VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Editor map pro Android

Map Editor for OS Android

2015 David Buček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **David Buček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Editor map pro Android
Map Editor for OS Android

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit nástroj, který převede do digitální podoby mapu zaznamenanou na čtverečkovaném papíře. Rozlišovat se budou zdi a případně i jiné významné objekty (průchody, předměty, apod.) Vstupní data bude možné získat z obrázku nebo přímo pomocí zabudovaného fotoaparátu zařízení s OS Androidem. Práce bude také obsahovat ukázkovou aplikaci využívající takto získanou mapu.

1. Rešerše algoritmů vhodných pro rozpoznání nákresů map.
2. Implementace knihovny vytvářející model mapy na základě zpracování obrazu plánku podlaží.
3. Testování úspěšnosti rozpoznání zpracovávaných nákresů.
4. Vytvoření demonstrační aplikace využívající tuto knihovnu.
5. Závěr a shrnutí dosažených výsledků.

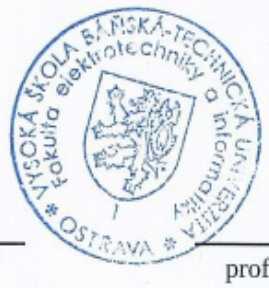
Seznam doporučené odborné literatury:

[1] Rafael C. Gonzalez, Richard E. Woods, Digital Image Processing (3rd Edition), Prentice Hall, 2007, ISBN 978-0131687288
[2] Mark Nixon, Feature Extraction & Image Processing for Computer Vision, Third Edition, Academic Press; 3 edition, 2012, ISBN 978-0123965493
[3] Wilhelm Burger, Mark J. Burge, Digital Image Processing: An Algorithmic Introduction using Java, Springer, 2012, ISBN 978-1846283796

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Ing. Michal Krumnikl, Ph.D.**

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 3. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 3. května 2015 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Cílem této práce je vytvoření funkční knihovny pro rozpoznávání ručně kreslených map. Implementace je realizována v jazyce Java pro OS Android i desktopovou edici Java SE. Součástí práce je shrnutí metod pro rozpoznávání obrazu, testování vzniklé knihovny a ukázková aplikace.

Klíčová slova: Java, Android, JSON, rozpoznávání obrazu

Abstract

The target of this thesis is to create library for handcrafted map recognition. Implementation for OS Android, as well as desktop edition, is realized in Java programming language. Thesis includes a summary of image recognition methods, test cases for final library and sample application.

Keywords: Java, Android, JSON, image recognition

Seznam použitých zkratek a symbolů

ADT	–	Android Development Toolkit
ASCII	–	American Standard Code for Information Interchange
BSD	–	Berkeley Software Distribution
EAN	–	International Article Number
GPS	–	Global Positioning System
GUI	–	Graphical User Interface
JSON	–	JavaScript Object Notation
JSON-LD	–	JavaScript Object Notation for Linked Data
NDK	–	Native Development Kit
NFC	–	Near Field Communication
OpenCV	–	Open Source Computer Vision
OS	–	Operační systém
RPG	–	Role Playing Game
TCP	–	Transmission Control Protocol
QR code	–	Quick Response Code
XML	–	Extensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V průběhu posledního desetiletí došlo k prudkému vývoji mobilních technologií, a s tím souvisejícího příslušenství. Součástí dnešních mobilních telefonů jsou senzory pro snímání polohy, pohybu, orientace, dále přední a zadní fotoaparát nebo přenosové služby zahrnující Bluetooth, Wi-Fi, NFC apod.

Tyto technologie lze prakticky využít v mapových aplikacích (GPS navigace, nalezení nejbližšího místa), čtení čárových kódů (EAN, QR code) či jiných typů značení, přenosu dat mezi zařízeními (sdílení souborů, vzdálené ovládání) nebo v herním průmyslu. Dobrým příkladem takovéhoto využití mohou být ovladače moderních herních konzolí (*PlayStation 3*, *PlayStation 4*, *Nintendo Wii*, atd.). Při ovládání her se rovněž využívá možností snímání a zpracování obrazu. Za zmínku stojí technologie *Microsoft Kinect*, která nachází využití i při vědeckých experimentech, nebo *PlayStation Move*, kdy je kamerou snímána poloha ovladače.

Účelem této práce je prozkoumat možnosti využití fotoaparátu mobilního telefonu k návrhu herních map. Využití lze najít u digitálních verzí klasických stolních her nebo při tvorbě vlastních úrovní do mobilních her.

Nejdříve jsou rozebrány způsoby, jakými se mapy kreslí, a jaké prvky jsou na nich vyobrazeny. Jsou zde stanovena pravidla, která musí mapa splňovat, a co vše bude výsledná knihovna zohledňovat. Podrobněji je tato problematika probrána v kapitole 1.1.

V části **Metody rozpoznávání obrazu** (viz 2) jsou pak rozebrány jednotlivé způsoby, jak z nakreslené mapy získat její digitální reprezentaci. Důraz je při tom kladen na obecné možnosti práce s obrazem a je zvolena konkrétní metoda, která bude v této práci použita.

Část **Návrh** (viz 3) pak obsahuje možné způsoby implementace, vybrané technologie a nástroje pro cílové řešení.

Kapitola **Implementace** (viz 4) rozebírá zajímavé části kódu výsledné knihovny. Současně je zde shrnuta celková struktura knihovny z pohledu kódu a popis jednotlivých tříd a způsob jejich použití.

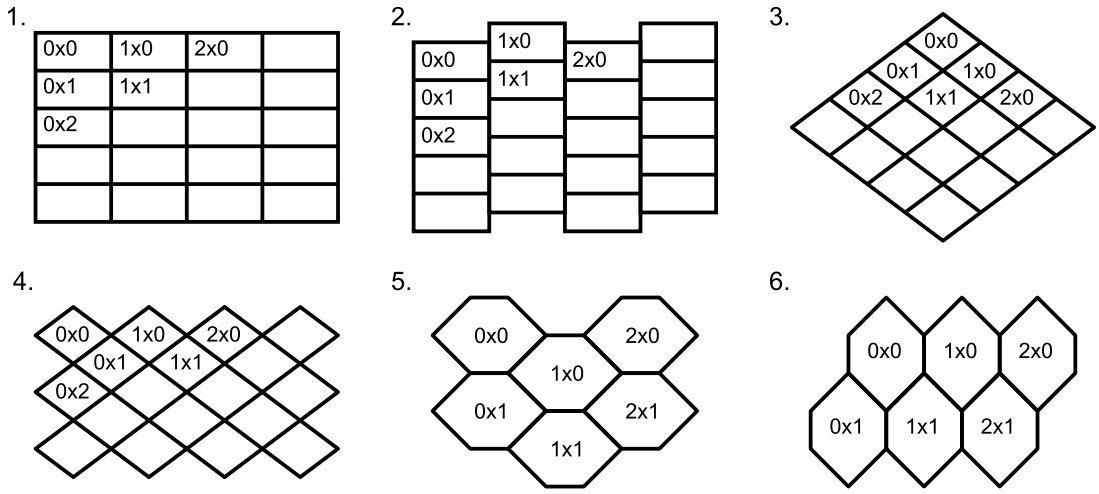
Testy knihovny a získané výsledky shrnuje kapitola **Testování** (viz 5). K práci je rovněž přiložena sada fotografií a naskenovaných snímků ručně kreslených map, včetně šablon pro vytvoření vlastní mapy.

Nedílnou součástí této práce je i ukázková aplikace, která demonstruje různé možnosti vzniklé knihovny. V kapitole **Ukázková aplikace** (viz 6) je uveden bližší popis, včetně návodu k použití.

1.1 Kreslení map

Mnoho deskových a společenských her využívá herní plán (mapu). V některých případech je pevně daný, v jiných je jeho tvorba závislá na kreativitě hráčů. Plán lze sestavovat z prefabrikovaných komponent nebo se celá mapa kreslí na papír. Kreslené mapy jsou typické pro většinu stolních her na hrdiny (RPG).

V mnoha případech je ale způsob vytvoření mapy závislý na herních pravidlech. Musí splňovat různá omezení, jakými jsou: Formát papíru, tvar dlaždic, jejich vzájemnou polohu, návaznost apod. Mapy se mohou skládat z dlaždic a nebo může být zarovnání



Obrázek 1: Typy map

č.	Český název	Anglický název
1.	Pravoúhlé	Orthogonal
2.	Pravoúhlé s posunem	Orthogonal, staggered
3.	Izometrické, normální	Isometric, diamond
4.	Izometrické, stupňovité	Isometric, staggered
5.	Šestiúhelníkové, vodorovné	Hexadecimal, horizontal
6.	Šestiúhelníkové, svislé	Hexadecimal, vertical

Tabulka 1: Rozdělení dlaždicových map

mapy omezeno sadou pravidel. V některých případech musí být mapa kreslena do předem připravené šablony. Stejně tak bývá omezen i seznam prvků, které mohou být na mapu zaneseny. [1]

Typickým zástupcem takových her je například *Dungeons & Dragons*, která využívá několik typů map současně. Nejčastěji používané formáty dlaždic zobrazuje obrázek 1. Rozdělení dlaždicových map zobrazuje tabulka 1. [2]

Pro samotné hraní je pak často důležité i číslování dlaždic, které usnadňuje orientaci hráčů na mapě. Všeobecně známé je například značení polí běžné šachovnice, kde sloupce jsou označeny písmeny abecedy od **A** po **H** a řádky číslicemi od **1** do **8**. Způsob zápisu je u každé hry jedinečný a závisí opět na pravidlech. Při číslování závisí často i na logice uspořádání dlaždic. Možná číslování jsou součástí obrázku 1. [3]

1.2 Digitální tvorba map

Při vytváření mapy je důležité rozlišit, zda-li je nutné uchovávat mapu jen jako obrázek anebo i data o ní.

1.2.1 Vizuální mapy

Nástroje pro tvorbu vizuálních dat slouží pouze k vytvoření obrázku mapy. Může jít o samostatné nástroje nebo rozšíření pro různé grafické editory.

Pro vytváření vlastních dlaždic lze použít nástroj *ShoeBox* postavený nad *Adobe Air*. Rozšíření obsahuje sadu funkcí pro zjednodušení práce, včetně exportu vzniklých dlaždic pro různé jiné nástroje. [4]

Zástupcem stand-alone editorů je například *Campaign Cartographer 3*. Jde o komerční produkt společnosti ProFantasy Software Ltd. Nástroj obsahuje velké množství dokoupitelných balíčků vzhledů a rozšíření. [5]

1.2.2 Mapy ve formě dat

Vedle nástrojů pro tvorbu vizuálních map existují i nástroje, jejichž výstupem není pouze obrázek mapy, ale její datová reprezentace. Většinou jde o specializované programy, ale lze najít i nástroje schopné pracovat obecně s mapou dlaždic.

Mezi takové nástroje patří open-source editor *Tiled*. Pro uložení mapy používá vlastní formát souborů s příponou `.tmx`, který je založen na XML. Podporováno je několik druhů dlaždic:

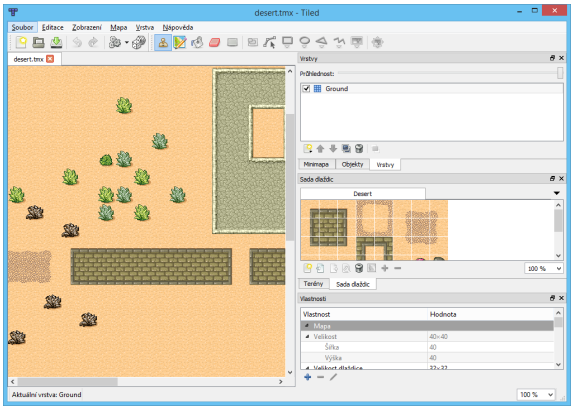
- pravoúhlé,
- izometrické,
- šestiúhelníkové.

Do nástroje lze nahrát vlastní sady dlaždic a obrázků, samotná mapa je pak členěna do vrstev. Samotné uložení dat vrstvy dlaždic může být v několika volitelných formátech, včetně kompresí. Editor umožňuje i exporty pro některé herní enginy nebo do jiných formátů, jako JSON. [2, 6]

V případě počítačových her bývají nástroje na tvorbu map často přímo jejich součástí nebo jsou distribuovány spolu s nimi. Existují rovněž programy, které umožňují mapy nejen vytvářet, ale rovnou je i využít v digitálních verzích deskových her.

1.3 Vlastní formát map

Tato práce se zaměřuje pouze na pravoúhlé mapy. Díky tomuto omezení je možné mapy zakreslovat na běžně dostupné čtverečkované papíry. Existují i specializované formáty papíru s šestiúhelníkovými vzory pro konkrétní RPG hry, které ale nejsou běžně dostupné. Velikost čtverečků v tomto případě nehraje roli, protože při vytvoření snímku papíru je velikost relativní vůči přiblížení objektivu.

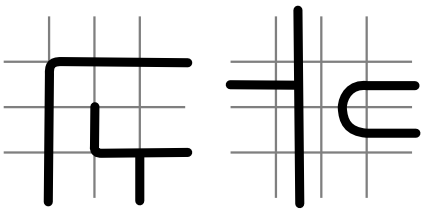


Obrázek 2: Snímek editoru Tiled

V rámci této práce budou uvažovány především formáty papíru **A4** a **A5** s velikostmi čtverečků **0,5 cm** a **1 cm**. Pro kreslení mapy lze použít jakoukoliv psací potřebu, nejlépe je použít permanentní lihový fix s hrotem velikosti **F** nebo **M**. Barva je s ohledem na postup rozpoznávání obrazu, který je popsán dále v práci, zanedbatelná. Nejčastěji byla v této práci používána barva **modrá** a **černá**.

Mapa bude na papíře zaměřena podle kót v rozích. Bližší popis a výběr tvaru je rozebrán později v této práci.

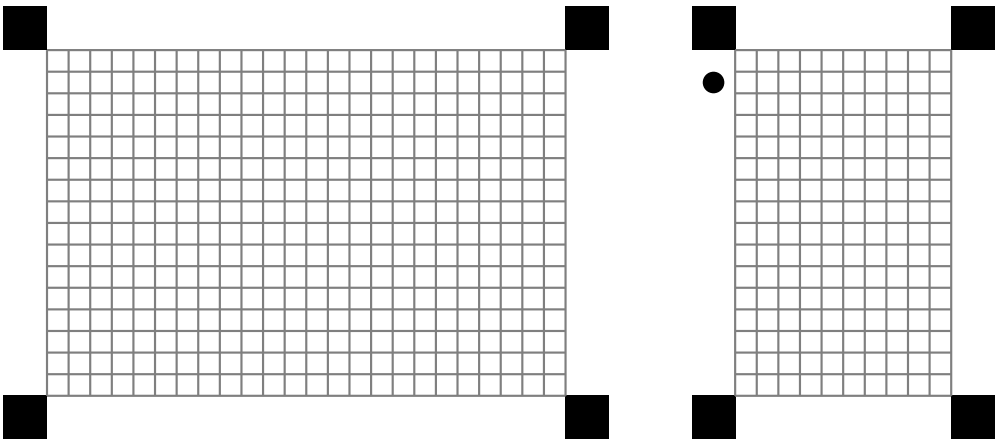
Jelikož má práce sloužit k návrhu map do her, je nejdůležitější částí rozpoznání zdí a cest na mapě. Ty mohou být zakresleny buď po hranách čtverečků nebo mohou být vedeny jejich středy. S ohledem na dlaždicovou představu reprezentace mapy bude použita druhá zmíněná možnost.



Obrázek 3: Odlišné způsoby zakreslení cesty do mapy

V rámci kreslení mapy není brán zřetel na velikost čtverečku na papíře, je ale nutné dodržet jeden z daných rozměrů mapy. Kóty jsou plně vybarvené čtverce zabírající oblast vždy 2x2 čtverečky a oblasti mezi jejich hranami musí zůstat prázdné. Výjimkou je doplňková značka pro velikost mapy. Samotná mapa se pak nachází uvnitř této vytyčené zóny a může mít rozměry:

- **Velká mapa**: Delší strana 24 čtverečků, kratší 16.
- **Malá mapa**: Delší strana 16 čtverečků, kratší 10.



Obrázek 4: Velikosti map včetně zobrazení značky

Velikosti map byly zvoleny s ohledem na běžné formáty papírů a velikosti čtverečků tak, aby se ideálně vešly na stránku. Aby bylo možné rozpoznat velikost mapy již z nákresu, je malá mapa opatřena značkou. Ta je umístěna jeden čtvereček od středu libovolné hrany kóty a jde o vyplněný kruh s průměrem velikosti čtverečku. Ukázka je zobrazena na obrázku 4.

2 Metody rozpoznávání obrazu

V rámci této práce je rozpoznání obrazu rozděleno do dvou větších celků, kdy každému z nich náleží specifické metody.

2.1 Morfologické zpracování obrazu

Morfologie je obecně chápána jako věda zabývající se tvarem a strukturou předmětů. Při zpracování obrazu je využívána matematická morfologie, která slouží jako nástroj pro jednoznačné vytyčení komponent na obrázku. Matematická morfologie vychází z vlastností bodových množin a její principy jsou založeny na nelineárních operacích. Nejčastěji se aplikuje nad binárními obrazy, lze ji ale snadno zobecnit i na barevné obrázky nebo na obrázky ve stupních šedi. Morfologické operace se používají pro předzpracování obrazu (odstranění šumu, zjednodušení tvaru), zdůraznění struktury objektů (kostra, ztenčování, zesilování) a pro popis objektů číselnými charakteristikami (plocha, obvod, projekce). [8]

Morfologické operace jsou realizovány jako relace obrazu (bodové množiny X) s jinou menší bodovou množinou B. Takováto množina se nazývá *strukturní element* (viz 2.1.1). Morfologickou transformaci si lze představit jako určitý systematický pohyb *strukturního elementu* B po obrazu X a vyhodnocení odezvy podle typu operace. Mezi dvě základní morfologické operace patří *dilatace* a *eroze*.

2.1.1 Strukturní element

Každý strukturní element se vztahuje ke svému lokálnímu počátku, někdy též nazývanému *aktuální* nebo *reprezentativní* bod. Výjimečným a nepříliš žádoucím případem je, když reprezentativní bod není součástí strukturního elementu. Uvažujme, že konkrétní strukturní element B je umístěn v nějaké poloze uvnitř obrazu. Výsledek jejich vzájemné relace se zapíše do výstupního obrazu na aktuální pozici. Pro binární obrazy je výsledek této relace buď 0 nebo 1.

Speciálním případem strukturního elementu je tzv. *izotropický strukturní element*, u něhož má morfologická transformace stejné chování ve všech směrech. [9]

2.1.2 Dilatace a eroze

Dilatace je operací, která zvětšuje objekty v binárním obraze, zatímco eroze je jejím přesným opakem. Obecné využití dilatace spočívá v zaplnění malých děr a nečistot obrazu. Eroze je naopak využívána k odstranění malých objektů nebo ke skeletonizaci. Z pohledu matematiky lze dilataci vyjádřit jako vektorový součet (rovnice 2) a erozi jako vektorový rozdíl (rovnice 3). K vyjádření dilatace je nutné nadefinovat reflexi (rovnice 1). [7, 10]

$$\hat{B} = \{w | w = -b, b \in B\} \tag{1}$$

$$X \oplus B = \{x | (\hat{B})_x \cap X \neq \varnothing\} \tag{2}$$

$$X \ominus B = \{x|(B)_x \subseteq X\} \tag{3}$$

Vlastnosti dilatace

- Komutativnost: $X \oplus B = B \oplus X$
- Asociativnost: $X \oplus (B \oplus C) = (X \oplus B) \oplus C$
- Rostoucí transformace: $X \subseteq Y \Rightarrow X \oplus B \subseteq Y \oplus B$
- Invariance k translaci: $X_T \oplus B = (B \oplus X)_T$
- Sjednocení posunutých bodových množin: $X \oplus B = \bigcup_{b \in B} X_b$

Vlastnosti eroze

- Antiextenzivnost: $(0,0) \in B \Rightarrow X \ominus B \subseteq X$
- Rostoucí transformace: $X \subseteq Y \Rightarrow X \ominus B \subseteq Y \ominus B$
- Invariance k translaci: $X_T \ominus B = (B \ominus X)_T$
- Průnik posunutých bodových množin: $X \ominus B = \bigcap_{b \in B} X_b$

2.1.3 Otevření a uzavření

Výše zmíněné operace se nejčastěji využívají ve vzájemné kombinaci se stejným strukturním elementem. Rozdíl mezi operací otevření (rovnice 4) a uzavření (rovnice 5) spočívá pouze v pořadí aplikace dilatace a eroze. [7, 11, 12]

$$X \circ B = (X \ominus B) \oplus B \tag{4}$$

$$X \bullet B = (X \oplus B) \ominus B \tag{5}$$

Tyto operace jsou využívány v rámci této práce při analýze obrázku s mapou k začištění obrazu.

2.1.4 Náhodná transformace

Náhodná transformace (Hit-or-Miss) je založena na dvou strukturních elementech. Element B_1 představuje vzor hledaného objektu, element B_2 pak koresponduje s pozadím obrázku. Transformaci lze tedy vyjádřit rovnicí 6. [7]

$$X \circledast B = \left(X \ominus B_1\right) - \left(X \oplus \hat{B}_2\right) \tag{6}$$

2.1.5 Skeletonizace

V mnoha případech se vyplatí omezit zkoumaný objekt pouze na jeho kostru. Kostra zachovává co nejlépe tvar objektu a je tvořena množinou čar minimální šířky. Každý bod na kostře tvoří střed kružnice vepsané, která se dotýká hranice objektu ve dvou nebo více bodech. [7, 11]

2.2 Segmentace obrazu

Jednou z významných částí analýzy obrazu je jeho segmentace. Jejím účelem je získat informace o objektech v obrazu, které jsou reprezentovány nepřekrývajícími se oblastmi. Segmentace může být úplná nebo částečná podle toho, zda oblasti souhlasí s objekty na původním obraze, či nikoliv. Pro účely analýzy obrazu není samotná segmentace dostačující a na její výsledky je potřeba aplikovat další metody. [7, 13]

2.2.1 Prahování

Jedna z jednoduchých a často používaných segmentačních metod je prahování. Většinou představuje část postupu v komplexnějších úlohách. Z pohledu implementace je tato metoda velmi jednoduchá a časově nenáročná. Matematicky lze prahování vyjádřit pomocí rovnice 7, kde g představuje výstup, f vstup a T hodnotu prahu. [7, 14]

$$g(x,y) = \begin{cases} 1 & f(x,y) > T \\ 0 & f(x,y) \leq T \end{cases} \tag{7}$$

Pokud je hodnota prahu v rámci celého obrázku stejná, nazývá se prahování *globálním*. V případě *adaptivního* prahování se hodnota prahu určuje pro každý zpracovávaný bod obrázku zvlášť (tzn. v průběhu zpracování se mění). [7]

3 Návrh

Výsledkem práce bude knihovna pro analýzu obrazu a rozpoznávání map, která bude rozčleněna na několik částí.

Základem bude sada tříd a rozhraní pro obecnou práci s obrazem, která musí splňovat následující požadavky:

- Třídy potřebné k základní práci (celočíselný bod a hranice obdélníkové oblasti v dvojrozměrném prostoru).
- Obecné rozhraní pro práci s obrázkovou bitmapou.
- Třídy pro uchování obrázků.
- Metody pro základní operace s obrazem, například:
  - Práce s celým obrázkem (vyplnění oblasti se stejnou hodnotou pixelu, vykreslení čar a převod indexů na barvy).
  - Práce s jednotlivými pixely obrázku (zvýšení/snížení hodnoty pixelu, vynásobení hodnoty pixelu zvoleným číslem, oříznutí hodnoty pixelu na danou mez nebo nahrazení hodnoty pixelu jinou hodnotou).
  - Vytváření nových obrázků nebo vyříznutí podoblasti jako nový obrázek.
  - Pokročilejší metody pro práci s obrázkem (dilatace a eroze).

Nedílnou součástí jsou i třídy pro analýzu obrazu a třídy zobecňující práci s mapou. Ty umožňují zpracovat a uchovat data dlaždic a mapy. Obecně se tedy tato sada dá rozdělit na:

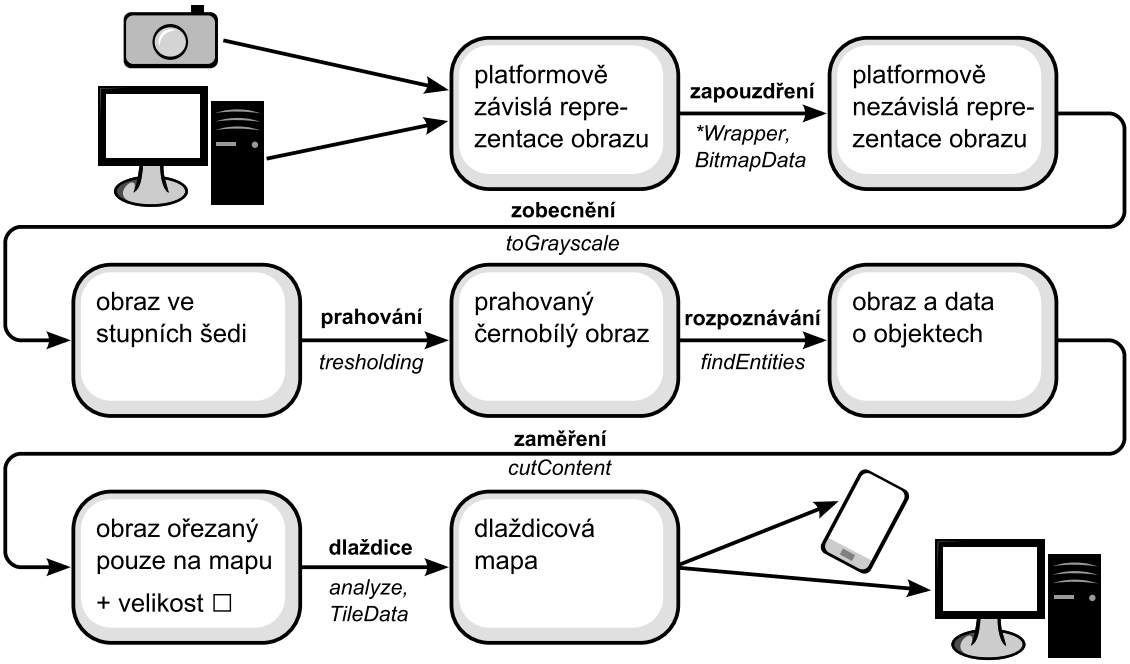
- Třídy pro práci s dlaždicovými mapami.
- Rozhraní a výčtové typy pro práci s formáty papíru.
- Třídy a rozhraní schopné rozpoznávat mapy.

Nezávisle na výše zmíněných částech knihovny jsou v práci zahrnuty i na platformě závislé komponenty. Konkrétně jde o sadu tříd, která přemosťuje existující třídy pro práci s obrázky tak, aby byly použitelné s knihovnou. Práce bude obsahovat toto řešení pro platformy **Android** a **Java SE**.

3.1 Pokročilé funkce

Vedle již zmíněných funkcí pro práci s obrazem bude knihovna obsahovat i funkce pokročilejší. Pro potřeby zpracování a rozpoznání mapy budou využity funkce pro prahování obrazu, vyplnění semínkovou metodou, hledání momentů apod. Funkce budou navrženy všeobecně s důrazem na jejich možnou znovupoužitelnost v jiných aplikacích, než rozpoznání dlaždic.

Samotný průběh zpracování od vstupního obrázku, až po výslednou dlaždicovou mapu je zobrazen na obrázku 5.



Obrázek 5: Průběh zpracování obrázku

3.2 Volba programovacího jazyka

S ohledem na cílovou platformu (desktop a Android) připadá v úvahu několik programovacích jazyků. Aplikace pro operační systém Android lze psát v jazyce Java, s využitím *NDK* pak v jazyce **C/C++**. Existuje i možnost vytvářet aplikace v jiných jazycích, ať už vystavěných na platformě Java (**Groovy**, **Scala)** nebo využívajících skriptovací prostředí (**Lua**, **Python**). [18, 19, 20, 21]

Jazyk Java umožňuje snadnou přenositelnost mezi platformami, má jednoduchou syntaxi, kvalitní dokumentaci a je to primární jazyk pro vytváření aplikací pro Android. Mezi nevýhody Javy patří komplikovaný přenos na platformu *Apple iOS* nebo *Microsoft Windows Phone*. V tomto ohledu je výhodnější jazyk C/C++, který ale vyžaduje doplňující propojení do každé z platforem. Rovněž není ustálený co se týče podporované syntaxe, verzí a překladačů, což může způsobovat problémy při přenosu kódu. Další zmíněné možnosti (především pak skriptovací jazyky) se pro vytváření knihovny příliš nehodí. [15, 16]

Jelikož je knihovna cílena na Android, bude implementace probíhat v programovacím jazyce Java. V rámci implementace a testování bude zohledněna i možnost propojení s jazykem C/C++ pro potřeby lepšího výkonu.

3.3 Vývojářské nástroje

V době zahájení implementace byly k dispozici dva nástroje, které umožňují vývoj pro Android v jazyce Java. První možností je integrované vývojové prostředí *Eclipse s balíčkem ADT*, které se používá od prvních verzí Androidu. Novým řešením pro vývoj aplikací pro platformu Android je nástroj *Android Studio*, jehož první stabilní verze vyšla v prosinci 2014.

3.3.1 Eclipse s ADT

Nástroj určený pro vývoj aplikací v jazyce Java, ve kterém je sám napsán. V roce 2001 byl Eclipse uvolněn jako open-source, spravovaný Eclipse Foundation. Pro potřeby vývoje aplikací pro Android je rozšířen o balíček ADT. Ten přidává funkce vizuálního návrhu GUI, automatickou kompilaci Android aplikací a emulátor pro jejich spuštění. Součástí je i sada nástrojů pro ladění. [24]

3.3.2 Android Studio

Vývoj nového nástroje pro vytváření aplikací platformy Android byl oznámen v květnu 2013 na konferenci Google I/O. [22] První ukázka se objevila záhy po oznámení, do fáze beta se nástroj dostal v červnu 2014. Stejně jako Eclipse je i Android Studio vyvíjeno v jazyce Java. Jako základ posloužilo vývojové prostředí *IntelliJ IDEA*. [23]

Tento nástroj nebyl použit, protože v době zahájení vývoje nebyl k dispozici ve stabilní verzi. Dalším důvodem je rovněž fakt, že aktuální verze (na rozdíl od Eclipse) nepodporuje NDK. [25]

4 Implementace

Základní struktura knihovny byla popsána již v části Návrh. Tato kapitola se věnuje podrobnému popisu kódu, včetně konkrétní implementace tříd a rozhraní. Součástí jsou také ukázky zajímavých částí kódu a komentáře ke klíčovým prvkům knihovny.

Celá knihovna se nachází v balíčku `buc0044.libbp` a případných podbalíčcích. Jednotlivé typy dlaždicových map a způsob jejich rozpoznání je rovněž v samostatných balíčcích. Projekt ukázkové aplikace a testovací případy jsou umístěny mimo balíček knihovny v samostatných projektech.

4.1 Verze a kompatibilita

Implementace je provedena v programovacím jazyce *Java* ve verzi **6**. V době vzniku této práce byl jazyk Java dostupný již ve verzi **8** a nejvyšší vydaná stabilní verze operačního systému *Android* byla **5.x** (Lollipop). Ta, ve srovnání s verzí **4.x**, již podporuje jazyk Java ve verzi **7**. [26] S ohledem na kompatibilitu knihovny jsou využity pouze prvky kompatibilní s verzí **6**. Platformově závislá část pro Android je cílena na zařízení s verzí systému **4.x**, verze pro desktop vyžaduje běhové prostředí Java ve verzi **6** a vyšší.

Oproti Javě verze **6** přináší její vyšší verze podporu dynamicky typovaných jazyků ve virtuálním stroji, rozšířené možnosti použití struktury **switch**, diamond operátor, vylepšené číselné literály a vícenásobné typy výjimek v bloku **catch**. Od verze **8** lze používat tzv. *Lambda výrazy* a zpracování kolekcí pomocí proudů. Jelikož by žádná z výše jmenovaných vlastností nepřinesla při psaní knihovny žádnou výhodu, je Java ve verzi **6** dostačující. [27, 28]

4.2 Obecné prvky

S ohledem na přenositelnost knihovny bylo nutné oddělit implementaci od balíčků, které nejsou multiplatformní. Proto bylo nutné vytvořit i třídy reprezentující prvky, které se v jednotlivých platformách běžně nacházejí. Pro tyto potřeby vznikly třídy:

- `buc0044.libbp.Bounds`
- `buc0044.libbp.Point`

Obě třídy představují celočíselnou reprezentaci (obdélníkové oblasti a bodu) ve dvojrozměrném prostoru. Po konstrukci objektu jsou data v nich neměnná (immutable). Třída `buc0044.libbp.Bounds` rovněž zajišťuje validitu dat (hodnota levé hranice musí být nižší než hodnota pravé, totéž pro horní a dolní hranici). Rovněž jsou plně implementovány metody `equals()`, `hashCode()` a `toString()`.

Dále se v balíčku nachází rozhraní `buc0044.libbp.Paper`, které reprezentuje rozměry čtverečkovaného papíru. Počtem čtverečků na stránce se myslí garantovaný počet úplných čtverečků v daném rozměru. Součástí rozhraní je i výčtový typ pro nejčastější formáty (včetně velikosti čtverečku), jak je vyobrazeno ve zdrojovém kódu 1.

```
public interface Paper
{
  public enum Format implements Paper
  {
    A4_10MM_24_16(24, 16),
    A5_10MM_16_10(16, 10),
    A6_5MM_24_16(24, 16),
    A7_5MM_16_10(16, 10);

    ⋮
  }

  public int getNumberOfSquaresOnLongerSide();
  public int getNumberOfSquaresOnShorterSide();
}
```

Výpis 1: Ukázka části implementace rozhraní `Paper`

Poslední obecnou třídou knihovny je `buc0044.libbp.StopWatch`, sloužící k zachycení doby trvání jednotlivých výpočtů v nanosekundách. Třída se chová jako klasické stopky, kdy volání metody `tap()` provede zaznamenání a výpis času na standardní výstup.

4.3 Zpracování obrazu

Ke zpracování obrazu v rámci práce nebyla použita žádná řešení třetích stran. Celá analýza obrazu probíhá skrze vlastní třídy knihovny.

4.3.1 Přístup k obrázkům

Pro analýzu obrazu je potřeba umožnit načítat a uchovávat obrázky, včetně možnosti přistupovat k hodnotám jejich pixelů, případně je měnit. Obecný přístup k jednotlivým pixelům obrázku je zajištěn pomocí `buc0044.libbp.BitmapData`. Rozhraní umožňuje získat rozměry bitmapy a hodnotu námi vybraného pixelu. Ta je reprezentována datovým typem **`int`**, který v jazyce Java nabývá hodnot $\langle -2^{31}, 2^{31}-1 \rangle$. Rozhraní, stejně jako většina komponent knihovny, neřeší význam hodnoty pixelu. [29]

```
package buc0044.libbp;

public interface BitmapData
{
  public int getPixel(int x, int y);
  public int getWidth();
  public int getHeight();
}
```

Výpis 2: Deklarace rozhraní `BitmapData`

V případě, že je metoda `getPixel()` volána s argumenty mimo rozsah bitmapy, musí volání této metody způsobit výjimku `IndexOutOfBoundsException`. Metody `getWidth()` a `getHeight()` musí vracet vždy kladná čísla.

Obecným kontejnerem pro obrázky je třída `buc0044.libbp.BitmapBuffer`, která implementuje výše zmíněné rozhraní. Obrázek je v tomto případě uchován jako dvojrozměrné pole typu **`int`**. Objekt typu `BitmapBuffer` lze zkonstruovat třemi různými způsoby:

1. zadáním rozměrů (všechny pixely budou mít hodnotu 0),
2. zadáním zdroje pixelů (skrze rozhraní `BitmapData`, hodnoty pixelů se zkopírují),
3. zadáním jiného objektu typu `BitmapBuffer` (hodnoty pixelů se zkopírují).

Pro získání nové instance obrázku ve stupních šedi lze využít i volání statické metody `toGrayscale`, která předpokládá předání barevného obrázku (složky (R, G, B) se zprůměrují).

Třída obsahuje i variantu metody `getPixel()`, která přijímá souřadnice ve formě desetinného čísla a vrací interpolovanou hodnotu pixelů. Vedle základních metod pro práci s obrazem (blíže popsáno v části 3) obsahuje i několik metod pro práci s různými typy hodnot pixelů. Příkladem může být metoda `replaceByColors()` pro indexované obrázky (viz 4.3.2), která nahradí každou hodnotu představující objekt barvou. Samozřejmostí je implementace metod zděděných ze třídy `Object`.

4.3.2 Význam hodnoty pixelu

Většina metod pro zpracování obrazu v této knihovně chápe hodnotu pixelu svým vlastním způsobem. Je proto nezbytné řídit se při programování dokumentací knihovny. Důvodem je urychlení provádění výpočtů, které nevyžadují převádět bitmapu na příslušný typ, ale pracují s bitmapou obecně. Knihovna tedy nehlídá formát pixelů bitmapy (neprovádí test platnosti hodnot při vstupu), správné použití a zacházení s bitmapou je čistě v režii programátora. Nejčastěji je v metodách hodnota pixelu chápána jako jedna z následujících možností:

- černobílý obrázek (hodnota pixelu nabývá pouze hodnot $\{0, 1\}$),
- obrázek ve stupních šedi (hodnota pixelu nabývá pouze hodnot $\{0, 1, 2, \ldots, 255\}$),
- barevný obrázek,
- indexovaný obrázek.

Hodnoty pixelu v barevném obrázku jsou tvořeny složkami (R, G, B). Jde tedy o klasickou reprezentaci mícháním červené, zelené a modré barvy. Každá z těchto složek nabývá hodnot $\{0, 1, 2, \ldots, 255\}$, tedy zabírá přesně jeden bajt z celkové hodnoty pixelu. Java uchovává celočíselné hodnoty vždy ve velké endianitě, nejvýznamnější bajt je tedy

nevyužit (běžně je používán jako alfa kanál) a zbylé tři obsahují kanály (R, G, B), přesně v tomto pořadí. Modrá složka je tedy v nejméně významném bajtu. [7, 29]

Indexované obrázky se v analýze obrazu používají pro reprezentaci rozpoznaných objektů. V tomto případě se hodnota 0 používá jako reprezentace prázdného místa a hodnota 1 reprezentuje objekt. Ve speciálním případě jsou využity i hodnoty $\{2, 3, 4, \ldots, n\}$ pro reprezentaci konkrétních objektů (v takovém případě se hodnota pixelu 1 v obrázku nenachází).

4.4 Dlaždicové mapy

Výsledkem procesu rozpoznání mapy je vrstva dlaždic, nesoucí informace o prvcích na mapě. K tomuto účelu se v knihovně nachází rozhraní `buc0044.libbp.Tile` reprezentující typ dlaždice na mapě, `buc0044.libbp.TileSet` reprezentující sadu typů dlaždic a rozhraní `buc0044.libbp.TileData`, které představuje obdélníkovou vrstvu dlaždic mapy.

```
package buc0044.libbp;

public interface Tile
{
  public String getName();
  public String getDescription();
  public int getCode();
  public TileSet<? extends Tile> getTileSet();
}
```

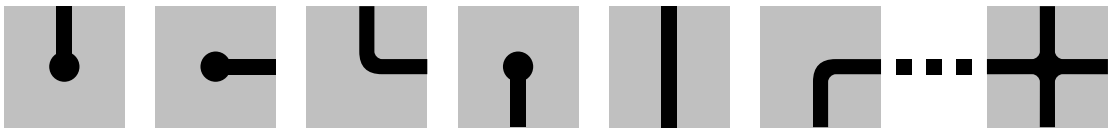
Výpis 3: Deklarace rozhraní `Tile`

Protože je knihovna navržena obecně, typ použitých dlaždic záleží na druhu rozpoznávané mapy. Každý typ dlaždice má vedle jména a popisu i jedinečný číselný kód (v rámci jedné sady), přičemž kód 0 je zpravidla prázdná dlaždice. Pro zjednodušení implementace rozhraní `TileData` je zde předpřipravená třída `AbstractTileData` s pomocnými metodami.

Samotné rozpoznání řeší rozhraní `Analyzer` s jedinou metodou `analyze()`, která přijímá obrázek a vrací příslušná `TileData`. Při zpracování je zaměřen obsah s mapou a následně je vyrovnán, což se děje v rámci metody `cutContent()`. Následně je zpracována každá dlaždice mapy zvlášť jako samostatný podobrázek. Zpracování dlaždice je specifické pro každou sadu dlaždic, případně se mohou lišit i další kroky průběhu celé analýzy. Během rozpoznávání může být vyvolána výjimka typu `AnalyzeException`.

4.4.1 Cesty

Součástí knihovny je typ dlaždic pro cesty, kdy cesta prochází vždy středem hrany. Tyto třídy jsou umístěny v samostatném balíčku `buc0044.libbp.roads`. Lze je použít k vytvoření uzavřeného okruhu (závodní dráha) nebo libovolné spleti cest s neomezeným počtem konců (bludiště). Vizuální ukázka těchto dlaždic je vidět na obrázku 6.



Obrázek 6: Ukázka dlaždic typu cesta (`buc0044.libbp.roads.RoadTile`)

Každý typ dlaždice musí být vždy reprezentován jedním objektem. Z tohoto důvodu je pro implementaci rozhraní `Tile` výhodné využít výčtový typ (**`enum`**). [29] Celá sada typů dlaždic je sdružena typem `RoadTileSet`, který implementuje rozhraní `TileSet`. Každá třída implementující toto rozhraní musí využívat návrhový vzor *singleton*. [30]

Pro rozpoznání jednotlivých dlaždic slouží metoda `analyzeTile()`, která je součástí třídy `buc0044.libbp.roads.RoadAnalyzer`. Metoda přijímá naprahovaný obrázek a hranice vyhodnocované dlaždice. Na základě hodnot na hranách pak zvolí a vrátí typ dlaždice.

4.5 Třída `LibBP`

Nejvýznamnější třídou celé knihovny je `LibBP`, která shlukuje různé statické metody. Obecně je lze rozdělit na metody zpracovávající pokročilé operace analýzy obrazu a pomocné obslužné metody.

4.5.1 Serializace dat z `TileData`

Pro práci s třídou `TileData` je zde umístěna metoda `loadTileData()`, zajišťující načtení z uložených dat, a její protějšek `saveTileData()`. Data dlaždic jsou v tomto případě serializována do formátu *JSON*, takže výsledný řetězec obsahuje pouze tisknutelné znaky v kódování *ASCII*. Tento formát byl zvolen pro svou jednoduchost, a protože jej lze snadno zpracovat ve většině programovacích jazyků a nástrojů. Oproti formátu *XML*, který se v této situaci také nabízel, má *JSON* při uložení dat jednodušší syntaxi. Obnovení dat zpátky do objektu je zajištěno pomocí reflexe jazyka Java, jelikož v serializovaných datech je rovněž obsažen i název původní třídy. Uložená data rovněž splňují podmínky formátu *JSON-LD*, tedy obsahují značku `"@context"`. [6, 31]

4.5.2 Prahování a vyplňování oblastí

Pro potřeby prahování obrazu je v knihovně obsažena metoda `threshold()`, případně je zde i pokročilejší `adaptiveThreshold()`. Obě metody přijímají i vracejí objekt typu `BitmapBuffer`, což umožňuje řetězit více akcí za sebe. Úpravy jsou prováděny přímo v předaném objektu, jehož instance je i vrácena.

U běžného prahování je hodnota prahu předána v druhém parametru metody. Pokud je hodnota pixelu nad prahem, je nahrazen hodnotou 1, jinak je nahrazen hodnotou 0. Fungování metody znázorňuje vzorec 8, kde *input* a *output* jsou vstupní parametry.

$$output(x,y) = \begin{cases} 1 & input(x,y) > hodnota\_prahu \\ 0 & jinak \end{cases} \tag{8}$$

Adaptivní prahování využívá dvě metody k vypočítání prahu: Průměr a Gaussovu funkci. Pro výpočet jádra Gaussovy funkce je v knihovně metoda `gaussianKernel()`, která vrací matici s normalizovaným jádrem. Matice musí obsahovat lichý počet řádků i sloupců, a zároveň musí být čtvercová. [32]

Hodnotu $sigma$, která je pro výpočet nezbytná, lze předat v parametru metody. Nutnou podmínkou je, aby hodnota $sigma$ byla kladná. V jiném případě je hodnota $sigma$ vypočítána podle vzorce 10, kde proměnná $kernel\_size$ představuje rozměr matice.

$$radius = \frac{kernel\_size - 1}{2} \tag{9}$$

$$sigma = 0.3 \cdot (radius - 1) + 0.8 \tag{10}$$

Výpočet prvku matice jádra pak probíhá podle vzorce 11.

$$gaussian_{xy} = \mathrm{e}^{\left(\frac{-(x-radius)^2}{2sigma^2} + \frac{-(y-radius)^2}{2sigma^2}\right)} \tag{11}$$

Výsledkem prahování je obraz obsahující pouze hodnoty 0 a 1. Po odstranění nežádoucích výsledků (chyby při prahování) je potřeba jednotlivé oblasti očíslovat. K tomuto účelu se používá metod `seed()` a `seedAll()`.

Metoda `seed()` přijímá souřadnice startu a hodnotu, kterou se má vybraná oblast vyplnit. Vyplňování probíhá od startovního bodu všemi směry, kde mají pixely stejnou hodnotu (semínková metoda). Pro automatické očíslování všech oblastí (číslováno je od hodnoty 2) lze použít metodu `seedAll()`. [33]

4.5.3 Momenty a rozpoznání entit

Jedna z klíčových metod při analýze obrazu je výpočet momentů. Ten zajišťuje metoda `moment()`, která přijímá parametry momentu p a q. Pro výpočet momentu m_{01} bude volání metody vypadat následovně:

```
double m01 = moment(buffer, 0, 1, value);
```

Výpis 4: Volání metody pro výpočet momentu

Proměnná *buffer* ve výpisu 4 představuje vstupní obrázek a parametr *value* číslo objektu v obrázku, pro který má být moment spočítán. Metoda `centerOfMass()` pak využívá výpočtu momentů k nalezení těžiště vybrané oblasti v obrázku.

Nejkomplexnější metodou je pak `findEntities()`, která vrátí pole objektů typu `Entity`. Vstupem metody je pouze objekt třídy `BitmapBuffer` obsahující předem naprahovaný obraz. Každý prvek ve vráceném poli představuje jeden nalezený objekt, včetně jeho vlastností, jak je vidět ve výpisu 5. [34]

```
static public class Entity
{
  public int value;           // hodnota pixelu
  public Bounds bounds;       // hranice obdélníkové oblasti
  public int edges;           // počet hran pixelů na hranici oblasti
  public int perimeter;       // počet pixelů na hranici oblasti
  public int area;            // počet pixelů v oblasti (obsah)
  public double moment00;
  public double moment01;
  public double moment10;
  public double moment02;
  public double moment20;
  public Point centerOfMass;  // těžiště oblasti
}
```

Výpis 5: Vnořená třída pro uchování informací entity

4.6 Knihovna OpenCV

V rámci vylepšení výkonu byla do projektu experimentálně zahrnuta knihovna pro zpracování obrazu OpenCV. Ta je naimplementována v jazyce C a obsahuje propojení do jazyků C++, Java, Python, MATLAB a dalších. Je plně svobodná (pod licencí BSD), a lze ji zkompilovat a použít na většině dostupných desktopových i mobilních platformách. Obsahuje mnoho modulů pro základní i pokročilou práci s obrazem. [35]

V rámci analýzy obrazu je v OpenCV několik funkcí, jejichž ekvivalent je součástí této práce. Protože OpenCV má dlouhodobou podporu ze strany vývojářů a je vysoce optimalizovaná, byly části OpenCV v rámci testů použity místo původních metod. Výsledky testů jsou zobrazeny a rozebrány v části **Testy výkonu** (viz 5.4).

4.7 Platformová závislost

Přestože je knihovna navržena univerzálně, obsahuje i části, které zjednodušují použití na dané platformě.

4.7.1 Implementace pro Android

Pro snadnější použití knihovny na operačním systému Android byla do projektu začleněna třída `buc0044.bp.BitmapWrapper`. Ke zpracování bitmap využívá Android třídu `android.graphics.Bitmap`. Instanci této třídy lze získat mnoha způsoby, nejčastěji nahráním ze zdrojů aplikace (resources), ze souboru nebo z fotoaparátu, apod. Objekt třídy `BitmapWrapper` pak umožňuje tento typ zapouzdřit a přistupovat k pixelům bitmapy přes jednotné rozhraní knihovny (Pro účely Java SE).

4.7.2 Implementace pro desktop

Pro účely použití knihovny na platformě *Java SE* vznikl balíček propojující ji se systémem komponent *Swing*. Při implementaci lze tedy použít klasické postupy pro práci s obrázky, jako je třída `BufferedImage`. Zapouzdření pro rozhraní `BitmapData` pak zajistí třída `BufferedImageWrapper`. [36]

Protože většina testů probíhala na desktopu, obsahuje tato implementace i specializovaný balíček `buc0044.libbp.testing`. Jeho součástí jsou metody pro rychlé nahrávání a ukládání obrázků ze třídy `BitmapBuffer`, včetně možnosti jejich zobrazení.

5 Testování

Již během vývoje knihovny probíhaly testy jednotlivých součástí, zda metody pracují správně. Po dokončení všech částí nutných k rozpoznání mapy na obrázku proběhla řada testů pro nalezení mezních případů. Za tímto účelem vznikla sada fotografií nakreslených map, zachycených pod různými úhly a v různém osvětlení. Součástí této sady jsou i fotografie se špatně zachycenou mapou, které mají za úkol otestovat opravné mechanismy knihovny.

5.1 Testovací zařízení

Knihovna byla testována pod oběma platformami, pro které je určena. Desktopová verze byla spouštěna na stolním počítači s operačním systémem *Windows 8.1 x64*, jehož hardwarovou specifikaci zachycuje tabulka 2. Mobilní verze byla testována na všech zařízeních, jejichž specifikace je v tabulce 3. [37, 38, 39]

Mateřská deska	MSI 970A-G43
Procesor	AMD FX-8350 VISHERA
Operační paměti	Kingston HyperX Fury 16GB (Kit 2x8GB) 1600MHz DDR3 CL10
Grafická karta	MSI NVIDIA GTX 750 Ti (N750Ti-2GD5/OC)
Pevný disk	Kingston HyperX 3K 120GB

Tabulka 2: Hardwarová specifikace počítače

	Nexus 5	**Lenovo A606**	**Lenovo Yoga 2**
Display	4.95″	5″	10.1″
Rozlišení	1920x1080	854x480	1920x1080
Fotoaparát	8Mpx (1.3Mpx)	8Mpx (2Mpx)	8Mpx (1.6Mpx)
Procesor	Qualcomm Snapdragon™ 800	MediaTek MT6290	Intel® Atom™ Z3745
Frekvence jádra	4x 2.26GHz	4x 1.3GHz	4x 1.86GHz
Grafická karta	Adreno 330, 450MHz	ARM Mali-400 MP2	integrovaná
Operační paměť	2GB	1GB	2GB

Tabulka 3: Hardwarová specifikace mobilních zařízení

Měření probíhala postupně na všech zmíněných zařízeních. Předpokladem je, že poměr časů by měl být u všech zařízení používajících OS Android obdobný, bez ohledu na jejich technickou specifikaci. Výsledky měření jsou uvedeny níže v části 5.4.

5.2 Podpůrná aplikace WannaPic

K usnadnění pořizování rozsáhlých kolekcí fotografií vznikla aplikace *WannaPic*. Aplikace je rozdělena do dvou částí: klient a server. Klientská část je určena pro zařízení s *OS Android*, zatímco serverová část je navržena pro *Java SE*.

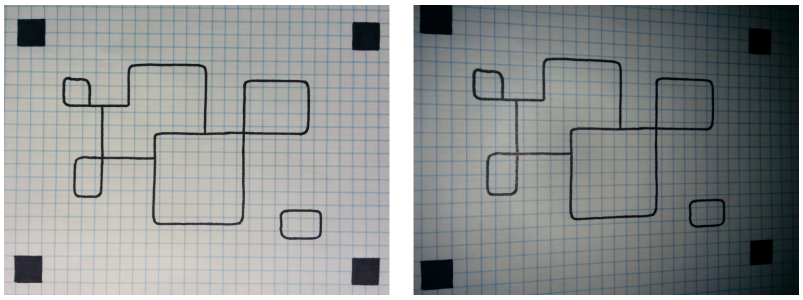
Klient je jednoduchá aplikace, která se automaticky připojuje k serverové části aplikace. Spojení probíhá nad protokolem TCP. Serverová část obsahuje řídící prvky, které umožňují přímo ukládat snímky získané z fotoaparátu.

5.3 Testy přesnosti snímání

Součástí příloh je sada nafocených snímků map za různých světelných podmínek a poloh fotoaparátu. Ty byly využity k vyhodnocení schopnosti knihovny rozpoznat nakreslenou mapu za nepříliš ideálních podmínek. Zřetel byl brán především na:

- Úhel fotoaparátu, tedy naklonění vůči vodorovné a svislé ose.
- Rotace fotoaparátu v rámci roviny.
- Přiblížení snímku, zda je na něm pouze mapa nebo i část okolí.
- Světelné podmínky (ideální, šero, blesk).
- Barva podkladu pod papírem nebo v jeho okolí.

Z výsledků těchto testů vyplynuly hraniční případy, při kterých analýza začne částečně nebo úplně selhávat. Zaměření mapy proběhne správně bez ohledu na rotaci a s náklonem do 15 stupňů, přičemž schopnosti rozpoznání se horšily s narůstajícím náklonem. Vzdálenost fotoaparátu od mapy neměla na schopnost zaměření a rozpoznání žádný znatelný vliv. V případě zhoršených světelných podmínek (focení za šera) selhalo zaměření i rozpoznání ve zhruba 40% případů. Při použití blesku bylo selhání prakticky 100%, což bylo způsobeno nerovnoměrným rozložením světla.



Obrázek 7: Porovnání fotografie mapy bez a s použitím blesku

Problém focení mapy za použití blesku je zanedbatelný z několika důvodů. Již z pořízeného snímku je znatelné, že je nekvalitní i na pohled uživatele. Odraz světla od bílého papíru způsobí, že se jas obrázku rozloží radiálně od přesvětleného středu směrem vně. Optimální je fotit vždy bez blesku, a to i za šera.

5.4 Testy výkonu

Vedle schopnosti správně rozpoznat mapu je důležitým rysem knihovny i rychlost zpracování. Během testů se ukázalo, že rozlišení obrázku neovlivňuje přesnost rozpoznání, pouze navyšuje čas nutný k jeho zpracování. Všechny testy proto probíhaly nad obrázky s rozměrem delší hrany 800 pixelů.

V rámci výkonnostních testů byl na všech jmenovaných zařízeních změřen průměrný čas potřebný k analýze mapy. Na nejméně výkonném zařízení se čas pohyboval v řádu jednotek sekund, naopak u zařízení s vysokým výkonem byl výsledek v řádech setin sekundy.

Jedna z možností, jak optimalizovat výpočet analýzy obrázku je již zmíněné nasazení knihovny *OpenCV* ve verzi $3.x$, která poskytuje některé používané funkce. Klíčovým krokem je prahování obrazu, které bylo zvoleno k porovnání ve výkonnostních testech. Měření probíhalo na všech zařízeních, popsaných v části 5.1. Čas byl měřen pro 100 volání prahovací funkce, a to pro prahování s pevnou hodnotou prahu (**fixed**) a adaptivní prahování s využitím Gaussovy funkce (**adaptive**). Pro *OpenCV* byly zohledněny dva způsoby volání. V prvním případě byla data obrázku předávána postupně (**OpenCV**$_1$), v druhém případě byl celý obrázek předán najednou (**OpenCV**$_2$). Výsledné časy jsou uváděny v jednotkách milisekund a byly zaokrouhleny na 3 desetinná místa. Pokud doba běhu překročila 10 sekund, je čas uveden v sekundách.

	Knihovna		**OpenCV**$_1$		**OpenCV**$_2$	
	celkem	průměr	celkem	průměr	celkem	průměr
Fixed	29.113	0.291	$\sim 14s$	138.647	66.930	0.669
Adaptive	951.111	9.511	$\sim 14s$	138.024	96.856	0.969

Tabulka 4: Časy výpočtu prahování na Nexus 5

	Knihovna		**OpenCV**$_1$		**OpenCV**$_2$	
	celkem	průměr	celkem	průměr	celkem	průměr
Fixed	2 530.224	25.302	$\sim 255s$	2 547.616	4 443.496	44.435
Adaptive	$\sim 99s$	987.676	$\sim 264s$	2 640.271	6 232.198	62.322

Tabulka 5: Časy výpočtu prahování na Lenovo A606

	Knihovna		**OpenCV**$_1$		**OpenCV**$_2$	
	celkem	průměr	celkem	průměr	celkem	průměr
Fixed	791.609	7.916	$\sim 81s$	808.371	1 965.840	19.658
Adaptive	$\sim 20s$	203.614	$\sim 81s$	812.041	2 402.251	24.023

Tabulka 6: Časy výpočtu prahování na Lenovo Yoga 2

	Knihovna		**OpenCV**$_1$		**OpenCV**$_2$	
	celkem	průměr	celkem	průměr	celkem	průměr
Fixed	48.208	0.482	5 620.549	56.205	143.720	1.437
Adaptive	1 957.515	19.575	5 730.879	57.309	229.029	2.290

Tabulka 7: Časy výpočtu prahování na platformě Java SE

Z výsledků výkonnostních testů je patrné, že u *OpenCV* hraje velkou roli režie předávání dat, proto budou použity hodnoty časů **OpenCV**$_2$. V případě pevné hodnoty prahu je čas knihovny a *OpenCV* srovnatelný a kolísá pouze v závislosti na použitém zařízení. Naopak u adaptivního prahování je *OpenCV* několikanásobně rychlejší, u jednoho ze zařízení až desetinásobně.

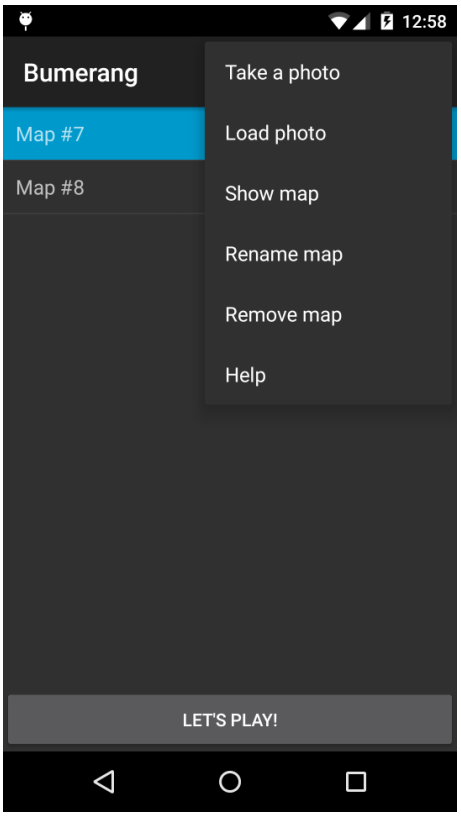
Využití *OpenCV* by mohlo vylepšit čas potřebný k rozpoznání mapy, ale jeho použití napříč platformami by bylo komplikované. Pro *Android* je potřeba nahrát binární verzi *OpenCV* nezávisle přes manažer, s ohledem na instrukční sadu procesoru. Zautomatizovat proces sestavení aplikace by bylo složité a není to ani účel této práce.

6 Ukázková aplikace

Jako aplikace demonstrující funkčnost knihovny byla zvolena jednoduchá hra *Bumerang*. Ta využívá dlaždice cest (balíček `buc0044.libbp.roads`) jako předlohu pro 3D bludiště, které lze procházet z pohledu první osoby. Hra má vlastní, ručně kreslenou grafiku. Účel hry je čistě rekreační, bludiště nemá žádný daný cíl a začíná se na náhodně vybraném místě. Aplikace je napsána pro Android, minimálně ve verzi 4.0.

6.1 Návrh

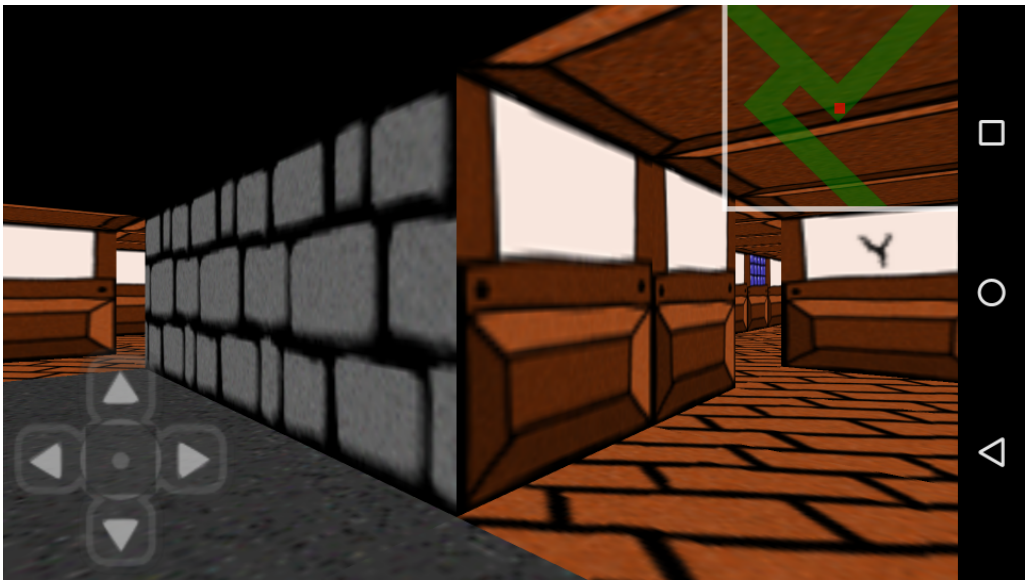
Aplikace je rozdělena do dvou obrazovek: startovní a hlavní. Hlavní komponentou na startovní obrazovce, která je orientována na výšku (*portrait*), je seznam uložených map, ze kterých může uživatel vybírat. Pod ní se nachází tlačítko **Let's play!**, umožňující zahájit hru. Všechny ostatní akce jsou řešeny nabídkou, která umožňuje přidávat a editovat mapy, včetně zobrazení jejich náhledu.



Obrázek 8: Startovní aktivita ukázkové aplikace

Akce **Take a photo** spustí aktivitu fotoaparátu s možností vyfotit mapu. Pokud byla mapa vyfocena mimo aplikaci a je uložena v galerii telefonu, lze ji nahrát pomocí akce **Load photo**.

Hlavní obrazovka je umístěna vždy na šířku (orientace *landscape*), přičemž celou plochu zabírá pohled hráče do bludiště. Vlevo dole se nachází ovládací kříž, v protilehlém rohu pak mini-mapa. [15]



Obrázek 9: Hlavní aktivita ukázkové aplikace

6.2 Implementace

Hra je napsána čistě pomocí tříd, které jsou součástí platformy Android. Nevyužívá žádné další pomocné frameworky nebo herní enginy.

6.2.1 Startovní aktivita

Startovní aktivita je postavena na základních GUI komponentách Androidu. K výběru mapy se používá komponenta `android.widget.ListView`, která je dále rozšířena o možnost označení položky, tedy námi zvolené mapy. Pro interakci s uživatelem se používá třída `android.app.AlertDialog`, krátké textové zprávy se zobrazují pomocí třídy `android.widget.Toast`. [15]

Rozpoznání dlaždic mapy probíhá ihned po nahrání (vyfocení) obrázku a mapy jsou poté uchovávány čistě ve formě dat. K uložení map je využíváno úložiště *SQLite*, které Android v základu podporuje. Data mapy jsou serializována do formátu JSON, a ten je následně jako text uložen spolu s popiskem mapy do databáze. [15, 42]

6.2.2 Hlavní aktivita

Obrazovka hlavní aktivity obsahuje dvě překrývající se komponenty. Vespodu se nachází komponenta `android.opengl.GLSurfaceView`, která slouží jako plátno pro vykres-

lování celé hry. Nad ní je umístěn `android.widget.ProgressBar`, který je zobrazen po dobu nahrávání potřebných částí hry při přechodu mezi aktivitami.

K vykreslování herní grafiky je použita technologie *OpenGL ES 2.x*. Obecně je práce s grafikou rozdělena na 2D a 3D část. O dvourozměrné vykreslování se stará pomocná třída `StaticDraw`, která umožňuje vykreslit část textury na libovolnou část obrazovky. Toho se využívá pro vykreslení ovládacího prvku. Jelikož je pro každé vykreslení prvku skrze třídu `StaticDraw` nutná inicializace a volání mnoha OpenGL funkcí, nehodí se toto řešení na vykreslování velkého množství objektů. Pro tento účel je zde třída `Model`, která udržuje předem připravenou sestavu polygonů. Tato třída je použita pro vykreslení mini-mapy i samotné herní scény. [43]

Mini-mapa je řešena jako matice čtvercových polygonů, na které je nanesena příslušná textura s průhledností. Oříznutí do rohu je řešeno pomocí funkce `glScissor()`. [43] Pro jednodušší orientaci se mini-mapa otáčí spolu s pohledem hráče.

Pro vykreslení bludiště je každá dlaždice cesty nahrazena krychlí, jejíž stěny jsou viditelné zevnitř. Pokud je mezi dvěma dlaždicemi cesta, stěna krychle se nezobrazí. Textury stěn jsou vybírány náhodně podle předem určených pravidel. Mapa se rozdělí na obdélníkové oblasti, které se mohou navzájem překrývat, přičemž jejich průnik je řešen jako exkluzivní disjunkce (operace **xor**). [41] Pro jednotlivá prostředí (oblasti) je zvolena jiná textura stropu, podlahy a zdí. Textura zdi může být v několika variantách. Vedle základního vzhledu je součástí i několik dekorativních zdí, které jsou vybírány náhodně, s nižší pravděpodobností výskytu. Díky tomuto řešení nepůsobí bludiště jednotvárně, a lze se v něm lépe orientovat. V rámci jedné mapy je kombinace oblastí i zdí vždy jedinečná, tzn. při opakovaném hraní jedné mapy vypadá bludiště vždy stejně.

7 Závěr

Výsledkem této práce je funkční knihovna schopná rozpoznat vlastnoručně nakreslenou herní mapu, vytvořenou tak, jak je uvedeno v úvodní části této práce. Pomocí vzniklé knihovny lze vytvářet aplikace pro *OS Andorid* a *Java SE*, bez nutnosti použití knihoven třetích stran nebo jakékoli dodatečné technologie.

V rámci návrhové fáze byly prozkoumány metody potřebné ke zpracování obrazu, z nichž byly následně vybrány části, použitelné pro rozpoznání mapy. Teoretické podklady analýzy byly shrnuty ve vlastní kapitole.

Jednotlivé kroky implementace byly zdokumentovány v kódu a podrobněji popsány v textu práce.

Celkové řešení bylo řádně otestováno a byly odladěny všechny nalezené chyby. Z provedených testů vyplynulo, že schopnost rozpoznání mapy je dostačující pro běžné použití. Doba nutná k celkovému rozpoznání obsahu mapy je přijatelná na všech testovaných zařízeních.

Pro demonstraci praktického použití knihovny vznikla ukázková aplikace *Bumerang*, kterou lze nalézt na přiloženém CD. Aplikace využívá všechny části knihovny a umožňuje uživateli projít se po vytvořené mapě z pohledu první osoby ve 3D bludišti.

8 Reference

[1] *Building A Dungeon | Simple DND* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://simplednd.wordpress.com/dungeon-mastering/building-a-dungeon/

[2] *TMX Map Format · bjorn/tiled Wiki · GitHub* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://github.com/bjorn/tiled/wiki/TMX-Map-Format

[3] *tiles - Difference between „staggered" isometric and „normal" isometric tilemaps? - Game Development Stack Exchange* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://gamedev.stackexchange.com/questions/49847/difference-between-staggered-isometric-and-normal-isometric-tilemaps

[4] *ShoeBox* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://renderhjs.net/shoebox/

[5] *Campaign Cartographer 3 (CC3) RPG and fantasy map making software* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://secure.profantasy.com/products/cc3.asp

[6] *JSON* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://json.org/json-cz.html

[7] GONZALEZ, Rafael C a Richard E WOODS. Digital image processing. 3rd ed. Upper Saddle River: Pearson, c2008, xxii, 954 s. ISBN 01-316-8728-X.

[8] *Morfologické operace* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://midas.uamt.feec.vutbr.cz/ZVS/Exercise10/content_cz.php

[9] *Základní morfologické pojmy* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://e-learning.tul.cz/cgi-bin/elearning/elearning.fcgi?ID_tema=67&ID_obsah=1164&stranka=publ_tema&akce=polozka_vstup

[10] *Matematická morfologie* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://midas.uamt.feec.vutbr.cz/POV/lectures-pdf/08_Matematicka_morfologie.pdf

[11] *Praktické využití metod digitálního zpracování obrazu* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://soc.nidv.cz/data/2007/01-2.pdf

[12] *Matematická morfologie* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://blade1.ft.tul.cz/elearning/Media/File/5/123/P7.pdf

[13] *Segmentace obrazu* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://e-learning.tul.cz/cgi-bin/elearning/elearning.fcgi?ID_tema=67&ID_obsah=1202&stranka=publ_tema&akce=polozka_vstup

[14] *Zpracování obrazu a jeho statistická analýza* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://e-learning.tul.cz/cgi-bin/elearning/elearning.fcgi?stranka=publ_tema&akce=tisk&ID_tema=67

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

[16] PRATA, Stephen. *Mistrovství v C++: 3. aktualizované vydání.* Brno: Computer Press, 2007. ISBN 978-80-251-1749-1.

[17] *Android NDK* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://developer.android.com/tools/sdk/ndk/index.html

[18] *groovy/groovy-android-gradle-plugin · GitHub* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://github.com/groovy/groovy-android-gradle-plugin

[19] *Scala on Android* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://macroid.github.io/ScalaOnAndroid.html

[20] *Corona SDK | Corona Labs* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://coronalabs.com/products/corona-sdk/

[21] *python-for-android* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://code.google.com/p/python-for-android/

[22] *Android Studio: An IDE built for Android* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://android-developers.blogspot.in/2013/05/android-studio-ide-built-for-android.html

[23] *IntelliJ IDEA* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://www.jetbrains.com/idea/

[24] *ADT Plugin Release Notes* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://developer.android.com/tools/sdk/eclipse-adt.html

[25] *Android Studio Overview* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://developer.android.com/tools/studio/index.html

[26] *Android 5.0 APIs* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://developer.android.com/about/versions/android-5.0.html

[27] *Java SE 7 Features and Enhancements* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html

[28] *Lambda Expressions* [online]. 2015 [cit. 2015-05-03]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

[29] HEROUT, Pavel. Učebnice jazyka Java. 5., rozš. vyd. České Budějovice: Kopp, 2010, 386 s. ISBN 978-80-7232-398-2.

[30] PECINOVSKÝ, Rudolf. *Návrhové vzory*. Vyd. 1. Brno: Computer Press, 2007, 527 s. ISBN 978-80-251-1582-4.

[31] *JSON-LD - JSON for Linking Data* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://json-ld.org/

[32] STRAKA, Stanislav. *Segmentace obrazu* [online]. 2009 [cit. 2015-05-03]. Dostupné z: http://is.muni.cz/th/72784/fi_m/dp.pdf. Diplomová práce. Masarykova univerzita. Vedoucí práce Mgr. Radka Pospíšilová.

[33] *Semínkové vyplňování* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://www.is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=6352

[34] *Momenty rozdělení* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://www-troja.fjfi.cvut.cz/ limpouch/sigdat/pravdh/node10.html

[35] *ABOUT | OpenCV* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://opencv.org/about.html

[36] *Buffered Images (Java Foundation Classes)* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://docstore.mik.ua/orelly/java-ent/jfc/ch04_10.htm

[37] *Nexus 5* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://www.google.com/nexus/5/

[38] *Lenovo A606* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://shop.lenovo.com/cz/cs/smartphones/a-series/a606/#tab-technická_specifikace

[39] *Lenovo Yoga Tablet 2 (10")* [online]. 2015 [cit. 2015-05-03]. Dostupné z: http://shop.lenovo.com/cz/cs/tablets/lenovo/yoga-tablet-series/yoga-tablet-2-10/#tab-technická_specifikace

[40] SHREINER, Dave. *OpenGL: průvodce programátora.* Vyd. 1. Překlad Jiří Fadrný. Brno: Computer Press, 2006, 679 s. DTP. ISBN 80-251-1275-6.

[41] DIVIŠ, Zdeněk, Zdeňka CHMELÍKOVÁ a Jaroslav ZDRÁLEK. *Logické obvody.* 2. vyd. VŠB - Technická univerzita Ostrava, 2008, 152 s. ISBN 978-80-248-1724-8.

[42] OWENS, Grant Allen and Mike. *The definitive guide to SQLite.* 2nd. ed. New York: Apress, 2010, 152 s. ISBN 978-143-0232-254.

[43] BROTHALER, Kevin. *OpenGL ES 2 for Android.* Raleigh: The Pragmatic Programmers, 2013. ISBN 978-193-7785-345.

A CD

Součástí této bakalářské práce je CD, obsahující výslednou knihovnu včetně zdrojových kódů ve složce *workspace*. Ta obsahuje podsložky s jednotlivými projekty pro *Eclipse* s pluginem ADT.

appcompat_v7 automaticky generovaný projekt pro udržení kompatibility

BP ukázková aplikace Bumerang pro OS Android

libBP samotná knihovna

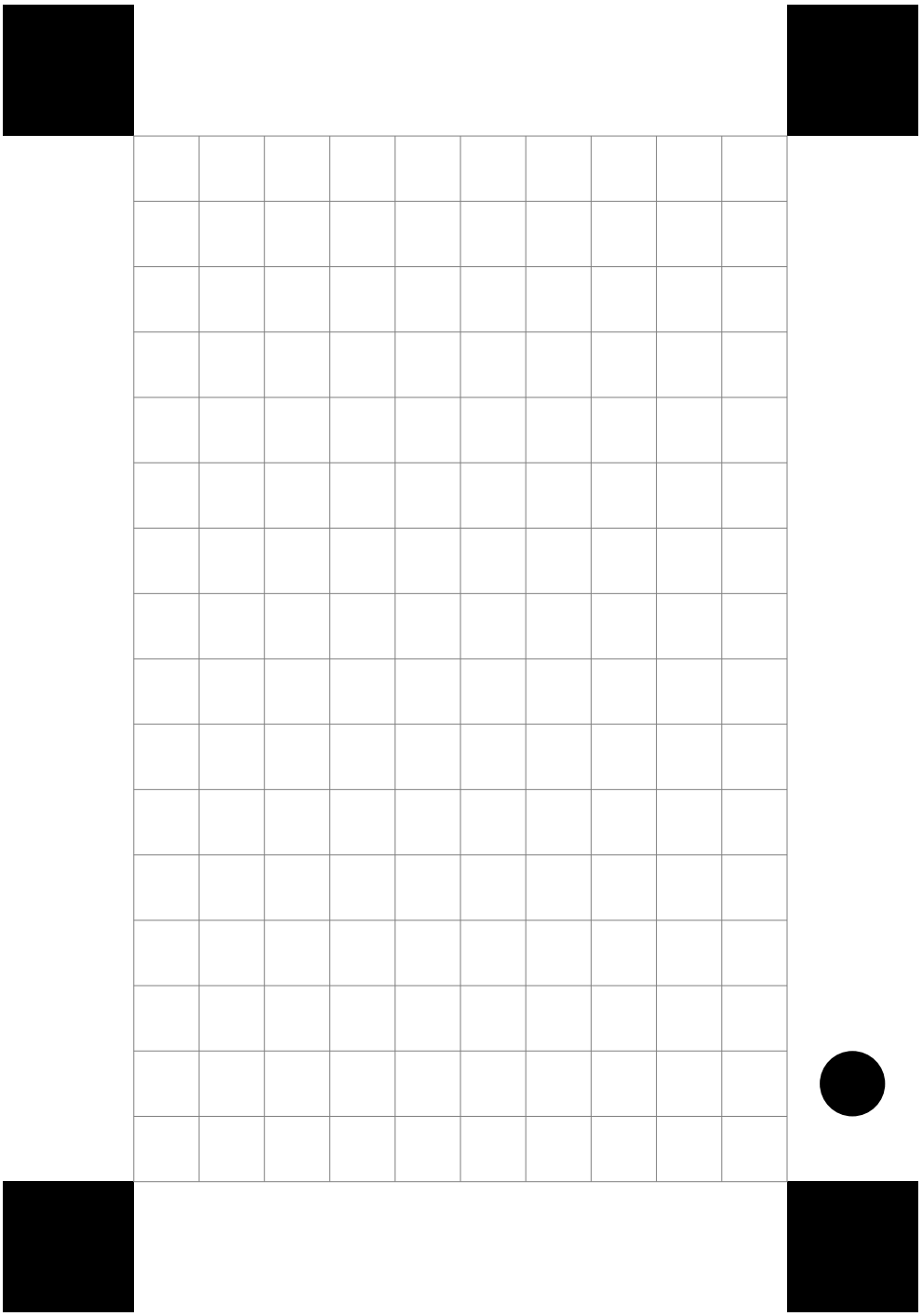
Testing testovací úlohy pro knihovnu

WannaPicClient klientská část podpůrné aplikace WannaPic pro OS Android

WannaPicServer serverová část podpůrné aplikace WannaPic pro Java SE

V kořenové složce disku je i digitální verze tohoto textu včetně zdrojových kódů v jazyce LaTeX (složka *BUC0044_BP*). Přiloženy jsou i šablony pro kreslení map a všechny testovací fotografie.

B Šablony



2015 (c) David Buček