Aplikace pro hru Mancala implementující optimální strategie

Bc. Jiří Andrla

Diplomová práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jiří Andrla**
Osobní číslo:	**A16417**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Aplikace pro hru Mancala implementující optimální strategie**
Téma anglicky:	**An Application for the Mancala Game Implementing Optimal Strategies**

Zásady pro vypracování:

1. **Popište deskovou hru Mancala a její varianty.**
2. **Zpracujte rešerši o existujících desktopových a mobilních aplikacích této hry.**
3. **Popište optimální strategie hry Mancala.**
4. **Vyberte a popište softwarové nástroje pro vytvoření vlastní aplikace.**
5. **Vytvořte aplikaci pro hru Mancala implementující popsané optimální strategie umožňující hrát člověku proti stroji.**
6. **Ověřte funkcionalitu a efektivitu vytvořené aplikace.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **BROWN, Seth. How to Win at Mancala? Basic Strategy. The Spruce Crafts [on-line]. 27. 9. 2018 [cit. 2018-10-02]. Dostupné z: https://www.thesprucecrafts.com/how-to-win-at-mancala-basic-strategy-411832.**
2. **Mancala. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 30. 9. 2018 [cit. 2018-10-02]. Dostupné z: https://en.wikipedia.org/wiki/Mancala.**
3. **MAŇAS, Miroslav. Teorie her a optimální rozhodování. Praha: SNTL, 1974.**
4. **HYKŠOVÁ, Magdalena. Teorie her a optimální rozhodování. Praha, 2009. Odborná publikace. ČVUT. [cit. 2018-10-02]. Dostupné z WWW: http://euler.fd.cvut.cz/predmety/teorie_her/hry.pdf.**
5. **HRUBÝ, Martin. Doprovodné texty ke kurzu Teorie her [online]. Brno, 2010. [cit. 2018-10-02]. Dostupné z: http://www.fit.vutbr.cz/ hrubym/THE/sk-2-nekoo.pdf. Brno University of Technology.**

Vedoucí diplomové práce: **doc. Ing. Libor Pekař, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání diplomové práce: **3. prosince 2018**

Termín odevzdání diplomové práce: **15. května 2019**

Ve Zlíně dne 7. prosince 2018

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 15. 5. 2019 ... Jiří Andrla v. r. ...

podpis autora

ABSTRAKT

Tato práce z oblasti Teorie her je zaměřena na hry typu Mancala, konkrétně v západním světě nejrozšířenější variantu – Kalahu. Hlavním cílem této práce je navrhnout a vytvořit aplikaci, která implementuje optimální strategii vedoucí k vítězství v této hře. Na základě analýzy pravidel Kalahy byly navrženy nové strategie s cílem zajistit hráči vítězství v každé herní partii. Tyto strategie byly implementovány v rámci nově vytvořené aplikace pro hraní hry Kalahy a byly vyhodnoceny jejich výkonnosti. Při analýze bylo zjištěno, že výsledek velkou měrou ovlivňuje pozice začínajícího hráče, který může od počátku volit výhodné tahy. Experimenty ukázaly, že lidský hráč v pozici druhého hráče proti některým implementovaným strategiím nedokázal ani jednou vyhrát. Naopak v případě, že začínal, vždy nalezl takovou posloupnost tahů, aby dokázal vyhrát. Obecně lze konstatovat, že navržené strategie mohou konkurovat lidským hráčům.

Klíčová slova: Mancala, Kalaha, Oware, Toguz kumalak, Teorie her, C++, Qt, strategie, implementace

ABSTRACT

This thesis belonging under the umbrella of the Game Theory is focused on games of the Mancala type — namely its most widespread variant in the western world, the Kalah game. The principal goal of this work is to design and create an application implementing a strategy leading to victory in this particular game. Based on the analysis of Kalah rules, we designed a novel winning strategies. This strategies were then implemented within a new Kalah game application and their performance were experimentally evaluated. Our analysis shows that the player who has the initial move usually has a decisive advantage as they can choose the most favorable moves from the very beginning. The experimental evaluation shows that when playing as the second player, the human competitor could not defeat some of the designed strategies at all. On the other hand, with the first move advantage, the human player always found a sequence of moves leading to victory. We conclude that the implemented strategies are able to thoroughly challenge human players.

Keywords: Mancala, Kalaha, Oware, Toguz kumalak, Game theory, C++, Qt, strategy, implementation

Děkuji všem, kteří mě podporovali v činnosti na závěrečné diplomové práci. Zvláště panu doc. Ing. Liborovi Pekaři, Ph.D. za trpělivost, vedení a nasměrování řešení této práce.

OBSAH

ÚVOD

Mancalové hry jsou jedny z nejstarších deskových her na světě, ale i přes svoje stáří nejsou v Evropě v obecném povědomí. I z tohoto důvodu jsou pro nás zajímavé. Vyznačují se poměrně jednoduchými pravidly, které mohou být zkoumány a analyzovány. Zajímavé jsou především z pohledu aplikace optimálního rozhodování na praktické problémy. Mancalové hry totiž unikly hlavnímu zájmu řešení pomocí umělé inteligence nebo jiných metod pro volbu optimálního rozhodování. Pravidla mancalových her by mělo být možné dobře analyzovat a tím pádem volit různé herní strategie pro optimální hru. Tím, že Mancala není náročná na herní vybavení, je snadné ji přiblížit i neznalým lidem a demonstrovat principy, které byly na řešení herních strategií použity. Další výhodou je, že svým principem je atraktivní pro děti a tudíž zvyšuje lepší vysvětlovací schopnost principů aplikace rozhodovacích strategií na reálné problémy. V konečném důsledku je hra Mancala vhodná pro implementaci strategií, které povedou ke hře hráče proti stroji. Počítačová inteligence umožňuje hru i v případě, kdy není k dispozici lidský protihráč a přidává do hry další rozměr a to ten, zda hráč dokáže porazit již naprogramovaný algoritmus.

Tato práce si klade za cíl aplikovat Teorii her na mancalové hry, konkrétně na variantu Kaluhu. Pro tento účel bude vytvořena aplikace, která bude implementovat zjištěné poznatky a bude umožněna hra proti stroji. Pro optimální rozhodování budou navrženy nové strategie s cílem zajistit vítězství v každé herní partii.

I. TEORETICKÁ ČÁST

1 Hra Mancala

Mancala je pojmenování celé skupiny strategických her, lišících se v konkrétních pravidlech. Variant her existuje velmi mnoho, všechny však mají společné rysy. Pro hru je zapotřebí hrací deska s důlky a kameny. Hrací deska bývá nejčastěji vyřezána ze dřeva, ale může být z kamene, slonoviny, nebo z kovu. Důležité jsou důlky v hrací desce, do kterých jsou na začátku hry umístěny kameny, které se podle pravidel přemísťují s cílem, aby hráč získal co nejvíce kamenů. Hra dle popisu není náročná na vybavení, které lze vyrobit z provizorních věcí. Například místo důlků v herní desce, mohou být v zemině vyhloubeny jamky a místo kamenů mohou být použity semena, oblázky, fazole, apod. Takto se hra nejčastěji hraje v rozvojových zemích.

1.1 Historie Mancaly

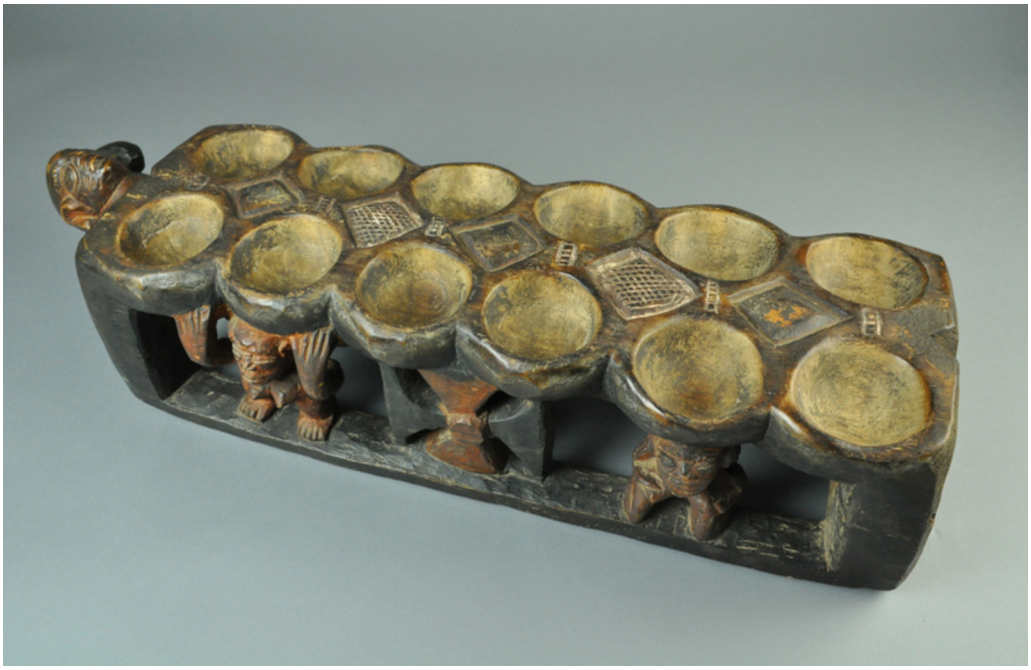
Mancala je považována za jednu z nejstarších her na světě. Její historii je však obtížné přesně zrekonstruovat. Je to dáno tím, že ji lze hrát na zemi, kde nezanechá žádné dlouhodobé stopy, nebo s dřevěnou deskou, která se v horizontu desítek let rozpadne. Badatelé označili některé objevy jako možné Mancala desky. Nejstarší z nich jsou kamenné desky s řadami důlků, nalezené v neolitických lokalitách Jordánska a Íránu. Pozdější objevy, které jsou určitou měrou považovány za Mancala desky jsou z doby bronzové z Kypru, Egyptu, oblasti Levantu a Mezopotámie. Přestože mnoho z těchto objektů nese silnou podobnost s Mancala deskami, nelze prokázat, že skutečně byly používány pro hru Mancala. Nejstarší plnohodnotný odkaz na Mancalu je z 10. století v encyklopedické sbírce *Kitab al-Afghani* z Persie. Mancala byla populární v celém středověkém muslimském světě, je to zmíněno i v jednom příběhu Arabských Nocí. Slovo, které dnes používáme pro hru Mancala pochází z arabského slova "*naqala*", které znamená "*pohybovat se*". Ať už je století vzniku hry jakékoli, odborníci se shodují, že se Mancala vyvinula v oblasti Blízkého východu a odtud se v důsledku arabské expanze dostala až do západní Afriky nebo do Indonésie.[Pearson(2016)]

Nedávné studie o rozdílných pravidlech a odlišných hrách Mancaly daly nahlédnout do rozložení typů her podle území. Po analýze těchto dat bylo zjištěno, že rozložení je spojeno s migračními trasami, které se utvářeli i několik staletí. Záznam o tom, že se hra hrála i v Evropě je ze 17. století z Anglie, kde ji hráli obchodníci. Avšak v Evropě si hra nikdy nezískala velký počet příznivců, s výjimkou oblasti Baltského moře a Bosny, kde se varianta Mancaly s názvem Ban-Ban hraje dodnes. Mancala byla také nalezena v Srbsku a v Řecku. Dvě tabulky Mancala z počátku 18. století se nacházejí v zámku Weikersheim v jižním Německu.

Ve spojených státech amerických byla tradiční Mancala hra zvaná *Warra* hrána v Louisianě na počátku 20. století a komerční verze zvaná *Kalah* se stala populární ve

čtyřicátých letech minulého století. Na Kapverdách je Mancala známá jako "*ouril*". Hraje se na ostrovech a přivezli ji do Spojených států kapverdští přistěhovalci. Hraje se dodnes v kapverdských komunitách v Nové Anglii.[Wik(2018a)]

Zatímco šíření Mancaly bylo podpořeno její jednoduchostí a nenáročností na hrací desku, Penn Muzeum v Philadelphii disponuje sbírkou vzácných uměleckých desek pro hru Mancala. Tyto desky jsou většinou z velmi kvalitního a drahého dřeva, do kterého jsou umělecky vyřezány jak hrací důlky tak nejrůznější ozdoby jako například leopardy, lidské tváře, hadi, apod. Mancala deska může tvořit celý hrací stůl, nebo může být sochou, do které jsou zakomponovány hrací důlky. Tyto luxusní hrací desky měli potvrzovat významnost, bohatství a postavení svých majitelů. Často se africké umělecké verze Mancaly vyznačovaly tím, že hrací deska byla socha ženské postavy. Celkově k ozdobným prvkům byl používán ikonografický jazyk, často zahrnující obrazy převzaté z legend a folklóru. Zástupce uměleckých desek reprezentuje obrázek 1.1 z Penn Muzea ve Philadelphii.



Obr. 1.1 Hrací deska Mancaly v Penn Muzeu

1.2 Příklady jednotlivých typů Mancala her

1.2.1 Kalaha

Kalaha je varianta Mancaly dovezené do Spojených států Williamem Juliusem Championem Jr. v roce 1940. Tato hra je někdy také nazývána "Kalahari", možná falešnou etymologií z pouště Kalahari v Namibii. Jedná se o nejoblíbenější a komerčně dostupnou variantu na Západě.

Hrací deska pro Kalahu má na každé straně 6 malých důlků, tzv. domů a velkou jámu na každém konci, nazvanou pokladnici. Cílem hry je zachytit více kamenů (semen) než soupeř.

Pravidla Kalahy

- Hrací deska je umístěna příčně mezi hráči.
- Na začátku hry jsou do každého domu vloženy čtyři kameny. Obě pokladnice jsou prázdné.
- Každý hráč kontroluje šest domů a jejich kameny umístěné na své straně desky. Skóre hráče je počet kamenů v pokladnici vpravo.
- Losem vybraný hráč začíná. Po každém tahu se hráči střídají, až na výjimky viz níže.
- Hráč si na své straně vybere jeden ze šesti domů, odebere z něho všechny kameny a poté je pokládá vždy po jednom proti směru hodinových ručiček do každého následujícího domu. Do domů je vkládán vždy jen jeden kámen.

  Dostane-li se hráč při rozdělování kamenů až ke své pokladnici, vloží i do ní jeden kámen. Pokud mu zbývají k rozdělení další kameny, pokračuje v přidělování kamenů do protivníkových domů. Pokud by k rozdělení měl hráč tolik kamenů, že by se dostal až k soupeřově pokladnici, musí ji přeskočit a pokračovat v přidělování kamenů opět do vlastních domů. Kameny, které se nachází v pokladnici se nevybírají a nepřerozdělují.
- Poslední kámen má dva důležité významy.

  Pokud připadne poslední kámen během tahu do pokladnice, získává hráč další tah. Toto pravidlo se užívá opakovaně.

  Jestliže připadne poslední kámen do prázdného domu na vlastní straně, smí být tento kámen a všechny kameny z protilehlého protivníkova domu odebrány a vloženy do vlastní pokladnice. Tímto je tah ukončen, ve hře pokračuje protivník. Pokud je protivníkův dům prázdný, zůstává kámen v domě a tah je ukončen.
- Hra končí, jakmile jeden hráč vyprázdní všechny své domy. Protivník přemístí zbylé kameny ze svých domů do své pokladnice a oba hráči si spočítají kameny ve svých pokladnicích. Vítězem hry je hráč, který má více kamenů.

V této práci je tah pojmenován přerozdělování kamenů. Můžeme se setkat s označením setí, odtud anglické *sowing*. Setí se označuje v kontextu, kdy místo kamenů jsou používána semena, pak se při tahu z domu vyberou semena a sejí se do ostatních domů.

I pro Kalahu existují různé varianty pravidel. Jedná se o hru s více kameny. Na začátku je do domů vloženo místo 4 kamenů 5, nebo 6 kamenů. Další změna pravidel může být rozdělování kamenů po směru hodinových ručiček (hraje se s více kameny). Alternativní pravidlo může být požito při vložení posledního kamene do svého prázdného domu, i přes stav, kdy soupeř nemá v protilehlém domě žádný kámen, je kámen vložen do pokladnice. Další varianta změny pravidla je taková, že zbylé kameny, po vyprázdnění domů jednoho hráče nejsou započítány do protihráčova skóre.[Wik(2018b)]

Na webové stránce *www.thesprucecrafts.com* jsou rady jak hrát Kalahu. Pro obecnou variantu, která je popsána zde.

- Začínající hráč má výhodu. Tu by měl zužitkovat tahem, při němž poslední kámen bude přiřazen do pokladnice. Tím získá tah navíc a poté hráč může uvolnit poslední dům hned u pokladnice. Tím si zajistí, že když se do tohoto domu dostane kámen, bude ho moci umístit do pokladnice a získá tah navíc.
- Čím více bude mít hráč vyprázdněné domy, tím bude mít více příležitostí k zajímání soupeřových kamenů.
- Zároveň by hráč měl kontrolovat, zda soupeř nemá prázdné domy, aby mu protihráč nezachytil jeho kameny. Pokud takový tah hrozí, je potřeba volit takový tah, aby se soupeřův prázdný dům zaplnil, nebo aby se přerozdělili kameny z ohroženého domu.
- Další rada se týká výhodnosti získání tahu navíc. Hráč může získat více kamenů v jednom tahu. [Brown(2019)]

1.2.2 Oware

Tato verze hry Mancala se vyskytuje v Karibiku a v oblasti Ashanti ve státe Ghaně. Oware má stejný základ jako Kalaha, tzn. hrací deska má 12 domů a dvě pokladnice, jednu pro každého hráče. Oware stejně jako Kalaha je standardně hrána s 48 kameny. Rozdíly jsou v přemisťování kamenů.

Hra začíná se čtyřmi kameny v každém domě. Hráč vybere dům, ze kterého vezme kameny. Ty rozdělí do ostatních domů proti směru hodinových ručiček, přitom vkládá kameny jen do domů, ne do pokladnic. Pokud se přerozděluje více jak 11 kamenů, vynechává se i dům, ze kterého se kameny vybraly, tzn. že dům ze kterého byly odebrány kameny zůstane vždy na konci tahu prázdný. V případě, že je poslední kámen vložen do soupeřova domu, ve kterém se nacházely před vložením jeden nebo dva kameny, získává všechny tyto kameny (včetně vloženého) hráč, který zahrál tento tah. Tímto však získávání kamenů nemusí končit. Hráč se podívá, zda v sousedním domě po pravé

straně nejsou také dva nebo tři kameny. Pokud ano, tak hráč získává i tyto kameny. Takto postupuje opakovaně vpravo, dokud není v kontrolovaném domě jiný počet kamenů než dva až tři, nebo nenarazí na pokladnici. Hra končí v případě, že hráč, který je na tahu, nemůže udělat žádný tah protože všechny jeho domy jsou prázdné. Vítězí ten hráč, který má více kamenů v pokladnici.

Stejně jako ve verzi Kalaha, tak v Oware existují alternativní pravidla. V případě, že by hráč svým tahem sebral všechny soupeřovy kameny a znemožnil mu tak pokračovat ve hře, nechá je ležet a nebere nic (i když tah provede a splní podmínky pro zajímání kamenů do pokladnice). Další změna pravidel může být následující. Pokud si hráč může vybrat mezi tahem končícím hru a tahem prodlužující hru, musí zahrát ten druhý (tzv. soupeře nakrmit). Pokud se hráči shodnou, že hra sklouzla do nekonečného opakování, každý si vezme kameny ze svých domů a hru ukončí.[Wikiknihy(2018)]

1.2.3 Toguz kumalak

Tato verze Mancaly se hraje ve střední Asii v Kazachstánu, Kyrgyzstánu, v Tádžikistánu, v Uzbekistánu, v Turkmenistánu, v severovýchodním Afghánistánu, v částech Ruska, v západním Mongolsku a severozápadní Číně. Název znamená v překladu z kazaštiny *"devět koulí"* [Gering(2018)] a právě i z názvu jde odvodit, že hrací deska se od desek pro hru Kalaha a Owari liší. Tvoří ji dvě řady po devíti jamkách - domech a dvě pokladnice, které jsou umístěny mezi řadami domů. Hráči patří vzdálenější pokladnice.

Hráč na tahu si vybere jeden neprázdný dům na své straně desky. Vezme z něho všechny kameny, jeden vrátí zpět, další dá do domu napravo a postupuje po jednom dál dokola (z konce své řady přeskočí na soupeřovu) proti směru hodinových ručiček, dokud nerozdělí všechny. Při tahu z domu jen s jedním kamenem se kámen nevrací zpět, ale přesune se do následující jamky. Po dokončení tahu (a případném zajímání nebo vytvoření tuzdeku) hraje soupeř. Pokud poslední kámen padl do domu na soupeřově straně a je v ní pak sudý počet kamenů, hráč je vezme a přesune do své pokladnice. Tuzdek neboli pastička vznikne, pokud poslední kámen z tahu skončí v domě na soupeřově straně a tento dům tak obsahuje celkem 3 kameny. V tu chvíli se z něho stává past. Vše, co do tohoto domu spadne, může hráč ihned přesunout do své pokladnice. Dům – past se označí, např. kamenem jiné barvy, mincí, apod. Každý hráč může vytvořit jen jednu past. Tuzdek nejde zrušit, zůstává do konce hry. Nedá se vytvořit v posledním domě řady (tedy vlevo nahoře a vpravo dole). Pokud první hráč vytvoří tuzdek v nějakém domě, protihráč nesmí vytvořit tuzdek v protilehlém domě. Takovýto tah provést lze, jen při něm nevznikne past.

Hra končí v okamžiku, kdy hráč, který je na tahu, má všechny své domy prázdné (a nemůže tedy provést žádný tah). Druhý hráč v tu chvíli získává všechny zbylé kameny

ve svých domech. Vítězí hráč, který nasbíral více kamenů.[Cach(2018)]

2 Rešerše řešení hry Kalaha

2.1 mancala.playdrift.com

Jedná se o webovou aplikaci hry Kalaha. Server umožňuje dva režimy hry. První je hra proti počítači, přičemž první tah je na hráči. Druhá varianta je možnost hrát s lidskými protivníky, kteří jsou online, ale až po registraci a přihlášení do systému. Server tedy umožňuje hrát hru s kýmkoliv na světě. Disponuje také chatem. Herní deska je přehledná, vyznačuje se jednoduchostí a přitom je efektní. Obsahuje kameny, které jsou uvnitř domů. Po kliknutí myši na dům, který chce hráč zahrát, hra simuluje rozdělování kamenů. Pro přehlednost je u každého domu a pokladnice ještě číselný údaj o tom, kolik kamenů je v domě respektive v pokladnici. [playdrift(2019)]

Při hře hráče proti stroji, je možnost nastavit tři úrovně obtížnosti. Jaké strategie jsou implementovány pro hru stroje, není nikde uvedeno. V rámci seznamování se s chováním zde implementovaného algoritmu označeného "*Hard Bot*", bylo zjištěno, že algoritmus se nechová racionálně. Nechrání své ohrožené domy a nepřesouvá jisté kameny do své pokladnice, místo toho provádí absolutně nelogické tahy. Tabulka 2.1 ukazuje statistiku deseti her. První tah měl znalý hráč a hrál proti algoritmu nazvaném "*Hard Bot*".

	číslo hry a skóre									
hráč	1	2	3	4	5	6	7	8	9	10
Znalý hráč	33	31	35	40	35	38	41	35	37	40
PC	15	17	13	8	13	10	7	13	11	8

Tab. 2.1 Tabulka skóre hry na mancala.playdrift.com

2.2 Superhry mancala

Hra je dostupná z *www.superhry.cz/games/240/*. Umožňuje hru proti stroji, nebo hru dvou hráčů na jednom počítači. V druhém případě se hráči střídají v ovládání hry pomocí signalizace aktivního hráče. Po ukončení hry, je ukázána malá statistika s údajem, kdo hru vyhrál, a pro každého hráče je uvedeno kolik kamenů mu zůstalo ve hře. Jedná se o zbylé kameny, které už hráč nemohl hrát po skončení hry, protože protihráč již nedisponoval kameny ve hře. Dále statistika obsahuje počet zachycených kamenů do pokladnice. Tyto dva počty tvoří výsledné skóre, které je také ve statistice zobrazeno. [Priebe(2019)]

Algoritmus se chová racionálně. Aktivně se brání ztrátě kamenů a naopak umí využít příležitost sebrat soupeři kameny.

	číslo hry a skóre			
hráč	1	2	3	4
Znalý hráč	33	40	34	30
PC	15	8	14	18

Tab. 2.2 Skóre hry na www.superhry.cz/games/240/

2.3 Arcadespot mancala

Hra je dostupná z www.arcadespot.com/game/mancala/. Umožňuje nastavit počáteční počet kamenů v domech od tří do šesti. V nabídce je výběr hry proti stroji, nebo hru dvou hráčů. Oproti ostatním online hrám je pozice hráče naproti desce. Po skončení hry je dána pouze informace o vítězi. [arcadespot(2019)]

Algoritmus pracuje určitě nějakým způsobem s náhodou, protože stejně rozehrané partie vyhodnocuje jinak.

	číslo hry a skóre				
hráč	1	2	3	4	5
Znalý hráč	31	38	36	38	30
PC	17	10	12	10	18

Tab. 2.3 Skóre hry na www.arcadespot.com/game/mancala/

2.4 Mathplayground mancala

Hra je dostupná z www.mathplayground.com/mancala.html. Nabízí pouze hrát proti stroji. Po skončení hry zobrazí informaci o tom zda hráč vyhrál, prohrál, nebo jestli nastala remíza a také zobrazí skóre. [LLC(2019)]

Jedná se zatím o nejlepší algoritmus. I pro zkušeného hráče není jednoduché volit optimální tahy.

	číslo hry a skóre				
hráč	1	2	3	4	5
Znalý hráč	23	29	30	30	19
PC	25	19	18	18	29

Tab. 2.4 Skóre hry na www.arcadespot.com/game/mancala/

2.5 AppOn mancala

Jde o mobilní aplikaci, která umožňuje výběr online hry, nebo offline. Ve verzi offline je pak možnost hrát jak proti stroji, tak hru dvou hráčů. Ve hře proti stroji je možné určit obtížnost hry. Na výběr jsou varianty lehká, střední a obtížná. V online hře je možnost přidělení náhodného hráče, nebo přítele. Hra umožňuje určit začínajícího hráče. [Ltd(2019a)]

2.6 MobileFusion Apps Ltd mancala

Mobilní aplikace hry mancala se změnou pravidla týkajícího se zajímání kamenů. Pokud byl dům, do kterého připadl poslední kámen prázdný, byl domem aktuálního hráče a protější dům byl prázdný, připadá kámen z tohoto domu do pokladnice i v případě, že v protějším domě žádné kameny nejsou. Hra umožňuje všechny tři režimy hry: hráče proti stroji, hráče proti hráči na jednom zařízení a hráče proti hráči propojených sítí. Oproti předešlým hrám disponuje zadáním jména hráče. [Ltd(2019b)]

2.7 Zhodnocení implementací uvedených her Mancala

Všechny zde uvedené implementace hry Kalaha jsou podobné. Některé umožňují hrát člověku proti stroji, jiné nabízejí hrát proti lidskému protihráči a to ve dvou variantách. První je hra na jednom zařízení, kdy se hráči střídají ve svých tazích, podle toho kdo je zrovna na řadě. Druhá varianta je hra po síti, která propojí dvě zařízení. Některé implementace umožňují změnit počet kamenů ve hře. V některých jsou v nabídce i jiná nastavení typu: vypnout vs. zapnout zvuk a pojmenovat hráče. Mnohem důležitější volba začínajícího hráče se vyskytuje pouze u menší části implementací. Na závěr hry se zobrazuje informace o tom, kdo hru vyhrál. Některé implementace nabízí i jednoduchou statistiku kdo, kolik a jakým způsobem získal kameny. Z výše uvedených her se jeví nejlépe varianta Mathplayground mancala a to z důvodu implementace dobrého herního algoritmu.

3 Teorie her

Teorie her je disciplína Aplikované matematiky [Wikipedie(2019)], analyzuje situace, v nichž se střetávají protichůdné zájmy jednotlivců nebo skupin, přičemž jednání jedné strany ovlivňuje, ale plně neurčuje, výsledek určité události, který závisí také na tom, jaká opatření podnikne strana druhá. Konfliktní situace se mohou vyskytovat nejen ve společenských hrách, podle nichž je tato vědní disciplína pojmenovaná, ale může se jednat o nejrůznější konfliktní situace charakteru ekonomického, vojenského, sportovního apod. [Říha(1972)]

Teorie her přímo neříká, co konkrétně jedinci v dané situaci opravdu udělají, a už vůbec negarantuje jistotu takové predikce. Teorie her umožňuje jejich situaci formálně popsat, analyzovat, pochopit a vyvodit nějaký závěr o situaci. V rámci studia her se zkoumá racionalita jedinců při rozhodování, jejich možnosti, preference a užitek dosažený ve hře. Z informatického hlediska je teorie her varianta umělé inteligence nebo všeobecně metod modelování. Pracuje s modely a mnohdy tyto modely počítačově zpracovává do formy simulačních modelů. [Hrubý(2010a)]

3.1 Hra

Pod pojmem hra se rozumí strategická interaktivní situace zahrnující alespoň dva hráče, kde oba sledují své vlastní cíle. Hráči jsou nuceni v těchto situacích provádět rozhodnutí, která vedou buď k výhře nebo prohře.

Hry jsou popsány souborem pravidel, mají určitou formální strukturu a určují chování hráčů. Pravidla zaručují, že hra je konečnou posloupností tahů, vykonané v určitém pořadí, a popisují možnosti každého tahu. [Blackwell(1964)]

3.2 Rozdělení her

Hry jsou rozděleny podle typu, podle něhož se pak hledá jejich nejlepší řešení. Pokud bude známa charakteristika hry a na základě ní určen typ, lze pak najít optimální strategii hry. Hry jsou rozděleny do tří hlavních okruhů:

- Strategické hry – jedná se o takové hry, ve kterých si hráč na začátku hry zvolí strategii a tu už v průběhu hry nemění [Hrubý(2010b)]. Příklad takové oblíbené hry může být Kámen-Nůžky-Papír.
- Poziční hry s úplnou nebo neúplnou informací – jedná se o hry obsahující sekvenční rozhodování. Hráči své rozhodování přizpůsobují předešlým akcím ostatních hráčů. Příkladem je hra Piškvorky.

- Kooperativní hry – tato skupina her, jak už název napovídá, je charakterizovaná spoluprací jednotlivých hráčů. Ti se mohou na svých tazích domlouvat a tím zvyšovat svůj zisk. Nové místo v těchto hrách nachází vyjednávání a přerozdělování zisku v rámci vytvořených koalic.

Uvedenému dělení odpovídá také dělení podle matematického zápisu hry:

- Hry v normálním tvaru – jsou popsány hráči, možnými strategiemi a výplatními funkcemi.
- Hry v explicitním tvaru – průběh hry lze znázornit pomocí stromu, zachycující sekvenční rozhodování
- Hry ve tvaru charakteristické funkce – charakteristická funkce označuje celou hru a přiřazuje každé koalici možnou výhru. [FORCH(2014)]

3.3 Užitek

Pokud důsledek nějakého tahu je označován jako zisk, pak musí být zkoumáno, jaký zisk přinese užitek. Rozdíl mezi ziskem a užitkem ilustruje následující příklad. Zisk 1000 CZK je pro každého člověka stejný, je roven 1000 CZK, ale užitek může být odlišný. Chudému člověku přinese větší užitek než miliardářovi.

Ke zkoumání dvou odlišných důsledků x a y mohou být použity dva přístupy zkoumání důsledků:

- Ordinalistický přístup – Slouží jen k porovnání alternativ. Výsledek je poznání zda x je lepší než y, nebo naopak, nebo jsou stejné.
- Kardinalistický přístup – kromě prostého porovnání x a y určí o kolik je x lepší než y. Umí tedy určit kvantifikovaně rozdíl $|x - y|$.

Při složitějším rozhodování se mohou protichůdné cíle uspořádat podle funkce užitku.

3.4 Rozhodování

Rozhodnutí je zdůvoditelná volba jedné z možných voleb (akcí, tahů, strategií).

Akce jsou buď racionální, nebo náhodné [Hykšová(2019)]. Pokud hráč volí svoji akci náhodně, je takováto hra v teorii her nazývána *"hra proti přírodě"*. Racionální hráči dochází ke svému rozhodnutí nějakou matematicky podloženou úvahou. Racionální chování je takové, které si vybírá tah po důkladném zvážení veškerých důsledků. Racionální jedinec maximalizuje svůj zisk. Zisk je číselné vyjádření, které čím je vyšší, tím je lepší.

Úvaha vede k rozhodnutí a každé rozhodnutí má nějaký důsledek. Důsledek je v teorii her zkoumaný pojem a je vyjádřen formou nějakého kvantifikovaného užitku nebo také zisku. V teorii her se často pojem užitek a zisk zaměňují. Pouze v některých případech se odlišuje objektivní zisk od vnímání užitku ze zisku.

Rozhodování se dělí do tří kategorií, podle toho, jaké informace jsou při tomto procesu známé.

- Rozhodování za jistoty – je situace, kdy hráč, který provádí rozhodnutí o volbě strategie, má možnosti z prostoru strategií $S = \{s_1, s_2, ..., s_k\}$ a je schopen definovat množinu veškerých důsledků svých rozhodnutí. Tím získá množinu $X = \{x_1, x_2, ..., x_n\}$, která představuje zisk. Hráč je poté schopen jednoznačně každé strategii $s \in S$ přiřadit právě jeden důsledek $x \in X$. Matematicky je tento fakt modelován funkcí užitku, která přiřazuje akcím $s \in S$ právě jeden důsledek z množiny X

$$u : S \to X \tag{3.1}$$

- Rozhodování za rizika – v tomto případě nemá hráč jistotu, k jakým výsledkům budou jednotlivá rozhodnutí směřovat, ale pouze pravděpodobnost, s jakou důsledek nastane. Funkce užitku u přiřazuje každé strategii $s \in S$ nějaké rozložení pravděpodobnosti P_x na množině důsledků X.

$$u : S \to P_x \tag{3.2}$$

- Rozhodování za neurčitosti – rozhodování probíhá za neurčitosti, jestliže výsledková funkce u přiřazuje každé strategii $s \in S$ nějakou množinu výsledků $X_s \subseteq X$, tj.

$$u : S \to X_s \tag{3.3}$$

3.5 Strategické hry

Definice strategické hry je (2N+1)-tice

$$\Gamma = (Q; S_1, S_2, ..., S_N; U_1, U_2, ..., U_N) \tag{3.4}$$

kde:

- $Q = \{1, ..., N\}$ pro $N \in Q$ a $N \geq 2$ je množina všech hráčů
- $S_i, i \in Q$ jsou množiny strategií hráčů $i \in Q$
- $U_i : S_1 \times S_2 \times \cdots \times S_N \rightarrow U$ jsou výplatní funkce. Jedná se o zobrazení z množiny S do reálných čísel, tedy $U_i : S \rightarrow \mathbb{R}$

3.5.1 Strategie

Pro kategorii strategických her platí, že strategie si jednotliví hráč volí nezávisle na ostatních hráčích. Zároveň však platí, že výplatní funkce je funkcí více proměnných, kde proměnné jsou zvolené strategie hráčů. I přesto, že hráč neví, jak se ostatní hráči rozhodnou, lze nalézt strategie, při jejichž zahrání si zajistí vyšší výhru než při zahrání strategií jiných.

3.5.2 Maticové hry

Jedná se o konečné hry dvou hráčů s nulovým součtem. Tento typ her může být reprezentován maticí hry. Řádky matice představují strategie prvního hráče a sloupce strategie druhého hráče. Prvek matice udává zisk prvního hráče a ztrátu druhého hráče. Ve hrách s nulovým součtem prvek matice udává kolik druhý hráč zaplatí prvnímu hráči, zisk prvního hráče je x a zisk druhého hráče je $-x$ [Maňas(1969)]. V těchto hrách lze určit maximální výši minimálního zisku prvního hráče, který si je schopen zajistit zisk bez ohledu na to, jak hraje druhý hráč. Také je možné určit minimální výši maximální ztráty druhého hráče, kterou si je schopen zajistit bez ohledu na to, jak hraje první hráč. Pokud se tyto dvě hodnoty rovnají, znamená to, že ve hře existuje tzv. sedlový bod. V takovém případě ani jeden z hráčů nemůže dosáhnout lepšího výsledku. Z toho vyplývá, že příslušné strategie jsou tím nejlepším, co mohou hráči hrát. [Sawa(2015)]

Řešení maticových her Pro získání optimálního řešení maticových her, je potřeba vytvořit matici hry. Musí být vypočteny prvky matic podle výplatní funkce, jak bylo uvedeno výše. V maticových hrách je hodnota výplatní funkce druhého hráče záporná hodnota prvního hráče. Stačí tedy výpočet hodnot jen s pomocí výplatní funkce jednoho hráče. Poté je hledán sedlový bod a_{ij}. Ten odpovídá největší hodnotě ve sloupci a zároveň nejmenší hodnotě ve svém řádku. Důvod proč se hledá maximum minima je následující. Oba hráči chtějí maximalizovat svůj zisk, výplata druhého hráče odpovídá $-a_{ij}$. Pokud je vybráno minimum v řádku, pro druhého hráče je to maximum. Nalezneme-li sedlový bod s hodnotou a_{ij}, potom i-tá strategie prvního hráče a j-tá strategie druhého hráče jsou optimální rovnovážné strategie a tato hodnota a_{ij} je nazývána cenou hry. Takto nalezené řešení je nazýváno Nashova rovnováha v ryzích strategiích.

Příkladem je matice hry pro hru Kámen-Nůžky-Papír, ta však sedlový bod nemá:

$$\begin{bmatrix} 0 & +1 & -1 \\ -1 & 0 & +1 \\ +1 & -1 & 0 \end{bmatrix} \tag{3.5}$$

Příkladem hry se sedlovým bodem může být hra, reprezentována následující maticí:

$$\begin{bmatrix} 5 & 6 & [(4)] & 5 \\ [-2] & 5 & 3 & (7) \\ (8) & (7) & [-2] & 6 \end{bmatrix} \tag{3.6}$$

V matici hry 3.6 jsou poznačeny v kulatých závorkách maxima ve sloupcích a hranatými závorkami minima v řádcích. Pokud je prvek jak v kulatých tak hranatých závorkách, jedná se o sedlový bod.

3.6 Hry v rozšířené formě

Hry v rozšířené formě modelují střet hráčů s posloupností rozhodování. Také jsou nazývány pozičními hrami. Je to z důvodu, že hráči reagují na aktuální stav hry. Hráči se postupně střídají v tazích, které musí volit tak, aby maximalizovali svůj zisk na úkor protihráčů. Hry v rozšířené formě se dále dělí na hry s úplnou informací a hry s neúplnou informací. Při hrách s úplnou informací je každý hráč obeznámen s předešlými tahy. Typickým příkladem takové hry jsou šachy.

Hra v rozšířené formě s úplnou informací je čtveřice $(N, H, P, (\succsim i)_{i \in N})$ kde:

- Q je množina všech hráčů
- H je množina posloupností, které splňují následující podmínky:
  1. Posloupnost délky 0 patří do H
  2. Pokud $(a^k)_{k=1,\dots,K} \in H$ a existuje nějaké $L < K$, pak i $(a^k)_{k=1,\dots,L} \in H$, kde $K \in \mathbb{N} \cup \{\infty\}$
  3. Jestliže posloupnost $(a^k)_{k=1}^{\infty}$ splňuje $(a^k)_{k=1,\dots,L} \in H, \forall L \in \mathbb{N}$ pak i $(a^k)_{k=1}^{\infty} \in H$

  Každý prvek $h \in H$ se nazývá *historie* a každá historie je posloupnost akcí zvolených hráči. Množina ukončených historií je označena Z.
- Funkce P, která každé neukončené historii přiřadí prvek z množiny všech hráčů Q. Tedy $P : H \setminus Z \to Q$. Tato funkce určí hráče, který volí následující akci po historii $h \in H$.

- Pro každého hráče $i \in Q$ porovnáme užitek na množině ukončených historií Z pomocí preferenčního uspořádání $\precsim i$.

Stejně jako pro strategické hry můžeme preferenční uspořádání na množině ukončených historií nahradit výherní funkcí u_i pro všechny hráče $i \in Q$.

Je-li h historie délky k, pak jako (h, a) je označena historie délky $k+1$, ve které po h následuje akce a. Po každé neukončené historii h vybere hráč $P(h)$ akci z množiny $A(h) = \{a : (h, a) \in H\}$. Počáteční historií je tzv. prázdná historie, poté vybere hráč akci z množiny A, ta je označena jako a^1, informace o této akci je uložena v historii h^1. Následně hráč $P(h^1)$ vybere akci z $A(h^1)$ a tak dále. Pokud po nějaké historii nenásleduje žádná volba, je označena za ukončenou.

Jak už z definice vyplývá, hry v explicitním tvaru mohou být popsány stromem. Uzly představují jednotlivé stavy hry a hrany možné akce (tahy) v těchto stavech. Hra obsahuje právě jeden uzel, do kterého nevstupuje žádná hrana. Takový uzel se nazývá kořen stromu. Uzly, ze kterých žádné hrany nevychází, jsou koncové uzly nebo také listy. Strom už z podstaty své struktury zaznamenává historii hry. Ta je reprezentována cestou k aktuálnímu uzlu od kořene stromu. V každém nelistovém uzlu se rozhoduje hráč, jakou variantu akce využije. Na základě vybrané akce se hra dostane do dalšího uzlu, kde akci vybírá další hráč. Tak probíhá výběr stále dokola, až se hra dostane do koncového uzlu.

Jak už bylo zmíněno, ve strategických hrách pojmy akce a strategie splývají. Ve hrách v rozšířené formě je rozlišujeme. Strategie pro tyto hry je plán akcí, který pro každý stav hry jasně udává akci, která má být provedena.

Strategie hráče $i \in Q$ ve hře v rozšířené formě $(Q, H, P, (\precsim i)_{i \in N})$ je funkce, která každé neukončené historii $h \in H \setminus Z$, pro kterou platí $P(h) = i$, přiřadí akci z $A(h)$, kde $A(h) = \{a : (h, a) \in H\}$. Strategii i-tého hráče označíme jako s_i, množinu všech jeho strategií pak S_i.

Strategie určuje hráči i volbu akce pro každou historii, kdy $P(h) = i$. Strategie s_i tedy hráči i diktuje jakou akci zvolit i po takové historii, která nenastane, řídí-li se hráč i touto strategií. [FORCH(2014)]

4 Technologie pro tvorbu aplikací

4.1 Programovací jazyk C++

Jazyk C++ vznikl v roce 1979 v Bellových laboratořích. Jeho autorem je Bjarne Stroustrup. Vychází z jazyka C, který byl velmi oblíbený a používaný strukturovaný jazyk. S postupem času však již strukturované paradigma nedostačovalo a na oblíbeném jazyce C byla vybudována nadstavba objektového paradigmatu. Vznikl tak multiparadigmatický jazyk C++, který zachoval původní strukturované paradigma, ale přidal i objektový přístup. Od roku 1998 je standardem ISO stejně jako jazyk C. Poslední aktualizace standardu je z roku 2017. I přesto, že C++ podporuje zpětnou kompatibilitu s jazykem C, existují výjimky. Ne všechny programy napsané v jazyce C jsou přeložitelné překladačem pro jazyk C++. Jazyk C++ patří mezi nejpoužívanější programovací jazyky na světě.

Objektové paradigma přináší do programování modelování objektů. Objekty mohou být jakékoliv předměty z reálného světa, se kterými se výpočetně v programu pracuje, mohou to být prvky grafického rozhraní aj. Důležité je, že k objetu se přistupuje jako k samostatnému prvku a komunikuje se s ním zasíláním zpráv. To realizují metody daného objektu (v C++ třídy). Objekt je popsán svými proměnnými, které by dle principů zapouzdření měli být pro okolí objektů skryté. Pracovat s nimi by měly pouze metody dané třídy. Zachováním tohoto pravidla se docílí zapouzdřenosti objektu a programátor se vyvaruje možné chybě někde v programu, kde by se s danou proměnou pracovalo přímo. Další výhoda objektového přístupu je znovupoužitelnost kódu. Tím, že jsou objekty samostatné jednotky, které implementují i obsluhu, je velmi jednoduché použít stejné objekty znovu v jiných aplikacích, nebo je jednoduše pozměnit a upravit. Další vlastnost objektového paradigmatu je dědičnost. I tuto C++ implementuje. Jedná se o situaci, kdy už z existujících objektů se vytvoří nové, přičemž zdědí vše od svého rodiče, ale navíc je objekt doplněn o další proměnné a metody. To umožňuje urychlení psaní kódu. Programátor nemusí implementovat to co už je napsáno, jen objekty rozšíří o novou funkcionalitu. C++ také implementuje polymorfismus. Ten je realizován tak, že potomci objektu mohou mít přepsanou metodu, která se bude pro rodiče i potomka chovat odlišně.

Jazyk C++ také obsahuje šablonování. Jedná se o silný nástroj, který zvyšuje univerzálnost kódu. Šablona je napsán obecně a až při samotném použití se specifikuje datový typ, pod kterým bude daná struktura fungovat. Souvisí to s tím, že C++ je typový jazyk, který se snaží datové typy proměnných důsledně kontrolovat a neumožňuje laxně přistupovat k přiřazování hodnot z různých datových typů. Zvyšuje se tím odolnost k chybám. Šablonové programování je požito i ve standardních knihovnách jazyku C++, kdy například datová struktura seznam je implementována šablonou. Progra-

mátor pak jen specifikuje datový typ nad kterým má být šablona použita a vytvoří se seznam libovolného datového typu.

Jazyk C++ je značně univerzální. Může být využit jak pro velmi nízkoúrovňové programování, tak pro nejsložitější aplikace. Pokud se programátoři drží pravidel objektového programování, dělá to z tohoto jazyka velmi mocný a bezpečný nástroj. Tím, že je jazyk standardizovaný, je zajištěna univerzálnost překladačů tohoto jazyka a to zvyšuje jeho použitelnost. [Liberty(2007)]

4.2 Grafické uživatelské prostředí GUI

Stejně jako programovací jazyky, tak i uživatelské rozhraní prošlo vývojem. Doba, ve které uživatelé komunikovali se stojem formou přepínačů a propojovacími dráty je nenávratně pryč. Po této velmi nízkoúrovňové komunikaci přišla varianta textová v podobě zadávání příkazů přes klávesnici a zobrazování dotazů respektive výsledků na monitoru. Dnes však pod pojmem uživatelsky přívětivé prostředí chápeme grafické, plně interaktivní rozhraní, které pro komunikaci mezi strojem a uživatelem používá srozumitelné grafické prostředky. První koncept grafického uživatelského rozhraní byl vytvořen ve firmě Xerox při práci na projektu Xerox Star. Tento projekt moc úspěšný nebyl, ale myšlenku grafického prostředí převzala konkurenční firma Apple a poté i Microsoft.

U aplikací s grafickým uživatelským prostředím se postupně přešlo od procedurálního přístupu k objektově-orientovanému, a začaly se používat programovací jazyky s podporou objektově-orientovaného programovaní v čele s jazykem C++.

GUI aplikace je tvořena množinou objektů, které zajišťují určitou činnost, tu zapouzdřují a vykonávají po přijetí zpráv. Zprávy jsou používány pro komunikaci s objekty, jak už bylo zmíněno v podkapitole věnované objektovému programování. Aby programátor nemusel implementovat grafické rozhraní sám, existují různé knihovny, API, toolkity a frameworky. Jednotlivé pojmy se často pletou, proto zde budou vysvětleny.

4.2.1 Knihovna

Knihovna je programový modul zapouzdřující určitou funkčnost. Jedná se o sbírku předprogramovaných funkcí, tříd, zdrojů a dalších programových prostředků, které pokrývají řešení často se vyskytujících požadavků na implementaci. Tím, že knihovny obsahují samotnou implementaci daných objektů nebo funkcí, je programátorovi ulehčena práce. Nemusí implementovat to, co už někdo vytvořil, jen použije univerzální implementaci pro svůj program nebo aplikaci. Velká výhoda knihoven je ta, že jsou psány univerzálně, tak aby jejich použití bylo co nejširší. Další jejich velká výhoda je, že jsou odladěné a neměli by obsahovat chyby. Alespoň pravděpodobnost, že chybu

obsahují je výrazně nižší, než při psaní vlastního kódu. Příkladem knihovny může být standardní C++ knihovna implementující seznam (list).

4.2.2 API

API neboli *Application Programming Interface* (aplikační programové rozhraní) úzce souvisí s knihovnami. API stanovuje jak bude probíhat komunikace mezi zdrojovým kódem programátora a knihovnami. Zároveň obsahuje kolekci veřejně dostupných funkcí, objektů a dalších datových struktur dané knihovny.

4.2.3 Toolkit a Framework

Toolkit znamená v překladu sada nástrojů, ale ve softwarovém inženýrství je takto označena množina základních stavebních prvků, které tvoří grafická uživatelská rozhraní (anglicky také *widget toolkit*). Framework je sada knihovních modulů s dalšími podpůrnými nástroji a programy usnadňujícími a urychlujícími vývoj aplikace. Na rozdíl od prosté knihovny, která je pouze sadou předprogramovaných funkcí, datových struktur, konstant a dalších programových komponent, framework nabízí navíc třeba i různé návrhové nástroje a možnosti rozšíření funkčnosti se zpětnou kompatibilitou. Ve zkratce se dá říci, že se jedná o komplexnější toolkit.

4.2.4 SDK

SDK je anglická zkratka *Software Development Kit* neboli sada nástrojů pro vývoj softwaru. SDK je kompletní vývojový balíček pro vývoj s danou technologií. Jedná se o komplexní sadu knihoven, vývojových, sestavovacích a lokalizačních nástrojů, příkladů a tutoriálů.

4.2.5 Možnosti tvorby GUI aplikací

Jak už z výše uvedeného vyplývá, vyplatí se pro ulehčení práce využít existující framework nebo přímo SDK, které je komplexně přizpůsoben vývoji software. Dále budou zmíněny nejvyužívanější technologie.

Microsoft Visual Studio patří jednoznačně do skupiny SDK. Jedná se o nativní vývojový nástroj pro operační systémy Microsoft Windows. Toto vývojové prostředí podporuje různé jazyky, např. C++, C#, F#, Visual Basic a další. Visual Studio je dostupné v mnoha komerčních licencích dle potřeb a zaměření jednotlivých vývojářů.

Vývojové prostředí pracuje s kolekcí knihoven *Windows API*. Jedná se o programové rozhraní tohoto operačního systému, nabízející programátorovi jeho funkce (zdokumentované i nezdokumentované). Windows API je implementováno jako knihovna C, která

nabízí volatelné funkce a datové struktury poskytované tímto operačním systémem a je dostupná jako součást Microsoft Windows SDK. Toto API zajišťuje základní i pokročilé funkce operačního systému, jako jsou grafické, multimediální, síťové a další služby. Windows API je mocným a robustním nástrojem, ale kvůli tomu, že je implementovaná v jazyce C, se hodí spíše pro nízkoúrovňové programy a utility. Chybějící objektové zapouzdření řeší knihovna MCF (*Microsoft Foundation Classes*), která je vhodnější pro začlenění nativních funkcí tohoto systému do objektově navržených programů.

Tvorba GUI aplikací na platformě GNU Linux/UNIX Základní knihovna pro tvorbu grafických aplikací v unixu je *Xlib*. Jedná se o nízkoúrovňového klienta protokolu X Window napsané v jazyce C, který slouží ke komunikaci s X Servrem zajišťujícím vlastní vykreslování. Knihovna Xlib je ovšem příliš nízkoúrovňová, nepodporuje pokročilejší prvky grafického rozhraní např. Widgety apod. Z tohoto důvodu vznikly nadstavby této knihovny, které ji doplňují a rozšiřují. Jako příklad může být uveden toolkit *X Toolkit Intrinsics*, který podporuje objektový přístup v C++ a tvorbu vlastních Widgetů, předpřipravené však nenabízí. Vlastní Widgety nabízí knihovny *X window Athena Widget set* a *(Open)Motif* [Chroboczek(2013)]

4.3 Qt

Qt je komplexní SDK, které nabízí potřebné knihovny, vývojové prostředí a podporu. Snaží se o maximální jednoduchost a účelnost vývoje aplikací. To nasvědčuje samotný fakt, že k instalaci celého SDK stačí jeden instalační balíček. SDK obsahuje všechny nástroje, které jsou potřeba od konceptu aplikace k nasazení aplikace na více platformách i mobilních zařízeních. Aplikace vytvořené v Qt mohou běžet jak na desktopech s operačními systémy Microsoft Windows, Mac OS X a Linux, tak i na mobilních zařízeních se systémy Android, Symbian, Maemo a MeeGo Harmattan. Z tohoto je jasně patrně, že Qt je opravdu multiplatformní a aplikace takto napsaná poběží bez zásahů nebo jen s minimálními úpravami na absolutně odlišných strojích, rozdílných jak softwarovým tak i hardwarovým vybavením. Z důvodu podporování různých platforem je součástí Qt i simulátor, ve kterém se jednotlivé aplikace mohou jednoduše otestovat. Qt je dobře a přehledně zdokumentované a dostupné buď přímo v Qt Creatoru nebo na webových stránkách.

Qt obsahuje tyto základní moduly:

- Qt Core – Negrafické třídy Qt Core používané jinými moduly
- Qt GUI – Základní třídy pro komponenty grafického uživatelského rozhraní (GUI), zahrnuje OpenGL

- Qt Multimedia – Třídy pro audio, video, rádio a funkčnost kamer
- Qt multimediální Widgety – Třídy založené pro implementaci multimediální funkce
- Qt Network – Třídy umožňují snadnější a přenosnější programování v síti
- Qt QML – Třídy pro jazyky QML a JavaScript
- Qt Quick – Deklarativní rámec pro vytváření vysoce dynamických aplikací s uživatelským rozhraním
- Qt Quick Controls – Poskytuje typy QML pro vytváření výkonných uživatelských rozhraní pro stolní počítače, vestavěné a mobilní zařízení. Tyto typy využívají jednoduchou architekturu a jsou velmi efektivní
- Qt Quick Dialogs – Typy pro vytváření a interakci se systémovými dialogy z aplikace Qt Quick.
- Qt Quick Layouts – jsou grafická rozvržení, která slouží k uspořádání položek založených na Qt Quick 2 v uživatelském rozhraní
- Qt Quick Test – Jednotkový testovací rámec pro aplikace QML, kde jsou testovací případy psány jako funkce JavaScriptu
- Qt SQL – Třídy pro integraci databáze pomocí SQL
- Qt Test – Třídy pro testování jednotek Qt aplikací a knihoven
- Qt Widgets – Třídy pro rozšíření Qt GUI s widgety C++

4.3.1 Licence

Qt je k dispozici v různých licenčních variantách, navržených tak, aby vyhovovaly potřebám různých uživatelů. Qt licencovaný na základě komerčních licencí je vhodný pro vývoj proprietárního softwaru, kde je nežádoucí sdílet zdrojový kód s třetími stranami nebo jinak nemůže splňovat podmínky GNU LGPL verze 3. Qt licencovaný pod licencí GNU Lesser General Public License (LGPL) verze 3 je vhodný pro vývoj aplikací Qt za předpokladu, že jsou splněny podmínky GNU LGPL verze 3. Qt obsahuje také kód třetích stran, který je licencován podle specifických open-source licencí od původních autorů.

4.3.2 Signál a slot

Signály a sloty se používají pro komunikaci mezi objekty. Mechanismus signálů a slotů je hlavním znakem a součástí Qt, která se nejvíce liší od vlastností poskytovaných jinými frameworky. Signály a sloty jsou možné pomocí meta-objektového systému Qt.

Při programování GUI je často potřeba aby na změnu jednoho widgetu reagoval druhý widget. Obecněji, je požadována vlastnost, aby objekty jakéhokoli druhu mohly komunikovat mezi sebou. Pokud například uživatel klikne na tlačítko Zavřít, pravděpodobně bude potřeba zavolat funkci pro zavření okna.

Ostatní toolkity dosahují tohoto druhu komunikace pomocí zpětných volání. Zpětné volání je ukazatel na funkci, takže v případě, že je potřeba aby funkce zpracování informovala o nějaké události, předá se ukazatel na jinou funkci – funkci zpracování (vyvolá zpětné volání). Zpětná volání ovšem mohou být neintuitivní a mohou trpět problémy při zajišťování správnosti typů argumentů funkce zpětného volání. Zatímco ostatní frameworky tuto techniku používají, Qt se rozhodlo jít vlastní cestou a vytvořilo koncept signálů a slotů.

Signál je vydáván, když nastane konkrétní událost. Slot je funkce, která je volána v reakci na konkrétní signál. Qt widgety mají mnoho předdefinovaných signálů a slotů. Je ovšem běžnou praxí, že widgety podtříd si přidávají své vlastní signály a sloty, takže mohou být zpracovány signály tak, jak požaduje programátor.

Signál je vysílán objektem, když se jeho vnitřní stav nějakým způsobem změnil. Tato situace může být zajímavá nebo důležitá pro klienta nebo vlastníka objektu. Signály jsou funkce veřejného přístupu a mohou být vysílány odkudkoliv, ale doporučuje se je vysílat pouze z třídy nebo podtřídy, která definuje signál.

Když je signál vysílán, sloty připojené k němu jsou obvykle prováděny okamžitě, podobně jako volání funkce. Mechanismus signálů a slotů je zcela nezávislý na jakékoliv smyčce událostí GUI. Po vrácení všech slotů dojde k provedení kódu následovaného příkazem *emit*. Situace je poněkud odlišná při použití spojení ve frontě, v takovém případě bude kód následující: za klíčovým slovem *emit* bude kód okamžitě pokračovat a sloty budou provedeny později. Pokud je k jednomu signálu připojeno několik slotů, sloty budou provedeny jeden po druhém v pořadí, v jakém byly připojeny. Signály jsou automaticky generovány modulem *moc* a nesmí být implementovány v souboru *.cpp*. Nikdy nemohou mít návratové typy.

Slot je připojen k signálu a je volán v případě vyslání signálu. Sloty jsou obyčejné C++ funkce a mohou být volány i normálně, jejich jedinou speciální vlastností je, že k nim mohou být připojeny signály. Vzhledem k tomu, že sloty jsou běžné členské funkce,

řídí se při volání přímo pravidly C++. Jako sloty však mohou být vyvolány přes spojení signál-slot libovolnou komponentou, bez ohledu na její úroveň přístupu. To znamená, že signál vysílaný z instance libovolné třídy může způsobit vyvolání privátního slotu v instanci nesouvisející třídy. Mohou být také definovány virtuální sloty, které jsou v praxi velmi užitečné.

Ve srovnání s zpětnými voláními jsou signály a sloty z důvodu zvýšené flexibility o něco pomalejší, ale rozdíl v reálných aplikacích je zanedbatelný. Obecně platí, že vyvolání signálu, který je připojen ke slotu, je přibližně desetkrát pomalejší než u přímého volání. Tyto režijní náklady jsou potřebné k lokalizaci objektu spojení, k bezpečné iteraci přes všechna připojení a ke generování jakéhokoliv parametru obecným způsobem. Desetinásobné zpomalení se může jevit jako velké, ale ve srovnání s prací s řetězci, vektory, seznamy nebo vytvářením nebo mazáním objektů, je to zanedbatelná režie. Jednoduchost a flexibilita mechanismu signálů a slotů stojí za režii, které si běžný uživatel ani nevšimne.

4.3.3 Qt Designer

Qt Designer je nástroj Qt pro navrhování a vytváření grafických uživatelských rozhraní (GUI) s Qt Widgets. V tomto designeru může programátor vytvořit a přizpůsobit okna nebo dialogy podle vzhledu, jak uzná za vhodné. Pracuje totiž s náhledem a fakticky posouváním a úpravami tvoří výsledné rozvržení, zarovnání a vzhled jednotlivých grafických komponent podle nejrůznějších stylů.

Widgety a formuláře vytvořené pomocí Qt Designer se bezproblémově integrují s naprogramovaným kódem, pomocí mechanismu Qt signálů a slotů, takže programátor může snadno přiřadit chování grafickým prvkům. Všechny vlastnosti nastavené v Qt Designer lze v rámci kódu dynamicky měnit. Mimo to, je možné, používat vlastní widgety a komponenty.

Při použití Qt Quick pro návrh uživatelského rozhraní se nepoužívají widgety. Jedná se o mnohem jednodušší způsob, jak psát mnoho druhů aplikací. Umožňuje zcela přizpůsobitelný vzhled. Prvky reagující na dotek a plynulé animované přechody, podporované silou akcelerace grafiky OpenGL. [Qt(2019)]

4.3.4 Widget

Třída QWidget je základní třídou všech objektů uživatelského rozhraní, přijímá události myši, klávesnice a okenního systému. Reprezentuje aplikaci pomoci barev na obrazovce. Každý widget je obdélníkový a je oříznut jeho rodičem a widgety před ním.

Widget, který není vložen do nadřazeného widgetu, se nazývá okno. Okna jsou obvykle opatřena rámečkem a záhlavím, ale je také možné vytvářet okna bez použití

vhodných příznaků okna. V Qt jsou nejběžnější typy oken *QMainWindow* a různé podtřídy *QDialog*.

Každý konstruktor widgetu akceptuje jeden nebo dva standardní argumenty:

QWidget * parent = 0 je rodič nového widgetu. Pokud je 0, nový widget bude okno. Pokud ne, bude podřízený rodič a bude až na výjimky omezen geometrií rodičů. Qt::WindowFlags f = 0 nastaví příznaky okna. Výchozí nastavení je vhodné pro téměř všechny widgety, ale například pro okno bez systému rámců, se musí použít speciálních příznaků. QWidget má mnoho členských funkcí, pomocí kterých se dá měnit funkčnost widgetu. Existuje mnoho podtříd, které poskytují skutečné funkce, jako je QLabel, QPushButton, QListWidget a QTabWidget.

Widgety nejvyšší úrovně a dítěte Widget bez nadřazeného widgetu je vždy nezávislé okno. Pro tyto widgety nastaví setWindowTitle() a setWindowIcon() záhlaví a ikonu.

Pomocí QWidgetu a jeho podtříd lze vytvořit libovolné GUI, plnící představu grafika i programátora.

II. ANALYTICKÁ ČÁST

5 Analýza hry Kalaha

Z výše uvedených pravidel hry Kalahy je možné pochopit, jak se hra hraje. Ovšem znát pravidla je jen nejzákladnější předpoklad k tomu, aby člověk mohl hru hrát. Pokud chce vyhrát, musí volit takovou strategii, aby jednotlivé tahy vedly k výhře. K tomuto cíli, ale vede složitá cesta. Dva protihráči zde stojí proti sobě a každý chce porazit svojí hrou toho druhého.

5.1 Získávání kamenů do pokladnice

5.1.1 Obyčejný tah získání kamene

Cílem hry je získat co nejvíce herních kamenů, nejméně však 25. Aby toho hráč dosáhl, musí volit tahy tak, aby získával kameny, které jsou ve hře. Aby hráč získal kámen do své pokladnice, musí vybrat pro tah takový dům, ze kterého při rozdělování padne alespoň jeden kámen do pokladnice. Hráč vezme kameny ze svého domu a postupně přerozděluje kameny proti směru hodinových ručiček, prvně do ostatních domů a pokud již žádný dům v tomto směru není, tak bude kámen přiřazen do pokladnice. Pokud zůstávají další kameny, dále se rozdělí proti směru hodinovým ručičkám do soupeřových domů. V případě, že se dostane hráč až k pokladnici protihráče, tu přeskočí a pokračuje v rozdělování zbylých kamenů do vlastních domů. Z takového tahu může získat ve většině případů jeden kámen. Pokud by v domě bylo více jak 14 kamenů tak i dva. Platí to pro případ, kdy z se kameny přiřadí do všech domů a dvakrát se přiřadí do pokladnice.

5.1.2 Získání tahu navíc

Další pravidlo Kalahy říká, že v situaci, kdy připadne poslední kámen do pokladnice, získá hráč další tah a to dokonce opakovaně. Situace k takovýmto tahům bývají v zásadě výhodné a to z několika důvodů. Za prvé, pokud bude proveden takovýto tah, bude v pokladnici o jeden kámen víc a hráč získá tah navíc, ve kterém může opět vylepšit skóre. Další výhodou je, že nebude změněno rozvržení kamenů v soupeřových domech. S přihlédnutím k tomuto faktu, je výhodná situace, kdy jsou v domech nalevo od pokladnice vzestupně seřazeny počty kamenů od 1 až teoreticky po 6, v takovém případě by šlo do pokladnice získat 18 kamenů z 21, aniž by se soupeř dostal ke svému tahu. Aby se tak stalo, hráč by musel udělat tyto kroky:

- Hráč zahraje 6. dům, tj. vezme kámen z posledního domu před pokladnicí a vloží kámen do pokladnice, tímto tahem získává nový tah.
- Hráč zahraje 5. dům, tj. vezme oba dva kameny z předposledního domu před pokladnicí a vloží jeden kámen do posledního domu a jeden kámen do pokladnice, tím získá další tah zdarma.

- Hráč zahraje 6. dům, tj. vezme kámen z posledního domu před pokladnicí a vloží kámen do pokladnice, tímto tahem získává nový tah.
- Hráč zahraje 4. dům, tj. vezme všechny tři kameny ze čtvrtého domu a vloží jeden kámen do předposledního domu, jeden kámen do posledního a jeden kámen do pokladnice, tím získá další tah zdarma.
- Hráč zahraje 6. dům, tj. vezme kámen z posledního domu před pokladnicí a vloží kámen do pokladnice, tímto tahem získává nový tah.
- Hráč zahraje 3. dům, tj. vezme všechny čtyři kameny ze třetího domu a vloží jeden kámen do čtvrtého domu, jeden kámen do pátého domu, jeden kámen do posledního domu a jeden kámen do pokladnice, tím získá další tah zdarma.
- Obdobně může hráč pokračovat až do situace, kdy mu zůstanou pouze tři kameny ve třech domech, které už nebude moct dostat do pokladnice tak, aby na tahu nebyl protihráč.

Jak lze vidět, tento postup je velmi dobře algoritmizovaný. V praxi takováto situace nenastane, ale může se vyskytnou alespoň částečná. Aby byl výtěžek kamenů co největší, musí být co nejdelší vzestupný počet kamenů (od 1 zvyšující se o jedničku) od posledního domu k prvnímu. Při reálných hrách se vyskytují stavy, které této situaci odpovídají, takové, že předepsanému počtu kamenů vyhovuje poslední, poslední a předposlední, nebo maximálně poslední, předposlední a 4. dům. Každopádně se vyplatí vyprázdnit domy blízko pokladnice, aby hráč v průběhu hry mohl tímto způsoben dostávat kameny do pokladnice. Pokud totiž vyprázdníme tyto domy a poté zahrajeme nějaký dům více vlevo, nebo zahraje protihráč dům ve kterém byl velký počet kamenů, dostanou se kameny do těchto prázdných domů, odkud je pak hráč může dostat do pokladnice.

5.1.3 Zajímání kamenů

Z výše uvedeného lze vyčíst, že z běžných tahů není možné získat větší množství kamenů. Tento problém řeší pravidlo, které říká, že pokud poslední kámen z přerozdělování padne do prázdného domu hráče, který je na tahu, a zároveň dům protihráče, který je na proti domu, do kterého byl vložen poslední kámen, není prázdný, tak všechny kameny, které se nachází v tomto protihráčově domě a i poslední kámen budou přesunuty do pokladnice hráče, který tah zahrál. V důsledku tohoto pravidla vyplývá, že takový tah je velmi výhodný, protože vede k možnosti získání potenciálně velkého množství kamenů. Například, pokud bude mít soupeř v domě 6 kamenů a hráč bude mít protější dům prázdný a zároveň bude moci zahrát tah, při kterém poslední kámen padne do

tohoto prázdného domu, získá hráč všech 6 protihráčových kamenů a kámen, který byl vložen do prázdného domu.

5.1.4 Obrana před zajmutí kamenů protihráčem

Výše popsané tahy se soustředily na získávání kamenů. Důležité je soustředit se také na obranu, jedná se o situace, při kterých protihráč může vzít kameny uložené v domě, který má soupeřův protější dům prázdny. Pokud by se protihráči povedlo poslední kámen uložit do tohoto domu, získal by tyto kameny on. Při takovém tahu by hráč mohl přijít o hodně kamenů. Právě v takovýchto tazích se rozhoduje o výsledku hry.

III. PROJEKTOVÁ ČÁST

6 Implementace herní strategie pro hru Kalaha

Praktickou část této práce představuje program Aplikace Mancala, napsaná v C++ společně s využitím Qt knihoven a Qt technologií. Mancala je implementována jako jeden program zahrnující jak grafické rozhraní tak samotnou výpočetní funkci.

6.1 Reprezentace hry

Jak už bylo popsáno v kapitole o hře Kalaha. Herní desku tvoří 12 domů a dvě pokladnice. V aplikaci byla herní plocha rozdělena na widgety, do nichž byly pomocí kreslící třídy QPainter nakresleny kružnice, reprezentující domy a po stranách obdélníky, které reprezentují pokladnice. Místo kamenů byla použita textová forma o počtu kamenů jak v jednotlivých domech, tak v pokladnicích.

Hru implementuje třída mancala. Obsahuje dvoudimenzionální pole domů. Pro každého hráče jeden řádek pole. Řádek obsahuje 6 položek, které představují jednotlivé domy. Vnitřní struktura není intuitivní. Za účelem, aby metody, které se budou starat o implementaci hry a zjišťování možných tahů, mohly pracovat co nejjednodušeji, je potřeba, aby hráči měli symetricky pojmenované herní prvky. Jde o to, že pokud by byly domy očíslovány pro indexaci do pole zprava doleva od 0 po 5, pro oba hráče, logicky by to neodpovídalo realitě. Hráči, sedící proti sobě, mají při stejné logice tahů, převrácené domy. Z důvodu aby algoritmy, zajištující tahy a rozhodování, nemusely řešit rozdíly v pohledech na desku, bylo rozhodnuto o odlišné logické reprezentaci domů v poli. Pro lepší pochopení je uvedena matice 6.1, která ukazuje převrácení jednoho řádku. Program pracuje s deskou tak, že se na ni dívá jakoby z vrchu. Při jakémkoli natočení (nebo můžeme chápat aktivaci jednoho nebo druhého hráče) bude herní plán identický.

Dále třída obsahuje pole čísel o dvou položkách, které představují aktuální počet kamenů v pokladnicích hráčů. Indexace pole domů a pole pokladnic je stejné, tz. i-tý řádek pole domů a i-tý prvek z pole pokladnic jsou stejného hráče.

$$Mancala\ domy = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix} \tag{6.1}$$

6.2 Deterministický přístup ke hře Kalaha

V rámci této kapitoly bude popsáno řešení pomocí hierarchického volání metod nad aktuálním stavem. Tato hierarchie znamená vzájemnou preferenci jednotlivých metod. Pokud je upřednostňovaný tah možný, tak bude vykonán, pokud ne, bude testován další. Implementaci řeší jednoduché větvení programu za pomocí podmínek. Pokud

metoda provedla tah při kterém získává zdarma nový, tak vrací hodnotu *true*.

```
1  bool mancala::PCplayer(unsigned short pl){
2    if(whichLastStoneMancala(pl)){
3      return true;
4    }else if(stoneToEmptyHouse(pl)){
5      return false;
6    }else if(defense(pl)){
7      return false;
8    }else if(firstHouse(pl)){//hrej prvni mozny dum
9      return false;
10   }
11   return false;
12 }
```

6.2.1 Metoda sowing

Tato metoda zajišťuje vykonání jednoho tahu ve hře včetně kontroly zda neobdrží hráč tah zdarma. Parametry, se kterými pracuje, jsou index hráče a index domu. Tyto dvě proměnné poslouží k přesné identifikaci domu, ze kterého se mají vybrat kameny. Ty se poté budou umísťovat po jednom do domů a hráčovy pokladnice, postupně, ve směru proti hodinovým ručičkám tak, jak je vysvětleno v pravidlech. Tato metoda využívá metodu *rotation*, která mění index domu ve směru proti hodinovým ručičkám. Pokud je na řadě pokladnice nastaví hodnotu na *MANCALA*. Tento stav je v kódu ošetřen a patřičně obsloužen. Musí se zvýšit počet kamenů v pokladnici o jeden, vyměnit hráče ve smyslu změny indexů domů (již nebudou kameny vkládány do domů aktuálního hráče, ale protihráče). O toto se stará metoda *changePlayer* (Změní index hráče na protivníka a zároveň tento index vrátí). V případě, že byl poslední kámen umístěn do pokladnice, získává hráč další tah. Metoda pak vrací hodnotu *true*. Testuje se, zda nebyl poslední kámen vložen do prázdného domu hráče. V takovém případě se musí ošetřit potenciální odebrání kamenů z protilehlého domu. Tuto situaci řeší metoda *lostStones*. Poslední věcí, která se musí kontrolovat, je zda nebude v další pozici pro vkládání kamene soupeřova pokladnice, pokud ano, tak se přeskočí na poslední hráčův dům. Řídící proměnnou cyklu je *count_stone*. Tato proměnná je inicializovaná na hodnotu vyjadřující kolik kamenů bylo odebráno z domu. Každou iteraci cyklu je snížena o hodnotu jedna. Cyklus probíhá dokud je hodnota nenulová.

```
1  bool mancala::sowing(unsigned short pl, unsigned short house){
2    unsigned short originalPlayer=pl;
3    unsigned short count_stone=Kalaha[pl][house];
4    Kalaha[pl][house]=0;
5    rotation(&house);
6    for(; count_stone!=0; count_stone--){
7      if(house==MANCALA){
8        mancala_count[pl]++;
9        changePlayer(&pl);
10       if(count_stone-1==0){//byl umisten posledni kamen => novy tah
11         return true; //priznak pro opakovany tah hrace
12       }
13     }else{
14     Kalaha[pl][house]++;          //zvyseni poctu kamenu o 1
15       //byl umisten posledni kamen?
```

```
16          if(pl==originalPlayer && count_stone-1==0 && Kalaha[pl][house]==1){
17            lostStones(pl,house); //odebrani kamenu
18            return false;
19          }
20        }
21        if(pl!=originalPlayer && house==0){//vynechani protivnikovy pokladnice
22          changePlayer(&pl);
23          house=COUNT_HOUSE-1;
24        }else{
25          rotation(&house);
26        }
27      }
28      return false;
29    }
```

6.2.2 Metoda lostStones

Metoda *lostStones* přijímá parametry k identifikaci domu a kontroluje protější dům, zda obsahuje kameny. Pokud ano, tak přeřadí tyto kameny do pokladnice hráče. Navíc do pokladnice musí být vložen i kámen z domu hráče. Počet kamenů v domech, které se vyprázdnily, musí být vynulovány.

```
1   unsigned short mancala::lostStones(unsigned short pl, unsigned short house){
2     unsigned short help=player[changePlayer(&pl)][COUNT_HOUSE-house-1];
3     if(help!=0){
4       Kalaha[pl][COUNT_HOUSE-house-1]=0;
5       mancala_count[changePlayer(&pl)]+=help+1;
6       Kalaha[pl][house]=0;
7     }
8     return help;
9   }
```

6.2.3 Obsluha hráčů na tahu

Před začátkem hry, se musí určit, zda bude začínat člověk nebo počítač. Oba dva hráči se v tazích střídají. Člověk hrající s počítačem aktivuje funkci *sowing* přes událost kliknutí myši na políčko domu. Toto kliknutí vyvolá signál, který obslouží slot a ten zavolá funkci *sowing*. Když hráč tah/y dokončí, podle toho jestli nezíská tah navíc, hraje počítač. Počítač se řídí podle strategie, která je předem naprogramovaná. Po výpočtu a provedení tahu počítačem, se opět čeká na událost myši (hru člověka).

6.2.4 Metoda stoneToEmptyHouse

Metoda na zjištění zda nelze zajmout soupeřovy kameny. Využívá pravidlo, kdy po vložení posledního kamene do svého prázdného domu, připadají všechny kameny z protějšího soupeřova domu, hráči, který tah zahrál. V cyklu se prochází domy patřící PC. Hledají se tři možné varianty:

- V první jsou hledány prázdné domy. Pokud je takový dům nalezen a protější dům není prázdný, tak se hledá dům ze kterého by šel zahrát takový tah, po kterém by

byl poslední kámen vložen do tohoto domu. Počet kamenů potenciálně získaných ze soupeřova domu bude uložen do pomocné proměnné *valueTheft*.

- Druhá možnost testuje speciální případ, kdy dům obsahuje přesně 13 kamenů (pro tuto konkrétní Kalahu se 48 kameny, obecně *COUNT_HOUSE*2+1* kamenů). V tomto případě se zahráním takového domu vloží poslední kámen zpět do tohoto domu a mohou se zajmout soupeřovy kameny z protějšího domu. V soupeřově domě však v tu dobu bude o jeden kámen navíc, právě o ten, který tam bude vložen při přerozdělování kamenů.

- poslední možnost je, když se přerozdělují kameny přes soupeře, ale nedojde se až k zdrojovému domu. Poslední kámen musí být opět umístěn do prázdného domu a soupeř v protějším musí mít alespoň jeden kámen. Druhá podmínka bude splněna vždy, protože než se kameny dostanou zpět k hráčovým domům, bude vloženo do každého soupeřova domu po jednom kameni.

Celý tento postup je uzavřen v cyklu, ve kterém se hledá nejlepší varianta. Pokud je nalezena výhodnější možnost, tak se aktualizují pomocné proměnné. Pokud byl nalezen alespoň jeden možný tah pro získání soupeřových kamenů, tak se ten nejvýhodnější provede. Metoda vrací *true*, pokud tah proběhl.

```
1  bool mancala::stoneToEmptyHouse(unsigned short p){
2    unsigned short help=0;
3    short valueTheft=0;
4    for(unsigned short i=0; i<COUNT_HOUSE-1; i++){     //prochazeni domu
5      //nalezeni prazdneho domu a protihracuv protejsi dum je obsazen
6      if(Kalaha[p][i]==0 && opositeHouse(p,i)>valueTheft){
7        //hledani domu ze ktereho padne posledni kamen do prazdneho domu
8        for(unsigned short j=i+1; j<COUNT_HOUSE; j++){
9          if(j-Kalaha[p][j]==i){                  //nalezeni hledaneho domu :)
10           valueTheft=opositeHouse(p,i);
11           help=j;
12           break;
13         }
14       }
15     }else if(Kalaha[p][i]==COUNT_HOUSE*2+1 && \\
16                 opositeHouse(p,i)+1>valueTheft){//padne do tohoto domu
17       valueTheft=opositeHouse(p,i)+1;
18       help=i;
19     //zajmuti kamenu lze provest po rozdeleni kamenu pres souperovy domy
20     //musi se kontrolovat meze indexu, kvuli pocitani se muze dostat pryc!!!
21     }else if(COUNT_HOUSE-(Kalaha[p][i]-COUNT_HOUSE-1-i)>=0 && \\
22       COUNT_HOUSE-(Kalaha[p][i]-COUNT_HOUSE-1-i)<COUNT_HOUSE && \\
23       Kalaha[p][COUNT_HOUSE-(Kalaha[p][i]-COUNT_HOUSE-1-i)]==0 && \\
24       opositeHouse(p,COUNT_HOUSE-(Kalaha[p][i]-COUNT_HOUSE-1-i))+1>valueTheft){
25             valueTheft=opositeHouse(p,COUNT_HOUSE-(player[p][i]-COUNT_HOUSE-1-i))+1;
26         help=i;
27     }
28   }
29   if(valueTheft>0){    //provedeni tahu
30     sowing(p, help);
31     return true;        //provedeno
32   }
33   return false;         //neprovedeno
34 }
```

6.2.5 Metoda whichLastStoneMancala

Jak už bylo popsáno v kapitole 5.1.2, je výhodné umístit kámen do pokladnice a získat tah navíc. Tato metoda zprava doleva hledá dům, který takový tah umožňuje. Tím, že tato metoda prochází domy zprava doleva, zajišťuje automatické odevzdávání kamenů do pokladnice, přesně tak, jak bylo vysvětlováno při analýze hry v připomínané kapitole. V takovém případě se metoda zavolá a vykoná opakovaně. Metoda vrací *true*, pokud provedla tah.

```
1  bool mancala::whichLastStoneMancala(unsigned short pl){
2    bool help=false;
3    for(unsigned short house=0; house<COUNT_HOUSE;){
4      if(Kalaha[pl][house]==house+1){
5        sowing(pl,house);
6        help=true;
7        house=0;
8      }else{
9        house++;
10     }
11   }
12   return help;
13 }
```

6.2.6 Metoda firstHouse

Tato metoda hledá zprava první neprázdný dům a přerozdělí kameny z tohoto domu.

```
1  bool mancala::firstHouse(unsigned short pl){
2    for(unsigned short house=0; house<COUNT_HOUSE;house++){
3      if(Kalaha[pl][house]!=0){
4        sowing(pl,house);
5        return true;
6      }
7    }
8    return false;
9  }
```

6.2.7 Metoda defense

Tato metoda pracuje obdobně jako metoda *stoneToEmptyHouse*. Při hledání domů, které jsou ohroženy ztrátou kamenů, jsou zapamatovány a následně je zahrán dům, který vykazuje největší ztrátu. Tím se docílí toho, že jsou kameny z tohoto domu přerozděleny a je již nemožné tyto konkrétní kameny získat. V zásadě není vyloučeno, že budou odstraněny všechny možnosti zajímání kamenů pro jiné domy. Dokonce zahráním tohoto tahu mohou vznikat další příležitosti k zajímání kamenů. Analýza takového vývoje už ale přesahuje deterministický přístup a týká se spíše kombinatorického výpočtu. Za povšimnutí stojí již poměrně komplikovaný výpočet indexu domu, který se má zahrát. Z výpočtu tahu zajmutí kamene se zjistí protilehlý (ohrožený dům) a ten se zahraje, aby byla odstraněna možnost této ztráty.

```
1  bool mancala::defense(unsigned short p){
2    changePlayer(&p);
3    unsigned short help=0;
4    short valueTheft=0;
5    for(unsigned short i=0; i<COUNT_HOUSE-1; i++){ //prochazeni domu
6          //nalezeni prazdneho domu a protihracuv protejsi dum je obsazen
7      if(Kalaha[p][i]==0 && opositeHouse(p,i)>valueTheft){
8        //hledani domu ze ktereho padne posledni kamen do praydneho domu
9        for(unsigned short j=i+1; j<COUNT_HOUSE; j++){
10         if(j-Kalaha[p][j]==i){                    //nalezeni hledaneho domu :)
11           valueTheft=opositeHouse(p,i);
12           help=i;
13           break;
14         }
15       }
16     }else if(Kalaha[p][i]==COUNT_HOUSE*2+1 && opositeHouse(p,i)+1>valueTheft){
17       valueTheft=opositeHouse(p,i)+1;
18       help=i;
19       //pripad, kdy zajmuti kamenu lze provest po rozdeleni kamenu pres souperovy domy
20     }else if(COUNT_HOUSE-(Kalaha[p][i]-COUNT_HOUSE-1-i)>=0 && \\
21       COUNT_HOUSE-(Kalaha[p][i]-COUNT_HOUSE-1-i)<COUNT_HOUSE &&\\
22       Kalaha[p][COUNT_HOUSE-(Kalaha[p][i]-7-i)]==0 && \\
23       opositeHouse(p,COUNT_HOUSE-(Kalaha[p][i]-7-i))+1>valueTheft){
24         valueTheft=opositeHouse(p,COUNT_HOUSE-(Kalaha[p][i]-7-i))+1;
25         help=COUNT_HOUSE-(Kalaha[p][i]-7-i);
26     }
27   }
28   if(valueTheft>0){
29     changePlayer(&p);
30     if(Kalaha[p][COUNT_HOUSE-help-1]!=0){//nekdy se stava ze je prazdny
31       sowing(p,COUNT_HOUSE-help-1);
32       return true;
33     }
34   }
35   return false;
36 }
```

6.3 Výběr největšího zisku

Tento algoritmus aplikuje teorii her na Kalahu. Vypočítává zisk kamenů z každého možného tahu. Všem tahům přiřadí ohodnocení, které se vzájemně porovná a bude vybrána nejlepší varianta, tz. tah při kterém se získá největší počet kamenů.

6.3.1 Metoda DoEver

Tato metoda řeší tahy, které se mohou udělat bez obav z jakékoli ztráty. Jedná se o situace, kdy zahrání tahu přinese jen pozitivní zisk a žádnou ztrátu. Metoda se velmi podobá metodě *whichLastStoneMancala*, ale provádí pouze podmnožinu tahů, tak aby byly provedeny jen ty tahy, které nepřinesou potenciální ztrátu. Postupuje se opět od domu s indexem 0, to znamená dům, který je nejvíce vpravo. Pokud obsahuje jeden kámen, tak při zahrání tohoto domu, půjde kámen přímo do hráčovy pokladnice. Proto se může bez obav tento tah provést. Hráč tím získá kámen a ještě další tah navíc. Kdyby byl proveden jiný tah, mohl by být do tohoto domu přidán další kámen, a hráč by nezískal tah navíc současně s tímto kamenem. Proto je pro nejlepší strategii zahrát tento tah. Tím ovšem metoda *DoEver* nekončí. V případě, že je počet kamenů

vzestupný od jednoho do teoretických šesti kamenů postupně v domech od nejpravějšího do nejlevějšího, tak lze kameny téměř všechny umístit do pokladnice. Takový postup je popsán v podkapitole 5.1.2 . Tento postup může být využit i v případě, kdy první dům je prázdný, přičemž je zachována posloupnost obsazenosti domů dle pravidla $pocet\ kamenu = i - ty\ dum + 1$. Situace vypadá například takto: [6 5 4 3 2 0]. I v tomto případě je nejlepší varianta zahrát dům se dvěma kameny. Hráč tím během dvou tahů získá oba kameny. V situacích, kdy se prázdný dům nachází dál od pokladnice než ve vzdáleností dva a méně, je situace složitější. V případě, kdy do prázdného domu může být umístěn poslední kámen z přerozdělovaných kamenů, tak aby byly zajaty soupeřovy kameny z protějšího domu, nemusí být pokračování v předešlém postupu nejvýhodnější. Zahrání takového tahu si hráč může znemožnit následné zajmutí soupeřových kamenů. Z tohoto důvodu metoda *DoEver* automaticky přeskočí pouze prázdný první dům. V ostatních případech dál zkoumá, zda není možné zahrát tah, který byl před chvíli popsán, aby se zajaly soupeřovy kameny. Pokud ano, tak metoda ukončí svou činnost a vrací hodnotu *true* v případě, kdy proběhl alespoň jeden tah. V každém takovém tahu se získá nový tah zdarma, proto stačí aby metoda vracela jen dvě hodnoty.

```
1  bool mm::DoEver(unsigned short p){
2    bool help=false;
3    for(unsigned short i=0; i<COUNT_HOUSE;){
4      if(Kalaha[p][i]==i+1){
5        sowing(p,i);
6        help=true;
7        i=0;
8      //podminka ze prvni dum je prazdny nebo protejsi dum je prazdny
9      }else if(Kalaha[p][i]==0 && (i==0 || oppositeHouse(p,i)==0)){
10       i++;//tak preskoc
11     }else{//jinak zkontroluj zda lze umistit kamen
12       for(unsigned short j=i; j<COUNT_HOUSE; j++){
13         if(j-Kalaha[p][j]==i){
14           return help;//existuje moznost zajmuti kamenu
15         }
16       }
17       i++;//preskoc
18     }
19   }
20   return help;
21 }
```

Velký význam tato metoda má v tom, že se nebudou muset složitě posuzovat i varianty, které by byli vybrány i po ohodnocení účelovou funkcí. Ušetří se jak výpočetní čas, tak paměť potřebná pro samotný výpočet. Při jednoúrovňovém ohodnocení variant to není zásadní, ale v případě hloubkového prohledávání stromu hry, tato metoda může mýt zásadní vliv na zjednodušení výpočtu.

6.3.2 Hodnotící funkce

Na hodnotící funkci musí být kladen požadavek, aby co nejpřesněji ohodnotila zisk z tahu. Pro úspěšnou hru Kalaha je zásadní zisk co největšího počtu kamenů. Z tohoto

důvodu musí hodnotící funkce posuzovat tah vzhledem k zisku kamenů. Hodnocení by mělo následovat až po dokončení celého kola, tzn. po odehrání hráče i protihráče. Protihráč totiž může zahrát takový tah, který předcházející tah hráče znehodnotí ve smyslu, že získá víc kamenů než on. Počet získaných kamenů se vypočítá z rozdílu přibytků kamenů v pokladnicích obou hráčů. Hodnotí se tedy stav hry, ve kterém bude pokračovat ve hře hráč, který danou situaci analyzuje. Z toho vyplývá, že bude mít k dobru volbu tahu z tohoto konečného stavu, který byl analyzován. Proto hráč může připočíst kameny které získá z tahů, které nebudou mít negativní následky. Popis těchto tahů je popsán v podkapitole 6.3.1. Ostatní tahy už nebudou brány na zřetel, protože na ně může být použit protitah se záporným ziskem. Pro dokonalou analýzu by bylo zapotřebí, aby byly vypočítány veškeré kombinace tahů až k listovým uzlům pomyslného grafu hry. Následně by byl vybrán tah, který by vedl k největšímu zisku kamenů. Hodnocení stavů řeší metoda *rate*. Primárně počítá zisk kamenů z pozice hráče, který analýzu spustil. Zisk vypočítá z rozdílů počtu kamenů v jednotlivých pokladnicích hráčů. Vychází se ze stavu, ve kterém se začaly analyzovat možnosti tahů. Stavy, ve kterých je volána metoda *rate*, jsou určeny předem stanoveným počtem zanoření algoritmu pro prohledávání stromu hry. Z takového posledního povoleného stavu je vypočítán přírůstek kamenů v pokladnicích vzhledem k poslednímu skutečně provedenému tahu hry. Hodnotící funkce zvyšuje měřítko hodnoty zisku kamenů, a to desetkrát. Činí tak z důvodu započítaní i jiných faktorů, které ovšem nemají takovou důležitost. Konkrétně přímo v metodě *rate* přihlíží k tomu, zda je prázdný první dům. Tato skutečnost znamená výhodu, že pokud do něho bude umístěn kámen, je vysoká pravděpodobnost možnosti umístění takového kamenu do pokladnice. Proto hodnotící funkce přičítá takovému stavu hodnotu pět. Další ovlivňování hodnoty se děje v případě, že je získán tah zdarma. Toto zvyšování ohodnocení stavů je ošetřeno v samotném rekurzivním řešení celého procházení stromu hry, které bude vysvětleno dále. Zde je ukázána implementace funkce *rate*.

```
1  short mm::rate(unsigned short p){
2    DoEver2(p); //prictou se jiste kameny
3    short scaleTreasury=10;
4    //nejvetsi vahu musi mit pribitek v pokladnici
5    short r=( mancala_count[p]-mancala_count_original[p])*scaleTreasury;
6    if(player[p][0]==0){
7      r+=5;
8    }
9    changePlayer(&p);
10   r-=(mancala_count[p]-mancala_count_original[p])*scaleTreasury;
11   return r;
12 }
```

6.3.3 Prostorové prohledávání stromu hry

Jak bylo zmíněno v teoretické části, Mancala patří do her v explicitním tvaru, pro které se může vytvořit strom hry. Princip je podobný jako u maticových her. Kde se vytváří tabulka tvořená hodnotami dle hodnotící funkce. Z ní se poté dle zvolené strategie

rozhodování vybírá nejvhodnější varianta akce. V této práci byla popsána snaha o deterministické rozhodování volby tahů z množiny seřazených základních strategií. V této části se práce bude zabývat využití Teorie her. Bude předpokládat hru dvou inteligentních hráčů, kteří se snaží vybrat nejlepší strategii pro výhru. V každém stavu hry, kdy se bude muset stroj rozhodnout, jaký tah vybrat, se bude dle pravidel analyzovat každý možný tah v daném stavu. Pro implementaci bude využito to co již bylo o hře Kalaha zjištěno v analýze a implementaci strategií na základě analýzy. To umožní, aby se zjednodušilo prohledávání. Například bude využívána metoda *DoEver*, která automaticky provede tahy, které ve výhodnosti nemají konkurenci. Pro ostatní možnosti musí být rozhodnuto, jaká je nejvýhodnější. To určí hodnotící funkce, která byla popsána výše. Aby ovšem hodnotící funkce měla co hodnotit, musí existovat více variant pro rozhodování volby tahů. Pokud jsou uvažovány obecné možnosti, tak existuje až 6 možností tahů v rámci jednoho stavu hry. Problém však nastává s možností získání tahu navíc. Tato varianta tahu vlastně rozšiřuje tento tah o dalších až 6 možností. Toto rozšíření nemusí být pouze jedno, ale může být opakované třeba až do relativně vysokého počtu. Opakování může atakovat až desítku získání nového tahu. Tato skutečnost však vnáší do implementace stromové struktury a především rozhodovací logiky komplikace.

Nejdříve se musí ujasnit struktura prohledávání stromu. O tuto činnost se starají dvě funkce, které se navzájem rekurzivně volají podle počtu prohledávaných pater stromu. Jsou to funkce *MAX* a *MIN*. Jak už název napovídá, první funkce bude hledat maximální hodnotu stavů, ohodnocených hodnotící funkcí, *MIN* pak minimum. Hráč, který analyzuje jednotlivé možnosti tahů, vybírá variantu nejvyššího zisku kamenů. Naopak protihráč by se měl dle racionality snažit o to, aby jeho tah vedl k co možná nejmenšímu zisku prvního hráče. Algoritmus předpokládá, že se protihráč bude chovat racionálně, neboli bude minimalizovat soupeřův zisk.

Nyní bude vysvětlena činnost funkce *MAX*. Na začátku je otestováno, zda již nenastalo maximální zanožení ve stromové struktuře. Pokud ano, tak je volána funkce pro ohodnocení aktuálního stavu hry a tato hodnota je vrácena k porovnání s ostatními možnostmi o úroveň výše. Poté je vytvořena proměnná pole stavů hry o velikosti šesti položek, které jsou inicializovány aktuálním stavem. V cyklu se pak řeší jednotlivé tahy. Nejprve je zkontrolováno, zda dům není prázdný. Pokud ano, tak nemůže být uvažováno zahrát tah s tímto domem, proto algoritmus obsluhu dalšího kódu přeskočí. Následuje provedení tahu v dané kopii stavu. Zároveň se řeší, zda tah nevede ke získání tahu navíc. Zjistí se to návratovou hodnotou metody *sowing*. Pokud vrátí hodnotu *false*, tak nový tah nebyl získán a hráčem na tahu se stane protihráč, jehož výběr nejlepší strategie řeší funkce *MIN*. Jako parametry jí předává stav hry po provedení analyzovaného stavu, hráče, který bude na tahu, a aktuální zanoření ve stromové struktuře. Pro změnu hráče se používá metoda *changePlayer* a nastavuje se jí hráč, který provádí tah, nebo z jehož

pohledu se provádí analýza. Pokud funkce *MIN* vrátí hodnotu *SHRT_MAX*, tak to znamená, že v daném stavu už protihráč nemá žádné kameny, neboli hra již skončila. Proto bude provedeno ohodnocení stavu pomocí metody *rate* a budou přičteny zbylé kameny v domech aktuálního hráče.

V případě že metoda *sowing* vrátí hodnotu *true*, znamená to, že poslední kámen připadl do pokladnice hráče a je získán nový tah. Proto se v tomto případě volá opět metoda *MAX*. Takto se může tato metoda několikrát zanořit. Ve většině případů jsou tahy se ziskem nového tahu výhodné, a proto je k ohodnocení konečného stavu přičtena hodnota jedna, aby se priorita těchto tahů zvýšila. Činí se tak ovšem s menší váhou než hodnocení zisku kamene. Zisk kamene je ohodnocen deseti body, zatímco zisk nového tahu pouze jedním bodem.

V cyklu se z až šesti možností vybere ta, která bude mít nejvyšší hodnotu. Při aktualizaci nejvýhodnější varianty tahu se ukládá nejen maximum zisku bodů pro porovnání s ostatními variantami, ale i index domu, pro který je maximální zisk možný. Po skončení cyklu se nastaví index domu do ukazatele aby s ním mohly pracovat funkce, která tuto funkci zavolala. Hodnotu zisku funkce *MAX* vrací jako návratovou hodnotu. Pro výběr maxima je použit operátor $>$ a znamená to, že jsou upřednostněny domy s nižším pořadovým číslem. Pokud by existovaly stejně ohodnocené tahy, bude proveden ten, který bere kameny pro přerozdělení z domu, který je blíže pokladnici z důvodu toho, že je výhodnější mít u pokladnice prázdné domy.

```
1  int MAX(mm *m, unsigned short p, unsigned short *index, unsigned short plunging){
2    if(plunging>=STOP_ITERATION){
3      return m->rate(p);
4    }
5    unsigned short in=0;
6    short help=SHRT_MIN;
7    short max=SHRT_MIN;
8    unsigned short best=0;
9    mm array[COUNT_HOUSE]=*m;
10   for(unsigned short i=0; i<COUNT_HOUSE; i++){
11     if(m->player[p][i]==0){//pokud je prazdny dum tak preskoc
12       continue;
13     }
14     if(array[i].sowing(p, i)==true){
15       help=MAX(&array[i], p, &in, plunging);
16       if(help==SHRT_MIN){
17         help=array[i].rate(p);
18       }else{
19         help+=1;
20       }
21     }else{
22       m->changePlayer(&p);
23       help=MIN(&array[i], p, plunging+1);
24       m->changePlayer(&p);
25       if(help==SHRT_MAX){//konec hry, musi se ale porovnat vyhodnost teto vetve
26         help=array[i].rate(p);
27         for(unsigned short j=0; j<COUNT_HOUSE; j++){
28           help+=m->player[p][j]*10;
29         }
30       }
31     }
32     //pozor nesmi mit maximalni hodnotu (kvuli ohodnoceni ke konci hry)
```

```
33      // kdy se neprovede kompletni zanoreni a uzel se hodnoti v min!
34      if(help>max && help!=SHRT_MAX){
35        max=help;
36        best=i;
37      }
38    }
39    *index=best;
40    return max;
41  }
```

Funkce *MIN* je téměř stejná jako *MAX*. Rozdíl je v tom, že nehledá stav s maximální hodnotu, ale s minimální. Vybírá tak minimální ztrátu. Musí si pohlídat korektní práci s indexem hráče, protože hodnotící funkce vypočítává hodnotu pro hráče, který chce zjistit ohodnocení jednotlivých stavů. Zde se tedy jedná o počítač. Pro porovnání funkcí a jejich rozdílností je zde uveden kód.

```
1   int MIN(mm *m, unsigned short p, unsigned short plunging){
2     if(plunging>=m->STOP_ITERATION){
3       m->changePlayer(&p);
4       return m->rate(p);
5     }
6     unsigned short in;
7     short help=SHRT_MAX;
8     short min=SHRT_MAX;
9    mm array[COUNT_HOUSE]=*m;
10    for(unsigned short i=0; i<COUNT_HOUSE; i++){
11      if(m->player[p][i]==0){//pokud je prazdny dum tak preskoc
12        continue;
13      }
14      if(array[i].sowing(p, i)==true){
15        help=MIN2(&array[i], p, plunging);
16        if(help==SHRT_MAX){
17          m->changePlayer(&p);
18          help=array[i].rate(p);
19          m->changePlayer(&p);
20        }
21      }else{
22        m->changePlayer(&p);
23        help=MAX2(&array[i], p, &in, plunging+1);
24        if(help==SHRT_MIN){//konec hry, musi se ale porovnat vyhodnost teto vetve
25          help=array[i].rate(p);
26        }
27        m->changePlayer(&p);
28      }
29      if(help<min && help!=SHRT_MIN){
30        min=help;
31      }
32    }
33    return min;
34  }
```

6.4 Validace Aplikace

Kontrola správnosti algoritmů byla provedena vyzkoušením aplikace s kontrolou chování. V případě, že bylo nalezeno chybné chování a nekorektní výsledek nějakého tahu, byl každý takovýto nedostatek analyzován, nalezena příčina a následně programově opravena nebo implementován dodatečný kód, který řešil zapomenutý stav nebo chování.

6.5 Vyhodnocení

Pro vyhodnocení úspěšnosti algoritmů musí být zvolena kritéria, podle kterých bude daný algoritmus hodnocen. Zásadní pro hru Kalahu je počet získaných kamenů, ten určuje, zda hráč zvítězil a jakou měrou ve hře dominoval. Mezi další kritéria patří rychlost výpočtu. Pokud bude algoritmus pracovat s příliš složitým výpočtem nebo bude počítat příliš mnoho kombinací tahů, jejichž výhodnost bude porovnávat, může být pomalý a pro interaktivní hru mezi počítačem a člověkem nepoužitelný. Dále může být algoritmus hodnocen podle toho, jaké tahy v konkrétní situacích bude preferovat.

			číslo hry a skóre									
název alg.	začíná	skóre	1	2	3	4	5	6	7	8	9	10
Deterministický	PC	PC	29	35	29	29	29	33	28	32	25	32
		Hráč	19	13	19	19	19	15	20	16	23	16
	Hráč	PC	7	21	9	7	12	8	22	8	17	12
		Hráč	41	27	39	41	36	40	26	40	31	35
MAX-MIN 1	PC	PC	30	32	41	32	36	33	38	23	26	22
		Hráč	18	16	7	16	12	15	10	25	22	26
	Hráč	PC	13	14	26	16	28	19	13	26	32	13
		Hráč	35	34	22	32	20	29	35	22	16	35
MAX-MIN 2	PC	PC	31	30	36	35	25	29	35	33	41	34
		Hráč	17	18	12	13	23	19	13	15	7	13
	Hráč	PC	25	25	20	23	23	23	38	19	24	16
		Hráč	23	23	28	25	25	25	10	29	24	32
MAX-MIN 3	PC	PC	35	35	30	25	25	30	33	26	30	32
		Hráč	13	13	18	23	23	18	15	12	18	16
	Hráč	PC	13	22	21	20	27	28	21	22	27	22
		Hráč	35	26	27	28	21	20	27	26	21	26
MAX-MIN 4	PC	PC	40	39	41	35	33					
		Hráč	5	9	6	5	5					
	Hráč	PC	26	32	17	29	28					
		Hráč	22	16	31	19	20					

Tab. 6.1 Skóre pro jednotlivé algoritmy při hře se znalým hráčem

Pro analyzování výsledků algoritmů bylo provedeno vždy 10 experimentů pro každou metodu s výjimkou *MAX-MIN 4* z důvodu velké časové náročnosti. Experiment se řídil těmito podmínkami: Hra probíhala mezi zkušeným hráčem a implementovaným algoritmem a výsledek byl zaznamenán do tabulky 6.1. Hráč se snažil o co nejlepší hru. Byl kladen důraz, aby se zkoumal algoritmus jak z pozice začínajícího hráče, tak z pozice druhého hráče. Zkušený hráč zkoušel různé varianty tak, aby zjistil chování algoritmu. Ve zmiňované tabulce je uvedeno pět implementovaných metod, pro které experimenty probíhaly. V zásadě se jedná o dva typy. První je pojmenovaný deterministický algoritmus. Jeho chování je rozebráno v kapitole 6.2. Algoritmus je sestaven na základě analýzy pravidel. Odlišný přístup volí ostatní algoritmy, které jsou zalo-

ženy na Teorii her. Jsou zde pojmenovány *MAX-MIN 1*, *MAX-MIN 2*, *MAX-MIN 3* a *MAX-MIN 4*. Všechny čtyři pracují na stejném principu, liší se pouze v hloubce, pro kterou vytváří podstrom hry. Algoritmus *MAX-MIN 1* pracuje s jedním zanořením, algoritmus *MAX-MIN 2* se dvěma zanořeními atd. Když je vytvořen podstrom, jsou ohodnoceny všechny jeho koncové uzly a následně vybrána varianta, která má přinést největší zisk kamenů, viz 6.3.2.

Jednotlivé algoritmy byly postaveny i proti sobě, aby se ukázalo, jak si budou vést ve vzájemném souboji. Tabulka 6.2 zachycuje jejich výsledky.

Začínající	Deter.	MAX-MIN 1	MAX-MIN 2	MAX-MIN 3	MAX-MIN 4
Deterministický	32:16	33:15	35:13	24:24	18:30
MAX-MIN 1	35:13	22:26	25:23	29:19	41:7
MAX-MIN 2	35:13	32:16	24:24	32:16	41:7
MAX-MIN 3	43:5	34:14	32:16	32:16	41:7
MAX-MIN 4	43:5	37:11	41:7	41:7	41:7

Tab. 6.2 Skóre soupeření jednotlivých algoritmů

Z tabulky 6.1 je vidět, že deterministický algoritmus dobře funguje v případě, že je použit pro začínajícího hráče. Ve všech deseti případech zvítězil počítač. Co se týče opačné pozice, když počítač hraje druhý v pořadí, tak je velmi nedostatečný. V tabulce je poměrně velké zastoupení zisku velmi malého množství kamenů. Co je ovšem ještě více podstatné, že nad lidským hráčem v těchto deseti případech ani jednou nezvítězil. V případě zkoumání tabulky 6.2 bylo zjištěno, že funguje také spolehlivě v případě protivníka se stejnou strategií. Co se týče algoritmů *MAX-MIN*, obstojí pouze nad dvěma nejjednoduššími variantami, nad těmi, které pracují pouze s jedním, nebo dvojím zanořením. Toto zjištění stojí za povšimnutí. V případě začátečníků ve hře Kalaha, může osvojení si v zásadě jednoduchého algoritmu (6.2) vést, za předpokladu začínající pozice, k úspěchům ve hře. Není zaručen stoprocentní úspěch, ale pravděpodobnost je vysoká.

Při podrobné analýze obou tabulek je zřejmé, že algoritmy vykazují jiné výsledky pro situace, ve kterých reprezentují začínajícího hráče, nebo druhého hráče. U deterministického algoritmu a algoritmu označeného *MAX-MIN 1* není možné vyvodit absolutní závěr z pohledu zaručené výhry. Při zkoumání zbylých variant, to vypadá jinak. U varianty *MAX-MIN 2* ze všech pokusů, kdy počítač zahajoval hru, neprohrál ani jednou, pouze jednou remizoval. Stejně si vedl i v porovnání s ostatními algoritmy (tabulka 6.2). Nezvítězil pouze jednou a to v případě, když remizoval pouze v souboji se sebou samým. *MAX-MIN 3* a *MAX-MIN 4* zvítězili vždy, ať už měly proti sobě kohokoli. V případě *MAX-MIN 4* je způsob výhry drtivý a to nejen vůči hráči, ale i proti zbylým algoritmům. Nevýhoda této metody, je v její časové náročnosti. Prohledává čtyři úrovně podstromu, což je zhruba $6^8 = 1679616$ uzlů. Reálně to bude více či

méně, protože existují tahy navíc a naopak prázdné domy neumožňují zahrát tah. V každém případě, při měření výpočtu na běžně se vyskytujícím počítači, probíhal výpočet tahu až 19 sekund, což je už nepřijatelně dlouhá doba. Hra s takovými dlouhými intervaly mezi výměnami tahů je nezáživná. Varianta *MAX-MIN 3*, která má přibližně $6^6 = 46656$ uzlů, je ve výpočtu rychlá a nezpůsobuje čekání na protivníka. Z tohoto rozboru je závěr, že algoritmy *MAX-MIN 3* a *MAX-MIN 4* v praktických příkladech, kdy zahajují hru, vedou k výhře.

V případech kdy začínal hráč, vždy dokázal nalézt strategii, která ho přivedla k výhře.

V rámci analýzy her byla potvrzena informace ze stánky *www.thesprucecrafts.com*, která tvrdí, že pro začínajícího hráče je výhodná varianta zahrát čtvrtý dům a poté první. Tímto získá hned na začátku dva kameny a navíc získá výhodné rozložení kamenů v domech.

Při porovnávání algoritmů bylo zjištěno, že je možné získat 43 kamenů z 48 do své pokladnice, což je velice velký počet.

Ve snaze matematicky dokázat, že začínající hráč má vždy možnost vyhrát nezávisle na protihráči, bylo naraženo na výpočetní problém. Rekurzivní algoritmus byl odvozen od *MAX-MIN*. Je ovšem příliš výpočetně velice náročný. Musí totiž projít všechny uzly podstromu, které nesplňují podmínku již jisté výhry některého hráče. Program, který toto zjišťoval, byl po třech dnech vypnut, aniž by tuto tezi potvrdil.

ZÁVĚR

V diplomové práci byly podrobně analyzovány pravidla hry Kalahy. Na základě analýzy byl vymyšlen algoritmus, který podle priorit prováděl tahy. V případech, kdy algoritmus reprezentoval začínajícího hráče, byly zjištěny dobré výsledky. Pokud byl naopak v roli druhého hráče, vykazoval nedostatečnou úspěšnost ve smyslu vyhrávání her.

Hlavním cílem práce byla aplikace Teorie her na tuto problematiku a následné vyhodnocení úspěšnosti takového přístupu. Na základě analýzy pravidel byla navržena herní strategie, která je reprezentovaná hodnotící funkcí a algoritmem, který prochází podstrom hry. Při analýze výsledků bylo zjištěno, že hra Kalaha zvýhodňuje začínajícího hráče a s velkou pravděpodobností existuje vždy taková posloupnost tahů, která nezávisle na protihráči povede k výhře. Tato skutečnost se z důvodu příliš rozsáhlého prohledávání stavů hry nepodařila matematicky dokázat. Provedené experimenty ovšem tento jev silně naznačují.

V rámci práce byla vytvořena strategie, která splňuje podmínky pro úspěšné hraní hry Kalahy. V situacích, kdy stroj hru začínal, nenašel v experimentech přemožitele. V případech, kdy algoritmus nezačínal, dokázal konkurovat hráči a případné chyby nemilosrdně trestat. Obecně lze konstatovat, že tato vytvořená strategie konkuruje lidskému hráči.

Dále byla ověřena informace, že pro začínajícího hráče je výhodné provést tah nad třetím domem, tím získá tah zdarma a poté, je vhodné zahrát první dům. Hráč tím získá dobré rozložení kamenů do hry.

Všechny poznatky byly implementovány do nové aplikace hry Kalaha. Aplikace obsahuje implementované herní strategie i jednoduché grafické prostředí, které umožňuje hru člověku proti stroji. Všechny experimenty byly provedeny ve zmíněné aplikaci.

V rámci možného pokračování této práce, by bylo možné se zabývat důkazem o výhodnosti začínajícího hráče a o vylepšení vytvořené aplikace.

SEZNAM POUŽITÉ LITERATURY

[Wik(2018a)] Mancala. In: Wikipedia: the free encyclopedia, 2018a. Dostupné z: <https://en.wikipedia.org/wiki/Mancala>. [Online; navštíveno 13. 01. 2019].

[Wik(2018b)] Kalah. In: Wikipedia: the free encyclopedia, 2018b. Dostupné z: <https://en.wikipedia.org/wiki/Kalah>. [Online; navštíveno 10. 02. 2019].

[arcadespot(2019)] ARCADESPOT. Mancala, 2019. Dostupné z: <https://www.arcadespot.com/game/mancala/>. [Online; navštíveno 28. 12. 2018].

[Blackwell(1964)] BLACKWELL, D. *Teorie her a statistického rozhodování.* Československá akademie věd, 1964.

[Brown(2019)] BROWN, S. Basic Strategy for Mancala, 2019. Dostupné z: <https://www.thesprucecrafts.com/>. [Online; navštíveno 25. 04. 2019].

[Cach(2018)] CACH, O. Toguz Kumalak, 2018. Dostupné z: <http://www.mankala.cz/toguzkumalak.html>. [Online; navštíveno 02. 03. 2019].

[Chroboczek(2013)] CHROBOCZEK, M. *Grafická uživatelská rozhraní v Qt a C++.* Computer Press, 2013. ISBN 978-80-251-4124-3.

[FORCH(2014)] FORCH, V. Hry v rozšířené formě [online]. Diplomová práce, Masarykova univerzita, Přírodovědecká fakulta, Brno, 2014. Dostupné z: <https://is.muni.cz/th/h1ojt/>.

[Gering(2018)] GERING, R. Toguz Kumalak, 2018. Dostupné z: <http://www.fandom.com/wiki/ToguzKumalak>. [Online; navštíveno 02. 03. 2019].

[Říha(1972)] ŘÍHA, O. Základy teorie her, 1972.

[Hrubý(2010a)] HRUBÝ, M. Doprovodné texty ke kurzu Teorie her, 2010a.

[Hrubý(2010b)] HRUBÝ, M. Doprovodné texty ke kurzu Teorie her, 2010b. Dostupné z: <http://www.fit.vutbr.cz/~hrubym/THE/sk-2-nekoo.pdf>. [Online; navštíveno 28. 03. 2019].

[Hykšová(2019)] HYKŠOVÁ, M. Teorie her a optimální rozhodování, 2019. Dostupné z: <http://euler.fd.cvut.cz/predmety/teorie_her/hry.pdf>. [Online; navštíveno 28. 03. 2019].

[Liberty(2007)] Liberty, J. *Naučte se C++ za 21 dní.* Computer Press, a.s., 2007.

[LLC(2019)] LLC, M. P. Mancala, 2019. Dostupné z: <https://www.mathplayground.com/mancala.html>. [Online; navštíveno 28. 12. 2018].

[Ltd(2019a)] Ltd, A. S. P. Mancala, 2019a. Dostupné z: <https://https://play.google.com/store/apps/details>. [Online; navštíveno 5. 4. 2019].

[Ltd(2019b)] Ltd, G. C. Mancala, 2019b. Dostupné z: <https://play.google.com/store/apps/details>. [Online; navštíveno 5. 4. 2019].

[Maňas(1969)] Maňas, M. *Teorie her a optimální rozhodování.* Vysoká škola ekonomická v Praze, 1969.

[Pearson(2016)] Pearson, K. Sowing the Seeds of Competitive Play. *Expedition.* 2016, 58, 1, pp. 24 – 27. ISSN 00144738.

[playdrift(2019)] playdrift, M. Mancala, 2019. Dostupné z: <https://mancala.playdrift.com/>. [Online; navštíveno 28. 12. 2018].

[Priebe(2019)] Priebe, L. Mancala, 2019. Dostupné z: <https://www.superhry.cz/games/240/>. [Online; navštíveno 28. 12. 2018].

[Qt(2019)] Qt, T. C. Qt, 2019. Dostupné z: <https://www.qt.io>. [Online; navštíveno 23. 03. 2019].

[Sawa(2015)] Sawa, Z. Teorie her, 2015. Dostupné z: <http://www.cs.vsb.cz/sawa/teh/opora/TEH-opora.pdf>.

[Wikiknihy(2018)] Wikiknihy. Pokladnice her/Mankalové hry/Oware, 2018. Dostupné z: <https://cs.wikibooks.org/w/index.php>. [Online; navštíveno 02. 03. 2019].

[Wikipedie(2019)] Wikipedie. Teorie her. Wikipedie: Otevřená encyklopedie, 2019. Dostupné z: <https://cs.wikipedia.org/w/index.php?title=Teorie_her&oldid=16868094>. [Online; navštíveno 28. 03. 2019].

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I. KÓD PRO OVĚŘENÍ ZARUČENÉ VÝHRY PRVNÍHO HRÁČE

V závislosti na podezření, že pravidla Kalahy zvýhodňují prvního hráče, byl vytvořen algoritmus, který měl zjistit, zda vždy existuje taková posloupnost tahů, která povede k výhře prvního hráče. Kvůli výpočetní náročnosti nebyl výsledek získán.

Algoritmus prohledává stromem hry. Zanořování ukončí až se dostane do stavu, kdy jedna pokladnice obsahuje více než polovinu kamenů. Pokud je to pokladnice prvního hráče, vrací hodnotu *true*, jinak vrací *false*. Prvního hráče reprezentuje funkce *MAX3*, která vybere vždy hodnotu *true*. Druhý hráč je naopak reprezentován funkcí *MIN3* a naopak vybírá hodnotu *false* (má totiž snahu zabránit výhře prvního hráče). Funkce vznikly modifikací funkcí *MAX* a *MIN*.

```
bool MAX3(mm *m, unsigned short p){
  if(m->mancala_count[p]>=25){
    return true;
  }else if(m->mancala_count[m->changePlayer(&p)]>=25){
    return false;
  }
    m->changePlayer(&p);//kvuli dotazovani v pefeslem if
    mm array[COUNT_HOUSE]=*m;
    for(unsigned short i=0; i<COUNT_HOUSE; i++){
      if(m->player[p][i]==0){//pokud je prazdny dum tak preskoc
        continue;
      }
      if(array[i].sowing(p, i)==true){
        if(MAX3(&array[i], p)==true){
          return true;    }
      }else{
        m->changePlayer(&p);
        if(MIN3(&array[i], p)==true){
          return true;      }
       m->changePlayer(&p);
  } }
  return false;
}
```

Funkce *MIN3* je opačná funkci *MAX3*.

```
bool MIN3(mm *m, unsigned short p){
  if(m->mancala_count[p]>=25){
    return false;
  }else if(m->mancala_count[m->changePlayer(&p)]>=25){
    return true;
  }
  m->changePlayer(&p);
  mm array[COUNT_HOUSE]=*m;
  for(unsigned short i=0; i<COUNT_HOUSE; i++){
    if(m->player[p][i]==0){//pokud je prazdny dum tak preskoc
      continue;
    }
    if(array[i].sowing(p, i)==true){
      if(MIN3(&array[i], p)==false){
        return false;    }
    }else{
      m->changePlayer(&p);
      if(MAX3(&array[i], p)==false){
        return false;    }
      m->changePlayer(&p);
  } }
  return true;
}
```