

Raport

Alesia Filinkova

336180

1 Treść ćwiczenia

1. Zaimplementować algorytm minimax z obcinaniem Alpha-beta dla gry w kółko i krzyżyk, korzystając 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 głębokości odcinania drzewa na jakość gry i czas obliczeń.

2 Doprecyzowanie

1. Zestaw parametrów: Głównym parametrem testowanym w algorytmie minimax jest maksymalna głębokość drzewa decyzyjnego. Testy wykonano dla głębokości $d = 0, 1, 2, 3, 4, \dots, 9$.
2. Konfiguracja w pliku: Parametry graczy (np. typ gracza i głębokość) definiowane są w pliku 'config.json'.
3. Metryki: Wykorzystano liczbe wygranych, przegranych i remisów jako podstawową miarę 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 głębokości 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 istotną przewagę.

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 większa głębokość zwiększa szanse na zwycięstwo.
 - 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 pomocą 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

Random wygrywa czasem wtedy, kiedy głębokość algorytmu minimax jest bardzo mała i tylko wtedy, kiedy random zaczyna gre.

5.2 Minimax vs Minimax same depth

depth	wins x	tie	wins o	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

Kiedy głębokość bardzo mała lub równa się 0 i głębokości algorytmów minimax są takie same, wygrywa gracz który zaczyna rundę

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

Kiedy jeden z graczy ma głębokość 1, a drugi gracz ma wielką głębokość, nastąpi remis, jeśli grę rozpocznie gracz, który ma głębokość 1.

6 Wnioski

1. Optymalna głębokość: Głębsze drzewo poprawia skuteczność algorytmu, ale zwiększa czas obliczeń.
2. Pierwszeństwo:
 - Głębsze drzewo poprawia skuteczność algorytmu, ale zwiększa czas obliczeń.
 - Grając minimax vs minimax pierwszy gracz wygrywa tylko przy głębokości 0 (to oznacza poprosu randomome ruchy) 1 dla obu, przy większej głębokości zawsze jest remis;
 - Przy różnej głębokości algorytmów, ten z większą głębokością wygrywa tylko wtedy, gdy przeciwnik ma głębokość 1. Jeśli przeciwnik ma większą głębokość, to wystarczy, aby zawsze sprowadzić grę do remisu
3. Wydajność algorytmu: Alpha-beta obcinanie znacznie poprawia wydajność minimax oraz evaluate function, szczególnie przy większych głębokościach.
4. Interakcja z graczem: Gracz minimax o średniej głębokości (np. 4) oferuje dobrą równowagę między wyzwaniem a grywalnością.