Sprawozdanie - SI

Lab 2

Wiktor Sadowy

Cel zadania

W ramach zadania mieliśmy zaimplementować:

a) grę Reversi z poprawnie zdefiniowanym stanem gry oraz funkcją generującą możliwe ruchu dla danego stanu i gracza
b) co najmniej 3 heurystyki pozwalające ocenić stan gry w oparciu o różne strategie
c) algorytm Minimax
d) algorytm alfa-beta cięcie
e) heurystykę adaptacyjnie zmieniającą strategię gracza w celu osiągnięcia zwycięstwa oraz zaimplementowanie możliwość przeprowadzenia gry pomiędzy dwoma programami

Opis teoretyczny

Zanim zdefiniujemy czym są algorytmy Minimax i alfa-beta cięcie zdefiniujmy czym jest drzewo decyzyjne.

Drzewo decyzyjne to skierowane drzewo w którym każdy węzeł reprezentuje stan problemu, a każda krawędź reprezentuje działanie lub wybór, który prowadzi do kolejnego stanu.

W naszej grze:

a) węzły będą reprezentowane przez obecne położenie pionków na mapie
b) krawędzie będą reprezentowane przez możliwe ruchy do wykonania
c) liściami (czyli węzłami od których nie wychodzą żadne krawędzie) będą stany, które oznaczają koniec gry
d) do oceny stanu gry będziemy używać heurystyk

Jedną ze strategii przeszukiwania drzewa gry jest Minimax. Algorytm przegląda rekurencyjnie każdy węzeł drzewa aż dojdzie do liścia drzewa lub osiągnie maksymalną głębokość (ustaloną wcześniej przez użytkownika).

Jeżeli algorytm dojdzie do liścia drzewa lub osiągnie maksymalną głębokość to zwracana jest ocena stanu gry dla gracza, który aktualnie wykonuje ruch.

W przeciwnym przypadku rekurencyjnie przeglądamy kolejne węzły drzewa i zwracamy wartość maksymalną jeżeli to my wykonujemy ruch albo wartość minimalną jeżeli to przeciwnik wykonuje ruch.

Metodą optymalizacji algorytmu Minimax jest użycie cięcia alfa-beta, które polega na zatrzymaniu rekurencyjnego przeglądania pewnej części drzewa, gdy z oceny stanu gry jesteśmy w stanie stwierdzić, że przeglądanie danej części drzewa nie wpłynie na końcowy wynik działania algorytmu

Główną różnicą pomiędzy algorytmem Minimax a cięciem alfa-beta jest użycie dodatkowych wartości alfa i beta, które pozwalają na przekazywanie pomiędzy kolejnymi rekurencyjnymi przeszukiwaniami drzewa odpowiednio najwyższą wartość stanu gry gdy my wykonaliśmy ruch i najniższą wartość stanu gry gdy przeciwnik wykonał ruch podczas przeglądania innych węzłów drzewa.

Dzięki użyciu cięcia alfa-beta jesteśmy w stanie zmniejszyć liczbę węzłów drzewa, które musimy przeglądnąć co zmniejszy czas wykonania algorytmu.

Przykładowe zastosowanie

Algorytm Minimax i alfa-beta cięcie mogą być użyte do implementacji gracza komputerowego, który będzie z nami grał w grę. Modyfikując poziom głębokości drzewa oraz używaną heurystykę jesteśmy w stanie ustawiać poziom trudności gry np. AI na niskim poziomie trudności nie będzie myślało zbytnio w przód i będzie używało najprostszych strategii. Natomiast AI na wysokim poziomie trudności będzie stosowało bardziej zaawansowane strategie.

Algorytmu Minimax i alfa-beta cięcie możemy też użyć do porównywania różnych strategii grania w grę. Możemy ustawić tą samą głębokość drzewa i porównywać która z heurystyk będzie najczęściej prowadziła do wygranej.

Wprowadzone modyfikacje + implementacja

W programie użyliśmy 3 klas do implementacji gry:

1. AI – klasa reprezentująca gracza. Wykonuje on ruchy w oparciu o algorytmy Minimax, alfa-cięcie lub decyzję użytkownika. Klasa ma też zapisane heurystyki, których używa do oceny stanu gry
2. Board – klasa reprezentująca obecny stan gry. Pozwala na ustalenie jakie ruchy możemy wykonać, wykonywanie ruchów oraz ustalenie czy gra się skończyła.
3. Players – klasa reprezentująca co dane oznaczenie na planszy znaczy. Pozwala uniknąć hardkodowania wartości do klasy Board i AI

Cała grę rozgrywa się w funkcji main

W grze AI może używać 5 heurystyk:

1. Maksymalizacja liczby swoich pionków na planszy
2. Maksymalizacja liczby swoich tzw. stabilnych pionków na planszy (czyli pionków, które nie mogą zmienić koloru do końca gry)

3. Minimalizacja liczby ruchów, które komputer może wykonać
4. Maksymalizacja liczby swoich pionków na planszy z zaznaczeniem, że preferowane są pionki na obrzeżach mapy
5. Heurystyka adaptacyjna, która wykorzystuje heurystyki 2, 3 i 4 żeby zmaksymalizować swoje szansę na wygranie

Materiały dodatkowe

Algorytmy Minimax i alphabeta były implementowane na podstawie kodu dostępnego na Wikipedii oraz na podstawie pdf z zadaniem

Heurystyki były bazowane na podstawie pracy inżynierskiej Roberta Wiatra pt. „Implementacja zaawansowanych strategii gry Reversi w C# dla platformy .NET. Projektowanie modułów AI i gra z użyciem połączenia sieciowego”.

Opis wykorzystanych bibliotek

Do zadania zostały wykorzystane następujące wbudowane biblioteki:

a) copy – biblioteka pozwalająca kopiować planszę (unikamy potencjalnego wykonania kilku ruchów jednocześnie podczas wykonywania Minimax lub alfa-beta cięcie)
b) time – biblioteka pozwalająca na mierzenie czasu
c) enum – biblioteka pozwalająca na łatwe zdefiniowane co znaczą liczby 0, 1 i 2 na planszy bez hardkodowania tych wartości

Oraz następujące dodatkowe biblioteki:

a) numpy – biblioteka pozwalająca zapisać planszę w postaci tablicy dwuwymiarowej

Napotkane problemy implementacyjne

Mieliśmy następujące problemy implementacyjne:

a) Jak zdefiniować brak możliwości wykonania ruchu w grze? Według reguł gry jeżeli my nie możemy wykonać ruchu, ale przeciwnik może to wówczas powinniśmy pomijać turę
b) Jak w łatwy sposób dodawać kolejne heurystyki do programu? Możemy po dodaniu nowej heurystyki w Minimax i alfa-beta cięcie dodawać kolejnego ifa na kolejną heurystykę, ale to bardzo mocno wydłuża kod

Rozwiązaliśmy te problemy implementacyjne w następujący sposób:

a) W momencie w którym wykryjemy brak możliwości wykonania ruchu dajemy graczowi specjalny ruch oznaczony jako None, który potem jest przetwarzany jako brak możliwości wykonania ruchu

b) Wykorzystujemy mechanizmy Pythona i definiujemy na samym początku wykonania programu, którą heurystykę będziemy używać

Przykładowe działanie programu

```
Turn 30
Current player: 2
Move: row 8, column 4
2 2 2 2 2 2 2 2
2 1 1 1 2 1 2 2
2 1 1 1 1 2 2 2
2 1 2 1 1 2 2 2
2 1 1 2 2 2 2 2
2 2 1 1 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 1 2 2


End results
2 2 2 2 2 2 2 2
2 1 1 1 2 1 2 2
2 1 1 1 1 2 2 2
2 1 2 1 1 2 2 2
2 1 1 2 2 2 2 2
2 2 1 1 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 1 2 2

No turns: 30, Winning Player: Player2
Number of visited nodes for Player1: 12347
Number of visited nodes for Player2: 11644
Length of the game: 20.22 seconds
```

Analiza algorytmów, głębokości drzewa oraz użytych heurystyk

Zacznijmy od dokonania analizy liczby odwiedzonych węzłów oraz czasu działania algorytmu w zależności od użytego algorytmu oraz maksymalnej głębokości drzewa.

Na potrzeby eksperymentu wyłączyliśmy wyświetlanie bieżącego stanu gry.

Założyliśmy, że gracz pierwszy zawsze gra z heurystyką 2 oraz że gracz drugi wykonuje ruchy w pomijalnym czasie (zagwarantowaliśmy to poprzez użycie heurystyki 1 i ustawienie maksymalnej głębokości drzewa na 1)

Głębokość drzewa \ Użyty algorytm	Minimax	Alfa-beta cięcie	Alfa-beta cięcie z sortowaniem węzłów
2	1517/1.03 sekund	661/0.62 sekund	606/0.72 sekund
3	14178/10.63 sekund	4236/3.57 sekund	3922/3.62 sekund
4	81878/50.12 sekund	7237/7.08 sekund	6266/6.36 sekund

Liczba odwiedzonych węzłów drzewa przez pierwszego gracza oraz czas gry w zależności od ustawionej maksymalnej głębokości drzewa i użytego algorytmu

Widzimy, że użycie algorytmu alfa-cięcie znacząco przyspiesza czas wykonania algorytmu oraz zmniejsza liczbę odwiedzonych węzłów. Wprowadzenie dodatkowego sortowania węzłów zmniejsza liczbę odwiedzonych węzłów i dla większych głębokości drzewa zmniejsza czas wykonania algorytmu.

Warto zauważyć, że nie zawsze alfa-beta cięcie będzie szybsze od Minimaxa. Do tego może dojść np. wtedy gdy żadnych węzłów nie odrzucamy podczas rekurencyjnego przeglądania drzewa. Wówczas wykonujemy dodatkowe obliczenia, które w żaden sposób nie przekładają się na zwiększenie efektywności algorytmu.

Podobny problem można zauważyć gdy będziemy sortować węzły w alfa-beta cięcia. Nie zawsze posortowane węzły będą ułożone tak, żeby móc wyeliminować którykolwiek z węzłów z analizy. Warto też zauważyć, że ustalenie w jakiej kolejności będziemy zwiedzać węzły wymaga wyliczenia wcześniej odpowiednio wartości heurystyk dla danych węzłów co zwiększa czas wykonania algorytmu. To wyjaśnia czemu dla małych głębokości drzewa czas wykonania algorytmu był dłuższy mimo mniejszej liczby odwiedzonych węzłów.

Jednak w większości przypadków alfa-beta cięcie będzie szybsze od Minimaxa. Wprowadzone sortowanie dodatkowo przyspieszy działanie algorytmu.

Kolejną rzeczą, którą przeanalizujemy są heurystyki. Poniższa tabela przedstawia, która heurystyka działa najlepiej. Założyliśmy, że pierwszy gracz używa algorytmu Minimax i maksymalnej głębokości drzewa równej 3. Gracz drugi nie został zmieniony

Użyte heurystyki	Liczba odwiedzonych węzłów drzewa	Czas działania algorytmu (w sekundach)	Liczba odwiedzonych węzłów w czasie jednej sekundy
Heurystyka 1	11362	2.47	4600.00
Heurystyka 2	14178	9.85	1439.39
Heurystyka 3	8018	4.91	1632.99
Heurystyka 4	16175	3.97	4074.31
Heurystyka 5	18018	19.98	901.80

Porównanie liczby odwiedzonych węzłów w czasie jednej sekundy dla algorytmu Minimax o maksymalnej głębokości drzewa równej 3 dla różnych heurystyk

Widzimy, że heurystyki 1 i 4 działają najszybciej gdzie w czasie jednej sekundy jesteśmy w stanie zwiedzić najwięcej węzłów. Najwolniej działała heurystyka 5 i 2. Różnicę w efektywności możemy wytłumaczyć stopniem zaawansowania heurystyki. Tym bardziej zaawansowana heurystyka tym dłuższy czas potrzebny na jej wykonanie.

Sprawdziliśmy też która z heurystyk radzi sobie najlepiej w pojedynkach.

Gracz 1 \ Gracz 2	Heurystyka 1	Heurystyka 2	Heurystyka 3	Heurystyka 4	Heurystyka 5
Heurystyka 1	X	Heurystyka 2	Heurystyka 3	Heurystyka 4	Heurystyka 5
Heurystyka 2	Heurystyka 2	X	Heurystyka 2	Heurystyka 2	Heurystyka 2
Heurystyka 3	Heurystyka 1	Heurystyka 2	X	Heurystyka 4	Heurystyka 5
Heurystyka 4	Heurystyka 4	Heurystyka 2	Heurystyka 4	X	Heurystyka 5
Heurystyka 5	Heurystyka 5	Heurystyka 2	Heurystyka 5	Heurystyka 5	X

Tabela przedstawiająca która heurystyka była lepsza w pojedynku dwóch graczy dla algorytmu alfa-beta cięcie i maksymalnej głębokości drzewa równej 3

Widzimy, że heurystyka 2 wygrała każdy pojedynek co znaczy, że jest najlepsza spośród wszystkich heurystyk. Na drugim miejscu jest heurystyka 5. Na ostatnich miejscach jest heurystyka 3 i heurystyka 1.

Warto tutaj zauważyć, że heurystyka 3 była jedną z najgorszych heurystyk mimo nie najkrótszego czasu potrzebnego na przeglądnięcie jednego węzła. Wynika to z tego, że dana heurystyka implementuje strategię, która jest dobra tylko w niektórych sytuacjach.

Mimo tego, że heurystyka 2 wypadła najlepiej to można zauważyć dwie wady w jej stosowaniu:

a) AI grające z użyciem tej heurystyki byłoby bardzo trudne do pokonania. Może to więc potencjalnie zniechęcać graczy grających w grę
b) AI grające z użyciem tej heurystyki będzie potrzebowało dużo czasu na wykonanie ruchu. Nie każdy gracz ludzki chce czekać aż AI zdecyduje się wykonać ruch.

Warto więc pamiętać, że heurystyka grająca najlepiej nie zawsze będzie najlepszą heurystykę do użycia, szczególnie jeżeli planujemy ją zaimplementować w np. grze komputerowej lub innym komercyjnym produkcie gdzie zależy nam na satysfakcji użytkownika.

Podsumowanie

W ramach zadania zaimplementowaliśmy algorytm Minimax i alfa-beta cięcie. Wyniki pokazały, że alfa-cięcie działa lepiej od algorytmu Minimax. Warto jednak pamiętać, że mogą być sytuacje w których alfa-beta cięcie będzie działało gorzej od Minimaxa. Jednak w większości wypadków algorytm alfa-beta cięcie będzie działał bardziej efektywnie od Minimaxa

Do alfa-beta cięcia można wprowadzać różnego rodzaju modyfikacje np. sortować węzły tak, żeby wpierw przeglądać węzły, które wydają się najbardziej atrakcyjne. Warto jednak pamiętać, że nie zawsze modyfikacja poprawi działanie algorytmu. Trzeba więc wpierw przeanalizować czy wprowadzone zmiany w kodzie wpłyną na efektywność algorytmu.

W ramach zadania mieliśmy też zaimplementować heurystyki. Podczas implementacji heurystyk warto pamiętać w jakim celu je implementujemy. Jeżeli naszym celem jest stworzenie najlepszego gracza komputerowego do gry to wówczas powinniśmy dążyć do implementacji jak najlepszej heurystyki. Jeżeli naszym celem jest stworzenie AI do gry z

graczem to powinniśmy dążyć do implementacji heurystyk, które nie tylko będą szybkie, ale też dadzą sensowne szanse na zwycięstwo graczowi.