Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Dominik Malý

Trenažér mariáše

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Martin Senft
Studijní program: Programování

2007

Poděkování patří Mgr. Martinu Senftovi za podnětné připomínky a vedení této práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 03/08/2007 Dominik Malý

Obsah

1 Úvod 6

2 Mariáš 8

2.1 Základní informace 8

2.2 Existující implementace a jejich srovnání s mým programem 9

3 Algoritmy pro hry s úplnou informací 12

3.1 Existující algoritmy 12

3.1.1 Minimax 13

3.1.2 Negamax 15

3.1.3 Alfa-beta prořezávání 16

4 MAPP algoritmus 18

4.1 Popis a vývoj algoritmu 18

4.1.1 Funkce MAPP 19

4.1.2 Generátor tahů 21

4.1.3 Ohodnocovací funkce 24

4.1.4 Vývoj funkce MAPP 25

4.2 Rozšiřitelnost algoritmu 27

4.3 Výhody a nevýhody algoritmu 28

5 Shrnutí 30

Literatura 31

A Uživatelská dokumentace 32

A.1 Instalace a spuštění aplikace 32

A.2 Pravidla mariáše 32

A.3 Ovládání aplikace 35

B Programátorská dokumentace 37

B.1 Rozvržení programu 37

B.2 Datové struktury 38

B.3 Grafické uživatelské rozhraní 41

B.4 Řízení běhu programu 43

B.4.1 Spuštění programu a inicializace nové hry 43

B.4.2 Zpracování událostí generovaných uživatelem 44

B.4.3 Průběh partie 45

C Obsah CD ROM 47

Název práce: Trenažér mariáše
Autor: Dominik Malý
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: Mgr. Martin Senft
e-mail vedoucího: Martin.Senft@mff.cuni.cz

Abstrakt: Mariáš je pravděpodobně nejznámější a nejoblíbenější karetní hrou v Čechách a přitom zůstává téměř výhradně výsadou našich luhů a hájů. V této práci studujeme, jakým způsobem je možné tuto zajímavou hru implementovat v řeči jedniček a nul výpočetních zařízení. Je zde také stručně popsán vývoj algoritmů pro implementaci her s úplnou informací a nulovým součtem, od stařičkého minimaxu, přes alfa-beta prořezávání až k mému vlastnímu upravenému negamaxu - algoritmu pro hry s neúplnou informací, používajícímu obecnější postup, který by teoreticky mohl být rozšiřitelný na všechny existující i neexistující karetní hry.

Klíčová slova: mariáš, hry s (ne)úplnou informací, minimax, alfa-beta prořezávání

Title: Mariáš trainer
Author: Dominik Malý
Department: Department of Software and Computer Science Education
Supervisor: Mgr. Martin Senft
Supervisor's email address: Martin.Senft@mff.cuni.cz

Abstrakt: Mariáš is probably the most well-known and favourite card game in Czech Republic and despite this remains to be seen almost exclusively in our country. In this work we study the ways, how can this interesting game be implemented in the speech of ones and zeros typical for computers. This work also briefly describes the evolution of full information games solving algorithms, from old minimax to alfa-beta pruning and my own adjusted negamax - algorithm for games without full information, using a more universal approach, which could be theorethically extended for all card games, whether existing or not.

Keywords: mariáš, games with (in)complete information, minimax, alfa-beta pruning

Kapitola 1
Úvod

Asi není jediný Čech, který by nikdy neslyšel o karetní hře mariáš. Většině lidí se při vyslovení tohoto slova nejspíš vybaví zakouřené hospůdky, ve kterých se skupinky nadšeně karty mastících tatíků snaží obrat jeden druhého o veškeré drobné. Je pravda, že mariáš je především hazardní hra. V našich reáliích se však výhry většinou počítají v desetnících, a když už někdo vyhraje v jedné partii několik korun, bývá to důvod k oslavě.

Mariáš je svým způsobem českou obdobou světoznámé hry poker. Obě hry spolu sice historicky nemají co do činění a nepatří ani do stejné skupiny karetních her. Přesto však mají mnoho společného. V obou hrách se hraje o peníze, na vrcholné úrovni je nezbytná skvělá paměť a dobrý odhad soupeřovi „ruky“. Mariáš se však na rozdíl od pokeru hraje s, v Čechách tradičnějším, dvaatřicítkovým balíčkem karet a není v něm ani tak důležitá schopnost chladnokrevně bluffovat, jako spíš umět odhadnout své možnosti a dokázat si vyjít vstříc se spoluhráčem.

V současnosti se zdá, že je mariáš (a karetní hry vůbec) tak trochu na ústupu. Pokud člověk vstoupí do hospody, ať už na venkově nebo ve městě, většinou se místní štamgasti, místo toho, aby trénovali paměť a hráčské dovednosti u mariáše, jen trpělivě zpíjejí do němoty.

Mariáš se čím dál více stává pouze výsadou starší generace. U těch mladších mají karetní hry tvrdou konkurenci v nových technologických výdobytcích, jako je třeba Internet. Přesto, že mladých hráčů už zřejmě nikdy nebude přibývat tolik, jako kdysi, není příliš pravděpodobné, že by tato fenomenální hra ztratila na popularitě natolik, aby zcela zanikla. Právě Internet je totiž pro mariáš tak trochu i živou vodou. Hráči se teď už nemusejí ke hře scházet v hospůdkách nebo doma. V Čechách už totiž existuje několik karetních serverů, specializovaných na tuto hru. Vznikají také oficiální mariášové Internetové ligy a vzhledem k množství hráčů do nich zapsaných má mariáš k odpisu ještě hodně daleko.

Já osobně bych určitě nepatřil v těchto oficiálních ligách k vrcholovým hráčům. Mariáš však hraji od dětství a hraji ho rád a snad i celkem dobře. Proto jsem se rozhodl pokusit se vytvořit algoritmus, který by byl schopen rozumně hrát nejen mariáš, ale třeba i další, více či méně podobné, karetní hry.

Tato práce je rozvržena následujícím způsobem. V následující kapitole nejprve popíši zařazení, varianty a princip karetní hry mariáš. Poté se zamyslím nad způsobem, jakým tuto hru implementovali jiní programátoři, a nastíním způsob, jakým jsem k problému přistupoval já sám.

Ve třetí kapitole se budeme věnovat některým ze základních existujících algoritmů pro řešení her s úplnou informací a nulovým součtem, konkrétně minimaxu, negamaxu a alfa-beta prořezávání. Ukážeme, proč jsou tyto algoritmy pro řešení mariáše nepoužitelné a jakým způsobem by bylo třeba je upravit.

Konečně v závěrečné čtvrté kapitole popíši svůj vlastní algoritmus na řešení (nejen) karetní hry mariáš a rozeberu jeho přednosti a nevýhody. Na závěr nastíním způsob, jakým by můj algoritmus byl teoreticky rozšiřitelný na další karetní hry.

Součástí práce je také uživatelská a programátorská dokumentace, obsažená v přílohách A a B.

Kapitola 2

Mariáš

2.1 Základní informace

Mariáš je nejpopulárnější karetní hrou v České republice a na Slovensku.

Patří mezi takzvané Trick Taking Games[1] (hry hrané na „štychy“). V těchto hrách je každému hráči obvykle rozdáno stejné množství karet a hra sestává z jednotlivých „štychů“, tedy z kol, kdy každý hráč položí na stůl jednu kartu obrácenou lícem nahoru. Podle pravidel dané hry se rozhodne, která karta je „nejvyšší“, a její majitel poté celý štych vyhraje (většinou ho položí před sebe na stůl lícovou stranou dolů). Výherce štychu poté obvykle zahajuje další kolo hry.

Podle dalšího rozřazení karetních her spadá mariáš do tzv. Point Trick Games (her hraných na body), ve kterých jsou jednotlivé karty bodově ohodnocené a kde výsledek hry není určen počtem vyhraných štychů, ale součtem bodů uhraných karet.

Mariáš patří mezi tzv. Ace-ten games, tedy hry, ve kterých jsou jedinými hodnocenými kartami esa a desítky.

Podle posledního zařazení patří mariáš do Marriage group, do skupiny her, v nichž se hraje na tzv. hlášky. Hláškou je, když má hráč na ruce krále a svrška (jinak také filka) stejné barvy.

Mariáš se hraje s balíčkem dvaatřiceti karet, rozdělených na čtyři barvy po osmi kartách. Barvami jsou červené nebo jinak také srdce, žaludy, kule a zelené. Jednotlivé karty, řazené podle hodnoty od nejvyšší, jsou: eso, desítka (v metahrách betl a durch patří desítka mezi spodka a devítku a nepatří mezi bodované karty, viz. níže), král, svršek (jinak také filek), spodek, devítka, osmička a sedma.

[1] Veškeré informace týkající se zařazení mariáše pocházejí z internetových stránek o karetních hrách www.pagat.com

Mariáš existuje ve třech základních variantách. Nejčastější je varianta pro tři hráče, která se dále dělí na licitovaný a volený mariáš. Ve voleném mariáši se hráči pravidelně střídají ve volení typu hry, pokud ovšem nedojde k „ukradnutí hry“ na betl nebo durch (viz. níže). V licitovaném mariáši se na začátku každé hry takzvaným „licitováním“ rozhodne, který hráč má nejlepší kartu (nebo si to alespoň myslí) a který tudíž bude i volit. Mariáš existuje i ve verzi pro čtyři hráče, jde o tzv. křížový mariáš. Pokud jsou hráči pouze dva, mohou se coby poslední záchrana pustit do tzv. „lízaného mariáše“, který je ovšem daleko méně zábavný než verze pro tři a čtyři hráče.

Má práce se věnuje výhradně volenému mariáši, ovšem je možné ji rozšířit i na ostatní typy (na licitovanou verzi je v programu přímo připravena půda).

2.2 Existující implementace a jejich srovnání s mým programem

Vzhledem k popularitě mariáše v našich končinách není divu, že se již v dobách operačního systému MS-DOS, nebo možná ještě dřív, začaly objevovat první mariášové programy.

Většina ze současných mariášových programů nabízí pěkné uživatelské rozhraní a v drtivé většině i velmi chytré protihráče. Pro všechny programy, od nichž jsem se dostal ke zdrojovému kódu, však byla typická také jedna ne už tak lichotivá záležitost. Jejich algoritmy fungovaly na principu užívání všelijakých heuristik typu „pokud máš takové a takové konkrétní karty, je většinou výhodné hrát tímto způsobem“. Programy samozřejmě počítaly, kolik karet dané barvy ještě zbývá a která z desítek ještě nebyla odehraná, vždyť k tomu jsou počítače jako stvořené. Kromě toho se však už jen pomocí všemožných heuristik a domněnek (slouží jim ke cti, že většina z nich se obvykle „domnívala“ zcela správně) snažila vypozorovat, jaké má který hráč karty a podle toho reagovala na tahy ostatních hráčů.

Je třeba spravedlivě uznat, že tyto programy skutečně v krátkém čase provádějí většinou dobrá rozhodnutí. Všechny tyto jejich heuristiky a pozorování

jsou však samozřejmě zcela závislé na typu hrané hry a jsou tedy naprosto nepřenositelné. Tyto programy proto musely používat jeden algoritmus pro klasickou hru, další více či méně odlišný pro hraní na sedmičku a ještě jeden, v zásadě zcela rozdílný, pro hraní betlu a durchu.

Tato práce se ubírá zcela opačným směrem. Jakési prapůvodní „předzadání“ této bakalářské práce totiž znělo „Univerzální trenažér karetních her“ (Slovo „trenažér“ je užíváno spíše z nedostatku lepších termínů. Program měl být schopen hry hrát a kontrolovat legálnost tahů. Neměl ani tak hráče „trénovat“.). Zadání práce bylo později přeformulováno do současné podoby, tedy „Trenažér mariáše“, jenž „se má pokusit o obecnější přístup, který by umožňoval rozšíření na hry podobného typu“. A přesně o to se tato práce pokouší.

Přílišná obecnost však s sebou přináší také mnohé nevýhody a omezení. I kdyby se podařilo vyvinout algoritmus a datovou strukturu dostatečně obecné na to, aby dokázaly popsat a hrát libovolnou karetní hru, budou tyto struktury velmi složité, méně robustní (z hlediska chybovosti) a především daleko pomalejší než programy zaměřené na jednu konkrétní hru. Tomu se téměř s určitostí nedá vyhnout, výhody specializace jsou lidstvu známé již odnepaměti.

V obecnosti takovéhoto programu by však zároveň byla i jeho síla. Sice by žádnou složitější hru (kterou mariáš určitě je) nedokázal hrát skutečně dobře, za to by si však dokázal poradit s daleko více hrami (v ideálním případě se všemi), než jen s jednou jedinou, jak je tomu u specializovaných programů. Proto by takový algoritmus určitě měl co nabídnout.

Na počátcích vývoje algoritmu jsem viděl dvě cesty, kterými by se takovýto projekt snad mohl ubírat. Jednou by byl program založený na principu samoučení. Program, který by se ve svých počátcích vlastně jen rozhlížel a sledoval tahy a způsob hry ostatních hráčů. Tato cesta se mi však zdála velmi trnitá. Věděl jsem o existenci neuronových sítí a dalších podobných konstrukcí, které jsou schopny principu učení využívat. Neměl jsem však v tomto směru sebemenší zkušenosti a celá myšlenka mi přišla příliš ambiciózní a neschůdná. Navíc by program ve své „učící fázi“ musel hrát zcela náhodně a byl tak pro všechny hráče snadnou kořistí.

Jako další variantu jsem uvažoval přístup aktivnější; spíše než na reaktivním učení se podle ostatních hráčů by byl založený na generování možného vývoje hry a ohodnocování jednotlivých jejích stavů. Od této myšlenky není daleko ke zjištění,

že takový algoritmus už vlastně existuje. Tímto způsobem funguje již tisíckrát použitý stařičký minimax a veškeré jeho klony. Tyto algoritmy se však s karetními hrami neslučují v několika zásadních věcech, z nichž nejdůležitější je to, že jsou použitelné pouze pro hry s úplnou informací. Tuto podmínku by karetní hry splňovaly pouze v tom případě, že by algoritmus „koukal“ do rukou protihráčů. A s takovým programem by jen těžko chtěl někdo hrát. Přesto však tato cesta nabízela zajímavé možnosti a proto se jí budeme v této práci dál věnovat.

Kapitola 3
Algoritmy pro hry s úplnou informací

Pokud se řekne slovo „hra“, vybaví se většině lidí klasické deskové hry typu šachy, dáma, v exotičtějších končinách go, anebo třeba i jednoduché, ale velmi populární piškvorky. Všechny tyto hry mají alespoň jednu věc společnou, jde o logické hry. To znamená, že mohou být popsány sadou pravidel a premisí. U každé z těchto her je možné v dané situaci rozpoznat, jaké jsou další možné tahy. Proto všechny také sdílejí další společnou charakteristiku, jde o hry s úplnou informací. Každý hráč totiž ví vše o možných tazích svého protivníka.

Vzhledem k popularitě těchto her není divu, že nedlouho po vzniku prvních počítačů se začaly objevovat i první pokusy o implementaci těchto her. Některé z nejznámějších a nejpoužívanějších algoritmů pro hry s úplnou informací si podrobněji představíme i v této práci.

3.1 Existující algoritmy

Algoritmy pro hry s úplnou informací jsou úzce svázány s pojmem prohledávacích stromů. Každý z těchto algoritmů totiž při svém běhu staví tzv. „strom hry“ – prohledávací strom, kde každý z vrcholů představuje určitý konkrétní stav dané hry a každá z hladin/úrovní stromu představuje tahy jednoho z hráčů v určitém kole hry.

Strom hry je generován pomocí techniky prohledávání do hloubky (depth-first search). Začíná se v kořeni, který představuje aktuální stav hry, a končí v konečných stavech hry (pokud je to možné). Pro každý právě procházený vrchol stromu se vygeneruje jeden potomek za každou z možných reakcí na daný vývoj hry.

Pokud se při prohledávání stromu dojde do konečného stavu hry anebo do takové hloubky, že rozsáhlejší prohledávání by už bylo časově nebo paměťově neúnosné, daný stav hry se číselně ohodnotí podle předem dané ohodnocovací funkce a započne se postupné zpětné ohodnocování předků daného vrcholu. Každý z vrcholů se při něm ohodnotí minimální nebo maximální hodnotou ze svých synů, podle toho, který z hráčů je na řadě. Hráč, který algoritmus uvedl do běhu totiž pochopitelně chce dosáhnout co nejlepšího ohodnocení hry a maximalizovat tak i své šance na výhru. Jeho protivník naopak chce jeho šance minimalizovat a tak zároveň i zvýšit své vlastní šance na úspěch.

Strom hry se tedy ohodnocuje zdola od listů po vrstvách, v jednotlivých vrstvách stromu se počítají střídavě minima a maxima z hodnot synů, dokud se nezíská hodnota kořene.

Zvoleným tahem se nakonec stává ten, který vede z kořene do toho uzlu, kde je ohodnocení stejné jako v kořeni.

Tímto relativně jednoduchým, avšak elegantním, způsobem funguje první z algoritmů, který je zároveň i základním kamenem všech ostatních. Algoritmus minimax.

3.1.1 Minimax

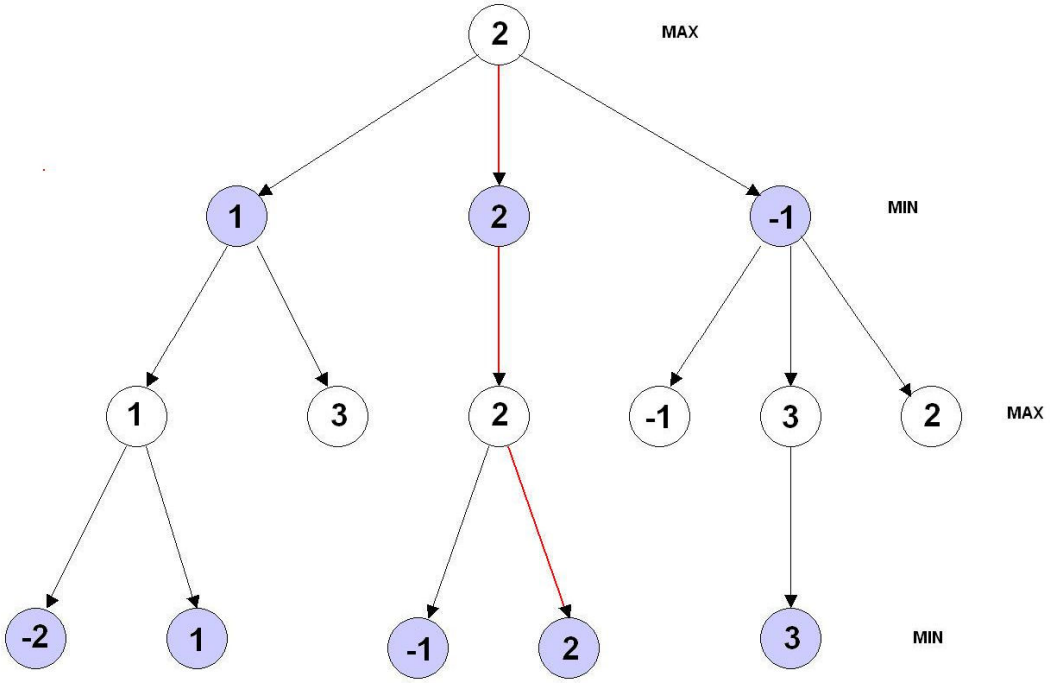
Minimax je metodou rozhodovací teorie pro *mini*malizování *max*imální možné ztráty, nebo alternativně také pro maximalizaci minimálního zisku. Své počátky odvozuje od teorie her dvou hráčů s nulovým součtem. Byl však rozšířen i na komplexnější hry a obecně na rozhodování v přítomnosti nejistoty.

Minimax obecně staví prohledávací strom do té doby, než narazí na konečný stav hry. U jednoduchých her, jako je třeba tic-tac-toe (= piškvorky na 3x3 poli), není problém projít všechny možné varianty vývoje partie od začátku až do konce. V případě sofistikovanějších her by to však typicky bylo časově či paměťově neúnosné. U takových her běží minimax maximálně tak dlouho, dokud neprojde tolik tahů, kolik udává parametr depth určený při volání funkce MinMax.

Algoritmus minimax, zapsaný v pseudokódu, vypadá takto:

```
Move MinMax (GamePosition game, int depth) {
       return MaxMove (game);
}

Move MaxMove (GamePosition game, int depth) {
       if (GameEnded(game) OR depth == 0)
              return EvalGameState(game);
       else {
              best_move <- {};
              moves <- GenerateMoves(game);
              ForEach moves {
                     move <- MinMove(ApplyMove(game), depth - 1);
                     if (Value(move) > Value(best_move))
                            best_move <- move;
              }
       }

       return best_move;
}

Move MinMove (GamePosition game, int depth) {
       best_move <- {};
       moves <- GenerateMoves(game);
       ForEach moves {
              move <- MaxMove(ApplyMove(game), depth - 1);
              if (Value(move) < Value(best_move))
                     best_move <- move;
       }

       return best_move;
}
```

Obrázek 3.1.1: Příklad prohledávacího stromu minimaxu. Světle označené jsou ty uzly, ve kterých táhne aktuální hráč (a ve kterých se tudíž volí maximum z hodnot potomků). V tmavých uzlech jsou situace před tahem protihráče. Červeně je vyznačena cesta nejlepšího nalezeného vývoje hry (pro aktuálního hráče – hráče, který je při volání algoritmu na tahu).



Minimax zdaleka není algoritmem dokonalým. Na jeho základě však bylo postaveno několik dalších algoritmů, které napravují některé nedostatky a zvětšují jeho efektivitu.

3.1.2 Negamax

Algoritmus negamax je v zásadě jen drobnou úpravou minimaxu. Jedná se o jinou realizaci téhož algoritmu, která však značně zjednodušuje programátorský kód a je jednoznačně elegantnější.

Negamax využívá jednoduchého faktu, že:

```
min(a,b) = -max(-a,-b)
```

Hodnota každého uzlu se tedy počítá jako maximum z hodnot synů a před předáním výsledné hodnoty z uzlu nahoru se změní její znaménko.

Algoritmus negamax, zapsaný v pseudokódu, vypadá takto:

```
int NegaMax (game, depth)
{
    if (GameEnded(game) OR depth == 0)
        return EvalGameState(game);
    best_move <- -INFINITY;
    moves <- GenerateMoves(game);
    ForEach moves
         move <- max(move, -NegaMax(ApplyMove(game), depth - 1));
    return move;
}
```

POZNÁMKA: Tento algoritmus je zapsán jednodušeji (a obvykleji) než algoritmus minimaxu v předchozí kapitole. Především vrací přímo číselnou hodnotu tahu a ne datovou strukturu popisující celý tah. Tento způsob je daleko praktičtější, než si mezi rekurzivními voláními předávat potenciálně složitou strukturu.

3.1.3 Alfa-beta prořezávání

Další značnou nevýhodou minimaxu (a negamaxu) je fakt, že prochází celý prohledávací strom i přesto, že některé větve by mohl beze ztráty dat vypustit a podstatně tím zrychlit celý výpočet.

Příklad takovéhoto bezpečného vypuštění je vidět na obrázku 3.1.1: Předpokládejme, že procházíme strom hry odleva a že už jsme dopočítali hodnotu prvních dvou synů kořene. Zbývá nám už tedy jen třetí potomek (a jeho podstrom). Jelikož v kořeni stromu se volí maximum ze synů, máme teď díky prostředními synovi jistotu, že výsledek bude minimálně roven dvěma. Když tedy začneme procházet podstrom třetího syna, je už u prvního uzlu vidět, že nedojde ke zlepšení a

že bychom tedy mohli zbylé větve podstromu vypustit. Hodnota prvního uzlu je totiž –1 a jelikož se v dané hladině stromu volí minimum z potomků, nebude hodnota třetího potomka kořene nikdy vyšší než právě oněch –1. Proto se ideální tah bude ubírat cestou prostředního syna kořene (2 je totiž větší než –1) a poslední tři uzly stromu bychom už vůbec nemuseli navštěvovat (jsou to ty s hodnotami 3, 3 a 2).

Tento problém řeší algoritmus alfa-beta prořezávání (anglicky alfa-beta prunning).

Algoritmus alfa-beta prořezávání, zapsaný v pseudokódu, vypadá takto:

```
int AlphaBeta(int depth, int alpha, int beta)
{
    if (GameEnded() OR depth == 0)
        return EvalGameState();
    moves <- GenerateMoves();
    ForEach moves
    {
        MakeNextMove();
        val = -AlphaBeta(depth - 1, -beta, -alpha);
        UnmakeMove();
        if (val >= beta)
            return beta;
        if (val > alpha)
            alpha = val;
    }
    return alpha;
}
```

POZNÁMNKA: Po odebrání modře zabarvených částí algoritmu dostaneme (trochu jinak zapsaný) negamax. Změn ve zdrojovém kódu tedy není mnoho.

Kapitola 4

MAPP algoritmus

Jak jsme zjistili, algoritmy pro hry s úplnou informací nabízejí pro naše potřeby mnoho zajímavých možností. Zároveň však také přinášejí několik problémů.

Nejzávažnějším z těchto problémů je určitě výše zmiňovaný fakt, že karetní hry nejsou hrami s úplnou informací. Skuteční hráči si totiž navzájem do rukou nevidí (nebo by alespoň neměli) a počítačové programy by tento fakt měly respektovat.

Z tohoto důvodu by však minimax a jemu podobné algoritmy nemohl stavět strom hry. Pro každý uzel prohledávacího stromu se totiž má vygenerovat jeden potomek za každý legální tah daného hráče. Protože však program netuší (nebo respektive jen předstírá, že netuší) jaké karty onen protihráč na ruce skutečně má, nemůže ani generovat strom hry a je tím pádem v koncích, alespoň co se algoritmů pro hry s úplnou informací týče.

Tento problém jsem ve své práci řešil zavedením prvku pravděpodobnosti. Algoritmus jsem pracovně pojmenoval MAPP – Maximum of Average Probable Profit algorithm, česky algoritmus nejvyšší průměrné pravděpodobnosti zisku[2].

4.1 Popis a vývoj algoritmu

Algoritmus MAPP se skládá ze dvou hlavních částí.

Prvním z nich je samotný upravený minimax, respektive některý z jeho klonů. Tato metoda zajišťuje rekurzivní procházení stromu hry a hledání ideálního tahu.

[2] Spojení pojmů „nejvyšší“ a „průměrný“ zní poměrně dost zvláštně. Důležité je však pořadí těchto slov. Ostatně minimax je přece také algoritmus pro „minimalizování“ „maximální“ možné ztráty. Důvody pro takové pojmenování jsou popsány níže.

Samotnou stavbu stromu hry však zajišťuje část druhá – generátor tahů, který zároveň i zprostředkovává řešení problému s neúplností informace hry.

O něco méně zásadní, ovšem také velmi důležité, jsou dvě metody pro virtuální provedení a zpětné od-provedení daného tahu. V pseudokódu jednotlivých algoritmů jsou popisovány jako MakeNextMove() a UnmakeMove().

Poslední součástí algoritmu je nezbytná ohodnocovací funkce jednotlivých stavů hry.

4.1.1 Funkce MAPP

Rekurzivní část algoritmu, kterou budu v dalším textu označovat jako „funkci MAPP“, vypadala v začátcích vývoje algoritmu přibližně takto:

```
float MAPP(depth, alfa, beta, playersOrder, /*ODKAZEM*/result) {
   int player = getFirst(playersOrder);
   int minmax = ( (MinOrMax(player)) ? +1 : -1 );

   if (GameEnded() OR depth == getAILevel())
       return EvalGameState();

   moves <- GenerateMoves();
   removeFirst(playersOrder);
   ForEach moves {
       MakeNextMove();
       if (TurnEnded()) {
           newOrder = getNewPlayersOrder();
           val = minmax * MAPP(depth + 1,-beta,-alpha,newOrder,result);
       }
       else
           val = minmax * MAPP(depth,-beta,-alpha,playersOrder,result);
       UnmakeMove();
       if (val >= beta)
           return beta;
       if (val > alpha) {
           alpha = val;
           if (depth == -1)     // rekurze se vratila do korene
               result.markMove();      // oznacime novy nejlepsi tah
       }
   }
   return minmax * alpha;
}
```

Funkce se volá jako MAPP(-1, -INFINITY, +INFINITY, playersOrder, result) a návratová hodnota je typu reálné číslo (z toho důvodu, že se v algoritmu pracuje s pravděpodobnostmi, viz. níže).

Důležité je povšimnout si, že parametr „depth“ se tentokrát nemění při každém postupu po úrovních stromu, ale jen při ukončení kola hry. V Mariáši, a v případě Trick Taking Games obecně, se totiž nedá skutečně určit výsledek daného kola, dokud není úplně u konce. Poslední hráč totiž může všechny položené karty přebít a veškeré zahrané karty se od dosavadního vítěze štychu přesunou k němu. Proto se hloubka prohledávání stromu neurčuje po jednotlivých hladinách, ale po kolech hry. Parametr depth se proto ve funkci MAPP inkrementuje a hodnota –1 označuje nejvyšší kořenovou úroveň (pro potřeby nastavení skutečné návratové hodnoty algoritmu, kterou představuje parametr „result“).

Parametr „playersOrder“ (a v těle metody také proměnná „newOrder“) představuje pořadí hráčů pro zbytek daného štychu. Při každém sestupu stromem se buďto odebere první z pořadí (tzn. aktuální hráč) nebo se, v případě, že štych už je u konce, vygeneruje nové pořadí pro další kolo.

Proměnná minmax určuje, zda se algoritmus právě nachází ve fázi minimalizace nebo maximalizace zisku. V algoritmu MAPP je stav hry ohodnocován vůči volícímu hráči. Pokud je tedy algoritmus v hladinách volícího hráče, hledá se maximum z hodnot synů a minmax je tedy roven jedné. V případě nevolících protihráčů je minmax nastaven na mínus jedna.

Rekurzivní prohledávání stromu se zastaví při dosažení zvenčí nastavené hodnoty „AILevel“, která má představovat výhled, a tedy i inteligenci, protihráčů.

POZNÁMKA: Ve výše uvedeném algoritmu je kvůli jeho zjednodušení opomenut jeden důležitý detail. Vzhledem k tomu, že se pravidelně nestřídají hladiny hledající maximum z potomků s hladinami hledajícími naopak jejich minimum, není možné na každé úrovni provádět alfa-beta prořezávání. Například v mariáši totiž mohou jít až čtyři „minimální“ hladiny za sebou (v případě, že nevolící hráči zakončují daný štych a zahajují štych následující). V tomto případě by alfa-beta prořezávání mohlo způsobit ztrátu některých platných možností. Program by tak mohl vynechat i některou z větví stromu na cestě k optimálnímu tahu.

4.1.2 Generátor tahů

Generátor tahů je procedura modelující možný vývoj hry pro účely stavby prohledávacího stromu. V případě karetních her není problémem stavět strom hry v hladinách odpovídajícím tahům aktuálního hráče. Jeho karty totiž máme právoplatně k dispozici. Potíže nastávají při pokusech generovat možné tahy ostatních.

Tento problém můj program řeší tím, že na úrovních protihráčů nepracuje s konkrétními kartami (jako je například srdcový spodek). Místo toho algoritmus simuluje jednotlivé možné reakce protihráčů na obecnější úrovni a pro takto vzniklé potenciální stavy potom počítá pravděpodobnosti, s jakou mohou nastat. Tato pravděpodobnost je určována na základě již odehraných karet, karet v ruce aktuálního hráče a také na základě předcházející hry protihráčů. Vypočítanou pravděpodobností se poté vynásobí ohodnocení jednotlivých stavů (dojde tedy ke zmenšení hodnot stavů - pravděpodobnost je vždy menší nebo rovna jedné).

Celá procedura bude demonstrována na následujícím PŘÍKLADU:

SITUACE:

Hraje se klasický mariáš, ale pro jednoduchost jen ve dvou hráčích.

Program nyní sestavuje podstrom stavu, kdy aktuální hráč (jehož AI program právě simuluje) právě zahrál kulovou devítku.

Následující hráč v dosavadním průběhu hry již stihl uhrát dvě desítky (= 2 bodově ohodnocené karty, tzn. esa nebo desítky).

Aktuální hráč uhrál pouze jednu desítku.

Ohodnocovací funkce jednoduše počítá rozdíl uhraných desítek soupeřících stran, kde výsledek je kladný, pokud více desítek uhrál volící hráč. Řekněme, že aktuální hráč je zároveň i hráčem volícím.
Aktuální ohodnocení stavu hry je tedy -1 (= 1 – 2)

Možné stavy, které by generátor mohl napojit na aktuální stav do další hladiny stromu (tzn. SEZNAM MOŽNÝCH TAHŮ PROTIHRÁČE):

1) protihráč zahraje nižší kulovou kartu
2) protihráč zahraje vyšší kulovou kartu

3) protihráč zahraje vyšší ohodnocenou (tedy v mariáši eso nebo desítku) kulovou kartu
4) protihráč přebije trumfem
5) protihráč přebije ohodnoceným trumfem
6) protihráč nemá ani kule ani trumfy --> může zahrát libovolnou kartu --> zahraje neohodnocenou
7) protihráč nemá ani kule ani trumfy --> může zahrát libovolnou kartu --> zahraje ohodnocenou

VÝSLEDKY FUNKCE POČÍTAJÍCÍ PRAVDĚPODOBNOST JEDNOTLIVÝCH STAVŮ:
(Předpokládejme například, že už se odehrálo hodně kulí, včetně esa a desítky; trumfy naopak téměř ještě nešly...):

1) 8%
2) 10%
3) 0% (eso i desítka už šly --> tento stav se tedy ze stromu vyřadí, respektive vůbec nedojde k jeho zařazení...)
4) 32%
5) 15%
6) 6%
7) 4% (Chytrý protihráč by samozřejmě nikdy „nenamazal“ svému protivníkovi hodnocenou kartu, pokud by mohl jinak. Funkce ovšem počítá pravděpodobnost, zda k situaci vůbec může dojít, ne zda si ji hráč opravdu zvolí. Samotnou volbu zajistí až ohodnocovací funkce ve spolupráci s minimaxem.)

POZNÁMKA: Program musí zohlednit všechny možné způsoby, jakými se partie může vyvíjet. To však ale neznamená, že by součet pravděpodobností jednotlivých tahů měl být roven 100% (typicky tomu tak není). Pravděpodobnost zahrání každé z možných karet se odvíjí od množství neodehraných karet dané barvy v poměru k množství všech neodehraných karet. U tahů, ve kterých nedošlo k přiznání barvy nebo trumfu, se musí také zohlednit pravděpodobnost, s jakou nedojde k zahrání tahu legálního (například pravděpodobnost zahrání trumfu musím zmenšit o šanci, s jakou by mohl daný hráč přiznat barvu).

VÝSLEDKY OHODNOCOVACÍ FUNKCE:
Všechny stavy měly implicitní hodnotu -1 (= hodnota otcovského vrcholu v prohledávacím stromě). Ve stavech 3 a 5 se hodnota sníží (protihráč uhraje další desítku, tzn. stav hry se z hlediska aktuálního hráče zhorší) o číslo [1 * pravděpodobnost_daného_stavu]. Ve stavu 7 se hodnota naopak zvýší.

1) -1
2) -1
3) -1 + 0 * (-1) (tento stav však ve stromě ve skutečnosti nebude!)
4) -1
5) -1 + 0,15 * (-1) = -1,15
6) -1
7) -1 + 0,04 * 1 = -0,96

Tento generátor tahů má však ještě pár nedostatků.

Hlavním z nich je fakt, že generuje poměrně velké množství možných tahů protihráče. Toto množství se dá několika úpravami algoritmu beze ztráty jakýchkoli možností snížit a tím samozřejmě i zrychlit běh programu (bude se prohledávat menší strom).

Klíčem k takovémuto bezeztrátovému ubírání možností je slučování podobných tahů.

Z výše uvedeného příkladu by se možná nabízelo sloučení tahů 1) a 2). Informace o tom, kdo zahrál nejvyšší kartu štychu, však určuje pořadí hráčů pro další kolo, a proto nesmí být takovýmto sloučením vypuštěna.

Je však možné sloučit takové tahy, které se navzájem liší pouze „ohodnocením“ zahrané karty. V příkladu jde například o tahy 4) a 5). Dosáhneme toho jednoduchou úpravou generátoru, kdy pro každý takovýto „dvojtah“ určíme hodnotu hrané karty jako poměr počtu hodnocených neodehraných karet dané barvy k počtu všech neodehraných karet.

Další nedostatek generátoru je tak trochu diskutabilní a jeho odstranění by vedlo naopak ke zvětšení počtu možných tahů. Jde o to, že algoritmus z příkladu shrnul karty, které ani nepřiznaly barvu ani nebyly trumfové, do pouhých dvou případů (který se navíc dá redukovat na pouhý jeden „dvojtah“, zmiňovaný o odstavec výše). Nezabýval se totiž tím, jakou bude mít zahraná karta barvu. Pouze se omezil na konstatování, že nepůjde ani o výchozí barvu ani o trumf. Tento postup má své výhody i nevýhody.

Výhodou je samozřejmě prostý fakt, že možností je méně a program tudíž bude o něco rychlejší. A protože v mariáši se dá přebít pouze trumfem nebo přiznáním barvy, na barvě podlézavší karty tedy vlastně ani nezáleží.

Nevýhodou je ale to, že algoritmus se má pokoušet o rozšiřitelnost na další karetní hry, ve kterých by už ztráta informace o potencionálně zahrané kartě mohla vadit.

Nevýhodou by se také mohlo zdát, že algoritmus při takovéto ztrátě informace nemůže upřesňovat své povědomí o počtu odehraných karet dané barvy

a ztrácí tedy na inteligenci. Nesmíme však zapomínat na to, že veškeré takto generované tahy jsou pouze hypotetické. Pokud si totiž program zaznamenával všechny tyto, ve své podstatě anonymní, tahy, docházelo potom k situacím, kdy se o některých hráčích domníval, že už žádnou barvu mít nemohou, přestože tito hráči ještě prokazatelně měli karty na ruce.

Po zvážení těchto důvodů jsem se nakonec rozhodl v mém generátoru tahů barvu podlézavší karty rozlišovat, a to především kvůli orientaci na „obecnost“ algoritmu.

Upravený generátor tahů tedy uvažuje následující možné situace:

1) protihráč zahraje nižší kulovou kartu
2) protihráč zahraje vyšší kulovou kartu
3) protihráč přebije trumfem
4) protihráč nemá ani kule ani trumfy --> zahraje první ze zbylých barev
5) protihráč nemá ani kule ani trumfy --> zahraje druhou ze zbylých barev

POZNÁMKA: Všechny z výše uvedených karet budou mít hodnotu odvozenou z poměru neodehraných hodnocených a nehodnocených karet dané barvy.

4.1.3 Ohodnocovací funkce

Podobně jako generátor tahů je ohodnocovací funkce silně závislá na typu hrané hry.

Zatímco u některých her je účelem zbavit se co nejrychleji karet na ruce (prší, černý Petr), u jiných záleží na množství uhraných štychů a u dalších zase na hodnotě vyložených karet (oko, kanasta).

V případě Trick Taking Games, mezi něž patří i mariáš, se však situace zdá být poměrně jednoduchou. Pokud problém zjednodušíme o hlášky (v jejichž uhrání danému hráči stejně nemůže nikdo zabránit, a tudíž je v rozhodovacím procesu algoritmu ani není třeba uvažovat) a o poslední štych (výherci posledního štychu se ohodnocení stavu hry jednoduše zvýší o jedna), jde v mariáši v zásadě jen o počet uhraných hodnocených karet (desítek a es). Problém ohodnocování stavu hry však naráží na několik menších úskalí.

Prvních z nich je nutnost ohodnocovat odehrané tahy až po skončení daného kola. Tento problém byl nastíněn už v kapitole 4.1.1 a nepředstavuje žádné vážnější úskalí. Při uložení odehraného (všichni hráči už zahráli svou kartu) kola do rozhodovacího stromu hry se určí vítěz štychu, přičte se mu skóre za ohodnocené karty v daném kole a celý výsledek se přičte ke skóre z předchozího průběhu hry.

Druhým problémem je existence hypotetických tahů, kdy je zahraná karta pouze uvažovaná - tudíž její ohodnocení nemusí být celočíselné a je nutné uvažovat pravděpodobnost jejího zahrání.

Funkce proto pro jednotlivé karty zahrané v daném kole vynásobí jejich ohodnocení pravděpodobností jejich zahrání, tato čísla sečte, vynásobí výsledek hodnotou –1, pokud štych bere tým nevolícího hráče, a výsledek přičte k ohodnocení předchozího průběhu hry.

Dalším úskalím je hra na sedmičku. Z tohoto důvodu funkce upravuje ohodnocení stavu hry podle toho, jestli už došlo k odehrání trumfové sedmičky. Stav hry je upravován přičítáním/odečítáním předem definovaných konstant, jejichž výše záleží na tom, zda některý z hráčů hlásil hru na sedmičku.

Posledním problémem je ohodnocování stavu hry v případě hraní betlu či durchu. V tomto případě je však řešení poměrně jednoduché. Dokud volící hráč zvládá přebíjet (u durchu) nebo naopak podlézat (betl) všechny štychy, ohodnocení zůstává nulové. Naopak v případě tahů, kdy by mělo dojít k narušení tohoto jeho snažení, se ohodnocení změní o nějakou předem definovanou (a zpravidla poměrně vysokou) konstantu, pochopitelně upravenou o pravděpodobnost, s jakou může k onomu tahu skutečně dojít.

4.1.4 Vývoj funkce MAPP

Když však algoritmus bude napsán tak, jak je uvedeno výše, bude se ve skutečné hře chovat velice zvláštně.

Opět totiž dojde ke střetu s prostou skutečností, že tahy neaktuálních hráčů ve stromu hry jsou pouze hypotetické a skuteční hráči je mnohdy vůbec nemají šanci

zahrát. Proto bude oblíbeným tahem tohoto algoritmu například zahájení štychu desítkou a spoléhání na to, že bude jejich spoluhráčem přebita a jejich tým tak získá rovnou dvacet bodů.

Tento problém je poměrně obtížně řešitelný a jeho úplná eliminace je ve spojení s algoritmy pro hry s úplnou informací nejspíše neproveditelná. Šance, že k takovýmto rizikovým tahům bude docházet, se však dají výrazně zmenšit.

Nejdůležitější z kroků pro zlepšení algoritmu je postižen v samotném jeho názvu. Program totiž zatím hledá nejvyšší pravděpodobnost zisku, jeho název však propaguje nejvyšší průměrnou pravděpodobnost zisku. A v průměrování výsledků jednotlivých podstromů aktuálního uzlu se skutečně skrývá klíč ke značnému zvýšení použitelnosti algoritmu.

Kdyby měl program k dispozici úplnou informaci o hře, pak by si skutečně mohl dovolit pracovat jen s nejlepšími (nebo nejhoršími, podle toho, zda jde o aktuálního hráče) stavy hry a vracet coby návratovou hodnotu funkce MAPP hodnotu extrému z ohodnocení podstromů. Problém však spočívá v tom, že tento ideální extrém nemusí být ve skutečné hře uskutečnitelný. Některý z tahů, který tento extrém předpokládá, totiž vyžaduje zahrání karty, kterou by daný hráč sice mohl mít, ale jednoduše nemá.

Proto je nezbytné nebrat v úvahu jen ty nejvyšší, či nejnižší, hodnoty, ale udělat průměr ze všech možností. Program tak bude brát v úvahu nejenom tu možnost, že zahranou desítku přebije svým esem spoluhráč, ale také méně příjemnou, ale přesto skutečnou, možnost, že celý štych, a tedy i onu desítku, vyhraje protihráč.

V tomto bodě však celý algoritmus začíná kolidovat s myšlenkou alfa-beta prořezávání. To sice eliminuje některé z „nevýhodných“ větví podstromu a tím celý program zrychluje, ovšem zároveň tak tyto větve vyčleňuje z počítání průměru, který je pro správný chod algoritmu nezbytný. A protože jsou tyto prořízнuté větve navíc pro aktuálního hráče těmi méně atraktivními, dochází k tomu, že použitelnost daného tahu se zvyšuje a zároveň s ní stoupá i výskyt zmíněných rizikových tahů.

Proto je u MAPP algoritmu alfa-beta prořezávání volitelné a jeho použití se spíše nedoporučuje. Přesto se však dá v případě potřeby v programu zapnout.

Po těchto úpravách vypadá funkce MAPP, ve své konečné podobě, takto:

```
float MAPP(depth, alfa, beta, playersOrder, /*ODKAZEM*/result) {
   int player = getFirst(playersOrder);
   int minmax = ( (MinOrMax(player)) ? +1 : -1 );
   int pocet = 0;
   float soucet = 0;

   if (GameEnded() OR depth == getAILevel())
       return EvalGameState();

   moves <- GenerateMoves();
   removeFirst(playersOrder);
   ForEach moves
   {
       MakeNextMove();
       if (TurnEnded()) {
           newOrder = getNewPlayersOrder();
           val = minmax * MAPP(depth + 1,-beta,-alpha,newOrder,result);
       }
       else
           val = minmax * MAPP(depth,-beta,-alpha,playersOrder,result);
       UnmakeMove();
       pocet++;
       soucet = soucet + val;
       if (val > alpha) {
           alpha = val;
           if (depth == -1)     // rekurze se vratila do korene
               result.markMove();      // oznacime novy nejlepsi tah
       }
       if ( val >= beta && isPrunning() )
           break;
   }
   return (minmax * soucet/pocet);
}
```

4.2 Rozšiřitelnost algoritmu

Na to, že je celý algoritmus MAPP alespoň teoreticky rozšiřitelný, poukazuje už samotný fakt, že je bez jakýchkoli úprav použitelný jak na klasický mariáš, tak i na betl a durch.

Co se funkce MAPP týče, je použitelná pro jakoukoli karetní hru a vlastně i pro hry s úplnou informací (v případě her s úplnou informací však má oproti

minimaxu a jeho klonům značné nevýhody v podobě průměrování a problémům s alfa-beta prořezáváním).

Horší je to už s generátorem tahů. Datové struktury, které popisují vlastnosti hry, jsou kostruované tak, aby byly v případě potřeby rozšiřitelné i na další hry. Algoritmus tedy je připraven na změny typu jiného počtu hráčů nebo změny hracího balíčku.

Hry, ve kterých se podstatně liší samotný průběh partie, by však vyžadovaly výraznější změny v datových strukturách, v generátoru tahů, ohodnocovací funkci, v grafickém uživatelském rozhraní (které by mělo být teoreticky připraveno na jednoho až čtyři hráče) a v dalších aspektech programu. Tyto změny by samozřejmě vedly k výraznému zesložitění algoritmu pro generování tahů a tedy i ke zpomalení celého programu.

Čím obecnější by tedy program byl, tím hůře by si poradil s jednotlivými hrami. To je však naprosto samozřejmé. Tento program si také nikdy nedával za cíl předstihnout v rychlosti a efektivitě specializované aplikace. Z toho důvodu jsem se také co nejdůsledněji vyhýbal používání všemožných herních heuristik, které by sice program z hlediska mariáše výrazně zefektivnily, ovšem z hlediska rozšiřitelnosti by šlo o krok zpět.

Obecně se domnívám, že by si algoritmus MAPP s jistými úpravami měl poradit se hrami typu Trick Taking Games, tedy se hrami na štychy. Takovýchto her je však na světě velké množství, podle serveru www.pagat.com jde dokonce o jednu z největších skupin karetních her.

4.3 Výhody a nevýhody algoritmu

Výhodou programu je především výše zmíněná rozšiřitelnost. Snahy o obecnou použitelnost algoritmu na karetní hry jako takové s sebou však také přinesly mnohé nevýhody.

Ve srovnání s aplikacemi specializovanými na hraní mariáše je tento program méně efektivní a občas předvádí rizikové tahy, které by si živí hráči většinou

nedovolili. Program je sice schopen hrát a vyhrávat i se zkušenými hráči, specializovaným algoritmům však může jen obtížně konkurovat.

Z tohoto důvodu program rezignoval na některé méně důležité aspekty mariáše, jako je licitování, flekování nebo počítání skóre. Pro implementaci těchto rozšíření je sice v programu připravena půda, ale v českých končinách už existuje několik freewarových aplikací, které tyto aspekty podporují a díky své specializaci si s nimi poradí efektivněji.

Kapitola 5
Shrnutí

V této práci jsme prezentovali program implementující známou karetní hru mariáš.

Při vyvíjení programu jsme se pokoušeli o obecnější přístup, který by umožňoval rozšíření na hry podobného typu. Podrobně jsme popsali problematiku takového úkolu a navrhli algoritmus Maximum of Average Probable Profit, neboli MAPP, který je schopen náš problém řešit.

Algoritmus MAPP má potenciál implementovat velké množství různých druhů karetních her. V současné formě se však zdá být vhodný především pro tzv. Trick Taking Games a po dalších úpravách by měl být schopen hrát a vyhrávat veškeré hry z této skupiny karetních her, jejíž příslušníci a příslušnice se již po mnoho století hrají na všech kontinentech.

Tento algoritmus vychází z algoritmů pro řešení her s úplnou informací, kterým jsme se v této práci podrobněji věnovali.

Vzniklý program jsme srovnali s jinými existujícími aplikacemi podobného účelu a kriticky zhodnotili výhody a nevýhody jednotlivých přístupů.

Literatura

Card games web site, http://www.pagat.com/

http://marias.webz.cz/

http://www.zzzz.cz/marias/

http://en.wikipedia.org/wiki/Minimax

Příloha A

Uživatelská dokumentace

A.1 Instalace a spuštění aplikace

Mariášový trenažér je možné spustit třemi různými způsoby.

V operačním systému MS Windows bude tím nejobvyklejším asi instalace aplikace z instalačního balíčku Cards_windows_1_0_1.exe. Spuštěním tohoto souboru se otevře klasický Windows Install Wizard, který nainstaluje program na počítač uživatele, a to včetně distribuce JRE (Java Runtime Enviroment) nutné pro spouštění programů v jazyce Java.

Dalším působem je spouštění aplikace ze souboru Cards.jar. Tento soubor je ve formátu Java Archiver a na počítači s distribucí JRE jeho otevření (dvojclickem nebo příkazem „java -jar Cards.jar“) způsobí rozbalení a spuštění programu. Tento způsob nevyžaduje žádnou instalaci ani žádný další prostor na disku (jen dočasně při spouštění souborů, kdy se celý archiv musí skrytě rozbalit).

Posledním způsobem, doporučovaným pouze pokročilejším uživatelům, je spuštění programu přímo v NetBeans IDE - vývojovém prostředí, ve kterém byla aplikace naprogramována. V tomto projektu je uživateli volně přístupný i zdrojový kód programu.

A.2 Pravidla Mariáše

Pravidla, která uvádím v této práci, jsou výtahem nejpodstatnějších bodů z pravidel voleného mariáše. Neobsahují některé méně podstatné aspekty hry (flekování, hlášení sedmy a stovky proti…), kterými se ve své práci nezabývám.

1. Hraje se ve třech hráčích vždy po směru hodinových ručiček.
2. Rozdává se ve směru hodinových ručiček, volícímu hráči (= tzv. „forhont“, hráč sedící vlevo od rozdávajícího) sedm karet, dvakrát po pěti kartách a dále třikrát po pěti kartách. Forhont se smí podívat jen na prvních sedm karet. Ostatní hráči mají okamžitě k dispozici všechny své karty.
3. Po rozdání volí forhont tzv. „trumfy“ - tedy hlavní barvu, jejíž všechny karty automaticky přebíjejí karty ostatních barev (nezávisle na jejich hodnotě). Forhont volí z prvních sedmi karet nebo naslepo (tzn. hráč náhodně vybere jednu z karet, dokud ještě nebyly otočeny lícem nahoru) z druhých pěti karet - takzvaně "z lidu". Po zvolení trumfu si forhont může vzít i své zbylé karty.
4. Poté je volící hráč povinen dát dvě karty do tzv. „talonu“, tzn. stranou na stůl, lícem dolů. Pokud se hraje klasická hra (ne betl nebo durch) a v talonu se nachází esa nebo desítky, přičítá se hodnota těchto karet do skóre nevolících hráčů.
5. Po zvolení talonu volí forhont typ hry. Existují tři základní možnosti. Hráč může zvolit základní hru (nebo některou z jejích variant - bonusů), ve které se hraje na trumfy a cílem je posbírat více bodů než oba protihráči dohromady. Další možností je tzv. betl, ve kterém je cílem nezískat jediný štych. Poslední možností je protipól betlu, durch, kde je k vítězství nutné uhrát naopak všechny štychy.
6. Bonusy ke klasické hře: Hráč se při volení klasické hry nemusí spokojit s obyčejnou hrou. Může nahlásit také určité „lepší“ hry, které při hře o peníze zvyšují cenu partie. Tyto bonusy, seřazené odzdola podle ceny, jsou: sedma, sto a stosedm. Při hře na sedmu (sto a stosedm) musí hráč vyhrát poslední štych trumfovou sedmičkou. Při hře na stovku musí získat alespoň sto bodů. Tyto lepší hry mohou uhrát stejným způsobem i nevolící protihráči, jde pak o takzvanou „hru proti“ (např. sedma proti). Pokud libovolná ze soupeřících stran některý z bonusů uhrála, aniž by ho nahlásila, jde pak o tzv. „tichou“ výhru (např. tichá stovka).
7. Po zvolení hry se forhont ptá ostatních na jejich kartu. Pokud zvolil betl nebo durch, oznámí to protihráčům. V případě klasické hry se tradičně ptá „barva?“, čímž oznamuje, že chce hrát klasickou hru nebo některou z jejích bonusů. Protihráči postupně odpovídají, a to buď „dobrá“, pokud jim hra vyhovuje nebo nemají lepší kartu, anebo v opačném případě mohou hru „ukrást“ (přičemž

klasická hra se dá krást na betl i durch, betl jde ukrást jen na durch a durch nejde krást vůbec). Pokud dojde k ukradení hry, kradoucí hráč se stává novým volícím hráčem (pořadí rozdávání však zůstává stejné). Vezme si talon, zvolí ze svých karet talon nový a oznámí, jakou hru chce hrát. Znovu se opakuje dotazovací ceremoniál a je možné i další kradení (tentokrát už jen případně z betlu na durch).

8. Po konečné dohodě hráčů na typu hrané hry oznámí volící hráč barvu trumfů, pokud se na ně hraje, a začne partii vynesením první karty.
9. Samotná hra sestává z jednotlivých štychů, tedy tří karet položených jednotlivými hráči postupně na stůl. Daný štych vyhrává buďto nejvyšší (pořadí hodnot karet viz. kapitola 2.1) zahraný trumf, anebo, v případě, že žádné trumfy nebyly zahrány, nejvyšší karta startovní barvy.
10. Hráči jsou povinni přiznávat barvu, tzn. zahrát kartu stejné barvy, jakou měla startovní karta štychu. Pokud nemají žádnou kartu startovní barvy, musí zahrát trumf. Pokud nemají ani ten, mohou hrát karty libovolné ze zbylých barev.
11. Hráči jsou povinni přebíjet, pokud to není v rozporu s pravidlem o přiznávání barvy.
12. Hraní hlášek: Pokud se hraje některá z variant klasické hry a libovolný z hráčů má při hraní filka na ruce i krále stejné barvy, uhrál tzv. „hlášku“. Filek se poté položí k hráči na stůl lícem nahoru a má hodnotu dvaceti bodů nebo dokonce čtyřiceti, pokud jde o hlášku trumfovou.
13. Hráč, který v klasické hře vyhraje poslední štych, si přičítá deset bodů.
14. Na konci klasické hry se sečtou body jednotlivých hráčů. Nevolící hráči sečtou své body dohromady. Vyhrává ten tým, který dosáhl více bodů.
15. Betl končí prohrou volícího hráče ve chvíli, kdy vyhrál svůj první štych. Výhrou pro něj končí, pokud hráč dokázal všechny své štychy prohrát.
16. Durch končí prohrou volícího hráče ve chvíli, kdy prohrál svůj první štych. Výhrou pro něj končí, pokud hráč dokázal všechny své štychy vyhrát.

A.3 Ovládání aplikace

Formulář aplikace je rozdělen do čtyř hlavních částí.

Ve spodním díle se nachází hrací prostor živého hráče, uživatele aplikace. V levé a pravé části se nachází prostor obou CPU protihráčů. V prostorech všech hráčů se vykreslují karty na jejich ruce, dále také případné uhrané hlášky, vyřčené dotazy na průběh hry a v neposlední řadě jméno daného hráče.

Uprostřed formuláře je samotné hrací pole. V tomto prostoru se vykreslují především karty odehrané v daném štychu. Také se zde vypisují informace o průběhu a typu aktuální partie. V případě volení typu hry uživatelem aplikace se ve spodní části objevuje výběrový seznam, ze kterého hráč volí druh hry pro danou partii.

Uživatel aplikace do hry zasahuje především prostřednictví výběru karet na ruce. Karty se vybírají pomocí kliknutí myši na zvolenou kartu. Aby se minimalizovalo riziko „ukliknutí“ hráče, proběhne po prvním kliknutí na obrázek karty pouze povytažení této karty z řady ostatních. Pokud chce hráč skutečně zahrát danou kartu, je nutné na povytaženou kartu kliknout podruhé, což vede k jejímu zahrání a k případným reakcím protihráčů.

Kvůli tomu, aby si hráč mohl každý dohraný štych dostatečně prohlédnout a případně zapamatovat odehrané karty, nedochází (po určité časové prodlevě) k automatickému zahájení dalšího kola. Místo toho hráč iniciuje začátek dalšího štychu pomocí tlačítka „new turn“, které se po odehrání všech tří karet daného kola objevuje uprostřed formuláře. Pro skončení celé partie se na stejném místě vykresluje tlačítko „next game“, které po stisknutí zahajuje další partii mariáše s novým volícím hráčem.

Pokud hráč chce ukončit hru nebo zahájit další partii ještě před skončením aktuální hry, může tak učinit pomocí volby „Game“ na horní liště aplikace nebo pomocí klávesových zkratek uvedených u jednotlivých voleb.

Pomocí menu hry na horní liště aplikace se dá také měnit nastavení některých z parametrů programu. Po zvolení položky menu „Settings“ - „Change“ se otevře dialog, ve kterém může uživatel nastavit jména jednotlivých hráčů a také úroveň

inteligence svých CPU oponentů. Pomocí checkboxu „pruning“ může hráč vypnout nebo zapnout algoritmus alfa-beta prořezávání, který zkracuje dobu výpočtu při volbě tahu protihráčů, ovšem za cenu zvýšení rizika určitých riskantních tahů.

Ve výběrovém seznamu „AI intelligence“ se nastavuje výhled CPU protihráčů, tedy množství štychů, které se tito umělí oponenti snaží předvídat dopředu. Při zvolení nižších hodnot se snižuje chytrost protivníků, odměnou za to je však zkrácení doby jejich výpočtů. Vyšší hodnoty mají efekt zcela obrácený.

Obecně se majitelům pomalejších počítačů doporučuje výhled rovný dvěma, který v závislosti na dané situaci představuje předvídání 2 – 7 zahraných karet. Majitelé středně rychlých strojů si mohou dovolit výhled rovný třem kolům (5 – 10 karet) a v případě těch nejrychlejších je možný i výhled na čtyři štychy dopředu (8 – 13 karet).

Příloha B

Programátorská dokumentace

Tato aplikace byla vytvořena v programovacím jazyce Java, konkrétně ve vývojovém prostředí NetBeans IDE 5.0 v prostředí operačního systému MS Windows XP.

B.1 Rozvržení programu

Zdrojový kód tohoto programu je členěn do několika souborů podle jeho nejvýznamnějších tříd.

Těchto tříd je devět a každá z nich implementuje některou z hlavních datových struktur programu. V některých případech není obsahem souboru pouze jedna jediná hlavní třída, ale ještě několik menších pomocných tříd, které s daným problémem úzce souvisí a jsou proto zahrnuty do téhož souboru.

Hlavní metoda main, vyvolávaná po spuštění programu jako první, se nachází ve třídě CardsMain. CardsMain je tedy hlavní třídou programu a skrze ni dochází k užívání všech devíti tříd.

Třída Random je jen jednoduchým generátorem náhodných celých čísel a nebudeme se jí v této práci dále věnovat.

Třída SwingWorker je implementací třetí verze třídy SwingWorker.java. Jde o abstraktní třídu sloužící k práci spojené s uživatelským rozhraním v samostatném vlákně. Dále se jí v této práci nebudeme samostatně věnovat.

B.2 Datové struktury

Jednotlivé třídy, a datové struktury v nich obsažené, jsou popisovány pokud možno hierarchicky, od těch nejzákladnějších, využívaných ostatními vyššími třídami, až po hlavní třídu CardsMain.

Card.java

Tato třída implementuje jednu konkrétní kartu z daného karetního balíčku.

Každá karta obsahuje řetězec popisující její celé jméno (např. spodek-srdce). Podle těchto jmen jsou pojmenovány i obrázky konkrétních karet používané v GUI programu, takže je důležité udržovat tyto názvy synchronizované.

Dále je karta popsána pomocí číselného indexu své barvy. Každá z barev má totiž v programu pevně přidělenou celočíselnou hodnotu, kterou popisuje třída Mariáš ze souboru GameSettings.java.

Dalším atributem každé karty je celočíselná hodnota udávající výšku karty vzhledem k přebíjení. Nejnižší hodnotu má tedy sedmička a nejvyšší eso.

Posledním údajem je cena karty používaná pro určení aktuálního stavu a výsledku hry. Při hraní betlu a durchu mají všechny karty tuto hodnotu rovnou nule. V případě klasického mariáše mají desítky a esa hodnotu rovnou jedné. Tato hodnota je určena reálným číslem, protože při generování tahů vznikají anonymní zahrané karty, jejichž cena je určována poměrně k počtu neodehraných desítek (viz kapitola Generátor tahů).

CardListClass.java

Tato třída implementuje balíček hracích karet pomocí dvourozměrného pole typu Card.

Třída také obsahuje metody pro určování počtu ohodnocených a neohodnocených karet v balíčku, potřebných pro určování neceločíselných hodnot karet v generátoru tahů.

PlayerClass.java

Jak je z názvu patrné, slouží tato třída především k implementaci jednotlivých hráčů.

Každá instance hráče musí být pro účely hry určena jednoznačným ID. V programu je živému hráči přisouzeno ID rovno nule. Ostatní z hráčů mají ID inkrementováno o jedna podle směru, v jakém jsou „usazeni u hracího stolu“.

Dalšími atributy hráče je jeho jméno a také informace o tom, jaké barvy ještě daný hráč může mít na ruce. Pokud totiž hráč někdy ve hře nepřizná barvu, poznamená si to program pomocí změny daného příznaku.

Nejdůležitější informací o každém hráči jsou však karty na ruce. Ty jsou implementovány pomocí třídy CardsInHandClass. Karty jsou rozděleny podle barev a každou z těchto barev reprezentuje jeden LinkedList – kontejner vhodný pro sekvenční procházení prvků (v programu se totiž obvykle procházejí postupně všechny karty dané barvy na ruce hráče). Jednotlivé LinkedListy jsou uspořádány do ArrayListu – kontejneru vhodného pro náhodný přístup.

GameSettings.java

Tato třída v sobě ukrývá veškeré informace o typu hrané hry a má proto zásadní význam pro rozšiřitelnost programu.

GameSettings v sobě nesou výše uvedené objekty typu balíček karet a jednotliví hráči.

Tato třída stručně řečeno obsahuje informace o veškerém nastavení dané karetní hry i konkrétní partie, od výhledu CPU protihráčů až po aktuálně volícího hráče.

Důležitá je metoda initialize, která vždy na začátku nové partie vygeneruje a nastaví karty na ruce hráčů.

V souboru GameSettings.java se nachází také menší třída Mariáš, která v sobě nese údaje o barvách a názvech jednotlivých dvaatřicítkových mariášových karet.

StackClass.java

Tato třída slouží k implementaci zásobníku, který pro algoritmus umělé inteligence představuje prohledávací strom hry.

Jednotlivé prvky zásobníku představují uzly stromu hry v tom pořadí, v jakém jsou algoritmem postupně prohledávány. Přechody mezi jednotlivými hladinami stromu jsou zaznamenány v menší pomocné datové struktuře – v poli celých čísel označujících místa na zásobníku, ve kterých dochází k přechodu na další úroveň myšleného prohledávacího stromu. Mezi každými dvěma takto označenými místy na zásobníku se tedy nacházejí synové uzlu, který je v předchozí skupince na posledním místě.

Jelikož v mariáši je možné ohodnotit výsledek štychu až po zahrání všech tří karet, musí každý uzel stromu hry (což ve skutečnosti znamená každý prvek zásobníku) nést plnou informaci o vývoji štychu.

Každý prvek zásobníku tedy představuje jedno kolo partie, ať už po zahrání všech karet nebo zatím jen rozehrané. Tyto objekty implementuje třída TurnClass, která sama sestává ještě ze tří menších podobjektů, reprezentujících jednotlivé tahy. Každý z těchto tahů je instancí třídy MoveClass a obsahuje informaci o zahrané kartě, o tom, kdo ji zahrál a s jakou pravděpodobností mohlo k této události dojít.

Gameplay.java

Třída Gameplay zajišťuje fungování algoritmu umělé inteligence.

Obsahuje metody implementující algoritmus MAPP, generátor tahů, ohodnocující funkci a funkce ExecVirtualTah a ExecReverseVirtualTah, sloužící pro virtuální provedení a zpětné od-provedení zkoumaných tahů.

Všechny tyto metody úzce spolupracují s vlastnostmi a specifiky hrané hry. Z tohoto důvodu je jednou z nejdůležitějších proměnných třídy Gameplay instance třídy GameSettings.

CardsMain.java

CardsMain je hlavní třídou celé aplikace. Řídí běh programu a iniciuje vznik instancí všech jeho ostatních tříd.

Tato třída se stará o vytvoření a chod grafického uživatelského rozhraní a jako jediná je tedy v přímé interakci s uživatelem.

B.3 Grafické uživatelské rozhraní

GUI tohoto programu je vytvořeno pomocí grafického uživatelského prostředí AWT (Abstract Window Toolkit).

Hlavní formulář aplikace ve své podstatě napodobuje stůl, na kterém se hraje karetní partie a kolem kterého jsou usazeni jednotliví hráči. Formulář je proto rozdělen na čtyři menší panely. Tři z nich představují „hrací prostory“ jednotlivých hráčů. Poslední, který je umístěn v prostřední části formuláře, představuje zmiňovaný stůl.

Nejpodstatnější částí hracích prostorů hráčů je plátno (Canvas), na které se vykreslují karty na ruce daného hráče. Pro grafické znázornění karet jsem použil obrázky klasických českých jednohlavých karet ve formátu PNG.

Karty CPU protihráčů musí pro uživatele aplikace zůstat utajené, a proto se vykreslují lícem dolů. Tyto karty nejsou s uživatelem v žádné interakci, jediným úkolem plátna je po každé zahrané kartě protihráče vykreslit na jeho ruce o jednu kartu méně.

Pomocí karet na ruce živého hráče naopak probíhá většina interakce uživatele s programem. Hráč volí ze svých karet pomocí myši, což je hráči naznačováno změnou kurzoru při najetí nad prostor plátna. Všechny tyto obrázky jsou přednatahovány pomocí rozhraní ImageObserver, které zároveň monitoruje úspěšnost načtení jednotlivých souborů. Pokud se některý z obrázků nepodaří načíst, program o tom informuje a ukončí svoji činnost.

Vlevo od karet na ruce uživatele, v případě CPU protihráčů pod jejich kartami, se nachází pole pro zobrazování uhraných hlášek daného hráče. Hlášky jsou symbolizovány tradičně obrázkem filka dané barvy položeného na stůl lícem vzhůru.

Poslední součástí panelů, symbolizujících hrací prostory hráčů, je plátno vykreslující výroky hráčů. Toto plátno je používané například u příležitosti hlášení typu hry.

Ve střední části formuláře se nachází zmiňovaný virtuální hrací stůl. V jeho horní části figuruje textové pole (TextArea) informující o dosavadním průběhu hry. Pod ním se nachází návěští (Label) vypisující pokyny pro hráče a informující o tom, jakou akci právě aplikace provádí nebo očekává.

Nejdůležitější součástí tohoto panelu však jsou tři plátna určená pro zobrazování karet zahraných v aktuálním štychu.

Další součástí „hracího stolu“ je pole informující o druhu právě probíhající partie. Při volení druhu hry uživatelem se na panelu zobrazí také výběrový seznam (Choice), ze kterého má hráč možnost zvolit požadovaný typ.

Poslední součást panelu se zviditelňuje pouze při zakončení štychu nebo celé hry. Toto tlačítko, které je ve skutečnosti obrázkem na Canvasu, po stisknutí zahájí dle situace další štych nebo partii.

Poslední součástí GUI aplikace je menu na liště formuláře. Pomocí položek menu je možné ukončit program, zahájit novou hru nebo nastavit některé parametry programu.

Nastavení probíhá v jednoduchém dialogovém okně. Uživatel určuje nové hodnoty výběrem z Choice, zaškrtáváním checkboxu nebo změnou obsahu textových polí.

Veškeré vykreslování obrázků probíhá v metodě paint konkrétních pláten. Informace o tom, odkud se mají soubory jednotlivých obrázků načítat, je uložena v třídě ImageLocation, která se nastavuje hned při spouštění aplikace.

B.4 Řízení běhu programu

B.4.1 Spuštění programu a inicializace nové hry

Okamžitě po spuštění aplikace dochází k výstavbě jejího GUI a ke konstrukci všech základních datových struktur, jako je vytvoření jednotlivých hráčů, balíčku karet a nastavení hry.

Po skončení této „tvůrčí fáze“ se volá metoda „gameInit“, která implementuje veškeré akce související s inicializací nové partie. Tato metoda zprostředkuje generování a nastavení karet na rukou hráčů a připraví formulář pro začátek nové partie.

Dále se činnost programu větví podle toho, zda je aktuálním volícím hráčem uživatel nebo jeden z CPU protihráčů.

V případě volení hráče se nastaví proměnné monitorující aktuální vývoj hry a do střední části formuláře se vypíšou pokyny, vedoucí uživatele průběhem partie. Uživatel pomocí voleb z karet na ruce a na konec i z výběrového seznamu postupně nastavuje parametry dané partie, jako jsou trumfy, talon a typ hry.

Pokud je na řadě některý z CPU protihráčů, volá se metoda „voleni“, která zhodnotí karty daného hráče a trumfy zvolí za něj (heuristickou metodou sčítající hodnotu karet a hlášky postupně od všech barev). Volba typu hry a talonu je o poznání méně jednoduchá než volba trumfů. Proto je pro ni vyhrazena samostatná metoda, funkce „voleniTypu“. Tato funkce postupně prochází jednotlivé druhy mariášových podher podle jejich ceny od nejvyšší, začíná tedy od durchu. Pro každou z her zkouší podle předem daných podmínek, zda se na ní karty hráče dají

použít. Pokud ne, zkusí se některá z nevhodných karet zahodit do talonu a celá procedura se opakuje, dokud není podmínkám vyhověno nebo dokud se talon nenaplní. Takto se postupně procházejí všechny hry, dokud není podmínkám první z nich vyhověno. Tato hra bude CPU protihráčem zvolena a funkce voleniTypu zároveň zajistí i volbu talonu.

Po zvolení všech parametrů hry mohou nevolící hráči partii ukrást. Hráč za sebe samozřejmě rozhoduje sám. CPU protihráčům opět poslouží výše zmíněná funkce voleniTypu, které se tentokrát pomocí parametru předá i hodnota nejnižší „zvolitelné“ hry.

B.4.2 Zpracování událostí generovaných uživatelem

Veškeré uživatelem generované události zachytává pomocí metody update přímo hlavní třída CardsMain. Tato třída totiž implementuje rozhraní Observer. Představuje tedy pozorovatele, který zachytává události od pozorovaných objektů. Těmito pozorovanými objekty jsou instance třídy NotifyEventClass, která je oděděna od třídy Observable a slouží právě jen k informování programu o akcích uživatele.

Parametry metody update je informace o zdroji události a řetězec, který událost dále specifikuje. Pomocí těchto dvou údajů může tato metoda rozlišit veškeré uživatelem generované události, od stisknutí tlačítka nového tahu až po výběr konkrétní karty z ruky hráče, a také na tyto události reagovat.

Hlavní vlákno aplikace však výpočetní část programu neprovádí přímo. Místo toho je v metodě update vytvořena instance třídy SwingWorker, která zajistí výpočet v samostatném vlákně. Tímto způsobem je v aplikaci oddělena pracovní část programu od grafického uživatelského rozhraní.

B.4.3 Průběh partie

Mariášová partie je rozdělena na jednotlivé štychy, tedy kola po třech zahraných kartách. V aplikaci jsou tyto štychy odděleny stisknutím tlačítka „new turn“. Je to řešeno tímto způsobem kvůli tomu, aby uživatel aplikace nebyl nucen rychle memorovat jednotlivé zahrané karty před tím, než mu je program skryje zahájením nového tahu. Po ukončení každého štychu se volá metoda „turnInit“, která připraví formulář aplikace a nastavení hry na další kolo. Tato funkce také kontroluje případné ukončení hry, ať už je to v případě odehrání všech karet nebo v případě prohry volícího hráče na betl/durch.

Pořadí hráčů v daném tahu určuje proměnná „playersOrder“, která udává posloupnost indexů těch hráčů, kteří v daném tahu ještě nehráli, a to v tom pořadí, v jakém mají jít za sebou.

Program je v pasivním vyčkávacím stavu v případech, kdy se čeká na tah hráče nebo na zahájení nového kola. Po vyvolání a provedení některé z těchto událostí se vyhodnotí, zda není na tahu některý z CPU protihráčů. Pokud ano, program pokračuje voláním metody „play“, která za pomoci algoritmu MAPP zjistí ideální tah daného hráče a tento tah provede. Po doběhnutí metody play se opět kontroluje nově aktualizovaná proměnná playersOrder. Podle ní program buďto znovu zavolá metodu play pro dalšího CPU protihráče nebo přejde do pasivního režimu čekání na reakci uživatele.

Průběh algoritmu umělé inteligence

Funkce MAPP je přesně popsaná (včetně zápisu v pseudokódu) v kapitole 4.1.3. Na tomto místě pouze zmíníme jednu drobnou, avšak z hlediska rychlosti algoritmu významnou, změnu. Funkce MAPP totiž při svém prvním zavolání (ještě z metody play, kdy je hodnota „depth“ rovna mínus jedné) po generování tahů kontroluje, zda náhodou na zásobník nepřibyl jen jeden jediný tah. V takovém případě už by byl další rekurzivní běh funkce zbytečný. Hráč totiž může zahrát jen jednu kartu, a proto tuto kartu funkce okamžitě vrátí coby ideální tah.

V této části se budeme blíže věnovat přesnému fungování generátoru tahů.

Generátor tahů sestává ze dvou hlavních metod. Obě mají za parametr instanci třídy StackClass, která reprezentuje aktuální stav stromu hry, a do které také pomocí metody push přidávají nové tahy. Před voláním generátoru se na instanci zásobníku (třída StackClass) volá metoda markLevelBorder, která oddělí tahy z vyšší hladiny stromu od tahů nově vznikajících.

Funkce generateMovesBasic je volána přímo z funkce MAPP a slouží ke generování tahů aktuálního hráče, tedy toho hráče, který algoritmus MAPP „zavolal“ a k jehož kartám máme přístup. Tato funkce jednoduše prochází jednotlivé karty hráče a ukládá na zásobník ty z nich, jejichž zahrání je v dané chvíli legální. S pravděpodobnostmi se pracuje jen v tu chvíli, kdy je aktuálně nejvyšší karta štychu anonymní a tudíž není jisté, jestli ji hráč kartou stejné barvy přebije nebo ne. Tyto situace se řeší tak, že se na zásobník umístí pro danou kartu dva tahy. Jeden, kdy se aktuální vítěz nemění, a druhý, ve kterém naopak k přebití došlo. Pravděpodobnosti těchto tahů jsou odvislé od hodnoty hrané karty a jejich součet je vždy roven jedné.

Pokud na řadě není aktuální hráč, volá se z funkce generateMovesBasic druhá funkce, GenerateMoves_Odds. GenerateMoves_Odds slouží ke generování „anonymních tahů“, tedy tahů, u nichž je možnost jejich zahrání pouze hypotetická. Fungování této metody je podrobně popsáno v kapitole 4.1.2.

Příloha C

Obsah CD-ROM

K této práci je přiloženo CD-ROM s její elektronickou verzí, zdrojovým kódem mariášového trenažéru (který je součástí složky projektu pro NetBeans IDE), distribucí JRE (Java Runtime Enviroment) pro operační systémy Windows a Linux a dvěma spustitelnými soubory, které umožňují přímé užívání této aplikace.

Jeden ze spustitelných souborů je instalační balíček pro MS Windows, obsahující distribuci JRE, která je pro spouštění programů v jazyce Java nezbytná. Druhý soubor je ve formátu JAR (Java Archiver) a ve spolupráci s nainstalovaným JRE na počítači uživatele umožňuje přímé spuštění aplikace bez nutnosti instalace.

Tabulka C. 1: Obsah CD-ROM

Soubor či adresář	**Obsah**
/bak_prace/	Adresář s touto prací ve formátech PDF a DOC
/program/NetBeans_projekt	Obsahem tohoto adresáře je mariášový trenažér coby projekt pro NetBeans IDE. Součástí projektu je mimo jiné zdrojový kód aplikace a vygenerovaná Javadoc dokumentace.
/program/exec/	Adresář se spustitelnými soubory mariášového trenažéru
/program/exec/Cards_windows_1_0_1.exe	Instalační balíček s aplikací pro MS Windows
/program/exec/Cards.jar	Aplikace ve formátu Java ARchiver
/src/	Adresář obsahující instalační balíčky s distribucí JRE pro Linux a Windows