rozdělil do třech balíčků. Toto rozdělení je možné vidět na obrázku 4.4. Dále následuje popis toho, co která třída implementuje.

- `MainActivity.java` – je hlavní aktivita, která se spouští jako první. Zobrazuje uživatelské rozhraní a stará se o jeho funkce.
- `ViewPagerAdapter.java` – sestavuje jednotlivé obrazovky uživatelského rozhraní. K seznamu jmen hráčů přidává zobrazení statistik.
- `GameListAdapter.java` – vytváří položky seznamů hráčů hry.
- `NewGameListAdapter.java` – vytváří položky seznamu jmen hráčů ve formuláři pro vytvoření nové hry.
- `About.java` – do aktivity, zobrazující informace o aplikaci, doplňuje verzi a název programu.
- `Player.java` – vytváří objekty hráčů.
- `Game.java` – vytváří objekty her.
- `GameConstruktor.java` – zpracovává informace zadané do formuláře pro vytvoření nové hry. Vytváří nové hry.
- `Camera.java` – implemetuje rozhraní pro přístup ke kameře zařízení. Zároveň obsluhuje tlačítka aktivity fotoaparátu.
- `Finder.java` – provádí analýzu každého snímku. Jeho fungování je blíže představeno v části 4.1.



Obrázek 4.4: Přehled rozdělení zdrojových souborů

4.5 Testování

Testování probíhalo jak průběžně během vývoje, tak i po jeho ukončení, aby bylo možné zhodnotit dosažené výsledky. K testování byly používány dva mobilní telefony s operačním systémem Android. V tabulce 4.1 jsou uvedené specifikace jejich parametrů. V poslední kolonce tabulky je zapsáno skóre, kterého dosáhly v banchmarku Antutu[1], aby si bylo možné udělat lepší představu o jejich výkonu.

Název telefonu	Pamět RAM	Procesor	Androidu	Antutu
Samsung Galaxy Nexus	1 GB	2x 1,2 GHz, Snapdragon 400	4.4.4	13 246
YotaPhone C9660	2 GB	2x 1,7 GHz, Cortex-A9	4.2.2	27 321

Tabulka 4.1: Telefony pro testování

Na obou telefonech bylo postupně provedeno šedesát rozpoznávání. Jako vzorky byly použity skupiny karet obsahující jednu až čtyři karty. Karty byly umístěny na kontrastním podkladu a testování probíhalo za dobrých světelných podmínek. Dobrými světelnými podmínkami je míněno přirozené denní světlo bez přímého slunečního záření, jež by vytvářelo na povrchu karet odlesky. Výsledky tohoto testu jsou zobrazeny v tabulce 4.2.

	Poměr chybně rozpoznaných karet [%]	
Barva karty	**Galaxy Nexus**	**YotaPhone**
1	13	13
2	6	0
3	13	20
4	26	33

Tabulka 4.2: Poměr chybně detekovaných karet k jejich počtu

Z naměřených hodnot vyplývá, že, s výjimkou první skupiny s jednou kartou, je počet chyb úměrný počtu najednou skenovaných karet. V tabulce 4.3 jsou zapsány průměrné časy tvrvání rozpoznávání během tohoto testu. Z nich je patrné, že rozpoznávání je výpočetně náročné a prodlužuje se úměrně počtu karet v obraze. Také se potvrdil vyšší výpočetní výkon telefonu YotaPhone.

	Průměrný čas potřebný pro detekci [s]	
Počet karet	**Galaxy Nexus**	**YotaPhone**
1	9,7	6,4
2	12,6	7,6
3	14,7	8,5
4	15,4	8,8

Tabulka 4.3: Průměrné doby trvání detekce

[1] http://www.antutu.com/en/index.shtml

Během prvního testu oba telefony vyhodnotily určitou skupinu vždy shodně špatně. Druhý test se pokouší odhalit karty, které jsou nejčastěji rozpoznány chybně a způsobovaly chyby i během prvního testu. Postupně byly oběma telefony naskenovány všechny karty z balíčku. V tabulce 4.4 jsou zaznamenány počty špatně detekovaných karet pro každou barvu. Z dat vyplývá, že všechny barvy jsou rozpoznávány stejně úspěšně v průměru s jednou chybou.

	Počet chybně detekovaných karet od každé barvy	
Barva karty	**Galaxy Nexus**	**YotaPhone**
Srdce	0	1
Káry	1	1
Piky	1	1
Kříže	1	1

Tabulka 4.4: Počet chyb v závislosti na barvě karty

Jako poslední zbývá ověřit závislost chyb na hodnotě karty. U karet s hodnotami 2 až 10 nedošlo k žádnému chybnému rozpoznání, proto jsou v tabulce 4.5 vynechány. U zbylých karet došlo k chybnému rozpoznámí alespoň na jednom z telefonů. Nejhůře dopadla karta s hodnotou K, ta se rozpoznala špatně celkem třikrát z osmi pokusů.

	Počet chybně detekovaných karet od každé hodnoty	
Hodnota karty	**Galaxy Nexus**	**YotaPhone**
2	0	0
⋮		
10	0	0
J	0	1
Q	1	1
K	1	2
A	1	0

Tabulka 4.5: Počet chyb v závislosti na hodnotě karty

Výsledky testování jsou uspokojivé pro většinu hodnot karet. Ukázalo se, že optimální počet pro skenování jsou tři karty (aplikace je navržená maximálně pro skupiny čtyř karet). Také se objevily slabiny, na něž by se bylo vhodné zaměřit v případě dalšího vývoje.

Kapitola 5

Závěr

Cílem práce bylo vytvořit aplikaci pro detekci karet v obraze a rozpoznání jejich hodnot. Vyzkoušel jsem několik řešení založených na různých metodách zpracování obrazu. Od nefunkčních pokusů jsem se propracoval až k funkčímu rozpoznávání. Vytvořil jsem způsob určování hodnot karet z analýzy více po sobě následujících snímků.

Navržené uživatelské rozhraní hráčům umožňuje vyhodnotit stav hry po každé sehrávce. Během jeho implementace jsem se seznámil s problematikou programování aplikací pro mobilní zařízení. Poznal jsem tak jejich výhody, ale i omezení.

Aplikaci by zajisté bylo možné rozšířit o detekci barev karet. Toho by bylo možné dosáhnout například vytvořením kaskádových klasifikátorů pro symboly všech barev. Pravděpodobně by se tím zjednodušil i problém rozpoznání karet J,Q a K od karty dva. Nicméně, toto rozšíření by postrádalo smysl bez přidání podpory pro další karetní hru, jež má pro karty stejných hodnot a odlišných barev různá bodová hodnocení.

Literatura

[1] *The OpenCV Reference Manual*. 2014. Dostupné na: <http://docs.opencv.org/opencv2refman.pdf>.

[2] Eikvil, L. *OCR – Optical Character Recognition*. Oslo: Norsk Regnesentral, 1993. Technical Report, 876.

[3] Gargenta, M. *Learning Android*. Sebastopol: O'Reilly Media, Inc., 2011. ISBN 978-1-449-39050-1.

[4] Rivera, J. a Meulen, R. van der. *Gartner Says Sales of Smartphones Grew 20 Percent in Third Quarter of 2014* [online]. 2014 [cit. 27. 4. 2015]. Dostupné na: <http://www.gartner.com/newsroom/id/2944819>.

[5] Svoboda, T. *Oficiální pravidla karetních her*. Praha: Eminent, 2002. 270–271 s. ISBN 80-7281-116-9.

[6] Viola, P. a Jones, M. *Rapid Object Detection using a Boosted Cascade of Simple Features*. [b.m.]: Accepted Conference on Computer Vision and Pattern Recognition, 2001.