Andrei Shumak
Ewa Kowalska
Yahor Melnik

Sprawozdanie do projektu z Podstaw sztucznej inteligencji

Treść zadania

Celem projektu jest napisanie gry kółko i krzyżyk (3x3) z wykorzystaniem algorytmu min-max z obcinaniem alfa-beta. Aplikacja umożliwia dwa tryby gry:

- sztuczny gracz vs człowiek
- sztuczny gracz vs sztuczny gracz

Interfejs graficzny jest zrealizowany konsolowo.

Realizacja

Cały projekt został zrealizowany w języku Python.

Wybraliśmy ten język bo ma bardzo prosta skladnię i jest fajny do napisania projektów zespołowych.

Projekt składa się z 2 folderów i 2 skryptów:

- ./logs/
- ./tictactoe/
- ./tests.py
- ./run_game.py

Folder ./tictactoe/ zawiera główne narzędzia do gry :

- Plik player_obj.py, który zawiera:
  - klasę User, reprezentującą gracza rzeczywistego
  - klasę AI, reprezentującą gracza sztucznego
- Plik board.py, zawierający naszą siatkę, na której gramy. Ta klasa zawiera funkcję oceniającą. Mechanizm działania tej funkcji będzie przedstawiony w

punkcie algorytmy.

- Plik minimax.py, zawierający sam algorytm minimax, opis w punkcie algorytmy.
- Plik init.py, zawierający wszystkie “stałe”

Plik run_game.py umożliwia grę w krzyrzek kółko. Automatycznie robi logowanie. Poziom jest ustawiony na ERROR.
Plik tests.py był urzyty dla badania, jak się zmienią wyniki gier, jeżeli będzie grał AI vs AI pod kątem zmiany funkcji oceniającej.

Algorytmy

Algorytm minimax z alfa-beta obcinaniem:

Pseudokod:

```
MMAB(board, player, top_level, cur_level, alpha, beta)
    if (board, top_level, cur_level)

    for pos in board:
        if board[pos] zajęta:
            continue
        board[pos] = player

        estimate = MMAB(board, -1*player, top_level, cur_level+1, alpha,
beta)

        board[pos] = 0

        if (maksymalizacja):
            if alpha < estimate:
                alpha = estimate
            if alpha > beta
                return alpha
        else:
            if estimation < beta:
                beta = estimation
```

```
                    if beta <= alpha:
                        return beta
        jeżeli maksymalizujemy:
            return alpha
        else
            return beta
```

Algorytm funkcji oceniającej:

```
   |  |    - oceniamy każdą linijkę poz.
---  --  ---
   |  |
---  --  ---
   |  |
 |       \
 |         oraz każdą przekątną
```

oceniamy każdą linijkę pion.
jeżeli w linijce stoi np. 1 kółko i nic więcej, to gracz z kółkami otrzymuje 10p, jeżeli 2 wtedy 60, jeżeli 3 wtedy 300 (zwycięstwo).
Taka sama logika dla krzyżyków.
Ale z zależności od gracza który ocenia to będziemy punkty jednego z graczy uwzględniali jako ujemne.
Przykład dla gracz, który ma wybrać klatkę gdzie wstawi X.

```
                koszt(przekątna) = -10
              /
 X | O |     - koszt tej linijki 0 (Bo możemy otrzymać punkty tylko
---  ---  ---    wtedy, gdy są symbole jednej zawartości)
   | O |         - koszt tej linijki odnośnie X = -10
---  ---  ---
   | X |     - koszt = 10
|     |   |  \
|     |   |   koszt(przekątna) = 0
|     |   koszt = 0
|   koszt = 0
koszt = 10        Suma = -10 + 10 + 10 -10 = 0
```

jeżeli po wstawieniu X będziemy mieli taką sytuację:

```
 X | O |      To koszt będzie = koszt linii poziomych + koszt linii
---  ---  ---  pionowych + koszt przekątnych =
   | O |      0 - 10 + 60 + 10 + 0 + 10 + 0 - 10 = 60
```

--- --- ---
| X | X

Testowanie

Testowaliśmy jak zachowa się AI vs real person. Nie można go wygrać (co najmniej nam się to nie udało). AI vs AI. I tutaj coś ciekawego bo zmieniając AI głębokość otrzymaliśmy że AI grający kółkami przegrywa w 5 przypadkach z 81 możliwych (wszystkie inne remis). To się dzieje jak głębokość 1 AI ustawiona na 7, a głębokość 2 AI 2,3,4,5,6.
Oraz zostały przeprowadzone testy po zmianie naszych kosztów w funkcji oceniającej. Wyniki można zobaczyć w pliczku res.txt

Wnioski

Algorytm minimax z alpha-beta obcinaniem jest bardzo dobrym algorytmem, dla AI, żeby ten wygrywał, lub co najmniej nie dawał wygrywać przeciwnikowi. Nie jest taki kosztowny jak zwykły minimax, ale wciąż potrzebuję sporo czasu przy dużej ilości gałęzi w drzewie oraz dużej głębokości algorytmu. Zależy od funkcji wyznaczenia kosztu.