Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Teorie her a její aplikace

Jakub Kolařík

Bakalářská práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2018/2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jakub Kolařík**
Osobní číslo:	**I16308**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Teorie her a její aplikace**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Práce by měla v úvodní části obsahovat základní pojmy a principy teorie her. Dále by se měla zabývat jednotlivými antagonistickými hrami s konečně mnoho strategiemi a zmínit také antagonistické hry s nekonečně mnoho strategiemi. Dále by práce měla obsahovat neantagonistické konflikty a to jak pro nekooperativní strategie, tak i pro kooperativní strategie, které by bylo dále vhodné rozdělit na strategie s přenosnou a s nepřenosnou výhrou. Vše by bylo vhodné doplnit reálnými hrami a situacemi, z nichž některé je možno implementovat ve vhodném programovacím jazyce a doplnit výsledky a grafy. Dále je možné zmínit některé další praktické využití teorie her, například v ekonomii.

Rozsah grafických prací:

Rozsah pracovní zprávy: **40**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

MAŇAS, Miroslav. Teorie her a konflikty zájmů. Vyd. 1. V Praze: Vysoká škola ekonomická v Praze, 2002, 114 s. ISBN 80-245-0450-2
MAŇAS, Miroslav. Teorie her a její aplikace. 1. vyd. Praha: Stát. nakl. techn. lit., 1991, 278 s. Teoretická knižnice inženýra. ISBN 80-030-0358

Vedoucí bakalářské práce: **Mgr. Alena Pozdílková, Ph.D.**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **31. října 2018**

Termín odevzdání bakalářské práce: **12. května 2019**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

Ing. Lukáš Čegan, Ph.D.
pověřený vedením katedry

V Pardubicích dne 20. března 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 17. 1. 2019

Jakub Kolařík

PODĚKOVÁNÍ

Rád bych poděkoval hlavně své vedoucí Mgr. Aleně Pozdílkové, Ph.D. za pozitivní přístup a trpělivé vedení při návrhu a zhotovování této práce.

ANOTACE

Na začátku této práce jsou obsaženy a vysvětleny základní pojmy a principy teorie her. Stručně se zabývá antagonistickými hrami s konečně i nekonečně mnoho strategiemi, neantagonistickými konflikty pro kooperativní i nekooperativní strategie. Poté se věnuje řešení umělé inteligence pro piškvorky, reversi a izolaci za použití minimax algoritmu, alfa-beta ořezávání, heuristiky, iterativního prohlubování a transpoziční tabulky. Pro tyto hry také obsahuje aplikaci, ve které je možné si proti AI zahrát jednotlivé hry.

KLÍČOVÁ SLOVA

teorie her, strategie, AI, minimax, optimalizace minimaxu, piškvorky, reversi, izolace.

TITLE

Theory of games and its applications.

ANNOTATION

At the beginning, this thesis explains the basic concepts and principles of game theory. It briefly explains antagonistic games with final or infinite strategies, non-antagonistic conflicts for cooperative or non-cooperative strategies. Later, it focuses on creating artificial intelligence for tic-tac-toe, reversi and isolation using minimax algorithm, alpha-beta pruning, heuristic, iterative deepening and transposition table. Those games are implemented in application, where i tis possible to play against AI individual games.

KEYWORDS

game theory, strategy, AI, minimax, minimax optimalization, tic-tac-toe, reversi, isolation.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

GUI	Grafické uživatelské rozhraní (Graphical User Interface)
AI	Umělá inteligence (Artificial Intelligence)
WPF	Windows Presentation Foundation

ÚVOD

Před několika tisíciletími byla velmi probírána myšlenka umělé inteligence, kterou by mohl stvořit sám člověk. Začalo to mytickými konstrukty, které byly postaveny pro ochranu chrámů, a táhlo se to až přes oživlé golemy z hlíny a alchymistické homunkuly. Už před sestrojením prvního počítače ale vznikaly reálnější představy o umělé inteligenci a o její možné roli v různých odvětvích. Nebyly to však představy o autonomně myslících entitách, ale o specializovaných algoritmech, které by byly schopny „inteligentního“ rozhodování ve specializovaných systémech. Se sestrojením počítače začaly vznikat první herní AI, které dokázaly hrát proti člověku. Úplně první hrou byly v roce 1950 piškvorky na 3x3 poli, kde dokonce bylo možné měnit obtížnost protivníka.

Tato bakalářská práce se bohužel nezabývá hlubokou filozofií myšlení a ani se nevěnuje v této době velmi modernímu a populárnímu hlubokému učení nebo strojovému učení, nevěnuje se žádným neuronovým sítím, ani samoučícím algoritmům.

Cílem bakalářské práce je sestrojit základní umělou inteligenci (AI) pro hraní několika deskových her, jako jsou například piškvorky, reversi, nebo izolace, která bude vycházet z rozsáhlé teorie her, přesněji z odvětví antagonistických her. Pro vytvoření AI bude použit zápis hry v explicitním tvaru, který poté bude procházen algoritmem zvaným minimax. Jelikož některé hry mají ohromný počet možných tahů, bude třeba zavést i optimalizační metody minimaxu, jako je například alfa-beta ořezávání, iterativní prohlubování, heuristika nebo transpoziční tabulka.

Výsledkem práce je také aplikace kolekce her, které si uživatel bude moct zahrát proti umělé inteligenci.

1 TEORIE HER

Teorie her je disciplínou aplikované matematiky, která se věnuje zkoumání konfliktů a problematiky rozhodovacích situací, a poskytuje nám teoretický i praktický nástroj pro implementaci různých řešení. Na různých hrách a jejich scénářích bude ukázána tvorba strategií pomocí výše zmíněné disciplíny, budou nalezeny optimální postupy pro výhru, či pro implementaci soupeře různých obtížností. Teorie her zde poskytne matematické definice, algoritmy a bude alfou i omegou celé bakalářské práce. Také jsou zde řešeny různé strategie pro mnoho her, které jsou teoreticky rozebrané a doplněné částmi algoritmů, řešící chování protivníků.

1.1 Historie Teorie her

Teorie her pomalu vznikala na počátku dvacátého století, kdy matematici začali řešit, jak matematicky popsat konflikty a rozhodovací situace. Historicky se jako oficiální vznik teorie her bere vydání „On the Theory of Strategy“ od John von Neumanna roku 1928, po kterém následovalo napsání „Theory of Games and Economic Behaviour“ opět od Neumanna, ale tentokrát ve spolupráci s Oskarem Morgensternem v roce 1944. V této knize byly shrnuty veškeré dřívější poznatky z teorie her, které ovšem byly také rozšířené o další teorie a dodatky. Jelikož se však dříve pohlíželo na teorii her jako na řešení kooperativních problémů, tato kniha se také zabývala situacemi s prvky konfliktu a položila základ maticovým hrám a obecně se stala alfou a omegou dalšího rozšiřování teorie her.

1.2 Základní pojmy

Zde si vymezíme základní pojmy, které budou využity v dalších kapitolách.

Hra

Značí kompletní sestavu pravidel, hráčů, všech jejich možných strategií. Obecně tento výraz zaobaluje řešený konflikt. [1]

Hráč

Instituce, osoba či mechanismus, který může svými rozhodnutími ovlivňovat výsledky hry. Hráčů je vždy konečný počet, množinu hráčů budeme označovat $Q = \{1, 2, \ldots, N\}$. [2]

Strategie

Označuje jednu možnost rozhodnutí hráče. Každý hráč má určitou množinu strategií X, která obsahuje všechny jeho možné strategie. Množina rozhodnutí hráče i se bude označovat jako X_i a bude se nazývat prostorem strategií hráče i.

Výplatní funkce

Vyjadřuje důsledek použití strategie x pro hráče i. Značí se M_i (x). Jestli výplatní funkce pro daného hráče dosáhne kladné hodnoty, výsledek se bere jako zisk hráče i, pokud je výsledek záporný, daný hráč utrpěl ztrátu o hodnotě | M_i (x) |. [2]

Výhra

Zisk nebo ztráta hráče ve hře, odvíjí se od výplatní funkce.

Optimální strategie

Je pro daného hráče nejvýhodnější rozhodnutí.

Inteligentní hráč

Hráč, který logicky volí své strategie tak, aby dosáhl nejvyšší výhry. Pokud jsou ve hře přítomní alespoň dva inteligentní hráči, jedná se o konflikt. [2]

Neinteligentní hráč

Reprezentace vlivu náhodného mechanismu ve hře. [2]

1.3 Matematický model her

Teorie her umožňuje zapisovat konflikty vícero zápisy. Patří mezi ně následující.

Hra v normálním tvaru

Je tvořena hráči v množině Q o počtu 1 … N, prostory strategií každého hráče a výplatními funkcemi pro každého hráče. Hráči v této hře vždy budou inteligentní. Takovou hru lze zapsat jako $\{Q; X_1, \ldots, X_N; M_1(x), \ldots, M_N(x)\}$. [2]

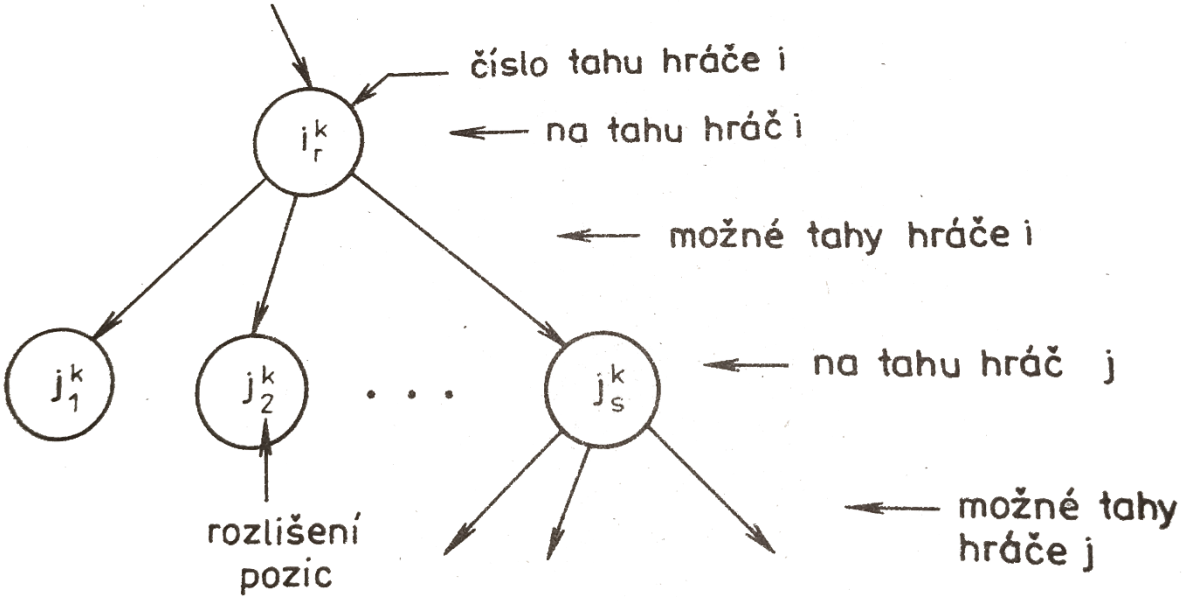
Hra ve tvaru charakteristické funkce

Tento zápis se převážně používá pro definici kooperativní hry, jde o funkci v, která je definovaná pro všechny podmnožiny množiny Q, tedy množiny hráčů (zde jde například o

firmy nebo jedince, mezi kterými vzniká možnost spolupráce). Hodnotou funkce je poté zisk koalice ať už kladný, nebo záporný. [1]

Hra v explicitním tvaru

Je popis využívaný pro hry, kde je nutné znázornit postupné střídání výběru strategií jednotlivých hráčů. Tato struktura se zapisuje do grafu, který je zadaný hranami a vrcholy. Rozhodování se potom znázorňuje k-cestným stromem a také se na něm dá implementovat. Poté se jedná o *strom hry*, ten je poté významově stejný jako hra v explicitním tvaru. Takový rozhodovací strom lze poté zakreslit následovně.



Obrázek 1.1 – Popis rozhodovací situace pomocí stromu hry [2]

Každá úroveň stromu tak označuje jiného hráče, hrany označují možné strategie (různé tahy). Při provedení tahu se stane aktuální jenom ta část stromu, která je dosažitelná z aktuálního uzlu a zbytek stromu reprezentuje situace, kterých v daném stádiu hry už nelze dosáhnout. Ukončení hry je ve stromu značeno listy, tedy koncovými uzly (nemají další potomky), které obsahují informaci o výsledku hry. Později se podíváme na konkrétní aplikace takového zápisu v explicitním tvaru. [2]

1.4 Hra s konstantním součtem

Jedná se o konflikt, ve kterém je výhra jednoho hráče přímo ovlivněna hráčem druhým. Tato výhra má konstantní celkovou hodnotu, o kterou se oba hráči určitým způsobem podle pravidel hry rozdělí. Jednoduše lze takovou situaci popsat následovně.

$M_1(x, y) + M_2(x, y) = konstanta$

Kde M_1 je výplatní funkce prvního hráče a M_2 hráče druhého, x označuje zvolenou strategii prvního hráče a y zvolenou strategii hráče druhého.

Taková hra se dá dále převést na *hru s nulovým součtem*, kde výsledná konstanta bude rovna nule. To lze zapsat jako

$M_1(x, y) = -M_2(x, y)$.

Nadále je pro nás jednodušší zapisovat a sledovat pouze výplatní funkci prvního hráče, jelikož výhra hráče druhého bude vždy stejná, akorát s opačným znaménkem. [1, 2]

1.5 Antagonistické hry

Neboli konflikty, budou značit hry, ve kterých se objevují dva inteligentní hráči, hra má nulový součet a je v normálním tvaru. V řešení takových konfliktů se budeme primárně zajímat o nalezení rovnovážných (optimálních) strategií každého inteligentního účastníka.

1.5.1 Maticové hry

Předpokládejme dva inteligentní hráče. Každý má finální počet strategií, přičemž výběr strategie neovlivňuje částku hry. Množinu strategií prvního hráče nazveme X a množinu strategií hráče druhého Y. Takto zadané prostory strategií generují při maticové hře za pomoci kartézského součinu *matici hry* neboli *výplatní matici*. Hodnota matice na souřadnicích strategie x a y tedy udává výplatní funkci $M_1(x, y)$. V následující tabulce lze vidět zmíněný zápis, řádky určují strategie prvního hráče, sloupce strategie hráče druhého, a na příslušných indexech jsou výplatní funkce označující výhru prvního hráče. Výhra druhého hráče je zbytková hodnota z částky hry, jelikož se jedná o hru s nulovým součtem. [3]

	y_1	y_2	y_3
x_1	$M(x_1, y_1)$	$M(x_1, y_2)$	$M(x_1, y_3)$
x_2	$M(x_2, y_1)$	$M(x_2, y_2)$	$M(x_2, y_3)$
x_3	$M(x_3, y_1)$	$M(x_3, y_2)$	$M(x_3, y_3)$

Tabulka 1.1 – Obecný popis základní maticové hry

V tuto chvíli bychom si měli vhodně definovat optimální strategii. Optimální strategie má podle [3] následující vlastnost: „*pokud kterýkoliv z hráčů zvolí nějakou jinou strategii než tyto*

své optimální, sníží si výhru.“. Takovému řešení se říká *Nashova rovnováha*. Optimální strategie budeme značit jako x^* a y^* a pro takové strategie tedy platí, že

$$M_1(x, y^*) \leq M_1(x^*, y^*),$$

$$M_2(x^*, y) \leq M_2(x^*, y^*).$$

Slovně jde o to, že výplatní funkce prvního hráče při použití jiné než optimální strategie a za předpokladu, že protivník použije svou optimální strategii je nižší (vyhraje méně), než kdyby ve stejné situaci použil svou optimální strategii. Stejně to platí pro druhého hráče. [2]

Podívejme se na jeden příklad takové maticové hry. Mějme následující matici.

$$\begin{bmatrix} -2 & 3 & 3 & -5 \\ 0 & -3 & -2 & 1 \\ 3 & 2 & 5 & 2 \end{bmatrix}$$

Druhý hráč má strategie $y_1 \ldots y_4$ a první hráč pouze $x_1 \ldots x_3$. Tato hra má nulový součet, takže výhra druhého hráče je záporná hodnota v matici. Při pohledu na tuto matici můžeme zjistit, že některé strategie jsou na první pohled pro hráče nevýhodné. Pokud strategie má všechny hodnoty menší než libovolná jiná, tak se jedná o *silně dominovanou strategii*, a pokud má hodnoty menší nebo rovny, tak se jedná o *slabě dominovanou strategii.* [1]

V tomto případě můžeme prohlásit, že třetí strategie druhého hráče je slabě dominovanou strategií a pro přehlednost jí můžeme odstranit.

$$\begin{bmatrix} -2 & 3 & -5 \\ 0 & -3 & 1 \\ 3 & 2 & 2 \end{bmatrix}$$

Optimální strategie lze v tomto případě najít jednoduše. Pokud existuje prvek, který je na řádku nejmenší a zároveň je největší na sloupci, jedná se o *sedlový prvek*, který znázorňuje výplatní funkci pro optimální strategie x^* *a* y^*. [2]

Při použití předchozí věty zjistíme, že optimální strategie pro tuto hru jsou x_3 *a* y_3 a při jejich použití bude výhra prvního hráče 2 a druhého -2.

Existují však případy, ve kterých nelze takto jednoduše určit sedlový prvek matice. Pro takové situace je vhodné použít *smíšené strategie*, které rozšiřují matici hry o pravděpodobnosti, anebo využít *lineárního programování* pro získání optimálních strategií.

1.5.2 Antagonistické hry s nekonečně mnoho strategiemi

Tyto hry mají většinou takové množství strategií každého hráče, že nelze počítat jejich matice her, natož je nějakým vhodným způsobem zobrazit. Množiny strategií hráčů jsou nekonečné,

ale musí být spočetné. V některých případech lze předpokládat průběh výplatních funkcí v matici a je možné určit optimální strategie na malém počátečním vzorku hry. [2]

V ostatních případech je ovšem nutné uplatnit rozšíření matice o smíšené strategie, nebo hledat maxima *úžlabinové funkce* a minima *hřebenové funkce.* [2]

1.6 Neantagonistické hry

Neantagonistickým konfliktem se myslí model hry, ve kterém jsou přítomni dva inteligentní hráči, hra je v normálním tvaru, ale nemá konstantní součet. To znamená, že nám nebude stačit jednoduchá matice, ale bude potřeba matic dvou, jelikož každý hráč má svou výplatní funkci, a hodnotu jedné funkce nebude možno dopočítat jen za pomoci funkce druhé. Tyto dvě matice však budeme zapisovat do jedné způsobem, že každá hodnota matice bude množinou dvou hodnot. [2]

Příkladem může být třeba spor dvou institucí. Každá z nich má tři strategie jednání s tou druhou.

1.6.1 Neantagonistické hry s nekooperativními strategiemi

Do této kategorie spadají hry neantagonistické hry, ve kterých nelze uzavírat dohody mezi hráči, ti se tedy nemohou vzájemně dohodnout na tom, jaká strategie je pro oba nejvíce výnosná.

1.6.2 Neantagonistické hry s kooperativními strategiemi

Zde patří neantagonistické hry, ve kterých lze uzavírat dohody mezi hráči ještě před zvolením strategií. Takové hry se dělí na *kooperativní hry s přenosnou výhrou*, kde se hráči mohou domluvit na společné strategii a zároveň si mohou přerozdělit výhru, a na *kooperativní hry s nepřenosnou výhrou*, ve kterých si mohou hráči domlouvat strategie, ale konečnou výhru si nemohou přerozdělit.

2 TEORIE PRO APLIKACI

Aplikace pro tuto bakalářskou práci se bude věnovat vytvoření schopné umělé inteligence pro hraní různých deskových her. Bude obsahovat základní GUI a několik příkladů deskových her, které bude možné hrát proti AI. Zde si popíšeme teorii použitých algoritmů a možných optimalizací.

2.1 Minimax algoritmus

Tento algoritmus přímo vychází z dříve probrané teorie her. V tomto případě se jedná o reprezentaci hry s nulovým součtem v explicitním tvaru, tedy o strom hry, který je vytvořen a poté algoritmem prohledáván, dokud není nalezen optimální tah hráče. Většina her má však bohužel velmi rozsáhlý strom hry, a tedy na ně nejde v reálném čase tento algoritmus aplikovat. Minimax prochází strom hry do hloubky, dokud nenarazí na terminální uzel (list stromu), který nese výsledek hry, což je číselná reprezentace výhry, prohry nebo remízy vzhledem k zvolené hře. Když minimax dojde k listu stromu, začne střídavě vybírat vždy optimální strategii pro hráče, který je zrovna na tahu. Je třeba si také určit, který hráč se bude snažit maximalizovat a který minimalizovat hodnotu výsledku hry, ta se může například pohybovat v hodnotách -1 pro prohru prvního hráče (výhra druhého), 0 pro remízu a 1 pro výhru prvního hráče (prohra druhého). Z takového rozestavení je tedy očividné, že hráč 1 se bude snažit ve svém tahu hodnotu výhry vždy maximalizovat a hráč 2 bude ve svém tahu naopak vybírat tahy s minimální hodnotou výhry. Minimax tedy musí od terminálních uzlů stromu probublat až ke kořeni, přičemž předpokládá, že každý hráč si vybere pro něj nejvíce optimální strategii (maximum nebo minimum hodnoty). Každý hráč by odchýlením od této strategie mohl maximálně poškodit své šance na výhru, a tak zde můžeme vidět aplikaci již zmíněné Nashovy rovnováhy.

Každý takový herní strom má určitý větvící faktor, tedy možný počet tahů z jednoho uzlu (počet hran), a hloubku stromu (délka hry v počtech tahů obou hráčů). Z těchto dvou hodnot jsme schopni vypočítat počet terminálních uzlů jako

$$n = b^d,$$

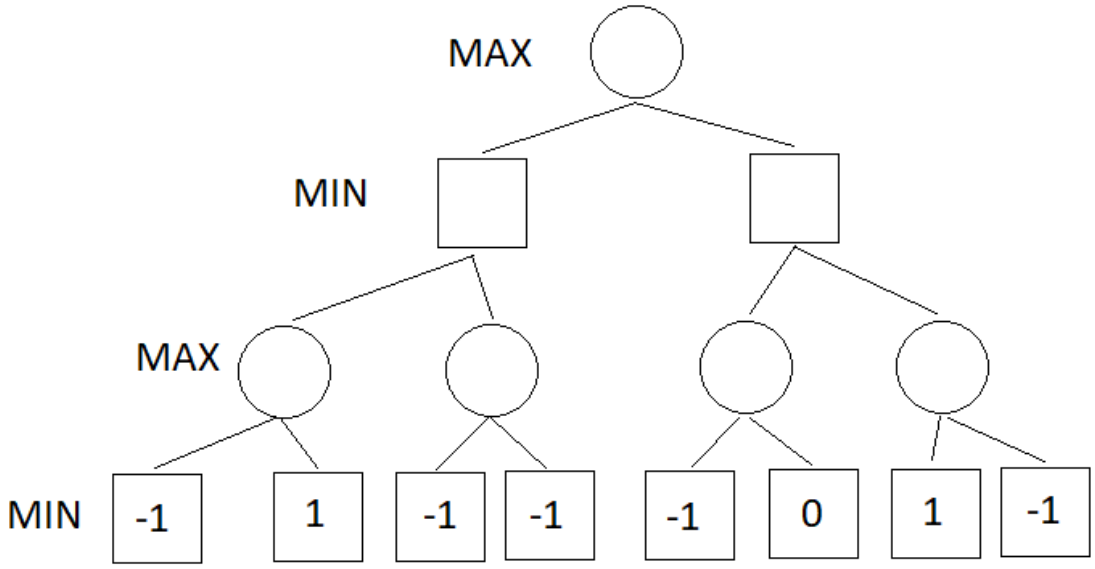
kde n je počet terminálních uzlů (listů), b je větvící faktor stromu a d je hloubka stromu.

Znát počet terminálních uzlů je pro nás velmi důležité, jelikož jen v případě takového uzlu je nutno počítat vyhodnocující funkci, která zhodnotí stav hry a vrátí hodnotu výhry.

Pro zajímavost, Patrick Winston ve své přednášce na MIT [4] ukázal, že pokud bychom vzali v úvahu aplikaci minimax algoritmu na šachy s průměrným větvícím faktorem 10 a pravidlem

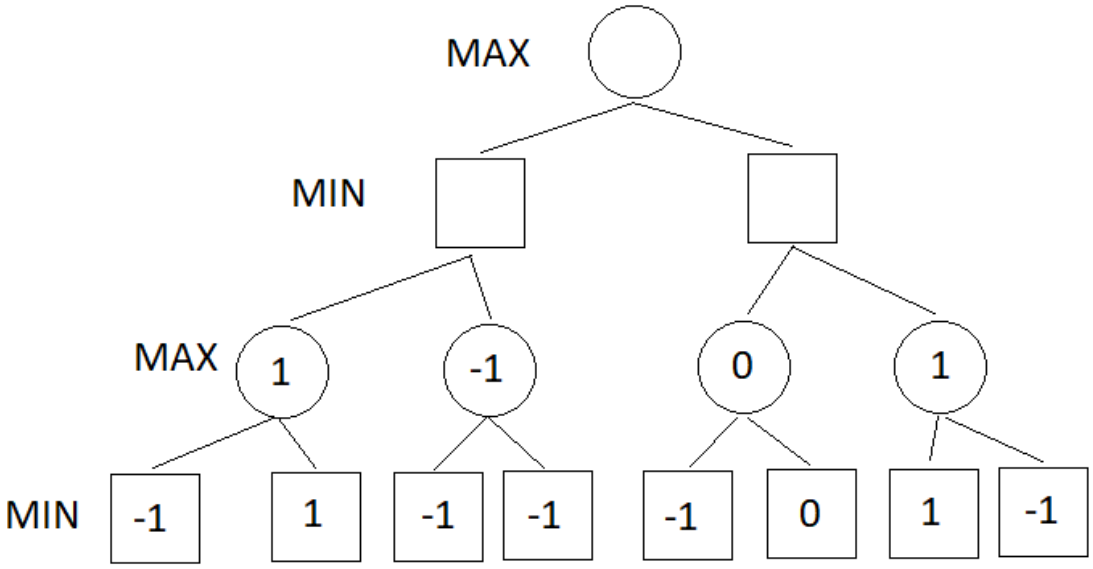
60 tahů pro každého hráče, došli bychom k 10^{120} terminálních uzlů. Ve známém vesmíru se nachází přibližně 10^{80} atomů, jeden rok má $\pi \times 10^9$ nanosekund, a historie vesmíru se datuje 14×10^{10} let. To je dohromady přibližně 10^{99}, což znamená, že kdyby všechny atomy ve vesmíru prováděly jeden výpočet každou nanosekundu od počátku vesmíru, tak by nám v tuto chvíli stále chybělo 21 řádů výpočtů.

Pro názornější představu tohoto algoritmu se můžeme podívat na následující ukázku. Mějme hráče 1 (MAX), který se bude vždy snažit maximalizovat svou výhru, a hráče 2 (MIN), který ji bude minimalizovat. Následující hra bude mít větvící faktor dva a hloubku stromu tři. Hra bude mít tři ukončovací stavy, -1 pro prohru prvního hráče, 0 pro remízu a 1 pro výhru. Vytvořený strom hry po vypočítání funkce v terminálních uzlech bude vypadat následovně.



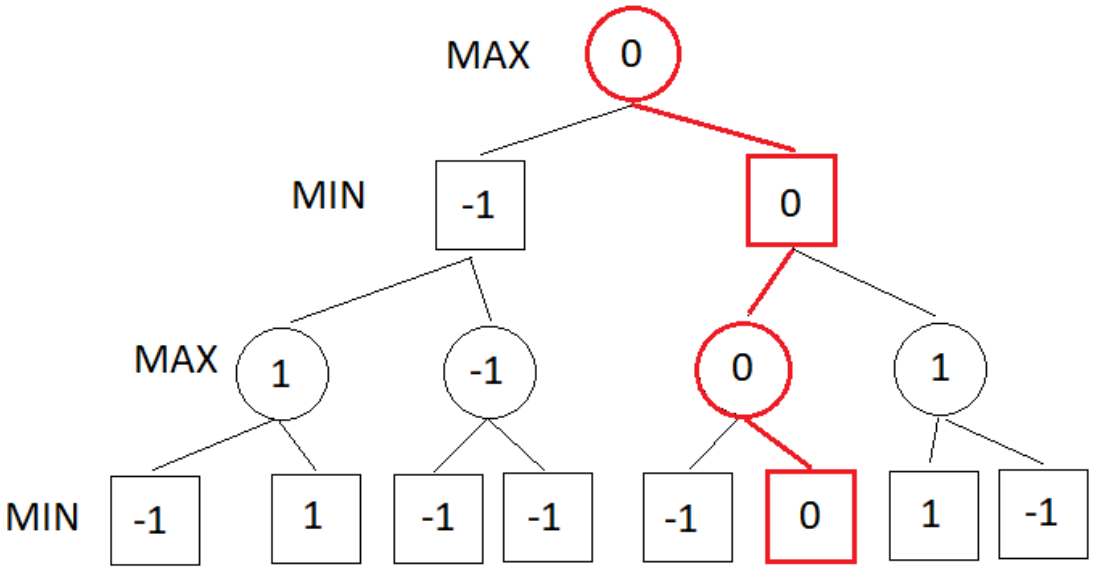
Obrázek 2.1 – Ukázka minimax 1

V dalším kroku bude minimax vybírat vždy maximální hodnotu, jelikož hráč MAX bude volit pouze tahy pro něj nejvíce optimální.



Obrázek 2.2 – Ukázka minimax 2

Poté je na řadě MIN, ten bude vybírat minimální hodnoty, které se k němu dostaly, jelikož mu zajišťují nejvyšší možnou výhru. Pak nastane poslední volba hráče MAX, ten opět vybere nejvyšší hodnotu a dostane se tak ke své nejvíce optimální strategii, od které když se odchýlí, tak si sníží šance na výhru. Finální stav vyhodnoceného stromu vypadá následovně.



Obrázek 2.3 – Ukázka minimax 3

Hráč MAX tedy dosáhl remízy. Pokud by se však MIN odchýlil od své optimální strategie, stejnou metodou by MAX dosáhl výhry. Červená trasa zvýrazňuje průběh hry, kdy oba hráči volí optimální strategie.

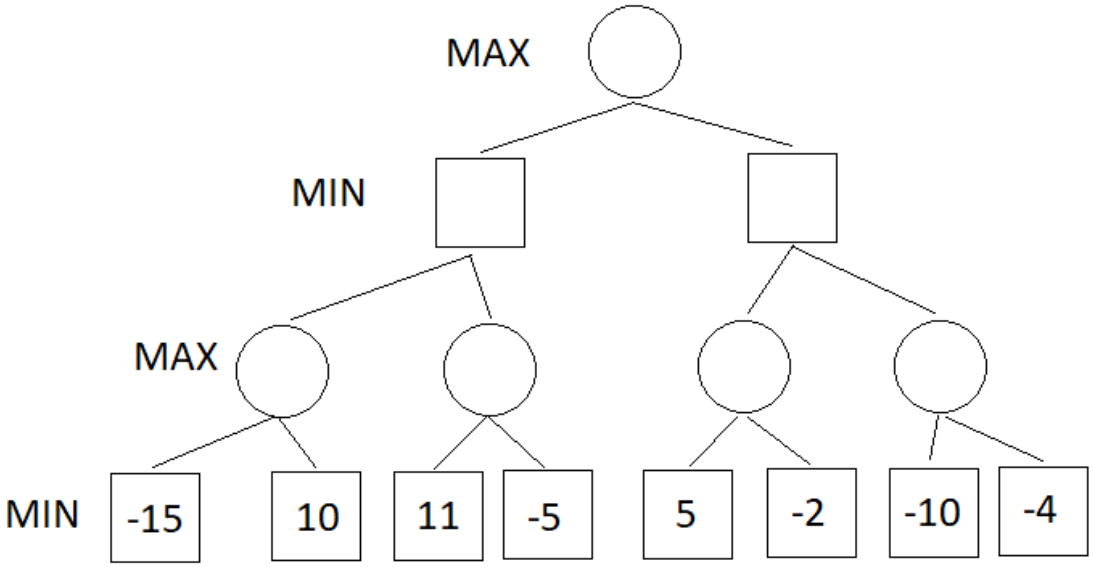
V praxi minimax funguje tak, že při každé potřebě získání tahu AI dostane stav hry, a od toho dopočítá všechny možné tahy, dokud nenarazí na terminální uzel, který vyhodnotí a postupně probublá optimální strategii až ke kořenu. Postupným klesáním možných tahů se tedy zmenšuje strom hry a výpočet probíhá mnohem rychleji s průběhem hry.

[4, 5]

2.2 Alfa-beta ořezávání

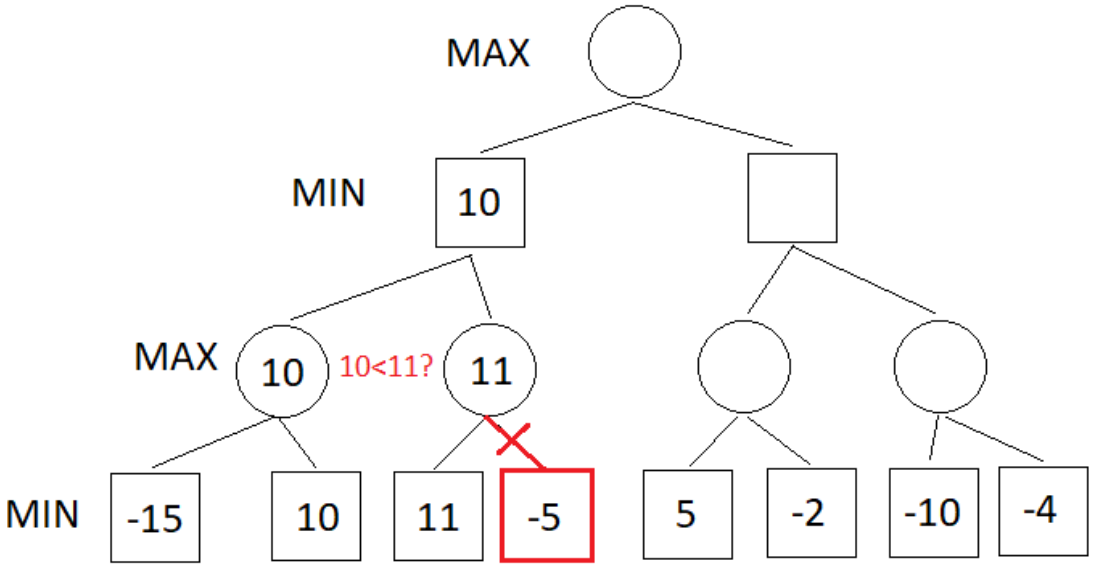
Je jedna z možností optimalizace minimax algoritmu, jde o praktiku zmenšování šířky stromu, která vysoce urychluje procházení herního stromu. S každou iterací minimaxu se předávají parametry alfa a beta, podle kterých se poté odsekávají větve stromu. Při každém prohledání sousedních uzlů se podle aktuálního hráče ptáme, zda větev může poskytnou výsledky, které by mohly ovlivnit výběr MAX nebo MIN hráče.

Jako příklad se můžeme podívat na následující strom, který už však má jiné hodnoty výhry. Čím větší číslo je, tím výhodnější je stav po MAX hráče a čím menší, tím je stav výhodnější pro MIN hráče.



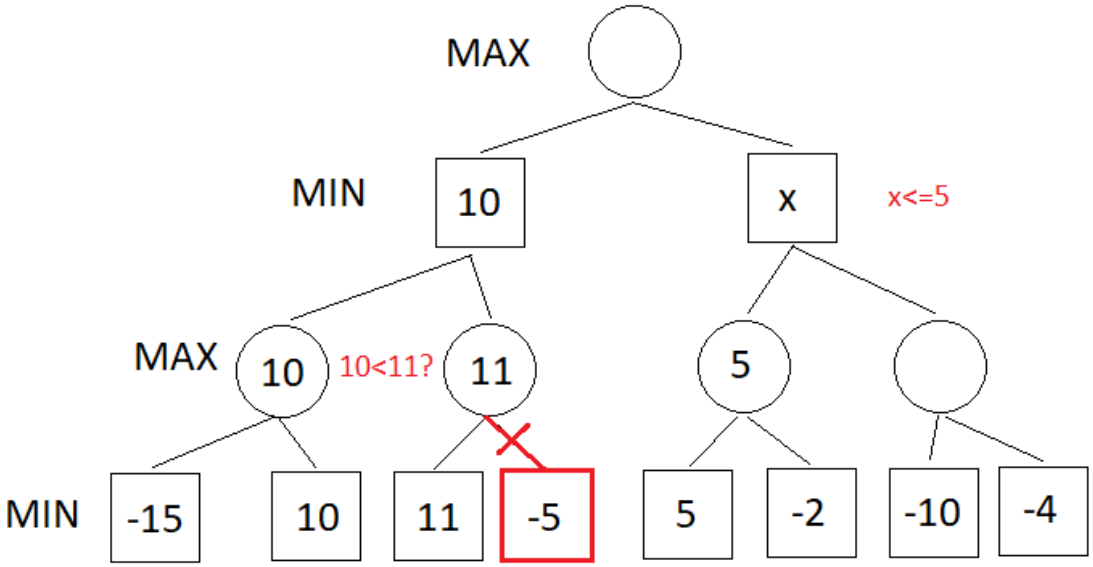
Obrázek 2.4 – Ukázka alfa-beta 1

První výběr hodnoty je nutný, MAX vybírá z čísel -15 a 10, vybere 10. Poté se přesune do dalšího uzlu, spočítá hodnotu 11 a bude porovnávat. Jelikož v další úrovni stromu má MIN vybírat z hodnoty 10 a částečné hodnoty 11, víme, že si určitě vybere 10. Do druhého listu pravé větve tedy nemusíme vůbec vstupovat, jelikož aby si toto druhé číslo vybral MAX, muselo by být větší než 11. Kdyby bylo, MIN by potom vybíral z čísla 10 a z čísla většího než 11, což znamená, že si vždy vybere hodnotu 10. Tento stav je zobrazen následovně.



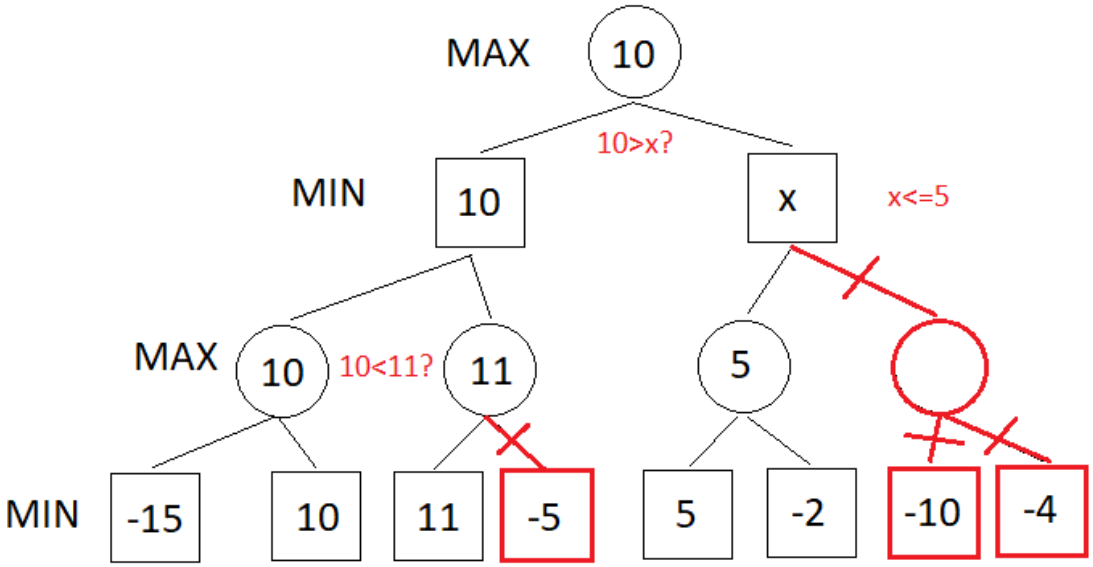
Obrázek 2.5 – Ukázka alfa-beta 2

Červeně se značí uzly a hrany, které byly oříznuty a není třeba je procházet. V tomto stádiu algoritmus projde do pravého podstromu a vybere hodnotu 5 z čísel 5, -2. V tuto chvíli víme, že MIN vybere číslo 5 nebo menší.



Obrázek 2.6 – Ukázka alfa-beta 3

Teď se už jen porovná hodnota 10 a *x*. Jelikož *x* je menší nebo rovno pěti, MAX hráč vždy vybere hodnotu 10.



Obrázek 2.7 – Ukázka alfa-beta 4

V tomto případě došlo k oříznutí tří listů z osmi což je přibližně 38 % zrychlení minimaxu.

Alfa-beta ořezávání má různou míru oříznutí. Velmi důležitým faktorem je pořadí výpočtu vyhodnocujících funkcí. Pokud by se nám nějakým způsobem podařilo seřadit prvky stromu, úspěšnost této optimalizační metody by mohla být o dost vyšší.

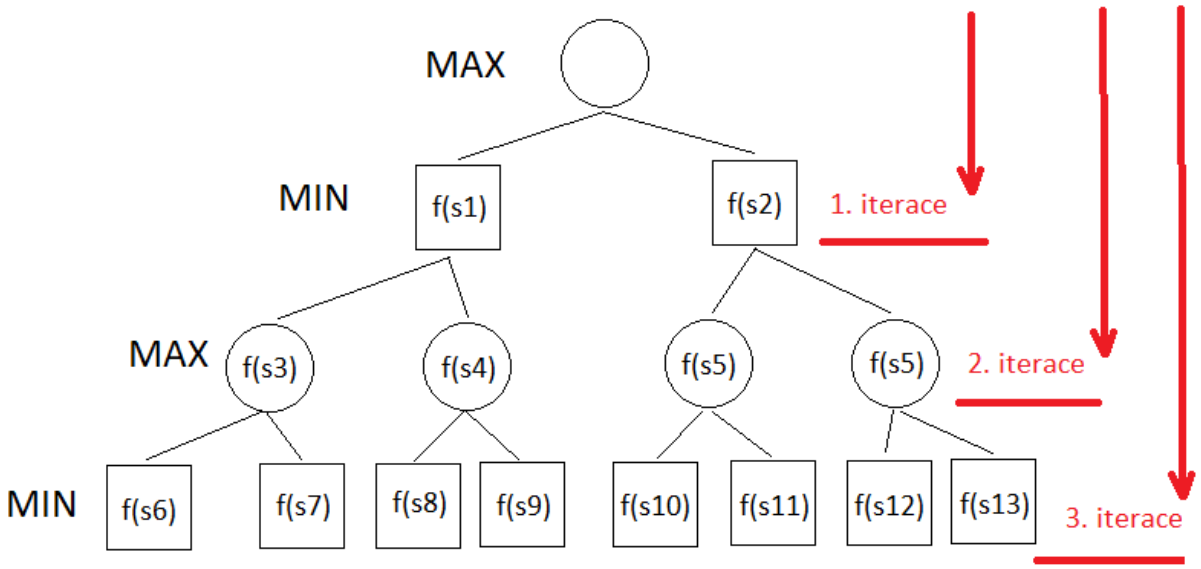
[4, 5]

2.3 Iterativní prohlubování

Metoda *iterative deepening*, neboli postupné prohlubování, řeší problém obrovské časové složitosti aplikace minimaxu na velmi rozsáhlé stromy. Dle dříve zmíněného příkladu hry šachů je prakticky nemožné vypočítat všechny terminální uzly stromu a vrátit tah, který by byl zaručeně optimální strategií a vedl by k výhře. Principem je procházení herního stromu do předem stanovené hloubky. Strom se vždy postupně projde od první hloubky do konečné hloubky, ale v každé iteraci minimaxu se uloží průběžný nejoptimálnější tah a přepíše se ten předchozí. Tento postup se volí z důvodu časového kritéria. Lze totiž nastavit časový limit pro hledání tahu, a když tento čas dojde, algoritmus je schopný vrátit tah z nejhlubší iterace, kterou stihl vypočítat. Jelikož však většinou nejsme schopni projít strom až k terminálním uzlům a vyhodnotit jejich výsledek, je třeba zavést průběžnou hodnotící funkci stavu hru pro každý uzel. Taková funkce se nazývá heuristická funkce a podíváme se na její řešení později.

Protože použití této metody znamená procházet strom od začátku tolikrát, kolik úrovní má, dalo by se čekat, že velmi zpomalí celkový algoritmus, to ale odpadá při použití transpoziční tabulky a paradoxně při správném seřazení prohledávaných uzlů je možné prohlídku ještě zrychlit.

Procházení stromu takovou metodou vypadá následovně.



Obrázek 2.8 – Ukázka iterativního prohlubování

Hodnota uzlu se počítá vždy až v maximální požadované hloubce, a až z té si vybírají hráči MIN a MAX své tahy.

[4, 5]

2.4 Heuristická funkce

Používá se při vyhodnocení uzlu, který není listem herního stromu. Jde o co nejlepší zhodnocení stavu hry pro oba hráče, což může být velmi obtížné. Pro každou hru je nutné vytvořit vlastní funkci, jelikož každá hra má jiná pravidla a strategie. Protože se věnujeme hrám s nulovým součtem, hodnoty takové funkce se budou pohybovat v hodnotách $\langle -x, x \rangle$. Tedy nulová hodnota bude stav neutrální, maximální x bude výhra prvního hráče a minimální x bude výhra hráče druhého. Velikost tohoto intervalu bude různá na základě požadavků každé hry.

[5]

2.5 Transpoziční tabulka

Využití takové tabulky funguje při iterativním prohlubování. Jde o vytvoření vhodné hashovací funkce, která je schopna převést stav hrací plochy s největší pravděpodobnosti na unikátní hash. Když je vyhodnocen stav hry v určité fázi, dostane se z něj hash, a na index hashe se do tabulky uloží vypočítaná hodnota. Tím pádem, když děláme více hloubkových iterací minimaxu, můžeme jít na index hashe stavů a rovnou vytáhnout hodnotu heuristické funkce z tabulky. Takto se může ušetřit velké množství výpočetního času.

3 HRY

V následující části budou popsány implementované hry, jejich pravidla a možnost využití různých algoritmů pro běh AI protivníka.

3.1 Piškvorky

Podle Claudie Zaslavsky [6] se historie piškvorek datuje do roku 1300 před naším letopočtem, přesněji do starověkého Egypta, kde byly nalezeny piškvorky na střešních deskách. Další verze piškvorek byly nalezeny na území Římské říše, a postupem času se objevily po celém světě.

Jedná se o jednoduchou deskovou hru, která se hraje se na poli o třech řádcích a sloupcích, hráči se střídají a zapisují do mřížky na střídačku křížky a kolečka. První hráč, který dosáhne tří stejných znaků v horizontálním, vertikálním, nebo diagonálním směru vyhrává.

Jelikož se piškvorky hrají na devět polí, mají relativně málo stavů hry. Průměrný větvící faktor je čtyři, hloubka stromu hry je 9, to znamená, že herní strom má 4^9 listů. Piškvorky jsou ovšem hra symetrická, a tak když vezmeme v potaz rotace a zrcadlení herního pole, zůstane nám 26 830 možných terminálních stavů hry [7].

Protože strom této hry je relativně malý, pro AI, která vždy bude hrát podle optimální strategie, stačí aplikovat surový minimax algoritmus. Není teda potřeba použít iterativní prohlubování, heuristickou funkci ani alfa-beta ořezávání i když to by značně zkrátilo počet výpočtů. Z tohoto hlediska jsou piškvorky hrou vyřešenou, jelikož při použití optimální strategie obou protivníků vždy dojde k remíze.

Také zde zkusíme vyřešit verzi piškvorků na poli deset na deset s požadavkem pěti po sobě jdoucích znaků na výhru, což může být docela komplexní. Problém je, že strom takové hry je velmi rozsáhlý, bude třeba využit postupné prohlubování. Velmi obtížným úkolem pak taky bude vytvoření heuristické funkce, která bude dostatečně vhodně hodnotit stavy hry.

3.2 Reversi

Reversi je desková hra z konce 19. století. Vznikla v Anglii a po druhé měla rozmach v Japonsku. Často se také objevuje pod názvem Othello.

Pravidla hry jsou jednoduchá. Hrací plocha se skládá z osmi řádků a sloupců, tedy dohromady 64 polí. Hraje se s oboustrannými tokeny, z jedné strany jsou bílé a z druhé černé. Začínající sestava je umístění dvou bílých a dvou černých tokenů doprostřed hrací plochy tak, aby stejná barva byla na diagonále. Hráč, který je na tahu pak vezme svůj token, a musí ho umístit na hrací plochu tak, aby izoloval posloupnost nepřátelských tokenů z obou stran svou

barvou (lze hrát horizontálně, vertikálně i diagonálně, lze jedním tahem využít i všechny tři směry). Když tak učiní, všechny izolované tokeny se převrátí na druhou barvu a hráč tak získá další kameny. Pokud hráč nemá žádné možné tahy, předává kolo protivníkovi. Když ani ten nemůže táhnout, hra končí a vyhrává ten, kdo má více tokenů své barvy na hrací ploše. Pokud taková situace nenastane, hra pokračuje, dokud není zaplněno všech 64 polí. Poté se tokeny opět spočítají a vyhrává vyšší počet. Posledním pravidlem je, že hráč musí táhnout, pokud existuje alespoň jeden validní tah.

V této hře je tolik možných tahů a stavů hry, že je nemožné procházet celý strom hry minimaxem. Proto je třeba aplikovat alfa-beta ořezávání, které může dramaticky snížit náročnost, ale přesto strom nejde prohledat celý. Je tedy nutné využít heuristiky.

Vytvořit ideální heuristickou funkci však není vůbec jednoduché. První nápad by mohlo být hodnotit stav hry podle počtu kamenů. Je-li hráč 1 MAX a má bílou barvu, tato funkce by mohla vracet počet bílých kamenů mínus počet černých kamenů. AI by teda vybírala tahy, které by jí do budoucna zařídily nejvíce kamenů. Toto řešení může znít jako optimální, ale rychle narazí, protože reversi je hra, ve které lze s dobrou pozicí kamenů otočit v posledním kole velké množství kamenů a zajistit si tak výhru, i když v předposledním kole měl drastickou převahu protivník. Takto implementovaná funkce vytvoří protivníka, který působí, že hraje inteligentně, ale obtížnostně moc vysoko nesahá.

Další možnost je využití síly pozice. Na hrací ploše se dají najít místa, které jsou pro hráče mnohem výhodnější než ostatní. Například všechny rohy jsou velmi strategickou pozicí, jelikož kámen v rohu nejde otočit, a to poté dává příležitost pro otáčení kamenů na okraji plochy a diagonále ke středu. Naopak pozice sousedící s rohem plochy jsou nevýhodné, protože dávají možnost soupeři je přeskočit a zabrat tak roh. Taková strategie už je o něco sofistikovanější než předešlá. Pro implementaci stačí vytvořit srovnávací matici, která se vždy porovná se stavem hry a podle hodnot na každém poli se hráčům přiřadí body.

Posledním řešením v této aplikaci je spojení obou uvedených strategií. Funkce bude odečítat počty kamenů, a zároveň bude hodnotit zabrané pozice. Problémem takové kombinace funkcí je nalezení správných parametrů pro adekvátní škálové vyrovnání obou metod.

[8]

3.3 Izolace

Je jednoduchá hra s různými pravidly. V tomto případě se budeme zabývat verzí, která se hraje na poli sedmi řádků a sedmí sloupců a jako figurky se používají šachoví koni. Každý hráč má

svého koně, který se vždy pohybuje o dvě políčka horizontálně nebo vertikálně, a poté jedno políčko kolmo k předešlému pohybu. Při volné hrací ploše má tedy osm možných pohybů. Začínající pozice figurek je vždy naproti sobě, na vzdálených stranách šachovnice a uprostřed řádku.

Cílem hry je izolovat nepřátelskou figurku, zde koně, tak, aby neměl žádný žádnou možnost dalšího pohybu. Toho se dosáhne díky tomu, že hráči nesmí postavit figurku na dříve už navštívené políčko jakýmkoli hráčem. První hráč, který nemá kde táhnout tedy prohrává.

Pro tuto hru bude opět použit minimax, alfa-beta optimalizace a heuristika. Základní heuristická funkce bude uvedena jako počet možných tahů MAX hráče minus počet možných tahů MIN hráče.

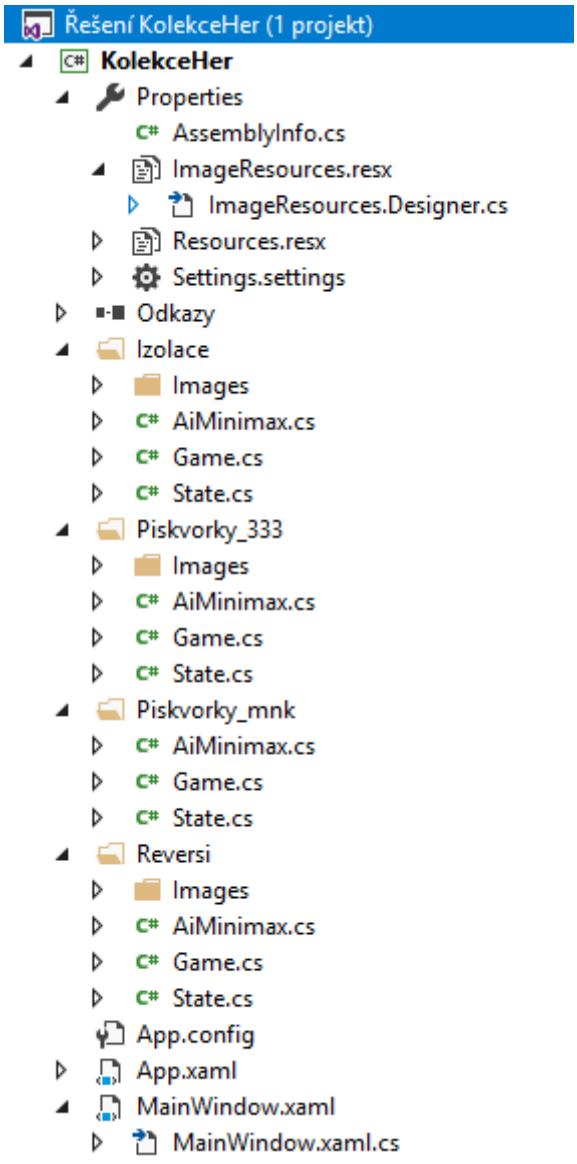
4 IMPLEMENTACE APLIKACE

Aplikace je napsána v jazyce C# v .NET frameworku za pomoci GUI knihovny WPF, ta umožňuje jednoduché psaní okenních aplikací pro platformu Windows. WPF byl vybrán místo běžnějšího Windows Forms převážně kvůli obsahu *Shapes* knihoven. Ty umožňují dynamicky generovat tvary, které se poté vykreslují na komponentu *Canvas*. V této aplikace je využití tvarů převážně pro generování čtverců, které slouží jako hrací plocha všech implementovaných her. Pro distinkci polí poté stačí aplikovat jiný obrázkový štětec na každý odlišný čtverec.

4.1 Struktura projektu

Při implementaci aplikace bylo vytvořeno řešení s názvem *KolekceHer*, s projektem *KolekceHer*, který poté ve vlastnostech obsahuje zdrojový soubor *ImageResources.resx*, ve kterém jsou uloženy veškeré grafické materiály pro aplikace. V tomto projektu se dále nachází složka *Izolace* s podložkou *Images*, ve které je využitá grafika v původní formě. Vedle obrázků se nachází tři hlavní třídy hry, *AiMinimax.cs*, ve které je implementován algoritmus AI, tedy minimaxu a jeho optimalizací, *Game.cs,* ve které je napsána grafická reprezentace hry, její pravidla, logika a tah hráče, a nakonec *State.cs*, která zároveň obsahuje třídu *Move*, které podle hry obsahuje informace nutné k uchování tahu hráče. Dále *State.cs* implementuje statické metody pro vyhodnocování stavů hry, získání možných tahů, a další. Stejná struktura jako v souboru *Izolace* je potom obsažena v souborech každé další hry, tedy *Piskvorky_333*, *Piskvorky_mnk* a *Reversi.*

Hlavní grafický vzhled aplikace je obsažen v souboru *MainWindow.xaml*, logika startu aplikace je potom v podsouboru *MainWindow.xaml.cs*, ve kterém se při zvolení různých her volá vytváření nových instancí *Game* tříd příslušné hry.



Obrázek 4.1 – Struktura projektu

4.2 Obecná implementace her

4.2.1 Třída Game

Třída *Game* řeší grafickou formu hry, její pravidla, validní tahy a celkovou funkčnost a logiku. Ve většině případů obsahuje následující metody.

SetField řeší vytvoření hracího pole a přichystání výchozích štětců pro barvení polí, které konvertuje z přiložených obrázků.

DrawField nastaví každému poli v hracím plánu nový objekt čtyřúhelníku s parametry podle logiky dané hry, také mu nastaví výchozí pozadí. Dále každému objektu přidá do jednotlivých

událostí požadované funkce, minimálně *RectangleClicked*, dále třeba *RectangleEntered* nebo *RectangleLeft*, které ošetřují zvýrazňování validních tahů.

RectangleClicked první testuje, zda je vybrané pole validní tah, poté provede tah hráče a následně zavolá *SetTurn.* V tuto chvíli se zkontroluje, zda hra už neskočila, pokud ne tak se volá metoda *MakeAiMove.*

SetTurn mění proměnnou hráče na tahu, v některých hrách řeší kolik tahů má nový hráč, zda hra opět nekončí.

MakeAiMove volá metodu třídy *AiMinimax.GeneralMinimax*, do které vkládá parametr nového stavu (*State)* hry, který získá zavoláním metody *ConvertBoard*. Poté graficky upravuje herní plochu podle tahu AI a volá opět metodu *SetTurn.*

ConvertBoard řeší transformaci uchovaného dvourozměrného pole objektů čtyřúhelníků na co nejmenší datový typ. V případě piškvorek se konvertuje na boolean s možnou hodnotou null. V jiných případech třeba na pole bytů.

U některých her se ještě objevuje metoda *GetMoves*, která vrací list možných pohybů hráče, který je právě na tahu.

4.2.2 Třída State

Tato třída obsahuje podtřídu *Move*, která má většinou jen atribut *X* a *Y* pozice tahu. Ve speciálních případech má i ostatní nutné informace pro provedení tahu.

State má vždy příslušné dvourozměrné pole s uloženým stavem hry a informaci, jaký hráč je na tahu. Konstruktor přebírá tedy jen tyto dva parametry. Ostatní metody jsou statické, a tedy vždy přebírají v parametru stav hry.

GetResult přijímá kromě stavu i *move*, tedy tah hráče. Vrací poté stav hry po provedení daného tahu.

GetMoves vrací list tahů, přičemž filtruje pouze možné tahy hráče, který je v přijatém stavu na tahu.

IsTerminal vyhodnocuje, zda je stav hry finální, tedy zda už nejsou k provedení žádné tahy, nebo byla splněna podmínka výhry.

GetUtility vrací celočíselnou hodnotu znázorňující zhodnocení herního pole. Čím je honota vyšší, tím je lepší pro MAX hráče a čím menší, tím je lepší pro MIN hráče. V každé hře je implementována jinak, reprezentuje heuristickou funkci.

CopyField je vedlejší metoda, která vytvoří hlubokou kopii dvourozměrného pole hrací plochy.

4.2.3 Třída AiMinimax

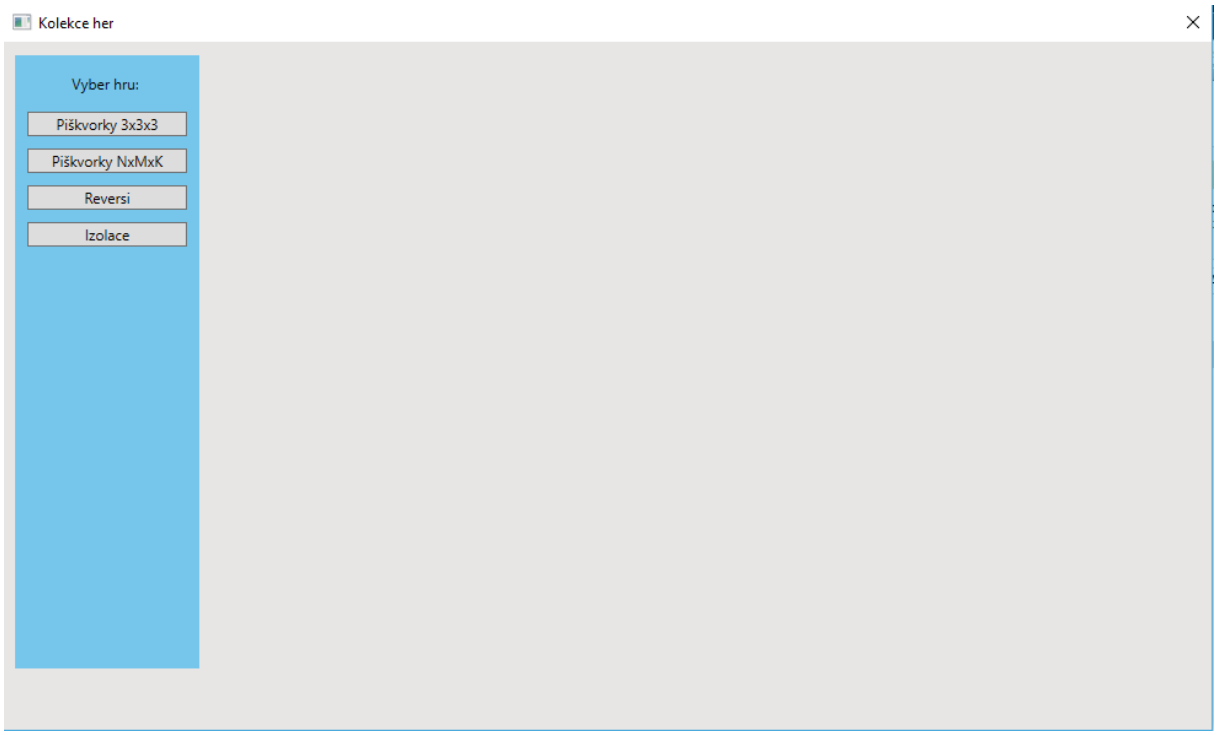
Tato třída obsahuje rozhodování AI pro každou hru. Velmi využívá třídu *State* a její statické metody pro vytváření a procházení n-cestného stromu tahů. Hlavní metoda se u každé hry liší podle toho, zda jde o klasický minimax, minimax s alfa-beta ořezáváním nebo minimax s iterativním prohlubováním.

Minimax je rekurzivní metoda, která pokud dostane v parametru terminální stav, vrací hodnotu jeho heuristické funkce. Pokud však stav není terminální a je na tahu hráč, jsou získány pomoci *State.GetMoves* jeho možné tahy a pro všechny takové je opět volán *Minimax* s parametrem *State.GetResult* daného tahu. Z hodnot všech tahů se poté vybere maximální hodnota která značí optimální tah. Pokud je na tahu AI, opět jsou získány jeho tahy, pro všechny je volán *Minimax* a je vybrána minimální získaná hodnota pro optimální tah. Mezitím se vždy do optimálního tahu ukládá nejúspěšnější provedený tah, aby po provedení metody mohl být uplatněn ve hře.

4.3 Uživatelská příručka

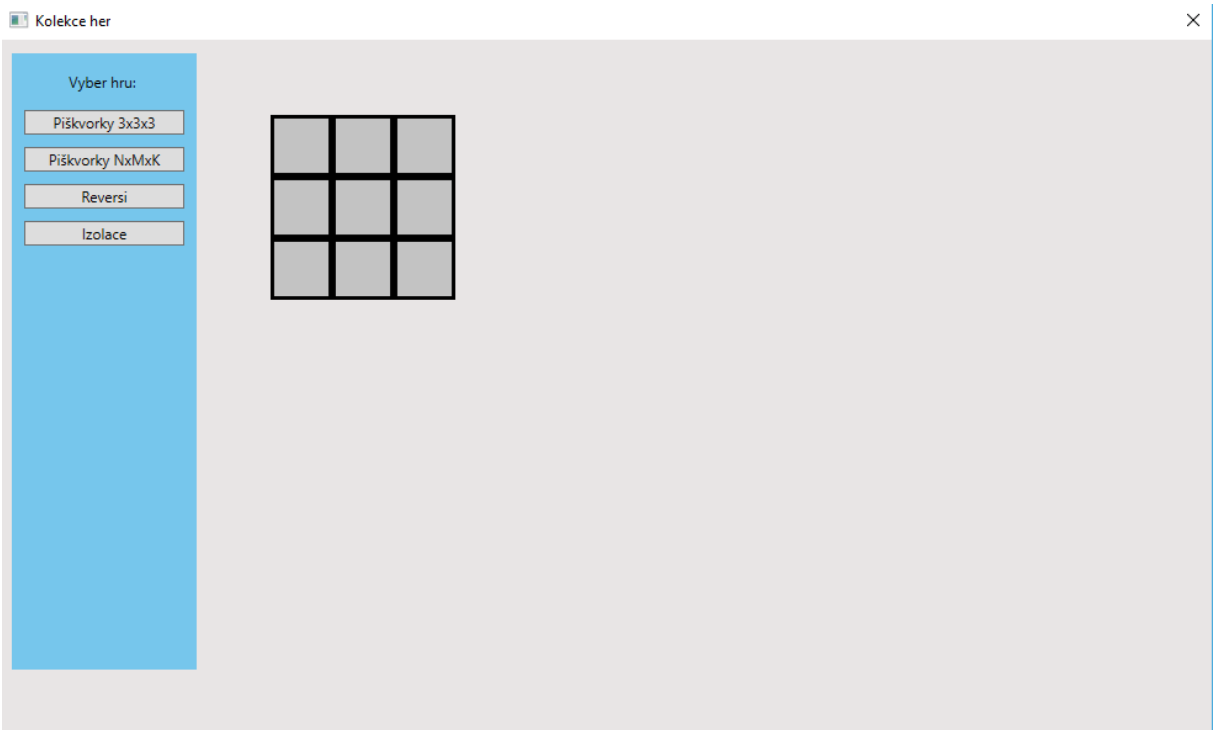
Celá aplikace se nachází v souboru *„Kolekce Her.exe“* a zapíná se pouhým spuštěním souboru.

Při zapnutí se otevře hlavní obrazovka.



Obrázek 4.2 – Menu aplikace

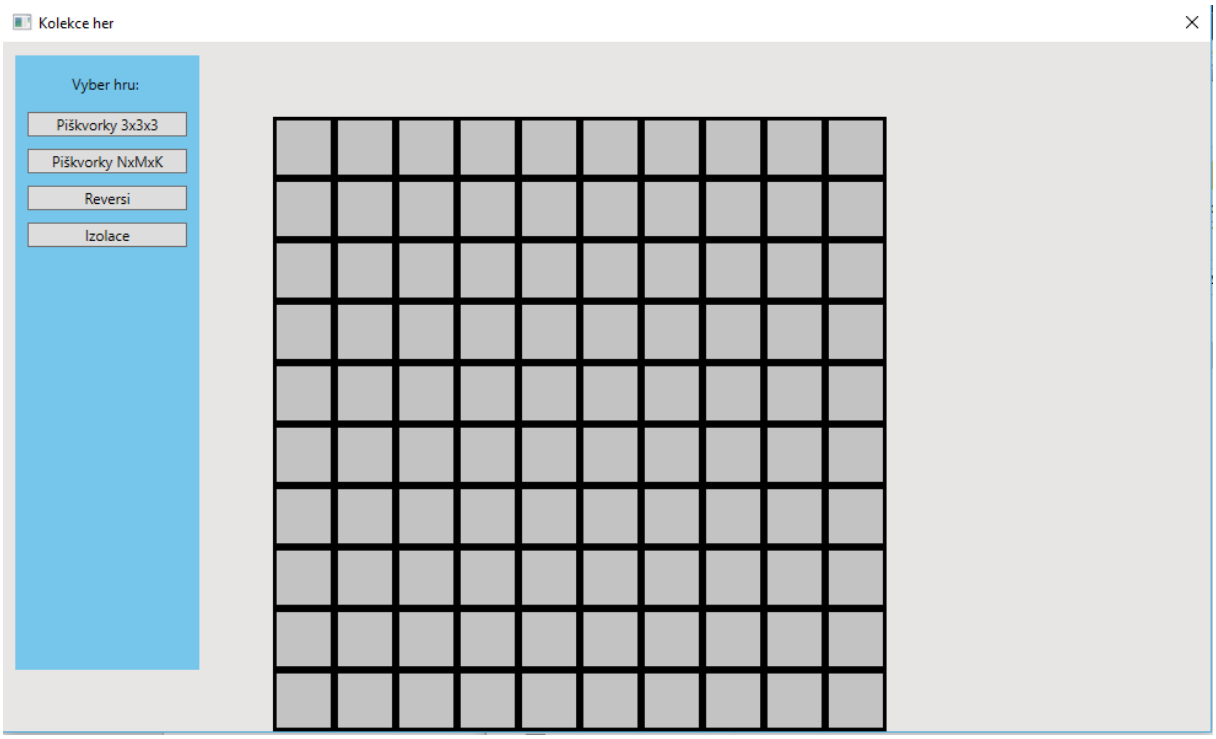
V tuto chvíle je nutné vybrat na levém panelu hru, kterou chcete hrát proti AI. Při vybrání první možnosti je okno následující.



Obrázek 4.3 – Malé piškvorky

AI hraje své tahy ve stejnou chvíli jako hráč, který však vždy začíná. Jelikož strom této hry je relativně malý, AI vždy uhraje remízu, anebo při chybě hráče vyhraje. Levým kliknutím na volné místo hráč umístí svůj piškvorek a při ukončení hry se na pravém panelu objeví stav hry.

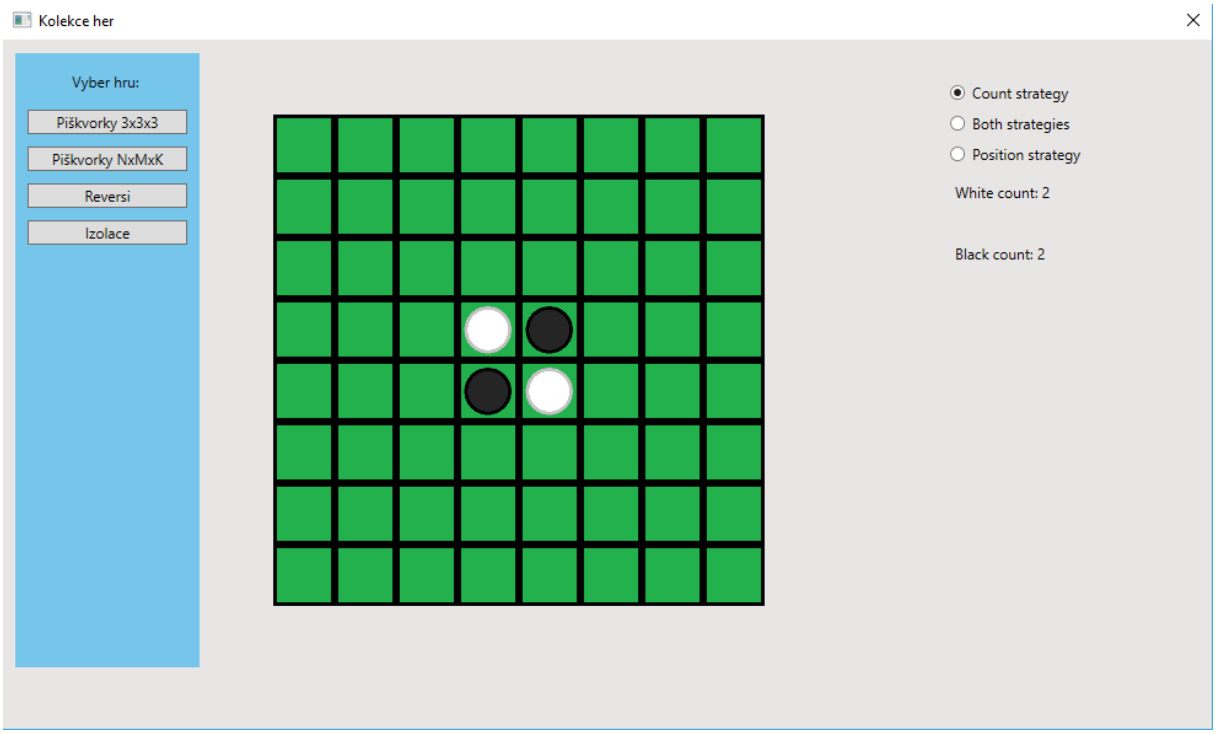
Při výběru druhé hry se objeví o dost větší pole.



Obrázek 4.4 – Velké piškvorky

Zde je ovládání a princip stejný jako u minulého okna. Pro resetování hry vždy stačí znovu kliknout na volbu hry na levém panelu.

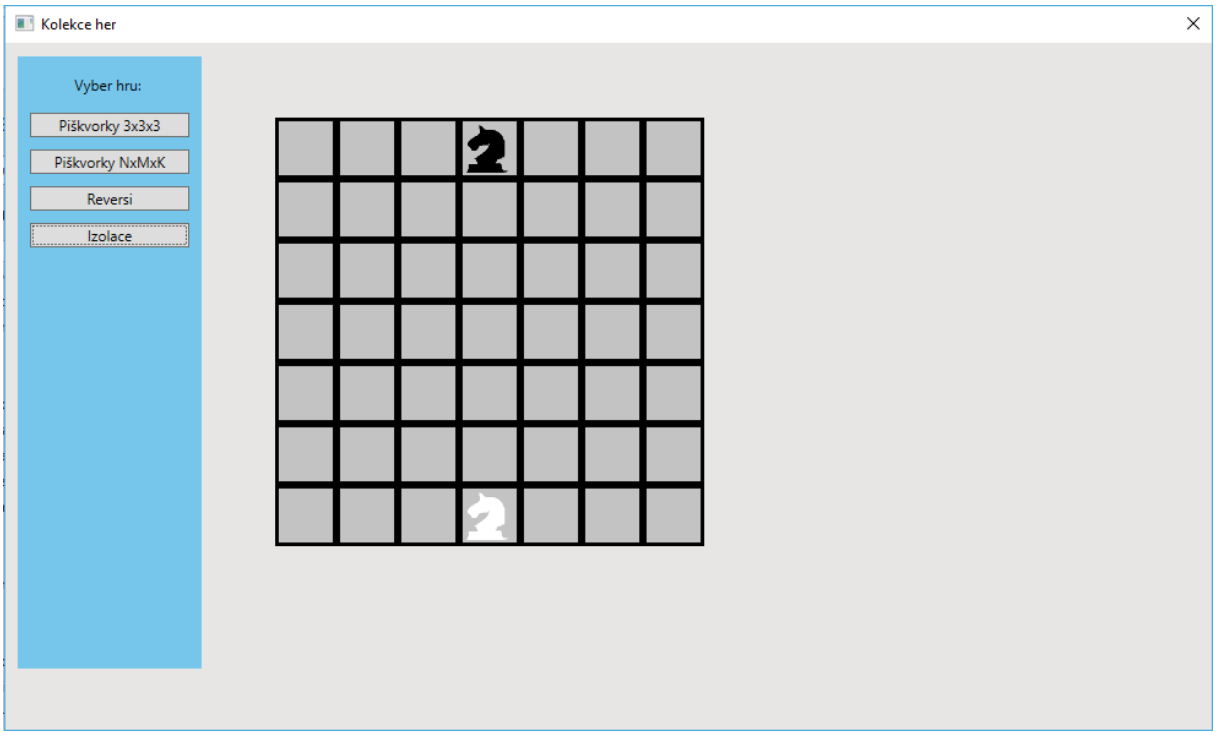
Okno Reversi zase vypadá takto.



Obrázek 4.5 – Reversi

Hrací okno Reversi je už odlišné od předchozích. Hráč hraje za bílé kameny a opět začíná, ale může volit pouze validní tahy. Ty jsou vyznačeny oranžově při ukázání myši na dané pole. Levým kliknutím se potom umístí kámen a provede se vyhodnocení. V pravém panelu jsou statistiky počtu kamenů každého hráče a také volba strategií, podle které hraje AI. Vždy může být zvolena pouze jedna strategie a strategie lze měnit v průběhu hry. Při konci hry je zobrazen přídavný text na pravém panelu a nejdou provádět další tahy.

Poslední tlačítko otevře hru Izolace.



Obrázek 4.6 – Izolace

Zde je princip ovládaní stejný jako v minulé hře. Validní tahy jsou zvýrazněny oranžově a při konci hry je výsledný stav ukázán na pravém panelu.

5 JINÁ VYUŽITÍ

Teorie her se kromě dříve ukázaného použití dá také aplikovat na velké množství klasických společenských her pro jakýkoli počet hráčů, dokonce se dá krajně použít i pro řešení moderních her pro jednoho hráče. Dále se využívá v biologii, kde řeší a popisuje vztahy mezi jednotlivými druhy, jejich reakce a chování, v historii a vojenství může zkoumat průběhy a možné výsledky bitev, v politologii se aplikuje pro řešení koalic a rozhodování. Velké uplatnění má také v matematice, kde se pomocí ní dají modelovat matematické problémy, které s hrami nemají nic společného, ale teorie her poskytuje vhodné aparáty pro jejich řešení. Podobným způsobem se dá využít v čisté informatice, kde se například vyhodnocují útočníkovy šance na ovládnutí sítě.

Přednostní využití teorie her je však ekonomie. Přímo se totiž nabízí pro popisování vztahů mezi jednajícími firmami nebo jinými subjekty, má vhodný aparát pro konfliktní situace a také je schopna analyzovat zisky při různých strategických volbách každého z účastněných subjektu. [9]

ZÁVĚR

V práci se úspěšně podařilo aplikovat teorii her na vytvoření AI protivníka pro určité deskové hry. V malých piškvorkách je protivník neporazitelný, vždy uhraje remízů, pokud hráč vybírá optimální tahy, nebo vyhraje, pokud se hráč odchýlí od optimální strategie. Velké piškvorky se ukázaly být relativně problematické, jelikož je složité pro ně najít vhodnou heuristickou hodnotící funkci, která by zároveň byla výkonově optimalizovaná pro velké pole. Výsledné AI u této hry tedy není zrovna vyhovující. Mnohem lepší variantou by zde bylo použít algoritmus Monte Carlo, který by zde našel lepší reakce na jednotlivé tahy. Protivník pro Reversi má naimplementované tři různé strategie, první (počet) je vhodná pro nenáročného hráče, druhá (kombinovaná) už dosahuje lepších výsledků, ale bylo by třeba lépe nastavit její parametry a poslední strategie (porovnávání pozic) je asi nejúčinnější. Izolace má pak průměrně silnou AI a je schopna relativně často porazit lidského hráče. Aby byli protihráči ještě silnější, bylo by třeba se mnohem více věnovat vhodným heuristickým funkcím a rozsáhlým strategiím jednotlivých her. Algoritmy pro sestavení AI se povedly naimplementovat i s příslušnými optimalizačními metodami. Samozřejmě by šlo docílit lepších výsledků a vyšší prohledávací hloubky úspornější implementací a dalšími, už méně známými, optimalizačními metodami.

POUŽITÁ LITERATURA

[1] M. Dlouhý a P. Fiala, *Úvod do teorie her*, 2. editor, Praha: Oeconomica, 2009, str. 119.

[2] M. Maňas, *Teorie her a její aplikace*, 1. editor, Praha: Teoretická knižnice inženýra, 1991, str. 278.

[3] M. Maňas, *Teorie her a konflikty zájmů*, 1. editor, Praha: Oeconomica, 2002, str. 114.

[4] M. OpenCourseWare, „*6. Search: Games, Minimax, and Alpha-Beta,*“ 10. 1. 2014. [Online]. Dostupné z: https://www.youtube.com/watch?v=STjW3eH0Cik&list=PLUl4u3cNGP63gFHB6xb-kVBiQHYe_4hSi&index=7&t=0s. [Přístup získán 19. 4. 2019].

[5] G. Bartel, „*Playing Strategy Games With The Minimax Algorithm*,“ 11. 12. 2017. [Online]. Dostupné z: https://medium.freecodecamp.org/playing-strategy-games-with-minimax-4ecb83b39b4b. [Přístup získán 22. 4. 2019].

[6] C. Zaslavsky, Tic Tac Toe: And Other Three-In-A Row Games from Ancient Egypt to the Modern Computer, Crowell, 1982.

[7] S. Schaefer, „*Tic-Tac-Toe (Naughts and Crosses, Cheese and Crackers, etc.),*“ 1. 1. 2002. [Online]. Dostupné z: http://www.mathrec.org/old/2002jan/solutions.html. [Přístup získán 20. 4. 2019].

[8] S. MacGuire, „*Strategy Guide for Reversi & Reversed Reversi,*“ 4. 1. 2011. [Online]. Dostupné z: http://samsoft.org.uk/reversi/strategy.htm. [Přístup získán 22. 4. 2019].

[9] Z. Sawa, „*Teorie her, studijní opora*,“ 24. 9. 2015. [Online]. Dostupné z: http://www.cs.vsb.cz/sawa/teh/opora/TEH-opora.pdf. [Přístup získán 5. 5. 2019].

[10] J. Pettersson, „*[Guide] Transposition tables,*“ 30. 1. 2007. [Online]. Dostupné z: http://mediocrechess.blogspot.com/2007/01/guide-transposition-tables.html. [Přístup získán 22. 4. 2019].

PŘÍLOHY

PŘÍLOHA A – APLIKACE A ZDROJOVÉ KÓDY

Úložiště obsahuje *KolekceHer.exe* pro spuštění aplikace a taky projektový adresář, ve kterém jsou všechny zdrojové kódy a obrázky pro sestavení aplikace.