Programátorská dokumentace: Piškvorky s překážkami

Jedná se o klasickou hru piškvorky s lehkou modifikací: uživatel bude hrát piškvorky proti počítači, na hracím poli, kde některá políčka budou zablokována.

Hrací pole má velikost 15\*15. Uživatel bude hrát s kolečky a počítač s křížky.

Algorytmus:

„Počítač“ bude hledat tahy ve stavovém prostoru pomocí mini-max algorytmu, vylepšeným alfa-beta prořezáváním, kde vybírá tah ze sousedních políček.

Proměné

* board\_size – integer určující velikost pole (doporučuju mezi 15 a 35)
* board – 2D pole charakterů tj. Hrací plocha, čísluje se od 1 do board\_size (ve skutečnosti má okraje z „#“, takže je to pole od 0 do board\_size+1),hrací pole je číslované jako matice (od 0 do board\_size+1), na každém políčku je právě jedno z:
  + char1
  + char2
  + „#“ – překážka
  + „\_“ – prázdné políčko

111111

123456789012345

1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 \_\_\_\_\_\_\_\_\_\_\_\_\_#\_

4 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6 \_\_\_\_\_\_\_\_\_\_\_\_#\_\_

7 \_\_\_\_\_XO\_\_\_\_\_\_\_\_

8 \_\_\_\_\_OXO\_\_\_\_\_\_\_

9 \_\_\_\_\_\_\_X#\_\_\_\_\_\_

10 \_\_\_\_\_\_\_\_\_\_\_\_\_#\_

11 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

12 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

13 \_\_\_\_\_#\_\_\_\_\_#\_\_\_

14 \_\_\_\_#\_\_\_\_\_\_\_\_\_\_

15 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* used – pole souřadnic, obsahující políčka obsahující kroužky nebo křížky
* active – pole souřadnic políček, které jsou prázdné a zároveň sousedí s obsazeným políčkem
* char1\_turn– True pokud zacina člověk neboli char1
* char1, char2 – klasická volba je char1=“O“, char2=“X“
* terminal\_score – nastaveno na 900000, pokud score>=terminal score (vyhrál char2) nebo pokud terminal\_score=< score (vyhrál char1) neboli podmínka na konec hry je abs(score)>=terminal\_score a vyhrát může jen ten kdo je na tahu

Utlity

utility –funkce jenž vypočítá a vrátí score aktuální pozice na hrací desce pomocí board a used. Skóre udává jak je daná situace výhodná pro X nebo O (0 => +- vyrovnaná pozice (score v sobe nepočítá kdo bude na tahu další), Kladné => vyhrává X, Záporné vyhrává O).

* Utility vypočítavá skóre tak, že spočítá všechny spojité řetězce char1 a char2 s cenami:

length\_value=[0,0,10,20,250,1000000, 1000000, 1000000, 1000000, 1000000, 1000000]

Ve skutečnosti n-tici počítám n-krát (z každého políčka n-tice)

Tady jsou skutečné hodnoty n-tic

* + 1-tice: 0
  + 2-ice: 20
  + 3-ice: 60
  + 4-řice: 1000
  + 5-tice tj vyhra: 1000000.
* Dále se určuje u n-tice stav:
  + otevřený – lze rozšířit na 5-tici do obou stran

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_OOO\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* + uzavřený – lze rozšířit na 5-tici pouze na jednu stranu

\_X\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_O\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_O\_\_\_\_\_\_\_\_\_#\_

\_\_\_\_O\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* + mrtvý – NE-lze rozšířit na 5-tici

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_ X\_OO\_X

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Dále se udělují bonusy za řetězce s mezerou, která není mrtvá např: X\_X nebo OO\_O,

Konstanty: bonus\_value=40, extra\_bonus\_value=60

bonus = "počet 1-niček v řetězci" \*bonus\_value + "2 za každou část 2-ky, 3 za kazdou řást 3-ky.. použitá v bonusu"\*extra\_vonus\_value

score+=sign(char)\*bonus

skutecne hodnoty bonusu (hodnota n-tice se zde nezapočitává jen hodnota bonusu)

pro bonus\_value=40 a extra\_bonus\_value=60:

        X\_X   -> bonus = (1+1)\*40        = 80

        XX\_X  -> bonus = 1\*40+(2+2)\*60   = 280

        X\_X\_X -> bonus = (1+1+1)\*40      = 120

        XXX\_X -> bonus = 1\*40+(3+3+3)\*60 = 580

        XX\_XX -> bonus = (2+2+2+2)\*60    = 480

* n-tice se od hráče char2 ke skóre přičítají, naopak od hráče char1 se odečítají
* state\_value – vrátí 5 za otevřený stav, 2 za uzavřený stav a 0 za mrtvý
* sign – vrátí 1 pokud na vstupu byl char2, -1 pokud na vstupu byl char1
* update\_state – z otevřeného stavu vrátí uzavřený a z mrtvého stavu vrátí mrtvý
* explore\_one\_directon – prohledá v jedné polopřímce: zjistí délku, stav, počet\_mezer a bonus v jedné POUZE polopřímce
* explore – prohledá v jednom smeru (horizontálně, vertikálně a 2x diagonálně) tj. prohledá 2x v jedné polopřímce, určí délku řetězce, stav, přičte (popř. odečte) ke skóre hodnotu řetězce podle jeho délky a stavu a přičte (popř. odečte) bonus

Skóre se vypočítá spuštením funkce explore ve vsech 4 smerech (horizontálně - , vertikálně |, diagonálně\_1 \, diagonálně\_2 / ) na vsechny použitá políčka

Minimax

minimax – funkce jenž vrátí „nejlepší“ tah s danou hloubkou, X je MAX hráč, O je MIN hráč

* valueMIN – vratí hodnotu kterou ziskam optimalni tahem Min hracem, následovanou optimalnim tahem MAX hrace, takhle stridaji dokud se nedostanu na hloubku 0,
* valueMAX - analogicky akorát zacina MAX hráč
* decisionMIN – vrátí nejlepšího tah s danou hloubkou pro MIN hráče
* decisionMAX – analogicky akorát pro MAX hrače

zároveň zde využívám alpha-beta pruning:

Funkce

* show\_board – vytiskne pole board i s číslováním sloupců a řádků
* check\_int - vrátí True pokud její argument byl integer
* check\_split – vrátí True pokud na string lze pouzit funkci split() se dvěma argumenty tj. pokud řetězec lze rozdělit na právě dvě části mezerou
* check input – vrátí True pokud její argument je dvojice celých čísel oddělených mezerou která je mezi 1 a board\_size
* add\_to\_active\_neighbour – pro zadané políčko dá všechny jeho sousedy, kteří již nejsou aktivní do pole active
* ran\_empty\_space - vrátí souřadnice prazného políčka na hrací ploše
* ran\_empty\_space\_middle – vrátí souradnice prázdného políčka +- uprostřed, podle argumentu n vygeneruje nahodne políčko ze ctverce o velikosti 2n+1 se středem ve prostřed hracího pole

funkce ran\_empty\_space a ran\_empty\_space\_middle se pouští na začátku, kdy drtivá většina políček je prazdná, nekontroluje zda volné políčko existuje (když by byly všechny polička obsazená bude náhodně generovat nekonečne dlouho)

* get\_mm\_active –vytvoří pole sousedních políček mm\_active a vratí ho
* minimax\_make\_move – udělá tah minimaxu s danou hloubkou pro daného hrače, zkontrokuje zda „počítač“ nevyhrál, updatuje pole active a přidaná právě zahraný tah do used
* get\_player\_input – chce input od hrace dokud není validní (dvojice celých čísel oddělených mezerou a na zadané políčko je volné) a pak vratí souřadnice (dvojici intů)
* player\_make\_move – bude chtít input od hrače dokud to nebude platný tah a pak na dané políčko zahraje a zkontrokuje zda nevyhrál, updatuje pole active a přidaná právě zahraný tah do used
* switch\_turn – obrátí hodnotu booleonu char1\_turn a vratí ho
* print\_char – vytiske „O“ pokud O\_turn==True, „X“ jinak
* victory – vytiskne skóre, ohlasí kdo vyhrál a ukončí program

Soubory

* board\_setup - definuje board\_size, sestaví pole board,used,active podle vstupu uživatele
* piskvorky\_console – hlavní soubor pomocí kterého se spouští celá hra
* functions\_one – soubor různých funkcí, které se využívají napříč projektem
* minimax\_f – soubor obsahující funkcí minimax
* utility\_function - soubor obsahující funkcí utility

Unit testy

Testuje drtivou většinu funkcí v daných souborech (Mockuje user input i náhodná čísla z randint)

* test\_minimax\_f
* test\_utility\_function
* test\_board\_size
* test\_functons\_one