Možnosti rozhodovacích agentov hrajúcich hracie karty

Bakalárska práca

Juraj Barič

Univerzita
FMFI
KI

9.2.1 Informatika

Vedúci bc. práce:

doc. RNDr. Mária Markošová , PhD.

Bratislava 2009

Čestne prehlasujem, že som túto prácu vypracoval samostatne s využitím uvedenej literatúry a s podporou mojej vedúcej.

Bratislava, máj 2009

Abstrakt

Touto prácou by som chcel preskúmať, ako sa dá simulovať myslenie človeka v počítači, a ako fungujú rozhodovací agenti, keď nemajú úplne informácie o svojom prostredi. Za týmto účelom som sa rozhodol naprogramovať agentov, ktorí budú hrať kartovú hru *šnaps*, a otestovať ich úspešnosť pri rôznych algoritmoch.

Abstract

In this thesis I would like to study how human beeings think. I do it with a help of computer simulation of decision agents. Agents do not have all the informations about their enviroment. Model example which I implement is "SNAPS"-card-game-player-agent. My goal is to test their performance using different algorithms.

.

1. Obsah

2. Úvod

Cieľom tejto práce je skúmať možnosti rozhodovacieho agenta v prostredí s neúplnými informáciami, kde sa jeho záujmy dostávajú do konfliktu s inými agentami. Ako príklad som si vzal agenta hrajúceho karty.

Predtým, než prejdem k samotným riešeniam, by som si dovolil niečo napísať o kartových hrách všeobecne. Prvé zmienky o hrách podobných kartovým pochádzajú z 12. Storočia po Kristovi, z oblasti Východnej Ázie, konkrétne Čína a Kórea. Jednalo sa o kartónové platničky, no pravidlá hry sa mi nepodarilo vypátrať. Neskôr, v 16. storočí, sa v Indií objavili kruhové kartičky.

Podľa môjho zdroja [3] nie je jasné, ako sa karty dostali do Európy (predpokladá sa, že ich priniesli Moslimovia z Orientu), no prvá zmienka o nich je z roku 1367, a to v dokumente „Verbot des Gebetbuch des Teufels“ (Zákaz čertovej modlitebnej knihy).

Ako som sa ďalej dopátral, našli sa dva skoršie zdroje. Prvý je údajný záznam v kronike Sandra di Pipozza z Benátok. Druhý akýsi zákaz biskupa z Würzburgu z roku 1329 pre všetkých duchovných v jeho diecéte. No ich pravosť bola vyvrátená výskumníkmi. [3]

Vplyv hracích kariet na spoločnosť a jednotlivca

Hoci ide v prvom rade o zábavu, z kariet sa postupom času vyvinul biznis. Treba priznať, že tento spôsob hazardu pripravil už veľa naivných hráčov o celý majetok. Samozrejme, väčšina ľudí hrajúcich o peniaze nehrá kvôli zárobku, ale svoje peniaze vsádza skôr pre väčšiu zábavu, a napätie pri hre, a sumy sa pohybujú v centoch (pôvodný slovenský krčmový výraz bol „päťdesiathalierníková“ hra, pri ktorej bola už výhra dvadsať korún– dnes cca 0.63 Eur - za celý večer považovaná za veľký jackpot).

Počítačové kartové hry

Keďže kartové hry su spoločenské, ich nástup na scénu počítačov mal mierne oneskorenie oproti single-player (jednohráčovým) hrám, a rozmohli sa až príchodom internetu do domácností. [2] No dnes je možné nahradiť spoluhráčov počítačom. A o to by som sa rád pokúsil. Za hru som si vybral *šnaps*. Dôvod môjho výberu je ten, že túto hru som doteraz ešte nenašiel.

V ďalšom texte budem často používať pojem „*agent*.“ Pokiaľ nebude uvedené inak, pod týmto pojmom budeme rozumieť systém slúžiaci na jednanie a rozhodovanie. Vo všeobecnosti sa skladá z následujúcich častí: vnemová, vykonávacia, vedomostná, a logická.

V prípade mojej práce bude vnemovú časť tvoriť rozhranie, ktorým sa agentovi pošle správa o ťahu iného hráča. Táto bude priamo meniť vedomostnú bázu „Knowledge base“ (ďalej len KB). Do KB si agent ukladá informácie o svojich kartách, o tom, kedy ktorý hráč ako ťahal, o nahrané body, a niekoľko podmienok, ku ktoré musia byť splnené vzhľadom na priebeh a pravidlá hry. Pod logickou časťou rozumieme celý rozhodovací systém, ktorý na základe KB zvolí, ktorú kartu je najvhodnejšie vyložiť. A nakoniec, konať bude mať agent možnosť 3 rôznymi akciami. Voliť kartu (na začiatku hry), ťahať kartu, a hlásiť „dosť“ – čo sa pod tým myslí, vysvetlím v časti „Pravidlá“.

3. Šnaps - pravidlá

V tejto kapitole by som stručne vysvetlil pravidlá hry šnaps.

Hra je určená štyrom hráčom, pri hre sa používajú sedmové karty bez sedmičiek a osmičiek, takže sa hrá s dvadsaťštyri rôznymi kartami.

Každý hráč na začiatku dostane šesť kariet. Jeden z hráčov zvolí na začiatku tromfovú kartu. Hráč, ktorý má danú kartu na začiatku hry na ruke, bude počas celej hry pre voliaceho hráča spoluhráčom, v prípade, že má sám voliaci danú kartu, je sám (tzv. „*sumec*“). Ostatní (dvaja alebo traja) hráči hrajú spolu.

Tromfová farba stojí počas celej hry nad ostatnými. Bodové hodnoty sa nemenia, ale platí, že akákoľvek karta tromfovej farby je silnejšia ako akákoľvek netromfová karta.

Cieľom hry je nazbierať šesťdesiatšesť bodov za vyhrané štichy, alebo vyhrať posledný štich. Prvý štich začína hráč, ktorý volil, ostatné štichy začína vždy ten hráč, ktorý vyhral predchádzajúci štich.

Pod pojmom *štich* sa rozumie jedno kolo, počas ktorého každý hráč vyloží jednu kartu, a na konci všetky karty vezme jeden hráč – ten, ktorý vyšiel najsilnejšou kartou. Vziať štich znamená, že dané karty sú jeho výhrou, odložia sa na kopu, a hráčovi sa priráta bodový zisk zodpovedajúci ohodnoteniam kariet.

Pri vykladaní kariet musí každý hráč vyložiť kartu tej istej farby, ako sa štich začal, a ak má, musí dať vyššiu kartu, ako najvyššia vyložená karta danej farby v danom štichu. Tejto povinnosti sa hovorí, že hráč musí *priznať* farbu, a o farbe, ktorú má prvá karta, sa hovorí, že sa *pýta*. Ak nemá farbu, musí potiahnuť tromfovou kartou, a pokiaľ možno, prebiť všetky ostatné karty(vyložiť silnejšiu tromfovú kartu, aká bola v danom štichu vyloýená). Ak nemá ani tromfovú kartu, môže vyhodiť ľubovoľnú.

Z týchto povinností plynie možnosť pre ostatných hráčov zistiť, aké karty ostatní hráči určite nemajú (a zúžiť tak výber hráčov, ktorý danú kartu mať môžu).

Príklad 3.1: Pre objasnenie týchto povinností uvediem príklad. Nech je tromfová farba žaluď a nech je takéto rozloženie kariet:

Hráč 1

- Červený dolník
- Zelené eso

Hráč 2

- Žaluď dolník
- Žaluď eso

Hráč 3

- Žaluď král
- Červená deviatka

Hráč 4

- Guľová desiatka
- Guľová deviatka

Nech posledný štich vyhral hráč č.1. Takže teraz začína. Vyloží červeného dolníka. Na rade je hráč 4. Ten nemá farbu, takže nemá čo priznať. Taktiež nemá tromfa, teda môže dať ľubovoľnú kartu.

Potom ťahá hráč 3. Pýta sa červená farba, hráč 3 má iba jedinú červenú kartu (deviatku), tak ju musí vyložiť. Štich končí hráč 2. Červenú farbu nemá, ale má tromfovú kartu, ktorou môže celý čtich vyhrať. Môže dať ľubovoľnú z nich.

Predstavme si, že by hráč 3 nemal červenú, ale zelenú deviatku. V tom prípade by nemohol priznať farbu, a tak by musel vyjsť tromfovou kartou (v tomto prípade žaluďový král). Potom by hráč 2, keďže nemá červenú kartu, musel vyložiť tromfa, ale nie ľubovoľného. Musel by dať vyššieho, ako král (ktorý je v danom štichu vyložený), teda by musel potiahnuť eso.

Štich vezme ten hráč, ktorý vyložil najvyššieho tromfa, prípadne ten, ktorý vyložil najvyššiu kartu tej farby, aká bola vyložená na začiatku štichu. Ináč ostáva štich hráčovi, ktorý ho začal.

Karty sú následovne zoradené podľa sily, a majú následovné hodnoty:

- Deväť – 0 bodov
- Dolník – 2 body
- Horník – 3 body
- Kráľ – 4 body
- Desať – 10 bodov
- Eso – 11 bodov

Na začiatku ťahu môže hráč povedať *dosť* – ak si je istý, že spolu so spoluhráčom nazbierali šesťdesiatšesť bodov. Hra sa končí, a v prípade, že mal tých bodov naozaj dosť, vyhral, v opačnom prípade prehral. Pre úplnosť dodám, že akonáhle vyjde nejaký hráč volenou kartou, je jasné, kto je komu spoluhráč. Samozrejme, hráč môže povedať

dosť aj v prípade, že ešte nevie, kto je jeho spoluhráč. V tom prípade sa po ukončení hry hráč, ktorý mal danú kartu, prizná, aby sa mohli zrátať body.

Okrem toho môže zahlásiť „*za dvadsať*“ – ak má na ruke horníka a kráľa jednej farby, a začína štich jednou z tých 2 kariet. V tom prípade, akonáhle získa on, alebo jeho spoluhráč čo i len jeden štich, majú bodový bonus + 20 bodov. V prípade, že sa jedná o tromfové karty, hlási 40, a bodový bonus je + 40 bodov.

Ešte som dlžný vysvetliť jednu štandartnú hlášku. Pred začatím hry môže hráč, ktorý hrá proti voliacemu hráčovi zahlásiť *kontru*, čo znamená, že na konci hry sa body zdvojnásobia, a to bez ohľadu na to, kto vyhral, a akým spôsobom. Touto hláškou sa samozrejme prezradí, a všetci okamžite vedia, že nemá na ruke zvolenú kartu. Túto hlášku obvykle hlási hráč, ktorý si je istý svojim víťazstvom, ináč je to samozrejme risk.

4. Hry s viac ako dvomi hráčmi

Ako sa také situácie v hrách vlastne riešia? Pre moje účely sa treba zamerať na hry s diskrétnym dejom – teda také, kde je každá udalosť oddelená od iných, následujú v určitom poradí a je ich obmedzený počet. Inými slovami, hráči sa striedajú vo svojich akciách, každý z nich sa dostane na rad, a zatiaľ ostatní nemôžu nič robiť.

K takým hrám patrí napríklad šach, pexeso, sabotér, bang, a iné spoločenské stolové hry. Najjednoduchšie zo spomínaných je pexeso , lebo pri ňom nie je treba žiaden zložitý algoritmus, keďže počítač má dokonalú pamäť, a väčšina výberov je náhodná.

Čo je úloha hráča? Samozrejme, nájsť svoj najoptimálnejší ťah – teda taký, ktorý ho čo najviac priblíži k výhre, alebo jeho výhru zvýrazní. Ak nie je možná, tak aspoň zmenší stratu pri prehre.

Predtým, než začnem popisovať hociktorý algoritmus, by som zadefinoval značenie, ktoré budem používať pri popise zložitostí algoritmov. Budeme predpokladať, že algoritmus má na vstupe dáta (pole, zoznam, graf, ...) o veľkosti (dĺžka, počet uzlov, ...) *n*. Jeho časovú zložitosť bude popisovať funkcia *f* : N->N, ktorá pre každú hodnotu *n* vráti počet operácii, ktoré algoritmus vykoná. Napríklad inicializácia pola o dĺžke *n* bude mať zloťitosť f(n) = 2*n + 1. Na začiatku algoritmus nastaví index na nula, a potom urobí algoritmus n-krát dve operácie: zvýši index, a do políčka so aktuálnym indexom sa priradí počiatočná hodnota.

Pre nás však nie je dôležité určovať presnú funkciu. Algoritmus so zložitosťou 4*n vykoná síce takmer dvakrát toľko operácii, ale vzhľadom na rýchlosť počítačov je celkom jedno, či bude výpočet trvať 0,003 sekundy, alebo 0,006 sekundy. Skutočne dôležitý je stupeň najvyššieho polynómu, respektíve základ exponentu, ak je funkcia exponenciálna. Pre malé vstupy (malé *n*) algoritmus zbehne tak rýchlo, že si uživateľ čakajúci na odpoveď programu ani neuvedomí zdržanie. Pre veľké vstupy sa algoritmus zdržuje, ak je funkcia rýchlo rastúca.

Pre popis zložitosti budeme preto používať jednoduché funkcie, ktorými ohraničíme konkrétnu funkciu algoritmu.

Def 4.1: Majme funkcie f(n) a g(n).

1. Potom značenie f(n) = O(g(n)) znamená, že: existuje konštanta $a > 0$ taká, že pre všetky *n* platí:

f(n) <= g(n) * a

2. Značenie f(n) = Θ(g(n)) znamená, že: existujú konštanty *a,b* > 0 také, že pre všetky *n* platí:

g(n) * b <= f(n) <= g(n) * a

4.1. Minimax

Ku štandartným algoritmom, ktoré sa používaju pri hľadaní najoptimálnejšieho ťahu, patrí minimax, a jeho aplikovateľná verzia α-β orezávanie.

Minimax ([1] - s. 215)

Algoritmus vygeneruje strom ťahov, kde každý uzol predstavuje situáciu v hre, a každá väzba (hrana) akciu, ktorá z nejakej situácie povedie k ďalšej možnej situácii (tzv. synovskej).

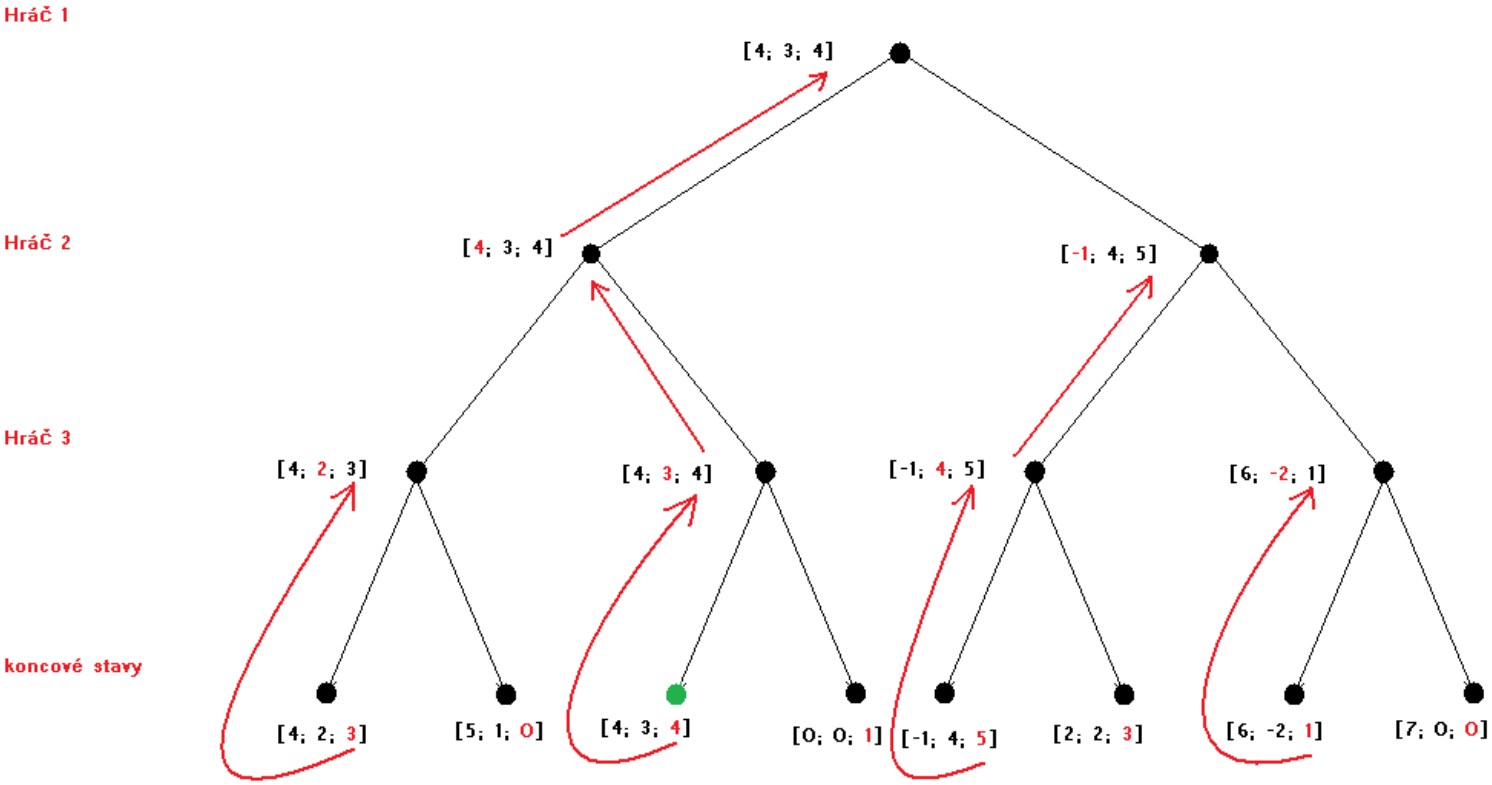
Koreň stromu je východzia pozícia hráča, a listami sú koncové stavy hry.

Každá hra by mala mať svoj vyhodnocovací systém konečných stavov, ktorý by každému stavu dokázal prideliť výsledkový vektor hráčov – teda akési ohodnotenie toho, ako hráči obstáli v hre. Hodnoty a ich rozsah sa menia od hry k hre, pričom platí, že kladné číslo znamená výhru, záporné prehru a nula remízu.

Majme hráčov h_1, ... , h_n, potom vyhodnocovacia heuristika priradí konečnému stavu vektor $(a_1,...,a_n)$, kde a_i je vyhodnotenie hráča h_i.

Algoritmus minimax generuje strom do hĺbky, a spätne priraďuje nelistovým uzlom jeden z vektorov svojich synov. Ako sa volí ťah, ukážeme na príklade hry troch hráčov.

Nech sme v nejakej situácii (a jej zodpovedajúcom uzle), a na ťahu je hráč i. Potom Algoritmus vyberie taký ťah, v ktorom je hodnota a_i zo všetkých vektorov synov (ďalších možných situácii) najväčšia. Automaticky pridelí dané ohodnotenie (celý vektor, nie iba jednu hodnotu z vektora!) aj otcovi. Priebeh je graficky znázornený na obrázku 4.1.



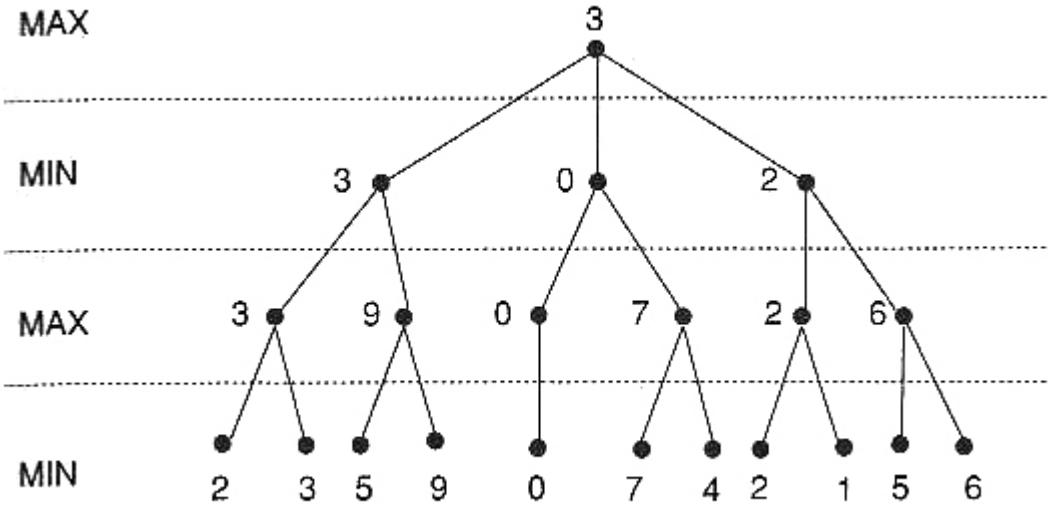
4.1. Na obrázku je vidieť algoritmus minimax pre troch hráčov, pričom pre jednoduchosť ukončujeme po troch ťahoch hru, aby sa nám strom zmestil do obrázky. Koncové stavy sa ohodnotia, hráč 3 si vyberie v každej situácii ten lepší, to isté hráč 2, a nakoniec hráč 1 si vyberie ten pre seba najlepší (v tomto prípade rozhoduje medzi 4 a -1)

4.1.1 Hry s nulovým súčtom

Špeciálnym prípadom sú hry, kde je súčet hodnôt každého vektora nula (Nullsummenspiel – „hra s nulovým súčtom“), teda v prípade dvoch hráčov výhra jedného je prehra pre druhého. V takom prípade je hodnota pre hráča A rovná negatívnej hodnote hráča B, teda miesto vektora (a,b) sa vracia iba hodnota a.

Potom algoritmus nevyberá pre každého hráča „svoju“ najvyššiu hodnotu, ale pre hráča, ktorý chce zistiť v koreni svoj ťah, vyberá syna s maximálnym ohodnotením, a pre súpera s minimálnym ohodnotením (pôjde tak, aby jeho protihráč utrpel čo najväčšiu porážku). Z toho aj pochádza názov Minimax, teda striedavé vyberanie min. a max. hodnoty.

Výber ťahov v tomto jednoduchšiom minimaxe je znázornený na obrázku 4.2.



Obr 4.2.: Ilustrácia algoritmu minimax pre dvoch hráčov, v koreni si vyberá hráč najlepšiu situáciu pre seba, na ďalšej úrovni si vyberá jeho protihráč najhorší ťah pre prvého hráča, v každej nastávajúcej situácii (na ďalšej úrovni) si zas prvý hráč vyberá pre seba najlepší ťah, atď.

4.1.2 Zložitosť algoritmu a jeho optimalizácia

Nech hru hrá x hráčov, ktorí sa striedajú, a nech po koniec hry bude mať každý t ťahov. Ďalej nech má každý hráč n možností, ako konať, keď sa dostane na rad. Je vidieť, že algoritmus začne vytvárať strom o hĺbke x*t s faktorom vetvenia n. Výsledná zložitosť bude teda O(n^(x*t)), čo nie je zrovna málo. Hlavne pre dlhotrvajúce hry, ako šach alebo dáma, by vznikal teda príliš veľký strom a algoritmus sa stáva prakticky nepoužiteľný.

Dá sa však optimalizovať. Spomeniem dva najznámejšie spôsoby, akým sa zmenšuje prehľadávaný strom:

- **α-β orezávanie** – ide o škrtanie vetví, o ktorých vieme dopredu, že nebudú vybraté, takže nemá zmyseľ ich ďalej prehľadávať.
- **Cut-off** – jednoduché zastavenie po dosiahnutí konštantnej hĺbky, alebo po uplynutí určitého času. Tu však treba dať pozor, lebo týmto spôsobom zastavíme hru na nekoncovom stave, a teda musíme mať heuristiku, ktorá dokáže odhadnúť ohodnotenie daného stavu.

4.2 Hry s neúplnými informáciami

V kartových hrách hráči nemajú informácie o tom, ktorá karta je kde umiestnená, a teda sa môžu iba domnievať, kde by asi ktorá mohla byť. Preto, predtým, ako pristúpim k riešeniu môjho problému, by som krátko predostrel všeobecne používané riešenia v situáciach, keď agent nemá dostupné všetky informácie, a musí sa rozhodovať iba na základe čiastočných vedomostí.

V tomto prípade patrí k agentovej úlohe často aj dopĺňanie chýbajúcich údajov z už dostupných. Niektoré fakty je možno si vydedukovať, iné možno iba tušiť. A práve tušenie bude v našom prípade kľúčové.

Pre každé tvrdenie *a*, ktoré by popisovalo agentove prostredie, zavedieme funkciu

P(a) -> <0,1> vyjadrujúcu našu vieru (je jedno či na základe doteraz nadobudnutých skúseností, alebo na základe nejakého pravidla) v to, že tvrdenie *a* je pravdivé.

4.2.1 Expect-minimax

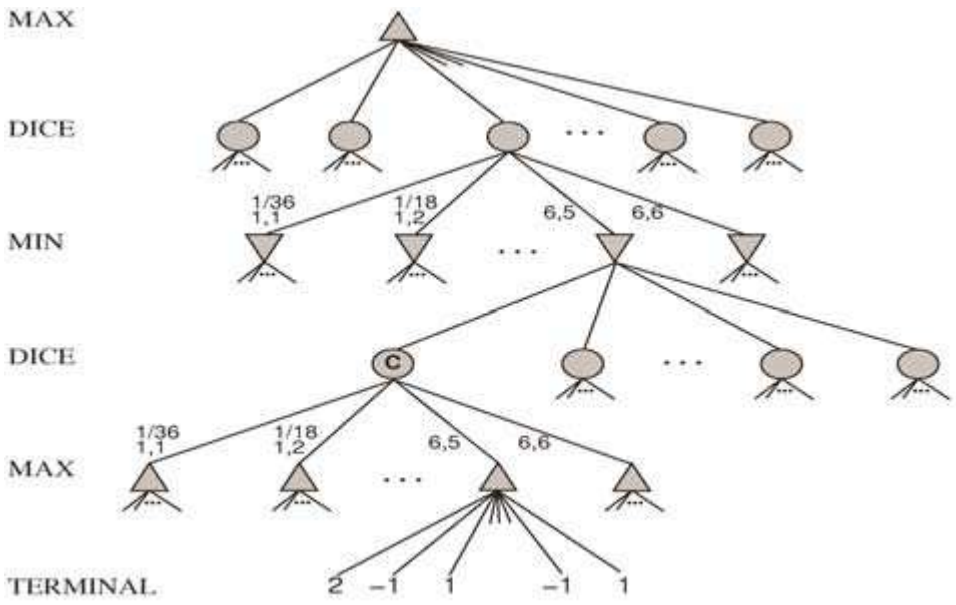
Algoritmus minimax sa dá použiť aj v prípade, keď nemáme k dispozícii úplné informácie. Avšak je potrebné, aby sme napriek nevedomosti o situácii vedeli presne určiť, čo sa všetko môže za akých okolností stať. Nevieme teda, v ktorej situácii sa nachádzame, ale vieme, aký je výber možných situácii. Pre každú z nich vieme, aké akcie má pre ne ktorý hráč k dispozícii. A takisto viem presne určiť, ktorá akcia ako danú situáciu zmení. Posledná vec, ktorú treba poznať, je pravdepodobnosť, že za danej situácie nastane nejaké ďalšia.

Ako teda vybrať najoptimálnejší ťah? Sledujme algoritmus minimaxu. Sme v situácii, keď máme na výber dve možnosti. Vieme, ako ovplyvní situáciu jeden, aj druhý ťah. Je na rade ďalší hráč.

Už vieme, ako sme situáciu ovplyvnili my, ale nevieme iné informácie o situácii. Na základe vplyvu nášho ťahu vytvoríme pre každú vetvu všetky možné situácie, ktoré môžu nastať, a k nim priradíme pravdepodobnosť(našu vieru), že po našom danom ťahu naozaj nastali.

Ďalej rozvinieme každú možnú situáciu podobne, ako v minimaxe, len s tým rozdielom, že každé ohodnotenie ťahu sa vynásobí pravdepodobnosťou, že daná situácia po danom ťahu nastane.

Na nasledujúcom obrázku bude algoritmus expect-minimax znázornený graficky:



Obr 4.3.: Ilustrácia algoritmu expect-minimax, krúžky značia možné ťahy hráča, trojuholníky možné situácie, do ktoré môžu vzniknúť (na začiatku je jedna jasná situácia). Tento priebeh simuluje hody dvoma kockami.

4.2.1.1 Zložitosť algoritmu Expect-minimax

Ako sme ukázali, ak hráči počas hry vystriedajú dokopy n ťahov, a pri každom ťahu majú b možností, časová zložitosť je O(b^(n)). V prípade, že pre každý ťah nastane ďalších m rôznych situácii, zložitosť sa vyšplhá na O(b^(n) * m^(n)), čo je pre trošku komplikovanejšie hry príliš veľké čislo.

Pre niektoré použitia sa určuje minimálna pravdepodobnosť situácie, aby sa vôbec rozvíjala a skúmala, čo čiastočne zredukuje prehľadávanie, avšak či to bude stačiť aj v našom prípade, uvidíme pri prípadnom pokuse o jeho implementáciu a použitie.

4.2.2 Axiómy pravdepodobností

Teraz by som uviedol tri základné axiómy pravdepodobností, zvané tiež Kolmogorovove [1] – 580.

Myslím, že všetky sú zrejme aj intuitívne, no vymenujem ich explicitne, aby som sa o ne mohol v niektorých prípadoch oprieť.

1. Pre každý výrok a platí $0 \leq P(a) \leq 1$

2. P (true) = 1 P (false) = 0
3. P(a \/ b) = P(a) + P(b) – P(a & b)

Pozorný čitateľ si iste všimne, že táto tretia axióma je použitá v dôkaze princípu inklúzie a exklúzie.

4.2.3 Náhodná premenná

Majme premennú X, a jej prípustnú množinu hodnôt $\{x_1, \ldots, x_n\}$. Ak je hodnot premennej X náhodne zvolená/nastavená, potom ju voláme *náhodná premenná.* Následne označíme $P(Xx_i)$ ako pravdepodobnosť, že premenná X nadobudne hodnotu x_i. Súčet hodnôt $P(Xx_j)$ pre všetky j od 1 po n je rovný 1.

4.2.4 Nepodmienená pravdepodobnosť

Začal som písať o pravdepodobnosti, a teraz by bolo na mieste napísať niečo o tom, ako s ňou pracovať. Doteraz som ju používal celkom jednoducho, a teda, že za každej situácie presne vieme, s čím môžeme ako isto rátať, ako to bolo pri *expect-minimaxe*.

Jednalo sa o tzv. *nepodmienenú pravdepodobnosť*, alebo aj *A-priori-pravdepodobnosť*. Jedná sa vlastne o naše presvedčenie, že výrok *a* platí, pri postrádaní akejkoľvek inej informácie. Teda nevieme o ničom, čo by na ne mohlo mať vplyv.

4.2.5 Podmienená pravdepodobnosť

Teraz si predstavme, že rozhodovací agent dostane nejakú evidenciu o udalosti (alebo nejakej skutočnosti), ktorá ovplyvňuje náhodnú premennú/tvrdenie (náhodná premenná *p* je premenná, ktorá nadobúda hodnoty z určitej množiny, a pre každú možnú hodnotu *h* existuje jedno tvrdenie A: p=h).

Napríklad náhodná premenná je výsledok hodu kockou *k*. Jej množina hodnôt je [1,..,6] a existuje 6 tvrdení k=I pre všetky i od 1 po 6. Nepodmienená pravdepodobnosť premennej je pre každé *i* P(k=i)=1/6.

Nech si agent všimne, že ťažisko kocky je posunuté bližšie k jednotke. Ak nemáme žiadne iné informácie o hádzajúcom, o povrchu, a o kocke, môžeme toto zistenie považovať za rozhodujúce. Nech agent vie, z predchádzajúcich skúseností, že ak je

ťažisko posunuté k nejakej stene, potom je pravdepodobnosť padnutia protiľahlej steny 5/16, zatiaľčo pre stenu, pri ktorej je ťažisko najbližšie, iba 1/18. Teda v našom prípade pravdepodobnosť hodu pre 6 bude 5/18, zatiaľčo pre 1 iba 1/18. Takto vytvorenú pravdepodobnosť voláme *podmienená* alebo aj *a-posteriori*.

Pre podmienenú pravdepodobnosť budeme používať notáciu **P(a|b)** a znamená to: „pravdepodobnosť, že platí *a*, ak poznáme iba *b*.“

Základný vzťah určujúci podmienenú pravdepodobnosť je

P(a|b) = P(a,b)/P(b)

4.2.5.1 Zopár užitočných vzorcov pre podmienenú pravdepodobnosť

Pred ďalším skúmaním vypíšem zopár základných vzorcov, ako s podmienenou pravdepodobnosťou zaobchádzať, ktoré môžeme neskôr využiť (ale tiež nemusíme):

1. P(a \/ not a) = P(a) + P(not a) – P(a & not a)
2. P(a) + P(not a) = 1

Ďalej tzv. *produktové pravidlo*, odvodené zo základnej definície podmienenej pravdepodobnosti:

P(a,b) = P(a|b) * P(b)

Z neho dostaneme rovnicu, ktorá sa používa v umelej inteligencii často používa, a volá sa *Bayesov vzorec*:

P(a|b) = P(b|a) * P(a) / P(b)

4.2.5.2 Kombinácie evidencí

Pri riešení problémov sa však nestretávame s takto jednoduchými pravidlami. Treba zopakovať, že P(a|b) = x neznamená to isté, ako implikácia b => (P(a)=x). Tvrdenie P(a|b) = x znamená, že P(a)=x za predpokladu, že nastala udalosť *b* a zároveň neevidujeme žiadnu inú okolnosť, ktorá by mohla ovplyvňovať pravdepodobnosť *a*. Ale v praxi sa stretávame skôr so situáciami, keď je tých evidencií ovplyvňujúcich nejaké tvrdenie oveľa viac.

Zatiaľčo vo výrokovej logike ak platí a implikuje b, a platí a, určite platí b(podľa pravidla modus ponens), a ak nastane máme v modeli výrok c implikuje not b, a dostaneme evidenciu o c, tak to znamená, že máme zlý model.

Lenže v modeloch s pravdepodobnosťami môže predpoklad *b* tvrdiť, že P(a) je 0.95, a *c* môže tvrdiť, že P(a) je 0.002, a obe evidencie môžu nastať súčasne. Prvý zápis tvrdí niečo v zmysle, že takmer určite platí *a* a druhý, že takmer určite neplatí.

A my potrebujeme vypočítať pravdepodobnosť P(a|b,c) – teda P(a) za predpokladu, že nastali udalosti *b* aj *c*.

Nech $\alpha = 1/P(b,c)$, potom z bayesovského pravidla po upravení dostaneme:

$$P(a|b,c) = \alpha\, P(b,c|a)*P(a)$$

Ak b a c nie su závislé, potom $P(b,c|a) = P(b|a) * P(c|a)$

Teda $P(a|b,c) = \alpha * P(b|a) * P(c|a) * P(a)$

K vyriešeniu tohto problému teda treba poznať P(b|a), P(c|a) a P(a), pričom každé P(x|y) sa dá pomocou bay. pravidla obrátiť na P(y|x). Takže z P(a|b) , P(a|c) vieme dostať P(b|a) , P(c|a), ak poznáme P(a), P(b) aj P(c).

5. Reálne konkrétne riešenia

V prvom rade treba zistiť, či už podobný problém nebol riešený. A nie iba v rovine teórií, a včeobecne používaných algoritmov vrámci umelej inteligencie, ale priamo sa pozrieť na kartovú hru podobnú šnapsu, a jej správanie.

Rozhodovacích agentov pre hry typu poker už bolo vytvorených nespočet, či už vrámci hracích prostredí, alebo dokonca trénerov a radcov, ktorí Vám radia, ako poťiahnuť. [8]

Šnaps je však zložitejšia hra ako poker. Preto rozhodovacieho agenta pre šnapsa prakticky nenájdete, ale podarilo sa mi nájsť pár programov s rozhodovacími agentami pre hru *Mariáš*. Presnejšie povedané pre zjednodušený mariáš. Mariáš je šnapsu veľmi podobná hra, a jej princíp je v podstate rovnaý. Rozhodol som sa teda programy vyskúšať. Nie pre všetky som ale rozbehal na mojom počítači. A nedopátral som sa k zdrojovým kódom, takže všetky moje závery budú iba domnienky, ktoré som nadobudol hraním, alebo čítaním diskusii (viď nižšie). Nechávam na čitateľa, aby si ich vyskúšal, ja napíšem moje pozorovanie pre tie, ktoré sa mi pošťastilo si zahrať.

Prvý program je ***Čtyřka*** – zjednodušená varianta pre štyroch hráčov. [4]

Tento progam má pravdepodobne preddefinované nejaké ťahy ako a za akých okolností, dvakrát mi chybne volil kontru, ale okrem toho to je hrateľná verzia.

Ďalší program je ***Pivoňkov mariáš.*** Je už vyše desať rokov starý, a tak som nenašiel download, iba diskusiu [5]. Avšak súdiac podľa diskusie veľmi kvalitný nebol, a pravdepodobne využíval minimax. Viaceri ľudia sa sťažovali, že niektoré ťahy mohol urobiť len ak videl ostatným do kariet.

Tretí program [6] mariáš ešte zjednodušil, a vytvoril verziu pre troch hráčov. Síce som ho nainštaloval, no .exe súbor sa mi nepodarilo spustiť, respektíve po spustení okamžite skončil, takže nemôžem posúdiť schopnosti agenta. Takisto som nenašiel žiadnu diskusiu.

Posledný program, ktorý som objavil, je ***Talon-marias***. Rýchlosť ťahov, tak ako pri *Čtyřke* bola vysoká (najdlhšie rozhodovanie trvalo cca niečo cez jednu sekundu!!!), takže predpokladám, že tam nebežali nejaké zložité algoritmy. Predpokladám tiež, že agent robil ťahy na základe nejakých preddefinovaných pravidiel agent urobil ťah. Na rozdiel od prvého programu tu sa mi nepodarilo vyhrať (možno preto, že sa nevyznám v pravidlách mariáša – hlavne v rôznych hláškach a spôsoboch hry). Mám pocit, že hoci to bola verzia len pre štyroch hráčov, mala v sebe množstvo rôznych mariášovských fintičiek, hlášok, doplnkových pravidiel, a na to všetko vedel agent reagovať. Skutočnú kvalitu by som však mohol posúdiť len za predpokladu, že by som dobre poznal túto hru, Dá sa nájsť na tomto mieste [7].

Závery o agentoch hrajúcich mariáš

Všetky tieto implementácie agentov sú, zdá sa, také že algoritmy pravdepodobne bežala radou if- rozhodovaní, ktoré sú od začiatku zakódované vývojárom. Jednou veľkou výhodou je rýchlosť, na druhej strane je to veľmi pracné pre programátora a návrhára vytvoriť také množstvo „odporúčaní“ ako kedy potiahnuť.

Na druhej strane, nikomu sa zrejme nepodarilo vytvoriť agenta, ktorý by sám naberal skúsenosti, učil sa z vlastných chýb, alebo aspoň sledovaním už skúsených hráčov.

6. Rozhodovací agent v hre *šnaps*

Cieľom tejto práce je zistiť, nakoľko inteligentne je agent schopný reagovať pri hre, a samotrejme tohto agenta aj naprogramovať. Mojou úlohou je navrhnúť agenta, ktorý má za úlohu hrať - a pokiaľ možno vyhrať - kartovú hru šnaps.

Predtým však, než sa začne agent rozhodovať, musí mať nejaké informácie o hre. Preto by som pristúpil najprv k riešeniu a návrhu znalostnej bázy(KB).

6.1. Znalostná báza (knowledge base - KB)

Asi to najdôležitejšie, čo musí agent vedieť, je kto a akú kartu volil na začiatku hry. Následne musí dostať karty z talóna, teda množinu 6 kariet. Kartu možno reprezentovať ako usporiadanú dvojicu (farba, znak).

Keď má agent tieto dve informácie, potrebuje si zaznamenávať priebeh hry. To znamená:

- Koľký štich práve prebieha
- Ktorý štich kto začínal
- Kto vyložil akú kartu

Na základe týchto informácii vie určiť v každom okamihu, kto je na ťahu, a ktorá karta je ešte v hre. Zároveň vie vypočítať, kto koľko bodov získal.

Aby však pri každej kontrole nebežal algoritmus, v KB budeme mať aj tabuľku s dvadsiatimi štyrmi políčkami (karta->{0,1}). V nej budeme zaznamenávať karty, ktoré boli vyložené. A naopak, v hociktorej chvíli sa môžeme pozrieť v čase $\Theta(1)$, či daná karta ešte je v hre a drží ju niekto z hráčov na ruke, alebo už bola vyložená.

6.1.1 Rolzoženie kariet

Hoci žiaden hráč nevidí do kariet iných hráčov, protihráči počas hry svojimi ťahmi prezradia, čo asi majú na ruke. Najideálnejšie by bolo držať pravdepodobnosti všetkých atomárnych rozložení. Avšak na začiatku, keď agent dostane šesť kariet, je počet

všetkých možných rozdelení kariet ostatným hráčom počet všetkých šesťprvkových podmnožín osemnásťprvkovej množiny, pre každú z nich počet šesťprvkových podmnožín dvanásťprvkovej množiny, pre každú dvojicu týchto podmnožín počet šesťprvkových podmnožín šesťprvkovej množiny(čo je jedna).

Teda (18 nad 6) * (12 nad 6) * (6nad 6) =

= (18! / (12!*6!)) * (12! / (6!* 6!)) * 1 =

= 18! / (6!) ^ 3

To je vyše 1,7 milóna. Po tom, ako agent dostanem šesť kariet, a predtým, ako sa rozdajú karty ostatným hráčom, bude pravdepodobnosť každej situácie práve 1/ (18! / (6!) ^ 3), ale pravdepodobnosti sa budú meniť skôr, ako sa začne redukovať počet atomárnych rozdaní.

Preto som sa rozhodol držať v malej tabuľke (3x24) pravdepodobnosti toho, že i-tu kartu má j-ty hráč.

Za každej situácie musí platiť, že súčet v každom riadku je rovný 1 (podľa 4.2.3).

	Hráč 1	**Hráč 1**	**Hráč 1**
Karta 1	1/3	1/3	1/3
Karta 2	1/3	1/3	1/3
Karta 2	1/3	1/3	1/3
...	...	...	...
Karta 24	1/3	1/3	1/3

Tab 6.1.1.: Ukážka tabuľky rozdelenia kariet na začiatku hry

Na začiatku je hodnota pre každú kartu a každého hráča 1/3 (pre každú kartu sú tri možnosti-hráči-komu môže byť rozdaná, pričom šance sú rovnomerne rozdelené) , ktoré hráč nemá na ruke, a 0 pre všetky, ktoré na ruke má. Počas hry sa budú hodnoty samozrejme meniť, avšak zo začiatku sa jedná o nepodmienené hodnoty. Tu nastal jeden z najväčších problémov mojej práce. A totiž, aké závislosti vytvoriť pre kombinácie pravdepobností. Za predpokladu, že jediné zmeny budú spôsobovať „škrtania“ kariet z rúk, a s tým spojená zmena na základe toho, že možných rôznych kariet je menej,

a hráč, ktorý mal danú kartu na ruke, má teraz o jednu pre agenta neznámu kartu menej, bude „upgrade“ pravdepodobností nasledovný:

Spočiatku je pre každú kartu a hráča hodnota 1/3, respektíve 6/18. Z možných 18 kariet má 6. Nech kartu i má hráč 1 (zistíme to tak, že ju vyloží). Potom pre hráčov 2 a 3 sa zmenia pravdepodobnosti pre kartu i na 0 (nemôžu ju mať), a pre všetky ostatné na 6/17, teda mierne stúpnu. Naopak pre hráča 1 bude P(má(i)) = 1.0, a pre všetky ostatné karty 5/17, teda mierne klesnú.

Ak týmto spôsobom vyrátame pravdepodobnosť pre každú možnú kombináciu kariet pre hráča 1, dostaneme (6/18 * 5/17 * 4/16 * 3/15 * 2/14 * 1/13), čo je

(6! * 12!) / 18!

A to je jedna ku (18 nad 6), teda jedna ku počet 6-prvkových podmnožín 18-prvkovej množiny. To zatiaľ sedí s tým, že každý z troch hráčov má na začiatku 6 kariet z 18 možných

	Hráč 1	**Hráč 1**	**Hráč 1**
Karta 1	5/17	6/17	6/17
Karta 2	1	0	0
Karta 2	5/17	6/17	6/17
…	…	…	…
Karta 24	5/17	6/17	6/17

Tab 6.1.2.: Ukážka tabuľky po zistení pozície karty 2 – má ju hráč 1

Ako som už spomínal, hodnoty sa budú počas hry meniť, a to hlavne v závislosti na reakciach jednotlivých hráčov (napríklad budeme predpokladať, že hráč pravdepodobne nemá nejakú kartu, lebo keby ju mal, tak by s ňou ťahal, keďže by to bolo preňho výhodné).

Má to jeden háčik, a to ten, že nevieme povedať, aký má vplyv takáto zmena pravdepodobností u jednej karty na pravdepodobnosti u ostatných kariet. To, čo sme urobili, je model, kde sa pravdepodobnosti budú meniť iba ak sa udeje niečo, čo má na tú ktorú kartu konkrétny vplyv, pravdepodobnosti pre ostatné karty sa meniť nebudú.

Napríklad v prípade, že z hry hráča x je veľmi pravdepodobné, že nemá nízku kartu (dolník a deväť hociktorej farby), zmenšíme pravdepodobnosť pre každú z nich nezávisle od seba (okrem tých, kde je všetko jasné – teda tam, kde je teraz 0 alebo 1) na nejakú malú hodnotu, ale pravdepodobnosti pre ostatné karty nezvýšime, hoci by mali byť vyššie, keby sme chceli byť presní.

Algoritmus 6.1.1 Tento jednoduchý algoritmus som navrhol na zmenu pravdepodobností.

Nech *k* je ľubovoľná karta, nech p_1 = P(U |má_kartu(1,k)), p_2 = P(U |má_kartu(2,k)) , p_3 = P(U |má_kartu(3,k)), a nech nastane udalosť *U* taká, že:

P(U |má_kartu(1,k)) = 0.05

Ako sme už uviedli, súčet p1, p2 a p3 musí byť rovný 1. Z udalosti U nastavíme p1 na 0.05. P2 a p3 budeme meniť v pôvodnom pomere tak, aby stále platilo p1 + p2 + p3 = 1. Predpokladajme, že pôvodne boli všetky hodnoty 1/3, teda p2 a p3 boli v pomere 1:1. Potom nové hodnoty p3 a p2 budú obe rovné zhodne 0.475.

Popíšme to teda všeobecnejšie. Majme kartu *k,* hodnoty p_1 , p_2 , p_3 , máme pravidlo P(U |má_kartu(1,k)) = $\underline{p_1}$ a potrebujeme vyrátať $\underline{p_2}$ a $\underline{p_3}$

Potom platí:

- $\underline{p_2} = (1 - \underline{p_1}) * p_2 / (p_2 + p_3)$
- $\underline{p_3} = (1 - \underline{p_1}) * p_3 / (p_2 + p_3)$

6.1.1.1 Kombinácie viac evidencií

Počas hry sa môže stať, a takmer určite sa stane, že dve a viac rôznych udalostí bude meniť pravdepodobnosť rozdelenia jednej karty. Nastáva teda ďalší problém. A to v prípade, že už je pravdepodobnosť ovplyvnená nejakou udalosťou, a nastane druhá udalosť, ktorá ju ovplyvňuje. Doteraz som to riešil tak, že pôvodná udalosť sa do úvahy nebrala vôbec, teda ak príde evidencia C, pričom platí P(má_kartu(h,k) | C) = 0.45, tak pravdepodobnosť, že hráč *h* má kartu *k* bude 0.45, bez ohľadu na to, či sa jedná o prvú zmenu (a do udalosti C platilo P(má_kartu(h,k)) je rovno 1/3), alebo či niekedy predtým nastala nejaká udalosť A, ktorá stanovila P(má_kartu(h,k)) na 0.62.

Najideálnejšie by bolo použiť vzorce pre kombinácie podmienených pravdepodobností (viď 4.2.5.2). Ale k tomu je treba poznať hodnoty
P(C |má_kartu(h,k)) pre všetky možné udalosti C, ktoré ovplyvňujú pravdepodobnosť výroku má_kartu(h,k). Získať tieto hodnoty však nie je vôbec jednoduché. Vlastne sa to dá iba sledovaním hry a zaznamenávaním štatistiky.

Samozrejme, nie každá karta má výrazný vplyv na každú udalosť, tým pádom nebude možné z každej udalosti odhadovať rozlosovanie každej karty. Preto je potrebné najprv vôbec určiť, aké vzťahy medzi tým, aké má nejaký hráč na ruke, a ktorou kartou ťahá, sú skutočne relevantné. To je ďalší problém. Keby sme sledovali závislosti medzi kartami, kto

Otázkou ostáva, ako určiť presné hodnoty. Buď ich môžeme zadať pri tvorbe agenta, alebo nechať agenta, nech počas svojho „života“ (hrania) zbiera skúsenosti a zisťuje, v koľkých prípadoch za akých okolností ako ostatní hráči ťahajú.

Tu však zachádzam do spôsobu samotného rozhodovania, a update pravdepodobností bude teda závisieť od riešenia agenta.

Zatiaľ pristúpim k najjednoduchšiemu spôsobu, ako pravdepodobnost meniť, a ukazuje to algoritmus 6.1.2.

Ak nastane udalosť, ktorá tvrdí, že P(má_kartu(h, k)) = *x*, tak pravdepodobnosť nastavím nasledovne: nech doterajšia P(má_kartu(h, k)) = *p*

Potom rozlišujem 3 možnosti, podľa ktorého vyberieme jednu z týchto hodnôt – na ďalšej strane

Algoritmus 6.1.2:

1. p < 1/3 =>
   I. ak (x > 1/3) potom P(má_kartu(h, k)) = x
   II. ak (x <= 1/3) potom P(má_kartu(h, k)) = MIN(x, p)
2. p = 1/3 => P(má_kartu(h, k)) = x
3. p > 1/3 =>
   I. ak (x < 1/3) potom P(má_kartu(h, k)) = x
   II. ak (x >= 1/3) potom P(má_kartu(h, k)) = MAX(x, p)

Inými slovami, ak ide o udalosť, ktorá zvyšuje pravdepodobnosť nad 1/3, tak nastavím najvyššiu pravdepodobnosť, ktorá je predpokladaná, naopak, ak pod 1/3, tak najmenšiu, aká môže byť z nejakej evidencie predpokladaná.

Pri tomto prístupe sú natvrdo v logike Knowledge-base zakomponované konktrétne hodnoty, a agent žiadnu skúsenosť nezbiera.

6.1.2 Fakty

Medzi ďalšie informácie patria fakty, ktoré vyplývajú z hry, a ktoré sa ťažko doterajším spôsobom zaznamenávajú. Napríklad tvrdenie „hráč 1 má určite aspoň 1 zelenú kartu“ alebo „hráč 3 má aspoň 2 červené karty vyššie ako dolník.“

V tabuľke 5.1.2 môžeme celkom jednoducho vyjadriť tvrdenie „hráč 3 nemá ani jednu červenú kartu“ tým, že priradíme každému políčku hodnotu nula pre všetky červené karty a hráča 3.

Avšak ako zmeniť pravdepodobnosti, ak viem, že v hre sú ešte 2 zelené karty, a hráč 1 má jednu z nich? Ak by v hre ostávala jediná karta, ktorá prichádza do úvahy, rovno jej priradíme hodnotu 1. Ak ich je viac, musíme situácia sa komplikuje. Určite budú pravdepodobnosti pre obe karty a hráča 1 vyššie, ako 1/3. Najprv treba rozlíšiť dve situácie. Ak existuje nejaká (aspoň jedna) zelená karta *kz*, pre ktorú platí P(má_kartu(h_1, kz)) = 1, potom netreba nič robiť (lebo vieme, že tá zelená karta, ktorú hráč 1 určite má, je *zk*). Ak však taká neexistuje, potom musíme pravdepodobnosti všetkých zelených kariet pre hráča 1 zdvihnúť.

Pravdepodobnosti nemôžme zdvihnúť rovnomerne. Vezmime nejaký príklad. Nech sme v situácii, keď ostávajú v hre (na rukách protihráčov) ešte dve karty zelenej farby (nech sú to eso a dolník), a kde platí:

P(má_kartu(h_2, zelené eso)) = 0.75

P(má_kartu(h_1, zelené eso)) = 0.125

P(má_kartu(h_1, zelený dolník)) = 1/3

Nech nastane udalosť, podľa ktorej vieme, že hráč 1 má zelenú kartu. Mohli by sme jednoducho povedať, že pravdepodobnosť toho, že má hráč 1 zeleného dolníka, bude 0.5, a že bude mať zelené eso, bude tiež 0.5. To by ale spôsobilo, že P(má_kartu(h_2, zelené eso)) sa zmení na 6/14 (0.5 * 0.75 / 0.875 podľa algoritmu 5.1.1), čo samozrejme pokazí všetky predchádzajúce evidencie, vďaka ktorým sme dospeli k hodnote 0.75

Môj nápad teda je zdvíhať pravdepodobnosti pre každú zelenú kartu a hráča 1 rôzne. V prípade, že je hodnota veľmi malá (blízka 0), tak musia byť pravdepodobnosti pre ostatných hráčov veľké (v našom prípade zelené eso), a preto by mala nastať čo najmenšia zmena.

Ak je pravdepodobnosť veľmi veľká (blízka 1), nemôžeme ju už príliš dvíhať – nemôžeme prekročiť hodnotu 1 (a nemala by ju ani nikdy dosiahnuť).

Najväčšiu zmenu by sme teda očakávali pre pravdepodobnosti v hodnote 1/3 až 2/3. Môžeme dokonca povedať, maximálne zvýšenie nastane pri hodnote 0.5.

Preto som sa rozhodol veľkosť zmeny určiť kvadratickou rovnicou s lokálnym maximom v bode 0.5 a s funkčnými hodnotami 0 pre vstupné hodnoty 0 a 1. Tieto podmienky dokonale spĺňa algoritmus 6.1.3, ktorý som pre tento problém zaviedol.

Algoritmus 6.1.3

Nech *p* je pôvodná pravdepodobnosť, potom novú pravdepodobnosť *pn* vyrátame takto:

pn = p*(1+p*(1-p))

Teda p sa zvýši o *p*(1-p)* , čo je kvadratická rovnica s lokálnym maximom pre *p* rovné 0.5, a s nulovými bodmi pre vstupy 0 a 1.

Musia však platiť 2 podmienky:

I. súčet pravdepodobností musí byť najmenej 1

II. aspoň jedna pravdepodobnosť musí byť väčšia ako 1/3

Ak tieto nie su splnené, algoritmus iterujeme dovtedy, kým ich splníme.

Vráťme sa k príkladu s dvoma zelenými kartami. Vezmime najprv zeleného dolníka:

P_{zd} = P(má_kartu(h_1, zelený dolník)) = 1/3

Potom

$\underline{\mathbf{P_{zd}}}$ =1/3 * (1 + 1/3*(1 - 1/3)) =

1/3 * (1 + 1/3*(2/3)) =

1/3 * (1 + 2/9) =

1/3 * 11/9 = 11/ 27

Teraz sa pozrime na zelené eso:

P_{ze} = P(má_kartu(h_1, zelené eso)) = 0.125

Potom

$\underline{\mathbf{P_{ze}}}$ =0.125* (1 + 0.125*(1 - 0.125)) =

0.125* (1 + 0.125*(0.875)) =

0.125* (1 + 0,109375) =

1/3 * 1,109375= 0,138671875

Druhá podmienka je síce splnená, ale prvá ešte nie (súčet je cca 0.54). Pre obe karty teda algoritmus niekoľkokrát zopakujeme, kým nebude ich súčet aspoň 1. Po piatom prebehnutí dostaneme následovné hodnoty:

$$P(má\_kartu(h_1, zelený\ dolník)) = 0.9133$$

$$P(má\_kartu(h_1, zelené\ eso)) = 0.2333$$

Pre hráča 2 ostane pravdepodobnosť, že má zelené eso, 0.66.

Stále to však nestačí jednoducho zmeniť pravdepodobnosti. Predstavme si predchádzajúci prípad, počiatočné hodnoty sú 1/3 pre obe zelené karty, zmeníme ich podľa vzorca na 0.5058, prejde jeden štich, a hráč 3 vytiahne zelené eso. Jeho pravdepodobnosť pre hráčov 1 a 2 sa zmení na 0, pravdepodobnosť pre poslednú zelenú kartu v hre a hráča 1 ostane 0.5058, hoci už je jasné, že ju musí mať.

Je teda vidno, že úpravy pravdepodobností pri niektorých faktoch o hre, ktoré sa hráč dozvedá, sú len bočným efektom, ale všetky tieto fakty sú defacto tvrdenia, ktoré si musí agent zaznamenávať, a pri ťahaniach hráčov ich prípadne uplatnovať. V predchádzajúcom prípade po vytiahnutí zeleného esa hráčom 3 by mal agent automaticky priradiť pravdepodobnosti poslednej zelenej karty pre hráča 1 na 1.00.

Naopak, môže sa stať, že takto zvýšime pravdepodobnosti, a hneď v ďalšom štichu vyjde hráč 1 zeleným esom. Teda už vieme, že mal aspoň jednu zelenú kartu, a práve ňou vyšiel. Aby sme boli korektní, mali by sme hodnoty pre zvyšné karty znížiť na pôvodnú hodnotu.

Musíme teda zabezpečiť, aby si každé toto tvrdenie „pamätalo“ ako a menilo pravdepodobnosti, v ktorých prípadoch ich má vrátiť do pôvodného stavu. Tu nastáva ďalší problém – čo ak nejaká iná udalosť zmenila medzičasom pravdepodobnosť, a teraz ju my budeme chcieť vrátiť naspäť – tým pádom zrušíme pôsobenie tej udalosti, ktorá nastala medzičasom. V tomto prípade uplatníme pre jednoduchosť štýl výberu, ako v algoritme 6.1.1.

6.1.2.1 Vylučovanie zbytočných viet z KB

Ako bolo spomenuté už niekoľkokrát, v priebehu hry môže nastať situácia, že niektoré fakty budeme môcť vyjadriť priamo v pravdepodobnostnej tabuľke rozdelenia kariet nulami alebo jednotkami. V takom prípade bude rozumné tieto tvrdenia z našej KB mazať, aby sme celé rozhodovanie urýchlili.

Toto mazanie bude prebiehať vždy po každom vyložení karty.

6.1.2.2 Operácie na tvrdeniach

Nakoniec by som zhrnul všetky operácie, ktoré budeme potrebovať na tvrdeniach vykonávať:

- kontrola platnosti (v prípadoch, že budeme chcieť simulovať situácie, tak budeme musieť kontrolovať napríklad to, či je daná situácia vôbec možná, aby neodporovala žiadnemu tvrdeniu z KB)
- vyplyv na pravdepodobnosti (táto operácia prebehne pravdepodobne pri zistení faktu, a následne pri každom ťahu bude kontrolovať, či sa pravdepodobnosť nemá opäť zmeniť)
- koniec platnosti (možno zavádzajúci názov, no treba po každom ťahu kontrolovať, či už nie je tvrdenie zbytočné)

6.1.3 Spoluhráči

Posledná vec, ktorú potrebuje agent evidovať, je to, kto je komu spoluhráč. K tomuto urobíme tabuľku 4x4, a hodnotami -1 (protihráči), 0 (ešte to nie je jasné) a 1 (spoluhráči). Počas hry sa budú tieto hodnoty meniť.

Myslím, že tu netreba nič vysvetlovať, je to celkom jednoduché, ako bolo napísané v pravidlách, ten, kto má zvolenú kartu, je s tým, kto zvolil (ak ju má ten istý hráč, je sám), ostatní sú spolu proti ním. Ide len o to zistiť, kto má, alebo mal (ak ňou potiahol) túto kartu.

Najideálnejšia situácia nastane, ak ju má na ruke agent – potom vie, že je s tým, kto volil, už na začiatku hry, a že ostatní hráči sú proti nim, a spolu.

6.3 Moje riešenia a výsledky

Na základe predošlých zistení a pozorovaní som sa pokúsil o moje riešenie. Pokúsil som sa zostrojiť takého agenta sám. Vyskúšal som niekoľko možností, postupne uvediem.

6.3.1 Podvodník

Ako názov napovedá, jedná sa o agenta, ktorý má prístup ku všetkým informáciam o rozložení kariet, a preto použije jednoduchý minimax. Dokonca môžeme použiť miesto vektora pre štyroch hráčov iba jednu hodnotu ako pri minimaxe pre dvoch hráčov, a pre agentovho spoluhráča budeme v simulácii uplatnovať MAX, a pre protihráča MIN.

Jedná sa o jeden z najjednoduchších algoritmov vrámci rozhodovacej logiky, takže jeho použitie nie je veľmi zaujímavé, poslúži iba ako ukážka minimaxu v praxi, samozrejme s použitím orezávacích techník.

6.3.2 Použitie expect-minimaxu

Pre toto riešenie by bolo potrebné vytvoriť 1.7 milióna (viď 6.1.1) možných rozdelení kariet, a následne by sme už neprichádzali do styku s neistotou, a pre každú situáciu by sme mohli spustiť jednoduchý minimax. No ak môžem pri orezaní minimaxu hovoriť o úspechu, pokiaľ sa rozhodne v priemere do 15 sekúnd, skúsme si predstaviť, ako by vyzeralo rozhodovanie pre 1.7 milióna možných situácii, pričom pre každú situáciu by algoritmus bežal v priemere tých 15 sekúnd (čo je až príliš optimistický predpoklad). Je to 295 dní, možno ešte o niečo dlhšie.

Aby toho nebolo málo, treba pripomenúť, že zmeny pravdepodobností sú nepresné, no expect-minimax potrebuje celkom presné čísla, nie hrubé odhady. Ak by sme mali pridať čas pre nejaký algoritmus, ktorý by ich vypočítal, čo sa však ani nedá (a tým pádom klesá správnosť rozhodnutia expect-minimaxu), určite by sme zaplnili celý rok čakaním.

Teda zhrnutie je, expect-minimax nie je použiteľný, a preto som ho neimplementoval.

6.3.3 Jednoduché pravidlá

Zatiaľ som sa dopracoval sem, a teda nevykročím z tieňov mariášových agentov. Pre rozhodovanie vymyslíme niekoľko (veľa) pravidiel pre agenta, za akej situácie má ako potiahnuť. Následne ich premietneme do softvéru pomocou if-ov, cyklov, a case-vetvení.

Z našej KB budeme potrebovať predovšetkým histórie ťahov, a potom nuly a jednotky z tabuľky rozdelenia kariet (teda tvrdenia o tom, kto ktorú kartu určite má, alebo nemá). A zijdu sa aj ostatné tvrdeni, ktoré nám povedia niečo o situácii v hre.

Avšak navrhoval by som tu pridať aj závislosť rozhodovania od pravdepodobností. Takto môžeme napríklad nastaviť agentovi mieru riskovania. Ako už bolo spomenuté, jednoduché zmeny v pravdepodobnostiach nie su veľmi presné, avšak to ani nepotrebujeme. Kľudne by sme mohli rozlišovať 3 stupne – menej ako 1/3, 1/3 až 2/3, viac, ako 2/3. Alebo sa rozdeliť pravdepodobnosti do piatich dielikov po 0.2. Podľa toho môžeme potom vytvoriť pravidlá napr. „ak má hráč danú kartu aspon na 75%, potom niečo …"

Pre tieto pravidlá však treba zaviesť tzv. priority. Totiž, môže sa stať, že viacero rôznych predpokladov bude splnených, a tým pádom dostaneme viacero kariet, ktoré sú vhodné pre najlepší ťah. Ale vrámci implementácie to nie je problém, môžeme jednoducho predpoklady kontrolovať postupne, a keď budú pre jedno pravidlo všetky splnené, kontrolovanie prerušíme a potiahneme kartou.

6.3.2 Učenie od skúseného hráča

Toto je riešenie, ktoré už som vyskúšal, a zatiaľ všetko napovedá tomu, že aj bude fungovať. Tak aby som bol konkrétnejší, navrhneme neurónovú sieť, ktorá bude sledovať karty hráča (teda bude mať informácie ako on), a ku každému jeho ťahu si pozrie, ako hráč potiahol. Neurónová sieť má schopnosť asociovať si vstupy (situáciu hráča) a výstupy (karta, ktorou hráč potiahol), pričom po viacerych pozorovaniach vie aj určiť, ktoré časti vstupu majú reálny vply na výstup, a ktoré sú nepodstatné, prípadne ktoré kombinácie vstupných atributov majú aký vplyv na výstup.

Doteraz sa mi podarilo naučiť sieť urobiť prvý ťah. Ako vstupy (24 neurónov – pre každú kartu jeden) som zakódoval do neurónových signálov 0 a 1 – podľa toho, či danú kartu mám, alebo nemám. Asi po 60 hrách, ktoré sieť „videla," bola schopná sama urobiť správne prvý krok. Stačila mi na to sieť s jednou skrytou vrstvou o šírke 3x väčšej, ako vstupná vrstva (teda 24 * 3).

6.4 Plány do budúcnosti

A ako by som chcel pokračovať? Už som napísal, že som zatiaľ nevykročil z tieňa ostatných agentov pre mariáš. No dúfam, že moje riešenie bude mať pokračovanie aj ďalej, a nezastaví sa touto prezentáciou. Teray by som popísal, v čom vidím príležitosť zlepšenia a vývoja agenta.

6.4.1 Skúsenosti s odhadovaním kariet

Prvá vec, ktorá by už agenta urobila interaktívnym, by bola schopnosť evidovať rôzne situácie, a následne, ako v nich ostatní ťahajú. Na to by bolo potrebné, aby sme vytvorili niekoľko druhov situácii a aby agent počas hry vždy vedel, ktorá situácia je aktuálna. Napríklad by zistil, že za určitej situácie hráč ťahá často kartou k. Po veľa hrách by sme mali teda približnú štatistiku, že situácia *s* vedie k ťahu kartou *k* a to by nebolo nič iné, ako P(k|s). A ako sme videli, z toho sa dá získať P(s|k), a teda až raz hráč potiahne kartou *k*, budeme vedieť, ako je pravdepodobná situácia *s.*

No nebude to až také jednoduché, keďže situácia *s* môže byť kombináciou viacerých situácií. Ako sa s týmto vysporiadať, by som rád vyriešil v budúcnosti.

6.4.2 Samoučiaca sa sieť

Inými slovami vytvoriť agenta, ktorý pozná iba pravidlá hry, „hodiť ho do hlbokej vody“ a nechať hrať. A nech sa po čase sám naučí na vlastných chybách.

Je to podľa mňa veľmi optimistický cieľ, a zatiaľ nerátam s tým, že by sa to podarilo, nakoľko dobrú predstavu, ako zostrojiť takú štruktúru siete. Každopádne, je to jeden odvážny nápad, a ak sa tadiaľto podarí v budúcnosti nájsť cestu, bude to pre mňa osobne úspech.

A toto je spôsob, ako vytvoriť lepšieho hráča, než sám strojca. Doteraz skutočne agent iba opakoval od stvoriteľa, v poslednom spôsobe mohol od veľa rôznych hráčov ich štýly a taktiky, no nikdy nemal šancu ich prekonať. Agent, ktorý by sa učil na vlastných chybách by to mohol po čase zvládnuť, keďže celý svoj život bude vnímať iba karty, a nič iné.

Možno raz takýto agent bude nám určovať taktiky a pravidlá, podľa ktorých pri hre ťahať.

7. Zhrnutie výsledkov a diskusia

Na záver by som rád zrekapituloval celú prácu. V zopár bodoch by som zhrnul, čo všetko som urobil, či už na papieri ako myšlienky, alebo na počítači:

- Preštudoval som niekoľko spôsobov a algoritmov, akými sa vo všeobecnosti riešia podobné problémy [1]. Potom som pohľadal niekoľko podobných implementovaných hier, vyskúšal ich funkčnosť a prečítal pár diskusií [4-7].

- Pokúsil som sám navrhnúť niekoľko riešení, a implementovať vlastného agenta, a to viacerými spôsobmi. Zistil som, že problém je značne zložitý a nebude jednoduché zostrojiť inteligentnejšie riešenie, ako už existujúce. Napriek tomu si myslím, že sa s mojimi riešeniami môžem posunúť ďalej, ako som spomínal v podkapitole 6.4. A rozhodne by som chcel vo vývoji a posúvaní hraníc možností pokračovať.

- Okrem samotných agentov som implementoval testovacie prostredie, kde má programátor možnosť vidieť karty všetkých hráčov a sledovať tak správanie sa agentov. K tomu som implementoval kompletnú logiku hry, kontroly, komunikačné rozhrania, a teda je jednoducho použiteľná v hociktorom programe.

- A nakoniec je implementovaná celá KB a základné správanie agenta s možnosťou nastavenia zdržania – minimálny čas rozhodovania. To pre prípad, aby náhodou pri príliš rýchlom agentovi dokázal hráč sediaci za počítačom sledovať, čo sa deje. Štandartne je toto zdržanie nastavené na päť sekúnd.

Celá implementácia je naprogramovaná v jazyku java, čo zaručuje nezávislosť od platformy. Na druhej strane môže táto skutočnosť spôsobiť pri použití zložitej neurónovej siete značné zbrzdenie.

8. Použitá literatúra

[1] Künstliche Intelligenz – Stuart Russel, Peter Norvig

[2] http://www.casinotropez.com/de/online_card_games_history.html

[3] http://www.kostenlose-kartenspiele.de/Geschichte.htm

[4] http://ctyrka.vrana.cz/

[5] http://www.zzzz.cz/marias/odpovedi.php?koren=1740&dummy=35

[6] http://marias.webz.cz/

[7] http://marias.ik.cz

[8] http://www.pokerbot.org/