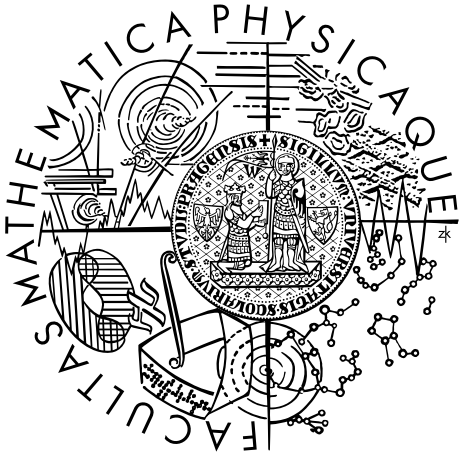
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Vojtěch Vašek

O-Hunter

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Petr Kučera, Ph.D.
Studijní program: Informatika
Studijní obor: IOI

Praha 2016

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 23. 5. 2016 Podpis autora

Název práce: O-Hunter

Autor: Vojtěch Vašek

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Petr Kučera, Ph.D., KTIML

Abstrakt: Práce se zabývá využitím technik digitálního zpracování obrazu v naučné hrě, která uživatele provede po zajímavých místech světa. Vymyšlená hra O-Hunter, která má dvě součásti – serverovou a klientskou, využívá porovnávání fotografií, rozhodování o denní době a filtry pro úpravu fotografií. Výklad je členěný do čtyř součástí. Nejdříve jsou popsána pravidla hry, poté je provedena analýza použitých metod, dále je vysvětlen způsob implementace a nakonec následuje popis způsobu ovládání součástí aplikace. Aplikace ukazuje, že některé použité techniky jsou pro tento účel velmi vhodné, jiné však v reálném nasazení mohou selhat. V závěru následuje shrnutí a několik návrhů k vylepšení. Hru O-Hunter lze využít k prozkoumávání významných míst nezvyklým způsobem, její návrh umožňuje případné nahrazení nebo vylepšení implementací použitých technik.

Klíčová slova: podobnost geocaching fotografie památky

Title: O-Hunter

Author: Vojtěch Vašek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Petr Kučera, Ph.D., KTIML

Abstract: This work looks into using digital image processing to create an educative game, which should guide its user through interesting places in the world. Invented game named O-Hunter, which consists of two parts – server and client, uses photo comparison, analyzes daytime and uses filters to modify images. In the first section, rules of the game are described, analysis of used methods follows. Implementation is described in third section and the next section covers how the program should be used. This program shows, that some of the selected approaches are very well suitable for this application and others can fail in real situations. Last section covers conclusions and suggestions for improvements. The game O-Hunter can be employed to discover attractive places in an unusual way, its design provides option to substitute or improve implementations of used techniques.

Keywords: similarity geocaching photo sights

Rád bych poděkoval všem, kteří mě při psaní této práce podporovali. Speciálně pak rodině a RNDr. Petrovi Kučerovi, Ph.D. za umožnění této práce, za vedení, rady, připomínky a nápady, které mi při psaní velmi pomohly.

Obsah

Úvod

Inspirací k tomuto tématu bakalářské práce byla hra z pomezí sportu a turistiky – geocaching.[1] V geocachingu je úkolem hledání ukrytých nádob tzv. kešek, o nichž je hráči známa pouze pozice GPS. Tyto nádoby obsahují různé předměty, nalezení kešky se potvrdí zápisem do deníku v kešce. Hlavní myšlenka vedoucí k vytvoření této práce byla použít namísto kešek reálné objekty.

Základní princip hry, jejíž implementace je náplní této práce, je následující: Na počátku „lovu“ si hráč vybere oblast, v níž je mu serverem náhodně vybraných několik konkrétních objektů. Objekty jsou specifikovány nejen svým umístěním (tedy GPS souřadnicemi), ale i nějakou jejich fotografií, kterou dříve vyfotil někdo jiný. Cílem hráče je navštívit co nejvíce takto vybraných objektů a vyfotit každý z nich na svém mobilním zařízení tak, aby byla výsledná fotografie co nejpodobnější vybrané předloze. Velmi důležitým je zde prvek náhodnosti, uživatel má jen částečnou kontrolu nad tím, které objekty jsou mu ve vybrané oblasti zvoleny k navštívení. To by mělo vést k tomu, že se hráč podívá i na místa, jež by jej jinak navštívit nenapadlo. Jedním z cílů této práce byla specifikace podrobných pravidel hry, tato jsou popsána dále v první kapitole.

Podobnou hrou je photogeocaching [2], který se snaží rovněž kombinovat geocaching s fotografováním. Jako kešky v něm figurují fotografie, které musí hráč na daném místě pořídit, každá keška má navíc specifikováno nějaké vlastní pravidlo dané jejím tvůrcem. Hráč si pak vybere kešku, přesune se na místo s pomocí souřadnic GPS a na místě pořídí fotografii tak, aby navíc splňovala pravidla tvůrce kešky. Pravidla jsou například fotka za určité denní nebo roční doby, na fotce musí být viditelný hráč, několik lidí, zvířat, věcí nebo má fotka obsahovat nějakou zajímavost. Od této hry má tato bakalářská práce několik odlišných záměrů. Jednak celý proces má probíhat automaticky. Již na počátku by mělo být dostupné velké množství objektů bez nutnosti jejich přidávání a proces ověřování má probíhat rychle a bez nutnosti zásahu ostatních hráčů. Dalším rozdílem je výběr objektů, hráč bude částečně omezován ve výběru a pořadí navštěvování objektů a konkrétní objekty ve zvolené oblasti budou voleny náhodně. Pravidla by také měla být jednotná pro každý objekt a jako objekty by měly být voleny spíše budovy a památky, než naprosto volitelné, pravidly definované obecné předměty.

Cíly této práce proto byly:

Vymyšlení pravidel hry zavedení herní terminologie, jak hra probíhá, co je přesně úkolem hráče, jak motivovat hráče k hledání dalších objektů, jak přesně nalezení objektu ověřovat apod.

Konkrétní volba technik z oblasti zpracování obrazu po definování pravidel hry bylo nutné zvolit patřičné metody k posouzení vyfoceného objektu, dále se ukázalo být vhodné mít k dispozici speciální filtr pro úpravu fotografií a kvůli nedostatečným informacím o fotografiích objektu bylo nutné zvolit metodu na jednoduché posouzení fotografie

[1] Je dostupná řada serverů, které nabízejí databáze kešek a vedení vlastních záznamů o jejich nalezení. Nejznámější je zřejmě `https://www.geocaching.com/`, nebo český `http://www.geocaching.cz/`.

[2] Domovské stránky projektu photogeocaching: `http://photogeocaching.com/`

Implementace hry volba platformy pro vývoj, rozdělení programu na součásti, nebylo jasné, kde získat fotografie a objekty samotné, jakými schopnostmi musí zařízení k hledání objektů disponovat apod.

Některé klíčové algoritmy byly převzaty z jiných aplikací a modifikovány tak, aby lépe vyhovovaly této práci. Výsledky série navržených testů mají ukázat, zdali je jejich použití opravdu vhodné nejen teoreticky, ale i v situacích, ve kterých je hra reálně využívá.

O pravidlech vymyšlené hry pojmenované **O-Hunter**[3], terminologii, herních objektech a řešení souvisejících problémů hovoří první kapitola.

Rozdělení aplikace na části server, knihovna a klient, přiřazení funkcí těmto částem a způsob komunikace mezi nimi, popis použitého zdroje objektů a jejich fotografií, volba a vysvětlení použitých metod digitálního zpracování obrazu jsou součástí kapitoly druhé.

Technický pohled do implementačních detailů, konkrétních řešení vývojových problémů a naznačení způsobu rozšíření či úprav nabízí kapitola třetí.

Kapitola čtvrtá pak pojednává z hlediska dokumentačního o způsobu konfigurace a možnostech správy technicky náročnější části aplikace a dále o používání a způsobu ovládání uživatelsky přívětivější části aplikace cílené na samotné hráče.

Na závěr jsou pak shrnuty výsledky práce a je navrženo několik způsobů rozšíření či vylepšení některých komponent aplikace.

[3] Jméno *O-Hunter* je inspirované prvotním nápadem hledání objektů a vychází ze zkrácené fráze „object hunting“.

1. Pravidla hry O-Hunter

Přihlášením, případně registrací do hry je uživateli umožněn přístup do jejího prostředí. Náplní této kapitoly je zavedení terminologie a vysvětlení průběhu celé hry.

1.1 Terminologie

Vysvětlení základních pojmů, se kterými se v této hře a aplikaci pracuje.

1.1.1 Cíl

Základním herním objektem je *cíl*. **Cíl** zastupuje reálný objekt, nejčastěji památku, a má vždy unikátní identifikátor ID, GPS pozici, alespoň jednu fotografii, stav a *aktivní zónu*. **Aktivní zóna** je kruhová oblast kolem cíle, která má pevně daný poloměr. Cíl může dále obsahovat název, adresu, odkaz na webové stránky, ikonu zastupující jeho typ a odkaz na oficiální webovou stránku Google pro tento objekt.

Vzdáleností od cíle se rozumí vzdálenost definová na geoidu WGS84 mezi GPS pozicí uživatele a GPS pozicí cíle bez ohledu na nadmořskou výšku.

1.1.2 Stavy cíle

Iniciální stav každého cíle je **nepřístupný**. Takový cíl se může stát *přístupným*, nebo *zamítnutým*. Úkolem hráče je cíl *splnit*. Cíl je **splněn** ve chvíli, kdy hráč cíl **navštíví**, tj. vstoupí do jeho *aktivní zóny*, a cíl *vyfotí*. **Vyfocením** cíle se rozumí vyfocení fotografie, která má být co nejpodobnější k **referenční**, což je jedna z fotografií cíle.

Za *splnění* dostane hráč body, ty je pak možné použít jako platidlo v jiných herních situacích.

Cíl se může stát **přístupným** buď automaticky po *vyfocení* některého cíle, případně pokud si hráč zpřístupnění zaplatí.

Nepřístupný nebo *přístupný* cíl může hráč **zamítnout**, pak takový cíl — stejně jako *splněný* cíl — nebude již nikdy hráči nabídnut a nebude ho ani možné *splnit*.

Přístupný cíl je také možné *přijmout*. Pokud je cíl ve stavu **přijmutý**, je u něj kontrolováno *navštívení*. Pokud hráč *přijmutý* cíl *navštíví*, stává se cíl **fotogenickým**. V takovém stavu je zpřístupněno jeho vyfocení.

Po vyfocení se automaticky stává **zamknutým**, *zamknutý* cíl je poté možné vyhodnotit a tím cíl *splnit*.

Možné stavy cíle a přechody mezi nimi jsou zachyceny v diagramu 1.2.

Jeden ze stavů není na klientské části aplikace hráči dostupný, jde o stav **blokovaný**, takový cíl byl označen za nevhodný na serveru. Takové rozhodnutí činí správce serveru a stav cíle manuálně nastavuje.

Přirozeně **zpřístupnění** je změna stavu cíle na *přístupný* apod. pro ostatní stavy.

Hra navíc pracuje s tzv. **historií cílů** (zkráceně **historií**), což je množina *zamknutých* a všech *splněných* cílů.

1.1.3 Lov

Hra probíhá pouze po dobu částí, které jsou časově omezené, každá jedna taková část se nazývá **lov**. *Lov* je část hry, kdy se hráč aktivně snaží navštěvovat a fotit, tedy *plnit* nové cíle, které mu po dobu lovu aplikace nabízí.

1.2 Jak hrát

Hra začíná zahájením nového lovu. Poté se hráč snaží *plnit* co nejvíce cílů až do ukončení lovu. Tato sekce pravidla celého průběhu lovu popisuje.

1.2.1 Zahájení lovu

Má-li uživatel dostatečný počet bodů[1] a jeho *historie* neobsahuje žádný *zamknutý* cíl, z hlavní nabídky je možné zahájit přípravu nového lovu. Pokud jeho *historie* nějaký *zamknutý* cíl obsahuje, je nejdříve povinen jej nechat *splnit*.

Příprava lovu začíná výběrem oblasti, kterou chce hráč prozkoumat. Tato oblast má kruhový tvar a je určena GPS souřadnicemi středu a poloměrem. Poloměr je povolen v rozsahu $1 - 50\,\text{km}$.

Nakonec je možné vybrat preferovanou denní dobu, standardně na ni není brán ohled, lze však preferovat pouze den nebo noc. Tato doba určuje typ fotografií, které budou uživateli nabízeny. Volba *den* z výsledků vynechá všechny fotografie, které budou příliš tmavé. Volba *noc* naopak ponechá pouze tmavé fotografie. Za podobnost tmavé fotografie je však možné získat pouze zhruba poloviční bodové ohodnocení než za fotografii světlou. Důvodem je snaha motivovat hráče zejména k focení denních snímků.

Na základě těchto parametrů je vygenerována a zahájena nová hra.

1.2.2 Nabídka cílů

První ze tří hlavních součástí hry je *nabídka* cílů. Po zahájení nové hry jsou postupně získávány dostupné cíle. Nabídka obsahuje pouze cíle z vybrané oblasti, které zároveň nebyly v některém z předcházejících lovů *splněné*, *zamítnuté* a dále neobsahuje cíle *blokované*.

Po získání všech cílů, kterých je vždy maximálně 30, je hra zahájena a je spuštěn odpočet času stráveného ve hře. Není-li k dispozici žádný cíl, uživatel není bodově penalizován a může zvolit jinou oblast.

Šest náhodných získaných cílů je *zpřístupněno*. Pokud není k dispozici více než 6 cílů, jsou *zpřístupněny* všechny *nepřístupné*.

Uživatel získá možnost *přijmout* 4 **přístupné** cíle a hra je nyní plně připravena.

V každém okamžiku má hráč možnost *přijmout* pouze určitý počet cílů. Je tedy nutné se na základě informací dostupných k cílům rozhodnout, které z cílů vyřešit nejdříve. Pokud mu žádný *přístupný* cíl nevyhovuje, může si koupit *zpřístupnění* některého *nepřístupného* cíle, případně pokud *přístupný* cíl zamítne, bude jeden z *nepřístupných* cílů *zpřístupněn* automaticky, nějaký *nepřístupný*

[1] Podrobněji dále sekci Bodování zahájení lovu 1.3.1.

cíl však musí být v takovém případě k dispozici v nabídce, jinak *zamítnutý* cíl nahrazen nebude.

1.2.3 Plnění cílů

U *přijmutých* cílů je průběžně kontrolována vzdálenost od uživatele. Bodový zisk pro *splněný* cíl má dvě složky: objevení a podobnost fotografií. Ve chvíli, kdy je cíl *navštíven*, je u něj zaznamenán počet bodů, které získal za splnění první složky, ten je závislý na době, která uběhla od zahájení hry. Poté je možné vybrat fotografii, která uživateli nejvíce vyhovuje a přejde k fotografování. Úkolem je pořídit co nejpodobnější fotografii ke zvolené. Zvolená fotografie je vykreslena na popředí náhledu fotoaparátu poté, co je upravena hranovým detektorem. Je nutné pořídit alespoň jednu fotografii, nejvýše však 3, podobnost je poté započítána podle té s nejvyšším bodovým výsledkem.

Po vyfocení cíle se cíl stává *zamknutým*. Dále jsou v nabídce *zpřístupněny* dva náhodné *nepřístupné* cíle. Pokud nabídka obsahuje pouze jeden *nepřístupný* cíl, ten bude *zpřístupněn*, pokud neobsahuje žádný *nepřístupný* cíl, uživateli pouze přibude, stejně jako i v předchozích případech, možnost *přijmout* jeden další cíl.

Poté co je vyhodnocena podobnost vyfoceného cíle, je cíl *splněn* a uživatel získá body za objevení/navštívení a podobnost fotografií.

Vyfocením cíle je tedy umožněno plnit další cíle a celý proces se opakuje, hra pokračuje až do chvíle, kdy je lov ukončen.

1.2.4 Ukončení lovu

Lov končí, pokud je každý cíl v nabídce *splněný* či *zamítnutý*, případně pokud vyprší časový limit na oblast, který je 24 hodin od zahájení lovu, nebo hráč zahájí lov nový ještě před nastáním jedné z předchozích situací.

1.3 Bodování

Průběh celé hry je závislý na hodnocení jednotlivých akcí, které uživatel provádí. Toto hodnocení je zvoleno tak, aby uživatele pobízelo k plnění úkolů, zároveň však aby zamezilo nevhodnému průběhu hry a umožnilo vytvořit tabulku pro porovnání všech hráčů mezi sebou.

1.3.1 Zahájení hry

Zahájení nové oblasti stojí **20** bodů. Toto množství bodů však není uživateli odstraněno v případě, že vybraná oblast žádný cíl obsahovat nebude.

Při registraci je novému uživateli přiděleno **100** bodů. To umožní okamžité zahájení nové oblasti hned několikrát po sobě, což je pro nového uživatele vhodné zejména proto, že může prozkoumat vhodné nastavení podmínek nového lovu, případně pro oblast obsahující mnoho cílů použít množinu cílů, která mu v této oblasti vyhovuje nejvíce.

1.3.2 Nabídka cílů

V libovolné fázi hry lze *nepřístupný* nebo *přístupný* cíl *zamítnout*. Toto *zamítnutí* stojí **5** bodů, výsledkem této akce je, že tento cíl již nikdy nebude uživateli ve hře nabídnut. Byl-li tento cíl *přístupný*, bude nahrazen jiným cílem, jak bylo popsáno v pravidlech dříve (viz pravidla plnění cílů 1.2.3).

Přijmutí cíle je zdarma, přijmout však lze vždy jen omezený počet cílů. Následným splněním tohoto cíle lze získat až **20** bodů, jak je detailněji vysvětleno dále.

Zpřístupnění cíle je prováděno automaticky náhodně po *splnění* cíle, lze ho však také vyžádat za poplatek **10** bodů.

Ostatní změny stavů cíle jsou bez bodové penalizace, jedná se o změny řízené funkcí aplikace.

1.3.3 Průběh hry

Plnění cílů je způsob, jakým lze ve hře získávat body. Částí bodů je uživatel ohodnocen již za navštívení cíle, druhá část bodů je získána po jeho vyfocení. Přidělení bodů za obě části splnění cíle je však možná až po ohodnocení podobnosti fotografií.

Pro výpočet počtu bodů, které hráč získal *splněním* cíle je použita funkce *kazič*. Hodnota této funkce je závislá na době, kterou probíhá aktuální lov t (v hodinách). Prvních pět hodin po započetí oblasti (tzv. doba **stagnování**) umožňuje získat maximální počet bodů, poté začne rapidně klesat. Konkrétně je definována takto:

$$kazič(t) = \begin{cases} m, & \text{je-li } t < s, \\ m \cdot \exp\left(\frac{t-s}{l-s} \cdot \log(1/m)\right), & \text{je-li } t \geq s, \end{cases}$$

kde $m = 10$ je maximální hodnota funkce, $s = 5$ je doba stagnování (v hodinách), $l = 24$ je maximální doba lovu (opět v hodinách) a člen $\log(1/m)$ zajišťuje minimum funkce: $kazič(l) = 1$ na konci lovu. Průběh funkce je možné vidět na obrázku 1.1.

Pro výpočet bodů za *navštívení* cíle, je použita dolní celá část z funkce *kazič*, její minimum je tedy 1. Toto bodování je vidět na stejném obrázku 1.1:

$$body_n(t_n) = \lfloor kazič(t_n) \rfloor,$$

kde t_n je čas navštívení cíle.

V případě výpočtu bodů za podobnost fotografií je použit vzorec

$$body_p(t_p, f_r, F) = \max\left(1, \lfloor podobnost(f_r, F) \cdot kazič(t_p)) \rfloor\right), \tag{1.1}$$

kde t_p je čas uložení všech fotografií cíle, f_r je fotografie referenční a F jsou fotografie cíle. Funkce $podobnost(f_r, F)$ je implementována na serveru a vrací hodnotu z intervalu $[0, 1]$, kde 0 odpovídá nejmenší a 1 největší podobnosti. Přesněji vrátí podobnost páru fotografií $(f_r, F(i))$ (pro $i \in [1,3]$), který je vyhodnocen jako nejpodobnější. Je-li však referenční fotografie označena jako tmavá, hodnota funkce *podobnost* bude vždy pouze z intervalu $[0, 0.5]$.

Celkově pak hráč získá:

$$body(t_n, t_p, f_r, F) = body_n(t_n) + body_p(t_p, f_r, F)$$

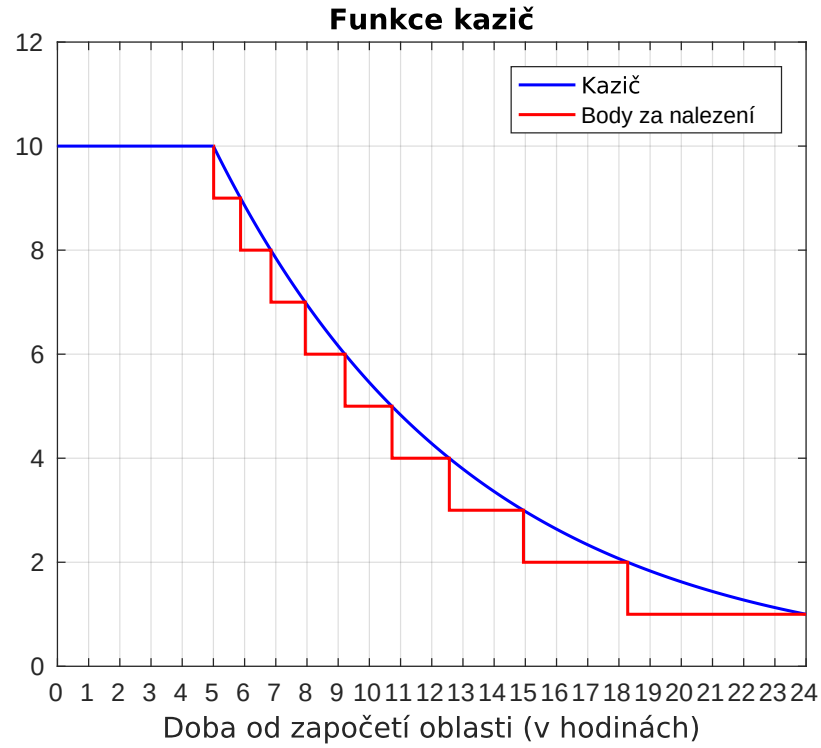
a díky vlastnostem funkcí $body_n$ a $body_p$ platí

$$2 \leq body(t_o, t_p, f_r, F) \leq 20.$$

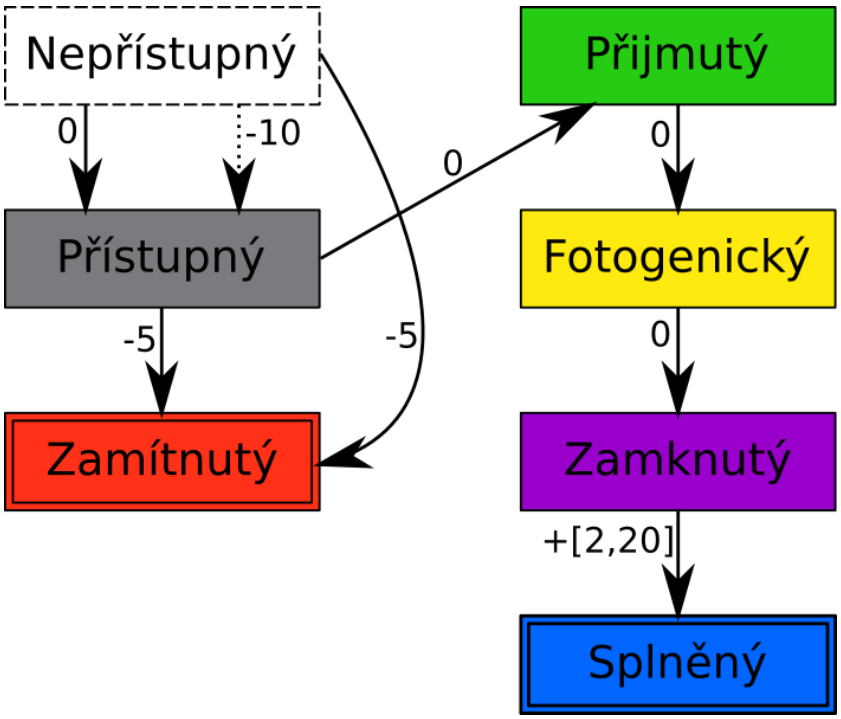
1.3.4 Zvláštní situace

Dostane-li se hráč pod hranici **20** bodů, je mu jich nastaveno **30** po skončení posledního (neúspěšného) lovu. V nejhorším případě je tedy nucen vyčkat až 24h, bez dvaceti bodů totiž nemůže aktuální lov ukončit zahájením lovu nového.

Jelikož hra uchovává *historii*, která obsahuje i všechny *zamknuté* cíle a za tyto cíle je proto ještě možné získat body jejich vyhodnocením, uživatel nedostává **30** bodů ihned po skončení lovu pokud jeho *historie* nějaké *zamknuté* cíle obsahuje, ale až v případě, že dolní hranice **20** bodů nedosáhne ani po *splnění* těchto cílů. **30** bodů tedy dostává, pokud splní všechny zamknuté cíle v *historii*, aktuálně žádný lov neprobíhá a hráč má méně než **20** bodů.



Obrázek 1.1: Funkce *kazič* a její dolní celá část.



Obrázek 1.2: Možné stavy cíle a vliv změny stavu na skóre.

2. Analýza

Projekt byl rozdělen do tří částí. Serverová část (OHS) uchovává data, analyzuje fotografie a komunikuje s externími zdroji, klientská část (OHC) slouží k prezentaci a ovládání hry na mobilních zařízeních a jako třetí vznikla speciální knihovna (OHL) pro komunikaci mezi dvěma předchozími částmi.

Tato kapitola přibližuje návrh a funkci hlavních komponent těchto součástí.

Jako jazyk pro programování software pro OHS a OHL byla zvolena Java, jejíž výhodou je především podpora v mnoha operačních systémech a navíc se vzájemně doplňuje s volbou vývoje části OHC pro platformu Android.

2.1 Knihovna OHL

Pro zajištění standardizované komunikace mezi serverovou a klientskou částí byla vytvořena speciální knihovna. Tato sekce podrobněji popíše její součásti a způsob využití.

2.1.1 Komunikační primitiva

Následující objekty byly navrženy k uchování konzistentních herních informací zejména pro vzájemnou komunikaci serveru a klienta, některé z nich však byly použity či rozšířeny i pro účely lokálního zacházení.

Photo a SImage

Objekt `Photo` je navržen k uchování jedné fotografie cíle. Krom fotografie přenáší také údaj o jejím typu, který je vyhodnocen a nastaven při jejím vytváření na serveru. Z důvodů serializovatelnosti a kompatibility není použit standardní objekt pro uchování fotografie samotné (např. `BufferedImage` není na platformě Android ani dostupný, `Bitmap` z Androidu naopak není dostupný ve standardní Javě), nýbž vlastní, serializovatelný obrázek `SImage`. Ten data uchovává v poli bytů reprezentující obrázek ve formátu JFIF (tj. po kompresi JPEG, která je pro tento účel vhodná, neboť obrázky cílů jsou fotografie a komprese je nutná pro docílení minimálního objemu přenesených dat).

Place

Tento objekt uchovává informace o *místu*, z kterého po přidání stavu typicky vzniká herní objekt *cíl*. *Místo* však obecně nemusí kromě GPS pozice a ID obsahovat žádné další informace.

Player

Každý uživatel má v této hře nějaké jméno, identifikátor a skóre. Objekt `Player` slouží k uchování všech těchto informací.

2.1.2 Návrh komunikace – třída Request a Response

K snadnému ovládání a rozšiřování komunikačních požadavků byl použit návrhový vzor Command pattern (Gamma a kol., 1994, kapitola 5). Knihovní třídy s koncovkou `Request` slouží pouze jako kostra příkazu, která uchovává parametry konkrétního požadavku. Její funkci pak zajišťuje až server. Na požadavek vytvořený třídou s koncovkou `Request` je vytvořena odpověď, která se sestává z několika hodnot zahrnutých ve třídě `Response`. Libovolný požadavek také může skončit výjimkou, v takovém případě je namísto `Response` použit objekt `OHException`, který vzniklý problém podrobněji popisuje.

2.2 Server OHS

Serverová část této aplikace vykonává úkoly, které jsou buď náročné na prostředky, nebo pracují s daty, která vyžadují sdílení mezi všemi uživateli, eventuálně je vhodné uchovávat je jako součást přirozeného vývoje aplikace. Mezi úlohy se řadí:

- výpočet podobnosti fotografií
- rozhodnutí denní doby fotografie
- získání cílů pro danou oblast
- vedení informací o hráčích
- uchovávání dat o cílech

Pro dlouhodobé uchovávání dat na lokálním úložišti se proto na serveru nachází databáze. K zajištění požadavku o cíle bylo využito Google Places API.[1]

Server naslouchá na portu, který je pevně určen při jeho spuštění. Pro klienty je připraveno několik vláken, proto umožňuje-li to typ úloh, požadavky jsou vyřizovány paralelně. Počet vláken je defaultně 8 a je možné jej změnit v konfiguračním souboru serveru. Ostatní klienti jsou mezitím nuceni vyčkat na uvolnění prostředků serveru a požádat o vyřízení později znovu.

K snažšímu ovládání serveru je navíc vždy spuštěno speciální vlákno, které pracuje se standardním vstupem a výstupem. Je možné z něj například sledovat připojené klienty, upravovat informace v databázi, nebo server bezpečně ukončit. (Více později v sekci 4.2.2.)

2.2.1 Komunikace s Google Places API

K vytvoření cílů pro uživatelem danou oblast bylo zvoleno použití existující rozsáhlé databáze přístupné přes Google Places API[1]. Ta umožnila rychlé a snadné vyřešení tohoto problému. Konkurenční řešení jako Global Places[2], nebo Foursquare[3], přestože dovolují větší počet dotazů na časový úsek, se zaměřují spíše na restaurace, hotely a informace k místům nejsou natolik podrobné.

[1] Hlavní rozcestník k Google Places API: `https://developers.google.com/places/`.
[2] Global Places od Factual: `https://www.factual.com/products/global`.
[3] Webové stránky Foursquare pro vývojáře: `https://developer.foursquare.com/`.

Základní objekt, se kterým Google Places API zachází se nazývá **místo** (orig. **place**). Místo má vždy unikátní *place_id* a může mít GPS pozici, jméno, adresu, reference na fotografie a další, pro tuto aplikaci méně zajímavé informace jako hodnocení ceny (např. vstupné), otevírací doby a uživatelské recenze. *Cíl* v herním smyslu je poté rozšířením místa, které má dostatečné množství informací a nějakou fotografii.

Pro všechny požadavky přes toto API je nutné využívat speciální klíč[4], který identifikuje aplikaci a umožňuje Google sledovat a omezovat množství požadavků. V základní verzi činí 1 000 požadavků na den, po poskytnutí dalších osobních informací je limit zvýšen na 100 000 požadavků za den.

Komunikace probíhá v doporučovaném formátu JSON, k parsování výsledků server využívá knihovnu JSON.simple[5] od Google.

Radar Search

V první fázi je nutné použít Place Search. Server konkrétně z dostupných možností využívá Radar Search[6], jehož hlavní výhodou (například vůči Nearby Search) je, že pro zadanou oblast vrací nejvíce výsledků (až 200). Nevýhodou je, že jeden výsledek obsahuje pouze minimální množství informací o místu. Jde o GPS souřadnice a *place_id*.

Požadovanými parametry radarového vyhledávání jsou:

- *API klíč* pro identifikaci aplikace
- zeměpisná šířka a délka udávající střed oblasti k prohledání
- poloměr oblasti v metrech z intervalu $[0; 50\,000]$
- *klíčové slovo*, *jméno* nebo *typ*

Oblast, kterou uživatel ve hře prohledává je kruh, parametry zeměpisné polohy a poloměr tedy přímo odpovídají požadavkům herních pravidel.

Jako poslední parametr byla zvolena možnost *typ*. Těchto *typů* Google poskytuje zhruba 100, mezi nimi například airport, hospital, cafe, park, doctor nebo zoo. Server využívá typů university, synagogue, city_hall, church, museum, mosque, které maximalizují počet cílů vhodných dle pravidel. Pro jednoduché specifikování všech typů najednou bylo využito notace

```
type1|type2|...
```

kde svislítko označuje logickou disjunkci. Tato notace však byla označena jako zastaralá a funkční bude do února roku 2017. Nově je nutné požadavky provádět pro jednotlivé typy zvlášť. Pro zachování menšího počtu požadavků a rychlosti odpovědi pro klienty server stále používá kompaktnější zastaralou notaci.

[4]Tento klíč lze získat přihlášením do konzole `https://console.developers.google.com`, kde je třeba vybrat či vytvořit projekt, který bude API využívat. Poté se pro projekt zvolené API povolí a tím je nový klíč uvolněn.

[5]Web projektu JSON.simple: `https://code.google.com/archive/p/json-simple/`.

[6]Popis Radar Search jako jedné z možností Place Search: `developers.google.com/places/web-service/search#RadarSearchRequests`.

Place Details

Funkce, která umožňuje získat dodatečné informace k místu se nazývá Place Details[7]. Jejími parametry jsou:

- *API klíč* pro identifikaci aplikace
- *placeid*, jedná se o údaj *place_id* získaný v předchozí fázi pomocí Radar Search (sekce 2.2.1)
- *jazyk* (kód jazyka, umožňuje preferenci daného jazyka ve výsledcích detailů)

Údaj *jazyk* je nepovinný a aplikace ho nevyužívá, i přesto je popis míst v České republice nejčastěji v českém jazyce.

Z výsledků tohoto dotazu je zjištěno a uloženo jméno, adresa, url (jde o oficiální webovou stránku od Google pro toto místo), website (autoritativní webová stránka místa) a ikonu (jde pouze o url na ni a ikona symbolicky označuje typ místa).

Poslední hodnotou, která je pro tuto aplikaci zásadní je seznam až deseti speciálních referencí na fotografie, ty je možné získat následujícím dotazem pro každou referenci zvlášť.

Place Photos

Reference na fotografie získané předchozím požadavkem (Place Details) je možné použít k získání samotné fotografie pomocí Place Photos[8] Je nutné vyplnit:

- *API klíč* pro identifikaci aplikace
- *photoreference*
- *max_výška* a *max_šířka* s hodnotami z intervalu $[1; 1600]$

Photoreference je reference jednoznačně identifikující fotografii a byla získaná předchozím krokem Place Details (viz 2.2.1). *Max_výška* a *max_šířka* udávají maximální povolené rozměry vrácené fotografie. Je-li fotografie menší, nebude její velikost upravena, je-li větší, bude zmenšena tak, aby nebyla větší jak žádný z těchto rozměrů, aby jeden z nich splňovala přesně a její poměr stran byl zachován.

2.2.2 Výpočet podobnosti fotografií

Jedná se o několikafázový postup, na vstupu jsou zapotřebí oba obrázky k porovnání ne nutně shodných rozměrů.

Jednotlivé fáze výpočtu, jenž jsou prováděny na obou obrázcích ze vstupu, vycházejí z metody vysvětlené v dokumentu (Qin Lv, 2004). Jejich popis následuje:

[7]Dokumentace k Place Details: `https://developers.google.com/places/web-service/details`.

[8]Funkce Place Photos je detailně popsaná na: `https://developers.google.com/places/web-service/photos`.

1. **Segmentace obrázku** – je prováděna pomocí algoritmu *K-Means*[9,10] na obrázku v HSV barevném modelu, ve kterém je eukleidovská metrika přirozená pro měření vizuální blízkosti barev. Počet segmentů je nejvýše 64, segmenty konvergující ke stejnému centroidu jsou zastoupeny pouze jedním segmentem, jednobarevný obrázek má tedy pouze 1 segment.

2. **Výpočet charakteristiky** – převedení všech získaných segmentů na menší data – vektory, které popisují některé z jejich vlastností. Tento vektor se sestává jednak z momentů, které charakterizují barevnou distribuci. Výpočet používá barevný model HSV, moment je počítán pro každou jeho složku, které má tento model 3 (hue, saturation a value), momenty tedy celkově přispějí do vektoru pro segment devíti hodnotami. Jejich definice následuje:

   1. moment střední hodnota barevné složky v segmentu, tj.

   $$E_i \coloneqq \frac{1}{N}\sum_{j=1}^{N} p_{i,j} \ ,$$

   kde N je počet pixelů v segmentu a $p_{i,j} \in [0;1]$ udává hodnotu j-tého pixelu v segmentu pro $i \in \{1,2,3\}$-tou složku HSV modelu, vyskytuje se na 1., 4. a 7. pozici vektoru, jeho hodnoty jsou z intervalu $[0;1]$.

   2. moment směrodatná odchylka, využívá 1. moment

   $$\sigma_i \coloneqq \sqrt{\frac{1}{N}\sum_{j=1}^{N}(p_{i,j} - E_i)^2} \ .$$

   Vyskytuje se na 2., 5. a 8. pozici vektoru s hodnotami z intervalu $[0;1]$.

   3. moment šikmost, tedy informace o tvaru distribuce

   $$s_i \coloneqq \sqrt[3]{\frac{1}{N}\sum_{j=1}^{N}(p_{i,j} - E_i)^3} \ .$$

   Hodnoty tohoto momentu jsou na 3., 6. a 9. pozici a patří do intervalu $[-1;1]$.

   Pro každý segment existuje nejmenší takový obdélník, zde označen O, který tento segment pokrývá a jeho hrany jsou rovnoběžné s okraji obrázku. Charakterizační vektor dále obsahuje 5 hodnot, které popisují jeho pozici, tvar a jsou již invariantní k jeho barvě:

   - Logaritmus z poměru šířky ku výšce O – popis tvaru segmentu invariantní ke vzdálenosti fotografa a objektu, vyskytuje se na 10. pozici s hodnotou z $[0;\ log(max(w,h))]$, kde w je šířka a h výška obrázku.

[9] Obecný algoritmus K-Means je popsaný například v knize (MacKay, 2002, str. 285-287).

[10] Zmíněný dokument používá pokročilou segmentační metodu JSEG `http://old.vision.ece.ucsb.edu/segmentation/jseg/`.

- Logaritmus z normované velikosti obsahu O – pokud je obrázek vyfocen z větší vzdálenosti, nebo pod odlišným úhlem, tato hodnota není invariantní ke vzdálenosti fotografa. Je na 11. pozici vektoru a jeho obor hodnot je $[0; log(1)]$.
- Hustota segmentu, tj. poměr počtu pixelů v segmentu vůči obsahu O – informace o tvaru segmentu invariantní ke vdálenosti fotografa a objektu. Jde o 12. údaj, který přispívá hodnotou z $[0; 1]$
- X a Y souřadnice těžiště oblasti ve vztahu k celému obrázku – jediná informace o pozici této oblasti. Tyto poslední, tj. 13. a 14. hodnoty vektoru nabývají hodnot $[0; 1]$.

Hodnoty HSV jsou v rozsahu $[0; 1]$ pro *hue*, *saturation* i *value*. Všechny hodnoty z vektoru pro segment jsou použity pro výpočet metriky, která je také součástí výsledné hodnoty EMD.

Logaritmus v případě 10. a 11. složky slouží k vhodnému přeškálování hodnot vektoru, nechceme například, aby příliš záleželo na mírně odlišné vzdálenosti od objektu při focení.

Jako výsledek dostáváme pro každý segment vektor dimenze 14.

3. **Vážení vektorů** – jako příprava pro další krok je nutné zajistit vážení jednotlivých segmentů a určit funkci udávající vzdálenost mezi nimi.

   Pro vážení, čili určení důležitosti segmentu v celkovém obrázku je použita druhá odmocnina z její plochy. Nakonec je nutné provést normalizaci tak, aby součet vah dával hodnotu 1.

   Pro měření vzdálenosti je použita Manhattanská metrika, tj. pro n-dimenzionální vektory p, q je $d(p, q) = \sum_{i=1}^{n} |p_i - q_i|$, kde n je v tomto případě počet složek charakterizačních vektorů, tedy 14. Stejně jako v Qin Lv (2004) je vzdálenost $d(p, q)$ navíc zhora omezena, zde hodnotou 1, tj.:

$$d(p, q) = \min\left(\sum_{i=1}^{n} |p_i - q_i|,\ 1\right)$$

4. **Podobnost distribucí** – samotné porovnávání je založeno na měření pomocí metody *Earth Mover's Distance*[11].

   Máme-li distribuci $P = (p_1, w_{p_1}^P), \ldots, (p_m, w_{p_m}^P)$ a druhou distribuci $Q = (q_1, w_{q_1}^Q), \ldots, (q_n, w_{q_n}^Q)$, kde w_p^R je váha vektoru p v rozdělení R, dále danou metriku $d(u,v)$ pro vektory u a v, pak EMD nejdříve najde tok f, kde $f_{i,j}$ je mezi hodnotami p_i a q_j tak, že minimalizuje

$$\text{PRÁCE}(P,Q) \coloneqq \sum_{i=1}^{m} \sum_{j=1}^{n} (f_{i,j} \cdot d(p_i, q_j))$$

   s podmínkami:

[11] Metoda je popsaná také na webové stránce `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm`.

$$f_{i,j} \geq 0 \qquad \ldots \text{tok pouze z } P \text{ do } Q \qquad (2.1)$$

$$\sum_{j=1}^{n} f_{i,j} \leq w_{p_i} \qquad \ldots \text{omezení odtoku z } P \qquad (2.2)$$

$$\sum_{i=1}^{m} f_{i,j} \leq w_{q_j} \qquad \ldots \text{omezení přítoku do } Q \qquad (2.3)$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j} = min(\sum_{i=1}^{m} w_{p_i}, \sum_{j=1}^{n} w_{q_j}) \qquad \ldots \text{přesunutelné množství} \qquad (2.4)$$

$$\text{kde } i = 1 \ldots m \text{ a } j = 1 \ldots n$$

EMD je poté definováno jako normalizovaná funkce PRÁCE, pro již spočtenou optimální f jako

$$EMD(P,Q) \coloneqq \frac{\text{PRÁCE}(P,Q)}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j}}.$$

Součet hodnot vah je v naší úloze normalizován $\sum_{i=1}^{m} w_{p_i} = \sum_{j=1}^{n} w_{q_j} = 1$, podmínka (2.4) je tedy zjednodušena na rovnost 1 a proto také

$$EMD(P,Q) = \text{PRÁCE}(P,Q).$$

Jde tedy o problém lineárního programování, na jeho vyřešení je použita externí knihovna SCPSolver[12].

Pro ověření funkčnosti algoritmu v této aplikaci byl proveden následující experiment, který je v příloze P.12. Byly přichystané dvě množiny deseti obrázků, sestávající se z pěti dvojic. První množina obsahovala fotografie pro testování algoritmu před jeho nasazením do hry a existencí OHC, tj. fotky jsou pořízené telefonem ze standardní aplikace pro focení.[13] Každá dvojice je v této první množině sestavena z fotografií, které mají být velmi podobné, jedná se o fotografie pořízené s minimální časovou prodlevou a umělou, mírnou chybou pozice při focení. Druhá množina byla vytvořena z dvojic (fotografie cíle, referenční fotografie) získaných po implementaci OHC ze samotné hry O-Hunter, tj. fotografie cíle je pořízená telefonem a referenční je z Google Photos.[14]

Do analyzátoru podobnosti OHS byly obě množiny postupně vloženy, bylo provedeno 10 porovnání na každé dvojici z 10-ti fotografií. Každá fotografie byla navíc testována i sama se sebou, tj. celkově 55 dvojic fotografií. Grafy 2.1a a 2.2a obsahují výsledky vynesené pomocí boxplotů pro obě množiny, každý boxplot vynáší vzorek deseti vyhodnocení podobnosti pro nějakou dvojici fotografií. Pořadí boxplotů je zvoleno tak, aby na začátku bylo 10 porovnání fotografií se sebou (shodné), dále 5 vyhodnocení s fotografií z dvojice (podobné) a nakonec zbytek

[12] Domovské stránky knihovny SCPSolver: `http://scpsolver.org/`.

[13] V příloze P.12 jsou testovací fotografie ve složce `vstup_2`, speciálně obrázky `IMG_03` a `IMG_04` schválně zachycují jiné portály.

[14] V příloze P.12 jsou herní fotografie ve složce `vstup_1`

porovnání (odlišné). Výsledky z první množiny naznačují maximální směrodatnou odchylku podobnosti pro jednu dvojici 23,96, minimální 0,05 a průměrnou 11,79 pro první množinu, 30,17, 7,08 respektive 18,16 pro druhou.

Z tohoto důvodu je algoritmus modifikován tak, že je spuštěn 5× po sobě a výsledná hodnota je spočtena jako aritmetický průměr. Hodnota počtu opakování je konfigurovatelná. Výsledky po této modifikaci jsou na grafech 2.1b a 2.2b. Výsledky směrodatných odchylek 9,52 maximální, 0,73 minimální a 5,40 průměrná pro první množinu, respektive 16,93, 4,56 a 8,79 pro druhou ukazují, že tato úprava metodu „stabilizuje“, nicméně za cenu zpomalení celého procesu.

Pro shrnutí všech výsledků jsou na obrázku 2.3 ještě boxploty, které jsou pro každou kategorii dvojic (shodné, podobné a odlišné) vytvořené z průměrovaných hodnot deseti podobností pro každou dvojici patřící do kategorie. Výsledná hodnota podobnosti je pak (i přes prakticky nevyužitý interval $[0{,}9; 1]$) přímo použita jako výstup tohoto algoritmu a pro určení bodového ohodnocení uživatele za podobnost fotografie podle vzorce (1.1).

2.2.3 Rozhodnutí denní doby fotografie

Pro učinění rozhodnutí, zdali je fotografie vyfocena v noci (noční), nebo ve dne (denní) byl použit algoritmus publikovaný v dokumentu (Chen, 2011). Jedná se o jednoduchou metodu, která dokáže velmi rychle toto rozhodnutí provést s uspokojivými výsledky.

Ze vstupního obrázku F se vybere náhodně N pixelů. Hodnota RGB z pixelu p_i pro $i \in [1; N]$ je převedena na intenzitu vzorcem

$$I_i = 0.2126r_i + 0.7152g_i + 0.0722b_i \tag{2.5}$$

, kde $0 \leq r_i,g_i,b_i \leq 255$ jsou postupně hodnoty červené, zelené a modré složky pixelu p_i. Z těchto N intenzit se spočítá průměr $A(F)$, tj. odhadovaná intenzita obrázku. Rozhodnutí o denní době je poté provedeno na základě porovnání s prahovou hodnotou $0 \leq T \leq 255$ následovně:

$$doba(F) = \begin{cases} \text{denní}, & \text{je-li } A(F) \geq T, \\ \text{noční}, & \text{jinak.} \end{cases}$$

V původním dokumentu byla prahová hodnota zvolena $T = 100$ a velikost množiny vzorků $N = 94$. Testy na fotografiích pořízených hráčem poukazovaly na velmi podobné hodnoty. Test na reálných fotografiích dostupných z Photo Details (viz sekce 2.2.1) však ukázal na jiné hodnoty. V tomto experimentu bylo 434 obrázků získaných pomocí mechanismu Google Places nejdříve protříděno a odstraněna byla loga, neboť algoritmus je cílený na fotografie (příklad nevhodného obrázku je 2.7). Zbylých $M = 403$ fotografií ručně roztříděno na denní, kterých bylo 337, a noční, tj. 66 fotografií. Na všech těchto rozkategorizovaných fotografiích byl proveden výpočet hodnoty $A_j := A(F_j)$ pro $j \in [1; 403]$. Hodnota T pak umožňuje spočítat počet denních fotografií, které jsou rozpoznané správně: $TP_D := \left|\{F_j : A_j \geq T \wedge F_j \text{ je denní fotografie} \wedge j \in 1 \ldots M\}\right|$ a podobně počet správně rozpoznaných nočních fotografií TP_N. Vhodný práh T byl pak zvolen tak, aby maximalizoval součet $TP_D + TP_N$.

Pro tento experiment byla nejvhodnější prahová hodnota $T = 74.4$ s výsledky: $TP_D = 330$ a $TP_N = 51$, tj. 94.54 % celková přesnost, 97.92 % přesnost

denních a 77.27 % u nočních. V algoritmu je použita prahová hodnota $T = 75$ a počet vzorků nastaven na $N = 128$. Výsledky experimentu jsou vykresleny na obrázku 2.5 a obrázku 2.6. Na obrázcích 2.8 jsou příklady špatně vyhodnocených obrázků s největší odchylkou od prahové hodnoty, na obrázcích 2.9 jsou dvě náhodně vybrané fotografie, které byly tímto algoritmem vyhodnoceny správně. Experiment je včetně skriptů v MATLABu, použitých obrázků a výsledků v příloze P.11.

2.2.4 Databáze

Pro dlouhodobé uchovávání velkého množství paměťově nenáročných dat hráčů je na serveru využita databáze Apache Derby[15]. Databáze obsahuje čtyři tabulky, které jsou vytvořeny ve chvíli, kdy je k jedné z nich poprvé vyžádán přístup.

Tabulka HRAC

Tabulka slouží k uchávání informací o každém registrovaném hráči.

Název	**Typ**	**Význam**
ID	INTEGER	Je automaticky přidělené po registraci nového hráče a je dále používané k jednoznačné identifikaci příchozích požadavků z klientské části.
PREZDIVKA	VARCHAR(16)	Přezdívka asociovaná s každým hráčem. Jde o znakový řetězec, který je unikátní a hráč si jej volí jako své jméno ve hře, které může být veřejné.
SKORE	INTEGER	Uchovává současný bodový zůstatek hráče. Tato hodnota je i v případě používání offline módu klienta aktuální, neboť klient získává/ztrácí body pouze při dostupném připojení k serveru, po aktualizaci informace v databázi.
HESLO	VARCHAR(64)	Řetězec uchovávající hash hesla uživatele.

Tabulka SPLNENE

Tato tabulka obsahuje splněné cíle pro všechny hráče hry.

[15] Odkaz na domovské stránky: `https://db.apache.org/derby/`, použita je verze 10.12.1.1.

Název	Typ	Význam
ID_HRACE	INTEGER	Jednoznačně určuje který hráč cíl splnil, použito je *ID* z tabulky HRAC.
ID_CILE	VARCHAR(64)	Řetězec jednoznačně identifikující cíl. Tato hodnota byla získána z Google Places API a je tedy případně možné znovu požádat o doplňující informace v případě potřeby, viz 2.2.1.
ID_FOTO	VARCHAR(512)	Jednoznačně identifikuje fotografii, která byla použita jako referenční ke splnění cíle. Jde podobně o identifikátor získaný z Google Place Photos.[a]
TIMESTAMP	TIMESTAMP	Uchovává datum a čas splnění cíle.
OBJEVENI	INTEGER	Udává počet bodů získaných za objevení cíle.
PODOBNOST	INTEGER	Udává počet bodů získaných za vyfocení cíle.
LOV	INTEGER	Uchovává pořadí lovu, ve kterém byl cíl splněn.

Pozn: [a] Viz sekce 2.2.1 a o fotografii je možné případně znovu zažádat přímým použitím této hodnoty. Google nespecifikuje jak dlouhá tato reference může být, maximální délka řetězce proto byla zvolena empiricky.

Tato tabulka uchovává nejvíce informací ze všech tabulek, neboť obsahuje cíle s nejzajímavějším stavem a může být použita k vytvoření historie lovů.

Tabulka ZAMITNUTE

Tabulka ukládá zamítnuté cíle všech hráčů.

Název	Typ	Význam
ID_HRACE	INTEGER	Jednoznačně určuje, který hráč cíl zamítnul. Použité je *ID* z tabulky HRAC.
ID_CILE	VARCHAR(64)	Jednoznačně identifikuje cíl a byla získána z Google Places API, viz sekce 2.2.1.

Jde o cíle, ke kterým se uživatel již nechce nikdy vracet, jsou pro něj nezajímavé, proto u nich čas zamítnutí zaznamenáván není.

Tabulka BLOKOVANE

Obsah této tabulky je tvořen správcem serveru, obsahuje všechny zablokované cíle.

Název	Typ	Význam
ID_CILE	VARCHAR(64)	Stejně jako v předchozích tabulkách jde o identifikátor získaný z Google Places API.

Cíle získané z Radar Search (viz sekce 2.2.1) umožňuje tato tabulka jednoduše odfiltrovat ještě před dalším zpracováním bez podrobnějšího zkoumání.

RadarSearchRequester

Odpovídající požadavek vyžaduje zadání zeměpisné šířky a délky a poloměr v metrech. Tím je specifikován střed a poloměr oblasti, pro kterou je úlohou vrátit seznam dostupných cílů.

Tato třída implementuje požadavek tak, že nejprve provede dotaz na Radar Search z Google Places API (viz sekce 2.2.1), čímž získá až 200 výsledků v podobě *place_id* odpovídajících *míst*. V souladu s herními pravidly uplatní informace z databáze, konkrétně tabulek SPLNENE, ZAMITNUTE a BLOKOVANE k tomu, aby byly z výsledků odstraněny cíle, které již hráč *splnil*, *zamítnul* nebo jsou *zablokované*. Takto odfiltrované výsledky posílá zpět v objektu `Response`.

FillPlacesRequester

Vstupem přidruženého `Requestu` je seznam *place_id* míst, ke kterým se mají získat další informace, tj. detaily a fotografie.

Implementace pro každé *place_id* přes Place Details (viz sekce 2.2.1) zažádá o vyplnění informací. Pro vyplňování detailů je k dispozici 32 vláken, pokud je seznam dlouhý, informace se mohou vyplňovat paralelně. Po vyplnění údajů je přečten údaj o počtu referencí na fotografie. Pokud žádná reference k dispozici není, cíl je zahozen. Pro dostupné reference (kterých je maximálně 10) server nabízí 10 vláken, které provádí vyřízení požadavku přes Photo Details (viz sekce 2.2.1). Počet vláken je možné změnit v konfiguračním souboru serveru. Pokud se požadavek o fotografii výjimečně nevydaří, pak je taková fotografie vynechána. Pokud je obrázek přijat na server bez problému, je na něm provedena analýza denní doby (viz sekce 2.2.3).

Po dokončení všech požadavků o detaily, přiřazení a vyhodnocení fotografií je seznam míst s důležitými údaji a nějakou fotografií odeslán zpět klientovi.

CompareRequester

Pro tento požadavek je vstupem hráč vytvářející požadavek, *referenční* fotografie a seznam fotografií, které uživatel vyfotil jako podobné k *referenční*. Výstupem je hodnota z rozsahu $[0; 1]$, kde 0 značí největší rozdílnost, 1 největší podobnost obrázků. Tato hodnota má být vrácena pro pár s největší podobnostní.

Implementace postupně na každý pár (F_r, F_i), kde F_r je *referenční* fotografie a F_i jsou fotografie ze seznamu, spustí analýzu podobnosti (viz sekce 2.2.2) a klientovi zpět vrací výsledek páru s nejvyšší podobností.

LoginRequester a RegisterRequester

Oba požadavky pracují především s objektem `Player`. Na server se dostává jméno uživatele a hash jeho hesla. Pomocí databáze, tabulky HRAC, je ověřena existence uživatele, je podle ní nastavena hodnota skóre, při registraci je vytvořen nový záznam a klientovi je v `Response` vrácen nový objekt `Player`.

V případě nesprávného hesla nebo neexistujícího uživatele při přihlašování a stejně tak při již existujícím jméně při registraci uživatele nového je odeslána speciální hodnota ve výjimce `OHException` tak, aby klient správně rozpoznal situaci, která na serveru v databázi nastala.

Databázové třídy Requester

Ostatní, méně zajímavé požadavky (např. `RejectPlaceRequest`, `CompletePlaceRequest`) operují nad databází a jejich funkce i implementace je přímočará.

2.3 Klient OHC

Klientská část aplikace slouží k hraní, ovládání a vizualizaci dat hry. Jejími úkoly jsou také:

- ukládání informací cílů
- výpočet pomocné úpravy referenční fotografie
- ukládání pořízených fotografií cílů

2.3.1 Pomocné úpravy referenční fotografie

Pro usnadnění focení cíle je upravená referenční fotografie při focení cíle umístěna na popředí náhledu fotoaparátu. Pro dosažení maximální nápomoci při hledání správné pozice, ze které byla fotografie pořízena se pomocné úpravy referenční fotografie sestávají ze tří hlavních kroků:

1. detekce hran (obrázek 2.12)
2. prahování hran (obrázek 2.13a)
3. barevná úprava (na obrázku 2.13b)

Jedna z technik úprav obrázků využívá operaci zvanou *konvoluce*. Binární operace **konvoluce** se značí $*$ a jednorozměrně je definována

$$(f * h)(x) \coloneqq \int_{-\infty}^{\infty} f(t) \cdot h(x-t)\,\mathrm{d}t \left(= \int_{-\infty}^{\infty} f(x-t) \cdot h(t)\,\mathrm{d}t \right)$$

pro funkci f, a takzvané *konvoluční jádro* h. Diskrétní verzi konvoluce ve dvou dimenzích je pak možné použít pro účely úprav obrázků. Její definice je:

$$(f * h)(x, y) = \sum_{i=-k}^{k} \sum_{j=-l}^{l} f(x - i, y - j) \cdot h(i, j)$$

kde f je jednokanálový obrázek, tj. $f(x, y)$ určuje intenzitu kanálu pixelu na příslušné pozici $[x, y]$, a h je konvoluční jádro s rozměry $(2k + 1) \times (2l + 1)$. **Konvolučním jádrem** je v této verzi konvoluce matice, typicky čtvercová, malých rozměrů (3×3, 5×5). Vhodnou volbou této matice je možné obrázek například rozmazat, zaostřit, nebo najít **hrany**, tj. místa v obrázku, kde je velký barevný rozdíl hodnot sousedících pixelů. Jednokanálový obrázek ze standardního RGB obrázku lze získat buď převedením do černobílého obrázku, nebo aplikací konvoluce na každou složku zvlášť. V tomto případě byl zvolen převod pomocí vzorce založeného na citlivosti oka pro jednotlivé složky (2.5).

Nějakým způsobem je navíc nutné dodefinovat hodnoty funkce $f(x, y)$ i pro souřadnice, které odpovídají pixelům v okolí obrázku (tzn., pro obrázek rozměrů $m \times n$, souřadnicím $(\mathbb{Z} \times \mathbb{Z}) \setminus (\{1, 2, \ldots m\} \times \{1, 2, \ldots n\})$). Existuje několik tradičně používaných postupů, pro účel této aplikace byla zvolena metoda naprostého vynechání těchto pixelů, tj. f je mimo obrázek dodefinována jako nulová.

Pro detekci hran byl použit Sobelův-Feldmanův filtr (publikovaný například v knize (Pratt, 2007)). Tento filtr využívající konvoluci je založený na měření 2D gradientu jednokanálového obrázku a v základní verzi dokáže vyznačovat horizontální a vertikální hrany. Jelikož funkce f není spojitá, gradient je možné pouze odhadnout na základě známých diskrétních bodů. Sobelův-Feldmanův filtr gradient odhaduje nepřesně, neboť používá pouze jádra rozměrů 3×3, výsledek je však pro praktické použití kvalitní, jádro je složeno z malých celých čísel a výpočet konvoluce je díky velikosti jádra rychlý. Tento filtr navíc provádí vyhlazování ve směru kolmém na detekovanou hranu (díky posloupnosti hodnot $[1, 2, 1]$, resp. $[-1, -2, -1]$ dává větší váhu prostředním hodnotám), což zvyšuje robustnost proti šumu v obrázku. Posloupnosti $\pm[1, 0, -1]$, $\pm[2, 0, -2]$ pak slouží k zmíněné aproximaci první derivace.

Pro vylepšení o detekci diagonál jsou použité, krom základních čtyř konvolučních jader, další čtyři. Čtyři jádra pro detekci svislých a vodorovných hran vzniknou rotací základního jádra o 90°, 180°a 270°, stejně tak vznikají i čtyři jádra pro detekci diagonálních hran, rotují pouze modifikované základní jádro. Některé příklady těchto konvolučních jader jsou na obrázcích 2.10 a 2.11.

Protože jader je osm, výpočet konvoluce je upraven takto:

$$(f * h)(x, y) = \max_{c=1,\ldots 8} \sum_{i=-k}^{k} \sum_{j=-l}^{l} f(x - i, y - j) \cdot h_c(i, j)$$

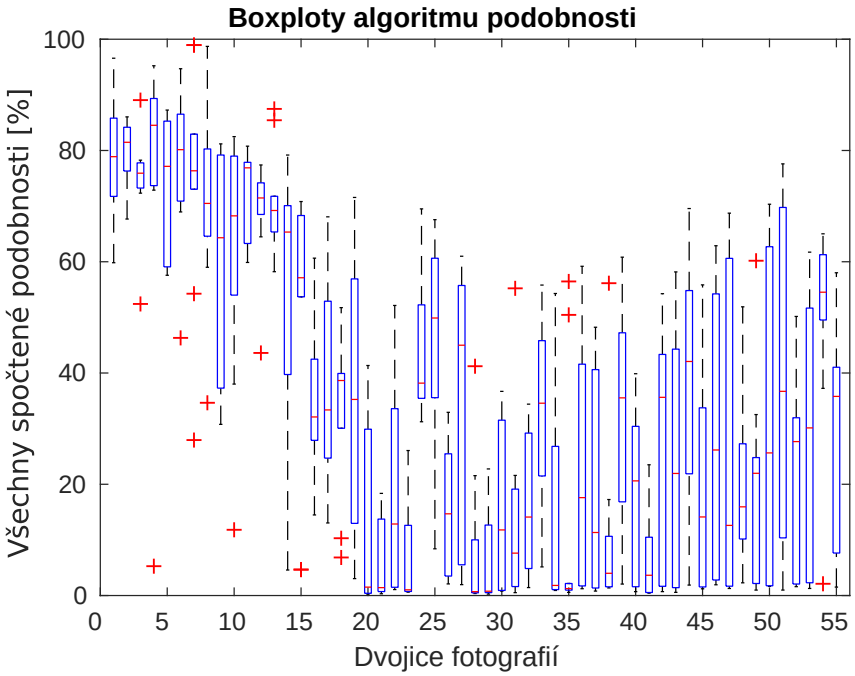
na konkrétní pixel je tedy použito jádro s maximální reakcí v tomto pixelu.

Obrázek hran je vytvořen tak, že má na hranách bílé pixely, jinde je průhledný. Tohoto výsledku je dosaženo přímým použitím výstupu filtru $(f * h)$ pro složku alfa, složky RGB mají všechny hodnotu 255. Sobelův-Feldmanův filtr má na jedné hraně až několik odezev a dodefinování funkce obrázku nulami mimo obraz samotný způsobuje typické zvýraznění okraje (nebyl-li obrázek u okraje černý).

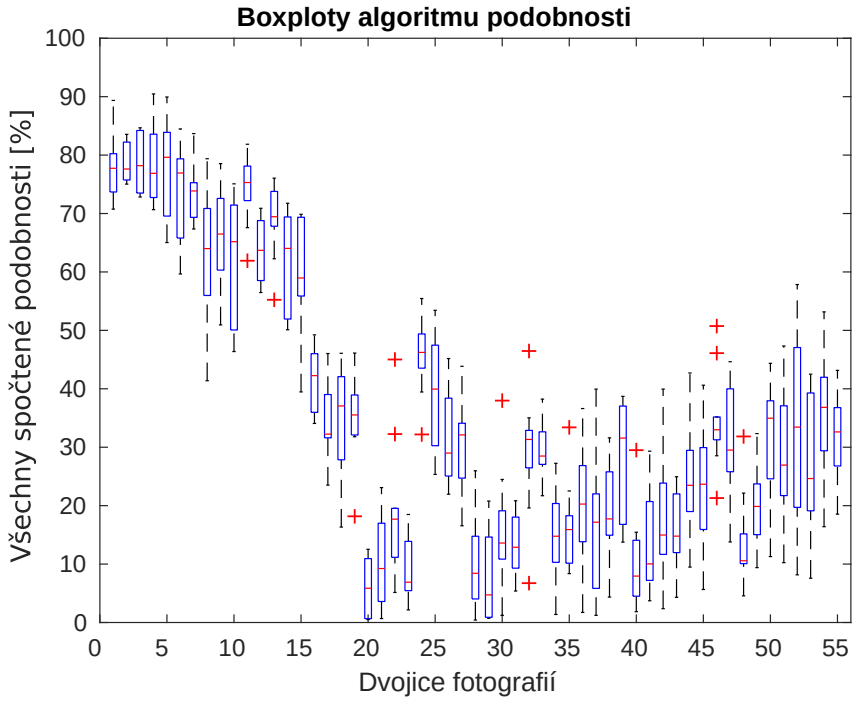
Oba tyto jevy (ačkoliv pro aplikaci samotnou nevýznamné) je možné na výsledku z první fáze algoritmu vidět na obrázku 2.12b.

Z obrázku 2.12b je také vidět, že většina označených pixelů má velmi malou, avšak neprůhlednou hodnotu. Tyto pixely uživateli nijak nepomohou, další fáze tedy provede jejich odstranění prahováním. „Vhodná“ prahová hodnota je pro každý obrázek odlišná, proto algoritmus pracuje tak, že vynechá 8/10 nejvíce průhledných pixelů. Výsledek je na obrázku 2.13a.

Pro zvýšení viditelnosti při použití na popředí fotoaparátu je navíc barva hran třetí fází algoritmu pozměněna v HSV modelu posunem složky hue. Konkrétní hodnotu této složky volí uživatel podle vlastní preference. Výsledek s příkladem použití je na obrázku 2.13b.

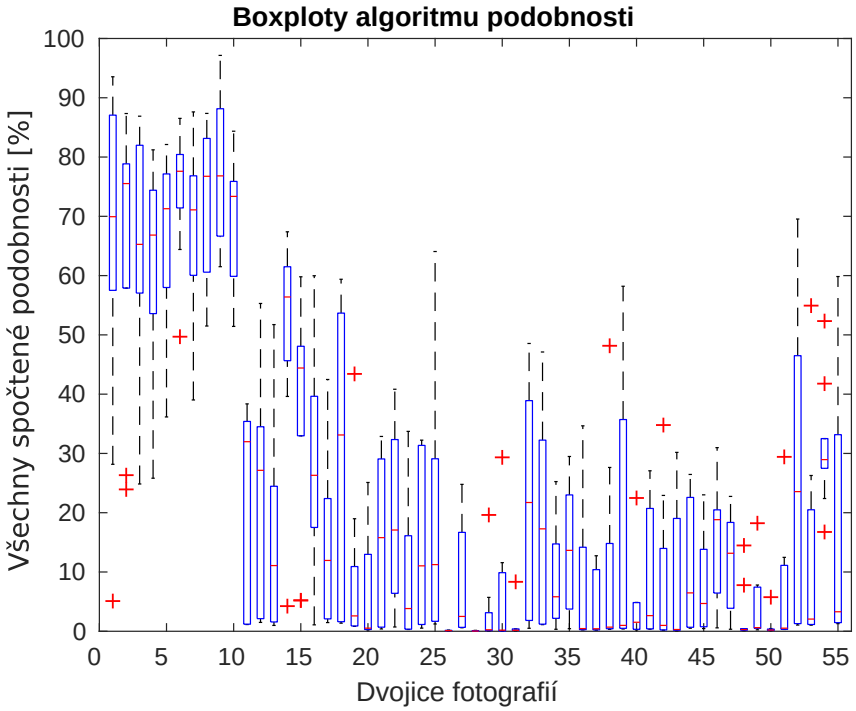


(a) Výkon metody podobnosti bez modifikace.

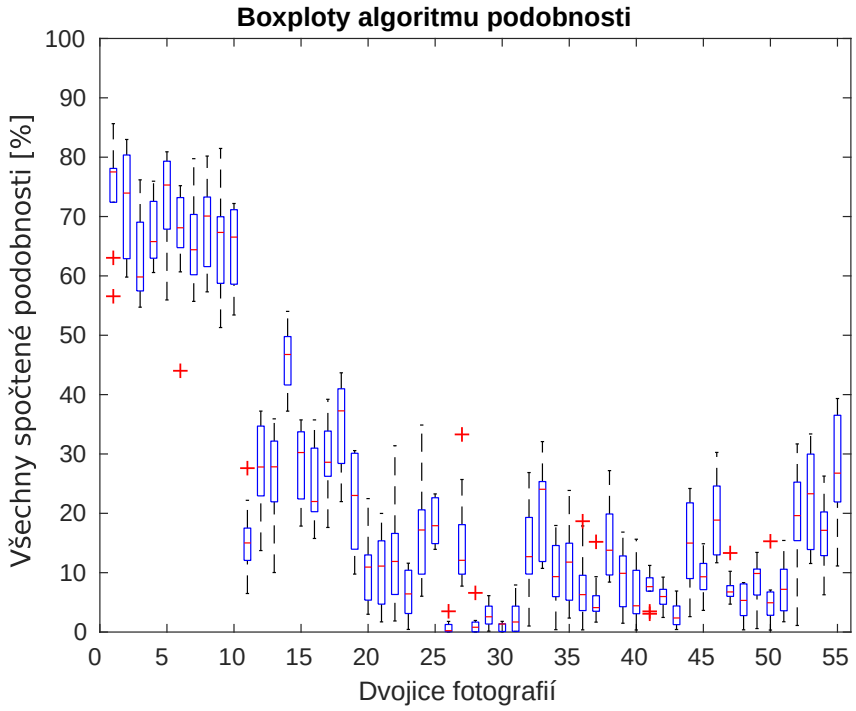


(b) Výkon metody podobnosti po průměrování pěti pokusů.

Obrázek 2.1: Boxploty pro podobnost každé dvojice deseti vlastních fotografií, na začátku sama se sebou (1.–10.), dále se svoji nejpodobnější (11.–15.) a nakonec s ostatními (16.–55.). Červené křížky označují *odlehlé hodnoty*, tj. hodnoty mimo interval $[q_1 - 1{,}5 \cdot (q_3 - q_1); q_3 + 1{,}5 \cdot (q_3 - q_1)]$, pro q_1,q_3 první a třetí kvartil.

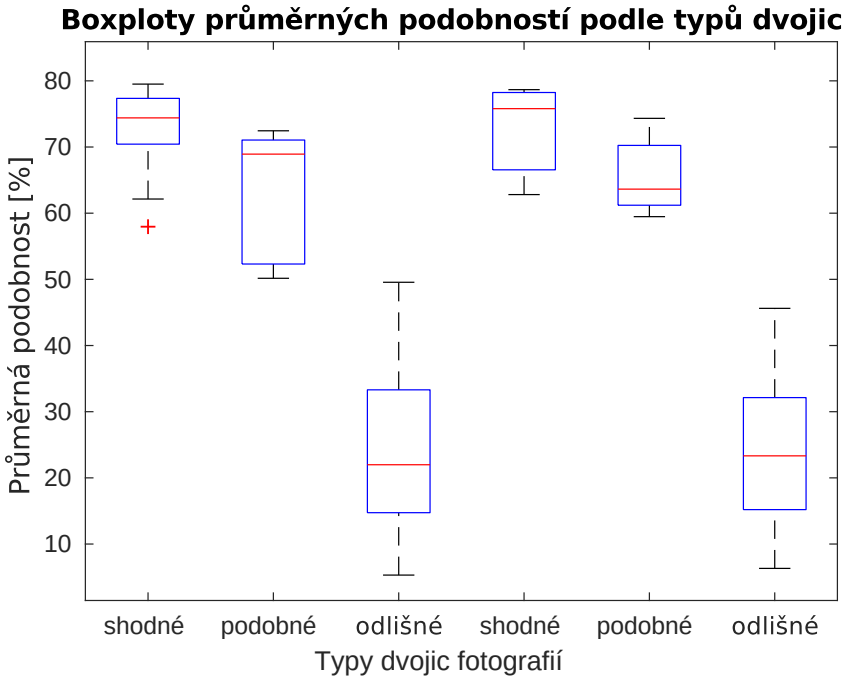


(a) Výkon metody podobnosti bez modifikace.

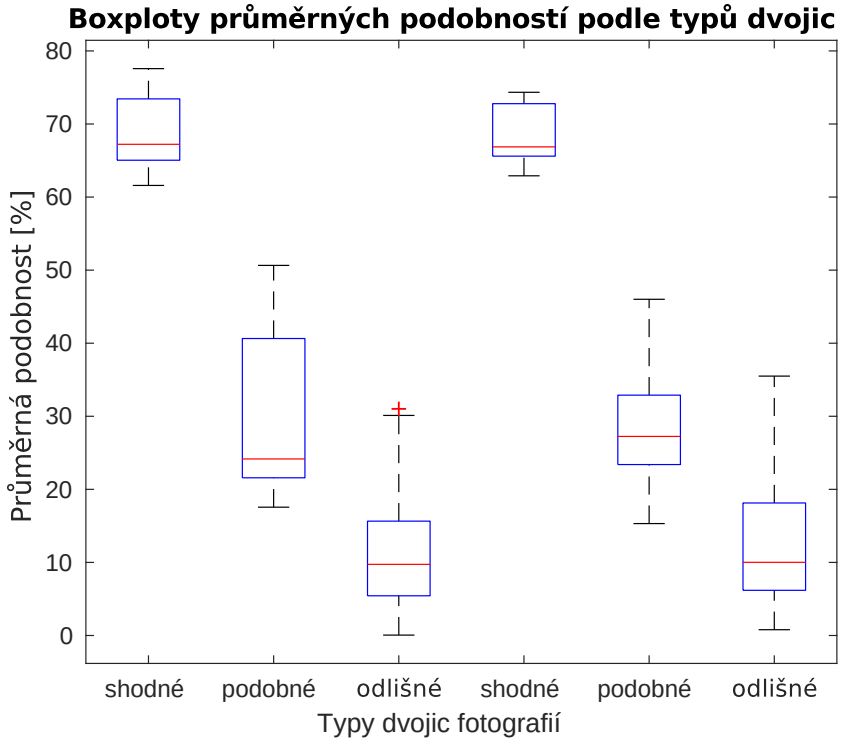


(b) Výkon metody podobnosti po průměrování pěti pokusů.

Obrázek 2.2: Boxploty pro podobnost každé dvojice deseti reálných herních fotografií, na začátku sama se sebou (1.–10.), dále se svoji nejpodobnější (11.–15.) a nakonec s ostatními (16.–55.). Červené křížky označují *odlehlé hodnoty*, tj. hodnoty mimo interval $[q_1 - 1{,}5 \cdot (q_3 - q_1); q_3 + 1{,}5 \cdot (q_3 - q_1)]$, pro q_1,q_3 první a třetí kvartil.



(a) Průměrné výsledky pro množinu vlastních testovacích fotografií.



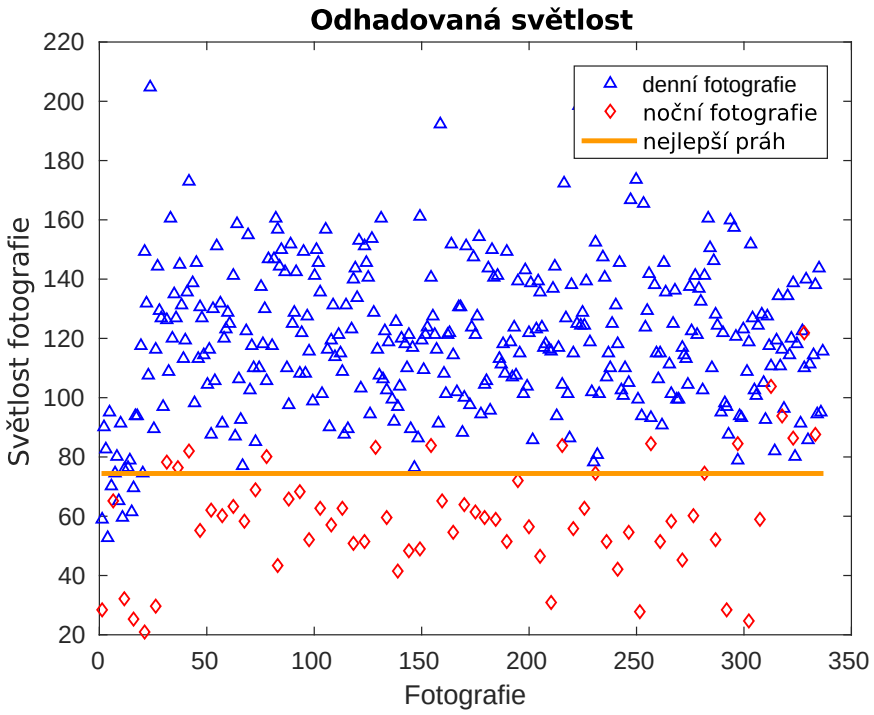
(b) Průměrné výsledky pro množinu reálných herních fotografií.

Obrázek 2.3: Boxploty průměrovaných výsledků podobnosti pro všechny tři typy dvojic fotografií. Na každém obrázku tři boxploty vlevo odpovídají metodě bez modifikace, vpravo po modifikaci průměrováním.

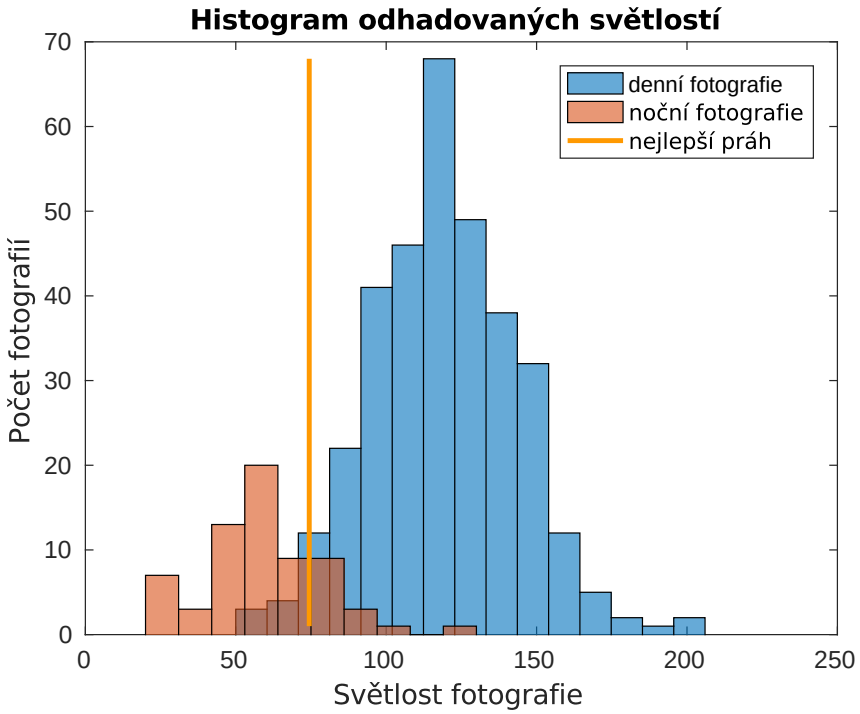
(a) Nejlépe ohodnocená dvojice v množině reálných herních fotografií.

(b) Příklad špatně ohodnocené dvojice v množině reálných herních fotografií ve skutečnosti podobných fotografií.

Obrázek 2.4: Příklady z reálného použití algoritmu podobnosti ve hře. Červené tečky označují centroidy segmentů, číslo podobnost v procentech.

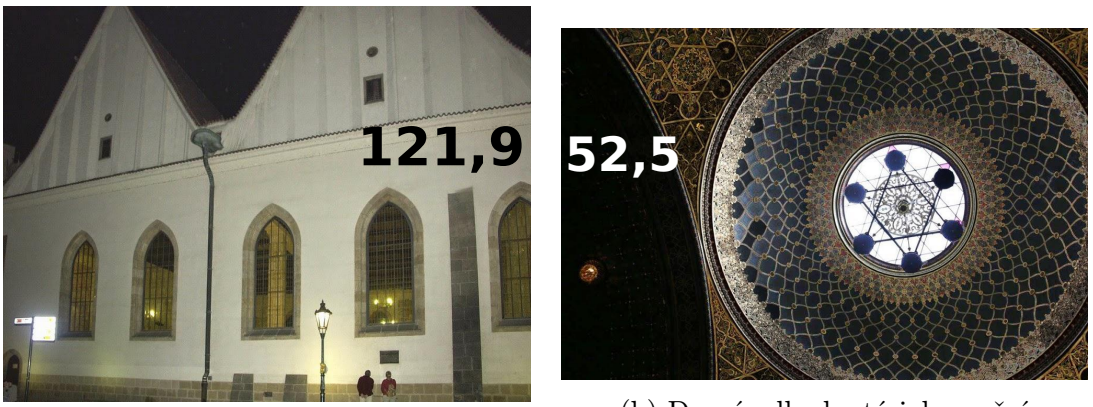


Obrázek 2.5: Výsledky analýzy světlosti fotografií.



Obrázek 2.6: Histogram odhadované intenzity fotografií.

Obrázek 2.7: Příklad odstraněných obrázků.



(a) Noční odhadnutá jako denní

(b) Denní odhadnutá jako noční

Obrázek 2.8: Obrázky s největší odchylkou odhadované světlosti.

(a) Denní fotografie

(b) Noční fotografie

Obrázek 2.9: Příklad správně rozhodnutých obrázků s hodnotou jejich odhadované světlosti.

1	2	1
-1	-2	-1

(a) Jádro detekující svislé hrany

1		-1
2		-2
1		-1

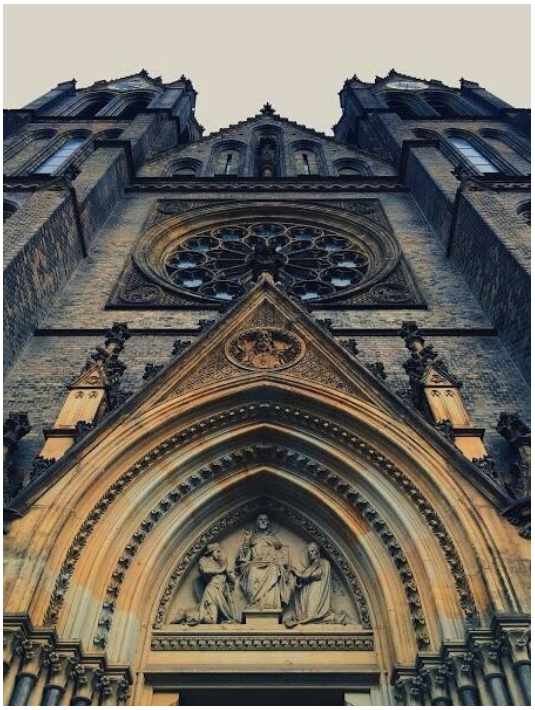
(b) Jádro detekující vodorovné hrany

Obrázek 2.10: Příklady jader použitých pro detekci horizontálních a vertikálních hran.

2	1	
1		-1
	-1	-2

2	1	
1		-1
	-1	-2

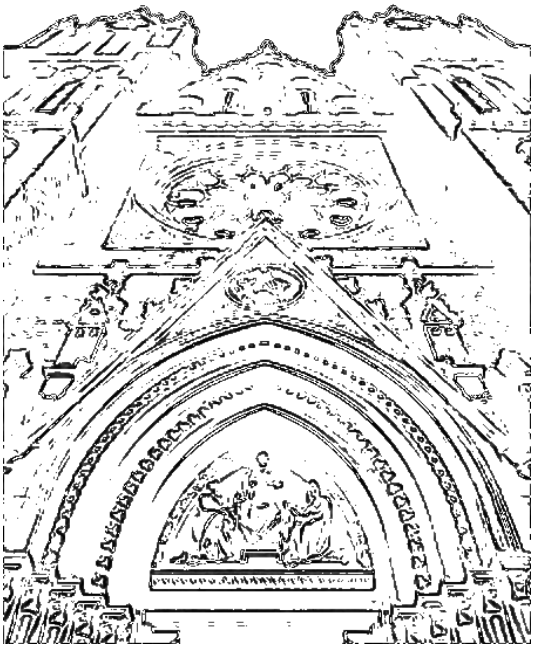
Obrázek 2.11: Příklady jader použitých pro detekci diagonálních hran.



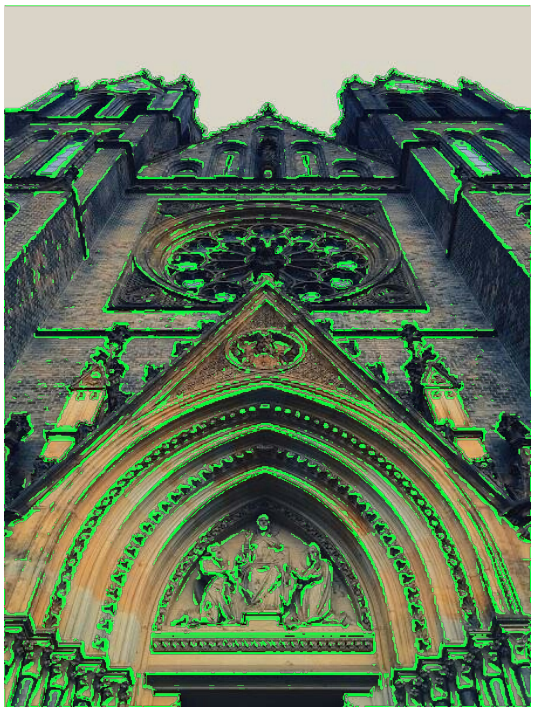
(a) Referenční fotografie

(b) Referenční fotografie po detekci hran

Obrázek 2.12: Detekce hran referenční fotografie.



(a) Oprahovaná referenční fotografie po detekci hran



(b) Vhodně přebarvená upravená referenční fotografie na popředí původní referenční fotografie

Obrázek 2.13: Prahování a přebarvení referenční fotografie po detekci hran s příkladem použití.

3. Vývojová dokumentace

Tato kapitola se věnuje některým z implementačních detailů, především problémům z kapitoly analýzy.

3.1 Knihovna

Knihovna OHL je napsána v jazyce Java 7, a navržena tak, aby umožnila komunikaci mezi serverem a klientem a její použití i rozšíření bylo co nejsnadnější. Všechny objekty, které knihovna obsahuje jsou nutně serializovatelné, tj. implementují rozhraní `Serializable`.

3.1.1 Návrh a rozšíření komunikace

Hlavními objekty, které zajišťují uchování všech informací potřebných ke komunikaci jsou abstraktní třída sloužící jako rozhraní všech požadavků `Request`, třída `Response` pro předávání výsledků požadavků a objekt `OHException` pro oznámení výjimek.

Abstraktní třídu `Request`, která uchovává ID žádajícího klienta, čas vytvoření požadavku a případně objekt `Player`, na níž je deklarována metoda

```
public Response execute() throws OHException {}
```

rozšiřují další třídy, jako `RadarSearchRequest`, `CompleteTargetRequest` nebo `CompareRequest`. Metoda `execute()` je pak na každé z nich na serveru implementována konkrétním způsobem, který řeší daný požadavek.

Pro přidání vlastního typu požadavku je nutné nejdříve deklarovat rozšíření abstraktní třídy `Request`, interní proměnné je vhodné volit s modifikátorem viditelnosti `protected`, aby k nim mohla snadno přistoupit rozšiřující třída na serveru. Pokud třída přijímá požadavek od přihlášeného hráče, pak bude jejím parametrem `Player player` a volat pak ve vlastním konstruktoru musí konstruktor předka `super(player)`. Není-li objekt `Player` parametrem, pak volá konstruktor bez parametrů pomocí `super()`. Konstruktor předka, tj. třídy `Request` zajistí vytvoření časové značky a uchování informací o tvůrci požadavku. Aby server mohl jednoduše rozšiřující třídy vyplnit daty odpovídající třídy z knihovny, je navíc nutné přidat speciální konstruktor, který podstatný obsah rozšiřující třídy překopíruje. Ukázková implementace následuje:

```
public class CustomRequest extends Request {
    protected int value;
    public CustomRequest(Player player, int value) {
        super(player);
        this.value = value;
    }
    public <R extends CustomRequest> CustomRequest(R request) {
        super(request);
        value = request.value;
    }
}
```

Metoda `execute()` vrací objekt `Response`, který obsahuje ID původního klienta, čas vytvoření této odpovědi a nepovinně objekty `Player` (například pokud byl požadavek na přihlášení), číselný údaj podobnosti (pokud bylo žádáno o výpočet podobnosti) a pole vyplněných objektů `Place` pokud byl požadavek na vyhledání cílů.

Při vykonávání požadavku na serveru může nastat chyba, z toho důvodu byla vytvořena třída `OHException`, která umožňuje na klienta přenést podrobnější informaci o typu chyby (např. chyba v databázi, konkrétní problém při registraci, nebo upozornění na zastaralou verzi aplikace při chybě deserializace). Na tu pak může klient podle typu problému vhodným způsobem zareagovat.

3.2 Server

Nadcházející sekce se zabývají detaily implementací na serveru, které zahrnují způsob obsluhy klientů, analýzu obrazu a přístup k databázi.

3.2.1 Komunikace s Google Places API

Všechny potřebné dotazy na Google Place API popsané v části 2.2.1 jsou implementované ve třídě `PlacesCommunicator`. Obstarávají je metody `radarSearch(...)`, `placeDetails(...)` a `photoRequest(...)`. Je zde také funkce `filterByName(String)`, která může být implementována vlastním způsobem tak, aby vracela `true` pro názvy míst, které mají být odstraněny z výsledků `placeDetails(...)`.

Všechny metody si nejdříve připraví URL odpovídající danému požadavku, ta vždy musí obsahovat Google API klíč, který si metody přečtou z načteného konfiguračního souboru. Poté se pomocí `java.net.HttpURLConnection` připojí k serveru Google a přečtou celý výstup, který je ve formátu JSON. V další fázi jsou tato data naparsována knihovnou JSON Simple a pokud jsou dostupné potřebné údaje, výsledek se ve formě objektů primitiv z OHL vrátí jako výsledek. Ve výjimečných případech dojde k odmítnutí spojení ze strany serverů Google. Pak je vytvořena výjimka `OHException` a reakcí na ni rozhodne vyšší vrstva. Pro RadarSearch je například vhodné požadavek provést znovu, pro PlaceDetails nebo PhotoRequest to není nutné, neboť je zpravidla dostupná naprostá většina ostatních míst, resp. fotografií.

3.2.2 Obsluha klientů

Obstarávání klientů je zajištěno fondem vláken s fixním počtem nastavených defaultně na 8. Pro vytvoření a správu fondu vláken je použita třída `ExecutorService` z balíku `java.util.concurrent`. Počet vláken tohoto fondu je nastavitelný v konfiguračním souboru (viz sekci 4.2.3), pro projevení změn je zapotřebí server spustit znovu. Server naslouchá na portu a adrese z konfiguračního souboru a pro každého klienta vytvoří třídu `ClientWorker`, která práci ve vlákně klienta vykonává. Pokud však k dispozici další vlákno není, klient je zařazen do fronty, kterou interně implementuje `ExecutorService`. Na klientovi je tedy nutné dobu čekání na vykonání požadavku omezit, fronta může být libovolně dlouhá.

Knihovna pro výpočet lineárního programu (SCPSolver) neumožňuje paralelizaci a je proto hlavním úzkým hrdlem obsluhy klientů.

`ClientWorker` dostává třídu `Requester` na ní tedy jednoduše zavolá metodu `execute()`, počká na výsledek a ten, nebo výjimku odešle klientovi. `ExecutorService` z fronty automaticky vybere dalšího klienta a je-li stále validní, obdobně ho obslouží.

Třída `Server` zastává všechny tyto úlohy a je instanciována z hlavní třídy `OHunterServer`, ve které je kromě metody na spuštění serveru několik dalších metod pro testování funkcí celého OHS. Server samotný je spouštěn metodou `runInBackground()` v samostatném vlákně, aby případně neblokoval běh hlavního programu a bylo možné testovat připojení klientů. Port a adresa serveru jsou nastavované v konstruktoru.

3.2.3 Implementace komunikace – třídy Requester

Aby bylo možné oddělit deklaraci API knihovny OHL od implementace, třídy typu `Request` metodu `execute()` v knihovně neimplementují. Pro tento účel byly na serveru navrženy třídy s názvem končícím `Requester`. Každá z nich rozšiřuje knihovní požadavek příslušného názvu (např. `LoginRequester` rozšiřuje knihovní `LoginRequest`), krom toho navíc implementuje metodu `execute()`.

Po příchozím požadavku od klienta, tj. typu `Request`, je pomocí Java Reflection API proveden převod na příslušný `Requester`, k tomu je použit překládající konstruktor, na třídě `Requester` je zavolána metoda `execute()` a výsledek, případně chyba je vrácena zpět klientovi.

Následující kód poukazuje na snadnost implementace ukázkového požadavku `CustomRequest` z části 3.1.1. Je nutné dodržet především konvenci koncovky a v konstruktoru pak zavolat konstruktor předka z knihovny:

```
public class CustomRequester extends CustomRequest {
    public BlockPlaceRequester(CustomRequest request) {
        super(request);
    }
    @Override
    public Response execute() throws OHException {
        // Implementace výkonné části vlastního požadavku
        // s přístupem k prostředkům serveru.
    }
}
```

3.2.4 Analyzátor podobnosti

Hlavní třídou, která zajišťuje analýzu obrazu je `Analyzer`. Na ní pak metoda `computeSimilarity(Photo, Photo)` poskytuje výpočet podobnosti dvou vstupních obrázků. Pro jejich předání je použit objekt knihovny OHL, neboť v tomto objektu jsou na server od klienta přijaty.

Prvním krokem je převedení objektů `Photo` na standardní `BufferedImage`. To je možné vykonat s pomocí třídy `ImageIO`, neboť `Photo` interně uchovává obrázek komprimovaný běžným JPEG. Dále jsou velikosti obou obrázků upraveny tak, aby jejich větší rozměr měl velikost danou v konfiguračním souboru a původní

poměr šířky a výšky nebyl změněn. Tento krok způsobuje zmenšení obrázků a je vykonáván z důvodů časové optimalizace.

Dále je obrázky nutné podrobit segmentační technice, tu obsahuje a provádí třída `Segmenter`. V tomto projektu byla zvolena metoda segmentace pomocí algoritmu K-Means, který jako segment označí pixely s podobnou barvou (barevná podobnost je hodnocena v barevném modelu HSV). Jednotlivé segmenty jsou reprezentované třídami `Segment`. Každý `Segment` uchovává množství základních charakteristik příslušného segmentu v obrázku. Jsou jimi pozice těžiště vůči celému obrázku, počet všech pixelů v segmentu, čtyři hraniční body a tři barevné momenty.

Algoritmus K-Means pracuje s třídou `Pixel`, která uchovává pozici pixelu v obrázku, jeho barevné složky v barevném modelu HSV, vzdálenost od nejbližšího centroidu a index tohoto centroidu. Na `Pixelu` je také definována funkce pro výpočet vzdálenosti k jinému `Pixelu`. Ta využívá eukleidovskou metriku na všech třech barevných složkách. Obrázek reprezentovaný těmito `Pixely` je pro lepší zacházení uložen ve třídě `MyImage`, která si navíc pamatuje i rozměry původního obrázku. K-Means nejdříve několikrát náhodně vybere požadovaný počet pixelů jako volbu počátečních centroidů a pro další postup zvolí kombinaci s nejmenší počáteční minimalizační funkcí, kterou je suma všech vzdáleností od nejbližších centroidů. Počet náhodných výběrů a velikost náhodného vzorku lze upřesnit v konfiguračním souboru. Dále algoritmus probíhá konvergencí ke správnému řešení úpravou voleb centroidů, jako zastavovací kritéria jsou volena jednak rychlost změny minimalizační funkce a navíc je dán maximální počet iterací, který lze upravit v konfiguračním souboru.

Optimalizační problém je reprezentován třídou `Problem`. Ta uchovává dva seznamy (pro každý obrázek jeden) dvojic (`Vector`, `weight`), kde `Vector` uchovává charakterizační hodnoty nějakého segmentu a `double weight` je jemu příslušící váha. Dvojice jsou ve třídách `DistrPair`. Třída `Problem` navíc umožňuje vygenerovat z těchto dvou seznamů lineární program ve formátu jazyka MathProg a v objektu knihovny SCPSolver `LinearProgram`. Před vygenerováním lineárního programu je však nutné ještě oba seznamy dopočítat, což je provedeno ve třídě `Analyzer` metodou `prepareDistribution(...)`.

Lineární program je poté vyřešen třídou `EMDSolver` z knihovny SCPSolver. Výsledná hodnota je posunuta do intervalu $[0; 1]$ pokud byla mimo něj a udává podobnost ve smyslu 0 nejpodobnější a 1 nejmenší podobnost. Celý proces je defaultně opakován $5\times$, neboť algoritmus projevuje poměrně vysoký rozptyl spočtených hodnot. Počet opakování je opět možné upravit v konfiguračním souboru. Výsledná hodnota je $1-p$, pro p spočtenou jako průměr všech opakování výpočtu podobnosti.

3.2.5 Analyzátor denní doby fotografie

K rozhodnutí denní doby fotografie v objektu `Photo` je třeba zavolat metodu `isNight(Photo)` ve třídě `Analyzer`.

Metoda nejdříve převede `Photo` na `BufferedImage`, poté z obrázku zvolí náhodný výběr několika pixelů, pro každý spočítá převod z RGB do černobílé barvy funkcí `argbToIntensity(int)` definované ve stejné třídě. Intenzita pixelu je reprezentovaná jako `double`, z černobílých intenzit pixelů náhodného vzorku se

spočte aritmetický průměr a podle výsledné hodnoty a prahu funkce vrátí predikovanou denní dobu fotografie. Velikost náhodného výběru a prahovou hodnotu lze upravit v konfiguračním souboru.

3.2.6 Přístup k databázi

Návrh přístupu k databázi odděluje nízkoúrovňové operace od objektových nadstaveb nad nimi. Třída `Database` obstarává připojení k databázi pomocí rozhraní `java.sql.Connection`, nabízí metody pro přímý přístup do databáze pomocí vytvořeného spojení a stará se vytváření celé databáze, tj. připravení všech databázových tabulek. Kontrolu existence tabulek provádí při zavedení do JVM, neboť metoda pro vytvoření tabulek je volána ve statickém bloku třídy. Tento okamžik nastává po prvním použití některé databázové operace, nebo příkazu `database` ve vlákně `ServerService`.

Třída `Database` je navržena pouze tak, aby nemusela přistupovat k objektům OHL. Jedinou výjimkou je metoda `getBestPlayers(...)`, jejíž úkolem je vyplnit pole `Player[]` několika nejlepšími hráči ve hře. Zde by nedávalo takový smysl informace vkládat do speciálních objektů a posílat je o úroveň výš, kde by z nich `Player` byl vytvořen.

Pro účely komunikace s databází na vyšší úrovni je určena třída `DatabaseService`, se kterou pracují především třídy `Requester`. Tato třída využívá kombinace metod z třídy `Database` a podle jejích reakcí vytváří objekty z knihovny OHL.

3.3 Klient

Popis některých implementačních detailů v klientské části aplikace.

3.3.1 Získávání cílů

Ve chvíli kdy uživatel zažádá o nový lov, jsou parametry *hráč*, *pozice* a *poloměr* odeslány pomocí požadavku `RadarSearchRequest`. Ten na serveru přeloží požadavek na dotaz na Google Radar Search (viz sekce 2.2.4), která vrátí seznam až 200 *Place ID*. To umožňuje rychlou odezvu GUI, neboť aplikace může po získání datově nenáročného seznamu přejít do prostředí nabídky, kde začnou postupně přibývat cíle.

Stahování cílů probíhá tak, že je provedena náhodná permutace seznamu *Place ID* a pro každé *ID* v permutovaném seznamu je nejdříve zkontrolováno, zdali není odpovídající cíl lokálně uložen z předchozích lovů, jinak je pro něj na server odeslán požadavek `FillPlacesRequest`, který vrátí cíl, nebo nevrátí nic pokud z místa s daným *Place ID* cíl vytvořit nejde (to je typicky z důvodu, že místo nemá žádnou (vhodnou) fotografii). Cíle jsou získávány postupně tak dlouho, dokud není seznam vyčerpán nebo bylo získáno 30 cílů.

3.3.2 Komunikace se serverem

Komunikace ze strany klienta probíhá tak, že klient vytvoří speciální vlákno, tzv. `AsyncTask`, které umožňuje plynulý běh grafického prostředí po dobu celé

komunikace se serverem. Během této doby je zobrazován dialog, který uživatele informuje o probíhající úloze.

Třída `ResponseTask` rozšiřuje `AsyncTask` a jejími úkoly jsou odeslání objektu `Request` na server OHS, vyčkání na výsledek, tj. `Response` nebo `OHException`, ukončení spojení, odstranění dialogu a informování o výsledku. O výsledku je informována třída, která implementuje rozhraní `OnResponseTaskCompleted`. Jeho jediná stejnojmenná metoda vrací původní `Request`, `Response`, `OHException` a `Object`, který může posloužit k identifikaci konkrétního požadavku.

Třída `Wizard` nabízí statické připravení mnoha různých `DialogFragment`ů, jeden z nich je připraven k použití pro standardní dialog komunikace. Pro vytvoření dialogu je nutné pouze vložit platný `Context`, který dialog hostuje.

Konkrétní příklad použití těchto tříd pro vytvoření požadavku s dialogem je pak:

```
Request request = new CustomRequest(player, 42);
DialogFragment dialog = Wizard.getServerCommunicationDialog(context);
ResponseTask task = new ResponseTask(dialog, this);
task.execute(request);
```

Třída odesílající požadavek poté implementuje `OnResponseTaskCompleted` například takto:

```
public void onResponseTaskCompleted(Request request, Response response,
                                    OHException ohex, Object data) {
    if (ohex != null) {
        /* Vyřeš konkrétní typ OHException */
        return;
    }
    if (response == null) {
        /* Server je nejspíš vytížen, nebo nedostupný */
        return;
    }
    /* Použij request, response a data */
}
```

3.3.3 Zacházení s pamětí

Herní objekt *cíl* má až 10 fotografií. Jelikož nabídka cílů obsahuje až 30 cílů a každá fotografie má v defaultním nastavení maximální rozměry 480×320 px, takové množství cílů by vyčerpalo až 180 MB paměti RAM, což je na současných mobilních zařízeních problém. Z tohoto důvodu je provedeno opatření, které tento problém redukuje na teoretické maximální nároky pouze 7,7 MB.

Získávání cílů se serveru OHS zajišťuje třída `TargetsManager`. Ta na server odesílá *Place ID* kandidátů na cíle, z výsledků filtruje místa, která jsou na objekt `Target` převedena konstruktorem `Target(Place, Bitmap)`, kde `Bitmap`a je ikona reprezentující cíl.

`Target` interně používá třídu `PhotosManager`, která fotografie cílů ukládá externě na úložiště telefonu. Na přední stranu karet v nabídce cílů je nutné vykreslovat vybranou fotografii, toho je docíleno tak, že tento náhled je jedinou

bitmapou, která se do objektu `Target` (a tedy i paměti RAM) skutečně ukládá. Využívá však faktu, že karty jsou čtvercové a typicky malých rozměrů, proto stačí ve velikosti 256×256 px. Je nutné ji přepočítat pouze pokud uživatel vybere jinou referenční fotografii pro daný cíl.

V této verzi je dosaženo velmi podobného výsledku s použitím minimálního množství paměti. Pokud je zapotřebí fotografie pro cíl načíst, je opět využita třída `PhotosManager`, konkrétně metoda `getPhotoOfTarget(String, int)`, kde `String` identifikuje cíl a `int` konkrétní fogografii cíle. Paměti nemusí být vždy dostupno pouze malé množství, proto `PhotosManager` navíc používá mechanismus cachování nejčastěji používaných fotografií pomocí třídy `LruCache`.

Toto řešení umožňuje plynulou práci s fotografiemi na zařízeních s dostatečným množstvím paměti RAM a pomalejší na reakce, nicméně funkční práci na zařízeních s výrazným omezením dostupné paměti RAM.

3.3.4 Hranová detekce

Algoritmus hranové detekce z části 2.3.1 je implementovaný v podtřídě `CountEdgesTask` třídy `Utils` a pro zajištění plynulého průběhu grafického vykreslování během jejího výpočtu je prováděna na pozadí pomocí `AsyncTask`. Ve svém vlákně pouze přímo volá metodu `sobel(Bitmap)`, která nejdříve provede rozmazání obrázku tak, že jej zmenší na poloviční rozměry a poté znovu zpátky zvětší na původní. Protože implementace konvoluce nevyužívá větu o konvoluci, která umožňuje výpočet konvoluce bez mnohonásobného navštívení stejných pixelů (za cenu výpočtu fourierovy transformace, což nemá pro malé obrázky význam), je pro hodnoty intenzit (černobílé hodnoty pixelu) využito dvourozměrné pole, které jednou spočtené hodnoty uchovává a urychluje tak celý výpočet.

Při výpočtu první fáze je zaznamenáván histogram všech hodnot intenzit konvoluce, ten je pak použitý k nalezení optimální prahové intenzity k odstranění nevýrazných neprůhledných pixelů. Algoritmus znovu projde celý obrázek a nastaví maximální průhlednost pixelům s podprahovou hodnotou.

Výpočet změny barvy pak obstarává ve třídě `CameraActivity` použitím třídy `ColorFilter` metoda `changeBitmapColor(Bitmap,.,int)`, kde `int` udává hodnotu složky hue barevného modelu HSV, kterou uživatel zadá posuvníkem `colorSeekBar`.

4. Uživatelská dokumentace

V této kapitole se správce serveru seznámí s možnostmi konfigurace a ovládání aplikace OHS, uživatel klientské části aplikace pak s instalací, rozhraním a ovládáním aplikace OHC v prostředí systému Android.

4.1 Knihovna

Knihovna OHL je nutná v případě kompilace OHS a OHC, pro úplnost tedy následuje popis kompilace této knihovny samotné.

4.1.1 Kompilace

Knihovna nemá žádné další závislosti, její kompilace je proto snadná. Příloha P.5 obsahuje projekt vývojového prostředí NetBeans IDE 8.1, které nabízí nejsnadnější variantu kompilace, např. klávesou F11. Je nutné kompilovat knihovnu pomocí JDK 1.7, vyšší verze totiž není na platformě Android podporovaná. Pro knihovnu je také možné vygenerovat dokumentaci nástrojem javadoc, v NetBeans snadno z menu `Run > Generate Javadoc`.

Zkompilovaná a zaarchivovaná knihovna OHL ve formátu `jar` je připravena v příloze P.6, její dokumentace pak v příloze P.7.

4.2 Server

Tato část popisuje detaily kompilace, spuštění, konfigurace a správy serveru pro zajištění jeho flexibility a maximálního výkonu v závislosti na aktuálních požadavcích klientů.

4.2.1 Kompilace a spuštění

Projekt OHS je závislý na několika knihovnách:

1. OHL, čili knihovna k této hře (její příprava je popsána v části 4.1.1)
2. JSON Simple (verze 1.1.1) [1]
3. Apache Derby (verze 10.12.1.1) [2]
4. SCPSolver (zapotřebí jsou všechny části, tj. `SCPSolver`, `GLPKSolverPack` a `LPSOLVESolverPack`) [3]

Příloha P.1 obsahuje projekt vývojového prostředí NetBeans IDE 8.1 a v podadresáři `/lib/` také všechny potřebné výše zmíněné knihovny. Projekt lze kompilovat s JDK 1.8 a výše [4] a nejjednodušší je varianta využívající NetBeans. K projektu

[1] Knihovnu JSON Simple lze najít na webu `http://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple`.

[2] Apache Derby je dostupné na `https://db.apache.org/derby/`.

[3] Domovské webové stránky SCPSolver jsou `http://scpsolver.org/`.

[4] Pod Windows funguje SCPSolver pouze po kompilaci 32-bitovou verzí JDK.

je možné vygenerovat dokumentaci, připravená je také v příloze P.3. Připravený soubor `jar` programu OHS je v příloze P.2.

Spuštění pak probíhá standardně pomocí odpovídající verze JRE, např.:

```
java -jar OHunterServer.jar
```

4.2.2 ServerService

Pro usnadnění obsluhy spuštěného serveru běží kromě fondu vláken obsluhujících klienty navíc speciální vlákno `ServerService`. Toto vlákno čte standardní vstup, vypisuje na standardní výstup a nabízí následující příkazy:

- `help` – vypisuje všechny správci dostupné příkazy a jejich krátký popis
- `state` – vypíše aktuální stav fondu vláken
- `database` – zkontroluje existenci a případně inicializuje databázi, tj. vytvoří neexistující tabulky, pro existující tabulky nevykoná nic. Databáze je defaultně vytvořena v domovském adresáři uživatele, např.: `/home/<user>/.oHunterDB/` nebo `C:\Users\<user>\.oHunterDB\`, cestu je možné specifikovat v konfiguračním souboru.
- `table show <tabulka>` – provede SQL dotaz do tabulky s daným jménem a vypíše její obsah
- `player remove <ID hráče>` – z tabulky HRAC odstraní záznam hráče s daným ID
- `player setscore <ID hráče> <skóre>` – nastaví skóre hráče s daným ID, pro aktualizaci údaje na klientovi je nutné provést odhlášení a příhlášení
- `player setname <ID hráče> <nové jméno>` – přenastaví přezdívku hráče s daným ID, pro uplatnění změny na klientovi je opět nutné se odhlásit a přihlásit, ovšem již s novým jménem
- `bests <počet>` – vypíše seznam nejlepších hráčů, jejich maximální počet je dán parametrem
- `config` – načte znovu obsah konfiguračního souboru, případně ho vytvoří či doplní chybějící hodnoty (více v sekci 4.2.3)
- `exit` – požádá správce fondu vláken o ukončení obsluhujících vláken a vyčká do splnění tohoto požadavku, poté uvolní systémové prostředky a ukončí server

Pro složitější SQL dotazy a zásahy do databáze viz část 4.2.4.

4.2.3 Konfigurace serveru

Pro usnadnění nastavení parametrů serveru je možné server konfigurovat úpravou hodnot v souboru `config.txt`, který je vytvořen s defaultními hodnotami vždy při jeho nepřítomnosti v adresáři, kde je server spuštěn.

Server při jeho čtení ignoruje řádky s bílými znaky a řádky začínající znakem #, který lze použít pro komentáře. Konfigurační řádka má pak formát:

```
komponenta1.komponenta2..komponentan = <číslo>
```

pro číselnou hodnotu a pro řetězec:

```
komponenta1.komponenta2..komponentam = "<řetězec>"
```

Konkrétní příklady číselného i řetězcového nastavení z defaultního konfiguračního souboru:

```
# The number of random samples to determine the daytime.
analyzer.daytime.random_photo_samples = 128

# The name of the database.
database.name = "oHunterDB"
```

Každé konfigurační řádce v defaultním vygenerovaném souboru předchází její komentář, celý defaultní soubor se nachází v příloze P.4.

Pro položku `server.address.ip` platí speciální pravidlo. Pokud je její hodnota prázdný řetězec `""`, bude správce při spouštění serveru požádán o vyplnění IP adresy, je-li však hodnota správně vyplněna, server ji po spuštění ihned využije bez dalších výzev.

Položku `game.google_api_key` je nutné nastavit tak, aby obsahovala Google API klíč (podrobněji popsaný v sekci 2.2.1). Po vygenerování konfiguračního souboru jej stačí vyplnit do prázdných uvozovek a server spustit znovu.

Konfigurační soubor je možné za běhu serveru načíst znovu (viz část 4.2.2). Pak je znovu kontrolováno, zdali je soubor přítomen, pokud ne, je vytvořen s defaultním nastavením. Jsou do něj také případně doplněny chybějící hodnoty. Načtení konfigurací za běhu má však omezení, ovlivnit lze pouze ty komponenty, které hodnoty pravidelně čtou. Mezi ně patří komponenty analýzy podobnosti a denní doby, všechny třídy `-Requester`. Bez nutnosti restartování serveru také funguje položka pro iniciální počet bodů nového uživatele.

4.2.4 Databáze

Pro obsluhu a manuální zásahy databáze je možné použít program SQuirreL[5] Pro připojení k databázi na serveru stačí standardní instalace bez pluginů, poté je nutné nastavit cestu k Derby Embedded JDBC Driveru a nakonec vytvořit alias k databázi. Je nutné aby byla databáze na serveru již vytvořena. Její vytvoření proběhne automaticky po první žádosti klienta, která k ní využívá přístup. Využít lze také příkaz `database` na standardní vstup spuštěného serveru.

[5]Domovská stránka programu SQuirrel: `http://squirrel-sql.sourceforge.net/`.

Přístupové jméno a heslo defaultně databáze nepoužívá, proto stačí vyplnit jen libovolný název aliasu a cestu k databázi. Server databázi standardně vytváří v domovském adresáři, URL tedy bude mít na unixových systémech tvar

```
jdbc:derby:/home/<user>/.oHunterDB/oHunterDB;create=false
```

na Windows pak

```
jdbc:derby:/C:/Users/<user>/.oHunterDB/oHunterDB;create=false
```

Cesta k databázi se však může lišit, neboť ji lze upravit v konfiguračním souboru serveru (podrobněji v 4.2.3).

Podrobný návod propojení SQuirreL s Apache Derby: `https://db.apache.org/derby/integrate/SQuirreL_Derby.html`.

4.3 Klient

Tato sekce je, s výjimkou podsekce kompilace, určená pro samotné hráče, tedy uživatele klientské části aplikace na systému Android. Obsahuje popis instalace a návod k ovládání.

4.3.1 Kompilace

Projekt OHC byl vyvíjen nástrojem Android Studio verze 2.1.1 a příloha P.8 celý projekt obsahuje. Projekt je závislý na knihovně OHL, jejíž příprava je popsaná v sekci 4.1.1. Stejně jako v OHS (viz sekci 2.2.1), je navíc nutné vyplnit vlastní klíč pro využití Google Maps Android API.[6] Kompilaci lze pak snadno provést například klávesovou zkratkou `Ctrl+F9`. Dokumentaci lze vygenerovat z menu: `Tools > Generate Javadoc`. Vygenerovaná dokumentace je připravená v příloze P.10.

4.3.2 Instalace & Hardware

Minimální systémové požadavky vyžadují Android API $\geq$ 19 (KitKat 4.4), fotoaparát, lokátor GPS pro přesné zaměření polohy a možnost připojení k Internetu. Doporučené množství paměti RAM pro běh této aplikace je 60MB, a alespoň 20MB interní paměti je nutných pro uchování dočasných souborů a *historie*.

Instalační soubor `ohunter.apk` je v příloze P.9. Tento soubor je nutné nejprve přesunout do úložiště telefonu, poté ho najít pomocí průzkumníka souborů v telefonu, spustit jej a postupovat dle instalačních pokynů. Po úspěšné instalaci je možné aplikaci ihned spustit.

Aplikace byla testována především na telefonu Samsung i9195 se systémem Android 5.1.1. Projekt využívá knihovny pro zpětnou kompatibilitu do verze API 7 (Eclair 2.1), nejnižší testovaná verze API byla 19 (KitKat 4.4.2) na tabletu

[6] Klíč se po vygenerování vyplní do souboru `AndroidManifest.xml`, položky `<meta-data android:name="com.google.android.geo.API_KEY"android:value=«klíč»"/>`.

Lenovo A3500FL. Celkové pokrytí zařízení se systémem Android je tedy teoreticky 75,6 % [7] (API $\geq$ 19).

4.3.3 Přihlášení a registrace do hry

Po prvním spuštění aplikace je zobrazeno okno s poli pro uživatelské jméno a heslo. Je nutné zvolit si takové jméno, které doposud nepoužívá jiný uživatel, tato informace je poskytnuta při pokusu o registraci duplicitního jména a dále heslo, které má alespoň 5 znaků. Po úspěšné registraci proběhne zároveň přihlášení a uživateli je umožněn přístup k hlavnímu menu hry.

Přihlášení do aplikace je platné až do okamžiku, kdy uživatel explicitně provede žádost o odhlášení, to lze provést z hlavního menu.

Přihlášení a registrace je umožněna pouze v případě, že je k dispozici připojení k serveru. Poslední použité jméno pro přihlášení bude na přihlašovací stránce při dalším přístupu předvyplněno. Náhledy příhlášení a registrace jsou na obrázku 4.1.

4.3.4 Hlavní menu

Stránka hlavního menu obsahuje jednak základní informace o přihlášeném hráči, tj. jeho jméno a počet bodů, dále čas do ukončení předchozího lovu a poté menu samotné.

Zahájení nového lovu

Spouští stránku pro nastavení parametrů nového lovu.

Předtím než samotný lov začne, je nutné specifikovat kruhovou oblast, kterou chce hráč prozkoumávat. To lze buď přímo zadáním GPS souřadnic, případně s pomocí interaktivní mapy lze střed oblasti na pozici mapy vybrat podržením prstu. Dále je nutné zvolit velikost oblasti, tj. její poloměr. Do pole pro poloměr není možné zadat hodnoty mimo povolený interval, hodnota je v jednotkách kilometrů. Denní dobu lze specifikovat z rozbalovacího seznamu.

Poslední použité hodnoty budou v příštím přístupu předvyplněné, je-li dostupná současná pozice, bude použita jako výchozí bod při prvním spuštění.

Pokračování ve hře

Dokud není předchozí lov ukončen, je možné se do něj vrátit tímto tlačítkem. V opačném případě není toto tlačítko přístupné.

Historie

Obsahuje *historii cílů* všech ukončených lovů, tj. přehled všech *zamknutých* cílů a *splněných* cílů včetně bodů získaných za jejich *splnění*. Na pozadí karet, podobných jako v nabídce cílů (viz část 4.3.5), je zobrazena jedna z pořízených fotografií. V pravé dolní části obrazovky je tlačítko, kterým lze odeslat všechny

[7] Podle současných údajů `https://developer.android.com/about/dashboards/index.html`.

zamknuté cíle k vyhodnocení a tím je *splnit*. Nový lov například nejde zahájit do té doby, kdy historie nějaké *zamknuté* cíle obsahuje.

Obsah historie je např. z paměťových důvodů možné vyprázdnit v nastavení hry (viz část 4.3.4).

Statistiky

Je-li dostupné připojení k serveru, horní část této stránky obsahuje tabulku s deseti nejlepšími hráči seřazenou od hráče s nejvíce, k hráči s nejméně body. Levá část každého řádku tabulky obsahuje přezdívku hráče, pravá pak jejich momentální skóre.

Pod tabulkou se nachází graf, který zobrazuje pomocí sloupců počet bodů získaných za podobnost a objevení cílů v daném lovu. Číslo lovu je pod každou dvojicí sloupců a je zvyšováno po každém spuštění nové hry, za kterou byly odečteny body, tj. v nabídce byly dostupné nějaké cíle.

Nápověda

Obsahuje stručný popis ovládání aplikace, text je provázen obrázky.

O aplikaci

Informace název aplikace a informace o používané verzi.

Nastavení

Umožňuje konfiguraci prostředí aplikace tak, jak uživateli vyhovuje. Například počet sloupců karet v nabídce, vymazání historie, resetování účtu, ovládání herního průvodce apod. Jednotlivé položky jsou stručně popsané v aplikaci.

Odhlášení

Slouží především k možnosti vybrat server k přihlášení, je možné přihlásit se i pod jiným uživatelským jménem, veškerá data lovů jsou však mezi uživateli sdílená, každý hráč by proto měl pro lov používat pouze vlastní telefon.

4.3.5 Nabídka cílů

Nabídka umožňuje přehledný seznam všech dostupných cílů, jejich stavů a základních informací o nich.

Jde o první ze tří hlavních součástí hry, kterou uživatel navštíví. Každý cíl je zde reprezentován kartou, která po vytvoření zobrazuje první z fotografií tohoto cíle, v pravém dolním rohu pak barvou označuje stav cíle a po jejím otočení tlačítkem rotace lze získat několik základních informací.

Po zahájení nového lovu v nabídce budou postupně přibývat cíle získané ze serveru. Po získání všech je 6 z nich ve stavu *přístupný*, tyto cíle budou umístněny na začátku celého seznamu a jejich barvou je šedá, ostatní jsou ve stavu *nepřístupný*, takové jsou označené barvu bílou, navíc je jejich fotografie na pozadí překryta bílým filtrem.

Pro akce *zpřístupnění*, *přijmutí*, *zamítnutí* jsou k dispozici tlačítka v dolní části nabídky, po řadě šedá, zelená a červená. Kartu cíle, u kterého se má změnit stav je nutné nejdříve označit dotykem, tlačítka pro možné stavy se poté zobrazí automaticky.

Opakovaný dotyk označené karty způsobí její odznačení, zobrazená tlačítka pak umožňují provádět akce vztahující se na všechny cíle v nabídce. Mezi nimi je tlačítko vyhodnocení všech *zamknutých* cílů, které mají barvu fialovou, po vyhodnocení je cíl *splněný* označen barvou modrou. Další tlačítko způsobuje setřídění nabídky podle stavů cílů a poslední způsobí rotaci všech karet.

Po navštívení přijmutého cíle je vytvořena notifikace odkazující do nabídky cílů. Navštívený, tj. *fotogenický* cíl má barvu žlutou a po jeho označení je možné přejít k focení, tj. fialovým tlačítkem s ikonou fotoaparátu.

Barva tlačítek tlačítek se řídí konvencí, že barva vždy odpovídá barvě stavu, na který tlačítko cíl mění. Barvy všech stavů cílů jsou vyznačeny na diagramu 1.2. Snímky obrazovky z aplikace a vysvětlivky jsou na obrázku 4.3. Na něm je také vidět dialog průvodce, který informuje hráče a snaží se ho učit pravidlům.

4.3.6 Focení cíle

Před focením je nutné vyčkat, než je referenční fotografie upravena hranovým detektorem. Poté je vykreslena na popředí náhledu fotoaparátu. Pro úpravu její výraznosti je možné posuvníky nastavit její barvu a průhlednost. Lze jí také rotovat dvěma přístupnými tlačítky o 90°.

V okolí referenční fotografie se může nacházet červeně vyznačené pole. Takto označená část bude z vyfocené fotografie oříznuta, pro docílení stejného poměru stran, jako u referenční fotografie. Číslo v pravé části obrazovky označuje počet fotografií, které je ještě možné vyfotit.

Pokud se uživatel náhle rozhodne cíl zatím nefotit, může se bez bodové penalizace vrátit zpět do nabídky cílů.

Po vyfocení a pokusu o opuštění focení je uživatel dotázán, zda-li chce pořízené fotografie použít ke *splnění* cíle. Má také možnost vrátit se a vyfotit případně další fotografii, pokud svůj limit nevyčerpal. V případě, že fotografie použít nechce, vrací se bez bodové penalizace do nabídky cílů. Pokud fotografie použít chce, vrací se také do nabídky cílů, stav vyfoceného cíle zde však bude změněn na *zamknutý*, bude tedy možné odeslat ho k vyhodnocení a tím ho *splnit*.

4.3.7 Detaily cíle

Již v průběhu stahování dostupných cílů v nabídce je možné prohlížet si detaily na stránce detailů cíle. Tato stránka je z nabídky cílů dostupná dvěma způsoby, buď vybráním z horního menu, nebo odsunutím stránky nabídky doleva.

Kromě všech dostupných informací o cíli je na této stránce dále dostupná galerie fotografií tohoto cíle. Je možné prohlížet si dostupné fotografie posuvníkem v dolní části stránky, fotografie, kterou si uživatel přeje později fotit je nutné ponechat vybranou. Tato fotografie bude také zobrazena na stránce nabídky na pozadí odpovídající karty cíle.

4.3.8 Interaktivní mapa

Třetí a poslední z hlavních součástí je stránka umožňující jednoduše sledovat pozice cílů i uživatele na mapě. Je možné se k ní dostat opět vybráním z menu nebo odsunutím detailů cíle doleva.

Tato mapa slouží dále k jednoduššímu plánování trasy. Pro všechny *přijmuté* cíle je zobrazena také jejich *aktivní zóna*. Barva značky cíle na mapě označuje jeho stav tak, jako v nabídce.

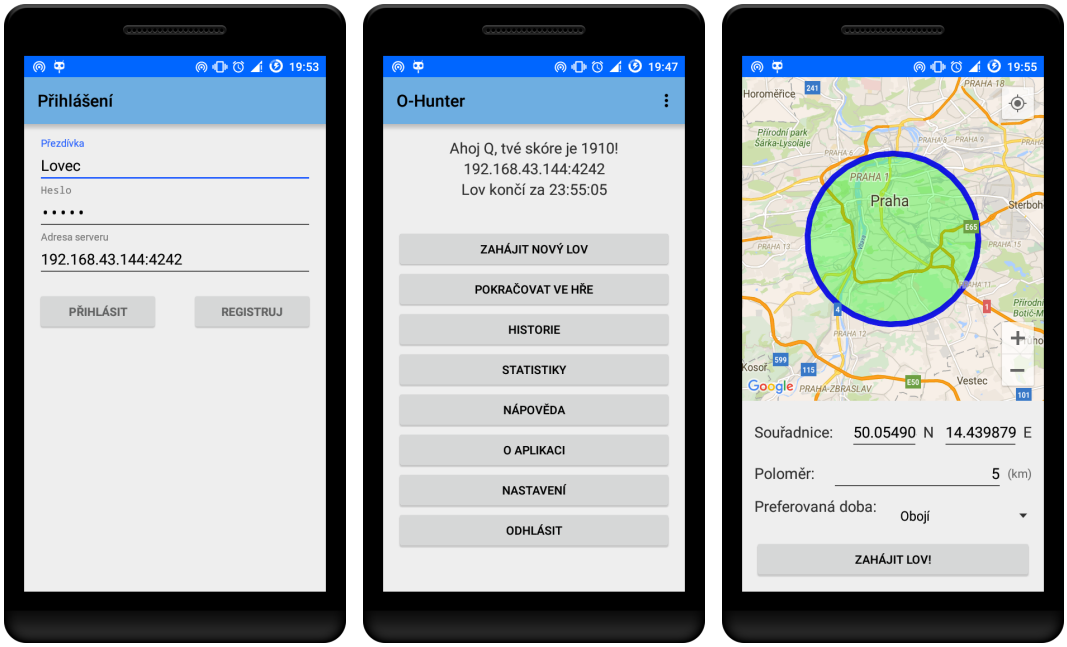
Ve spodní části stránky jsou zobrazeny číselné údaje o aktuální GPS pozici uživatele, pozici cíle vybraného v nabídce a vzdálenosti k němu. Snímek obrazovky reálného rozložení je na obrázku 4.2.

4.3.9 Offline režim

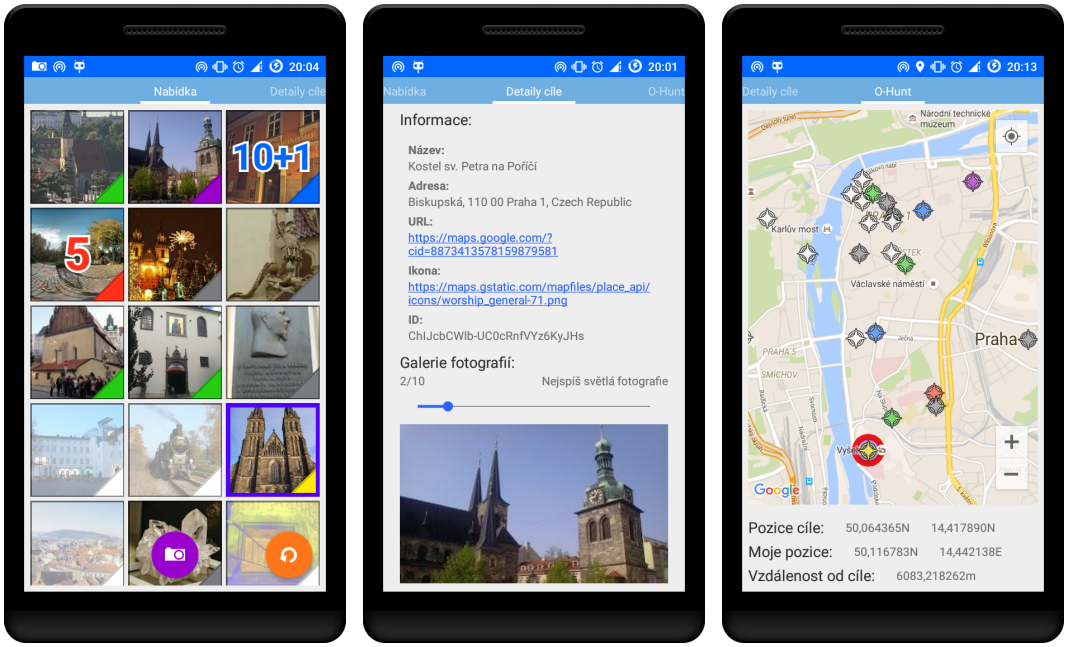
Hra je konstruována tak, aby nebylo nutné využívat připojení k Internetu po celou dobu lovu. Toto připojení je však nezbytné v případech:

- přihlášení a registrace
- zahájení nového lovu
- zakoupení *zpřístupnění* cíle a *zamítnutí* cíle
- výpočet podobnosti fotografií a *splnění* cíle

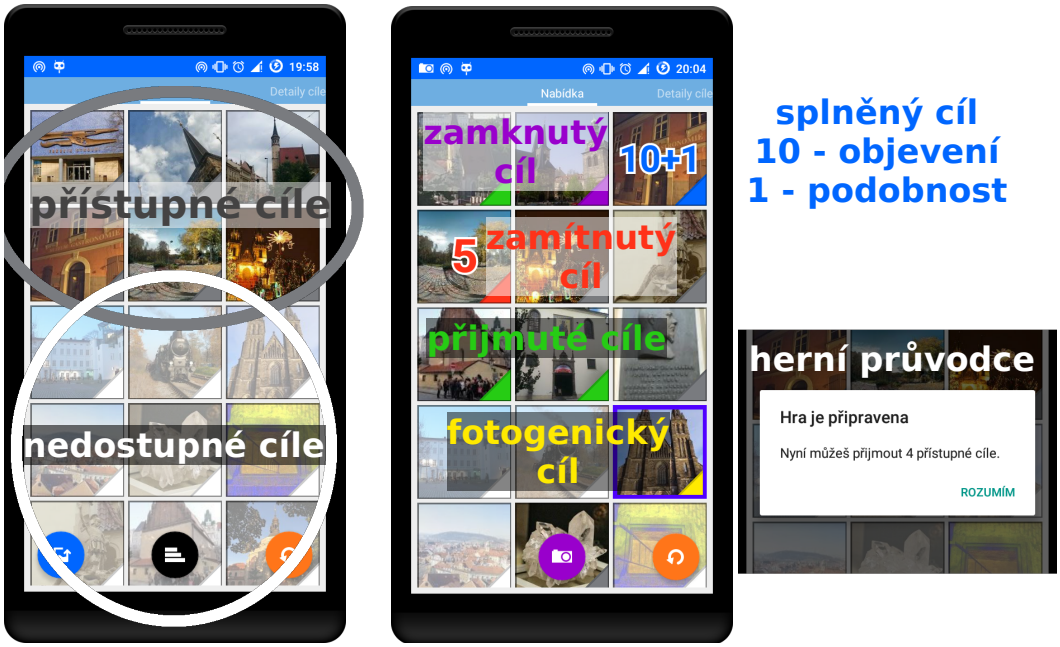
Pro nejčastější a datově náročný případ, tedy výpočet podobnosti a *splnění* cíle, je uživateli umožněno vyfocený cíl uložit, a požadavek o vyhodnocení odeslat až při opětovném připojení k Internetu.



Obrázek 4.1: Přihlášení a registrace do hry, hlavní menu a obrazovka přípravy nového lovu.



Obrázek 4.2: Nabídka cílů, detaily pro vybraný cíl a interaktivní mapa.



Obrázek 4.3: Vysvětlivky nabídky cílů a ukázka dialogu průvodce.

Závěr

Výsledkem této bakalářské práce je hra O-Hunter, která může svým uživatelům poskytnout zajímavý přístup k prozkoumávání neznámých oblastí kompetitivním způsobem.

Podařilo se vytvořit pravidla, která umožňují motivovat uživatele pro další hraní, srovnání hráčů mezi sebou, omezují hráče v určitých situacích pro rozložení zátěže na Google Places API a zaručují pokračování hry i v případě, že se uživateli nějaký lov nevydaří. Některá pravidla navíc může správce nástroji OHS i během hry upravit.

Použitá technika porovnávání obrázků v reálném použití ve hře nevykazuje příliš dobré výsledky, na to má však pravděpodobně vliv omezení rozměrů porovnávaných obrázků (které je nutné z hlediska urychlení výpočtu, paměťových nároků klienta a snížení množství dat komunikace) a použitá segmentační technika. Lepší segmentační algoritmus by mohl například vybírat souvislejší oblasti, které by obrázek mohly identifikovat přesněji. Jednotlivé charakterizační hodnoty segmentů by mohly mít váhy, které by šly nastavit například s využitím genetických algoritmů. Před samotným provedením výpočtu by také mohla být aplikována metoda registrace obrazu, aby se obrázky skutečně co nejpřesněji překrývaly.

Metoda rozpoznání denní doby se přes svou jednoduchost ukázala jako velmi vhodná, celý experiment včetně použitých obrázků a Matlab skriptů je v příloze P.11. Hranová detekce referenční fotografie je velmi nápomocná při samotném hraní hry a umožňuje najít správné místo k vyfocení cíle.

Google Places API nabízí velké množství použitelných míst včetně až deseti fotografií pro ně a stal se tak pro tuto aplikaci velmi vhodnou volbou. Nevýhodou jsou občasné odmítnutí připojení, omezení maximálního počtu dotazů na určitou dobu a malé množství informací o místě. Jako možné vylepšení by mohlo být vhodné využít další prostředky k získání dodatečných informací o cíli, které by uživatele mohly zajímat. Jelikož toto API také nabízí přidávání vlastních objektů, další možností by bylo přidat například „zkušeným“ hráčům přístup k další části aplikace, která by takové přidávání obstarávala.

Výsledná hra O-Hunter je rozdělená do tří částí – knihovna, server a klient, které problémy projektu rozdělují do ucelených fragmentů a ulehčují tak způsob kontroly nad nimi.

Seznam použité literatury

Chen, Y. (2011). *Algorithmic Approach to Differentiating between Day-Time and Night-Time Photographs.* `http://web.mit.edu/rsi/2012/minisubmit/yimochen/main.pdf.gz` [Online; přístupné 3. 5. 2016].

Gamma, E., Helm, R., Johnson, R. a Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* First Edition. Addison-Wesley Professional, Boston. ISBN 978-0201633610.

MacKay, D. J. C. (2002). *Information Theory, Inference & Learning Algorithms*, chapter 20.1, pages 285–287. Cambridge University Press, New York, NY, USA. ISBN 0521642981.

Pratt, W. K. (2007). *Digital Image Processing: PIKS Scientific Inside.* John Wiley and Sons, Inc., New Jersey, USA. ISBN 978-0-471-76777-0. URL `http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=2007dip..book.....P`.

Qin Lv, Moses Charikar, K. L. (2004). Image similarity search with compact data structures. In *13th ACM Conference on Information and Knowledge Management (CIKM)*, pages 208–217. `http://www.cs.princeton.edu/courses/archive/spr05/cos598E/bib/p208-lv.pdf` [Online; přístupné 1. 5. 2016].

Seznam obrázků

Seznam použitých zkratek

Zkratka	Význam
OHL	O-Hunter Library, knihovna pro zajištění komunikace (Java 7)
OHS	O-Hunter Server, server zajišťující náročné výpočty a databázi pro obsluhu klientů (Java 8)
OHC	O-Hunter Client, aplikace, kterou využívají hráči na svých telefonech (Android API $\geq$ 19 (KitKat 4.4))
JVM	Java Virtual Machine, interpreter bytekódu Javy

Přílohy

P.1 Projekt OHS v NetBeans IDE 8.1 obsahující zdrojové soubory, v adresáři `/ohs/project/`

P.2 Zkompilovaný OHS v archivu jar, v souboru `/ohs/OHunterServer.jar`

P.3 Dokumentace serveru OHS, v adresáři `/ohs/javadoc/`

P.4 Defaultní konfigurační soubor serveru, v souboru `/ohs/config.txt`

P.5 Projekt OHL v NetBeans IDE 8.1 obsahující zdrojové soubory, v adresáři `/ohl/project/`

P.6 Zkompilovaná OHL v archivu jar, v souboru `/ohl/OHunterLibrary.jar`

P.7 Dokumentace knihovny OHL, v adresáři `/ohl/javadoc/`

P.8 Projekt OHC v Android Studiu 2.1.1 obsahující zdrojové soubory, v adresáři `/ohc/project/`

P.9 Instalační soubor OHC pro Android, v souboru `/ohc/ohunter.apk`

P.10 Dokumentace knihovny OHC, v adresáři `/ohc/javadoc/`

P.11 Experiment výkonu detekce denní doby, v adresáři `/experiment/nocden/`

P.12 Obrázky, skripty a výsledky experimentu porovnávání fotografií, v adresáři `/experiment/podobnost/`