UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Libor Machovec

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Rozpoznání a složení Rubikovy kostky

Libor Machovec

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Libor Machovec**
Osobní číslo:	**I14135**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Rozpoznání a složení Rubikovy kostky**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce bude aplikace, která nasnímá foroaparátem reálnou Rubikovu kostku a zobrazí postup jejího složení.

V teoretické části se požaduje porovnat několik podobných referenčních aplikací na Google Play a stanovit jejich výhody a nevýhody. Navrhnout vlastní řešení, které nebude mít vetšinu nevýhod referenčních aplikací.

V praktické části se požaduje realizovat softwarovou aplikaci pro operační systém Android, která nasnímá otačející Rubikovu kostku a navrhne postup jejího složení.

Rozsah grafických prací:	**Podle potřeby**
Rozsah pracovní zprávy:	**Nejméně 40 stran textu**
Forma zpracování bakalářské práce:	**tištěná**

Seznam odborné literatury:

[1] ABLESON, W, Robi SEN a Chris KING. Android in action. 2nd ed. Greenwich, Conn.: Manning, c2011, xxix, 562 p. ISBN 19-351-8272-2.

[2] MURPHY, Mark L. Android 2: průvodce programováním mobilních aplikací. Vyd. 1. Brno: Computer Press, 2011, 375 s. ISBN 978-80-251-3194-7.

[3]BRADSKI, Gary R. Learning OpenCV. Sebastopol: O'Reilly, c2008, xvii, 555 s. ISBN 978-0-596-51613-0.

Vedoucí bakalářské práce:	**Ing. Karel Šimerda** Katedra softwarových technologií
Datum zadání bakalářské práce:	**31. října 2016**
Termín odevzdání bakalářské práce:	**12. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 10. 5. 2017

podpis autora
Libor Machovec

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu práce Ing. Karlu Šimerdovi za odborné vedení v průběhu psaní této bakalářské práce. Dále bych poděkoval rodině za trpělivost a podporu při celém studiu na vysoké škole. V neposlední řadě bych poděkoval kolegům studentům, kteří mi vždy ochotně odpověděli na dotazy.

ANOTACE

Tato bakalářská práce popisuje načítání a algoritmizaci hlavolamu Rubikova kostka (dále jen „hlavolam“), které jsem použil při tvorbě aplikace pro mobilní zařízení. V první části dokumentu je popsán vlastní hlavolam, digitální zpracování obrazu při snímání hlavolamu, systém Android a referenční aplikace, které byly v této souvislosti testovány. V druhé časti je zdokumentován konkrétní návrh aplikace, která umí načítat a určit postup pro složení hlavolamu. V poslední části jsou navrhnuta vylepšení vytvořené aplikace a zhodnocení očekávaných funkčností a uživatelského rozhraní.

KLÍČOVÁ SLOVA

Android, Rubikova kostka, barva, digitální obraz, aplikace, hlavolam

TITLE

Detecting and passing the Rubik's Cube.

ANNOTATION

This bachelor thesis describes the loading and algorithmization of the puzzle Rubik's Cube ("puzzle"), which I used to create applications for mobile devices. The first part of the document describes a custom teaser, brain imaging digital imaging, Android, and reference applications that have been tested in this context. In the second part is documented a specific application design, which allows to record and determine the procedure for the composition of the puzzle. The last part proposes improvements to the created application and evaluation of the expected features and user interface.

KEYWORDS

Android, Rubik's Cube, Color, Digital Image, Applications, Puzzle

Obsah

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

HW	Hardware
SW	Software
IT	Informační Technologie
MLL	Machine Learning Library
RGB	Red Green Blue
CMYK	Cyan Magenta Yellow Key
HSV	Hue Saturation Value
HSL	Hue Saturati Lightness

0 ÚVOD

Rubikova kostka (snad nejpopulárnější hlavolam světa) si díky své náročnosti řešení a oblíbenosti široké veřejnosti zasloužila pozornost vývojářů aplikací a tím naprogramování nejrůznějších simulátorů a návodů na řešení. Ne však všechny aplikace skloubí rychlé načtení Rubikovy kostky (dále jen „hlavolam“), jednoduchou editaci hlavolamu a následnou prezentaci kroků ke složení hlavolamu. Díky moderní době a „chytrým“ telefonům, které mají již dostatečný výkon, lze naprogramovat detekční aplikaci přímo do mobilního zařízení, které uživatel nosí stále u sebe. Dnešní telefon díky přítomné kameře pohodlně nasnímá kvalitní obraz a pomocí dotykového displeje jednoduše umožní editovat obsah.

Tato práce se primárně zaměřuje na problematiku načtení, implementace a algoritmizace Rubikovy kostky na platformě Android. Navrhuje aplikaci, která je schopna nalézt postup složení Rubikovy kostky.

První teoretická část je věnována popisu problematiky Rubikovy kostky. Dále se zabývá digitálním zpracování obrazu v mobilním telefonu. Popisuje také některé části operačního systému Android. Na závěr se věnuje testování několika referenčních aplikací z Google play. Testy jsou zaměřené na funkčnost a snadnost obsluhy aplikace.

Druhá část navrhuje možnou implementaci aplikace pro platformu Android. Mé návrhy reagují na nedostatky z testování aplikací. Jsou zde popsány důležité třídy pro práci s obrazem a uložení Rubikovy kostky. Dále je naprogramována aplikace včetně designu uživatelského prostředí.

Konvence a výrazy použité v dokumentu.

Tučně Důležité informace, názvy, zkratky, odkazy na obrázky, tabulky a pojmy jsou zobrazeny v textu tučně.

`Smaz()` Třídy, metody, funkce a části kódu jsou v textu označeny neproporcionální fontem.

1 TEORETICKÁ ČÁST

1.1 Hlavolam Rubikova kostka

Rubikova kostka je mechanický hlavolam v nejčastější a zároveň původní podobě tvaru krychle. Tato krychle je tvořena z dílčích krychliček, které mají z vnějších (viditelných) stran barvu dle stěny krychle hlavolamu, na které se nachází. Hlavolam celkem obsahuje 3×3×3 jednotlivých krychliček, které podle umístění na kostce obsahují odpovídající počet barevných stěn. Tedy od jedné (středová kostička stěny) do maximálně tří pro rohovou kostičku stěny.

Z celkových 27 krychliček je 1 středová, ta není vidět a neobsahuje žádnou barvu, ale obsahuje mechanizmus otáčení jednotlivých stěn. Středových kostiček stěn obsahuje kostka 6, tyto kostičky jsou vždy na stejném místě vůči středu a ostatním středům stěn a obsahují pouze jednu barevnou stěnu. Dvanáct hranových kostiček spojují vždy dvě sousední stěny a obsahují dvě rozdílné sousední barvy. Nakonec 8 rohových kostiček obsahujících 3 rozdílné barvy.

1.1.1 Historie hlavolamu

V Budapešti roku 1974 mladý profesor architektury Erno Rubik vytvořil těleso, které by nemělo být možné. Krychle, ve které je možné otáčet všemi stěnami všemi směry, aniž by se krychle rozsypala na dílčí krychličky. Barevným polepením jednotlivých stěn krychliček vznikl hlavolam „Magic Cube“. Samotnému tvůrci trvalo více než měsíc, než tento hlavolam dokázal složit. Původní důvod vytvoření takového tělesa byl demonstrovat studentům Erna Rubika vztahy v prostoru. Nikdy by ho nenapadlo, že se tento hlavolam jednou stane nejprodávanější hračkou na světě. Po uvedení na trh se přejmenovala na „Rubik’s Cube“ neboli Rubikova kostka. [1]

Po více než 40 letech je tento hlavolam stále populární a vyrábí se v nejrůznějších variantách tvarů a počtů krychliček. Původní varianta 3×3×3 se stále vyvíjí s novými mechanizmy otáčení pro snadnější a přesnější skládání. Po celém světě se pořádají soutěže ve skládání kostek různých velikostí a tvarů, a to na čas nebo třeba po slepu.

1.1.2 Princip hlavolamu

Originální kostku 3×3×3 lze rozložit v 43 252 003 274 489 856 000 (43 triliónů) různých permutací [2]. Existuje několik postupů, jak hlavolam složit od základního – vrstva po vrstvě,

kdy stačí použít několik málo algoritmů, které se na vhodném místě opakují, dokud se nedocílí složení kostky. Další postupy se blíží pokročilým metodám s minimalizací počtu kroků a tím zrychlení skládání hlavolamu. Tyto postupy využívají poznání více algoritmů, které jsou užity při skládání kostek na soutěžích. Pomocí výpočetní techniky bylo dokázáno, že každou z permutací lze složit přibližně 20 kroky.

Cíl hlavolamu je přemístit nebo otočit rohové a hranové kostičky tak, aby středem stěny a jejich barvou na téže stěně měly stejnou barvou. Toho lze docílit otáčením jednotlivých stěn kolem své osy v určitém pořadí kroků.

Popis kroků

Pro popis algoritmů v textové podobě se zavedl popis všech možných pohybu kostky. Značení stěny, která se má otáčet vychází z anglického názvu:

a) **F – Front,** přední stěna,
b) **B – Back.** zadní stěna,
c) **L – Left,** levá stěna,
d) **R – Right,** pravá stěna,
e) **U – Up,** horní stěna,
f) **D – Down,** spodní stěna.

Pokud je zkratka uvedena samostatně, znamená to pohyb o 90° příslušné stěny po směru hodinových ručiček při pohledu na stěnu. Pokud je doplněna zkratka o malé i nebo apostrof například **Ri, R',** znamená to, že se stěna bude otáčet o 90° proti směru hodinových ručiček. Doplněním dvojky před zkratku pohybu znázorňuje pohyb o 180° příslušné stěny například **2B**.

1.1.3 Metody skládání

Existuje mnoho metod a postupů, jak vyřešit hlavolam. Metody se zaměřují na snadnost naučení, rychlost složení nebo minimalizaci počtu kroků. Nejjednodušší metody obsahují minimum potřebných algoritmů pro složení. Složitější metody jsou založené na předvídání a plánování několika kroků dopředu. Tyto metody vyžadují více algoritmů na naučení a správné použití určitého algoritmu.

Metoda vrstva po vrstvě (layer by layer)

Metoda vrstva po vrstvě je nejrozšířenější metoda začátečníků. Metoda postupně složí nejprve kříž na první vrstvě. Dalším krokem je složení rohů první vrstvy. Po první vrstvě se složí druhá vrstva přesunutím odpovídajících hran na svojí pozici. Skládání poslední vrstvy se zahajuje opět orientací hran vrstvy a vytvoření kříže. Posledním krokem je postupné umístění a otáčení rohů poslední vrstvy.

Metoda Fridrich

První krok metody je stejný s předešlou opět se vytvoří kříž. V druhé fázi se ale skládají rohy první vrstvy spolu s druhou vrstvou. Na poslední vrstvě se otočí všechny kostičky barvou poslední vrstvy na poslední stranu. Následně dochází k prohazování kostiček této vrstvy. Tato metoda je mnohem rychlejší a potřebuje v průměru méně kroků, ale vyžaduje znalost více delších algoritmů.

Ostatní metody

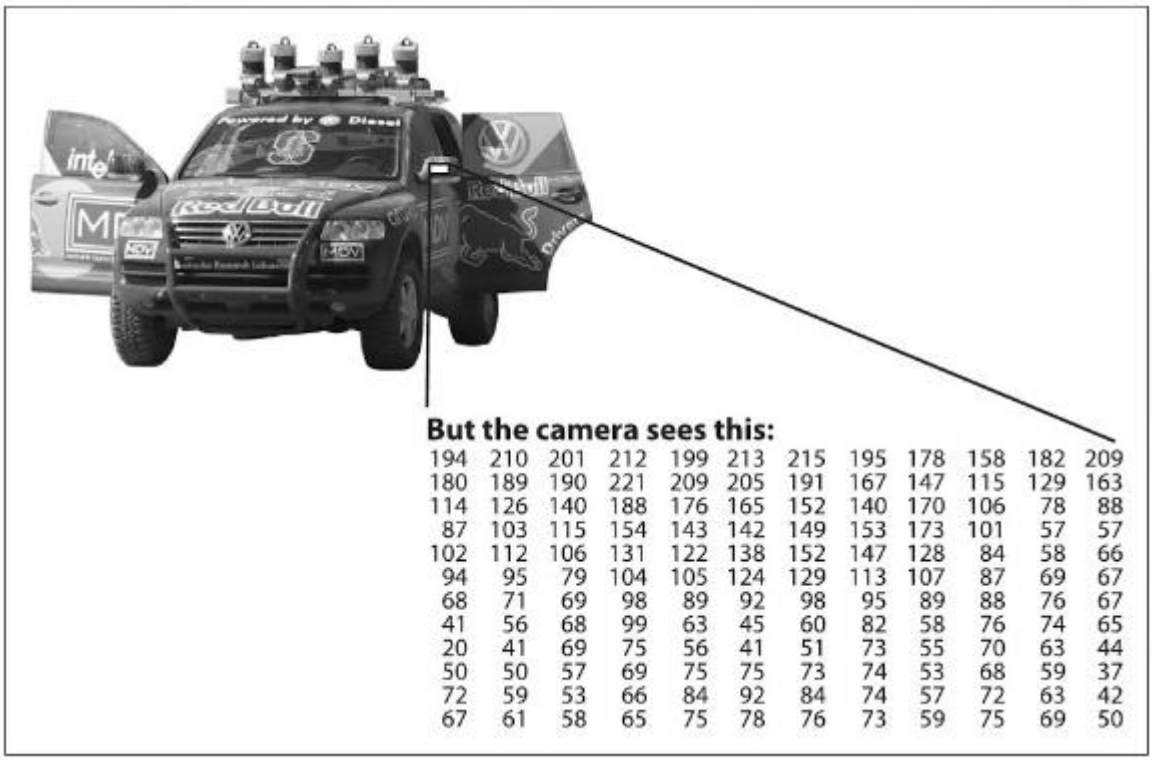
Existuje nespočet dalších metod zaměřených na nízký počet kroků nebo k zapamatování a složení kostky poslepu. Samozřejmostí je kombinace algoritmů ze všech technik, která umožňuje zkušeným efektivněji skládat Rubikovu kostku.

1.2 Digitální zpracování obrazu

Pro načtení kostky nebo barev stěn je potřeba zpracovat nasnímaný obrázek nebo video v digitální podobě. Tato kapitola popisuje digitální zpracování obrazu obecně, následně využitou knihovnu **OpenCV** a na závěr popisuje barvu a barevné modely.

Digitální zpracování obrazu je rychle se rozvíjející moderní disciplína v **IT**. Výsledky zpracování obrazu jsou dnes běžně používaný k úpravám, kompresím obrazů, rozpoznání objektů a textů. Dostatečný výkon a technologická úroveň dnešních výpočetních systémů pomohla proniknout těmto technologiím i do spotřební elektroniky, jako je například mobilní telefon. [3]

Pokud se člověk podívá na fotografii nebo video hned dokáže popsat, co vidí nebo co se děje ve videu či fotografii, ale jak je znázorněno na **obrázku 1** počítač vidí pouze matici hodnot.



Obrázek 1 – Co vidí počítač na obrázku – převzato z [4]

Proto vznikly techniky počítačového vidění, které umožní počítačům do jisté míry porozumět tomu, co je na obrázku a na základě toho provést určitou akci. Například při focení lidí za

pomoci chytrého telefonu lze u některých zařízení spustit automatické sejmutí snímku, pokud se focené osoby usmívají nebo automaticky ostřit obličeje.

1.2.1 Knihovna OpenCV

Pro jednoduchou práci s detekcí a zpracováním digitálního obrazu je dobré využít vhodnou knihovnu. **OpenCV** je volně šiřitelná knihovna počítačového vidění pod licencí open-source[1]. Knihovna je napsaná v jazyce **C** a **C++** a funguje na platformách **Linux, Windows** a **Mac OS**. Knihovna **OpenCV** je navržena s důrazem na výpočetní efektivitu a zpracování v reálném čase. Umožňuje běh na více jádrových procesorech. [4]

Jedním z cílů knihovny je poskytnout jednoduše použitelnou infrastrukturu počítačového vidění k rychlému vytvoření sofistikovaných aplikací počítačového vidění. Knihovna obsahuje více než 500 funkcí, které pokrývají nejrůznější oblasti počítačového vidění, kontroly kvality výroby, medicíny, zabezpečení, uživatelského rozhraní, kalibrace kamer, stereo kamer, a robotů. [4]

Na **obrázku 2** je vidět, že v současné době se tato knihovna dělí na několik modulů. Protože počítačové vidění je úzce spjato se strojovým učením, **OpenCv** také obsahuje všestranně použitelnou knihovnu **Machine Learning Library**[2] (**MLL**). [4]

Základní dělení **OpenCV** na moduly:

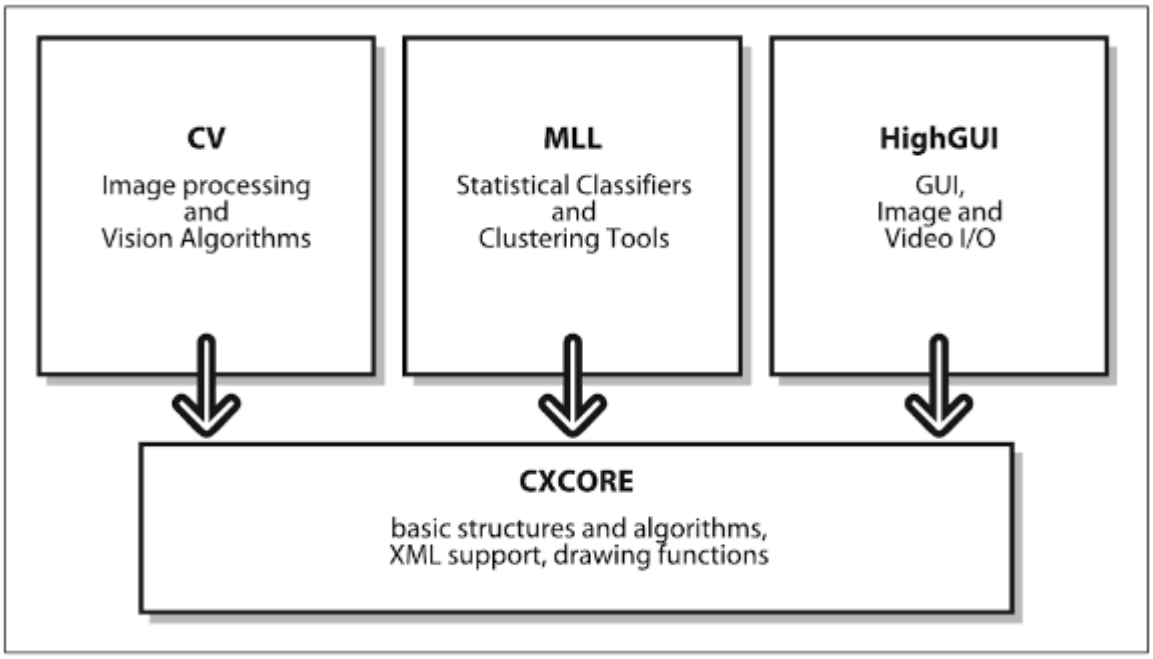
a) **CV** – moduly zajišťující zpracování obrazu a základní zobrazovací algoritmy.

b) **MML** – obsahuje nástroje strojového učení.

c) **HighGui** – obsahuje komponenty pro práci s **GUI** a komponenty pro práci se vstupně výstupními operacemi nad obrazem a videem.

d) **CXCore** – jádro umožňující zpracovávat a pracovat se základními strukturami a algoritmy, přidává podporu XML a vykreslovací funkce.

[1] Open-source je počítačový software s otevřeným zdrojovým kódem.
[2] Machine Learning Library knihovna strojového učení



Obrázek 2 – Základní struktura knihovny OpenCV – převzato z [4]

1.2.2 Barva v digitálním obrazu

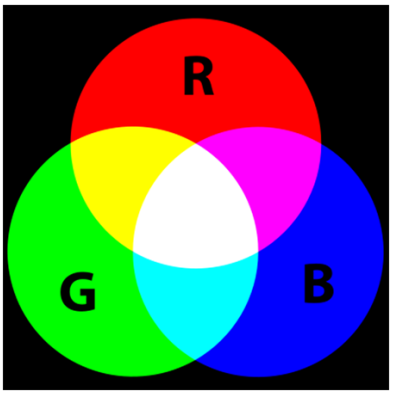
Pro načtení hlavolamu hraje hlavní roli barva. Rozpoznáním barvy kostičky na snímku lze určit, o kterou kostičku se jedná.

Digitální obraz se skládá z jednotlivých bodů určité barvy. Tyto body mají označení pixel[3]. Každý pixel nese informaci o barvě a jasu bodu. Každá barva je tvořena ze tří základních **červená, zelená a modrá**. Sčítáním nebo odečítáním jednotlivých složek lze zobrazit všechny barvy spektra. Tyto informace se uchovávají v několika modelech, které budou popsány níže.

[3] Pixel (picture element) je nejmenší bezrozměrná obrazová jednotka počítačové grafiky

Model RGB

Model **RGB** je nejčastěji používaný aditivní[4] model, kdy se skládáním červené, zelené a modré barvy na černém podkladu vytvoří všechny možné odstíny barev, viz **obrázek 3**. Složením všech barev se vytvoří bílá.



Obrázek 3 – Barevný model RGB zdroj images.google.com

Protože se jedná o sčítání jednotlivých barev, toho se využívá ve většině zobrazovacích zařízení, jako jsou monitory nebo displeje. V takových zařízeních je každý pixel tvořen třemi elementy, konkrétně je to červeným, zeleným a modrým elementem. [5]

Jednotlivé složky se v počítačích dají vyjádřit v desítkové soustavě jako číslo od 0 do 255 nebo v šestnáctkové soustavě jako číslo od 0 do FF. Běžný zápis v programovacím jazyce může vypadat takto `rgb(0,255,255)` nebo `#00FFFF`. [5]

[4] Aditivní barvy se skládají sčítáním

Model CMYK

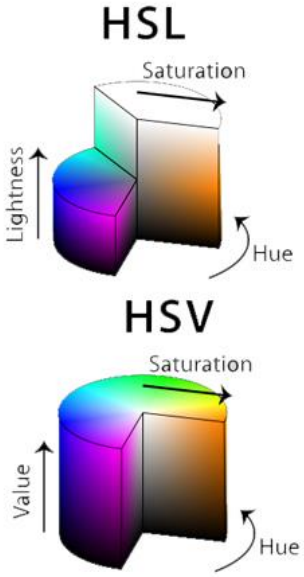
Model **CMYK** je subtraktivní[5], je tedy opakem **RBG** a využívá bílého podkladu pro zobrazení všech potřebných barev. Tento model se využívá při tisku, kde se složením všech barev dá zobrazit černá viz **obrázek 4**. [5]

Obrázek 4 – Barevný model CMYK převzato z [5]

[5] Subtraktivní barvy se skládají odečítáním

Modely HSV a HSL

Modely **HSV** a **HSL** jsou zaměřeny na intuitivní vnímání barev. Popisují barvu odstínem **H** (Hue), sytostí **S** (Saturation) dále u modelu **HSV** hodnotou **V** (Value) a u modelu **HSL** světelností **L** (Lightness), viz **obrázek 5**. [5]



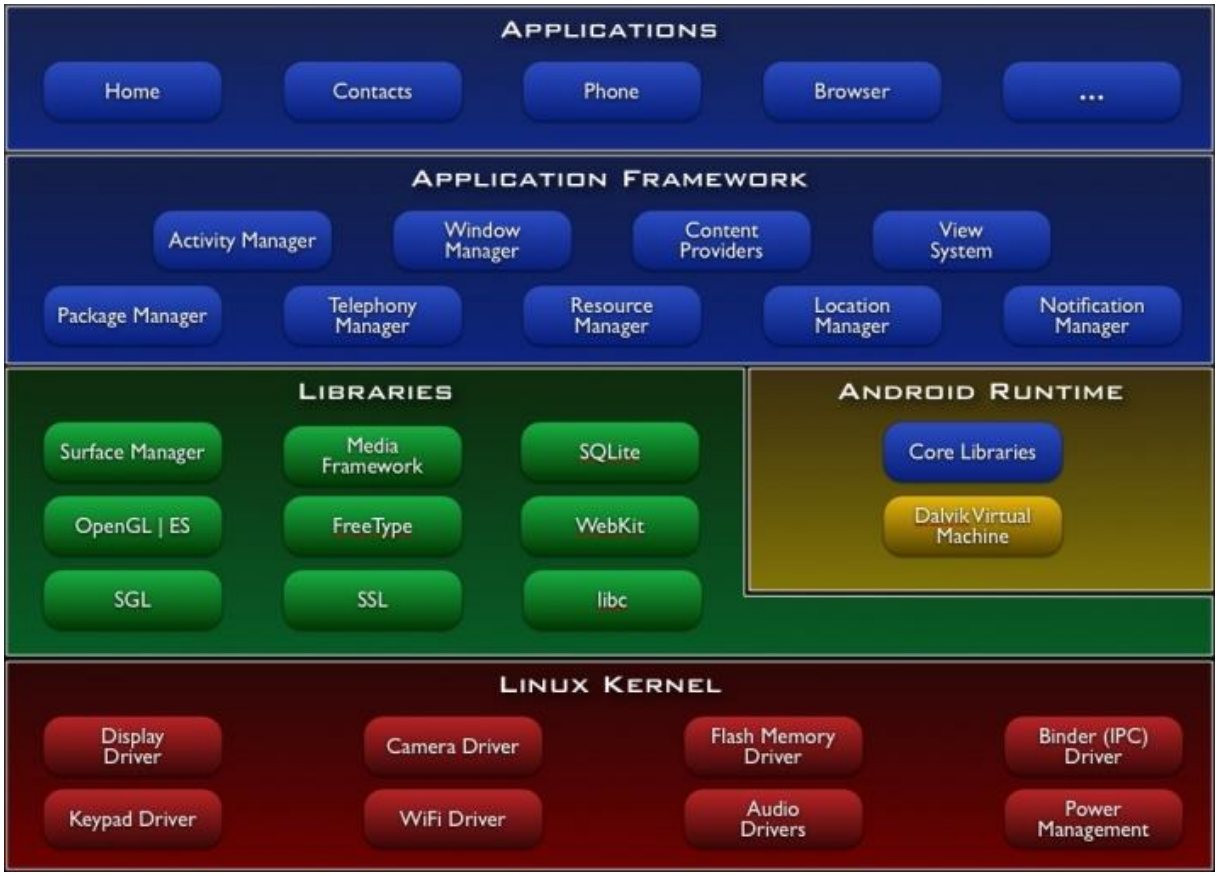
Obrázek 5 – Modely HSV a HSB zdroj images.google.com

1.3 Operační systém Android

Android je open-source[6] platforma určená primárně pro mobilní zařízení na bázi **Linuxu**. **Android** ve svých modifikacích obsluhuje mobilní telefony, tablety, fotoaparáty, hodinky, televize, navigace a jiná chytrá zařízení. Systém je vyvíjen organizací **Open Handset Alliance**, jejíž součástí jsou gigantické firmy jako **Google, HTC, Intel, NVIDIA, Qualcomm, Samsung** a další. Jako jeden z mála operačních systémů podporuje více platforem, ale to je zároveň nevýhoda v podobě slabší optimalizace. [6]

1.3.1 Architektura operačního systému Android

Architektura zobrazena na **obrázku 6** bude popsána od nejnižší vrstvy.



Obrázek 6 – Architektura Android zdroj developer.android.com

[6] Open-source je počítačový software s otevřeným zdrojovým kódem.

Linux Kernel

Nejnižší vrstvou architektury je jádro upraveného operačního systému **Linux**. Tento systém byl upraven pro potřeby mobilních zařízení. **Jádro přímo komunikuje s hardwarem zařízení**. Toto řešení úplně odděluje vyšší softwarové vrstvy od hardwaru. Jádro zabezpečuje správu paměti, správu procesů, ovladače a síťovou komunikaci. Na této úrovni je také zajištěno zabezpečení systému, správa napájení, vstupně-výstupní operace nebo základní grafika. **Aplikace a služby** jsou spuštěny v oddělených procesech pro zajištění komunikace mezi nimi slouží **Binder**. Tento modul řídí komunikaci mezi procesy a předává informace pomocí balíčků. **Binder** mapuje a počítá veškerou komunikaci a zabezpečuje přístupy to zajistí vysledování nebo zrušení některých komunikací. [6]

Libraries

Vrstva nativních knihoven napsaných v jazyce **C** a **C++**. Knihovny tvoří mezivrstvu mezi jádrem a vyššími vrstvami. Modul **Surface Manager** zabezpečuje funkcionalitu multidotykového displeje a směřuje grafický výstup z více aplikací do souvislého datového toku, který je předán grafické vykreslovací paměti. **Webkit** vykresluje a zobrazuje webové stránky. Grafické knihovny s podporou **OpenGL** se starající o vykreslení 2D a 3D grafiky. [6]

Android Runtime

Sada základních knihoven pro virtuální stroje **Dalvik Virtual Machine**. Tyto stroje jsou nutné pro běh **Java** aplikací. **Dalvik Virtual Machine** je obdoba **Java Virtual Machine** běžící na klasických počítačích. Při překladu aplikace napsané v jazyce **Java** kompilátor přeloží zdrojové kódy do binárních souborů. Tyto soubory spolu s jinými zdroji, které aplikace využívá jako obrázky, jsou nástrojem **DX** transformovány do jednoho souboru formátu **DEX**. Virtuální stroj následně začne tento soubor vykonávat. [6]

Application Framework

Obsahuje opakovaně použitelný software jako ovládací prvky, ikony a podobně. Celý framework je napsán v jazyce **Java**. Jedná se o nejdůležitější vrstvu pro vývojáře aplikací, protože poskytuje aplikacím základní služby systému. [6]

Package Manager modul spravující balíčky je databáze udržující aktuální seznam všech nainstalovaných aplikací. Vizuálně obrazem správce je plocha zařízení, kde každá ikona aplikace reprezentuje jeden balíček. [6]

Window Manager spravuje okna tvořící aplikaci. Většina aplikací využívá dvě a více oken současně. Dialogové nabídky jsou samostatná okna vyvolaná aplikací. [6]

View System spravuje paletu společných prvků grafického uživatelského rozhraní například ikony, tlačítka, editory textů a další. [6]

Application Framework obsahuje i modul **Activity Manager** sledující životní cyklus aktivity. [6]

Aplikace

Nejvyšší vrstva architektury systému **Android** jsou samotné aplikace jako navigace, seznam kontaktů, zprávy, email, webový prohlížeč a jiné. Každá aplikace pro přístup k některým komponentám a údajům potřebuje potřebné oprávnění. Toto zabezpečení neumožní aplikaci bez vědomí uživatele pořizovat snímky, posílat SMS, vytáčet telefonní hovorů a podobně. Veškerá oprávnění musí být deklarována v **manifestu**. [6]

Manifest je soubor **AndroidManifest.xml**, který popisuje celou aplikaci. Každá aplikace musí obsahovat tento soubor a veškeré prostředky, které bude aplikace využívat musí být zapsány. [6]

1.3.2 Historie operačního systému Android

Společnost **Android Inc**. založili čtyři zakladatelé v roce 2003. V roce 2005 odkoupil tuto společnost **Google**. V listopadu 2007 byla založena zmiňovaná organizace, která vyvíjí systém do dnes. V témže roce byl vydán první vývojářský balík **SDK**[7]. **HTC Dream** byl uveden v září 2008 jako první chytrý telefon s **Androidem 1.0**. První masový **Android** byl vydán v dubnu 2009 ve verzi 1.5 a označením **Cupcake**. Všechny další verze vždy nesou číselné označení a název nějakého pamlsku. [6]

[7] SDK – Software development kit je typická sada vývojových nástrojů umožňující vytváření aplikací

Android 1.5 Cupcake

Muffin v košíčku se šlehačkou představen 30. 4. 2009 podporuje virtuální klávesnici třetích stran se slovníkem. Nahrává a přehrává videa ve formátu **MPEG-4** a **3GP**. Podporuje videa **YouTube** a obrázky **Picasa**. Nabízí možnost widgetů a animaci přechodů. [6]

Android 1.6 Donutd

Kobliha s otvorem vychází 1. 9. 2009. Integruje vyhledávání **Google**, **vylepšuje Market s aplikacemi**. Podporuje displeje s vyšším rozlišením, **VPN** připojení a nabízí bezplatnou navigaci od **Googlu**. [6]

Android 2.0/2.1 Eclair

Podlouhlý zákusek s náplní a čokoládou na povrchu vydán 20. 5. 2010. Nový design uživatelského prostředí. Optimalizace výkonu, podpora standardu **HTML5**, **Bluetooth 2.1**, **Google Maps 3.x** a živé tapety. V době uvedení na trh **Android Market** nabízel 20 000 aplikací. [6]

Android 2.2 Froyo

Šlehačková špička se objevuje 20. 5. 2010. Přináší nové uživatelské prostředí, webový prohlížeč s podporou **Flash 10**, možnost instalace aplikací i na paměťovou kartu, možnost vytváření **Wi-Fi hotspot** a možnost automatických aktualizací aplikací z **Android Marketu**, který již obsahoval 100 000 aplikací. [6]

Android 2.3 Gingerbread

Perníček vypuštěn 6. 12. 2010 podporuje více fotoaparátů, komunikaci přes **NFC**[8], senzory jako gyroskop nebo barometr, internetové hovory a vylepšuje virtuální klávesnici. [6]

Android 3.0 Honeycomb

Medová plástev uvedena 22. 2. 2011 je verze určená jen pro tablety. Podporuje **USB** příslušenství. [6]

[8] NFC – Near field communication je modulární technologie radiové bezdrátové komunikace mezi elektronickými zařízeními na velmi krátkou vzdálenost

Android 4.0 IceCream Sandwitch

Ruská zmrzlina odhalena 19. 10. 2011 sjednocuje verze pro tablety a telefony. Jako novinky umožnuje aplikace na zamčené obrazovce, nedávno spuštěné aplikace a vytváření adresářů. Umožňuje odemknutí rozpoznáním tváře, podporuje video ve **Full HD** a integruje sociální sítě do kontaktů. [6]

Android 4.2 Jelly Bean

Želatinové bonbóny uvolněn 9. 7. 2012 přináší podporu uživatelských účtů, zlepšuje notifikace, integruje vyhledávání **Google Now**, umožňuje přepínání uživatelských účtů. [6]

Android 4.4 KitKat

Čokoládová tyčinka prezentována 3. 9. 2013 nabízí lepší výkon, podporu krokoměru a infračerveného ovládání a inteligentní vypínání procesů na pozadí. [6]

Android 5.0 Lollipop

Lízátko představen 25. 6. 2014 má nový vzhled rozhraní **material design**, nový systém správy baterie s možností zapnout úsporný režim. [6]

Android 6.0 Marshmallow

Maršmelou uveden 5. 10. 2015 přináší podporu **USB typu C** a rozpoznávání otisků prstů. [7]

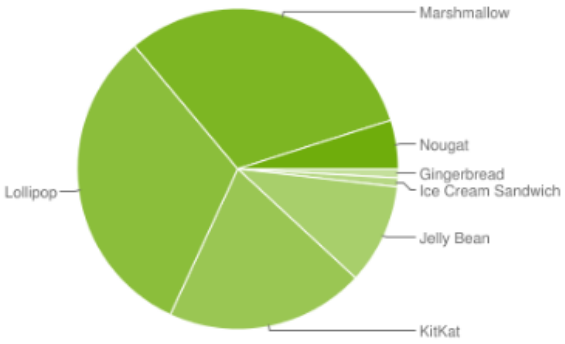
Android 7.0/7.1 Nugat

Nugát je poslední verze, která byla uvolněná 22. 8. 2016. Přinesla možnost práce s více okny, podpora **API pro virtuální realitu** a vykreslovací **API Vulkan 3D**. Změna politiky oprávnění nově je nutné povolit aplikaci přístup ke zdrojům telefonu i ve správci aplikací. [7]

1.3.3 Srovnání verzí operačního systému Android

Dnešní rozložení zastoupení **Androidu** na trhuje je znázorněno na **obrázku 7,** kde je také vidět používané rozhraní API jednotlivých verzí.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.9%
4.1.x	Jelly Bean	16	3.5%
4.2.x		17	5.1%
4.3		18	1.5%
4.4	KitKat	19	20.0%
5.0	Lollipop	21	9.0%
5.1		22	23.0%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	4.5%
7.1		25	0.4%



Obrázek 7 – Zastoupení verzí na trhu – zdroj developer.android.com

1.4 Porovnání referenčních aplikací

Na **Google Play** je nespočet aplikací, které simulují nebo ukazují postup řešeni **Rubikovy kostky**. Pro porovnání je vybráno několik aplikací, které jsou zdarma. Každou aplikaci bude nainstalovaná na testovací zařízení **Samsung SM-N9005 (Galaxi Note 3)** a otestována z pohledu jednoduchosti obsluhy a funkčnosti.

1.4.1 Referenční aplikace RubikSolver

První aplikace je **RubikSolver** podle recenzí od uživatelů je velmi spolehlivá, a především je chváleno animované zobrazení potřebných kroků ke složení zadané kostky. Dále tato aplikace nabízí režim simulace. V tomto režimu si muže uživatel nechat rozložit kostku a pomocí tahů v aplikaci se pokusit o složení.

Výhody

Hlavní výhodou je animovaná prezentace kroků spuštěných jako video s možností nastavení rychlosti. Aplikace také rychle detekuje špatně zadanou kostku. Kontroluje počet jednotlivých barev a správnou orientaci jednotlivých kostiček. Pokud zjistí nesložitelnou kostku, aplikace vyzve uživatele k opravě.

Nevýhody

Zásadní nevýhodou je absence načtení kostky pomocí fotoaparátu. Je nutné zadat celou reálnou kostku ručně políčko po políčku. Aplikace zároveň při animaci otáčí celou kostkou. Uživatel proto musí sledovat nejen odpovídající krok, ale i správnou orientaci kostky.

1.4.2 Referenční aplikace Rubik's Cube Fridrich Solver

Aplikace zaměřená na skládání kostky **Fridrichovou metodou**. Načtení kostky pomocí fotoaparátu je intuitivní. Aplikace zobrazí jednoduše potřebné otočení kostky, to umožní rychlé načtení celé kostky. Ruční zadání vyžaduje od uživatele prostorovou představivost při zadání barev do sítě. Výsledný algoritmus složení je prezentován na síťovém modelu kostky.

Výhody

Výhodou je intuitivní nafocení celé kostky. Aplikace přehledně popisuje celou Fridrichovu metodu skládání i s obrázky. Pomocí této aplikace je velmi snadné naučení Fridrichovy metody.

Aplikace detekuje špatně načtené barvy a zvýrazní je pro snadné opravení. Možnost skládání optimální metodou s minimem kroků nebo právě Fridrichovou metodou.

Nevýhody

Hlavní nevýhoda této aplikace je přesnost detekce jednotlivých barev, zejména rozdíly mezi červenou, oranžovou a žluto a samotná detekce bílé barvy. Detekce je výrazně závislá na světelných podmínkách. Složitější editace, která vyžaduje nejdříve vybrat barvu a poté jí umístit. Aplikace neumožnuje režim simulace. Výsledné kroky jsou v textové podobě doprovázeny změnou sítě kostky.

1.4.3 Referenční aplikace Rubik's Cube Free

Oficiální aplikace **Rubikovy kostky** nabízí simulovanou kostku, kterou lze skládat na čas nebo režim složení vlastní reálné kostky. Barvy jednotlivých stěn lze zadávat ručně po jednotlivé stěně nebo stěnu po stěně fotoaparátem. Následné kroky jsou jednoduše animovány na trojrozměrném modelu.

Výhody

Výhody této aplikace jsou přehledné zobrazení kroků skládání na trojrozměrném animovaném modelu. Algoritmus skládání obsahuje málo potřebných kroků. Aplikace je obsahem velmi rozsáhlá. Velmi propracovaný simulátor kostek od $2\times2\times2$ až po $5\times5\times5$. V režimu simulace je také měřen čas s možností zapnutí času na prohlédnutí hlavolamu před skládáním.

Nevýhody

Kostku pro načtení fotoaparátem musí uživatel předem podle návodu orientovat pro každou stranu zvlášť. Fotoaparát barvy nedetekuje přesně. Při kontrole složitelnosti nezobrazí, ani nepopíše nedostatek.

1.4.4 Referenční aplikace Easy Cube Solver

Aplikace nabízí povedený simulátor Rubikovy kostky a zároveň možnost zadat vlastní kostku. U vlastní kostky přehledně na animaci ukazuje postup složení. V nabídce je také možnost tutoriál, který ale popisuje ovládání aplikace, a nikoliv skládání kostky. Aplikace používá snadno naučitelnou metodu Fridrich.

Výhody

Velmi přehledná animace kroků s možností spuštění nebo posunu po kroku. Aplikace detekuje složitelnost kostky. V nabídce lze změnit barevné schéma celé kostky na jiné rozmístění barev.

Nevýhody

Absence načtení pomocí fotoaparátu. Při editaci kostky se rotuje příliš rychle, proto je složité nastavit kostku do požadované pozice.

1.4.5 Referenční aplikace Magic Cube Solver

Poslední z testovaných aplikací, která nenabízí možnost nasnímání. Neobsahuje simulátor. Aplikace nabízí možnost zobrazit kroky nebo animovat.

Výhody

Algoritmus skládání používá malý počet kroků. Lze si kostku při skládání otočit dle potřeby.

Nevýhody

Zadání vlastní kostky je značně nepřehledné. Všechny stěny jsou pod sebou. Uživatel proto nemá možnost vidět, jak na sebe navazují.

1.4.6 Výsledek srovnání referenčních aplikací

Ze srovnání konkurenčních aplikací nejlépe vychází aplikace **Rubik's Cube Fridrich Solver**, viz **tabulka 1**. Tato aplikace jednoduše nafotí kostku a přehledně zobrazí kroky. I tato aplikace ovšem detekuje špatně barvy, což není možné ovlivnit úpravou úhlu natočení fotoaparátu nebo přisvětlením. Uživatel nevidí detekované barvy, dokud nedokončí celé načítání.

Nejhorší funkčnost má aplikace **Magic Cube Solver**, která nepřehledně zobrazuje zadávání kostky na jednotlivé plochy bez vizuálního spojení hranami. Aplikaci zároveň nefunguje možnost v menu.

Tabulka 1 – Porovnání referenčních aplikací

Aplikace	Hodnocení na Google Play	Výhody	Nevýhody
RubikSolver	4,4	přehledné zobrazení kroků animací, detekce špatného zadání	nelze načíst fotoaparátem, animace často otáčí kostkou
Rubik's Cube Fridrich Solver	4,3	intuitivní nafocení kostky, postupy skládání Fridrichovou metodou, přehledný tutoriál	přesnost detekce jednotlivých barev, složitější editace modelu na sítí a zobrazení kroků
Rubik's Cube Free	3,7	animovaná prezentace, nízký počet kroků, propracovaný simulátor	složité a nepřesné focení, nepopsání chyby zadání
Easy Cube Solver	4,2	animovaná prezentace, možnost změnit barvy, detekce složitelnosti	nemožnost načtení fotoaparátem, špatné ovládání kostky
Magic Cube Solver	3,3	malý počet kroků, otočení kostky dle potřeby	nepřehledné zadání kostky

2 PRAKTICKÁ ČÁST

2.1 Požadavky aplikace

Hlavním cílem této bakalářské páce je vytvořit uživatelsky jednoduchou aplikaci pro **platformu Android**. Aplikace bude poskytovat návod na složení **Rubikovy kostky**. Vyřeší hlavní nevýhodu všech testovaných aplikací načítání kostky. V tomto ohledu je potřeba zabezpečit přesné načtení kostky pomocí fotoaparátu. Dále by aplikace měla přesně vést uživatele všemi kroky složení bez nutnosti otáčení celé kostky, které je velice náchylné na chybu při skládání. **Tabulka 2** obsahuje přehled požadavků na aplikaci.

Tabulka 2 - Požadavky aplikace

ID	Popis požadavku	Typ	Priorita
P1	Aplikace bude detekovat a zobrazovat barvy v reálném čase.	Nefunkční	Musí mít.
P2	Aplikace bude využívat dvě aktivity CameraActivity a EditorActivity.	Funkční	Měla by mít.
P3	Aplikace bude umožňovat editaci celé kostky.	Funkční	Měla by mít.
P4	Aplikace bude popisovat chyby zadání.	Funkční	Musí mít.
P5	Aplikace bude zobrazovat přehledně potřebné kroky.	Funkční	Musí mít
P6	Aplikace bude umožňovat přisvětlení diodou.	Nefunkční	Může mít.
P7	Aplikace bude uchovávat model kostky.	Nefunkční	Musí mít.
P8	Aplikace najde řešení validních kostek.	Funkční	Musí mít.
P9	Aplikace bude podporovat operační systém Android s verzí API 21 a vyšší.	Nefunkční	Musí mít.
P10	Aplikace bude navržena pro běh ve více vláknovém systému.	Nefunkční	Musí mít.
P11	Aplikace bude využívat vibrační zpětnou vazbu.	Funkční	Může mít.
P12	Aplikace se bude přizpůsobovat velikostem a rozlišením obrazovek zařízení.	Nefunkční	Měla by mít.
P13	Aplikace bude odstraňovat duplicitní kroky.	Funkční	Může mít.
P14	Aplikace bude kontrolovat validitu kostky.	Funkční	Měla by mít.

2.2 Implementace požadavků

2.2.1 P1 – Aplikace bude detekovat a zobrazovat barvy v reálném čase

Pro detekci barev je použita knihovna **OpenCV**. Knihovna nabízí jednoduché funkce pro zpracování obrazu v reálném čase. Při detekování každého snímku docházelo k rychlé změně detekovaných barev. Bylo obtížné zaznamenávat barvy, které byly správné. Z tohoto důvodu v metodě `onCameraFrame()` jsou jednou za sekundu z jednoho snímku kamery detekovány barvy. Tyto barvy jsou zakresleny vlevo nahoře a v detekčních bodech. Pokud je uživatel spokojen s detekcí barvy zaznamená.

Zdrojový kód 1 – Metoda onCameraFrame()

```
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame){
        rgba = inputFrame.rgba();
        int vzdalenost = rgba.rows() / 5;
        Point stredObrazovky = new Point(rgba.cols() / 2, rgba.rows() / 2);
        Point[][] velkaMatice = Kresleni.vratNaklonenouMatici3x3(stredObrazovky, vzdalenost,
vzdalenost / 5);
        if(ukaz) {
            // urci barvy pod matici
            ukaz = false;
            cvtColor(inputFrame.rgba(), hsv, COLOR_RGB2HSV);
            barvy = Detektor.barvaOblasti(hsv, velkaMatice);
        }
            Scalar barvaZnacek = new Scalar(255, 0, 255, 255);
            Point[][] maticeVysledku = Kresleni.vratMatici(new Point(80, 80), 3, 3, 50);
            //nakresli matici detekcnich bodu
            Kresleni.kreslyTvar(rgba, velkaMatice, 45, barvaZnacek, 5);
            //matice vysledku
            Kresleni.kreslyTvary(rgba, velkaMatice, 30, barvy, -1);
            //matice vysledku vlevo
            Kresleni.kreslyTvary(rgba, maticeVysledku, 30, barvy, -1);
        return rgba;
    }
```

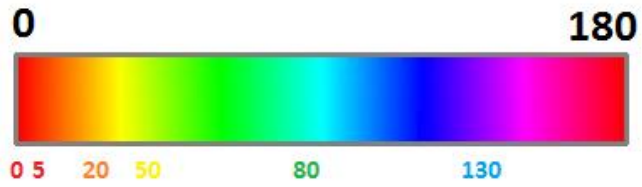
Rozpoznání barvy

Nejdříve jsou detekované barvy pod body detekce, které jsou následně zobrazeny. K detekci slouží samostatná třída `Detektor`. Z důvodu nestálých vnějších světelných podmínek a rozdílných odstínů nálepek různých **Rubikových kostek** je pro detekci třeba určit interval pro každou barvu. Z **obrázku 8** je například patrné, že z barevného modelu **RGB** lze těžko určit interval všech složek, který by popisovali červenou barvu. Pro snadnější detekci pěti rozdílných po sobě jdoucích barev je lepší převést vstupní snímek z barevného modelu **RGB** do modelu **HSV**. V modelu **HSV** stačí určit hodnotu odstínu, protože odstín se nemění pro různé úrovně jasu.

JMÉNO	HEXA KÓD	DECIMÁLNÍ KÓD	UKÁZKA BARVY
IndianRed	#CD5C5C	rgb(205,92,92)	
LightCoral	#F08080	rgb(240,128,128)	
Salmon	#FA8072	rgb(250,128,114)	
DarkSalmon	#E9967A	rgb(233,150,122)	
LightSalmon	#FFA07A	rgb(255,160,122)	
Crimson	#DC143C	rgb(220,20,60)	
Red	#FF0000	rgb(255,0,0)	
FireBrick	#B22222	rgb(178,34,34)	
DarkRed	#8B0000	rgb(139,0,0)	

Obrázek 8 – Červená barva v RGB – zdroj www.klikzone.cz

Knihovna **OpenCV** používá pro popis odstínu v **HSV** barevném modelu **hodnoty 0 až 180**. Pro popis sytosti a **hodnoty rozmezí 0 až 255**. Testováním nejrůznějších intervalů pro detekci byli nalezeny intervalové hodnoty pro 5 barev kostky. Hranice jsou znázorněny na **obrázku 9**.



Obrázek 9 – Hranice detekovaných barev

Bílou barvu je dobré detekovat velikostí jasu. Jas lze výhodou určit z modelu **RGB**. V tomto modelu je jas určen následující rovnicí:

$$\boldsymbol{Y} = 0.2126 \times \boldsymbol{R} + 0.7152 \times \boldsymbol{G} + 0.0722 \times \boldsymbol{B}$$

kde **R**, **G** a **B** jsou jednotlivé složky **RGB** barvy.

Metoda `jeBila()` určí hodnotu jasu barvy a jestli je jas vyšší než hranice .

Zdrojový kód 2 – Metoda `jeBila()`

```
private static boolean jeBila(Mat vzorek, Point stred) {
        Scalar rgbBarva = getScalarBodu(vzorek, stred);
        int hranice = 230;
        return  (0.2126*rgbBarva.val[0] + 0.7152*rgbBarva.val[1] +
0.0722*rgbBarva.val[2])>hranice;
    }
```

Vykreslení výsledku

Pro snadné zobrazení detekovaných barev v reálném čase je vytvořena třída `Kresleni`. Hlavní funkce této třídy je vytváření matic bodů a kreslení čtverců do matice. Při vykreslení detekčních oblastí je určen střed obrazovky a zvolena vhodná velikost detekční matice tak, aby zabírala přibližně 2/3 výšky obrazovky. Matice je zobrazena čtverci. Po detekování barvy je do výplně detekčních čtverců vyplněna detekovaná barva.

2.2.2 P2 – Aplikace bude využívat dvě aktivity `CameraActivity` a `EditorActivity`

Aplikace využívá dvě aktivity `CameraActivity` a `EditorActivity`. Obě aktivity jsou zobrazeny pouze v režimu na šířku.

Aktivita `CameraActivity`

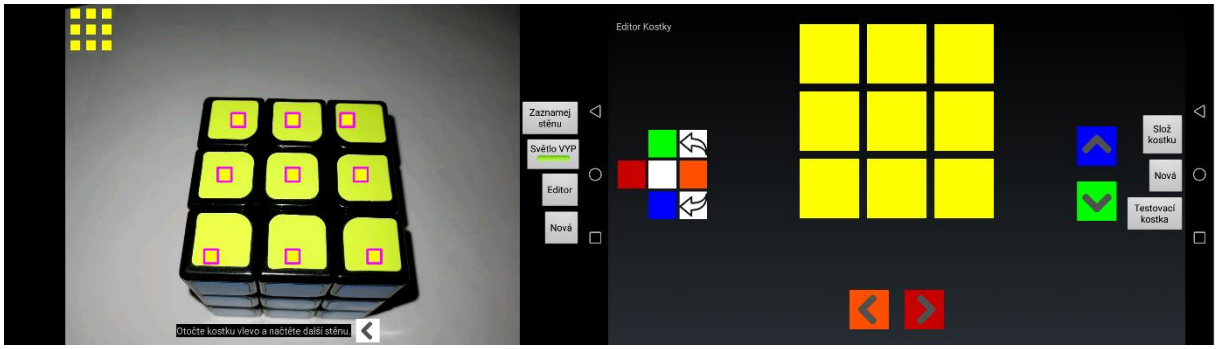
Startovací aktivita aplikace zobrazená na **obrázku 10 vlevo**. Zobrazuje detekci pomocí fotoaparátu v reálném čase. V levém horním rohu jsou vyobrazeny aktuálně detekované barvy. Ve spodní části je popisový řádek, který popisuje potřebné natočení kostky. Na pravé straně jsou ovládací tlačítka, která nabízí možnost:

a) **zaznamenat stěnu** detekované barvy se uloží do modelu,
b) **zapnou/vypnout světlo** podle **požadavku P6,**
c) **editor** přepne do druhé aktivity,
d) **nová** nastaví novou prázdnou kostku.

Aktivita `EditorActivita`

Editační aktivita aplikace zobrazená na **obrázku 10 vpravo**. Umožnuje zadat kostku ručně nebo editovat již načtenou kostku podle **požadavku P3**. Šipky umožnují rotaci kostky ve zvoleném směru. Tlačítka na pravé straně nabízí následující funkce:

a) **slož kostku** provede validaci a pokusí se složit kostku, následně zobrazí dialog kroků nebo chybu validace,
b) **nová** nastaví prázdnou kostku a přepne na hlavní aktivitu,
c) **testovací kostka** nastaví rozloženou testovací kostku.



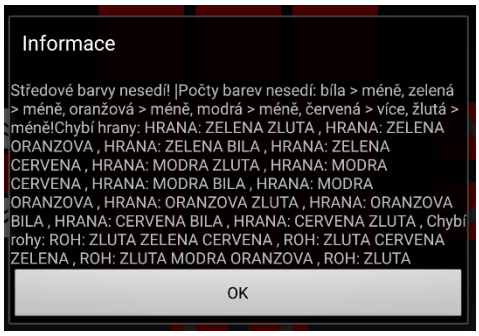
Obrázek 10 – Aktivity aplikace vlevo `CameraActivity`, vpravo `EditorActivity`

Po zvolení políčka modelu je zobrazen dialog na **obrázku 11** pro změnu barvy políčka.

Obrázek 11 – Dialog změny barvy

2.2.3 P4 – Aplikace bude popisovat chyby zadání

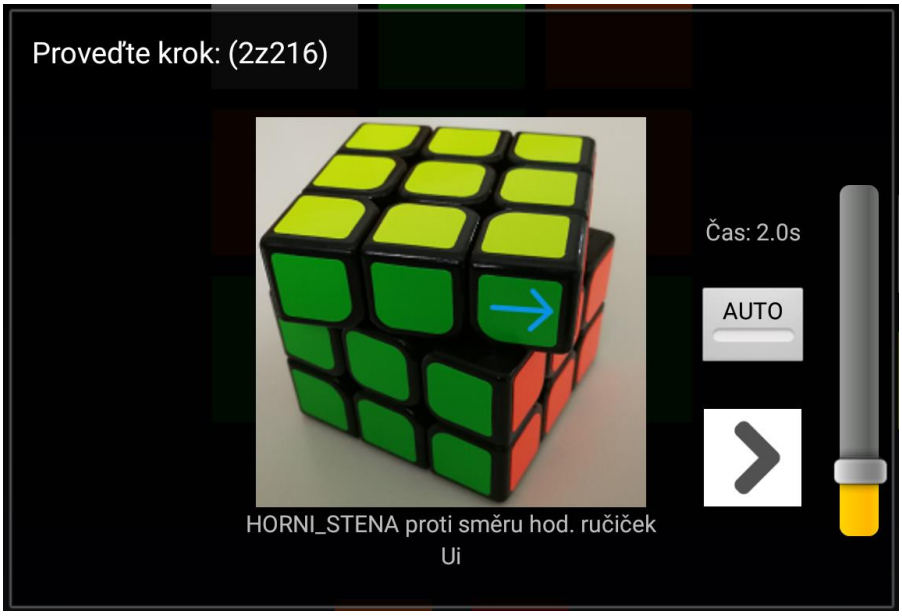
Chyby zadání jsou zobrazeny v dialogu na **obrázku 12**. Dialog popisuje, co přesně neprošlo validací a co by měl uživatel upravit, aby byla kostka a složitelná.



Obrázek 12 – Informační dialog

2.2.4 P5 – Aplikace bude zobrazovat přehledně potřebné kroky

Na **obrázku 13** dialog zobrazuje postupně kroky, které by měl uživatel provést. Pomocí šipky lze zobrazit další krok, stlačení šipky doprovází vibrační zpětná vazba podle **požadavku P11**. Pomocí přepínacího tlačítka lze zapnout nebo vypnout automatický režim procházení kroků. V tomto režimu jsou kroky automaticky procházeny po zvoleném čase. Tento čas lze nastavit pomocí posuvníku vpravo v rozmezí 0,5 sekundy až 10,5 sekundy.



Obrázek 13 – Dialog kroků

2.2.5 P7 – Aplikace bude uchovávat model kostky

Pro uchování Rubikovy kostky slouží hlavní třídy `Kostka`, `ModelKostky`, `Kosticka` a `Krok`. Tyto třídy zabezpečí potřebnou funkcionalitu pohybů kostky a samotného složení.

Třída `Kosticka`

Třída představuje jednu kostičku Rubikovy kostky. Na kostce existují 4 druhy kostiček, které se liší počtem viditelných stěn. K rozlišení druhů slouží `enum DruhKosticky`.

Zdrojový kód 3 - `Enum DruhKosticky`

```
public enum DruhKosticky {
    STRED(1),
    HRANA(2),
    ROH(3),
    PRAZDNA(0);
    private int pocetBarev;
    DruhKosticky(int pocet){
        pocetBarev = pocet;
    }
    /**
     * Metoda vrátí počet možných barev podle druhu kostičky.
     * @return počet možných barev
     */
    public int getPocetBarev() {
        return pocetBarev;
    }
}
```

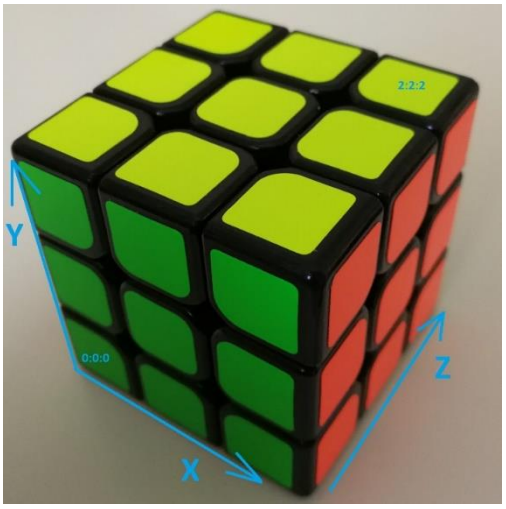
Třída implementuje rozhraní `IKosticka` (**Příloha A**). Rozhraní zabezpečuje veškeré potřebné metody pro nastavení barev na kostičce a otáčení kostičkou všemi možnými směry. Třída kostička obsahuje tovární metodu `vratKostisku()`, která vrací v závislosti na počtu parametrů odpovídající instanci kostičky s nastavenými barvami.

Zdrojový kód 4 - Tovární metoda `vratKosticku()`

```
public static Kosticka vratKostisku(Barva... barvy){
         Kosticka k =null;
         switch (barvy.length){
             case 0:
                  k = new Kosticka(PRAZDNA);
                  return k;
             case 1:
                  k = new Kosticka(STRED);
                 k.barvaPredni=barvy[0];
                  k.pocetVidet = 1;
                 return k;
            case 2:
                k = new Kosticka(HRANA);
               k.barvaPredni=barvy[0];
               k.barvaVpravo=barvy[1] ;
                k.pocetVidet = 2;
                return k;
            case 3:
                k = new Kosticka(ROH);
                k.pocetVidet = 3;
               k.barvaPredni=barvy[0];
                k.barvaVpravo=barvy[1];
               k.barvaDole=barvy[2];
                return k;
        }
        return k;
    }
```

Třída `ModelKostky`

Třída implementuje rozhraní `IModelKostky` (**Příloha B**). Třída obsahuje trojrozměrné pole kostiček a provádí potřebné operace nad celým polem. V této třídě jsou implementované veškeré funkce algoritmu složení kostky metodou **vrstva po vrstvě**, dále obsahuje funkce pro validaci modelu a vnitřní třídu `Pozice`. Tato vnitřní třída slouží pro hledání kostiček v modelu a vrácení či porovnává souřadnice kostičky. Model využívá souřadnicový systém XYZ viz **obrázek 14**.



Obrázek 14 – Souřadnicový systém modelu XYZ

Funkce `sloz()`

Funkce pro složení modelu. Nejdříve je model nastaven do výchozí pozice. To znamená přední stěna je zelená, spodní stěna je bílá. Postupně jsou volány funkce:

a) složení bílého kříže,
b) složení bílých rohů,
c) složení druhé řady,
d) složení žlutého kříže,
e) složení žlutých rohů.

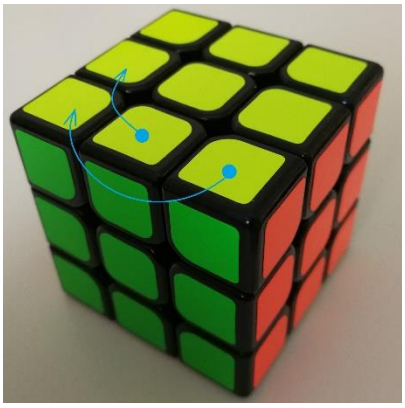
Všechny funkce vrací `Vektor` potřebných kroků. Tyto kroky zároveň provádí na modelu. To zajišťuje, že je možné kontrolovat stav modelu před zahájením dalšího dílčího skládání modelu. Pokud předchozí funkce neprovedla potřebné složení je vystavena výjimka.

Složení bílého kříže, bílých rohů a druhé řady je prováděno čtyřikrát po sobě s relativním natočením modelu k relativnímu výchozímu bodu. Například skládání hrany bílého kříže je pro všechny čtyři hrany stejné, jen se mění relativní roh, ke kterému se umísťuje hrana. Výsledkem tohoto faktu je, že stačí jedna logika pro složení jedné hrany ze všech možných startovacích pozic a možných natočení na modelu. Pro každou hranu se pouze změní označení pohybů pro kroky a startovací pozice. Tento princip je využit i u rohů bílé stěny a hran druhé řady.

Při skládání poslední vrstvy se využívá možnost natočení této vrstvy směrem k uživateli tak, aby nemusel potřebnou logiku provádět ze všech směrů kostky. Logika totiž nebortí složení kostky do druhé řady.

Provádění kroků

Při provádění kroků, jako je například otáčení celé kostky směrem vlevo, jsou nejdříve všechny kostičky otočeny požadovaným směrem a následně přesunuty podle **obrázku 15.**



Obrázek 15 – Otočení modelu

Při otáčení pouze jedné vrstvy se otočí všechny kostičky této vrstvy a následně jsou přemístěny stejným principem jako u celého modelu pouze na odpovídající vrstvě.

Třída `Krok`

Třída pro vytvoření potřebného kroku na pohyb modelem. Využívá výčet `Pohyb`, který popisuje všechny možné pohyby na kostce.

Zdrojový kód 5 - Enum `Pohyb`

```
public enum Pohyb {
    HORNI_STENA("U"),
    DOLNI_STENA("D"),
    PRAVA_STENA("R"),
    LEVA_STENA("L"),
    PREDNI_STENA("F"),
    ZADNI_STENA("B"),
    NAHORU_KOSTKA("X"),
    VLEVO_KOSTKA("Y"),
    POSMERU_KOSTKA("Z");
    private String zkratka;

    Pohyb(String zkratka) {
        this.zkratka = zkratka;
    }

    public String getZkratka() {
        return this.zkratka;
    }
}
```

Samotný krok uchovává navíc počet, kolikrát má nebo je proveden. Třída obsahuje metodu `proved()`. Tato metoda provede nad modelem odpovídající krok.

Třída `Kostka`

Hlavní úkol třídy je uchovat po celou dobu spuštění aplikace aktuální konzistentní model kostky. Tato třída je jedináčkem, to lze zajistit následující částí kódu. Statická metoda `getInstance()` je navržena pro běh ve více vláknových aplikacích vyhovující **požadavku P10** tak, aby nedocházelo k nekonzistentnosti instance třídy.

Zdrojový kód 6 - Jedináček kostky

```
public class Kostka implements IKostka {
    private static Kostka instance = null;
    private IModelKostky model;
    private int pocetNactenych;
    public static Kostka getInstance() throws Exception {
        if (instance == null)
            synchronized (Kostka.class) {
                if (instance == null) {
                    instance = new Kostka();
                }
            }
        return instance;
    }
    /**
     * Konstruktor nastaví všechny dílčí kostičky podle druhu na své místo a nastaví prázdnou
kostku.
     */
    private Kostka() throws Exception {
        pocetNactenych = 0;
        model = new ModelKostky();
        model.nastavPrazdny();
    }…
```

2.2.6 P8 – Aplikace najde řešení validních kostek

Aplikace skládá kostku metodou **vrstva po vrstvě**. Vlastní složení kostky je prováděno v následujícím pořadí:

a) je vytvořena kopie modelu,
b) kopie modelu je složena a jsou vráceny potřebné kroky ve vektoru,
c) z vektoru kroků je odstraněna duplicita (popsána níže),
d) je vrácen vektor potřebných kroků ke složení a model kostky je v původním stavu.

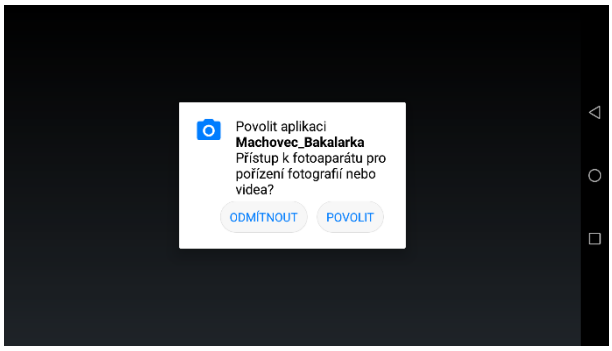
2.2.7 P9 – Aplikace bude podporovat operační systém Android s verzí API 21 a vyšší

Pro oprávnění aplikace přistupovat ke kameře telefonu, přisvětlovací diodě a vibračnímu motorku je nutné zadat do **android manifestu** (**Příloha C** – *AndroidManifest.xml*) nutná oprávnění.

Zdrojový kód 7 - Oprávnění aplikace

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.camera.autofocus"/>
<uses-feature android:name="android.hardware.camera.front"/>
<uses-feature android:name="android.hardware.camera.front.autofocus"/>
<uses-permission android:name="android.permission.FLASHLIGHT" />
<uses-permission android:name="android.permission.VIBRATE"/>
```

K zajištění kompatibility s novou verzí **Android 7.x** je nutné vyžádat při prvním spuštění aplikace povolení k přístupu k fotoaparátu telefonu viz **obrázek 16**.



Obrázek 16 - Povolení přístupu k fotoaparátu

Potřebné vyvolání dialogu lze zajistit následujícím kódem v metodě `onCreate()` startovací aktivity. Kód nejdříve zjistí nastavení správce aplikací na potřebné oprávnění. Pokud není povolené, pokusí se vyvolat dialog pro povolení oprávnění.

Zdrojový kód 8 - Vyvolání povolení oprávnění

```
if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.CAMERA)
                != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.CAMERA},
                        MY_PERMISSIONS_REQUEST_READ_CONTACTS);
        }else {
…
```

2.2.8 P12 – Aplikace se bude přizpůsobovat velikostem a rozlišením obrazovek zařízení

Ovládací prvky aplikace jsou navrženy tak, aby se vhodně přizpůsobovaly nejrůznějším variantám velikostem a rozlišením obrazovek zařízení. Toho lze docílit použitím jednotek **dp** při návrhu rozložení a velikostí prvků. Jednotky **dp** jsou vykreslovacím zařízením automaticky přepočítány v závislosti na velikosti displeje na velikost v reálných pixelech. To zajistí, že ovládací prvky budou mít vhodnou velikost tak, aby byly pohodlně použitelné na nejrůznějších zařízeních.

2.2.9 P13 – Aplikace bude odstraňovat duplicitní kroky.

Protože algoritmus může některé kroky stejného pohybu umístit za sebou, například třikrát krok otočení pravé stěny. Je vhodné tyto kroky odstranit a nahradit jedním krokem otočením pravé stěny zpět. U kroků provedených násobkem 4 jsou kroky odstraněny úplně. Tento princip se opakuje, dokud výsledný vektor je kratší než vstupní. Přiklad odstranění:

a) vstupní posloupnost (**B, R, U, U, U, U, R, R, D, L)**,
b) první odstranění odstraní po sobě jdoucí (**B, R, U, U, U, U, R, R, D, L)**
   výsledek **(B, R, 2R, D, L)**
c) druhé odstranění odstraní nově zvyklou posloupnost **(B, R, 2R, D, L)**
   na výsledné **(B, Ri, D, L)** čtyři kroky z deseti původních.

2.2.10 P14 – Aplikace bude kontrolovat validitu kostky

Pro splnění **požadavku P14** je vytvořena funkce. Funkce kontroluje pozici všech středových kostiček v modelu vůči sobě. Následuje kontrola počtu všech barev. Každé barvy musí být právě devět. Konečná kontrola sleduje umístění barev na jednotlivých kostičkách. Je kontrolováno, jestli nesousedí například barvy bílá a žlutá na jedné hraně (bílá stěna je vždy přesně na opačné stěně se žlutou, proto spolu nesousedí). Dále jsou kontrolovány duplicitní barvy na stejné kostičce. V případě nedostatku je vrácen řetězec popisující nedostatek.

2.3 Výsledné řešení

Aplikace je schopna rozpoznat 9 barevných čtverečků (kostiček) stěny **Rubikovy kostky**. Detekce je prováděná v reálném čase. Při uspokojivé detekci uživatel zaznamená stěnu a otočí kostku podle pokynů. Po načtení kostky přejde aplikace do režimu editace, kde je možné procházet celou kostku a měnit barvy kostiček. Následně po stisku tlačítka aplikace najde řešení kostky a zobrazí kroky. Při skládání může uživatel kdykoliv sekvenci přerušit a v aplikaci bude kostka ve stavu, jak by měla vypadat v reálném stavu. Pokud skládaný hlavolam nebude v daném kroku shodný s aplikací kostka nebudou sedět (uživatel se spletl), je možné v aplikaci zvolit novou kostku a nasnímat hlavolam znovu.

Před skládáním je na pozadí provedena nejprve validace kostiček. Následně se aplikace pokusí najít kroky ke složení. V této části může dojít k výjimce, která nastane, pokud bude kostka nesložitelná. Při nalezení řešení se zobrazí dialog s kroky, kde je možné kroky posouvat vpřed. Kroky jsou popsány větou, zkratkou a odpovídající fotografií. Celý proces skládání je kostka držena ve stejné orientaci.

2.4 Instalace aplikace na mobilní zařízení

Instalace je prováděná pomocí jednoho instalačního balíčku. Pro instalaci vytvořené aplikace musí být do telefonu nainstalován balíček **OpenCV Manager**, který je nezbytný pro běh aplikací s knihovnou **OpenCV**. Pokud při prvním spuštění aplikace zjistí, že není nainstalován balíček **OpenCV Manager**, tak vyzve uživatele pro nainstalování a přesměruje ho do **Google play**. Uživatel stáhne a nainstaluje balíček. Při dalším spuštění aplikace vše funguje bez problému.

2.5 Možná budoucí vylepšení aplikace

Animace

Hlavní nevýhodou dosavadního prezentování kroků je nedynamické zobrazení aktuálního stavu kostky. Bylo by vhodné kroky prezentovat pomocí animace. Animace by odrážela v každém okamžiku stav kostky. Uživatel by byl schopen včas odhalit krok, který provedl nesprávně a tím by se vyvaroval nechtěnému znovu nasnímání kostky při chybě.

Animaci by bylo vhodné nasadit i do editoru, kde by byla snazší editace kostky pomocí otáčení modelu v prostoru.

Algoritmus

Aplikace by mohla nabízet možnost několika algoritmů skládání. Uživatel by měl možnost skládat pomocí logicky zapamatovatelných kroků nebo by skládal s minimem potřebných kroků. Minimum kroků by bylo využíváno pro rychlé složení kostky.

Validace

Aplikace by mohla při nedostatku validace přesně určit kostičku, která není korektní a navrhnou možnou opravu.

Detekce barev

Aplikace by mohla nabízet kalibraci detekovaných barev. Každý výrobce kostek má vlastní odstíny barev. To může způsobovat časté nepřesnosti detekce zejména u odstínů červené, žluté a oranžové barvy, které v modelu **HSV** jsou blízko u sebe.

Bylo by vhodné navrhnout takový algoritmus, který by zaznamenal sekvenci několika snímků bez přisvětlení a s přisvětlením. Na základě vyhodnocení by byl schopen rozhodnout správnost detekované barvy.

Možnost procvičování

Aplikace by mohla nabízet cvičící nástroje pro naučení uživatele skládání celé kostky. Tato možnost by uměla naučit potřebnou posloupnost kroků k určitému složení například bílého kříže a sama by rozkládala kostku tak, aby uživatel byl nucen opakovaně provádět kroky na

nejrůznějších variantách. Celé učení by bylo doprovázeno logickým vysvětlením, co který algoritmus přesně dělá a jak by měl vypadat výsledek.

3 ZÁVĚR

Hlavní cíl této bakalářské práce byl vytvořit funkční aplikaci pro platformu Android. Aplikace je napsaná v jazyce Java a využívá open source knihovnu OpenCV. Aplikace načte kostku pomocí fotoaparátu telefonu. V aplikaci je možno nasnímanou kostku ručně editovat, zkontrolovat správnost načtení (zadání) a navrhnout kroky ke složení. Tato aplikace plně vyhovuje zadání bakalářské práce.

Výsledná aplikace byla navržena na základě testování referenčních aplikací. Aplikace především špatně načítaly barvy kostek nebo nevhodně zobrazovali kroky.

Při návrhu aplikace bylo z počátku testováno načítání kostky pomocí počítačového vidění bez nutnosti umisťovat kostku jednou stěnou do detekčních bodů. Takto navržená aplikace nebyla použitelná z důvodu nízkého výkonu telefonu a různým typům kostek. Některé kostky používají jinou barvu materiálu, ze kterého jsou vyrobeny nebo jiný tvar nálepek. Proto nebylo možné navrhnout takový algoritmus, který by s jistotou detekoval postupně všechny barvy a pozice barev vůči sobě.

Výsledná aplikace byla vyzkoušená na několika desítkách různě rozložených hlavolamů. Aplikace hlavolam vždy bez problému složila. V průměru navržený způsob skládání používá kolem 200 kroků ke složení celé kostky. Nedostatky aplikace jsou však stále v načítání kostky. Největší chybu v načítání způsobují okolní světelné podmínky, které ve výsledku značně mění odstíny barev. Zkušený uživatel je ale schopen vhodně použít diodu a, nebo natočit telefon pod správným úhlem ke kostce tak, aby bylo načtení přesné.

Pro zlepšení použitelnosti aplikace by bylo vhodné doplnit aplikaci o vylepšení, která byla popsána v předchozí kapitole.

4 POUŽITÁ LITERATURA

[1] THE HISTORY OF THE RUBIKS CUBE. *THE HOME OF RUBIK'S CUBE* [online]. Rubiks Brand, 2017 [cit. 2017-04-03]. Dostupné z: https://eu.rubiks.com/about/the-history-of-the-rubiks-cube/

[2] CUBE FACTS. *THE HOME OF RUBIK'S CUBE* [online]. Rubik's Brand, 2017 [cit. 2017-04-03]. Dostupné z: https://eu.rubiks.com/about/cube-facts/

[3] **BURGER, Wilhelm a Mark James BURGE**. *Digital Image Processing: An Algorithmic Introduction using Java* [online]. London: Springer, 2008 [cit. 2017-04-03]. ISBN 978-1-84628-968-2. 10.1007/978-1-84628-968-2. Dostupné z: https://link.springer.com/

[4] **BRADSKI, Gary R. a Adrian KAEHLER**. *Learning OpenCV: Computer Vision with the OpenCV Library* [online]. Sebastopol: O'Reilly Media, c2008 [cit. 2017-04-04]. ISBN 978-0-596-51613-0. Dostupné z: https://www.safaribooksonline.com/

[5] **YŪKI, Inoue.** Úvod do počítačové grafiky: Základy optiky, barevné modely. *ITnetwork.cz* [online]. 2015 [cit. 2017-04-04]. ISSN 2464-6326. Dostupné z: http://www.itnetwork.cz/grafika/uvod-do-pocitacove-grafiky-optika-modely

[6] **LACKO, Ľuboslav.** *Vývoj aplikací pro Android*. Brno: Computer Press, 2015. ISBN 978-80-251-4347-3.

[7] *Android Developers* [online]. Google, 2017 [cit. 2017-04-25]. Dostupné z: https://developer.android.com/index.html

5 PŘÍLOHY

Příloha A - *Rozhraní IKosticka*

```
public interface IKosticka {

    /**
     * Metoda změní barvu na přední straně kostičky na barva.
     * @param barva na kterou se změní
     */
     void zmenPredniBarvu(Barva barva);

    /**
     * Metoda nastaví barvu dole na barvaDole.
     * @param barvaDole barva nastavení
     * @throws Exception pokud kostička má již maximální počet barev
     */
     void setBarvaDole(Barva barvaDole) throws Exception ;

    /**
     * Metoda nastaví barvu nahoře na barvaNahore.
     * @param barvaNahore barva nastavení
     * @throws Exception pokud kostička má již maximální počet barev
     */
     void setBarvaNahore(Barva barvaNahore) throws Exception ;

    /**
     * Metoda nastaví barvu přední na barvaPredni.
     * @param barvaPredni barva nastavení
     * @throws Exception pokud kostička má již maximální počet barev
     */
     void setBarvaPredni(Barva barvaPredni) throws Exception ;

    /**
     * Metoda nastaví barvu vlevo na barvaVlevo.
     * @param barvaVlevo barva nastavení
     * @throws Exception pokud kostička má již maximální počet barev
     */
     void setBarvaVlevo(Barva barvaVlevo) throws Exception ;

    /**
     * Metoda nastaví barvu vpravo na barvaVravo.
     * @param barvaVravo barva nastavení
     * @throws Exception pokud kostička má již maximální počet barev
     */
     void setBarvaVpravo(Barva barvaVravo) throws Exception ;

    /**
     * Metoda nastaví barvu vzadu na barvaZadni.
     * @param barvaZadni barva nastavení
```

```
     * @throws Exception pokud kostička má již maximální počet barev
     */
     void setBarvaZadni(Barva barvaZadni) throws Exception ;

    /**
     * Metoda vrátí barvu na pozici.
     * @return Barva na pozici
     */
     Barva getBarvaDole();

    /**
     * Metoda vrátí barvu na pozici.
     * @return Barva na pozici
     */
     Barva getBarvaNahore();

    /**
     * Metoda vrátí barvu na pozici.
     * @return Barva na pozici
     */
     Barva getBarvaPredni();

    /**
     * Metoda vrátí barvu na pozici.
     * @return Barva na pozici
     */
     Barva getBarvaVlevo();

    /**
     * Metoda vrátí barvu na pozici.
     * @return Barva na pozici
     */
     Barva getBarvaVpravo();

    /**
     * Metoda vrátí barvu na pozici.
     * @return Barva na pozici
     */
     Barva getBarvaZadni();

    /**
     * Metoda vrátí počet viditelných barev.
     * @return počet viditelných
     */
     int getPocetVidet();

    /**
```

```
     * Metoda vrátí druh kostičky.
     * @return Barva na pozici
     */
     DruhKosticky getDruhKosticky();

    /**
     * Metoda přesune barvy predni>vlevo, vlevo>zadni, zadni>vpravo a vpravo>predni
     */
     void otocVlevo() ;

    /**
     * Metoda přesune barvy přední>vpravo, pravo>zadní, zadní>vlevo a vlevo>přední
     */
     void otocVpravo() ;

    /**
     * Metoda přesune barvy přední>nahoru, nahore>zadní, zadní>dole a dole>přední
     */
     void otocNahoru() ;

    /**
     * Metoda přesune barvy přední>dolu, dolu>zadní, zadní>nahoru a nahoru>přední
     */
     void otocDolu() ;

    /**
     * Metoda přesune barvy horní>vpravo, vpravo>dole, dole>vlevo a vlevo>nahoru.
     */
     void otocPoSmeru() ;

    /**
     * Metoda přesune barvy horní>vlevo, vlevo>dole, dole>vpravo a pravo>nahoru.
     */
     void otocProtiSmeru() ;


}
```

Příloha B – *Rozhraní IModelKostky*

```
public interface IModelKostky {

    /**
     * Metoda nastaví všechny barvy přední stěny podle barvy.
     * @param barvy nastavované barvy
     * @throws Exception pokut nejde nastavit barva kostičky
     */
    void setPredniBarvy(Barva[][] barvy) throws Exception;

    /**
     * Metoda nastaví testovací model.
     *
     * @throws Exception pokut nejde nastavit barva kostičky
     */
    void nastavTestovaci() throws Exception;

    /**
     * Metoda nastaví model jako složenou kostku.
     *
     * @throws Exception pokut nejde nastavit barva kostičky
     */
    void nastavSlozeny() throws Exception;

    /**
     * Metoda nastaví na všechny strany modelu neznámou barvu.
     *
     * @throws Exception pokut nejde nastavit barva kostičky
     */
    void nastavPrazdny() throws Exception;

    /**
     * Metoda vrátí matici barev přední stěny modelu. Index [0,0] je levá dolní barva
     *
     * @return matice barev 3x3
     */
    Barva[][] getPredniBarvy();

    /**
     * Metoda vrátí barvu středové kostičky levé strany.
     *
     * @return Barva středové kostičky levé strany.
     */
    Barva getStredBarvaVlevo();

    /**
     * Metoda vrátí barvu středové kostičky pravé strany.
```

```
 *
 * @return Barva středové kostičky pravé strany.
 */
Barva getStredBarvaVpravo();

/**
 * Metoda vrátí barvu středové kostičky horní strany.
 *
 * @return Barva středové kostičky horní strany.
 */
Barva getStredBarvaNahore();

/**
 * Metoda vrátí barvu středové kostičky dolní strany.
 *
 * @return Barva středové kostičky dolní strany.
 */
Barva getStredBarvaDole();

/**
 * Metoda vrátí barvu středové kostičky přední strany.
 *
 * @return Barva středové kostičky přední strany.
 */
Barva getStredBarvaVpredu();

/**
 * Metoda vrátí barvu středové kostičky zadní strany.
 *
 * @return Barva středové kostičky zadní strany.
 */
Barva getStredBarvaVzadu();

/**
 * Metoda nastaví přední barvu přední vrstvě kostiček na pozici [x,y].
 *
 * @param x index x 0-vlevo, 1-střed, 2-vpravo
 * @param y index y 0-dole, 1-střed, 2-nahore
 * @param b nastavovaná barva
 */
void zmenBarvu(int x, int y, Barva b);

/**
 * Metoda otočí kostkou do výchozí pozice to je zelená barva vpředu a bílá barva dole.
 *
 * @return true - lze takto otočit, false - nelze otočit kostku do výchozí pozice.
 */
```

```
boolean natocDoNullPozice() throws Exception;


/**
 * Metoda provede veškeré validační kroky-
 *
 * @return true pokud byli všechny úspěšné.
 */
boolean getValidita() throws Exception;

/**
 * Metoda vrátí řetězec popisující nerovnosti ve validaci.
 *
 * @return popisující řetězec
 */
String getValidacniString() throws Exception;

/**
 * Vrací pole kostiček modelu.
 *
 * @return pole kostiček
 */
Kosticka[][][] getModel();

/**
 * Metoda vytvori kopii modelu kostky a generuje kroky potrebne ke slozeni.
 *
 * @return Vector kroku vedouci ke slozeni.
 * @throws Exception pokud nebylo možné složit model
 */
Vector<Krok> sloz() throws Exception;

/**
 * Metoda provede krok na modelu.
 *
 * @param krok který se má provést.
 */
void provedKrok(Krok krok);

/**
 * Metoda provede opačný krok na modelu.
 *
 * @param krok který se má provést.
 */
void provedOpacnyKrok(Krok krok);

/**
```

```
     * Metoda provede kroky ve Vektoru na modelu.
     *
     * @param kroky který se má provést.
     */
    void provedKroky(Vector<Krok> kroky);


    /**
     * Metoda vrátí kopii modelu.
     *
     * @return kopie modelu
     * @throws Exception pokut nejde nastavit barva kostičky
     */
    ModelKostky vratKopii() throws Exception;
}
```

Příloha C – *AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.libor.machovec_bakalarka" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
        <activity android:name=".CameraActivity"
            android:screenOrientation="landscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".EditorActivity"
            android:screenOrientation="landscape"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
            <intent-filter>
                <action android:name=".EditorActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature android:name="android.hardware.camera.autofocus"/>
    <uses-feature android:name="android.hardware.camera.front"/>
    <uses-feature android:name="android.hardware.camera.front.autofocus"/>
    <uses-permission android:name="android.permission.FLASHLIGHT" />
    <uses-permission android:name="android.permission.VIBRATE"/>



</manifest>
```