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

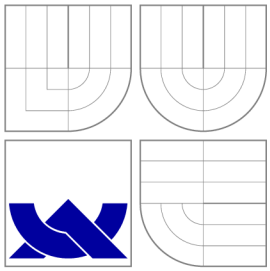
ŘEŠITEL LOGICKÉ HRY PRO ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

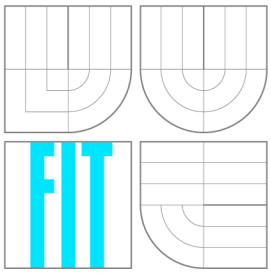
AUTOR PRÁCE
AUTHOR

DAN URBÁNEK

BRNO 2014



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

ŘEŠITEL LOGICKÉ HRY PRO ANDROID

LOGIC PUZZLE SOLVER FOR ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DAN URBÁNEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEXANDER PÁLDY

BRNO 2014

Abstrakt

Tato práce řeší návrh a implementaci aplikace pro mobilní platformu Android. Aplikace načte vstupní fotografii, vyhledá na ní zadání logické hry Light Up (Akari) a následně toto zadání vyřeší. V práci je krátce popsáno programování aplikací na platformě Android a také jsou popsány všechny použité algoritmy pro zpracování obrazu. Čtenář je také seznámen s testováním aplikace a zhodnocením dosažených výsledků. V práci jsou porovnány dva způsoby pro detekci znaků.

Abstract

This thesis describes the design and implementation of an application for the mobile platform Android. Initially the application reads an input image, searches for the game Light Up and afterwards solves the recognized game. The thesis briefly describes the programming on the Android platform and also describes all the algorithms used for the needed image processing. The reader is also informed about the testing of the application with an evaluation of the achieved results. A comparison of two algorithms for detecting characters is presented, too.

Klíčová slova

Android, Java, OpenCV, Tesseract, Akari, Light Up, zpracování obrazu

Keywords

Android, Java, OpenCV, Tesseract, Akari, Light Up, image processing

Citace

Dan Urbánek: Řešitel logické hry pro Android, bakalářská práce, Brno, FIT VUT v Brně, 2014

Řešitel logické hry pro Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Alexandera Páldyho. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dan Urbánek
9. května 2014

Poděkování

Rád bych tímto poděkoval vedoucímu mé bakalářské práce, Ing. Alexanderovi Páldymu, za odbornou pomoc a užitečné rady.

Obsah

Kapitola 1

Úvod

Obor počítačového vidění je obor, který má čím dál tím větší oblibu. Může to být způsobeno i cenovou dostupností a výkonností zařízení potřebných pro tento obor. V současné době je možné pozorovat také rychlý nástup chytrých mobilních zařízení. Tato zařízení nezřídka disponují poměrně dobrými fotoaparáty a na kapesní zařízení i vysokým výpočetním výkonem. Vzestupný trend rozvoje počítačového vidění i mobilních zařízení lze předpokládat i v budoucnu.

Lidé si rádi zkracují volné chvíle využíváním chytrých mobilních zařízení, ať už jde o konzumaci internetového obsahu či hraní her. Mezi známé hry, například z tištěných novin, patří jistě logická hra Sudoku či křížovky.

Tato práce se zabývá méně známou logickou hrou Light Up (též známa jako Akari). Uživatel zadání této hry vyfotografuje mobilním telefonem, aplikace fotografii zanalyzuje a hru vyřeší. Hráč si tak může zkontrolovat, zda sám hru vyřešil správně.

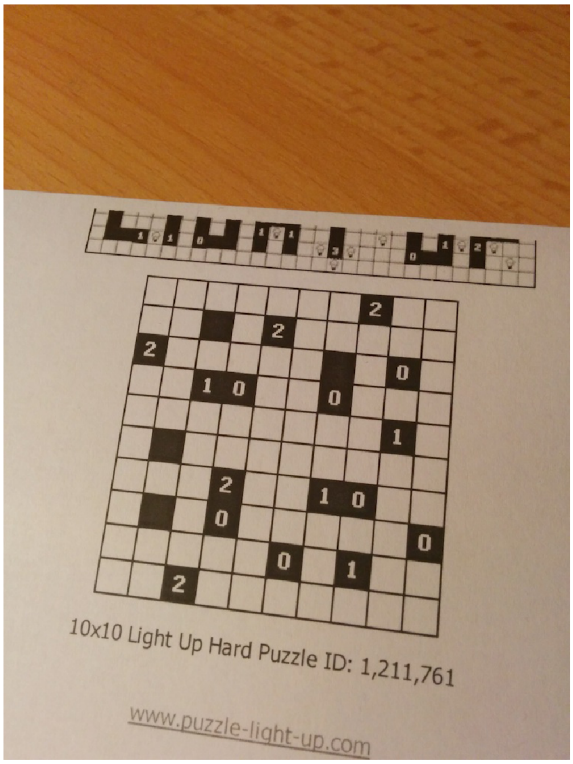
Cílem této práce je seznámit se s vývojem na mobilní platformě Android, navrhnout aplikaci na řešení hry Light Up z fotografie a tuto aplikaci implementovat. Pro účely této práce je nutné nastudovat různé metody zpracování obrazu.

Ve 2. kapitole bude čtenář seznámen s algoritmy pro zpracování obrazu, které jsou důležité pro tuto práci, a také s pravidly logické hry Light Up. 3. kapitola popisuje vývojovou platformu Android s knihovnami, které jsou použité v této práci, a také jsou popsány aplikace, které se zabývají podobnou tématikou. Samotným návrhem aplikace se zabývá kapitola 4. Implementace a testování navržené aplikace je tématem kapitoly 5. V poslední 6. kapitole jsou pak dosažené výsledky shrnuty.

Kapitola 2

Rozpoznání hry

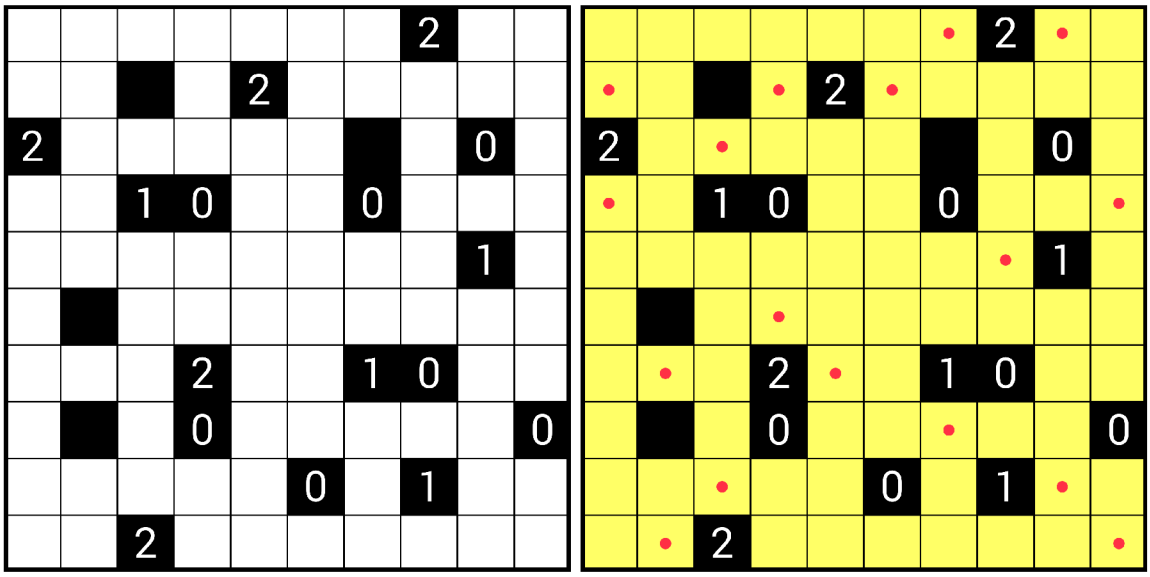
V této kapitole budou popsány způsoby, jak rozpoznat hru Light Up na fotografii. Jednotlivé kroky zpracování obrazu budou předvedeny na zadání z obrázku 2.1. Budou popsány algoritmy pro lokalizování hry v obraze, způsob rozpoznání velikosti detekované hry a také způsob rozpoznání jednotlivých hracích políček. Těmto krokům musí předcházet řada algoritmů, které obraz patřičně upraví do vhodnější reprezentace. I tyto algoritmy budou popsány v této kapitole. Nejprve však budou uvedena základní pravidla logické hry Light Up.



Obrázek 2.1: Vstupní fotografie

2.1 Pravidla logické hry Light Up

Light Up je logická hra vytvořená roku 2001 japonskou firmou Nikoli[1]. Tato hra je také známa pod názvem Akari. Hra se uskutečňuje na čtvercové nebo obdélníkové matici políček, která mohou mít černé nebo bílé zbarvení. Úkolem hráče je rozmístit žárovky tak, aby osvětlily celou hrací plochu. Žárovky se pokládají pouze na bílá pole. Položením žárovky na pole se rozsvítí horizontální i vertikální sloupec se žárovkou. Černé pole má význam zdi, kdy přes toto pole světlo neprojde. V tomto poli může být vepsáno číslo v rozsahu 0-4. Toto číslo určuje omezení kladená na požadovaný počet žárovek v okolí tohoto pole. Okolím je myšleno jedno sousední pole nahoře, dole, vlevo a vpravo. Zároveň platí omezení, že žárovka nesmí svítit na jinou žárovku ve stejném horizontálním nebo vertikálním sloupci. Ukázkové zadání hry a řešení je na obrázku 2.2.



Obrázek 2.2: Ukázkové zadání hry a její řešení

2.2 Barevné modely

V práci budou využity dva barevné modely pro práci s barvami - RGB model a šedotónový model. Fotografie na vstupu aplikace je dodána v RGB modelu. Pro použití s algoritmy pro rozpoznání hry je využit šedotónový model.

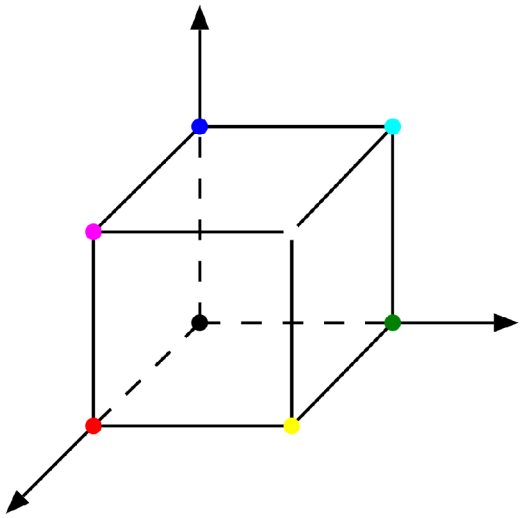
RGB barevný model[14]

Tento model využívá aditivního míchání barev. Výsledná barva vzniká mícháním jednotlivých základních barev. Je to červená (Red), zelená (Green) a modrá (Blue) barva. Aditivní míchání je charakteristické tím, že složením všech barevných složek s maximální intenzitou dostaneme bílou barvu. Naopak při nulové intenzitě všech složek dostaneme černou barvu. Způsob skládání barev je znázorněn na jednotkové krychli na obrázku 2.3.

Přidělením 8 bitů pro každou složku máme 24 bitů na jednu barvu. Dostaneme tak cca 16 milionů barev. Model RGB patří mezi nejpoužívanější modely a je používán například

[1] http://www.nikoli.co.jp/en/puzzles/akari.html

v zobrazovacích zařízeních. Nevýhodou tohoto modelu může být, že nelze intuitivně pracovat s barvou. Nelze jednoduše regulovat světlost, barevný tón nebo sytost barvy. Tento model není vhodný pro přímé zpracování obrazu.



Obrázek 2.3: RGB krychle [2]

GrayScale model[12]

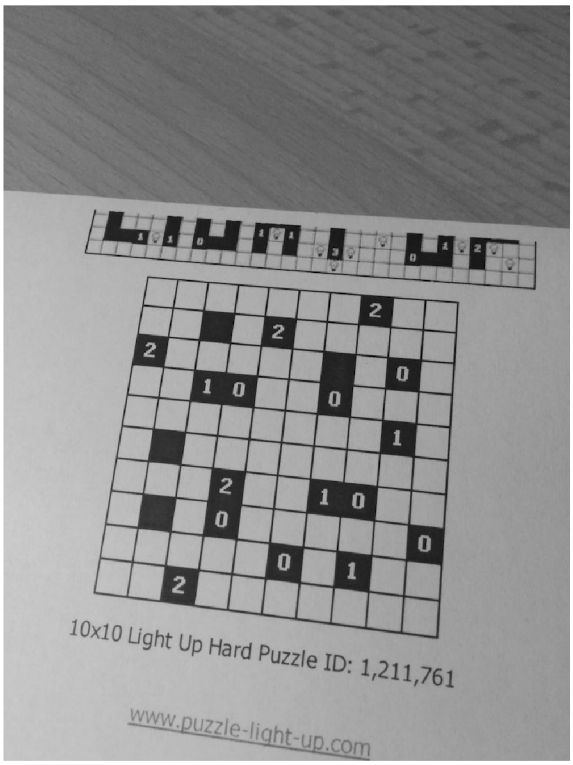
Šedotónový model vzniká redukcí barevného modelu. Pro převod se využívá empirického vztahu[3]:

$$I = 0.299R + 0.587G + 0.114B \tag{2.1}$$

Výstupem je hodnota pixelu šedotonového obrazu. Vstupem jsou jednotlivé hodnoty R, G a B barevného modelu RGB, přičemž každý bod má jinou váhu. Hodnoty vah byly určeny empiricky podle citlivosti lidského oka na jednotlivé vlnové délky světla. Oko je nejcitlivější na zelenou barvu.

V počítači je tento model reprezentován dvojrozměrnou maticí, kdy hodnota určuje jas v daném bodě. Bývá uložen s barevnou hloubkou 8 bitů. Tento model se používá pro kompresi s omezením barevného prostoru, pro zařízení, které neumí zobrazit barvy nebo pro jednodušší zpracování obrazu. Výsledek převodu vstupní fotografie 2.1 lze vidět na obrázku 2.4.

[2] Zdroj obrázku http://commons.wikimedia.org/wiki/File:Barevny_model_rgb.svg
[3] http://www.itu.int/rec/R-REC-BT.601-7-201103-I/en



Obrázek 2.4: Vstupní fotografie v šedotónovém modelu

2.3 Předzpracování obrazu

Samotnému rozpoznávání musí předcházet předzpracování obrazu. K redukci šumu a jemnému vyhlazení obrazu je použit algoritmus Gaussova rozostření obrazu. V různých částech práce je využito také segmentace obrazu, tedy rozčlenění obrazu do několika částí, které mají společné vlastnosti. Pro segmentaci je využit algoritmus prahování a také Cannyho hranový detektor. Při předzpracování obrazu je také využito matematické morfologie. Tyto algoritmy budou popsány.

Gaussovo rozostření obrazu[14][10]

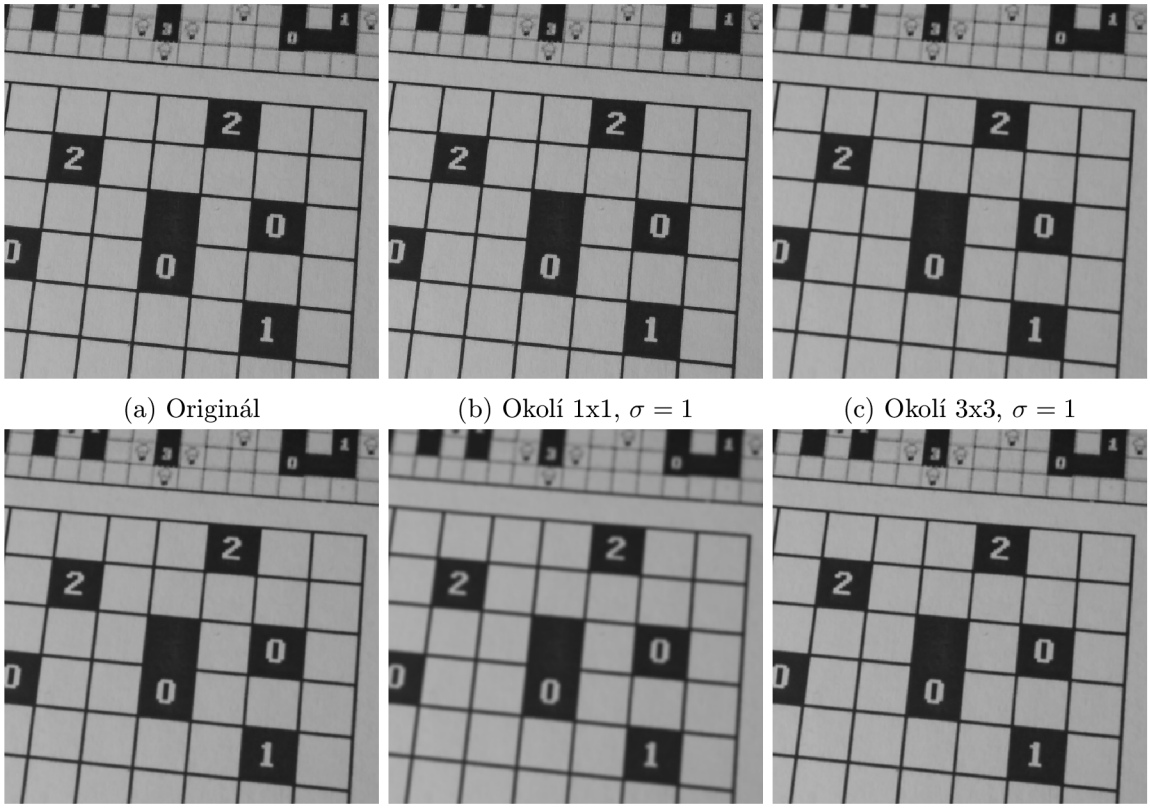
Rozostření obrazu je používáno pro odstranění šumu obrazu, vyhlazení obrazu a také pro zvýraznění určitých částí. Gaussovo rozostření je často používáno v souvislosti s algoritmy pro detekci hran. Dojde totiž k odstranění šumu, na který jsou detekční algoritmy velmi citlivé.

Hlavní myšlenkou je, že nové hodnoty vstupních bodů jsou vytvořeny váženým průměrem bodů v okolí. Výpočet nové hodnoty každého bodu je proveden pomocí konvoluce s maskou, která je dána transformační maticí. Pro výpočet nové transformační masky pro každý bod obrazu se využívá 2D Gaussova funkce. Ta je definována vztahem (2.2), kde x je vzdálenost od počátku na horizontální ose, y je vzdálenost od počátku na vertikální ose a σ je směrodatná odchylka Gaussovy funkce.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

Vstupní bod bude mít dle Gaussova rozdělení pravděpodobnosti největší váhu. Váha okolních bodů se pak snižuje s rostoucí vzdáleností od vstupního bodu.

Problémem tohoto typu rozostření může být správnost volby směrodatné odchylky a velikosti okolí vstupního bodu. Zvolíme-li malé hodnoty, dojde k jemné filtraci při zachování hran, u vyšších hodnot může dojít až k zániku méně výrazných hran, které však mohou být pro detekci důležité. Výsledek různého nastavení parametrů je ukázán na sérii fotografií obrázku 2.5.



(a) Originál (b) Okolí 1x1, $\sigma = 1$ (c) Okolí 3x3, $\sigma = 1$

(d) Okolí 9x9, $\sigma = 1$ (e) Okolí 9x9, $\sigma = 2$ (f) Okolí 3x3, $\sigma = 2$

Obrázek 2.5: Vstupní fotografie po aplikaci Gaussova rozostření obrazu s různými parametry

Prahování[14]

Tato metoda slouží k převodu obrazu šedotónového modelu na obraz černobílý. Na počátku je určena prahová hodnota (threshold) T. Využívá se převodního vztahu (2.3), kde $f(x, y)$ je vstupní šedotónový obraz, T je prahová hodnota a $G(x, y)$ je výstupní binární obraz.

$$G(x,y) = \begin{cases} 1 & \text{pro } f(x,y) < T \\ 0 & \text{jinak} \end{cases} \tag{2.3}$$

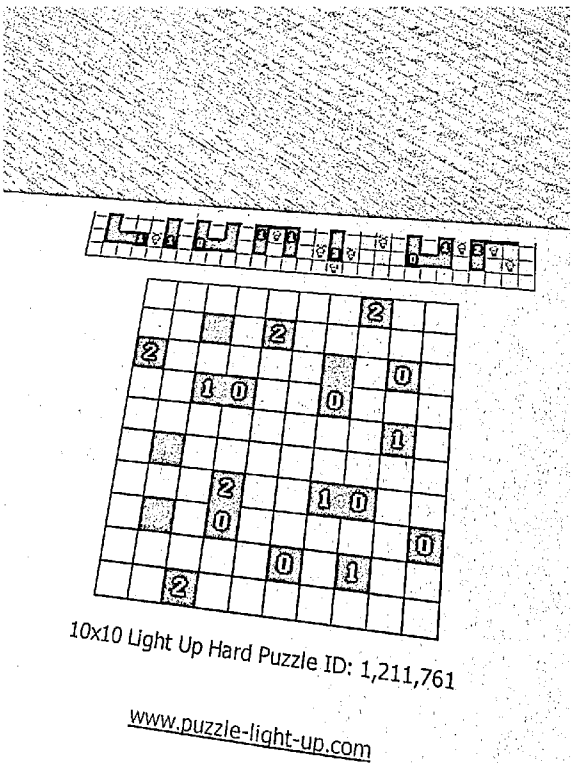
Každé hodnotě bodu pod prahovou hodnotou je přiřazena hodnota 1 (černá barva). Pokud je hodnota nad touto hranicí, je přiřazena hodnota 0 (bílá barva). Výsledkem převodu je tedy binární obraz, který obsahuje dvě hodnoty bodů - 0 a 1. Rozdělením bodů na dvě hodnoty lze při správně zvoleném prahu rozlišit popředí a pozadí obrazu.

Tento algoritmus je použit buď v základní variantě, nebo ve variantě adaptivní. V základní variantě je prahová hodnota globální pro celý obraz. V adaptivním prahování je

obraz nejprve rozdělen na několik částí, kdy pro každou část je nastavena lokální prahová hodnota. Poté je aplikováno prahování na každou tuto část zvlášť.

Problémovým místem je právě určení prahové hodnoty. Lze ji určit manuálně nebo automaticky. Manuálně může být určena analýzou dat z histogramu obrazu či prostým odhadem. Mezi automaticky vypočtené hodnoty patří například vypočtení střední hodnoty okolních bodů či využití váhového koeficientu vypočteného pomocí Gaussova rozdělení pravděpodobnosti. Tyto hodnoty jsou pro každý bod vypočteny z okolních bodů.

Výsledek aplikace adaptivního prahování na vstupní fotografii lze vidět na obrázku 2.6. Je zřejmé, že hranice hry více vystupuje z obrazu.



Obrázek 2.6: Fotografie po aplikaci adaptivního prahování

Cannyho hranový detektor[10]

Tento algoritmus patří k nejpopulárnějším detektorům hran. Je stanoveno několik základních podmínek pro správnou detekci hran. První podmínkou je požadavek na minimální počet chyb. Musí být detekovány všechny významné hrany a zároveň nesmí dojít k falešné detekci hran, které v obraze nejsou. Je tedy potřeba redukovat šum v obraze. Další podmínkou je přesnost detekce. Hrana musí být v obrazu detekována přesně. Rozdíl mezi nalezenou hranou a skutečným umístěním hrany v obraze by měl být minimální. Poslední podmínkou je jednoznačnost detekce. Nesmí dojít ke zdvojení detekovaných hran. To je nutné zejména u zašuměných obrázků, které nemají hladké hrany. Tyto základní podmínky stanovil autor algoritmu, John Canny, roku 1986 v odborném článku[4].

Algoritmus detekce lze rozdělit do 4 základních kroků. Nejdříve je potřeba eliminovat šum v obraze. Většinou je tento krok proveden za pomoci konvoluce vstupního obrazu (f) Gaussovým filtrem (G). Nicméně lze použít libovolný jiný filtr umožňující odstranění šumu.

Poté je potřeba určit směr a velikost hran v obraze. K tomu detektor může použít různé konvoluční masky pro určení horizontálního, vertikálního nebo diagonálního směru hrany. Pro konvoluční jádro můžeme použít například operátory: Robertsonův, Sobelův či operátor Prewittové. Směr normálových vektorů zjistíme vztahem (2.4).

$$\vec{n} = \frac{\nabla(f * G)}{|\nabla(f * G)|} \tag{2.4}$$

Směr není určen zcela přesně, ale jedná se o odhad. Velikost hrany je vyjádřena vztahem (2.5), kde G_n je první derivace G ve směru hrany $\vec{n}$, viz vztah (2.6).

$$|G_n * f| = |\nabla(G * f)| \tag{2.5}$$

$$G_n = \frac{\partial G}{\partial \vec{n}} \tag{2.6}$$

Dalším krokem je nalezení správné polohy hran. Využívá se techniky *nonmaximum suppression*, kdy odebíráme body, které nejsou lokálními maximy vypočtených derivací. Dojde tak ke ztenčení detekovaných hran. Správná poloha hrany je tehdy, když platí rovnice (2.7). Tedy konvoluce obrazu f s operátorem G_n. Po substituci rovnice (2.6) do rovnice (2.7) dostaneme rovnici (2.8).

$$\frac{\partial(G_n * f)}{\partial \vec{n}} = 0 \tag{2.7}$$

$$\frac{\partial^2(G * f)}{\partial \vec{n}^2} = 0 \tag{2.8}$$

Čtvrtým a posledním krokem je eliminace přebývajících bezvýznamných hran. Výsledné hrany získáme pomocí prahování s hysterezí. Pro tento typ prahování jsou stanoveny dva prahy, jeden vyšší (T_v) a druhý nižší (T_n). Pokud jsou hodnoty hran vyšší jak T_v pak jsou hrany potvrzeny jako skutečné. Jsou-li hodnoty hran nižší jak T_n, pak dojde k zahození příslušné hrany. Uvnitř tohoto intervalu jsou hrany uznány pouze tehdy, když byla uznána některá z okolních hran.

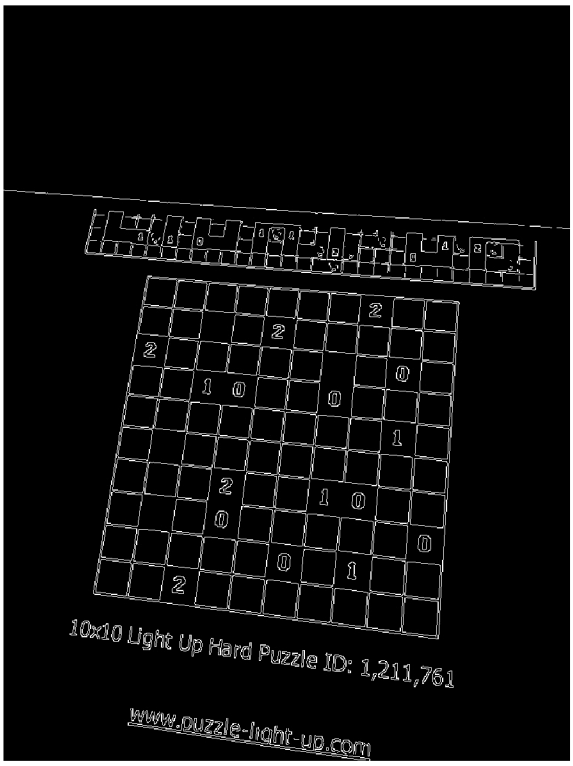
Vstupní fotografii z obrázku 2.1 po aplikování Cannyho hranového detektoru lze vidět na obrázku 2.7. Na fotografii je patrné zvýraznění hran hry a přilehlých nápisů.

Matematická morfologie[10][14]

Jedná se o zpracování obrazu v závislosti na jeho tvaru. Matematická morfologie se opírá o teorii množin, integrální algebru a algebru svazů.

Operace binární morfologie berou jako vstup binární obraz B a strukturní element S, který je dalším binárním obrazem, ale typicky je o hodně menší. Tento strukturní element představuje tvar. Může být libovolné velikosti i struktury, ale existuje několik běžně používaných struktur. Může se jednat například o obdélník, kruh či jakýkoliv jiný útvar. Morfologická transformace využívá relace binárního obrazu B se strukturním elementem S. Jeden bod strukturního elementu je určen jako počátek. Často se jedná o středový bod u symetrických objektů, ale jinak může být určen kdekoliv. Transformace probíhá umístěním počátku strukturního elementu na každý bod objektu. Strukturní element je pak rozprostírán přes celý obraz. Na počátku je výstupní obraz inicializován nulovými hodnotami.

Mezi základní operace binární morfologie patří dilatace a eroze. Existují také varianty těchto operací a to uzavření a otevření. Stejnou morfologickou operaci nemá smysl provádět znovu. Provedením by nedošlo k žádné změně. Tato vlastnost se nazývá idempotence.



Obrázek 2.7: Fotografie po aplikaci Cannyho hranového detektoru

Výsledkem eroze je obraz, který je jednodušší. Dojde ke ztenčení objektů v obraze. Výstupní obraz dostaneme postupným prováděním binární operace OR nad počátkem strukturního elementu a příslušným bodem vstupního obrazu.

Eroze je vyjádřena vztahem (2.9).

$$B \ominus S = \bigcap_{s \in S} B_{-s} \tag{2.9}$$

Pomocí dilatace můžeme zaplňovat menší části obrazu. Výsledné objekty v obraze jsou tak zvětšeny. Je to duální operace k erozi. Vždy, když se počátek strukturního elementu dotýká bodu s hodnotou 1 vstupního binárního obrazu, je provedena binární operace OR bodu strukturního elementu s výstupním obrazem. Operaci dilatace vyjadřuje vztah (2.10).

$$B \oplus S = \bigcup_{b \in B} S_b \tag{2.10}$$

Operace otevření je aplikace eroze následovaná dilatací. Dochází k odstranění menších objektů. Výsledný obraz má tak méně šumu. Vztah popisující operaci je (2.11).

$$B \circ S = (B \ominus S) \oplus S \tag{2.11}$$

Operaci uzavření získáme aplikací dilatace, po níž následuje eroze. Uzavírá menší vnitřní díry v objektech. Tuto operaci lze popsat vztahem (2.12).

$$B \bullet S = (B \oplus S) \ominus S \tag{2.12}$$

1	1	1	1	1	1	1	
			1	1	1	1	
			1	1	1	1	
		1	1	1	1	1	
			1	1	1	1	
		1	1				

(a) Binární obraz B

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1
	1	1	1	1	1	1	1
	1	1	1	1	1	1	1
	1	1	1	1	1	1	1
	1	1	1	1			

(b) Dilatace $B \oplus S$

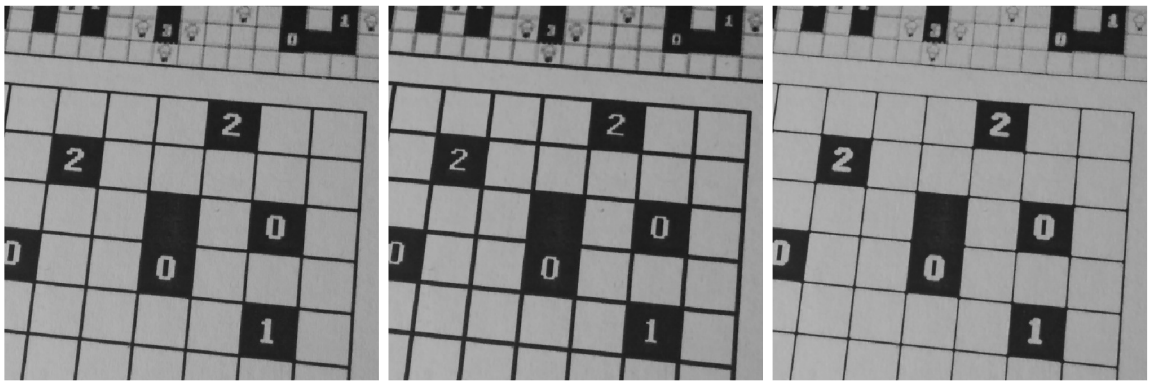
				1	1		
				1	1		
				1	1		

(c) Eroze $B \ominus S$

Tabulka 2.1: Ilustrace eroze a dilatace se strukturním elementem S

Ilustrace operace morfologie na binárním obrazu B je na tabulkách 2.1. Strukturní element S má tvar čtverce o velikosti 3x3 bodů a počátek má umístěn doprostřed.

Původně se morfologie používala na úpravu binárních obrazů. Byla pak však rozšířena a nyní ji lze použít i k úpravě obrazů ve stupních šedi. Efekt dilatace a eroze lze vidět na sérii fotografií na obrázku 2.8. Je použit stejný strukturní element jako v ilustračních tabulkách 2.1. Originál je výřezem ze vstupní fotografie ve stupních šedi 2.4.



(a) Originál (b) Eroze (c) Dilatace

Obrázek 2.8: Vstupní fotografie po aplikaci eroze a dilatace se strukturním elementem S

2.4 Detekce objektů

V této podkapitole budou popsány algoritmy pro rozpoznání, ve které části fotografie se hra nachází. Nejprve bude představen algoritmus Border following, který poslouží k detekci, ve které části obrazu se hra nachází. Dále bude představen algoritmus Houghovy transformace. Pomocí něj je možné detekovat čáry v obraze. Bude tak možné rozpoznat počet řádků a sloupců hry. Bude také představen algoritmus K-means pro shlukování dat na základě podobných vlastností.

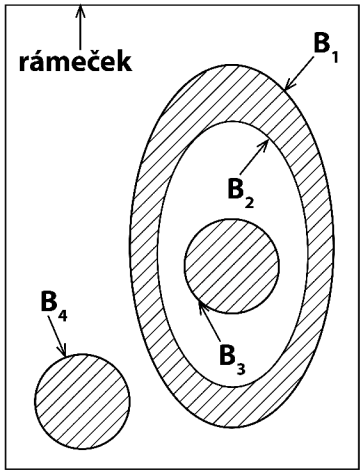
Border following[13][16]

Tento algoritmus slouží k detekci a rozpoznání obrysů v binárním obraze, který byl nejdříve zpracován například Cannyho hranovým detektorem či prahováním.

Nejprve je třeba zavést souřadný systém. Souřadnice bodů v obraze jsou v pořadí (řádek, sloupec). Číslo řádku se zvyšuje shora dolů, číslo sloupce pak zleva doprava.

Na začátku jsou určeny tři typy značení bodů. Body, které mají hodnotu 0, nejsou hranou ani obrysem. Body, které mají hodnotu 1, jsou určeny jako body hrany, které zatím nejsou označeny jako obrys. Body, kterým náleží jakákoliv jiná hodnota (kladná nebo záporná), jsou použity pro označení již detekovaného obrysu.

Detekované obrysy jsou děleny do dvou kategorií, na vnější či vnitřní. Vnější obrys je takový, který má oblasti složené z bodů nabývajících kladné hodnoty a je obklopován oblastí, která je složená z bodů nulových hodnot. Naopak vnitřní obrys je složen z kladných hodnot, které přímo obklopují oblast obsahující nulové hodnoty bodů. Znázornění jednotlivých typů obrysů je na obrázku 2.9. B_1, B_3 a B_4 jsou vnější obrysy, obrys B_2 je vnitřní.



Obrázek 2.9: Typy obrysů

Procházením obrazu zleva doprava po jednotlivých řádcích hledáme takový bod (x, y), který splňuje podmínku bodu obrysu. Může se jednat o bod z kterékoliv kategorie obrysů. Podmínky pro určení, zda se jedná o počáteční bod vnitřního nebo vnějšího obrysu jsou uvedeny v tabulce 2.2.

	y-1	**y**
x	0	1

(a) Vnější hrana

	y	**y+1**
x	≥1	0

(b) Vnitřní hrana

Tabulka 2.2: Určení typu hrany

Pokud bod splňuje podmínku pro oba typy obrysu, je považován za počáteční bod vnějšího obrysu. Nově nalezenému bodu hranice přiřadíme unikátní identifikační číslo. Nazveme jej sekvenčním číslem hranice a označíme jej NBD.

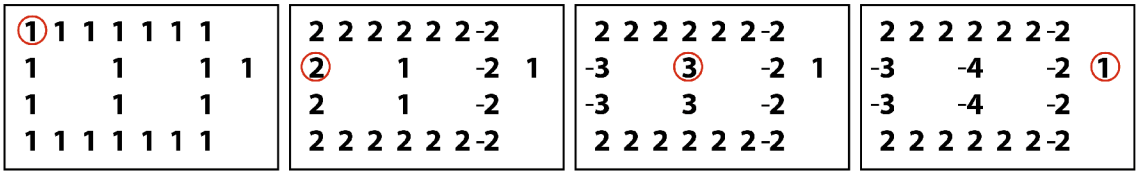
Následujeme nalezenou hranici od počátečního bodu a označujeme další nalezené body. Pro označování bodu platí dvě pravidla:

(a) Pokud bod $(x, y + 1)$ obsahuje nulovou hodnotu a (x, y) obsahuje hodnotu 1, změní se hodnota bodu (x, y) na zápornou hodnotu čísla NBD.

(b) Jinak pokud nebyl bod (x, y) již jednou označen jako obrys, je nastavena hodnota bodu (x, y) na hodnotu čísla NBD.

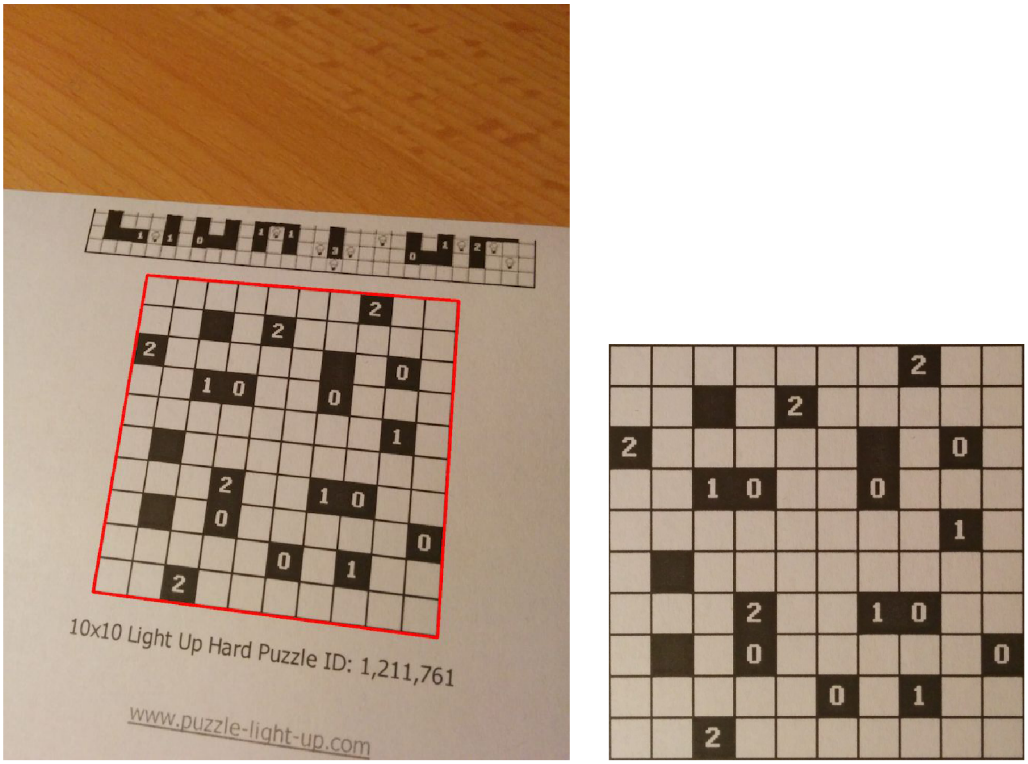
Algoritmus poté pokračuje, dokud nedosáhne konce obrazu v pravém dolním rohu. Poté záleží již na způsobu interpretace výsledků. V případě této práce je vybrána hranice, která má největší plochu. U nalezené hranice můžeme použít všechny nalezené body nebo můžeme využít aproximace a vybrat jen rohové body.

Algoritmus je znázorněn na sérii ilustračních obrázků 2.10. Čísla ukazují hodnoty bodů. U zakroužkovaného čísla se jedná o startovní bod. Na obrázku a) lze vidět, že počáteční bod byl rozpoznán jako vnější obrys a proto je označen sekvenčním číslem $NBD = 2$. Z obrazu b) je vidět, že bod v kroužku je označen za vnitřní hranu dle tabulky 2.2. Bude proto označen číslem $NBD = 3$. Podobně tomu je i u obrázku c). Na posledním obrázku d) je už jen vidět detekce dalšího obrysu sestávajícího se pouze z jednoho bodu. Bude tedy označen následujícím sekvenčním číslem $NBD = 5$.



Obrázek 2.10: Ilustrace procesu rozpoznání hranice algoritmem Border following

Na obrázku 2.11 je vyznačena nalezená hlavní hranice, tedy ta s největší plochou. Po aproximaci na 4 rohové body může dojít k ořezu fotografie a jejímu vycentrování.



Obrázek 2.11: Vyznačená rozpoznaná hranice hry algoritmem Border following a oříznutá a vycentrovaná fotografie

Houghova transformace

Pro napsání této kapitoly byly využity dvě knihy[1][10] popisující základní algoritmy pro zpracování obrazu. Cenné informace byly čerpány také z knihy[7] popisující zpracování obrazu programem Matlab[4]. Tento program byl využit pro generování některých grafů této kapitoly.

V této kapitole bude popsán algoritmus Houghovy transformace, který bude využit pro rozpoznání velikosti hry v počtu řádků a sloupců. Princip Houghovy transformace publikoval Paul Hough roku 1959, k patentování došlo roku 1962 ([3]). V roce 1972 došlo pak k zobecnění této metody[6].

Tato technika je určená k lokalizování jednoduchých tvarů v obraze, které lze popsat křivkami. Hledáme tedy parametrický popis objektu. Původně byla tato transformace využívána pro rozpoznání přímek, nicméně s úpravami lze tuto techniku použít i pro rozpoznání jiných útvarů, například kružnic či elips. Potřebujeme znát analytický popis tohoto objektu. Pro detekci libovolného tvaru slouží upravená verze tohoto algoritmu, která je známa jako zobecněná Houghova transformace (generalized Hough transform).

Přímka může být reprezentována v kartézském souřadném systému směrnicovým tvarem (2.13), kde k je směrnice (sklon přímky, $k = \tan\varphi$) a q je y souřadnice místa, kde přímka protne osu y.

$$y = kx + q \tag{2.13}$$

Může být také vyjádřena v polárních souřadnicích normálovým tvarem (2.14), kde ρ je délka normály od přímky k počátku souřadnic a θ je úhel, který svírá normála přímky s osou x. Tento úhel může nabývat hodnot od 0 do 2π.

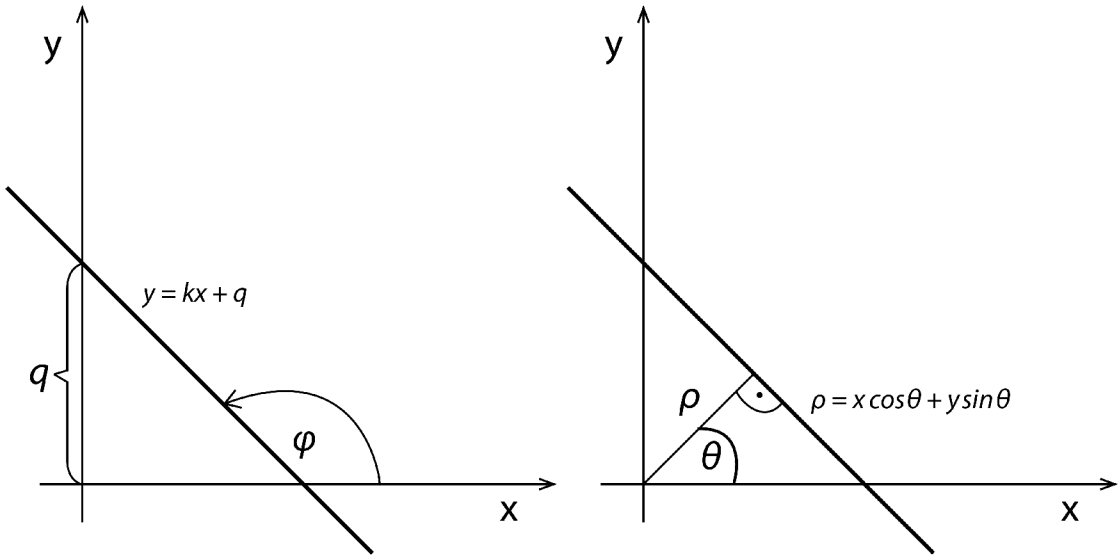
$$\rho = x\cos\theta + y\sin\theta \tag{2.14}$$

Při použití směrnicového tvaru se může u vertikálních přímek směrnice blížit nekonečnu. Proto je pro použití s Houghovou transformací vhodnější normálový tvar přímky. Znázornění zobrazení přímky v obou tvarech je na obrázku 2.12.

Transformace využívá převodu křivky z vyjádření v kartézských souřadnicích do parametrického prostoru. Transformovaný prostor se nazývá akumulátor a může být implementován například pomocí 2D matice, která má jako rozměr velikost ρ a θ. Na počátku jsou hodnoty akumulátoru inicializovány hodnotou 0. Na vstupu je očekáván obraz v binárním tvaru, tedy obraz, který byl předzpracován například Cannyho hranovým detektorem nebo prahováním. Známe vstupní souřadnice bodů nalezených hran, avšak neznáme hodnoty parametrů ρ a θ. Kombinace těchto dvou parametrů určuje právě jednu přímku. Hledáme tedy takovou kombinaci hodnot, která má nejblíže k určení příslušné přímky v obraze. Generujeme proto nové hodnoty θ inkrementací a parametr ρ dopočítáme. Dosazením do rovnice (2.14) dostaneme spojitou křivku funkce sinus v Houghově prostoru. Toto opakujeme pro každý bod nalezené hrany, tedy tam, kde platí $f(x, y) = 1$. Všechny sinusoidy jedné hrany se protínají v jednom bodě. Pro každou kombinaci hodnot ρ a θ proto inkrementuje hodnotu uloženou v akumulátoru. Největší hodnota uložená v akumulátoru patří parametrům přímky, která má nejvíce bodů v obraze. V akumulátoru můžeme najít hrany například jen určitých velikostí, úhlů nebo jen vybrat jednu nejdelší.

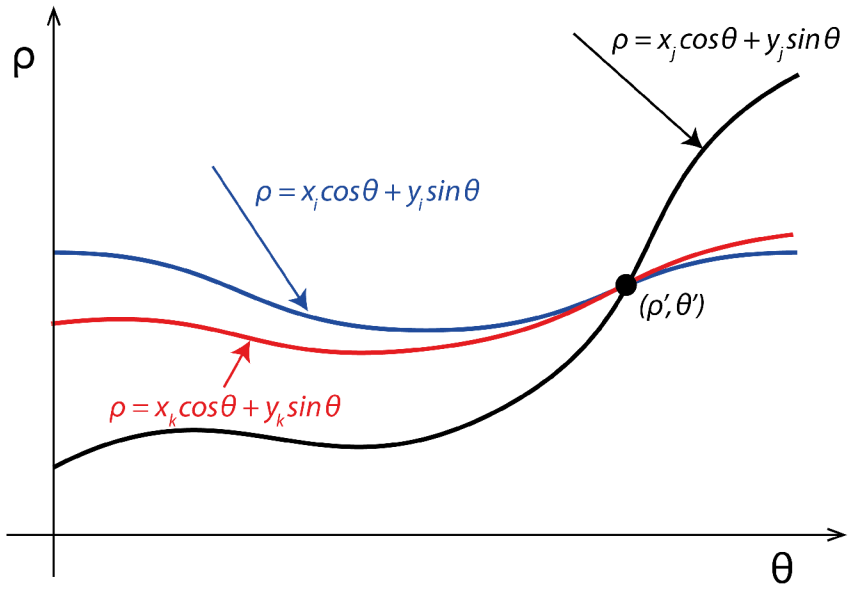
Znázornění parametrického prostoru je na obrázku 2.13. Na obrázku je vyznačen průsečík (ρ', θ'), který určuje parametry přímky, jež prochází body (x_i, y_i), (x_j, y_j) a (x_k, y_k).

[4] http://www.mathworks.com/products/matlab/



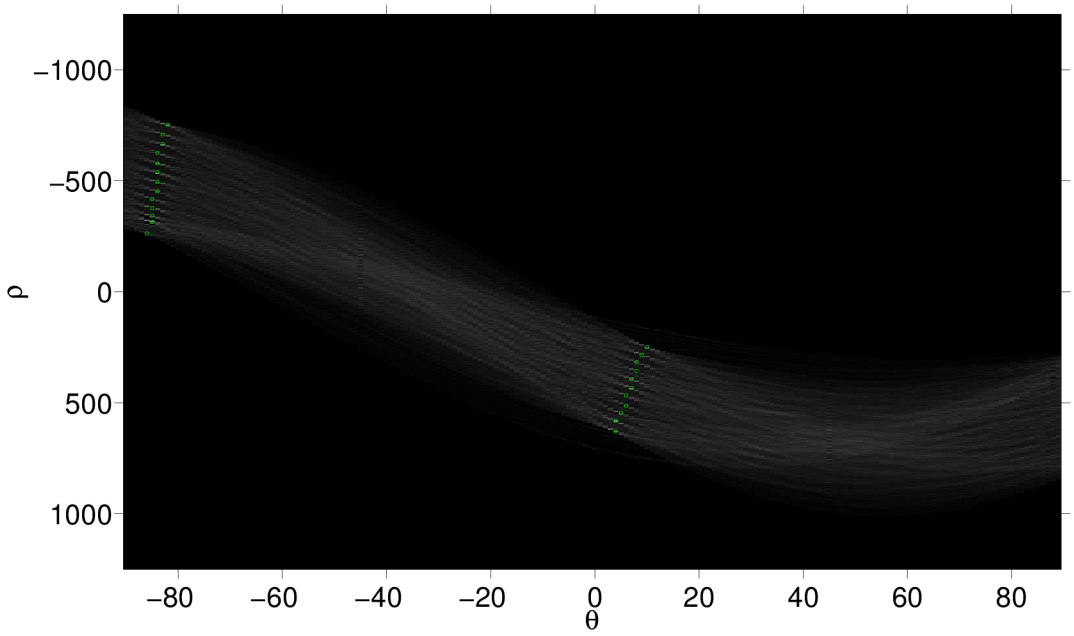
(a) Směrnicový tvar (b) Normálový tvar

Obrázek 2.12: Znázornění rovnic přímky

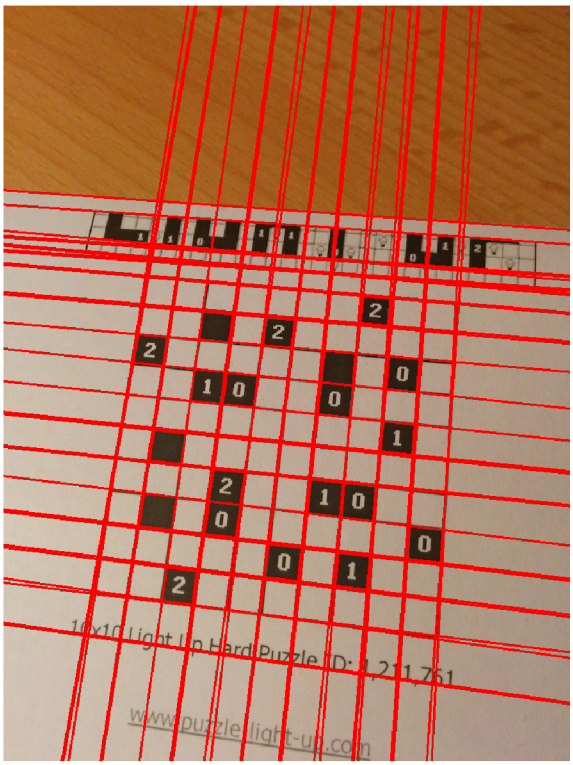


Obrázek 2.13: Znázornění parametrického prostoru

Výsledný akumulátor vytvořený ze vstupní fotografie 2.1 je na grafu v obrázku 2.14. Na grafu jsou vyznačeny nejvýznamnější hodnoty akumulátoru. Vstupní fotografie se zvýrazněním detekovaných přímek je na obrázku 2.15.



Obrázek 2.14: Zobrazení akumulátoru po detekci přímek na vstupní fotografii



Obrázek 2.15: Fotografie s nalezenými přímkami

Shlukování[14][10]

Shlukování je technika pro sdružování dat do shluků na základě jejich podobnosti. Zde existuje celá řada různých algoritmů. Avšak nejpoužívanějším je algoritmus K-means, kdy

shlukujeme n vstupních bodů do k shluků. Algoritmus přestavil John McQueen v článku[9] z roku 1967.

Algoritmus je iterativní. Musíme rozdělit množinu vektorů dimenze n do k podmnožin. Přitom musí platit, že suma vzdáleností příslušných vektorů musí mít co nejmenší vzdálenost od středu podmnožiny. Nejmenší vzdálenost K shluků C_1, C_2, ..., C_K s jejich středy m_1, m_2, ..., m_K popisuje i vzorec (2.15).

$$D = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - m_k\|^2 \tag{2.15}$$

Algoritmus probíhá následovně.

1. Určíme počet shluků, množinu všech vektorů a náhodně zvolíme středy podmnožin.
2. Vypočítáme vzdálenost každého z vektorů ke středům a přiřadíme jej do shluku, který má tuto vzdálenost nejmenší.
3. Vypočítáme nové středy podmnožin pomocí aritmetického průměru všech bodů shluku.
4. Opakujeme kroky 2 a 3, dokud se přiřazení vektorů do shluků mění nebo jsou změny minimálního charakteru.

Problémem algoritmu může být počáteční volba středu.

Kapitola 3

Použité vývojové prostředky

V kapitole budou popsány vývojové prostředky, které bylo potřeba nastudovat pro návrh a implementaci této aplikace. Jedná se především o programování na platformě Android. Dále bude popsána knihovna OpenCV, která slouží ke zpracování obrazu. V práci bude využita také knihovna Tesseract sloužící k rozpoznání textu.

3.1 Vývojová platforma Android

Pro napsání této kapitoly bylo využito především oficiálních internetových stránek[1] systému. Pro seznámení s programováním na této platformě byly kromě zmíněných stránek ještě použity knihy[11][17], které popisují některé programové konstrukce srozumitelněji.

Android je poměrně mladá open source platforma určena především pro mobilní zařízení. Nejprve v roce 2003 vznikla společnost Android Inc., která se zabývala vývojem aplikací pro mobilní zařízení. O dva roky později ji odkoupila společnost Google. Poté došlo k vývoji systému Android, který je založen na jádře Linux. Primárním jazykem, ve kterém se pro tento systém programují aplikace, je Java. Systém byl představen roku 2007 a současně byla založena organizace Open Handset Alliance (OHA), která zastřešuje několik desítek firem od mobilních operátorů, výrobců mobilních zařízení po výrobce jednotlivých polovodičových součástek.

První komerční verze 1.0 společně s prvním telefonem byla představena roku 2008. Od té doby vyšlo dalších 18 verzí systému, přičemž současná verze má označení 4.4. Každá verze přináší opravy chyb a dochází také k přidávání funkcionality či změnám vzhledu. Verze 3 byla určena jen pro tablety a přinesla především nové grafické prostředí. Pozdější verze 4, která sjednotila verze pro mobilní telefony a tablety, přišla s novým vzhledem systému založeným na tabletové verzi 3. Nová verze systému vychází zpravidla dvakrát ročně.

Ne každému zařízení je umožněna aktualizace na nejnovější verzi. Stále jsou v oběhu zařízení, která mají i tři roky starou verzi. Pro vývojáře na této platformě je tak důležité si nejdříve stanovit, kterou nejnižší verzi systému budou podporovat ve své aplikaci. Od toho se odvíjí, které funkce systému budou moci vývojáři použít. Každý měsíc společnost Google vydává tabulku[2] s aktuálním procentuálním zastoupením jednotlivých verzí systému. Pro zajištění zpětné kompatibility některých nových funkcí se staršími verzemi systému existuje knihovna Android Support Library.

[1] http://developer.android.com/

[2] https://developer.android.com/about/dashboards/

Architektura systému obsahuje pět vrstev, které pracují samostatně, ale dochází k jejich vzájemné spolupráci. Nejnižší vrstvou je samotné jádro Linux. Zde dochází k abstrakci mezi hardwarem a softwarem vyšších vrstev. Následující vrstvou jsou knihovny, které obsahují všechny základní API pro vývoj aplikací. Jsou to například knihovny pro práci s XML, základní obsluhu systému, grafické prvky či obsluha komunikačních prvků zařízení. Kromě Android API jsou obsaženy ještě knihovny napsané v jazyce C/C++. Zde se jedná například o knihovnu OpenGL pro 3D grafiku, FreeType pro vykreslování písma, SQLite pro relační databáze či SSL pro šifrování internetové komunikace. Dále je zde vrstva obsahující virtuální stroj Dalvik Virtual Machine (DVM). Je to obdoba standardního virtuálního stroje, který se používá při vývoji Java aplikací - Java Virtual Machine (JVM). Dalvik je však více optimalizován pro výkon a spotřebu mobilních zařízení. Také neobsahuje všechny knihovny, které obsahuje JVM. Jsou to především knihovny pro grafické rozhraní, které má Android řešené jiným způsobem. Pro vývojáře na této platformě je nejdůležitější aplikační vrstva. Ta poskytuje programátorům přístup ke službám systému. Jsou to prvky pro tvorbu grafického rozhraní, přístup k jiným aplikacím, řízení životního cyklu aplikací či přístup k jednotlivým souborům zařízení. Poslední vrstvou jsou již samotné aplikace. Zde se může jednat o aplikace již předinstalované v systému či uživatelsky doinstalované.

Android Standard development Kit (SDK)[3] obsahuje základní vývojové nástroje, emulátor platformy, knihovnu API rozhraní, ukázkové zdrojové kódy aplikací a taky dokumentaci. SDK je dostupné pro operační systémy GNU/Linux, OS X i Windows. Základem jsou nástroje pro vývoj a ladění aplikace a také nástroj Android Development Bridge (ADB), který umožňuje komunikaci s připojeným fyzickým či emulovaným zařízením.

Android Native Development Kit (NDK)[4] je sada nástrojů, která umožňuje programování v jazycích C/C++. To může být vhodné pro aplikace, které provádějí výpočetně náročné operace či pro znovupoužití již vytvořených knihoven v těchto jazycích. Pomocí NDK tak můžeme klíčové funkce implementovat v jazyce C/C++. Ty jsou pak volány pomocí Java Native Interface (JNI).

Vývoj probíhá na počítači a aplikace je pak kompilována pro spuštění na mobilním zařízení. Aplikaci můžeme spustit také v emulátoru. Existuje více variant používaných emulátorů. Jednou variantou je oficiální emulátor, který emuluje přímo architekturu ARM. Tento emulátor však nedisponuje velkou rychlostí. Další variantou je použít emulátor Genymotion[5], který je oproti oficiálnímu emulátoru podstatně rychlejší, což sami tvůrci zmiňují. Neemuluje totiž přímo architekturu ARM, ale běží na architektuře Intel. Toto může být nevýhodou, pokud potřebujeme v aplikaci používat přímo některé speciální instrukce architektury ARM. Emulátor obsahuje také další nadstandardní služby, mezi které patří například propojení kamery počítače s emulovaným Android zařízením či simulace různých senzorů. V základní verzi je zdarma, pokročilé funkce pak za příplatek. Tento emulátor je používán v této práci.

Aplikace je obvykle vyvíjena ve vývojovém prostředí. Pro Android existují dvě oficiální vývojová prostředí. Nejstarším prostředím je Eclipse[6], do kterého je nutné nejprve doinstalovat zásuvný modul Android Development Tools (ADT)[7]. Druhou variantou je využití prostředí Android Studio[8], které však prozatím nemá finální sestavení. Toto prostředí je

[3] http://developer.android.com/sdk/
[4] https://developer.android.com/tools/sdk/ndk/
[5] http://www.genymotion.com/
[6] https://www.eclipse.org/
[7] http://developer.android.com/tools/sdk/eclipse-adt.html
[8] http://developer.android.com/sdk/installing/studio.html

založeno na produktu IntelliJ IDEA[9] od firmy JetBrains, který slouží pro vývoj Java aplikací. Prostředí IntelliJ IDEA má však rozšířenou podporu i pro vývoj aplikací pro Android. Toto prostředí je využito k vývoji aplikace v této práci. Nicméně lze používat i jiné prostředí umožňující programování v Javě či lze programovat v klasickém textovém editoru a kompilovat aplikace pomocí příkazů.

Hlavní komponentou aplikace této práce bude tzv. aktivita. Aktivita obsahuje grafické rozhraní jedné obrazovky aplikace. V aplikaci bývá obvykle více těchto aktivit a jsou navzájem provázány. Aktivita se může nacházet v několika různých stavech. Jednotlivé aktivity mezi sebou komunikují pomocí komponenty Intent.

3.2 Existující aplikace s podobnou tématikou

Pro mobilní platformy existuje již několik aplikací s podobnou tématikou. Většina je zaměřena na logickou hru Sudoku[10]. Pro logickou hru Light Up obdobná aplikace nebyla nalezena. V následujících sekcích budou popsány jednotlivé aplikace zaměřené na hru Sudoku, způsoby jejich řešení a taky shrnutí jejich kladů a záporů.

Jako referenční zařízení byly použity telefon Google Nexus 5[11] a tablet Google Nexus 7 (2012)[12].

Google Goggles[13]

Jedná se o aplikaci firmy Google, která je určená pro rozpoznávání fotografií. Je dostupná, jak pro platformu Android, tak i pro Apple iOS. Umožňuje rozpoznat z fotografie např. známou památku, čárový kód a další objekty. Jedním z objektů je i hra Sudoku. Mezi další funkce patří rozpoznávání znaků z fotografie (OCR).

Aplikace po spuštění zobrazí přímo výstup fotoaparátu. Po kliku na určené tlačítko vytvoří fotografii a provede analýzu této fotografie. Na fotografii pak zobrazí detekované oblasti a typ objektu(znaky, památky, ...). Průběh detekce lze vidět na obrázcích 3.1. Aplikace pro svou činnost potřebuje internetové připojení. Všechny vytvořené fotografie zasílá na servery společnosti Google, kde také probíhá analýza. Konkrétní způsob řešení rozpoznávání nebyl představen.

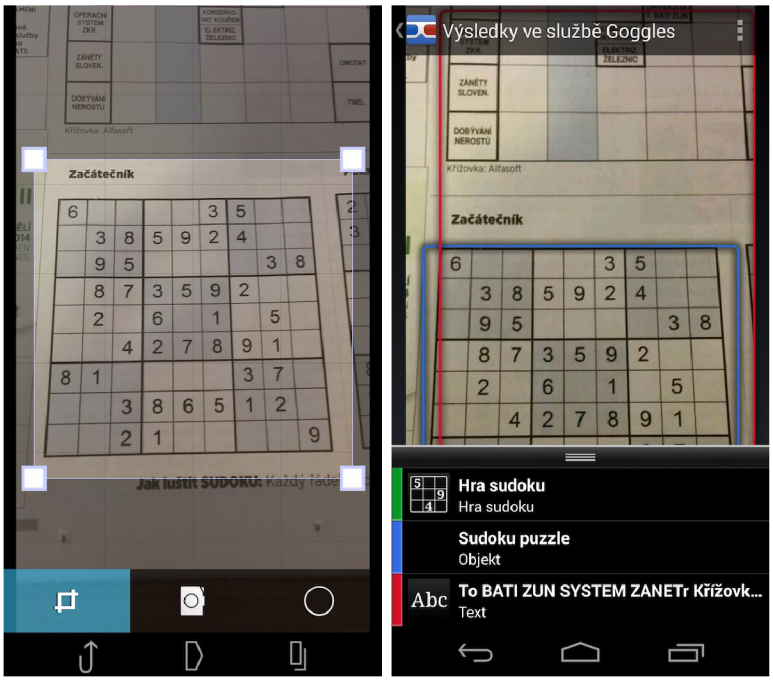
Mezi klady aplikace lze zařadit hlavně široký záběr druhů rozpoznaných objektů. Záporem pak může být především nutnost připojení k internetu.

AR Sudoku Solver[14]

Aplikace je k dispozici pouze jako technické demo. Po spuštění zobrazí výstup z fotoaparátu a rovnou se pokouší o detekci Sudoku. Pokud detektor nezjistí výskyt, je celý výstup aplikace zabarven do červena, viz obrazovkové snímky na obrázku 3.2. V případě, že se detekce povede, je rovnou obraz doplněn o správné řešení hry.

Autor uvádí[15], že zobrazovaný výstup odpovídá zhruba 5 snímkům za sekundu. Zobrazovaný výstup tedy není zobrazován plynule v reálném čase. Program žádá o další snímek

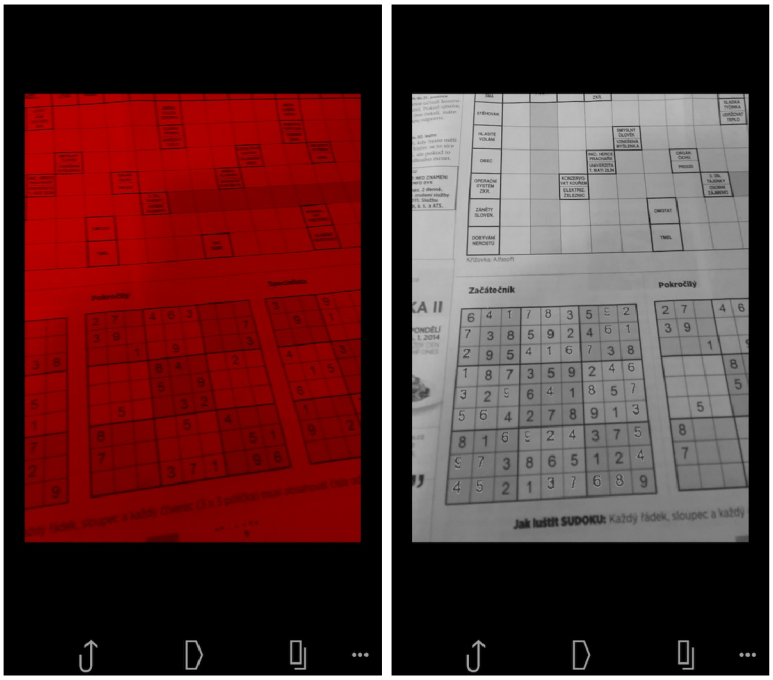
[9] http://www.jetbrains.com/idea/
[10] http://en.wikipedia.org/wiki/Sudoku
[11] https://www.google.com/nexus/5/
[12] http://en.wikipedia.org/wiki/Nexus_7_(2012_version)
[13] https://play.google.com/store/apps/details?id=com.google.android.apps.unveil
[14] https://play.google.com/store/apps/details?id=com.enigon.sudokusolver
[15] http://enigon.com/misc/en/arsudokusolver.html



Obrázek 3.1: Google Goggles

až po zpracování aktuálního. Tento program má zároveň omezení, že neumožňuje řešení her, kde je méně než 20 symbolů. Dalším omezením je nutnost výskytu všech symbolů hry.

Oproti předchozí aplikaci je výhodou zpracování bez připojení k internetu a také zobrazení řešení přímo do pořízené fotografie. Nevýhodou pak je nízká rychlost aplikace a již zmíněné omezení řešitelnosti hry.



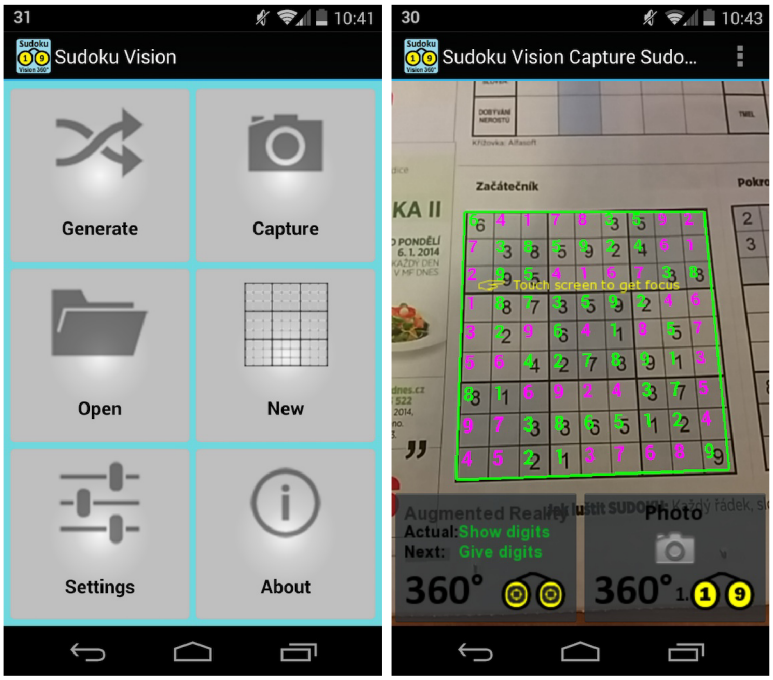
Obrázek 3.2: AR Sudoku Solver

Sudoku Vision[16]

Aplikace po spuštění svým vzhledem nezaujme. Ten příliš neodpovídá designovým standardům[17] Androidu 4. Menu obsahuje několik možností - načtení Sudoku ze souboru, vyfocení a rozpoznání hry a také možnost hraní z uložených variant nebo vygenerování hry nové. Aplikace se tedy oproti předchozím liší již v počátku tím, že umožňuje i hraní Sudoku. Celý proces rozpoznání byl na referenčním mobilním telefonu subjektivně rychlý. Řešení hry je možné zobrazit v pořízené fotografii.

Autor na svém blogu[18] popisuje, jak jednotlivých 9 částí rozpoznávání v jeho aplikaci funguje. Aplikace také může uživateli ukázat výstupy jednotlivých částí. Autor využívá semínkového vyplňování pro nalezení rozhraní hry a číslic. Pro nalezení rohových bodů používá Houghovu transformaci. Pomocí nalezených bodů provede perspektivní transformaci. Následuje už jen provedení rozpoznání čísel, které je implementováno pomocí semínkového vyplňování oblastí kolem čísla.

Výhodou je určitě možnost hraní hry. Velkou výhodou v porovnání s předchozími aplikacemi je rychlost zpracování. Za nevýhodu lze považovat hlavně překomplikované grafické uživatelské rozhraní, které může být pro uživatele místy matoucí.



Obrázek 3.3: Sudoku Vision

3.3 Knihovna OpenCV pro zpracování obrazu

Open Source Computer Vision Library (OpenCV)[19] je knihovna zaměřená na počítačové vidění a strojové učení. První verze byla vyvinuta v roce 1999 společností Intel. Knihovna

[16] https://play.google.com/store/apps/details?id=com.rogerlebo.sudokuvision
[17] https://developer.android.com/design/get-started/principles.html
[18] http://sudokuvision.blogspot.cz/
[19] http://opencv.org/

je napsána v jazyce C++, ve kterém je i hlavní rozhraní knihovny. Je uvolněna pod licencí BSD, tudíž ji lze použít zdarma i pro komerční účely. OpenCV v současnosti obsahuje přes 2500 různých algoritmů. Obsahuje mnoho funkcí pro práci s obrázky, aplikaci různých filtrů, detekci objektů apod.

OpenCV podporuje všechny hlavní desktopové operační systémy (GNU/Linux, OS X, Windows), mobilní platformy (Android, iOS) a řadu dalších systémů. Podporovány jsou především jazyky C, C++, Python a Java. Na Androidu lze použít OpenCV ve verzi pro Javu či ve verzi C/C++ při programování pomocí Android NDK. Při použití s Javou je nutné na cílovém zařízení nainstalovat program OpenCV Manager, který zajistí instalaci aktuální verze knihovny. Tento program také zajišťuje aktualizaci knihovny.

Hlavní a základní třídou knihovny je třída `Mat`[20]. Je to n-dimenzionální pole, které slouží k uložení vektorů, matic, obrazů a dalších. Tuto třídu využívá většina funkcí pro zpracování obrazu.

Pro seznámení se s knihovnou bylo potřeba nastudovat především oficiální dokumentaci[21]. Bylo využito také dvou knih[8][5], které popisují různé užitečné tipy a triky knihovny OpenCV.

3.4 Knihovna Tesseract pro rozpoznání textu

Tesseract OCR[22] je považován za jednu z nejpřesnějších knihoven pro rozpoznání textu, které jsou volně dostupné. Byla vyvíjena v letech 1985-1995 společností Hewlett Packard. Od roku 2006 se na jejím vývoji podílí společnost Google.

Zpočátku knihovna podporovala jenom anglický jazyk, od verze 2 se počet podporovaných jazyků zvyšuje a ve verzi 3 již podporuje i češtinu. Pro podporu dalších jazyků či vlastních fontů je nutné Tesseract tyto znaky naučit[23].

Knihovna je dostupná pro operační systémy GNU/Linux, OS X a Windows. Pro použití s Androidem je možné použít knihovnu Tesseract Tools for Android[24], která poskytuje Android API pro Tesseract OCR.

[20] http://docs.opencv.org/modules/core/doc/basic_structures.html#mat
[21] http://docs.opencv.org/modules/refman.html
[22] https://code.google.com/p/tesseract-ocr/
[23] https://code.google.com/p/tesseract-ocr/wiki/TrainingTesseract3
[24] https://code.google.com/p/tesseract-android-tools/

Kapitola 4

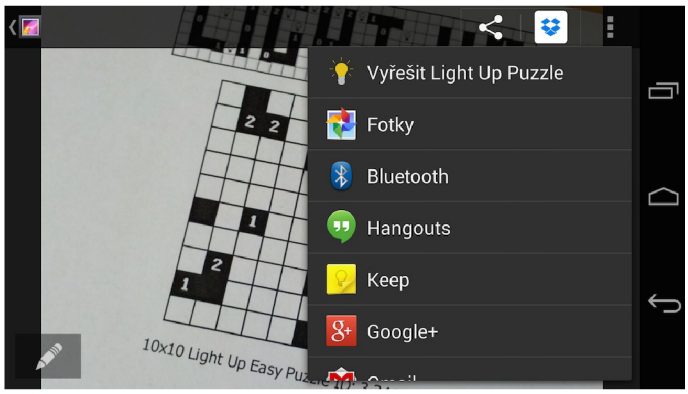
Návrh

V této kapitole bude rozebrán návrh aplikace plnící všechny požadavky plynoucí ze zadání. Pro návrh samotné aplikace bylo potřeba nastudovat především programování na platformě Android a také techniky pro zpracování obrazu, jež byly popsány v předchozích kapitolách. Některé způsoby zpracování obrazu jsou popsány i v knize[2], která se zabývá přímo implementací těchto algoritmů v jazyce Java.

4.1 Návrh uživatelského rozhraní

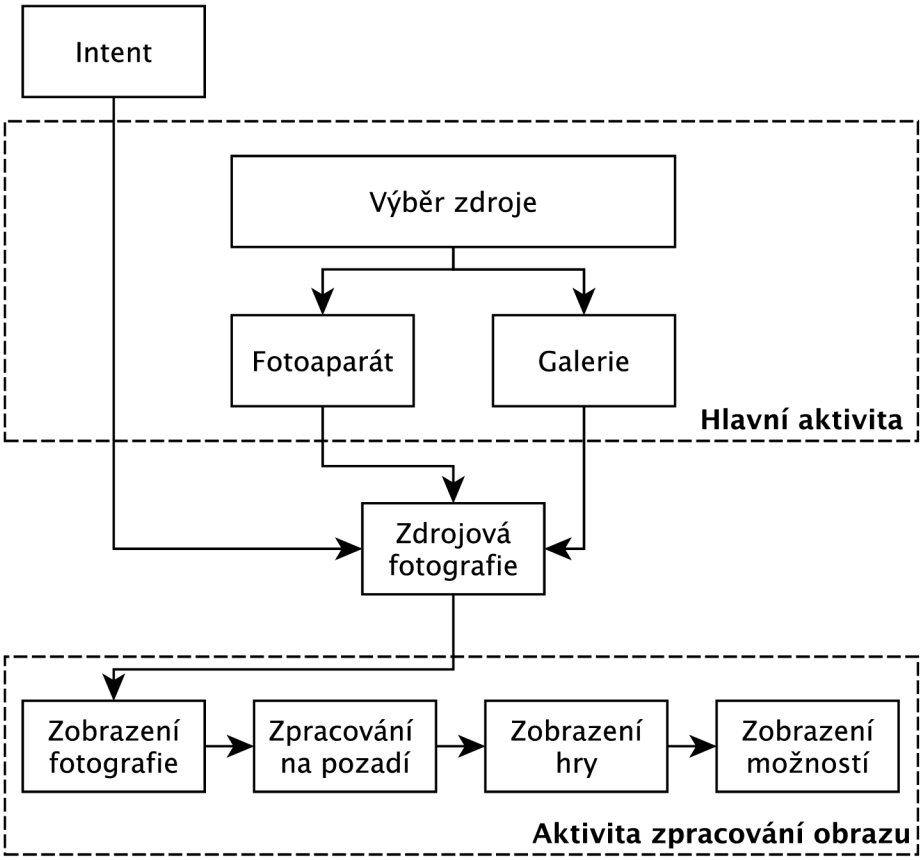
Uživatelské rozhraní aplikace by mělo být, co nejjednodušší. V kapitole 3.2 byly popsány aplikace, které mají podobnou tématiku. Některé aplikace disponovaly příliš komplikovaným rozhraním. I proto bude mít popisovaná aplikace jen ty nejnutnější uživatelské volby.

Aplikace nebude provádět operace v reálném čase v rozšířené realitě. Bude zpracovávat jen jeden uživatelem vybraný snímek. Proto aplikace musí obsahovat minimálně tlačítka pro vytvoření nové fotografie a taky pro výběr fotografie uložené v telefonu. Vhodné je také využít Android komponenty Intent, která byla krátce zmíněna v kapitole 3.1. U aplikace je nutné zaregistrovat Intent filtr na obrázky. Aplikace pak bude moci získat obrázek z jiné aplikace, která umožňuje zasílání obrázků komponentou Intent. Ukázka sdílení obrázku ze systémové galerie do různých aplikací přijímajících toto volání je na obrázku 4.1. V aplikaci budou dvě hlavní aktivity. Jedna pro zmíněný výběr zdroje fotografie, druhá se bude zabývat již samotným rozpoznáním hry. Pro zasílání dat z jedné aktivity do druhé se běžně užívá komponenty Intent.



Obrázek 4.1: Sdílení pomocí komponenty Intent

Uživatelské rozhraní druhé aktivity musí opět obsahovat jen to nejdůležitější. Bude tedy disponovat rozhraním, které zobrazí vybraný obrázek. Na pozadí bude mezitím obrázek zpracováván. Uživatel musí být o této činnosti nějakým způsobem informován, aby nedošlo k mylné domněnce, že je aplikace nečinná. Po rozpoznání dojde k zobrazení načtené hry. Poté bude uživateli nabídnuta možnost editace hry. Té využije, pokud došlo ke špatné detekci některého z políček. Jinak může také stisknout tlačítko pro vyřešení hry. Diagram návrh práce dvou aktivit je znázorněn na obrázku 4.2.

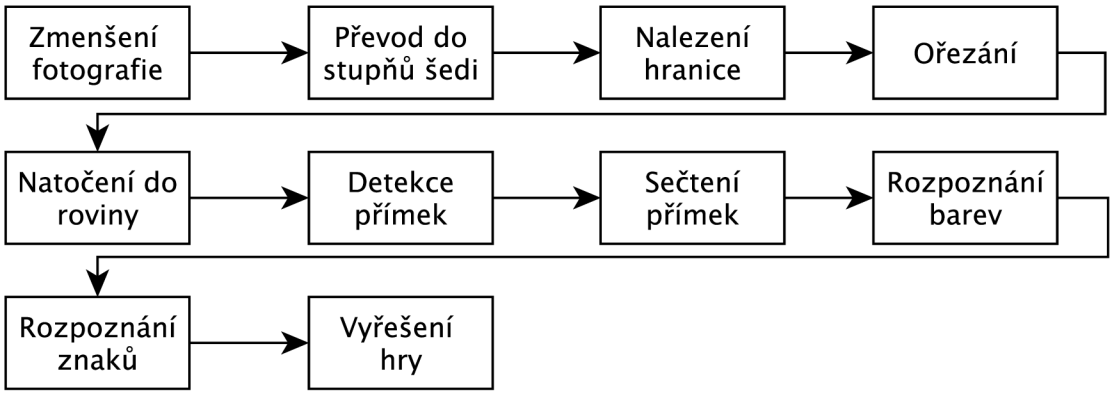


Obrázek 4.2: Diagram návrhu rozhraní

4.2 Rozpoznání hry a její řešení

Dnešní mobilní telefony nezřídka disponují snímači produkujícími fotografie v rozlišení 8 MPx a vyšší. Taková fotografie je pak pro zpracování telefonem příliš velká a časově náročná. Proto bude nutné jako první z kroků provést zmenšení vstupní fotografie. Pro další zpracování nepotřebujeme barevné informace fotografie. Je tedy vhodné převést zmenšenou fotografii do šedotónového modelu. Tento model vyžadují i některé následné algoritmy pro zpracování obrazu. Tím dojde také k dalšímu zmenšení velikosti fotografie. Následným krokem bude nalezení hranice hry. Před provedením dalších kroků je vhodné udělat ořez obrazu podle nalezené hranice a také natočení do roviny. Zadání hry Light Up může disponovat různými počty řádků i sloupců. Bude tedy nutné detekovat jednotlivé přímky v obraze.

Sečtením počtu horizontálních a vertikálních přímek je možné určit počet řádků a sloupců hry. Po detekci počtu řádků a sloupců budeme moci rozdělit hru na jednotlivá hrací políčka. Musí ještě následovat rozpoznání, zda je políčko víceméně černé či bílé. V případě, že je detekována černá barva, musí dojít ještě k analýze políčka, zda neobsahuje číslo. Závěrečným krokem musí být už jen samotné vyřešení rozpoznané hry. Diagram s posloupností jednotlivých kroků je znázorněn na obrázku 4.3.



Obrázek 4.3: Diagram návrhu rozpoznání hry

Kapitola 5

Implementace a testování

Tato kapitola se zabývá implementací aplikace, která byla navržena v předchozí kapitole. Bude popsána struktura aplikace, uživatelské rozhraní a také budou popsány použité algoritmy pro detekci hry.

V další částí této kapitoly bude popsáno testování aplikace.

Vývojové prostředky

Jak již vyplývá ze zadání i názvu práce, aplikace musí být implementována pro platformu Android. Proto je jako hlavní jazyk použit jazyk Java. Některé knihovny jsou však implementovány i částečně v jazycích C/C++. Uživatelské rozhraní je popsáno pomocí XML souborů. Pro vývoj na platformě Android je použito vývojové prostředí IntelliJ IDEA od společnosti JetBrains.

Každá aplikace musí mít v souboru `AndroidManifest.xml` uvedeno několik informací týkajících se názvu aplikace, použitých oprávnění, obsažených aktivit, podporovaných verzí systému apod. Právě podporovaná verze určuje, kterou nejnižší verzi API může programátor použít. Pro implementaci mé aplikace byla použita jako nejnižší podporovaná verze API 14, tedy Android 4.0, který byl představen v říjnu 2011. Dle, v době psaní práce aktuálního, procentuálního zastoupení[1] verzí má starší verzi než 4.0 méně než 20 % uživatelů. I z tohoto důvodu byla zvolena tato verze.

Testování aplikace probíhalo nejčastěji pomocí emulátoru Genymotion. Bylo však využito i skutečných zařízení, která budou podrobněji popsána v jedné z dalších podkapitol.

Struktura aplikace

Aplikace obsahuje dvě aktivity. První je `MainActivity`, která je spuštěna po kliknutí na ikonu aplikace a zajišťuje jen menu aplikace. Druhou aktivitou je `RecognizeGameActivity`, která má na starosti volání jednotlivých funkcí pro zpracování obrazu a zobrazení výsledku. Uživatelské rozhraní bude popsáno v jedné z dalších podkapitol.

Aplikace má pracovní název `Logic Puzzle Solver` a označení balíku celé aplikace je `cz.durbanek.lightup`. Nejdůležitějším balíkem je balík `recognize`, který obsahuje třídy pro jednotlivé části práce s obrazem. Tento balík bude podrobněji popsán v následující podkapitole. Důležitou třídou je `GameView` balíku `gui`, který obsahuje rozšíření standardní systémové třídy `View`. Tato třída se stará o vykreslení rozpoznané hry. Neméně důležitým balíkem je `solver`, zejména pak jeho třída `AkariSolver`. Ta obsahuje kód pro vyřešení

[1] http://developer.android.com/about/dashboards/

načtené hry. Hrací plocha hry je definována v balíku `gameboard`, přesněji jeho třída `Board`. Políčka popisuje třída `Field`. Tyto třídy obsahují především metody pro vytvoření, úpravu a kontrolu hry. Dále pak má aplikace balík `utils`, který obsahuje některé často používané funkce, např. funkce pro práci se soubory či poli.

Hlavní zdrojové soubory v jazyce Java jsou umístěny ve složce `src/`. Soubory popisující grafické rozhraní aplikace, grafické soubory, řetězcové konstanty či spouštěcí animace jsou ve složce `res/`.

5.1 Postup detekce a řešení hry

Implementace vychází z návrhu popsaného v kapitole Návrh. Postup detekce hry je znázorněn na obrázku 5.1. Tato podkapitola se bude zabývat především implementací, která je umístěna v balíku `recognize`.

Aplikace byla navržena tak, aby vyhověla více typům her než jen Light Up. Například pro některé hry nepotřebujeme detekovat počet řádků a sloupců, proto tuto část lze vypnout. Nebo lze také povolit detekci čísel na všech polích a ne jen na černých, jako je tomu u hry Light Up. Pro aplikaci na jinou hru než Light Up bude však nutné manuálně upravit některé hodnoty funkcí pro zpracování obrazu. Je zapotřebí algoritmus pro detekci znaků naučit znaky, které má aplikace detekovat, pokud se nemají detekovat jen znaky 0-4 potřebné pro Light Up. Celý proces rozpoznání hry je řízen ze třídy `Game`. Jednotlivé vlastnosti detekce jsou ovlivněny prvotním voláním konstruktoru.

Nejprve je nutné provést zmenšení fotografie. To zajišťuje standardní systémová třída `BitmapFactory`. Předzpracování fotografie má na starosti třída `Preprocess`. Tato třída volá funkce pro převod obrazu do stupňů šedi (kapitola 2.2), Gaussovské rozostření obrazu (kapitola 2.3) a také adaptivní prahování (kapitola 2.3).

Po úspěšném předzpracování je potřeba najít hranici hry. K tomu slouží algoritmus Border following (kapitola 2.4). Následně je vybrána taková oblast, která je v obraze největší. Na této oblasti dojde k aproximaci na 4 rohové body. Se čtyřmi body lze provést perspektivní transformaci a hru tak narovnat.

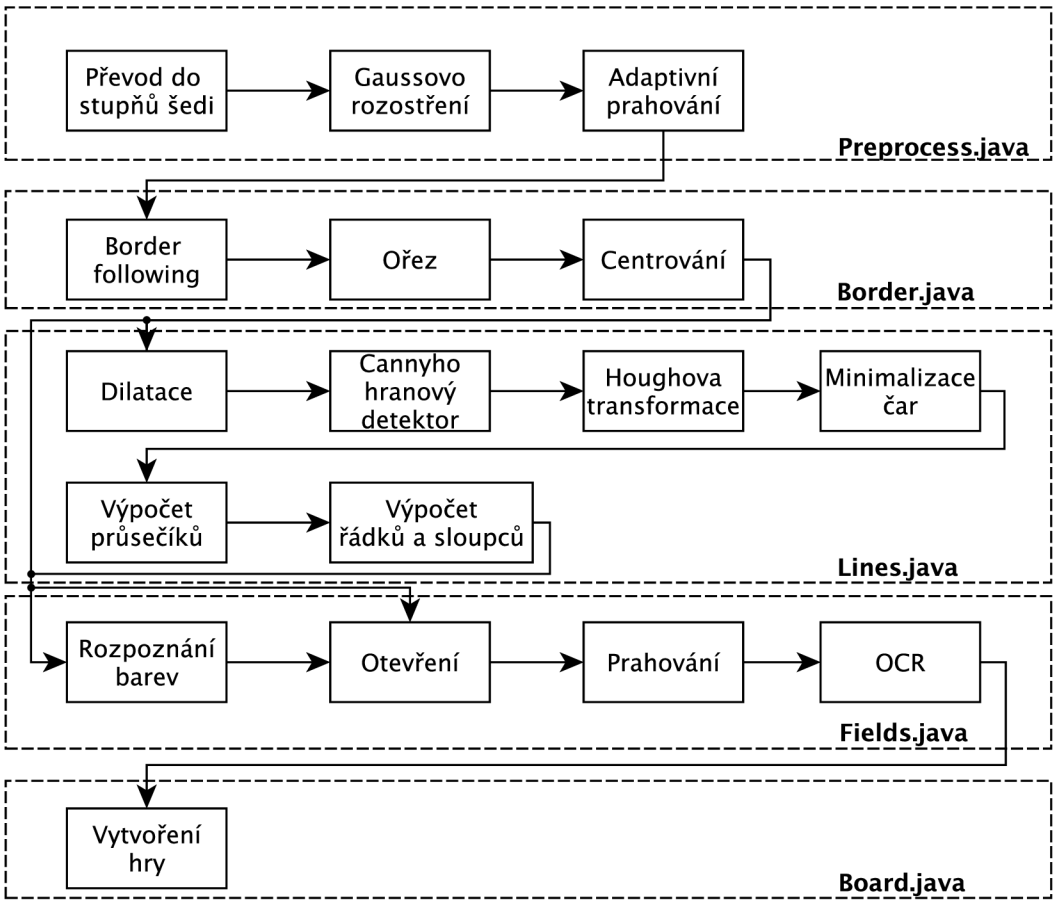
V případě, že byla konstruktorem zapnuta detekce počtu řádků a sloupců, je potřeba zavolat příslušnou metodu třídy `Lines`. Metoda `recognizeRowsCols` nejprve provede dilataci a nalezení hran pomocí Cannyho hranového detektoru (kapitola 2.3). Poté je využito algoritmu Houghovy transformace (kapitola 2.4) pro detekci čar. Často dochází k detekci zdvojených čar, které je tak nutné eliminovat. V dalším kroku jsou vypočítány průsečíky nalezených čar a tím pádem můžeme určit počet řádků a sloupců.

Následuje část pro rozpoznání jednotlivých políček hry. Tato část je implementována v třídě `Fields`. Nejdříve jsou detekovány barvy políček, pokud byla tato možnost aktivována. Každé políčko projde prahováním a poté je spočten počet černých bodů. Pokud je počet černých bodů políčka vyšší než nastavená hranice, je políčko označeno jako tmavé. V další části je spuštěna detekce znaků na políčku. Buď se detekuje na všech polích, nebo jen na polích určité barvy. Druhá varianta je v případě detekce hry Light Up. Před samotnou detekcí je každé pole upraveno erozí (kapitola 2.3) následovanou dilatací a prahováním. Pro detekci znaků je použit algoritmus K-means (kapitola 2.4), jehož implementace bude blíže popsána v další části. V podkapitole testování bude taky provedeno srovnání s knihovnou Tesseract (kapitola 3.4).

V posledním kroku už je jen vytvořena hra v reprezentaci balíku `gameboard`. Uživateli je dále zobrazena detekovaná hra a je mu nabízena možnost vyřešení hry.

Pro řešení hry je použit algoritmus zpětného navracení Backtracking[15]. Tento algoritmus funguje na principu prohledávání stavového stromu zadaného problému. Algoritmus prohledává stavový strom do hloubky. Využívá se zde zásobníku, do kterého postupně ukládáme jednotlivé stavy. Nejprve umístíme počáteční uzel. V dalším kroku generujeme jednoho bezprostředního následníka. Jestliže je následník platným stavem, tak pokračujeme generováním dalšího následníka. Pokud je však následník neplatný, tak se zpětně navracíme do posledního platného stavu a generujeme dalšího následníka, je-li takový k dispozici. V případě, že již není další následník a poslední uzel je označen jako neplatný, končí algoritmus neúspěchem. Vždy před generováním dalšího následníka musí dojít k ověřování, zda není uzel na vrcholu zásobníku cílovým uzlem. Algoritmus končí, pokud je nalezena cesta k cílovému stavu, nebo pokud byly všechny uzly označeny jako neplatné.

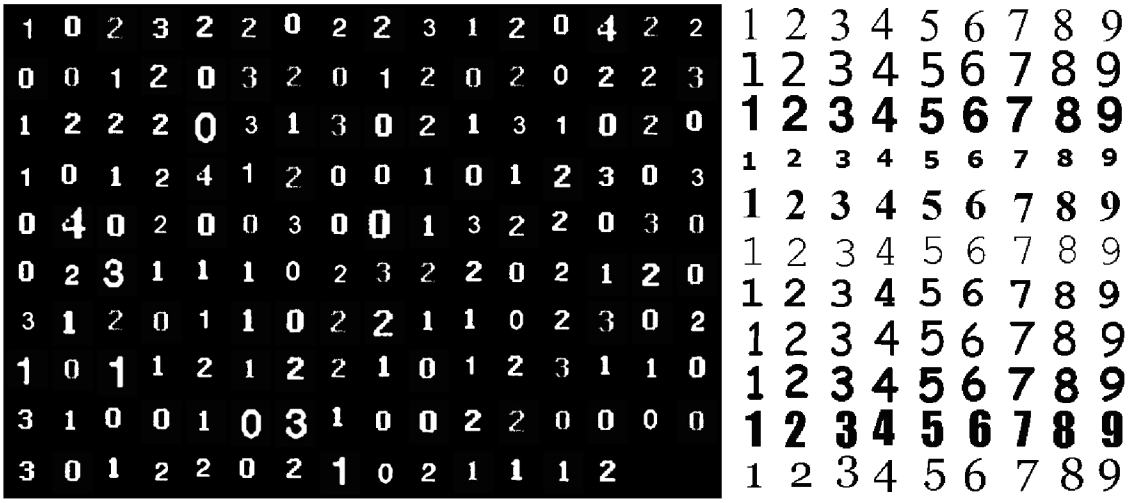
Algoritmus Backtracking má exponenciální časovou náročnost. Paměťová náročnost je nízká, protože se v zásobníku ukládá jen aktuální cesta k uzlu. Backtracking je nejvíce časově náročný, když je problém neřešitelný. V takovém případě totiž musí projít všechny možné kombinace.



Obrázek 5.1: Diagram rozpoznání hry

5.1.1 K-means

Tento algoritmus je použit v této práci k rozpoznání čísel hry. Nejdříve je potřeba algoritmus naučit tvary jednotlivých znaků. Je nutné vytvořit trénovací vzorek s používanými znaky. Pro lepší detekci je vhodné zahrnout i různé typy písma či chybně vyfocené znaky. Ukázka dvou trénovacích vzorků je vidět na obrázcích 5.2. Pro natrénování znaků trénovacího vzorku je potřeba uložit všechny body popisující znak a přiřadit k němu příslušnou hodnotu znaku. Toto se opakuje pro každý znak trénovacího vzorku. Výsledkem trénování jsou na výstupu 2 soubory. Jeden popisuje tvary všech znaků, druhý popisuje hodnoty znaků.



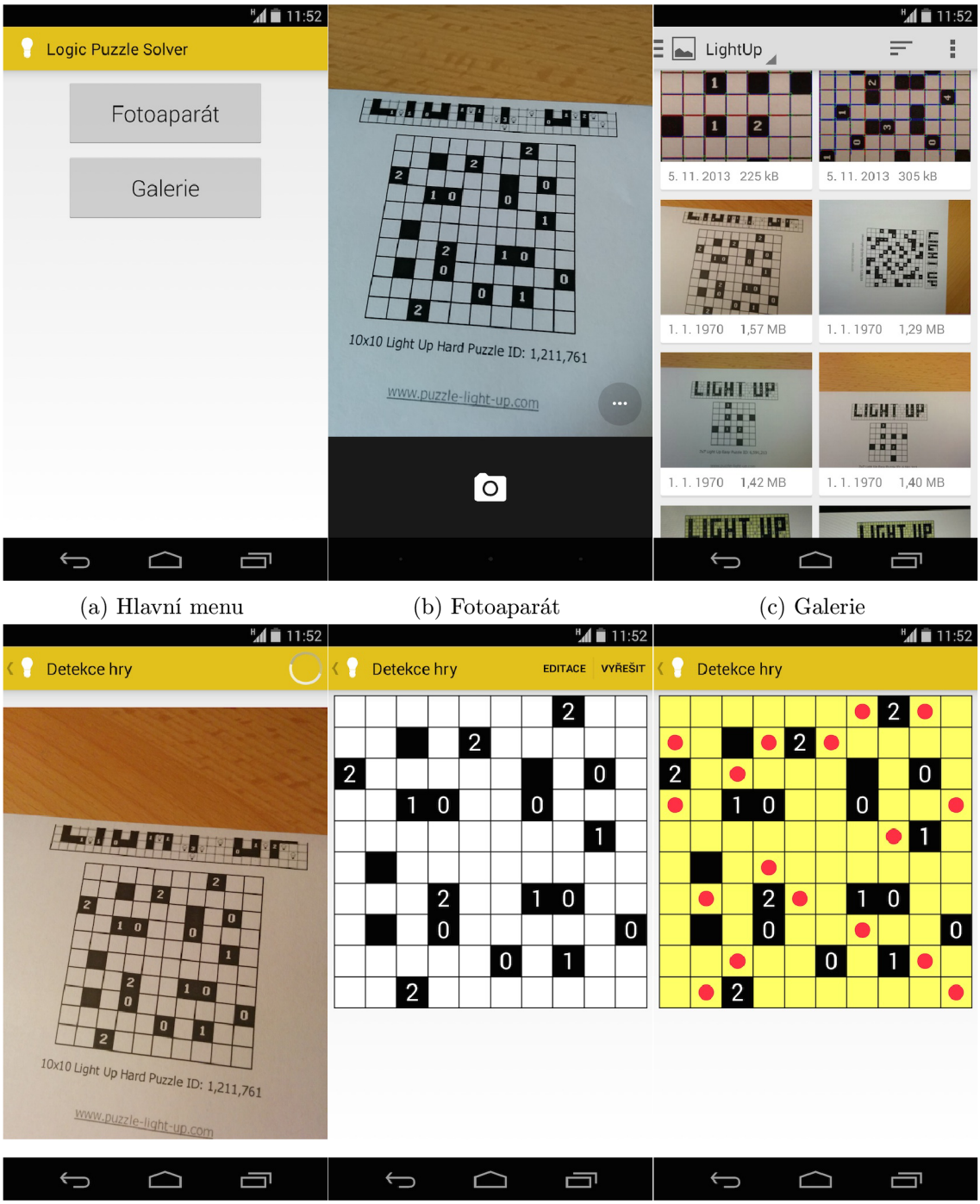
Obrázek 5.2: Trénovací soubory

Knihovna OpenCV obsahuje klasifikátor umožňující natrénování znaků za pomocí vytvořených souborů. Pro nalezení čísla je použita podobná technika jako u trénování. Jsou vybrány všechny body, které popisují jeden znak a poté je zavolána funkce knihovny OpenCV pro nalezení nejbližší shody.

5.2 Uživatelské rozhraní aplikace

Uživatelské rozhraní tvoří dvě aktivity. První aktivita je nejjednodušší. Stará se jen o obsluhu dvou tlačítek. Buď uživatel vybere fotografii z již uložených nebo vytvoří fotografii novou. Základní aktivitu s menu a reakcemi na klik lze vidět na prvním řádku série obrázků 5.3. Popis rozhraní této aktivity je v souboru `main.xml`. Pro výběr fotografie z již uložených i k vytvoření nové používám standardní systémové komponenty systému Android.

Po vybrání fotografie nebo vytvoření nové je fotografie předána do další aktivity. V této aktivitě je nejprve zobrazena vybraná fotografie a uživatel je na činnost upozorněn animovanou ikonou v pravém horním rohu. Po zpracování fotografie je uživateli, v případě správné detekce, zobrazena detekovaná hra. Ten má pak na výběr dva kroky. Buď editaci načtené hry nebo zobrazení řešení. Editaci hry využije zejména v případě, kdy došlo k chybné detekci některého z polí. Uživatelské rozhraní v různých fázích zpracování je na druhém řádku série obrázků 5.3. Rozhraní aktivity je definováno v souboru `recognize_game.xml`.



(a) Hlavní menu (b) Fotoaparát (c) Galerie

(d) Průběh zpracování (e) Detekovaná hra (f) Vyřešená hra

Obrázek 5.3: Uživatelské rozhraní aplikace

5.3 Testování

Důležitou části vývoje aplikace je nepochybně testování. Testování by mělo probíhat v průběhu vývoje, ale i na samotném konci. V průběhu vývoje jsem testoval především přesnost detekce hry. Ke konci vývoje jsem se pak zaměřil na optimalizaci celého procesu rozpoznání

hry. Poté bylo potřeba aplikaci znovu důkladně otestovat. V rámci testování jsem také porovnal dva navržené způsoby detekce znaků.

Použité zařízení

Aplikace byla testována na zařízeních uvedených v tabulce 5.1. Výběr zahrnuje zařízení od různých výrobců, různě velkým výpočetním výkonem a rozdílnými verzemi systému.

Název zařízení	Verze systému	Procesor	Paměť
Asus Nexus 7 (2012)	4.1	4x 1,5 GHz ARMv7	1 GB
LG Nexus 4	4.4	4x 1,512 GHz ARMv7	2 GB
LG Nexus 5	4.4	4x 2,26 GHz ARMv7	2 GB
Samsung Nexus S	4.1	1x 1 GHz ARMv7	512 MB
Samsung Galaxy Nexus	4.3	2x 1,2 GHz ARMv7	1 GB
Sony Xperia Sola	4.1	2x 1 GHz ARMv7	512 MB
Emulátor Genymotion	4.2	1x 2,3 GHz Intel	512 MB

Tabulka 5.1: Tabulka zařízení

Testování správnosti rozpoznání hry

V této části testování jsem testoval zejména přesnost detekce. Všechny zadání hry jsem vytiskl ze stránky Light Up Puzzle[2]. Na této stránce se vyskytují zadání her různých obtížností o velikostech 7x7, 10x10, 14x14 a 25x25. Takto vytištěné puzzle jsem následně mobilním telefonem vyfotil a spustil detekci. Při horších světelných podmínkách nebyla hra detekována vůbec či některá čísla byla detekována nesprávně. Pro druhý výsledek je v aplikaci implementována možnost editace rozpoznané hry. Při dobrých světelných podmínkách detekce probíhá v pořádku. Problém samozřejmě může být se chybně zaostřenými fotografiemi nebo s fotografiemi, které mají příliš mnoho šumu. Tento výsledek je však očekávaný.

Bylo zjištěno, že aplikaci dělají problém hry o velikosti 25x25. U hry o této velikosti jsou jednotlivá políčka na fotografii příliš malá. Dochází zde především k chybné detekci počtu řádků a sloupců. Velikost 14x14 je detekována bez problému. Žádné zadání mezi těmito velikostmi není na stránce dostupné. Na další stránce[3] zabývající se touto hrou jsem nalezl další zadání. Zde se vyskytují i zadání mezi velikostmi 14x14 a 25x25. Vyzkoušel jsem proto dostupné velikosti 15x15 a 20x20. V obou případech došlo k bezproblémové detekci. Detekovaná hra je však při zobrazení na testovacím zařízení, které má úhlopříčku displeje 4,95", hodně malá.

Testování rychlosti

Testování rychlosti probíhalo na 30 fotografiích různé kvality a různé velikosti hry Light Up. Nejmenší velikost hry byla 5x5 a největší pak 20x20. Naměřené časy jsou v tabulce 5.2.

Při podrobnějším zkoumání dílčích výsledků detekce, bylo zjištěno, že nejdéle trvá Houghova transformace. V případě, že by nebyla zapnuta detekce čar, by byl výsledek znám v průměru o polovinu času dříve. Tento algoritmus je implementován v knihovně OpenCV.

[2] http://www.puzzle-light-up.com/
[3] http://www.brainbashers.com/lightup.asp

Název zařízení	Průměrný čas	Minimální čas	Maximální čas
Asus Nexus 7 (2012)	3,22 s	2,25 s	4,18 s
LG Nexus 4	3,29 s	2,25 s	4,83 s
LG Nexus 5	1,84 s	1,25 s	2,48 s
Samsung Galaxy Nexus	4,24 s	2,58 s	20,55 s
Samsung Nexus S	26,35 s	14,74 s	36,12 s
Sony Xperia Sola	6,68 s	5,46 s	8,84 s
Emulátor Genymotion	2,08 s	1,69 s	2,44 s

Tabulka 5.2: Testování rychlosti detekce

Proto není možno tento krok zoptimalizovat. Bylo by nutné hledat jiný způsob detekce čar v obraze.

Na výsledcích je vidět podstatné zrychlení aplikace u zařízení, které disponují větším počtem jader.

Čas řešení hry se pohybuje v setinách až desetinách sekundy u zadání do velikosti 14x14. U her o velikosti 14x14 a vyšší obtížnosti zadání dochází však již k značnému zpomalení řešení a to i o desítky sekund. V práci je použit algoritmus backtracking. Pro zrychlení řešení by tedy bylo nutné použít vhodnější algoritmus.

Testování způsobu detekce znaků

V této části je popsáno testování dvou způsobů detekce znaků hry. Jedním ze způsobů je použití knihovny Tesseract, která byla blíže popsána v kapitole 3.4. Druhým způsobem je detekce pomocí algoritmu pro shlukování, K-means. Oba algoritmy jsem naučil sadu znaků, která čítá několik set výskytů všech znaků hry. Každý znak je zde v různých variantách, které zahrnují různé typy písma či výskyty znaků s poškozením. Tyto naučené znaky jsou uloženy v souborech aplikace. U knihovny Tesseract má takový soubor velikost cca 150 kB. U algoritmu K-means je soubor zhruba čtvrtinový, velikost je 40 kB. Samotná knihovna Tesseract zabírá přes 10 MB. Algoritmus K-means je implementován v knihovně OpenCV. Nezabírá tedy další místo. Výsledky měření na 20 fotografiích je v tabulce 5.3. Testování probíhalo na mobilním telefonu LG Nexus 5.

Způsob detekce	Přesnost	Průměrný čas	Minimální čas	Maximální čas
Tesseract	84 %	0,12 s	0,036 s	0,226 s
K-means	98 %	0,05 s	0,02 s	0,146 s

Tabulka 5.3: Výsledky testování detekce čísel 0-4

Z testování je vidět, že použití algoritmu K-means je při hře Light Up přesnější. Detekce znaků zabere také v průměru menší čas. Dalším kladem, oproti použití knihovny Tesseract, je snížení velikosti aplikace o více než 10 MB.

Další částí testování detekce znaků bylo testování na hře Sudoku, kdy potřebujeme detekovat čísla v rozsahu 1-9. Oba způsoby detekce jsem tedy naučil stejnou sadu bezmála 500 znaků různých variant čísel 1-9. Předzpracování detekce nebylo nijak upravováno oproti detekci hry Light Up. Algoritmus K-means dopadl srovnatelně s knihovnou Tesseract. K-means vykazoval výsledky okolo 59 % správných detekcí čísel. Avšak mnoho znaků bylo detekováno tam, kde bylo prázdné pole. Knihovna Tesseract dokázala správně rozpoznat

cca 65 % čísel. Falešně detekovaná čísla zde byla v menším měřítku než u algoritmu K-means.

Důvodem chybné detekce u Sudoku může být fakt, že jednotlivé hodnoty předzpracování obrazu nebyly upraveny. K lepší detekci by přispělo také vybrání jiného vzorku znaků pro naučení. U algoritmu K-means také častěji docházelo k záměně některých tvarově blízkých znaků. K-means tedy bude vhodnější pro hry, kde se nevyskytuje velký počet znaků.

Pro tuto práci je tedy použit algoritmus K-means, avšak aplikace je připravena i na nasazení knihovny Tesseract.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat aplikaci, která nalezne na vstupní fotografii hru Light Up a následně ji vyřeší. Tato aplikace byla implementována na mobilní platformě Android. Cílem tedy bylo i seznámení se s programováním na této platformě. Tyto cíle byly naplněny.

Pro návrh aplikace bylo potřeba nastudovat příslušnou literaturu. Tato literatura zahrnovala především knihy zabývající se zpracováním obrazu. Nastudoval jsem také literaturu o programování na platformě Android. Většinu znalostí jsem však získal z oficiálních stránek platformy.

Zpracováním tohoto tématu jsem si osvojil vývoj aplikací pro mobilní platformu Android a také některé důležité algoritmy pro zpracování obrazu. Zdokonalil jsem se v programování v jazyce Java a také jsem pronikl do některých méně používaných technik. Odborně mě obohatila také práce s použitou knihovnou pro zpracování obrazu - OpenCV.

V rámci testování jsem aplikaci odzkoušel na několika zařízeních. Testy zahrnovaly testování rychlosti rozpoznání, správnosti rozpoznání a další. Za dobrých světelných podmínek dochází ke správné detekci hry. Test rychlosti ukázal některá slabší místa aplikace. Především se jedná o detekci čar, která je potřebná pro určení počtu řádků a sloupců. Tento algoritmus je však implementován přímo v knihovně OpenCV.

Aplikaci jsem také dal k vyzkoušení několika osobám a získal jsem tak hodnotnou zpětnou vazbu. Uživatelům se aplikace převážně líbila. Výhrady měli především k neznámosti logické hry Light Up. Jeden z uživatelů měl také připomínku, že by aplikace mohla dokázat zobrazit posloupnost jednotlivých kroků zpracování obrazu. Celkově uživatelé aplikaci hodnotili jako dobrou, ale vzhledem k použité hře neviděli praktické využití aplikace. V případě použití jiné hry (např. Sudoku) by si již dovedli představit, že by takovou aplikace někdy i použili.

Dalším pokračováním projektu by mohlo být rozšíření na další hry než jen Light Up. Aplikace je na toto rozšíření částečně připravena. Pro každou hru je však nutné manuálně nastavit jednotlivé hodnoty rozpoznání a případně také naučit rozpoznávat jiné znaky než jen 0-4. Aplikace by se dala také vylepšit přidáním možnosti zahrát si rozpoznanou hru. Zajímavým rozšířením by mohlo být upravení aplikace tak, aby detekce probíhala v reálném čase a řešení se zobrazovalo přímo do výstupního snímku fotoaparátu.

Téma i průběh práce včetně získaných závěrů byly pro mě velmi přínosné. Vývoj mobilních aplikací považuji za zajímavou dynamicky rozvíjející a do budoucna perspektivní část odvětví informačních technologií.

Literatura

[1] Burger, W.; Burge, M.: *Principles of digital image processing: Core Algorithms*. Springer, 2009, ISBN 18-480-0190-8, 327 s.

[2] Burger, W.; Burge, M. J.: *Digital image processing: an algorithmic introduction using Java*. Springer, první vydání, 2008, ISBN 9781846283796, 564 s.

[3] C, H.: Method and means for recognizing complex patterns. Prosinec 18 1962, uS Patent 3,069,654.

[4] Canny, J.: *A Computational Approach to Edge Detection*. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 1986, 679-698 s.

[5] Daniel Lelis Baggio, S. E.: *Mastering OpenCV with practical computer vision projects*. Packt Pub, první vydání, 2011, ISBN 978-184-9517-829, 287 s.

[6] Duda, R. O.; Hart, P. E.: Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Commun. ACM*, ročník 15, č. 1, Leden 1972: s. 11–15, ISSN 0001-0782, doi:10.1145/361237.361242.

[7] Gonzalez, R. C.; Woods, R. E.; Eddins, S. L.: *Digital Image processing using MATLAB*. Gatesmark Publishing, druhé vydání, 2009, ISBN 978-0982085400, 826 s.

[8] Laganiere, R.: *OpenCV 2 computer vision application programming cookbook*. Packt Publishing, první vydání, 2011, ISBN 978-1-84951-324-1, 287 s.

[9] MacQueen, J. B.: Some Methods for Classification and Analysis of MultiVariate Observations. In *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, ročník 1, editace L. M. L. Cam; J. Neyman, University of California Press, 1967, s. 281–297.

[10] Mark S Nixon, A. S. A.: *Feature extraction and image processing*. Academic Press, druhé vydání, 2008, ISBN 978-0-12372-538-7, 406 s.

[11] Murphy, M. L.: *Android 2*. Computer Press, vyd. 1. vydání, 2011, ISBN 9788025131947, 375 s.

[12] Poynton, C.: *Digital video and HDTV*. Morgan Kaufmann Publishers, 2003, ISBN 15-586-0792-7, 692 s.

[13] Rosenfeld, A.: Connectivity in Digital Pictures. *J. ACM*, ročník 17, č. 1, Leden 1970: s. 146–160, ISSN 0004-5411.

[14] Shapiro, L.; Stockman, G.: *Computer vision*. Prentice-Hall, první vydání, 2001, ISBN 01-303-0796-3, 580 s.

[15] Skiena, S. S.: *The algorithm design manual*. London: Springer, druhé vydání, 2010, ISBN 978-1849967204.

[16] Suzuki, S.; Abe, K.: Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, ročník 30, č. 1, 1985: s. 32–46.

[17] Ujbanyai, M.: *Programujeme pro Android*. Grada, vyd. 1. vydání, 2012.