VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PODPORA HRANÍ DESKOVÉ HRY MLÝNY MOBILNÍ APLIKACÍ

MOBIL APPLICATION SUPPORT FOR NINE MEN'S MORRIS PLAYING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — DANIEL KOLÍNEK
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.
SUPERVISOR

BRNO 2019

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

22062

Student: **Kolínek Daniel**
Program: Informační technologie
Název: **Podpora hraní deskové hry Mlýny mobilní aplikací**
Mobil Application Support for Nine Men's Morris Playing
Kategorie: Umělá inteligence
Zadání:

1. Seznamte se s pravidly deskové hry Mlýny a dále s prostředími pro vytváření aplikací pro operační systém Android.
2. Navrhněte a implementujte algoritmus, který umožní volbu optimálního tahu v této hře při známém rozložení kamenů.
3. Pro systém Android navrhněte aplikaci, která na základě pořízených snímků herního pole zjistí aktuální stav hry.
4. Rozšiřte aplikaci tak, aby na základě zjištěného stavu navrhne optimální tah.
5. Ověřte fungování této hry s reálnými hráči a diskutujte použitelnost aplikace z hlediska její funkčnosti i z hlediska uživatelské přívětivosti.

Literatura:

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

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Cílem této práce je vytvořit aplikaci pro mobilní zařízení řešící úlohou určení nejlepšího tahu ve hře Mlýny ze snímku kamery chytrého zařízení. Úloha je rozdělena na podproblémy detekce pozice a následného určení nejlepšího tahu. Rozpoznání pozice je řešeno použitím detekce hran, hledání kružnic s využitím Houghovy transformace a následné detekce barvy v nalezených kružnicích. Nalezení nejlepšího tahu je řešeno vlastním ohodnocením pozice a prohledávání stavového prostoru pomocí algoritmu Alfa-Beta. S využitím knihovny OpenCV a vývojového prostředí Android Studio byla vytvořena vzorová aplikace spustitelná pod systémem Android verze 5 a vyšší. Vzorová aplikace řeší obě úlohy.

Abstract

The aim of this work is to create an application for mobile device solving the task of determination the best move in game Nine men's morris from a camera snapshot taken on smart device. The task is divided into the problem of position detection and determination of the best move. Position recognition is solved by using edge detection, finding circles using Hough transform and subsequent color detection in found circles. Finding the best move is solved by own position evaluation and state space search using the Alpha-Beta algorithm. Using the OpenCV library and the Android Studio development environment, a sample application executable under Android version 5 and higher was created. The sample application solves both tasks.

Klíčová slova

Hra Mlýny, Alfa-Beta, Zpracování obrazu, Houghova transformace, Konvoluce, RGB, CIELAB, CLAHE, CIEDE2000, Android, OpenCV

Keywords

Nine men's morris, Alpha-Beta, Image processing, Hough transform, Convolution, RGB, CIELAB, CLAHE, CIEDE2000, Android, OpenCV

Citace

KOLÍNEK, Daniel. *Podpora hraní deskové hry Mlýny mobilní aplikací.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František Zbořil, Ph.D.

Podpora hraní deskové hry Mlýny mobilní aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Ing. Františka Zbořila, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Daniel Kolínek
13. května 2019

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce panu Doc. Ing. Františku Zbořilovi, Ph.D za pomoc a cenné rady při konzultacích.

Obsah

1 Úvod **3**

2 Teoretická část **4**
2.1 Hra mlýny 4
2.2 Teorie kombinatorických her 5
2.3 Počítačové vidění 8
2.4 Obrazové filtry 11
2.5 Barevný model 14
2.6 Delta E 15

3 Návrh **18**
3.1 Ohodnocení pozice 18
3.2 Detekce pozice 20
3.3 Návrh GUI 21

4 Implementace **25**
4.1 Použité nástroje pro vývoj 25
4.2 Android 25
4.3 Detekce kružnic 28
4.4 Přehled tříd (Hra Mlýny) 29
4.5 Přehled tříd (detekce pozice) 30
4.6 Problémy při implementaci 31

5 Testování **32**
5.1 Testování určení nejlepšího tahu 32
5.2 Testování detekce pozice 34

6 Závěr **37**

Literatura **38**

A Obsah přiloženého paměťového média **40**

Kapitola 1

Úvod

Ačkoli se v dnešní době hledí u her na grafické zpracování, velikou roli hrají i algoritmy umělé inteligence. Bez algoritmů umělé inteligence se počítačové hry neobejdou a jsou jejich součástí již řadu let. Ať se již jedná o určení nejlepšího tahu u deskových her, či postavy, která nám pomáhá s průchodem hry, jako je tomu například u hry God of War. Je také jasné, že čím bude umělá inteligence lépe propracována, tím bude hra více zábavná.

Tato práce obsahuje popis jednoho z možných způsobů, jak nalézt optimální možný tah v deskové hře Mlýn. Pro hledání optimálního tahu byl použit vyhledávací algoritmus Alfa-Beta, který s pomocí vyhodnocovací funkce vrací optimální možný nalezený tah, který byl nalezen pro danou hloubku vyhledávání.

Cílem této práce není pouze nalezení optimálního tahu, jak by se mohlo z názvu zdát, ale také detekce pozice pomocí kamery. Detekce objektů je v dnešní době jedno z nejvíce řešených témat v oblasti IT. Dnes již nepomáhá jen firmám při výrobě, či policii pro detekci přestupků na silnici a následné naskenování SPZ. Například nám tato vymoženost pomáhá při řízení, kdy je možné detekovat osobu, která vběhne do vozovky a auto samo zastaví.

Jelikož jsou důležitá pro hru pravidla, jsou v teoretické části popsána. Vzorová aplikace je vyvinuta pro mobilní zařízení běžící na systému Android. Pro tento operační systém je poskytnuto vývojové prostředí Android Studio do kterého jde naimportovat knihovna OpenCV, proto je část věnována i jim.

Po teoretické části následuje návrh kde je popsána hodnotící funkce. Dále se v této kapitole dozvíte, jak je možné pomocí kamery detekovat daný stav. Po návrhu je zmíněna výsledná implementace aplikace s problémy, které se při ní vyskytly. V části implementace nechybí ani stručný přehled tříd, které jsou implementovány.

Poslední částí je testování, kde je možné vidět úspěšnost zvolené hodnotící funkce v různých úrovních zanoření a popis kdy docházelo k největším chybám při detekci pozice. Na základě celé práce a následného testování je sepsán závěr.

Kapitola 2

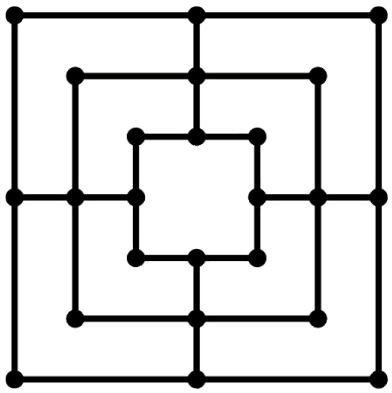
Teoretická část

V této kapitole jsou probrána základní témata potřebná pro uvedení do problematiky, která tvoří popis samotné hry s pravidly, algoritmus Alfa-Beta pro hledání optimálního tahu, nahlédnutí do rozpoznávání obrazu, malá část je také věnována samotnému prostředí, ve kterém je vzorová aplikace vytvořena i s knihovnou OpenCV.

2.1 Hra mlýny

Jedná se o jednu z nejstarších strategických her, zmínky o ní jsou již v dávné historii. Nejstarší herní deska byla nalezena vyryta v Egyptském chrámu, její původ je datován do 14. století před naším letopočtem. Jedná se o velice podobnou hru Morabaraba, Three Men's Morris, nebo Six Men's Morris, ze kterých pravděpodobně vznikla.

Hru hrají dva hráči. Hráč má na začátku hry 9 hracích kamenů. Herní deska je tvořena 24 body, po kterých je možné kameny rozmístit, nebo po spojnicích táhnout. [12]



Obrázek 2.1: Hrací deska

Hra se dělí na 3 fáze:

1. Rozmístění kamenů
2. Přesouvání kamenů
3. Skládání kamenů

Rozmístění kamenů

Hráči se v tazích střídají. Na začátku hry hráči pokládají kameny na volná pole. Pokud se hráči podaří položit 3 kameny svojí barvy do jedné linie, uzavřel mlýn. Pokud hráč právě uzavřel mlýn, odstraňuje protivníkovi jeden kámen. Smí sebrat pouze kámen, který neuzavírá mlýn. Pokud všechny kameny protihráče uzavírají mlýn, odstraňuje libovolný kámen uzavírající mlýn.

Přesouvání kamenů

Hráč na tahu posune kamenem své barvy na volné sousední pole. Pokud je hráč na tahu a nezbývá mu žádné prázdné sousední pole kam by mohl táhnout, hráč prohrává. Při uzavření mlýnu se opakuje situace s odebráním kamene. Je možné se z mlýnu odsunout a v příštím tahu mlýn opětovně uzavřít. Hráč, kterému zůstávají pouze tři kameny, se dostává do fáze skákání kamenů.

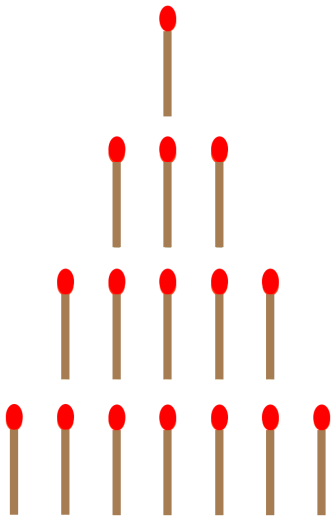
Skákání kamenů

Hráč, který má pouze tři kameny, může skákat na jakékoli volné místo na šachovnici. Při uzavření mlýnu se chová jako v předešlých fázích. [11]

2.2 Teorie kombinatorických her

Kombinatorické hry jsou hry pro dva hráče, kde o konci hry nerozhoduje náhoda. Mohou skončit pouze třemi možnými způsoby (výhrou, prohrou a remízou) a jsou určeny počtem možných stavů / operací, do kterých se počítá i počáteční. Hráči se obvykle v tazích střídají. Věc, kterou musí splňovat kombinatorická hra je, že má konce, což znamená, že nesmí dojít k nekonečnému opakování tahů. Proti nekonečnému opakování tahů je určeno vždy nějaké pravidlo, jako například u šachu, kde existuje pravidlo 50 tahů. Díky němuž je hra ustanovena remízou, pokud v posledních 50 tazích nebylo taženo pěšcem, či nedošlo k sebrání figury.

Kombinatorické hry se dělí na „nestranné“ a „partyzánské“ hry. Rozdíl mezi nimi je, že u nestranných v jakékoli pozici mají hráči možnost stejných tahů. Příkladem takové hry může být Nim, kde dva hráči střídavě odebírají sirky z několika hromádek. V jednom tahu je možno odebrat z libovolné hromádky libovolný počet sirek. Ten který již nemůže tah uskutečnit prohrává.



Obrázek 2.2: NIM (inspirováno oporou IZU [19])

Partyzánské hry jsou hry, u nichž nemají v každé pozici možnost stejného tahu, jako oponent, příkladem může být právě hra Mlýn.[11]

Algoritmus Minimax

Minimax je druh algoritmu pracujícího na zpětném vyhledávání. Je používán při rozhodování v teorii her pro určení optimálního tahu.

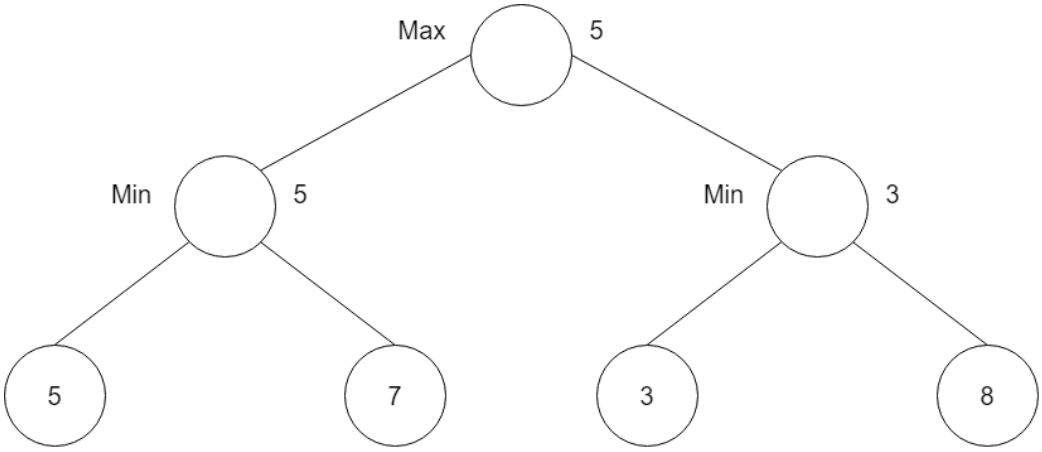
Každý stav hry je ohodnocen pomocí hodnotící funkce. Kladné hodnoty znamenají příznivý stav pro hráče na tahu (hráč Max), záporné potom příznivý stav pro protihráče (hráč Min). Vítězství je poté hodnoceno maximální a prohra minimální hodnotou.

Minimax tedy pro hráče Max vybírá stavy s největším ohodnocením a pro hráče Min naopak s nejmenším. Řešení je poté určení nejvýhodnějšího tahu pro hráče Max.

Algoritmus Minimax je rekurzivní, první spuštění se provádí pro hráče Max. Vstupní parametry algoritmu jsou stav hry (X), maximální hloubka prohledávání a informaci o hráči, pro kterého je vyhledávání spuštěno (Max nebo Min). Algoritmus vrací ohodnocení aktuálního stavu a tah, který k tomuto ohodnocení vede.

Popis algoritmu:

- Pokud je uzel X konečným stavem hry, nebo uzlem s maximální hloubkou vrať ohodnocení tohoto uzlu.
- Pokud je na tahu hráč Max, postupně pro všechny jeho možné tahy volej Minimax pro hráče Min, vrať maximální nalezenou hodnotu a tah, který vede k bezprostřednímu následníkovi.
- Pokud je na tahu hráč Min, tak postupně pro všechny možné tahy volej Minimax pro hráče Max a vrať minimální z navrácených hodnot.[19]



Obrázek 2.3: Minimax

Algoritmus Minimax je neefektivní, prochází všechny listy, i když některé již řešit nemusí. Podíváme-li se na obrázek 2.3 je zřejmé, že Minimax projde i poslední list s číslem 8, ačkoli je jasné, že číslo 3 je menší než dříve nalezené maximum 5 a funkce Min již větší číslo nevrátí. I na tak malém stromu prochází zbytečně jeden stav. Právě tento problém řeší algoritmus Alfa-Beta.

Algoritmus Alfa-Beta

Vychází z Minimax algoritmu a spíše jej optimalizuje, než se pokouší o nové techniky. Díky optimalizaci redukuje čas výpočtu o znatelnou část. Díky čemuž umožňuje průchod stromem do větší hloubky. Ořezává větve stromu, které již není potřeba prohledávat, jelikož už existuje lepší tah. Název Alfa-Beta získal díky dvěma parametrům (alfa, beta), které jsou připojeny k parametrům algoritmu Minimax.

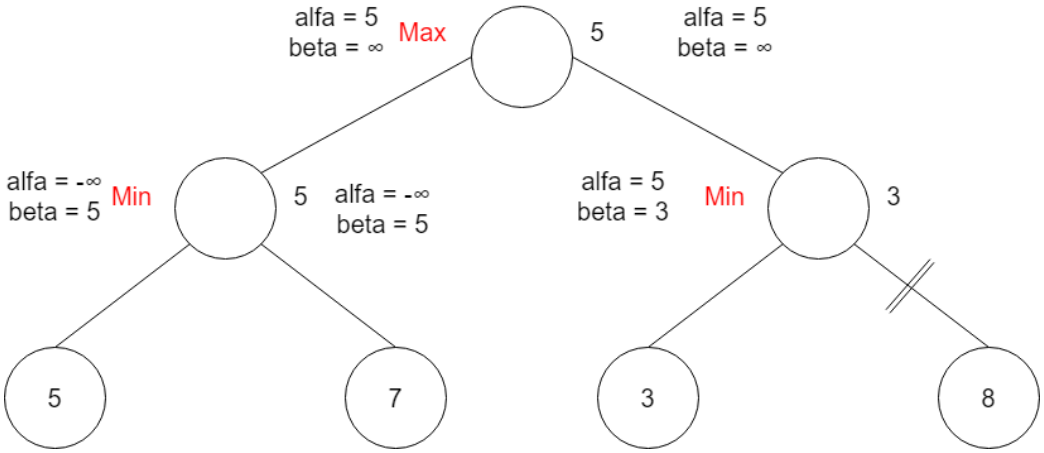
Parametr alfa obsahuje dosavadní největší nalezenou hodnotu pro aktuální, či vyšší úroveň stromu. Beta poté obsahuje dosavadní nejmenší nalezenou hodnotu pro aktuální, či vyšší úroveň stromu. Při prvním volání se alfa nastaví na $-\infty$ a beta na ∞.

Popis algoritmu:

- Pokud je uzel X konečným stavem hry, nebo uzlem v maximální hloubce, vrať ohodnocení uzlu
- Pokud je na tahu hráč Max:
  - pokud je alfa >= beta, nebo nemá uzel X žádného dalšího bezprostředního následníka, vrať aktuální hodnotu alfa a tah, který vede k nejlépe ohodnocenému bezprostřednímu následníku(pokud je více možných tahů se stejným ohodnocením jako je alfa, alfa je roven prvnímu takhle ohodnocenému tahu).
  - jinak postupně pro dalšího bezprostředního následníka uzlu X volej Alfa-Beta s aktuálními hodnotami parametrů alfa, beta. Po každém tahu kontroluj hodnotu alfa a uchovávej v ní maximum z aktuální a navrácené hodnoty
- Pokud je na tahu hráč Min:
  - pokud je alfa >= beta, nebo nemá uzel X žádného dalšího bezprostředního následníka, vrať aktuální hodnotu beta a tah, který vede k nejlépe ohodnocenému

bezprostřednímu následníku(pokud je více možných tahů se stejným ohodnocením jako je beta, beta je rovna prvnímu takhle ohodnocenému tahu).

– jinak postupně pro dalšího bezprostředního následníka uzlu X volej Alfa-Beta s aktuálními hodnotami parametrů alfa, beta. Po každém tahu kontroluj hodnotu beta a uchovávej v ní minimum z aktuální a navrácené hodnoty[19]



Obrázek 2.4: Alfa-Beta

Na vzorovém obrázku 2.4 je vidět, jak Alfa-Beta plní a využívá parametry alfa, beta, kde v posledním kroku (algoritmus jde postupně zleva doprava) se do parametru beta nahraje minimum 3, porovná se s alfa, který je roven 5, což znamená, že alfa je větší než beta, další větve se neprochází a vrací se hodnota 3.

2.3 Počítačové vidění

Počítačové vidění má dva cíle. Z pohledu biologie, jde o to přijít na výpočetní modely, které by byly podobné lidskému zrakovému vjemu. Z technického hlediska se jedná o autonomní systém, který vykonává určité úkoly, které umí lidské zrakové ústrojí. Mnoho těchto úkolů vyžaduje extrakci 3D informace z poskytnutých 2D dat, která mohou být dodávána například pomocí kamer.

Oba tyto cíle jsou samozřejmě úzce spjaty. Inženýři, kteří navrhují systémy počítačového vidění se často inspirují vlastnostmi lidského zraku. Naopak algoritmy počítačového vidění mohou ukázat, jak lidský zrak pracuje.

Prahování

Vstupem do operace prahování je typicky barevný obraz nebo obraz ve stupních šedi. V nejjednodušším provedení je výstupem binární obraz představující segmentaci. Černé pixely odpovídají pozadí a bílé pixely odpovídají popředí (nebo naopak). V jednoduchých implementacích je segmentace určena jedním parametrem známým jako prahová hodnota intenzity. V jednom průchodu je každý pixel na snímku porovnán s touto prahovou hodnotou. Pokud je intenzita pixelu vyšší než prahová hodnota, pixel je ve výstupu nastaven na bílou. Pokud je menší než prahová hodnota, je nastavena na černou.

U sofistikovanějších implementací může být specifikováno více prahových hodnot, takže pásmo hodnot intenzity může být nastaveno na bílou, zatímco vše ostatní je nastaveno na černou. Pro barevné nebo multi-spektrální obrazy může být možné nastavit různé prahové hodnoty pro každý barevný kanál, a tak vybrat pouze ty pixely v rámci specifického kvádru v prostoru RGB. Další obyčejná varianta je nastavit k černé všechny ty pixely odpovídající pozadí, ale nechat pixely popředí na jejich originální barvě / intenzitě, takže informace nejsou ztraceny.

Ne všechny snímky mohou být přehledně rozděleny do popředí a pozadí pomocí jednoduchého prahování. To, zda může být obraz správně segmentován tímto způsobem, lze určit pohledem na histogram intenzity obrazu. Budeme uvažovat pouze histogram ve stupních šedi.

Pokud je možné oddělit popředí obrazu na základě intenzity pixelů, pak musí být intenzita pixelů v objektech popředí výrazně odlišná od intenzity pixelů v pozadí. V tomto případě očekáváme, že se v histogramu objeví výrazný vrchol odpovídající objektům v popředí, takže prahy mohou být vybrány tak, aby byl tento vrchol odpovídajícím způsobem izolován. Pokud takový vrchol neexistuje, pak je nepravděpodobné, že jednoduché prahování povede k dobré segmentaci. V takovém případě může být lepším řešením adaptivní prahování. V adaptivním prahování algoritmus určuje práh pro pixel založený na malé oblasti kolem něj. Čímž dostáváme tedy různé prahové hodnoty pro různé oblasti stejného obrazu, které poskytují lepší výsledky pro obrazy s různým osvětlením.[15]

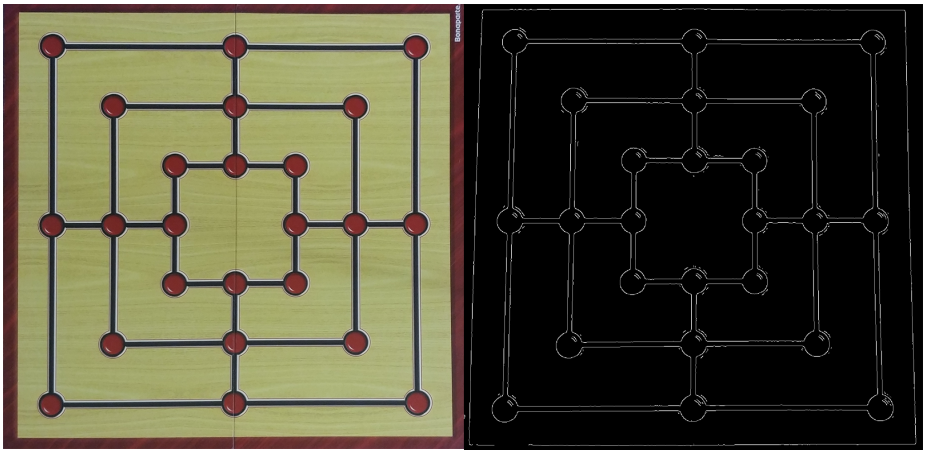
Detekce hran

Jedná se o postup v digitálním zpracování obrazu, který slouží k nalezení oblastí pixelů, ve kterých se výrazně mění jas.

Každý nesymetrický obraz obsahuje kromě skutečných hran i nenulovou míru šumu, který se lokálně chová stejně jako hrana. Samostatné detektory hran jsou sestaveny tak, aby byly citlivé na libovolnou skokovou změnu jasové funkce obrazu, což zahrnuje i změny způsobené šumem. Proto je šum potřeba redukovat pomocí vhodného filtru, k čemuž lze využít dříve zmíněné Gaussovo rozostření.

Po redukci šumu lze aplikovat hranový filtr. Existuje již několik standardizovaných hranových filtrů (Robertsův, Prewittové, Sobelův, Robinsonův, ...). Každý detektor má své nevýhody, proto nemůžeme určit univerzální nejlepší filtr. Po použití hranových filtrů dostaneme signál (snímek) obsahující všechny hrany obrazu. Jde-li nám o kvalitu získaných hran, musíme se tímto problémem zabývat. V nejjednodušším případě lze výsledný signál konvoluce prahovat a získat tak binární masku indikující místa hran. Nestačí-li pouhé prahování, je nutné se zabývat kvalitou hrany hlouběji.

Prahování s hysterezí využívá Cannyho hranový detektor. Ten po filtraci obrazu dvourozměrným Gaussovým filtrem nalezne pomocí standardních detektorů velikosti hran ve všech čtyrech základních směrech. Poté je provedena operace nemaximální suprese tzn. potlačení okolí lokálních maxim a následně je aplikováno prahování s hysterezí. Tento mechanizmus pracuje tak, že jsou libovolnou metodou určeny dva prahy (vyšší T_H a nižší T_L). Hodnoty vyšší, než práh T_H jsou uznány jako hrany, již při první iteraci a hodnoty nižší jak T_L jsou zamítnuty. Díky čemuž hysterezní řeší pouze hrany v intervalu $<T_L;T_H>$. Tyto hrany jsou postupně uznávány pouze pokud sousedí s již nalezenými hranami. Algoritmus se vykonává dokud se obraz hran nezmění za poslední dvě iterace. [13] Na obrázku 2.5 můžeme vidět jak vypadá výsledná detekce hran (vlevo originální obraz, vpravo obraz obsahující detekované hrany).



Obrázek 2.5: Detekce hran

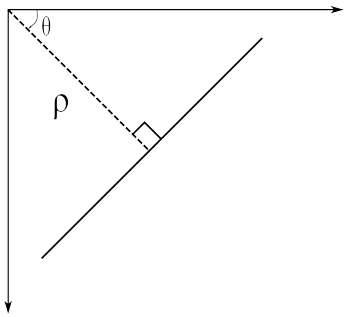
Houghova transformace

Jedná se o populární techniku pro detekci jakéhokoli tvaru, který lze popsat matematicky. Jsme díky této technice schopni detekovat i necelé, či nějakým způsobem trochu nepřesné tvary. Základním detekovatelným tvarem je přímka a na ní bude i Houghova transformace vysvětlena.[1]

Houghova transformace přímky

Přímka lze vyjádřit vzorcem prostoru parametrů $\rho = x * cos\theta + y * sin\theta$, kde ρ je délka kolmé úsečky k přímce, začínající ve středu souřadnicového systému a končící na přímce a θ je úhel, který svírá tato kolmá úsečka a osa X měřený proti směru hodinových ručiček (směr se liší dle reprezentace souřadnicovým systémem. Zde uvedenou reprezentaci využívá použité OpenCV).

Což znamená, pokud přímka prochází pod počátkem, ρ bude nabývat kladné hodnoty a bude svírat úhel menší než 180°.



Obrázek 2.6: rovnice přímky[1]

Nyní se můžeme podívat, jak pracuje Houghova transformace. Jakákoli přímka může být zapsána pomocí dvou parametrů (ρ, θ). Proto se pro tyto hodnoty vytvoří dvourozměrné pole, či nějaký akumulátor a inicializuje se na 0. Nechť řádky označují ρ a sloupce θ. Velikost pole závisí na přesnosti, kterou si stanovíme. Pokud bychom chtěli mít přesnost úhlu 1°, potom bychom potřebovali pole o velikosti 180 sloupců. Nejhorší možná přesnost pro ρ je

délka diagonály obrazu. Pokud bychom tedy chtěli mít přesnost 1 pixel, počet řádků pole by muselo být rovno délce diagonály obrazu.

Uvažujme obrázek 100x100 s vodorovnou čárou uprostřed. Vezmeme první bod přímky, známe tedy souřadnice X a Y. Nyní je dosadíme do rovnice a za θ dosazujeme 0,1,2,....,180 a kontrolujeme jaké ρ dostáváme. Pro každý pár (ρ, θ) zvyšujeme hodnotu o jeden v našem poli v odpovídajících buňkách (ρ, θ). Což znamená, že se nyní v poli nachází buňka $(50, 90) = 1$ spolu s některými jinými buňkami.

Nyní vezměme druhý bod na přímce a použijeme stejný postup jako u prvního bodu. Zvýšíme hodnotu v buňkách odpovídajícím (ρ, θ), které máme. Tentokrát jsme dostali buňku $(50, 90) = 2$. V každém bodě bude u buňky (50,90) navýšena hodnota. Tímto způsobem se dopracujeme k výsledku, kdy bude mít buňka (50,90) největší hodnotu, což nám říká, že obrázek obsahuje přímku ve vzdálenosti 50 od středu souřadnicového systému a s osou x uzavírá úhel 90°. [1]

Houghova transformace kružnice

Houghova transformace kružnice pracuje podobně jako Houghova transformace přímky. Pro detekci přímky bylo potřeba pouze dvou parametrů (ρ, θ), kdežto v případě kruhu je zapotřebí tří parametrů. (a, b, r). a a b definují pozici středu a r poloměr.[5] Rovnicí kružnice, podle které získáváme hodnoty je:

$$\begin{cases} x = a + r \cdot \cos(\theta) \\ y = b + r \cdot \sin(\theta) \end{cases} \tag{2.1}$$

2.4 Obrazové filtry

V dnešní době se již každý, kdo trochu fotí, setkal s rozmanitou škálou filtrů (doostření, rozmazání, zdůraznění kontur a další stylizace). Dnes již i obyčejný grafický editor, spustitelný na telefonu obsahuje nabídku filtrů, ze kterých si stačí pouze vybrat.

Konvoluce

Jedná se o nástroj používaný při úpravách všech možných signálů, nemusí tedy jít pouze o obraz. Konvoluční filtry lze aplikovat na zvukovou stopu za účelem odstranění šumu, dále na fotografie a další signály. V případě fotografie se jedná o dvourozměrnou konvoluci, jelikož je obraz 2D struktura.

Vzorec základní diskrétní 2D konvoluce: [17]

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

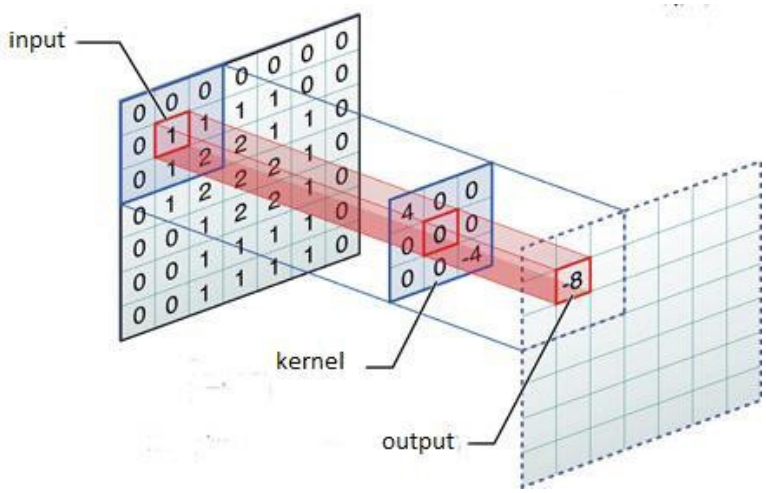
Konvoluce snímku

Uvažujme zpracovávaný snímek, jako matici hodnot. Poté snímek o rozlišení 1920 na 1080 pixelů je tedy maticí o rozměru 1920 sloupců na 1080 řádků. Jednotlivé části matice násobíme jinou maticí - tzv. konvolučním jádrem. Konvoluční jádro může nabývat různých rozměrů.

Konvoluční jádro postupně posouváme buňku po buňce postupně po celém obrazu, kde pokaždé provádíme následující kroky:

- Provedeme vynásobení hodnot konvolučního jádra s hodnotami části matice, nad kterou se právě nachází konvoluční jádro.
- Všechny násobky sečteme a uložíme do zpracovávané buňky.
- Přesuneme konvoluční jádro na následující buňku.

Princip konvoluce je ukázán na obrázku 2.7.[17]



Obrázek 2.7: Konvoluce (převzato z webu [1])

Vyhlazení obrazu

Vyhlazení obrazu je používáno převážně pro odstranění šumu z obrazu. Jedná se o rozostření obrazu, přijdeme tedy o tenké linie, které jsou součástí obrazu. Nejčastěji je využíváno takzvané Gaussovo rozostření. Matice 3x3 pro Gaussovo rozostření:

1	2	1
2	4	2
1	2	1

Konvoluční matice obsahuje pouze kladné koeficienty, což znamená, že k hodnotě zpracovávaného pixelu se přičte hodnota sousedního pixelu. Neboli světlo, které je obsaženo ve vedlejší buňce bude přičteno ke světlu aktuální buňky, což zapříčiní rozmazání obrazu. Proto matice pro vyhlazení snímku obsahují ve většině případů pouze kladné koeficienty. [8]

AHE

Adaptivní ekvalizace histogramu (AHE) je počítačová technika zpracování obrazu používaná ke zlepšení kontrastu v obrazech. Od běžné ekvalizace histogramu se liší tím, že adaptivní metoda počítá několik histogramů, z nichž každý odpovídá zřetelné části obrazu, a používá je k redistribuci hodnot světlosti obrazu. Je proto vhodný pro zlepšení místního kontrastu a zlepšení definic hran v každé oblasti obrazu.

[1] https://mc.ai/convolutional-neural-networks-from-the-ground-up/

AHE má však tendenci zvyšovat výskyt šumu v relativně čistých oblastech obrazu. Varianta adaptivního vyrovnávání histogramu nazvaná adaptivní histogramová ekvalizace s omezeným kontrastem (CLAHE) tomu zabraňuje omezením amplifikace.

Obyčejné vyrovnání histogramu používá stejnou transformaci odvozenou z histogramu obrazu k transformaci všech pixelů. Což funguje dobře, když je distribuce hodnot pixelů v celém obrazu podobná. Pokud však obraz obsahuje oblasti, které jsou podstatně světlejší nebo tmavší než většina obrazu, kontrast v těchto oblastech nebude dostatečně zesílen.

AHE se na tom zlepšuje transformací každého pixelu transformační funkcí odvozenou od sousední oblasti. Ve své nejjednodušší formě je každý pixel transformován na základě histogramu čtverce obklopujícího pixel.

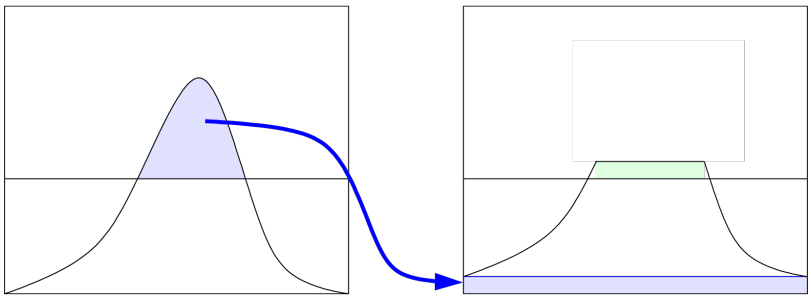
Pixely v blízkosti hranice obrazu musí být zpracovány speciálně, protože jejich okolí by nebylo zcela uvnitř obrazu. Problém lze vyřešit rozšířením obrazu zrcadlovými pixelovými čarami a sloupci s ohledem na hranici obrazu. Prosté kopírování řádků pixelů na ohraničení není vhodné, protože by vedlo k vysoce vrcholnému histogramu okolí. [10]

CLAHE

Obyčejný AHE má sklon ke zvyšovaní výskytu kontrastu v konstantní oblasti obrazu, a to z důvodu že histogram v takových oblastech je vysoce koncentrovaný. V důsledku toho může AHE způsobit zesílení šumu v téměř konstantních oblastech. Kontrast omezený AHE (CLAHE) je varianta adaptivního vyrovnávání histogramu, ve kterém je zesílení kontrastu omezeno, aby se tento problém zesílení šumu omezil.

V CLAHE je zesílení kontrastu v blízkosti dané hodnoty pixelu dáno sklonem transformační funkce. To je úměrné sklonu distribuční funkce sousedství (CDF) a tedy hodnotě histogramu při této hodnotě pixelu. CLAHE omezuje zesílení oříznutím histogramu na předem definovanou hodnotu před výpočtem CDF. To omezuje sklon CDF a tedy transformační funkci. Hodnota, při které je histogram oříznut, tzv. limit střihu závisí na normalizaci histogramu a tím na velikosti oblasti sousedství. Je výhodné neodstraňovat část histogramu, která přesahuje limit, ale rovnoměrně ji šířit mezi všechny histogramové hodnoty.

Přerozdělování opět zatlačí některé hodnoty nad limitem střihu (na obrázku 2.8 zelenou barvou), což vede k efektivnímu limitu střihu, který je větší než předepsaný limit a přesná hodnota závisí na obrázku. Pokud je to nežádoucí, postup redistribuce může být rekurzivně opakován, dokud není přebytek zanedbatelný. [14]



Obrázek 2.8: CLAHE (obrázek převzat z webu[2])

[2]https://upload.wikimedia.org/wikipedia/commons/thumb/5/5f/Clahe-redist.svg/1920px-Clahe-redist.svg.png

2.5 Barevný model

Barevný model je systém pro vytváření celé škály barev z malé sady primárních barev. Existují dva typy barevných modelů: aditivní a subtraktivní. Aditivní barevné modely používají světlo k zobrazení barev, zatímco subtraktivní modely se používají k tisku.

RGB

RGB je barevný model, který se používá převážně v zobrazovacích technologiích, které využívají světlo k zobrazení. V tomto modelu jsou barvy červená (Red), zelená (Green) a modrá (Blue) sčítány v různých intenzitách a vytvářejí tak miliony různých barev.

Model RGB je "aditivní"model: jak jsou přidávány barvy ve formě světla, výsledek bude světlejší. Například plná kombinace červené, zelené a modré barvy vytváří bílou. [7]

CIE XYZ

XYZ se typicky používá pro zjištění spektrální odezvy vzorku měřeného kolorimetrem nebo spektrofotometrem. Kolorimetr obsahuje pouze tři senzory, červený, zelený a modrý, nebo X, Y a Z. Obvykle se používá pro kalibraci a zobrazení profilů. Spektrofotometr zaznamenává celou spektrální odezvu v častých intervalech podél spektra, řekněme každých 10 nanometrů, a bude typicky používán pro měření vytištěných listů, buď pro řízení tisku, nebo ve spojení s ICC profilovacím softwarem, pro vytvoření profilu tiskárny.

Zatímco CIE XYZ se používá k zjištění barvy pomocí měřicích přístrojů, není pro lidi tak užitečné pomocí něj popisovat barvu. [9]

CIELAB

Barevný prostor CIELAB bylo definováno roku 1976 společností International Commission on Illumination. Popisuje barvu třemi číselnými hodnotami:

- L* - světelnost
- a* - poměr zelené a červené barvy
- b* - poměr modré a žluté barvy.

CIELAB bylo navrženo aby co nejlépe reprezentovalo barvu, tak jako ji vidí lidé. Což znamená, že změny v hodnotách L*a*b* odpovídají přibližně podobným vizuálním změnám vnímanými lidmi. S ohledem kladeným na bílý bod je model CIELAB nezávislý na zařízení definuje barvy nezávisle na tom, jak jsou vytvořeny nebo zobrazeny.

Jedná se tedy o třírozměrný prostor reálných čísel, díky kterému je možné zobrazit neomezené množství barev. V praxi je obvykle prostor namapován na třídimenzionální prostor celých čísel, aby jej bylo možné reprezentovat digitálně. Poté L* = 0 znamená nejtmavší bod a L* = 100 nejsvětlejší bod. Podíváme li se na a* a b*, pokud jsou obě hodnoty rovny 0, jedná se o neutrální šedou barvu. Osa a* reprezentuje zeleno červenou složku, zelená náleží zápornému směru a červená kladnému. Osa b* reprezentuje modro žlutou složku, modrá je v záporném směru a žlutá v kladném. Hranice a* a b* poté mohou mít hranici +-128 podle 8-bitového unsigned integeru. [9]

Převod RGB na CIELAB

Převod z barevného modelu RGB nelze provést přímo, je zapotřebí tak učinit "mezi-převodem"na barevný model CIE XYZ. Před převodem je zapotřebí si zvolit správnou matici pracovního prostoru RGB (v našem případě je nejvhodnější sRGB s referenční bílou D65). Poté se již jen jedná o dosazení do jednotlivých, které je možné nalézt ve webové prezentaci Bruce Justina Lindblooma [3].

2.6 Delta E

Pomocí ΔE lze porovnávat dvě barvy, podobně, jako je vidí člověk. Termín Delta pochází z matematiky, kde je využívána pro změnu proměnné, či funkce. Přípona E odkazuje na německé slovo Empfindung, což znamená pocit.

ΔE nabývá hodnot od 0 do 100:

- <= 1 Rozdíl barev není poznatelný lidským okem
- 1-2 Rozdíl je poznatelný, po nahlédnutí z blízka
- 2-10 Rozdíl je znatelný na první pohled
- 11-49 Barvy jsou si více podobné, nežli rozdílné
- 100 Barvy jsou přesným opakem

Je ovšem možné dostat dvě barvy, které jsou rozdílné a ΔE bude menší než 1. Tento problém se týká použití vzorců CIE76 a CIE94, ve kterých se saturace, či světelnost buď neberou v úvahu, nebo se neporovnávají správně. Díky nekonzistenci se hodnota ΔE liší, dle použitého vzorce.[16]

CIEDE2000

Vzorec CIEDE2000 řeší problémy předešlých vzorců. Je ovšem mnohem obtížnější, co se týká výpočtu. Jedná se o dosud nejvíce komplikovaný vzorec v ohledu na CIE porovnávací algoritmy. Vzorec CIE94 vyhodnotí jako velice podobné barvy zelenou a černou, stejně jako bílou a zelenou, k čemuž dochází díky špatné práci se saturací.[16]

Porovnání barev pomocí vzorce CIEDE2000 je založeno na barevném spektru CIELAB. CIEDE2000 krom dvou barev (L_i^*, a_i^*, b_i^* ; i= 1, 2) přijímá na vstupu i váhové faktory k_L, k_C a k_H.

Vzorec pro výpočet CIEDE2000:

1) Výpočet C_i a h_i

$$C^*_{i,ab} = \sqrt{(a^*_i)^2 + (b^*_i)^2} \quad i = 1, 2 \tag{2.2}$$

$$\overline{C}^*_{ab} = \frac{C^*_{1,ab} + C^*_{2,ab}}{2} \tag{2.3}$$

$$G = 0.5\left(1 - \sqrt{\frac{C^{*7}_{ab}}{\overline{C}^{*7}_{ab} + 25^7}}\right) \tag{2.4}$$

$$a'_i = (1 + G)a^*_i \quad i = 1, 2 \tag{2.5}$$

$$C'_i = \sqrt{(a'_i)^2 + (b^*_i)^2} \quad i = 1, 2 \tag{2.6}$$

$$h'_i = \begin{cases} 0 & b^*_i = a'_i = 0 \\ \tan^{-1}(b^*_i, a'_i) & \text{jinak} \end{cases} \qquad i = 1, 2 \tag{2.7}$$

2) Výpočet $\Delta L', \Delta C', \Delta H'$:

$$\Delta L' = L^*_2 - L^*_1 \tag{2.8}$$

$$\Delta C' = C'_2 - C'_1 \tag{2.9}$$

$$\Delta h' = \begin{cases} 0 & C'_1 C'_2 = 0 \\ h'_2 - h'_1 & C'_1 C'_2 \neq 0; |h'_2 - h'_1| \leq 180^\circ \\ (h'_2 - h'_1) - 360 & C'_1 C'_2 \neq 0; (h'_2 - h'_1) > 180^\circ \\ (h'_2 - h'_1) + 360 & C'_1 C'_2 \neq 0; (h'_2 - h'_1) < -180^\circ \end{cases} \tag{2.10}$$

$$\Delta H' = 2\sqrt{C'_1 C'_2}\sin\left(\frac{\Delta h'}{2}\right) \tag{2.11}$$

3) Výpočet rozdílů barev CIEDE2000 ΔE_{00}:

$$\overline{L}' = \left(L_1^* + L_2^*\right)/2 \tag{2.12}$$

$$\overline{C}' = \left(C_1' + C_2'\right)/2 \tag{2.13}$$

$$\overline{h'} = \begin{cases} \frac{h_1'+h_2'}{2} & |h_1' - h_2'| \leq 180°; C_1'C_2' \neq 0 \\ \frac{h_1'+h_2'+360°}{2} & |h_1' - h_2'| > 180°; (h_1' + h_2') < 360°; \\ C_1'C_2' \neq 0 & \\ \frac{h_1'+h_2'-360°}{2} & |h_1' - h_2'| > 180°; (h_1' + h_2') \geq 360°; \\ C_1'C_2' \neq 0 & \\ (h_1' + h_2') & C_1'C_2' = 0 \end{cases} \tag{2.14}$$

$$\begin{aligned} T &= 1 - 0.17\cos\left(\overline{h}' - 30°\right) + 0.24\cos\left(2\overline{h}'\right. \\ &\quad + 0.32\cos\left(3\overline{h}' + 6°\right) - 0.20\cos\left(4\overline{h}' - 63°\right) \end{aligned} \tag{2.15}$$

$$\Delta\theta = 30\exp\left\{-\left[\frac{\overline{h}' - 275°}{25}\right]^2\right\} \tag{2.16}$$

$$R_C = 2\sqrt{\frac{\overline{C}'^7}{\overline{C}'^2 + 25^j}} \tag{2.17}$$

$$S_L = 1 + \frac{0.015\left(\overline{L}' - 50\right)^2}{\sqrt{20 + \left(\overline{L}' - 50\right)^2}} \tag{2.18}$$

$$S_c = 1 + 0.045\overline{C}' \tag{2.19}$$

$$S_H = 1 + 0.015\overline{C}'T \tag{2.20}$$

$$R_T = -\sin(2\Delta\theta)R_C \tag{2.21}$$

$$\begin{aligned} \Delta E_{00}^{12} &= \Delta E_{00}\left(L_1^*, a_1^*, b_1^*; L_2^*, a_2^*, b_2^*\right) \\ &= \sqrt{\left(\frac{\Delta L'}{k_L S_L}\right)^2 + \left(\frac{\Delta C'}{k_C S_C}\right)^2 + \left(\frac{\Delta H'}{k_H S_H}\right)^2 + R_T\left(\frac{\Delta C'}{k_C S_C}\right)\left(\frac{\Delta H'}{k_H S_H}\right)} \end{aligned} \tag{2.22}$$

[16]

Kapitola 3

Návrh

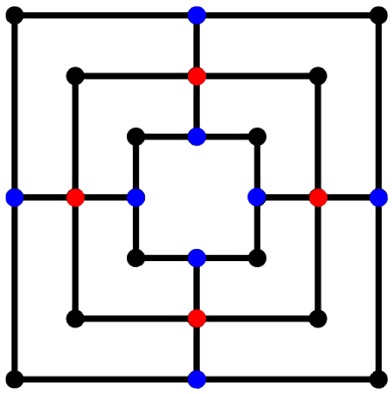
Před tím než se pustíme do samostatné implementace je zapotřebí se zamyslet nad některými částmi práce. Především je zapotřebí si stanovit kritéria, podle kterých se bude hodnotit daná pozice a také nad nemalou problematikou detekce pozice. Ke konci je uvedeno i něco málo o návrhu GUI.

3.1 Ohodnocení pozice

Při hře mlýny nelze zaručit výhra, což ukazuje i algoritmus nalezený Ralphem Gasserem [6]. Pomocí jehož ohodnocení lze dosáhnout nejhůře remízy. Ovšem je zapotřebí projít velké množství stavů a aby bylo možné tah hledat v přiměřeném časovém rozmezí je zapotřebí databáze, což u mobilních aplikací není žádoucí situací.

Z toho důvodu jsem se inspiroval věcmi, které hodnotí Ralph Gasser a přidal i své, čímž jsem se pokusil o optimalizaci na mobilní zařízení.

Na začátku ohodnocení algoritmus zjistí jestli hráč nevyhrál, kdy vrací maximální číslo, naopak pokud prohrává, vrací minimální číslo.



Obrázek 3.1: Výhodná pole

Na desce se nachází pozice, u kterých lze předpokládat, že jsou výhodnějším strategickým místem a to z důvodu buď větší možnosti tahů, či lepší možnosti uzavření mlýnu. Tyto pozice se nachází uprostřed stran jednotlivých čtverců, ze kterých se hrací deska skládá.

Vyznačená pozice jsou zvýrazněny na obrázku 3.1. Tyto pozice jsou dvojího typu, proto jsou použity dvě barvy pro zvýraznění. Červené pozice označují pozice s větší výhodou, popřípadě větším počtem tahů, a proto je zapotřebí je zvýhodnit více, nežli modrá (obsazené červené pole: 3, modré pole :1)

Hlavní částí celého ohodnocení jsou samozřejmě hlavní body, díky kterým se hráč přibližuje výhře. Což znamená pokud hráč uzavře mlýn, či zmenší možný počet polí, po kterých může táhnout zvýší se ohodnocení. Hodnotí se tedy počet uzavřených mlýnů, počet dvou kamenů v jedné linii s jedním volným polem, počet kamenů hráče a počet všech možných tahů. Z daných počtů se odečítají počty stejných vlastností protihráče.

Abych započítal do ohodnocení i 3 fáze hry, tak v každé fázi se přikládá jiný důraz na počty v hlavním ohodnocení pomocí koeficientů, kterými jsou počty násobeny. Koeficienty jsou zobrazeny v tabulce 3.1.

Popis algoritmu ohodnocení:

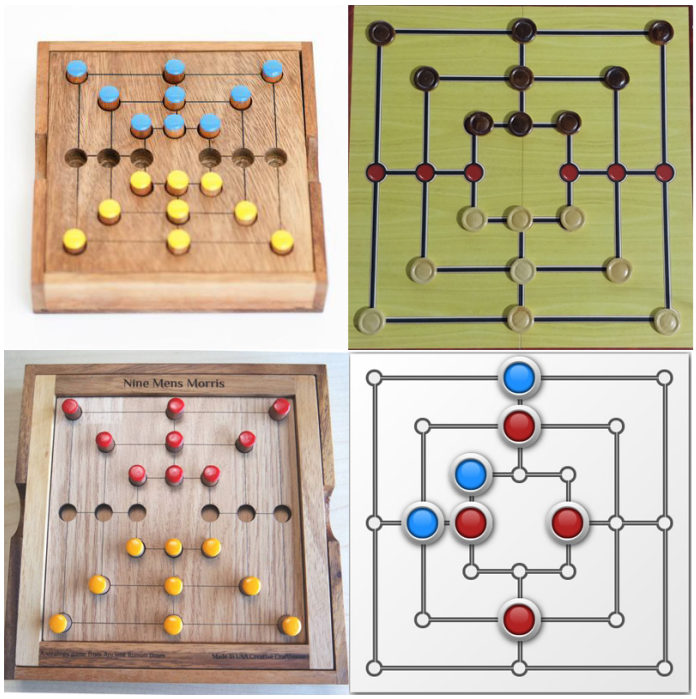
- Pokud hráč vyhrál, vrať maximální hodnotu, pokud prohrál, vrať minimální hodnotu.
- Proměnnou ohodnocení (value) inicializuj na 0.
- Najdi všechny řady, ve kterých se nachází pouze jeden kámen, který náleží hráči. Za každou takovou řadu přičti do value 1.
- Najdi všechny řady, ve kterých se nachází pouze jeden kámen, který náleží protihráči. Za každou takovou řadu odečti 1.
- Projdi všechna zvýhodněná pole. Za každé více zvýhodněné pole(4, 10, 13, 19) na kterém se nachází kámen hráče, přičti do value 3 a za každé zvýhodněné pole (1, 9, 14, 22, 7, 11, 12, 16) přičti do value 1. Při výskytu kamene protihráče na zvýhodněných polích odečti od value stejné hodnoty, jako by byly přičteny za hráčovi kameny.
- Zjisti stav hry a podle ní urči koeficienty z tabulky 3.1.
- Do value přičti coef_0 * počet uzavřených mlýnů hráčem, naopak odečti od value coef_0 * počet uzavřených mlýnů protihráčem.
- Do value přičti coef_1 * Počet možných uzavřených mlýnů hráčem a naopak odečti coef_1 * Počet možných uzavřených mlýnů protihráčem.
- Do value přičti coef_2 * Počet kamenů hráče a odečti coef_2 * Počet kamenů protihráče.
- Do value přičti coef_3 * Počet možných tahů hráče a odečti coef_3 * Počet možných tahů protihráče.
- Vrať hodnotu value.

	Počet uzavřených mlýnů (coef_0)	Počet možných uzavřených mlýnů (coef_1)	Počet kamenů (coef_2)	Počet možných tahů (coef_3)
Rozmístění	630	91	85	380
Přesouvání	940	76	70	205
Skákání	1400	80	48	195

Tabulka 3.1: Koeficienty

3.2 Detekce pozice

Při detekci pozice jsem se snažil najít řešení, které by pokrylo co nejvíce hracích desek. Herní desky se od sebe liší nejen barevností, ale prakticky každý výrobce hry / vývojář aplikace si vytvořil vlastní kameny i políčka. Po menším průzkumu jsem došel k závěru, že většina implementací dodržuje kruhový tvar políček i kamenů, jako můžeme vidět na obrázku 3.2. Z uvedených důvodů jsem se rozhodl detekci pozice založit na detekování hran a v detekovaných hranách následně hledat kruhy, určit jejich pozici a následně rozpoznat, zda se jedná o políčko, či kámen.



Obrázek 3.2: Rozdílné hrací desky (obrázky převzaty z webů [1])

[1] https://www.creativecrafthouse.com/pub/media/catalog/product/cache/c687aa7517cf01e65c009f6943c2b1e9/n/i/nine_mens_morris_premium.jpg
https://cdn.shopify.com/s/files/1/0181/8685/products/9manmorris.jpg?v=1352180584

Detekce políčka a kamene

Jak políčka, tak kameny jsou tedy kruhového tvaru. Z toho důvodu jsem se rozhodl pro detekci kružnic pomocí detekce hran a následné Houghovy transformace kružnice. Následně je potřeba nalezené pozice seřadit podle jejich nalezené polohy, čehož lze dosáhnout porovnání pozice oproti ose y, kde se dostaneme na určitou řadu a následné porovnání oproti ose x, kde dostaneme pozici ve sloupci. Při porovnávání na ose y je zapotřebí použití tolerance, jako toleranci jsem použil průměr největšího nalezeného kamene na hrací desce.

U některých herních desek je výrazný rozdíl mezi velikostí kamene a políčkem, jako příklad mohu uvést implementaci od Smart Little Games [2], kde si detekci můžete vyzkoušet bez toho, abyste potřebovali fyzickou deskovou hru. Ačkoli faktor rozdílných velikostí neplatí u všech možností, je kvůli zpřesnění výsledků započítán následujícím způsobem. Nalezne se největší nalezený kruh a nejmenší nalezený kruh, pokud má nejmenší kruh výrazně menší poloměr, než-li největší, poté se do vyhodnocení počítá i velikost poloměru. Pokud má detekovaný kruh výrazně menší poloměr než-li největší, poté je vyhodnocen jako políčko a následující vyhodnocení barev se neprovádí. Jak je již na první pohled vidět, při použití této metody na herní desku s políčky o stejném, nebo velice podobném průměru, jako je průměr kamene, metoda nebude mít žádný dopad. Proto není potřeba její eliminace při takovýchto herních deskách.

Hlavním faktorem je barva políčka a kamenů. Před první detekcí pozice je tedy potřebné získat barvu políčka a obou barev kamenů, aby bylo možné detekci pomocí barvy provést. Naskenované barvy se následně uloží do paměti aplikace a k jejich smazání dojde až při smazání proměnných prostředí uživatelem aplikace. Při detekci pozice se provede zprvu detekce kružnic a z daných kružnic se vypočítá průměrná barva. Pro zpřesnění detekce a následného vypočtu průměrné barvy je průměr detekovaného kruhu zmenšen o 7 obrazových bodů. Získaná barva je ovšem popsána pomocí barevného modelu RGB, který je používaný pro zobrazení obrazu na display zařízení. Model RGB se pro porovnávání barev příliš nehodí a to z důvodu, že neklade příliš důraz na světelnost a tím pádem nepopisuje změnu barvy jako to umí model CIELAB, který se mnohem lépe přibližuje vnímání barev lidským zrakovým systém. Proto je zapotřebí provést převod barvy v získaném modelu RGB do barevného modelu CIELAB, čehož nelze dosáhnout přímo a musí být použit "mezi-převod"do barevného modelu CIE XYZ.

Pro následné porovnání barev byl zvolen vzorec CIEDE2000, se kterým jsem bylo dosaženo nejlepších výsledků. Porovnání barev je prováděno přímo mezi detekovanou barvou a barvou políčka, či barvami kamenů, které jsou uloženy v paměti aplikace. Nemá smysl brát v potaz tabulku podobnosti ΔE z důvodu osvětlení hrací desky a kvality snímku pořízeného kamerou chytrého telefonu, či tabletu. Proto je porovnání barev založeno na porovnání barev uložených v paměti aplikace a aktuální naskenovanou barvou.

3.3 Návrh GUI

Jako většina aplikací, které obsahují GUI se ani tahle nevyhnula návrhu grafické stránky. Jelikož se jedná o aplikaci, která nepotřebuje žádné velké efekty a grafické zázraky, byl při návrhu kladen důraz spíše na přehlednost a snadné ovládání. Jednotlivé důležité obrazovky z aplikace jsou popsány níže.

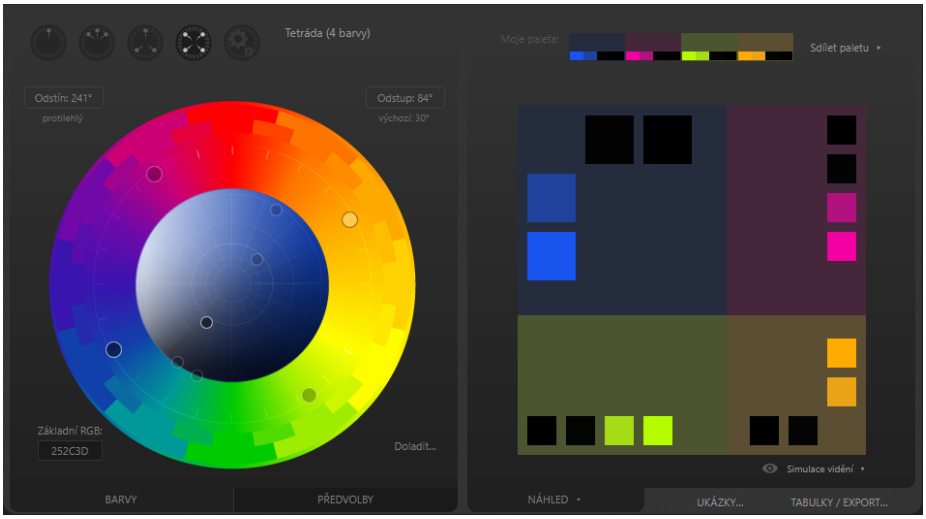
Při návrhu byly použity prvky Material designu, který využívá většina ne příliš graficky náročných aplikací pro android (např. Messenger). Jedná se o vizuální jazyk sjednocující

[2] http://www.smartlittlegames.com/ninemensmorris

klasické prvky designu s moderními technologiemi a inovativními prvky. Dále při tvoření designu aplikace je dobré, aby bylo myšleno na responsivitu, čehož lze poměrně jednoduše v Android studiu dosáhnout a to použitím jednotek dp.

Výběr barev

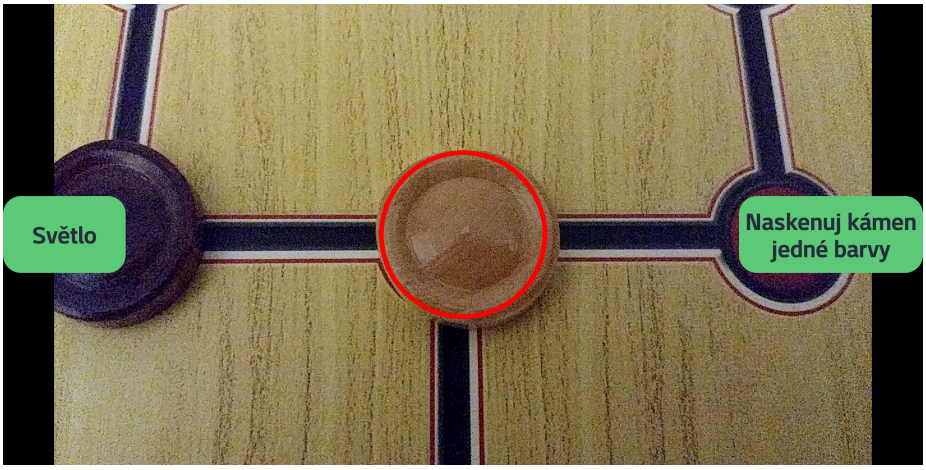
Před návrhem je zapotřebí si zvolit barvy. Na začátku si zvolíme hlavní barvu, od které se návrh může odrazit. Pro můj návrh byla taková barva pozadí (RGB #252c3d). Následně je možné použít jeden z nástrojů pro pomoc výběru s barvami schéma. Takový nástroj dokáže zaručit, že vybrané barvy budou do sebe zapadat, a celý layout bude jednotvárný. Já k takovému návrhu použil nástroj Color Scheme Designer 3, který je možné vidět na obrázku 3.3.



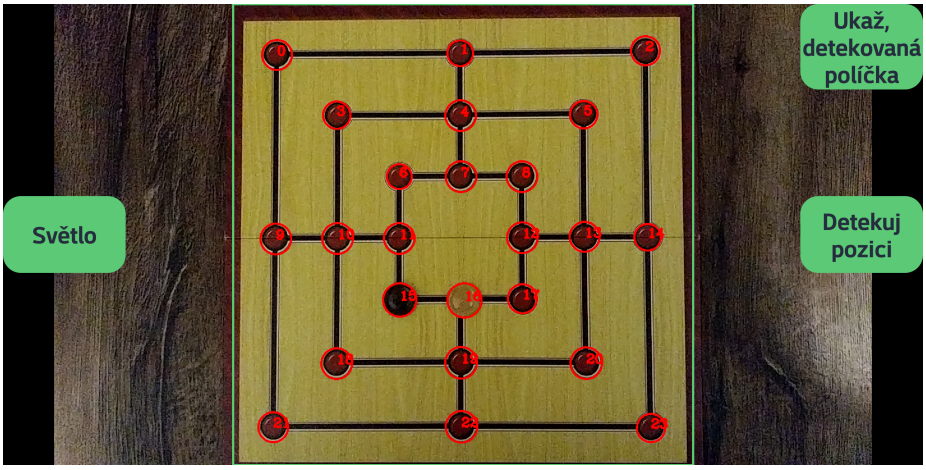
Obrázek 3.3: Color Scheme Designer 3

Návrh GUI samostatné aplikace

Aplikace má za úkol naskenovat pozici na dané šachovnici a následovně určit nejlepší tah, proto jsou tyto položky usazeny v hlavní nabídce spolu s možností hry a nastavením. Pro možnost naskenování jsou zapotřebí dvě obrazovky (skenování barev 3.4 a následné pozice obrázek 3.5). Obrazovky obsahují tlačítko pro zapnutí a vypnutí blesku (Světlo), spuštění skenování. Detekce pozice je rozšířena o možnost zobrazení detekovaných políček.

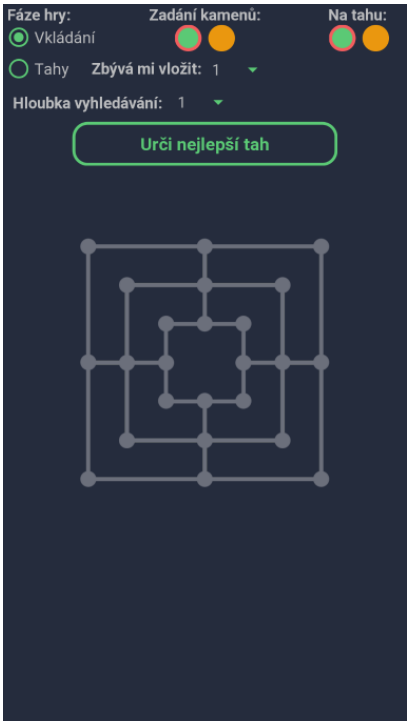


Obrázek 3.4: layout ScanColor



Obrázek 3.5: layout CameraDetection

Následně nás aplikace přenese tedy do layoutu pro určení nejlepšího tahu, do kterého je možné se dostat i přes menu a pozici zadat ručně. Z důvodu nepřesné detekce je layoutu doplněna možnost o zadávání kamenů. Layoutu tedy musí obsahovat zadání fáze hry, barvu zadávání kamenů, určení kdo je na tahu, v případě fáze vkládání kolik zbývá vložit kamenů a nakonec hloubku vyhledávání v Alfa-Beta stromu. Layout pro možnost Hrát je podobný layoutu pro určení nejlepšího tahu, ovšem ztrácí všechny možnosti, jako je zadaní fáze hry, barvy kamenů, ... Před tím, než hra započne, je možné zvolit si barvu a následně i obtížnost. Tlačítka zbytku aplikace jsou stejného stylu, jako je tlačítko, pro určení nejlepšího tahu, které je vidět na návrhu nalezení nejlepšího tahu 3.6. Pod položkou nastavení se nachází pouze možnost resetování naskenovaných barev.



Obrázek 3.6: layout Manualy

Kapitola 4

Implementace

V následujících podkapitolách je popsaný vývoj samostatné aplikace pro operační systém Android s využitím jazyku Java a knihovny OpenCV. Implementace se dělí na dvě hlavní části, jimiž jsou hra mlýny(implementace pravidel, ohodnocení a hledání nejlepšího tahu) a detekce pozice pomocí kamery.

4.1 Použité nástroje pro vývoj

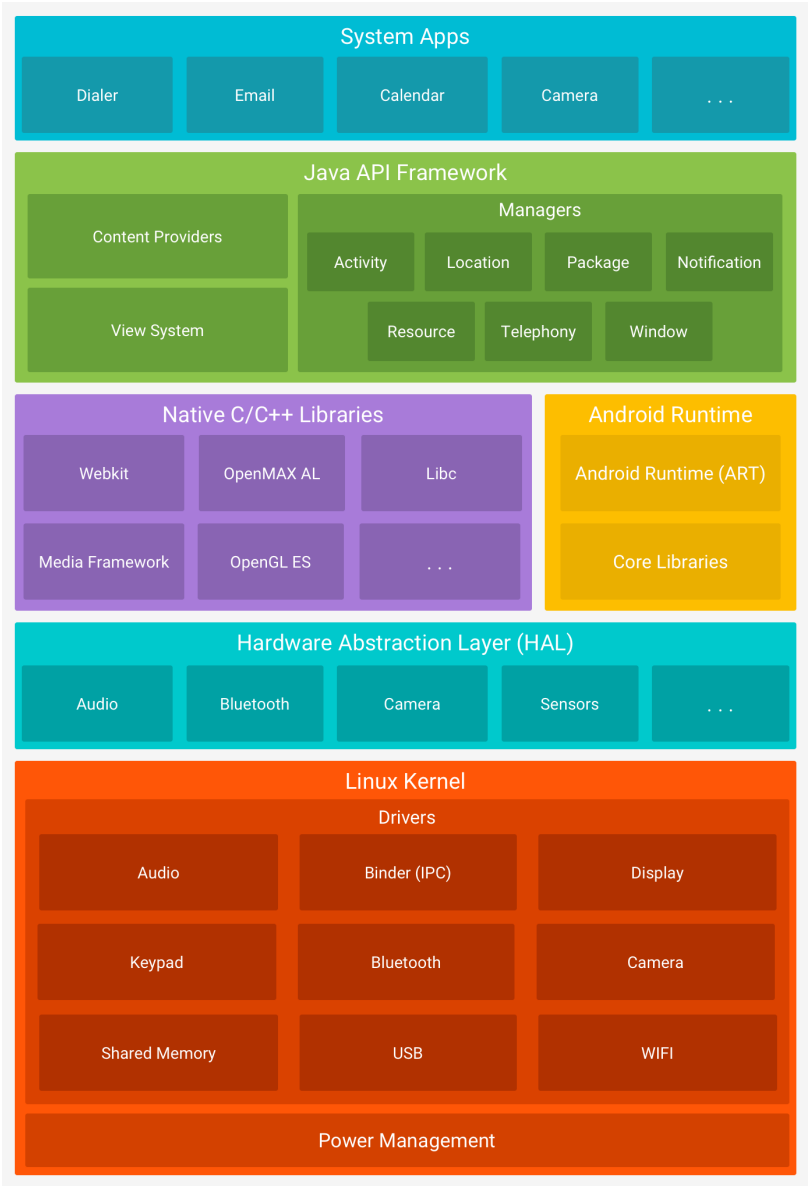
Před popisem samostatné implementace je uveden výčet nástrojů použitých pro vývoj s krátkým popisem.

4.2 Android

Android je mobilní operační systém vyvíjený společností Google. Primárně je navrhnut pro zařízení s dotykovou obrazovkou, jako jsou chytré telefony či tablety. Google samozřejmě pokračuje dále a vyvíjí Android TV pro televize, Android Auto pro auta a Wear OS pro chytré hodinky / náramky. Varianty androidu jsou také použity na herních konzolách, digitálních kamerách, počítačích a jiných elektronických zařízeních.

Architektura Android

Android je open source software založený na Linuxu, vytvořený pro širokou škálu zařízení a formovacích faktorů. Na obrázku 4.1 jsou znázorněny hlavní komponenty platformy Android.



Obrázek 4.1: Architektura Android [2]

Linux Kernel

Základem platformy Android je jádro Linuxu. Například Android Runtime (ART) spoléhá na jádro Linuxu pro základní funkce, jako je vlákno řízení a správa nízké úrovně paměti.

Použitím jádra Linuxu Androidem je umožněno využívat klíčové bezpečnostní funkce a výrobcům zařízení vyvíjet hardwarové ovladače pro již dobře známé jádro.

Abstrakční vrstva hardware

Hardware Abstraction Laye(HAL) je vrstva poskytující standardní rozhraní, které poskytuje funkce hardware vyšší vrstvě rozhraní Java API. Skládá se z několika modulů knihoven, každá z nich implementuje interface pro specifický typ hardwarové komponenty, jako

je například kamera, nebo modul Bluetooth. Při každém volání hardwarové komponenty Android načte modul knihovny určený pro danou komponentu.

Android Runtime

U zařízení se systémem Android verze 5.0 nebo vyšší je každá aplikace spuštěna ve vlastním procesu a vlastní instanci služby Android Runtime (ART). ART je napsán tak, aby dokázal spustit více virtuálních počítačů na zařízeních s nízkou pamětí pomocí spouštění souborů DEX, formátu bytecode, který je navržený speciálně pro Android a je optimalizovaný pro minimální nároky na paměť.

Nativní knihovny C / C ++

Mnoho základních komponent a služeb systému Android, jako je ART a HAL, je postaveno z nativního kódu, který vyžaduje nativní knihovny napsané v jazyce C a C ++. Android poskytuje JAVA framework poskytující některé z těchto nativních knihoven aplikacím. Díky čemuž lze přistupovat například k OpenGL API.

Java API Framework

Všechny funkce operačního systému jsou zpřístupněny prostřednictvím rozhraní napsaném v jazyce JAVA. Tato rozhraní tvoří bloky, které jsou potřeba při tvorbě Android aplikací a to zjednodušením opětovného použití základních, modulárních komponent a služeb systému.

Systémové aplikace

Android je dodáván se sadou základních aplikací (email, posílání SMS, kalendář, ...). Aplikace, které jsou součástí platformy, nemají mezi aplikacemi, které se uživatel rozhodne nainstalovat, žádný zvláštní status. Aplikace třetí strany se tak může stát výchozí aplikací pro danou činnost (např. webový prohlížeč). [2]

Android Studio

Android Studio je oficiální integrované vývojové prostředí(IDE) pro vývoj aplikací pro Android. Je založen na IntelliJ IDEA, integrovaném vývojovém prostředí pro software psaný v jazyce Java a obsahuje jeho nástroje pro editaci kódu.

Pro podporu vývoje aplikací v operačním systému Android používá Android Studio sestavovací systém založený na technologii Gradle, emulátoru, ukázkové kódy a integraci Github. Každý projekt v Android Studiu má jednu nebo více modulů se zdrojovým kódem a zdrojovými soubory. Tyto moduly zahrnují moduly aplikací pro Android, moduly Knihovny a moduly Google App Engine.

Aplikace Android Studio používá funkci Push Instant Push k posunu změn kódu a prostředků do spuštěné aplikace. Editor kódu pomáhá vývojáři psát kód našeptáváním, adresováním a analýzou kódu. Aplikace vytvořené v aplikaci Android Studio jsou zkompilovány do formátu APK aby mohly být následně nainstalovány do systému Android.

Aplikace Android Studio byla poprvé zveřejněna na webu Google I / O v květnu 2013 a první stabilní verze byla vydána v prosinci 2014. Android Studio je k dispozici pro platformy Mac, Windows a Linux. Nahradila Eclipse Android Development Tools (ADT) jako primární IDE pro vývoj aplikací pro Android. Aplikaci Android Studio a sadu Software Development Kit lze stáhnout přímo od společnosti Google. [18]

OpenCV

OpenCV (Open Source Computer Vision Library) je open source knihovna pro počítačové vidění a strojové učení. OpenCV byla vytvořena aby poskytovala běžnou infrastrukturu potřebnou pro počítačové vidění a urychlila užívání strojového vnímání v komerčních produktech. Jelikož se jedná o BSD-licenci, tak se OpenCV lehce používá a upravuje.

Jedná se o komplexní sadu klasických i nejmodernějších algoritmů pro počítačové vidění a strojové učení. Algoritmy mohou být použity pro detekci tváře, identifikaci objektů, klasifikaci lidských činností ve videu, zaznamenávaní pohybů, extrakci 3D modelů objektů a mnohem více.

Existují rozhraní pro jazyky C++, Python, Java a MATLAB a je podporována operačními systémy Windows, Linux, Android a Mac OS. OpenCV převážně směřuje k aplikacím využívající vidění v reálném čase a pokud je možno, tak využívá instrukce z MMX a SSE.[4]

Git

Hlavní rozdíl mezi Gitem a jakýmkoliv jiným VCS (Verzovacím systémem) je způsob, jakým Git přemýšlí o svých datech. Koncepčně většina ostatních systémů ukládá informace jako seznam změn založených na souborech. Tyto systémy (CVS, Subversion, Perforce, Bazaar atd.) uvažují nad informacemi, které uchovávají, jako nad sadou souborů a změny provedených nad každým souborem v čase.

Git nepřemýšlí ani neukládá data tímto způsobem. Namísto toho Git pracuje s daty spíše jako se snapshoty miniaturního souborového systému. Pokaždé, když dojde ke commitu nebo uložení projektu v Gitu, vezme v podstatě obrázek toho, jak všechny soubory vypadají v té chvíli a ukládá odkaz na tento snímek. Pokud se soubory nezměnily, soubor Git neukládá soubor znovu - pouze odkaz na předchozí identický soubor, který již uložil.

Jedná se tedy o užitečný nástroj umožňující přehlednou a jednoduchou správu kódu nejen při týmovém vývoji. Pomocí "commitů", které reprezentují změny kódu, pomáhá ukládat historii změn v kódu. Slouží také jako záloha pro zdrojové kódy, protože se ve většině případů jedná o externí úložiště.

4.3 Detekce kružnic

Pro detekci kružnic byly využity funkce knihovny OpenCV. V této podsekci je popsáno jejich použití s parametry, se kterými jsem dosáhl nejlepších výsledků.

Při detekci kružnic je zapotřebí si obraz nejprve převést z barevného obrazu na stupňů šedi, díky čemuž docílíme rychlejšího zpracování.

```
Imgproc.cvtColor(original, gray, Imgproc.COLOR_RGB2GRAY);
```

- original = vstupní obraz
- gray = výstupní obraz
- Imgproc.COLOR_RGB2GRAY = typ převodu

Pro efektivnější a rychlejší zpracování jsem zvolil oříznutí obrazu pouze na část, kde se nachází zobrazená část pro detekci (viz. obrázek 3.5).

Před samostatnou detekcí jsem použil funkci pro Gaussovo vyhlazení

```
Imgproc.GaussianBlur(gray, gray2, new Size(5, 5), 0);
```

- gray = vstupní obraz
- gray2 = výstupní obraz
- new Size(5, 5) = velikost matice (5x5)
- 0 = sigmaX

Pro detekci kružnic slouží funkce:

```
private int maxRadius = 50;
private int minRadius = 10;

Imgproc.HoughCircles(gray, circles, Imgproc.HOUGH_GRADIENT,
  1.0,maxRadius*2, 100.0, 20.0, minRadius, maxRadius);
```

- gray = vstupní obraz
- circle = vektor pro uchování tří hodnot, x_{stredu}, y_{stredu} a poloměr pro každou kružnici
- Imgproc.HOUGH_GRADIENT = definování metody pro detekci
- 1.0 = inverzní poměr rozlišení
- maxRadius*2 = minimální vzdálenost mezi
- 100.0 = vrchní hranice pro vnitřní Cannyho hranový detektor
- 30.0 = hranice pro detekovanou kružnici (čím je číslo vyšší, tím se detekuje s větší pravděpodobností kružnice, čím naopak větší, tím se detekuje více šumu, jako kružnice)
- minRadius nejmenší detekovaný poloměr
- maxRadius největší detekovaný poloměr

4.4 Přehled tříd (Hra Mlýny)

V této podkapitole je uveden přehled tříd a jejich funkcionality použité při implementaci týkající se pravidel hry Mlýny a nalezení nejlepšího tahu. Tyto třídy se nachází v balíčku game.

Manualy

Manualy je třída implementující aktivitu k ovládání layoutu pro nalezení nejlepšího tahu (obrázek 3.6). Je propojena s detekcí pozice, která jí předává právě detekovanou pozici pomocí proměnných, předávaných při vytváření aktivity. Pro předání proměnných mezi aktivitami lze využít Bundle, kterým lze předat i pole. Aktivita umí pracovat i bez předešlé detekce a to tak, že si pozici nastaví na počáteční, pokud jí není žádná zadána. Pro určení nejlepšího tahu slouží třída AplhaBeta z balíčku logic. Pro jednotlivé tahy poté alouží třída Board ze stejného balíčku.

Play

Play je třída implementující aktivitu k ovládání layoutu pro možnost zahrát si proti aplikaci. Pracuje stejně, jako monualy s třídami AplhaBeta a Board. Play i s layoutem pro ni byly vytvořeny pro snazší testování určení nejlepšího tahu. Při zapínání možnosti hraní je možné pomocí třídy GameSetup si navolit barvu a obtížnost. Obtížnost znamená hloubku prohledávání pomocí Alfa-Beta (Lehká = hloubka 2, Střední = hloubka 3, Těžká = hloubka 4). Tato hloubka je používána po celou hru.

AplhaBeta

Jak je již z názvu patrné třída slouží pro vyhledání nejlepšího tahu pomocí algoritmu AlphaBeta. Obsahuje i ohodnocení pozice založené na pravidlech popsaných v návrhu. Při vytváření instance třídy na vstupu očekává hloubku, do které bude vyhledávání provedeno a instanci třídy Board, ve kterém je zaznamenána aktuální pozice. AplhaBeta využívá při hledání třídu Board z balíčku logic, ve které si při postupném vyhledávání tahá kameny a na konci vrací pozici do původního stavu. Po zavolání funkce bestMove, dostaneme nejlepší tah.

Board

Třída Board obsahuje veškerou možnou práci uskutečněnou nad hrou Mlýny. Všechny možné uzavřené mlýny, či tahy při fázi přesouvání kamenů jsou implementovány pomocí polí. Hlavní funkcí, která je ve třídě implementována je allTurns, která vrací všechny možné tahy v danou chvíli pro daného hráče. Vrácené tahy jsou uloženy v datovém typu list, jehož prvky jsou instancemi třídy Turn. Zbytek funkcí využívá právě funkce allTurns (kameny, které mohou být sebrány, kameny tvořící mlýn, ...), či funkce informační (kolik kamenů hráči zbývá, kdo je na tahu, ...).

Turn

Třída Turn slouží k uložení tahu (odkud, kam, který kámen je sebrán), je používán jako podpůrná třída ke třídě Board. Vytváření instancí Turn je trojího typu, stejně jako je tomu v pravidlech hry. Záleží na počtu vstupních parametrů, kde jeden znamená vkládání (pouze kam), dva tah (odkud, kam) a tři tah se sebráním kamene (odkud, kam, kde).

4.5 Přehled tříd (detekce pozice)

V této podkapitole je uveden přehled tříd a jejich funkcionality použité při implementaci týkající se detekce pozice pomocí kamery. Tyto třídy se nachází v balíčku detection.

ScanColor

CameraDetection je třída implementující aktivitu k ovládání layoutu pro detekci barvy políčka a kamenů (obrázek 3.4). Pro výpočet průměrné hodnoty barvy slouží třída CircleColorDetection. Vypočítaná hodnota je ukládána do SharedPreferences, což jsou proměnné, které zůstávají uložené i po vypnutí aplikace.

CameraDetection

CameraDetection je třída implementující aktivitu k ovládání layoutu pro detekci pozice. CameraDetection vytváří při vzniku 3 tlačítka na display, která jsou vidět na obrázku 3.5. Hlavní funkcí třídy je detectPosition, která detekuje a vrací detekovanou pozici ze vstupního obrazu. Funkce detectionType umožňuje detekci dvojího typu. Typ 0 slouží pro pouhou detekci kruhů v obraze, detekce vrací pozici, která obsahuje pouze políčka (tato detekce je spuštěna po kliknutí na tlačítko "Ukaž detekovaná políčka". Typ 1 slouží pro detekci celé pozice, k čemuž využívá třídu CircleColorDetection (tato detekce je spuštěna při zmáčknutí tlačítka "Detekuj pozici").

CircleColorDetection

Třída CircleColorDetection je podpůrná třída pro detekci políčka. Obsahuje 3 důležité funce:

- getCircleColor, díky níž je vypočítána a navrácena průměrná hodnota barvy v zadané kružnici na zachyceném obrazu.
- rgbToLab, která slouží pro převod barvy z barevného modelu RGB do CIELAB
- deltaE2000, která je využívána k porovnání barev pomocí vzorce deltaE2000

4.6 Problémy při implementaci

První část bakalářské práce se týkala hry samotné se zaměřením na nejlepší tah. Kde mi největší problém dělala funkce pro ohodnocení pozice. Snažil jsem se inspirovat od různých řešení, která jsem našel na Gitu. Skončil jsem ovšem u vlastního ohodnocení, které je mírně inspirováno algoritmem nalezeným Ralphem Gasserem.

Jelikož jsem s detekcí v obraze nikdy nepracoval, tak se problémů vyskytlo hned několik a rád bych je zde zmínil. Při detekci kružnic jsem bojoval nejvíce se správným nastavením filtrů pro odstranění šumu. Jelikož jsem nebyl schopný nastavit vše pouze pomocí parametrů ve funkci pro nalezení kružnic 4.3 musel jsem použít i Gaussovo rozostření pro vyhlazení obrazu.

Největším problém ovšem nastal při detekci barev. Při porovnání barev dochází k velkým rozdílům, které jsou závislé na osvětlení, odrazu světla, provedení desky (lesklý povrch, či velice podobná políčka s jednou barvou kamene jsou problémem) a kvalitě snímku. Prvotní návrh, který jsem chtěl realizoval byl, nastavit určitou mez ΔE, která pokud by se překročila, barva by byla určena, jako neplatná. Tento způsob se ovšem ukázal jako nereálný. Poslední u čehož jsem skončil, je přímé porovnání ΔE jednotlivých barev mezi sebou a prostým porovnáním je pomocí ΔE určena barva. Dále jsem se snažil vylepšit samostatnou detekovanou barvu pomocí odstranění odlesků, k čemuž jsem se snažil využít ekvalizaci histogramu CLAHE, nebo vlastní pokus o detekci odlesku a následné upravení kontrastu. Dále jsem se pokoušel porovnávat barvy v RGB. Nic ovšem z uvedených způsobů nepomohlo. Po vyzkoušení všech tří vzorců pro porovnání barev jsem došel k nejlepším výsledkům pomocí CIEDE2000. Nesrovnalosti v osvětlení jsem se snažil odstranit pomocí adaptivního prahování, ač marně. Výsledek snímání pozice je tedy stále závislý na osvětlení.

Kapitola 5

Testování

Testování bylo nutné rozdělit do dvou částí, stejně jako tomu bylo v návrhu na testování určení nejlepšího tahu a skenování pozice. Jedná se totiž o dva rozdílné úkony a není je možné testovat naráz.

5.1 Testování určení nejlepšího tahu

Testování jsem započal hledáním nejoptimálnější možné hloubky vyhledávání pro Alfa-Betu. K prohledání největšího počtu pozic dochází při skákání kamenů, kde oběma hráčům zbývají pouze 3 kameny, při takové pozici je možné vytvořit nejvíce (54) různých tahů (testování bylo provedeno na pozici z obrázku 5.1).

Obrázek 5.1: testovaná pozice

V následující tabulce 5.1 je ukázáno, jak se při daném zanoření do určité hloubky prodlužuje čas vyhledávání s závislostí na počtu prohledávaných pozic.

Pro větší hloubku, než je hloubka 4, testování postrádalo smysl, z toho důvodu byla hloubka 4 zvolena jako maximální pro možnost hry (těžká úroveň).

Jak je uvedeno v implementaci, pro snazší testování určení nejlepšího tahu slouží třída Play, díky které si lze zahrát proti aplikaci. Testování bylo prováděno, jak s reálnými hráči, tak s aplikacemi, na kterých byla testována i detekce pozice.

První testování proběhlo na hráčích, které osobně znám. V tomto případě algoritmus zvítězil ve 100%. Nejspíše z důvodu, že hra mlýny není zase tak hranou hrou. Z toho důvodu jsem se rozhodl při testování přejít na možnost, kde jsou hráči zdatnější. Pro takové

Hloubka zanoření	Počet procházených pozic	Čas prohledávání
1	54	13ms
2	1370	118ms
3	64278	3244ms
4	1642054	82391ms

Tabulka 5.1: Časová náročnost

Úroveň obtížnosti	Úspěšnost proti ELO 0-100	Úspěšnost proti ELO 100-300	Úspěšnost proti ELO 300+
Lehká	7 z 10	4 z 10	0 z 5
Střední	10 z 10	7 z 10	2 z 5
Těžká	10 z 10	8 z 10	3 z 5

Tabulka 5.2: Úspěšnost

testování jsem si zvolil aplikaci od Boardgamearena [1]. Aplikace umožňuje hrát online proti hráčům. Velikou výhodou pro mé testování byla ukázka ELO (dovednost hráče v dané hře), díky které jsem mohl vyzkoušet aplikaci na hráčích s různou zkušeností. Testování bylo provedeno na různých úrovních obtížnosti. Úrovně jsou uvedeny v podkapitole 4.4. Při každém testování bylo vybráno pár hráčů s různým ELO. Následně podle ELO byli rozděleni do skupin (0-100, 100-300, 300+). Pro testování skupin nebyl bohužel dostatek hráčů. Výsledky testování jsou znázorněny v tabulce 5.2

Další testování bylo prováděno jen proti aplikacím. První aplikací, u které bylo testování provedeno byla implementace nalezena na webu Playpager [2]. Aplikace má zřejmě dva různé vyhodnocovací algoritmy, které se střídají mezi hrami. Pokud je v mé implementaci použita úroveň Lehká, dostaneme se do stádia, kde se pozice stále opakuje, tuto situaci jsem vyhodnotil jako remízu. Při použití úrovní Střední a Těžká má implementace vyhrává.

Jako druhá testovaná aplikace byla implementace, kterou jsem nalezl na webu Mathplayground [3]. Aplikace používá pravděpodobně měnící se ohodnocení stavů, nabývají ovšem stejné úrovně. Při použití úrovně Lehká, se dostáváme do stejné situace, jako tomu bylo při předchozí aplikace. Dostáváme se tedy do patové situace, kdybychom počítali s nejlepšími tahy aplikace. Při použití Střední a Těžké úrovně má implementace vítězí.

Pro poslední testování určení tahu jsem si zvolil inplmentaci od Smartlittlegames [4], která má mnohem více propracovaný rozhodovací algoritmus pro určení nejlepšího tahu. Umožňuje volbu několika úrovní(Novice, Beginner, Intermediate, Advanced, Expert). Při použití mé úrovně Lehká,se dostavám do patové situace proti úrovni Novice, proti zbytku prohrávám. Při zvolení mé úrovně Střední, se dostáváme do patové situace v případě nastavení soupeřovi úrovně Novice i Beginner, se zvolením vyšší úrovně prohrává. Při zvolení mé úrovně Těžká, při Novice i Beginner vyhrává mé určení tahu, při volbě Intermediate nebo posledních dvou možností Advanced a Expert má implementace prohrává.

[1] https://boardgamearena.com
[2] https://playpager.com
[3] www.mathplayground.com/logic_nine_mens_morris.html
[4] http://www.smartlittlegames.com/ninemensmorris

5.2 Testování detekce pozice

Testování detekce pozice jsem provedl na více platformách, abych vyzkoušel svůj záměr pokrytí co nejvíce desek. Pro testování byly použity celkem tři chytré telefony, aby byla porovnána úspěšnost detekce v závislosti na kvalitě zabudované kamery. Testované telefony byly LG G6, Huawei Mate 10 lite, Huawei Y6 Prime.

Fyzická deska

Při použití fyzické hrací desky (Dáma Mlýn od výrobce Bonaparte 5.2) bylo testování zkoušeno za 4 světelných podmínek. První testování proběhlo na denním světle, kde byla deska i s kameny rovnoměrně osvícena, při takovém prostředí jsem dosáhl 10 z 10 správných výsledků u všech tří testovaných zařízení.

Obrázek 5.2: Dáma Mlýn od výrobce Bonaparte (převzato z webu [5])

Druhé testování proběhlo v zatemněné místnosti za rozsvícení bílého světla, kde deska byla umístěna tak, aby při detekci nepůsobil žádný odlesk, v takovém případě jsem dosáhl stejného výsledku, jako při denním světle.

Třetí detekce probíhala v uzavřené místnosti při osvětlení se žlutým světlem. V takových podmínkách jsem na dané hrací desce nebyl schopen detekovat pozici a to z důvodu, že pod takovým osvětlením na kameře všech tří telefonních zařízení barva desky a bílých kamenů splývá a není je tak možné detekovat.

V poslední fázi bylo testování provedeno nad nerovnoměrným osvícením desky bílým světlem, kde na horní část hrací desky padal stín. Jelikož má šachovnice velice podobnou velikost políčka a hracích kamenů, vypadává možnost rozpoznání pomocí velikosti. Dostáváme se tedy do situace, kde políčko nabývá podobné barvy, jako má černý kámen a v takovém případě dochází k náhodnému rozpoznání mezi políčkem a černým kamenem. Při takové detekci jsem došel k 50% úspěšnosti s použitím LG G6. Při použití obou telefonů Huawei jsem byl schopen správně detekovat pozici jen zřídka. Nejhůře na tom byl s detekcí Huawei Y6 Prime, který má v takovéto situaci velkou potíž se vzniklými odlesky a následným porovnáním barev. Často nastala situace, kdy tímto telefonem byly detekovány kameny správně, ale barva pole byla vyhodnocena jako barva kamene a naopak. Pro vylepšení detekce hran a následně kruhů jsem se pokoušel využít adaptivní prahování. Bohužel jsem dostával spíše více šumu, čímž se detekce ještě horšila.

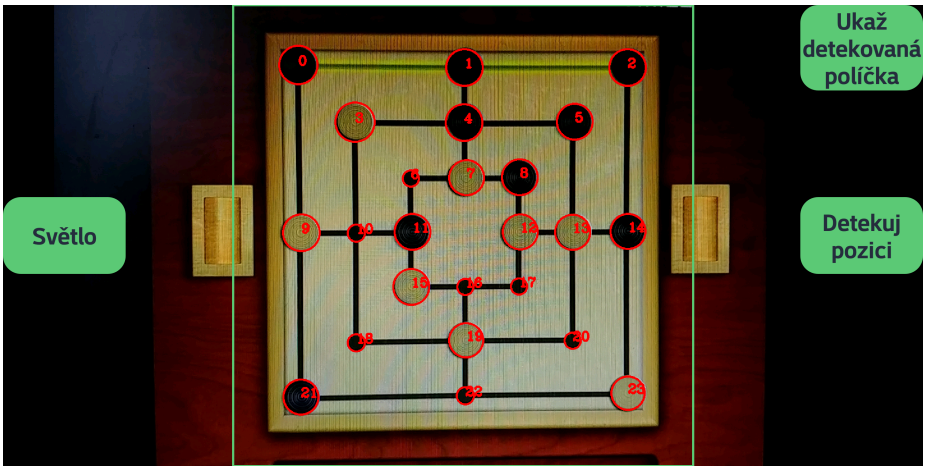
[5] https://www.papirnictvikbely.cz/obrazky/str_maly_bn07697.jpg

Jelikož bylo testování prováděno na desce s lesklým povrchem, využití světla (blesku zařízení) se nevyplatilo používat. Záleželo na odlesku, docházelo k situacím, kdy políčka nebyla detekována, nebo díky nerovnoměrnému osvětlení došlo ke špatné detekci barvy. Světlo by mohlo být využito na šachovnicích s nelesklým povrchem, případně na deskách s rozdílnou velikostí polí a kamenů, či deskách, kde se barva políček a kamenů každého hráče výrazně liší. Problém s odleskem jsem se snažil vyřešit použitím metody adaptivní vyrovnávání histogramu s omezeným kontrastem (CLAHE), ovšem problém se mi vyřešit nepodařilo.

Detekce z monitoru

Detekce byla zkoušena na více aplikacích, kde nejlepší detekce lze dosáhnout v aplikaci od Smartlittlegames [6], kde při testování dosahuji 10 z 10 správných výsledků u LG G6 a Huawei Mate 10 lite. Huawei Y6 Prime chytá z monitoru příliš mnoho šumu a je trochu potíž najít správnou pozici, kdy je šum odstraněn. Pro detekci z monitoru je detekce ve většině případů přesnější z důvodu možnosti použití dvou faktorů (velikost políčka + barev kamene). Dalším důvodem je rovnoměrné osvětlení. Veliký dopad má ovšem rozlišení monitoru, kde čím menší rozlišení, tím horší obraz a tím vzniká i více šumu v obraze.

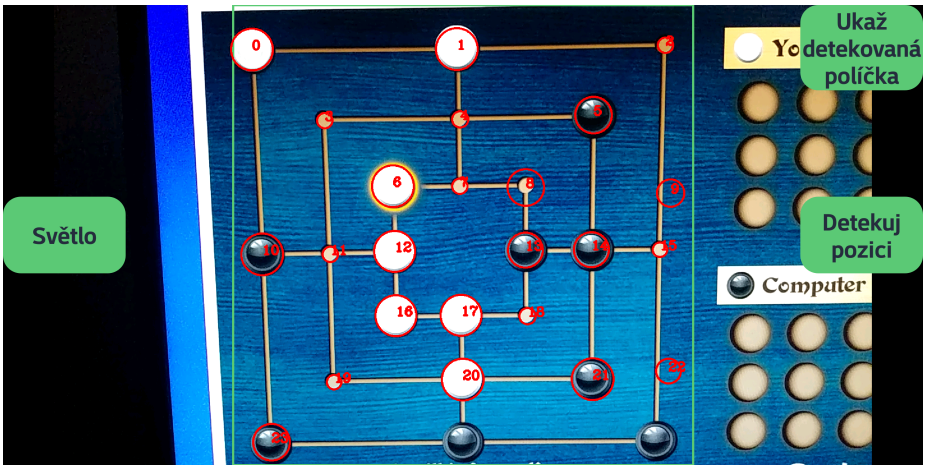
Druhá zkoušená aplikace je z portálu Playpager [7]. Zde je zapotřebí si aplikaci zobrazit v režimu celé obrazovky a kameru namířit co nejvíce přímo to jde (v tomto případě je lepší si nechat zobrazit, co je detekováno pomocí tlačítka ukaž detekovaná políčka), viz. obrázek 5.3. Ovšem z důvodu textury černých kamenů dochází k zaměnění políčka za kámen. Ve většině případů se tak stane pouze u jednoho kamene. Úspěšnost detekce je 40% při použití LG G6, kde chyba v jednom kamenu byla v 5 případech. Při použití Huawei Mate 10 lite jsem překvapivě dosáhl lepších výsledků, úspěšnost dosahovala 100%. Ovšem Huawei Y6 Prime zde nebyl schopný detekci uskutečnit, při detekci snímal příliš mnoho šumu. Chybu se mi nepodařilo eliminovat z důvodu , že při zvýšení filtru pro odstranění šumu dochází k ignoraci polí.



Obrázek 5.3: detekce playpager

[6] http://www.smartlittlegames.com/ninemensmorris
[7] https://playpager.com/play-mill/index.html

Na aplikaci od Mathplayground [8] se detekci daří provést zřídka až na poslední Huawei Y6 Prime, který zde opět není schopný detekovat pozici. Dochází k tomu díky zvolenému pozadí hrací desky, které vytváří velké množství šumu, který pokud se snažím odstranit přicházím i o nalezená políčka. Nacházení falešných políček, díky výraznému šumu je vidět na obrázku 5.4.



Obrázek 5.4: detekce Mathplayground

[8] https://www.mathplayground.com/logic_nine_mens_morris.html

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vytvořit mobilní aplikaci pro načtení pozice pomocí kamery zařízení a následné určení nejlepšího tahu. Zvolenou platformou pro vývoj byl operační systém Android. K dosažení žádaného cíle bylo nutné se seznámit se samotnou hrou a jejími pravidly, dále se základy teorie her. Druhou částí problematiky bylo počítačové vidění a témata s ním spojené, které byly potřebné pro řešení práce. Mezi taková témata patří detekce hran, Houghova transformace, barevné modely a operace nad nimi, jako jsou převody mezi modely, porovnání barev a další.

Při návrhu celé aplikace byla práce rozdělena na tři části. První část se věnovala návrhu ohodnocení pozice, kde byl kladen důraz na zefektivnění co nejvíce pravidel a výhod, kterých může pozice nabídnout. Druhou částí byla detekce pozice. Při návrhu detekce bylo zjištěno, že existuje veliké množství různých zhotovení, či implementací samostatné hry. Kvůli podobnosti tvaru políček a kamenů ve většině implementací byla zvolena detekce kružnic a následné porovnání jejich poloměrů a barev. Poslední částí návrhu byla grafická stránka aplikace. Při návrhu byla využita pravidla Material designu, kterými se drží většina dnešních Android aplikací.

Pro implementaci aplikace bylo využito vývojové prostředí Android Studio, které je oficiálním vývojovým prostředím pro systém Android. Celá aplikace je naprogramována v jazyce Java za pomocí knihovny pro počítačové vidění a strojové učení OpenCV. Při implementaci byla také přidána možnost zahrát si danou hru a to hlavně z důvodu jednoduššího testování určení nejlepšího tahu. Celá implementace byla zálohována pomocí verzovacího systému Git.

Výsledná aplikace je funkční, ovšem existují omezení pro detekci pozice pomocí kamery. Nastává problém při vzniklém odlesku, nerovnoměrném osvětlení hrací desky, či použití různých zařízení. Další vývoj v tomto směru by se mohl zabývat odstraněním těchto problémů. Řešení by mohla být například kalibrace kamery, či využití algoritmů pro odstranění odlesku. Pokusy o odstranění odlesku byly provádění s použitím algoritmu pro vyrovnávání kontrastu CLAHE, vlastní detekcí odlesku a jeho eliminací, či snahou zlepšit alespoň detekci hran při výskytu odlesku pomocí adaptivního prahování. Ovšem všechny pokusy o odstranění odlesku byly neúspěšné. Následné ohodnocení pozice by bylo možné rozšířit o učící se neuronovou síť, čímž by došlo k ještě lepším výsledků, než jsem došel s aktuální implementací.

Literatura

[1] Hough Line Transform. https://docs.opencv.org/3.4.3/d3/de6/tutorial_js_houghlines.html, cit. 2019-19-03.

[2] Platform Architecture. https://developer.android.com/guide/platform, cit. 2019-29-03.

[3] Useful Color Equations. http://www.brucelindbloom.com/, cit. 2019-25-04.

[4] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

[5] Carlson, C. F.: Hough Circle Transform. https://docs.opencv.org/trunk/d4/d70/tutorial_hough_circle.html, cit. 2019-19-03.

[6] Gasser, R.: Solving nine men's morris. *Computational Intelligence*, ročník 12, č. 1, 1996: s. 24–41.

[7] Ibraheem, N. A.; Hasan, M. M.; Khan, R. Z.; aj.: Understanding color models: a review. *ARPN Journal of science and technology*, ročník 2, č. 3, 2012: s. 265–275.

[8] Jain, R.; Kasturi, R.; Schunck, B. G.: *Machine vision*, ročník 5. McGraw-Hill New York, 1995.

[9] Kang, H. R.: *Computational color technology*. Spie Press Bellingham, 2006.

[10] Ketcham, D. J.; Lowe, R. W.; Weber, J. W.: Image enhancement techniques for cockpit displays. Technická zpráva, HUGHES AIRCRAFT CO CULVER CITY CA DISPLAY SYSTEMS LAB, 1974.

[11] Nowakowski, R.; Levy, S.: *More games of no chance*, ročník 42. Cambridge University Press, 2002.

[12] Nowakowski, R. J.: *Games of no chance*, ročník 29. Cambridge University Press, 1998.

[13] Peregrin, J.; Materna, P.: Detekce hran. http://midas.uamt.feec.vutbr.cz/ZVS/Exercise08/content_cz.php, cit. 2019-25-03.

[14] Pizer, S. M.; Amburn, E. P.; Austin, J. D.; aj.: Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, ročník 39, č. 3, 1987: s. 355–368.

[15] Sezgin, M.; Sankur, B.: Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, ročník 13, č. 1, 2004: s. 146–166.

[16] Sharma, G.; Wu, W.; Dalal, E. N.: The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur*, 2005.

[17] Straka, S.: *Segmentace obrazu.* Dizertační práce, Masarykova univerzita, Fakulta informatiky, 2009.

[18] Wolfson, M.; Felker, D.: *Android Developer Tools Essentials: Android Studio to Zipalign.* "O'Reilly Media, Inc.", 2013.

[19] Zboril, F: IZU - opora.

Příloha A

Obsah přiloženého paměťového média

- doc/ - Složka obsahuje dokumentaci a její zdrojové soubory
  - tex/ - zdrojové soubory k této dokumentaci
  - xkolin05_BP.pdf - tato bakalářská práce ve formátu PDF
- app/ - Složka obsahuje aplikaci a její zdrojové soubory
  - src/ - zdrojové soubory k výsledné aplikaci
  - NineMensMorris.apk - výsledná aplikace