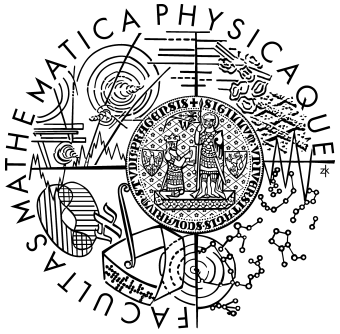
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Caithaml

Trénovací program na mariáš

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Petr Pudlák

Studijní program: Informatika, programování

2007

Rád bych na tomto místě poděkoval RNDr. Petru Pudlákovi za trpělivé vedení mé bakalářské práce.
Také bych rád poděkoval Janu Fialovi, z jehož mariáše byly převzaty obrázky karet.

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

V Praze dne Tomáš Caithaml

Obsah

Název práce: Trénovací program na mariáš
Autor: Tomáš Caithaml
Katedra: Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: RNDr. Petr Pudlák
e-mail vedoucího: Petr.Pudlak@mff.cuni.cz

Abstrakt: Předložená práce se zabývá návrhem programu na sehrávání trénovacích partií mariáše (tradiční české karetní hry) proti počítači. Uvažují se dvě varianty této hry pro tři hráče – mariáš volený a licitovaný. Program umožňuje sehrávat partie jak v turnajovém módu, který simuluje skutečnou hru, tak i v trénovacím módu, kdy hráč může nahlížet hráčům do karet, vracet tahy, přehrávat různé varianty, nechat si poradit tah a tak analyzovat herní situaci. Program lze rozšířit o další umělé hráče a lze mezi nimi pořádat turnaje. Práce dále představuje implementace umělého hráče, které jsou založeny na variantách alfa-beta prořezávání s několika vylepšeními.

Klíčová slova: hraní her, mariáš, alfa-beta prořezávání

Title: Training program for marias
Author: Tomáš Caithaml
Department: Dpt. of Theoretical Computer Science and Mathematical Logic
Supervisor: RNDr. Petr Pudlák
Supervisor's email: Petr.Pudlak@mff.cuni.cz

Abstract: In the present thesis we deal with designing a program to play training games of mariáš (traditional czech card game) against computer. Two three-player versions of this game are considered – volený mariáš and licitovaný mariáš. The program enables to play games in tournament mode simulating the real game as well as in training mode where player can look into opponent's hand, undo moves, replay different variations, get advice from computer and so analyse the game situation. It is possible to extend the program by artificial players and to set up tournament between them. The thesis introduces several implementations of artificial player based on enchanced alfa-beta prunning.

Keywords: game playing, marias, alfa-beta prunning

Kapitola 1

Úvod

1.1 Motivace

Mariáš je všeobecně známá a oblíbená karetní hra (tedy alespoň v Čechách), která má mnoho různých variant (čtyřka, licitovaný mariáš, volený mariáš, mariáš lízaný ...).

Existuje několik počítačových programů, které umožňují nějakou variantu hrát. Legendární a v jistém smyslu asi nepřekonaná je dvojice programů Flek! (volený mariáš) a RE! (licitovaný mariáš) od firmy Pivoňka Software, která pochází až z roku 1993. Tyto programy sdílí stejný vzor. Uživatel hraje proti oponentům, které řídí počítač. Před sebou vidí ruku s vlastními kartami a stůl, kam se odkládají štychy. Hráči se střídají v tazích, dokud není hra skončena. Pak se zobrazí výsledek a rozdá se na další hru.

Cílem bakalářské práce je navrhnout program, který by umožňoval víc než jen pouhou simulaci hry. Podstata tréninku spočívá v tom, že uživatel má kontrolu nad hrou. Může nahlížet hráčům do karet, nechat si poradit od počítače optimální tah, vracet zahrané tahy, hrát místo protihráčů a podobně. Uživatel si tak může vyzkoušet různé možnosti, jak zahrát v dané situaci.

Půjde o otevřený systém, do kterého lze přidávat umělé hráče. Ti budou hrát jak proti člověku tak i proti sobě. Výsledky automatických turnajů se budou zaznamenávat do souboru, aby mohly být později použity (například k analyzování různých strategií a jejich vylepšování).

Součástí základní instalace bude i implementace umělého hráče. Zde jsem se zaměřil hlavně na výběr správného tahu při hře. Vycházel jsem z prověřených a efektivních technik, které se používají v šachových programech.

Fakt, že stavový prostor mariáše je podstatně menší než u šachů, zde dává naději na dosažení velmi dobrých výsledků.

1.2 Přehled kapitol

Druhá kapitola pojednává o samotném mariáši. Jsou zde vyložena hlavně pravidla, tak aby se neznalý čtenář mohl zorientovat v dalším textu a aby nedocházelo k nejasnostem neboť pravidla mariáše nejsou všeobecně ustálená.

Třetí kapitola popisuje vzhled, chování a funkce programu. Jde o výtah z uživatelské dokumentace.

Čtvrtá kapitola je věnovaná návrhu programu, hlavním datovým strukturám a problémům, které byly při návrhu řešeny.

Pátá kapitola je celá věnovaná implementacím umělých hráčů, které jsou s aplikací standardně dodávány.

Šestá poslední kapitola obsahuje shrnutí a závěrečné zhodnocení. Diskutují se dosažené výsledky, možná vylepšení do budoucna a srovnání s ostatními programy s podobnou tématikou.

Kapitola 2

O mariáši

2.1 O mariáši

Mariáš je oblíbená karetní hra. Jde o hru specificky českou, neodmyslitelně spjatou s prostředím českých hospůdek a putyk. Mimo střední Evropu ji ke své škodě ale zná jen málokdo. Nejde jen o „veskrze hospodskou hru“, jak se o ní někdy hanlivě mluví. Těžko se dá říct, že by šlo o hru vyloženě hazardní, protože částky, o které se hraje, jsou spíše symbolické. Daleko větší uspokojení může hráči přinést vtipné polapení soupeřovi desítky nebo vzorně sehraná obrana betla. Mariáš má své osobité kouzlo, které se projevuje i v jazyce a specifickém humoru. Nápaditost při vymýšlení různých hlášek a průpovídek je opravdu velká. Stačí vzpomenout Haškových „Sedm kulí, jako v Sarajevu!!!“.

Krásná charakteristika mariáše pochází z pera Bohumila Hrabala [3]:

> „Mariáš se jmenuje proto mariage, protože je to neustálá svatba a manželství smutných králů a krásných královen, jejich sňatek má cenu dvaceti bodů, čtyřicet bodů má cenu jejich vztah, když je předem ohlášen jako trumfový. Zdánlivě nejvyšší karty jsou esa, desítky, ovšem pravý půvab hry je v tom, že obyčejná sedma, osma, devítka, obyčejný bezvýznamný kluk mají tu možnost, že donutí šintnutím desítky, aby se přiznaly, a spoluhráč tu cennou desítku zabije esem, někdy i opuštěným maličkým trumfem. Jak je mariáš královen a králů demokratický!“

Mariáš není jen jedna hra. Existuje celá řada variant. Pravidla navíc nejsou pevně stanovena a často se na různých místech liší ve spoustě drob-

ných detailů. Bez ohledu na variantu se však mariáš vždy hraje s balíčkem o 32 listech (tzv. německé karty). V Čechách se většinou hraje kartami jednohlavými, na Moravě jsou v oblibě karty dvouhlavé.

Základní varianty mariáše jsou:

- lízaný
- volený
- licitovaný
- čtyřka

Lízaný mariáš je varianta pro dva hráče a mariášníky je na něj pohlíženo s despektem. Lze citovat například Poláčka[5]: „že tento je dobrý pro staré manžele v důchodu nebo pro odsouzence, jenž tráví poslední noc před popravou hrou v mariáš se svým dozorcem.“

Volený mariáš a mariáš licitovaný jsou si v mnohém podobné. Oba vyžadují tři hráče. Volený mariáš lze s trochou nadsázky považovat za zjednodušenou variantu licitovaného mariáše vhodnou pro začátečníky. Licitovaný mariáš pak bývá označován jako „královská hra“ mezi mariáši.

Čtyřka, jak už název napovídá, se hraje ve čtyřech lidech. I tato varianta má mnoho zastánců a je velmi oblíbená.

Mě osobně si však nezískala a i proto tento program podporuje mariáš licitovaný a volený.

2.2 Obecná pravidla

Volený i licitovaný mariáš se hraje ve třech hráčích, kdy proti sobě hrají dvě strany – osamocený hráč (aktér) proti dvojici spoluhráčů. Jeden hráč je **forhont** (lidově též fořt), po jeho levici sedí prostřední hráč – **předák** a po jeho pravici („na vocase“) **zadák**. Pro jednoduchost si můžeme hráče očíslovat. Forhont je hráč číslo 1, předák číslo 2 a zadák číslo 3. Možná rozestavení jsou tedy tato:

```
   1          3          2
 3   2      2   1      1   3
```

Je vidět, že pořadí hráčů je dáno **po směru hodinových ručiček**. To je důležitá mariášnická zásada. Všechny další činnosti ať už licitace nebo zahrávání štychů se také řídí tímto uspořádáním. Forhont má vždy oproti ostatním hráčům výsadní postavení. Nejde však o výhodu trvalou. Po sehrání jedné hry se hráči ve svých rolích střídají – jak jinak – ve směru hodinových ručiček. Z předáka se stane forhont, ze zadáka se stane předák a z forhonta zadák (viz diagram výše).

Karty jsou rozděleny do čtyř barev: srdce (červený), listy (zelený), kule a žaludy. Každá barva obsahuje osm karet: sedma, osma, devítka, desítka, spodek, svršek (filek), král, eso. Karty se občas řadí do kategorii:

- Desítky a esa se souhrně nazývají *ostré*;
- spodek, svršek, král *figury*;
- sedma, osma, devítka *plívy*.

Karet je 32, tedy počet, který nelze rovnoměrně rozložit mezi tři hráče. Dvě karty jsou proto vždy mimo hru – v talonu.

2.3 Průběh partie

Před započetím vlastní hry probíhá jakési „schvalování“. V podstatě jde o to, aby se hráči dohodli na tom, kdo hraje proti komu a co hraje. Právě v této části se od sebe nejvíce liší licitovaný a volený mariáš.

Po této fázi ještě mohou protihráči přijaté závazky oflekovat. Pokud je závazků víc, flekuje se každý zvlášť. Je-li závazek oflekován protihráčem, může aktér flekovat zpátky, pokud tak učiní, může protihráč opět flekovat a tak dále. Každý flek zdvojnásobuje sazbu za závazek.

Některé varianty pravidel obsahují takzvané povinné fleky, kdy hráč musí za určitých okolností dát flek, i když to pro něj není výhodné, a také se při některé kombinaci oflekování hra ani nehraje pouze se složí karty. Trénovací program na mariáš tyto varianty neimplementuje.

Bez ohledu na výsledek schvalování se pak samotná hra řídí několika obecnými pravidly. Hra se skládá ze štychů (zdvihů). Hráč, který je na řadě, vynese kartu, ostatní v pořadí daném směrem hodinových ručiček přihodí další karty. Jeden hráč štych získá, sebere jej, přihodí si ho k sobě na hromádku a otočí rubem navrch. Potom výnosem začíná další štych a tak to

pokračuje, dokud se neodehrají všechny karty (v některých případech může hra skončit předčasně).

Při odehrávání štychů se musí dodržet dvě *relativně* jednoduchá pravidla:

1. **cti barvu**
2. **přebíjej**

Je-li vynesena karta, musí ji hráč přebít vyšší kartou stejné barvy. Pokud takovou kartu nemá, musí aspoň zahrát kartu této barvy (ctí barvu). Nemá-li kartu této barvy, pak musí zahrát trumf, nemá-li trumf, může zahrát cokoliv.

Důležité je toto:

- Pravidlo **cti barvu** se váže na první kartu ve štychu.
- Pravidlo **přebíjej** se váže na celý štych, tj. hráč je povinen se snažit pro sebe získat celý štych.
- První pravidlo má přednost před druhým.

Uvedu pár příkladů, aby by to bylo trochu jasnější.

- 1. hráč hraje kulového krále
- 2. hráč má kulovou sedmičku a kulové eso a nějaké trumfy.

Druhý hráč musí zahrát eso (pravidlo přebíjej). Kdyby neměl eso, musel by zahrát sedmičku, i když tak štych nezíská, přestože pomocí trumfů by mohl. Kdyby neměl ani sedmičku, musel by samozřejmě zahrát libovolný trumf.

Daleko důležitější je následující příklad, začátečníci v takovém případě často chybují:

- 1. hráč hraje kulového krále
- 2. hráč zabíjí trumfem
- 3. hráč má kulové eso a sedmičku a i nějaké vyšší trumfy. Co má hrát?

První byla hrána kule, hráč kule má, musí tedy bezpodmínečně ctít barvu a zahrát kulovou kartu. Eso přebíjí krále, sedmička nikoli. Král však byl již přebit trumfem, proto není naděje na zisk štychu a třetí hráč si může vybrat, kterou kuli zahraje.

2.4 Betl s durchem

Dvě hry – betl a durch – jsou velmi odlišné od ostatních her. Jako první vždy vynáší aktér (hráč, který hraje sám). Pořadí karet v jedné barvě dáno takto:

sedm, osm, devět, **deset**, spodek, svršek, král, eso.

Neexistují trumfy, nepočítají se body ani hlášky. Jde jen o to, kdo štych uhraje.

Betl také zvaný *žebrák* nebo *malej* je závazek, že se hráči nepodaří uhrát ani jeden štych. Pokud aktér získá nějaký štych, hra je u konce a platí protihráčům příslušnou částku.

Durch zvaný *velkej* je pravý opak betla. Hráč se zavazuje, že uhraje štychy všechny. Prohrává, pokud se protihráčům podaří uhrát byť jediný štych.

2.5 Ostatní hry

V ostatních hrách (hry kromě betla a durcha) jako první vynáší forhont. Pořadí karet je:

sedm, osm, devět, spodek, svršek, král, **deset** a eso

Desítka tedy přebije krále. Jedna z barev je trumfová a karty této barvy přebijí libolnou kartu jiné barvy. Do talonu se nesmí odhazovat desítky a esa.

Úkolem hráče v těchto hrách je uhrát více bodů než soupeři případně splnit nějaký závazek navíc. Každá desítka a eso je za 10 bodů (ty si připíše strana, která získala štych s takovou kartou). Poslední štych ve hře je oceněn také 10 body.

Pokud má jeden hráč v ruce svrška i krále jedné barvy, má takzvanou *hlášku*. Chce-li nést jednu z těchto karet, vynese svrška a oznámí „dvacet“ případně „čtyřicet“, pokud jde o hlášku v barvě trumfů. Svrška nepřikládá do štychu, ale přihodí si ho k sobě na hromádku a nechá ho otočeného lícem navrch. Na konci hry si za každou obyčejnou hlášku připočte 20 bodů bez ohledu na to, kdo vyhrál štych, ve kterém byla hlášena (za trumfovou si připočte 40 bodů).

Ve hře je 90 až 190 bodů, vždy však lichý počet (díky poslednímu štychu). Nemůže tedy dojít k remíze.

Jednotlivé hry se liší podle toho, jaké další závazky na sebe hráč bere. Hry dvě sedmy a dvě sedmy se stovkou se hrají jen v licitovaném mariáši.

Hra – Obě strany hrají o to, kdo bude mít bodů.

Sedma – Hráč se zavazuje uhrát poslední štych svojí trumfovou sedmou. Hraje se vždy společně se hrou, každá část se flekuje i platí zvlášť.

Stovka – Hráč se zavazuje uhrát alespoň sto bodů. Ale pozor! K dosažení této hranice se mu započítává jen jedna hláška. Má-li trumfovou, stačí mu uhrát 60 bodů (může ztratit 30 bodů), má-li netrumfovou musí uhrát 80 bodů (může ztratit jen 10 bodů). Bez hlášky nelze tento závazek uhrát.

Stosedm – Hráč se zavazuje uhrát jak stovku, tak i sedmičku. Každá část se flekuje i platí zvlášť.

Dvě sedmičky je hra, kdy hráč, který hraje sám, musí nejenom získat poslední štych trumfovou sedmičkou (jako u sedmy), ale navíc získá i předposlední štych a to sedmičkou pomocné barvy, kterou dopředu ohlásí (v praxi se hlásí například: "zelená sedma, žaludová jí pomůže").

Zvláštností této hry je, že narozdíl od sedmy a stovky se zde nepočítají body, hlášky nehrají žádnou roli. Je však důležité neplést si tuto hru s betlem nebo durchem – ostatní pravidla zůstávají v platnosti. Desítka je stále mezi esem a králem. **A ostré nelze odhodit do talonu**! Což je obzvláště nepříjemné, pokud vám přijde osamocená desítka v talonu.

Dvě sedmy a sto je nejvyšší možná hra. Hráč se zavazuje uhrát jak dvě sedmy a tak i stovku. Z pochopitelných důvodů v této hře odpadají zvláštnosti dvou sedem, tj. body se opět počítají. Tuto hru lze hrát jen s extrémně silným listem.

2.6 Konec hry, placení

Po skončení hry se vyhodnotí výsledek. Základní cena za každý závazek je uvedena v tabulce 2.1 (tabulka obsahuje i hry pro licitovaný mariáš, ve voleném mariáši platí tabulka po durcha). S každým flekem se cena zdvojnásobuje, stejně jako v případě, že srdce byly trumfy. Pokud se hrála stovka, určuje se cena podle počtu získaných desítek. Uhrál-li hráč čistou stovku, dostane základní sazbu. Za každou desítku navíc (zde se už počítají všechny

Stupeň	Název	Cena
	Hra	0,10 Kč
1	Sedm	0,20 Kč
2	Sedm lepších	0,40 Kč
3	Sto	0,40 Kč
4	Sto lepších	0,80 Kč
5	Sto sedm	0,60 Kč
6	Sto sedm lepších	1,20 Kč
7	Betl	0,50 Kč
8	Durch	1,00 Kč
9	Dvě sedmy	3,00 Kč
10	Dvě sedmy lepší	5,00 Kč
11	Sto a dvě sedmy	3,40 Kč
12	Sto a dvě sedmy lepší	5,40 Kč

Tabulka 2.1: Hry v licitovaném mariáši

hlášky) se sazba zdvojnásobuje. Naopak pokud prohrál, každá desítka, která chyběla do stovky, zdvojnásobuje prohru.

Výsledkem je cena, kterou platí hráči, kteří hráli spolu. Osamocený hráč tedy vždy dostane nebo zaplatí dvojnásobek.

Uhrála-li nějaká strana závazek, aniž by ho ohlásila, jedná se o takzvaný tichý závazek, za který je pouze poloviční zisk.

Po finančním vyrovnání se cyklicky posunou role všech hráčů. Dřívější forhont dá sejmout balíček hráči po své pravici a jako nynější zadák opět rozdá na další hru.

2.7 Volený mariáš

Hra se zahajuje rozdáním karet. Pokud jde o první hru, balíček se důkladně promíchá. Později se však už **nikdy nemíchá**.

Zadák rozdá forhontovi sedm karet, poté zadákovi a sobě po pěti kartách. Nakonec rozdá dokola každému po pěti kartách.

Forhont zvolí z prvních sedmi karet, aniž by se podíval na hodnotu zbývajících pěti, trumfovou barvu. Jednu z trumfových karet položí lícem dolů před sebe na stůl. Pokud se nedokáže rozhodnout, může „zavolit z lidu“, tj. vybrat naslepo jednu ze zbývajících pěti karet.

Pak si prohlédne zbývající karty a rozhodne se, zda chce hrát betla nebo durcha (špatná barva), či zda se spokojí s obyčejnou hrou případně vylepšenou o nějaký závazek (dobrá barva).

Pokud se rozhodne pro špatnou barvu, může vzít ohlášenou kartu zpět. Odhodí dvě libovolné karty do talonu a ohlásí buďto betla nebo durcha.

Pokud se rozhodne pro dobrou barvu, vybere ze všech karet dvě, které se mu nehodí, a odloží je rubem navrch stranou. Nesmí však odložit ostré (desítky a esa). Poté se zeptá protihráčů, jaká je barva. Protihráči se postupně vyjádří, zda dobrá nebo špatná. Hráč, který ohlásí špatnou barvu, si vezme talon, přidá jej do svého listu a poté odloží další dvojici karet (případně stejnou) do talonu. Nakonec ohlásí, zda chce hrát betla nebo durcha. Pokud by oba hráči chtěli dát špatnou barvu, má přednost ten, který byl první na řadě.

Nechce-li nikdo hrát ve špatné barvě, otočí forhont trumfovou kartu a ohlásí svůj závazek. Základem je hra v barvě trumfů. Má-li dostatečně silný list, může zahlásit sedmu, stovku nebo jejich kombinaci – stosedm.

2.8 Licitovaný mariáš

Hra se zahajuje rozdáním karet. Pokud jde o první hru, balíček se důkladně promíchá. Později se však už nikdy nemíchá.

Zadák rozdá každému pět karet v pořadí v forhont, předák, zadák. Pak odhodí dvě karty do talonu a opět každému rozdá pět karet.

Po rozdání následuje licitace, po které získala tato varianta jméno. Tabulka 2.1 shrnuje seznam všech 12 licitačních stupňů.

Licitace probíhá po dvojicích. Začíná předák, obrací se na forhonta a říká, co by chtěl hrát. Forhont buďto řekne „mám“, čímž si ponechá možnost volit hru a zároveň se zavazuje, že bude hrát alespoň takovou hru, na kterou kývl, nebo přepustí hru předákovi. Předák se forhonta může ptát opakovaně, ale svou nabídku musí zvyšovat. Po té co buď forhont nebo předák vzdá licitaci, může do ní vstoupit zadák a licituje s tím, kdo vyhrál předešlé kolo.

Hráč, který v licitaci zvítězí, si vybojoval právo získat talon a hrát sám proti ostatním. Na druhou stranu má povinnost vyhlásit buďto vylicitovanou hru nebo libovolnou vyšší.

Poznámka: Forhont je zřejmě ve výhodě, protože mu stačí jen kývat na nabídky ostatních hráčů. Pokud dva hráči chtějí hrát stejnou hru, získá ji forhont. Také není nezbytně nutné licitovat stupeň po stupni.

Pokud se nikdo nikoho na nic neptá, je na forhotovi, aby se rozhodl, zda se „podívá do talonu“. Pokud ne, složí se karty. Forhont zaplatí částku za netrumfovou hru (tuto hru nelze v licitovaném mariáši hrát samostatně) a začíná další kolo. Pokud si talon vezme, bere se to jako by některému z protihráčů kývnul na sedmičku.

V licitovaném mariáši lze hrát jednu z dvanácti her, každé z nich přísluší jeden licitační stupeň, viz 2.1. U hry není uveden žádný licitační stupeň, protože v licitovaném mariáši nelze tuto hru vylicitovat a hrát samostatně. Pokud je u hry uvedeno „lepší“ znamená to, že trumfy budou červené. Hry 1 až 8 jsou společné pro oba druhy mariáše. Hry 9-12 jsou specialitou licitovaného mariáše.

Kapitola 3

Ovládání programu

Trénovací program na mariáš je GUI aplikace. Po nastartování programu vás přivítá hlavní okno. Uprostřed je hrací plocha, která je prozatím prázdná, protože ještě nezačala hra. Na pravé straně se nachází informační panel, kde se zobrazují důležité informace o probíhající hře. Všechny funkce programu jsou dostupné z menu. Program byl navržen tak, aby jej šlo snadno ovládat jak pomocí myši tak i pomocí klávesnice.

3.1 Informační panel

Informační panel se nachází na pravém okraji hlavního okna. Jak název napovídá, obsahuje informace o právě hrané hře. Tedy, jaká hra se hraje, kdo onu hru vyhlásil, kdo je právě forhont, který hráč je právě na tahu a podobně. Také obsahuje ovládací tlačítka (podrobnosti viz sekce 3.7) a tlačítko na ukončení aktuální hry.

3.2 Hrací plocha

Hrací plocha zabírá většinu okna aplikace. Narozdíl od většiny programů na hraní mariáše jsou ruce hráčů uspořádány do třech souvislých pruhů u levého okraje obrazovky. Karty hráče, za kterého hraje člověk, jsou zobrazeny vždy vespod. Štychy se nikam neodhazují, ale zahrané karty se posunují na pravou stranu. Tento způsob zobrazování ač nestandardní je velmi přehledný a usnadňuje práci s partií při tréninkovém režimu.

Obrázek 3.1: Menu Partie

3.3 Jak začít hru

Novou partii lze začít pomocí menu Partie (obr. 3.1). Vybráním menu *Partie\Nová partie ...* zobrazíte průvodce, který vám umožní začít novou partii (obr. 3.2). V první obrazovce si musíte zvolit jeden ze tří druhů podporovaných partií. Každému typu je věnovaná zvláštní kapitola viz 3.6, 3.7, 3.9. Pro zrychlení práce s programem jsou v menu také položky, které rovnou vedou na daný typ partie. Jde o příkazy *Nový trénink*, *Nový turnaj*, *Nový automat*. Ty mají navíc mají přiřazeny klávesové zkratky, takže hru lze zahájit stisknutím správného tlačítka.

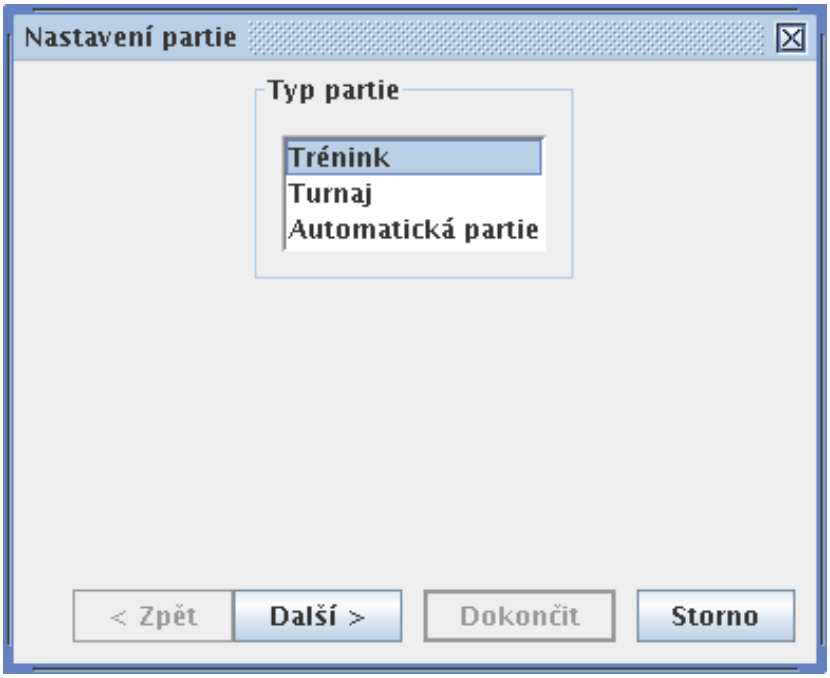
Ať už si vybererete jakýkoliv typ, následuje další obrazovka průvodce, kde je potřeba vyplnit základní údaje (obr. 3.3).

Pole *Forhont*, *Předák* a *Zadák* obsahují jména hráčů v nově vytvářené partii. V comboboxech napravo lze pro každého hráče určit, kdo za ně bude hrát. Program je navržen tak, aby bylo možné přidávat různé umělé hráče (více viz 3.10). Kromě nich je v nabídce vždy i Lidský hráč, který znamená, že hráče bude ovládat uživatel. Combobox *Trenér* má obdobný význam jako u hráčů. Je zde vybrán algoritmus, který bude hráči napovídat při hře.

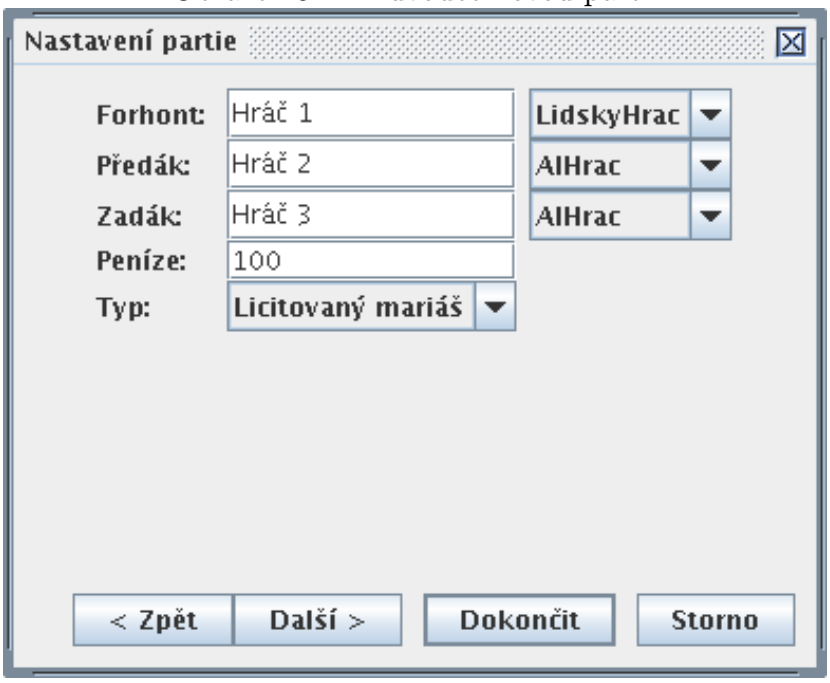
Pole *Peníze* umožňuje určit počáteční stav kont všech hráčů.

V poli *Typ* lze nastavit, zda si přejete hrát licitovaný nebo raději volený mariáš.

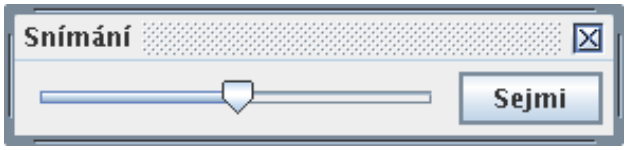
Stisknutím tlačítka *Dokončit* se vytvoří nová partie. Tlačítko *Další* umož-



Obrázek 3.2: Průvodce novou partií



Obrázek 3.3: Nastavení základních paremetrů nové partie



Obrázek 3.4: Snímání karet

ňuje další nastavení, které se liší podle zvoleného typu partie a je popsáno v jednotlivých sekcích.

Další možností jak zahájit partii, je nahrát ji ze souboru. To lze pomocí klávesy F6, nebo přes menu *Partie/Nahrát partii*.

3.4 Jak hrát

Abyste mohli začít hrát je nejprve nutné znát pravidla. Pokud je neumíte, jsou uvedena v části 2.2 na straně 9. Vřele doporučuji, aby do nich alespoň nahlédl i protřelý mariášník, protože mohou existovat rozdíly mezi pravidly zde uvedenými a těmi, na které je hráč zvyklý.

Mariášový trenér je naprogramován tak, že neumožní zahrát tahy proti pravidlům.

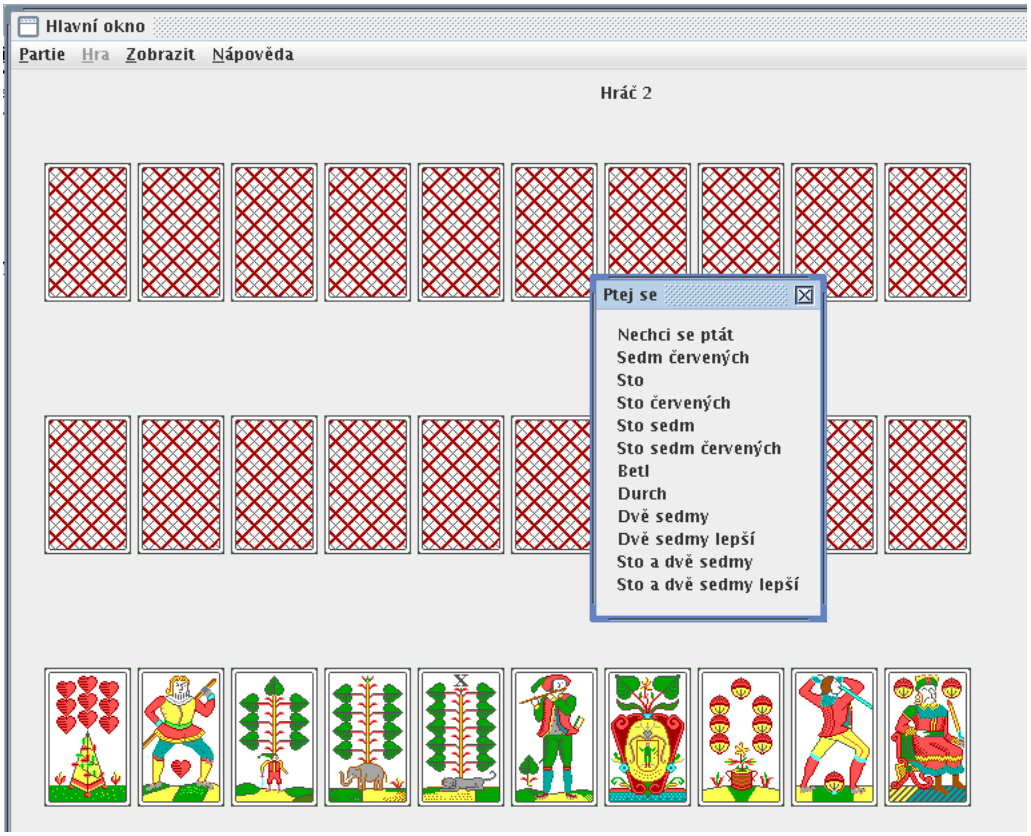
Hra začíná rozdáváním. Pokud je na Vás řada se snímáním, objeví se výzva k sejmutí dvou až třiceti karet (obr. 3.4). Rozdané karty se zobrazí do třech pruhů. Hráč ovládaný člověkem se vždy zobrazí úplně dole. Po rozdání okamžitě následuje proces schvalování, kterým je uživatel proveden pomocí série dialogů, ve kterých odpovídá na otázky svých spoluhráčů nebo jim klade své návrhy.

Konkrétní průběh se u licitovaného mariáše trochu liší od voleného, způsob ovládání je ale stejný. Na obr. 3.5 je vidět ukázku z licitovaného mariáše.

Celý proces se řídí pravidly popsanými v části 2.2.

Po skončení schvalování začíná samotná hra. Ovládání je velmi intuitivní. Ve chvíli, kdy je hráč na řadě, vybere kartu, kterou chce zahrát. To může udělat, jak pomocí kliknutí myši, tak i pomocí klávesnice. Právě aktivní karta je označena nahoře červeným trojúhelníkem. Označení lze posouvat kurzorovým tlačítky. Stisknutím tlačítek mezera nebo enter bude karta zahrána. Odehraná karta se přesune na pravý okraj, kde s ostatními kartami v jednom sloupci vytváří jeden štych.

Na konci hry se zobrazí vyúčtování, kde je shrnut výsledek právě dohrané hry.



Obrázek 3.5: Ukázka licitace

3.5 Ložené hry

Pojmem *ložená hra* se v mariáši označuje situace, kdy má hráč takový list, že protihráči neuhrají ani štych, pokud jim to nedovolí. V takovém případě je slušnost – v některých variantách pravidel dokonce i povinnost – hru prohlásit za loženou a složit karty, aby se zbytečně nemíchaly. Pokud by protihráči tvrzení o ložené hře nepřijali, řeší se to tak, že hráč, který chtěl hru ukončit, dohrává s otevřenými kartami.

Důležitým detailem je, že hra se považuje za loženou pouze pokud protihráči neuhrají jediný štych při **jakémkoliv** rozložení karet.

Mariášový trenér je vybaven schopností takové stavy rozpoznávat. Lze nastavit do kolikátého štychu se budou ložené hry kontrolovat. Pokud se hra dostane do situace, kdy je hra ložená a do konce zbývá více štychů než je uvedeno v nastavení, program sám dohraje zbytek hry.

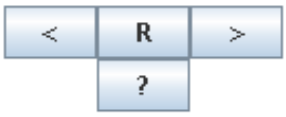
3.6 Turnaj

Tento mód slouží k hraní partií mariáše proti počítači. Oproti tréninku jsou zde zablokovány některé pokročilé funkce. Nelze vracet tahy ani jinak se hrou manipulovat. Jediné, co lze dělat, je hrát přesně podle pravidel.

3.7 Trénink

Tento režim nabízí nejvíce funkcí. Narozdíl od turnaje (viz 3.6), jsou všechny funkce zpřístupněny.

Nejdůležitější změnou oproti turnajovému módu je odblokování tlačítek na informačním panelu.



Prostřední tlačítko signalizuje, zda je hra zastavena. Pokud hra běží, je na něm napsáno písmeno P (pauznout) a jeho stisknutím lze hru zastavit. Po té se popisek přepíše na R (restart) a jeho stisknutím lze hru opětovně

rozběhnout. Šipka doleva znamená *Krok zpět* a vrátí hru o tah zpátky, zároveň pauzne hru. Šipka doprava znamená *Krok vpřed* a posune hru o tah dopředu, zároveň pauzne hru. Pokud uživatel zahrává tah, může si stisknutím tlačítka s otazníkem vyžádat *Nápovědu*. Algoritmus trenéra začne s hledáním nejlepšího tahu v dané situaci a až ho spočte zobrazí se výsledek uživateli. V případě, že probíhá výpočet, je místo otazníku zobrazen vykřičník. Jeho stisknutím si uživatel vyžádá urychlení výpočtu tahu.

Pomocí tlačítek *Krok zpět* a *Krok vpřed* lze tedy snadno procházet historií hry. Navíc se hra přeruší, aby nedocházelo k interferenci mezi hráči snažícími se zahrát své tahy a uživatelem, který tyto tahy vrací.

Při zastavené hře lze buďto procházet historií nebo zahrávat tahy místo hráčů. To se děje jednoduše prostým vybráním karty, která má být zahrána, stejně jako při běžné hře. Pokud je však karta zahrána, ruší se historie vrácených tahů a nelze se již vrátit o tahy dopředu (tlačítko *Krok vpřed* se znepřístupní).

Kromě tlačítek v informačním panelu, má uživatel při tréninku také zpřístupněna menu *Hra* a *Zobrazit*. V menu *Zobrazit* lze nastavit, které karty se mají odhalit. Možnosti jsou: odehrané štychy, talon, ruce hráčů, poslední štych. Menu *Hra* obsahuje příkazy, kterými lze manipulovat s hrou.

3.8 Sestavení partie

Tréninkové partie lze i sestavit na míru, tj. definovat herní situaci, která se má zahrát.

Stačí, když při vytváření tréninkové partie (viz obr. 3.3 na straně 19) místo tlačítka *Dokončit* stisknete tlačítko *Další*. Průvodce pak zobrazí další obrazovku, na které se vyplní detaily.

Na výběr jsou dvě možnosti *Začátek licitace* a *Hra*. První možnost znamená, že si přejete nastavit pouze jak byly rozdány karty – zbytek už proběhne normálně. Druhá možnost znamená, že si přejete nastavit rozložení karet v rukách hráčů (a v talonu), hru, která se bude hrát, a hráče, který bude hrát sám.

Tlačítka v dolní části nastavují ruce jednotlivých hráčů a také talon. Stiknutím tlačítka *Nastav* vyvoláte dialog), kde v dolní polovině je pole 4x8 karet – balíček, ze kterého můžete vybírat. Karta otočená rubem navrch značí, že už je v ruce jiného hráče. Kliknutím na takovou kartu ji lze z ruky dotyčného hráče odstranit a tak si ji zpřístupnit. V horní části dialogu je

Obrázek 3.6: Sestavování hry

pak aktuální výběr karet pro daného hráče. Kliknutím do takového výběru se karta opět vrátí do balíčku.

3.9 Automatická partie

Automatické partie jsou vhodné pro testování umělých hráčů (viz 3.10). Mohou se jich účastnit jen umělí hráči. Při sestavování takové partie se v průvodci určí podmínky ukončení turnaje a soubor, do kterého se má uložit výsledek.

3.10 Umělí hráči

Umělí hráči jsou hráči ovládaní počítačem. S aplikací se dodává základní sada hráčů, jejichž algoritmy jsou inspirovány šachovými programy. Mariášový trenér je však navržen tak, aby šlo přidávat další hráče, kteří mohou implementovat své vlastní strategie a používat jiné algoritmy.

Instalace umělého hráče je extrémně snadná. Stačí class soubor, který představuje zkompilovaný kód umělého hráče, nahrát do podadresáře `plugin` a přidat do konfiguračního souboru `plugins.xml` záznam o tomto hráči. Přesný popis, jak tento záznam vytvořit, by měl být součástí dokumentace umělého hráče.

Existenci dalších typů umělých hráčů poznáte při vytváření nových partií, kde vám v bude nabídnuta možnost od nich vytvářet hráče (viz 3.3, na str. 18).

Více o tom, jak takový soubor vyrobit a co musí splňovat, se můžete dozvědět z programové dokumentace programu.

Kapitola 4

Struktura programu

4.1 Volba platformy

Trénovací program na mariáš je jednouživatelská GUI aplikace. Jako vývojová platforma byl zvolen jazyk JAVA 5.0, pro GUI část byla použita knihovna Swing.

Mariášový trenér je relativně nenáročná aplikace, která nevyžaduje pro svůj běh skoro žádné služby, pouze GUI knihovnu. Obsahuje však jednu část, která spotřebovává hodně zdrojů a tou je algoritmus na výpočet optimálního tahu.

Vzhledem k této skutečnosti by možnou volbou implementačního jazyka bylo C/C++, které je kompilované a dá se předpokládat, že kód v něm napsaný by byl asi nejefektivnější. Proti hovoří fakt, že GUI část by vyžadovala knihovnu, se kterou by se program musel linkovat.

Nakonec jsem se rozhodl pro jazyk Java. Výsledný kód bude nejspíš o něco pomalejší, než kdyby byl napsán v C/C++. S výjimkou počítání tahů, to však vůbec nevadí. Algoritmus na počítání nových tahů má exponenciální časovou složitost, takže se domnívám, že určité zpoždění (dle mého odhadu v řádu procent) by nemělo výrazněji vadit.

Velkým plusem naopak je, že odpadnou problémy s instalací knihoven a překládáním na jiných platformách. Výsledný projekt lze nezměněný rovnou spouštět na libovolném OS, pro který existuje JVM, což je taky jediná závislost programu. Instalace se tak zjednoduší na pouhé nakopírování do instalačního adresáře.

Ze stejných důvodů byla vybrána knihovna Swing. Její nevýhodou je relativní náročnost na zdroje, určitá pracnost při vytváření jednotlivých formu-

lářů a také bývá kritizována (a to asi nejvíce) za nepěkný vzhled grafických prvků.

Po dokončení projektu mohu říct, že šlo o správnou volbu. Nezaznamenal jsem žádné komplikace způsobené režií danou použitím vybraného jazyka. Běžný počítač má dostatečný výkon na to, aby prohledávaní mohlo dosáhout dostatečné hloubky.

U knihovny Swing se potvrdily některé její špatné vlastnosti. Výsledné GUI není 100% nativní a někomu může přijít nepěkné, ale nemyslím si, že by šlo o nějakou závažnou chybu. Vytváření celého GUI však bylo relativně dost pracné. To je však přijatelné cena, vezmeme-li v úvahu, že jsme tím získali program, který kromě JVM nemá žádné další závislosti a běží na široké paletě operačních systémů.

Není mi například znám žádný program, který by umožňoval hrát mariáš na systémech jiných než ty od firmy Microsoft. Pominu-li možnost emulace a hraní přes web.

4.2 Přehled balíčků

Vzhledem k tomu, že celá aplikace má přes třináct tisíc řádků (včetně javadocové dokumentace, bez ní má aplikace přes deset a půl tisíce řádků), je pro větší přehlednost rozdělena do několika balíčků:

- `marias.core` Zde jsou uloženy třídy, které modelují samotný mariáš. Ústřední třídou je *Partie*.
- `marias.gui` Zde jsou soustředěny třídy, které mají na starost vizuální stránku programu. Jde vesměs o panely a dialogy – potomky tříd z knihovny Swing.
- `marias` Tento balíček obsahuje třídy, které logicky nespadají do žádného z ostatních balíčků. Jde hlavně o pomocné třídy, výjimky apod. Obsahuje důležité třídy *Aplikace*, *Display*, *Nastaveni*, *PluginLoader*.
- `marias.ai` Tento balíček není přímou součástí programu. Obsahuje jednu konkrétní implementaci sady algoritmů pro umělé hráče. Třídy pro tyto algoritmy se nahrávají až za běhu a uživatel je může nahradit jinými. Více o mechanismu nahrávání algoritmů viz 4.9, implementaci dodávanou s aplikací se zabývá následující kapitola.

4.3 Přehled základních tříd

Vstupním bodem do programu je třída *Aplikace*, která obsahuje pouze metodu *main*. Tato metoda se stará o inicializaci aplikace (nastavení GUI manažera, nahrání nastavení ze souborů a lokalizaci standardních dialogů) a vytvoření instance třídy *Display*, které předá řízení.

Display představuje hlavní okno a má na starosti interakci s uživatelem a řízení celé aplikace.

Nejdůležitější třídou je však asi *Partie*. Instance této třídy fungují jako manažeři hry a spravují všechny informace, které je o nich potřeba evidovat. Většina objektů v systému si drží referenci na partii, ve které se nacházejí, a přes tento odkaz komunikují se zbytkem systému. Mimo jiné si partie udržuje odkaz na tři hráče, kteří se jí účastní, informace o tom jaká fáze hry právě probíhá, pokud už skončila licitace, tak jaká hra se hraje, s jakými trumfy, jaké jsou fleky, jaký je dosavadní průběh hry apod.

4.4 Typy partií

Trénovací program na mariáš je relativně komplexní aplikace. Umožňuje hrát jak licitovaný, tak i volený mariáš, lze hrát v několika módech: turnajovém, tréninkovém a automatickém. Turnajový a tréninkový mód se od sebe liší jen nepatrně, u automatické partie je však potřeba některé části implementovat různě. Při vytváření je tedy třeba určit:

- typ partie (interaktivní nebo neinteraktivní)
- typ mariáše (volený nebo licitovaný)

Klasickým řešením takové situace je použití dědičnosti a definování potomků abstraktní třídy. To by ale znamenalo definování čtyř potomků – pro každou kombinaci jednoho, což je zbytečně mnoho pro nastavení dvou nezávislých parametrů.

Místo toho jsem definoval dvě rozhraní (*IPartie* a *IMarias*), která zapouzdřují části, které je potřeba implementovat různě. Třída *Partie* má pak privátní konstruktor, který přijímá instance těchto rozhraní. Činnosti, které závisí na typu partie, se pak jednoduše delegují na předaná rozhraní. Pokud chce někdo získat instanci třídy*Partie*, musí zavolat jednu z veřejných statických metod, které se postarají o vytvoření správného typu partie.

Rozhraní *IPartie* má dvě implementace *NormalniPartie* pro interaktivní partii, které se účastní uživatel, a *AutomatickaPartie* pro partii, kde proti sobě hrají pouze hráči řízení počítačem. První implementace úzce spolupracuje s třídou *Display*, aby byly správně zobrazovány informace o partii, druhá implementace pak hlavně kontroluje podmínky, kdy má partie skončit.

Aby byla *Partie* schopná hrát jak licitovaný tak volený mariáš, byl zvolen tento model: rozlišuje se „herní fáze“, kdy je už ohlášena hra a hráči sehrávají jednotlivé štychy, a „licitační fáze“, která začíná snímáním balíčku a končí tím, že některý hráč ohlásí hru a začne hrát proti ostatním. Licitační fáze je u licitovaného a voleného mariáše dost různá, proto byla tato činnost delegována na objekt typu *IMarias*.

I herní fáze se může mít více podob podle toho, jak dopadlo vyhlášení. Nejvíce je rozdíl patrný mezi hrami typu betl/durch a ostatními hrami, kdy se dokonce mění i pořadí karet. Není ovšem rozdíl mezi licitovaným a voleným mariášem. Rozdíly mezi jednotlivými hrami byly zapouzdřeny do rozhraní *IHra*.

4.5 Hráči

V mariáši spolu hrají tři hráči – forhont, předák a zadák. *Partie* obsahuje speciální kolekci *Hraci*, která obsahuje hráče zároveň s jejich pořadím a je schopná toto pořadí po skončení jedné hry pootočit.

Hráč (instance třídy *Hrac*) je autonomní agent, který svým chováním ovlivňuje hru. Pro každou situaci, kdy je třeba, aby hráč učinil nějaké rozhodnutí je v třídě *Hrac* definovaná abstraktní metoda. To jakým způsobem jsou tyto metody naimplementovány určuje hráčovo chování a toto chování zase určuje, jakým způsobem se bude hra odvíjet.

Třída obsahuje metody jak pro licitovaný, tak i pro volený mariáš. Dále obsahuje odkaz na nadřízenou partii. Díky tomu si může zjistit všechny informace, které potřebuje pro svá rozhodnutí. Je to jednodušší model než předávat informace jako argumenty. Platí zásada, že *Hrac* nemění stav partie (metody definované na třídě používají odkaz na *Partii* jen na čtení a nic nemodifikují) a že *Hrac* hraje podle pravidel (návratové hodnoty metod se neověřují).

Hrac je pouze abstraktní třída, která implementuje pouze pomocné metody. Má dva potomky: třídy *LidskyHrac* a *UmelyHrac*.

Třída *LidskyHrac* implementuje hráče, kterého ovládá uživatel. Je provázána s třídou *Display*, na kterou deleguje většinu svých úkolů.

Obdobně, třída *UmelyHrac* implementuje hráče, kterého ovládá počítač. Třída samotná je pouhá obálka. Své úkoly deleguje na instanci rozhraní *IMozek*, kterou dostane jako parametr konstruktoru.

IMozek je rozhraní, jehož metody odpovídají abstraktním metodám třídy *Hrac*. Samotná aplikace neobsahuje žádnou jeho implementaci. Ty se nahrají až za běhu (více viz 4.9).

4.6 Hry

V herní fázi závisí konkrétní pravidla na vyhlášené hře a nezávisí na typu mariáše (zde se volený mariáš od licitovaného liší jen tím, které hry lze hrát). Tyto rozdílnosti byly zapouzdřeny do rozhraní *IHra*. Instance her nelze libovolně vytvářet, lze je získat jen pomocí *HraFactory*. Objekty typu *IHra* odpovídají jednotlivým licitačním stupňům v licitovaném mariáši. V případě voleného mariáše se používá jen jejich podmnožina.

Jednotlivé hry jsou implementovány pomocí několika tříd provázaných dědičností.

4.7 Vlákna a komunikace mezi třídami Partie a Display

Trénovací program na mariáš je vícevláknová aplikace. Vzhledem k tomu, že byla použita knihovna Swing, probíhají veškeré GUI záležitosti (zpracování událostí, překreslování, apod.) v tzv. *event-dispatch* vlákně. Většina metod třídy *Display* je volána právě v rámci tohoto vlákna. Je proto vhodné, aby v něm byly vykonávány jen činnosti, které bezprostředně souvisí s GUI a které nezabírají mnoho času. Pokud by totiž zpracování nějaké události zabralo například několik vteřin, GUI by po tuto dobu neodpovídalo na uživatelovy příkazy a ani by se nepřekreslovalo – aplikace by "zamrzla".

Nejjednodušší způsob, jak navrhnout GUI aplikaci, je vytvořit třídu pro hlavní okno a v jejím rámci spouštět kód aplikace. Tento přístup je vhodný pro event-driven aplikace jako je například grafický editor, který nic nepočítá a jen čeká na reakce uživatele. Vykonávaný kód závisí pouze na tom, co dělá uživatel.

V případě mariášového trenéra je však situace odlišná. Hra "žije" svým vlastním životem. Hráči se střídají ve hře, hned jak zahrají kartu. Běh je

více určován pravidly hry, než vstupy od uživatele, který hru většinou ovlivňuje jen jako jeden ze tří rovnocenných hráčů. Navíc vypočtení nového tahu v případě umělých hráčů může trvat delší dobu a to by mělo negativní dopad na interaktivitu uživatelského rozhraní.

Jedním z možných řešení by bylo udržovat si informace o aktuálním stavu partie v pomocných proměnných, rozdělit logiku aplikace na dostatečně malé kousky a vykonávání kódu zorganizovat tak, že se podle pomocných proměnných určí jaký kód by měl být vykonán, po jeho vykonání se upraví pomocné proměnné, aby byl příště spuštěn kód, který by měl logicky následovat, a pak se vrátí řízení event-dispatch vláknu, aby se mohly obsloužit případné události a GUI zůstalo responsivní.

Aplikace by tak vlastně implementovala svépomocný multithreading, kde by simulovala práci dvou vláken – jedno pro GUI, jedno pro běh mariáše – v jednom vlákně pomocí stavového automatu.

Vzhledem k tomu, jak komplexní hrou mariáš je, by to vedlo k velmi složitému a nepřehlednému kódu. Navíc takovéto uspořádání neřeší problém dlouhé odezvy umělých hráčů. Tato volání by stejně musela být prováděna asynchronně, čímž by se složitost programu ještě zvětšila.

Proto jsem se rozhodl pro jiné řešení. GUI část bude mít na starosti instance třídy *Display*, která poběží na event-dispatch vlákně, a která ve chvíli, kdy si uživatel bude přát začít novou hru, vytvoří instanci třídy *Partie*, jež se bude starat o udržování informací o samotné hře. Třída *Partie* obsahuje metodu *start*, po jejímž zavolání se vytvoří a spustí nové vlákno, které bude vykonávat hlavní smyčku.

Aplikace se tak zpřehlední, protože dojde k oddělení bussiness logiky od prezentace – *Display* se nemusí zabývat pravidly mariáše, *Partie* nemusí nic vědět o GUI, které se tak přirozeně stane responzivní, protože "výpočty" budou probíhat v jiném vlákně.

Na druhou stranu toto řešení přináší i problémy. *Display* a *Partie* nejsou úplně nezávislé. *Display* musí být schopen sledovat stav *Partie*, aby ji mohl vykreslit a musí jí být schopen předat příkazy od uživatele. Protože oba objekty běží v jiných vláknech nástává nutnost synchronizace.

Zajistit, aby instance třídy *Display* nezůstávala pozadu za aktuálním stavem partie je vcelku jednoduché. Knihovna Swing obsahuje pomocnou metodu *SwingUtilities.invokeAndWait(Runnable)*, která zablokuje volající vlákno a vloží argument do fronty událostí. Kód, který je v metodě *run* (jediná metoda rozhraní *Runnable*), je později spuštěn v event-dispatch vlákně stejně jako kdyby to byla obsluha nějaké události a dokud tento kód není

vykonán, je volající vlákno zablokované.

Vždy, když nastane nějaká změna ve stavu partie (hráč zahraje kartu a podobně), zavolá *Partie* interní metodu *updateDisplay*, která pomocí *invokeAndWait* zavolá na nadřízené instanci třídy *Display* metodu *update*. Ta si jednoduše přečte z *Partie* relevantní informace a překreslí podle nich okno aplikace.

Předávání informací opačným směrem je podobné jen o něco složitější, protože nešlo použít předpřipravenou metodu. Pro předávání příkazů je použita třída *BlockingQueue<Udalost>* z balíčku *java.util.concurrent*. Jde o thread-safe implementaci fronty, která je navržena pro problémy typu producent-konzument. Třída *Udalost* reprezentuje předávaný příkaz, obsahuje string, který identifikuje konkrétní příkaz, a případné další argumenty.

Fronta má nastavenou nenulovou kapacitu a tudíž akce přidání události do fronty narozdíl od *invokeAndWait* neblokuje. Není k tomu ani důvod, všechny důležité informace jsou obsaženy přímo v objektu typu *Udalost* a není třeba číst informace z *Displaye*.

Partie nemůže zároveň provádět hlavní smyčku a zpracovávat příchozí zprávy. Proto každá příchozí zpráva nastaví příznak přerušení, který způsobí, že *Partie* přestane provádět hlavní smyčku a začne zpracovávat příchozí události.

Typ přerušení může nastaven na tyto možnosti:

- NONE partie není přerušena, probíhá hlavní smyčka
- ABORT partie je přerušena, při nejbližší příležitosti vyhodí hlavní smyčka výjimku *InterruptedException* (před tím se uvede do konzistentního stavu) a pak se předá řízení zpracování událostí
- RETURN partie není přerušena, jde o pouhý příznak, že by hráč, který je na tahu měl pokud možno ihned zahrát svůj tah.

O obsluhu fronty událostí se pak stará metoda *zpracujUdalost*. Mód, kdy se zpracovávají události, lze zrušit dvěma způsoby:

- zasláním události *Ukončit* – způsobí ukončení vlákna *Partie*
- zasláním události *Pokračovat* – způsobí, že *Partie* začne opět vykonávat hlavní smyčku a přestane zpracovávat události

4.8 Ukládání a nahrávání her

V rámci trénovací partie lze uložit právě rozehranou hru a později ji opět nahrát. Pro ukládání jsem zvolil vlastní formát XML. Java sice obsahuje funkcionalitu pro automatickou serializaci objektů, ale ta ukládá pouze do binárního formátu a ten je navíc velmi citlivý na změny – stačí nepatrně pozměnit definici třídy a uložená data se stanou nekompatibilními. Formát XML je proto podle mě lepší volbou. Serializace není sice automatická, je potřeba pro každý objekt implementovat, jak se má načíst z a uložit do XML, ale za to má programátor kontrolu nad tím jaká data uložit.

Díky rozhraní DOM ani není konverze příliš složitá. Objekty, které je potřeba ukládat mají definovanou metodu *toXML(Document,Element)*, která uloží do předaného elementu svůj obsah a třídy těchto objektů, pak mají implementovány statickou metodu *fromXML*, která z elementu zrekonstruuje původní objekt. Pro některé jednodušší objekty je serializace umístěna v třídě XMLConverter spolu s pár pomocnými statickými metodami, které používají objekty při konverzi.

Tento způsob má také další výhody. Soubor je v čitelném formátu, proto si ho případně může uživatel ručně napsat nebo pomocí nějakého nástroje vygenerovat či naopak zpracovat.

V neposlední řadě se XML serializace používá nejen pro ukládání partií, ale i při nahrávání umělých hráčů a při vypisování výsledků automatické partie.

4.9 Pluginy

Trénovací program na mariáš vychází z konceptu, že možností jak implementovat strategii hráče je mnoho a proto obsahuje mechanismus jak algoritmy pro umělé hráče zavádět z externích souborů. Samotné jádro aplikace pak žádný takový algoritmus neobsahuje, pouze je s ní dodávána základní sada, která stačí k zprovoznění hry, ale která se nijak neliší od jiných, které si může kdokoli implementovat sám.

Standardní implementace je inspirována šachovými programy a využívá prohledávání stavového prostoru s ořezáváním. Podrobněji je popsána v následující kapitole.

Pluginovací systém je navržen tak, aby byl jednoduchý a přitom měl dostatečnou flexibilitu a neomezoval případného implementátora.

```
<?xml version="1.0" encoding="windows-1250" ?>
<root>
<hraci>
<hrac class="marias.ai.Mozek" name="Standardní hráč">
 <!-- user defined content -->
 <snimac  class="marias.ai.NahodnySnimac" />
 <licitator class="marias.ai.StandardniLicitator" />
 <hrac class="marias.ai.SlozenyTrener" >
   <betl class="marias.ai.StandardniBetlTrener" />
   <durch class="marias.ai.StandardniDurchTrener" />
   <normal class="marias.ai.StandardniNormalniHraTrener"/>
  </hrac>
</hrac>
</hraci><treneri>
 <trener class="marias.ai.SlozenyTrener" name="Trenér">
  <betl class="marias.ai.StandardniBetlTrener" />
  <durch class="marias.ai.StandardniDurchTrener" />
  <normal class="marias.ai.StandardniNormalniHraTrener" />
 </trener>
</treneri>
</root>
```

Obrázek 4.1: Ukázka souboru plugins.xml

Pluginy nejsou nic jiného než obyčejné javovské class soubory umístěné v adresáři `plugin`. Aby bylo možné poznat, který soubor obsahuje implementaci nějakého algoritmu, je v kořenovém adresáři umístěn soubor `plugins.xml`, který obsahuje jednotlivé konfigurace.

Struktura konfiguračního souboru je jednoduchá. Kořenový element se jmenuje `root`, ten obsahuje elementy `hraci` a `treneri`. Element `hraci` obsahuje posloupnost elementů `hrac`, element `treneri` zase obsahuje posloupnost elementů `trener`. Tyto podelementy mají povinný atribut `class`, který používá *PluginLoader* pro nahrání třídy do aplikace a `name`, který určuje jméno dané jednotlivé konfigurace a musí být jednoznačný (stačí v rámci elementu `hraci` nebo `treneri`, podle toho, ve kterém je). Obsah elementu je nedefinován, každá implementaci si sem může uložit informace, jaké potřebuje.

O nahrávání tříd se stará *PluginLoader*, který při svém vzniku načte konfigurační soubor a extrahuje z něj jména hráčů a trenérů. Pro dané jméno také umí podle konfigurace nahrát patřičnou třídu do paměti.

Velmi důležitá je metoda *getObjekt(Class interface,Element el)*. Předávaný element musí obsahovat atribut `class` s jménem třídy, která se má nahrát. Metoda načte tuto třídu do paměti, ověří, že z dědí rozhraní, které dostala jako první parametr, a že obsahuje statickou metodu *getInstance(Element)*. Tuto metodu pak přes java reflection zavolá a jako parametr jí předá ten samý element, který sama dostala. Tato statická metoda funguje jako konstruktor, který, protože je uložen u třídy pro daný element, zná strukturu jeho obsahu, a dokáže z něj vytvořit novou instanci.

V případě elementů `hrac` a `trener` je požadováno, aby vytvářené objekty dědily z rozhraní *IMozek* respektive *IHraAI*. Konstruktory těchto objektů, ale můžou rekurzivně volat opět getObjekt, aby získali pomocné objekty. Jednotlivé položky v konfiguračním souboru ani nemusí být různé třídy, pokud má daný algoritmus nějaké parametry, může být v souboru uveden vícekrát pokaždé s jinými parametry a s jiným jménem. Díky tomu je celý systém velmi flexibilní. To je vidět i z ukázkového souboru, který obsahuje kus konfigurace pro standardní sadu algoritmů.

Mozek je triviální implementace rozhraní *IMozek*, která své povinnosti pouze deleguje na své podobjekty. Metoda *getInstance()* této třídy tedy zkontroluje, že element obsahuje tři podelementy a pokusí se z nich pomocí *getObjekt()* vyextrahovat objekty, které implementují po řadě rozhraní *ISnimaniAI*, *ILicitaceAI* a *IHraAI*, a zkonstruovat novou instanci. Tyto podelementy uvnitř tedy mohou odkazovat na libovolnou třídu, která implementuje správný interface a splňuje všechny konvence. Je tedy možné použít standardní implementaci a nahradit například jen část, která se stará o licitaci.

Pro dynamické nahrávání tříd a vytváření jejích instancí se používá Java Reflection API. O nahrávání tříd se stará `java.net.URLClassLoader`. Kvůli omezení JVM zůstávají po nahrání třídy v paměti a nelze jednoduše upravit jejich definici. Pokud se změní class soubory je potřeba aplikaci restartovat, aby se změny projevily. To však podle mne není na škodu, protože "dynamika" v tomto případě je důležitá z toho důvodu, aby bylo možné vyrábět algoritmy pro umělé hráče nezávisle na hlavní aplikaci a aby nebylo potřeba aplikaci znova překládat nebo upravovat pokud se objeví nová verze pluginy nebo třeba rovnou úplně nový plugin. Navíc konfigurační soubor se načítá při každém vytváření znovu, takže změny v nastavení jednotlivých hráčů a trenérů se projeví i bez restartu.

Kapitola 5

Struktura trenérů

Algoritmy, které jsou potřeba pro oživení umělých hráčů, nejsou přímou součástí aplikace. Trénovací program na mariáš je naprogramován tak, že tyto algoritmy jsou jen obyčejné objekty implementující daná rozhraní.

Je definována konvence, kam třídy těchto objektů ukládat, jak vytvářet jejich instance a jak ukládat tyto informace do konfiguračního souboru. Systém je pak schopen tyto algoritmy nahrát na požádání, bez toho, aby dopředu znal tyto třídy. Tj. aplikaci lze snadno rozšiřovat. Více viz 4.9.

Základní instalace obsahuje jednu takovou implementaci algoritmů pro umělé hráče, umístěnou do balíčku `marias.ai`. Tento balíček slouží k několika účelům:

- umožňuje, aby bylo se základní instalací vůbec možné hrát mariáš
- poskytuje určité základní třídy pro vytváření algoritmů pro umělé hráče
- obsahuje zajímavé algoritmy sám o sobě.

Jak již bylo napsáno výše, program je napsán tak, že algoritmy pro umělé hráče jsou jen rovnocenné moduly, které jsou mezi sebou zaměnitelné. Náhrada jednoho modulu jiným není jediná strategie při psaní vlastních algoritmů. Balíček `marias.ai` obsahuje spoustu užitečných pomocných tříd, které může jiný modul použít (generátory tahů, ohodnocovače pozic). Navíc je balíček implementován tak, že lze algoritmy z ostatních modulů zakomponovat do tohoto. Tedy jiné moduly nemusí implementovat kompletní sadu algoritmů (licitace, hra, ...), ale mohou nadefinovat jen část a pro zbytek použít tuto implementaci.

Každý umělý hráč potřebuje ke své činnosti objekt implementující interface *IMozek*. Balíček `marias.ai` obsahuje jeho jednoduchou implementaci *Mozek*, která deleguje své povinnosti na tři podobjekty: jeden se stará o snímání (implementuje rozhraní *ISnimaniAI*), druhý o licitaci a flekování (implementuje rozhraní *ILicitaceAI*), třetí pak o výběr nejlepšího tahu (implementuje rozhraní *IHraAI*).

Tyto objekty mohou být klidně i z jiného balíčku (dokud implementují daný interface), ale `marias.ai` obsahuje pro každý typ jednu implementaci: *NahodnySnimac*, *StandardniLicitator* a *SlozenyTrener*. Třída *SlozenyTrener*, jak název napovídá, neobsahuje žádný vlastní algoritmus, pouze obsahuje tři další trenéry – pro betla, pro durcha a pro ostatní hry.

To, které implementace se použijí, se načítá z konfigurace, je tedy možné například jenom implementovat vlastní algoritmus pro betla a nového hráče vytvořit změnou jednoho elementu a zbytek neměnit.

5.1 Snímání

O snímání se stará *NahodnySnimac* – velmi jednoduchá implementace rozhraní *ISnimaniAI*. Jak název napovídá, tato implementace na otázku, kolik karet se má sejmout, odpoví jednoduše tím, že vygeneruje náhodné číslo.

5.2 Licitace

StandardniLicitator je založen na myšlence přiřadit každé hře míru důvěry v to, že se jí podaří uhrát.

Licitovaný mariáš obsahuje 12 licitační stupňů (her, které lze vyhlásit), volený mariáš jenom jejich podmnožinu. Licitátor obsahuje kolekci, kde je pro každou hru připravena instance třídy *Duvera*.

Třída *Duvera* obsahuje jen několik jednoduchých informací:

- Míru důvěry. Celé číslo mezi -50 a 50. Čím vyšší, tím větší jistota, že lze danou hru uhrát.
- Karty, které by se měly odhodit do talonu.
- Barvu trumfů.

Licitátor postupuje tak, že nejdřív nainicializuje kolekci obsahující důvěru v jednotlivé hry. To probíhá tak, že postupně bere jeden licitační stupeň po druhém a generuje všechny možné kombinace trumfů a odhození do talonu. Pro každou kombinaci pak zavolá ohodnocovač. Ta kombinace, která je ohodnocena maximální hodnotou je pak zapsána do kolekce.

Na základě těchto hodnot se pak licitátor rozhoduje. V případě licitace přihazuje dokud existuje hra, která má dostatečně velkou míru dúvěry (větší než daná mez). Podaří-li se získat právo na vyhlášení hry, vybere se jednoduše hra s nejvyšší důvěrou. Pro detaily vyhlášení (barva trumfů, jak odhodit do talonu) se použijí informace, které jsou asociované s danou hrou v objektu typu *Duvera*.

Při flekování se postupuje obdobně. Nejdřív se spočte míra důvěry v to, že protihráč (resp. hráč, který hru vyhlásil) hru uhraje. Pokud je dostatečně nízká, poradí licitátor umělému hráči, aby dal flek.

Aby se předešlo možnému zacyklení (míra důvěry zůstává pochopitelně pořád stejná a tudíž by hráč flekoval do nekonečna) zvyšuje se při každém fleku práh nutný k oflekování. Pokud si je licitátor hodně jistý, může flekovat vícekrát, ale vždy skončí, pokud si není příliš jistý bude flekovat třeba jen jednou.

Jednotlivé ohodnocovače jsou uloženy ve formě statických metod v třídách *BetlAI*, *DurchAI* a*NormalniHraAI*.

Ohodnocovač pro durcha identifikuje chytáky a přiřazuje důvěru podle nich. Betlový ohodnocovač používá prohledávání. Ohodnocení pro ostatní hry je nejsložitější. Je volán nejčastěji (více licitačních stupňů + pro každou barvu trumfů) a musí být tedy dostatečně rychlý. Proto jsem upustil od prohledávání (které je ostatně narozdíl od betla složitější). Ohodnocení je čistě statické a snaží se odhadnout pravděpodobnost výhry podle síly listu. Analyzuje zvlášť závazky týkající se sedmiček (sedma, dvě sedmy) a her (hra, stovka). U sedmiček se snaží podle počtu trumfů a síly doprovodných barev odhadnout, zda je možné sedmu uhrát. U her pak nejdříve spočítá kolik musí hráč uhrát desítek, aby hru vyhrál. Pak zanalyzuje, které desítky jsou jisté, tj. u kterých lze odhadnout, které straně připadnou. O zbytku nepředpokládá nic a uvažuje, že obě strany mají stejnou šanci desítku získat.

5.3 Výběr tahu při normálních hrách

5.3.1 Hraní her

Hraní her stálo u počátku výzkumu umělé inteligence. Od doby, kdy byly vytvořeny první počítače, se lidé snažily o sestrojení stroje, který by byl schopen porazit člověka v šachu. Prvním pokusem o návrh počítačového programu pro hraní šachů je nejspíše proslulý článek Clauda Shannona [8] *Programming a Computer for Playing Chess* zveřejněné roku 1950.

V podstatě se cíl – hrát srovnatelně s člověkem – podařilo splnit, ale trochu jinými prostředky než se původně předpokládalo. Myslím, že Hans Berliner to vystihl velmi dobře [1]: „Myslím, že nejdůležitějším trendem bylo, že počítače za těchto posledních 50 let značně zrychlily. Během tohoto procesu jsme přišli na to, že mnoho věcí, pro které máme přinejlepším antropomorfické řešení, kterým se v mnoha případech nepodařilo zachytit pravé jádro lidské metody, může být vykonáno metodou založené daleko více na hrubé síle, která toliko prohledává, dokud není nalezeno uspokojivé řešeni. Pokud je toto kacířství, budiž.“

Karetním hrám není věnovaná taková pozornost. Mezi nejvíce studované patří nejspíše bridž. I zde zvítězilo prohledávání [2]:

Do roku 1997 se obecně pokoušely bridžové programy hrát podle metodologie lidských hráčů. Úspěšný program Bridge Baron^{TM}, používal například hierarchické plány. GIB představený roku 1998 pracuje jinak, používá “brute-force” prohledávání k analyzování situace, ve které se nechází. Tato technika byla natolik úspěšná, že většina bridžových programů přešla z přístupů založených na bázi znalostí na metody založené na prohledávání.

Mariáš je hra o dost jednodušší a nabízí se tak možnost použít podobné techniky, které se osvědčily u šachových programů. Díky neustálému zrychlování počítačů a menší kombinatorické složitosti lze očekávat, že prohledávácí algoritmy dosáhnou velké úrovně.

5.3.2 Minimax

Teoretickým základem nalezení optimálního tahu je minimax. Tento algoritmus zaručuje optimální strategii pro antagonistickou hru dvou hráčů s úplnou informací a nulovým součtem. To znamená, že dva hráči hrají proti sobě, oba mají všechny informace vztahující se ke hře a na konci hry je zisk vítěze roven ztrátě poraženého.

Prakticky libovolná taková hra může být formálně popsána takto [7]:

- *Počáteční stav*
- *Množina operátorů*, která definuje možné tahy hráčů
- *Test konce*, který rozpozná, kdy hra skončila
- *Účelová funkce*, která vrací číselnou hodnotu jako výsledek hry.

Z této definice lze pak vytvořit strom hry. Kořenem stromu je počáteční stav odkud pro každý legální tah vede hrana do stavů, které vzniknou, pokud se tento tah zahraje. Odtud opět vedou hrany za každý tah, který může udělat protihráč, a tak dále, až se dosáhne listů – koncových stavů, kdy hra skončila a žádný další tah není možný.

Pokud by šlo o normální prohledávání, tak jediné co by MAX musel udělat, by bylo najít sekvenci tahů, které vedou do koncové stavu, který je pro něj příznivý. Naneštěstí MIN asi bude chtít výsledek ovlivnit ve svůj prospěch. MAX tedy musí najít *strategii*, která povede do vítězného stavu bez ohledu na to, jak bude MIN reagovat.

Minimaxový algoritmus dokáže takovou optimální strategii najít a určit tak tah, který by měl hráč zahrát. Sestává z pěti kroků:

1. Vygenerovat celý strom hry.
2. Použít užitkovou funkci na koncové stavy.
3. Použít užitek koncových stavů ke spočtení užitku uzlů o jednu úroveň výše.
4. Pokračovat v počítaní hodnot vyšších uzlů z nižších.
5. Nakonec použít hodnoty spočítané pro úroveň pod kořenem k určení tahu, který zahraje hráč, který je právě na řadě.

Spočítání hodnot uzlu pomocí hodnot všech jeho synů je triviální. MAX vezmeme maximum ze všech hodnot, MIN minimum. Právě toto je nazýváno *minimaxové rozhodnutí*, protože maximalizuje užitek za předpokladu, že oponent hraje dokonale na jeho minimalizování. Není ani potřeba držet celý strom v paměti – na určení hodnoty kořeny lze použít rekurzivní proceduru, která generuje strom postupně, tak jak ho prochází.

5.3.3 Aplikace na mariáš

Při adaptaci minimaxového algoritmu na mariáš je potřeba se vypořádat s určitými detaily. Podmínka nulového součtu je triviálně splněna. V mariáši hrají sice hráči tři, ale jsou rozděleni do dvou táborů, budeme tedy na ně pohlížet jako na dva hráče.

Aktéra budeme považovat za hráče MAX, který se snaží zvýšit hodnotu užitkové funkce. Protihráči pak vytvoří jednoho hráče MIN, který se snaží hodnotu užitkové funkce (zisk aktéra) minimalizovat. Oba se pak řídí společnou strategií pro hráče MIN.

Otázkou zůstává, jak se vypořádat s tím, že mariáš není hra s úplnou informací.

Jednoduchou možností je hru modifikovat a řešit variantu s odhalenými kartami. Lze sice namítat, že v tom případě program podvádí a je oproti lidem ve výhodě, ale ta není tak velká, jak by se mohlo zdát. V mariáši se totiž karty nemíchají pouze snímají. Řekněme tedy, že známe přesné rozložení karet v balíčku (a to po odehrané hře známe). Pozice karet si můžeme očíslovat z vrchu $1, 2, \ldots, 32$. Dále řekněme, že bylo sejmuto x karet ze shora. Měla-li nějaká karta v původním balíčku pozici s, má v novém balíčku pozici $s^{'} \equiv s - p \bmod 32$.

Odtud je snadno vidět, že známe-li složení balíčku, tak po rozdání snadno spočítáme parametr p, tim zjistíme složení balíčku po sejmutí a tím i rozložení karet u soupeřů.

Další námitkou může být, že počítání karet je neetické a nemělo by se to dělat. To je nejspíš pravda u bridže, ale nikoli u mariáše. Pokud je mi známo, tak většina hráčů se víceméně snaží tyto informace použít, i když většinou jen k odhadu, jak jsou rozloženy barvy, či zda je ve hře hláška, či nikoli. Znemožnit toto počítači by naopak poskytovalo výhodu lidem. Počítač dělá to, co mohou dělat lidé, jenom díky své neomylnosti a přesné paměti to dokáže daleko lépe.

Jedině tedy na úplném začátku hry, kdy se hraje se zamíchaným balíčkem, má program k dispozici znalosti, které by jinak nemohl získat. Myslím si, že lze takový případ opomenout a soustředit se čistě na hru s otevřenými kartami.

Počáteční stav odpovídá herní situaci na začátku prohledávání, koncové stavy jsou stavy v hloubce 27, kdy hráčům zbývá po jedné kartě. Výsledek hry je v tu chvíli už jednoznačně určen (hráčům nezbývá než zahrát tu kartu, kterou mají v ruce) a navíc je technicky snadnější určit ohodnocení koncové pozice.

Pokud je minimaxový algoritmus použit na klasické hry dvou hráčů (šachy, dáma, piškvorky, ...), začíná MAX hráč a pak se oba hráči pravidelně střídají. Ustálilo se označovat jako *půltah* (angl. ply) tah jednoho hráče (ať už MAX nebo MIN) a jako *tah* (angl. move) tah jednoho hráče i odpověď na něj.

V mariáši je situace jiná. Nezačíná vždy MAX hráč, ale forhont. Hráči se nestřídají pravidelně, ale hrají v pořadí daném směrem oběhu hodinových ručiček, MIN hráč má dva tahy, a teprve po skončení štychu se na základě toho jaké karty byly odehrány, určí, který hráč bude začínat další štych.

Vzhledem k těmto odlišnostem budu nadále používat trochu jiné označení: *tahem* se myslí zahrání karty (odpovídá *ply*), *štychem* pak série tří tahů, kdy zahrají všichni hráči (odpovídá zhruba *move*).

Abychom mohli realizovat prohledávácí algoritmus potřebujeme:

- reprezentaci karet, rukou jednotlivých hráčů i celé herní pozice
- způsob, jak vytvářet legální tahy
- způsob, jak ohodnotit jednotlivé pozice

5.3.4 Reprezentace karet

V programu je karta reprezentována stejnojmennou třídou. Její instance jsou víceméně čísla od 0 do 31, která na sobě mají definována několik užitečných metod. Ruce hráčů jsou pak reprezentovány pomocí standardních kolekcí jazyka Java.

Tyto objekty jsou vhodné manipulaci v jádře programu, kde se zajišťuje logika hry. Pro prohledávání je ale potřeba o něco kompaktnější reprezentace. Datový typ **int** má v Javě 32 bitů, stejný počet jako je karet v mariáši. Toto pozorování lze využít k snadné reprezentaci karet. Každá karta bude mít přiřazeno číslo s právě jedním bitem nastaveným a to na pozici odpovídající pořadovému číslu karty. Konverze z objektu typu *Karta* na tuto reprezentaci lze provést pomocí jedné instrukce – shiftování jednotky doleva.

Další výhodou tohoto způsobu je snadná reprezentace množiny karet. Není potřeba žádná kolekce, která zabírá proměnlivou velikost paměti. Na zapamatování množiny karet stačí opět jeden jediný integer. Nastavené bity odpojvídají kartám, které jsou v kolekci obsaženy. Navíc množinové operace lze provádět pomocí jediné instrukce.

Jedinou relativně náročnou operací je opětovná extrakce karet z této reprezentace. Převést jednotlivé karty jde ještě lehce – jsou reprezentovány pomocí čísel 2^x, kde $x \in [0, 31]$ a například dělení modulo 37 dává pro všechna tato čísla rozdílné zbytky (vytváří vlastně perfektní hašovací funkci). Není nic jednoduššího než použít tyto zbytky jako indexy do tabulky, kde jsou uloženy skutečné reprezentace karet. Bohužel pro extrakci karet z kolekce asi nezbývá než projít celý integer bit po bitu a čísla postupně extrahovat.

Třída *Karta* navíc obsahuje některé přepřipravené kolekce typu všechny karty dané barvy, všechny karty větší než daná hodnota. Díky tomu lze velice rychle určit např. karty, které přebíjejí jinou kartu.

5.3.5 Pozice

Herní situace je reprezentována pomocí třídy *Pozice*. V ní je uložena:

- karty, které drží hráči v ruce
- karty ve štychu
- hráč na tahu
- historie - počet desítek, které uhrál aktér

První položka je uložena pomocí jednoho integeru jako bitová maska obsahující všechny karty, které jsou ještě ve hře. Abychom z ní mohli zpětně extrahovat ruce jednotlivých hráčů, je potřeba si ještě někde stranou uložit počáteční rozložení karet mezi hráči. Aktuální ruku hráče pak snadno získáme pomocí bitové operace AND.

Důležitou položkou je historie. Pomocí ní je možné v listech určit cenu pozice.

Navíc jsou v pozici ještě uloženy další hodnoty:

- nejlepší tah, který lze z této pozice zahrát
- hloubku, do které byl strom pod touto pozicí prohledáván
- ohodnocení uzlu

Tyto hodnoty nejsou potřeba pro samotný prohledávácí algoritmus, naopak jsou to neinicializované složky, do kterých prohledávácí algoritmus zapíše své výsledky předtím než je zapíše do transpoziční tabulek (viz 5.3.10).

5.3.6 Tahy

Tahy jsou reprezentovány pomocí instancí třídy *Karta*. Na třídě *Pozice* jsou definovány metody, které vytvářejí nové pozice aplikací karet.

O vygenerování legálních tahů se pak stará třída *Tahy*. Pomocí bitových operací a bitmap definovaných v třídě *Karta* je schopna relativně rychle zkonstruovat množinu všech povolených tahů. Tato je pak zkonvertována do seznamu karet (používá se standardní kolekce `java.util.ArrayList`). Vzhledem k tomu, že v mariáši je nutno ctít barvu a přebíjet je většinou množina povolených tahů jednobarevná. Proto se při generování testuje, zda povolené tahy obsahují danou barvu. Pokud ano pokračuje extrakce bit po bitu od nejvyšších hodnot po nejnižší.

Je však použita velmi důležitá optimalizace. Jestliže má hráč sled po sobě jdoucích karet, nezáleží na tom, kterou zahraje. Pozice, které se budou lišit jen tím, která karta byla z toho sledu zahrána jsou si ekvivalentní a vedou ke stejnému výsledku.

Je ovšem třeba dát pozor na to, že to neplatí u desítek a es. Zde sice platí, že možnost získat nebo prohrát další štychy je stejná. Ale zahrání ostré karty má ten vedlejší efekt, že hráč, který ji získá, si připíše body.

Na druhou stranu sledy karet nemusí být jen "statické". Stejná úvaha platí i pro posloupnost karet, která sice není souvislá, ale žádný z protihráčů nemá chybějící karty (buďto jsou v talonu, nebo už byly zahrány). Detekce sledů je tedy "dynamická" a využívá informace a aktuálním rozložení karet ve hře.

Při generování tahů je tedy vždy do seznamu přidána jen nejvyšší karta každého sledu! Toto vylepšení má velký vliv na rychlost prohledávání, protože v mariáši nejsou takové sledy ničím výjimečným a takto se ve většině případu podstatně sníží větvící faktor prohledávaného stromu.

5.3.7 Ohodnocovací funkce

Minimaxový algoritmus má obrovské nároky. Jeho praktický význam je pouze jako teoretický postup, který definuje optimální tah. Pro naprostou většinu her je strom generovaný minimaxem příliš velký, než aby bylo možné ho v rozumném čase projít.

Řešením je nepočítat strom až k listům, ale v určité hloubce ho zaříznout a na listy tohoto nově vzniklého stromu aplikovat *ohodnocovací funkci*, která vrací odhad ceny daného uzlu.

Zavedení ohodnocovací funkce umožňuje redukovat náročnost spočtení tahu na únosnou mez. Cenou za to je, že ztracíme jistotu, že nalezený tah je opravdu optimální.

Častým modelem ohodnocovací funkce je *lineární vážená funkce*, kterou lze vyjádřit jako

$$f(\text{pozice}) = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n,$$

kde f_i jsou příznaky, podle kterých se určuje hodnota pozice a w_i jsou váhy přiřazené těmto příznakům.Ohodnocovací funkce použitá v trenéru je také tohoto druhu.

5.3.8 Alfa-Beta prohledávání

Správnost odhadu většinou závisí na hloubce prohledávání. S tou ale exponenciálně roste i počet uzlů ve stromu. Slabinou minimaxu je, že prozkoumává úplně všechny větve hry bez ohledu na to, zda ovlivní výsledek.

Základním algoritmem pro spočítání minimaxové hodnoty pozice, aniž by se musel prohledávat celý strom je alfa-beta prohledávání – algoritmus vynalezený koncem 50.let minulého století [4].

Vylepšení oproti minimaxu vychází z jednoduchého pozorování: pokud např. hráč MIN je na tahu a zjistí, že nějakým tahem dosáhne ohodnocení menšího než doposud nalezené minimum, může přestat hledat dál a vrátit se, protože hráč MAX v takovém případě raději zvolí strategii, která mu zajistí už nalezené minimum, než by dovolil hráči MIN zahrát právě objevený tah, který by vedl k nižšímu zisku.

Algoritmus je realizován funkcí, která má parametry α a β ($\alpha < \beta$), hloubku do které má počítat, a vrací hodnotu pozice.

Parametry α a β tvoří takzvané *okénko* – otevřený interval, ve kterém se nacházejí hodnoty nalezených pozic. Hodnoty α a β se v průběhu volání mění na nejlepší dosud nalezené odhady. Pokud se hodnota ocitne mimo okénko, můžeme větev zaříznout, v opačném případě nám funkce vrátí správnou minimaxovou hodnotu pozice.

Základní idea alfa-beta algoritmu je: jakmile máme dobrý tah, můžeme rychle eliminovat alternativy, které by vedly k horším výsledkům.

```
function alfabeta(node,depth,a,b)
    if uzel je koncový nebo depth = 0
        return eval(node)
    if hraje MAX
        for each child in childs(node)
            b := min(b,alphabeta(child,depth−1,a,b))
            if a >= b
                return a
        return b;
    else /* MIN je na tahu */
        for each child in childs(node)
            a := max(a,alphabeta(child,depth−1,a,b))
            if a >= b
                return b
        return a
```

Příklad 5.3.1: Pseudokód pro alfabeta algoritmus (na začátku bývá zavolán pomocí: alfabeta(node, d, -INFINITY , +INFINITY)).

5.3.9 Metoda okénka

Procedura alfa-beta prohledávání má dva parametry alfa a beta, které určují tzv. okénko, ve kterém se může hodnota pozice pohybovat. Na začátku je okénko široké – od minus nekonečna do plus nekonečna. Postupem času se zužuje, tak jak se vylepšuje náš odhad ohodnocení pozice. Čím užší je okénko, tím lépe. Více hodnot se do něj nevejde a tedy více větví výpočtu je jednoduše odříznuto jako neperspektivní a nejsou vůbec prohledávány. Proto je vhodné snažit se prohledávat nejdříve nejlepší tahy. Jejich prozkoumáním získáme nejpřesnější odhad a tím i užší okénko.

Nabízí se tedy otázka, zda tohoto efektu nevyužít více a okénko uměle nezmenšit. Takový postup by přinesl větší efektivitu (více prořezávání), ale také by porušil správnost algoritmu. Za předpokladu, že bychom se okénkem trefili do ohodnocení pozice, tj. ohodnocení by leželo uvnitř, bude prohledávání stále korektní. Pokud by ale leželo vně okénka, zahodili bychom optimální tah spolu s ostatními. V takovém případě mohou nastat dvě možnosti:

- naše očekávání je nadhodnocené, v tom případě neexistuje tah, kterým by bylo možné dosáhnout tak vysokého ohodnocení a všechny je

zahodíme jako neperspektivní. Alfa-Beta prohledávání vrátí hodnotu alfa.

- naše očekávání je podhodnocené, v tom případě existuje tah, který je ještě lepší než jsme doufali. Alfa-Beta prohledávání vrátí hodnotu beta.

Pokud tedy spekulativně zmenšíme okénko, je možnost, že prohledávání selže a vrátí neplatnou hodnotu. Naštěstí lze takový stav snadno detekovat (vrácená hodnota je buďto alfa nebo beta). V případě, že taková situace nastane nám nezbývá než upravit okénko a začít prohledávat od znovu.

Existuje více strategií jak volit velikost okénka. Extrémním případem je takzvané nulové okénko - beta je rovno alfa + 1. Takto nastavené prohledávání nemůže nikdy uspět (předpokládám celočíselné ohodnocení), ale na druhou stranu ho lze použít na zodpovězení binární otázky: je ohodnocení menší nebo rovno alfa anebo je ostře větší než alfa?

Vícenásobným položením této otázky, lze dosáhnout odpovědi na skutečné ohodnocení pozice. Efekt většího prořezávání je při této metodě na maximu.

I když se tato myšlenka zdá kontraproduktivní a samostatně asi není příliš účinná, našla si svoji cestu do nejúspěšnějších algoritmů založených na alfa-beta prohledávání.

5.3.10 Transpoziční tabulka

Ta samá pozice se může při prohledávání objevit vícekrát. Stavový prostor pro mariáš je spíše acyklický graf než strom. Proto je vhodné zachovat hodnoty expandovaných uzlů, aby nemusely být počítány znovu. Ukládání pozic spolu s jejich ohodnocením je účelem transpoziční tabulky.

Transpoziční tabulka je implementována jako pole s pevnou velikostí, do kterého jsou pozice ukládány pomocí hašovací funkce.

Položky tabulky jsou přímo *Pozice*. Ty bylo potřeba za tímto účelem rozšířit o nějaké položky. Alternativou by bylo mít separátní datový typ pro prohledávané pozice a položky tabulky, což by vyžadovalo konverze mezi těmito typy a umožnilo by to úspornější využití paměti. Paměťová zátěž není podle mně tak velká a tak v zájmu jednodušší formulace algoritmů jsem se rozhodl pro první možnost.

Do tabulky jsou také ukládáné jen pozice na začátku štychu. Je tomu tak proto, že k transpozici (přejití do společné pozice) může dojít právě jen

na začátku nového štychu.

Použití hašovací funkce s sebou přináší možnost kolizí. Díky tomu, že jsou v tabulce uloženy přímo *Pozice*, lze snadno kolizi detekovat. Při porovnávání dvou pozic se porovnávají: ruce hráčů, historie a hráč, na tahu.

Pokud by se *Pozice* lišila v nějaké z těchto položek, mohla by mít úplně jiné ohodnocení a použití v ní uložených hodnot by mohlo mít za následek nekorektnost algoritmu. Ostatní položky mají význam pomocných údajů, které jsou s pozicí uloženy, a není je tedy potřeba porovnávat.

Kolize při zápisu jsou ošetřeny velmi jednoduchou metodou. Nově ukládáná pozice přepíše starou, pokud byla prohledána do větší hloubky.

5.3.11 NegaScout

NegaScout je dalším algoritmem pro spočítání minimaxové hodnoty pozice. Jde o variantu alfa-beta prohledávání.

Algoritmus je založen na jednoduché myšlence. Alfa-beta prohledávání funguje nejlépe, pokud se nejdříve prohledávají nejlepší tahy. Čím užší je okénko tím rychlejší je výpočet hodnoty, protože může nastat více zaříznutí. NegaScout tedy první tah prohledá normálně a u ostatních tahů se snaží pouze dokázat, že jsou horší. K tomu používá okénko nulové velikosti okolo hodnoty prvního prohledávání. Pokud hodnota pozice vyšší než mez okénka znamená to, že tento tah je lepší než dosud zpracované a provede znovu vyhodnocení tohoto uzlu tentokrát už s dostatečně širokým okénkem.

NegaScout se objevil roku 1989 v disertační práci A.Reinefelda [6] a rychle se stal oblíbeným algoritmem v řadě šachových programů.

5.3.12 MTD(f) algoritmus

MTD(f) algoritmus je poslední z algoritmů, které jsem si pro svou práci vybral. Je ze všech nejmladší. Askee Plat představil tento algoritmus ve své dizertační práci [4] v roce 1996.

Tento algoritmus v sobě snoubí jednoduchost s efektivitou. Vychází z toho, že prohledávací algoritmy byly postupně vylepšovány několika standardními postupy (transpoziční tabulky, iterativní prohlubování). MTD(f) algoritmus tyto vylepšení oproti jiným algoritmům velice silně využívá a je na nich v podstatě závislý.

Sám žádné prohledávání neobsahuje, k tomu využívá jinou prohledávácí proceduru. Jde vlastně o jakousi nadstavbu, která řídí prohledávání.

```
function negascout(node, depth, a, b)
  if uzel je koncový nebo depth = 0
      return eval(node)
  if hraje MAX
    t := b
    for each child in childs(node)
      v := negascout (child, depth−1, a, t)
      if a < v < b and not the first child
        v := negascout(child, depth−1, v, b) /* re−search */
      a := max(a, v)
      if a >= b
        return a /* cut−off */
      t := a+1 /* nulové okénko */
    return a
  else ...
```

Příklad 5.3.2: Pseudokód pro negascout.

Písmeno f v názvu odkazuje na fakt, že procedura vyžaduje nějaký prvotní odhad ohodnocení uzlu. Čím blíže bude skutečnému ohodnocení tím lépe.

Algoritmus po té zorganizuje prohledávání následujícím způsobem. Vytvoří okolo prvotní hodnoty nulové okénko a spustí prohledávací algoritmus. Podle toho, na kterou stranu prohledávání selže posune okénko doleva nebo doprava. Potom v posunování pokračuje, dokud se "nezmění znaménko". V tu chvíli byla nalezena správná hodnota uzlu.

Tento postup se může zdát jako zbytečné plýtvání, protože nalezení správného ohodnocení vyžaduje vždy minimálně dvě prohledávání. Idea algoritmu je taková, že prohledávání je sice víc, ovšem díky nulovému okénku jsou daleko rychlejší. První prohledávání také zaplní transpoziční tabulku pozicemi a další iterace tak nemusí uzly opakovaně expandovat.

5.3.13 Iterativní prohlubování

Uvedené metody prohledávání mají jednu slabinu. Nedokážou vyhledat optimální tah v reálném čase. Dopředu nelze přesně odhadnout jak velký strom bude potřeba prohledat a kolik času na to bude potřeba. Je třeba algoritmus nechat doběhnout. Bylo by možné vrátit nejlepší doposud nalezený

```
function MTDF(root, f, d)
    g := f
    upperBound := +INFINITY
    lowerBound := −INFINITY
    while lowerBound < upperBound
        if g = lowerBound then
            beta := g + 1
        else
            beta := g
            g := AlphaBeta(root, beta−1, beta, d)
        if g < beta then
            upperBound := g
        else
            lowerBound := g
    return g
```

Příklad 5.3.3: Pseudokód pro MTD(f).

tah, tato volba by ale byla silně ovlivněna kvalitou uspořádání tahů. Pokud bychom dobrý tah zařadili někam na konec, nebyl by zahrán.

Jednoduchou odpovědí na požadavek předběžného nejlepšího tahu je technika nazývaná iterativní prohlubování (iterative deepening) [7].

Začneme nejdříve prohledávat stromy menších hloubek a postupně zvyšujeme hloubku. Nejlepší tah na konci každého prohledávání si zapamatujeme. Čím déle necháme tento postup běžet, tím prozkoumáme větší hloubku a tím přesnější máme výsledek. Vždy ale máme relativně dobrý odhad (odpovídající času, který jsme měli na jeho získání), který můžeme v případě potřeby ihned vrátit.

Na první pohled se tato technika může jevit jako neskutečné plýtvání, protože vrchní vrstvy stromu budeme prohledávat opakovaně. Důležité je si uvědomit, že strom má typicky na nižších úrovních několikanásobně více uzlů než na úrovni o jedna výš. Nezáleží tudíž tolik na tom, že jsou uzly blíže kořenu prohledávány vícekrát.

Navíc tento způsob velmi dobře kooperuje s transpozičními tabulkami. Při prohledávání jsou pozice ukládány do tabulky. Při přechodu na vyšší hloubku jsou zde uložené hodnoty zastaralé, ale není vůbec vhodné ji pročistit! Naopak zde uložené hodnoty lze použít jako odhady a použít je v další

fázi prohledávání na příklad k uspořádání tahů od nejlepších po nejhorší.

Na iterativní prohlubování se tak vlastně také dá dívat jako na způsob, jak si naplnit traspoziční tabulky užitečnými daty a ty pak použít ke zrychlení prohledávání.

5.4 Výběr tahu při betlu

Hledání optimálního tahu pro betla je postaveno na stejných myšlenkách jako pro ostatní hry. Jsou tu však určité rozdíly. Hra může skončit předčasně, je-li solo hráč donucen vzít štych – to vede k velkému zrychlení. Další zrychlení plyne z toho, že betl se většinou hraje za takového rozložení karet, kdy mají hráči souvislé bloky karet. Díky tomu se výrazně snižuje větvící faktor stavového prostoru.

Pro hledání tahu je použit jednoduchý prohledávácí algoritmus oddělený od algoritmů pro normální hry.

Je tomu tak proto, že u betla lze dopočítat celý strom hry a není potřeba se snažit ohodnotit jednotlivé pozice. Navíc odlišnosti (jiná pravidla pro generování tahů apod.) by zbytečně zkomplikovala algoritmy pro ostatní hry, kde je potřeba co největší výkon, zatímco v případě betla postačí jednodušší varianta. Nakonec nejpádnějším důvodem pro separaci je to, že algoritmus pro betla je starší a byl implementován daleko dříve, než prohledávací algoritmy pro ostatní hry.

5.5 Výběr tahu při durchu

Při durchu musí aktér uhrát všechny štychy. Pokud má nějaký protihráč kartu, která může přebít kartu aktéra a zároveň k ní má dostatečný počet karet stejné barvy, tak aby ho aktér nemohl této karty zbavit pomocí pravidla o ctění barvy, říkáme, že má takzvaného trháka.

Strategie pro durcha je jednoduchá. Snadnou analýzou rozložení karet ve hře lze identifikovat trháky. Protihráči hrají tak, aby vyhověli pravidlům, ale nezbavovali se trháků.

Aktér hraje podle strategie, co nejdéle odkládat zahrání karty, kterou mu může protihráč přebít (a doufat, že ji protihráč dříve zahodí).

Kapitola 6

Závěr

Cílem této práce bylo vytvořit program na hraní mariáše a navrhnout algoritmy pro hráče řízeného počítačem. Výsledkem je jednouživatelská GUI aplikace napsaná v jazyce Java (verze 5.0).

6.1 Srovnání s ostatními programy

Mariáš je oblíbená karetní hra, proto je s podivem, že neexistuje mnoho programů na jeho hraní. Při zkoumání podobně zaměřených programů, jsem jich mnoho nenašel. Legendární jsou programy Flek! a Re!, které pocházejí až roku 1993. Jedná se ještě o DOSové aplikace.

Další programy, na které jsem narazil byly Čtyřka Jakuba Vrány, reklamní Becherovka mariáš Jana Fialy a Mariáš Martina Staufčíka – všechny pouze pro platformu Windows.

První z nich umí čtyřku (variantu mariáše pro čtyři hráče), ostatní pouze volený mariáš. Vyjma Čtyřky mají všechny programy napevno nastavené algoritmy pro umělé hráče. Všechny programy v podstatě nabízejí simulaci mariáše a snaží se hru co nejvíce přiblížit skutečné hře. Většinou obsahují konfigurační možnosti, které umožňují změnit vzhled aplikace, případně jednoduché modifikace pravidel. Způsob jejich fungování mi není znám, ale lze předpokládat, že používají nějaký druh prohledávání.

Trénovací program na mariáš tedy přináší několik nový rysů.

- Je to jediný program, který je schopen hrát zároveň licitovaný a volený mariáš (a kromě Re! jediný program, který umí hrát licitovaný mariáš, o kterém vím).

- Díky tomu, že je napsaný v Javě, je velice přenositelný a lze si tak zahrát mariáš například i na unixových systémech (více viz 4.1).

- Trénovací program na mariáš má především odlišnou filosofii. Narozdíl od předchozích programů umožňuje program nejenom hraní, ale i analýzu herní situace. V GUI je kladen důraz víc na přehlednost než na hezké animované zpracování hry. Ukládání her, jejich nastavování, možnost odhalení karet a schopnost prochát různé větve hry je naprosto unikátní.

- Další významnou vlastností je otevřenost celého systému. Snadno lze naprogramovat vlastní strategii pro umělého hráče. Stačí jen napsat třídu implementující daná rozhraní. Navíc není potřeba programovat vše, lze využít třídy, které už jsou napsané, a připsat si jen určitou část.

- Lze pořádat turnaje mezi umělými hráči. Průběh a výsledek takového turnaje se zapisuje do XML souboru. Tento výstup pak může být použit k porovnání různých algoritmů. Nabízí se taky možnost použít metod strojového učení a pokusit se umělého hráče "natrénovat".

6.2 Zhodnocení algoritmů

Detailní popis použitých algoritmů je předmětem kapitoly 5. Při jejich návrhu jsem vycházel z toho, že s určitou dávkou nadsázky lze mariáš brát jako hru dvou hráčů s úplnou informací. Za tohoto předpokladu se nabízí využít výsledků, kterých bylo dosaženo v oblasti šachových programů za posledních 50.let a pokusit se přenést některé postupy na mariáš. Vzhledem k tomu, že stavový prostor mariáše je podstatně menší než u šachů, daly se předpokládat dobré výsledky.

Součástí práce je implementace třech prohledávácích algoritmů: alfa-beta prohledávání, NegaScout a MTD(f). Důraz byl kladen na rychlé a co nejefektivnější prohledávání stavového prostoru. Protože takové prohledávání má velkou časovou složitost a nelze dopředu odhadnout, jak dlouho bude výpočet trvat, je celý systém navržen tak, aby bylo možné prohledávací algoritmus přerušit a použít nejlepší dosud zahraný tah.

Lze konstatovat, že implementace prohledávacích algoritmů dopadla výborně. Trénovací program na mariáš je běžně schopen v krátkém čase (<1s) z počáteční pozice prohledávat až do hloubky 6 tahů! Ve většině případů

dokáže také spočítat hru až do úplného konce a najít tak opravdu optimální tah.

Provedl jsem jednoduché testy na 20 vybraných partií, ze kterých vyplynulo, že největší vliv na efektivitu prohledávání mají transpoziční tabulky. Rozdíly mezi jednotlivými algoritmy nebyly nijak markantní, i když se potvrdil předpoklad, že novější postupy budou lepší a pořadí algoritmů bylo: alfa-beta prohledávání, NegaScout a MTD(f).

Implementace má i svoje slabiny.

Licitace používá pouze statické ohodnocení, protože prohledávání všech možných variant by zabralo příliš mnoho času. Z toho také plyne její nedokonalost.

Hraní jednotlivých tahů je na tom už podstatně lépe, ale zde se projevuje "počítačový styl hry". Strategie, kterou nalezne prohledávací algoritmus vychází z toho, že všichni hrají optimálně. Počítač se tedy nesnaží spoluhráči předávat informace o tom jaké má karty, spoléhá na to, že on bude hrát také optimálně, a pokud to nezmění ohodnocení hry, je schopen klidně zahrát nesmyslný tah (jako např. namazat protihráči desítku, pokud ji stejnak nemůže uhrát nebo dvacet bodů nezmění výsledek hry).

6.3 Možná vylepšení

Trénovací program na mariáš by šlo určitě vylepšit. Možné nápady jsou:

- Vylepšit grafické rozhraní. Důraz byl sice kladen na přehlednost než na líbivost, ale i tak by šlo GUI vylepšit.
- Statistiky. Program by šlo rozšířit o sledování statistik hráče.
- Více variant pravidel. Pravdila mariáše nejsou ustálené existuje spousta modifikací. Program by šlo rozšířit o jejich nastavení.
- Lepší licitace.
- Lidštější způsob hraní. Pokud by se použila nějaká heuristická funkce pro uspořádání tahů, mohly by "logičtější" tahy přijít na řadu první a program by hrál více jako člověk.
- Nepodvádět. Nahlížení do karet lze považovat za podvádění. Jak jsem psal v 5.3.3, není velký problém si spočítat rozložení karet z předchozích her. Variantu se zakrytými kartami lze řešit pomocí Monte Carlo

metody, kdy hráč vybere několik rozložení karet, provede na nich prohledávání a výsledky zprůměruje. Filosofickou otázkou ovšem je, jak generovat rozložení. Pokud by algoritmus nic nepředpokládal a každé rozložení bral se stejnou pravděpodobností byl by oproti lidskému hráči v nevýhodě. Pokud by používal všechny dostupné informace, měl by prakticky dokonalý přehled o hře a nemusel by hádat.

Literatura

[1] H.Berliner: *AI's greatest trends controversies*, IEEE Intelligent Systems Magazine 15 (January /February) (2000) 9

[2] M.Ginsberg: *GIB: Imperfect information in a computationally challenging game*, Journal of Articial Intelligence Research, 2001, 303–358

[3] B. Hrabal: Hovory lidí. Československý spisovatel, Praha 1984

[4] Aske Plaat: *Research RE: Search & Re-Search*, 1996

[5] K. Poláček: *Hráči*, Nava (2006), ISBN: 80-7211-216-3

[6] A. Reinefeld: *Spielbaum-Suchverfahren*. Informatik-Fachbericht 200, Springer-Verlag, Berlin (1989)

[7] Russell S., Norvig P.: *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence. (1995), 123–126, ISBN: 0-13-103805-2

[8] Claude Shannon: *Programming a Computer for Playing Chess*, Philosophical Mag., 41(7), 256–277 (1950).