Katarzyna Kanicka, 331388

WSI Sprawozdanie 3 – Dwuosobowe gry deterministyczne

Treść polecenia:

1. Zaimplementować algorytm minimax z obcinaniem α − β grający w kółko i krzyżyk, używając przygotowanego tym repozytorium kodu.

2. Przeprowadzić następujące symulacje gier próbując znaleźć jak najlepsze parametry algorytmu minimax:
   - Gracz minimax vs gracz losowy
   - Gracz minimax vs gracz minimax

3. Rozegrać samemu kilka gier przeciwko:
   - Graczowi losowemu
   - Graczowi minimax ze znalezionymi parametrami

4. Zbadać eksperymentalnie wpływ głębokości odcinania drzewa gry.

Instrukcja do używania skryptu

Po pobraniu repozytorium należy:

- przejść do głownego katalogu – lab3

```
$ cd lab3
```

- stworzyć środowisko wirtualne i pobrać requirements.txt

```
$ python3 -m venv .venv
$ source .venv/bin/activate
$ pip install -r requirements.txt
```

- uruchomić plik main.py, w terminalu za pomocą:

```
$ python3 main.py [argumenty]
```

Możliwe argumenty:

- **config** – plik konfiguracyjny
- **seed** – ziarno

- **turns** – liczba rund dla symulacji
- **display_board** – wyświetlanie planszy przy każdym ruchu gracza
- **change_started** – zmiana gracza rozpoczynającego w kolejnych rundach

W pliku ‘config.json’ znajdują się parametry algorytmu: player_x i player_o (human, random, minimax), oraz gui (true, false). Dodatkowo w przypadku gracza minima określa się głębokość przycinania.

Parametry algorytmu Minimax z obcinaniem α – β

- **bez użycia obcinania α – β** – algorytm wykonuje się bardzo długo, co widać po ilości wywołań:

Pruning depth	Numer of minimax function calls for the starting board
8	549945
7	422073
6	221625
5	73449

- **głębokość** – maksymalna głębokość potrzebna do przyanalizowania wszystkich ruchów to 8, lecz wydłuża to działanie algorytmu.
- **obcinanie α – β** – umożliwia ono ograniczenie liczby przeanalizowanych stanów gry. Zdecydowałam, że wystarczającą głębokością jest depth=5 do prawidłowego działania. Widać znaczne poprawienie wyników

Pruning depth	Numer of minimax function calls for the starting board
8	30709
7	23310
6	13831
5	5265

- Aby zoptymalizować liczbę wywołań funkcji minimax postanowiłam sortować ruchy dostępne w danym stanie gry za pomocą funkcji heurystycznej.
- **Funkcja heurystyczna** – ocenia dany ruch ze względu na to w ilu wygranych konfiguracjach się znajduje:

3	2	3
2	4	2
3	2	3

- Z sortowaniem ruchów liczba wywołań funkcji wynosi teraz:

Pruning depth	Numer of minimax function calls for the starting board
8	15674
7	13048
6	8953
5	4647

- Dla większych głębokość liczba wywołań znacznie się zmniejsza

Symulacje

Gracz minimax vs gracz losowy

- Gracz minimax rozpoczyna (x – minimax, o - random)

Results:
x - 20
o - 0
t - 0

Results:
x - 19
o - 0
t - 1

- Gracz minimax z łatwością pokonuje gracza losowego, rzadko dochodzi do remisu

- Gracz losowy rozpoczyna (x – random, o - losowy)

Results:
x - 0
o - 17
t - 3

Results:
x - 0
o - 16
t - 4

- Gracz losowy, z powodu tego, że zaczyna, ma większą szansę na spowodowanie remisu. Jednak gracz minimax wciąż w większości wygrywa

Gracz minimax vs gracz minimax

Results:
x - 0
o - 0

Results:
x - 0
o - 0
t - 20
t - 20

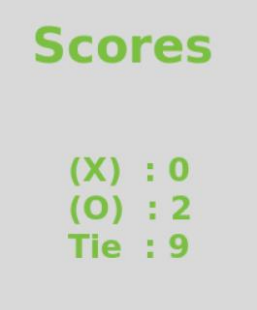
- Tutah oba gracze mają taką samą głębokość przeszukiwania, więc nie ma znaczenia kto zaczyna. Za każdym razem zachodzi remis

Testowanie gry

Przeciwko graczowi losowemu

Bardzo łatwo wygrać przeciwko graczowi losowemu. W niektórych przypadku może doprowadzić do remisu, ale szansa jest niewielka.

Przeciwko graczowi minimax



Nie udało mi się wygrać ani razu z graczem minimax z sortowaniem za pomocą funkcji heurystycznej. Gdy gracz minimax rozpoczyna, to zawsze na początku wybierze środek, przez co jedynie mogę doprowadzić do remisu. Gdy ja zaczyna również mogę doprowadzić jedynie do remisu, ponieważ przeciwnik dobrze blokuje każdą możliwość mojej wygranej.

Wpływ głębokości odcinania drzewa – małe wartości

(Uwaga: te testy zostały przeprowadzone bez sortowania ruchów za pomocą funkcji herytycznej, ponieważ ona bardzo pomaga w wyborze korzystnych kroków nawet przy małych głębokościach)

Gracz minimax (depth = 2) vs gracz losowy

- Gracz minimax rozpoczyna (x – minimax, o - random)

Results:
x - 16
o – 2
t - 2

Results:
x – 18
o – 1
t - 1

Results:
x – 19
o – 1
t - 0

- Minimax wciąż w większości wygrywa, ale gracz losowy ma większą szansę na wygraną niż w przypadku większkzej głębokości przeszukiwania
- Gracz losowy rozpoczyna (o – minimax, x - random)

Results:
x – 4
o – 13
t - 3

Results:
x – 2
o – 12
t - 6

Results:
x – 3
o – 12
t – 4

- Dzięki temu, że gracz losowy rozpoczyna, ten ma jeszcze większą szansę na wygraną

Gracz minimax (depth = 2) vs gracz minimax (depth = 5)

- rozpoczyna gracz z depth = 5

Results:
x – 20
o – 0
t - 0

Results:
x - 20
o - 0
t - 0

- Gracz z mniejszą głębokością wcześniej ucina drzewa przeszukiwania, dlatego nie wybiera zawsze najlepszeo ruchu. W tym wypadku gracz z depth = 5 zawsze wygrywa
- rozpoczyna gracz z depth = 2

Results:
x - 0
o - 0
t - 20

- Dzięki temu, że gracz z depth = 2 rozpoczyna, jest w stanie doprowadzić zawsze do remisu

Wnioski końcowe

- Algorytm Minimax z obcinaniem α-β znacząco poprawia efektywność przeszukiwania przestrzeni stanów gry, redukując liczbę wywołań funkcji Minimax w porównaniu do pełnego przeszukiwania.
- Głębsze przeszukiwanie umożliwia lepszą ocenę przyszłych stanów gry, ale odbywa się to kosztem wydłużonego czasu obliczeń.
- Optymalizacje, takie jak sortowanie ruchów, pozwalają znacząco poprawić czas działania algorytmu bez utraty skuteczności.
- Zatrzymanie przeszukiwania na określonej głębokości może prowadzić do decyzji, które nie są optymalne, bo algorytm nie bierze pod uwagę wszystkich możliwości.