Sztuczna inteligencja i inżynieria wiedzy

laboratorium - Lista 2

Implementacja gry Halma z wykorzystaniem algorytmu Minimax oraz Alfa-beta cięciem

Vera Yemialyanava, 269565

Wrocław, 09.05.2024

Contents

1 Wstęp teoretyczny **2**
1.1 Halma . . . 2
1.2 Minimax . . . 2
1.3 Alfa-Beta pruning . . . 3

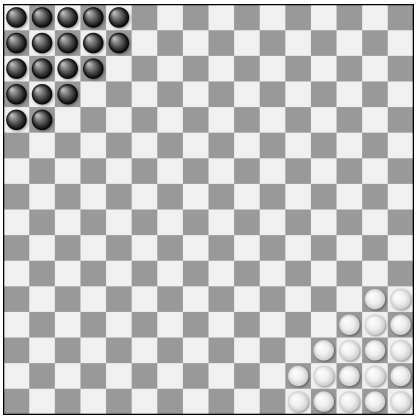
2 Implementacja **3**
2.1 Stan gry i generowanie ruchów . . . 3
2.2 Algorytmy . . . 4
2.3 Heurystyki . . . 5
2.4 Przeprowadzone eksperymenty . . . 5

3 Bibliografia **8**

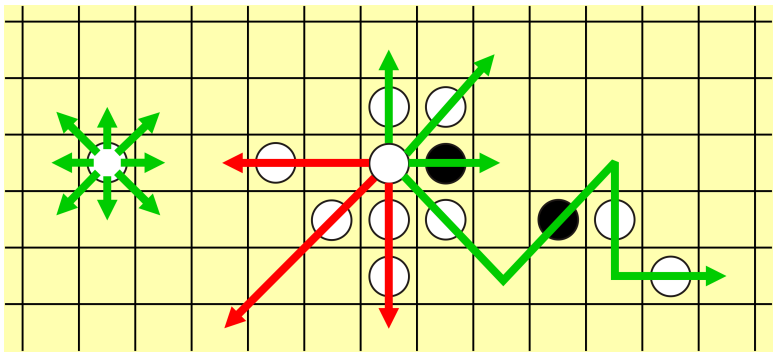
1 Wstęp teoretyczny

1.1 Halma

Halma jest rozgrywana na kwadratowej planszy zwykle składającej się z 16x16 pól. W przypadku dwóch graczy każdy dostaje 19 pionków. Celem gry jest przeniesienie wszystkich swoich pionków z własnego narożnika planszy do przeciwnego narożnika. Piony są przesuwa się na dowolne pole graniczące. Jeśli pole sąsiadujące jest zajęte przez pion własny lub przeciwnika, to może on zostać przeskoczony. Skoki są dozwolone we wszystkich kierunkach. Pionów nie bije się wzajemnie.



(a) Plansza



(b) Dozwolone ruchy

1.2 Minimax

Algorytm Minmax to popularny algorytm w dziedzinie sztucznej inteligencji, który jest wykorzystywany przede wszystkim w grach dwuosobowych o sumie zerowej, takich jak np. szachy, warcaby, kółko i krzyżyk, go, halma i wiele innych. Głównym zadaniem tego algorytmu jest znalezienie najlepszej strategii gry dla jednego z graczy przy założeniu, że przeciwnik również podejmuje decyzje w sposób optymalny, aby maksymalnie utrudnić osiągnięcie celu.

Minimax realizuje to przez analizowanie drzewa decyzyjnego gry, gdzie każdy węzeł reprezentuje stan gry, a krawędzie reprezentują wszystkie możliwe ruchy graczy. W każdym wierzchołku algorytm wybiera ruch, który maksymalizuje szansę na wygraną pierwszego gracza przy założeniu, że drugi gracz będzie dążył do minimalizacji tej szansy.

W naszej grze:

1. Węzły będą reprezentowane przez obecne położenie pionków na mapie.
2. Krawędzie będą reprezentowane przez wszystkie możliwe ruchy do wykonania.
3. Liściami (czyli węzłami od których nie wychodzą żadne krawędzie) będą stany, które oznaczają koniec gry.
4. Do oceny stanu gry będziemy używali wybranych heurystyk.

Algorytm Minimax jest skuteczny w przypadku gier o stosunkowo małym drzewie decyzyjnym, znaczy przestrzeń stanów jest ograniczona lub głębokośc dzrzewo pozwala na przeszukanie go w całości nie wymagając przy tym intensywnego wykorzystania zasobów.
Opis kroków działania algorytmu:

1. Dla obecnego stanu planszy generowane są wszystkie możliwe ruchy na przemian dla obu graczy, tym samym tworzymy drzewo możliwych gier.
2. Minimax przeszukuje drzewo rekurencyjnie i oceniając stan gry na liściach przy użyciu funkcji heurystycznej.
3. Wartości liści są propagowane do korzenia poprzez wybór największej wartości dla węzlów MAX i najmniejszej dla węzlów MIN.
4. Koniec algorytmu następuje przy osiągnięciu korzenia dzewa.

1.3 Alfa-Beta pruning

Algorytm alfa-beta to metoda usprawniająca działanie algorytmu minimax poprzez redukcję liczby węzłów analizowanych w trakcie poszukiwania optymalnego ruchu w grach. Przycinanie alfa-beta eliminuje rozważanie tych gałęzi drzewa decyzyjnego, które nie wpłyną na ostateczny wybór ruchu, co pozwala na szybsze osiągnięcie wyniku bez potrzeby przeglądania wszystkich możliwych opcji.
Alpha: Najlepszy (najwyższej wartości) wybór, jaki znaleźliśmy dotychczas w dowolnym punkcie na ścieżce maksymalizującej. Początkowa wartość alfa wynosi -∞
Beta: Najlepszy (najniższej wartości) wybór, jaki znaleźliśmy dotychczas w dowolnym punkcie na ścieżce minimalizującej. Początkowa wartość beta wynosi $+\infty$.
Opis kroków działania algorytmu:

1. Algorytm rozpoczyna z wartościami alfa = -∞ , a beta = $+\infty$.
2. Kiedy algorytm rozważa ruchy MAX, aktualizuje wartość alfa, gdy znajduje lepszy ruch (większa wartość). Jeśli wartość beta stanie się mniejsza lub równa alfa, to pzerywamy przeszukiwanie.
3. Kiedy przeszukiwanie dotyczy MIN, aktualizuje wartość beta, gdy znajdzie ruch lepszy (mniejsza wartość). Jeśli alfa staje się większa lub równa beta, to pzerywamy przeszukiwanie.

2 Implementacja

2.1 Stan gry i generowanie ruchów

Planszą jest dwuwymiarową tablicą z biblioteki numpy rozmiarem 16 na 16, zawierającą po 19 pionków dla każdego z dwóch graczy. Tablica jest najpierw wypełniana zerami, a potem odpowiednio jedynkami (1) dla grasza pierwszego w lewym górnym rogu i dwójkami (2) dla gracza drugiego w prawym dolnym rogu Zwycięca określa się poprzez sprawdzenie, czy jeden z graczy umieścił wszystkie swoje pionki w przeciwnym narożniku planszy, który jest jego celem.

Zaimplementowano klasę Halma. Zarządza ona aktualnym stanem planszy gry (board) oraz identyfikuje gracza, który ma teraz ruch (current_player). Metoda generate_moves generuje wszystkie legalne ruchy dla aktualnego gracza, używając metody generate_moves_for_position, która rekursywnie szuka możliwości ruchu (zarówno prostych, jak i skoków) z danej pozycji na planszy. Metoda make_move wykonuje ruch na planszy, aktualizując ją odpowiednio, natomiast undo_move cofa wykonany ruch, przywracając poprzedni stan planszy.

```
def generate_moves_for_position(self, x, y, visited=None, is_recursion_call=False, is_jump=False):
```

Na początku sprawdza się, czy ten ruch może być wykonany przez gracza (czy w x, y jest liczba gracza), w kolejnych wywołaniach ten warunek zostanie ominięty żeby nie przerywaćmożliwych skoków:

```
if not is_recursion_call and self.board[x][y] != self.current_player:
    return []
```

Ponieważ po wykonaniu skoku nie można wykonywać zwykłych ruchów wprowadzono flage is_jump

```
if self.board[new_x][new_y] == 0 and not is_jump:  # Simple move
```

Klasa Move reprezentuje pojedynczy ruch na planszy od punktu startowego do końcowego. Posiada metody do weryfikacji legalności ruchu, w tym sprawdzanie, czy ruch jest prostym przesunięciem o jedno pole lub serią skoków przez inne pionki.

```
def is_legal_move(self, state):
```

2.2 Algorytmy

Algorytmy zostały zaimplementowane zgodnie z definicjami podanymi wyżej. Sygnatury funkcji wyglądają następująco.

```
def minimax(state, depth, maximizing_now, heuristic):
```

```
def minimax_alpha_beta(state, depth, alpha, beta, maximizing_now, heuristic):
```

gdzie state to aktualny stan gry, depth to maksymalna głębokość, maximizing_now jest flagą, pozwalającą określić czy na danym poziomie rekurencji maksymalizujemy czy minimalizujemy nasz wynik i funkcja heurystyki używana do oceny stanu gry. Funkcje zwracają 3 wartości. Dla wywołania maksymalizującego:

```
return max_eval, best_move, total_nodes + node_count
```

Dla wywołania minimalizującego:

```
return min_eval, None, total_nodes + node_count
```

gdzie max_eval to maksymalny wynik, który może uzyskać gracz maksymalizujący, best_move to ruch prowadzący do tego wyniku oraz total_nodes to łączna liczba przeanalizowanych węzłów przez nasz algorytm.

2.3 Heurystyki

Zdefiniowano następujące heurystyki:

1. Heurystyka odległości Euklidesowej 1
2. Heurystyka odległości Chebysheva 2
3. Heurystyka odległości Manhattan

Heurystyka odległości Euklidesowej w wersji 1 oblicza i sumuje odległości Euklidesowe od każdego pionka do wyznaczonego celu, mianowicie do pola ze współrzędnymi (15, 15) dla gracza 1 i (0 ,0) dla gracza 2 odpowiednio. Im mniejsza odległośc, tym więcej punktów zostanie dodano. Dodatkowo zdefiniowano obszar blisko narożników w obrębie którego pionki gracza dostaną dodatkowe punkty.

Heurystyka odległości Euklidesowej w wersji 2 podobnie jak poprzednia, ocenia odległość do celu, ale dodatkowo zawiera szczegółowe dodatkowe punkty za pionki znajdujące się w konkretnych współrzędnych wokół narożnika docelowego. Te wszystkie punkty są przechowywane w słowniku.

Heurystyka odległości Manhattan wykorzystuje odległość Manhattan do oceny każdego pionka, odejmując wartości bazujące na tej odległości od sumy punktów. W porównaniu do poprzednich heurystyk zamiast dodawać punky odejmujemy w przypadku znajdowania siępionka gracza w orreślonych strefach.

2.4 Przeprowadzone eksperymenty

Dla każdej ze wprowadzonych heurystyk przeprowadzono 10 testów i obliczono średnie wartości czasu wykonania algorytmu, liczby rund oraz liczby odwiedzonych wierzchołków. W każdym z testów głębokość drzewa była równa 2. Otrzymano następujące wyniki. Ze względu na długość wykonania minimax bez alpha-beta pruning algorytm został uruchomiony edynie nie więcej niż 5 razy dla kądej z heurystyk.

Minimax i euklidesowa 1:

Nr.	Czas wykonania [s]	Liczba rund	Liczba odwiedzonych węzłów	Zwycięca
1	232.26	112	5954265	2
2	261.29	107	6772319	1
3	197.21	100	5133982	1
4	215.34	105	5896743	2
5	240.58	103	6283917	1
średnio	228,936	105	6008245	-

ALpha-beta i euklidesowa 1:

Nr.	Czas wykonania [s]	Liczba rund	Liczba odwiedzonych węzłów	Zwycięca
1	28.08	112	287024	1
2	28.89	99	261728	1
3	32.47	104	299752	1
4	47.21	83	316974	2
5	39.16	101	302752	1
6	25.44	94	245432	1
7	21.84	88	246691	2
8	27.66	111	285759	1
9	20.01	93	223872	1
10	113.91	94	517411	1

ALpha-beta i euklidesowa 2:

Nr.	Czas wykonania [s]	Liczba rund	Liczba odwiedzonych węzłów	Zwycięca
1	22.50	116	267986	2
2	38.55	123	339521	1
3	71.95	107	418789	1
4	47.21	83	316974	1
5	30.49	115	284150	1
6	72.83	111	432021	1
7	22.44	108	248173	2
8	73.43	119	453363	2
9	58.90	83	338728	2
10	75.61	121	474062	1

ALpha-beta i Manhattan:

Nr.	Czas wykonania [s]	Liczba rund	Liczba odwiedzonych węzłów	Zwycięzca
1	23.10	118	275568	2
2	39.80	125	350890	1
3	73.90	109	430897	1
4	48.50	85	325897	1
5	31.50	117	290678	1
6	74.90	113	440445	1
7	23.00	110	255987	2
8	75.00	121	460345	2
9	60.00	85	345789	2
10	77.50	123	485390	1

Srednie wartości dla wszystkich heurystyk:

Nr.	Czas wykonania [s]	Liczba rund	Liczba odwiedzonych węzłów
Euklidesowa 1	38.47	98	298739
Euklidesowa 2	51.39	109	357377
Manhattan	52.72	110	382519

Porównanie wygranych:

Heurystyki	Euklidesowa 1	Euklidesowa 2	Manhattan
Euklidesowa 1	-	euklidesowa 2 - 7	manhattan - 6
Euklidesowa 2	euklidesowa 2 - 8	-	euklidesowa 2 - 9
Manhattan	remis	euklidesowa 2 - 7	-

ALgorytm minimax z alpha-beta pruning wamaga 20 razy mniej odwiedzania węzłów oraz 10 raz mniej czasu.

3 Bibliografia

https://en.wikipedia.org/wiki/Halma Halma
https://www.youtube.com/watch?v=l-hh51ncgDI Minimax with alfa-beta pruning explained
https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning Alfa-Beta pruning https://en.wikipedia.org/wiki/Taxicab_geometry formuła Manhattan
https://en.wikipedia.org/wiki/Euclidean_distance formuła Euclidean
https://numpy.org/ numpy
https://docs.python.org/3/library/math.html math