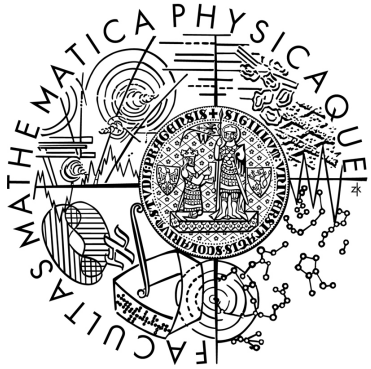
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Matej Moravčík

Umelá inteligencia pre hru Texas Holdem poker

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Zuzana Reitermanová

Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2011

Poďakovanie

Na úvod svojej bakalárskej práce by som sa rád poďakoval všetkým, ktorý ma podporovali v priebehu prípravy tejto práce.

V prvom rade ďakujem svojej vedúcej, Mgr. Zuzane Reitermanovej, za odborné vedenie, cenné rady a prínosné pripomienky.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne ..... Matej Moravčík

Název práce: Umelá inteligencia pre hru Texas Holdem poker

Autor: Matej Moravčík

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Zuzana Reitermanová, Katedra teoretické informatiky a matematické logiky

Abstrakt

V poslednej dobe nastal veľký rozmach pokru. Týka sa to živej hry, rovnako ako hry na internete. Pre začínajúcich hráčov však môže byť problém nájsť protihráčov so solídnymi schopnosťami a zlepšovať tak svoju hru bez vkladu vlastných finančných prostriedkov. Ako riešenie sa ponúka využitie umelej inteligencie. Dostupných programov, venujúcich sa turnajovej hre je však málo.

Táto práca ukazuje celkový návrh a tvorbu takejto aplikácie, určenej špeciálne pre turnajový variant hry Texas Holdem poker. Najväčšia pozornosť je venovaná umelej inteligencii. Sú rozobraté dva hlavné postupy jej tvorby a to *aproximácia Nashovho equilibria* a použitie expertného systému. Dôraz je kladený na prvú z možností. Hlavný prínos práce spočíva v podrobnom predstavení a porovnaní troch algoritmov na na výpočet *aproximácie Nashovho equilibria.* V dvoch prípadoch sa jedná o originálne heuristiky, využívajúce špecifickú štruktúru pokrovej hry. Algoritmy boli implementované a boli empiricky vyhodnotené ich vlastnosti.

Konečným výsledkom práce je plnohodnotná aplikácia, určená pre koncového používateľa. Simuluje pokrovú hru, poskytuje silnú umelú inteligenciu a atraktívne grafické užívateľské prostredie.

Klíčová slova: Poker, Texas Holdem, Umelá inteligencia, Nashovo equilibrium

Title: Artificial intelligence for Texas Holdem poker

Author: Matej Moravčík

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Zuzana Reitermanová, Department of Theoretical Computer Science and Mathematical Logic

Abstract

Recently there has been a great expansion of poker. This includes live games, as well as games on the internet. For beginners, it may be difficult to find opponents skilled enough and thus improve their gaming performance without deposit of their own funds. Using of artificial intelligence seems as good solution for the problem, but there are only few suitable programs available.

This thesis describes the overall design and development of such an application, specially designed for tournament variant of Texas Hold'em poker. Most attention is devoted to the artificial intelligence. There are two main approaches discussed - *approximate Nash equilibrium* and the use of expert system. Emphasis is placed on the first option. The main contribution of this thesis is detailed description and comparison of three algorithms for calculating the *approximation of Nash equilibrium*. Two of them are original heuristics algorithms, that take advantage of specific structure of poker game. Algorithms have been implemented and their properties have been empirically evaluated.

The final result is a full-featured application designed for end users. It simulates poker game and provides a powerful artificial intelligence with attractive graphical user interface.

Keywords: Poker, Texas Holdem, Artificial Intelligence, Nash equilibrium

Obsah

1. Úvod

1.1. Motivácia

V poslednej dobe nastal veľký rozmach pokru. Týka sa to živej hry, rovnako ako hry na internete. Pre začínajúcich hráčov však môže byť problém nájsť protihráčov so solídnymi schopnosťami a zlepšovať tak svoju hru bez vkladu vlastných finančných prostriedkov. Väčšina internetových herní síce ponúka turnaje v ktorých sa nehrá o reálne peniaze ale len o fiktívne (tzv „play money“), ale takého hry nemotivujú oponentov k zodpovednej hre. Preto si hráč, ktorý sa zúčastňuje týchto turnajov, často namiesto zlepšenia svojej hry osvojí zlé návyky, ktoré ho potom v hre o reálne peniaze môžu stáť nemalé finančné prostriedky.

1.2. Stručný popis práce

Cieľom tejto práce bolo vyvinúť umelú inteligenciu, proti ktorej by bolo možné poker trénovať a doplniť ju o simuláciu samotnej hry a o grafické rozhranie.

Úvodná časť práce je venovaná priblíženiu pokrovej hry čitateľovi. V druhej kapitole najprv stručne opíšeme pravidlá pokru. V tretej kapitole sa potom pozrieme na základné aspekty stratégie hry v jednostolových turnajoch.

Nasleduje časť práce venovaná umelej inteligencii. Najprv, v štvrtej kapitole, popíšeme dva hlavné prístupy použité na tvorbu umelej inteligencie a to *aproximáciu Nashovho equilibria* a použitie expertného systému. V piatej kapitole sa budeme venovať modelu používanému na prevod žetónov v hre na očakávanú výhru. Ukážeme, že tento model hrá pri tvorbe umelej inteligencie dôležitú úlohu. Bude tiež vysvetlené fungovanie modelu a možnosti jeho implementácie.

Ďalší blok práce, kapitoly šesť až jedenásť, sa bližšie venujú *aproximácii Nashovho equilibria*. V posledných troch kapitolách tohto bloku rozoberáme postupne tri algoritmy určené na *aproximáciu Nashovho equilibri*a a empirické vlastnosti ich implementácie. V dvoch prípadoch sa jedná o originálne algoritmy, využívajúce štruktúru našej abstrakcie hry. Kapitola dvanásť je venovaná pomocnému výpočtu *equity,* potrebnému v každom z predchádzajúcich algoritmov. Od rýchlosti prevádzania tohto výpočtu najviac závisí časová zložitosť implementácie umelej inteligencie. Kapitola trinásť stručne opíše výpočet predpočítaných tabuliek, potrebných pre umelú inteligenciu. V nasledujúcej kapitole sa rýchlo pozrieme na výpočet *Nashovho equilibira* pre špeciálny prípad dvoch

hráčov, kde môžeme použiť lineárne programovanie. Posledná kapitola, ktorá sa venuje hre s malým počtom žetónov je kapitola pätnásť, v ktorej opíšeme špeciálne výpočty pre jednotlivé kartové kombinácie.

Nasleduje kapitola venovaná expertnému systému a časť práce venovanú umelej inteligencii uzatvára zhodnotenie umelej inteligencie.

V krátkosti sa ďalej v kapitole osemnásť venujeme zvyšným aspektom programu a to najmä grafickému rozhraniu. Prácu uzatvára užívateľská dokumentácia a záverečné zhodnotenie.

Vyvíjaný softvér pracuje pre variantu pokru Texas Holdem. Špecializuje sa na verziu no limit, kde nie je obmedzená veľkosť stávky, a na turnaje odohrávajúce sa na jednom stole. Vývoj prebehol pre operačný systém Linux. Program bol napísaný v jazyku C++ v prostredí Qt Creator. Beh programu bol testovaný v distribúcii Ubuntu.

Samotný program, jeho zdrojový kód a programátorská dokumentácia sú uložené na DVD priloženom k práci.

2. Pravidlá Texas Holdem pokru

Poker sa delí na viacero variantov. V súčastnosti najpopulárnejším a najviac hraným je variant Texas Holdem. Pre tento variant je aplikácia určená a budeme sa mu ďalej venovať. Hrá sa z 52 žolíkovými kartami.

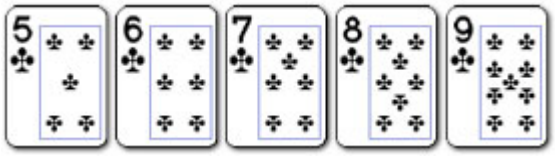
Vo variante pokru Texas Holdem dostane najprv každý hráč dve karty (vlastné karty známe ako „hole cards“ alebo tiež „hand“), ktoré patria iba jemu. Ďalších päť kariet je postupne rozdaných lícom nahor, aby zostavili tzv. „board“. Všetci aktívni hráči hrajú potom s týmito spoločnými kartami spolu s vlastnými dvomi a snažia sa vytvoriť čo najlepšiu kombináciu piatich kariet. V Holdem pokri môže hráč použiť týchto sedem kariet aby zostavil najlepšiu kartovú kombináciu o piatich kartách, s použitím žiadnej, jednej alebo oboch vlastných kariet.

2.1. Hodnoty kartových kombinácií

Uvádzame prehľad možných kartových kombinácií usporiadaných od najsilnejšej po najslabšiu [3].

2.1.1. Postupka vo farbe (Straight Flush)

Jedná sa o päť kariet rovnakej farby, ktorých hodnoty nasledujú bezprostredne za sebou (Obrázok 1). V prípade rovnosti kombinácií: Postupka zakončená vyššou kartou vyhráva. Najlepšia postupka sa nazýva Kráľovská postupka (Royal Flush) a tvorí ju eso kráľ, dáma, jack a desiatka rovnakej farby. Kráľovská postupka je neporaziteľná kombinácia.



Obrázok 1: Postupka vo farbe

2.1.2. Štvorica (Four of Kind)

Jedná sa o štvoricu kariet rovnakej hodnoty a jednu doplnkovú kartu (tzv. Kicker, Obrázok 2). V prípade rovnosti kombinácií vyhráva vyššia štvorica, v prípade rovnosti štvoríc vyhráva hráč s vyšším kickrom.

Obrázok 2: Štvorica

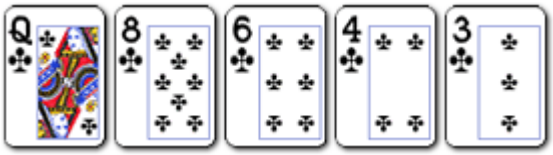
2.1.3. Full House

Jedná sa o trojicu a dvojicu kariet rovnakej hodnoty (Obrázok 3). V prípade rovnosti kombinácií vyhráva vyššia trojica, v prípade rovnosti trojíc vyhráva hráč s vyššou dvojicou.

Obrázok 3: Full House

2.1.4. Flush

Jedná sa o päť ľubovoľných kariet rovnakej farby (Obrázok 4). V prípade rovností kombinácií vyhráva hráč s najvyššou hodnotou karty. V prípade potreby sa berie druhá, tretia, štvrtá, alebo piata hodnota najvyššej karty.



Obrázok 4: Flush

2.1.5. Postupka (Straight)

Postupka zložená z piatich kariet ľubovoľnej farby, ktorých hodnoty nasledujú bezprostredne za sebou (Obrázok 5). V prípade rovností kombinácií vyhráva postupka zakončená vyššou kartou.

Obrázok 5: Postupka

2.1.6. Trojica (Three of kind)

Jedná sa o tri karty rovnakej hodnosti a dve ľubovoľné doplnkové karty (Obrázok 6). V prípade rovností kombinácií vyhráva vyššia trojica. V prípade zhodných trojíc rozhoduje najvyššia, prípadne druhá najvyššia doplnková karta.

Obrázok 6: Trojica

2.1.7. Dva páry (Two Pair)

Jedná sa o dve dvojice kariet rovnakej hodnosti a jednu doplnkovú kartu (Obrázok 7). V prípade rovnosti kombinácií rozhoduje vyššia dvojica. V prípade rovnosti týchto dvojíc rozhoduje nižšia dvojica, v prípade ich rovnosti vyššia doplnková karta.

Obrázok 7: Dva páry

2.1.8. Pár (One pair)

Jedná sa o dvojicu kariet rovnakej hodnosti a tri ľubovoľné doplnkové karty (Obrázok 8). V prípade rovnosti kombinácií vyhráva vyšší pár. V prípade rovností párov vyhráva hráč s najvyššou prvou druhou, alebo treťou doplnkovou kartou.

Obrázok 8: Pár

2.1.9. Vysoká karta (High card)

Jedná sa o prípad, že hráč nemá žiadnu z uvedených kombinácií (Obrázok 9). V prípade rovnosti kombinácií: Vyhráva najvyššia karta, v prípade zhody druhá najvyššia, až posledná najvyššia.

Obrázok 9: Vysoká karta

2.2. Priebeh hry a sádzkové kolá

Samotná hra hráčov spočíva vo vykonávaní sádzok, prípadne zložení kariet hráčov.

2.2.1. Blindy a button

Na začiatku každého kola hry Texas Holdem sú dve veci, ktoré sa nemenia: button a blindy. Button je kotúč umiestnený pred hráčom, ktorý označuje krupiéra tohto kola.

Blindy sú nútené stávky (nazývajú sa tak preto, lebo hráči ich musia vložiť „naslepo“ predtým, ako vidia akékoľvek karty), ktoré vkladajú dvaja hráči nachádzajúci sa bezprostredne naľavo od značky button. Prvý hráč vloží small blind a druhý hráč vloží big blind. Small blind obvykle predstavuje polovicu žetónov big blindu. Po každom kole sa button posunie o jedno miesto doľava, rovnako ako aj oba blindy.

2.2.2. Preflop

Jedná sa o fázu hry pred vyložením spoločných kariet. Keď hráč dostane karty na ruku, čelí prvému rozhodnutiu: hrať alebo nehrať. Hráč sa musí rozhodnúť, či pokračuje alebo vystúpi, a toto urobí zrovnaním otvorenej stávky (tzv. call), keď sa na neho dostane poradie, alebo zvýšením tejto sumy.

Dobrí hráči takmer nikdy nezrovnajú a volia navýšenie, aby hráč na veľkom blinde nemal šancu zo slabou štartovnou kartovou kombináciou zadarmo pokračovať v hre a zostaviť lepšiu kombináciu pomocou spoločných kariet.

Stávkovanie ďalej prebieha v slede, počnúc prvým hráčom naľavo od krupiéra a pokračuje postupne okolo stola v smere hodinových ručičiek, až kým sa dorovnajú všetky stávky, alebo hráči zložia svoje karty.

Hráč má na výber z nasledujúcich možností: zložiť, dorovnať, alebo navýšiť. Hra pokračuje okolo stola v preflope a hráči, ktorí si želajú zostať v hre, musia najmenej dorovnať big blind, alebo, ak predchádzajúci hráč navýšil (zvýšil) stávku - potom musia stávku najmenej dorovnať, aby zostali v hre. Hráč má vždy možnosť zložiť karty, keď na neho príde rad.

Ak nedôjde k zvýšeniu stávky, keď sa dostane na ťah hráč, ktorý vložil big blind, tento hráč má stále možnosť zvýšiť stávku, ak si to želá. Všetky peniaze, ktoré hráč vložil do banku pred rozdaním kariet, sa považujú za súčasť jeho celkovej stávky. Takže, ak sú blindy $1 (small blind) a $2 (big blind) a predtým, ako sa dostane rad na

small blind, sa nezvýši stávka, potom musí vložiť do banku len ďalší $1, aby dorovnal $2, pretože už $1 investoval pred rozdaním kariet. Po zrovnaní všetkých stávok sa hrá flop.

2.2.3. Flop

Jedná sa o fázu hry keď sa prvý krát rozdajú spoločné karty, tzv folp. Flop tvoria tri karty, ktoré sa rozdajú odkryté v strede stola. Ide o spoločné karty, čo znamená, že ich zdieľajú všetci hráči a môžu tvoriť súčasť kombinácie ktoréhokoľvek hráča.

Teraz nasleduje kolo stávkovania, počnúc hráčom bezprostredne naľavo od značky button a pokračuje v smere hodinových ručičiek okolo stola.

Každý hráč bude mať znova možnosť zložiť, hlásiť check, dorovnať, alebo navýšiť. Check je situácia, keď momentálne nie je otvorená žiadna stávka, ktorú by hráč musel dorovnať, takže hráč sa môže rozhodnúť pokračovať hlásením check (v podstate zrovnaním nulovej stávky). Stávkovanie pokračuje podľa vyššie uvedených pravidiel, až do zrovnania všetkých stávok. Potom sa rozdá ďalšia karta

2.2.4. Turn

Karta turn (alebo štvrtý street) sa položí odkrytá na stred stola a stáva sa ďalšou spoločnou kartou. Kolo stávkovania potom pokračuje rovnakým spôsobom, ako v prípade flopu. Hráči, ktorí pokračujú v hre po tomto kole stávkovania, sa dostanú k poslednej karte.

2.2.5. River

Posledná karta je river (alebo piaty street), ktorá sa položí odkrytá na stred stola a stáva sa tiež spoločnou kartou. Hráči teraz majú všetky karty, ktoré mohli získať. Cieľom je dosiahnuť najlepšiu kombináciu piatich kariet, zo siedmych použiteľných kariet (dvoch kariet na ruke a piatich spoločných kariet).

Posledné kolo stávkovania teraz pokračuje rovnakým spôsobom, ako predchádzajúce kolá. Hráči, ktorí zostanú v hre po dokončení stávkovania, potom vyložia karty.

2.2.6. Vyloženie kariet

Po dokončení všetkých stávok nasleduje vyloženie kariet hráčmi, ktorí zostali v hre. Ak sa v poslednom kole vložila stávka, potom hráč, ktorý urobil poslednú kladnú akciu (hráč, ktorý ako posledný stavil – nie dorovnal) vyloží karty ako prvý.

Ak v poslednom kole stávkovania všetci hráči zvolili „check“, potom ako prvý vyloží karty hráč, ktorý sa nachádza bezprostredne naľavo od značky button a pokračuje sa ostatnými hráčmi v smere hodinových ručičiek.

V hre Texas Holdem Poker hráč môže použiť akúkoľvek kombináciu dvoch kariet na ruke a piatich kariet na stole (spoločných kariet). Hráč môže dokonca len „hrať s kartami na stole“ a použiť len spoločné karty na vytvorenie vlastnej kombinácie (toto sa líši od ostatných foriem pokeru).

Hráč s najlepšou kombináciou piatich kariet vyhráva bank. Ak hráči dosiahnu kombinácie rovnakej hodnoty, pokrový bank sa rozdelí rovnakým dielom medzi hráčov.

Button sa teraz premiestni v smere hodinových ručičiek na nasledujúceho hráča a začne sa nové kolo.

2.2.7. Veľkosti sádzok, all-in situácie

Podľa maximálnej veľkosti sádzok sa Texas Holdem poker delí na tri druhy:

-limit: veľkosti stávok sú pevne určené

-pot limit: maximálna stávka je rovná veľkosti potu(banku)

-no limit: maximálna stávka nie je obmedzená, hráč môže staviť všetky žetóny, ktoré má hráč k dispozícii (tzv. stack).

All-in (alebo push) sa nazýva akcia, keď hráč vsadí všetky svoje žetóny. Potom sa už hráč ďalej nezúčastňuje na hre a nie je donútený dorovnať ďalšie sádzky. Ak ostatní hráči nezložia svoje karty, každá sádzka naviac po dorovnaní all-inu tvorí druhý pot (side pot), o ktorý sa potom delia len títo hráči.

2.3. Turnajové pravidlá

Poker sa delí na dva základné smery:

Cash game – hrá sa priamo o peniaze – hráč si môže kedykoľvek prisadnúť ku hraciemu stolu a vymeniť svoje peniaze vymeniť za hracie žetóny, v prípade prehry čipy dokúpiť. Keď sa hráč rozhodne stôl opustiť vymení žetóny naspäť za hotovosť.

Turnaje – hra sa začína keď sa zíde vopred stanovený počet hráčov. Každý z nich potom zaplatí vstupný poplatok (buy-in), ktorý pozostáva zo sumy určenej na ceny pre hráčov (prize pool) a poplatku herni (rake, zvyčajne 4-10% z celkovej čiastky). Každý turnaj má danú výplatnú štruktúru, ktorá určuje rozdelenie prize pool-u medzi hráčov podľa ich umiestnenia v turnaji.

Turnaje sa podľa veľkosti delia na dve základné skupiny:

-S**ingle table turnaje** (tiež známe ako sit and go - **SNG** turnaje) – turnaj sa odohráva len na jednom hracom stole

-M**ulti table turnaje** (MTT) – turnaj sa odohráva na viacerých stoloch súčasne. Ďalej sa budeme venovať už len SNG turnajom pre ktoré je aplikácia určená.Na začiatku turnaja každý hráč dostane rovnaký počet žetónov. Samotná hra prebieha podľa opísaných pravidiel, hráči postupne vypadávajú, keď prídu o všetky svoje žetóny. Hráč sa umiestni na n-tom mieste v prípade, že v hre pri ktorej vypadol z turnaja ostávalo ešte n hráčov. Ak viac hráčov vypadne naraz ich umiestnenie sa určí podľa ich stackov na začiatku danej hry.

V prípade že za dané miesto bola vypísaná odmena hráč ju zinkasuje. Turnaj pokračuje, až kým neostane v hre iba víťaz.

Špecifikom turnajov je zvyšovanie blindov po určitom čase. Deje sa tak, aby boli donútení hrať aj pasívni hráči a turnaj sa skončil v rozumnom čase.

Ďalšie špecifikum je Ante. Jedná sa o ďalšiu povinnú stávku, ktorú je povinný vložiť naslepo každý hráč pri stole. Je používaná len v niektorých herniach v neskoršej fáze hry. Robí hru agresívnejšou a akčnejšou, pretože sa hrá o väčší pot.

3. Stratégia hry jednostolových turnajov

Hra jednostolových turnajov má špecifické strategické aspekty. Tie sú dôležité pre ľudských hráčov, aj pre umelú inteligenciu. Poďme sa na ne bližšie pozrieť.

3.1. Equita

Pretože v pokri o výsledku do značnej miery rozhoduje náhoda, zavádzame pojem equita. Jedná sa o strednú hodnotu získaných žetónov, alebo peňazí cez všetky výsledky hry, podľa ich pravdepodobností.Rozdeľujeme ju na dva varianty:

cequity – chip equity – očakávaná výhra čipov

$equity – očakávaná výhra peňazí

Pri cash game je $equita priamo úmerná cequite, pretože čipy predstavujú skutočné peniaze. Pri turnajovej hre to však neplatí.

Dôvodom je to, že cieľom hráča nie je turnaj vyhrať, ale v dlhodobom horizonte sa umiestňovať na takých pozíciách, aby bola dosiahnutá čo najvyššia stredná hodnota výhry za dané rozdelenie pozícii.

V prípade že je vyplácaná iba prvá pozícia, zase môžeme obe hodnoty považovať za priamo úmerné. Na prepočítanie počtu žetónov na $equity sa používa model ICM (Independent Chip Model) ktorý bude vysvetlený neskôr.

3.2. Neskorá fáza turnaja

Hra v turnaji je rozdelená do dvoch fáz:

-**Skorá fáza** kde sú blindy relatívne malé vzhľadom k veľkosti stackov.

-**Neskorá fáza** kde sú blindy relatívne veľké vzhľadom k veľkosti stackov.

Neskorá fáza hry je vzhľadom k počtu vyhraných čipov a konečného výsledku turnaju najdôležitejšia.

Keď blindy dosiahnu určitú veľkosť stráca význam zvyšovať stávku o menšiu čiastku pred flopom [1].

Deje sa tak z nasledujúceho dôvodu: Keď navýšime pot o menšiu čiastku, je už veľkosť potu taká veľká, že prípadný all-in súpera sme nútený dorovnať s takmer každou kombináciou. Napríklad kombinácia 23o (2, 3 mimo farby - jedna z najslabších) má proti AKs (eso s kráľom vo spoločnej farbe - jedna z najsilnejších) pravdepodobnosť výhry 1:2. V prípade, že navyšujeme s rozumne silnými kartami, je pravdepodobnosť ešte vyššia a čiastka ktorú musíme doplatiť oproti veľkosti celkového potu, robí call výhodnejším ako zloženie kariet. Súperovi však nechávame

dve strategické možnosti all-in alebo call.

Získa tak možnosť dorovnať, kým my nemáme možnosť karty zložiť. Z teórie hier vieme, že keď súper získa novú strategickú možnosť, jeho zisk z hry sa môže pri správnej hre len zvýšiť, čo znamená zníženie nášho zisku, keďže ide o hru s nulovým súčtom.

Optimálna stratégia sa potom redukuje na dve možnosti (vsadiť všetky čipy - push, alebo zložiť karty - fold). Deje sa tak pri veľkosti stackov okolo 10bb (big blinds), v prípade že je v hre Ante skôr [1]. Ďalej je túto stratégiu v niektorých prípadoch vhodné využiť aj pri väčších stackoch:

- Pri hre z malého blindu po flope, bude mať protivník na veľkom blinde výhodu v tom, že má lepšiu pozíciu (hrá až po nás a tým získava naviac informáciu o našej akcii). Túto výhodu môžeme eliminovať vsadením všetkých čipov pred flopom.
- Proti silným hráčom takisto môžme eliminovať výhodu ich silnej hry po flope tak, že vsadíme všetky čipy už pred flopom.
- Proti slabým hráčom, slabí hráči nie sú väčšinou schopní dorovnávať správne kombinácie po našom all-ine. Často skladajú príliš veľa kariet a tým vytvárajú situácie s veľkou ziskovosťou.

Zostáva už len určiť s ktorými kartami zahrať all-in a ktoré zložiť. To je možné našťastie pomerne presne spočítať pomocou počítača v prípade, že poznáme stratégiu súperov. Pomocou konceptu Nashovho equilibira (bližšie definovaného v časti 8.2) je možné hrať silnú hru aj bez znalosti stratégií súperov.

4. Základné postupy použité pri tvorbe umelej inteligencie

V tejto kapitole uvedieme dva postupy, ktoré sme sa rozhodli použiť pri tvorbe umelej inteligencie.

4.1. Aproximácia Nashovho equilibira

Silnou stánkou umelej inteligencie je fáza hry s vysokými blindami. Táto sitácia v turnajoch z rýchlym rastom blindov (tzv. turbo), nastáva pomerne rýchlo po začiatku turnaja. V niektorých typoch turnajov (tzv. superturbo), s veľkými blindami a malými stávkami, nastáva prakticky ihneď po začiatku hry.

Hra v tejto fáze je riešená pomocou aproximácie Nashovho equilibria. Pri hre 2 hráčov s nízkymi stackmi to zaručuje silnú stratégiu, ktorú je možné exploatovať len o minimálnu čiastku (v závislosti na kvalite aproximácie) a dosahuje dobré výsledky bez ohľadu na hru protihráča.

Pri hre viacerých hráčov síce takéto garancie neexistujú, ale takto vyprodukované stratégie majú stále zaujímavé vlastnosti. Platí, že v prípade keď stratégie všetkých hráčov tvoria Nashovo equilibrium, nemôže si žiaden hráč polepšiť odchýlením sa od svojej stratégie. V praxi však zväčša odchýlka zaznamená stratu. Pretože v prípade pokru sa jedná o hru s nulovým súčtom, keď sa jeden hráč od equilibria odchýli, ostatní hráči v priemere zaznamenajú zisk. Preto je v v prípade hry, kde sa vyskytujú silní hráči a nie je známa možnosť ako ich exploatovať, rozumné hrať stratégiu ktorá je aproximáciou Nashovho equilibria. To je práve často prípad pokru, pretože kvôli vysokej variácii, neúplnej informácii a veľkému množstvu rôznych herných situácií je mimoriadne ťažké odhadnúť súperovu stratégiu a je na to potrebné veľké množstvo dát o tom ako sa súper v minulosti správal. To robí samotné exploatovanie stratégie obzvlášť obtiažne. Samotný postup hľadania aproximácie equilibria a jeho implementácia budú opísané ďalej v tejto práci.

4.2. Expertný systém

Problematická však stále zostáva hra s veľkým počtom blindov, vzhľadom k stacku hráčov. Vytvoriť umelú iteligenciu, ktorá by mohla súperiť s ľudskými expertmi, je pre vriantu pokeru no-limit stále otvorený problém dokonca aj pre hru 2 hráčov [7]. Pre hru troch a viacerých hráčov je však tento problém rádovo ťažší. Deje sa tak kvôli stromu hry narastajúcemu exponenciálne s počtom hráčov.

Pre tento typ hry bol preto vytvorený v predloženej práci expertný systém, ktorý na

základe viacerých strategických aspektov aktuálnej hernej situácie určí akciu hráča.

Báza znalostí na základe ktorej sa systém rozhoduje je v prehľadnej forme uložený v podobe xml súborov. Je preto jednoduché prispôsobiť chovanie umelej inteligencie a vytvárať tak nových hráčov, ako aj postupne chovanie systému vylepšovať. Podrobnejšie bude tento systém opísaný neskôr.

5. Model výpočtu očakávanej výhry na základe počtu žetónov

V nasledujúcej kapitole si predstavíme model, ktorý prevádza počet žetónov v hre na očakávanú výhru. Tento model je pre tvorbu umelej inteligencie kľúčový.

5.1. Predstavenie modelu výpočtu očakávanej výhry

Aby sme mohli uvažovať o čo najlepšej hre, musíme si ujasniť čo chceme v hre dosiahnuť. Ako prvá možnosť nás možno napadne pokúsiť sa v strednej hodnote získať čo najviac žetónov. To však nie je väčšinou pri turnajoch optimálne riešenie (s výnimkou turnajov kde víťaz berie celú odmenu).

5.1.1. Príklad hry

Uvažujme hru vo formáte DON, kde je vyplácaných rovnakým dielom len prvých 5 miest a nasledujúci príklad: V hre ostáva 6 rovnako silných hráčov a každý hráč má rovnaký počet žetónov. Všetci hráči bú mať teda pravdepodobnosť výhry takmer rovnakú a to 5/6 (v zjednodušenom prípade, keď zanedbáme pozíciu hráčov pri stole a vplyv povinných sádzok).

Predpokladajme teraz, že dvaja hráči sa ocitnú v all-in situácii, keď obaja vsadili všetky svoje žetóny. Keď jeden z nich vyhrá ostane v hre len 5 hráčov ktorí si medzi seba rozdelia výhry. Víťaznému hráčovi stúpne počet žetónov na dvojnásobok, pravdepodobnosť výhry však bude mať teraz 1. To je však zlepšenie len o 1/6 oproti predchádzajúcej pravdepodobnosti, o rovnakú hodnotu sa teda zdvihne aj očakávaná finančná výhra.

Pre správne výpočty je potrebný model pomocou ktorého môžeme na základe počtu žetónov jednotlivých hráčov a výplatnej štruktúry turnaja vypočítať finančné očakávanie hráčov.

Takýto model predstavuje ICM [1, 4, 5] (Independent Chip Model). Jedná sa o model široko vyžívaný pokrovou komunitou, a softwarom (ako napr. [4]). Jeho vlastnosti boli overené na modelových situáciách a ukázalo sa, že pri aproximácii očakávanej výhry dosahuje dobré výsledky [5].

5.2. Fungovanie modelu

Najprv potrebujeme odhadnúť rozdelenie pravdepodobnosti, že hráč skončí na danom mieste medzi jednotlivé výplatné miesta. Po tom je už jednoduché vypočítať

očakávanú finančnú výhru – tá je rovná sume pravdepodobností umiestnenia vynásobenej odmenou za dané miesto cez všetky ohodnotené miesta čiže:

$\sum (\textit{pravdepodobnosť umiestenia n.} * \textit{odmena za n. miesto})$ cez všetky miesta.

5.2.1. Výpočet pravdepodobnosti umiestnenia hráča na danom mieste

Pri výpočtoch vychádzame z dvoch predpokladov:

1. Pravdepodobnosť výhry hráča *p* je počet jeho žetónov delený počtom všetkých žetónov v hre

2. Pravdepodobnosť že hráč *p* skočí na *n*-tom mieste, za predpokladu že hráči p_1 ... p_{n-1} skončili na miestach *1* ... *n-1* (v ľubovoľnom poradí), je rovný pomeru veľkosti stacku hráča *p*, k počtu všetkých žetónov v hre, bez žetónov hráčov, ktorí skončili pred ním. (Predpoklad je že zvyšní hráči za použitia svojich žetónov bojujú o *n*-té miesto) čiže:

$$\frac{\text{stack (p)}}{\sum_{j=1}^{\text{počet hráčov}} \text{stack}(p_j) - \sum_{i=1}^{n-1} \text{stack}(p_i)}$$

Kde je v p_j zahrnutý každý hrajúci hráč. Aby sme konečne dostali pravdepodobnosť že hráč skončí na *n*-tom mieste stačí vynásobiť predchádzajúcu pravdepodobnosť pravdepodobnosťou, že ostatný hráči sa umiestnenia na predpokladaných miestach a spraviť súčet všetkých takýchto pravdepodobností.

Konečný vzorec je pre výpočet očakávanej výhru hráča i pre n výplatných miest je:

$$U_i = P_1 * \frac{s_i}{S} + P_2 * \sum_{j \neq i} \left[\frac{s_j}{S} \times \frac{s_i}{S - s_j} \right] \ldots$$

$$+ P_n * \sum_{j_1 \neq i} \sum_{j_2 \neq i,\, j_2 \neq j_1} \ldots \sum_{j_{n-1} \neq i,\, k < n-1 \Rightarrow j_{n-1} \neq j_k} \left[\prod_k \frac{s_{j_k}}{S - \sum_{l<k} s_{j_l}} \times \frac{s_i}{S - \sum_k s_k} \right]$$

Kde *S* značí súčet všetkých čipov v turnaji, P_j je výhra za miesto *j*, a s_j je stack hráča *j*.

Časová zložitosť výpočtu je exponenciálne veľká vzhľadom k počtu vyplácaných miest. To však nie je veľký problém pretože vo formáte SNG je týchto miest spravidla maximálne 5.

5.3. Implementácia modelu ICM

Ukážeme si teraz bližšie samotnú implementáciu modelu.

5.3.1. Priamočiara implementácia ICM

Najjednoduchšia implementácia je založená na rekurentnom volaní funkcie, ktorá prevedie výpočet pravdepodobnosti pre každého hráča, že skončil na n-tom mieste. Pre tento prípad sa potom zavolá rekurzívne a spočíta pravdepodobnosť pre zvyšných hráčov, že sa umiestnili na nieste $n+1$. Rekurzia sa začne pre prvé miesto a zastaví sa pri poslednom vyplácanom mieste.

Príklad implementácie v jave prevzatý zo stránky [4]. Kód bol pre lepšiu názornosť dodatočne okomentovaný :

```
//Funkcia ktorá inicializuje premenné a zavolá
//rekurzívnu funkciu
//zodpovednú za samotný výpočet, vstupné parametre:
//double [] payouts – pole v ktorom sú ceny za jednotlivé
//vyplácané miesta
//double [] stacks- pole v ktorom sú stacky( počty
//žetónov) jednotlivých hráčov
//int player – index hráča ktorého icm nás zaujíma

public static double getEquity ( double [] payouts, double []
stacks, int player )
 {
 double total= 0 ; počet všetkých žetónov v hre
 for ( int i= 0 ;i<stacks.length;i++ )

//Vypočítam súčet všetkých žetónov ako súčet stackov

 total+=stacks [ i ] ;

//vrátim výstup rekurzívnej funkcie

 return getEquity ( payouts,stacks.clone () totalplayer 0
        );
 }

//Rekurzívna funkcia prevadzajúca výpočet, v porovnaní z
//predchádzajúcou funkciou pribudli na vstupe 2
//parametre:
//double total – počet čipov ktoré zostávajú v hre bez
//čipov hráčov ktorý sa umiestnili na nižších miestach
//ako je aktuálne počítané
//int depth- aktuálne počítane miesto

 Private static double getEquity ( double [] payouts double
[] stacks, double total, int player, int depth )
  {
```

```
//pravdepodobnosť že sa hráč umiestni na aktuálnom
//mieste je rovná podielu jeho stacku a všetkých čipov v
//hre, equitu eq pre hráča dostaneme tak že túto
//pravdepodobnosť vynásobíme výhrou  za dané miesto.

  double  eq = stacks [ player ]  / total * payouts [
  depth ] ;

//ak sa nejedná o posledné výplatné miesto, pokračujeme
//ďalej vo výpočte pre ďalšie výplatné miesta
//rekurzívnym volaním funkcie

  if ( depth +  1  < payouts.length )
      for ( int i= 0 ;i<stacks.length;i++ )

//pre každého hráča okrem hráčov ktorý sa už umiestnili
//budeme počítať  equitu hráča player ak sa daný hráč
//umiestni na aktuálnom mieste rekurzívnym volaním
//funkcie z náležite upravenými parametrami a výsledok
//sa vynásobí pravdepodobnosťou tohto umiestnenia čo je
//opäť podiel počtu čipov hráča k všetkým čipom v hre,
//potom túto hodnotu pripočítame k výslednej equite

        if  ( i != player && stacks [ i ]  >  0.0 )
        {
            double  c = stacks [ i ] ;
            stacks [ i ]  =  0.0 ;
            eq += getEquity ( payouts, stacks, total – c,
            player, depth +  1 )  * c / total;
            stacks [ i ]  = c;
         }
//Nakoniec vrátime výslednú equitu
     return eq;
  }
```

Ako je vidieť je implementácia veľmi jednoduchá, vo väčšine prípadov je aj dostatočne rýchla.

V našom programe je implementácia podobná, na rozdiel od ukážkového príkladu, kvôli potrebám ďalších výpočtov počítam naraz equitu pre viacero zúčastnených hráčov. O implementáciu sa stará trieda ICM pomocou statických metód.

Ďalší fakt vyplývajúci z ukážkovej implementácie je, že časová náročnosť výpočtu rastie exponenciálne s počtom vyplácaných miest. To sa prejavilo pri použití programu na výpočty pre hry typu DON z piatimi vyplácanými pozíciami. Výpočty prebiehali príliš pomaly, aby bolo možné použiť aplikáciu v reálnom čase. Problém

sa mi podarilo vyriešiť pomocou dynamického programovania.

5.3.2. Implementácia ICM využívajúca dynamické programovanie

Pozorovanie: Keď máme hráčov h_1 až h_n, ktorí sa umiestnia na prvých n miestach je pravdepodobnosť, že sa hráč h_{n+1} umiestni na mieste $n+1$ rovná podielu jeho čipov a všetkým čipov bez čipov hráčov h_1 až h_n. Je teda nezávislá na konkrétnom umiestnení jednotlivých hráčov.

Pri výpočte to využijeme tak, že zmenšíme vetvenie výpočtu. V predchádzajúcom prípade existovala samostatná vetva pre každú permutáciu hráčov na n-miestach. Teraz je dostatočná jedna vetva pre každú množinu n-hráčov. Možné zrýchlenie je teda až $n!$.

Postup výpočtu pre spracovanie rozdelenia ceny za miesto n medzi hráčov je nasledovný. Vygenerujem všetky množiny hráčov veľkosti $n-1$. Pre každú takúto množinu mám z predchádzajúceho kroku uloženú jej pravdepodobnosť.

Pre každého hráča ktorý sa v množine nenachádzal, vypočítam pravdepodobnosť že sa umiestni na n-tom mieste a túto pravdepodobnosť vynásobím pravdepodobnosťou množiny. Výslednú hodnotu vynásobenú cenou za miesto n, prirátam k equite daného hráča. Takisto príslušne zvýšim pravdepodobnosť množiny veľkosti n obsahujúcej pôvodnú množinu a aktuálneho hráča.

V mojej implementácii som algoritmus optimalizoval, aby použil čo najmenší počet aritmetických a logických operácií. Výsledná implementácia sa nachádza v triede FastICM.

V praktickom nasadení pre 10 hráčov a 5 výplatných miest, bola oproti predchádzajúcej implementácii približne 30-krát rýchlejšia, čo umožnilo prevádzať výpočty v reálnom čase.

6. Aproximácia Nashovho equilibria

Aproximácia Nashovho equilibria je najsilnejšia stránka vytvorenej umelej inteligencie. Nasledujúce kapitoly venujeme práve jej.

6.1. Požiadavky na výpočet aproximácie

Ako už bolo spomínané táto metóda sa používa v záverečnej fáze hry. Samotná equlibristická stratégia závisí na viacerých parametroch, konkrétne na stackoch hráčov, ich pozícii pri stole, veľkosti blindov a ante a výplatnej štruktúre hry. Pri počte hráčov n a súčte čipov v hre C dostávame $\binom{C+n-1}{n-1}$ možností rozdelenia čipov, ktoré sú zo strategického hľadiska rozdielne (pri hre záleží aj na pozícii hráčov, vzhľadom na polohu blindov, preto nemôžeme využiť izomorfizmy). Typický súčet čipov v jednostolových turnajoch sa pohybuje v rozpätí 3000 – 20000 a počet hráčov z ktorými pri našej aproximácii budeme musieť rátať je v rozmedzí 3-10. Preto nie je z časových a priestorových dôvodov možné equilibriá vypočítať dopredu a uložiť. Program musí byť z tohto dôvodu schopný aproximovať equilibrium v reálnom čase.

Teraz sa otvára otázka čo znamená „počítať v reálnom čase“, teda ako rýchlo musí výpočet prebehnúť. V praxi hráči na internete hrajú väčšinou viac stolov naraz, niekedy dokonca viac ako 50, preto sú zvyknutý hrať veľmi rýchlo a rozhodovať sa v niekoľkých sekundách. Aby sme uspokojili požiadavky týchto hráčov je potrebné aby umelá inteligencia pracovala naozaj rýchlo, výpočet by mal prebiehať maximálne v rámci niekoľkých sekúnd. Preto bola nad hrou vytvorená abstrakcia ktorá bude opísaná neskôr.

6.2. Strom Hry

Aby sme mohli určiť optimálne stratégie, potrebujeme vedieť určiť výsledky pre jednotlivých hráčov v prípade, keď sa každý bude držať svojej pevne zvolenej stratégie. Na výpočet využívame strom hry.

6.2.1. Priebeh hry

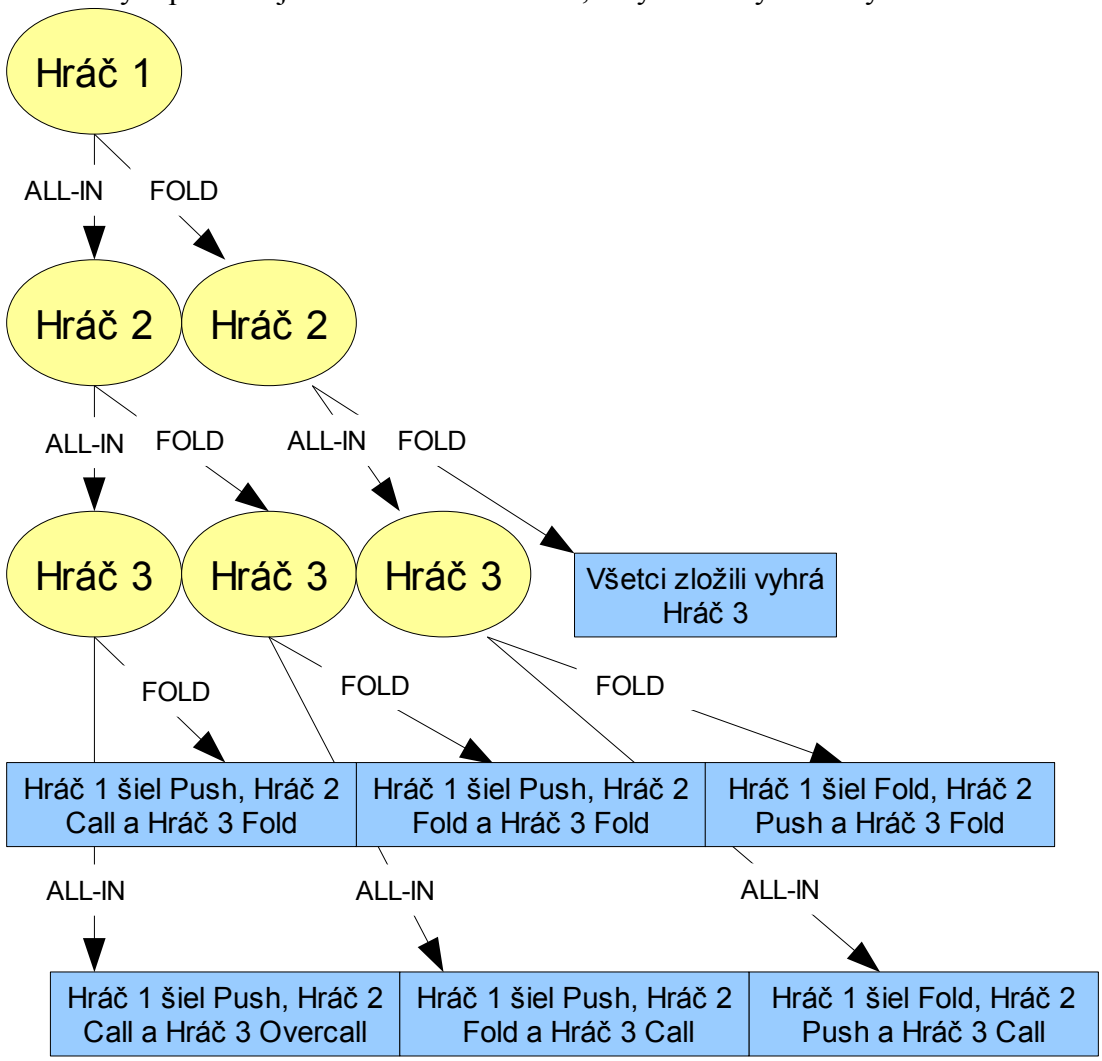
Ako sme už spomínali, obmedzíme akcie hráčov na all-in, alebo fold. Hráči hrajú v pevne danom poradí, posledný na rade je hráč na veľkom blinde. Na začiatku má prvý hráč možnosť ísť all-in (tzv. Push) alebo zložiť, v prípade že zložil dostávajú

rovnakú možnosť hráči ktorý sú na rade za ním. V prípade, že všetci hráči zložia svoje karty, vyhráva hráč na veľkom blinde.

Keď niektorý hráč Push-uje, opäť majú hráči po ňom možnosť vsadiť all-in (v tomto prípade tzv. Call), alebo zložiť. V prípade, že všetci ostatní hráči zložili, vyhráva hráč ktorý Push-oval.

Po hráčovi ktorý Call-uje má v našej abstrakcii hry možnosť vsadiť all-in ešte jeden zo zostávajúcich hráčov (v prípade že necall-oval hráč na veľkom blinde). V reálnej hre je možné, aby šli all-in naraz všetci hráči. Situácia keď sa ocitnú v all-ine viacerí, ako traja hráči, je v reálnej hre mimoriadne zriedkavá a kvôli obmedzenému dostupnému výpočtovému výkonu a pamäti ju nie je možné zahrnúť do našej abstrakcie.

Na presný výsledok potrebujem poznať všetky možnosti, ako môže hra dopadnúť. Pre názornosť zobrazíme tieto možnosti v strome hry pre troch hráčov na obrázku č.10. Uzly reprezentujú rozhodovanie hráčov, listy koncový stav hry.



Obrázok 10: Strom hry troch hráčov

6.2.2. Implementácia stromu hry

Tento strom je reprezentovaný v programe ako viacrozmerný vektor koncových stavov (rozhodovacie uzly nie sú pre výpočet zaujímavé, všetky potrebné informácie sú zahrnuté v týchto stavoch).

Každý z nich je zachytený pomocou inštancie triedy StrategyLeaf v triede EquilibristicTreeBase. Jednotlivé súradnice vo vektore označujú pozíciu push-ujúceho, call-ujúceho a overcall-ujúceho hráča.

Každý zo stavov je schopný vypočítať pravdepodobnosť, že nastane a equitu jednotlivých hráčov v prípade že nastane, za predpokladu, že sú známe stratégie všetkých hráčov. Takže vieme zistiť výhodnosť ľubovoľnej stratégie vybraného hráča, pomocou súčtu equity, cez všetky koncové stavy, váženého ich pravdepodobnosťou za predpokladu, že stratégie ostatných hráčov budeme tiež poznať. Pri skutočnom výpočte beriem z dôvodu optimalizácie v úvahu len stavy, ktoré môže hráč svojim rozhodnutím naozaj ovplyvniť.

7. Abstrakcia Hry

Ako sme už spomenuli, neaproximujeme equilibrium v celej hre ale pre zrýchlenie výpočtov je použitá abstrakcia hry.

7.1. Reálne Nashovo equilibrium v našej hre

Až na izomorfizmus existuje 169 štartovných kombinácií. Stratégia hráča spočíva v tom, že pre každú kombináciu akcií hráčov pred ním má ku každej kartovej kombinácii priradenú pravdepodobnosť, s akou ju bude hrať a s akou ju zloží (tzv. Range). Aproximovať equilibrium v takejto hre by však bolo podľa odhadov príliš pomalé, preto bola vytvorená abstrakcia hry.

7.2. Abstrakcia hry

Naše zjednodušenie spočíta v tom že hráč sa pri každej kartovej kombinácii rozhodne len medzi dvoma možnosťami - či ju bude hrať so 100% pravdepodobnosťou, alebo ju z rovnakou pravdepodobnosťou zloží. Tento predpoklad nie je až taký obmedzujúci, ako by sa mohlo zdať. Pri zvolenom algoritme „fictitious play“ sú v priebehu výpočtu používané iba čisté stratégie. Ďalším dôvodom prečo toho obmedzenie nie je zásadné, je empirické zistenie, že randomizácia je v skutočnom equilibriu používaná veľmi zriedkavo [5].

Ďalším zjednodušením je, že všetky kombinácie sú usporiadané podľa ich predpokladanej sily. Hráč musí do svojej range pridávať karty len na základe tohto poradia. To znamená že nie je možné, aby vsadil s kombináciou s poradovým číslom 4 a nevsadil s kombináciou 2. Silou sa rozumie pravdepodobnosť výhry proti súperovej range. Tá však mení v závislosti na na range, preto je toto usporiadanie kariet iba heuristika. Toto usporiadanie bolo prevzaté z práce [2].

Počet kariet ktoré hráč hrá jednoznačne určuje range. Tú potom v programe môžeme reprezentovať pomocou jediného čísla. Napr. číslo 2 reprezentuje range skladajúcu sa z 2 najsilnejších kombinácií: AA a KK. Celkovo máme teda 170 možných range, ktoré určujú počet kombinácií, ktoré bude hráč hrať (range 0 znamená že hráč všetky karty skladá). Každý hráč má viacero range pre rozličné situácie. Jedna určuje s akými kartami bude push-ovať, ak pred ním všetci zložili (pokiaľ sa nejedná o hráča na veľkom blinde). Pre každého hráča ktorý pred ním mohol push-ovať má stratégiu s akou range ho bude call-ovať. A nakoniec pre každú dvojicu hráčov s ktorých mohol jeden push-ovať a druhý call-ovať, má stratégiu

určujúcu s čím ich bude overcall-ovať. Uloženie stratégií je implementované pomocou viacrozmerného poľa celých čísel.

8. Alogritmy použité na výpočet aproximácie equilibria

V tejto časti práce sa bližšie pozrieme na potupy ktoré sme použili na výpočet aproximácie Nashovho equilibria.

8.1. Asymptotická zložitosť výpočtu Nashovho equilibria

Výpočet Nashovho equilibria pre hry troch a viac hráčov je PPAD-úplný problém [6]. Nepredpokladá sa preto že existuje polynomiálny algoritmus ktorý by dokázal tento problém riešiť. To však nie je príklad hry 2 hráčov s konštantným súčtom. Preto sú situácie keď v turnaji ostanú iba dvaja hráči (tzv. heads up, HU) riešime osobite pomocou lineárneho programu počítaného v reálnom čase. Nie je známy ani špeciálny algoritmus ktorý by obišiel PPAD-úplnosť využitím špecifických vlastností určitej triedy hry, tak aby do tejto triedy patril poker obecne, alebo naša abstrakcia.

8.2. Epsilon equilibrium

Nashovo equilibrium je v teórii hier stav, keď si žiadny z hráčov nemôže jednostrannou zmenou svojej stratégie vylepšiť svoju situáciu. Pretože ako sme spomínali skôr, je výpočet takéhoto equlibira v praxi veľmi obtiažny, zavádza sa pojem epsilon-Nashovo equilibrium. Epsilon-equilibrium je stratégia ktorá približne spĺňa požiadavky Nashovho equilibria.

8.3. Definícia ε-equilibria

Pri danej hre a nezápornom parametri ε, je stratégia ε-equilibrium, ak žiadny hráč nemôže získať viac, ako ε očakávanej výhry jednostrannou zmenou svojej stratégie. Každé Nashovo equilibrium je equivalentné ε-equilibriu pre ε rovné 0.

8.4. Algoritmy používané na aproximáciu Nashovho equilibria

Aproximačný algoritmus, ktorý je schopný nájsť v polynomiálnom čase ε-equilibrium pre malú konštantnú hodnotu ε, je stále otvorený problém. Preto uvedené algoritmy nebudú spravidla poskytovať teoretické garancie konvergencie k equilibriu. V pokrovej doméne sú na aproximáciu equilibria používané najmä dva postupy. Fictitious play [8], a regret minimalization [9]. Algoritmus fictitious v pokrovej abstrakcii podobnej našej empiricky dosahoval konvergenciu [5]. Na implementáciu do AI bol vybratý algoritmus implementujúci fictitious play, pretože

počas samotného výpočtu na rozdiel od algoritmov implementujúcich regret minimalization nepotrebuje rátať zo zmiešanými stratégiami. Tento fakt je možné využiť na efektívny výpočet. Ďalej sa budeme teda zaoberať len týmto algoritmom z uvedenej dvojice.

9. Fictitious Play

Jedným zo široko využívaných modelov učenia v hrách je proces fictitious play a jeho varianty. Vrámci tohto modelu sa agenti správajú, ako keby čelili stacionárnemu, ale neznámemu rozdeleniu stratégií oponentov.

9.1. Definícia algoritmu

Aby sme definíciu zjednodušili, začneme z prípadom dvoch hráčov.

9.1.1. Definícia algoritmu pre Prípad dvoch hráčov

Algoritmus definujem pre hru 2 hráčov, s konečnými priestormi stratégií S^1, S^2 a výplatnými funkciami u^1, u^2. Model fictitious play predpokladá, že hráči vyberajú svoje akcie v každej perióde hry tak, aby maximalizovali svoju očakávanú výhru vzhľadom na predpoklady o distribúciu akcií oponentov v každej perióde hry. Tieto predpoklady majú špeciálnu formu:

Hráč i má počiatočnú váhovú funkciu $\kappa^i_0 : S^{-i} \to \Re_+$. Táto váha je potom aktualizovaná pridávaním 1 k váhe každej stratégie oponenta, vždy keď je táto akcia zahratá čiže:

$$\kappa^i_t(s^{-i}) = \kappa^i_{t-1}(s^{-i}) + \begin{cases} 1 \, ak \, s^{-i}_{t-1} = s^{-i} \\ 0 \, ak \, s^{-i}_{t-1} \neq s^{-i} \end{cases}$$

Pravdepodobnosť ktorú hráč i priradí tomu že hráč *-i* hrá stratégiu s^{-i} v čase t je daná vzorcom:

$$\gamma^i_t(s^{-i}) = \frac{\kappa^i_t(s^{-i})}{\sum_{\tilde{s}^{-i} \in S^{-i}} \kappa^i_t(\tilde{s}^{-i})}$$

Fictitious Play je potom definovaná ako akékoľvek pravidlo $\rho^i_t(\gamma^i_t)$, ktoré priradí $\rho^i_t(\gamma^i_t) \in BR^i(\gamma^i_t)$. Kde $BR^i(\gamma^i_t)$ je množina stratégií, ktoré sú najlepšou odpoveďou na γ^i_t [8]. V tradičnom prístupe, keď hráči môžu vybrať medzi viacerými najlepšími odpoveďami, vyberú čistú stratégiu.

9.1.2. Prípad viacerých hráčov

Pre prípad troch alebo viacerých hráčov nastáva zaujímavá otázka. Ukážeme ju na príklade troch hráčov. Pre hráča 1 sa naskytá otázka, či má uvažovať o stratégiách protihráčov ako o nezávislých zmiešaných stratégiách, čiže pre príklad dvoch protihráčov bude predpokladať že $\gamma^1(s^2, s^3) = \gamma^1(s^1)\gamma^1(s^3)$, alebo má zahrnúť do

svojich predpokladov aj korelované stratégie, teba bude jeho predpoklad o súperových stratégiách tvoriť pravdepodobnostné rozdelenie na priestore $\Delta(S^2 \times S^3)$?

Pozrime sa na časovú zložitosť jednotlivých možností, vzhľadom na počet stratégií. Predpokladajme hru 3 hráčov a nech $|S^1|=|S^2|=|S^3|$. Pozrime sa teraz, ako rýchlo dokáže hráč 1 vypočítať svoju najlepšiu odpoveď. V prvom prípade musí na základe stratégií použitých v minulom kole aktualizovať predpoklady o hráčoch 2 a 3 a na základe týchto predpokladov potom nanovo spočítať najlepšiu odpoveď za využitia všetkých možných kombinácií troch stratégií. Teda zložitosť výpočtu bude $\Theta\left(\left|S^1\right|^3\right)$. Teraz sa pozrime na druhý prípad. Stačí pre každú stratégiu hráča 1 vypočítať očakávanú výhru proti dvojici stratégii z minulého kola. Za predpokladu, že máme uložený aj aktuálny súčet týchto hodnôt, dokážeme najlepšiu odpoveď hráča 1 spočítať v čase $\Theta\left(\left|S^1\right|\right)$, čo je výrazné asymptotické zrýchlenie. Naskytá sa teda otázka, či nám prvý variant algoritmu dokáže že poskytnúť lepšie teoretické vlastnosti a lepšiu konvergenciu, ktorou by vyvážil pomalý výpočet. Odpoveď je však záporná [8]. Preto sme zvolili druhý variant algoritmu.

9.2. Niektoré zaujímavé vlastnosti algoritmu

Algoritmus fictitious play neposkytuje vo všeobecnosti teoretickú garanciu konvergencie k equilibriu. Je však známe, že v niektorých triedach hier [8] algoritmus konverguje. Medzi týmito prípadmi však zatiaľ nie je taká, ktorá by pokrývala našu pokrovú abstrakciu. Algoritmus má však niektoré zaujímavé vlastnosti, ktoré môžu byť využité k analýze praktických výsledkov jeho nasadenia.

Vo fictitious play sú silné Nashove equilibriá absorbujúce stavy. Platí:

(a) Ak je statégia *s* silné Nashovo euilibrium a je hraná v kroku *t, s* bude tiež hraná v každom ďalšom kroku hry.

(b) Akákoľvek čistá stratégia, ktorá je stabilným stavom fictitious play, musí byť Nashovo equilibrium [8].

Ak vo fictitious play konverguje empirické rozdelenie stratégií, ktoré každý hráč hrá, stratégia zodpovedajúca tomuto rozdeleniu je Nashovo equilibrium [8].

9.3. Implementácia algoritmu

Výhodu algoritmu je jednoduchá a priamočiara implementácia. Algoritmus bol implementovaný triedou FictitiousPlayTree a jej metódou *ComputeEquilibrium*. Základnými dátovými štruktúrami použitými v algoritme boli vektory obsahujúce aktuálne stratégie hráčov a vektory obsahujúce akumulovanú equitu jednotlivých stratégií za všetky iterácie algoritmu, ktoré je potom možné použiť na výpočet najlepšej odpovede v danej iterácii.

9.3.1. Ukážka Implementácie metódy *ComputeEquilibrium*

Pozrime sa teraz bližšie na kód tejto metódy:

```
void FictitiousPlayTree::ComputeEquilibrium()
{
    //zinicializujem vektor obsahujúci akmulované equity jenotlivých
    stratégií
    InizializeEuities();
    //nastavím stratégie hráčov pre krok 0 na východziu hodnotu
    SetDefaultStrategies();
    //nastavím požadovaný počet iterácií výpočtu
    const int intrationsCount = 500;
    //vykonám jednotlivé iterácie
    for (int time =1; time <= intrationsCount; ++time)
        SIngleIteration(time);

}
```

Ako vidno, metóda výpočtu aproximácie equilibria je mimoriadne jednoduchá. Pozrime sa teda bližšie na jednotlivé iterácie:

```
void FictitiousPlayTree::SIngleIteration(int time)
{
    //prepočítam equitu všetkých stratégií na základe stratégií z
    predchádzajúceho kola
    UpdateEuities(time);
    //pre každého hráča vyberiem stratégiu z najlepšou equitou
    SetBestResponse();
}
```

Kód jednej iterácie je podobne priamočiary. Zaujímavým sa stáva výpočet equity jednotlivých stratégií. Deje sa tam na základe vzorca:

$$1/t * (equita\ proti\ stratégii\ v\ poslednej\ iterácii) + (t-1)/t * (doterajšia\ equita)$$

Equita vypočítaná týmto spôsobom je ekvivalentná z equitou stratégií proti empirickej distribúcii hry protihráčov. Jej porovnaním cez všetky stratégie teda môžeme rýchlo získať najlepšiu odpoveď hráča. Výpočet equity proti pevnej stratégii bude vysvetlený neskôr.

9.4. Experimentálne výsledky

Po naimplementovaní algoritmu bolo jeho chovanie experimentálne otestované a vyhodnotené.

9.4.1. Očakávané chovanie algoritmu

Ako sme už spomínali, pri našej hre čakáme zriedkavé využitie zmiešaných stratégií. Preto v prípade, že algoritmus konverguje k Nashovmu equilibriu, mal by skončiť v pohlcujúcom stave (v prípade že je v hre čisté equilibrium), alebo skákať v jednotlivých krokoch cyklu medzi malým počtom stratégií. To vyplýva z špecifických vlastností našej hry. Ako už bolo spomenuté, randomizácia je v našej abstrakcii zriedkavá. Ďalšia zaujímavá vlasnosť je predpoklad dobrého usporiadania kariet v abstrakcii. Napríklad range {AA, KK} má proti ľubovolnej range súperov väčšiu pravdepodobnosť výhry ako range {AA, KK, QQ}. Preto by sa v empirickom rozdelení pravdepodobností jednotlivých stratégií zastupujúcom euilibrium nemal vyskytovať cyklus oscilujúci medzi range {AA} a {AA, KK, QQ}. Je tomu tak z dôvodu, že v prípade keď prevedieme rozdelenie pravdepodobností hry {AA} a {AA, KK, QQ} na pravdepodobnosť hry jednotlivých kariet, dostávame slabšiu distribúciu pretože QQ je hrané rovnako často, ako KK čo je silnejšia kombinácia. Preto by mal namiesto toho mal vo výpočte vyskytovať cyklus využívajúci range {AA, KK} a niektorú zo susedných kombinácií kariet. Pri range ktoré zastupujú vysoký počet kariet sa môžu v priebehu výpočtu vyskytnúť aj o niečo väčšie skoky v rámci cyklu, spôsobené nejednoznačnosťou usporiadania, čakáme však že range ktoré budú súčasťou cyklu, zas budú veľmi blízko seba počtom kariet ktoré zastupujú.

9.4.2. Zistené výsledky

Pri pokusných výpočtoch sa spravidla po dostatočne veľkom počte krokov výpočet zastavil na pevne danej stratégii, alebo sa vyskytol cyklus zahŕňajúci dve stratégie. Cyklus zahŕňajúci tri, alebo viac stratégií, nebol pozorovaný a stratégie v cykloch sa spravidla líšili iba o pár kartových kombinácií. Z týchto výsledkov môžeme usudzovať, že algoritmus aproximoval equilibrium dobre, čo bol vzhľadom na prácu [5] očakávaný výsledok.

10. Heuristika založená na najlepšej odpovedi na aktuálnu stratégiu súpera

Výpočet pomocou fictitious play prebiehal pomerne rýchlo (v rádoch sekúnd, trvanie výpočtu závisí na počte iterácií, počte hráčov a použitom hardware). Pre zvýšenie užívateľského komfortu poskytnutím ešte rýchlejšej odozvy protivníkov boli však vyskúšané ešte dve heuristiky.

10.1. Princíp práce heuristiky

Naskytá sa teda otázka ako navrhnúť heuristiku ktorá by bola rýchlejšia ako fictitious paly a dosahovala podobné výsledky. Návrh berie ako základ algoritmus použitý vo fictitious play. Najpomalší krok iterácie fictious play je zisťovanie equity každej stratégie. Preto sa ako logická varianta zrýchlenia ponúka možnosť vypočítať iba equitu stratégií ktoré sú blízko odhadovanému equilibriu. Teraz sa však naskytá otázka ako odhadnúť equilibrium.

Na odhad bol použitý predpoklad, že stratégia sa v postupe výpočtu približuje k equilibiru, preto vždy rátame iba equitu stratégií, ktoré sa od aktuálnej stratégie líšia len o určitý maximálny počet kariet. Problém nastáva, keď sa zmení aktuálna stratégia a tým aj jej okolie. V tom prípade dostaneme stratégie v okolí aktuálnej stratégie, o ktorých nemáme informácie o ich equite v minulosti. Ako riešenie tohto problému, bolo zvolené namiesto najlepšej odpovede na empirickú distribúciu stratégií súperov zvoliť len najlepšiu odpoveď na aktuálnu stratégiu súperov. Tým sa vytvoril posledný problém.

Predstavme si ho na príklade 2 hráčov. Hráč 1 hrá v prvom kole stratégiu s_1^1 a hráč 2 hrá v prvom kole stratégiu s_1^2. V ďalšom kole hry bude teraz hráč 1 hrať stratégiu s_2^1, ktorá je najlepšia odpoveď na stratégiu s_1^2 a hráč 2 zas stratégiu s_2^2, ktorá je najlepšia odpoveď na s_1^1. V prípade že s_1^1 je najlepšia odpoveď na s_2^2 a s_1^2 na s_2^1 nastane cyklus a algoritmus nevydá ani približnú aproximáciu Nashovho equilibria.

Ak by sa jednalo o hru heads up, kde je súčet nulový, algoritmus fictitious play by dokonca teoreticky garantoval konvergenciu k equilibriu [8]. Aby sme sa takýmto cyklom vyhli a zároveň využili predpoklad, že algoritmus by sa mal postupne približovať k Nashovmu equilibriu, budeme uvažovať ako kandidátov na najlepšiu odpoveď stratégie z postupne zmenšujúceho sa okolia aktuálnej stratégie.

10.2. Implementácia algoritmu

Narozdiel od fictitious play v našej implementácii hráči menia svoje stratégie striedavo. O implementáciu sa stará trieda EquilibristicTreeV2 a jej metóda ComputeEquilibrium. Pozrime sa teraz bližšie na výpočet aproximácie equilibria:

```
void EquilibristicTreeV2::ComputeEquilibrium()
{
    //podľa počtu hráčov určíme okolie aktuálnej stratégie ktoré
    //budeme prehľadávať
    int playersCount = this->state->players.count();

     //konštanty určujúce veľkosť okolia
    int firstJumps[] = {25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 16};

    //zistíme veľkosť okolia pre aktuálny prípad
    int firstJump = firstJumps[playersCount];

    //stratégie meníme postupne podľa hráča ktorý pushuje,
    //prepočítame stratégie len pre aktuálny podstrom
    for(int pusher=pusherStrategies.size()-1;pusher>-1;--pusher)
    {
        //nainicializujeme range v aktuálnom podstrome
        InitSubtreeRanges(pusher);

        //zmenšujeme postupne zmenšujeme veľkosť okolia (a tým aj
        //maximálneho možného skoku )
        for(int maxJump = firstJump; maxJump > 0; maxJump--)
        {

            //funkcia TryToImprovePush nájde najlepšiu možnú
            //stratégiu v dovolenom okolí pre pushera
            //následne zavolá funkcie ktoré to isté zopakujú pre
            //callerov a  overcallerov.
            if(!TryToImprovePush(pusher, maxJump))
            {

                //Funkcia TryToImprovePush vráti false
                //v prípade že sa nikoho stratégia nezmení. Keďže sa
                //najlepšia odpoveď ráta iba na základe aktuálnych
                //stratégií súperov a uvažované okolie sa môže len
                //zmenšovať, nemôže sa stratégia
                //meniť ani ďalej vo výpočte a výpočet môžme ukončiť

                break;
            }
        }
    }
}
```

V priebehu výpočtu využívame fakt, že stratégia hráča pre push nezávisí na stratégiách hráčov ktorí pred ním zložili. V skutočnosti to, že zložili určitú range ovplyvní pravdepodobnosť výskytu kartových kombinácií (tzv. card removal effect).

Tento fakt však zanedbávame, jednak z dôvodu extrémnej výpočtovej zložitosti a neveľkého efektu, a v druhom rade preto, že to môže viesť k nepríjemným paradoxom. Card removal effect vzhľadom na range hráčov ktorí šli all-in, však počítame korektne.

Keď teda začnem výpočet od hráča na malom blinde (posledný hráč ktorý môže push-ovať) a postupne budeme prepočítavať ďalších hráčov až k hráčovi UTG (prvý hráč na rade) a odpovede protihráčov na ich stratégiu, výpočet prebehne správne.

10.3. Experimentálne výsledky

Na overenie výsledkov výpočtu sme použili stratégie získané pomocou algoritmu fictitious play. Náš algoritmus počíta mimoriadne rýchlo, a zároveň zachováva veľkú presnosť vzhľadom k algoritmu fictitious play. Od jeho výsledku sa výsledné range spravidla odlišovali len v minimálnom počte kartových kombinácií. Preto bola táto heuristika nakoniec zvolená ako hlavný spôsob aproximácie equilibia pre umelú inteligenciu hry.

11. Heuristika založená na predpoklade dobrého uspriadania kariet v stratégii

Predošlý algoritmus dosahoval veľmi dobré výsledky, bola však navrhnutá a otestovaná heuristyka, ktorá počíta ešte rýchlejšie.

11.1. Návrh Algoritmu

Základná vlastnosť equilibria je, že žiadny hráč nemôže vylepšiť svoj výsledok zmenou stratégie. Pri tejto aproximácii to využijeme ako fakt, že keď sa rozšíri range hráča ktorý psuhuje, range hráčov ktorý ho môžu dorovnať sa tiež len rozšíri alebo ostane rovnaká. Ako v predchádzajúcom algoritme budeme počítať abstrakciu pre podstromy hry dané hráčom ktorý môže push-ovať. Začneme pri hráčovi na malom blinde. Samotný algoritmus bude potom definovaný nasledovne:

Postupne, pokiaľ to bude pre neho výhodné, bude hráč ktorý može push-ovať rozširovať svoju range. Výpočet pre aktuálny podstrom sa končí v prípade, že sa mu jeho range neoplatí rozšíriť proti aktuálnej stratégii súperov. V opačnom prípade hráč svoju range rozšíri, a o to isté sa potom pokúsia v odpovedi na tento krok protihráči.

Celý algoritmus musí dodržiavať nasledujúci invariant. Pred tým, ako rozšíri svoju range push-ujúci hráč, nesmie byť žiadny z call-ujúcich hráčov schopný so ziskom rozšíriť svoju range o jednu kartu. To isté platí aj o call-ujúcicom hráčovi a overcall-ujúcom hráčovi.

Pri tejto definícii sa núka implementácia, pri ktorej po prispôsobení stratégií oponentov opäť prerátame, či sa hráčovi equita skutočne zväčšila. Iba v prípade, že tomu tak je, potom stratégiu zväčšíme. Táto možnosť je však nie len výpočtovo pomalšia, ale dokonca dosahuje oveľa horšie výsledky kvôli tendencii zasekávať sa na lokálnych maximách.

Najzaujímavejšou vlastnosťou algoritmu je, že pri každom kroku je pre každého hráča potrebné spočítať iba equitu dvoch stratégií. To robí algoritmus mimoriadne rýchlym. Problematickou vlastnosťou je však, že výkon algoritmu je oveľa viac závislý na kvalite usporiadania kariet podľa sily, ako predošlé algoritmy. Napríklad keď by sme dali v usporiadaní na niektoré z predných miest slabú kartovú kombináciu (napríklad 23o), range call-ujúcich hráčov by sa takmer určite zasekla na tejto kombinácii. To by taktiež zmenilo range push-ujúceho hráča a výsledný stav by bol ďaleko od equilibria.

11.2. Implementácia algoritmu

V programe má výpočet na starosti metóda ComputeEquilibrium triedy EquilibristicTree. Pozrime sa teda na jej kód:

```
void EquilibristicTree::ComputeEquilibrium()
{
    for(int i=pusherStrategies.size()-1;i>-1;--i)
        while(this->TryToImprovePush(i));
}
```

Ako vidno samotná metóda na výpočet equilibria jej mimoriadne jednoduchá - začne od posledného hráča ktorý môže push-ovať a postupne každému hráčovi ktorý môže push-ovať zväčšuje jeho range, dokiaľ je to pre neho výhodné.

Teraz sa pozrieme bližšie na metódu TryToImprovePush ktorá sa pre hráča push-er pokúsi zväčšiť jeho range, ak sa jej to nepodarí vráti fasle, inak vráti ture:

```
bool EquilibristicTree::TryToImprovePush(int pusher)
{
    //Ak je stratégia 169, hráč pushuje všetky svoje karty a range
    //nie je možné zväčšiť

    if(this->pushRange(pusher)==169)
        return false;
    //spočítame si aktuálnu equitu hráča
    double oldEquity = this->GetEquity(pusher);
    //zväčšíme stratégiu hráča o 1
    this->pusherStrategies[pusher]++;
    //znovu spočítame equitu  hráča zo zmenenou stratégiou
    double newEquity =this->GetEquity(pusher);

    //ak je nová equita menšia ako stará hráčovi sa neoplatí
    //zmeniť stratégiu, takže jej hodnotu vrátime do pôvodného
    //stavu a ako výstup vrátime false.
    if(newEquity<oldEquity)
    {
        pusherStrategies[pusher]--;
        return false;
    }

    //v prípade že sa zmena oplatí najprv skúsime zlepšiť stratégie
    //všetkých dorovnávajúcich hráčov volaní metódy
    //TryToImproveCall ktorá funguje analogicky ako metóda
    //TryToImprovePush (a rovnako neskôr volá metódu
    //TryImproveOverCall ktorá sa snaží zlepšiť stratégie
    //dorovnávajúcich hráčov ), stratégiu hráča nechá nastavenú na
    //zvýšenú hodnotu a na  výstup vráti true pretože bol pokus o
    //zmenu stratégie úspešný.

    for(int i=state.activePlayersCount-1;i>pusher;--i)
        while(this->TryToImproveCall(pusher,i));
    return true;
}
```

11.3. Experimentálne výsledky

Podľa očakávaní algoritmus bežal mimoriadne, rýchlo. Bol suverénne najrýchlejším z trojice testovaných algoritmov. Jeho presnosť však bola už značne horšia, čo vyplynulo porovnaním výsledkov z algoritmom fictitious play. Preto algoritmus nakoniec nebol zvolený, ako hlavný algoritmus pre umelú inteligenciu.

12. Výpočet equity

Ako sme mohli v ukážkovom kóde vidieť, veľmi dôležitý krok je výpočet equity hráča pri konkrétnej stratégii. Na to využijeme už spomínané koncové stavy hry. V našom algoritme však nepotrebujeme presne poznať equitu hráča, ale porovnať dve stratégie a zistiť ktorá z nich je výhodnejšia. Preto namiesto váženej equity cez všetky stavy hry počítame len so stavmi, ktoré môže hráč svojim rozhodnutím ovplyvniť - sú to práve tie stavy ktoré môžu nastať po jeho akcii.

Preto je metóda na výpočet equity GetEquity preťažená a funguje osobite pre push-ujúceho, call-ujúceho a overcall-ujúceho hráča. Ukážka kódu pre push-ujúceho hráča:

```
double EquilibristicTree::GetEquity(int player)
{
    double equity=0;
    for(int i=player;i<this->leafs.size();++i)
        for(int j=0;j<this->leafs.at(i).size();++j)
            for(int k=0;k<this->leafs.at(i).at(j).size();++k)
                equity+=leafs[i][j][k].CountEquity(player);

    return equity;
}
```

Opäť sa jedná o priamočiaru implementáciu. Trojrozmerný vektor leafs má v sebe uložené všetky stavy hry. Prvá súradnica určuje push-ujúceho hráča, druhá call-ujúceho hráča a posledná overcall-ujúceho hráča. Sú vyhradené špeciálne hodnoty pre prípad, že všetci hráči po push-i alebo call-e zložia.

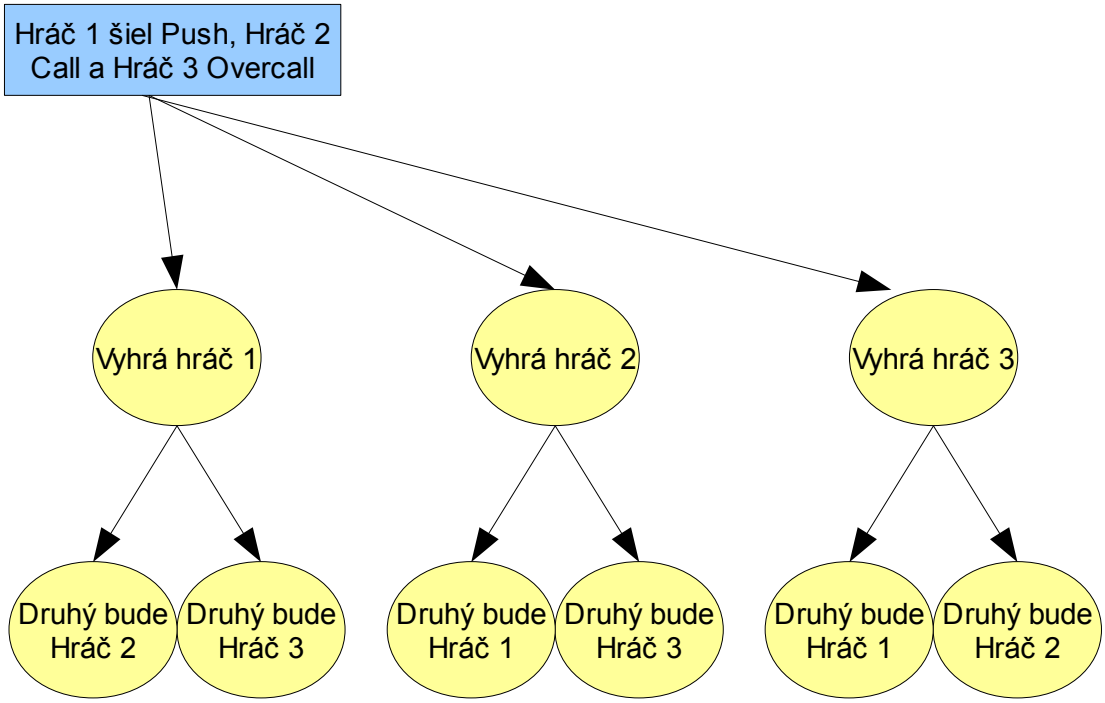
Ako je vidieť telo funkcie sa skladá z troch vnorených cyklov. Prvý cyklus určuje súradnice hráčov, ktorí môžu zahrať push po hráčovi, pre ktorého je equita počítaná a jeho samotného. Ďalšia súradnica potom pre každého z týchto hráčov prejde všetkých hráčov, ktorí môžu dorovnať a nakoniec sa spracujú všetci hráči ktorí môžu ovrecall-ovať. Tieto tri súradnice jednoznačne určia koncový stav hry. Pre každý takýto koncový stav sa zavolá jeho funkcia CountEquity (player), ktorá vráti equitu hráča v player danom stave vynásobenú pravdepodobnosťou, že tento stav nastane. Funkcia nakoniec vráti ako svoj výstup súčet týchto hodnôt.

12.1. Výpočet equity hráča v koncovom stave hry

Každý koncový stav, reprezentuje len stav pred flopom. Po vo vyložení všetkých

kariet však môže v prípade konfrontácie nastať viacero výsledkov v závislosti od toho ktorý hráč vyhrá a ako sa umiestnia zvyšní hráči.

Ako príklad, zobrazím na obrázku č. 11, možnosti ako môže dopadnúť stav z obrázku 10, keď išli all-in hráči č. 1, 2 a 3.



Obrázok 11: Možný vývin hry po vyložení spoločných kariet.

Uvedené možnosti nie sú všetky možné, vo veľmi malej časti prípadov môže dôjsť k tomu, že dvaja hráči majú na konci hry rovnako silnú výhernú kombináciu a časť výhry sa medzi nich rozdelí. Tento prípad má však zanedbateľný vplyv na výsledok. Namiesto toho, aby som ho uvažoval ako samostatný prípad, rozdelím jeho pravdepodobnosť rovnomerne medzi zúčastnených hráčov a pripočítam ju k pravdepodobnosti ich výhry tak, aby celková suma pravdepodobností bola opäť 1.

12.1.1. Štruktúra koncového stavu hry

V našom programe je stav hry reprezentovaný triedou StrategyLeaf. Trieda si pre každú možnú akciu, ktorú mohli hráči vykonať, drží vektor hráčov ktorí ju vykonali. V prípade, že ju nevykonal nikto, je tento vektor prázdny. Tieto vektory jednoznačne určujú akcie hráčov a teda aj o aký stav sa jedná. Ukážeme si teraz kód s najdôležitejšími dátovými štruktúrami:

```
QVector<int> foldPlayers; //hráči ktorí zložili namiesto push-u
```

```
QVector<int> pushPlayers; //hráči ktorí push-li(max pocet 1)
QVector<int> foldToPushPlayers; //hráči ktorí fold-li na push
QVector<int> calledPlayers;//hráči ktorí call-ovali
QVector<int> foldToCallPlayers;//hráči ktorí flod-ovali po calle
QVector<int> overcallPlayers; //hráči ktorí over-callovali
```

Ďalšou dôležitou štruktúrou je vektor icmValues, ktorý drží equitu pre každého hráča pre všetky prípady ako môže hra dopadnúť. Keď sa all-in ocitnú traja hráči je jeho veľkosť 6, ako sme mohli vidieť na predchádzajúcom obrázku. Pre dvoch hráčov je jeho veľkosť 2, a v prípade jedného hráča, alebo keď všetci zložia je veľkosti 1. Poslednou premennou, ktorú spomenieme, je pointer na rodičovskú triedu EquilibristicTree, ktorá obsahuje aktuálne stratégie hráčov potrebné pre výpočty.

Najdôležitejšia funkcia triedy StrategyLeaf, používaná pri výpočte equilibria je funkcia CountEquity, ktorá vracia equitu hráča, v tomto stave vynásobenú pravdepodobnosťou, že stav nastane.

Posledný problém pri výpočte equlibria je, ako rýchlo spočítať pravdepodobnosti výhry jednotlivých hráčov a pravdepodobnosť, že nastane daný koncový stav. Tu môžeme opäť využiť, že v našej abstrakcii hry sme počet stratégií zredukovali na 170 a obmedzili sme počet hráčov, ktorí sa zúčastnia konfrontácie na 3. Máme teda 170^3 všetkých možných kombinácií stratégií týchto hráčov. To nám umožňuje držať v operačnej pamäti tabuľky s predpočítanými hodnotami pre každú takúto kombináciu.

13. Výpočet equilibria pre hru dvoch hráčov (Heads Up)

Heads Up predstavuje hru dvoch hráčov s nulovým súčtom. Na rozdiel od hry viacerých hráčov, alebo hry dvoch hráčov s nekonštantným súčtom, nepatrí hľadanie equilibria do triedy PPAD úplných problémov. Sú známe algoritmy na ktoré nájdu equilibrium v polynomiálnom čase. Celý herný strom je však stále rádovo väčší, ako umožňuje zvládnuť súčasný stav výpočtovej techniky [7], preto sa opäť obmedzíme na situácie, keď hráči vsadia celý svoj stack preflop.

Pre tento prípad je možné vypočítať equilibrium v reálnom čase. Najprv sa generuje lineárny program popisujúci hru. Ten je následne vyriešený za pomoci externej knižnice. Výsledné equilibrium je použité pri hre. Na rozdiel od prípadu pre viacerých hráčov sa nejedná o aproximáciu equilibira, ale jeho presný výpočet (až na numerické nepresnosti), ktorý pre každú kartu určí optimálnu hru.

14. Generovanie tabuliek

Ako sme uviedli skôr, zásadnú úlohu pri výpočte equilibria hrajú tabuľky s predpočítanými pravdepodobnosťami. Teraz sa bližšie pozrieme na proces ich generovania.

14.1. Výpočet pravdepodobnosti výhry hráča

Na výpočet ďalších tabuliek budeme potrebovať najprv poznať pravdepodobnosť výhry konkrétnej kartovej kombinácie proti ďalším konkrétnym kombináciám. Tento výpočet je pomerne zložitý. Preto sme použili kód tretej strany. V zásade sa používajú dva rôzne postupy:

Exhaustive Exploration – vygenerujú všetky možné kombinácie spoločných kariet. Konečná pravdepodobnosť sa spočíta z pomeru, koľkokrát každý hráč vyhrá, prehrá a remizuje.

Monte Carlo Simulation – náhodne sa generujú spoločné karty, zaznamenáva sa pomer výhier, prehier a remíz pre každého hráča. Po dostatočnom počte iterácií ich pomer začne konvergovať k skutočnej hodnote.

Pri výpočtoch boli využité programy používajúce oba postupy.

14.1.1. Výpočet pravdepodobnosti výhry jednej z dvojice kartových kombinácií

Na vytvorenie tabuľky pre dvoch hráčov bol využitý upravený open source program Hold'Em Showdown [11]. Program pracuje enumeratívnym spôsobom, preto sú ním vygenerované výsledky presné. Dokáže však podať výsledok len pre karty s presne zadanými farbami. Napr. pri výpočte pravdepodobnosti výhry páru sedmičiek proti páru desiatok nie je možné zadať ako karty 77 a TT, ale je potrebné aj špecifikovať ich farbu, napr ThTc 7d7h.

My však potrebujeme poznať pravdepodobnosť v obecnom prípade (pri hre napr. držíme pár sedmičiek a vieme že súper nebude skladať akýkoľvek pár desiatok, bez ohľadu na farbu). Kombinácia farieb kariet však ovplyvňuje pravdepodobnosť výhry. Napríklad, v ukážkovom prípade boardy, ktoré budú obsahovať štyri srdcové karty vyhrá hráč s párom desiatok, pretože Th je vyššia karta ako 7h. Keby ale mal prvý hráč namiesto toho napríklad Ts tak by v tomto prípade by prehral.

Priamočiare riešenie problému je počítať hodnotu pre všetky možné kombinácie (6

pre každého hráča teda dokopy 36). Namiesto toho sa pri našom výpočte uvažujú iba tri možnosti – podľa počtu kariet, ktoré majú u oboch hráčov rovnakú farbu.

Pre každú možnosť vyberieme jednu dvojicu, ktorá ju reprezentuje (napr. TsTc a 7h7d pre žiadnu spoločnú farbu). Pretože pre každú možnosť vieme jednoducho spočítať jej pravdepodobnosť, dokážeme na základe týchto údajov určiť výslednú hodnotu. Výpočet bude prebiehať v uvádzanom prípade 12-krát rýchlejšie (výpočet pre jednu konkrétnu dvojicu je omnoho zložitejší ako ostatná réžia).

Aby bolo možné vypočítať celú tabuľku, bol napísaný kompletný rozhodovací strom s podmienkami. Príklad takej podmienky môže byť napríklad dotaz, či sa jedná o dvojicu vo farbe a mimo farby, ktorá zároveň nie je pár a obe dvojice majú práve jednu spoločnú karu. Tento strom pokrýva všetky možné stavy.

Výpočet tabuľky pre všetky možné kombinácie (veľkosti 169^2) však aj napriek spomínanej optimalizácii trval na viacerých počítačoch súčasne niekoľko dní strojového času. Preto bol pre tabuľku troch hráčov (veľkosti 169^3) zvolený výpočet pomocou knižnice používajúcej metódu Monte Carlo.

Použitá bola knižnica Poker Sleuth [12], ktorá dokáže spočítať pravdepodobnosť pre dvojicu bez presne zadaných farieb kariet.

14.1.2. Výpočet podmienenej pravdepodobnosti štartovných kombinácií

Ďalším parametrom, ktorý je potrebné poznať pre generovanie tabuliek, je tzv. card removal effect (v našom prípade sa jedná o podmienenú pravdepodobnosť jednej kartovej kombinácie za predpokladu, že je známa iná).

Príklad uplatnenia: pred nami išiel all-in hráč, o ktorom vieme že hrá iba AA a KK, my na ruke držíme AA. Chceme vedieť s akou pravdepodobnosťou vyhráme. Nesprávnym postupom by bolo spočítať pravdepodobnosť, že naše AA vyhrá proti AA, obdobne proti KK a tieto pravdepodobnosti spriemerovať. Pravdepodobnosť, že súper bude držať AA, bude výrazne nižšia ako pre KK pretože v balíčku ostávajú už len dve esá (dve držíme my).

Správny postup je vážiť každú súperovu kombináciu jej podmienenou pravdepodobnosťou. Kvôli rýchlosti ďalších výpočtov boli opäť predpočítané tabuľky podmienených pravdepodobnosti pre 2 a 3 hráčov.

14.2. Výpočet konečných tabuliek

Konečné tabuľky ktoré sú potrebné k výpočtu equilibria majú tvar:

```
//Tabuľka určujúca pravdepodobnosť výhry push-ujúceho hráča proti
//call-ujúcemu hráčovi s určitou range
double WinProbPushVsCall[170][170];

//Tabuľka určujúca pravdepodobnosť že call-ujúci hráč nezloží
double PlayProbPushVsCall[170][170];

//Tabuľka určujúca pravdepodobnosť výhry call-ujúceho hráča proti
//call-ujúcemu (call-range sa od push-range líši pri rovnakej
//hodnote)
double WinProbCallVsCall[170][170];

//Tabuľka určujúca pravdepodobnosť výhry push-ujúceho hráča proti
//dvom call-ujúcim
double WinProbPalyer1PushVsCallVsCall[170][170][170];

//Tabuľaka určujúca pravdpodobnosť výhry call-ujúceho hráča proti
//push-ujúcemu a call-ujúcemu
double WinProbPalyer2PushVsCallVsCall[170][170][170];

//Tabuľka určujúca pravdepodobnosť že overcalujúci hráč nezloží
double PlayProbPalyer3PushVsCallVsCall[170][170][170];

//Tabuľka určujúca pravdepodobnosť že pushujúci hráč bude pushovať
double PlayProbPush[170];
```

Tieto tabuľky sa však kvôli urýchleniu výpočtov nepočítali priamo. Najprv sa predpočítali tabuľky jednotlivých kombinácií proti range, potom sa z týchto kariet na ruke (ďalej len ruka) zostavila range hráča a výsledná hodnota vznikla ako súčet hodnôt pre jednotlivé ruky vážený ich pravdepodobnosťou.

Podobne boli zostavené aj tabuľky pre jednotlivé kombinácie proti range. Pre kombináciu bol vypočítaný výsledok proti každej samostatnej súperovej kombinácii v range a do tabuľky bol zapísaný súčet týchto hodnôt vážený podmienenou pravdepodobnosťou kombinácií.

V programe sa nachádzajú aj ďalšie tabuľky, ktoré slúžia však na výpočty vykonávané až po spočítaní equilibria.

Všetky tabuľky sú pred spustením programu načítané do operačnej pamäti, čo umožňuje rýchly prístup k potrebným hodnotám počas výpočtu.

15. Určenie equity konkrétnej štartovnej kombinácie

Aby sme nakoniec zmenšili nevýhody ktoré vyplývajú z pevne daného usporiadania kombinácií hráča, po spočítaní equilibria vypočítame pre každú kombináciu equitu osobitne. Pre hru nás zaujíma vzťah equity konkrétnej ruky po akcii fold a po akcii all-in. V prípade že equita po akcii all-in je väčšia, hráč kombináciu bude hrať, v opačnom prípade ju zloží. Výpočet equity prebieha samostatne pre akciu fold a akciu all-in. Priebeh výpočtu je podobný ako pri výpočte equity pre aproximáciu equilibria. Obstaráva ho samostatná sada funkcií využívajúca ďalšiu skupinu tabuliek.

16. Expertný systém

Ako už bolo spomínané, súčasný stav umelej inteligencie je pomerne ďaleko od stavu, keď bude možne porážať ľudských expertov v hre z vysokými stackmi. Je tomu tak dokonca aj v prípade dvoch hráčov. Preto bol na tento typ hry použitý expertný systém. Tento systém sa tiež využíva pri hre s nízkymi stackmi, keď nastane hra postflop. Táto situácia nastane keď ľudský hráč dorovná a žiadny hráč sa nerozhodne vsadiť all-in.

16.1. Požadované vlastnosti

Jeden zo základných požiadaviek na systém je ľahká úprava chovania systému, aby ho bolo možné prispôsobiť rôznym typom hry a tiež zohľadniť praktické skúsenosti získané z jeho použitia. Ďalšou dôležitou vlastnosťou je, aby systém pri rozhodovaní zohľadňoval dôležité strategické aspekty hry a mohol tak podať kvalitnú expertnú radu.

16.2. Návrh systému

Systém bol navrhnutý tak, aby boli hlavné parametre jeho chovania uložené externe v xml súboroch. Exprertná rada je vydaná na základe týchto parametrov a niekoľkých strategických aspektov aktuálnej hry.

Spomeňme teraz najdôležitejšie charakteristiky hry zohľadnené v rozhodovacom procese. Prvá vec, ktorú musíme pred rozhodnutím zohľadniť je fáza hry. Najväčší rozdiel je, či sa jedná o hru prefolp, alebo postfop, avšak aj pre jednotlivé fázy postflop sa stratégia líši. Ďalšia prirodzená skutočnosť, ktorú hráč musí zohľadniť, je sila kariet ktoré drž hráč na ruke. Na tento účel je použitá metrika E(HS) [10]. Táto metrika prevádza kartu na reálne číslo, pričom jeho hodnota udáva pravdepodobnosť víťazstva kartovej kombinácie hráča proti náhodnej kombinácii súpera. Je počítaná za pomoci externej knižnice. Pred flopom je sila kombinácie hodnotená podľa usporiadania, ktoré bolo použité na tvorbu abstrakcie pre preflop hru. Ďalej sú použité parametre ako agresivita vyskytujúca sa v hre, veľkosť potu, akcie súperov a veľkosť ich stávky a veľkosti stacku hráčov.

Program je nastavený tak, aby sa snažil dodržiavať základné strategické aspekty predpokladanej optimálnej hry konkrétne modelu nazývaného 0-1 game [1]. Ako príklad môžeme uviesť hru na riveri. Po predchádzajúcom checku súpera hráč vsádza svoje najlepšie karty (tzv. Valuebett) a na blufovanie používa svoje najslabšie karty.

V prípade, že súper vsadil tak umelá inteligencia zo svojimi najsilnejšími kartami raisuje, o niečo slabšie karty dorovnáva a najsilnejšie karty, ktoré nie sú dosť silné na dorovnanie zase blufuje. Frekvencie jednotlivých akcií pre jednotlivé prípady sú však určené expertnými radami uloženými xml súbore. Je možné vytvárať a editovať viacero takýchto súborov a tým dosiahnuť požadované chovanie hráčov, pričom každý z nich môže dodržiavať svoju vlastnú stratégiu.

17. Zhodnotenie umelej inteligencie

Ako sme už uviedli, zásadné je rozdelenie hry na hru s malými blindami, vzhľadom k stackom a hru s veľkými blindami. Pri hre s veľkými blindami sme sa obmedzili na hru push/fold, ktorá je vcelku blízka optimálnej hre a a je málo exploitovateľná. Následne sme predstavili tri algoritmy na aproximáciu equilibria. Vzhľadom k faktu, že človek sa v takejto hre musí spoliehať na svoj odhad a intuíciu zatiaľ čo počítač dokáže počítať equitu jednotlivých stratégii a aproximáciu equilibria pomerne presne verím, že umelá inteligencia dokáže v takejto situácii poraziť aj špičkových ľudských protivníkov. Proti programu však nebola zahratá dostatočná vzorka hier s dostatočne veľkou vzorkou kvalitných hráčov, aby bolo možné tento predpoklad štatisticky overiť.

Pri hre s veľkými stackmi môže umelá inteligencia pri dobrom nastavení parametrov predstavovať solídneho súpera priemernému hráčovi. Pri správne vyváženom pomere blufov a sádzok zo silnými kartami (tzv. valuebet) môže byť problematické exploitovať umelú inteligenciu bez explicitnej znalosti jej nastavení aj pre dobrého hráča.

Celková sila umelej inteligencie teda závisí od typu turnaja, ktorý určuje pomer týchto dvoch možných situácií. Predpokladáme, že v turnajoch s nízkou veľkosťou štartovných stackov a rýchlym rastom blindov dokáže program porážať ľudských protivníkov, v ostatných prípadoch však tiež môže poskytnúť solídnu hru.

18. Ďalšie funkcie programu

Umelá inteligencia bola doplnená o simuláciu hry a grafické používateľské rozhranie.

18.1. Grafické Rozhranie programu

Grafické užívateľské rozhranie sa skladá z doch častí a to menu hry a samotného zobrazenia turnaja. V menu je možné nastaviť základné parametre turnajov. Aby program vyhovoval súčasným trendom v online pokri, je možné spustiť a hrať viacero nezávislých turnajov súčasne. Interakcia grafického rozhrania a zvyšku hry, ako aj interakcia v rámci samotného rozhrania je riešená pomocou technológie signálov a slotov, ktorú poskytuje knižnica Qt.

18.2. Grafické Rozhranie turnaja

Pri návrhu grafického rozhrania som myslel najmä na jeho prehľadnosť a dobré chovanie pri prispôsobovaní veľkosti okna zobrazujúceho turnaj, aby bolo možné hrať viac turnajov naraz.

Aby bolo vyhovené týmto požiadavkom, celá geometria zobrazenia turnaja sa plynulo prispôsobuje veľkosti stolu. Pre dosiahnutiu prehľadnosti bola použitá grafika ktorá zobrazuje pokrový stôl a hráčov symbolicky. Zároveň sú dôležité informácie, napríklad karty a stávky hráčov väčšie, vzhľadom k veľkosti okna, ako na bežnej internetovej herni, čo ďalej zvyšuje prehľadnosť. Aby sa v takomto nastavení zabránilo tomu, že po vzájomnom prekrývaní budú dôležité prvky zle zobrazené, zobrazujú sa karty súperov obrátené rubom priehľadne (myslíme tým transparentnosť prvku, nie viditeľnosť hodnoty). Samotné obrázky kariet sú pre krajšie zobrazenie v rozličných veľkostiach uložené vo vektorovom formáte SVG. Pre plynulú odozvu grafického prostredia sú po svojom vyrendrovaní na požadovanú veľkosť catchované.

Bola implementovaná aj originálna vlastnosť, ktorá zvýšila možnosť nastavenia zobrazenia turnaja ako aj užívateľskú atraktivitu. Pohľad na stôl je možné myšou jednoducho a plynule otáčať pričom sa mení aj pozícia hráčov pri stole.

19. Užívateľská dokumentácia

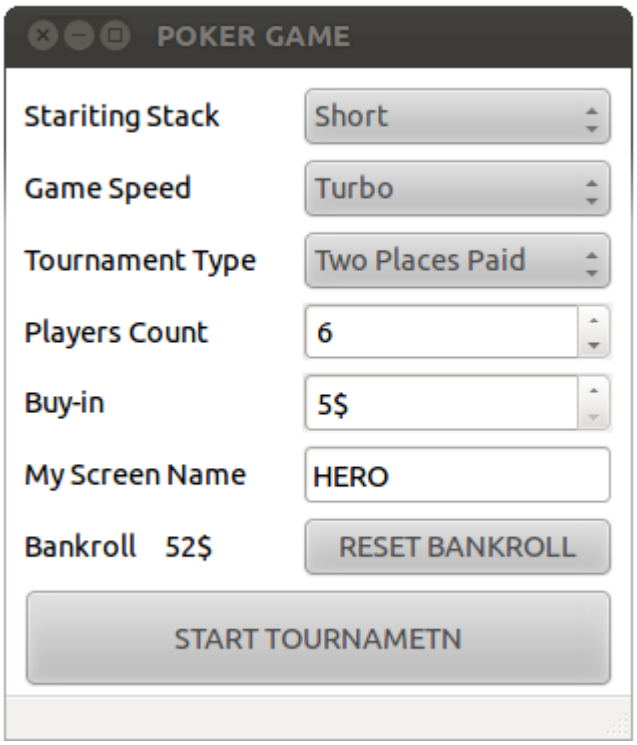
V tejto kapitole sa pozrieme na výslednú aplikáciu prácu z pohľadu používateľa. Ukážeme bežné používanie programu ako aj možnosti pokročilého nastavenia niektorých parametrov.

19.1. Grafické užívateľské prostredie

Najprv sa pozrieme na ovládanie programu pomocou grafického prostredia.

19.1.1. Menu hry

Spustenie programu môže kvôli načítaniu údajov z pevného disku trvať aj niekoľko sekúnd. Po spustení programu sa otvorí menu hry zobrazené na obrázku č. 12.



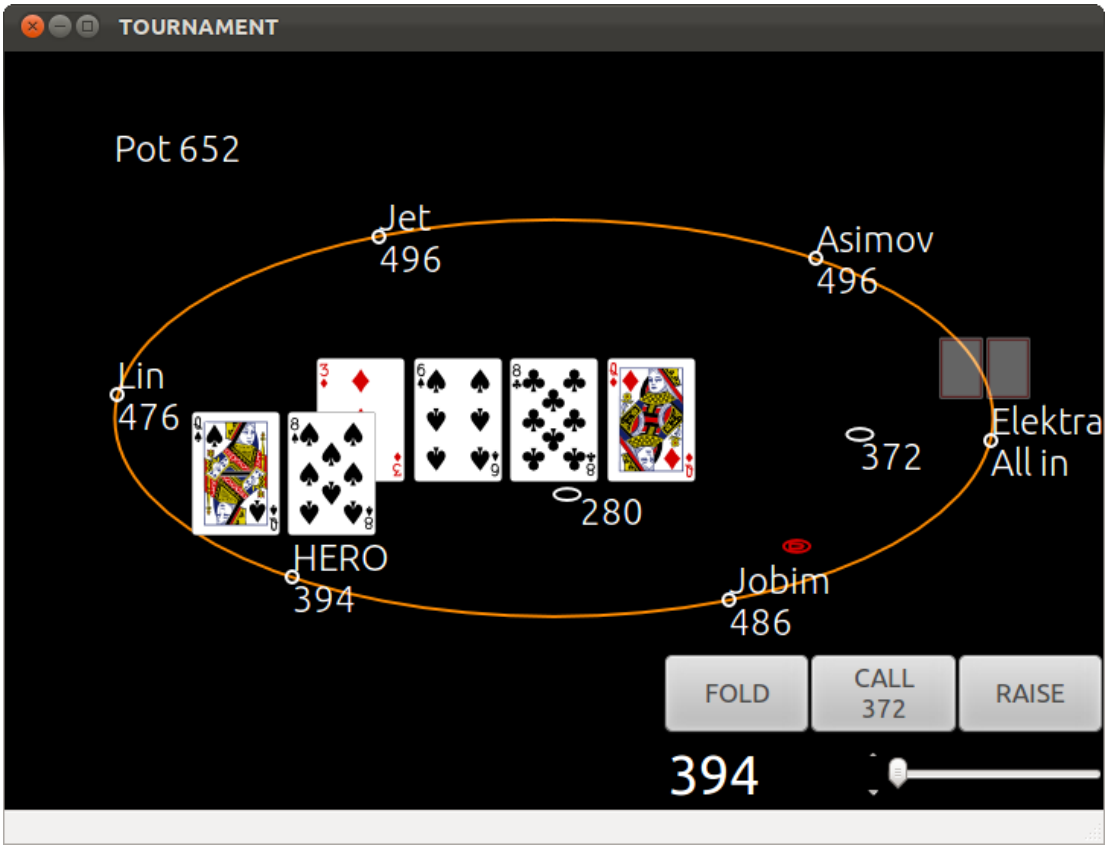
Obrázok 12: Menu hry

Menu je jednoduché, Prvých päť nastavení sa týka parametrov turnaja. Dôležitým nastavením hry je Buy-in. Nastavuje sa ním čiastka ktorú hráč do turnaja vloží. Táto čiastka sa potom odpočíta od celkovej sumy, ktorú hráč v aktuálnej chvíli vlastní (tzv. bankroll). Táto hodnota je zobrazená v kolónke Bankroll. Keď hráč prehrá svoje peniaze, môže zase bankroll nastaviť na základnú hodnotu tlačítkom RESET BAKROLL. Toto tlačítko nie je možné použiť, keď sú otvorené turnaje, aby hráč nemohol podvádzať tým, že bude postupne otvárať turnaje a bankroll si dopĺňať.

Cieľom hry je vybudovať čo najväčší bankroll. V spodnej časti menu sa nachádza tlačítko START TOURNAMENT, ktoré vytvorí turnaj s nastavenými parametrami. Počet turnajov, ktoré je možné naraz spustiť obmedzuje len bankroll hráča, prípadne parametre hardware.

19.1.2. Hra Turnaju

Po spustení turnaja sa ukáže okno zobrazené na obrázku č. 13.



Obrázok 13: Zobrazenie turnaju

Turnaj prebieha podľa pravidiel hry Texas Holdem. V okne turnaja sú zobrazené všetky údaje potrebné na hru. Hráč volí svoje akcie tlačítkami v pravom dolnom rohu hry. Tlačítka sa objavia keď sa hráč dostane na ťah. Sú potom dve možnosti ako nastaviť veľkosť stávky. Jedna z nich je pomocou posuvného ukazovateľa, druhá je možnosť označiť zobrazenú stávku a zadať hodnotu pomocou klávesnice. Je tiež možné zmeniť otočenie stola ťahom myšou po okne turnaja. Hráč môže kedykoľvek opustiť turnaj zatvorením okna turnaja. V prípade, že v dobe zatvorenia turnaja hráč ukončil svoju hru, v turnaji sa mu pripíše výhra podľa výhernej štruktúry turnaja.

19.2. Pokročilé možnosti nastavenia

Jedná sa o možnosti nastavenia niektorých parametrov počítačových hráčov, ktoré

sú však určené iba skúseným užívateľom.

19.2.1. Nastavenie hráčov v turnaji

Pomocou prepísania xml súborov s menami hráčov je možné zmeniť množinu hráčov, z ktorej sa pri generovaní turnaja hráči vyberajú. Pri prepisovaní zložky treba dodržať štruktúru tohto súboru. Odporúča sa pred tým súbor zálohovať. Súbor nie je pri načítaní kontrolovaný, preto chyba v súbore môže spôsobiť pád programu.

19.2.2. Nastavenie stratégií hráčov

Parametre podľa ktorých sa správa umelá inteligencia sú taktiež uložené v xml súboroch. Súbor je nazvaný menom hráča pre ktorého uchováva parametre. Pre hráčov ktorých chovanie nie je ošetrené špeciálnym súborom sa údaje načítajú zo súboru Default.xml. Preto nie je možné vytvoriť hráča z menom „Default“ a jeho hru nastaviť osobitne. Je potrebné zachovať štruktúru xml súboru, v súbore musia byť zachované všetky parametre, môže sa meniť len ich hodnota. Pri načítaní nie sú súbory kontrolované, preto chyby v súboroch môžu spôsobiť pád programu. Nie je kontrolovaná ani samotná hodnota parametrov v súbore, preto aj ich zlé nastavenie môže spôsobiť pád. Na to aby užívateľ zistil presnú funkciu parametrov je potrebné naštudovať programátorskú dokumentáciu programu. Prípadne je možné postupovať skusmo, v takom prípade sa však dôrazne doporučuje súbory zálohovať.

20. Záver

V práci sme porovnali niekoľko možných postupov, ktoré môžu byť použité na tvorbu umelej inteligencie pre poker. Na implementáciu sme si potom vybrali dva postupy a to aproximáciu Nashovho euilibria a využitie expertného systému.

Najväčšiu pozornosť sme venovali aproximácii Nashovho equilibria. Predstavili sme tri algoritmy, z toho dve originálne heuristiky, využívajúce špecifickú štruktúru pokrovej abstrakcie. Tieto algoritmy umožňujú dosahovať výrazne väčšiu rýchlosť výpočtu, ako pri bežne používaných postupoch. Tento predpoklad sme empiricky overili a zároveň sme sa pozreli na presnosť výpočtu týchto algoritmov. Optimálne výsledky dosiahol originálny heuristický algoritmus, ktorý používa najlepšiu odpoveď len na aktuálnu stratégiu súpera, preto bol nakoniec vybraný, ako hlavný algoritmus pre implementáciu. Pre prípady, kde nebolo možné použiť aproximáciu Nashovho euilibria, bol rozobraný návrh a implementácia expertného systému.

Výsledkom je silná umelá inteligencia, ktorá dosahuje dobré výsledky, najmä pri hre s nízkym počtom žetónov. Pri takejto hre dokonca predpokladáme, že umelá inteligencia je schopná porážať špičkových ľudských expertov. Na overenie tohto predpokladu sa však bohužiaľ nepodarilo získať dostatočne kvalitné štatistické údaje.

Text sa stručne venuje aj ostatným stránkam implementácie programu. Konečným produktom práce je plnohodnotná aplikácia, určená pre koncového používateľa. Simuluje pokrovú hru, poskytuje silnú umelú inteligenciu a atraktívne grafické užívateľské prostredie. Tým bol cieľ práce úspešne splnený.

20.1. Ďalšia práca

Ďalej by som sa rád zameral na vylepšenie modelu, podľa ktorého sa prevádza počet žetónov na očakávanú výhru oproti modelu použitému v súčastnosti. Druhý smer práce, ktorý sa prirodzene ponúka je pokúsiť sa zlepšiť umelú inteligenciu pre hru s veľkým počtom žetónov aspoň pre prípad dvoch hráčov. Tomuto problému by som sa v budúcnosti tiež rád venoval.

21. Zdroje

[1] Chen, Bill - Ankenman, Jerrod. The Mathematics of Poker, Conjelco 2006, ISBN 1-886070-25-3.

[2] Melcher, Helmuth. Nash ICM Library FAQ [online]. July 14, 2008 [cit. 2012-11-05]. URL: <http://www.insight-pokerhound.com/Nashicmfaq.pdf >

[3] Pokerstars - Poker Hand Rankings [online]. [cit. 2012-11-05]. URL: <http://www.pokerstars.net/poker/rules/hand-rankings/ >

[4] Holdemresources - Sample ICM Implementation [online]. [cit. 2012-11-05]. URL: <http://www.holdemresources.net/hr/sngs/icm/icmjava.html>

[5] Ganzfried, Sam Ganzfried - Sandholm, Tuomas. Computing an Approximate Jam/Fold Equilibrium for 3-player No-Limit Texas Hold'em Tournaments [online]. 2008 [cit. 2012-11-05]. URL: <http://www.cs.cmu.edu/~sandholm/3-player%20jam-fold.AAMAS08.pdf>

[6] Daskalakis, Constantinos – Papadimitriou, Christos H. Three-Player Games Are Hard [online]. 2005 [cit. 2012-11-05]. URL: <http://www.cs.berkeley.edu/~christos/papers/3players.pdf >

[7] Schnizlein, David Paul. STATE TRANSLATION IN NO-LIMIT POKER [online]. University of Alberta publicetions, 2009 [cit. 2012-11-05]. URL: <http://poker.cs.ualberta.ca/publications/schnizlein.msc.pdf>

[8] Fudenberg, D. and D.K. Levine. The Theory of Learning in Games. 1. vyd. MIT Press, 1998. 292 pp. ISBN 0-262-06194-5.

[9] Szafron, Duane – Risk, Nick, Abou. Using Counterfactual Regret Minimization to Create Competitive Multiplayer Poker Agents [online]. University of Alberta publicetions, 2010 [cit. 2012-11-05]. URL: <http://poker.cs.ualberta.ca/publications/AAMAS10.pdf>

[10] Johanson, Michael, Bradle. Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player [online]. University of Alberta publicetions, 2007 [cit. 2012-11-05]. URL: <http://poker.cs.ualberta.ca/publications/johanson.msc.pdf >

[11] Steve Brecher. Hold'Em Showdown. [online]. [cit. 2012-11-05]. URL: <http://www.brecware.com/Software/software.html>

[12] Poker Sleuth Software. [online]. [cit. 2012-11-05]. URL: <http://pokersleuth.com>