Sierakowski Adam, 246662
17.05.2021, poniedziałek 17:05

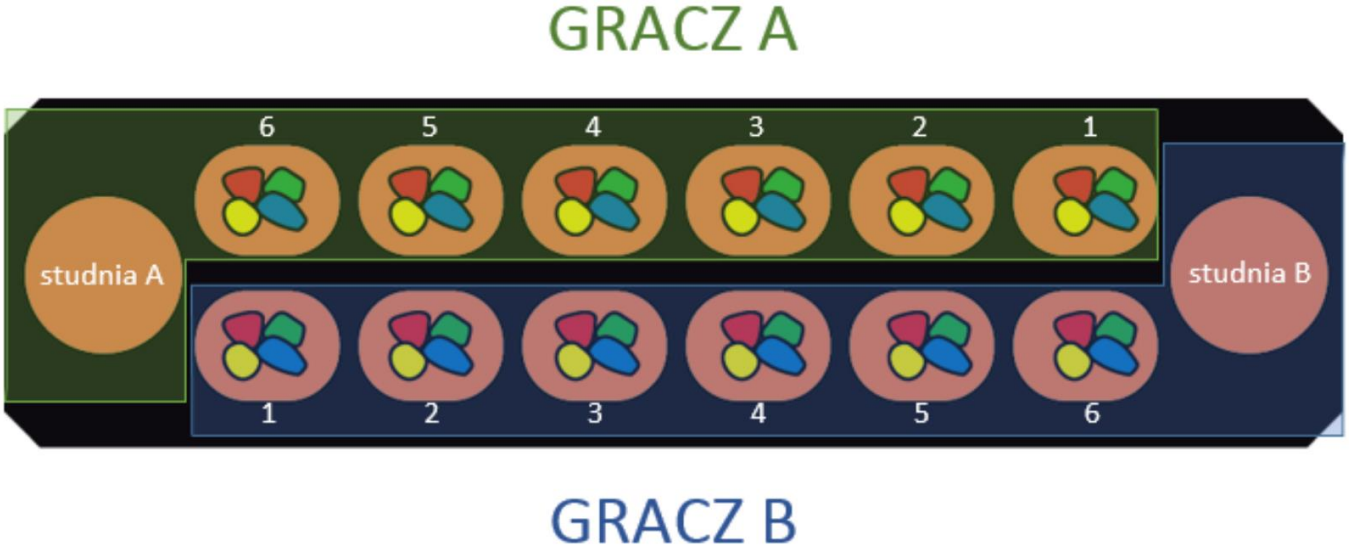
Gry Logiczne i algorytmy grające
Minmax vs Aplha-beta cięć
oraz heurystyki oceny aktualnej sytuacji na planszy

Spis treści

1. Przedstawiony problem

Omawianym problem dotyczy wykorzystania sztucznej inteligencji w grach logicznych takich jak kółko i krzyżyk, szachy, itd. W tym badaniu przyjrzymy się działaniu sztucznej inteligencji w grze Mancala

1.1. Omówienie gry



Rysunek 1 Źródło https://www.ymimports.com/pages/how-to-play-mancala

Zasady:

Losowany jest gracz rozpoczynający. W swojej kolejce gracz wybiera jeden z kontrolowanych przez siebie niepustych dołków. Z tego dołka usuwa wszystkie kamienie, a następnie „rozsiewa" je po jednym we wszystkich następnych dołkach, przeciwnie do ruchu wskazówek zegara (włącznie z własną studnią). Studnię gracza przeciwnego w swoim ruchu gracz pomija.

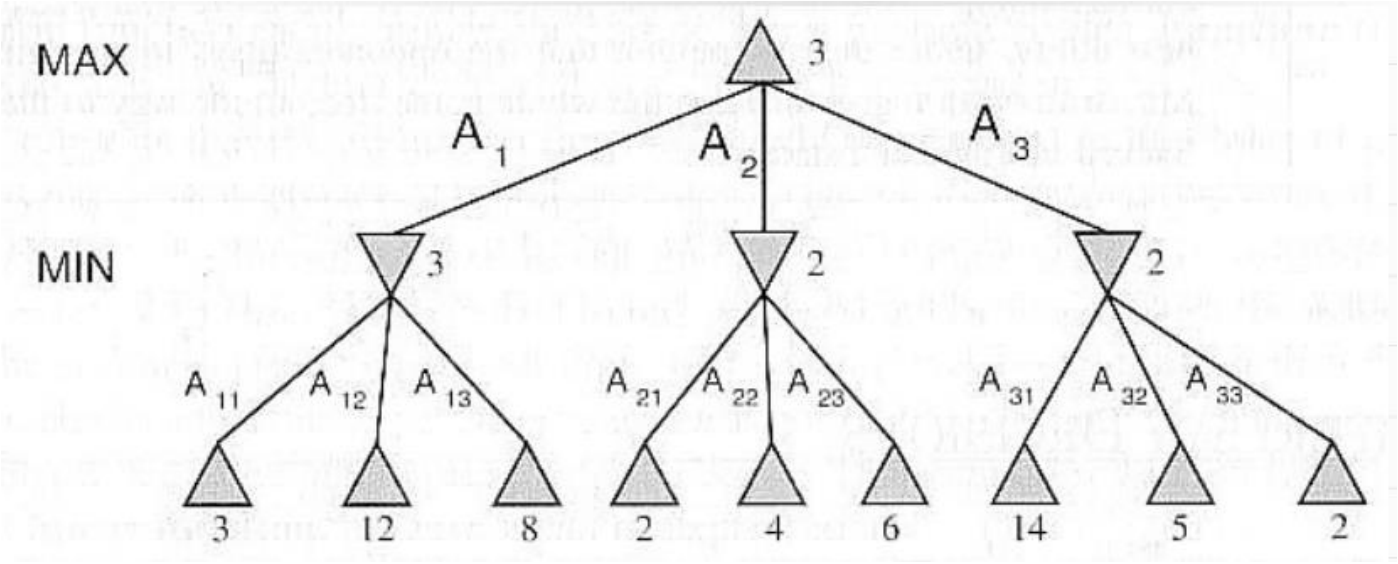
Przykłady:

- gdyby grę rozpoczynał gracz B, mógłby wybrać dołek 4, co spowodowałoby umieszczenie po jednym kamieniu w jego dołku 5 i 6, w jego studni, oraz w dołku 1 gracza A;
- gdyby w swojej kolejce gracz A wybrał dołek 6, a ten zawierał 8 kamieni, zostałyby one rozsiane następująco: pierwszy do studni A, następnie po jednym kamieniu do każdego dołka gracza B, a ostatni kamień do dołka 1 gracza A (studnię B pomijamy).
- Jeżeli ostatni rozsiewany przez gracza kamień trafia do jego własnej studni, gracz ten musi wykonać ponowny 2 ruch (tzn. znowu wybrać dołek i rozsiać kamienie). Jeżeli ostatni rozsiewany przez gracza kamień trafia na jego własny pusty dołek, a przeciwległy dołek przeciwnika nie jest pusty, gracz zabiera wszystkie kamienie z przeciwległego dołka wraz z ostatnim rozsianym przez siebie kamieniem i umieszcza je wszystkie w swojej studni (zachodzi bicie). Bicie zawsze kończy kolejkę gracza.
- Gra kończy się, gdy któryś z graczy nie może wykonać swojej kolejki, tzn. wszystkie jego dołki są puste w momencie, gdy musi wykonać ruch. Wtedy wszystkie pozostałe na planszy (tj. nie w studniach) kamienie trafiają do studni przeciwnika.
- Zwycięzcą jest gracz mający więcej kamieni w swojej studni.
- Status gry: nierozwiązana, ale gracz pierwszy prawdopodobnie ma przewagę. Z tego względu czasami wprowadza się dodatkowe reguły wyrównujące szanse obu zawodników (np. pie rule).

Źródło PDF zawierający instrukcje do ćwieczenia. Jacek Gruber, Przemysław Dolata, Bartosz Perz

1.2. Wykorzystywane algorytmy oraz ich podstawowe działanie

- **Minmax** – algorytm tworzy drzewo możliwych ruchów i wybiera na zmianę, nasz najlepszy ruch (**max**) oraz najlepszy ruch przeciwnika (**min**). W Mancali istnieje możliwość wielokrotnego ruchu tego samego gracza, w mojej implementacji w takim przypadku, ruch przeciwnika jest ruchem pustym (nie zmienia sytuacji na planszy)
- **Alfa-beta cięć** – działa bardzo podobnie jak algorytm **minmax**, z tą różnicą, że pomija niepotrzebne przeszukiwanie poddrzew, gdy i tak wynik nie będzie lepszy od dotychczas napotkanego. Np.



PDF z wykładu czwartego „Gry logiczne – klasyczne wyzwanie, algorytmy grające i ich rozwój”. Halina Kwaśnicka, Maciej Piasecki

W tym przypadku rozpoczynamy szukanie od lewego poddrzewa, znajdujemy tam najlepszy ruch przeciwnika dający mu 3 pkt (na tym poziomie szukamy **min**). Następnie przeszukujemy środkowe poddrzewo i znajdujemy od razu wynik 2 pkt (zakładamy że przeciwnik jest inteligentny i to wybierze).

Wiemy wcześniej że już mamy możliwy lepszy dla nas ruch w lewym poddrzewie i jako że **3>2**, to nie musimy sprawdzać reszty poddrzew. W A3 pechowo musimy przeszukać wszystkie dostępne opcje ponieważ zarówno **14** i **5 jest większe od 3**.

Algorytm w teorii powinien mocno ograniczać niepotrzebne przeszukiwanie poddrzew, sprawdźmy teorie w praktyce!

Funkcja badania aktualnego stanu planszy to:

SimpleGetPoint – prosta heurystyka polegająca na liczeniu liczby punktów z studni danego gracza, a następnie odejmująca od tej liczby liczbę kamieni w studni przeciwnika.

2. Wyniki algorytmu minmax.

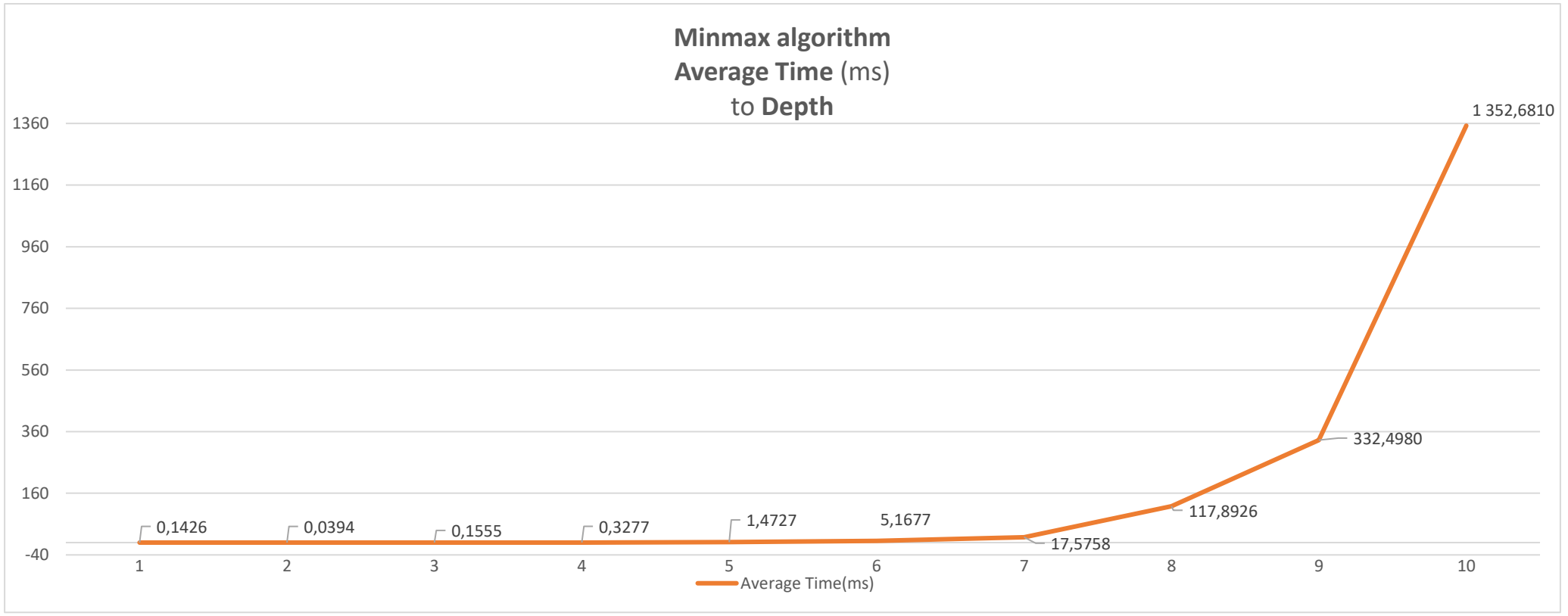
Minmax algorithm (Walczą ze sobą te same głębokości)		
Minmax Ai (Gracz wygrywający, średnia z 6 rozgrywek)		
Depth	**Average Time(ms)**	**Average Number of moves**
1	0,142557777	13,125
2	0,039429072	14,5
3	0,155536032	15,75
4	0,327714958	11,375
5	1,472666769	9,75
6	5,167730011	15,875
7	17,57578214	11,75
8	117,8925758	11,25
9	332,4980385	14,125
10	1352,680969	14,25

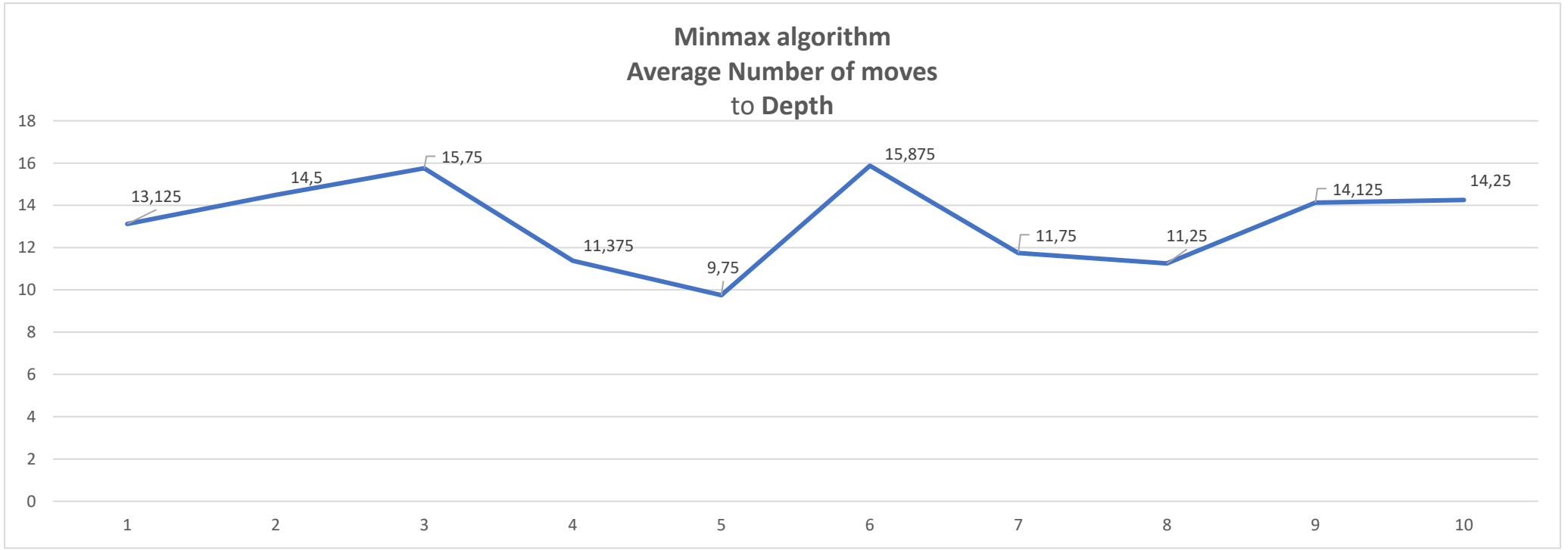
Możemy zaobserwować znaczący wzrost w ilości czasu. Złożoność obliczeniowa w tym przypadku nie jest taka prosta do policzenia, biorąc pod uwagę, że w każdej turze możemy mieć od 1 do 6 możliwych ruchów (nie wspominając o ruchach pominiętych). Możemy zaokrąglić średnią liczbę ruchów do 4 i wtedy złożoność obliczeniowa wynosi:

b – liczba możliwych ruchów

d – głębokość drzewa

O(b^d) = **O(** 4^10 **)** = 1 048 576 **przeszukanych liści**.


Minmax algorithm
Average Time (ms)
to Depth
1360
1160
960
760
560
360
160
-40
1 352,6810
332,4980
117,8926
17,5758
5,1677
1,4727
0,3277
0,1555
0,0394
0,1426
1
2
3
4
5
6
7
8
9
10
Average Time(ms)


Minmax algorithm
Average Number of moves
to Depth
18
16
14
12
10
8
6
4
2
0
13,125
14,5
15,75
11,375
9,75
15,875
11,75
11,25
14,125
14,25
1
2
3
4
5
6
7
8
9
10

3. Wyniki algorytmu **alfa-beta cięć**.

Alpha-beta algorithm (Walczą ze sobą te same głębokości)		
Alpha-beta Ai (Gracz wygrywający, średnia z 6 rozgrywek)		
Depth	**Average Time(ms)**	**Average Number of moves**
1	0,053746968	12
2	0,028581308	14,875
3	0,050710579	14,125
4	0,117902652	14,75
5	0,221296266	15
6	0,428815801	15,125
7	1,435139717	11,375
8	2,4857161	13,625
9	6,647911936	14,375
10	15,58120769	15,5
11	40,63037367	12,375
12	84,68789949	12,375
13	227,7718583	13,125
14	411,7829469	11,875
15	977,8269231	8,25
16	1106,092482	11,875

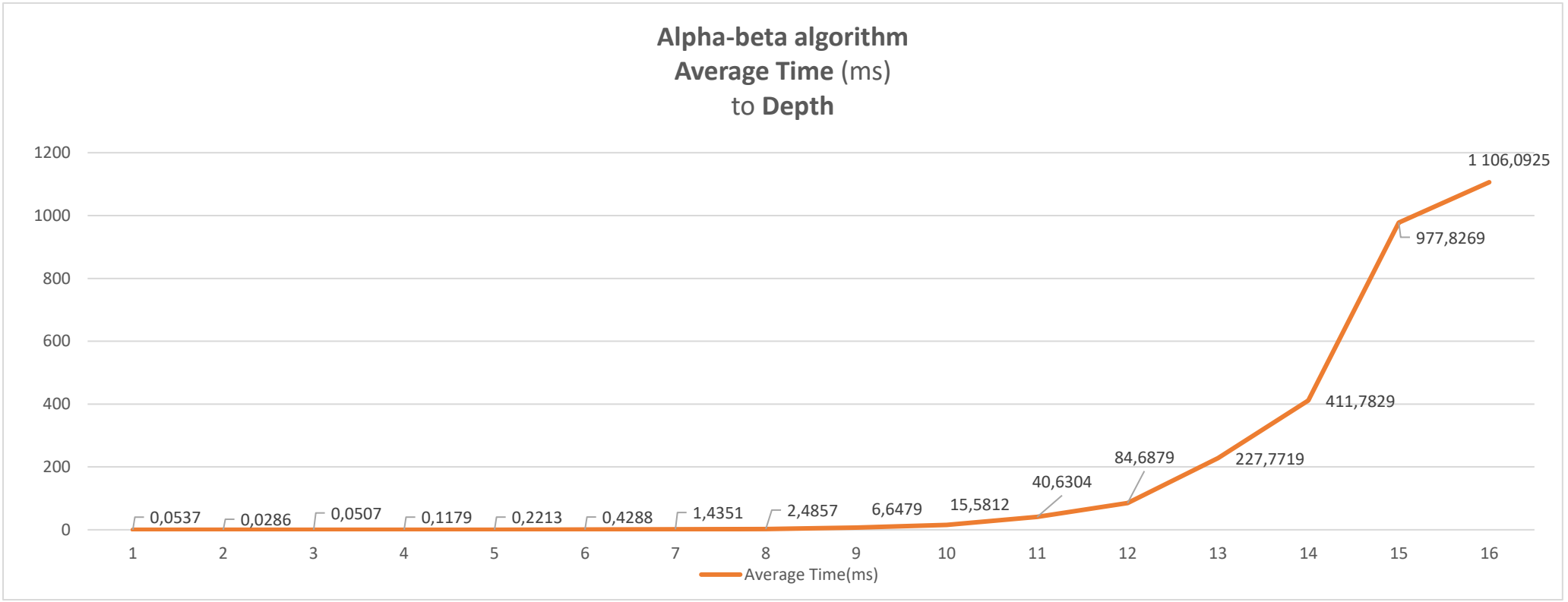
Możemy zaobserwować znaczący wzrost w ilości czasu. Złożoność podobnie jak poprzednio nie jest prostą rzeczą:

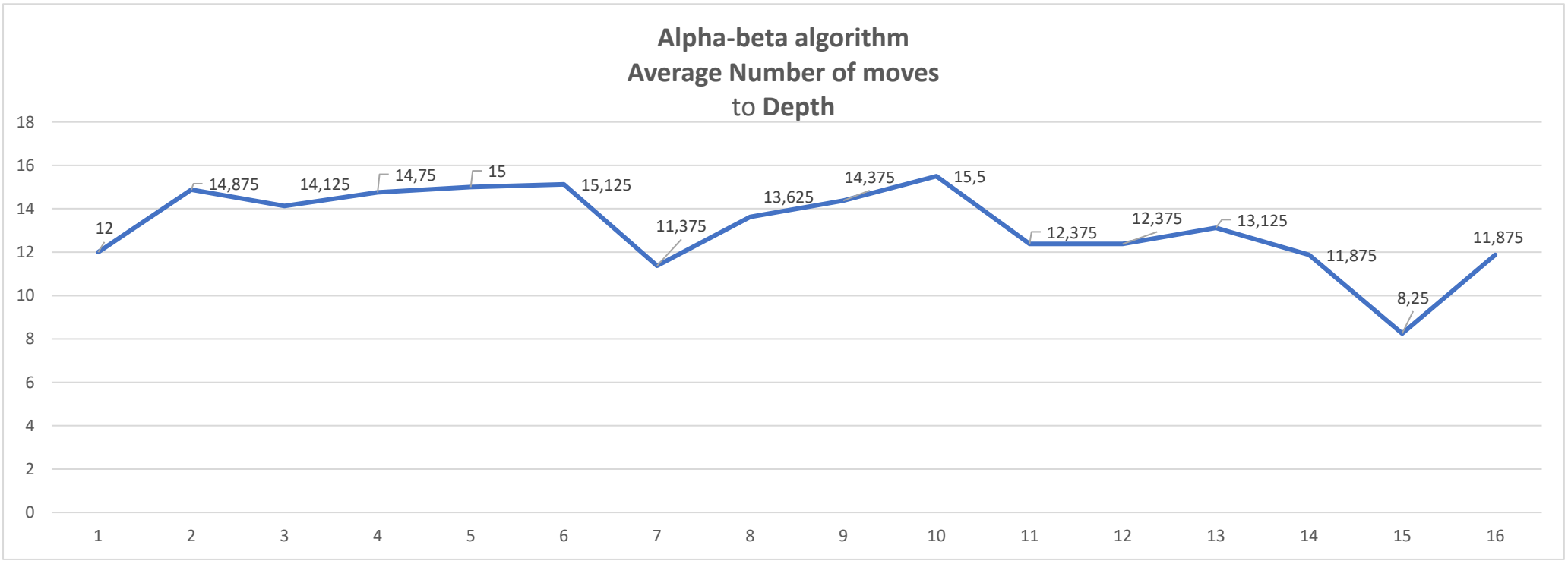
b – liczba możliwych ruchów
d – głębokość drzewa

Alfa-beta Pesymistyczny scenariusz:
O((**b**/log**b**)^**d**) = **O(** 1 048 576 / log^10(4) **)** ≈ 40 000 **przeszukanych liści**. (Wygląda to i tak znacznie lepiej, niż w przypadku **minmax**)

Alfa-beta Optymistyczny scenariusz:
O(b^(**d**/2) **)** = **O(** 4^(10/2) **)** = **O(** 4^5 **)** = 1024 **przeszukanych liści**. (prędkość niewyobrażalnie lepsza niż w **minmax**)

Bardzo ciężko ocenić średni scenariusz, ponieważ bardzo zależy od kolejności przeszukiwanych węzłów.


Alpha-beta algorithm
Average Time (ms)
to Depth
1200
1000
800
600
400
200
0
1 106,0925
977,8269
411,7829
227,7719
84,6879
40,6304
15,5812
6,6479
2,4857
1,4351
0,4288
0,2213
0,1179
0,0507
0,0286
0,0537
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Average Time(ms)


Alpha-beta algorithm
Average Number of moves
to Depth
18
16
14
12
10
8
6
4
2
0
12
14,875
14,125
14,75
15
15,125
11,375
13,625
14,375
15,5
12,375
12,375
13,125
11,875
8,25
11,875
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

4. Minmax vs alfa-beta cięć.

Depth	Minmax Time(ms)	Alpha-beta Time(ms)
1	0,142557777	0,053746968
2	0,039429072	0,028581308
3	0,155536032	0,050710579
4	0,327714958	0,117902652
5	1,472666769	0,221296266
6	5,167730011	0,428815801
7	17,57578214	1,435139717
8	117,8925758	2,4857161
9	332,4980385	6,647911936
10	1352,680969	15,58120769
11	**n/a**	40,63037367
12	**n/a**	84,68789949
13	**n/a**	227,7718583
14	**n/a**	411,7829469
15	**n/a**	977,8269231
16	**n/a**	1106,092482

Algorytmy będę porównywał **tylko na podstawie czasu**, ponieważ gdy posiadają taką samą głębokość to wyniki będą analogiczne. W czasie natomiast gołym okiem widzimy ogromną różnicę. Porównajmy, czy teoretyczna złożoność obliczeniowa sprawdza się w praktyce.

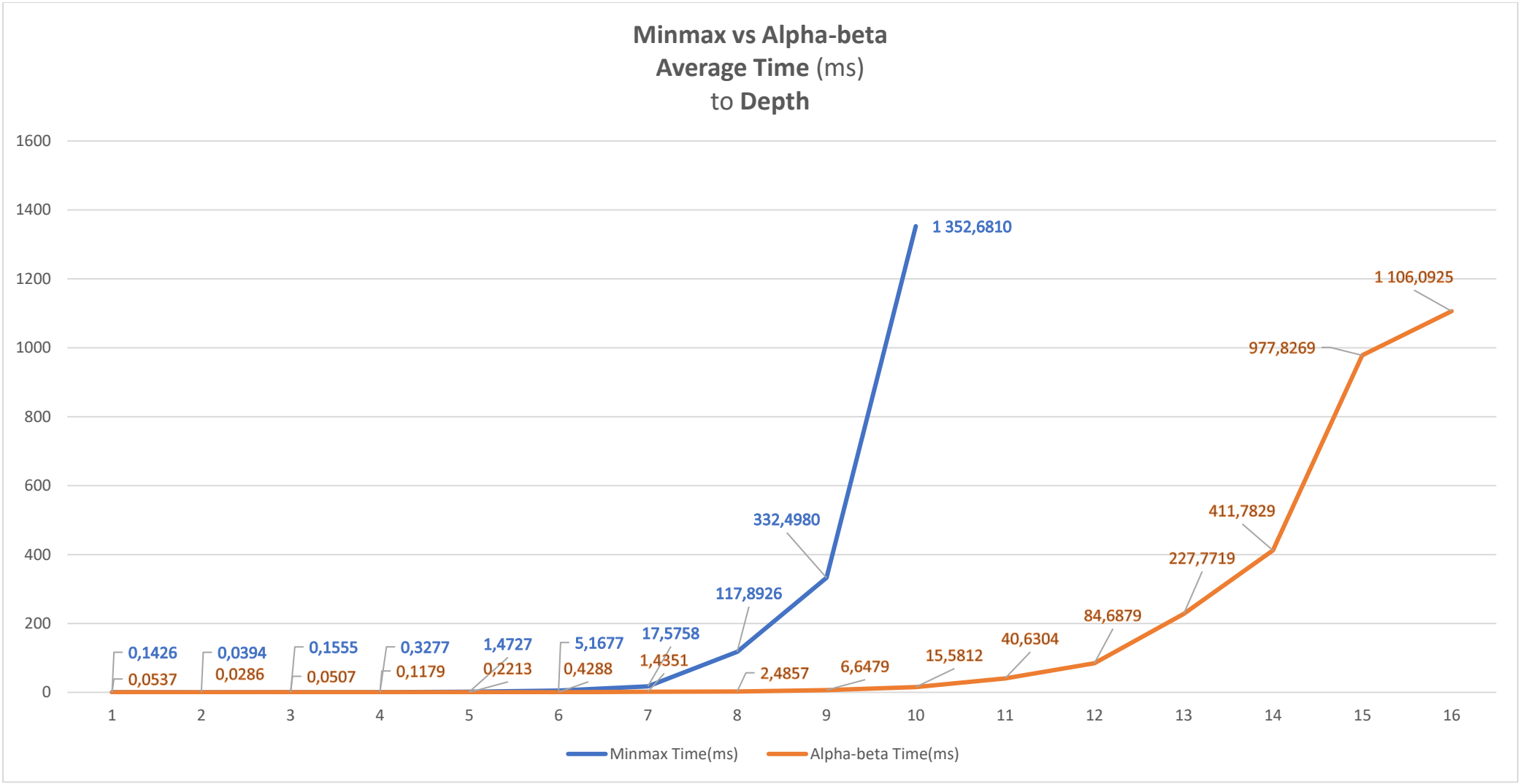
4.1. Porównanie złożoności obliczeniowej

b – liczba możliwych ruchów
d – głębokość drzewa

Minmax:
O(b^d) = **O(** 4^10 **)** = 1 048 576 **przeszukanych liści**.

Alfa-beta cięć:
Alfa-beta Pesymistyczny scenariusz:
O((**b**/log**b**)^**d)** = **O(** 1 048 576 / log^10(4) **)** ≈ 40 000 **przeszukanych liści**. (Wygląda to i tak znacznie lepiej, niż w przypadku **minmax**)

Alfa-beta Optymistyczny scenariusz:
O(b^(**d**/2) **)** = **O(** 4^(10/2) **)** = **O(** 4^5 **)** = 1024 **przeszukanych liści**. (prędkość niewyobrażalnie lepsza niż w **minmax**)



4.2. Minmax vs Alfa-beta – wniosek.

Możemy gołym okiem dostrzec, że **alfa-beta cięć** bije na głowę czasowo zwykły algorytm **minmax**, co zgadza się z złożonością obliczeniową algorytmów.

Wynik praktyczny (z pomiarów) znajduje się w rozsądnym miejscu między **wynikiem pesymistycznym** a **wynikiem** (bardzo) **optymistycznym**.

s

1352,681 / 15,5812 ≈ **86,815 razy szybciej** ← Wynik praktyczny

1 048 576 / 40 000 ≈ **26,2144 razy szybciej** ← Wynik pesymistyczny

1 048 576 / 1024 = **1024 razy szybciej** ← Wynik bardzo (bardzo) optymistyczny

5. Heurystyki oceny planszy z algorytmem Minmax oraz Alfa-beta.

5.1. Wykorzystywane heurystyki:

- **SimpleGetPoint** – prosta heurystyka polegająca na liczeniu liczby punktów z studni danego gracza, a następnie odejmująca od tej liczby liczbę kamieni w studni przeciwnika
- **ExpandedGetPoints** – bardziej skomplikowana heurystyka, która tylko w momencie końca gry działa podobnie jak poprzednio ale dodatkowo:
    - Bierze pod uwagę ułożenie kamieni w studniach punktując dodatkowo kamienie ułożone w sposób sprzyjający do powtórzenia tury (np. w dołku 6 jest 1 kamień, więc jest fajniej, ponieważ możemy wykonać dodatkowy ruch) + dodatkowy punkt, jeżeli pierwszy dołek jest pusty, co pozwala na wykonanie jeszcze dodatkowego ruchu
    - Sumę kamieni po stronie danego gracza z dołków * wagaDołku odjąć liczba kamieni w dołkach drugiego gracza
    - Różnica kamieni w studniach obu graczy * wagaStudni

5.2. Założenia badań heurystyk.

Do testów heurystyk skupie się na algorytmie **Alfa-beta**, nie z lenistwa a z prostego wniosku:

Minmax oraz **Alfa-beta** na danej wysokości dadzą dokładnie taką samą odpowiedź, więc w kwestii % wygranych tych algorytmów na takiej samej głębokości oraz przy takiej samej heurystyce oceny byłby taki sam.
Co do kwestii czasu, tutaj oczywiście bardziej skomplikowana funkcja oceny zajmie dłuższy czas również niezależnie od algorytmu. Różnica polega na tym, że **Minmax** będzie miał stałą liczbę wywołań tej funkcji, więc tutaj czas tylko by się zwiększył. Co innego możemy powiedzieć o **Alfa-beta**, tutaj jest szansa na szybsze porzucanie słabych gałęzi, co możliwe że doprowadzi do krótszego czasu.

Finalnie dokonam porównania czasów tych 4 opcji:

- **Minmax** z **SimpleGetPoint**
- **Minmax** z **ExpandedGetPoints**
- **Alfa-beta** z **SimpleGetPoint**
- **Alfa-beta** z **ExpandedGetPoints**

Zostanie zbadanie wpływu heurystyki na czas, oraz % wygranych.

UWAGA!
Czasy w badaniach mogą się różnić od poprzednich, ponieważ na potrzeby wprowadzania nowej heurystyki, liczenie punktów zamiast zwracać **int** zwraca **double**. Co pociągnęło za sobą parę zmian mających na celu zachowania poprawnego działania programu. Poprzednie badania jednak wciąż mają sens, ponieważ porównanie obu algorytmów miało miejsce w takich samych warunkach.

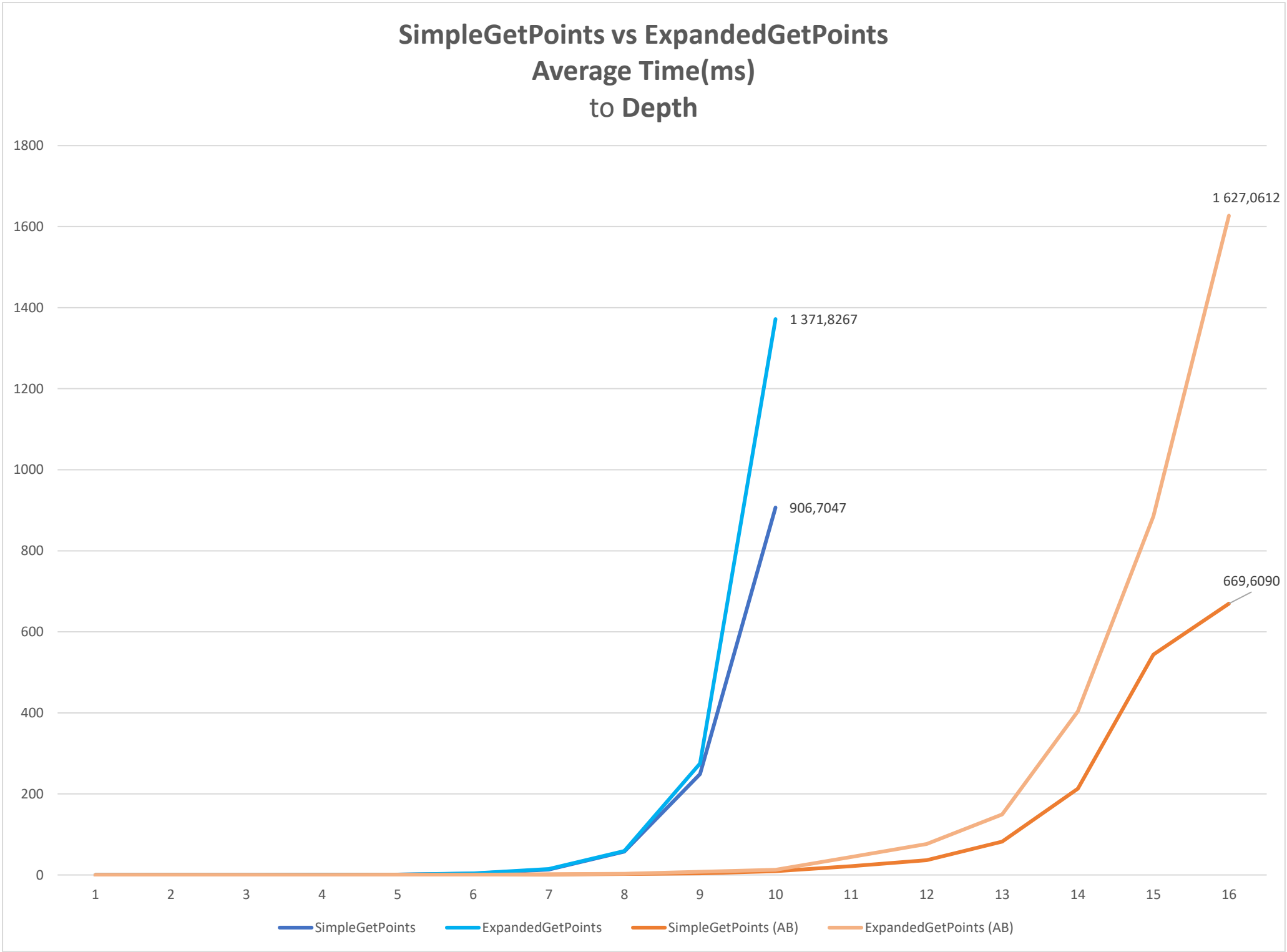
5.3. Wynik badań czasu.

Wpływ czasu.

Depth	Minmax Average Time(ms)		Alpha-beta Average Time(ms)	
	SimpleGetPoints	ExpandedGetPoints	SimpleGetPoints	ExpandedGetPoints
1	0,063933794	0,102456985	0,026831667	0,043905306
2	0,024553215	0,032818576	0,021463906	0,028659344
3	0,080626749	0,091372157	0,03614387	0,059381929
4	0,228292057	0,246582142	0,093287538	0,098790217
5	1,129316675	1,044657426	0,206749794	0,297407547
6	3,426445144	4,02407615	0,430933583	0,472317624
7	13,46229117	14,75469689	0,84832908	1,856468718
8	57,54481768	59,29995684	2,168882949	2,780380347
9	248,7325427	275,2237011	3,926346962	7,931759878
10	906,7047019	1371,826673	9,286348181	12,87343056
11	**n/a**	**n/a**	21,66809892	44,49978069
12	**n/a**	**n/a**	36,23184096	76,58648198
13	**n/a**	**n/a**	82,06748515	149,4861992
14	**n/a**	**n/a**	212,8951131	403,7551705
15	**n/a**	**n/a**	543,7985497	885,0249907
16	**n/a**	**n/a**	669,6089658	1627,061215

Tutaj bez większych zaskoczeń co do czasu w **Minmax**, czas zwiększył się o koszt wykonania trudniejszej heurystyki.

Trochę zdziwił mnie wynik z **Alfa-beta**, ponieważ spodziewałem się ucinania niepotrzebnych gałęzi, a wnioskując po znacznie większej dysproporcji niż przy **Minmax**, więc prawdopodobnie osiągnęliśmy wynikj odwrotny do zamierzonego, jednak pamiętajmy, naszym głównym celem jest wygrana, sprawdźmy czy na tym polu heurystyka się obroni!


SimpleGetPoints vs ExpandedGetPoints
Average Time(ms)
to Depth
1800
1600
1400
1200
1000
800
600
400
200
0
1 627,0612
1 371,8267
906,7047
669,6090
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
SimpleGetPoints
ExpandedGetPoints
SimpleGetPoints (AB)
ExpandedGetPoints (AB)

5.4. Wyniki badań współczynnika zwycięstwa oraz średniej przewagi w punktach.

Zaczynał **SimpleGetPoints** dla 50 rozgrywek

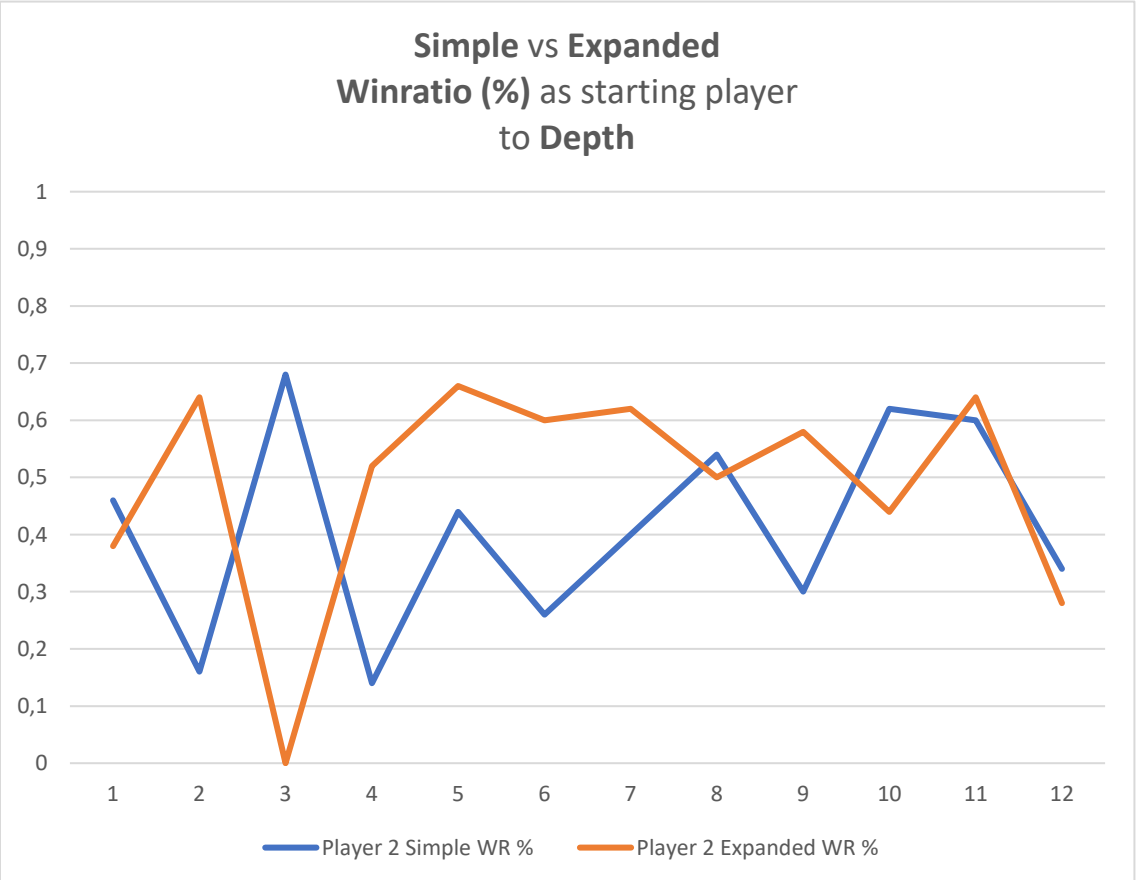
Depth	Player 1 Simple WR %	Player 2 Expanded WR %	AvgLead P1 Simple	AvgLead P2 Expanded
1	0,62	0,38	8,6897	14,0000
2	0,24	0,64	11,4634	6,0000
3	0,88	0	12,6818	2,0000
4	0,38	0,52	20,2857	8,9091
5	0,18	0,66	14,5789	20,0000
6	0,4	0,6	6,6667	4,0000
7	0,38	0,62	6,7857	6,6364
8	0,5	0,5	18,2222	8,8125
9	0,42	0,58	11,4118	5,6800
10	0,36	0,44	12,5714	6,0000
11	0,36	0,64	11,3333	15,4375
12	0,72	0,28	16,6667	13,0000

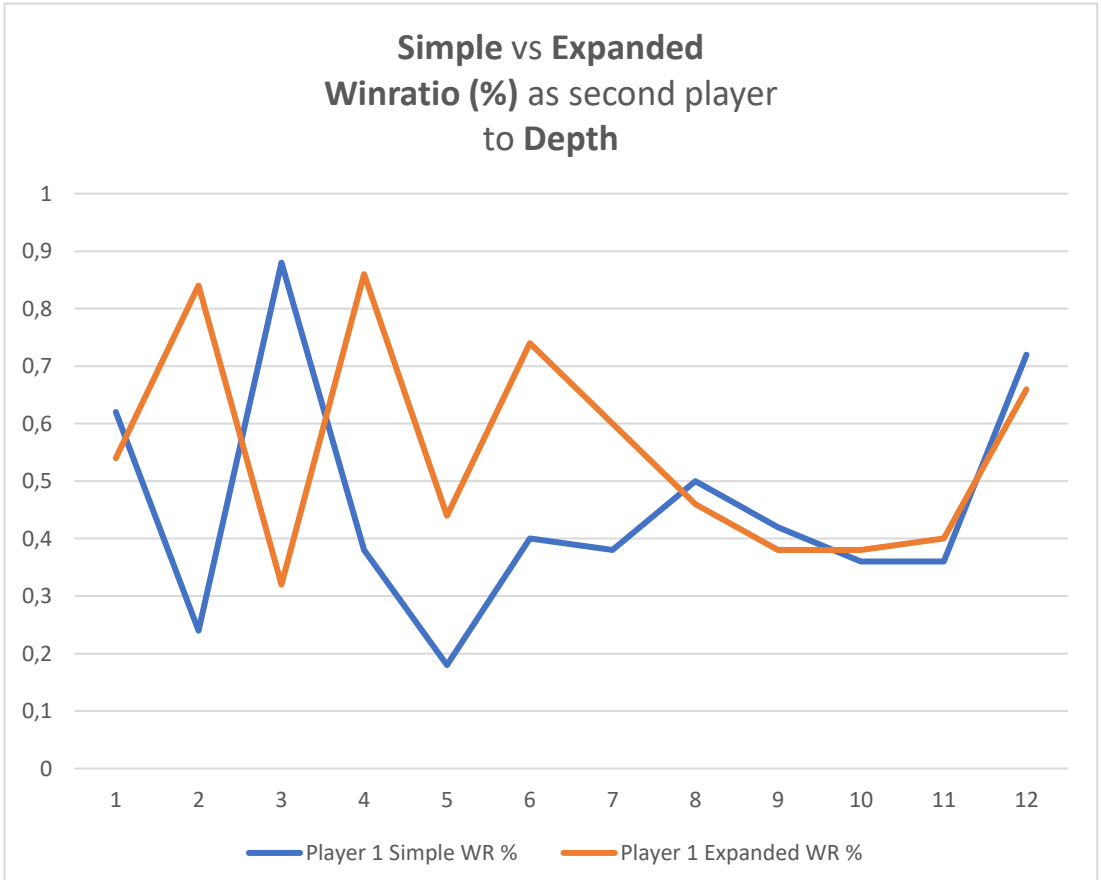
Zaczynał **ExpandedGetPoints** dla 50 rozgrywek

Depth	Player 1 Expanded WR %	Player 2 Simple WR %	AvgLead P1 Simple	AvgLead P2 Expanded
1	0,54	0,46	7,333333333	8,782608696
2	0,84	0,16	9,142857143	10
3	0,32	0,68	29	12,35294118
4	0,86	0,14	9,069767442	4
5	0,44	0,44	7,272727273	8,090909091
6	0,74	0,26	17,62162162	14,76923077
7	0,6	0,4	8,666666667	4,1
8	0,46	0,54	10,26086957	9,703703704
9	0,38	0,3	17,78947368	12,53333333
10	0,38	0,62	20,52631579	7,612903226
11	0,4	0,6	15,8	15,4
12	0,66	0,34	11,27272727	15,52941176

Bezpośrednie porównania heurystyk

Depth	Player 2 Simple WR %	Player 2 Expanded WR %		Player 1 Simple WR %	Player 1 Expanded WR %
1	0,46	0,38		0,62	0,54
2	0,16	0,64		0,24	0,84
3	0,68	0		0,88	0,32
4	0,14	0,52		0,38	0,86
5	0,44	0,66		0,18	0,44
6	0,26	0,6		0,4	0,74
7	0,4	0,62		0,38	0,6
8	0,54	0,5		0,5	0,46
9	0,3	0,58		0,42	0,38
10	0,62	0,44		0,36	0,38
11	0,6	0,64		0,36	0,4
12	0,34	0,28		0,72	0,66





5.5. Wniosek badań heurystyk.

Z przykrością muszę oznajmić, że niestety moja innowacyjna propozycja heurystyki oceny stanu planszy się nie obroniła przed prostotą.

Nie odczuwam, że to źle, ponieważ moim zdaniem prosta heurystyka w cale nie jest taka słaba, jakby można było się spodziewać. Możliwe, że nie wybrałem idealnych parametrów do wag danych, jednak nie jest to łatwe zadanie i możliwe że jakiś algorytm genetyczny przyszedł by tu z pomocą. Dodatkowym faktem jest też z względów czasowych wykonaniu względnie małej ilości gier oraz duża waga losowości pierwszego ruchu.

Podsumowując heurystyki, tym razem wygrała prostota!

SimpleGetPoints > ExpandedGetPoints