Raport

Alesia Filinkova

336180

1 Treść ćwiczenia

1. Zaimplementować algorytm minimax z obcinaniem Alpha-beta dla gry w kółko i krzyżyk, korzystajac z dostarczonego repozytorium.
2. Przeprowadzić symulacje rozgrywek:
   - Gracz minimax vs gracz losowy.
   - Gracz minimax vs gracz minimax.
3. Rozegrać kilka gier jako gracz ludzki przeciwko:
   - Graczowi losowemu.
   - Graczowi minimax z optymalnymi parametrami.
4. Zbadać eksperymentalnie wpływ głebokości odcinania drzewa na jakość gry i czas obliczeń.

2 Doprecyzowanie

1. Zestaw parametrów: Głównym parametrem testowanym w algorytmie minimax jest maksymalna głebokość drzewa decyzyjnego. Testy wykonano dla głebokości d = 0, 1, 2, 3, 4, ..., 9.
2. Konfiguracja w pliku: Parametry graczy (np. typ gracza i głebokość) definiowane sa w pliku ‘config.json‘.
3. Metryki: Wykorzystano liczbe wygranych, przegranych i remisów jako podstawowa miare jakości gry. W przypadku badań wydajności zmierzono czas wykonania ruchu.

3 Cel i opis eksperymentów

3.1 Cel Eksperymentów:

1. Ocena skuteczności algorytmu: Porównanie wyników gracza minimax z różnymi przeciwnikami oraz wpływ głebokości drzewa.

2. Wpływ parametrów: Zbadanie, jak zmiana maksymalnej głębokości wpływa na jakość rozgrywki oraz czas wykonania ruchów.

3. Wpływ pierwszeństwa: Sprawdzenie, czy rozpoczynanie gry daje istotna przewage.

3.2 Opis Eksperymentów:

1. Eksperyment 1: Gracz minimax vs gracz losowy:
   - Cel: Sprawdzić skuteczność minimax przeciwko losowym ruchom.
   - Metodyka: 100 symulacji dla każdego poziomu głębokości; Minimax rozpoczyna w 50 grach.
2. Eksperyment 2: Gracz minimax vs gracz minimax:
   - Cel: Zweryfikować, czy wieksza głębokość zwieksza szanse na zwyciestwo.
   - Metodyka: 100 symulacji z dwoma graczami minimax o różnych głębokościach; Obserwacja wyników i czasu gry.

4 Instrukcja odtworzenia wyników

`Skrypt można uruchomić przez terminal za pomoca` polecenia:

1. git clone https://gitlab-stud.elka.pw.edu.pl/afilinko/wsi.git
2. python3 -m venv venv
3. source venv/bin/activate
4. cd /lab3
5. pip install -r requirements.txt
6. `python3 main.py --config config.json`

5 Wyniki

5.1 Minimax vs Random

depth	wins (Minimax)	tie	looses (Random)	Czas ruchu
0	43	7	50	0.000231933594
1	88	2	10	0.0016784191
2	95	4	1	0.004676532745
5	90	10	0	0.03849869967
8	94	6	0	0.277585268

depth	wins (Minimax)	tie	looses (Random)	Czas ruchu
0	50	0	50	$3.09228897 \times 10^{-4}$
1	50	0	50	0.002569770813
2	0	100	0	0.004860745536
5	0	100	0	0.07350757387
8	0	100	0	0.1281554434

5.2 Minimax vs Minimax same depth

5.3 Minimax vs Minimax different depth

depth x	depth o	wins x	tie	wins o	Czas ruchu x	Czas ruchu o
1	9	0	50	50	0.001762342453	0.05726391077
2	8	0	100	0	0.006449651718	0.05123120546
9	2	0	100	0	0.004676532745	0.004571855068
5	4	0	100	0	0.2006055355	0.03115457296

6 Wnioski

1. Optymalna głebokość: Głebsze drzewo poprawia skuteczność algorytmu, ale zwieksza czas obliczeń.
2. Pierwszeństwo:
   - Głebsze drzewo poprawia skuteczność algorytmu, ale zwieksza czas obliczeń.
   - Grajac minimax vs minimax pierwszy gracz wygrywa tylko przy głebokości 0(to oznacza poprosu randomome ruchy) 1 dla obu, przy wiekszej głebokości zawsze jest remis;
   - Przy różnej głebokości algorytmów, ten z wieksza głebokościa wygrywa tylko wtedy, gdy przeciwnik ma głebokość 1. Jeśli przeciwnik ma wieksza głebokość, to wystarczy, aby zawsze sprowadzić gre do remisu
3. Wydajność algorytmu: Alpha-beta obcinanie znacznie poprawia wydajność minimax oraz evaluate function, szczególnie przy wiekszych głebokościach.
4. Interakcja z graczem: Gracz minimax o średniej głebokości (np. 4) oferuje dobra równowage miedzy wyzwaniem a grywalnościa.