Inteligencja obliczeniowa w analizie danych cyfrowych

Projekt:

Nimby

AGH Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Autorzy
Michał Burda
Kamil Poniewierski

Spis treści

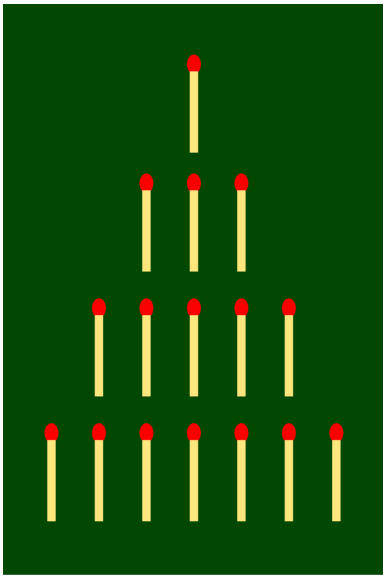
1. Opis gry Nim

Nim to klasyczna gra matematyczna dla dwóch graczy, w której kluczową rolę odgrywa strategia. Gra toczy się na zbiorze *kupek* zawierających pionki.

Zasady gry:

1. Pionki dzieli się na kupki, które powinny spełniać następujące warunki:
    - Liczba kupek: co najmniej **trzy**.
    - Liczba pionków w każdej kupce: **co najmniej cztery**.
    - Każda kupka zawiera inną liczbę pionków.
2. Gracze wykonują ruchy na zmianę.
3. W jednym ruchu gracz może zabrać **dowolną, niezerową liczbę pionków**, ale **tylko z jednej kupki**.
4. Istnieją dwie wersje warunków zwycięstwa:
    - **Normalna wersja** (*misère*): przegrywa gracz, który zabierze ostatni pionek.
    - **Klasyczna wersja**: wygrywa gracz, który zabierze ostatni pionek.

Gra Nim jest dobrze znana w teorii gier, a jej optymalna strategia opiera się na operacjach w systemie binarnym (*nim-sum*).



Układ zapałek w formie piramidy sugeruje klasyczną wersję tej gry, w której gracze na zmianę zabierają dowolną liczbę zapałek z jednego rzędu.

2. Wykorzystane technologie

W projekcie **Nimby**, implementującym probabilistyczny wariant gry **Nim**, wykorzystaliśmy następujące technologie i narzędzia:

Język programowania: Python

- Python został użyty do implementacji całej logiki gry oraz eksperymentów z algorytmami sztucznej inteligencji.

Biblioteka `easyAI`

- `easyAI` to biblioteka do tworzenia gier turowych, szczególnie przydatna do implementacji sztucznej inteligencji opartej na algorytmach przeszukiwania drzewa gry.
- Umożliwia łatwe wdrażanie algorytmów takich jak **Negamax** oraz jego warianty.

3. Algorytmy

Negamax bez odcięcia alfa-beta

Negamax to zoptymalizowana wersja algorytmu MiniMax, używana w grach dwuosobowych z pełną informacją, takich jak szachy, warcaby czy Nim. W przeciwieństwie do MiniMaxa, który ocenia osobno wartości dla obu graczy, Negamax używa jednej wartości punktowej, zmieniając jej znak w zależności od aktualnego gracza.

Jak działa?

1. Algorytm rekurencyjnie analizuje wszystkie możliwe ruchy do określonej głębokości.
2. Po osiągnięciu maksymalnej głębokości lub końca gry ocenia stan gry przy użyciu funkcji oceny.
3. Wybiera ruch o najlepszej wartości, zakładając, że przeciwnik zawsze gra optymalnie.

Negamax z odcięciem alfa-beta

Negamax z odcięciem alfa-beta to ulepszona wersja Negamax, w której stosuje się tzw. odcinanie alfa-beta (alpha-beta pruning), co pozwala znacznie ograniczyć liczbę przeszukiwanych ruchów.

Jak działa odcięcie alfa-beta?

1. Dwa parametry:
    - α (alfa) – najlepszy znaleziony wynik dla gracza maksymalizującego.
    - β (beta) – najlepszy znaleziony wynik dla gracza minimalizującego.
2. Jeśli wartość węzła jest gorsza niż już znalezione β, przerywamy dalszą eksplorację tego ruchu, bo przeciwnik nigdy go nie wybierze.
3. Dzięki temu drzewo gry jest przeszukiwane szybciej, bez utraty jakości wyników.

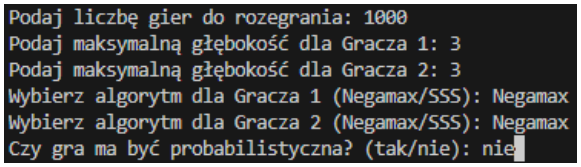
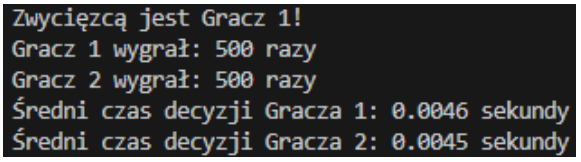
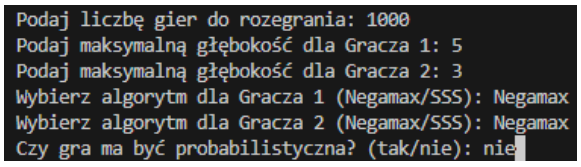
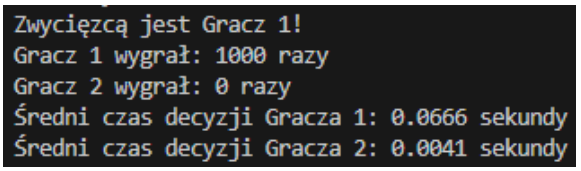
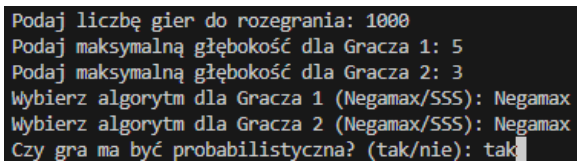
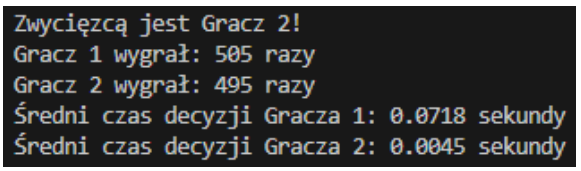
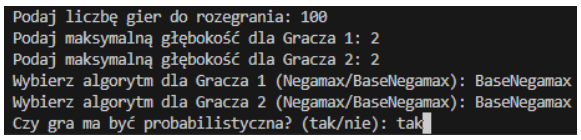
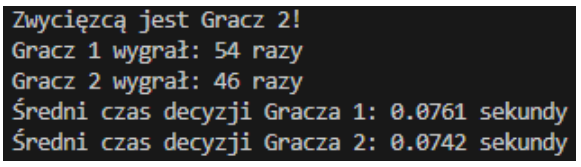
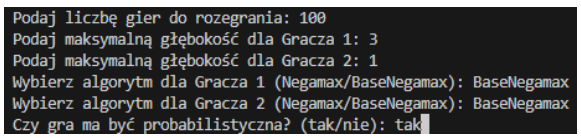
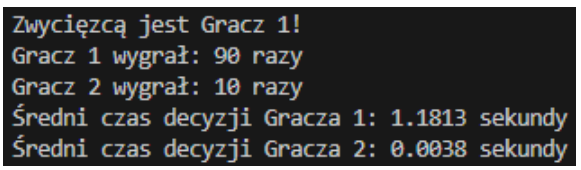
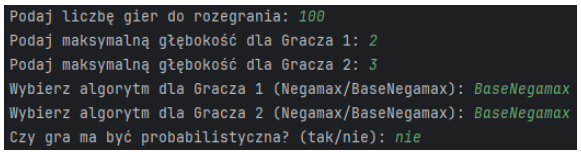
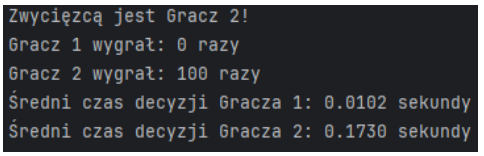
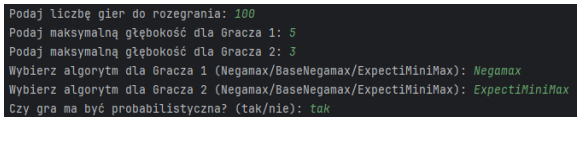
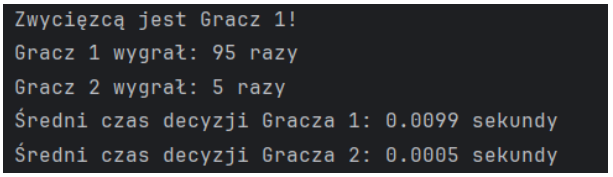
Expecti-Minimax (z losowością)

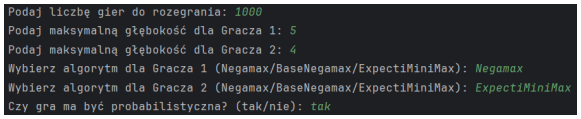
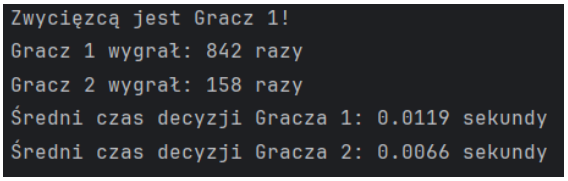
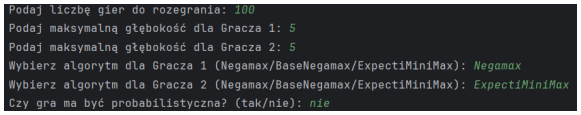
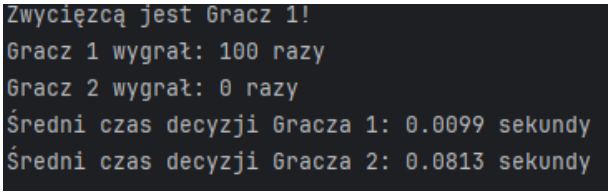
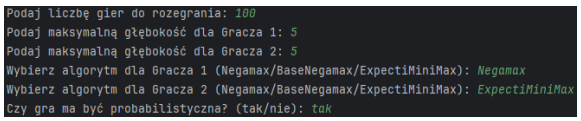
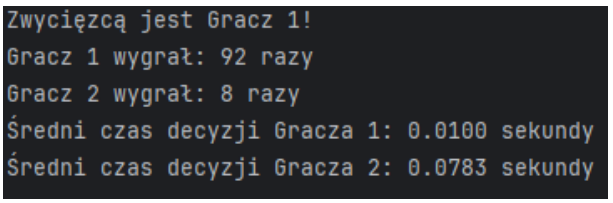
Expecti-Minimax to rozszerzenie klasycznego MiniMaxa, które obsługuje elementy losowe w grze. Jest wykorzystywane w grach, gdzie oprócz ruchów graczy występują zdarzenia losowe, np. rzuty kostką lub losowe ograniczenia ruchu.

Jak działa?

1. Algorytm działa podobnie do MiniMaxa, ale dodatkowo wprowadza węzły losowe (CHANCE nodes), które reprezentują zdarzenia probabilistyczne.
2. Węzły MAX i MIN działają tak samo jak w MiniMaxie.
3. Węzły CHANCE liczą średnią wartość wszystkich możliwych wyników, uwzględniając ich prawdopodobieństwo.

4. Przykładowe eksperymenty z AI, które przeprowadziliśmy

Input	Wynik działania
	
	
	
	
	
	
	

 Podaj liczbę gier do rozegrania: 1000 Podaj maksymalną głębokość dla Gracza 1: 5 Podaj maksymalną głębokość dla Gracza 2: 4 Wybierz algorytm dla Gracza 1 (Negamax/BaseNegamax/ExpectiMiniMax): Negamax Wybierz algorytm dla Gracza 2 (Negamax/BaseNegamax/ExpectiMiniMax): ExpectiMiniMax Czy gra ma być probabilistyczna? (tak/nie): tak	 Zwycięzcą jest Gracz 1! Gracz 1 wygrał: 842 razy Gracz 2 wygrał: 158 razy Średni czas decyzji Gracza 1: 0.0119 sekundy Średni czas decyzji Gracza 2: 0.0066 sekundy
 Podaj liczbę gier do rozegrania: 100 Podaj maksymalną głębokość dla Gracza 1: 5 Podaj maksymalną głębokość dla Gracza 2: 5 Wybierz algorytm dla Gracza 1 (Negamax/BaseNegamax/ExpectiMiniMax): Negamax Wybierz algorytm dla Gracza 2 (Negamax/BaseNegamax/ExpectiMiniMax): ExpectiMiniMax Czy gra ma być probabilistyczna? (tak/nie): nie	 Zwycięzcą jest Gracz 1! Gracz 1 wygrał: 100 razy Gracz 2 wygrał: 0 razy Średni czas decyzji Gracza 1: 0.0099 sekundy Średni czas decyzji Gracza 2: 0.0813 sekundy
 Podaj liczbę gier do rozegrania: 100 Podaj maksymalną głębokość dla Gracza 1: 5 Podaj maksymalną głębokość dla Gracza 2: 5 Wybierz algorytm dla Gracza 1 (Negamax/BaseNegamax/ExpectiMiniMax): Negamax Wybierz algorytm dla Gracza 2 (Negamax/BaseNegamax/ExpectiMiniMax): ExpectiMiniMax Czy gra ma być probabilistyczna? (tak/nie): tak	 Zwycięzcą jest Gracz 1! Gracz 1 wygrał: 92 razy Gracz 2 wygrał: 8 razy Średni czas decyzji Gracza 1: 0.0100 sekundy Średni czas decyzji Gracza 2: 0.0783 sekundy

5. Opis otrzymanych wyników

Porównanie liczby zwycięstw dla różnych algorytmów

- Wyniki dla wersji Negamax z obcinaniem i bez były podobne, obcinanie gałęzi nie wpływa na decyzję algorytmu.
- Głębokość drzewa dla algorytmów Negamax nie miała dużego znaczenia w wersjach probabilistycznych -- algorytmy na głębokościach 3 i 5 prawie zremisowały.

  Miała za to duże znaczenie w wersjach deterministycznych -- jakakolwiek różnica głębokości powodowała dominację, 100-0 dla graczy z głębokościami 3-2, 1000-0 dla 5-3
- Expecti-Minimax radził sobie lepiej w wersji probabilistycznej, ze względu na branie pod uwagę możliwość wystąpienia zdarzenia losowego, za to przegrywał w wersji deterministycznej, gdyż brał pod uwagę zdarzenia, które nigdy nie zachodziły.
- W wersji deterministycznej, gdy obaj gracze mają taką samą głębokość przeszukiwania w algorytmie Negamax, wygrywają równie często. Natomiast gdy jeden z nich ma większą głębokość, zawsze wygrywa, ponieważ przewiduje więcej ruchów i unika przegrywających strategii.

Porównanie średniego czasu decyzji

- Negamax bez alfa-beta był znacznie wolniejszy, ponieważ analizował wszystkie możliwe ruchy bez żadnej optymalizacji.

  Wersja z obcinaniem na głębokości 5 podejmowała ruch w podobnym czasie co wersja bez obcinania na głębokości 2.
- Negamax z alfa-beta był szybszy, ponieważ skutecznie odcinał niepotrzebne ruchy, redukując liczbę analizowanych stanów gry.
- Expecti-Minimax był nieco wolniejszy od Negamax z alfa-beta, co wynika z konieczności obsługi węzłów losowych i obliczania średnich wartości.

6. Napotkane Problemy

Jednym z głównych problemów był długi czas oczekiwania na decyzję AI, szczególnie dla Negamax bez alfa-beta, który przy głębokościach powyżej 3 analizował zbyt wiele stanów. Rozwiązaniem było zastosowanie Negamax z odcięciem alfa-beta, co znacząco przyspieszyło obliczenia.

Kolejnym wyzwaniem była losowość w grze, która utrudniała AI przewidywanie ruchów, prowadząc do nieoptymalnych decyzji. Zwiększenie głębokości przeszukiwania pozwoliło Expecti-Minimax lepiej reagować na zmiany w rozgrywce.

Ostatecznie trudność sprawiało dobranie najlepszego algorytmu – Expecti-Minimax sprawdzał się lepiej w wersji probabilistycznej, ale gorzej w deterministycznej, dlatego konieczne było testowanie różnych wariantów, aby uzyskać optymalne wyniki.

7. Podsumowanie

1. Negamax bez alfa-beta był poprawny, ale bardzo wolny, szczególnie dla większych głębokości.
2. Negamax z alfa-beta był najlepszy dla gier deterministycznych, ponieważ znacznie redukował liczbę przeszukiwanych stanów.
3. Expecti-Minimax sprawdzał się najlepiej w grze probabilistycznej, ale miał trudności z adaptacją do losowych zmian.
4. Wysokie głębokości przeszukiwania znacząco wydłużały czas podejmowania decyzji, co wymagało optymalizacji algorytmów.