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 SUDOKU PRO ANDROID

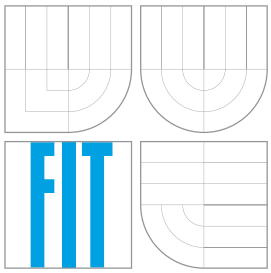
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH HRBAS

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 SUDOKU PRO ANDROID

SUDOKU SOLVER FOR ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VOJTĚCH HRBAS
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEXANDER PÁLDY
SUPERVISOR

BRNO 2014

Abstrakt

Tato práce se zabývá řešením hry Sudoku pořízené pomocí kamery mobilního zařízení se systémem Android. Probírá možnosti zpracování obrazu, možnosti rozpoznávání textu v obraze a princip a řešení hry Sudoku. Zkoumá také již existující aplikace pro Android řešící Sudoku. Dále navrhuje vlastní aplikaci pro řešení Sudoku a shrnuje dosažené výsledky testování aplikace z hlediska výkonu a uživatelů.

Abstract

This work deals with solving Sudoku game which is taken by a camera of a mobile device running Android. It discusses possibilities of image processing, possibilities of recognizing the text in the image and principle and solving of Sudoku game. It also examines existing applications for Android that solve Sudoku. Then it proposes the application itself for solving Sudoku and summarizes the results of testing the application in terms of performance and users.

Klíčová slova

Android, Sudoku, řešitel Sudoku, zpracování obrazu, rozpoznávání textu, OCR.

Keywords

Android, Sudoku, Sudoku solver, image processing, text recognition, OCR.

Citace

Vojtěch Hrbas: Řešitel sudoku pro Android, bakalářská práce, Brno, FIT VUT v Brně, 2014

Řešitel sudoku pro Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Alexandera Páldyho.

.....
Vojtěch Hrbas
16. května 2014

Poděkování

Rád bych poděkoval svému vedoucímu panu Ing. Alexanderovi Páldymu za poskytování nápadů, připomínek a rad, které přispívaly ke zlepšování práce. Dále chci poděkovat svým rodičům, bez nichž by mi studium nebylo umožněno a také své přítelkyni Gabriele, jejíž motivace mi pomáhala v práci pokračovat. V neposlední řadě bych rád poděkoval všem, kteří se zúčastnili testování aplikace a poskytli mi tak cennou zpětnou vazbu.

Obsah

Kapitola 1

Úvod

Cílem této práce je poskytnout jednoduchou pomůcku pro kontrolu řešení hry Sudoku. Pokud se Sudoku nachází tištěné například v časopise, jeho řešení bývá často uvedené na jiných stránkách nebo ve zmenšené podobě a ještě vzhůru nohama. Občas lze také přijít k zadání, jehož řešení není k dispozici vůbec.

V takovýchto případech je výhodné mít možnost si řešení zkontrolovat jiným způsobem. Ovšem počítač, který se nabízí jako jedna varianta, nemusí být vždy po ruce. Oproti tomu jsou stále rozšířenější mobilní zařízení s operačními systémy. Je potom docela snadné vzít toto zařízení a Sudoku do něj přepsat. Takových aplikací již existují spousty a stále lze jít dále. Většina těchto mobilních zařízení totiž obsahuje také kameru. Nabízí se proto možnost Sudoku jednoduše nasnímat kamerou a tím přeskočit i nutnost zdlouhavého ručního přepisování celého zadání.

Podobných aplikací již není mnoho a lze je tak rozšířit další variantou přístupu. Velká část mobilních zařízení běží na systému Android. Tato práce se věnuje návrhu a vytvoření aplikace umožňující pořízení snímku Sudoku a jeho vyřešení kamerou mobilního zařízení právě se systémem Android.

V kapitole 2 je shrnuta veškerá problematika spojená s teorií nutnou pro úspěšný návrh aplikace. Tato kapitola shrnuje několik různých oblastí a to samotnou platformu Android, dále zpracování obrazu z kamery, které je nutné pro zjištění, kde se v obraze nachází mřížka Sudoku. Jelikož ta obsahuje číslice, je probírána také problematika rozpoznávání znaků v obraze. Následně je nutné vyřešit samotnou hru Sudoku, proto se tato kapitola zabývá také jejími pravidly a technikami řešení. Poslední část této kapitoly probírá některé již existující aplikace pro řešení Sudoku pomocí kamery.

Kapitola 3 se zabývá návrhem vlastní aplikace a to jak jejího vzhledu tak i funkčnosti a návrhem rozdělení částí řešení na jednotlivé moduly. Implementace a popis výsledné aplikace se nachází v kapitole 4. V poslední kapitole 5 je popsáno testování aplikace z hlediska výkonu a uživatelů, dále jsou shrnuty dosažené výsledky a jejich význam.

Kapitola 2

Problematika řešení hry Sudoku na platformě Android

Vyřešit Sudoku pořízené pomocí kamery mobilního zařízení zahrnuje několik významných oblastí, jejichž problematika je řešena v této kapitole. Je nutné pracovat s mobilním zařízením na platformě Android, tím se zabývá část 2.1. Dále je potřeba zpracovat obraz pořízený kamerou, to je vysvětleno v části 2.2. Rozpoznáním číslic v obraze se zabývá část 2.3. Samotná hra Sudoku je řešena v části 2.4. Již existující aplikace a jejich přístup k řešení Sudoku jsou probírány v části 2.5.

2.1 Platforma Android

Tato část řeší pouze několik detailů systému Android, které souvisí s vývojem aplikace využívající kameru zařízení.

Celá architektura systému Android je složená z několika komplexních vrstev, jimiž jsou *linuxové jádro*, *nativní knihovny*, *běhové prostředí*, *aplikační framework* a *aplikace a widgety*, jak lze najít v knize [3]. Pro vývoj aplikací je nejdůležitější vrstvou aplikační framework a jemu se tato část věnuje.

2.1.1 Aplikační framework

Popis této části vychází z knihy [1]. Aplikační framework poskytuje vysokoúrovňové stavební bloky pro tvorbu aplikací, které jsou zde krátce zmíněny. Poté je popsán také přístup k souborům a ke kameře zařízení.

Nejvýznamnějšími bloky jsou *aktivity*, *služby*, *poskytovatelé obsahu* a *záměry*. Aktivity jsou základním blokem uživatelského rozhraní a proto se jim více věnuje další část. Poskytovatelé obsahu jsou také významným blokem, protože umožňují přístup například právě k souborovému systému, jehož obsluha je také zmíněna dále.

Aktivity

Systém Android je z velké části instalován na telefonech, které mají omezené paměťové možnosti. Spouštěním různých aktivit se paměť spotřebovává a občas je nutné ji uvolnit, proto se aktivity řídí určitým životním cyklem.

Samotná aktivita se může nacházet ve čtyřech různých stavech a to jsou stavy *aktivní*, *pozastavená*, *zastavená* a *mrtvá*. Při přepínání mezi jednotlivými stavy dochází k volání

různých metod, v nichž je vhodné vytvářet a uvolňovat jiné prostředky. Například při vytvoření aktivity alokovat paměť pro zobrazení obrázku na obrazovku a při ukončování aktivity ji opět uvolnit. Zabrat nebo uvolnit kameru je však vhodné při přechodech aktivity mezi stavy aktivní nebo pozastavená, protože ji může chtít využít jiná aktivita, která je právě přenesena na popředí.

Systém Android si však vyhrazuje právo kdykoli zabít aplikační proces a tím i běžící nebo pozastavenou aplikaci. Je to z důvodu naléhavého nedostatku paměťových prostředků. Při tom ovšem nemusí být volány metody, které vedou k běžnému ukončení aktivity a nedochází tak ke korektnímu uvolňování paměti a prostředků, které mohlo být implementováno v těchto metodách.

Práce se soubory a komponentami zařízení

Práce se soubory v Androidu probíhá několika různými způsoby. Přístup k souborům přibaleným k aplikaci probíhá pomocí správce zdrojů. Tomu je předán identifikátor, pokud se jedná o strukturované zdroje nebo relativní cesta v rámci obecných přibalených souborů. Takovéto soubory slouží pouze ke čtení.

Dále je možné číst a zapisovat soubory umístěné v interním úložišti zařízení. Každá aplikace má přístup pouze k vlastním souborům a běžně k nim nelze jinak přistupovat. Poslední možností je čtení a zapisování souborů v externím úložišti. K takovým souborům může přistupovat jakákoli aplikace nebo i uživatel sám. Aby bylo možné využít externího úložiště, je nutné si vyžádat patřičné oprávnění.

Pokud má aplikace využívat některé komponenty zařízení, například kameru nebo snímače určující polohu zařízení, je také nutné si vyžádat dané oprávnění. Je možné si vyžádat i přítomnost určité komponenty, například kamery, a aplikaci pak není možné nainstalovat na zařízení nesplňující tuto podmínku. Přístup ke kameře si poté vyžaduje zabrání prostředků a jejich následné uvolnění.

2.2 Zpracování obrazu z kamery

Tato část se zabývá možnostmi zpracování obrazu pro Android, poté řeší základní operace zpracování obrazu a nakonec vyhledávání mřížky Sudoku v upraveném obraze.

2.2.1 Knihovny pro Android

Z oblasti zpracování obrazu a počítačového vidění lze nalézt několik významných knihoven běžících na platformě Android:

- *OpenCV4Android*[1] – je dostupný jako nástroj pro vývoj software a jedná se o Java rozhraní pro knihovnu OpenCV.
- *FastCV*[2] – je knihovna pro počítačové vidění optimalizovaná pro mobilní zařízení. Tato knihovna je speciálně optimalizovaná pro architekturu procesorů Snapdragon, kde dosahuje vyšších výkonů.
- *BoofCV*[3] – je open source Java knihovna pro počítačové vidění a robotiku v reálném čase.

[1] Více o OpenCV4Android na stránkách: http://www.opencv.org/android
[2] Více o FastCV na stránkách: https://developer.qualcomm.com/computer-vision-fastcv
[3] Více o BoofCV na stránkách: http://www.boofcv.org

V tabulce 2.1 je porovnání rychlostí zpracování některých funkcí pro knihovny OpenCV a FastCV. Toto porovnání je vzhledem k samostatné knihovně OpenCV, nejedná se o nástroj OpenCV4Android.

Funkce	**OpenCV**	**FastCV**	**FastCV Snapdragon**
Normalizovaná vzájemná korelace	1.0x	9.0x	23.1x
Skalární součin 128x4	1.0x	4.0x	10.0x
Konverze YUV420	1.0x	1.4x	1.3x
Sobelův operátor	1.0x	1.8x	7.8x
Mediánový filtr 3x3	1.0x	3.8x	51.9x
Gaussův filtr 3x3	1.0x	2.6x	4.1x
Gaussův filtr 5x5	1.0x	1.4x	2.9x
Prahování	1.0x	0.7x	9.7x
Integrální obraz	1.0x	1.1x	1.3x
Harrisova detekce rohů	1.0x	2.8x	8.6x
Dilatace	1.0x	1.4x	15.0x
Eroze	1.0x	1.3x	15.0x
Perspektivní transformace	1.0x	21.5x	37.8x
Optický tok Lucas-Kanade	1.0x	2.0x	14.3x

Tabulka 2.1: Porovnání rychlostí zpracování některých funkcí mezi OpenCV a FastCV. Tabulka je převzatá z prezentace dostupné na webu [4].

Nástroj OpenCV4Android nabízí od verze 2.4.3 servisní aplikaci OpenCV Manager pro Android. Tato aplikace snižuje paměťovou velikost aplikací, které OpenCV využívají, jelikož je instalovaná samostatně a není nutné knihovnu OpenCV přibalovat ke každé aplikaci zvlášť. Toto řešení navíc přináší snadno dostupné aktualizace a umožňuje hardwarové optimalizace podporovaných platforem, jak je uvedeno na webu [7].

2.2.2 Základní úprava obrazu

Pořízený obraz je výhodné upravit do podoby vhodné pro zpracování. Základní úpravy obrazu lze najít v této části.

Vyhlazování

Vyhlazování slouží především k odstranění šumu a artefaktů kamery. Nejjednodušším filtrem je pouhé zprůměrování vstupních bodů pomocí konvoluce každého vstupního bodu obrázku s jádrem (maskou) a poté sumarizací k vyprodukování výstupního bodu. Konvoluční jádro může vypadat například takto:

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Užitečnější je Gaussův filtr, který směrem od středu masky snižuje váhy jednotlivých bodů podle hustoty pravděpodobnosti $g(x)$ Gaussova rozložení, jak uvádí kniha [11]:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Gaussovo jádro je definováno centrované, lze proto vynechat parametr μ. Dvourozměrná Gaussova funkce pak nabývá podoby, kterou uvádí kniha [9]:

$$g(x, y) = ce^{-\frac{x^2+y^2}{2\sigma^2}}$$

kde parametr c určuje měřítko tak, aby součet všech členů Gaussova jádra byl roven 1. Příklad aproximovaného Gaussova jádra je zde:

$$h = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Nevýhodou obou těchto filtrů je, že vyhladí ostré okraje, jak tvrdí kniha [11]. Tuto nevýhodu odstraňuje bilaterální filtr, který neprůměruje vstupní pixely podle vzdálenosti od centrálního bodu masky, ale každému bodu přiřazuje váhu podle rozdílu intenzity světlosti vzhledem k centrálnímu bodu. Pixely s menším rozdílem intenzity mají přiřazenu vyšší váhu než pixely s vyšším rozdílem, jak je uvedeno v knize [2]. Výstupy Gaussova a bilaterálního filtru jsou zobrazeny na obrázku 2.1.

Obrázek 2.1: Vlevo je vstupní obraz, uprostřed výstup Gaussova filtru a vpravo výstup bilaterálního filtru.

Převod obrazu do binární podoby

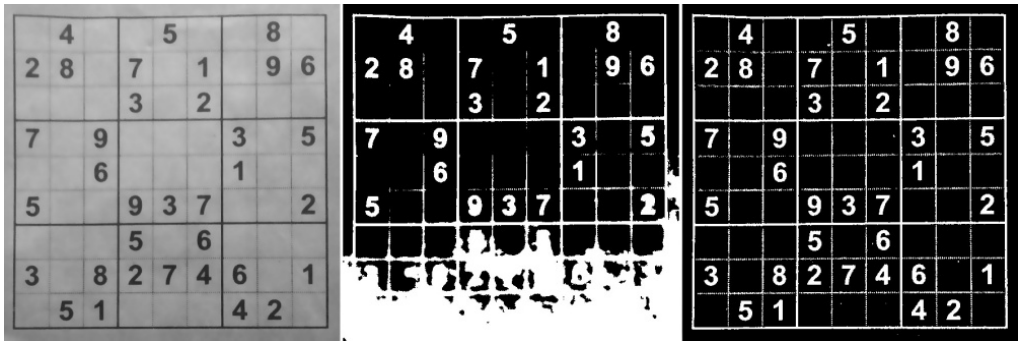
Binární podoba obrazu využívá pouze dvou barev k rozlišení, zda se daný bod v obraze nachází nebo ne. K převodu obrazu do takové podoby lze využít prahování. To spočívá v porovnání každého vstupního bodu s hodnotou prahu.

Obecně lze prahování popsat touto rovnicí:

$$i' = \begin{cases} L & \text{pro } i < T \\ H & \text{pro } i \geq T \end{cases}$$

kde i' je výstupní hodnota bodu po prahování, která nabývá hodnoty L nebo H podle toho, zda hodnota vstupního bodu i je nižší nebo vyšší než hodnota prahu T, jak lze nalézt v knize [11]. Pro binární podobu obrazu nabývá hodnota L minima a hodnota H maxima a výstupní bod je tedy černý nebo bílý. Lze využít také invertované prahování, kdy vyhodnocování probíhá opačně.

Pokud je v obraze velký rozdíl v intenzitách světel a stínů, toto prahování nemusí být vyhovující. V takovém případě lze využít adaptivní prahování, které neudává globální hodnotu prahu, ale jeho hodnota je vypočtena podle okolí hodnoceného bodu, jak uvádí kniha [2]. Výstupy obou prahování jsou zobrazeny na obrázku 2.2

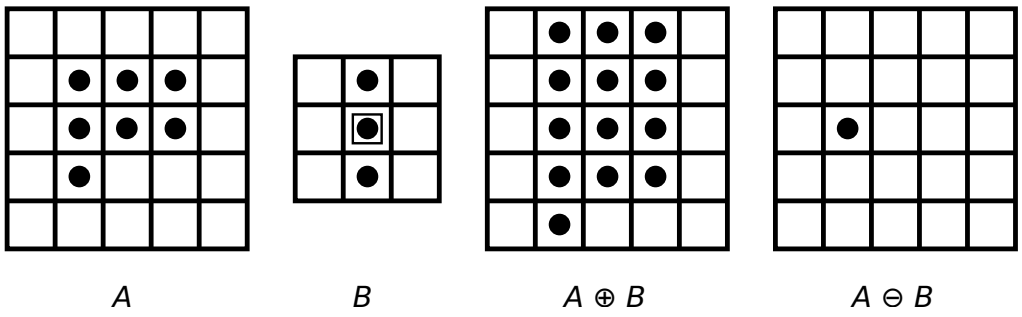


Obrázek 2.2: Vlevo je vstupní obraz, uprostřed výstup invertovaného prahování s globálním prahem a vpravo výstup invertovaného adaptivního prahování.

Obrazová morfologie

Základními morfologickými operacemi jsou dilatace a eroze. Dilatace pracuje tak, že ze vstupních bodů pomocí masky získá maximální hodnotu obsaženou v masce a výstupní bod určený kotvou (nejčastěji centrální bod masky) nastaví na toto maximum. Výsledkem dilatace je zvětšení a často spojení světlých oblastí. Eroze pracuje stejným způsobem, ale výstupní bod nastavuje na minimální hodnotu získanou z masky. Výsledkem eroze je zvětšení a často propojení tmavých oblastí, jak tvrdí kniha [2].

Na obrázku 2.3 je zobrazeno maticové vyjádření dilatace a eroze. Matice A představuje vstupní obraz, matice B je maska se zvýrazněnou kotvou v centrálním bodu. Operace $A \oplus B$ představuje výsledek dilatace a $A \ominus B$ je výsledek eroze.

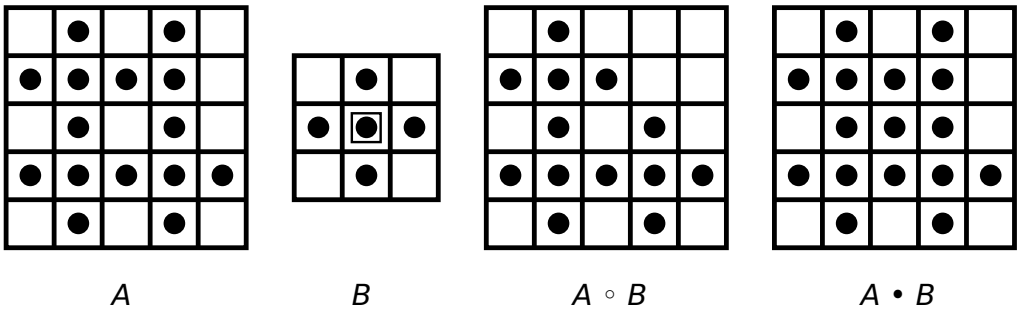


Obrázek 2.3: Maticové vyjádření operací dilatace a eroze. Obrázek je převzatý z knihy [10].

Odvozenými operacemi jsou otevření a zavření obrazu. Otevření obrazu je provedení nejprve eroze a následně dilatace, výsledkem operace je často propojení tmavých oblastí, které si ale zachovají svou původní velikost. Světlé oblasti jsou často izolovány. Zavření obrazu je naopak provedení nejprve dilatace a následně eroze. Výsledkem je často propojení světlých oblastí, které si také zachovají původní velikost a často dochází k izolaci tmavých oblastí, jak uvádí kniha [2].

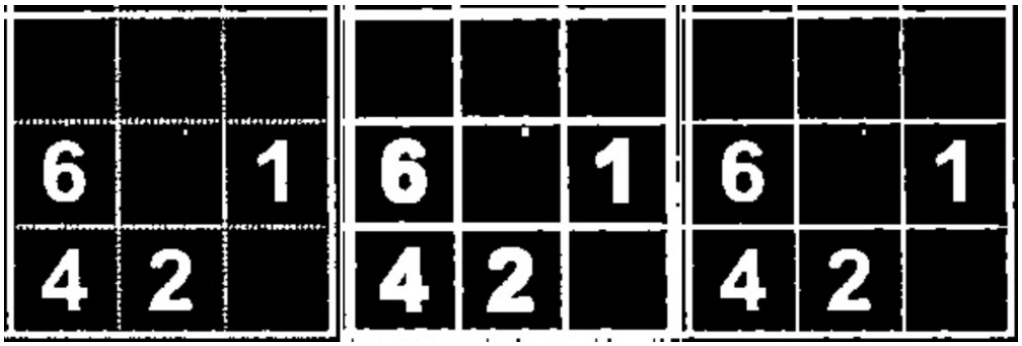
Na obrázku 2.4 je zobrazeno maticové vyjádření otevření a zavření. Matice A představuje vstupní obraz, matice B je maska se zvýrazněnou kotvou v centrálním bodu. Operace $A \circ B$ představuje výsledek otevření a $A \bullet B$ je výsledek zavření.

Rozdíl v dilataci a zavření konkrétního binárního obrazu lze vidět na obrázku 2.5. Eroze



Obrázek 2.4: Maticové vyjádření operací otevření a zavření. Obrázek je převzatý z knihy [10].

a otevření obrazu jsou operace opačné, v takovém případě by došlo k zúžení bílých číslic a okrajů a naopak rozšíření černých oblastí.



Obrázek 2.5: Vlevo je vstupní obraz, uprostřed výstup po dilataci a vpravo výstup po zavření obrazu.

2.2.3 Hledání mřížky Sudoku

Práce se zabývá řešením Sudoku, je tedy nutné najít mřížku se zadáním Sudoku. Tato část řeší dva možné přístupy vyhledání okrajů mřížky.

Vyhledání přímek pomocí Houghovy transformace

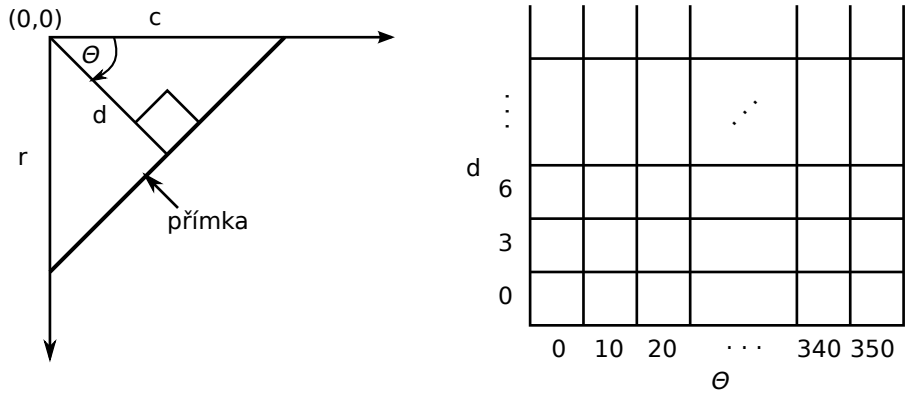
Houghova transformace spočívá v převedení obrazu do Houghova prostoru, kde se promítnou parametry hledaných útvarů, v tomto případě přímek. Je-li v původním obraze více bodů v jedné přímce, promítnou se v Houghově prostoru jako bod s vyšší intenzitou, tedy jako lokální maximum. Vyhledáním všech lokálních maxim v Houghově prostoru lze zjistit průběh přímek v původním obraze.

Standardní Houghův prostor pro vyhledávání přímek je tvořen dvourozměrným akumulátorem. Tyto rozměry jsou určeny parametry přímky. První parametr udává délku kolmice d od přímky k počátku obrazu (levý horní roh). Obraz je tvořen řádky r a sloupci c. Druhým parametrem hledané přímky je úhel Θ, který svírá kolmice d ke sloupcům c. Je-li takto nalezena přímka s odpovídajícími parametry, je akumulátor A inkrementován v místě

$A[\Theta, d]$. Rozměr d akumulátoru je závislý na velikosti obrazu a tedy na maximální možné vzdálenosti od počátku obrazu. Samotný parametr d lze spočítat rovnicí:

$$d = c \cdot cos\Theta - r \cdot sin\Theta$$

Jednotlivé parametry přímky a náčrt akumulátoru jsou zobrazeny na obrázku 2.6.



Obrázek 2.6: Vlevo jsou zobrazeny parametry přímky, vpravo je náčrtek akumulátoru. Obrázky jsou převzaté z knihy [9].

Jinou variantou této standardní Houghovy transformace je progresivní pravděpodobnostní Houghova transformace, která navíc vypočítá také délku každé přímky. Je pravděpodobnostní, protože přímku počítá pouze z určité části všech bodů na rozdíl od standardní Houghovy transformace, která počítá se všemi body přímky, jak uvádí kniha [2].

Vyhledání okrajů v obraze

Okraj je definovaný jako sekvence bodů, která reprezentuje křivku v obraze. V binárním obraze je okraj nalezen na rozhraní mezi bílou a černou barvou. Pokud je sekvence bodů vyhodnocena jako jedna přímka, je možné ponechat pouze její koncové body. Okraje lze také aproximovat a zjednodušit tak jejich reprezentaci, jak lze nalézt v knize [2]. Potom je možné vyhledat pouze takové okraje, které obsahují čtyři propojené body, tedy čtyřúhelník. Tento čtyřúhelník poté reprezentuje přímo okraj mřížky Sudoku.

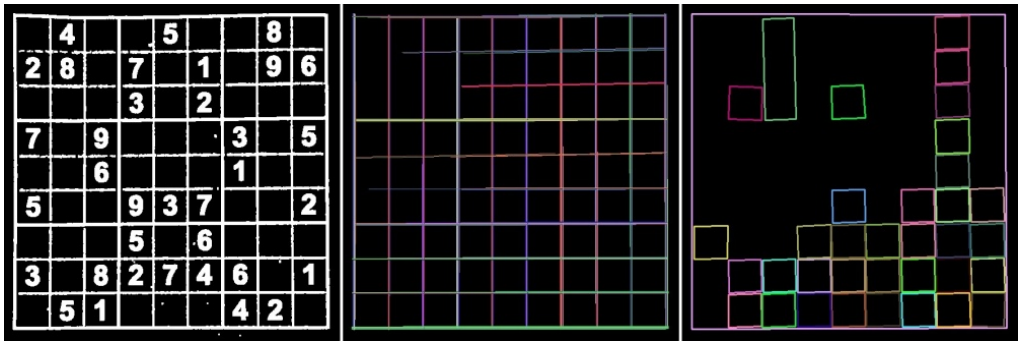
Příklady výstupů nalezených přímek pomocí progresivní pravděpodobnostní Houghovy transformace a nalezených čtyřúhelníkových okrajů po aproximaci jsou na obrázku 2.7.

2.3 Rozpoznávání textu v obraze

Získání textu z obrazu se věnuje optické rozpoznávání znaků (dále OCR). Tato problematika je značně rozsáhlá, ale tato část se jí zabývá pouze obecně. Dále se řeší také možnosti využití OCR na systému Android.

2.3.1 Optické rozpoznávání znaků

Celá tato podkapitola čerpá informace z knihy [5]. Optické rozpoznávání znaků je způsob rozeznání textových znaků ze statického obrazu bez dalších informací. Obecné rozpoznávání



Obrázek 2.7: Vlevo je vstupní obraz, uprostřed výstup vyhledání přímek Houghovou transformací a vpravo výstup vyhledání čtyřúhelníkových okrajů po aproximaci.

znaků se dělí na *on-line*, kdy se nejedná pouze o optické rozpoznávání. Probíhá zatímco jsou jednotlivé znaky psány a k dispozici jsou tak údaje o pohybu pera po podložce. Druhou větví je *off-line* rozpoznávání znaků ze statického obrazu. Dále se dělí na rozpoznávání:

- Jednotlivých znaků – znaky jsou od sebe navzájem odděleny a lze je tedy rozpoznávat po jednom. Tyto znaky mohou být tištěné nebo ručně psané.
- Ručně psaného skriptu – jedná se o text psaný rukou, kde jsou jednotlivá písmena ve slově navzájem propojena, nelze je tedy snadno oddělit. Dále se dělí na *rozpoznávání* psaných znaků nebo pouze *ověřování*, které slouží k porovnání se vzorem jako je například ověření podpisu.

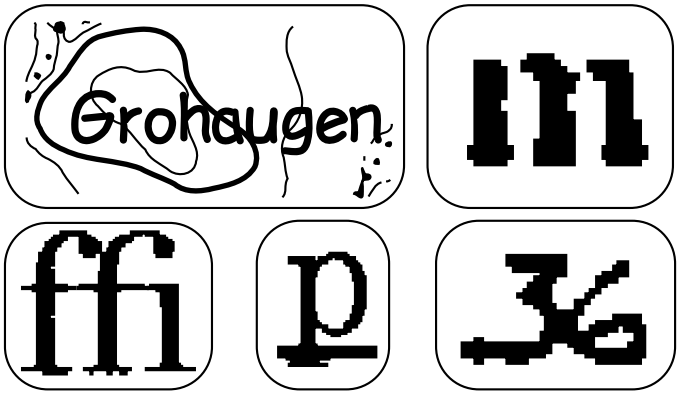
Metody OCR

Rozpoznávání znaků je založeno na učení stroje třídám vzorů. Tyto třídy mohou odpovídat například písmenům, číslicím a jiným speciálním znakům. Každá třída obsahuje určité vzory, se kterými může být porovnán rozpoznávaný znak. Nejlepší shoda se vzorem je určena jako výstup a tedy rozpoznání zkoumaného znaku. Celá problematika rozpoznání znaku se dá rozdělit na následující části.

Optické skenování je pořízení obrazu, ve kterém se nachází text k rozpoznání. Tato část zahrnuje převod obrazu do odstínů šedi a dále prahování z důvodu zjednodušení nesené informace a urychlení následných operací.

Umístění a segmentace zahrnuje nejprve zjištění pozice znaků v obraze a jejich následné oddělení na slova a poté na jednotlivé znaky. Hlavní problémy segmentace znaků jsou extrakce dotýkajících se znaků a fragmentovaných znaků, rozlišení šumu od textu a zaměnění grafiky nebo geometrie za text a opačně. Příklady degradovaných znaků lze vidět na obrázku 2.8.

Předzpracování zahrnuje normalizaci, což je změna jednotlivých znaků na jednotnou velikost, vyrovnání sklonu a rotace znaků. Dále následuje vyhlazení znaku, které zaplní drobné mezery a díry ve znaku a zúží linie znaku. Příklad předzpracování je na obrázku 2.9.



Obrázek 2.8: Příklad degradovaných znaků pro segmentaci. Obrázek je převzatý z knihy [5].

Obrázek 2.9: Příklad normalizace a vyhlazení. Obrázek je převzatý z knihy [5].

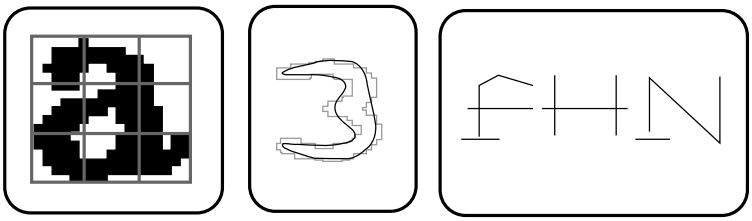
Extrakce příznaků je zjištění charakteristických příznaků znaku tak, aby tyto příznaky byly pro každý znak jedinečné. Hlavním problémem extrakce je *robustnost*, která zahrnuje citlivost na šum, zkreslení, odlišnosti ve tvarech stejných písmen a umístění a otočení znaků. Dalším významným bodem je *praktické použití*, které zkoumá rychlost rozpoznávání, složitost implementace a nezávislost na jiných metodách.

Samotná extrakce může být založena pouze na *porovnávání vzorů a korelačních technikách*, kdy tato metoda ze znaků žádné příznaky skutečně neextrahuje. Spočívá v zachování matice bodů znaku, která se následně porovnává se vzory.

Další metodou extrakce jsou *techniky založené na příznacích*. Tyto metody definují různý přístup k popisování znaků a dále se dělí na:

- Rozložení bodů – tato metoda zkoumá statistické rozložení bodů znaku podle různých podmínek, mezi které spadá členění na zóny, momenty, průsečíky a vzdálenosti, n-tice a charakteristika loci.
- Transformace a rozložení řad – tyto metody jsou založené na popisu křivek získaných z okrajů znaků.
- Strukturální analýza – zaměřuje se na popis geometrie a topologickou strukturu znaků.

Příklady některých metod extrakce příznaků jsou na obrázku 2.10.



Obrázek 2.10: Příklad zónování, Fourierovy transformace a strukturální analýzy. Obrázky jsou převzaté z knihy [5].

Klasifikace porovnává znaky s třídními vzory podle extrahovaných příznaků. Klasifikace využívá buď přístupu *teoretických rozhodovacích metod*, které se dále dělí na:

- Porovnávání – identifikuje znaky podle nejlepší shody se vzorem podle daných příznaků.
- Optimální statistické klasifikátory – využívají k identifikaci znaků pravděpodobnost. Pro každý znak je vypočítána pravděpodobnost, že spadá do dané třídy a znak je následně přiřazen třídě s nejvyšší pravděpodobností příslušnosti.
- Neuronové sítě – pracují v několika úrovních. Znak je přiveden na vstup a jednotlivé spoje sítě produkují výstupy s určitými vahami, které jsou následně sumarizovány a určují celkový výstup.

Dalším přístupem klasifikace jsou *strukturální metody*, které jsou založené na syntaktické analýze. Jednotlivé znaky jsou budovány pomocí gramatik, které odpovídají různým třídám znaků.

Následné zpracování se dělí na *seskupování*, které spočívá ve spojování jednotlivých znaků do slov. Dalším bodem následného zpracování je *detekce a korekce chyb*, která umožňuje rozpoznat chybné znaky v jednotlivých slovech. Detekce chyby v samostatném znaku je obtížná, zatímco detekovat chybu ve slově lze pomocí specifických pravidel jazyka, ve kterém rozpoznávání probíhá. Jinou metodou, která umožňuje i opravu chyb, je slovníková metoda, která rozpoznané slovo vyhledává ve slovníku a pokud jej nenalezne, může jej opravit podle nejlepší shody.

Možnosti OCR

Různé OCR systémy nabízejí několik možností jak přistupovat k sadám znaků, které rozpoznávají a to:

- Fixní font – Tyto systémy pracují s jedním daným fontem. Nejčastěji se jedná o font určený ke strojovému čtení, například OCR-A, OCR-B, Pica, Elite a další. Tyto systémy dosahují vysoké přesnosti a rychlosti.
- Multifont – systémy obsahují více samostatných fontů. Jsou následníky systémů s fixními fonty.
- Omnifont – systémy obsahují místo fontů charakteristické popisy jednotlivých znaků. Jsou tedy variabilní pro více různých druhů fontů.

- Omezené ruční psaní – Tyto systémy umožňují čtení rukou psaných znaků, které však musí splňovat určité požadavky. Často se používají pro skenování formulářů.
- Skript – Tyto systémy se zaměřují na rozpoznávání rukou psaného textu. Jednotlivé znaky ve slovech tohoto textu se přitom vzájemně dotýkají a slovo tak tvoří jeden celek. Rozpoznání znaků v takovém textu je velice obtížné. Tyto systémy se proto používají převážně pro ověřování podpisů.

Další problematika OCR

Jedná se především o typické chyby OCR systémů, protože jejich přesnost je závislá převážně na kvalitě vstupu. Nejčastější problémy jsou odlišnosti v tvarech znaků, deformace, odlišnosti v mezerách mezi znaky nebo slovy a směs textu a grafiky.

Další významnou problematikou je vyhodnocení výkonu OCR systémů, protože nemají žádné standardizované testy a pro jejich srovnávání se často používá hodnocení v těchto oblastech:

- Míra rozpoznání – poměr správně rozeznaných znaků.
- Míra zamítnutí – poměr znaků, které nebyly rozpoznány.
- Míra chybovosti – poměr chybně rozpoznaných znaků.

2.3.2 Optické rozpoznávání znaků pro Android

Tato část se zabývá možnostmi využití OCR systémů pro Android. První variantou je využití dostupných knihoven. Mezi nejvýznamnější patří:

- *Tess-two*[4] – je odnož nástroje *Tesseract OCR*[5]. Obsahuje také knihovnu Leptonica pro zpracování obrazu.
- *ABBYY Mobile OCR Engine*[6] – je nástroj pro vývoj software, který sdružuje více nástrojů dohromady. Umožňuje například použití překladače a slovníků pro překlad rozpoznaného textu do různých jazyků. Tento nástroj je zdarma dostupný pouze jako zkušební verze.

Další možností, jak zprovoznit OCR pro Android je využití knihoven pro zpracování obrazu, které byly probírány v části 2.2.1. Tyto knihovny obsahují různé prvky pro extrakci příznaků a klasifikaci znaků a umožňují tak sestavit vlastní OCR systém.

2.4 Hra Sudoku

V této části je uveden zjednodušený popis Sudoku. Variant této hry je více a také se liší velikost mřížky, která se může pohybovat například od rozměrů 4x4 po 25x25. Sudoku může být vyplňováno čísly, písmeny nebo jinými znaky, jejichž počet odpovídá jednomu rozměru mřížky. Tato práce se zabývá nejčastější variantou Sudoku velikosti 9x9, které se vyplňuje číslicemi od 1 do 9. Dále jsou popsány principy řešení Sudoku.

[4] Více o Tess-two na stránkách: https://github.com/rmtheis/tess-two
[5] Více o Tesseract OCR na stránkách: http://code.google.com/p/tesseract-ocr
[6] Více o ABBYY Mobile OCR Engine na stránkách: http://www.abbyy.com/mobileocr

2.4.1 Popis hry

Příklad zadání Sudoku je na obrázku 2.11. Celá mřížka se dělí na devět řádků *a* až *i*, devět sloupců *1* až *9* a devět bloků zvýrazněných tlustými čarami. Například levý horní blok je označen jako *abc123*. Cílem hry je doplnit chybějící čísla v zadání a to takovým způsobem, že v každém sloupci, řádku a bloku se vyskytují všechna čísla od 1 do 9, každé pouze jednou.

Například v řádku *a* se již vyskytují čísla 2, 6, 7 a 9 a na zbývající políčka v tomto řádku mohou být doplněna pouze čísla 1, 3, 4, 5 a 8. Ovšem v blocích *abc123* a *abc456* se číslo 1 již vyskytuje a jsou tak vyloučena políčka *a2*, *a5* a *a6*. Číslo 1 se také objevuje ve sloupci *9* a tím je vyloučeno pole *a9*. Jediné políčko v řádku *a*, které může obsahovat číslo 1, je tak na pozici *a8*.

	1	2	3	4	5	6	7	8	9	
a	2		9	7			6			a
b	5		1	9	8					b
c					1					c
d	4	1							5	d
e			6		4		7			e
f	7							9	4	f
g				1	3					g
h					9	2			6	h
i			7			8	4		1	i
	1	2	3	4	5	6	7	8	9	

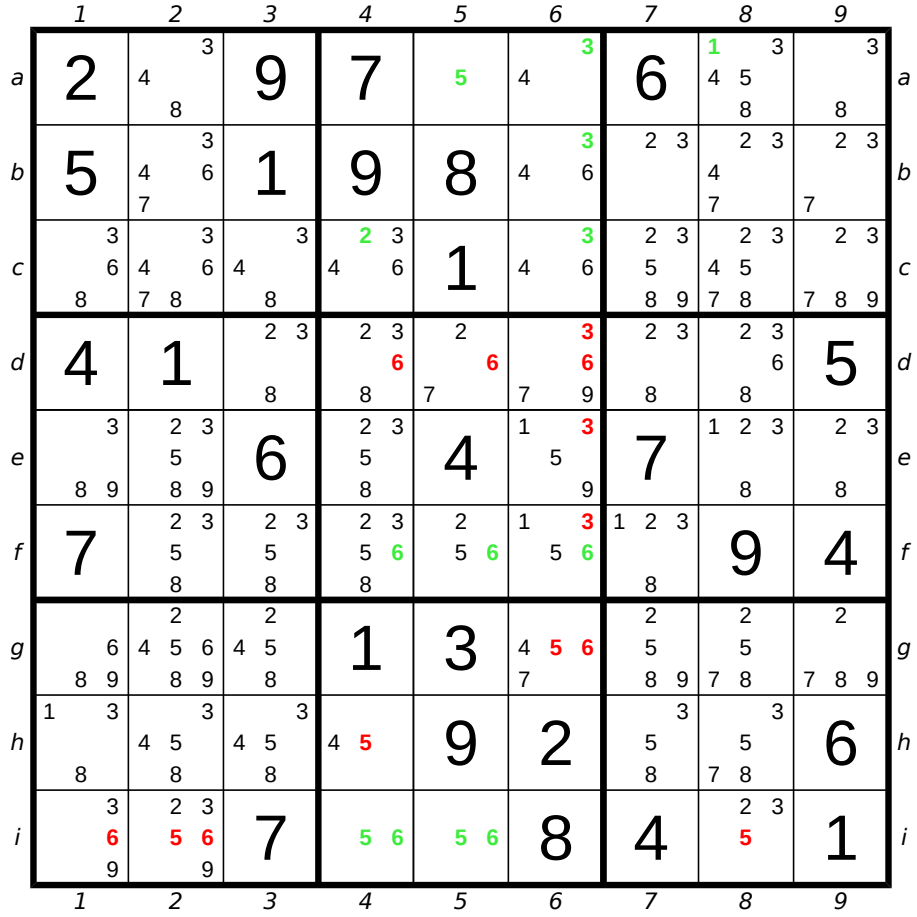
Obrázek 2.11: Příklad zadání Sudoku

2.4.2 Princip řešení

Pro řešení Sudoku lze využít dvou různých metod, první je založena na *eliminačních technikách*, které vycházejí z pravidel hry. Druhou metodou je *hádání* a to spočívá ve zkoušení všech možných variant. Následuje popis obou variant, který vychází z článků [6, 8].

Eliminační techniky

Pro každé políčko mřížky je podle pravidel hry určeno, která čísla může toto políčko obsahovat. Tato čísla se nazývají kandidáti. Zkoumáním kandidátů lze eliminovat jiné kandidáty a postupně tak dojít k vyřešení Sudoku. Tímto zkoumáním a eliminováním kandidátů se zabývají následující techniky, které k popisu využívají obrázek 2.12.



Obrázek 2.12: Zobrazení kandidátů v mřížce

Odhalený jedinec je takový kandidát, který se v daném políčku vyskytuje osamoceně a na dané políčko tedy není možné doplnit jiné číslo. Jedná se o číslo 5 zobrazené zeleně na pozici *a5*. Doplněním tohoto čísla lze poté eliminovat všechny výskyty daného čísla ve stejném řádku, sloupci a bloku. V příkladu by se jednalo o odstranění čísla 5 jako kandidáta z řádku *a*, sloupce *5* a bloku *abc456*. Ovšem výskyt čísla pět v políčku *i5* je použit pro demonstraci jiných technik a proto není zvýrazněna jeho eliminace.

Skrytý jedinec je kandidát, který podobně jako odhalený jedinec může být doplněn pouze na jedinou pozici. Rozdíl je v tom, že takový kandidát se v daném políčku nevyskytuje samostatně. Jeho určení vychází z pravidla, že dané číslo se může v každém řádku, sloupci nebo bloku vyskytovat právě jednou. Je-li tedy daný kandidát jedinečný v rámci řádku, sloupce nebo bloku, může být doplněn. V příkladu se jedná například o zeleně zobrazené číslo 1 na pozici *a8*, které je jedinečné v rámci řádku *a* nebo bloku *abc789* a o číslo 2 na pozici *c4*, které je jedinečné v rámci řádku *c* nebo bloku *abc456*. Doplněním takového čísla by dále došlo k eliminaci dalších kandidátů jako v případě odhaleného jedince.

Uzamčený kandidát je jedno stejné číslo, které se vyskytuje na více pozicích v jednom řádku, sloupci nebo bloku. Jsou dvě možné varianty jeho výskytu a to buď, že číslo je jedi-

nečné v rámci řádku nebo sloupce a zároveň se objevuje pouze v jednom bloku. Příkladem je zelené číslo 6, které je jedinečné v řádku *f* a vyskytuje se pouze v bloku *def456*. Toto číslo se musí vyskytovat na svém řádku *f* a je tedy možné odstranit další výskyty tohoto čísla z bloku *def456*, což jsou čísla 6 zobrazená červeně v tomto bloku.

Druhou variantou je jedinečnost čísla v rámci bloku, přičemž se toto číslo vyskytuje v jednom řádku nebo sloupci. Pokud by se v příkladu doplnila zelená 2 na pozici *c4*, zelená čísla 3 v bloku *abc456* by se stala jedinečnými ve svém bloku ve sloupci *6*. To umožňuje vyloučit všechny další výskyty čísla 3 ve sloupci *6*, která jsou v příkladu zobrazena červeně.

Odhalené dvojice a trojice jsou taková dvě (tři) políčka, která obsahují právě dva (tři) stejné kandidáty a vyskytují se v rámci jednoho řádku, sloupce nebo bloku, ze kterého mohou být další výskyty těchto kandidátů eliminovány. V příkladu se jedná o zelená čísla 5 a 6 na pozicích *i4* a *i5*. Protože leží ve stejném řádku, je možné z něj další výskyty těchto čísel odstranit, v příkladu jsou to červená čísla 5 a 6 v řádku *i*. Dále leží tato čísla také ve stejném bloku, proto lze z bloku *ghi456* odstranit výskyty těchto čísel, v příkladu zobrazených červeně v tomto bloku.

Výskyt čísel se může rozšířit i na čtveřice, pětice a dále, ale pro většinu zadání Sudoku je toto zbytečné. Navíc se tím zvyšuje náročnost této metody a poté je již snazší přistoupit k hádání.

Skryté dvojice a trojice se mají stejně jako skrytý jedinec k odhalenému jedinci. Nejsou v políčkách jako samostatní kandidáti, ale v rámci jednoho řádku, sloupce nebo bloku se vyskytují právě dvakrát (třikrát) ve dvou (třech) stejných políčkách. Jako odhalené dvojice a trojice je lze rozšířit na čtveřice a dále, ale stejně tak je to přebytečné.

Další techniky zahrnují například metody X-Wing, XY-Wing nebo Swordfish, které vychází z technik předchozích nebo jsou navíc založené na hádání. Tyto metody jsou ojediněle využitelné a je snazší přistoupit přímo k hádání.

Hádání

Samotné hádání je nejčastěji založeno na metodě backtrackingu. Je vybráno první volné políčko a do něj je dosazeno číslo 1. Poté se zkontroluje, zda nebyla porušena pravidla a pokud ano, dosadí se číslo 2 a tak dále. Pokud pravidla nebyla porušena, přechází se na další volné políčko a opět se zkouší dosazovat čísla. Tato metoda je časově náročná a zkouší i takové možnosti, které porušují pravidla a lze je přeskočit.

Vhodné je kombinovat metodu hádání s eliminačními technikami, kdy se nejprve doplní a vyřadí všichni možní kandidáti. Poté, co již eliminační metody nenaleznou další postup, přistoupí se k hádání. V prvním volném políčku se dosadí první možný kandidát a poté se opět pokračuje eliminačními metodami. Pokud by došlo k porušení pravidel, v políčku, kde došlo k hádání, se přistoupí k dalšímu kandidátovi. Tuto metodu lze ještě posunout a nehádat v prvním volném políčku, ale v takovém políčku, které obsahuje nejmenší počet kandidátů.

2.5 Existující Android aplikace pro vyfocení a řešení Sudoku

Zde je výpis některých aplikací dostupných na Google Play, které dokáží vyfotit a rozpoznat Sudoku. Těchto aplikací lze nalézt více, ale jejich funkčnost je podobná jako zde uváděných a je jich pouze několik jednotek.

- *Sudoku Vision*[7] – nejobsáhlejší aplikace, která umí generovat zadání Sudoku, umožňuje hrát Sudoku a nabízí nápovědu pro řešení pomocí různých metod nebo přímo Sudoku vyřeší. Při pořízení snímku zobrazuje postup jeho zpracování a umožňuje takto načíst zadání Sudoku. Po pořízení je možnost Sudoku hrát nebo vyřešit. Aplikace zvládá vyřešit Sudoku i v rámci rozšířené reality.
- *AR Sudoku Solver*[8] – řešitel Sudoku, který umožňuje řešení pouze v rozšířené realitě. Nelze pořídit fotografii a načíst tak zadání.
- *Sudoku Camera*[9] – po vyfocení umožňuje upravit umístění rohů Sudoku, poté rozpozná čísla a umožní jejich opravu a zobrazí je ve vlastní mřížce. Nakonec je nabídnuta možnost Sudoku vyřešit. Slouží pouze k rozpoznávání a řešení, neslouží pro hraní hry.
- *Sudoku Grab'n'Play*[10] – umožňuje vyfotit Sudoku, které je načteno do mřížky. Aplikace neumí Sudoku vyřešit, umožňuje pouze jeho kontrolu a hraní.

[7] Sudoku Vision dostupné na stránkách:
https://play.google.com/store/apps/details?id=com.rogerlebo.sudokuvision
[8] AR Sudoku Solver dostupný na stránkách:
https://play.google.com/store/apps/details?id=com.enigon.sudokusolver
[9] Sudoku Camera dostupné na stránkách:
https://play.google.com/store/apps/details?id=eu.codepoetry.sudokucamera
[10] Sudoku rab'n'Play dostupné na stránkách:
https://play.google.com/store/apps/details?id=com.onegravity.sudoku.grabandplay

Kapitola 3

Návrh aplikace

Tato kapitola se zabývá návrhem samotné aplikace z pohledu, co má aplikace nabídnout, jak má být rozvržen vzhled a jak pojmout přístup k řešení Sudoku v pořízeném snímku z kamery.

3.1 Co aplikace nabízí

Z krátkého přehledu již existujících aplikací v části 2.5 vyplývá, že podobných aplikací není mnoho. Většina aplikací tento problém potom řeší způsobem, že Sudoku načtené z obrázku vykresluje do vlastní mřížky. Pouze dvě aplikace byly schopné vyřešit celé Sudoku v rámci rozšířené reality a jedna z těchto ani neumožňovala pořízení snímku. Proto jsem se rozhodl, že moje aplikace bude výsledek řešení zakreslovat přímo do snímku po jeho pořízení. Tento návrh navíc umožňuje snadné doplnění aplikace o rozšířenou realitu, kdy se řešení bude vykreslovat přímo v náhledu z kamery.

Při pořízení snímku nemusí být nalezené zadání správné, proto by aplikace měla obsahovat prvky pro úpravu načteného zadání. Tato aplikace je pouze řešitelem a tyto prvky tedy neumožňují hraní Sudoku.

Aplikace je pak vhodná pro snadnou kontrolu výsledku řešení. Pokud je dostupné pouze zadání nebo se řešení nachází například v časopise na jiné stránce, je snazší vzít zařízení s kamerou, vyfotit dané zadání a mít ihned k dispozici řešení. Je to výhodnější i z toho pohledu, že zařízení s Androidem, která toto umožňují, jsou stále rozšířenější a jsou obvykle blíže po ruce než například počítač, do kterého by se zadání muselo navíc složitě přepisovat.

3.2 Vzhled a možnosti aplikace

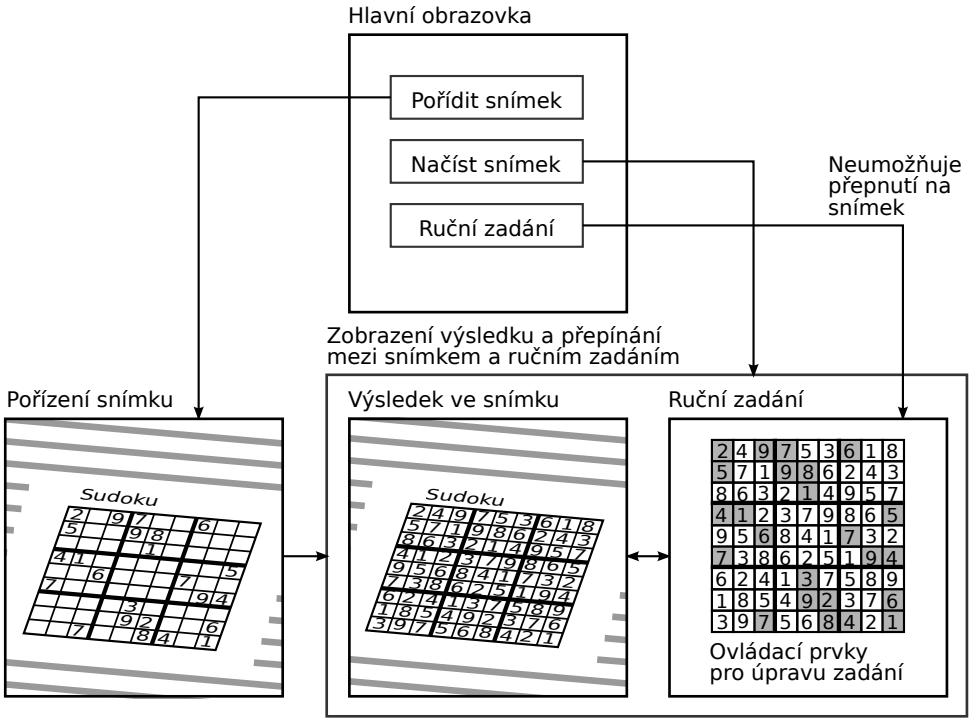
Podle rozhodnutí z předchozí části jsem vytvořil návrh aplikace, který je na obrázku 3.1. Při spuštění aplikace se zobrazí hlavní obrazovka, která bude nabízet zobrazené možnosti.

Po zvolení možnosti pořídit snímek se bude zobrazovat náhled z kamery zařízení. V tomto momentě by také mohla fungovat rozšířená realita, kdy by bylo vykresleno celé řešení nebo pouze jeho část, jako je nalezení a zvýraznění mřížky. Po pořízení snímku se obrazovka přepne na zobrazení výsledku. Další možností v hlavní nabídce je načtení snímku, které přeskočí jeho pořizování a rovnou přistoupí k zobrazení výsledku.

Toto zobrazení výsledku jsem se rozhodl rozdělit na dvě části. V první řadě zde bude výsledek řešení zobrazený přímo ve snímku. Snímek však může být pořízený takovým způsobem, že by nebylo snadné přistupovat k jednotlivým políčkům mřížky pro jejich ruční

úpravu. Z toho důvodu je zde možnost přepnout obrazovku na ruční zadávání, kde bude vykreslena samostatná mřížka, ve které bude umožněno provádět úpravy. Ty se následně promítnou zpět do pořízeného snímku.

Poslední možností v hlavní nabídce je přistoupit pouze k samostatnému ručnímu zadávání. Pokud by se nedařilo vyřešit Sudoku v pořízeném snímku, je tak stále možnost zkontrolovat řešení alespoň ručním přepsáním.



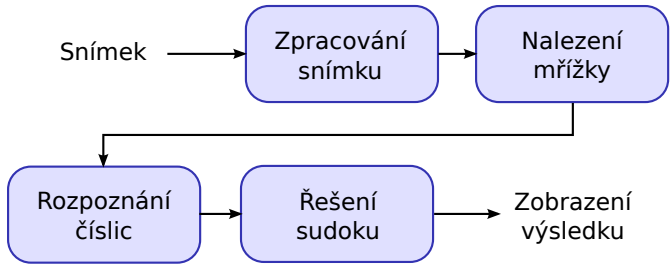
Obrázek 3.1: Obrazovky a možnosti výsledné aplikace

3.3 Návrh modulů pro implementaci

Nalézt v obrázku Sudoku a vyřešit ho je značně rozsáhlý problém, jehož hlavní složky jsou zobrazeny na obrázku 3.2. Tímto rozdělením jsem se také řídil při návrhu modulů, které celý problém zjednodušují.

Zpracování snímku je zkráceně převod obrazu do binární podoby a následně jeho vyhlazení ve smyslu propojení malých mezer v obraze. V takto upraveném obraze je potřeba nalézt mřížku, o což se stará další modul. Ten by si měl pamatovat, kde se mřížka nachází, aby bylo možné zpětně vykreslovat výsledek řešení nebo pouze vykreslovat zvýraznění mřížky.

Dalším krokem je poté v mřížce rozpoznat čísla a načíst tak zadání Sudoku. Poslední částí je vyřešení Sudoku a získání výsledku. Výsledek by měl mít možnost rozlišit, která čísla byla určena jako vstupní nebo vyřešená a také určit, která čísla v zadání porušují pravidla Sudoku. Z celkového výsledku by také mělo jít snadno zjistit, zda bylo nalezeno řešení nebo byl neplatný vstup nebo zda se řešení nalézt nezdařilo.



Obrázek 3.2: Zjednodušený pohled na části řešení Sudoku

Toto rozdělení je vhodné i z hlediska návrhu vzhledu a chování, které bylo popsané v předchozí části. Pokud se má při pořizování snímku jako rozšířená realita kolem nalezené mřížky pouze vykreslovat zvýraznění, bylo by zbytečné přistupovat k dalším krokům řešení. Takto stačí propojit první dva moduly. Při ručním zadávání jsou naopak nepodstatné veškeré části, které pracují s obrazem a stačí použít pouze modul pro řešení Sudoku.

Kapitola 4

Implementace a popis aplikace

Tato kapitola stručně popisuje implementované třídy, poté se zaměřuje na některé detaily implementace, například jaké knihovny byly využity, způsob zjišťování mřížky v obraze nebo porovnání metod pro řešení Sudoku. Poslední částí kapitoly je přehled vzhledu a možností výsledné aplikace.

4.1 Přehled implementovaných tříd

Pro představu, jak jsem rozdělil zdrojové soubory do balíčků, se lze podívat do přílohy A. Dále následuje popis toho, co která třída obsahuje.

Třída `CameraView.java` je rozšířením třídy, kterou implementuje knihovna OpenCV a slouží pro obsluhu kamery. Toto rozšíření je vhodné z hlediska získání parametrů kamery, například rozlišení, ve vlastní třídě a není potřeba tyto parametry obsluhovat jinde.

Třída `HomeScreen.java` je úvodní obrazovkou aplikace. Další obrazovkou aplikace je třída `CapturePhoto.java`, která umožňuje pořídit snímek z kamery. Tato třída využívá k zobrazení náhledu z kamery knihovnu OpenCV a implementuje také rozšířenou realitu ve formě zvýraznění nalezené mřížky. Dále třída `ViewResult.java` je obrazovkou pro sloučení tříd `ViewPhoto.java` a `SudokuManualEntry.java` pomocí panelů. Třída `ViewPhoto.java` slouží pouze pro zobrazení snímku s doplněným řešením. Třída `SudokuManualEntry.java` pak vykresluje vlastní mřížku a umožňuje ruční úpravu zadání. Zde jsem s výhodou využil faktu, že jazyk Java předává ukazatele na objekty a tak například tabulka se zadáním Sudoku je pouze jedna a její změna ve třídě pro ruční úpravu se promítne i ve třídě pro zobrazení snímku.

Třída `OCRReader.java`, která využívá knihovny tess-two pro rozpoznávání znaků, slouží k inicializaci této knihovny, jejímu ukončení a především k rozpoznání znaku v předaném obrázku. Pokud knihovna rozpozná něco jiného než číslice, jsou následně tyto znaky ignorovány. Knihovně je vždy předán pouze obrázek jednoho políčka z mřížky.

Třída `SudokuHolder.java` je implementací tabulky Sudoku pomocí dvourozměrného pole a navíc drží informace o výsledku řešení a o typu čísel načtených v tabulce, zda se jedná o vstupní nebo vyřešené číslo, o prázdné políčko nebo o detekovanou chybu. Třída `SudokuImageParser.java` slučuje zpracování obrazu a vyhledání mřížky pomocí knihovny OpenCV a navíc si v paměti drží vstupní obraz a obraz mřížky z důvodu urychlení operací pro překreslení výsledku řešení. Poslední třída `SudokuSolver.java` je implementací řešení Sudoku pomocí eliminačních metod kombinovaných s hádáním, pokud by zadání nebylo jednoznačné. Třída vyhledá první vhodnou variantu nebo informuje o nemožnosti řešení.

4.2 Detaily implementace

V této části podrobněji popisuji jak a proč jsem se rozhodl řešit některé problémy a to převážně z hlediska výkonnosti aplikace. Občas v textu zmiňuji dobu řešení určitých problémů. Jak jsem tuto dobu měřil a na jakých zařízeních, je popsáno v části 5.1 v následující kapitole.

4.2.1 Nastavení Androidu

V aplikaci využívám knihovnu OpenCV a ta vyžaduje minimální verzi Androidu 2.2 a vyšší, moje aplikace proto tuto verzi také podporuje jako minimální. Dalším požadavkem je, aby zařízení, kde se má aplikace instalovat, obsahovalo kameru. Při instalaci si poté aplikace vyžádá oprávnění používat kameru a potom také externí úložiště. To je z toho důvodu, že knihovna tess-two pro rozpoznávání znaků vyžaduje soubor s klasifikačními daty uložený v paměti zařízení.

Nastavit parametry kamery jsem se rozhodl tak, že pro náhled při pořizování snímků je povoleno maximální rozlišení 480 pixelů v libovolném rozměru. Při této podmínce se nastaví největší možné rozlišení, které zařízení nabízí. Pro pořízení snímku jsem se naopak rozhodl nastavit nejnižší možné rozlišení, které však bude minimálně 800 pixelů v libovolném rozměru. Tyto hodnoty jsem ověřoval experimentálně a zjistil jsem, že i v takovémto rozlišení lze číslice snadno rozpoznávat a není nutné zvyšovat nároky na výkon nastavováním vyšších rozlišení.

4.2.2 Zpracování obrazu a vyhledání mřížky

Zpracování obrazu jsem se rozhodl řešit pomocí knihovny OpenCV, jelikož například oproti knihovnám FastCV nebo BoofCV lze najít více dotazů a funkčních příkladů na různých stránkách[1]. Podle porovnání uváděného v tabulce 2.1 v části 2.2.1 není pro většinu operací rozdíl v rychlostech knihoven OpenCV a FastCV pro běžné procesory příliš výrazný a navíc se zde nejedná o knihovnu OpenCV optimalizovanou pro Android.

V základních úpravách obrazu jsem se rozhodl využít Gaussova filtru, protože při zkoušení bilaterálního filtru jsem zjistil, že je časově několikanásobně náročnější. Následuje invertované dynamické prahování a poté operace zavření obrazu. Časová náročnost zavření byla podobná jako u dilatace, avšak nedocházelo k zaplnění děr uvnitř číslic, které je nežádoucí.

Knihovna OpenCV nabízí vyhledávání přímek pomocí Houghovy transformace. Ovšem při hledání mřížky touto variantou je zapotřebí, aby mřížka Sudoku byla jediným objektem v obraze. Jinak jsou často nalezeny přímky i v okolí a ty následně znemožňují detekci skutečných okrajů mřížky. Proto jsem tuto variantu zavrhl a rozhodl jsem se pro využití funkce, která v obraze vyhledává okraje. Takto nalezené okraje je možné aproximovat a vyhledávat například pouze čtyřúhelníky. Navíc po použití této funkce jsou nalezeny přímo dané okraje mřížky a ne pouze přímky, u kterých se ještě musí určit, které jsou okrajové.

4.2.3 Orientace mřížky a detekce čísel

Pro detekci čísel jsem se rozhodl využít možností knihovny tess-two. K té je potřeba mít soubor s daty určitého jazyka, které slouží pro klasifikaci znaků. K aplikaci jsem proto přibalil soubor s daty pro angličtinu z funkčního příkladu aplikace využívající tess-two.

[1] Počet nalezených výsledků pomocí Google vyhledávače je pro heslo *OpenCV* asi 2,5 milionu, zatímco pro hesla *FastCV* nebo *BoofCV* je výsledků asi 15 tisíc.

Bylo to z toho důvodu, že tento soubor má velikost necelé 2 MB oproti více než 20 MB dat, které nabízí k dispozici Tesseract OCR pro anglický jazyk[2].

Zjišťovat orientaci mřížky jsem se rozhodl tak, že se mřížka postupně otáčí do čtyř směrů a získané obrázky se předávají knihovně tess-two. Ze získaných znaků se filtrují pouze číslice a porovnává se počet načtených číslic. Podle toho, kde bylo nalezeno nejvíce číslic, bylo možné určit orientaci mřížky. Toto řešení bylo ovšem příliš pomalé a proto jsem se rozhodl detekci orientace zcela vynechat a pouze instruovat uživatele, aby snímek Sudoku pořizoval správně otočený.

4.2.4 Vykreslování výsledku zpět do snímku

V kódu jsem se rozhodl nejprve uložit pouze pozici mřížky, avšak při zpětném vykreslování, kdy bylo nutné znovu provádět perspektivní transformaci mřížky, docházelo k znatelnému zpomalení. Proto jsem se na úkor paměťových nároků rozhodl uchovat již transformovanou mřížku, do které se pouze vykreslí výsledek. Poté stačí mřížku opět transformovat zpět do původního obrázku. Toto řešení urychlilo celé vykreslení v řádu až několika sekund na slabších zařízeních.

4.2.5 Řešení Sudoku

Původně jsem řešení Sudoku implementoval pouze hádáním pomocí backtrackingu, ale toto řešení se ukázalo pomalé, pokud bylo v zadání malé množství čísel. Trvalo v řádu několik sekund. Proto jsem se rozhodl pro implementaci eliminačních technik a to konkrétně pouze technik odhalený a skrytý jedinec a uzamčený kandidát. Pokud by použitím těchto technik nebylo možné dále postupovat, přistoupí se opět k technice hádání pomocí backtrackingu ovšem kombinovaného právě s eliminací. Toto řešení zkrátilo čas do řádu desítek milisekund nebo stovek milisekund v případě prázdného zadání.

4.2.6 Rozšířená realita

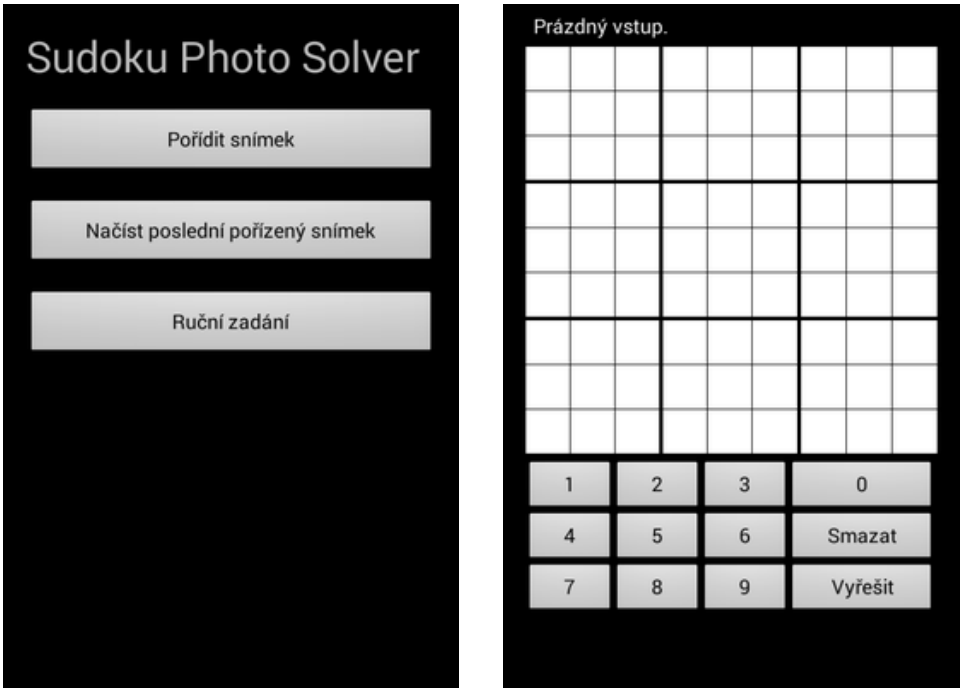
Po implementaci všech těchto postupů jsem se rozhodl otestovat běh rozšířené reality. Ovšem již pouze vyhledání okrajů mřížky a její následné zvýraznění zhoršilo počet snímků za sekundu z průměrně 15 na pouhé přibližně 3 snímky za sekundu u nejslabšího zařízení. Přesto jsem se rozhodl toto řešení v implementaci ponechat, protože se nejedná o aplikaci se zaměřením na plynulost videa a pomalejší zobrazování snímků tak nepovažuji za kritické.

4.3 Výsledná aplikace

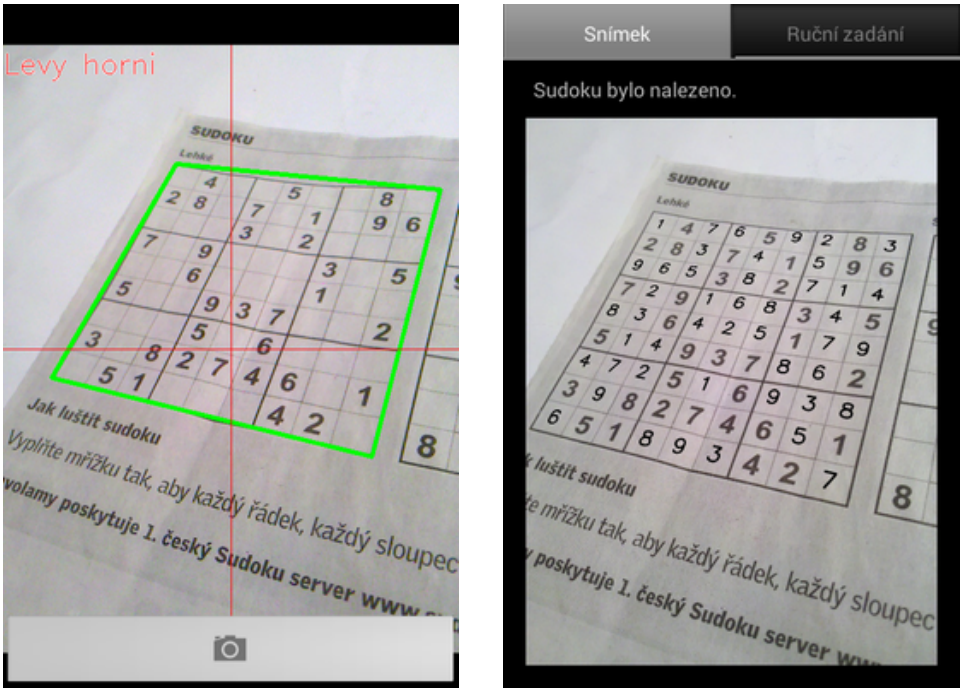
V této části je ukázka vzhledu a popis možností výsledné aplikace. Na obrázku 4.1 je vlevo zobrazena úvodní obrazovka, která nabízí tři možnosti. Poslední možnost *Ruční zadání* vede na obrazovku, která je na obrázku vpravo. Zde je možné pouze ručně doplnit čísla ze zadání a nechat Sudoku vyřešit. Doplňování probíhá výběrem čísla a poté klikáním na jednotlivá políčka mřížky, kam se vybrané číslo doplní.

Druhou možností na úvodní obrazovce je *Načíst poslední pořízený snímek*, která vede přímo na obrazovku s řešením naposledy pořízeného snímku. Tuto variantu jsem se rozhodl uvést, aby nebylo nutné znovu pořizovat jeden snímek, kdyby uživatel chtěl opakovaně kontrolovat řešení jednoho Sudoku.

[2]Soubory s daty lze nalézt na stránkách Tesseract OCR: http://code.google.com/p/tesseract-ocr



Obrázek 4.1: Vlevo je úvodní obrazovka, vpravo je samotné ruční zadávání.

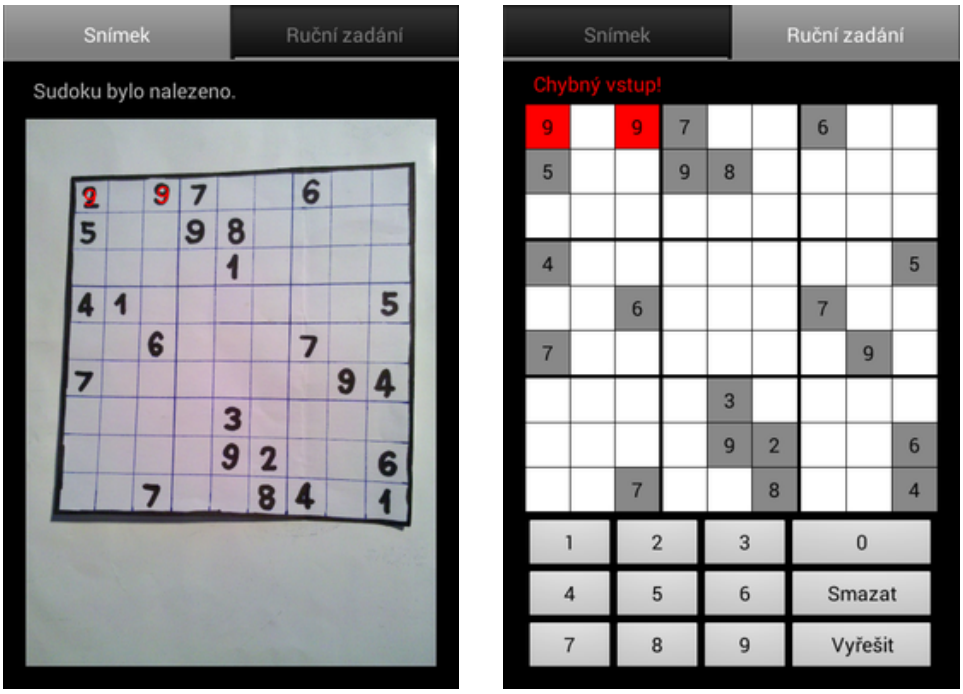


Obrázek 4.2: Vlevo je snímek pořizování, vpravo je výsledek řešení ve snímku.

Nakonec se dostávám k první možnosti *Pořídit snímek*, která je nejzajímavější. Tato možnost přepíná na obrazovku, která je na obrázku 4.2 vlevo. Na této obrazovce je jednak informace o umístění levého horního rohu Sudoku do levého horního rohu obrazovky a dále je na obrázku ukázka zvýraznění nalezené mřížky, které probíhá v rámci rozšířené reality.

Na obrázku 4.2 vpravo je ukázka obrazovky s výsledkem po pořízení snímku. Na tuto obrazovku vede i možnost *Načíst poslední pořízený snímek*. Snímek byl záměrně pořízený pod zkoseným úhlem a přesto byla všechna vstupní čísla rozpoznána správně. Tento výsledek je docílen i díky tomu, že zadání Sudoku obsahuje číslice, jejichž všechny čáry mají stejnou výraznou tloušťku a i díry uvnitř číslic, například u číslice 8, jsou dostatečně velké.

Další ukázkou je výsledek pořízení snímku Sudoku s ručně psanými číslicemi na obrázku 4.3 vlevo. V obrázku jsou červeně překresleny vstupní číslice, které byly načteny špatně a jsou v konfliktu. V tomto případě je možnost přepnout na panelu nahoře na možnost *Ruční zadání*, ukázka je na stejném obrázku vpravo. Na této obrazovce je vykreslená vlastní mřížka, kde jsou zobrazeny načtené číslice a zvýrazněná chyba. V obrázku je vidět, že ne všechny číslice byly rozpoznány, například všechny číslice 1, přičemž ta v pravém dolním rohu byla navíc zaměněna za číslici 4. Chybovost je dána například i tím, že číslice jsou různě zkosené a díry uvnitř číslic, hlavně u číslic 4, jsou příliš malé.



Obrázek 4.3: Vlevo je snímek s detekcí chyby, vpravo je ruční zadání s detekcí chyby.

V tomto momentě přichází na řadu ruční úprava, jejíž výsledek po vyřešení je na obrázku 4.4 vlevo. Po přepnutí na panelu na *Snímek* se výsledek řešení promítne také do pořízeného snímku, což lze vidět na stejném obrázku vpravo.

Na těchto snímcích lze také spatřit, že řešitel je schopný načíst i ručně psané číslice, pokud nejsou příliš deformované. Z celkového počtu dvaceti tří čísel byly dvě číslice zaměněny za jiné a čtyři číslice nebyly rozpoznány vůbec. Celkem je to špatná čtvrtina načtených číslic, ovšem při takto ručně napsaném zadání to považuji za obstojný výsledek.

Obrázek 4.4: Vlevo je opravené a vyřešené zadání, vpravo je řešení promítnuté ve snímku.

Kapitola 5

Testování aplikace

Tato kapitola je rozdělená na dvě části. První část je zaměřena na testování výkonu aplikace na několika různých zařízeních. Sledovaným parametrem byl čas trvání různých operací nutných pro vyřešení Sudoku. Druhá část se již věnuje testování na uživatelích a to jednak z pohledu, jak s aplikací zacházeli a poté jak aplikaci hodnotili.

5.1 Výkon aplikace

Měření výkonu jsem prováděl na zařízeních uvedených v tabulce 5.1. Zaměřil jsem se především na dobu provádění jednotlivých operací. Dobu operací jsem měřil odečítáním času v milisekundách. U každého zařízení jsou také v naměřených údajích uvedeny rozměry náhledu z kamery a rozměry pořízeného snímku. Všechny doby byly určovány průměrem z několika měření a poté zaokrouhleny na desetiny sekund, popřípadě desítky milisekund.

Prvním měřením bylo zjišťování počtu snímků za sekundu při samotném náhledu a poté při vykreslování zvýraznění mřížky v rámci rozšířené reality. Dalším měřením bylo zjištění doby nalezení mřížky v pořízeném snímku, doby přípravy mřížky, což zahrnuje převážně její perspektivní transformaci, dále doby rozpoznávání čísel pomocí knihovny tess-two včetně její inicializace a nakonec celkové doby zpracování snímku. Tato celková doba je uváděna z důvodu probíhajících operací mezi uvedenými hlavními výpočty. Další měření se zaměřovalo na určení doby vykreslení výsledku zpět do obrázku a to s využitím paměti pro uložení transformované mřížky a bez ní. Nejvýznamnější operací v tomto měření byla zpětná perspektivní transformace mřížky do snímku.

Speciálně prováděným měřením bylo také určení, jak dlouho trvá zjištění orientace mřížky. Posledním měřením byla doba řešení Sudoku pomocí samostatného hádání a poté s využitím eliminačních technik. Měření bylo prováděno na třech různých zadáních pro lepší porovnání obou metod. První zadání bylo specifikované jako lehké a obsahovalo 35 číslic v zadání a celé se dalo vyřešit technikou odhalených jedinců. Druhé zadání bylo specifikováno jako těžké a obsahovalo 24 číslic v zadání a k využití bylo potřeba technik odhalených a skrytých jedinců a také uzamčených kandidátů. Posledním zadáním byla prázdná mřížka.

5.1.1 Výsledky testování výkonu

Zaokrouhlené naměřené hodnoty jsou uvedeny v tabulce 5.2. Jak lze vidět u zařízení *ZTE* a *Alcatel*, ne vždy se podaří nastavit požadované rozlišení náhledu nebo snímku. Výsledné hodnoty jsou tak oproti jiným zkreslené a to obzvláště u zařízení *Alcatel*, jehož pořízený snímek má nízké rozlišení a i v rozpoznávání čísel pak častěji docházelo k chybám.

Značka	**Motorola**	**LG**	**Samsung**
Model	Defy+	Optimus One	Galaxy Young
Označení	MB526	P500	GT-S6310N
Obrazovka	3.7”	3.2”	3.27”
Rozlišení	480x854 px	320x480 px	320x480 px
Procesor	Cortex-A8	ARM 11	Cortex-A5
Frekvence	1 GHz	600 MHz	1 GHz
Počet jader	1	1	1
Paměť RAM	512 MB	512 MB	768 MB
Systém	CyanogenMod	Android	Android
Verze Android	4.4.2	2.3.3	4.1.2

Značka	**Sony**	**Alcatel**	**ZTE**
Model	Xperia Mini Pro	OneTouch	Blade V
Označení	SK17i	997D	Blade V
Obrazovka	3.0”	4.3”	4.0”
Rozlišení	320x480 px	480x800px	480x800px
Procesor	Scorpion	Cortex-A9	Snapdragon 200
Frekvence	1 GHz	1 GHz	1.2 GHz
Počet jader	1	2	4
Paměť RAM	512 MB	1 GB	1 GB
Systém	Android	Android	Android
Verze Android	4.1.2	4.0.4	4.1.2

Tabulka 5.1: Specifikace testovacích zařízení

Nejvýraznější rozdíly ve výkonu jsou zřetelné u zařízení *Motorola* a *LG*, jelikož tato zařízení byla z testovacích nejslabší. V počtu snímků za sekundu znamenalo samotné zvýraznění mřížky, které zahrnovalo pouze její nalezení a vykreslení, propad pouze na přibližně 3 snímky za sekundu u zařízení *LG*. Přesto jsem se rozhodl tuto formu rozšířené reality ponechat protože u podobné aplikace nepovažuji toto zpomalení za kritické.

Dalším bodem bylo zpětné vykreslení mřížky. Opět u *Motoroly* a *LG* znamenal rozdíl ve využití paměti pro uložení transformované mřížky několikasekundové urychlení. Proto jsem se rozhodl tohoto řešení využít, i když u jiných zařízení byl rozdíl méně zřetelný.

Důležitým zjištěním pro mě bylo, že nejdéle trvajícím prvkem v nalezení čísel je jejich rozpoznání pomocí knihovny tess-two. Čas v tomto případě zabírá nejen samotné rozpoznávání ale také inicializace této knihovny. Její rychlost lze spatřit také v čase zjišťování orientace mřížky. Jelikož mřížka může být natočena ve čtyřech různých směrech, knihovna tess-two čtyřikrát prováděla čtení celé této mřížky. Protože výsledná doba zjišťování orientace by navýšila ve všech případech celkový čas řešení několikanásobně, rozhodl jsem se ji ve výsledné aplikaci zavrhnout.

Posledním výsledkem je porovnání dob řešení Sudoku samotným hádáním a eliminací. Nejvýraznější rozdíl je opět vidět na zařízeních *Motorola* a *LG*, kdy obzvláště pro těžké zadání Sudoku znamenalo využití eliminačních metod mnohonásobné urychlení.

Značka	**Motorola**	**LG**	**Samsung**
Rozměry			
Náhledu	288x352 px	320x480 px	240x320 px
Snímku	960x1280 px	960x1280 px	960x1280 px
Průměrně FPS			
Bez zvýrazňování	15	15	14
Se zvýrazňováním	6	3	8
Nalezení číslic			
Nalezení mřížky	1,1 s	2,5 s	0,7 s
Příprava mřížky	0,7 s	1,4 s	0,4 s
Rozpoznání čísel	4,5 s	5,9 s	2,2 s
Celkem	6,5 s	10,1 s	3,4 s
Zjištění orientace	13 s	21 s	8 s
Řešení Sudoku			
Hádáním			
Lehké	30 ms	60 ms	60 ms
Těžké	2,7 s	3,1 s	0,9 s
Prázdné	1,7 s	3,2 s	0,7 s
Eliminací			
Lehké	10 ms	10 ms	10 ms
Těžké	20 ms	20 ms	20 ms
Prázdné	0,5 s	0,7 s	0,1 s

Značka	**Sony**	**Alcatel**	**ZTE**
Rozměry			
Náhledu	320x480 px	368x480 px	720x1280 px
Snímku	960x1280 px	480x640 px	1200x1600 px
Průměrně FPS			
Bez zvýrazňování	16	17	10
Se zvýrazňováním	8	8	2
Nalezení číslic			
Nalezení mřížky	0,6 s	0,1 s	0,2 s
Příprava mřížky	0,4 s	0,1 s	0,5 s
Rozpoznání čísel	3,2 s	1,8 s	3,5 s
Celkem	4,3 s	2,0 s	4,3 s
Zjištění orientace	8 s	5 s	9 s
Řešení Sudoku			
Hádáním			
Lehké	40 ms	20 ms	60 ms
Těžké	1,4 s	1,0 s	1,6 s
Prázdné	1,3 s	0,6 s	1,6 s
Eliminací			
Lehké	10 ms	10 ms	10 ms
Těžké	10 ms	10 ms	20 ms
Prázdné	0,2 s	0,1 s	0,2 s

Tabulka 5.2: Naměřené hodnoty v testování výkonu

5.2 Zpětná vazba od uživatelů

Zpětnou vazbu od uživatelů jsem se rozhodl zjišťovat dvěma způsoby rozvedenými dále. Prvním bylo pozorování uživatele při seznamování a práci s aplikací a druhým bylo sestavení krátkého dotazníku.

5.2.1 Pozorování uživatelů

Pro pozorování uživatelů jsem se jim rozhodl zadat dva jednoduché úkoly. Prvním bylo prohlédnout si aplikaci a vyzkoušet si ji a to bez toho, že by jim byly poskytnuty jakékoli další informace. V tomto případě jsem se zaměřil především na to, zda uživatelé sami objeví, jaké možnosti aplikace nabízí, jak těchto možností využijí a jak budou reagovat na dobu řešení, která dosahuje až několika sekund. Pro toto první vyzkoušení aplikace jim bylo nabídnuto Sudoku, které lze načíst, aniž by u něj docházelo k častým záměnám číslic.

Po tomto seznámení jsem uživatelům vysvětlil zbývající možnosti, které popřípadě sami neobjevili a popsal jsem, jak bylo ovládání aplikace zamýšleno. Pro druhý úkol jim bylo nabídnuto ručně psané zadání Sudoku, ve kterém často docházelo k různým záměnám při rozpoznávání a cílem bylo, aby uživatelé toto zadání nasnímali, opravili chyby, které se objeví a nechali znovu proběhnout řešení. Při tomto jsem sledoval především, jak se změnila rychlost provádění potřebných úkonů a jak s touto změnou poté uživatelé reagovali na celkovou dobu řešení Sudoku.

Výsledky zjištěné při pozorování

Ze zjištěných skutečností jsem se rozhodl vybrat ty, které mi přišly nejzajímavější. V prvním úkolu to znamenalo, že několik uživatelů si možnosti přepnout na ruční zadávání po pořízení snímku všimlo až po upozornění. Toto ruční zadávání jsem zamýšlel tak, že se nejprve klikne na tlačítko s číslicí a ta se poté vyplňuje do mřížky klikáním na políčka. Několik uživatelů však mělo opačný nápad a nejprve klikali na políčka a čekali, že pak jim bude nabídnuto, jakou číslici chtějí doplnit.

Dalším problémem bylo, že jeden uživatel zaměnil funkčnost tlačítek *0* a *Smazat*. Tlačítko *0* slouží k vyprázdnění políčka, zatímco tlačítko *Smazat* k vyprázdnění všech políček, která byla doplněna řešením Sudoku. Uživatel označil toto pojmenování za nelogické. Stejný uživatel se pokoušel pořídit snímek s telefonem orientovaným na šířku. Toto nebylo zamýšleno i vzhledem k umístění nápisu *Levý horní*, který se nachází v levém horním rohu obrazovky při orientaci telefonu na výšku.

Posledním zjištěným faktem bylo, že při tomto zkoušení žádnému uživateli nevadilo několikasekundové zdržení při načítání zadání ze Sudoku a naopak označili řešení za rychlé.

Ve druhém úkolu jsem již při ovládání aplikace nepozoroval výrazné váhání až na uživatele, kteří by raději klikali první na políčko a poté volili číslo. Přesto všichni uživatelé zvládli všechny úkony včetně pořízení snímku, opravy zadání a nového vyřešení během jedné až dvou minut. Sám jsem se takto dostal při nejrychlejším provedení na asi třicet sekund, ale tento rozdíl považuji za daný faktem, že jsem již znal zadání Sudoku a nemusel jsem číslice kontrolovat ve skutečném zadání a poté je přepisovat do telefonu.

Důležitějším zjištěním při druhém úkolu pro mě bylo, že při čekání na výsledek řešení již uživatelé začínali být mírně netrpěliví, ačkoli původně hodnotili rychlost řešení jako velmi dobrou. I přes toto zjištění uživatelé nepovažovali zdržení za příliš nepříjemné, i když zrychlení by uvítali. Upozorňuji, že se jednalo o zařízení, kde celková doba řešení byla přibližně tři až čtyři sekundy.

5.2.2 Dotazníková metoda

Do dotazníku jsem zvolil několik variant, které uživatelé mohli oznámkovat a poté jsem je také vyzval, že mohou doplnit libovolné slovní hodnocení. V prvním případě to bylo speciálně zaměřené hodnocení na ovládání aplikace a v druhém případě libovolné další připomínky. Známkování bylo dáno na stupnici od 1 jako nejlepší po 5 jako nejhorší. Uživatelé mohli známkovat ovládání ve smyslu pohodlnosti nebo intuitivnosti, dále přesnost, jak byli spokojeni s výsledkem načítání a to i včetně ručně psaného zadání a poté známkovali rychlost, jak byli spokojeni s celkovou dobou řešení. Pro tuto poslední variantu byli uživatelé vyzvání také k doplnění průměrné doby řešení v sekundách.

Výsledky dotazníku

Výsledky známkování lze nalézt v tabulce 5.3. Výsledky slovního hodnocení jsou shrnuty dále a opět jsem se zaměřil na nejzajímavější postřehy z mého pohledu.

Ovládání	Přesnost	Rychlost	Průměrná rychlost
1	2	1	3 s
1	4	2	5 s
2	2	2	2 s
1	5	5	-
1	2	1	1 s
1	5	5	-
3	1	2	3 s
2	1	3	6 s
2	1	2	4 s

Tabulka 5.3: Známkování aplikace z dotazníku

Ve známkování se projevilo to, že dvěma uživatelům, kteří přesnost a rychlost hodnotili známkou 5 a neuvedli průměrnou rychlost, se nepodařilo Sudoku pořídit tak, aby bylo nalezeno a vyřešeno. U jednoho uživatele to bylo způsobeno kamerou bez automatického zaostřování a pořízený snímek byl vždy příliš rozmazaný. Uživatel, který hodnotil přesnost známkou 4, si podle jeho hodnocení nevšiml možnosti přepnout na ruční zadávání a upravit špatně načtené číslice.

Při pohledu na hodnocení rychlosti jsem byl překvapen, že i čtyř nebo pěti sekundové zdržení dva uživatelé oznámkovali 2. Na současné poměry, kdy je všude vyžadována vysoká rychlost, mi toto známkování připadá mírné.

Z hodnocení ovládání jsem zjistil, že se objevili další uživatelé, kteří by dali přednost klikání nejprve na políčko a poté volbě čísla oproti stávající možnosti. Objevili se také někteří uživatelé, kteří by ocenili možnost focení při orientaci telefonu na šířku a tím pádem také možnost vyfotit Sudoku libovolně otočené.

Z dalších připomínek se objevil fakt, že jednomu uživateli aplikace spadla při přijetí SMS zprávy. Jako nápady na vylepšení se objevilo například procházení historie pořízených snímků Sudoku a ukládat obrázky s vyřešenými výsledky. Poté například přepínání mezi snímkem a ručním řešením tahem obrazovky místo pouhého klikání na panely. A jako poslední zajímavou možnost uvádím, že jeden uživatel by rád označil mřížku ručně, pokud není nalezena automaticky a poté by zkusil znovu nalézt zadání Sudoku.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout aplikaci, která umožní pořídit snímek Sudoku pomocí kamery mobilního zařízení se systémem Android a následně toto Sudoku vyřeší. Při návrhu aplikace jsem se seznámil s vývojem pro systém Android a také s již existujícími aplikacemi, které Sudoku umí nasnímat a vyřešit. Samotný problém vyhledání mřížky ve snímku mě naučil možnostem zpracování obrazu obecně a také konkrétně na systému Android. Dalším problémem, který jsem řešil, bylo rozpoznávání textu v obraze a poslední probíranou záležitostí bylo řešení samotného Sudoku.

Vlastní aplikaci jsem navrhl a sestavil tak, že výsledné řešení zakresluje zpět do pořízeného snímku, protože s touto variantou jsem se u existujících aplikací nesetkal. Vyzkoušel jsem také zobrazování rozšířené reality, ale již pouhé nalezení a zvýraznění mřížky zpomalilo běh aplikace obzvláště na pomalejších zařízeních. Samotné nalezení mřížky a rozpoznání číslic je prováděno až po pořízení snímku a to s využitím knihoven pro zpracování obrazu a rozpoznávání znaků, které celou práci usnadnily.

S přesností rozpoznávání a rychlostí aplikace byla spokojena také vetší část uživatelů, kteří aplikaci vyzkoušeli. Překvapila mě pozitivní reakce právě na rychlost aplikace vzhledem k dnešní době, kdy je vše často vyžadováno okamžitě, protože celé řešení znamenalo až několikasekundové zdržení, které uživatelé přesto považovali za chvalitebné.

Z hodnocení ovládání aplikace již vyplývají možnosti rozšíření projektu z pohledu výsledné aplikace. Někteří uživatelé by uvítali možnost ručního zadávání číslic odlišným způsobem, než jsem se rozhodl využít. Toto vidím jako první variantu rozšíření, která by umožňovala způsob zadávání zvolit. Dalším nápadem bylo umožnit ručně označit mřížku Sudoku, pokud není automaticky nalezena. Poslední variantou, která se mi jeví jako vhodné rozšíření, je ukládání historie pořízených snímků a jejich řešení.

Další možností, jak pokračovat v projektu, je také zaměření se na rozpoznávání číslic, které bylo z časového hlediska nejnáročnější operací řešení. Momentálně jsou číslice rozpoznávány pomocí knihovny, avšak celá knihovna je komplexní nástroj pro rozpoznávání znaků v libovolném textu. Také zabírá výrazné místo v paměti telefonu. Pokud uvážím, že by stačilo rozpoznávat pouze jednotlivé číslice a že základní nástroje pro sestavení systému pro rozpoznávání znaků nabízí i knihovny pro zpracování obrazu, jeví se právě vlastní systém rozpoznávání znaků jako další zajímavá možnost pokračování v projektu.

Literatura

[1] Allen, G.: *Android 4: Průvodce programováním mobilních aplikací.* Brno: Computer Press, 2013, 656 s., ISBN 978-80-251-3782-6.

[2] Bradski, G. R.; Kaehler, A.: *Learning OpenCV: Computer Vision with the OpenCV Library.* Sebastopol: O'Reilly, 2008, 555 s., ISBN 978-0-596-51613-0.

[3] Burnette, E.: *Hello, Android: Introducing Google's Mobile Development Platform.* Raleigh: The Pragmatic Programmers, třetí vydání, 2010, 293 s., ISBN 978-1-93435-656-2.

[4] Cohen, Y.: *Cross platform computer vision optimization.* [online]. 2012-04-18 [cit. 2014-4-21].
URL http://www.slideshare.net/DSPIP/cross-platform-computer-vision-optimization-12584172

[5] Eikvil, L.: *OCR - Optical Character Recognition.* Oslo: Norsk Regnesentral, 1993, ISBN 9788253903712.

[6] Maji, A. K.; Roy, S.; Pal, R. K.: A Novel Algorithmic approach for solving Sudoku puzzle in Guessed Free Manner. *European Academic Research*, ročník 1, č. 6, září 2013, ISSN 2286-4822.
URL http://www.euacademic.org/UploadArticle/76.pdf

[7] OpenCV Development Team: *OpenCV4Android SDK.* [online]. 2014-04-21 [cit. 2014-4-23].
URL http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html

[8] Santos-García, G.; Palomino, M.: Solving Sudoku Puzzles with Rewriting Rules. *Electronic Notes in Theoretical Computer Science*, ročník 176, č. 4, červenec 2007, ISSN 1571-0661.
URL http://www.sciencedirect.com/science/article/pii/S1571066107005142

[9] Shapiro, L. G.; Stockman, G. C.: *Computer vision.* New Jersey: Prentice-Hall, 2001, 580 s., ISBN 0-13-030796-3.

[10] Shih, F. Y.: *Image Processing and Mathematical Morphology: Fundamentals and Applications.* Boca Raton: CRC Press, 2009, 415 s., ISBN 978-1-4200-8943-1.

[11] Žára, J.: *Moderní počítačová grafika.* Brno: Computer Press, 2004, 609 s., ISBN 80-251-0454-0.

Příloha A

Struktura zdrojových souborů