VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MOBILNÍ APLIKACE PRO DEMONSTRACI ŘEŠENÍ DESKOVÝCH HER

SUPPORT FOR PLAYING BOARD GAMES DEMONSTRATION ON MOBILE PLATFORMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ ŠIROKÝ**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Široký Ondřej**
Program: Informační technologie
Název: **Mobilní aplikace pro demonstraci řešení deskových her**
Support for Playing Board Games Demonstration on Mobile Platforms
Kategorie: Umělá inteligence
Zadání:

1. Zvolte a seznamte se s pravidly několika (tří či více) populárních deskových her. Dále se seznamte s metodami zpracování obrazu, které by umožnily ze snímků hrací desky zjistit typ a aktuální stav hry.
2. Navrhněte aplikaci, která na základě těchto snímků navrhne další tah, případně analyzuje odehrané tahy ze sekvence snímků.
3. Implementujte navržené řešení pro systém Android.
4. Vyzkoušejte výslednou aplikaci na dostatečném počtu osob (alespoň šest) a shrňte jejich názory, postřehy a kritiky. Na základně tohoto navrhněte další vylepšení aplikace, pokud bylo požadováno, a diskutujte užitečnost systému pro širší veřejnost.

Literatura:

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

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Cílem této bakalářské práce je vytvořit mobilní aplikaci na platformu Android, která pomůže hráči určit nejlepší možný tah z pořízené fotografie, která zachycuje některý z několika typů deskových her. Aplikace ze snímku pomocí funkcí knihovny OpenCV identifikuje hru a extrahuje její aktuální stav. Tyto informace jsou poté zobrazeny uživateli, který má možnost stav hry ručně upravit, jestliže došlo k nepřesnému načtení stavu hrací desky. Následně aplikace pomocí Minimax algoritmu s Alfa-beta ořezáváním určí nejlepší tah a zobrazí jej uživateli.

Abstract

The aim of this bachelor's thesis is to create a mobile application for the Android platform that will help the player determine the best possible move from the captured image. The application identifies the game and extracts its current state by using OpenCV library functions. This information is then displayed to the user, who has the ability to manually adjust the game status if the game board is inaccurately read. Subsequently, the application uses the Minimax algorithm with Alpha-beta pruning to determine the best move and displays it to the user.

Klíčová slova

Dáma, Gomoku, Mlýny, Android, mobilní aplikace, zpracování obrazu, počítačové vidění, OpenCV, Houghova transformace, MiniMax, Alfa-beta ořezávání

Keywords

Checkers, Gomoku, Nine men's morris, Android, mobile application, image processing, computer vision, OpenCV, Hough transform, MiniMax, Alpha-beta pruning

Citace

ŠIROKÝ, Ondřej. *Mobilní aplikace pro demonstraci řešení deskových her*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Zbořil František, Ph.D.

Mobilní aplikace pro demonstraci řešení deskových her

Prohlášení

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

.....
Ondřej Široký
11. května 2021

Poděkování

Děkuji panu Doc. Ing. Františku Zbořilovi, Ph.D za pomoc, cenné rady a čas, který mi věnoval při psaní této bakalářské práce. Děkuji také všem, kteří mě při psaní této bakalářské práce podporovali.

Obsah

Kapitola 1

Úvod

Ačkoliv v poslední době nabírá na popularitě především hraní videoher, nemůžeme opomenout ty klasické deskové hry. Deskové hry jsou v určité podobě s námi již tisíce let. Každý se s nimi minimálně v dětství setkal a mnohým tento druh zábavy zůstal i v dospělosti. Pro některé jsou hry více než zábavou a rádi pronikají hlouběji do samotných herních mechanik. Pro mě osobně je u her velice zajímavá i technologická či strategická stránka. Algoritmy pro umělou inteligenci soupeře jsou nedílnou součástí jakékoliv hry, kde protivníkem není jiný živý hráč. Přestože jsou již na trhu deskové hry, které svým zpracováním připomínají videohry, nesmíme zapomínat ani na klasické hry, jako jsou Šachy, Go, Gomoku, Dáma či Mlýny. Posledním třem zmíněným se tato práce věnuje.

Informační technologie jsou všude kolem nás a každodenní život si bez nich nedokážeme představit. Počítačové vidění patří mezi obory IT, které pronikají do každodenního života čím dál více. Počítačové vidění můžeme dnes najít takřka všude. V lékařství pomáhá s diagnózou nemocí. Bezpečnostním složkám dokáže počítačové vidění pomoci odhalit hledaného člověka nebo hrozbu na bezpečnostních záznamech či identifikovat poznávací značku rychle jedoucího auta. Z druhé strany pohledu pak nalezneme počítačové vidění i u moderních automobilů, kde tato technologie napomáhá k bezpečnosti či automatizaci řízení. Možnosti využití tohoto oboru jsou obrovské.

Počítačové vidění mě velice zaujalo a jelikož volný čas často trávím u her s přáteli, rozhodl jsem se spojit příjemné s užitečným tím, že vytvořím mobilní aplikaci, která spadá do obou kategorií. Aplikací, které využívají počítačem řízeného soupeře je mnoho. Já jsem se rozhodl uchytit problém z druhé strany a metodami pro hraní her poskytnout pomoc samotnému hráči. Cílem práce je vytvořit mobilní aplikaci pro systém Android, která pomocí snímku rozpozná podporovanou hru, identifikuje její aktuální stav a doporučí hráči, jak nejlépe táhnout.

Práce jako taková se skládá ze čtyř kapitol teoretické části, kde se čtenář může seznámit s potřebnou problematikou. Kapitola 2 se věnuje samotným hrám a jejich pravidlům. Následuje kapitola o teorii hraní her, kde jsou blíže popsány použité algoritmy k určení nejlepšího tahu. Kapitola 4 se věnuje počítačovému vidění a popisuje metody používané ke zpracování obrazu. Poslední teoretickou částí je samotný operační systém Android, pro který je tato aplikace vyvíjena.

V následujících dvou kapitolách je popsán samotný návrh a implementace GUI, funkcí pro rozpoznaní her a ohodnocovací funkce pro určení nejlepšího tahu. Poslední kapitolou před závěrem je samotné testování jednotlivých částí aplikace a shrnutí zpětné vazby od uživatelů.

Kapitola 2

Hry podporované aplikací

Různých druhů deskových her existuje na světě velice mnoho. V této kapitole jsou popsána pravidla tří her, která moje aplikace podporuje. Konkrétně to jsou hry Dáma, Mlýny a Gomoku.

2.1 Hra Dáma

Hra dáma, v anglickém jazyce známá pod jménem *Checkers*, či v britské angličtině *Draughts*, je desková hra, v níž se dva hráči utkávají na čtvercové šachovnici, a to nejčastěji velikosti 8x8 nebo 10x10. Hráči mají kameny rozdílné barvy, standardně bílé a černé, které jsou na začátku hry rozestavěny proti sobě na koncích herní desky. Hráči se ve svých tazích střídají a pohyb kameny je povolen pouze diagonálně. Hraje se po celou dobu výhradně na tmavých polích šachovnice. Hrací figurky se rozlišují na základní kameny, se kterými hráč začíná, a dámy, které hráč získá přesunutím základního kamene na protější konec hrací plochy. Kameny se mohou pohybovat pouze o jedno pole vpřed směrem k soupeři. U dámy toto omezení neplatí a může se pohybovat i opačným směrem. Další možnosti pohybu dámy pak závisí na konkrétních pravidlech. K sebrání soupeřových kamenů je nutné daný kámen přeskočit, musí být tedy v diagonálním směru za branným kamenem volné políčko. Vítězem se stává hráč, který soupeři sebere všechny jeho kameny, případně je zablokuje tak, aby protihráč nemohl provést žádný tah.

Jelikož je hra dáma velice oblíbenou hrou po celém světě, je evidováno více než 150 různých variací pravidel. Níže jsou zmíněná pravidla u nás velice rozšířené české varianty dámy a pravidla varianty anglické, kterou jsem se rozhodl implementovat, a jejíž pravidla jsou níže vysvětlena v porovnání s českými. [14]

Česká dáma

Pravidla české varianty jsou definována Českou federací dámy. Hra probíhá na čtvercové desce o velikosti 8x8 a každý hráč má po 12 kamenech. Braní soupeřových kamenů je povinné a hráč nesmí táhnout jinak, pokud má možnost sebrat soupeřův kámen. Je také dovoleno v jednom tahu brát více kamenů, jestliže to jejich rozložení dovoluje. Pokud může hráč brát zároveň kamenem i dámou, musí brát dámou. Dáma nemá omezený pohyb na vzdálenost jedna a může se tak pohybovat o libovolný počet polí, a to i při přeskakování. [17]

Obrázek 2.1: Ukázka deskové hry Dáma s kameny v počáteční poloze

Anglická dáma

Jedná se o nejhranější variantu dámy. Anglická verze hry má s českými pravidly dost podobností. Prvním drobným rozdílem, který však nijak neovlivňuje hraní hry, je pojmenování samotné Dámy. Dáma totiž v anglické variantě nese název král. Král se stejně jako česká dáma smí pohybovat oběma směry. Znatelným rozdílem oproti české verzi je pak to, že král se smí pohybovat pouze o jedno pole stejně jako běžné kameny. Také v anglických pravidlech dámy nenajdeme nic o přednosti v braní soupeřových kamenů. Braní je sice povinné stejně jako je tomu v české variantě, ale pokud má hráč možnost brát králem i kamenem, je na něm, který tah zvolí. [14]

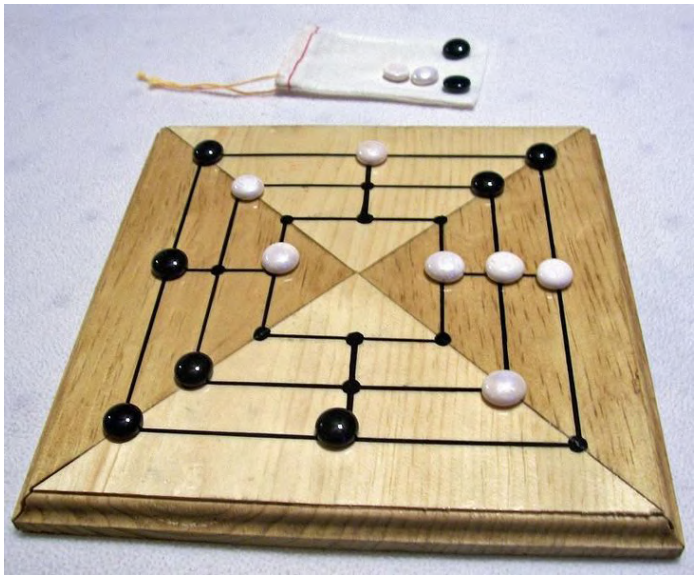
2.2 Hra Mlýny

Hra mlýny je jednou z nejstarších her, které se těší popularitě i dnes. Historie této hry sahá až do starověkého Egypta, kde byla nalezena prozatím nejstarší deska této hry, která byla vytesána do střešní břidlice tamního chrámu. Hra byla také silně rozšířena v Cejlonu, Troji či Irsku. [14]

Hra mlýny se vyskytuje v několika různých variantách. Tyto varianty se od sebe liší převážně počtem kamenů, počtem hracích polí a rozložením spojnic mezi nimi. Existují verze s 9, 16 a 24 poli. Nejrozšířenější je 24 políčková varianta, kde hráči mají každý po 9 kamenech. V angličtině je kromě pojmenování *Mills* hra také nazývaná jako kombinace počtu kamenů a slovního spojení *men's morris*. Nejznámější varianta je tedy v anglickém jazyce známá také pod názvem *Nine men's morrris*. Právě této variantě se ve své práci věnuji a její pravidla jsou popsána níže.

Hrací desky můžeme najít v různém vyhotovení, které se liší různým vyobrazením políček. Ty jsou ve většině případů menší jak samotné kameny. Existují ale i varianty, kde políčka nejsou zobrazena vůbec a deska se tak skládá pouze z čar. Na obrázku 2.2 můžeme vidět jedno z vyhotovení hry Mlýny.

Hra samotná je rozdělena do dvou fází.



Obrázek 2.2: Vyhotovení hry Mlýny s políčky menšími, než jsou herní kameny [1]

Fáze umísťování kamenů

Hra startuje fází umísťování kamenů, ve které se začíná s prázdnou herní deskou. Hráči ve střídavých tazích umisťují všech svých 9 kamenů na volné pole herní desky. Umístění tří kamenů stejné barvy do jedné horizontální či vertikální linie vzájemně propojených bodů se nazývá uzavření mlýnu. Hráč, který takto uzavře mlýn, může soupeři odebrat z desky jeden z jeho kamenů. Pokud je to možné, hráč musí odebrat kámen, který není součástí soupeřova mlýnu. Jestliže jsou všechny soupeřovy kmeny součástí mlýnu, může vybrat libovolný z nich. S odebranými kameny se již nehraje. Po umístění všech 18 kamenů následuje druhá fáze.

Fáze přesouvání kamenů

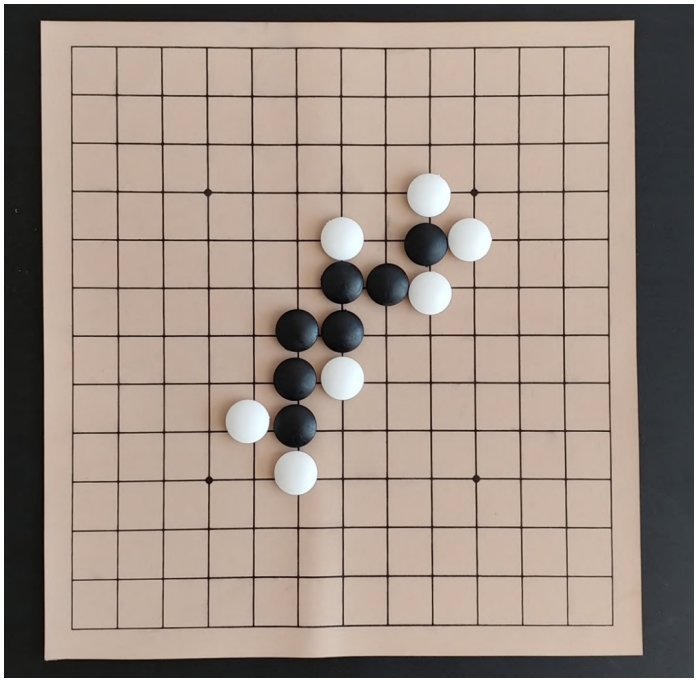
Stejně jako v první fázi se hráči střídají. Hráč na tahu musí posunout po spojnici svým kamenem na některé ze sousedních volných polí. Pokud dojde při tomto pohybu k vytvoření mlýnu, hráč odebírá kámen soupeři podle stejných pravidel, jako tomu bylo v předchozí fázi. Stejné mlýny mohou být uzavírány opětovně. Lze tedy odsunout kámen z mlýnu a v následujícím tahu mlýn znovu uzavřít. Jestliže dojde k situaci, kdy hráč nemá vedle svých kamenů žádné volné pole, kam by mohl táhnout, prohrává. Hráči, kterému zůstávají pouze tři kameny, je umožněno skákání. To znamená pohyb na libovolné volné pole. Uzavře-li hráč mlýn v situaci, kdy má soupeř jen 3 kameny, stává se vítězem.

2.3 Hra Gomoku

Gomoku, též známá pod jménem Renju, je desková hra, která využívá herní desku a kameny ze hry Go. Vznik hry samotné se odhaduje kolem roku 2000 př. n. l, čemuž nasvědčují nálezy v oblasti okolo řeky Hwang Ho v Číně. Její název je odvozen ze dvou částí. "Go"v tomto případně znamená 5 a "moku"je označení pro kámen. Jak již z tohoto názvu vyplývá, cílem hry je spojit pět svých kamenů dříve, než tak učiní soupeř. Spojit kameny je možno

[1] Obrázek 2.2 převzat z webu https://boardgamegeek.com/image/438639/nine-mens-morris

vertikálně, horizontálně i diagonálně a hráči tyto kameny pokládají střídavě na průsečíky čar herní desky. Hra se dá hrát na deskách různé velikosti, jedna je k vidění na obrázku 2.3. Nejčastěji je využíváno velikostí 13x13, 15x15 či 19x19. Jelikož ve hře nedochází k odebírání kamenů, je možné ke hraní využít i obyčejný papír a tužku. Tato varianta je velice známá a u nás ji můžeme znát pod jménem Piškvorky. [11]



Obrázek 2.3: Ukázka rozehrané partie Gomoku na desce 13x13

Výzkum hry Gomoku matematicky dokázal, že začínající hráč má prokazatelnou výhodu a téměř vždy vyhraje. Samozřejmě zkušenější hráč porazí toho slabšího za jakékoliv situace. Pokud ale dojde ke střetu hráčů stejné úrovně, vyhrává vždy ten, kdo byl první na tahu. Díky tomuto objevu bylo do turnajových klání zavedeno několik různých pravidel pro začátek hry, konkrétně:

1. Pravidlo Pro
    - **Pro**
        - První hráč na tahu musí umístit kámen do středové oblasti desky 5x5,
        - druhý hráč nemá žádné omezení pro jeho tah,
        - první hráč ve svém druhém tahu musí umístit kámen mimo středovou oblast 5x5, což zajistí vyrovnanější hru,
        - i přes toto omezení je v této variantě druhý hráč mírně znevýhodněn.
    - **Long Pro**
        - Oproti původním pravidlům **Pro** je počáteční oblast zvětšena na 7x7,
        - tímto rozšířením dochází takřka k vyrovnání obou stran.
2. Pravidlo Swap
    - **Swap**

  - V této variantě první hráč položí dva černé a jeden bílý kámen kamkoliv na herní plochu,
  - druhý hráč si může vybrat, zda si chce ponechat bílé kameny a pokračovat položením druhého bílého kamene, nebo si vezme kameny černé a přenechá bíle prvnímu hráči,
  - tato varianta je mnohem vyrovnanější než předešlé zmíněné, jelikož začínajícího hráče nutí nevytvářet výhodně kombinace.
- **Swap 2**
  - Toto počáteční pravidlo vychází z výše zmíněného pravidla **Swap**,
  - druhý hráč má v této variantě na výběr ještě třetí možnost a tou je položit navíc ještě jeden kámen od každé barvy a volbu strany přenechat soupeři,
  - jedná se o nejpoužívanější pravidlo na turnajích.[4]

Kapitola 3

Metody pro strojové hraní her

Teorie her je obor aplikované matematiky zabývající se analýzou situací, kde dva nebo více hráčů činí rozhodnutí, která jsou na sobě závislá. Tato závislost vede k tomu, že hráči činí strategická rozhodnutí v reakci na rozhodnutí protihráčů. Výsledkem analýzy je pak řešení, které popisuje optimální rozhodnutí hráčů, které mohou mít podobné, protichůdné, nebo smíšené zájmy. Jedním z pododvětví teorie her jsou hry kombinatorické, kterým se věnuje následující podkapitola. [2]

3.1 Kombinatorické hry

V kombinatorických hrách proti sobě hrají dva hráči, kteří se ve svých tazích střídají. Jedná se o takzvané hry s nulovým součtem (zero-sum games). Pro vysvětlení to znamená, že zisk jednoho z hráčů znamená ztrátu pro druhého hráče. Dále pak kombinatorické hry musí mít jasně určená pravidla, oba hráči v nich při svém tahu mají přesné informace o stavu hry a hra neobsahuje žádné prvky náhody (hod kostkou atd.). Kombinatorické hry také musí mít konec, tedy musí být možné tuto hru dohrát v konečném počtu tahů. Výsledkem hry je výhra jednoho hráče a prohra druhého, nebo remíza. Stavy konce hry bývají definovány určitými pravidly. Prohra obvykle nastává, jestliže byly jednomu hráči odebrány všechny jeho herní kameny, bylo mu znemožněno jakéhokoliv dalšího pohybu (např. dáma), nebo došlo ke splnění určité závěrečné podmínky (například šach mat ve hře šachy). Remízy docílíme tím, že ani jeden z hráčů již nemůže udělat nic, co by hru proměnilo v jeho prospěch, případně bude použito pravidlo zabraňující nekonečnosti hry. Toto pravidlo obvykle spočívá v omezení počtu tahů, ve kterých nedojde ke změně stavu hry v prospěch některého z hráčů. Kombinatorické hry lze dělit na "jednoduché“ a "složité". U jednoduchých můžeme bez problémů prohledávat celý stavový prostor. U složitých by prohledávání všech možných stavů bylo nereálné. Pro určení optimálního tahu her, u kterých je nemožné prohledat graf do větších hloubek, můžeme využít například algoritmus Minimax. [8][22]

3.1.1 Algoritmus Minimax

Jedná se o základní algoritmus k určení optimálního tahu u her pro dva hráče, který pracuje rekurzivně s informacemi o aktuálním stavu hry. Při jeho prvním volání je nastavena hloubka (počet tahů, který se má simulovat), je předán aktuální stav hry, určen hráč na tahu s označením Max a jeho soupeř je označován jako Min. V algoritmu se počítá s tím, že oba hráči budou volit nejlepší možné tahy. Tahy jsou tedy střídavě prováděny a ohod-

nocovány. Pokud je zrovna na tahu hráč Max, algoritmus vybírá stav s největší hodnotou. U hráče Min naopak vybírá hodnotu nejmenší.

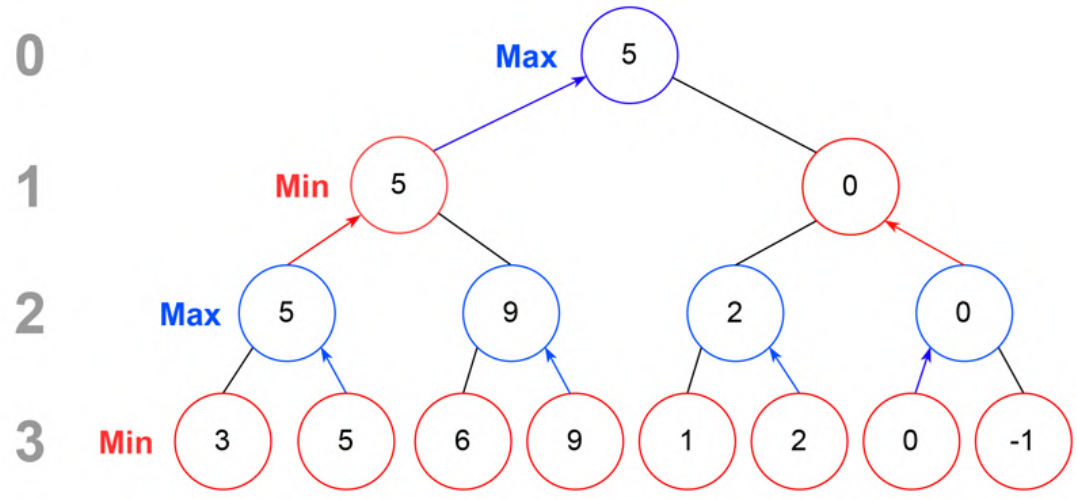
Tyto hodnoty se vypočítávají pomocí ohodnocovací funkce. Ta bere v potaz důležitost různých vlastností dané hry a na základně ní je vráceno výsledné číselné ohodnocení aktuálního stavu hry. Při dosažení výhry je hodnota nastavena na maximální možnou, u prohry naopak nejnižší možnou hodnotu. Výsledek algoritmu je nejlepší možný tah pro hráče Max, tedy takový tah, který získal nejvyšší ohodnocení. Ohodnocovací funkce použité v mé bakalářské práci jsou popsané v návrhu aplikace, konkrétně v podkapitole 6.2.

Samotný postup algoritmu probíhá následovně:

1. Jestliže je dosažena maximální dovolená hloubka, nebo nastala některá z konečných podmínek (výhra, prohra nebo remíza), je vráceno ohodnocení daného listu.

2. Je-li je na tahu hráč Max, projdi postupně všechny jeho možné tahy z aktuálního stavu hrací desky (bezprostřední následníky předchozího listu) a zavolej Minimax pro hráče Min. Návratová hodnota se rovná maximální nalezení hodnotě, tedy nejlepšímu možnému tahu hráče Max.

3. na tahu hráč Min, projdi postupně všechny možné taky hráče Min, které je možné provést po předchozím táhnu hráče Max. Vrať nejmenší nalezené ohodnocení, tedy nejlepší možný tah hráče Min. [22]



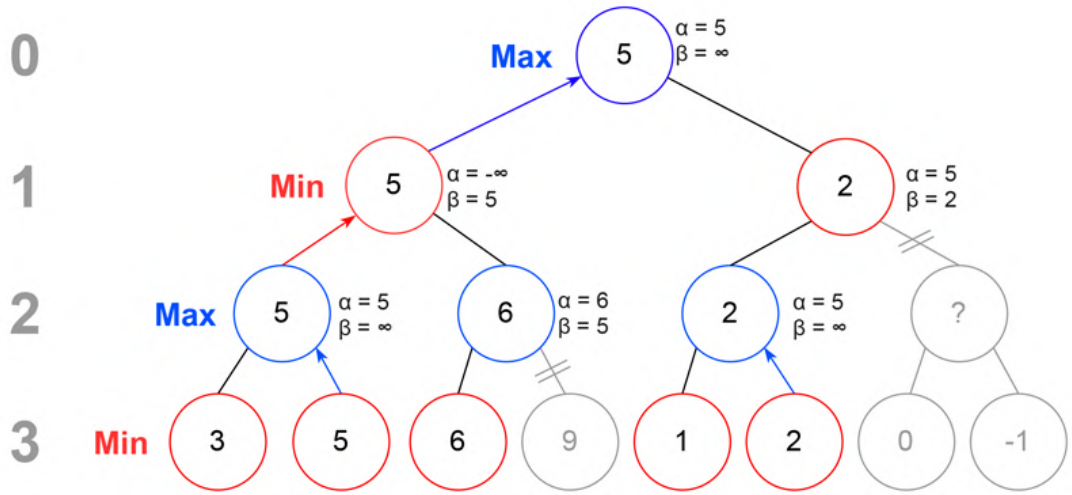
Obrázek 3.1: Ukázka algoritmu Minimax hloubky 3

Na obrázku 3.1 můžeme vidět strom algoritmu Minimax o hloubce 3. Jelikož Minimax je jednoduchý a nevyužívá žádná optimalizační pravidla, musí algoritmus projít všechny listy stromu. Tento přístup je značně neefektivní, protože prochází listy, u kterých je předem jasné, že nebudou použity. Aby se zbytečnému procházení předešlo, byl algoritmus rozšířen o optimalizační řešení nazývané Alfa-beta ořezávání.

3.1.2 Minimax s Alfa-beta ořezáváním

Alfa-beta ořezávání je optimalizační metoda pro dříve zmíněný algoritmus Minimax. Algoritmus je obohacen o dvě proměněné alpha a beta, podle čehož je i metoda pojmenována. α slouží k optimalizaci vyhledávání tahu maximalizujícího hráče, β pak optimalizuje vyhledávání minimalizujícího hráče. Při prvotním volání se nastavují hodnoty alpha na -∞ a beta na $+\infty$. Díky tomuto nastavení dojde vždy hned při prvním ohodnocovaní hráčova tahu k přepsání. Do hodnoty α se ukládá dosud nejvyšší nalezená hodnota. Beta naopak reprezentuje nejnižší nalezenou hodnotu. Díky těmto hodnotám lze ukončit prohledávání určité větve předčasně, jelikož víme, že byl nalezen lepší tah již dříve. Postup algoritmu vypadá následovně:

1. Jestliže je maximální dovolená hloubka, nebo nastala některá z konečných podmínek (výhra, prohra nebo remíza), je vráceno ohodnocení daného listu.
2. Je-li na tahu hráč Max:
   (a) Platí-li $\alpha \geq \beta$ nebo již neexistuje další tah z aktuálního stavu, vrať aktuální hodnotu α a tah odpovídající tomuto ohodnocení. Jestliže je více tahů ohodnoceno stejně, bere se v potaz pouze první nalezený.
   (b) Platí-li $\alpha < \beta$, volej další bezprostřední tah s aktuálními hodnotami α a β. Po každém ohodnocení tahu zjisti, zda je návratová hodnota větší než α, pokud ano přepiš hodnotu α navrácenou hodnotou a tah si pamatuj jako nejlepší.
3. Je-li je na tahu hráč Min:
   (a) Platí-li $\alpha \geq \beta$ nebo již uzel nemá bezprostředního následníka, vrať aktuální hodnotu β.
   (b) Platí-li $\alpha < \beta$, volej další možný tah s aktuálními hodnotami α a β. Po každém ohodnocení tahu zjisti, zda je návratová hodnota menší než β, pokud ano přepiš hodnotu β navrácenou hodnotou. [22]



Obrázek 3.2: Ukázka algoritmu Minimax s Alfa-beta ořezáváním

Na obrázku 3.2, který demonstruje ořezávání Alfa-beta, si můžeme všimnout šedých listů s přeškrtnutou cestou. To znázorňuje neprozkoumané listy, které byly ořezány metodou Alfa-beta. Na obrázku lze vidět, že hodnota 9 v hloubce tři není vůbec prohledávána. To je zapříčiněno tím, že v hloubce dva je na tahu maximalizující hráč, který hledá maximální hodnotu, a v předešlém prohledávání na stejné úrovni nalezl hodnotu 6. Každá další hodnota, která by ovlivnila rozhodnutí, může být jedině větší. Z toho vyplývá, že není nutné dále prohledávat tahy z aktuálního stavu, jelikož hráč Min vybere hodnotu nejmenší. Tu již má v jiném listu, protože $5 < 6$.

Podobný případ můžeme vidět v pravé časti stromu, kde v hloubce 1 vybíráme minimální možnou hodnotu. Jelikož jsme dostali hodnotu 2, která je menší než již předem nalezená hodnota 5, nemá smysl dále hledat menší hodnoty, jelikož v hloubce 0 volíme maximální možnou hodnotu, kterou zde je hodnota 5.

Kapitola 4

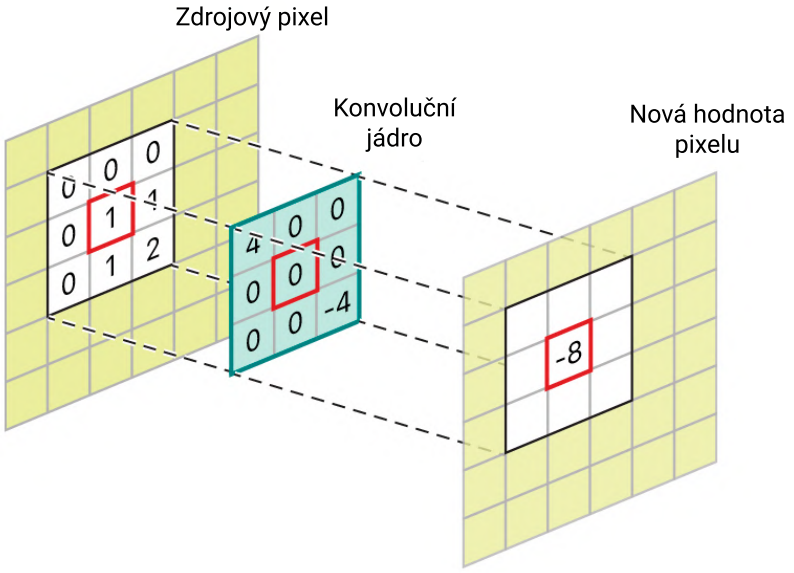
Počítačové vidění

Informace pro tuto kapitolu jsem čerpal z publikací *Computer Vision: algorithms and applications* [20] a *Image Processing, Analysis, and Machine Vision* [19].

Lidé vnímají svým okem trojrozměrnost světa kolem nás bez větších obtíží. Cílem počítačového vidění je co nejlépe tuto schopnost lidského oka duplikovat. Pro počítač to ale dokáže být obtížné a algoritmy jsou často náchylné k chybám, protože kamery ve většině případů zachycují 3D svět jako 2D obraz, čímž dojde ke ztrátě mnoha důležitých informací. Samotné zpracování obrazu můžeme v oboru počítačového vidění rozdělit na mnoho částí. Níže jsou uvedeny základní informace nutné k pochopení problému řešeného v této práci. Informace jsou rozdělené do tří částí, a to:

- Předzpracování obrazu
- Segmentace obrazu
- Extrakce příznaků

Než se ale pustím do popisu samotných způsobů zpracování obrazu, je třeba si definovat pojem konvoluce obrazu, který se v popisu metod vyskytuje. Jednoduše řečeno konvoluce obrazu je proces, při kterém máme velice malou matici nazývanou konvoluční jádro. Touto maticí poté vynásobíme každý pixel a jeho okolí v matice reprezentující původní obraz. V závislosti na hodnotách konvolučního jádra tak dokážeme výsledný obraz vyhladit, doostřit, najít hrany a podobně. Pro lepší pochopení je princip konvoluce znázorněn na obrázku 4.1.



Obrázek 4.1: Grafická reprezentace 2D diskrétní konvoluce [1]

Matematická formulace 2D diskrétní konvoluce je dána vztahem 4.1, kde x reprezentuje matici vstupního obrazu, h je matice jádra konvoluce a y je nově vzniklá matice.

$$y[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m,n] \cdot x[i-m, j-n] \tag{4.1}$$

4.1 Předzpracování obrazu

Digitální obraz, který získáme snímáním, může být vlivem okolních podmínek nebo typem snímání značně zkreslen. O potlačení těchto nežádoucích jevů, jako je například šum, se stará právě předzpracování obrazu. Metody předzpracování obrazu ale nemusí nutně jen odstraňovat nežádoucí vlastnosti obrazu, mohou naopak některé informace zvýrazňovat, pokud s těmito informacemi pracuje některý z dalších procesů zpracování. Tyto metody ovšem nijak nezvyšují počet informací, které se v obraze nachází, naopak je většinou snižují. To znamená, že lze pracovat pouze s informacemi, které jsou již ve snímku obsaženy před samotným předzpracováním.

Metody pro předzpracování obrazu lze rozdělit různými způsoby. V této práci vycházím z rozdělení, které je zmíněno v knize *Image Processing, Analysis, and Machine Vision* [19]. Rozdělení v této knize je dle velikosti okolí pixelů, která je použita pro výpočet jasu pixelu nového. Toto rozdělení vypadá následovně:

- jasové transformace,
- geometrické transformace,
- metody předzpracovaní obrazu pracující s lokálním okolím,
- metody pro obnovu obrazu

[1] Obrázek 4.1 převzat z webu https://developer.apple.com/documentation/accelerate/blurring_an_image a přeložen

Dále se v této kapitole zmiňuji pouze o metodách zpracování obrazu, které jsem využil při implementaci výsledné aplikace, která je popsaná v kapitole 7.

4.1.1 Vyhlazování obrazu

Vyhlazování obrazu, také nazývaná jako filtrace obrazu, spadá do kategorie metod pracujících s pixely v lokálním okolí. Vyhlazování využívá redundance v obrazových datech k potlačení šumu, a to tak, že průměruje hodnotu jasu v určitém okolí pixelů $\mathcal{O}$. Existuje mnoho různých metod pro vyhlazování obrazu. Dále se v textu zabývám pouze Gaussovým rozostřením, jelikož právě to jsem ve své implementaci použil.

4.1.2 Gaussovo rozostření

Gausovo rozostření je jedna z metod, která slouží k rozmazání obrazu a odstranění nežádoucího šumu. Jak již z názvu napovídá k výpočtu rozmazaní se používá Gaussova funkce. Ta je v dvojrozměrném prostoru definována jako:

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

kde x a y jsou souřadnice určující vzdálenost od středového pixelu (0,0) a σ je směrodatná odchylka, která je úměrná k velikosti okolí, ve kterém filtr pracuje. Čím vyšší je hodnota σ tím větší bude výsledné rozostření. Hodnoty z tohoto rozdělení jsou použity k vytvoření matice konvolučního jádra. Hodnoty této matice jsou váženým průměrem hodnot okolních pixelů, kdy ve středu je hodnota původního pixelu a směrem od něj se váhá daných pixelů postupně zmenšuje. Matici je nutné vydělit součtem hodnot pixelů tak, aby se hodnota konvolučního jádra rovnala 1. Níže je uveden příklad Gaussova koevolučního jádra velikosti 5x5.

$$h = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

4.1.3 Detekce hran

Hranové detektory jsou velmi důležité metody spadající do kategorie metod pracujících s lokálním okolí pixelů. Díky metodám pro detekci hran je možné z obrazu získat informace, které mohou být důležité pro jeho další zpracování, jako je například segmentace nebo detekce objektů.

Hrany lze dělit dle jejich směru na vodorovné, svislé a diagonální. Hrany jako takové jsou v obraze zaznamenány jako náhlá změna intenzity jasu sousedících pixelů. Tyto změny intenzity jsou nejčastěji způsobené nespojitostí v orientaci povrchu nebo hloubky, změně barvy či osvětlení v zachyceném obraze.

Samotné hranové detektory lze rozdělit do dvou hlavních kategorií podle toho, zda jsou založené na aproximaci extrémů první derivace nebo hledají nulové průchody v druhé derivaci. Metody pracující s první derivací pracují gradientem. Pro výpočet jednotlivých složek tohoto gradientu se používají hranové operátory. Tyto operátory jsou reprezentovány konvolučním jádrem, kterým se poté daný obraz násobí. Mezi nejznámější operátory

patří například Robertsův, Prewittův či Sobelův. Nevýhodou těchto metod je závislost na velikosti obrazu a citlivost na šum. Z operátorů využívající druhou derivaci je nejznámější Laplacianův operátor, který je ale také náchylný na šum. Proto se často používají takzvané LOG (Laplacian of Gaussian) metody, které aplikují Gaussův filtr k odstranění šumu.

John F. Canny definoval ideální kriteria hranového detektoru, a to:

- **Spolehlivá detekce** - Hranový detektor musí nalézt všechny důležité hrany a nesmí přitom detekovat hrany falešné.
- **Přesná lokalizace** - Vzdálenost mezi reálnou a detekovanou hranou musí minimální.
- **Jednoznačná odezva** - Každá hrana musí být detekována pouze jednou. Toto kritérium je tak částečně pokryto pravidlem prvním, jelikož dvojitá detekce by znamenala detekci falešné hrany. Toto kritérium řeší problém s narušením hrany šumem.[19]

Aby se mu podařilo co nejvíce přiblížit ideálnímu detektoru, vyvinul svoji vlastní metodu, která kombinuje různé postupy zpracování obrazu.

4.1.4 Cannyho hranový detektor

Cannyho hranový detektor je jedním z nejpopulárnějších hranových detektorů současnosti. Jak již název napovídá, byl vytvořen Johnem F. Cannym v roce 1986. Aby tento detektor co nejvíce naplnil kritéria zmíněná výše a stal se tak co nejideálnějším detektorem, je rozdělen do několika kroků. Při popisu kroků vycházím z konkrétní implementace z knihovny OpenCV [1], kterou ve své práci používám.

Redukce šumu

Jak jsem zmínil výše, hranové detektory jsou citlivé na šum. Z tohoto důvodu je prvním krokem Cannyho metody odstranění nežádoucího šumu. K tomu Cannyho hranový detektor využívá Gaussův filtr, popsaný v kapitole 4.1.2, a to konkrétně s velikostí konvolučního jádra 5x5.

Nalezení gradientu

Ve druhém kroku se na vyhlazený obraz aplikuje jeden z klasických filtrů pro nalezení hran. V případě implementace z knihovny OpenCV je k filtraci použit Sobelův operátor. Tímto operátorem je obraz filtrován jak ve vodorovném, tak i svislém směru. Výsledkem je získání hodnoty první derivace ve vodorovném směru (G_x) a svislém směru (G_y). Z těchto dvou obrazů je poté vypočítána velikost a směr gradientu pro jednotlivé pixely následovně:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

kde G je velikost gradientu a θ jeho úhel, který určuje směr. Tento směr je zaokrouhlen na jeden ze čtyř případů. Vodorovný (0°), svislý (90°) a dva diagonální (45°, 135°).

Potlačení nemaximálních hodnot

Protože po předchozím kroku v obraze zůstali tlusté hrany tvořené nechtěnými pixely, je nutné tyto pixely odstranit. V tomto kroku tedy dochází ke ztenčení hran tím, že se odstraní pixely, které netvoří samotnou hranu. Algoritmus postupně projde každý pixel a jeho okolí ve směru jeho gradientu. Jestliže je v tomto okolí pixel lokálním maximem je ponechán, v opačném případě je pixel potlačen (jeho hodnota je nahrazena nulou). Výsledkem je poté binární obraz reprezentující tenké hrany.

Prahování s hysterezí

Posledním krok rozhoduje o tom, zda všechny nalezené hrany jsou hranami, které hledáme. V této části algoritmu je potřeba definovat dva prahy (více o prahování v kapitole) – minimální a maximální. Hrany, které mají velikost gradientu větší než je hodnota maximálního prahu, jsou automaticky brány jako hrany které hledáme. Naopak hrany s velikostí gradientu menší, než je minimální práh, jsou vyhodnoceny jako falešné a jsou zahozeny. Všechny ostatní hrany, které leží mezi těmito dveřmi prahy, jsou označené jako hrany v případě, že jsou napojeny na pixely hran, které leží nad maximálním prahem. Pokud hrana ležící mezi prahy není napojená na tyto pixely, je zahozena. Z tohoto důvodu je velice důležité správně zvolit hodnoty obou prahů.

4.2 Segmentace obrazu

Hlavním cílem segmentace je rozdělit obraz do částí, které spolu nějakým způsobem souvisí, a tím usnadnit další analýzu obrazu. Tyto segmenty pak můžou například reprezentovat různé objekty nebo oblasti zachycené na jedné fotografii. Metody pro segmentaci se liší dle toho, jaké obrazové informace využívají pro vytvoření segmentu. Nejednoduší segmentační metodou je prahování. Existují ale i metody, které jsou založené na detekci hran nebo na hledání oblastí pixelů, které mají určitou podobnost. Jelikož jsem při implementaci využil k segmentaci právě prahování, budu se v této podkapitole věnovat pouze této metodě.

4.2.1 Prahování

Prahování je nejjednodušší metodou pro segmentaci obrazu. Ačkoliv se jedná o jednu z nejstarších metod, je stále široce používaná, a to zejména díky tomu, že jde o metodu výpočetně nenáročnou a především rychlou (lze ji provádět v reálném čase). Vstupem je obvykle obraz ve stupních šedi, výstupem pak binární obraz reprezentující segmentaci. Pro úspěšnou segmentaci prahováním je velice důležité, aby objekty, které mají být segmentovány, byly dostatečně rozlišitelné od pozadí a vzájemně se nedotýkaly. Také je zapotřebí zvolit správnou hodnotu pro práh. Samotná metoda pak prochází všechny pixely obrazu a porovnává je s hodnotou prahu. Pokud je intenzita pixelu větší nebo rovna hodnotě prahu, je pixel ponechán, v opačném případě je vyhodnocen jako pozadí a je odstraněn. Tento postup se nazývá *globální prahování* a není příliš vhodný pro obrazy, ve který není osvětlení rovnoměrné, jelikož používá pro všechny pixely stejnou hodnotu prahu.

Přesnější metodou, která se vypořádá s nerovnoměrným osvětlením obrazu, je *adaptivní prahování*. Tato metoda prahovou hodnotu pro jednotlivé pixely vypočítává zvlášť na základě jejich blízkého okolí. Velikost tohoto okolí se může stejně jako způsob výpočtu lišit. Knihovna OpenCV podporuje pro výpočet adaptivní hodnoty prahu dva způsoby. Prvním způsobem je určení prahu ze střední hodnoty pixelů v definovaném okolí. Druhá z nich poté

počítá práh jako vážený součet okolních pixelů. Váha jednotlivých pixelů je určena pomocí Gaussova okna. [9]

Existují ještě i další metody pro prahování, ze kterých mohu jmenovat například již dříve zmíněné prahování s hysterezí, které využívá maximální a minimální hodnotu prahu a je blíže popsané v kapitole 4.1.4 o Cannyho hranovém detektoru.

4.3 Extrakce příznaků z obrazu

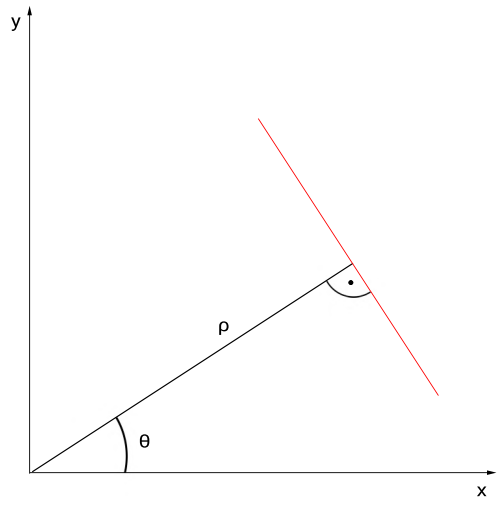
Extrakce příznaků je v počítačovém vidění velice důležitou technikou. Označuje skupinu metod, která má za úkol najít a izolovat z obrazu určité příznaky. Těmito příznaky mohou být čáry, rohy objektů, různé geometrické tvary, ale například i určité barvy. Já v této kapitole popíši metodu Hougovy transformace, kterou v implementaci využívám pro detekci čar a kruhů.

4.3.1 Houghova transformace

Houghova transformace je metoda původně navržená pro detekci přímek, později rozšířená tak, že dokáže detekovat jednoduché tvary, které lze popsat parametricky. Hlavní výhodou Houghovy transformace je to, že metoda dokáže najít i tvary, které jsou nějakým způsobem zkreslené či narušené. Vstupem pro tuto metodu je již předem zpracovaný obraz například detektorem hran, díky čemuž je snazší hledané tvary detekovat.

Houghova transformace přímky

Přímku můžeme reprezentovat jako $y = mx + c$ nebo také parametricky, $\rho = x\cos\theta + y\sin\theta$, kde ρ délka normály od přímky k počátku souřadnic a θ je úhel mezi touto normálou a osou x, což lze vidět na obrázku 4.2.



Obrázek 4.2: Definice přímky

Přímku tedy lze reprezentovat těmito dvěma hodnotami (ρ, θ). Algoritmus vytvoří 2D pole a nastaví všude výchozí hodnotu 0. Nechť ρ reprezentuje řádky a θ sloupce. Velikost

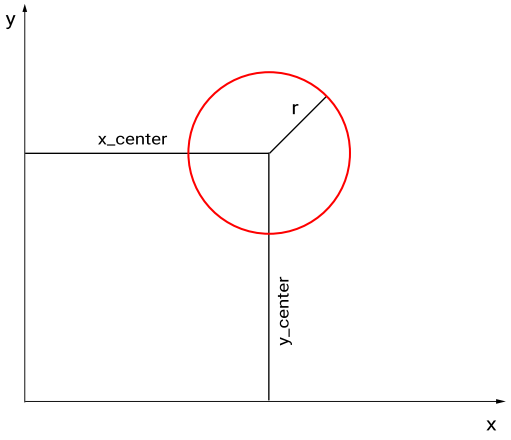
samotného pole závisí na požadované přesnosti. To znamená, že pokud budeme chtít přesnost úhlu na 1°, bude mít pole o 180 sloupcích. Pro řádky pak platí, že při přesnosti na jeden pixel je maximální velikost pole rovna uhlopříčce obrazu.

Samotný algoritmus pak pracuje tak, že vezmeme první bod přímky (x,y) a dosadíme do rovnice každé θ dle nastavené přesnosti a kontrolujeme získané ρ. Pro každý pár (ρ,θ) zvyšujeme v poli hodnotu o 1. Poté pokračujeme na další bod přímky a stejně jako u bodu prvního navyšujeme hodnoty získaných buněk o 1. Na konci tohoto procesu nalezneme nejvyšší hodnotu. Tuto nejvyšší hodnotu bude obsahovat například buňka (50,90). To znamená, že ve vzdálenosti 50 od počátku souřadnicového systému se nachází přímka, která svíra s osou x úhel 90°.

OpenCV obsahuje i optimalizovanou pravděpodobnostní metodu Houghovy transformace. Ta na rozdíl od klasické metody neprochází každý jeden bod, ale pouze náhodnou podmnožinu bodů, která je dostatečná pro detekci přímky. [7]

Houghova transformace kružnice

Pro detekci kružnice postupujeme podobně jako při detekci přímek. Matematickým zápisem pro kružnici je $(x - x_{center})^2 + (y - y_{center})^2 = r^2$ kde (x_{center}, y_{center}) je střed kruhu a r je poloměr viz. obrázek 4.3. Jak lze odvodit z rovnice, v tomto případě bychom potřebovali použít 3D akumulátor, jelikož operujeme s hodnotami (x,y,r). Tento přístup je velice neefektivní, protože pro každý bod je nutné počítat s kruhem o různém poloměru r. OpenCV proto pro výpočet používá metodu, která využívá informace o gradientu. Ta místo aby kreslila kružnici pro každý bod a inkrementovala buňky, které kruhu odpovídají, jako tomu bylo u přímky, inkrementuje pouze body, které leží ve směru gradientu daného bodu.[6]



Obrázek 4.3: Definice Kružnice

Kapitola 5

Operační systém Android

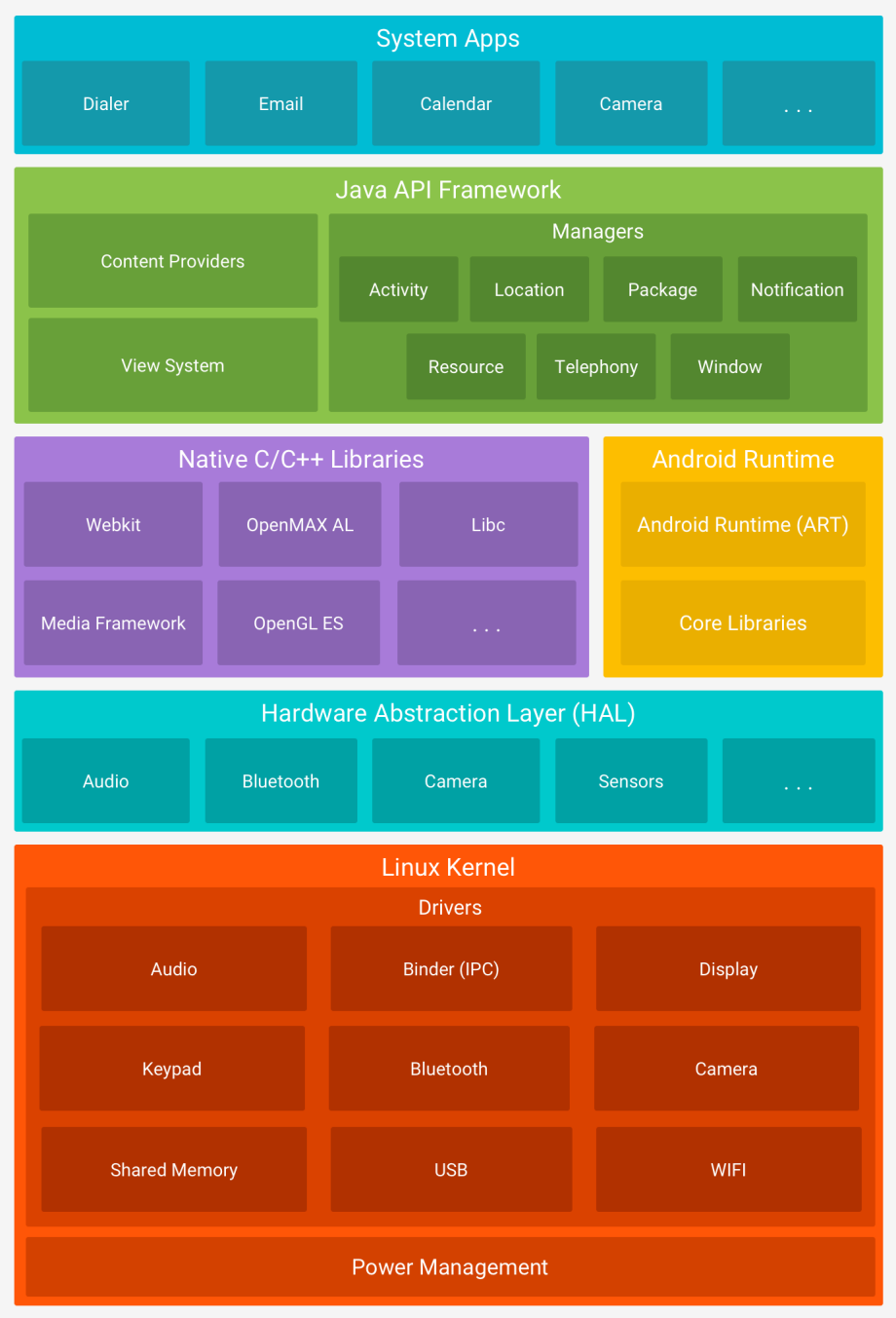
Android je open source operační systém postavený na linuxovém jádře vyvíjen pod záštitou Open Handset Alliance včele se společností Google, která původní firmu Android Inc. odkoupila v říjnu 2003. Android je nejrozšířenější mobilní platformou současnosti, a ačkoliv byl zprvu systém vyvíjen primárně jako systém pro fotoaparáty, podařilo se mu dobít trh chytrých telefonů a následně proniknout do dalších odvětví elektronky. Dnes již můžeme tento systém najít v různých podobách v televizorech, chytrých hodinkách, herních konzolích, či dokonce jako součást palubních počítačů moderních automobilů.

Otevřenost platformy Android má i své nevýhody od škodlivých kódu až po nekonzistentnost verzí. Jelikož o verzi systému rozhoduje výrobce zařízení a ne samotný Google, setkáme se v oběhu s mnoha verzemi, což může vývojářům komplikovat funkčnost jejich aplikací. Google se tento problém snaží řeší minimální verzí, pro kterou musí být nové aplikace funkční. Podpora starších verzí je pak na samotném vývojáři aplikace.

Nerozšířenější verzí na trhu je Android 10.0 (40.52 %), následuje Android 9 Pie (19.14 %) a Android 8.1 Oreo (9.34 %). Nově vydaný Android 11.0 je na čtvrté pozici s 8.9 %. Zajímavostí pak může být, že Android 4.4 vydaný před více jak 7 lety má stále 1 % zastoupení. [21]

5.1 Architektura systému

Operační systém Android je softwarová sada komponent uzpůsobená pro potřeby různých zařízení. Jeho architektura se skládá z šesti vrstev, které jsou k vidění na obrázku.



Obrázek 5.1: Jednotlivé vrstvy architektury systému Android [1]

Linuxové jádro

Základem celé architektury je linuxové jádro, které obstarává klíčové funkcionality, jako je například správa procesoru, obsluha nízkoúrovňové paměti, a umožňuje vyšším vrstvám využít klíčové bezpečnostní funkce.

[1] Obrázek 5.1 převzat z webu https://developer.android.com/guide/platform

Abstrakční hardwarová vrstva

Druhou vrstvou je Hardware Abstraction Layer (HAL) neboli Abstrakční hardwarová vrstva. Ta usnadňuje využití hardwarových prvků zařízení, jako je například kamera tím, že poskytuje jejich funkcionalitu frameworku vyšší úrovně Java API, který je popsaný níže. Tato vrstva je složená z několika knihovních modulů pro jednotlivá zařízení. Java API v případě použití daného zařízení vyšle požadavek na načtení daného modulu.

Nativní C/C++ knihovny a Android Runtime

Na třetí vrstvě nativní knihovny C/C++ a Android Runtime (ART). Od verze Androidu 5.0 je každá aplikace běžící v systému Android spuštěna jako vlastní proces s instancí ART. ART je naprogramován tak, aby umožnil běh vícero virtuálních strojů na zařízeních s nízkou pamětí tím, že spouští soubory DEX. Soubor DEX je formát bytecode navržený specificky pro Android a optimalizovaný tak, aby měl co nejmenší paměťové nároky. Mezi hlavní přednosti tohoto řešení patří optimalizovaný garbage colletor, lepší podpora ladění s detailními výjimkami a podrobným hlášení pádů pro snadnou diagnostiku.

Další částí třetí vrstvy jsou nativní knihovny C/C++. Ty se zde vyskytují proto, že mnoho základních komponent a služeb systému Android je vytvořeno z nativního kódu, který vyžaduje nativní knihovny naspané v C/C++. Platforma Android funkcionalitu některých těchto knihoven poskytuje skrze Java API framework dále do aplikací. Do této vrstvy patří například i knihovna OpenCV, kterou v rámci této bakalářské práce využívám a která je blíže popsána v kapitole.

Java API Framework

Předposlední vrstvou architektury Android je již dříve zmíněný Java API framework. Díky této vrstvě jsou všechny funkce operačního systému zpřístupněny prostřednictvím rozhraní API napsaných v jazyce Java. Tato rozhraní tvoří potřebné stavební bloky pro tvorbu aplikací, a to tím, že je zjednodušeno opětovné použití základních funkcí, modulárních komponent a služeb. Vývojáři aplikací mají plný přístup ke stejným API, které jsou využívány v systémový aplikacích. To umožňuje využít kupříkladu správce notifikací, který aplikaci dovoluje zobrazovat vlastní upozornění na stavovém řádku, nebo mohou sáhnout po prostředcích jako jsou lokalizační řetězce pro snadný překlad aplikace do více jazyků či předpřipravené grafické prvky.

Systémové aplikace

Poslední vrstvou jsou samotné systémové aplikace, které jsou dodávány jako součást operačního systému. Tato základní sada obvykle obsahuje aplikace, jako je e-mailový klient, SMS messenger, kalendář a další. Tyto aplikace nejsou ničím zvláštní a uživatel je může nahradit libovolnou aplikací třetí strany. [16]

5.2 Vývoj aplikací pro Android

Jelikož se chytré mobilní telefony staly každodenní součástí běžného života, popularita mobilních aplikací v poslední době vzrostla a můžeme tak najít aplikaci téměř na cokoliv. To také zapříčinilo, že vývoj aplikací přilákal mnoho vývojářů z různých odvětví specializace. Ačkoliv oficiálními jazyky pro vývoj aplikací na platformu Android jsou Java a Kotlin, je zde

i mnoho jiných možností. Jelikož webové aplikace jsou čím dál více spjaté s těmi mobilními a Javascript je mezi vývojáři velice oblíbený jazyk, vzniklo také mnoho frameworků, které umožnují psaní mobilních aplikací v Javascriptu. Tyto frameworky se pak starají o překlad JavaScript kódu na nativní aplikaci, popřípadě vytváří webovou stránku spustitelnou v aplikaci. To je docíleno tím, že je aplikace zobrazí pomocí komponenty Web-View a navozuje tak pocit standardní mobilní aplikace. Tomuto druhu mobilních aplikací se říká hybridní mobilní aplikace. Mezi nejznámější Frameworky překládající JavaScript na nativní aplikaci patří React Native[2] vyvíjený firmou Facebook. Druhou možnost, tedy vytvoření hybridní mobilní aplikace, nabízí například framework Cordova[3] od společnosti Apache. Já jsem si pro svoji bakalářskou práci zvolil jazyk Java, pro který je přichystáno oficiální vývojové prostředí Android Studio popsané v kapitole níže.

5.2.1 Android Studio

Původně bylo pro vývoj aplikací na platformu Android využíváno integrované vývojové prostředí (IDE) Eclipse. Toto vývojové prostředí bylo velice populární mezi Java vývojáři a s rozšířením Android Developement Tools (ADT) umožnovalo vyvíjet i androidí aplikace. Společnost Google nahradila v roce 2013 Eclipse za nástroj Android Studio. Toto vývojové prostředí je postavené na populárním IntelliJ IDEA od české společnosti JetBrains s.r.o. Oproti originálnímu IntelliJ IDEA je Android Studio rozšířeno o sadu nástrojů nazývanou Android Software Development Kit (Android SDK), dále také obsahuje emulátory mobilních zařízení pro testování aplikací bez nutnosti využití reálných mobilních zařízení a další nástroje a knihovny například pro tvorbu uživatelského rozhraní. V Android SDK jsou obsaženy nástroje pro ladění a testování, dokumentace, ukázkové části kódu pro demonstraci fungování API či podporu pro integraci Google Play plateb v aplikaci a další. Android studio využívá také automatizační nástroj Gradle. Ten umožňuje automatizaci sestavovaní, testování, nasazování a mnoho dalšího. Konkrétní nastavení Gradle se provádí v souboru build.gradle, který využívá syntaxi jazyka Groovy. [5]

Výsledným produktem sestavení je soubor APK (Android application package), který je možné s povolením spustit na chytrém zařízení, popřípadě soubor podepsat elektronickým klíčem a publikovat na některém z marketů pro aplikaci. Oficiálním marketem pro aplikace na telefonech Android je Obchod Google Play[4].

[2]Framework React Native dostupný na https://reactnative.dev.
[3]Framework Cordova dostupný na https://cordova.apache.org
[4]Obchod Google Play - https://play.google.com

Kapitola 6

Návrh řešení

Důležitou částí při vývoji software je zamyšlení se, jaké problémy v aplikaci je potřeba vyřešit a navrhnout postup, jakým se daný problém bude řešit. Kvalitní návrh aplikace dokáže značně usnadnit samotnou implementaci a předejít některým problémům. Tato kapitola je tedy zaměřena na návrh jednotlivých částí aplikace. Je rozdělená do tří podkapitol dle řešených problémů. Prvním problémem je samotná identifikace a extrakce hry z obrazu. Návrhu této části se věnuje podkapitola 6.1. Za ní následuje podkapitola 6.2 věnovaná návrhu ohodnocovacích funkcí, které jsou důležité pro určení co možná nejlepšího tahu u jednotlivých her. Poslední podkapitolou 6.3 je pak návrh uživatelského rozhraní aplikace.

6.1 Identifikace a extrakce hry ze snímku

Jedním z hlavních funkcionalit výsledné aplikace je identifikování herní desky a analyzování stavu hry ze snímku od uživatele. Rozhodl jsem se využít pouze funkcí openCV bez zapojení neuronové sítě.

Jako první je nutné získat ze snímku samotnou desku, poté určit, o jakou hru se jedná, a následně nalézt herní kameny a přiřadit je jednomu z hráčů. Pro úspěšnou detekci vyžaduji po uživateli snímek pořízený co nejvíce kolmo nad deskou za dobrých světelných podmínek.

Extrakce herní desky

Herní desky se vyrábí převážně ve čtvercovém provedení. Za tohoto předpokladu aplikace najde pomocí funkce pro hledání obrysů největší čtvercový obrys na pořízeném snímku. Tím získám souřadnice oblasti, ve které se nachází herní deska. Jelikož je možné, že na snímku bude v okolí desky mnoho rušivých elementů a algoritmus desku nenalezne, je důležité, aby měl uživatel možnost tuto oblast vybrat sám. Ruční výběr uživateli zobrazí na displeji čtverec, který znázorňuje prostor, ve kterém bude analýza hry probíhat. Uživatel má možnost měnit velikost tohoto čtverce podle potřeby. Tuto oblast získanou automaticky či ručně vyříznu z hledaného obrazu, pro usnadnění další práce změním jeho velikost na 600x600 pixelů a další analýza je prováděna již pouze v tomto výřezu.

Identifikace typu deskové hry

Získáním výřezu oblasti, ve kterém se deska nachází, můžeme přistoupit k samotné identifikaci hry. Zde bylo nutné navrhnout univerzální způsob, kterým od sebe odlišit jednotlivé hry. Po analýze vzhledu různých druhů desek jsem došel k závěru, že nejuniverzálnějším

přístupem bude identifikovat hru na základě nalezených průsečíků. Jako první v hledané oblasti pomocí Houghovy transformace naleznu vodorovné a svislé přímky, u kterých poté najdu průsečíky. Protože jedna čára může být identifikovaná jako více čar v lehce odlišných směrech, dochází k nalezení mnoha průsečíků blízko sebe. Tento problém jsem vyřešil filtrováním těchto bodů, které jsou příliš blízko sebe. Každý průsečík porovnávám s již zaznamenanými průsečíky, a pokud v jeho okolí nějaký takový je, tento zahodím. Velikost okolí pro zahazování jsem určil ze znalosti desky. Vím například, že deska pro hru Dáma je velikosti 8x8. Oblast pro zahození průsečíku je tedy $\frac{1}{8}$ z šířky desky, která je ještě mírně zmenšená, aby nedocházelo k zahazování bodů z vedlejších čar. Na základě počtu zbylých průsečíku identifikuji hru. Gomoku 13x13 má průsečíků 169, dáma 81 a hra mlýny 49. Pokud počet průsečíků neodpovídá ani jedné hře, vyhodnotím rozpoznání jako neúspěšné. V opačném případě pokračuji k nalezení herních kamenů.

Nalezení kamenů

Kameny jsou ve snímku po aplikování hranového detektoru reprezentovány kružnicí. Pro nalezení těchto kružnic jsem využil, stejně jako u hledání příjmech, Houghovy transformace. Parametry pro hledání kružnic jsou nastaveny různě dle typu hry, protože například ve hře Gomoku se kameny dotýkají, u hry Mlýny naopak mezi kameny vždy nějaká mezera bude.

Po nalezení samotných kamenů je nutné určit jejich pozici vzhledem k desce. Tu určím tak, že si seřadím nalezené průsečíky podle hodnot osy x a y. U Dámy ze seřazených bodů vytvořím pomyslné čtverce reprezentující políčka. Každou kružnici poté porovnám s těmito čtyřmi body, a pokud je střed kružnice uvnitř tohoto pomyslného čtverce, vyhodnotím, že na tomto políčku se nachází kámen. U her Gomoku a Dáma naopak porovnávám, zda průsečík reprezentující políčko je uvnitř kružnice. Pokud ano, znamená to, že na dané pozici leží kámen.

Pro všechny takto nalezené kameny je nutné najít jejich průměrnou barevnou hodnotu. Poté je barva každého kamene provovnána s tímto průměrem. Pokud je hodnota menší jak průměr je kámen přiřazen prvnímu hráči, v opačném případě hráči druhému.

6.2 Ohodnocovací funkce

Algoritmy pro hraní her využívají ohodnocovací funkci, na niž závisí jejich schopnost volit strategie co možná nejoptimálnějšího tahu. Každý stav hry se dá ohodnotit mnoha různými způsoby. Všechny ale mají jedno společné. Snaží se určit, co je v konkrétní hře důležitý faktor ovlivňující výhru, a jakou váhu tento faktor vůči ostatním má.

6.2.1 Ohodnocení hry Dáma

Za jedním z velkých úspěchů strojového hraní ve hře Dáma stojí program Chinook vyvinutý Jonathanem Schaefferlem [18]. Chinook byl prvním programem v historii, který vyhrál šampionát proti lidem. Chinook je unikátní svým přístupem, oproti dalším velice úspěšným algoritmům pro hraní hry Dáma nevyužívá strojové učení. Jeho úspěch tkví ve velice detailně vytvořené s ohodnocovací funkcí s mnoha faktory a databází všech možných konečných stavů. Jelikož komplexnost řešení je enormní a využívá databázové řešení, které není pro offline aplikaci ideální, rozhodl jsem se na základě mých poznatků ze hry a poznatků zkoumaných v článku *Evolutionary-based heuristic generators for checkers and give-away checkers* [13] vytvořit následující pravidla pro ohodnocení:

1. Počet obyčejných kamenů.
2. Počet kamenů typu dáma.
3. Počet kamenů bránící soupeřovi získat dámu.
4. Počet kamenů v prostředních dvou řádcích herní desky.
5. Počet kamenů, které jsou v bezpečí. To znamená kameny, kterým neohrozí sebrání od soupeře.
6. Počet kamenů, které jsou v nebezpečí. To znamená kameny, které může soupeř v následujícím tahu sebrat.

Po provedení několika testů s různými hodnotami jsem dospěl k finálnímu ohodnocení koeficientů, které vykazovalo nejlepší výsledky. Je znázorněno v tabulce 6.1. Ohodnocovací funkce vypočítá pro každý tah skóre dle rovnice (6.1).

$$Score = C_1 * SUM(1) + C_2 * SUM(2) + C_3 * SUM(3) + C_4 * SUM(4) + C_5 * SUM(5) + C_6 * SUM(6) \tag{6.1}$$

kde $C_1, .., C_6$ jsou bodové hodnoty koeficientů jednotlivých pravidel a $SUM(n)$ je počet kamenů splňující pravidlo n.

Pravidlo	1	2	3	4	5	6
Hodnota koeficientu	1250	2000	1000	320	750	-750

Tabulka 6.1: Koeficienty ohodnocovací funkce pro hru Dáma

6.2.2 Ohodnocení hry Mlýny

Ralph Gasler v roce 1996 publikoval článek [3] ve kterém dokázal, že ani znalostní databáze všech závěrečných scénářů nedokáže zajistit výhru, ale pouze remízu. Nelze tedy vytvořit ohodnocovací funkci, která zajistí vítězství pokaždé.

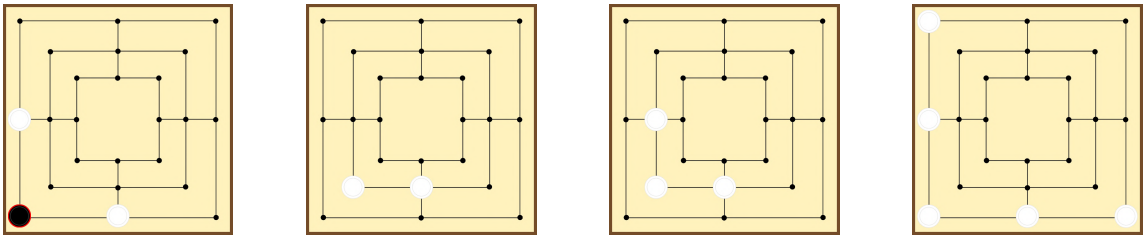
Hra Mlýny, jak bylo popsáno v kapitole 2.2, je rozdělena do dvou základních fází. V druhé fázi je hráči, který má pouze 3 kameny, dovoleno skákání na libovolné pole. Toto pravidlo zásadně ovlivňuje hru a je nutné ho brát v potaz. Pro získání nejlepších výsledků je tedy nutné počítat při ohodnocení i s touto skutečností. Ohodnocovací funkce je rozdělena na tři části, dle aktuálního stavu hry, a je založena na článku [15]. K ohodnocení stavu funkce počítá s těmito případy:

1. Uzavření mlýnu v posledním tahu.
2. Počet všech mlýnů.
3. Počet blokovaných kamenů (6.1a).
4. Celkový počet kamenů.
5. Dva kameny v řadě (6.1b).

6. Tři kameny ve formaci L (6.1c).

7. Dvojitý mlýn (6.1d).

8. Vítězství (soupeř má méně než 3 kameny nebo má všechny kameny zablokované).



(a) Blokovaný kámen soupeře

(b) Dva kameny v řadě

(c) Tři kameny ve formaci L

(d) Dvojitý mlýn

Obrázek 6.1: Ukázka některých pravidel ohodnocovací funkce hry Mlýny

Ve fázi vkládání kamenů je v rovnici 6.2 počítáno s případy 1-6. Po dokončení rozmísťování kamenů jsou brána v potaz všechny pravidla kromě čísla 6, což lze vidět v rovnici 6.3. Jakmile jednomu z hráčů zbývají pouze 3 kameny, jsou v rovnici 6.4 uplatněna pravidla 1,5,6 a 8. Váhu koeficientů v různých fázích hry lze vidět v tabulce 6.2.

$$ScorePhase1 = C_1 * SUM(1) + C_2 * SUM(2) + C_3 * SUM(3) + C_4 * SUM(4) + C_5 * SUM(5) + C_6 * SUM(6) \quad (6.2)$$

$$ScorePhase2 = C_1 * SUM(1) + C_2 * SUM(2) + C_3 * SUM(3) + C_4 * SUM(4) + C_5 * SUM(5) + C_7 * SUM(7) + C_8 * SUM(8) \quad (6.3)$$

$$ScorePhase2jumps = C_1 * SUM(1) + C_5 * SUM(5) + C_6 * SUM(6) + C_8 * SUM(8) \quad (6.4)$$

kde $C_1, .., C_8$ jsou bodové hodnoty koeficientů jednotlivých pravidel a $SUM(n)$ je počet kamenů splňující pravidlo n.

Číslo pravidla:	1	2	3	4	5	6	7	8
Fáze rozmísťování	18	26	1	6	12	7	-	-
Fáze přesouvání	14	43	10	8	7	-	42	1086
Fáze přesouvání se skákání	10	-	-	-	1	16	-	1190

Tabulka 6.2: Koeficienty ohodnocovací funkce pro hru Mlýny

6.2.3 Ohodnocení hry Gomoku

Doposud nejúspěšnějším algoritmem pro hraní hry Gomoku je *Gomoku Yi Xin*, který v roce 2017 dokázal porazit nejlepšího lidského protihráče a vyhrát Gomoku šampionát čtyřikrát

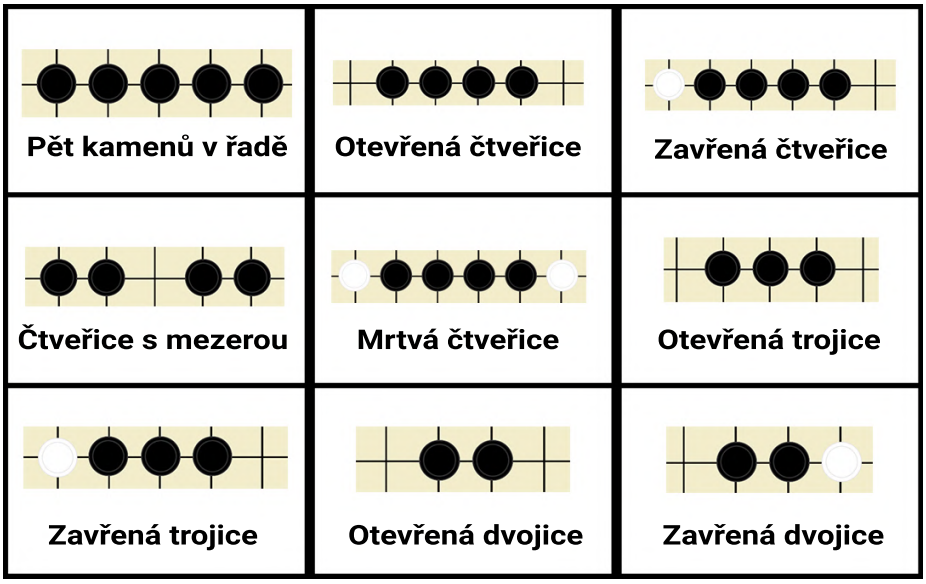
po sobě. Přestože jsou výsledky tohoto algoritmu velice úspěšné, jeho nevýhodou je zdlouhavé rozhodování na začátku hry, kdy je deska hrací plochy prázdná a minimax tak zbytečně prochází každou z možností. To převážně v prvních čtyřech korcích znamená projití mnoha zbytečných variant. Tento problém řeší některé algoritmy pro hraní hry Gomoku databází všech různých variací rozhlížení prvních 4 kamenů.[12] Jelikož moje aplikace má pomoci převážně určit nejlepší tah a ne odehrát celou hru za hráče, rozhod jsem se o snadnější a méně výpočetně náročnou variantu řešení. Ta, pokud se jedná o první tah, vybere náhodně jedno ze středových políček ve čtverci 3x3. Pro ostatní tahy poté funkce analyzuje vzory. Každý z těchto vzorů má přidělenou jinou bodovou hodnotu. Při vymýšlení hledaných vzorů jsem se také kromě vlastních zkušeností ze hry inspiroval některými vzory uvedenými v článku [12]. Hledané vzory jsou následující:

1. **Zavřená dvojice** – Dvojice vedle sebe ležících kamenů, která je z jedné strany zavřená kamenem soupeře.
2. **Otevřená dvojice** – Dvojice sousedících kamenů, která má z obou stran prostor pro expanzi. Dvojice mezi sebou může mít i jedno volné pole.
3. **Zavřená trojice** – Trojice vedle sebe ležících kamenů, která je na jedné straně uzavřena kamenem soupeře.
4. **Otevřená trojice** – Trojice kamenů, která je z obou stran otevřená.
5. **Mrtvá čtveřice** – Čtyři kameny zavřené z obou stran soupeřem.
6. **Čtveřice s mezerou** - Kombinace trojice a jednoho kamenu s mezerou mezi, nebo dvě dvojice s mezerou uprostřed.
7. **Zavřená čtveřice** – Čtyři kameny uzavřené z jedné strany soupeřovým kamenem.
8. **Otevřená čtveřice** – Formace, která garantuje vítězství v příštím tahu.
9. **Pět kamenů v řadě** – Vítězství

Jednotivé vzory jsou k vidění na obrázku 6.2 a hodnoty koeficientů v tabulce 6.3.

Číslo pravidla	1	2	3	4	5	6	7	8	9
Fáze rozmísťování	50	800	1000	3000	100	8000	30000	100000	10000000

Tabulka 6.3: Koeficienty ohodnocovací funkce pro hru Gomoku



Obrázek 6.2: Ukázka některých vzorů, které využívá ohodnocvací funkce hry Gomoku

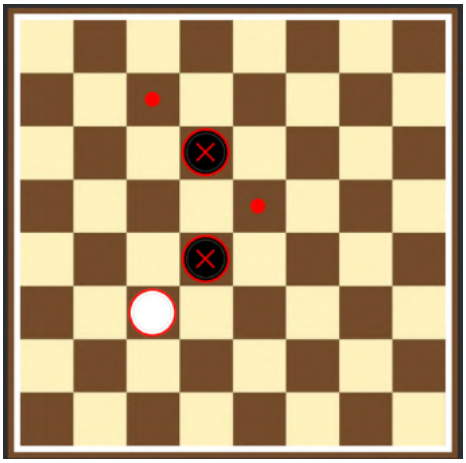
6.3 Uživatelské rozhrani

Aby se aplikace uživatelům dobře používala, je nutné správně navrhnout uživatelské grafické rozhraní (GUI). GUI by mělo být přehledné, intuitivní, jednoduché na ovládání a hezké, aby přitáhlo pozornost uživatele. Jako první jsem si musel definovat, jaké jednotlivé obrazovky bude aplikace obsahovat. První obrazovkou po spuštění je samotné menu aplikace. To obsahuje logo s názvem aplikace a položky pro přechod do dalších obrazovek. Jelikož hlavní částí aplikace je skenování herní desky, rozhodl jsem se tlačítko pro přechod do skenovací obrazovky umístit jako první. Protože je ale možné, že některý z uživatelů nebude chtít desku skenovat a raději zadá rozložení ručně, je zde i tlačítko pro přechod do ručního režimu. To vede na druhé menu, kde si uživatel vybere jednu, kterou ze tří her chce zadávat. V dolní části první obrazovky je pak tlačítko pro zobrazení nápovědy a nastavení, které umožňuje vypnout obrazovku zobrazující naskenovanou desku.

Následujícím krokem bylo navrhnout samotnou obrazovku pro skenování. Tato obrazovka je situovaná na šířku pro lepší manipulaci s telefonem během snímání. Vzhledem k navrženým funkcím obsahuje tři tlačítka. V pravé části se nachází tlačítko pro provedení skenu. Na opačné straně je umístěno ovládání pro přepínání režimu automatické/manuální detekce, a nad ním se nachází vypínač svítilny, díky které je možné zpřesnit detekci za horších světelných podmínek. V režimu manuální detekce je v dolní částí zobrazen posuvník, díky němuž lze manipulovat s velikostí plochy pro detekci.

Další důležitou částí aplikace je i samotná obrazovka pro zobrazování nejlepšího možného tahu. Jelikož stejná obrazovka slouží i k editaci aktuálního stavu desky, bylo nutné logicky rozdělit tyto ovládací prvky. Rozhodl jsem se tedy do horní části nad zobrazenou herní deskou umístit ovládací prvky pro vkládání kamenů. V dolní části pod deskou se pak nachází ovládací prvky pro učení dalšího tahu. Aby byl výsledný tah pro uživatel přehledně znázorněn, využil jsem výraznou červenou barvu pro označení tahu. Kámen, který se v daném tahu přesouvá nebo vkládá, je znázorněn červeným obrysem, a místo, kam se kámen přesouvá, znázorňuje červená tečka. Pokud v tahu dojde k odebrání soupeřova kamene, je

tento kámen znázorněn přeškrtnutím. Zobrazení tahu v aplikaci je znázorněno na obrázký 6.3



Obrázek 6.3: Ukázka zobrazení dvojitého přeskoku ve hře Dáma

Výsledné aplikaci je také nutné určit celkovou podobu, tedy definovat barevné schéma a vzhled jednotlivých prvků. Jelikož Google doporučuje pro Android aplikace využívat postupů definovaných v příručce Material designu[1], rozhodl jsem se těchto postupů držet. Zvolil jsem modro-černé barevné schéma vybrané z online palety barev[2] určené pro Material design. V aplikaci také využívám pro některé tlačítka Material ikony[3].

[1]Příručka Material Designu dostupná z https://material.io/design
[2]Paleta barev pro Material design dostupná z https://www.materialpalette.com/
[3]Material ikony dostupné z https://fonts.google.com/icons?selected=Material+Icons

Kapitola 7

Implementace aplikace

Tato kapitola se věnuje samotné implementaci aplikace. Aplikace je implementovaná výhradně pro platformu Android, která je popsaná v kapitole 5. Ve stejné kapitole je také popsané oficiální vývojové prostředí Android Studio, které bylo k vývoji využito. Z dvou jazyků, které toto vývojové prostředí podporuje, jsem zvolil Javu. Jako první jsou v této kapitole zmíněny použité knihovny, po nich následní jednotlivé třídy a metody v nich. Na závěr kapitoly je pak popsána implementace grafického rozhraní.

7.1 Použité knihovny

Kromě standardních knihoven programovacího jazyku Java a knihoven pro operační systém Android jsem k implementaci aplikace využil také knihovnu pro počítačové vidění OpenCV.

7.1.1 Knihovna OpenCV

Knihovna OpenCV je open-source knihovna pro počítačového vidění a strojové učení vydaná pod licencí BSD. Byla vytvořena za účelem poskytnout základní infrastrukturu pro počítačové vidění a akcelerovat tak jeho využití v komerční sféře. Knihovna jako taková obsahuje stovky optimalizovaných algoritmů, které pokrývají široké spektrum počítačového vidění a strojového učení. Knihovna je vyvíjena v jazyce C++, ale poskytuje rozhraní i pro jazyky Python, Java či MATLAB. Knihovna OpenCV je složena z několika modulů. Já jsem při implementaci použil tyto moduly:

- **Core** – Modul obsahující základní datové struktury, jako například více dimenzionální pole *Mat* a základní funkce používané ostatními moduly.
- **Imgproc** - Modul obsahující funkce pro zpracování obrazu. Mezi ně patří například lineární a nelineární filtry, geometrická transformace, převody barevného prostoru, histogramy a další.

Dále knihovna obsahuje moduly pro zpracování videa, 3D rekonstrukci, detekci objektů a další. [10]

7.2 Detekce a identifikace deskové hry a jejího stavu

První částí samotné aplikace je získání informací o hře ze snímku. K řešení tohoto problému bylo zapotřebí využít metody počítačového vidění. K tomu jsem využil metody implemen-

tované ve výše zmíněné knihovně OpenCV. Samotnou implementaci této části jsem rozdělil na tři problémy, které bylo potřeba vyřešit a to:

- Nalezení herní desky.
- Identifikace typu hry.
- Získání aktuálního stavu.

O celý proces se stará třída rozšiřující aktivitu[1] skenování, nesoucí název `GameScan`. Tato třída při vytvoření spustí zadní fotoaparát, ze kterého bude zachycen snímek k dalšímu zpracování. Poté volá jednotlivé funkce ze tříd `GameBoard` a `GameType`.

7.2.1 Detekce herní desky

Detekce herní desky je naimplementována ve třídě `GameBoard`. Funkci `detectGameBoard` je předán aktuální snímek kamery, se kterým ve funkci dále pracuji. Nejprve daný snímek převedu na obraz ve stupních šedi pomocí funkce `cvtColor`, která slouží k převádění snímků do různých barevných modelů. Poté, jelikož obraz z fotoaparátu může obsahovat nežádoucí šum, bylo potřeba tento šum redukovat. Toho jsem docílil použitím funkce `GausianBlur`, která na snímek aplikuje Gaussovo rozostření. Nejlepších výsledků jsem dosáhl při zvolení konvolučního jádra velikosti 7x7 se směrodatnou odchylkou 3. Zavoláním funkce `adaptiveThreshold` poté aplikuji na snímek segmentační metodu adaptivního prahování. Jako způsob prahování jsem pomocí parametru `ADAPTIVE_THRESH_GAUSSIAN_C` zvolil vážený součet pixelů za pomocí Gaussova okna a to na okolí 11x11. Jako výstup jsem zvolil invertovaný binární obraz. Na ten poté použiji funkce `dilate` a `erode`, díky kterým spojím případné nespojitosti v segmentovaných obrysech.

V takto předpřipraveném snímku se pokusím nalézt obrysy herní desky. Volám tedy další funkci knihovny OpenCV `findContures`, která slouží právě k nalezení obrysů. Parametrem `RETR_EXTERNAL` určím, že chci hledat pouze vnější obrysy. Všechny nalezené obrysy poté v cyklu procházím aproximuji je funkcí `aproxPolyDP`, abych z obrysů získal body. Jestliže počet takto získaných bodů je roven 4, a zároveň je daný obrys větší jak předchozí nalezený, uložím jej jako nalezenou desku.

Jelikož tento přístup nemusí fungovat ve snímku, kde je v pozadí mnoho rušivých elementů, implementoval jsem také ruční metodu detekce desky. V té uživatel zachytí desku do předem vyznačeného čtverce na jeho obrazovce.

Detekovanou oblast ze snímku vyříznu a upravím její rozměry na 600x600, což mi usnadní další zpracování a předám třídě `GameType`.

7.2.2 Identifikace konkrétní hry

Výřez herní desky ve funkci `identifyGame` převedu do odstínů šedi. Poté funkcí `Canny`, která reprezentuje OpenCV implementaci Cannyho detektoru hran. Na výsledný obraz nalezených hran ještě aplikuji funkci `dilate`, abych spojil případné mezery mezi detekovanými hranami. Dalším krokem je určit, které z hran jsou čarami. K tomu používám funkci `HoughLinesP`, která hledá přímky za pomocí pravděpodobnostní Houghovi transformace. Pro všechny čáry poté naleznu průsečíky. Nalezení průsečíků jsem naimplementoval ve funkci `FindIntersection`. Jelikož pravděpodobnostní Hougova transformace může jednu čáru nalézt jako více malých u sebe je nutné ještě průsečíky vyfiltrovat, aby některé z nich

[1] Aktivita = Jedna obrazovka uživatelského rozhraní

nebyly započítány vícekrát. Ponechávám pouze takové průsečíky, které jsou vzdálené alespoň 15 pixelů od okraje snímku a zároveň kolem sebe nemají žádné jiné průsečíky, které jsou již uložené v poli. Oblast, ve které se může nacházet pouze jeden průsečík, je různá u každé z her. Proto si uchovávám tři různé pole průsečíku. Kontrola zda se v oblasti nachází jiný průsečík je ve funkci `isNearOtherPoint`. Ta jako parametry vyžaduje souřadnice aktálně kontrolovaného průsečíku, pole již započítaných průsečíků a číslo, které reprezentuje kolik herních políček hra na šířku obsahuje. Herní deska o velikosti 600 px je poté rodělena dle zvoleného čísla. To je ještě zmenšeno o konstantu 0.8, aby nedocházelo k odstranění průsečíků ze sousedních polí. Pomocí Pythagorovy věty je poté vypočtena vzdálenost mezi jednotlivými již uloženými průsečíky. Pokud je vzdálenost větší než stanovené minimum, průsečík uložím. V opačném případě dojde k jeho zahození.

Poté výsledný počet průsečíků porovnávám s hodnotami, které odpovídají daným hrám. Jestliže bylo nalezeno 169 průsečíků jedná se o hru Gomoku, při počtu 81 pak o Dámu. Číslo 49 odpovídá hře Mlýny. Pokud počet průsečíků není roven ani jedné z hodnot, je uživateli zobrazeno chybové hlášení o neúspěšné identifikaci hry.

7.2.3 Extrakce aktuálního stavu

Poté, co aplikace úspěšně rozpozná hru, je nutné získat její aktuální stav. Podle toho, jakou hru aplikace detekovala, zavolám jednu z funkcí: `getGomokuState`, `getMillsSate` nebo `getCheckersState`. V těchto funkcích dojde k seřazení nalezených průsečíků. Samotné řazení jsem implementoval ve funkci `sortPointsBy`, která přijímá pole průsečíků a řetězec "x"nebo "y", který určuje podle které osy mají být průsečíky seřazeny. K řazení poté využívám výchozí funkce `sort` s vlastním komparátorem. Seřazené body poté uložím do 2D pole, které reprezentuje herní mřížku.

Dalším důležitým krokem je nalézt na desce kameny obou hráčů. K tomu jsem využil funkce `HoughTransform` pro kružnice z knihovny OpenCV. Parametry použité pro detekci se opět liší v závislosti na hře. Minimální průměr je určen jako odhaovaná velikost políčka zmenšená o koeficient. U Dámy to například znamená to, že velikost desky (600px) je poděleno 8, jelikož vím, že deska pro Dámu je tvořena políčky o mřížce 8x8. Maximální je naopak lehce větší než samotné políčko. Tím jsem docílil určité tolerance v detekci daného kruhu.

Nalezené kruhy je poté nutné zařadit na políčko herní desky kam patří patří. U hry Gomoku a Mlýny testuji všechny průsečíky reprezentující políčko na to, zda leží uvnitř některého z nalezených kruhů. Pokud ano je na tomto políčku přítomný kámen. U hry Dáma kameny neleží na průsečících, nýbrž ve čtvercích, které reprezentují hrací pole. Při získávání stavu hry Dáma tedy vytvořím ze seřazených průsečíků pomyslné čtverce a kontroluji, zda střed nalezené kružnice leží vně tohoto čtverce. Tato kontrola probíhá ve funkci `isInSquare`.

Posledním krokem je pak určit, jakému hráči daný kámen patří. To zjistím tak, že nejprve převedu výřez daného kamene do barevného spektra HSL. Poté vypočítám pomocí funkce `mean` z knihovny openCV průměrnou hodnotu světlosti (lightness). Světlost jednotlivých kamenů poté porovnávám s průměrnou hodnotou. Pokud je menší, přiřadím kámen hráči 1, v opačné případě hráči 2. Výsledek reprezentující rozložení kamenů na herní desce je poté jako pole předán dál do dalších aktivit.

7.3 Implementace podporovaných her

Implementace každé hry je rozdělena do čtyř tříd. Každá z těchto tříd začíná prefixem označující konkrétní hru, poté následuje název, co daná třída reprezentuje. První třídou je `Setup` ve které je implementováno ovládání a zobrazení dané hry. Třída `Move` slouží k uložení důležitých informací o pohybu kamenů na desce. Třetí třída pojmenovaná `Board` obsahuje veškerou herní logiku. Poslední ze tříd `AI` pak obsahuje implementovaný Minimax algoritmus s Alfa-beta ořezáváním a metody pro ohodnocení tahu.

První třídu `Setup` zde popíši dohromady pro všechny hry, jelikož se její funkčnost u jednotlivých her příliš nelišší. Tato třída rozšiřuje Aktivitu dané hry. Po inicializaci se volá metoda `onCreate`. V té se ověří zda do třídy nebyl předán aktuální stav herní desky ze skenování. Pokud ne, zobrazí se deska prázdná. Poté dochází k inicializaci ovládacích prvků pro vkládání kamenů a určování tahu. Kromě ovládání se tato třída stará také o zobrazení vypočítaného tahu. Za zmínku zde stojí metoda `showTheMove`, která se stará o vykreslení zobrazeného tahu. A jelikož aplikace vyžaduje po zobrazení daný tah provést, aby bylo možné vypočítat tah náseldující, je zde metoda `executeTheMove`. Ta volá metody hry potřebné pro provedení tahu a následně aktualizuje zobrazenou herní desku.

Zbylé třídy jsou popsány zvlášť u jednotlivých her. Ještě před nimi popíši pomocné třídy, které jsou používané v ostatních třídách.

7.3.1 Pomocné třídy

Vlastnosti a metody, které jsou používaný napříč všemi třídami jsem oddělil a naimplementoval je jako pomocné třídy.

Alerts

Třída alerts obsahuje implementaci vyskakovacího okna pro upozornění. Obsahuje metodu `showAlert`, která jako parametry využívá dva textové řetězce a referenci na aktivitu, ve které se má zobrazit. První řetězec slouží pro nadpis daného upozornění. Druhý pak určuje samotné sdělení daného upozornění.

Decoder

Tato třída slouží k překlad informací mezi grafickým rozhraním a logikou hry. Jelikož kliknutí na políčko v GUI identifikuji dle parametru `tag`, je nutné tuto značku převést na souřadnice dané hry. K tomu slouží metody uvnitř této funkce. Jsou rozdělené na metody `decode` a `encode`. První zmíněné slouží k převodu textové reprezentace `tag` na pozici v poli. Využívají se v místech, kde po kliknutí na položku v grafickém rozhraní je nutné změnu zaznamenat i v poli, které udržuje aktuální stav hry. Metody s prefixem `encode` naopak převádí pozici na řetězec označující položku v grafickém rozhraní. Ty se využívají při zobrazování vypočítaných tahů.

Enums

Jak již samotný název napovídá, tato třída v sobě nese uložené výčtové typy `enum`, konkrétně pak `GameType` obsahující označení pro jednotlivé hry. Výčtový typ `Type` slouží k reprezentaci obsahu herního políčka a obsahuje hodnoty WHITE,BLACK a NONE. Posledním výčtovým typem je pak `CheckerType`, který obsahuje navíc od předchozího výčtového typu ještě označení pro Dámu obou barev (WHITE_QUEEN a BLACK_QUEEN).

StoneCoordinates

Třída `StoneCoordinates` slouží k uložení a práci se souřadnicemi herních kamenů.

7.3.2 Hra Dáma

V této podkapitole je popsán obsah tříd pro hru Dáma, ve které jsou naimplementovány pravidla hry stejně jako ohodnocení a nalezení nejlepšího tahu. Třídy hry dáma obsahují prefix `Checkers`.

CheckersMove

Tato třída slouží k uložení jednotlivých tahů a obsahuje metody důležité pro práci s těmito tahy. Třída obsahuje atributy pro uložení původní pozice kamene a pozice nové. Jestliže tah je vícenásobný přeskok, jsou využita dvě pole: jedno pro uložení souřadnic mezikroků a druhé pro označení pozic sebraných kamenů. Jako poslední je v třídě také uložen typ daného pohybu. Ten identifikuje, zda se jedná o obyčejný pohyb, sebrání kamene soupeřovi, nebo vícenásobné přeskočení.

CheckersBoard

Třída `CheckersBoard` obsahuje implementaci herních pravidel hry Dáma. Herní desku jsem implementoval jako 2D pole typu `CheckerType`, který jsem zmiňoval již dříve. Toto pole je v konstruktoru naplněno hodnotami `EMPTY`. Třída také obsahuje metody `setStone` a `removeStone`, které jak již název napovídá, slouží k přidávání a odebírání herních kamenů. Pro získání souřadnic všech kamenů jednoho hráče slouží metoda `getAllStones`, která jako parametr vyžaduje barvu kamenů které chceme získat. Pole získaných kamenů pak lze použít v metodě `getAllValidMovements`, která se nejprve pokusí získat pole všech možných přeskoků. Pokud je toto pole přeskoků prázdné, je zavolána metoda `getValidMove`, která vrací pole možných pohybů. Pro získání přeskoků slouží metoda `getValidAttacks`. Ta nejprve získá všechny možné jednokrokové přeskoky tak, že kontroluje, zda je na sousedním poli v povoleném směru kámen soupeře, který za sebou má volné místo. Pole těchto přeskoků poté předá do metody `getMultiAttack`. Ta rekurzivně získá možnosti vícenásobných přeskoků a rozšíří o ně původní pole. Poslední důležitou metodou je `makeMove`, která provede tah předaný v parametru. Kroky, které se v metodě provádí, určuje typ tahu. Při pohybu metoda pouze přepíše pozici kamene v poli. V případě přeskoků i odebere soupeřovo kameny z pole.

CheckersAI

Poslední třídou hry Dáma je `CheckersAI`. Tato třída obsahuje implementovanou Minimax s Alfa-beta ořezáváním ve stejnojmenné metodě. Dále třída obsahuje metodu `getEvalNumbers`, ve které dochází k získaní počtu kamenů splňujících definovaná pravidla, která byla zmíněna v kapitole návrhu. Tyto hodnoty jsou poté v metodě `heuristic` vynásobené danými koeficienty a předány jako výsledné ohodnocení metodě `MiniMaxAB`.

7.3.3 Hra Mlýny

V této kapitole je popsán obsah tříd hry Mlýny. Tyto třídy jsou označeny prefixem `Mills`.

MillsMove

Třída `MillsMove` reprezentuje jednotlivé tahy ve hře Mlýny. Pro uchování informace o tahu ve hře Mlýny jsou zapotřebí tři položky. První z nich je číslo představující pozici v herním poli, kam byl kámen uložen nebo přesunut. Druhá je místo, odkud byl kámen přesunut. Poslední pak označuje pozici kamene, který byl odebrán. Jelikož poslední dvě se využívají jen v určitých případech, jsou ve výchozím stavu nastavené na -1.

MillsBoard

Stejně jako u Dámy tato třída představuje herní desku a pravidla hry Mlýny. Herní deska je implementována jako jednorozměrné pole, kde nejnižší index reprezentuje políčko v levém spodním rohu. Posledním indexem je pak políčko ležící v pravé horní části. Indexování je tedy implementování zleva doprava po řádcích a směrem nahoru. Opět jsou zde nutné metody pro odebírání a přidávání kamenů, získaní kamenů jedné barvy, provedení tahu či nalezení kamenů, které je možné odebrat. Jelikož kameny lze odebírat pouze pokud nejsou součástí mlýnu, bylo nutné implementovat tuto kontrolu. Kontrolu jsem implementoval v metodě `isMill`.Ta obsahuje příkaz switch s návěštími pro každou pozici. V návěští je pak volána jako návratová hodnota metoda `checkMills` s parametry sousedních pozic, které reprezentují mlýn. Metoda `checkMills` vrací true/false, podle toho zda na všech pozicích leží kameny stejné barvy a je zde tedy uzavřený mlýn. Obdobný způsob použití příkazu `switch`, jsem použil i v metodě `getNeighbours`. Ta vrací pole souřadnic sousedících políček.

MillsAI

V této třídě se opět nachází implementace metody pro určení nejlepšího tahu `MiniMaxAB`, jako tomu bylo u hry Dáma. V metodě `heuristic` je podle aktuální fáze hry voleno z třech různých formulí pro ohodnocení. Kontrola jednotlivých pravidel při ohodnocení probíhá ve funkcích `getConfigs` a `getMillsConfigs`.

7.3.4 Hra Gomoku

Poslední implementovanou hrou je hra Gomoku. Její třídy začínají prefixem `Gomoku`.

GomokuMove

Jelikož ve hře Gomoku dochází pouze k vkládání kamenů, obsahuje tato třída pouze dva atributy. První atribut reprezentuje souřadnice kam byl kámen položen. Druhý pak ukládá ohodnocení tohoto tahu.

GomokuBoard

Herní deska, kterou reprezentuje tato třída je implementována jako 2D pole. Protože by bylo velice neefektivní, aby algoritmus pro určení nejlepšího tahu zkoušel pokládat kameny na všechny volná místa na desce 13x13, je výběr pozic pro možný tah omezen pouze na sousední políčka okolo již vložených kamenů. Metoda `getAllPosiblePlaces` nalezne všechny vložené kameny a jejich souřadnice předa metodě `addNearEmptyPlaces`. Ta prozkoumá okolní políčka a ta, která jsou volná uloží do pole. S tímto polem pak pracuje třída `GomokuAI`. Pro hledání vzorů jsem využil standardní metody `match`, kterou třída `String` v jazyce Java nabízí.

GomokuAI

Třída `GomokuAI` stjeně jako tomu bylo u ostatních her obsahuje `Minimax` a metoda `heuristic`. Metoda `heuristic` u hry Gomoku převede 2D pole, které reprezentuje herní desku na textový řetězec. Tento řetězec je sestavován postupně ve všech směrech, ve kterých hra Gomoku probíhá. Je tedy postupně sestaven řetězec reprezentující řádek, sloupec a oba diagonální směry. V těchto řetězcích jsou následně v metodě `matchPattern` hledané vzory zmíněné v kapitole návrhu.

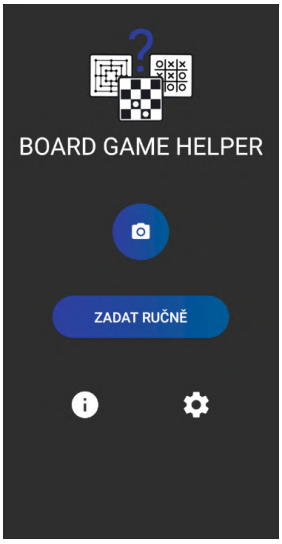
7.4 Grafické uživatelské rozhraní

K vytvoření uživatelského prostředí jsem využil nástroj *Layout Editor*, který je součástí Android Studia. Tento nástroj umožňuje tvorbu grafického uživatelského rozhraní za pomocí jednoduchého grafického editoru, kde je možné přetahovat jednotlivé prvky GUI přímo do scény obrazovky. Obrazovku lze také tvořit a upravovat přímo v XML kódu, který je výstupem grafického editoru. Při implementaci jsem používal obě z nabízených možností a to tak, že po přetažení požadovaných prvků jsem jejich parametry upravoval přímo v kódu.

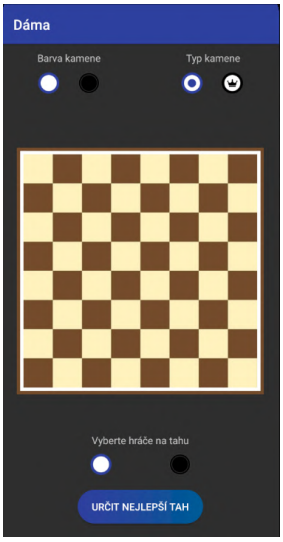
Každý prvek GUI je možné upravit vlastními styly, které lze nadefinovat jako `drawable` v jazyce XML. Toho jsem v implementaci využil hned několikrát. Protože jsem potřeboval přepínače `RadioButton` pro výběry kamenů od sebe nějakým způsobem odlišit, a nechtěl jsem z důvodu narušení designu používat text, zvolil jsem právě vlastní definici stylu v XML. Vlastních stylů využívám také u tlačítek, kde jako pozadí používám přechod z tmavě na světle modrou barvu.

Herní desky jsou do grafického rozhraní vykresleny jako běžné obrázky pomocí komponenty `ImageView`. Pro jednotlivé políčka využívám `FrameLayout`. Ve třídě, která rozšiřuje danou aktivitu jsem na událost `onClick` implementoval přidání kamene. Při kliknutí na políčko je na základě aktuálně vybraných přepínačů přidán kámen. Obrázek kamene ve `FrameLayout` je zobrazen jako popředí (parametr `Foreground`). Tento přístup jsem zvolil, protože jde s popředím v kódu snadno manipulovat a lze ho jednoduše překreslovat, bez nutnosti odstraňovat původní obrázky.

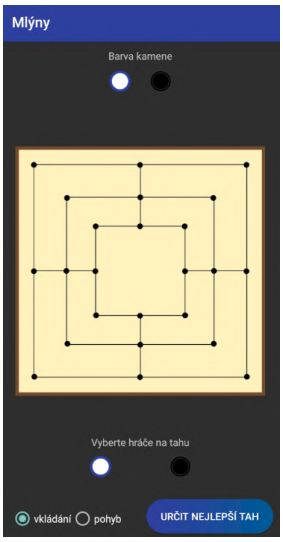
Výsledný vzhled jednotlivých obrazovek je možné vidět na obrázku 7.1.



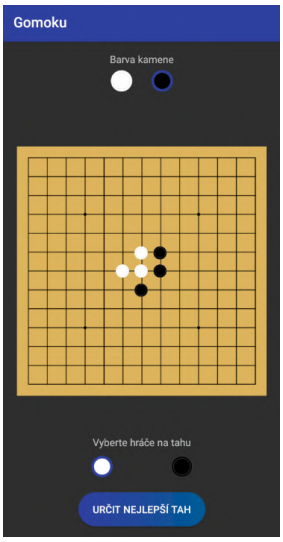
(a) Hlavní menu



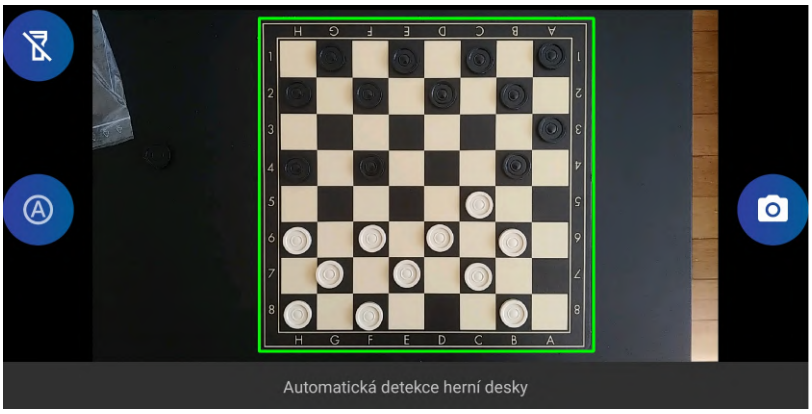
(b) Obrazovka Dáma



(c) Obrazovka Mlýny



(d) Obrazovka Gomoku



(e) Obrazovka detekce

Obrázek 7.1: Ukázka jednotlivých obrazovek aplikace

Kapitola 8

Testování

Důležitou částí vývoje aplikace je také testování. Samotné testování jsem rozdělil na tři části. V první části se věnuji testování detekce herní desky a identifikaci podporovaných her. Druhá část je věnovaná určení nejlepšího tahu. Zaměřuji se v ní na její úspěšnost a časovou náročnost. V poslední podkapitole jsem shrnul poznatky z uživatelského testování.

8.1 Testování detekce a identifikace hry

Při testování detekce a identifikace hry jsem využil kromě emulátoru také několik fyzických zařízení. Konkrétně se jednalo o chytré telefony LG V30, LG G6 a Samsung Galaxy A71. Cílem této části bylo otestovat aplikaci na co největším množství herních desek. Testování probíhalo na několika fyzických hrách (2x - dáma, 2x - mlýny, 1x - gomoku) od různých výrobců, deskách vytisknutých na obyčejný papír a detekci jsem také vyzkoušel při skenování z monitoru. Testování probíhalo v různých světelných podmínkách a to konkrétně při denním světle, umělém osvětlené a v šeru za použití svítilny.

Detekce herní desky

Jako první na řadu přišlo testování automatické detekce herní desky. Při testování se ukázalo, že světelné podmínky v tomto případě nemají příliš velký vliv na nalezení desky ve snímku. Jak jsem očekával, daleko větší vliv mělo pozadí. Testování jsem provedl na několika stolech různé barvy (černý, dub sonoma a ořech), kde se detekce ukázala jako úspěšná. Neúspěch detekce zapříčinily až ostatní objekty, které se vyskytovaly v okolí hry. K neúspěchu také došlo u skenování, při kterém hra ležela na dřevěných parketách, a nebo v případě skenování z obrazovky monitoru, na kterém byly v okolí zobrazené hry i jiné objekty. Ve zmíněných případech je uživatel nucen použít funkci ruční detekce.

Identifikace konkrétní hry

Rozpoznání hry bylo dalším krokem testování. U her vytištěných na obyčejný papír bylo dosaženo za všech světelných podmínek vysoké úspěšnosti. Při provedení 14 skenů bylo správně rozpoznáno 11 z nich. Neúspěšná identifikace hry nastala u hry Mlýny, při použití příliš velkých kamenů. Ty totiž zakrývaly značnou část čáry, kterou se algoritmu nepodařilo detekovat.

Jako druhé byly testovány hry zakoupené v obchodě. Desky her Dáma a Mlýny měly lesklý povrch. Zde se ukázalo jako kontraproduktivní používat svítilnu z důvodu odlesků.

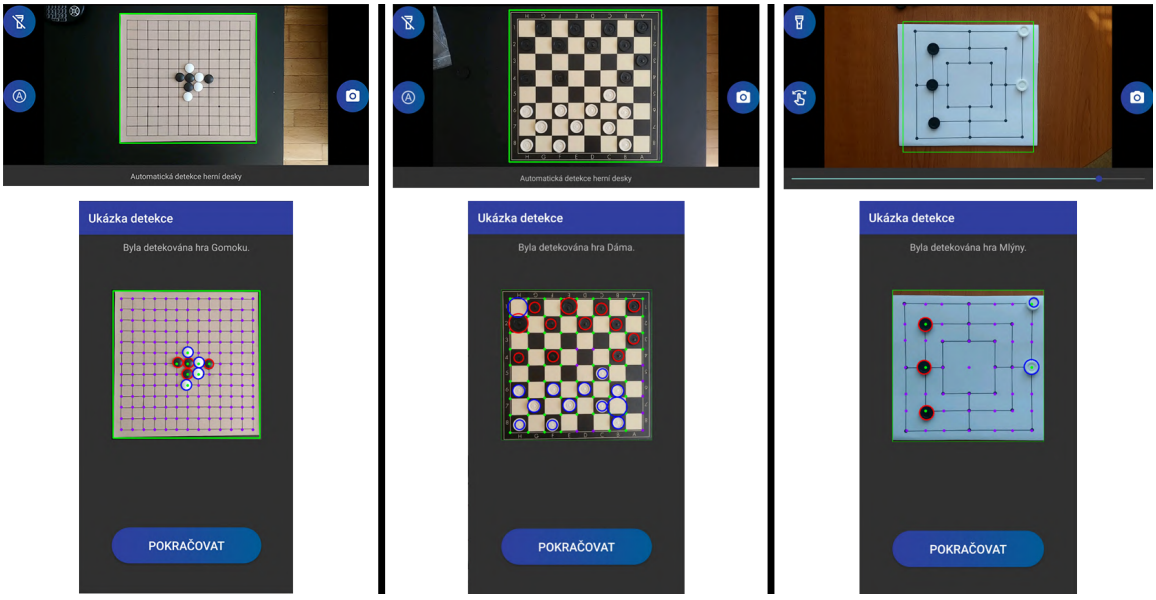
Algoritmus měl také problém s rozpoznáním her z desek, u kterých byla políčka pokrytá výraznou texturou s čarami (např.: imitace dřeva). Naopak hry, které měly pozadí políček jednobarevné byla úspěšnost identifikace větší než 80%. Naopak u zakoupené hry Gomoku se mi podařilo dosáhnout 100% úspěšnosti. Její deska je vyhotovena z kůže, na které jsou výrazné černé čáry. Nedocházelo tedy k odleskům ani k detekci falešných čar.

U detekce z monitoru záleželo na kvalitě skenovaného obrázku, stejně jako na typu daného monitoru. U monitorů, kde na fotoaparátu bylo zřetelně vidět rozložení jednotlivých pixelů docházelo k častým falešným nálezům čar. Na monitoru AOC 24G2U se z 10 pokusů podařilo hru správně rozpoznat pouze dvakrát. Druhým testovaným monitorem byl BENQ G2220HD, kde se úspěšnost identifikace pohybovala okolo 50 %.

Detekce kamenů na herní ploše

Poslední částí testování detekce je hledaní herních kamenů. Toto testování ukázalo, že při detekci za zhoršených světelných podmínkách docházelo k falešným detekcím z důvodu přílišného šumu, který byl na snímcích přítomen. Stejný jev jsem zaznamenal také při snímaní hry zobrazené na monitoru. K tomu docházelo zřejmě ze stejných důvodů, které byly popsány výše u identifikace her. U hry Dáma, která měla vyhotovení políček ve velmi tmavé barvě docházelo k nezaznamenání některých černých kamenů. Tento problém vyřešilo použití svítilny. U některých provedení hry Mlýny, která měla velikost polí podobnou s velikostí kamenů docházelo k mylnému označení těchto polí. V ostatních případech byla detekce kamenů úspěšná.

Rozpoznání barvy kamene bylo ve většině případů správné, a to i u odlišných barevných kombinacích, než je černá/bílá. Ke špatnému rozpoznání barvy kamene docházelo převážně při použití svítilny a lesklých kamenů. V takovém případě byly některé z černých kamenů špatně označeny jako bílé.



Obrázek 8.1: Ukázka detekce u jednotlivých her

8.2 Testování určení nejlepšího tahu

Testování určení nejlepšího tahu bylo provedeno proti několika různým počítačem řízených protivníkům, stejně jako proti lidským protihráčům. V tabulce 8.1 jsou vidět výsledky testování u jednotlivých her. U her Dáma a Mlýny bylo k testování využito prohledávání do hloubky 4. U hry Gomoku pak hloubky 2. V případě, že u počítačem řízeného protivníka nebyla uvedená obtížnost je v tabulce zařazen do obtížnosti lehká.

Pro účely testování byly využity webové hry 247checkers[1], cardgames.io[2], onlinesologames.com[3], playok.com[4], gomokuonline.com[5] a gomoku.yjya[6].

Testování odhalilo vysoukou časovou náročnost při určování nejlepšího tahu ve hře Gomoku, což připisuji tomu, že je v mé implementaci pro každou zkoumanou možnost znovu vytvářen textový řetězec reprezentující herní pole. Proto jedním z vylepšení při budoucím vývoji bude změna fungovaní ohodnocení pro hru Gomoku. Toho by mohlo být docíleno jinou metodou hledání vzorů. Případně řazením možných tahu dle určitých vlastností hry Gomoku tak, aby vhodnější tahy byly nalezeny co nejdříve a nedocházelo tak ke zbytečnému prohledávání ostatních stavů.

Dáma	Lehká	Střední	Těžká	Lidský protihráč
247checkers	3 ze 3	2 ze 3 (1x remíza)	1 ze 3 (1x remíza)	-
cardgames.io	2 ze 3 (1x remíza)	-	-	2 z 5 (1x remíza)
Hra s kamarády	-	-	-	1 z 3
Mlýny	Lehká	Střední	Těžká	Lidský protihráč
onlinesologames.com	3 ze 3	-	-	-
playok.com	-	-	-	2 z 5
Hra s kamarády	-	-	-	2 z 3
Gomoku	Lehká	Střední	Těžká	Lidský protivník
gomokuonline.com	3 ze 3	-	-	-
gomoku.yjyao	0 ze 3	-	-	-
Hra s kamarády	-	-	-	1 ze 3

Tabulka 8.1: Přehled výher z testování ohodnocovacích funkcí

8.3 Uživatelské testování

Uživatelského testování se zúčastnilo celkem 9 uživatelů z mého okolí. Uživatelé si nainstalovali aplikaci, kterou následně zkoušeli používat bez jakékoliv poznámky z mé strany. Poté byla s každým z uživatelů vedena diskuse ohledně použitelnosti aplikace a případných vylepšeních. Na závěr byl uživatelům předložen krátký dotazník, kde jednotlivé prvky mohli hodnotit 0-5 body. Výsledek dotazníku je shrnut v tabulce 8.2. Uživatele jsem rozdělil do tří skupin:

- **Začátečníci** - používají telefon primárně ke komunikaci a aplikace používají pouze výjimečně

[1] Dostupné z https://www.247checkers.com/
[2] Dostupné z https://cardgames.io/checkers/
[3] Dostupné z http://www.onlinesologames.com/ninemensmorris/
[4] Dostupné z https://www.playok.com/en/mill/
[5] Dostupné z https://gomokuonline.com/
[6] Dostupné z https://gomoku.yjyao.com/

- **Středně pokročilí** - Mají zkušenosti s používáním některých aplikací, ale nevyužívají je denně.
- **Zkušení** - Aplikace na chytrém zařízení používají na denní bázi. Mají zkušenosti s mnoha mobilními aplikacemi

Testování se zúčastnily dva uživatelé začátečníci, jeden středně pokročilý a šest zkušených uživatelů.

	Vzhled	Ovladání	Skenovací funkce	Zobrazení tahu	Doporučení tahu
Začátečníci	5	4,5	4	5	5
Středě pokročilí	5	5	3	5	5
Zkušení	4,6	4,8	3,5	5	4

Tabulka 8.2: Průměrné hodnoty hodnocení z dotazníku

Po vyhodnocení dotazníku a získaných poznatků od uživatelů jsem dospěl k závěru, že mnou navržené uživatelské rozhraní je dostatečně intuitivní a uživatelům se líbí. Zobrazení tahů je pro uživatele přehledné a dokážou jej replikovat v reálně hře. Ovládání změn na herní desce nedělalo středně pokročilým a zkušeným uživatelům problém. U začátečníků bylo nejprve nutné si aplikaci pořádně vyzkoušet. Po chvíli s ovládáním již problém neměli ani oni. S výsledky doporučených tahů byli uživatelé ve většině případů spokojeni. Většina uživatelů se shodla, že další směr vývoje by měl být zaměřen na vylepšení a přesnost identifikace jednotlivých her. Některým uživatelům se totiž naskenování stavu hry podařilo až na několikátý pokus. Na základě uživatelské zpětné vazby byla do aplikace také přidána možnost vypnout mezikrok zobrazující výsledky detekce.

Kapitola 9

Závěr

Cílem této bakalářské práce bylo vytvořit aplikaci pro operační systém Android, která ze snímku dokáže rozpoznat jednu ze tří podporovaných her a extrahovat její aktuální stav. Získaný stav je následně uživateli zobrazen a ten má možnost jej upravit, případně si nechat určit a zobrazit nejlepší možný tah.

Před začátkem implementace bylo nutné vybrat a seznámit se s pravidly her, které budu implementovat. Následně jsem se seznámil s možnostmi vývoje aplikací pro operační systém Android. K samotnému návrhu klíčových funkcí aplikace bylo nutné nastudovat principy počítačového vidění jako je detekce hran, prahování, Houghova transformace a další. Bylo také nutné si zopakovat metody pro hraní her, které jsem již z dřívějška znal z předmětu o základech umělé inteligence (IZU).

Aplikace byla naimplementována v jazyce Java s využitím oficiálního vývojového prostředí Android Studio. K implementaci prvků počítačového vidění jsem využil knihovnu OpenCV, která obsahuje velké množství již naimplementovaných metod pro zpracování obrazu.

Výsledná aplikace je spustitelná na zařízeních s operačním systémem Android. Správnost fungování byla ověřena na zařízeních s verzí Android 8 nebo novější. Funkčnost jednotlivých částí aplikace byla ověřena testováním. Na základě vlastního testovaní a zpětné vazby od uživatelů, bylo odhaleno několik omezení detekce a identifikace hry. Spolehlivost rozpoznání her je ovlivněna kvalitou osvětlení, materiálem a grafickým zpracováním dané hry. Na základě tohoto testování je primárním cílem budoucího vývoje vylepšit detekci a identifikaci. Pro tyto účely by mohlo být využito například neuronové sítě. Dalším možným vylepšením aplikace v budoucnu je rozšíření počtu podporovaných her či vylepšení stávajících funkcí pro ohodnocení.

Literatura

[1] *Canny Edge Detection*. 2021 [cit. 2021-04-21]. Dostupné z: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html.

[2] Davis, M. D. a Brams, S. J. *Game theory*. [cit. 2021-03-30]. Dostupné z: https://www.britannica.com/science/game-theory.

[3] Gasser, R. *Solving nine men's morris*. 1996. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8640.1996.tb00251.x.

[4] *Opening rules | GomokuWorld.com* [online]. [cit. 2021-04-04]. Dostupné z: http://gomokuworld.com/gomoku/2.

[5] Griffiths, D. a Griffiths, D. *Head First Android Development: A Brain-Friendly Guide*. 2. vyd. O'Reilly Media, Inc., 2017. ISBN 1491974052.

[6] *Hough Circle Transform*. 2021 [cit. 2021-04-23]. Dostupné z: https://docs.opencv.org/master/d4/d70/tutorial_hough_circle.html.

[7] *Hough Line Transform*. 2021 [cit. 2021-04-23]. Dostupné z: https://docs.opencv.org/master/d3/de6/tutorial_js_houghlines.html.

[8] Huggan, M., Nowakowski, R. J. a Ottaway, P. *Simultaneous Combinatorial Game Theory*. 2018 [cit. 2021-03-30].

[9] *Image Thresholding*. [cit. 2021-04-21]. Dostupné z: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html.

[10] *Introduction*. 2021 [cit. 2021-04-27]. Dostupné z: https://docs.opencv.org/master/d1/dfb/intro.html.

[11] Lasker, E. *Go and Go-moku: The Oriental Board Games*. 2. vyd. Dover Publications, 1960. ISBN 9780486206134.

[12] Liao, H. *New heuristic algorithm to improve the Minimax for Gomoku artificial intelligence*. Iowa State University, 2019. Dostupné z: https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=1491&context=creativecomponents.

[13] Mańdziuk, J., Kusiak, M. a Walędzik, K. *Evolutionary-based heuristic generators for checkers and give-away checkers*. Blackwell Publishing, 2007. Dostupné z: https://pages.mini.pw.edu.pl/~mandziukj/PRACE/es_init.pdf.

[14] Nowakowski, R. a Mathematical Sciences Research Institute (Berkeley, C. *Games of No Chance*. Cambridge University Press, 1998. Mathematical Sciences Research Institute Publications. ISBN 9780521646529.

[15] PETCU, S.-A. a HOLBAN, S. *Nine Men's Morris: Evaluation Functions*. Politehnica University of Timisoara, 2008. Dostupné z: http://www.dasconference.ro/papers/2008/B7.pdf.

[16] *Platform Architecture* [online]. 2021 [cit. 2021-03-26]. Dostupné z: https://developer.android.com/guide/platform.

[17] *Pravidla české dámy* [online]. Česká federace dámy, 2021 [cit. 2021-03-20]. Dostupné z: http://www.damweb.cz/pravidla/cdfull.html.

[18] Schaeffer, J., Lake, R., Lu, P. a Bryant, M. *Chinook The World Man-Machine Checkers Champion*. Mar. 1996. Dostupné z: https://ojs.aaai.org/index.php/aimagazine/article/view/1208.

[19] Sonka, M. *Image processing, analysis, and machine vision*. 4. vyd. Stamford, CT, USA: Cengage Learning, 2015. ISBN 978-1133593607.

[20] SZELISKI, R. *Computer vision : algorithms and applications*. 1. vyd. SPRINGER NATURE, 2020. ISBN 978-3030343712.

[21] *Mobile and Tablet Android Version Market Share Worldwide* [online]. StatCounter, 2021 [cit. 2021-03-26]. Dostupné z: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide.

[22] Zbořil, F. V. *Základy umělé inteligence – Studijní opora* [online]. 2021 [cit. 2021-04-01]. Dostupné z: https://www.fit.vutbr.cz/study/courses/IZU/private/2021-opora-IZU.pdf.

Příloha A

Obsah přiloženého datového média

- **doc/** - Složka obsahující bakalářskou práci ve formátu PDF a její zdrojové kódy.
  - **latex/** - Zdrojové soubory textu bakalářské práce.
  - **xsirok09_BP.pdf** - Tento dokument ve formátu PDF.
- **app/** - Složka obsahující aplikaci a její zdrojové soubory.
  - **src/** - Zdrojové kódy aplikace (projekt Android Studio)
  - **GameBoardHelper.apk** - Instalační balíček aplikace
- **print/** - Složka obsahující PDF podklady k tisku herních desek podporovaných her.