UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA APLIKOVANEJ INFORMATIKY

Úvod do Umelej Inteligencie
Cvičenie 6 - MiniMax

Október 26, 2022

6. MINIMAX A ALPHA-BETA OREZÁVANIE

Budeme programovať agenta, ktorý bude hrať piškvorky pomocou algoritmu MiniMax a Alpha-Beta orezávania.

Program:

Program (aj jeho popis tu v zadaní) je kompletne prevzatý z prvého cvičenia.
V súbore *games.py* je definovaná trieda *Game* a od nej odvodené *TicTacToe* a *Gomoku*, ktoré nemusíte vôbec meniť. V *players_minimax.py* sú funkcie-hráči a hlavný program. Hľadajte klasické *YOUR CODE GOES HERE* a/alebo *EXAMPLES*.

Vašou úlohou bude dorobiť funckiu **choose_move(game, state)** v triedach *MinimaxPlayer* a *AlphaBetaPlayer* tak, aby bol použitý MiniMax algoritmus a príp. Alpha-Beta orezávanie. Tieto funkcie dostanú dva argumenty:

- **game** - inštancia triedy **Game**, v ktorej sú implementované všetky funkcie týkajúce sa samotnej hry (viď nižšie).
- **state** - aktuálny stav hry (t.j. stav hracej plochy, kto je na ťahu, ...), na ktorý je potrebné odpovedať najlepšou možnou akciou (najlepším možným ťahom). Stav nemusíte parsovať, iba ho použiť ako argument do ďalších funkcií (viď nižšie).

Výstupom z funkcie je akcia (t. j. ťah, ktorý sa má vykonať) - teda číslo zo zoznamu možných akcií **game.actions(state)** pre daný stav **state**.

Niektoré funkcie triedy **Game**, ktoré môžu byť nápomocné:

- **game.player_at_turn(state)** - zistí, kto je v stave **state** na ťahu ('X' alebo 'O')
- **game.other_player(player)** - ako argument dostane písmenko hráča, vráti písmenko druhého hráča
- **game.board_in_state(state)** - vráti stav hracej plochy *(list(list(char)))* v stave **state**
- **game.w** - šírka hracej plochy
- **game.h** - výška hracej plochy
- **game.k** - koľko znakov v rade musím mať, aby som vyhral

- **game.actions(state)** - vráti zoznam *(list(int))* možných akcií (platných ťahov) zo stavu **state**
- **game.state_after_move(state, a)** - vráti nový stav, ktorý vznikne vykonaním akcie **a** zo stavu **state**
- **game.is_terminal(state)** - otestuje, či je stav **state** terminálny, t.j. niekto vyhral alebo je remíza
- **game.utility(state, player)** - vráti 'utilitu' stavu **state** pre hráča **player** : 1 ak vyhral, -1 ak prehral, 0 inak. Majte na pamäti, že jednotlivé utility sú známe až na konci hry, t.j. nemá zmysel sa pýtať na utilitu stavu **state**, pokiaľ to nie je terminálny stav.

Jednoduché ukážky týchto funkcií sú zahrnuté aj v kóde.

Na konci kódu sa spúšťajú hry - odkomentujte čo budete potrebovať. Pri debuggingu odporúčam hrať aj MiniMax vs. človek.
Ako zistiť že to funguje? Piškvorky sú tzv. m,n,k hra, okolo ktorej prebehlo celkom dosť teoretických výskumov. Nás z nich bude zaujímať najmä to, že na hracej ploche 3x3 hráč, ktorý hrá optimálne, nemôže prehrať. Ak je teda váš agent spravený správne, tak vždy buď vyhrá alebo remízuje, pričom nezáleží na tom či začína hru alebo ide druhý. Ak hrá proti druhému optimálnemu hráčovi (MiniMax vs. MiniMax), tak vždy remízuje.

Úloha 1 (1b): Do triedy MinimaxPlayer naprogramujte MiniMax algoritmus. Pseudokód nájdete napr. v prednáške alebo TU. Použite rekurziu!
Úloha 2 (0.4b): Ako ste si mohli všimnúť, MiniMax je dosť pomalý, najmä keď začína hru. Riešením je orezať (t.j. neprehľadávať) tie vetvy, ktoré určite neprinesú lepšie riešenie - tzv. Alpha-Beta orezávanie. Do triedy AlphaBetaPlayer naprogramujte MiniMax s využitím Alpha-Beta orezávania. (AIMA pseudokód)
Úloha 3 (0.6b): Štandardné piškvorky 3x3 sú pomerne jednoduché - celý prehľadávací priestor má "iba" $3^9 = 19683$ stavov, takže si môžeme dovoliť prehľadávať strom možností až po listy. Altermatívna verzia s hracím plánom 15x15 - Gomoku - má až $3^{255} \approx 4.6 \times 10^{121}$ stavov. Nemôžeme teda prejsť celý prehľadávací strom, musíme sa zastaviť v určitej hĺbke. Daný stav potom nebude terminálny - nebudeme vedieť jeho reálnu utilitu, a tak ju musíme odhadnúť pomocou vyhodnocovacej funkcie. Tá ohodnotí daný stav podľa toho, či je "sľubný" alebo nie. Neexistuje žiadna zaručene dobrá evaluačná funkcia - musíte si vymyslieť vlastnú (tu môžete použiť aj váš už existujúci kód z prvého cvičenia). V tejto časti nepotrebujete vymyslieť najlepšiu vyhodnocovaciu funkciu, stačí niečo, čo bude rozumné a bude dávať zmysel.
Do triedy AlphabetaEvalPlayer teda doprogramujte orezanie stromu v určitej hĺbke a evaluačnú funkciu. Takýto agent by mal následne vedieť hrať Gomoku celkom rozumne.