
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# MOBILNÍ APLIKACE PRO DEMONSTRACI ŘEŠENÍ RUBIKOVY KOSTKY

**RUBIK'S CUBE DEMONSTRATION ON MOBILE PLATFORMS**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **DAŠA LIPTÁKOVÁ**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Doc. Ing. FRATIŠEK ZBOŘIL, Ph.D.**
**SUPERVISOR**



**BRNO 2020**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

# Zadání bakalářské práce


Studentka: **Liptáková Daša**
Program: Informační technologie
Název: **Mobilní aplikace pro demonstraci řešení Rubikovy kostky**
**Rubix Cube Demonstration on Mobile Platforms**
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s algoritmy řešení problému Rubikovy kostky. Dále se seznamte s metodami zpracování obrazu, které by umožnily ze snímků Rubikovy kostky zjistit její aktuální konfiguraci.
2. Navrhněte aplikaci, která na základě snímků Rubikovy kostky předvede postup jejího řešení.
3. Implementujte navržené řešení pro systém Android tak, že po nasnímání stavu kostky aplikace navrhne nastavený počet tahů, které vedou k řešení.
4. Vyzkoušejte výslednou aplikaci na dostatečném počtu osob (alespoň deset) a shrňte jejich názory, postřehy a kritiky. Na základně tohoto navrhněte další vylepšení aplikace, pokud bylo požadováno a diskutujte užitečnost systému pro širší veřejnost.

Literatura:

- Russel, S., Norvig, P.: Artificial Intelligence, A Modern Approach, Pearson, 2009

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

# Abstrakt

Cieľom bakalárskej práce je navrhnúť a implementovať mobilnú aplikáciu, ktorá bude slúžiť ako návod na zloženie Rubikovej kocky. Aplikácie je určená pre operačný systém Android. Užívateľ nasníma za pomoci fotoaparátu šesť stien Rubikovej kocky. Aplikácia z nasnímaných fotografií určí aktuálnu konfiguráciu Rubikovej kocky, ktorú zobrazí ako grafický 3D model. Pokiaľ bude nájdená konfigurácia správna, aplikácia nájde sekvenciu algoritmov vedúcich k zloženiu Rubikovej kocky, ktoré sa budú postupne zobrazovať užívateľovi.



# Abstract

Bachelor thesis describes mobile application, which will be ser. The application is implemented for Android OS. User takes six photos of each side of Rubik's cube. The application will convert them into graphical 3D model and show it to user. After the user confirms the detected configuration, the application will solve it and shows the user how to solve it using graphical steps. Each step will be displayed to the user to follow along.



# Kľúčové slová

Rubikova kocka, Android, mobilná aplikácia, spracovanie obrazu, Cannyho hranový detektor, OpneCV, OpenGL ES

# Keywords

Rubik's Cube, Android, mobile application, image processing, Canny edge detector, OpenCV, OpenGL ES


# Citácia

LIPTÁKOVÁ, Daša. *Mobilní aplikace pro demonstraci řešení Rubikovy kostky.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Fratišek Zbořil, Ph.D.

# Mobilní aplikace pro demonstraci řešení Rubikovy kostky

## Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Doc. Ing. Fratišek Zbořil, Ph.D. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Daša Liptáková
28. mája 2020


## Poďakovanie

Ďakujem Doc. Ing. Fratiškovi Zbořilovi, Ph.D za jeho rady, pripomienky a čas, ktorý mi venoval na konzultáciách. Ďakujem svojej rodine za možnosť štúdia a hlavne za podporu, ktorú mi počas neho venovali.

# Obsah

# Kapitola 1

# Úvod

Počítačové videnie je obor, ktorého využitie v súčasnosti značne narastá. Svoj význam našiel v rôznych odvetviach. V zdravotníctve umožňil jednoduchšie určovanie diagnóz z príslušných snímkov. Počítačové videnie ďalej uľahčuje prácu policajtom, ktorý ho využívajú napríklad pri hľadaní osôb alebo pri identifikácii vozidiel vďaka rozpoznávaniu ŠPZ. Mňa osobne tento obor veľmi zaujal a chcela som si vyskúšať naimplementovať niečo s prvkami počítačového videnia.

Rubikova kocka je jeden z najznámejších hlavolamov na svete. Pre mnohých ľudí sa tento hlavolam ale stáva viac ako len obyčajnou hračkou. Stáva sa z nej istý druh koníčka, v ktorom má skladateľ priestor na stále zlepšovanie sa. Rubikova kocka je dobrá na rozvýjanie motoriky a pre mnohých sa stáva tiež antistresovou hračkou. Keďže sa skladaniu Rubikovej kocky venujem už od detstva, viem že nie je jednoduché sa naučiť ju zložiť. Bolo náročné zapamätať si všetky algoritmy, ale ešte náročnejšie bolo zo začiatku vyznať sa v označeniach stien a ich pohybe a algoritmus previesť správne.

Keďže som chcela pri práci spojiť zábavné s užitočným, napadlo ma naprogramovať aplikáciu, ktorá bude slúžiť ako návod na zloženie Rubikovej kocky. Podobných aplikácií síce niekoľko existuje, ale ani jedna z nich neponúka nasnímanie Rubikovej kocky. Ako som už spomenula, obor počítačového videnia ma zaujal a chcela som skúsiť vytvoriť aplikáciu s jeho prvkami. Spojila som tieto dve veci a rozhodla sa vytvoriť mobilnú aplikáciu pre Android, ktorá užívateľovi umožní nasnímať svoju Rubikovu kocku a následne mu predvedie návod na jej zloženie. Cieľovú skupinu užívateľov nechcem obmedziť a preto budem chcieť v aplikácii implementovať metódu pre začiatočníkov a tiež pre pokročilých. Cieľom mojej bakalárskej práce je naimplementovať začiatočnícku metódu, ale v budúcnosti plánujem doimplementovať aj metódu pre pokročilých.

Práca je rozdelená do ôsmych kapitol. Kapitola 2 popisuje Rubikovu kocku, rôzne označenia stien a metódy skladania. V ďalšej kapitole je detailnejšie vysvetlené spracovanie obrazu, metódy predspracovania obrazu, detektor hrán a tiež farebné priestory. Nasledujúca kapitola bližšie popisuje operačný systém Android a vývojové prostredie určené práve pre vývoj aplikácii pre tento operačný systém – Android Studio. Uvedené tri kapitoly poskytujú teoretické informácie o metódach využitých pri následnej implementácii. V ďalšej časti práce je detailne popísaný návrh a implementácia aplikácie. V predposlednej kapitole sú zhrnuté výsledky testovania, ktoré bolo rozdelené do dvoch častí. Prvou časťou testovania bolo aplikačné testovanie. Druhou časťou bolo testovanie aplikácie užívateľmi, ktorých spätnú väzbu som získala prostredníctvom dotazníku.

# Kapitola 2

# Rubikova kocka

Rubikova kocka je jeden z najznámejších a najobľúbenejších mechanických hlavolamov na svete. Táto magická kocka, ako sa jej kedysi hovorilo, sa teší svojej sláve už viac ako 40 rokov a nadšených skladateľov každým rokom pribúda.

Táto kapitola sa zaoberá tým, ako a prečo Rubikova kocka vlastne vznikla a ďalej obsahuje základné pojmy, skratky a algoritmy, ktoré je potrebné poznať k jej poskladaniu.

## 2.1 Od histórie po súčasnosť

Rubikovu kocku vynašiel Ernő Rubik v roku 1974. Ernő Rubik bol maďarský vynálezca, sochár, architekt a tiež profesor architektúry. Ako profesor premýšľal, ako inšpirovať svojich študentov a tak rád vytváral modely, ktoré by vyjadrovali jeho myšlienky. Jedným z týchto modelov bola práve aj známa Rubikova kocka. Tou chcel Rubik naučiť svojich študentov originálne premýšľať nad tým, ako vytvoriť niečo nové.

Prvá Rubikova kocka, ktorú zhotovil ručne sám Rubik, sa skladala z drevených kociek a gumičiek. Pre všetkých to bol len obyčajný objekt, ktorým sa dalo krútiť a otáčať jeho časťami. Neskôr si ale Rubik uvedomil, že je tento objekt veľmi ťažké pustiť z rúk a že by sa dal ľahko zmeniť na hlavolam, stačilo nalepiť farebné nálepky. A tak sa z obyčajnej školskej pomôcky stal najznámejší hlavolam na svete, ktorý sa dal dookola rozložiť a znovu poskladať. Tento hlavolam Rubik nazval Magic Cube alebo inak Magická kocka. Samotnému Rubikovi trvalo mesiac, kým prišiel na algoritmus, ktorým kocku zložil.

Prvé Magické kocky sa vyrábali v Maďarsku a začali sa predávať v roku 1977. Kocka okamžite zožala veľký úspech, no vzhľadom k vtedajšej dobe bolo veľmi náročné predávať kocku v zahraničí. V roku 1979 sa vďaka obchodníkovi Tiborovi Laczimu kocka dostala na veľtrh hračiek v Norimbergu, kde sa okamžite stala veľkým hitom. Záujem o kocku ihneď prejavila aj americká spoločnosť Ideal Toys, ktorá ju začala vyrábať a predávať. V roku 1980 sa Magická kocka premenovala na dnes známu Rubikovu kocku alebo tiež Rubik's Cube. Už v roku 1982 sa konali prvé Majstrovstvá sveta v skladaní Rubikovej kocky, ktoré sa konajú dodnes. Rubikova kocka sa stala jednou z najpredávanejších hračiek všetkých čias a môže sa tiež pýšiť titulom Hračky roka z rokov 1980 a 1981.

Rubikova kocka mala vplyv na rôzne odvetvia. Umelci vytvárali rôzne diela ako najväčšiu či najmenšiu funkčnú Rubikovu kocku. Zostavujú sa roboty, ktoré vedia kocku zložiť v čo najkratšom čase. Matematici sa stále snažia odhaliť maximálny počet pohybov na vyriešenie rôznych hlavolamov inšpirovaných práve Rubikovou kockou. V súčasnej dobe existuje aj takzvané rýchlostné skladanie tiež známe pod názvom speedcubing alebo speedsolving.

Každoročne sa po celom svete koná mnoho súťaží v tomto rýchlostnom skladaní a rekordy zloženia kocky sa v súčasnosti výrazne znížili. Súťaže prebiehajú tak, že každý z účastníkov dostane od rozhodcu zakrytú už rozloženú kocku a následne má približne 15 sekúnd na to, aby si ju poriadne preskúmal. Po zložení kocky rozhodca zapíše čas zloženia a prinesie ďalšiu. Súťažiaci skladá kocku presne 5x. Na týchto súťažiach sa merajú dva typy rekordov a to: *Single*, čo je najlepší čas zo všetkých 5-tich pokusov a *Average*, ktorý sa vypočíta tak, že sa škrtne najlepší a tiež najhorší čas a zo zostávajúcich 3-roch sa urobí priemer. Aktuálnym držiteľom svetového rekordu *Single* je čínsky speedcuber Yusheng Du, ktorý dokázal zostaviť kocku za 3,47 sekundy. Držiteľom rekordu *Average* je v dobe písania práce Feliks Zemdegs, Austrálčan, ktorý tento rekord drží s časom 5,53 sekundy. [10]

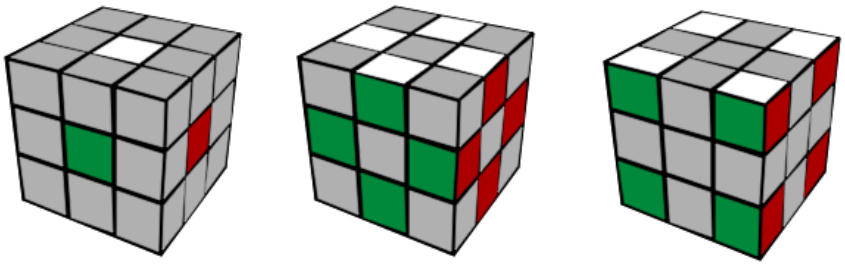
Rubikovu kocku v súčasnosti vyrábajú viacerí výrobcovia. Každý z nich sa snaží prísť vždy s lepším mechanizmom otáčania a inými vylepšeniami ako sú napríklad magnety. Vďaka týmto vylepšeniam sa steny kocky ľahšie a presnejšie otáčajú, čím umožňujú skladateľovi zlepšovať čas, za ktorý kocku postaví. Postupom času sa začali vytvárať aj ďalšie podobné hlavolamy. V dnešnej dobe existujú Rubikove kocky rôznych veľkostí – kocka 2x2x2, 4x4x4, 5x5x5 a ďalšie hlavolamy inšpirované práve Rubikovou kockou ako Rubikova veža – kváder o veľkosti 2x2x4 alebo Rubikova kocka void, ktorej stredové kocky sú nahradené dierami. Táto kapitola sa bude ďalej zaoberať len klasickou Rubikovou kockou 3x3x3. [15]

## 2.2 Rubikova kocka 3x3x3

Štandardná Rubikova kocka 3x3x3 je šesťfarebná – jednej stene kocky zodpovedá jedna farba. Skladá sa z 26-tich menších kociek a uprostred týchto kociek sa nachádza mechanizmus otáčania jednotlivých stien. Tento hlavolam je možné mechanicky rozobrať na príslušné časti, je ale potrebné dávať pozor pri ich opätovnom zložení, pretože pokiaľ by bola niektorá z kociek vložená na iné miesto alebo nesprávne otočená, mohlo by sa stať, že sa hlavolam nebude dať vyriešiť. Preto, pokiaľ je Rubikovu kocku z nejakého dôvodu nutné mechanicky rozobrať, odporúča sa potom skladať ju späť podľa farieb.

Malé kocky, z ktorých je Rubikova kocka zložená, delíme na tri typy a to na stredové, hranové a vrcholové. Stredových kociek je 6, v strede každej steny Rubikovej kocky jedna. Sú to jednofarebné kocky, ktoré na každej stene Rubikovej kocky odpovedajú práve jednej zo šiestich farieb. Stredové kocky sú jediné kocky, ktoré sa nedajú presúvať. Hranových kociek je 12, v strede každej hrany Rubikovej kocky sa nachádza práve jedna. Tieto kocky sú dvojfarebné, farby sú vždy rozdielne a odpovedajú práve dvom farbám susedných stien Rubikovej kocky. Jedna hranová kocka spája dve stredové kocky. Posledným typom sú vrcholové kocky, tých je 8 a nachádzajú sa každá v jednom vrchole Rubikovej kocky. Vrcholové kocky sú trojfarebné, tieto farby odpovedajú trom rozdielnym farbám stien vytvárajúcich daný vrchol Rubikovej kocky.

S týmito malými kockami a s ich rôznymi farbami súvisia aj dva podstatné pojmy – orientácia a permutácia. Orientácia alebo inak natočenie znamená, že kocka sa nachádza na správnom mieste, ale jej farby neodpovedajú farbe steny a je potrebné túto kocku správne otočiť. Permutácia znamená presunutie alebo prehodenie. V tomto prípade môže byť kocka otočená správne, ale nenachádza sa na svojom mieste. Rubikovu kocku je možné rozložiť na 43 252 003 274 489 856 000 (43 triliónov) rôznych kombinácií. Pre správnu pozíciu a otočenie kocky existuje viacero metód skladania, tie budú popísane v sekcii 2.3.



Obr. 2.1: Na obrázku sú znázornené jednotlivé typy 26-tich malých kociek, z ktorých sa skladá Rubikova kocka – zľava stredové, hranové a vrcholové kocky.[1]

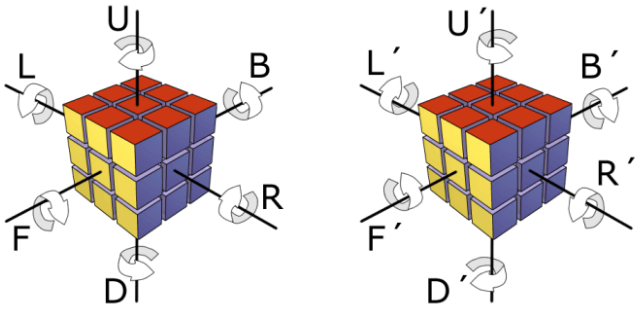
## 2.3 Princíp riešenia

Cieľom tohto kombinačného hlavolamu je správne presunúť a otočiť všetky malé kocky tak, aby na všetkých stenách bola práve jedna zo šiestich farieb. Všetkých 6 stien sa dá otáčať okolo vlastnej osi v smere hodinových ručičiek a tiež v protismere. Pokiaľ budeme stenami otáčať v správnom poradí krokov, docielime tým poskladanie Rubikovej kocky. Tieto kroky si môže skladateľ intuitívne vymyslieť sám, alebo môže použiť už existujúce postupy.

V súčasnosti existuje mnoho metód a postupov, ktoré obsahujú rôzne počty algoritmov a zameriavajú sa na rôzne aspekty skladania ako napríklad rýchlosť zloženia, jednoduchosť naučenia sa daného postupu či minimálny počet krokov. Jednoduchšie metódy sa zameriavajú hlavne na minimálny počet krokov potrebných pre zloženie kocky. Tieto kroky sa pri skladaní často opakujú a tým sa zvyšuje čas zostavenia, ale sú oveľa jednoduchšie na zapamätanie. Najjednoduchšou metódou, vhodnou pre začiatočníkov, je metóda Layer by layer alebo inak vrstva po vrstve. Táto metóda bude popísaná v sekcii 2.3.1. Pokročilejších metód je v dnešnej dobe podstatne viac, poznáme napríklad metódy ZZ, Roux, Petrus, ZB, CFOP/Fridrich a ďalšie. Tieto metódy sú zložitejšie hlavne na zapamätanie, pretože obsahujú viac algoritmov. Sú ale efektívnejšie a zaručia oveľa nižší čas zostavenia. Skladateľ okrem algoritmov musí mať dobrú predstavivosť a vedieť si naplánovať aj viacero krokov dopredu. Jednou z najznámejších metód, ktorú používa aj mnoho súťažiacich, je práve CFOP alebo inak metóda Fridrich. Tá je bližšie popísaná v podkapitole 2.3.2.

Pre pochopenie daných algoritmov je potrebné poznať určité označenia. Medzi tie najzákladnejšie patria označenia jednotlivých stien: F – Predná stena, B – Zadná stena, L – Ľavá stena, R – Pravá stena, U – Horná stena a D – Dolná stena. Tieto označenia sú odvodené od anglických názvov Front, Back, Left, Right, Up a Down. Pokiaľ je označenie steny uvedené samostatne napríklad R, znamená to, že sa pravá stena má otočiť o 90° v smere hodinových ručičiek. Pre otočenie steny o 90° proti smeru hodinových ručičiek sa používa označenie steny doplnené o apostrof alebo v niektorých prípadoch o malé i, teda R' alebo Ri prípadne označovanie R+ pre pohyb v smere hodinových ručičiek a R- pre pohyb proti smeru hodinových ručičiek. Pokiaľ je potrebné stenu otočiť o 180°, bude označenie doplnené o číslicu 2, teda napríklad 2R alebo R2, znamená dvakrát otočenie pravej steny v smere hodinových ručičiek. Platí tu rovnosť 2R = 2R'. Toto označenie je najčastejšie používaným označením vo všetkých metódach a pre jednoduchšie, začiatočnícke metódy je úplne postačujúce. Pre pokročilejšie zložitejšie algoritmy boli postupne zavedené aj ďalšie označenia.

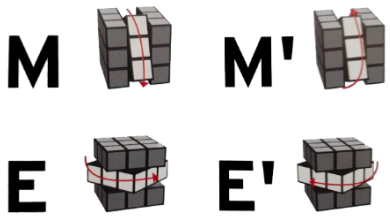
[1] Obrázok 2.1 bol prevzatý z https://hlavolam.maweb.eu/rubikova-kostka-3x3x3.




Obr. 2.2: Na obrázku sú znázornené jednotlivé označenia stien a smer otáčania.[2]

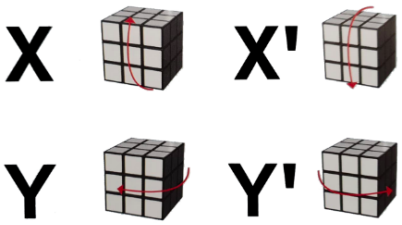
Medzi ďalšie označenia patria: otáčanie dvoch vrstiev, rezné otáčanie a rotácie. Otáčanie dvoch vrstiev je svojím označením veľmi podobné označeniu stien, jediným rozdielom je veľkosť písmen. Sú to teda označenia: f – dve predné vrstvy, b – dve zadné vrstvy, l – dve ľavé vrstvy, r – dve pravé vrstvy, u – dve horné vrstvy a d – dve dolné vrstvy. Ako už názov napovedá, pri tomto druhu otáčania sa otáčajú dve susedné vrstvy súčasne.

Rezné otáčanie je otáčanie strednej vrstvy a označuje sa M – otočenie strednej vrstvy vo vertikálnom smere zozadu dopredu, S – taktiež označuje otočenie vo vertikálnom smere, ale zľava doprava a E – otočenie strednej vrstvy v horizontálnom smere zľava doprava. Pri týchto otáčaniach sa nesmú vonkajšie vrstvy pohnúť.




Obr. 2.3: Na obrázku sú zobrazené M a E – najpoužívanejšie označenia.

Niektoré algoritmy pokročilejších metód vyžadujú otočenie celej Rubikovej kocky. Kvôli tomu boli zavedené rotácie, ktoré označujeme písmenami X – otočenie kocky vo vertikálnom smere zdola nahor, Y – otočenie kocky v horizontálnom smere sprava doľava a Z – otočenie kocky vo vertikálnom smere sprava doľava.




Obr. 2.4: Na obrázku sú zobrazené X a Y – najpoužívanejšie označenia.

Ako už je z uvedených obrázkov vidieť, všetky doplňujúce označenia majú rovnaký význam ako pri základných označeniach stien.

[2]Obrázok 2.2 bol prevzatý z https://hlavolam.maweb.eu/rubikova-kostka-3x3x3.

V niektorých algoritmoch sa tiež môžu vyskytovať zátvorky, napríklad (R, U, R', U'). Zátvorky na algoritme nič nemenia, postup krokov ostáva rovnaký. Tieto zátvorky slúžia pre lepšie zapamätanie si niektorých bežne používaných menších sekvencií krokov. Tieto menšie sekvencie sa vždy skladajú zo 4-roch otočení stien, sú podobné ako na vyššie uvedenom príklade, ale môžu obsahovať ľubovoľné steny kocky. V oblasti speedsolvingu sú tieto sekvencie pomenované *Sexy move*.

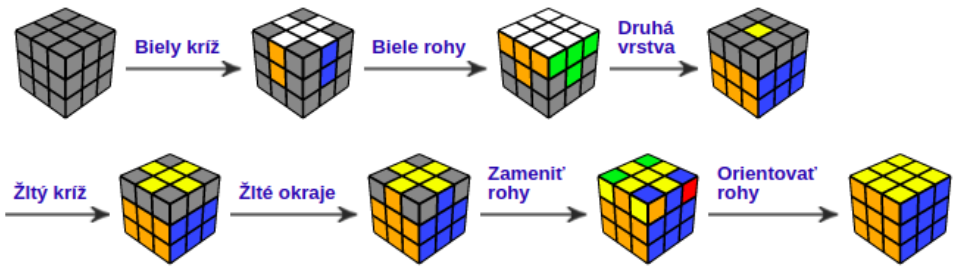
### 2.3.1 Metóda pre začiatočníkov – Layer by layer

Ako už samotný názov metódy napovedá, princípom tejto metódy je skladať vrstvu po vrstve, anglicky layer by layer. Je to najrozšírenejšia metóda pre začiatočníkov. Dalo by sa povedať, že má viacero verzií, alebo inak povedané, existuje viacero algoritmov, ktorých prevedenie dosiahne rovnakého prehodenia daných kociek. Je potrebné pamätať si približne len šesť algoritmov, asi preto je táto metóda takou obľúbenou u začiatočníkov.

Prvým krokom je zostaviť na prvej vrstve kríž – okolo stredu na vrchnej stene sú na správne miesto dosadené príslušné hranové kocky so správnou orientáciou. Zvyčajne sa odporúča začínať bielou stenou, ale nie je to podmienkou. Po zostavení kríža sa dokončí zloženie prvej vrstvy uložením správne natočených vrcholov na ich miesto. Tieto kroky sú intuitívne, pretože neexistujú žiadne algoritmy, ktorými by skladateľ prvú vrstvu dokázal postaviť. Dajú sa vyhľadať ukážky, ako na to.

Ďalším krokom je zostavenie druhej vrstvy. To sa dá docieliť presunutím štyroch hranových kociek druhej vrstvy na správne miesto a ich správnym otočením.

Na tretej vrstve sa najprv správne otočia a presunú hranové kocky, čím vznikne kríž. Záverečnými krokmi je postupne prehodiť a správne zorientovať vrcholy poslednej vrstvy.




Obr. 2.5: Na obrázku je znázornený postup krokov metódy „Layer by layer“.[3]

### 2.3.2 Metóda CFOP/Fridrich

Táto metóda dostala názov „Fridrich method“ po Jessice Fridrich, českej rodáčke, ktorá ju spopularizovala. Na rozdiel od predchádzajúcej metódy umožňuje skladateľovi poskladať kocku s čo najmenším počtom krokov a tým výrazne znížiť čas skladania. Obsahuje ale podstatne viac rôznych algoritmov. Táto metóda má aj ďalší, známejší názov „CFOP“. Tento názov je odvodený od štyroch krokov riešenia: C – Cross (kríž), F – First two layers (prvé dve vrstvy, skratka F2L), O – Orient last layer (orientácia poslednej vrstvy, skratka OLL), P – Permute last layer (permutácia poslednej vrstvy, skratka PLL). Ako už odvodenie názvu napovedá, rozdielom oproti predošlej metóde je, že sa vrcholy prvej vrstvy skladajú

[3]Obrázok 2.5 bol prevzatý z https://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/.

súčasne s hranami druhej vrstvy. Na poslednej vrstve sa najprv správne otočia všetky malé kocky a potom sa jedným ďalším algoritmom prehodia na svoje miesta.

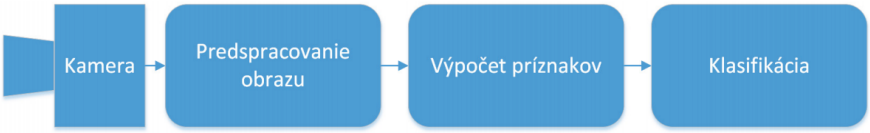

Obr. 2.6: Na obrázku je znázornený postup krokov metódy CFOP.[4]

[4]Obrázok 2.6 bol prevzatý z https://ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/.

# Kapitola 3

# Spracovanie obrazu

Spracovanie obrazu spadá do odvetvia počítačového videnia, ktoré je súčasťou umelej inteligencie. Počítačové videnie sa po technickej stránke snaží napodobniť funkcionalitu ľudského zraku. Zaoberá sa teda získavaním rôznych informácií zo zachyteného obrazu. Medzi obrazové dáta, ktoré je možné spracovať, patrí napríklad fotka, video, kamerový záznam a ďalšie. Proces spracovania obrazu sa skladá z niekoľkých fáz – *Predspracovanie obrazu*, *Výpočet príznakov* a *Klasifikácia*. Postupnosť týchto fázy je zobrazená na nasledujúcom obrázku 3.1.




Obr. 3.1: Na obrázku sú znázornené jednotlivé fázy spracovania obrazu, od samotného nasnímania obrazu až po jeho klasifikáciu. [16]

Po tom ako z digitálnej kamery získame signál obrazu, je vhodné obraz predspracovať. *Predspracovanie obrazu* zahŕňa odstránenie rušivých a nežiaducich javov, ktoré by mohli výrazne narušiť celkové spracovanie. Jedná sa napríklad o odstránenie šumu zo signálu obrazu alebo prevod na iný farebný priestor. Pod fázou *Výpočet príznakov* sa rozumie výpočet, ktorého výsledkom je nejaká vlastnosť objektu ako napríklad tvar objektu, jeho hrany, vrcholy, farba a ďalšie. *Klasifikácia* znamená isté porozumenie obrazu počítačom, teda určenie objektu na základe získaných vlastností z vypočítaných príznakov. V týchto fázach sú využívane rôzne metódy, ako napríklad filtrácia pre odstránenie šumu, segmentácia obrazu, detekcia hrán či transformácia obrazu. Táto kapitola podrobnejšie popisuje hlavne metódy, ktoré boli následne využité pri implementácii. [16]

## 3.1 Predspracovanie obrazu

Predspracovanie obrazu je dôležitou fázou spracovania obrazu. Ako už bolo spomenuté, pri snímaní, prenose alebo inom spracovaní obrazu môžu nastať nežiaduce javy, medzi ktoré patrí aj šum. Existuje viacero druhov šumu a tiež viacero filtrov, vďaka ktorým je možné šum z obrazu odstrániť. Jeden z týchto šumov je Gaussov šum, ktorý postihuje všetky

pixely obrazu. Tento šum veľmi efektívne potláča Gaussovo rozostrenie obrazu, tomu sa detailnejšie venuje nasledujúca podkapitola 3.1.1.

Medzi ďalšie metódy predspracovania obrazu patrí aj prahovanie. Prahovanie vytvára binárny obraz – čierne a biele farby pixelov a tým zaistí, že jeho veľkosť je menšia. Vzhľadom k menšej veľkosti obrazu sa urýchli jeho následné spracovanie. Preto je tento binárny obraz žiadaným vstupom pre viaceré metódy určené na výpočet príznakov. Princípu prahovania sa venuje podkapitola 3.1.2, ktorá tiež vysvetlí rozdiel medzi klasickým a adaptívnym prahovaním.

### 3.1.1 Gaussovo rozostrenie obrazu

Jedná sa o jednu z metód, ktoré slúžia na potlačenie nežiaduceho šumu v obraze. Gaussovo rozostrenie obrazu je založené na konvolúcii s vhodným konvolučným jadrom. Pri konvolúcii si predstavíme obraz ako maticu hodnôt, ktorú následne násobíme inou maticou – konvolučným jadrom. Na výpočet hodnôt konvlučného jadra sa používa Gaussova funkcia, ktorá má pre 2D tvar:

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

kde $x$ a $y$ odpovedajú vzdialenosti od pôvodného pixelu na odpovedajúcej osi a $\sigma$ je smerodajná odchýlka, ktorá určuje polomer rozostrenia v pixeloch. V praxi sa tento vzorec aproximuje do matice znázornenej na príklade 3.1, ktorej hodnoty vytvárajú sústredené kruhy smerujúce od stredu matice. Nová hodnota každého pixelu sa vypočíta ako vážený priemer hodnôt susedných pixelov. Váhy daných pixelov sú získané z konvolučného jadra. V strede matice sa nachádza hodnota pôvodného pixelu, ktorá má najťažšiu váhu a váha ostatných pixelov sa zmenšuje v závislosti na vzdialenosti od pôvodného pixelu.

$$h = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{3.1}$$

Na príklade 3.1 je znázornené konvolučné jadro s veľkosťou 5x5 a so smerodajnou odchýlkou $\sigma = 1$. Túto maticu je potrebné vydeliť súčtom všetkých hodnôt z jej vnútra, tým dostaneme Gaussovo rozostrenie alebo inak Gaussov filter. Hodnota konvolučného jadra sa totiž musí rovnať jednej. Pokiaľ by táto hodnota bola menšia, spôsobila by, že výsledný obraz bude tmavší a naopak s vyššou hodnotou by bol výsledný obraz svetlejší. [13]

### 3.1.2 Prahovanie

Metód prahovania existuje viacero, napríklad globálne prahovanie, adaptívne prahovanie či prahovanie s hysteréziou. Všetky metódy sú si do určitej miery podobné, ich hlavný rozdiel spočíva v získavaní prahovej hodnoty, čo následne ovplyvní aj výsledný obraz. Preferovaným vstupom pre prahovanie je obraz v odtieňoch sivej farby, ale existujú aj techniky prahovania pre farebný obraz. Výstupom je typický binárny obraz odpovedajúci práve čiernej a bielej farbe. Princíp tohto prevodu spočíva v tom, že na základe intenzity farby každého pixelu sa porovnaním s hodnotou prahu určí výsledná farba daného pixelu. Pokiaľ je intenzita pixelu menšia ako prahová hodnota, tak je tomuto pixelu na výstupe priradená hodnota bielej

farby. Naopak, pokiaľ je táto intenzita vyššia, hodnota pixelu bude na výstupe nastavená na farbu čiernu. Tento princíp je možné popísať nasledujúcou funkciou:

$$f(x) = \begin{cases} A & \text{pre } x < \textit{hodnota prahu} \\ B & \text{pre } x \geq \textit{hodnota prahu} \end{cases}$$

kde premenná $x$ prestavuje pôvodnú hodnotu intenzity farby daného pixelu, *f(x)* je jeho hodnota na výstupnom binárnom obraze, $A$(biela) a $B$(čierna) sú nové hodnoty pixelu.

*Adaptívne prahovanie* v porovnaní s klasickým dosahuje pomerne lepšie výsledky. Rozdiel spočíva v prahovej hodnote. Na rozdiel od klasického prahovania, ktoré pre každý pixel používa tú istú globálnu prahovú hodnotu, adaptívne prahovanie vypočíta prahovú hodnotu pre každý pixel samostatne. Výpočet tejto hodnoty je založený na vopred definovanom okolí daného pixelu, ktoré sa môže líšiť veľkosťou aj tvarom. Pre tento výpočet existuje viacero metód, napríklad môže byť hodnota prahu vypočítaná na základe stredovej hodnoty pixelov z danej oblasti alebo môže byť určená na základe váženého súčtu týchto pixelov. Pre tento vážený súčet je potrebné každému pixelu priradiť určitú váhu, ktorá je priradená pomocou Gaussového okna. [14]

Existuje tiež *prahovanie s hysteréziou*, ktoré sa od klasického prahovania líši v tom že nepoužíva jednu, ale dve prahové hodnoty. Jednou je horná prahová hodnota(maximum), druhou je dolná prahová hodnota (minimum). Pokiaľ je intenzita pixelu vyššia ako hodnota horného prahu, tak výsledná hodnota pixelu bude mať bielu farbu. Pixely, ktoré nesú bielu farbu, môžeme tiež nazvať významnými. Ak je intenzita daného pixelu nižšia ako dolný prah, pixel pre nás významný nie je a preto jeho výsledná hodnota bude mať čiernu farbu. Pokiaľ intenzita pixelu spadá do intervalu $\langle \textit{maximum, minimum} \rangle$, je potrebné určiť, či je tento pixel významný alebo nie. Pixel je považovaný za významný, ak sa v jeho susedste nachádza aspoň jeden pixel s vyššou intenzitou ako je hodnota horného prahu. V takom prípade bude výsledná hodnota pixelu biela. Naopak, pokiaľ sa v jeho susedstve nenachádza ani jeden takýto pixel, bude spracovávaný pixel považovaný za nevýznamný a vo výslednom obraze bude niesť hodnotu čiernej farby. [11]

## 3.2 Detekcia hrán

Detekcia hrán sa využíva k rozoznávaniu obsahu obrazu. Pri spracovaní obrazu sa za hranu považujú pixely, kde nastáva prudká zmena intenzity jasu. Spojením pixelov, ktoré disponujú touto prudkou zmenou jasu, vznikajú hrany objektu. Výstupom hranových detektorov býva väčšinou binárny obraz (čierna a biela), ale existujú aj prípady, kde je potrebné dané hrany rozlišovať a preto sú znázornené viacerými farbami. Vo výslednom binárnom obraze sú hrany zvyčajne znázornené bielou farbou na čiernom pozadí. Na základe detekovaných hrán je teda možné získať štruktúru objektov vyskytujúcich sa v obraze. Za pomoci týchto štruktúr je následne možné detekovat požadované objekty.

V súčasnosti exituje viacero hranových detektorov. Tieto detektory sa odlišujú tým, aký postup na detekciu hrán v obraze využijú. Používajú síce rôzne postupy, ale všetky sa snažia dosiahnuť najlepšie možné výsledky v troch zásadných kritériách pre ideálnu detekciu hrán daného obrazu.

Prvým kritériom je čo najmenšia chybovosť detekovaných hranách. Teda nemali by byť detekované falošné hrany a žiadna z existujúcich hrán by nemala byť pri detekovaní vynechaná. Druhým kritériom je správna lokalizácia hrany, teda vzdialenosť detekovanej hrany od tej reálnej by mala byť minimálna. Tretím kritériom je zabezpečenie, aby bola každá hrana detekovaná len raz.

Spočiatku hranové detekektory tieto kritéria nespĺňali. Tieto tri kritéria vymyslel John F. Canny, v roku 1986 ich popísal vo svojom článku *A Computational Approach to Edge Detection*. V závislosti na svojej teórii o troch kritériách vyvinul Canny svoj vlastný hranový detektor, ktorý je po ňom pomenovaný a aj v súčasnosti je jeden z najvyužívanejších. [11]

### 3.2.1 Cannyho hranový detektor

Informácie uvedené v tejto podkapitole som čerpala z vyššie uvedeného článku [11], ktorý napísal sám John F. Canny. Ako už bolo spomenuté Cannyho hranový detektor sa riadi troma kritériami – minimálna chybovosť, presná lokalizácia a jednoznačné detekovanie hrany. Postup tohto detektora sa skladá zo štyroch nasledujúcich krokov:

1. **Eliminácia šumu**

   Pre presnejšiu detekciu hrán je potrebné vstupný obraz zbaviť šumu, tým sa predíde prípadnej detekcii falošnej hrany. Cannyho hranový detektor väčšinou využíva na elimináciu šumu Gaussovo rozostrenie obrazu, ktoré je podrobnejšie popísané v podkapitole 3.1.1.

2. **Výpočet veľkosti a smeru gradientu**

   Jedná sa teda o štandardnú detekciu hrán pomocou metód založených na prvej derivácii. Môže sa jednať napríklad o Robertsov, Kirschov, Prewittov alebo Sobelov operátor. Každý z nich má svoje výhody a nevýhody, nedá sa teda určiť jeden najlepší, ale najčastejšie je používaný Sobelov operátor. V práci je pre spracovanie obrazu použitá knižnica OpenCV, ktorá bude bližšie popísaná v sekcii 6.1.1. Táto knižnica pri použití Cannyho detektoru využíva práve Sobelov operátor, preto bude ďalej popísaný len tento jeden operátor.

   Pre zaistenie hodnoty gradientu v každom bode obrazu, využíva Sobelov operátor dve konvolučné jadrá o veľkosti 3x3, ktorých hodnoty sú znázornené v nasledujúcich maticiach:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

   Tieto dve matice môžu byť použité každá zvlášť, jedna pre horizontálny a druhá pre vertikálny smer hrany. Ale môžu byť použité aj súčasne, výsledkom potom bude absolútna hodnota gradientu v danom pixely. Vo výslednom obraze budú farebne vyznačené hrany, kde farba určuje uhol danej hrany. Veľkosť a smer gradientu sa vypočíta pomocov vzťahov:

$$G = \sqrt{G_x^2 + G_y^2} \qquad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

   kde G je celková veľkosť intenzity gradientu a $\theta$ je smer alebo inak uhol gradientu. Tento uhol býva zaokrúhlený na 0°, 45°, 90° alebo 135° a podľa tejto veľkosti je následne určená farba daného pixela.

3. **Nájdenie lokálnych maxím**

   Tento krok sa nazýva aj stenšenie hrán (thinning), ale najčastejšie sa stretávame s anglickým názvom *nonmaximum suppression*. V tomto kroku dochádza k nájdeniu lokálnych maxím. Lokálne maximum sa určí porovnaním hodnôt gradientu susedných pixelov, ktoré sú vybraté na základe smeru gradientu. Tieto hodnoty boli získané v predchádzajúcom kroku. Pixely, ktorých veľkosť intenzity gradientu nedosahuje lokálne maximum, sú odobraté a naopak pixely, dosahujúce lokálne maximum, sú ponechané. Tým dochádza k stenšeniu detekovaných hrán. Vďaka stenšeniu sú hrany oveľa presnejšie.

4. **Prahovanie s hysteréziou**

   Posledným krokom Cannyho detektoru je prahovanie s hysteréziou. Práve táto metóda zaistí, že pixely, ktoré nie sú skutočnou hranou, sa odstránia. Táto metóda používa dve prahové hodnoty, ktoré zaistia, že budú na výslednom obraze ponechané len významné pixely, teda skutočné hrany. Princíp, ktorým prahovanie s hysteréziou rozhoduje o tom, ktorý pixel je významný a ktorý nie, je detailnejšie popísaný v podkapitole 3.1.2.

## 3.3 Modely a priestory farieb

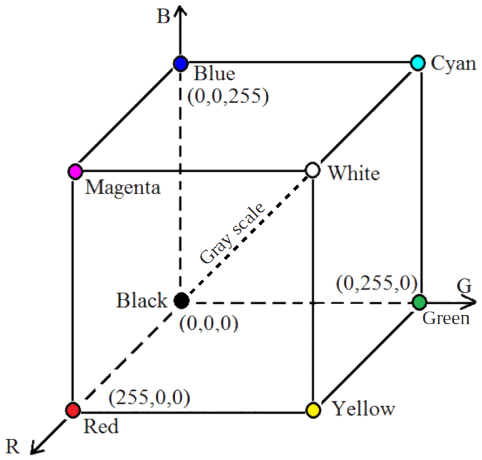
Farby sú pre ľudí veľmi dôležité a stretávajú sa s nimi každý deň. Vďaka farbám ľudia istým spôsobom komunikujú s okolitými objektmi. Vnímanie farieb je síce prirodzenou vlastnosťou ľudského zraku, ale na to, aby boli farby premietnuté do počítača alebo iných zariadení, bolo potrebné ich vyjadriť tak, aby im dané zariadenie rozumelo. Na základe tejto požiadavky vznikli farebné modely, ktoré vyjadrujú farby numericky a to pomocou matematických vzorcov. Tieto modely pomocou primárnych komponentov farieb dokážu vytvoriť celú škálu rôznych farieb, ktorá sa nazýva farebný priestor. Ten následne dokáže počítač vizualizovať. Zvyčajne majú modely tri alebo štyri farebné komponenty.

V súčasnosti existuje viacero farebných modelov, ktoré sú využívané v rôznych zariadeniach či aplikáciách. Podľa spôsobu kombinovania farieb delíme farebné modely na dva typy – aditívne a subtraktívne. Ďalším spôsobom delenia je intuitívnosť daného modelu. Poznáme hardwarovo orientované modely, ktoré zvyčajne využívajú zariadenia pri zobrazovaní obrazového signálu a ďalej tu sú používateľsky orientované modely, vďaka ktorým je pre užívateľa jednoduchšie vybrať danú farbu. Detailnejšia odlišnosť týchto modelov spočíva zvyčajne v rozsahu farieb, ktoré môžu byť pomocou neho nakombinované. Pre rozlišovanie konkrétnej farby sa používajú tri vlastnosti – jas, odtieň a sýtosť. Tieto vlastnosti sú pri farebných modeloch rozdelené na dva komponenty – jas a chrominancia, ktorá zahŕňa odtieň a sýtosť.

Pri spracovaní digitálneho obrazu nám použitie farebného spektra môže zjednodušiť analýzu obrazu, napríklad rozpoznávanie objektov z obrazu na základe ich farby. Kamera alebo fotoaparát snímajúci digitálny obraz dokážu rozpoznať tri primárne komponenty farieb a to červenú, zelenú a modrú. Jedná sa o najčastejšie využívaný farebný model pre kódovanie farieb v obraze – model RGB, ktorý bude detailnejšie popísaný v nasledujúcej podkapitole. Podkapitola 3.3 čerpá informácie z publikácie [12].

### 3.3.1 Model RGB

Jedná sa o hardwarovo orientovaný model, ktorý využíva adaptívne kombinovanie farieb. Čím vyššiu hodnotu majú farebné zložky, tým svetlejšia je výsledná farba. Jeho primárne komponenty farieb sú červená (Red), zelená (Green) a modrá (Blue), ktorých kombinovaním vznikajú takzvané sekundárne farby a to modrozelená (Cyan), fialová (Magenta) a žltá (Yellow). V priestore je tento model zobrazený pomocou jednotkovej kocky, ako je vidieť na obrázku 3.2. Každý vrchol kocky predstavuje jednu z vyššie uvedených farieb a zvyšné dva vrcholy nesú čiernu a bielu farbu. Osi tejto súradnicovej sústavy odpovedajú množstvu príslušnej primárnej farby. Stred súradnicovej sústavy, teda vrchol [0,0,0] odpovedá čiernej farbe a vrchol naproti, teda vrchol [1,1,1] odpovedá farbe bielej. Diagonála medzi týmito vrcholmi zodpovedá farbám v odtieňoch sivej.




Obr. 3.2: Na obrázku je znázornený farebný model RGB v priestore ako jednotková kocka, súradnice jednotlivých bodov sú v implementačnom kódovaní.[1]

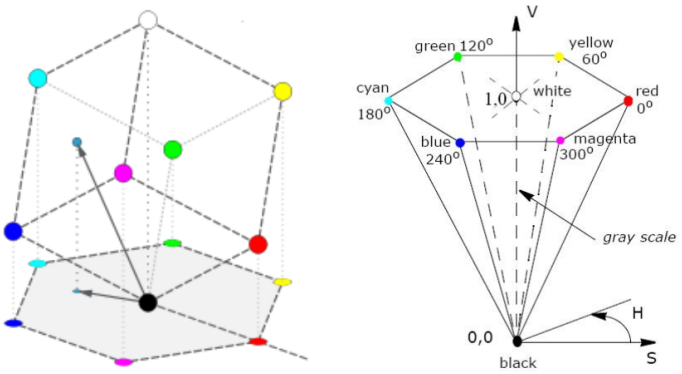
Model RGB je jeden z najviac technicky orientovaných modelov. Využíva sa v zariadeniach, ktoré využívajú k zobrazovaniu daného obrazového signálu svetlo, napríklad monitory či televízie. Pri implementácii je rozsah kódovaný pomocou 8 bitov, teda hodnoty 0-255. Na základe toho je v tomto modeli možné vytvoriť 16,8 milióna rôznych farieb. Tento model však nie je úplne vhodný pre prácu s obrazom, preto je často potrebné ho previesť na iný, viac vyhovujúci model. To, aký model je najvhodnejší, sa môže líšiť, záleží od danej úlohy.

### 3.3.2 Model HSV

Pre užívateľa môže byť vyjadrenie niektorých farieb pomocou primárnych komponentov náročné, preto vznikli aj farebné modely, ktoré sú intuitívnejšie a umožňujú užívateľovi si danú farbu jednoducho vybrať. Tieto modely nazývame používateľsky orientované a sú vlastne lineárnou transformáciu z farebného priestoru RGB. Používajú tri základné parametre – odtieň (hue), sýtosť (saturation) a jasovú zložku (value, lightness, brightness, luminance) farby. Medzi tieto farebné modely patria HSV, HSL, HSI a HCI. V nasledujúcej časti bude popísaný farebný model HSV, ktorý je veľmi často využívaný v počítačovej grafike.

[1]Obrázok 3.2 bol prevzatý z https://www.researchgate.net/figure/Cartesian-coordinate-system-of-RGB-color-space_fig1_322688596.

Parametre farebného modelu HSV, ktorý sa niekedy nazýva aj HSB, sú odtieň (Hue), sýtosť (Saturation) a jasová hodnota (Value). Ako je vidieť, názov modelu je odvodený z parametrov, ktoré sú v modeli využité. V priestore je tento model väčšinou zobrazený ako obrátený pravidelný šesťboký ihlan, ktorý je vytvorený transformáciou RGB kocky, ako to je znázornené na obrázku 3.3. Jeho vrchol leží v bode [0,0,0], teda v začiatku súradnicovej sústavy a odpovedá čiernej farbe. Odtieň farby je uhlová hodnota, kde červená farba je pri 0°. Sýtosť je určená vzdialenosťou danej farby od stredovej osi. Samotná stredová os určuje jasovú hodnotu, ktorá klesá od bielej farby v strede podstavy ihlanu k čiernej farbe na jeho vrchole.




Obr. 3.3: Na obrázku vľavo je zobrazený model RGB tak, aby diagonála predstavovala stredovú os ihlanu na obrázku vpravo, teda modelu HSV.[2]

### Prevod z modelu RGB do modelu HSV

Na prevod z modelu RGB do modelu HSV sa používa algoritmus, ktorý obsahuje nasledujúcich 5 krokov:

1. Nájsť minimálnu a maximálnu hodnotu v modely RGB:
   $$Min = min(R, G, B)$$
   $$Max = max(R, G, B)$$

2. Normalizovať RGB hodnoty tak, aby spadali do intervalu $\langle 0{,}1 \rangle$:
   $$r = \frac{Max - R}{Max - Min} \qquad g = \frac{Max - G}{Max - Min} \qquad b = \frac{Max - B}{Max - Min}$$

3. Vypočítať hodnotu parametra $V$:
   $$V = max(R, G, B)$$

4. Vypočítať hodnotu parametra $S$. Pri tomto výpočte záleží na hodnote $Max$.

   Pokiaľ je hodnota $Max$ rovná 0, potom $S = 0$ a súčastne $H = 180°$.
   Pokiaľ sa hodnota $Max$ nerovná 0, potom $S = \frac{Max - Min}{Max}$.

[2]Obrázky 3.3 boli prevzaté z https://www.wikiwand.com/en/HSL_and_HSV a https://scc.ustc.edu.cn/zlsc/sugon/intel/ipp/ipp_manual/IPPI/ippi_ch6/ch6_color_models.htm

5. Vypočítať hodnotu parametra $H$. Pri tomto výpočte záleží na hodnotách $R$, $G$ a $B$. Vzhľadom k tomu, že hodnota $H$ je uhol, musí spadať do intervalu $\langle 0{,}360 \rangle$. Preto je po získaní tejto hodnoty potrebné upraviť tie hodnoty $H$, ktoré do daného intervalu nespadajú.

Pokiaľ sa hodnota $Max$ rovná hodnote $R$, potom $H = 60(b - g)$.
Pokiaľ sa hodnota $Max$ rovná hodnote $G$, potom $H = 60(2 + r - b)$.
Pokiaľ sa hodnota $Max$ rovná hodnote $B$, potom $H = 60(4 + g - r)$.

Pokiaľ je hodnota $H$ väšia alebo rovná 360, potom $H = H - 360$.
Pokiaľ je hodnota $H$ menšia ako 0, potom $H = H + 360$.

# Kapitola 4

# Operačný systém Android

Android je v súčasnosti jedna z najrozšírenejších bezplatných open source mobilných platforiem. Operačný systém Android je založený na linuxovom jadre a je najviac používaný v inteligentných mobilných telefónoch, ale tiež v inteligentných hodinkách, tabletoch či televízoroch. Architektúru tejto platformy ďalej popisuje podkapitola 4.2. Android OS je vyvíjaný spoločnosťou Google a Open Handset Alliance. Open Handset Alliance je skupina 84-roch mobilných a technologických spoločností, ktoré sa spojili, aby priniesli užívateľom nové, lepšie možnosti v oblasti mobilných technológií. [1] Vďaka tomuto neustálemu vývoju a vylepšeniam vznikajú stále novšie verzie Androidov, tie sú bližšie popísané v podkapitole 4.1.

Android OS umožňuje vývojárom pomerne jednoducho vytvárať rôzne aplikácie a hry, ktoré je možné zverejniť pomocou oficiálnej distribučnej služby pre zariadenia s týmto systémom – Google Play. Služba Google Play spočiatku ponúkala len hry a aplikácie, ale v súčasnosti ponúka tiež hudbu, filmy, elektornické knihy, magazíny a ďalšie. Oficiálnym vývojovým prostredím pre tieto aplikácie je Android Studio, ktoré bližšie popisuje podkapitola 4.3.

## 4.1 Verzie Android OS

Operačný systém Android prišiel so svojou prvou verziou Android 1.0 23. septembra 2008. [2] Prvou verziou s *Linux Kernel* (verzia 2.6.27) je Android 1.5, ktorý dostal kódové označenie Cupcake. [3] *Linux Kernel* spolu s ďalšími komponentami platformy Android sú bližšie popísané v nasledujúcej podkapitole 4.2. S príchodom linuxového jadra tiež prišli aj kódové označenia jednotlivých verzií. Tieto tematické označenia nesú názvy nejakých sladkých pochutín a v abecednom poradí ako napríklad Lollipop – Andorid 5.0, Marshmallow – Andorid 6.0, Oreo – Android 8, Pie – Android 9.0. Práve Android 9.0 bol posledný Android s týmto kódovým označením. Momentálne je toto označenie Android $X$, kde $X$ je číslica odpovedajúca verzii systému. K dátumu 30. marec 2020 bol najnovšou verziou systému Andorid 11, ten je momentálne zverejnený len na testovanie a dodatočný vývoj. Aktuálnou verziou, ktorá bola k vyššie uvedenému dátumu už nasadená v niektorých zariadeniach, je Android 10. V súčasnosti nie je možné vyhľadať oficiálne informácie o zastúpenosti jednotlivých verzií v zariadeniach. Podľa posledných oficiálnych informácií mali najväčšie percentuálne zastúpenia vyššie uvedené verzie Androidu, z nich najväčšie mala verzia Android 8 a to približne 27,4%. [5] Tieto dáta boli spracované ešte pred tým, než prišiel Android 10 a preto už nie sú aktuálne. Podľa najaktuálnejších neoficiálnych informácií je v súčasnosti vo vyše

40% zariadení využívaná verzia Android 9 – Pie. Ďalej je využívaný hlavne Android 8.1 – 12.2% a Android 6.0 – 10.1%. [7]

## 4.2 Architektúra Android OS

Architektúra operačného systému Android sa skladá zo šiestich komponentov, tieto komponenty sú zobrazené na obrázku 4.1. V tejto podkapitole sú ďalej popísané jednotlivé komponenty v smere zdola nahor.

*Linux Kernel* alebo inak linuxové jadro sa stará o správu pamäti, spravovanie procesov a tiež umožňuje systému Android využívať kľúčové funkcie zabezpečenia. Výrobcom týchto zariadení umožňuje vyvíjať hardwarové ovládače pre známe a už osvedčené jadro.

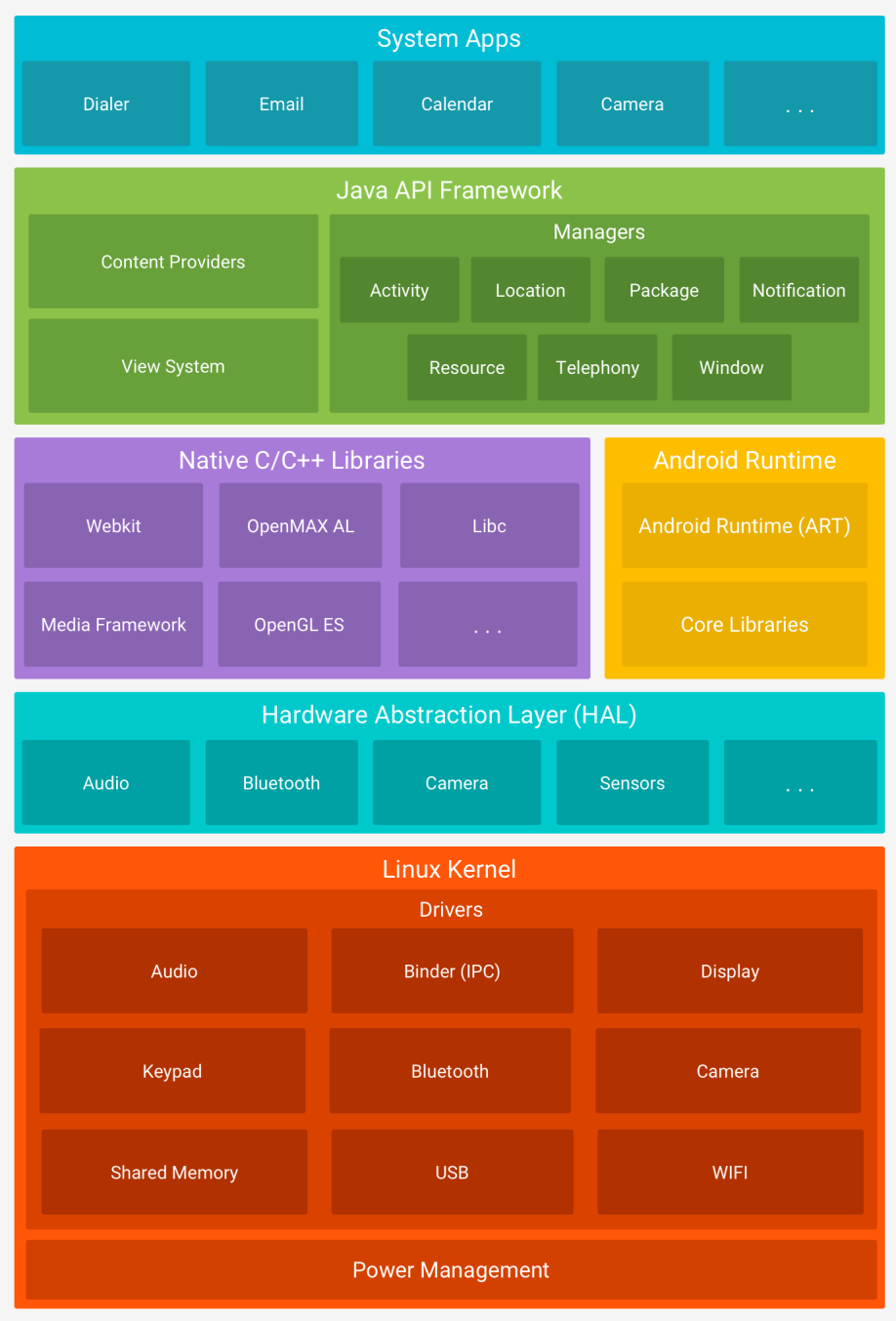
Ďalším z komponentov je *Hardware Abstraction Layer* (HAL) – abstrakčná vrstva pre hardware. Táto vrstva poskytuje štandardné rozhranie umožňujúce využívať hardware telefónu bez nutnosti poznať detaily hardwaru či jeho obsluhy, napríklad kameru, modul Bluetooth alebo iné komponenty. Skladá sa z niekoľkých modulov knižníc. Každá z knižníc implementuje interface pre daný typ hardwarového komponentu.

*Native C/C++ Libraries* sú natívne knižnice napísané v jazyku C a C++. Tieto knižnice vyžadujú natívny kód, z ktorého je zostavených mnoho základných komponentov a služieb systému Android, ako napríklad ART (Android Runtime), ktorý bude popísaný v nasledujúcom odstavci alebo HAL (Hardware Abstraction Layer) popísaná v predchádzajúcom odstavci. Pre poskytnutie funkcií niektorej z týchto natívnych knižníc aplikáciam, poskytuje platforma Android framework. Tento framework sa nazýva Java framework APIs a je taktiež jedným z komponentov a bude mu venovaný jeden z nasledujúcich odstavcov. Okrem iných do tejto vrstvy spadajú aj knižnice OpenCV, ktorá je zameraná predovšetkým na počítačové videnie a spracovanie obrazu v reálnom čase a OpenGL ES, ktorá sa zameriava na podporu pre kreslenie a manipuláciu s 2D a 3D grafikou v aplikáciách. V bakalárskej práci sú využité obe tieto knižnice a ich bližšiemu popisu sa venujú podkapitoly 6.1.1 – OpenCV a 6.1.2 – OpenGL ES.

Na rovnakej vrstve ako natívne knižnice sa nachádza tiež *Android Runtime* (ART) alebo inak povedané „behové prostredie“. ART je napísaný tak, aby dokázal spustiť viacero virtuálnych strojov na zariadení s nízkou pamäťou a to pomocou spustení DEX súborov. Súbor DEX je formátu bytecode navrhnutého špeciálne pre systém Android a je optimalizovaný na minimálne nároky na pamäť. Pri zariadeniach so systémom Android verzie 5.0, čo je API level 21, alebo vyššej verzie, sú jednotlivé aplikácie spúšťané ako inštancie virtuálneho stroja ART – Dalvik Virtual Machine (DVM). Tento komponent tiež obsahuje základné runtime knižnice poskytujúce väčšinu funkcií programovacieho jazyka Java, vrátane tých, ktoré používa rámec Java API framework.

*Java API framework* je predposledným zo 6-tich komponentov. Všetky funkcie Android OS sú k dispozícii prostredníctvom API rozhrania, napísaného v jazyku Java. API rozhranie tvorí stavebné bloky, ktoré sú potrebné pri vývoji aplikácií pre systém Android a to zjednodušením opakovaného použitia rôznych Java tried – niektorých základných a modulárnych komponentov a tiež služieb systému. Medzi služby systému patrí správca zdrojov, upozornení a aktivít, ďalej systém zobrazenia, ktorý je možné použiť pri vytvorení užívateľského rozhrania a tiež poskytovatelia obsahu, ktorí umožňujú aplikáciám prístup k údajom iných aplikácií. Všetky tieto Java API sú bežne dostupné programátorom, ktorí sa venujú vývoju mobilných aplikácií. Sú to rovnaké Java API ako tie, ktoré používajú systémové aplikácie Android.

Posledným komponentom sú *System Apps*. Jedná sa o systémové a užívateľské aplikácie. Systémové aplikácie sú už vopred nainštalované v zariadení samotným Googlom alebo iným dodávateľom hardwaru, sú to napríklad Google Play, Gmail, Google kalendár a mnoho ďalších. Užívateľské aplikácie sú tie, ktoré si užívateľ do svojho zariadenia sťahuje sám, tie sú bežne sťahované práve už zo spomínanej služby Google Play. [9]




Obr. 4.1: Na obrázku sú znázornené jednotlivé komponenty architektúry operačného systému Android. [9]

## 4.3 Android Studio

Android Studio je oficiálne integrované vývojové prostredie (IDE) pre vývoj aplikácií pre systém Android. Je založené na integrovanom vývojovom prostredí pre software, ktorý je napísaný v jazyku Java – JetBrains' IntelliJ IDEA. Andorid Studio okrem vývojárskych nástrojov a editorov spoločnosti IntelliJ ponúka ešte viac funkcií.

Toto vývojové prostredie je prehľadné, ľahko ovládateľné a umožňuje programátorovi prispôsobiť si ho tak, aby práca v ňom programátorovi vyhovovala a bola preňho čo najpohodlnejšia. Už pri inštalácii, ktorá je rýchla a intuitívna, je možné si zvoliť mód, v ktorom bude aplikácia spustená – light (svetlý) alebo dark (tmavý). Ďalej je možné meniť veľkosti a farby fontov, ale tiež usporiadanie okien ako napríklad konzola, debbuger či editor kódu. Andorid Studio je navrhnuté tak, aby nie len svojím výzorom alebo aj funkcionalitou čo najviac pomáhalo vývojárovi. Editor kódu napríklad ponúka pomoc pri písaní kódu našepkávaním, analýzou kódu či adresovaním.

Pri vytváraní projektu je možné zvoliť najnižšiu podporovanú verziu Androidu. Hneď pri tejto voľbe je vidieť približné percento zariadení, na ktorých bude aplikácia podporovaná a spustiteľná. Každý projekt obsahuje jeden alebo viacero modulov so zdrojovými kódmi a súbormi. Tieto moduly delíme na tri typy – moduly aplikácií pre Android, moduly knižníc a moduly Google App Engine. Pre rýchly prístup k hlavným zdrojovým kódom a súborom Android Studio implicitne zobrazuje súbory projektu usporiadané podľa týchto modulov.

Na zostavenie projektu využíva Android Studio systém Gradle. Každý projekt obsahuje na úrovni modulu samostatné súbory pre zostavenie daného modulu a na najvyššej úrovni obsahuje jeden súbor na zostavenie celého projektu. Tieto súbory sa nazývajú `build.gradle`. Sú to obyčajné textové súbory používajúce Groovyho syntax na konfiguráciu zostavenia. Zostavovací systém Gradle ponúka viacero funkcií ako napríklad opätovné použitie kódu naprieč zdrojmi, vytváranie rôznych verzií tej istej aplikácie alebo tiež podpora viacerých APK súborov. APK alebo inak Android Package je formát balíka používaný operačným systémom Android na distribúciu a inštaláciu mobilných aplikácií. [6]

# Kapitola 5

# Návrh aplikácie

Pred samotnou implementáciou aplikácie je dôležité sa zamyslieť nad jej návrhom. Návrh aplikácie predstavuje rozdelenie problému, ktorý aplikácia rieši, na menšie podproblémy. Vďaka tomu je potom následná implementácia omnoho jednoduchšia. V prvom rade je vhodné sa zamyslieť nad detekciou Rubikovej kocky v obraze a zistenia jej aktuálnej konfigurácie. Tento krok návrhu je popísaný v podkapitola 5.1. V nasledujúcej podkapitole 5.2 je popísaný návrh postupu riešenia Rubikovej kocky. V neposlednom rade je potrebné popísať aj návrh grafického uživateľského rozhrania, tomu je venovaná podkapitola 5.3.

Je potrebné tiež uviesť, že aplikácia je navrhovaná pre klasickú Rubikovu kocku 3x3x3, ktorá má na čiernom podklade farebné štvorce. Tieto farebné štvorce nesú jednu z nasledujúcich farieb – biela, žltá, oražová, červená, modrá a zelená. V súčasnosti sa totiž vyrábajú aj kocky, kde nie je čierny podklad alebo kocky obsahujúce iné farby, napríklad miesto bielej čiernu.

## 5.1 Detekcia aktuálnej konfigurácie Rubikovej kocky

Datekciu aktuálnej konfigurácie Rubikovej kocky som rozdelila do viacerých krokov. Od užívateľa očakávam 6 snímok, na každej práve jednu stenu kocky. Prvým krokom bude nájdenie kontúry samotnej steny kocky, teda štvoruholníka. Následne vďaka nájdenej kontúre získam súradnice vrcholov tohto štvoruholníka. Na každej stene sa nachádza 9 menších štvorcov, ktoré môžu niesť rôzne farby. V ďalšom kroku pomocou získaných súradnic vrcholov vypočítam súradnice pixelov nachádzajúcich sa približne v strede menších štvorcov. Okolo určeného pixelu vytvorím menšiu oblasť pixelov, z ktorých následne zistím ich farbu. Z takto získaných farieb urobím priemernú hodnotu, ktorá zodpovedá farbe daného menšieho štvorca. V poslednom kroku detekcie aktuálnej konfigurácie spracovávanej steny Rubikovej kocky uchovám získané hodnoty. Pre uchovávanie získaných hodnôt som zvolila 2D pole o veľkosti 3x3, kde každá jeho položka obsahuje danú farbu menšej kocky na spracovávanej stene kocky. Následne prejdem k spracovaniu ďalšej snímky, kde prevediem rovnaké kroky. Po spracovaní všetkých snímok vytvorím pole o veľkosti 6x3x3, ktoré bude uchovávať všetkých 6 získaných stien obsahujúcich pole 3x3 uchovávajúce farby danej steny. Z dát uložených v poli 6x3x3 vytvorím 27 inštancií tried reprezentujúcich malé kocky, teda stredy, hrany a vrcholy. Tie budú spoločne tvoriť kocku v priestore, kde každá malá kocka bude mať súradnice v rozmedzí od [0,0,0] - [2,2,2]. Na základe týchto malých kociek vytvorím a zobrazím kocku ako grafický 3D model zodpovedajúci Rubikovej kocke užívateľa. Užívateľ

bude môcť tento model dôkladne prezrieť zo všetkých strán a skontrolovať, či je nájdená konfigurácia správna a následne ju schváliť prípadne zamietnuť.

## 5.2 Postup riešenia Rubikovej kocky

Vzhľadom k tomu, že moja aplikácia má slúžiť ako návod, vďaka ktorému sa užívateľ môže naučiť kocku poskladať, rozhodla som sa preto implementovať už existujúce metódy a nevyužívať prvky umelej inteligencie, ktoré umožňujú vynechať niektoré kroky potrebné k zloženiu kocky. Ako som už spomenula v úvode, nechcela som cieľovu skupinu užívateľov obmedziť len na skupinu ľudí, ktorí Rubikovu kocku nedokážu zostaviť. Mojim cieľom je implementovať metódu pre začiatočníkov, ale časom chcem implementovať tiež metódu pre pokročilých. Túto skutočnosť som pri návrhu zohľadnila. Navrhla som preto architektúru aplikácie tak, aby bolo rozšírenie o pokročilú metódu čo najjednoduchšie. Obe metódy sú detailnejšie vysvetlené v sekcii 2.3.1 – metóda pre začiatočníkov a 2.3.2 – metóda pre pokročilých.

Na základe získanej konfigurácie bude nájdená sekvencia algoritmov danej metódy, ktorá bude následne krok po kroku prezentovaná užívateľovi. Keďže nie je jednoduché zo začiatku rozumieť označeniam stien a smeru ich otočenia, rozhodla som sa, že užívateľovi nebude zobrazená len sekvecia označení stien, ale tiež model jeho kocky. Tento model užívateľovi uľahčí naučenie sa daného algoritmu a tiež jeho prevedenia, pretože bude vidieť názornú ukážku, ako a ktorou stenou kocky otočiť. Vďaka tomu bude môcť priebežne kotrolovať, či postupuje správne.

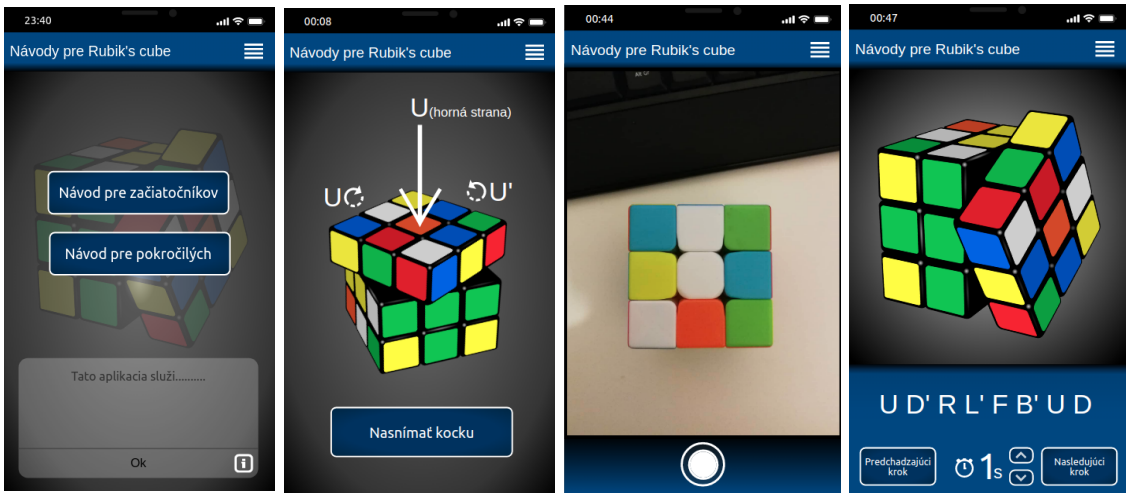
## 5.3 Grafické užívateľské rozhranie

Každá aplikácia obsahujúca grafické užívateľské rozhranie (GUI), by mala spĺňať požiadavky na intuitívnosť a jednoduchosť ovládania. Pri návrhu mojej aplikácie som sa snažila tieto požiadavky čo najviac dodržať. Vzhľadom k tomu, že skladanie Rubikovej kocky nie je zábavou len určitej vekovej kategórie, je potrebné to pri návrhu zohľadniť a navrhnúť aplikáciu tak, aby žiadneho užívateľa ničím neodradila. Preto som sa snažila aplikáciu navrhnúť tak, aby nebol potrebný žiadny manuál a aby sa v nej užívateľ každej vekovej kategórie vedel intuitívne orientovať.

Ďalšou dôležitou časťou návrhu grafického rozhrania je výber farieb. Nakoľko som predpokladala, že užívateľ bude pozerať na display dlhšiu dobu, rozhodla som sa zvoliť tmavšie farby, ktoré menej namáhajú zrak. Pozadie všetkých častí aplikácie nesie tmavosivú farbu, na ktorej Rubikova kocka dobre vynikne. Keďže samotná Rubikova kocka obsahuje šesť rôznych farieb bolo potrebné pre ostatné komponenty zvoliť vhodnú farbu tak, aby celý layout pôsobil jednotvárne a prirodzene. Po dlhšej úvahe a skúšaniu kombinácií farieb som zvolila tmavomodrú farbu. Tá na sivom pozadí zvýraznila potrebné komponenety ako napríklad tlačítka, dodala aplikácii nádych hry, ale nepôsobí príliš rušivo a ponecháva dominantnosť farbám Rubikovej kocky. Vzhľadom k spätnej väzbe od užívateľov, ktorí aplikáciu testovali, som sa rozhodla takto ponechať farby aj vo finálnej verzii aplikácie, pretože väčšine užívateľov táto kombinácia farieb vyhovovala.

Samotná aplikácia je rozdelená do viacerých častí. Medzi tieto časti patrí vysvetlenie dôležitých pojmov pred začiatkom návodu na zloženie Rubikovej kocky. Ďalej nasnímanie danej Rubikovej kocky a určenie jej aktuálnej konfigurácie. A poslednou časťou je grafický návod, ktorý krok za krokom vedie k zloženiu danej kocky. Aplikácia obsahuje tiež menu,

vďaka ktorému sa uživateľ môže ľahko a rýchlo dostať k opätovnému spusteniu návodu. Jednotlivé časti sú znázornené na obrázku 5.1




(a) Úvodná stránka (b) Potrebná teória (c) Nasnímanie kocky (d) Návod na zloženie

Obr. 5.1: Návrh grafického užívateľského rozhrania mobilnej aplikácie.

### Úvodná strana

Úvodná strana bola navrhnutá tak, aby bola čo najprehľadnejšia a bolo jednoduché sa v nej zorientovať. Obsahuje len dve tlačítka a informačnú ikonu. Tlačítka slúžia pre voľbu návodu. Ako už bolo spomenuté, užívateľ si môže vybrať medzi dvoma typmi návodov – návod pre začiatočníkov a návod pre pokročilých. V pravom dolnom rohu je zakomponovaná informačná ikona. Po kliknutí na ikonu sa zobrazí stručná informácia o tom, načo táto aplikácia slúži. Túto ikonu som sa rozhodla zakomponovať do úvodnej strany z dôvodu, že by sa mohol užívateľ dostať k aplikácií bez toho, aby vedel načo daná aplikácia slúži.

### Potrebná teória

Po voľbe druhu návodu z úvodnej stránky bude užívateľovi vysvetlená potrebná teória. Túto časť som navrhla tak, aby sa skladala z viacerých stránok a tým bola zrozumiteľnejšia. Na prvej stránke je vysvetlená metóda, ktorú daný návod používa. Následne na ďalších stránkach je graficky znázornené označenie strán, potrebné k danej metóde, to je zobrazené na obrázku 5.1b. Ako už z obrázku vyplýva, po prejdení základných označení strán Rubikovej kocky sa užívateľovi zobrazí tlačítko, ktoré vedie k nasnímaniu jeho Rubikovej kocky. Tento návrh sa od finálnej verzie líši. Vzhľadom na spätnú väzbu od testerov som sa rozhodla, že bude lepšie vytvoriť iba jednu stránku, ktorá bude obsahovať všetky tieto informácie. Dôvod tejto zmeny bude bližšie vysvetlený v podkapitole 7.2.

### Nasnímanie Rubikovej kocky

Ako je vidieť na obrázku 5.1c, fotoaparát bol graficky navrhnutý tak, aby zodpovedal bežným aplikáciám využívajúcim fotoaparát. Aby bolo nasnímanie Rubikovej kocky pre užívateľa pohodlnejšie a rýchlejšie, rozhodla som sa, že pred samotným nasnímaním steny kocky predkreslím požadovanú pozíciu kocky. Po nasnímaní všetkých stien Rubikovej kocky sa

zobrazí užívateľovi jeho nasnímaná kocka, ktorú bude musieť odsúhlasiť. V prípade, ak by kocka nemala správnu konfiguráciu, bude užívateľ požiadaný o opätovné nafotenie. Užívateľ sa ocitne znovu na stránke s fotoaparátom, nebude vrátený na úvodnú stránku, kde by si musel znovu zvoliť druh návodu.

### Návod na zloženie Rubikovej kocky

Keď je Rubikova kocka nasnímaná a jej konfigurácia je užívateľom odsúhlasená, dostávame sa k poslednej časti. Touto časťou je samotná prezentácia návodu na jej zloženie. Vzhľadom k tomu, že zo začiatku nie je jednoduché si zapamätať dané označenia stien Rubikovej kocky, rozhodla som sa návod prezentovať pomocou animácií. V hornej časti bude teda zobrazená kocka, ktorá bude daný krok predvádzať. V spodnej časti bude zobrazený krok pomocou označenia strán, talčítka pre predchádzajúci a nasledujúci krok a tiež možnosť zvoliť si počet sekúnd, po ktorých sa na kocke otočí ďalšia stena. Tento návrh je zobrazený na obrázku 5.1d. Vo finálnej verzii som sa rozhodla tlačítko venované predchadzajúcemu kroku odstrániť. Rozhodla som sa na základe spätnej vezby od užívateľov, ktorí aplikáciu testovali. Vysvetlenie, prečo som sa pre túto zmenu rozhodla, bude popísané v podkapitole 7.2.

# Kapitola 6

# Implementácia

Aplikácia je implementovaná výhradne pre operačný systém Android, ktorý bol detailnejšie popísaný v kapitole 4. Na vývoj aplikácií pre Android OS existuje oficiálne integrované vývojové prostredie – Android Studio. Android Studio bolo popísané v podkapitole 4.3. Moja aplikácia je naprogramovaná v tomto vývojovom prostredí a teda napísaná primárne v programovacom jazyku Java.

Na začiatku tejto kapitoly sú bližšie popísané knižnice využité pri implementácii. Jedná sa o knižnice OpenCV – podkapitola 6.1.1 a OpenGL ES – podkapitola 6.1.2. V nasledujúcich častiach kapitoly sú popísané jednotlivé časti implementácie rozdelené podľa návrhu. Hlavná funkcionalita aplikácie sa delí na dve implementačné časti a to detekcia aktuálnej konfigurácie Rubikovej kocky užívateľa – podkapitola 6.2 a postup jej riešenia – podkapitola 6.3. V závere tejto kapitoly, konkrétne v podkapitole 6.4, je popísaná implementácia grafického užívateľského rozhrania.

## 6.1 Využité knižnice

### 6.1.1 OpenCV

OpenCV je open source knižnica zameraná na počítačové videnie, ktorá je vydávaná pod licenciou BSD. Je napísaná v jazyku C++, ale je možné ju využívať aj pri programovaní v jazyku Java, C, Python a ďalších. Knižnica sa skladá z viacerých modulov. Pri implementácii boli využité dva z týchto modulov – modul `core` a modul `imgProc`, preto budú v krátkosti popísané len tieto moduly.

Modul `core` je hlavným modulom, ktorý predstavuje základnú funkčnosť, definuje základné dátové štruktúry, operácie s nimi a funkcie, ktoré využívajú všetky ostatné moduly. V module `core` sa definuje aj viacrozmerné pole `Mat`, ktoré bude pri implementácii využité. Modul `imgProc` slúži na samotné spracovanie obrazu. V tomto module sa nachádzajú rôzne druhy filtrovania obrazu, transformácie, konverzia farebného priestoru a ďalšie. [4]

### 6.1.2 OpenGL ES

Open Graphics Library for Embedded Systems alebo skrátene OpenGL ES je knižnica, ktorá je zamenraná na 2D a 3D grafiku v mobilých zariadeniach, teda v mobilných telefónoch, tabletoch a ďalších. Vznikla z OpenGL, multiplatformového programovateľného rozhrania na klasických počítačoch.

Android podporuje OpenGL pomocou rozhrania API. V rámci Android OS definujeme dve základné triedy. Tieto triedy umožňujú vytváranie a manipuláciu s grafikou. Prvou triedou je `GLSurfaceView`, ktorá zaisťuje priestor, kam sa vykreslí vytvorená grafika. Táto trieda tiež poskytuje preddefinované udalosti, ako napríklad `onTouchEvent`, vďaka ktorej je možné implementovať rotáciu objektov. V tejto práci sa jedná o rotáciu grafického 3D objektu Rubikovej kocky. Druhou základnou triedov je `GLSurfaceView.Renderer`, ktorý vykresľuje implementovanú grafiku priamo do `GLSurfaceView`. Pri tejto triede je nutné implementovať niektoré metódy, ktoré určujú, aké vlastnosti bude mať daná grafika. Jedná sa napríklad o metódu `onDrawFrame`, ktorá je volaná vždy, keď je kvôli nejakej zmene potrebné prekresliť `GLSurficeView`. [8]

## 6.2 Detekcia aktuálnej konfigurácie Rubikovej kocky

Implementáciu aktuálnej konfigurácie Rubikovej kocky som sa rozhodla rozdeliť do dvoch hlavných častí. Prvou časťou je samotné spracovanie nasnímanej steny Rubikovej kocky, kde sú využité prvky počítačového videnia. Toto spracovanie snímky danej steny sa opakuje pre každú stenu. Druhou časťou je prevedenie získaných informácií do 3D objektu, teda objektu kocky v priestore. Tento proces sa skladá z nájdenia, uchovania a následného zobrazenia aktuálnej konfigurácie Rubikovej kocky.

### 6.2.1 Spracovanie nasnímanej Rubikovej kocky

Proces detekcie Rubikovej kocky a následne rozoznanie farieb je naimplementovaný v triede `DetectRubiksCube`. V tomto procese sú využité funkcie z knižnice OpenCV. Keďže každý mobilný telefón môže mať iný fotoaparát a teda veľkosť snímok môže byť rôzna, rozhodla som sa pred samotným procesom upraviť veľkosť snímky tak, aby bol proces spracovania rýchlejší. Veľkosť snímky som upravila tak, že jeho šírka bude 600 pixelov a výška sa následne dopočíta tak, aby bol zachovaný pomer z originálnej snímky.

Prvou časťou procesu spracovania je detekcia hrán, tú som rozdelila do troch krokov. Prvým krokom procesu spracovania je redukcia šumu v obraze. Zvolila som Gaussovo rozostrenie obrazu, ktoré som implementovala pomocou funkcie `GaussianBlur`. Veľkosť konvolučného jadra som zvolila 5x5 a smerodajnú odchýlku rovnú 0. Tieto hodnoty dosahovali najlepší výsledok pri väčšine vstupov. Pred použitím Gaussovho rozostrenia som obrázok ešte previedla do odtieňov sivej farby pomocou funkcie `cvtColor`, kde je možné vybrať si z mnohých prevodov medzi farebnými modelmi. Ďalej je prevedené prahovanie. Tu som zvolila adaptívne prahovanie, ktoré dosahuje najlepšie výsledky. Tento krok je implementovaný pomocou funkcie `adaptiveThreshold`. Pri tejto funkcii bolo tiež potrebné zvoliť vhodné parametre. Najlepšie výsledky dosahovala funkcia s veľkosťou susedstva 103x103 a hodnotou prahu vypočítanou ako vážený súčet pixelov z daného susedstva. Na výpočet tejto hodnoty je nutné využiť Gaussovo okno, to sa predáva do funkcie ako parameter `ADAPTIVE_THRESH_GAUSSIAN_C`. Na samotnú detekciu hrán som využila funkciu `Canny`, teda Cannyho detektor hrán, ktorej vstupom je výstup adaptívneho prahovania. Pomer hodnôt minimálneho a maximálneho prahu hysterézie je 1:3, čo bolo doporučené v dokumentácii OpenCV.

Ďalšou časťou je nájdenie kontúr Rubikovej kocky. Knižnica OpenCV ponúka funkciu `findContours`, ktorá vďaka parametru `RETR_EXTERNAL` vráti iba vonkajšie kontúry. Keďže predpokladám, že na snímke bude Rubikova kocka dominovať, tak z nájdených kontúr vyberiem kontúru, ktorá má najväčší obsah. Následne na tejto kontúre prevediem apro-

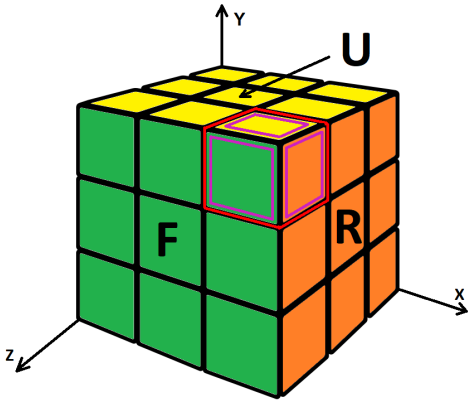
ximáciu pomocou funkcie `approxPolyDP`. Aproximácia je nutná z dôvodu, že je potrebné danú kontúru vyjadriť pomocou štyroch bodov – vrcholov kocky. Takto získané body najprv postupne zoradím, aby som predišla prípadným problémom, ak by body boli vyhľadané v inej postupnosti ako očakávam.

Poslednou časťou je získanie farieb jednotlivých malých kociek. Po zoradení bodov získaných v predchádzajúcej časti bude volaná funcia `detectColor`, kde okrem získaných bodov bude vstupným parametrom aj pôvodný zmenšený obraz. Ako prvé v tejto funkcii sa z bodov vypočítajú približná šírka a výška kocky, ktoré sú potrebné na nasledujúce výpočty. Postupne sa vypočíta za pomoci vyššie získaných informácii pixel, ktorý bude ležať približne v strede malej kocky. Kedže by sa mohlo stať, že daný pixel bude ovplyvnený šumom, rozhodla som sa, že farbu vypočítam ako priemernú hodnotu susedstva 10x10 od nájdeného pixelu. Pre toto susedstvo vytvorím masku, v ktorej sa za pomoci funkcie `mean` vypočíta priemerná hodnota farby oblasti v maske. Priemerná hodnota farby je uložená do premennej typu `Scalar`. Typ `Scalar` slúži na uchovávanie troch alebo štyroch hodnôt a jedná sa práve o uchovanie farebného priestoru. Takto získaná farba je vo formáte RGB modelu farieb a určiť, o akú farbu konkrétne sa jedná, je pomerne zložité, pretože treba zohľadniť všetky tri zložky. Preto som sa rohodla využiť farebný model HSV, ktorého parameter H odpovedá odtieňu farby, vďaka čomu je potrebné pri určení farby zohľadniť len jednu hodnotu. V prípade bielej farby, ktorá sa medzi odtieňmi nevyskytuje, je potrebný len parameter S. Parameter S predstavuje sýtosť farby, kde viem povedať, že ak je sýtosť nižšia ako 50%, jedná sa práve o bielu farbu. Kedže nie je možné urobiť prevod nad typom `Scalar`, previedla som premennú tohto typu na obrázok o veľkosti jedného pixelu, ktorý nesie danú farbu. Následne som za pomoci funkcie `cvtColor` a parametru `COLOR_RGB2HSV` previedla tento obrázok do farebného modelu HSV. Po prevedení obrázku je potrebné ešte určiť danú farbu vyššie uvedeným spôsobom. Táto funkcionalita je implementovaná vo funkcii `GetColor`, ktorá je v triede `Colors`. Táto funkcia na vstupe dostane pole obsahujúce hodnoty H, S a V, z ktorých je následne určená farba. Proces získania danej farby sa zopakuje pre všetkých 9 menších kociek, ktoré budú uložené do poľa o veľkosti 3x3. Funkcia `detectColor` toto pole vráti.

Celý proces spracovania obrazu, detekcie steny Rubikovej kocky a získania farieb sa zopakuje pre každú nasnímanú stenu, teda šesťkrát. Tak, ako bolo spomenuté v návrhu, všetky steny budú uložené do poľa o veľkosi 6x3x3. Toto pole po ukončení spracovania poslednej steny bude obsahovať šesť stien, kde každá z nich obsahuje 3x3 hodnôt rôznych farieb.

### 6.2.2 Nájdenie aktuálnej konfigurácie Rubikovej kocky

Ako bolo spomenuté v návrhu, posledným krokom nájdenia aktuálnej konfigurácie je prevod poľa 6x3x3 do poľa o veľkosti 3x3x3, ktoré bude znázornovať Rubikovu kocku v priestore. Tento prevod je implementovaný v triede `CubeConfugiration`, kde bude postupne prechádzaná každá stena a jednotlivé farebné štvorce budú prevedené na inštancie triedy malej kocky. Táto trieda má názov `Cubie` a uchováva všetky potrebné informácie o danej malej kocke. Medzi tieto informácie patria súradnice malej kocky v priestore – [X,Y,Z], typ malej kocky – hrana, vrchol alebo ani jedno, teda stred, informácia o tom či je na správnom mieste a pole inštancií triedy `CubieColor`, ktorá v sebe uchováva názov farby a jej orientáciu. Pod orientáciou sa rozumie označenie steny, na ktorej sa farba nachádza, teda F, B, U, D, L, R. Pre lepšie porozumienie spôsobu uchovania Rubikovej kocky, som vytvorila obrázok 6.1.




Ukážka triedy `Cubie` pre znázornenú malú kocku:

```
Cubie {
  X = 2;
  Y = 2;
  Z = 2;
  edge = false;
  corner = true;
  correct = true;
  CubieColor cubie_colors[3];
}
```

Ukážka triedy `CubieColor` pre znázornený zelený štvorec na malej kocke:

```
CubieColor {
  direction = 'F';
  color = Color.colors_mane.Green;
}
```

Obr. 6.1: Na obrázku je uvedený príklad triedy `Cubie` a `CubieColor` pre lepšie pochopenie uchovávanie Rubikovej kocky.

Tieto malé kocky potom budú vykreslené ako grafický 3D model užívateľovi, ktorý skontroluje, či bola Rubikova kocka správne rozpoznaná. V prípade, že by sa nájdená konfigurácia nezhodovala s jeho kockou, bude užívateľ požiadaný o opätovné nasnímanie a upozornený, že musí steny nasnímať ako na ukážke.

## 6.3 Postup riešenia Rubikovej kocky

Predtým, než bude implementovaná samotná začiatočnícka metóda, je potrebné naimplementovať otáčanie jednotlivých stien do oboch smerov, zistenie, či malá kocka je na správnom mieste, nájdenie danej malej kocky a ďalšie pomocné funkcie. Po implementácií týchto funkcií je možné začať implementovať samotný solver.

### 6.3.1 Pomocné funkcie

Pomocné funkcie sú implementované v triede `CubeConfiguration`, pretože sú úzko späté s konfiguráciou Rubikovej kocky. Pomocných funkcií je viacero, väčšina z nich sa skladá z volania najkomplexnejších a najvýznamnejších funkcií. Z tohto dôvodu v nasledujúcej časti budú popísané len tie najvýznamnejšie.

#### Funkcia `FaceMove`

Vo `FaceMove` sú naimplementované jednotlivé otáčania stien. Na vstupe dostáva `String` obsahujúci označenie steny a príslušného smeru otočenia, teda v smere alebo v protismere hodinových ručičiek. Na začiatku funkcie je vytvorená kópia aktuálnej konfigurácie kocky, pretože prepísaním malej kocky inou by mohlo dôjsť k strate niektorých informácií. Následne pre každú stenu a smer je vypočítané, ktoré súradnice budú zmenené a tiež ako. Po zmene súradníc bude zavolaná pomocná funkcia `ChangeCubieAfterMove`, ktorá zaistí, že sa po

presunutí malej kocky na novú pozíciu správne zmení aj orientácia všetkých farieb, ktoré obsahuje, na základe smeru otočenia steny.

#### Funkcia `IsCorrenct`

Na začiatku sa všetky malé kocky považujú za nesprávne umiestnené, preto bolo potrebné zaistiť konrolu správnej pozície pomocou samostatnej funkcie, ktorá je využívaná v mohých ďalších funkciách solveru. Funkcia na vstupe dostane malú kocku, ktorej pozíciu je potrebné skontrolovať. Následne prejde všetky farby, ktoré obsahuje a skontroluje, či sa farba zhoduje s farbou stredovej kocky na danej stene. V prípade, že sú farba aj smer malej kocky a stredu zhodné, bude označená za správne umiestnenú a natočenú kocku.

#### Funkcia `FindCubie`

Ďalšou často využívanou funkciou pri solveri je práve `FindCubie`, vďaka ktorej nájdeme požadovanú kocku. Funkcii je predaná informácia, či sa jedná o hranu alebo vrchol a pole obsahujúce farby hľadanej malej kocky. Následne budú postupne prejdené všetky malé kocky hľadaného typu, mimo tých, ktoré sú umiestnené a natočené správne. Na základe porovnania všetkých jej farieb je funkciou vrátená hľadaná kocka spolu so všetkými informáciami o nej. Funkciu je možné zavolať aj bez úplnej špecifikácie farieb, a to tak, že v poli farieb bude len jedna požadovaná farba. V takomto prípade bude vrátená prvá nájdená malá kocka hľadaného typu okrem korektných.

### 6.3.2 Solver

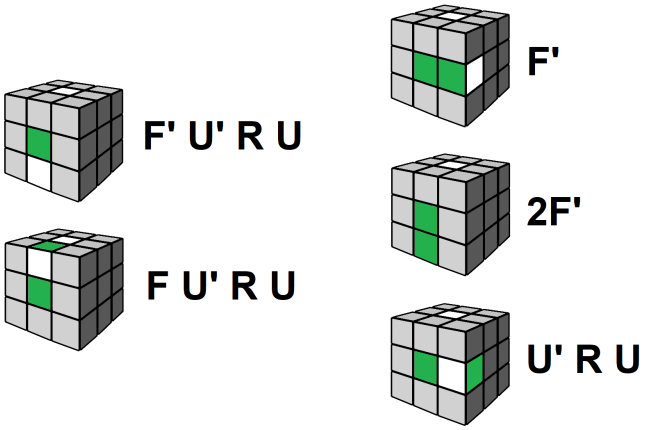
Ako bolo popísané v návrhu, solver bol navrhnutý tak, aby rozšírenie o pokročilú metódu bolo čo najjednoduchšie. Základný solver je naimplementovaný ako abstraktná trieda s názvom `BasicSolver`. Samotná začiatočnícka metóda je následne implementovaná v triede `BeginnerSolver`, ktorá dedí od vyššie uvedenej abstraktnej triedy. Detailnejší popis tejto metódy je popísaný v podkapitole 2.3.1.

#### `BasicSolver`

V tejto triede sú implementované funkcie, ktoré je možné využiť pri začiatočníckej metóde, ale tiež pri pokročilej metóde. Prvou spoločnou funkciou je abstraktná funkcia `Solve`, ktorú každá trieda vychádzajúca z triedy `BasicSolver` bude musieť implementovať samostatne.

Kedže obe metódy majú spoločný prvý krok a to biely kríž, pre tento krok bola implementovaná ďalšia spoločná funkcia a to `MakeCross`. Algoritmus spočíva v tom, že za pomoci fukcie `FindCubie` bude nájdená taká hrana obsahujúca bielu farbu, ktorá nie je korektná. Aby bol algoritmus univerzálny a neviazal sa vždy na jednu konkrétnu bielu hranu, bol naimplementovaný v nasledujúcich krokoch. Pokiaľ sa nájdená hrana nenachádza na prostrednej vrstve, prvý krok ju tam umiestni. Z tejto pozície je totiž najjednoduchšie následné vloženie hrany na korektné miesto, a to tak, že bude v požadovanom smere otočené stenou, na ktorej sa nachádza druhá farba hrany. Vloženie, teda druhý krok, bolo nutné naimplementovať tak, aby nebola porušená už postavená časť kríža. Preto je algoritmus postavený tak, aby bolo pri vkladaní vždy otočené len stenou, ktorá zodpovedá práve druhej farbe hrany. Keďže nás zvyšok kocky zatiaľ nezaujíma, tak je na základe zostaveného vzroca vypočítané, o koľko je potrebné otočiť bielu stenu tak, aby pri vkladaní bola hrana vložená priamo na korektnú pozíciu. Po vložení je biela stena vrátená do pôvodnej pozície a teda

hrana na svojom mieste. Tento proces sa zopakuje pre všetky biele hrany mimo konkrétnej pozície. Pre lepšie pochopenie, ako funguje spôsob vkladania bielych hrán prikladám obrázok 6.2.




Obr. 6.2: Na obrázku sú znázornené jednotlivé situácie pri vkladaní bieleho vrcholu.

Keďže bolo potrebné pri nájdení algoritmu vhodného pre danú situáciu zostaviť určité podmienky, boli prevedené otáčania stien, ktoré vo výsledku nebolo vôbec potrebné užívateľovi prezentovať. Z tohto dôvodu bola naimplementovaná posledná spoločná funkcia `OptimizeMoves`, ktorá optimalizuje prevedené kroky tak, aby kroky podstatné pre určenie algoritmu, ale zároveň neužitočné pre užívateľa, neboli prezentované v grafickom rozhraní. Napríklad pokiaľ pri hľadaní algoritmu bolo prevedené otáčanie U a U' hneď po sebe, boli odstránené z krokov, ktoré sú užívateľovi zobrazené, pretože sa po ich vykonaní stav Rubikovej kocky nezmení.

#### `BeginnerSolver`

Ako už bolo spomenuté trieda `BeginnerSolver` dedí od abstraktnej triedy `BasicSolver` a je teda nutné, aby definovala funkciu `Solve`. Začiatočnícku metódu, ktorú som sa rozhodla implementovať, som detailnejšie popísala v podkapitole 2.3.1. Funkcie `Solve`, teda postupne volá 7 hlavných funkcií, reprezentujúcich jednotlivé časti metódy *Layer by layer*:

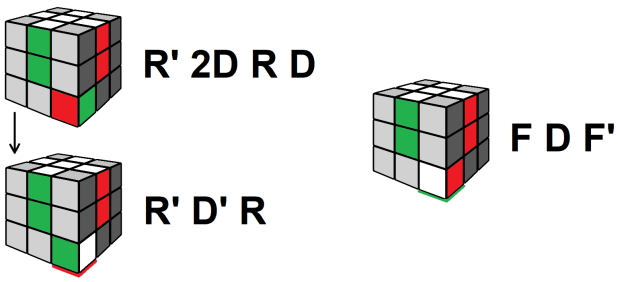
1. **Funkcia MakeCross**

   Jedná sa o jednu zo spoločných funkcií a preto je ako jediná implementovaná v abstraktnej triede `BasicSolver` a už bola detailne popísaná vyššie v sekcii venujúcej sa práve tejto triede.

2. **Funkcia FirstLayer**

   Tak, ako pri predchádzajúcej funkcii, je prvým krokom pomocou funkcie `FindCubie` nájdenie takého vrcholu obsahujúceho bielu farbu, ktorý nie je korektný. V tomto algoritme bolo tiež potrebné zaistiť univerzálnosť. Funkcia bola teda naimplementovaná tak, že nájdený vrchol umiestni nad vrchol, na ktorého pozíciu patrí a následne prevedie vloženie pomocou algoritmu. Tento algoritmus má viac podôb, ktoré znovu závisia od smeru bielej farby. To, akým spôsobom sú vrcholy vkladané, je zobrazené na obrázku 6.3. Pri tejto implementácii je ale biela stena spodná, pretože pokiaľ by

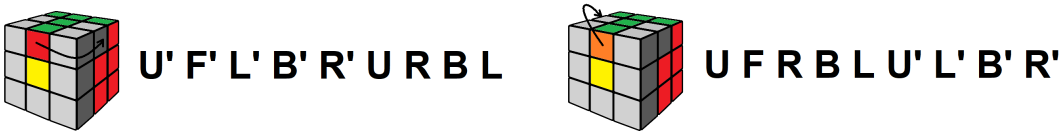
sa užívateľ chcel naučiť časom tiež pokročilú metódu, ktorá tento spôsob vyžaduje, bude to preňho omnoho jednoduchšie.




Obr. 6.3: Na obrázku sú znázornené jednotlivé situácie pri vkladaní bieleho vrcholu.

**3. Funkcia SecondLayer**

Funkcia slúži na zloženie prostrednej vrstvy Rubikovej kocky. Je naimplementovaná tak, že na poslendej vrstve, teda na žltej stene, nájde hranu, ktorá neobsahuje žltú farbu. Následne otočí touto stenou tak, aby sa farba nájdenej hrany, ktorá má iný smer ako žltá stena, zhodovala s farbou stredu steny, kam po otočení smeruje. Keďže algoritmus na vloženie hrany na korektné miesto vyžaduje, aby bola vrchnou stenou tá stena, ktorej stred i jeho smer sa zhodujú s farbou a smerom nájdenej hrany, bude celá kocka otočená pomocou funkcií implementujúcich označenia X, X', Y, Y', ktoré sú vysvetlené v podkapitole 2.3. Následne bude prevedený jeden z algoritmou uvedených na obrázku 6.4, ktorý závisí od druhej farby tejto hrany. Po prevedení algoritmu bude kocka vrátená do pôvodnej pozície. V prípade, že sa v poslednej vrstve nevyskytuje už žiadna hrana, ale nie sú všetky na korektnom mieste, bude tento algoritmus prevedený s hociktorou hranou obsahujúcou žltú farbu tak, aby sa nesprávne umiestnená hrana dostala do poslednej vrstvy.



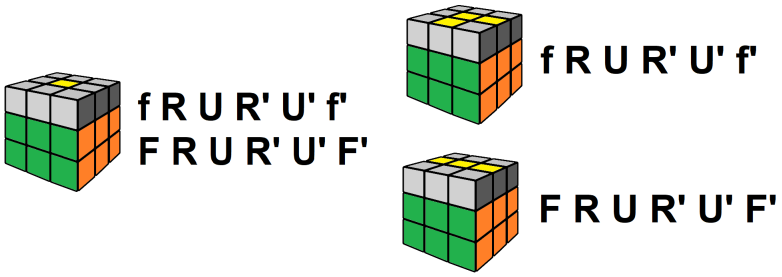

Obr. 6.4: Na obrázku sú znázornené algoritmy, ktoré zaistia presunutie hrany na korektné miesto. Pri implementácii je zelená stena vrchnou a biela stena zadnou.

**4. Funkcia MakeYellowCross**

Ako prvé v tejto funkcii je potrebné zistiť, ktoré hrany smerujú žltou farbou tým istým smerom ako stred poslednej vrstvy, teda vrchnej (žltej) steny. Pre rýchle zaistenie tejto informácie bola implementovaná pomocná funkcia `isYellowUp`, ktorá vráti hodnotu `true` v prípade, že žltá farba smeruje nahor alebo v opačnom prípade vráti hdonotu `false`. Pri tomto kroku môžu nastať štyri prípady.

Prvým prípadom je, že budú žltou farbou nahor smerovať dve hrany ležiace vedľa seba, teda spoločne vytvoria akoby písmeno L. V tomto prípade je nutné zaistiť, aby bola vrchná stena otočená tak, aby jedna z týchto hrán smerovala dopredu a druhá vpravo, následne bude prevedený daný algoritmus. Druhým prípadom je, že budú

tieto dve hrany smerujúce žltou nahor ležať oproti sebe. V tomto prípade je nutné vrchnou stenou otočiť tak, aby bola jedna hrana na ľavej stene a druhá na pravej stene. Následne sa zavolá daný algoritmus, ktorý rieši túto situáciu. V treťom prípade nebude žltou nahor smerovať ani jedna z hrán. Tu sú potrebné oba vyššie uvedené algoritmy. V poslednom prípade budú žltou nahor smerovať všetky hrany a teda je možné prejsť rovno k ďalšiemu kroku tejto metódy Oba algoritmy sú zobrazené na nasledujúcom obrázku 6.5.




Obr. 6.5: Na obrázku sú znázornené jednotlivé algoritmy pre vytvorenie žltého kríža.

**5. Funkcia CorrectYellowCross**

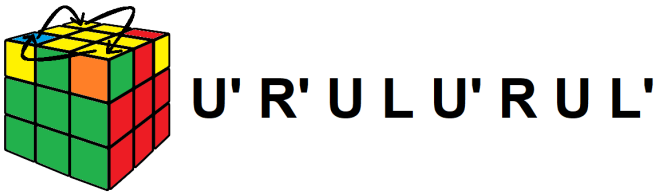
Hrany v kríži vytvorenom v predchádzajúcej funkcii síce smerujú správnou farbou nahor, ale nemusia byť na korektonom mieste. Funckia `CorrectYellowCross` zaistí, že budú presunuté všetky hrany na správnu pozíciu. Na začiatku za pomoci funkcie `IsCorrect` zistím, ktoré hrany sú korektné. Cieľom tohto kroku je nájsť dve korektné hrany, ktoré ležia vedľa seba. V prípade, že tieto hrany existujú, bude vrchnou stenou otočené tak, aby sa správne umiestnené hrany nachádzali jedna na pravej stene a druhá na zadnej stene. Následne bude vykonaný algoritmus z obrázku 6.6 vrchná stena bude otočená do pôvodnej pozície. V prípade, že bola nájdená len jedna korektná hrana, alebo žiadna, bude otočené vrchnou stenou v smere hodinových ručičiek a znovu sa algoritmus pokúsi nájsť dve korektné hrany. V prípade, že korektné budú všetky hrany, prejdeme k nasledujúcej funkcii.


Obr. 6.6: Na obrázku je znázornený algoritmus, ktorý zaistí prehodenie hrán v žltom kríži tak, aby boli všetky korektné.

**6. Funkcia CorrectYellowCornerPosition**

Táto funkcia funguje podobne ako predošlá funkcia `CorrectYellowCross`. Ako prvé je za pomoci funckie `IsCorrect` zistené, ktoré vrcholy sú na správnom mieste. Cieľom je nájsť jeden korektný vrchol tak, aby nebolo nutné otočenie vrchnou stenou. V prípade, že bude tento vrchol nájdený, otočí sa celou kockou tak, aby korektný vrchol bol

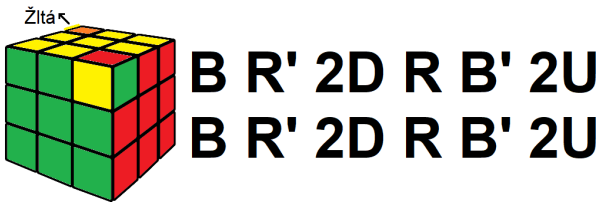
v pravom hornom rohu. Následne sa prevedie algoritmus z obrázku 6.7 V prípade, že nebude nájdený žiaden korektný vrchol, prevedie sa algoritmu nad aktuálnou pozíciou kocky a znovu bude vyhľadávaný korektný vrchol. V inom prípade sa jedná o kocku, kde niekto omylom alebo zámerne vybral a mechanicky otočil jeden vrchol a kocku nie je možné zostaviť. V prípade, že budú všetky vrcholy na korektnom mieste, bude zavolaná posledná funkcia `MakeYellowCorner`.




Obr. 6.7: Na obrázku je znázornený algoritmus, ktorý zaistí prehodenie troch žltých vrcholov, aby boli všetky na správnom mieste.

**7. Funkcia MakeYellowCorner**

Vrcholy sú po prevedení predchádzajúcej funkcie síce na korektnom mieste, ale nemusia byť správne natočené. Tento problém vyrieši funkcia `MakeYellowCorner`. Ako prvé bude vďaka pomocnej funkcii `isYellowUp` zistené, či daný vrchol je natočený správne alebo nie. Budú nájdené len tie vrcholy, ktoré nie sú natočené správne. Algoritmus z obrázku 6.8 je prevedený vždy nad vrcholom v ľavom hornom rohu a súčastne vrcholom v pravom dolnom rohu vzhľadom k pozícii. Funkcia preto najprv skontorluje korektnosť vcholu v ľavom hornom rohu. Pokiaľ bude vrchol korektný, otočí vrchnou stenou v smere hodinových ručičiek a znovu skontroluje korektnosť vrcholu v ľavom hornom rohu. Pokiaľ bude vrchol nekorektný, skontrolujú sa ostatné vrcholy. Pokiaľ sa druhý nekorektný vrchol nenachádza v pravom dolnom rohu, funkcia otočí príslušnou stenou tak, aby sa tam nachádzal. Prevedie sa daný algoritmus a stena bude vrátená do predchádzajúcej polohy. Tento proces sa opakuje, kým nebudú všetky vrcholy korektné, teda na korektnom mieste a správne natočené. V prípade, že bude nekorektný iba jeden vrchol, kocku nie je možné zložiť, pretože ju niekto porušil. Funkcia `MakeYellowCorner` je posledným krokom k úplnému zloženiu Rubikovej kocky.




Obr. 6.8: Na obrázku je znázornený algoritmus, ktorý zaistí otočenie vrcholov tak, aby boli všetky korektné.

## 6.4 Grafické užívateľské rozhranie

Android Studio, v ktorom je aplikácia implementovaná, poskytuje sadu základných objektov pre tvorbu grafického užívateľského rozhrania. Vývoj GUI je v Android Studio jednoduchý a intuitívny, pretože poskytuje *Layout Editor*, v ktorom je možné si layout vybudovať graficky a len malé detaily upraviť prostredníctvom kódu. Zo sady základných objektov API som pri implementácii zväčša využívala úplne základné objekty ako: `ImageView` – pre zobrazenie obrázku, `TextView` – pre zobrazenie textu, `Button` – pre zobrazenie tlačítka a ďalšie. Preto som sa rozhodla popísať len implementáciu zložitejších aktivít. Všetky obrazovky aplikácie sú znázornené na obrázku 6.9 v závere tejto podkapitoly.

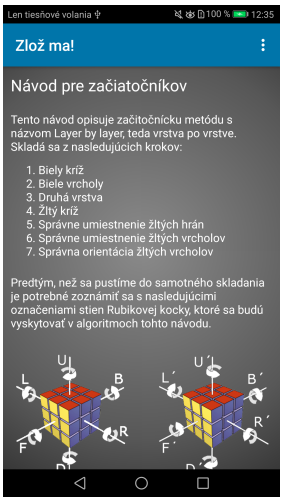
### Fotoaparát

Ako prvé bude užívateľovi poskytnutá informácia o tom, akú snímku očakávam. Následne po kliknutí na tlačítko „Nasnímať stenu kocky“ sa mu zobrazí fotoaparát. Ten bol implemetovaný pomocou preddefinovanej `Intent` akcie `MediaStore.ACTION_IMAGE_CAPTURE`. Vďaka tejto akcii je práca s foroaparátom zaistená pomocou preddefinovaných funkcií, teda je zaistené automatické zaostrenie, blesk a ďalšie typické funkcie fotoaparátu. Po vyfotení snímky bude snímka zobrazená užívateľovi, ktorý ju bude musieť odsúhlasiť. V prípade, že snímku zamietne, bude fotoaparát automaticky zobrazený a užívateľ bude môcť danú stenu nasnímať ešte raz. Po odsúhlasení je táto snímka vrátená aplikácii v podobe `uri`, ktorú následne načítam do datoveho typu `Bitmap`. Keďže OpenCV pracuje s obrázkom dátového typu `Mat`, je nutné pred volaním samotného spracovania obrazu prevod z `Bitmap` do `Mat`. Tento prevod je možný vďaka `Utils`, ktorý ho ponúka ako funkciu `bitmapToMat`. Po tomto prevode je obrázok predaný do procesu detekcie Rubikovej kocky, ktorý bol popísaný v podkapitole 6.2.

### Grafický 3D model Rubikovej kocky

Grafický model Rubikovej kocky je zložený z 27 malých kociek, vyskladaných ako kocka v priestore. Ako som už spomenula, na prácu s grafikou bola využítá knižnica OpenGL ES, ktorá definuje dve základne triedy potrebné k vytvoreniu a manipulácii s grafikou. Vytvorený grafický 3D model Rubikovej kocky je vykreslený do objektu `GLSurficeView`, ku ktorému sa viaže naimplementovaná trieda `CubeSurficeView`, ktorá dedí od `GLSurficeView`. V tejto triede je ďalej implementovaná preddefinovaná funkcia `onTouchEvent`, vďaka ktorej je zaistená rotácia objektu v súvislosti so zmenou polohy dotyku na obrazovke. Pre samotné vykreslenie je potrebné implementovať tiež druhú základnú triedu. Túto triedu som nazvala `CubeRenderer` a dedí od už definovanej triedy `GLSurfaceView.Renderer`. Vďaka nej je možné nie len vykresliť vytvorenú grafiku, ale tiež prekresľovanie tejto grafiky, ktoré je potrebné napríklad pri už spomínanej rotácii alebo iných zmenách grafiky. Tento grafický 3D model je znázornený napríklad na obrázku 6.9e.

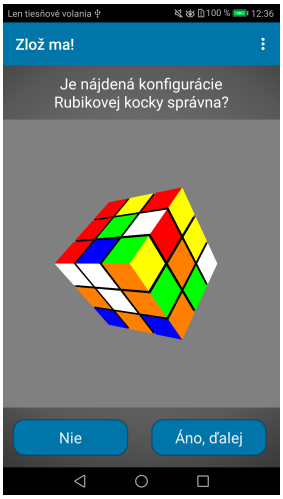

(a) Úvodná stránka



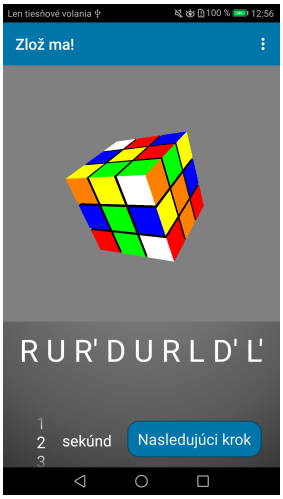

(b) Potrebná teória


(c) Nasnímanie kocky (pred volaním fotoaparátu)

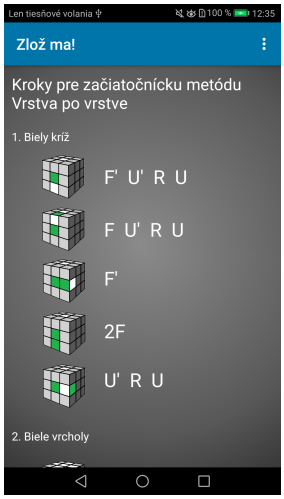
(d) Nasnímanie kocky (po nafotení)




(e) Kontrola konfigurácie




(f) Solver




(g) Prehľad krokov metódy

Obr. 6.9: Grafické užívateľské rozhranie aplikácie.

# Kapitola 7

# Testovanie aplikácie

Testovanie je neoddeliteľnou súčasťou vývoja akejkoľvek aplikácie, programu, informačných systémov a ďalších vývojov nie len v oblasti informačných technológii. Testovanie môžeme rozdeliť na dva základné typy: *aplikačné testovanie*, ktoré je popísané v podkapitole 7.1 a *užívateľské testovanie* popísané v podkapitole 7.2.

## 7.1 Aplikačné testovanie

Aplikačné testovanie je súčasťou každej fázy vývoja. Po každej implementačnej časti som ju dôkladne otestovala. Ako už bolo spomenuté, implementácia bola rozdelená do dvoch hlavných implementačných častí a to detekcia aktuálnej kofigurácie Rubikovej kocky a postup riešenia. V každej implementačnej časti prebehlo testovanie viacerých fáz danej implementačnej časti. Po dokončení implementácii prebehlo ešte jedno testovanie a to testovanie celej aplikácie, teda všetkých súčastí spoločne. Pri testovaní som využívala tri klasické Rubikove kocky. Každá z nich je od iného výrobcu a líšia sa hlavne odtieňom a sýtosťou jednotlivých šiestich farieb.

### 7.1.1 Testovanie detekcie aktuálnej konfigurácie Rubikovej kocky

Ako prvé som testovala detekciu steny Rubikovej kocky v obraze. Toto testovanie prebehlo na 72 snímkoch steny Rubikovej kocky. Každú stenu Rubikových kociek som nafotila v štyroch rôznych situáciach – za denného svetla s neutrálnym pozadím, za denného svetla s rušivým pozadím, za horšieho svetla s neurálnym pozadím a za horšieho svetla s rušivým pozadím. Toto testovanie dopadlo s 94,4% úspešnosťou. Pri štyroch obrázkoch nebola stena správne detekovaná z dôvodu, že pri nafotení s rušivým pozadím sa vyskytoval objekt, ktorý splýval s vonkajšou hranou steny. Funkcia `findContours` následne určila tento objekt ako súčasť steny Rubikovej kocky a teda nebolo možné túto kontúru zapísať pomocou štyroch bodov. Tento problém sa pri finálnej verzii aplikácie vyriešil nasnímaním danej steny znovu s neutrálnejším pozadím respektíve pri lepších svetelných podmienkach.

Ďalej bolo potrebné dôkladne otestovať rozpoznávanie jednotlivých farieb. Toto testovanie prebehlo taktiež na 72 snímkach Rubikovej kocky za už spomenutých štyroch situácii. Testovanie dopadlo s úspešnosťou 88,8%. Jedna z testovaných Rubikovych kociek má mierne poškodenú a vyblednutú oražovú farbu, ktorá sa pri horšom svetle javila ako svetločervená. Farby zvyšných dvoch kociek boli detekované za každej situácie správne. Táto situácia sa dala riešiť len nasnímaním kocky za dobrého svetla.

Posledným testovaním v tejto implementačnej časti bolo nájdenie aktuálnej konfigurácie. Jednotlivé steny Rubikovej kocky boli prevedené na grafický 3D model tejto kocky. Tento model následne bolo potrebné stenu po stene odkotrolovať s reálnou Rubikovou kockou. Testovala som to na 12-tich rôznych konfiguráciach na troch Rubikovych kockách, teda 4 rôzne konfigurácie jednej Rubikovej kocky. Toto testovanie dopadlo s úspešnosťou 100%. Jedinou vínimkou bola už vyššie spomenutá Rubikova kocka, kde bola na dvoch stenách zle rozpoznaná oražová farba, to ale nesúviselo so správnym prevedením stien na grafický 3D model. Prevedenie bolo správne.

### 7.1.2 Testovanie postupu riešenia Rubikovej kocky

Ako prvé bolo potrebné otestovať naimplementovaný pohyb alebo inak otočenie steny. Toto testovanie bolo veľmi dôkladné, pretože nesprávne otočenie steny by znamenalo, že následne implementované algoritmy nebudú prevedené správne a Rubikovu kocku nebude možné zložiť. Otáčania som sa rozhodla testovať najprv na zloženej Rubikovej kocke, pretože po otočení danej steny je hneď vidieť, či bola otočená správne. Pokiaľ bola stena otočená správne, skontrolovala som ju na rozloženej Rubikovej kocke, kde je toto skontrolovanie oveľa zložitejšie.

Po implementácii všetkých otočení stien v smere aj v protismere hodinových ručičiek prebehlo testovanie tak, že na zloženej Rubikovej kocke bola prevedená sekvencia otáčaní niektorých stien. Túto sekvenciu som taktiež previedla na reálnej Rubikovej kocke a skontrolovala, či sa po prevedení Rubikova kocka a grafický 3D model v aplikácii zhodujú. Toto testovanie dopadlo taktiež na 100%. Tým, že som dôkladne kontorlovala každý pohyb po jeho implementácii, som predišla zlému pobyhu pri testovaní viacerých pohybov súčasne.

Testovanie samotnej začiatočníckej metódy prebiehalo podobne ako pri otáčaní stien. Toto testovanie prebehlo na jednej nájdenej konfigurácii Rubikovej kocky. Po implemetovaní danej funkcie predstavujúcej jeden zo siedmych krokov metódy som tento krok najprv otestovala. Po implementácii všetkých krokov a ich samotnom skontrolovaní som solver mohla začať testovať dôkladnejšie. Kedže existuje viac než 43 triliónov rôznych konfigurácii Rubikovej kocky, nie je možné otestovať všetky. Rozhodla som sa otestovať 10 rôznych konfigurácii, a to tak, že som na zloženej kocke previedla niekoľko otočení rôznych stien. Tie isté otočenia som tiež previedla na svojej Rubikovej kocke. Následne som všetky algoritmy, ktoré solver použil k jej zloženiu, previedla na reálnej Rubikovej kocke. Prevedením nájdených algoritmov som zložila všetky zadané konfigurácie kocky a solver som považovala za úspešný.

### 7.1.3 Testovanie celej aplikácie

Po doimplementovaní posledných detailov ako napríklad optimalizácie krokov, ktoré budú zobrazené užívatelovi, nasledovalo testovanie celej aplikácie. Na každej z Rubikovych kociek som vytvorila štyri rôzne kofigurácie, tie som následne nasnímala a zložila podľa návodu. Každá konfigurácia bola snímana za štyroch rôznych situácii tak, ako bolo uvedené už pri testovaní detekcie. Výsledky som vložila do tabuľky 7.1.

**Situácie, v ktorých bola nasnímaná daná konfigurácia:**

1. Rubikova kocka nasnímaná za denného svetla s neutrálnym pozadím.
2. Rubikova kocka nasnímaná za denného svetla s rušivým pozadím.

3. Rubikova kocka nasnímaná za horšieho svetla s neutrálnym pozadím.
4. Rubikova kocka nasnímaná za horšieho svetla s rušivým pozadím.

**Rubikove kocky, na ktorých prebehlo testovanie:**

(a) Rubikova kocka č. 1 (b) Rubikova kocka č. 2 (c) Rubikova kocka č. 3

Obr. 7.1: Na obrázkoch sú znázornené Rubikove kocky, ktoré boli použité na testovanie.

**Výsledky testovania:**

| **Rubikova kocka** | **Situácia č. 1** | | **Situácia č. 2** | | **Situácia č. 3** | | **Situácia č. 4** | |
|---|---|---|---|---|---|---|---|---|
| | Detekcia | Solver | Detekcia | Solver | Detekcia | Solver | Detekcia | Solver |
| **Rubikova kocka č. 1** | 6 / 6 | úspešný | 5 / 6 | úspešný po nafotení zlej steny | 4/ 6 | nevyužitý | 3 / 6 | nevyužitý |
| **Rubikova kocka č. 2** | 6 / 6 | úspešný | 6 / 6 | úspešný | 6 / 6 | úspešný | 4 / 6 | nevyužitý |
| **Rubikova kocka č. 3** | 6 / 6 | úspešný | 6 / 6 | úspešný | 6 / 6 | úspešný | 5 / 6 | úspešný po nafotení zlej steny |

| | | | | |
|---|---|---|---|---|
| **Celkový počet detekovaných stien v určitej situácii** | 100% | 94,45% | 88,89% | 66,67% |
| **Celková úspešnosť detekcie stien** | 87,5% | | | |
| **Úspešnosť solveru** | 100% | | | |

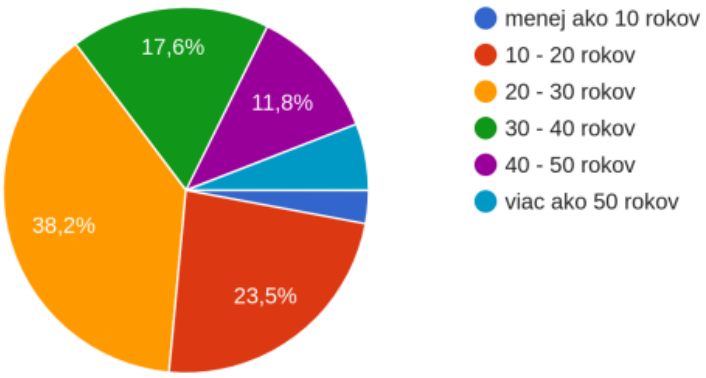
Tabuľka 7.1: Tabuľka výsledkov testovania

Ako je vidieť v tabuľke výsledkov, testovanie dopadlo s úspešnosťou vyššou ako 87,5%. Najväčší problém nastal pri snímkoch nasnímaních za horšieho svetla s rušivým pozadím, či už v podobe zle detekovanej farby alebo nesprávne detekovanej pozície. Naopak, pokiaľ bola daná Rubikova kocka nasnímaná za denného svetla, tento problém bol minimálny. Z tabuľky tiež vyplýva, že pokiaľ bola Rubikova kocka nasnímaná za denného svetla na neutrálnom pozadí, je detekcia 100%. Ako je vidieť, pokiaľ bola kocka správne detekovaná, tak solver tiež fungoval správne vo všetkých týchto prípadoch.

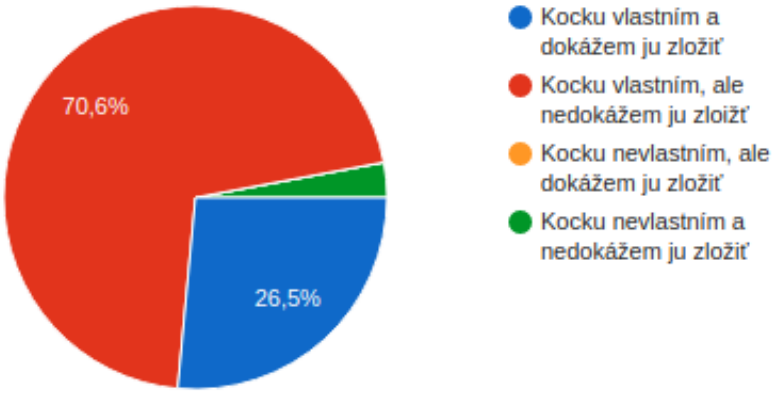
## 7.2 Užívateľské testovanie

Užívateľské testovanie prebehlo na 34 užívateľoch z môjho okolia. Chcela som, aby sa do testovania zapojili užívatelia, ktorí Rubikovu kocku vlastnia a nevedia ju postaviť, ale tiež tí, ktorí ju zložiť dokážu. Na obrázku 7.2 je znázornené percentuálne zastúpenie vekovej skupiny užívateľov, ktorí aplikáciu testovali a na obrázku 7.3 je znázornené percentuálne zastúpenie užívateľov podľa toho, či vlastnia alebo nevlastnia a súčasne vedia alebo nevedia Rubikovu kocku zložiť. Spätnú väzbu od užívateľov som dostala prostredníctvom dotazníka, ktorý je uvedený v prílohe A.

S výsledkami užívateľského testovania som spokojná, pretože ohlasy na aplikáciu boli pozitívne. Viacerí užívatelia mi prostredníctvom pripomienok oznámili, že pre nich bola aplikácia naozaj prínosom a boli šťastní, že vďaka nej prvý krát dokázali Rubikovú kocku zložiť. Niektorí sa v pripomienkach zmienili, že vďaka aplikácii sa naučili nový algoritmus, ktorý bol pomerne jednoduchší ako ten, ktorý doposiaľ ovládali. Farebné prevedenie aplikácie vyhovovalo 73,5% užívateľov a aplikácia bola pre nich intuitívna na 97,05%. Úspešnosť detekcie aktuálnej konfigurácie Rubikovej kocky bola 92,65%. Pri správne detekovanej konfigurácii zložili podľa nájdených algoritmov kocku všetci užívatelia, pokiaľ sa v priebehu skladania nepomýlili. Pre túto situáciu som implementovala vrátenie sa k predchádzajúcemu kroku, ktoré bolo z aplikácie po užívateľskom testovaní odstránené. Až 76,5% užívateľov túto možnosť označilo za neprínosnú. Vo väčšine prípadov bolo ako dôvod uvedené, že po ich chybe pri skladaní sa zmenila aktuálna konfigurácia ich Rubikovej kocky natoľko, že vrátiť sa k predchádzajucemu kroku v aplikácii im vôbec nepomohlo. Viedlo to k nasnímaniu novej konfigurácie Rubikovej kocky. V pripomienkach sa okrem vyššie uvedených pozitívnych ohlasov až v 82,3% vyskytla pripomienka, že by bolo vhodné teóriu potrebnú pre pochopenie algoritmov napísať na jednu stránku a označiť steny na jednom obrázku. Tento proces bol pôvodne implemetovaný na viacerých obrazovkách, čo užívateľom nevyhovovalo a dosť ich to odrádzalo, pretože to bolo príliš zdĺhavé.




Obr. 7.2: Graf, ktorý znázorňuje percentuálne zastúpenie vekovej skupiny užívateľov.




Obr. 7.3: Graf, ktorý znázornené percentuálne zastúpenie užívateľov podľa toho, či vlastnia alebo nevlastnia a súčasne vedia alebo nevedia Rubikovu kocku zložiť.

# Kapitola 8

# Záver

Cieľom tejto bakalárskej práce bolo vytvoriť mobilnú aplikáciu, ktorá slúži ako návod na zloženie Rubikovej kocky. Aplikácia umožňuje užívateľovi nafotiť steny Rubikovej kocky, ktoré sú následne rozpoznané. Zo získaných dát sa určí aktuálna konfigurácia danej kocky. Takto rozpoznaná kocka bude prevedená do grafického 3D modelu Rubikovej kocky, ktorý bude následne prezentovaný užívateľovi spoločne s nájdenou sekvenciou algoritmov začiatočníckej metódy potrebných k jej zloženiu.

Pred samotnou implementáciou bolo nutné naštudovať si teóriu potrebnú k dosiahnutiu požadovaného cieľa. Ako prvé bolo potrebné oboznámiť sa so spôsobom skladanie Rubikovej kocky, označeniami stien a rôznymi metódami skladania. Ďalej bolo nutné si naštudovať spracovanie obrazu, ktoré je pre dosiahnutie cieľa najpodstatnejšie. K tejto tematike patrí detekcia hrán, prahovanie, vyhladzovacie filtre, farebné modely a ďalšie. Aplikácia bola vyvinutá pre operačný systém Android a bola naimplementovaná vo vývojom prostredí Android Studio. Preto bolo nutné oboznámiť sa so samotným Andoid OS a tiež spomenutým vývojovým prostredím. Pri implementácii bolo potrebné využiť knižnicu OpenCV, ktorá bola využitá pri spracovaní obrazu a knižnicu OpenGL ES, vďaka ktorej je možné zobraziť grafický 3D model Rubikovej kocky.

Po implementácii bolo nutné aplikáciu otestovať na viacerých Rubikových kockách, ktorých farby sa mierne líšili v odtieni alebo sýtosti danej farby a mali odlišnú konfiguráciu. Súčasťou testovania bolo aj testovanie aplikácie užívateľmi. Toto testovanie dopadlo dobre a užívatelia ohodnotili aplikáciu pozitívne.

Výsledná aplikácia je funkčná pre klasickú Rubikovu kocku 3x3x3. Môžu však nastať problémy pri rozpoznávaní farieb alebo pri detekcii hrán. V prípade, že je stena Rubikovej kocky odfotená pri zlom svetle alebo vznikol veľký odlesk pri použití blesku, nemusí byť daná farba rozpoznaná správne, čo vedie k nesprávne určenej konfigurácii. V prípade, že bude Rubikova kocka nasnímaná na podklade, ktorý bude príliš splývať s vonkajšou kontúrou steny, môže sa stať, že nebude kocka vôbec detekovaná.

V budúcnosti bude aplikácia ešte rozšírená o pokročilú metódu. Ďalší vývoj aplikácie v oblasti počítačového videnia by mohol riešiť vyššie uvedené problémy. Riešením by mohol byť napríklad vylepšený detektor hrán alebo odstránenie odlesku. Okrem týchto vylepšení by mohla byť aplikácia časom rozšírená aj o metódy skladania pre Rubikove kocky rôznych veľkostí.

# Literatúra


[1] *Open Handset Alliance*. 2007. [Online; navštíveno 15.03.2020]. Dostupné z: http://www.openhandsetalliance.com.

[2] *Announcing the Android 1.0 SDK, release 1*. September 2008. [Online; navštíveno 30.03.2020]. Dostupné z: https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html.

[3] *Android 1.5 Platform Highlights*. Apríl 2009. [Online; navštíveno 30.03.2020]. Dostupné z: https://developer.android.com/about/versions/android-1.5-highlights.html.

[4] *Introduction*. OpenCV, júl 2018. [Online; navštíveno 13.04.2020]. Dostupné z: https://docs.opencv.org/3.4.2/d1/dfb/intro.html.

[5] *Distribution dashboard*. 2020. [Online; navštíveno 30.03.2020]. Dostupné z: https://developer.android.com/about/dashboards.

[6] *Meet Android Studio*. 2020. [Online; navštíveno 30.03.2020]. Dostupné z: https://developer.android.com/studio/intro.

[7] *Mobile and Tablet Android Version Market Share Worldwide*. Marec 2020. [Online; navštíveno 30.03.2020]. Dostupné z: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide.

[8] *OpenGL ES*. Marec 2020. [Online; navštíveno 15.04.2020]. Dostupné z: https://developer.android.com/guide/topics/graphics/opengl.

[9] *Platform Architecture*. 2020. [Online; navštíveno 01.04.2020]. Dostupné z: https://developer.android.com/guide/platform/.

[10] *Rankings | World Cube Association*. World Cube Association, 2020. [Online; navštíveno 03.03.2020]. Dostupné z: https://www.worldcubeassociation.org/results/rankings/333/single.

[11] Canny, J. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1986, PAMI-8, č. 6, s. 679–698.

[12] Noor, A., Mokhtar, M., Rafiqul, Z. a Pramod, K. Understanding color models: A review. *ARPN Journal of Science and Technology*. 2012, zv. 2, č. 3, s. 265–275.

[13] Pei-Yung Hsiao, Chia-Hsiung Chen, Shin-Shian Chou, Le-Tien Li a Sao-Jie Chen. A parameterizable digital-approximated 2D Gaussian smoothing filter for edge detection in noisy image. In: *2006 IEEE International Symposium on Circuits and Systems*. 2006, s. 4 pp.–.

[14] Roy, P., Dutta, S., Dey, N., Dey, G., Chakraborty, S. et al. Adaptive thresholding: A comparative study. In: *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. 2014, s. 1182–1186.

[15] UK, E., ed. *Ako zložiť Rubik's kocku*. EGMONT, 2018. Rubik's Cube Series. ISBN 978-80-252-4253-7.

[16] Šikudová, E., Černeková, Z., Benešová, V., Haladová, Z. a Kučerová, J. *Počítačové videnie. Detekcia a rozpoznávanie objektov*. 1. vyd. Wikina, 2011. ISBN 978-80-87925-07-2.

# Príloha A

# Dotazník pre užívateľské testovanie

# Testovanie aplikácie Zlož ma!

Tento dotazník slúži ako spätná väzba od užívateľov, ktorý sa zapojili do testovania aplikácie "Zlož ma!". Aplikácia slúži ako návod na zloženie Rubikovej kocky.

* Povinné

1. Vek *

*Označte iba jednu elipsu.*

- menej ako 10 rokov
- 10 - 20 rokov
- 20 - 30 rokov
- 30 - 40 rokov
- 40 - 50 rokov
- viac ako 50 rokov

2. Ako často využívate mobilný telefón? *

*Označte iba jednu elipsu.*

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Nevyužívam vôbec | | | | | | | | | | | Využívam často |

3. Na akú aktivitu využívate svoj mobilný telefón? *

*Začiarknite všetky vyhovujúce možnosti.*

- [ ] Hovory a SMS
- [ ] Sociálne siete
- [ ] Online noviny
- [ ] E-maily
- [ ] Fotografie a videá
- [ ] Náučné aplikácie
- [ ] Hry

4. Vlastníte a dokážete Rubikovu kocku zložiť? *

*Označte iba jednu elipsu.*

- Kocku vlastním a dokážem ju zložiť
- Kocku vlastním, ale nedokážem ju zloižť
- Kocku nevlastním, ale dokážem ju zložiť
- Kocku nevlastním a nedokážem ju zložiť

5. Ako sa Vám páči celkový vzhľad aplikácie? *

*Označte iba jednu elipsu.*

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Nepáči sa mi | | | | | | | | | | | Páči sa mi |

6. Vyhovuje Vám zvolené farebné prevedenie aplikácie? *

*Označte iba jednu elipsu.*

- Áno
- Nie, radšej by som zvolila tmavšie prevedenie
- Nie, radšej by som zvolila svetlejšie prevedenie

7. Intuitívnosť aplikácie (jednoduchosť ovládania, celková prehľadnosť aplikácie) *

*Označte iba jednu elipsu.*

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Neintuitívne | | | | | | | | | | | Intuitívne |

8. Úspešnosť nájdenia aktuálnej konfigurácie (v približných percentách) *

*Označte iba jednu elipsu.*

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Nedošlo vôbec k nájdeniu aktuálnej konfigurácie | | | | | | | | | | | Vždy došlo k nájdeniu aktuálr |

9. Viedli nájdene algoritmy k zloženiu Rubikovej kocky? *

*Označte iba jednu elipsu.*

- Áno, vždy bola Rubikova kocka zložená
- Nie vždy
- Vôbec, ani raz sa Rubikovu kocku nepodarilo zložiť

10. Bola možnosť vrátiť sa k predchádzajúcemu kroku prínosná? *

*Označte iba jednu elipsu.*

- Áno
- Nie

11. V prípade, že pre Vás možnosť predchádzajúceho kroku nebola prínosná, uveďte prosím dôvod.

12. Doporučili by ste aplikáciu známym?

*Označte iba jednu elipsu.*

- Áno
- Nie

13. Pripomienky *

Sem prosím uveďte akých nedostatkov ste si v aplikácii všimli, nápady pre ďalšie vylepšenie a podobne.

Google Formuláre