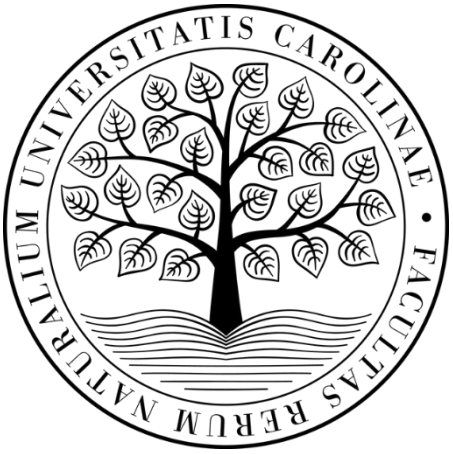
Univerzita Karlova

Přírodovědecká fakulta



Hra piškvorky

Úvod do programování

Dokumentace

Eliška Králová

2. B-GEKA

Protivín 2023

Zadání

Příklad č. 115: Hra piškvorky (Tic-tac-toe) s použitím minimax algoritmu pro hraní proti počítači.

Rozbor problému

Pravidla

Hráči se střídají v umísťování svých kamenů na hrací plán. Hráč, který má nepřerušenou horizontální, vertikální či diagonální řadu daného počtu kamenů, vyhrává. Běžně je potřebné mít v řadě 3 kameny (pro plán 3x3) nebo 5 kamenů (pro větší plán). K nejvýhodnějšímu umístění kamenů se využívá algoritmus minimaxu.

Použitý algoritmus

Minimax algoritmus se snaží maximalizovat zisk počítače a minimalizovat jeho ztráty. Před začátkem je nutné si definovat pro maximalizujícího hráče nejhorší možný výsledek, tedy záporné nekonečno (nejmenší zisk počítače), a naopak pro minimalizujícího hráče kladné nekonečno (největší ztráta hráče). Algoritmus poté prochází rekurzivně všechny možné tahy do dané hloubky, v níž ohodnotí herní plán dle rozdílu počtu kamenů v řadě mezi hráči. V menší hloubce, během tahu počítače, vybere pole s nejvyšším ohodnocením. Naopak při tahu hráče vybere to s nejmenším ohodnocením. Následně vybere nejvýhodnější pole pro hráče na tahu. Pomocí hloubky se dá nastavit obtížnost počítače, jelikož tak uvažuje nad vývojem hry daný počet tahů dopředu.

Při větším počtu polí v hracím plánu či při velké obtížnosti dochází ke zdlouhavému ohodnocování polí. Proto bylo za potřebí využít alfa-beta ořezávání, které tuto dobu výrazně snížilo. Alfa-beta ořezávání snižuje počet procházených tahů, nemůže-li nastat změna výsledku. Tedy je-li při tahu počítače v dané hloubce některé pole s větší hodnotou, než je hodnota ostatních polí, vezme toto pole a u ostatních polí již neprovádí propočty v další hloubce. Při tahu hráče bere naopak nejmenší ohodnocení a pole s vyšší hodnotou zanedbává.

Pseudokód algoritmu[1]

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            if value > β then
                break (* β cutoff *)
            α := max(α, value)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            if value < α then
                break (* α cutoff *)
            β := min(β, value)
        return value
(* Initial call *)
alphabeta(origin, depth, −∞, +∞, TRUE)
```

Struktura programu

Program používá knihovny TkInter, Math, Enum a Random. Z datových struktur jsou využity seznamy.

Třídy

Field – Pole

Datové položky třídy

empty_field– zápis volného pole v plánu

human – zápis hráče v plánu

computer – zápis počítače v plánu

Tic_tac_toe – Piškvorky

Datové položky třídy

canvas – interaktivní hrací plán na němž se vykreslují položené kameny

width – šířka herního plánu

height – výška herního plánu

for_win – počet kamenů potřebných pro vítězství

difficulty – hloubka minimaxu

board – seznam polí, v němž jsou vnořené seznamy polí jednotlivých sloupců

moves – počet umístěných kamenů, pokud jejich počet odpovídá počtu polí v hracím plánu, je plán plný

[1] Zdroj: Wikipedia.org

Funkce

move – tah

Přepočítá souřadnice kliknutí na hrací pole na souřadnice 0, 0 až 2, 2 (hrací pole 3x3). Zavolá funkci **placement** s danými souřadnicemi. Jsou-li souřadnice již použity (pole je obsazené), upozorní program hráče do terminálu a čeká, dokud neklikne na volné políčko. Pro tah počítače nejdřív zavolá funkci **minimax**, ze které si vezme souřadnice pole a přidá je jako parametr funkci **placement**.

placement – umístění

Určuje, který hráč je na tahu a po zavolání funkce s daným znakem jej umístí na příslušné pole. V seznamu herního plánu se přepíše znak pro hráče (1 nebo -1) a na pole již nebude možné nic zapsat. Pomocí funkce **no_in_row** kontroluje, zda hráč na tahu nevyhrál (počítá počet kamenů v řadě pro umístěný kámen) a zaplnění hracího plánu.

draw_cross

Funkce nakreslí křížek na dané pole dle přiřazených souřadnic z parametru.

draw_circle

Funkce nakreslí kolečko na dané pole dle přiřazených souřadnic z parametru.

minimax

Dle hráče na tahu, pro kterého byla funkce volána, určí počáteční nejhorší pole a následně pole hracího plánu v náhodném pořadí postupně prochází. Pomocí funkce **in_row** ohodnotí jednotlivá pole v aktuálním stavu hry. Je-li hloubka rovna 0, hodnota daného pole se zjistí z rozdílu nejlepšího umístění obou hráčů po vzájemném porovnání ohodnocení polí (**in_row**) a počtu kamenů v řadě umisťovaného kamene (**no_in_row**). Při jiné hloubce se spustí znovu funkce **minimax** pro druhého hráče, než byla předtím spuštěna, dokud není hloubka 0. Z ohodnocení všech polí se pro maximalizujícího hráče vezme pole s největší hodnotou a uloží se i jeho souřadnice pro umístění. V opačném případě (minimalizující hráč) se vybere pole s nejmenší hodnotou. Funkce také kontroluje, zda hráč na tahu umístěním na dané pole nevyhraje. V tom případě vrátí toto pole s nejlepším možným ohodnocením.

no_in_row – počet v řadě

Funkce prochází všechny možné směry (horizontální, vertikální a diagonální) a počítá počet již umístěných kamenů jednoho znaku. Zároveň kontroluje, aby neprocházela i neplatná pole pomocí funkce **on_board**.

in_row – ohodnocení plánu

Funkce prochází jednotlivá pole herního plánu a ukládá jejich hodnotu dle počtu kamenů v řadě pro hráče z parametru. Z těchto hodnot se vybere ta největší. Výsledná hodnota se poté využije ve funkci **minimax**, kde se s její pomocí určí nejvýhodnější pole pro umístění kamene.

on_board – pole na plánu

Při procházení počtu kamenů v řadě zajišťuje tato funkce to, že se neprochází neplatná pole (souřadnice mimo hrací plán).

Alternativní programová řešení

Nejdříve jsem pro grafické znázornění používala Želví grafiku, avšak při dalším rozvíjení programu se stávaly souřadnice zmatečnými, a tak jsem přešla na TkInter. Díky němu lze umisťovat kameny pouhým kliknutím na dané pole, a ne psaním souřadnic do terminálu.

Program při hloubce 0 pokaždé znovu ohodnocoval celý herní plán pro každé pole, což prodlužovalo dobu výpočtu. Nyní je plán v nulové hloubce ohodnocen pouze jednou pro aktuální stav hracího plánu. Poté je spočítán počet v řadě pro pole s umístěným kamenem, neboť pouze na tomto místě se mohl zvýšit.

Spuštění programu

Při svém spuštění program nejprve zkontroluje, zda jsou zadané parametry číselné a kladné. Následně se otevře okno s hracím plánem a již stačí kliknout na požadované políčko. Po kliknutí se na dané pole umístí kolečko (znak hráče) a minimax algoritmus vypočítá pole pro počítač, na které se vykreslí křížek. Takto se hráči střídají a program pokračuje, dokud jeden z hráčů nedosáhl požadovaného počtu kamenů v řadě či se nezaplnil celý hrací plán. Jakmile někdo vyhraje nebo nastane remíza, otevře se nové okno, které hráče informuje o nastalém stavu. Po potvrzení informace se program ukončí.

Možná vylepšení

Vhodné vylepšení je uživatelské rozhraní, kde by bylo tlačítko menu a v něm příslušné možnosti jako volba počtu hráčů (2 proti sobě, hráč a počítač či 2 počítače), začít novou hru a další. Program by mohl být vylepšen o možnost výběru pořadí hráčů a výběr jejich znaku. Díky tomu by hráč neměl výhodu začínajícího hráče a hra proti počítači na vyšší obtížnosti by mohla být o něco těžší. Dále by jistě šlo algoritmus minimaxu ještě o něco zrychlit, neboť stále při vyšší obtížnosti trvá propočet dlouho.

Zdroje dat

Programujte: Úvod do Tkinter
http://tkinter.programujte.com/, [cit. 28.12.2022]

Python Software Foundation: Python 3.10.2 documentation
https://docs.python.org/3/, [cit. 14.1.2023]

Wikipedia: Minimax
https://en.wikipedia.org/wiki/Minimax, [cit. 7.1.2023]

Wikipedia: Alpha_beta pruning
https://en.wikipedia.org/wiki/Alpha-beta_proning, [cit. 30.1.2023]