Projektowanie i Analiza Algorytmów	
Kierunek *Informatyczne Systemy Automatyki*	**Termin** *Wtorek* $15^{15} - 16^{55}$
Imię, nazwisko, numer albumu *Mikołaj Nowak 280082*	**Data** *17.06.2025*
Link do projektu `https://github.com/devoid-of-thought/Szachy`	

Sprawozdanie Projekt nr 3

Spis treści

1 Wstęp

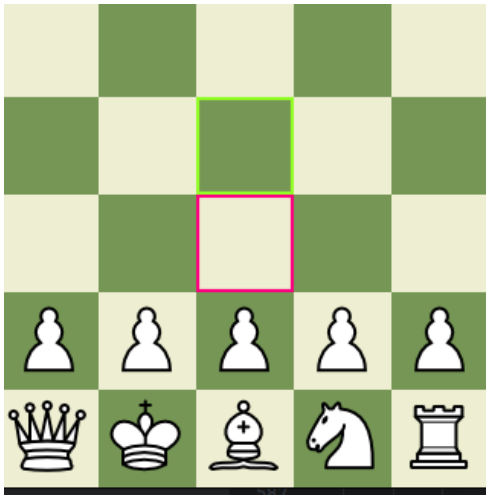
Celem niniejszego sprawozdania jest stworzenie gry komputerowej z wykorzystaniem elementów sztucznej inteligencji. Wybrana gra to szachy. Został zaimplementowany algorytm negamax z alpha-beta pruning. Biblioteka użyta do stworzenia interfejsu użytkownika to SDL3.

2 Opis gry

2.1 Szachy

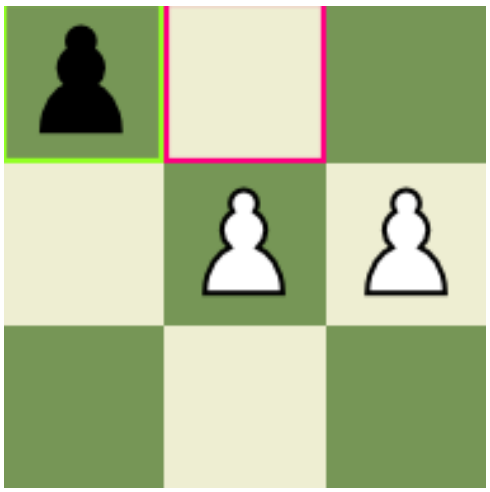
2.1.1 Pionek

- Pionek rusza się jedeno polę do przodu.
- W przypadku gdy jest to jego pierwszy ruch może ruszyć się o dwa pola do przodu.



Rysunek 1: Ruch Pionka

- Bicie odbywa się kiedy figura o przeciwnym kolorze znajduje się o jedno polę do przodu w prawo lub lewo od pionka.

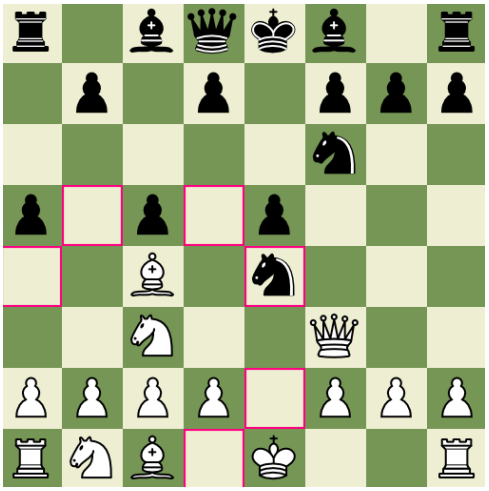


Rysunek 2: Bicie Pionkiem

- Gdy pionek dotrzę do końcowego pola następuje promocja do Hetmana, nie został zaimplementowany wybór figury.
- En passant również nie zostal zaimplementowany.

2.1.2 Skoczek

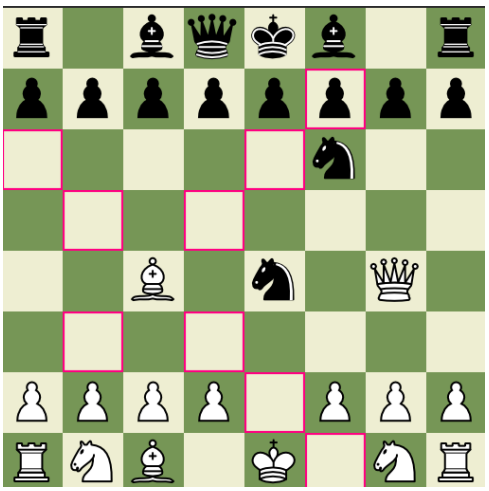
- Skoczek porusza się w kształcie litery L:
- Dwa pola do przodu i jedno w bok lub
- Jedno pole do przodu i dwa w bok lub
- Dwa pola do tyłu i jedno w bok lub
- Jedno pole do tyłu i dwa w bok lub



Rysunek 3: Ruch Skoczka

2.1.3 Goniec

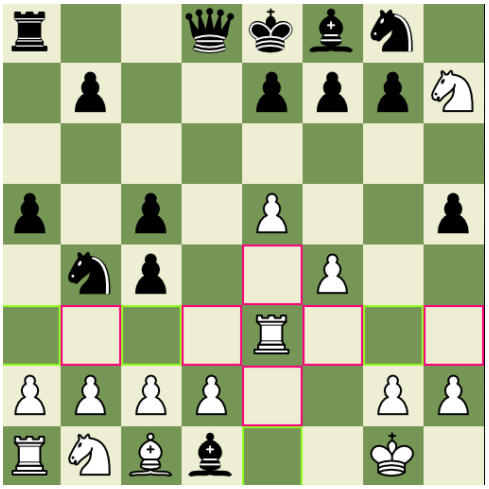
- Goniec porusza się dowolną liczbę pól po skosie



Rysunek 4: Ruch Gońca

2.1.4 Wieża

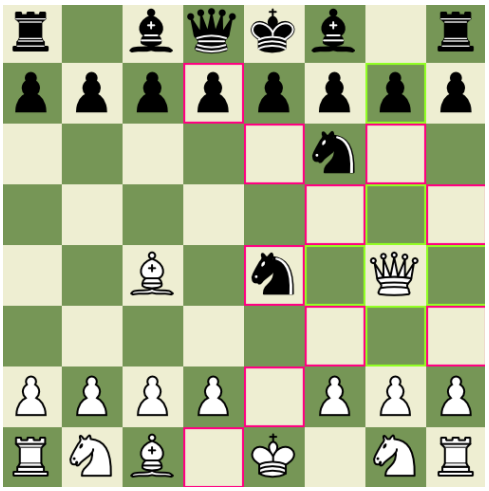
- Wieża porusza się dowolną liczbę pól do przodu lub do tyłu lub w prawo lub w lewo



Rysunek 5: Ruch Wieży

2.1.5 Hetman

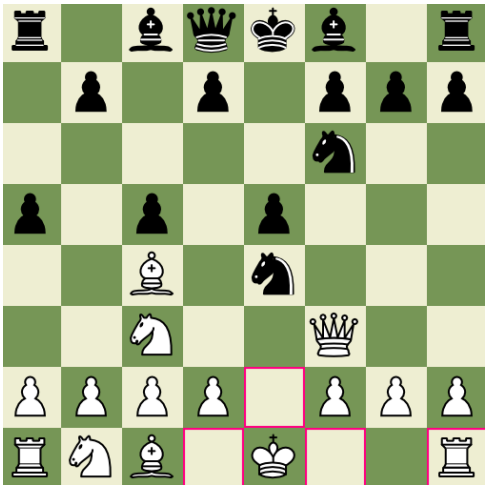
- Hetman łączy możliwości wieży i gońca co czyni go najbardziej wartościową figurą



Rysunek 6: Ruch Hetmana

2.1.6 Król

- Król może poruszać się o jedno polę w każdą stronę, ale nie może poruszyć się na polę poddane atakowi.
- W implementacji roszada obsługiwana jest przez króla. Jeśli zarówno król jak i wieża nie zostały ruszone, pomiędzy nimi nie znajdują się żadne figury oraz nie są atakowane, a król nie podlega szachowi.



Rysunek 7: Ruch Króla

2.2 Algorytm negamax

Algorytm negamax to wersja algorytmu MinMax, któray łączy funkcje Min i Max w jedną. Algorytm działa ponieważ ruch pozytywny dla przeciwnika, będzie negatywnym dla nas. W rekurencyjnym odwołaniu negujemy wynik dziecka. Zastosowanie Alphy i Bety pomgaga ograniczyć rozmiar drzewa. Alpha to najlepszy możliwy ruch jaki możemy zagwarantować w danym momencie, co oznacza że nie ma potrzeby przeszukiwać dalej drzewa. Beta to najlepszy ruch jaki przeciwnik może zagwarantować, więc jeśli znajdziemy gorszy wynyk, to najprawdopodobniej go nie wybierze, co oznacza że nie ma potrzeby przeszukiwać dalej drzewa.Możemy spodziewać się złożoności rzędu $O(b^d)$ lub przez zastosowanie Alpha-Beta Pruning $O(b^{d/2})$, gdzie b to współczynnik drzewiastości, oznaczający ilość dzieci, każdego z rodziców, a d głębokość przeszukiwania

3 Analiza Algorytmu

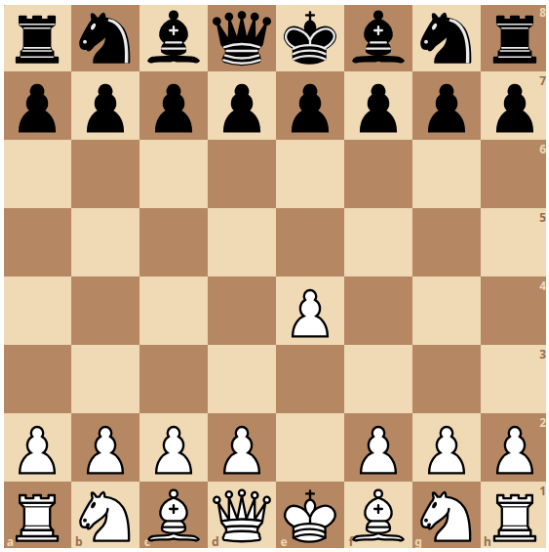
Algorytm został przetestowany dla głębokości 1, 3, 5, 7. Dla głębokości 1 i 3 w celu uśrednienia wyniku zostało wykonane 10 prób, dla głębokości 5 zostało wykonane 5 prób, natomiast dla głębokości 7 ze względu na długi czas wykonywania została wykonana jedynie jedna próba. Dodatkowo dla porównania została przetestowana ta sama wersja algorytmu tylko bez Alpha-Beta Pruning. Algorytm został przetestowany dla 4 plansz (podane w notacji FEN) :

- Pozycja startowa, jedyny wykonany ruch to biały pionek na e4:
  - rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq - 0 1
- Pozycja rozbudowana, gdzie żadna z figur nie została jeszcze zbita, a duża ilość ruchów jest możliwa:
  - rn2kb1r/pp3ppp/2p1pn2/q3Nb2/2BP2P1/2N5/PPP2P1P/R1BQK2R b KQkq - 0 8
- Gra środkowa:
  - 1r3rk1/1pp2ppp/p7/2PPQ3/2n1P3/6P1/5PBP/3R2K1 b - - 0 26

- Gra końcowa:
  - b4r2/p3R1PP/1p5k/2p4P/1P1p2n1/8/P7/6K1 b - - 0 48

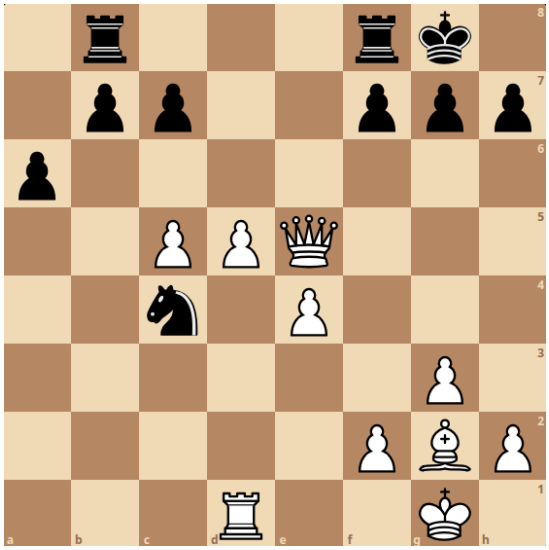
Rysunek 8: Użyte do testowania algorytmów planszr

(a) Pozycja startowa

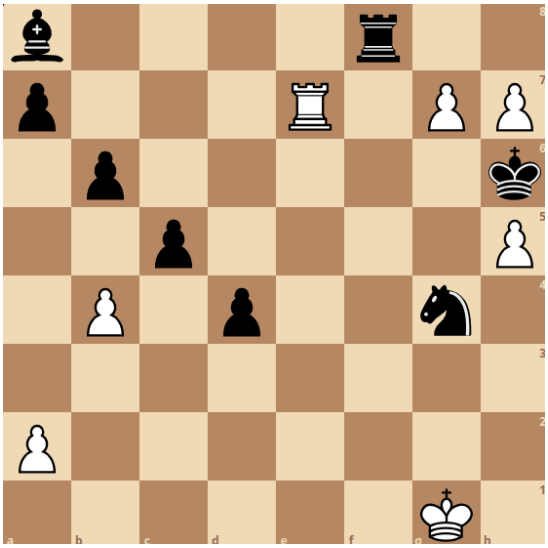


(b) Pozycja rozbudowana

(c) Gra środkowa



(d) Gra końcowa



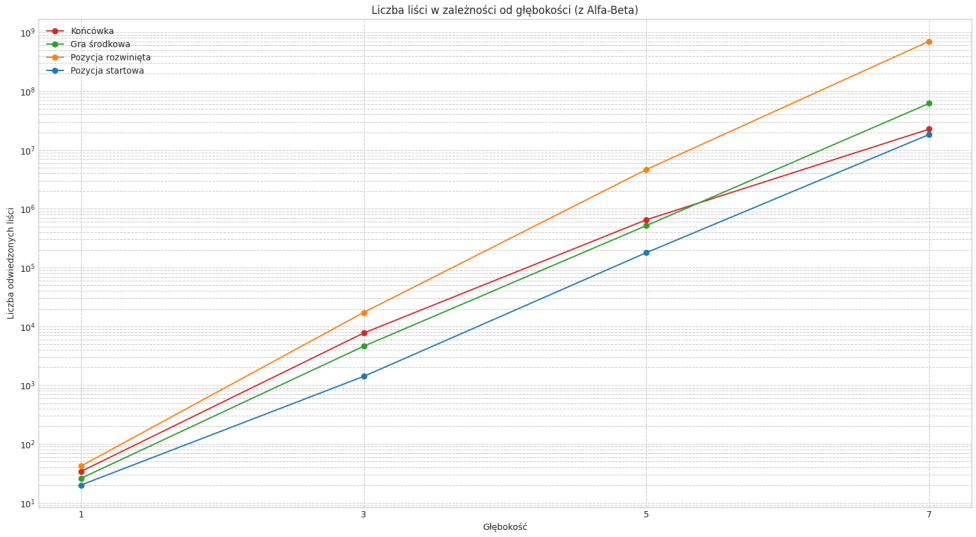
3.1 Liczba Węzłów Końcowych

Tabela 1: Ilość możliwych ruchów na danej głębokości wyszukiwania dla algorytmu z Alpha-Beta Pruning

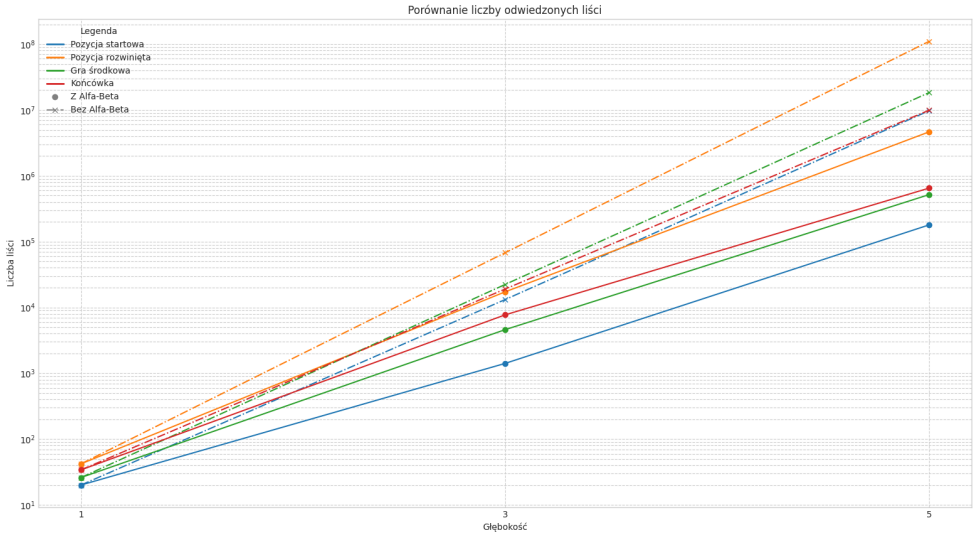
Pozycja	**Głębokość**			
	1	**3**	**5**	**7**
Pozycja startowa	20	825	73662	18347972
Pozycja startowa rozwinięta	42	17246	4658349	706077472
Gra środkowa	26	4593	518454	62053519
Końcówka	33	8549	1034744	22638269

Tabela 2: Ilość możliwych ruchów na danej głębokości wyszukiwania dla algorytmu bez Alpha-Beta Pruning

Pozycja	**Głębokość Przeszukania**			
	1	**3**	**5**	**7**
Pozycja startowa	20	13160	9770281	x
Pozycja rozwinięta	42	67713	110758973	x
Gra środkowa	26	22314	18423490	x
Końcówka	34	19026	9940134	x



Rysunek 9: Ilość liści na danej głębokości wyszukiwania dla algorytmu z Alpha-Beta Pruning



Rysunek 10: Porównanie ilości liści na danej głębokości wykonania dla danej planszy pomiędzy algorytmem z Alpha-Beta Pruning i bez niego

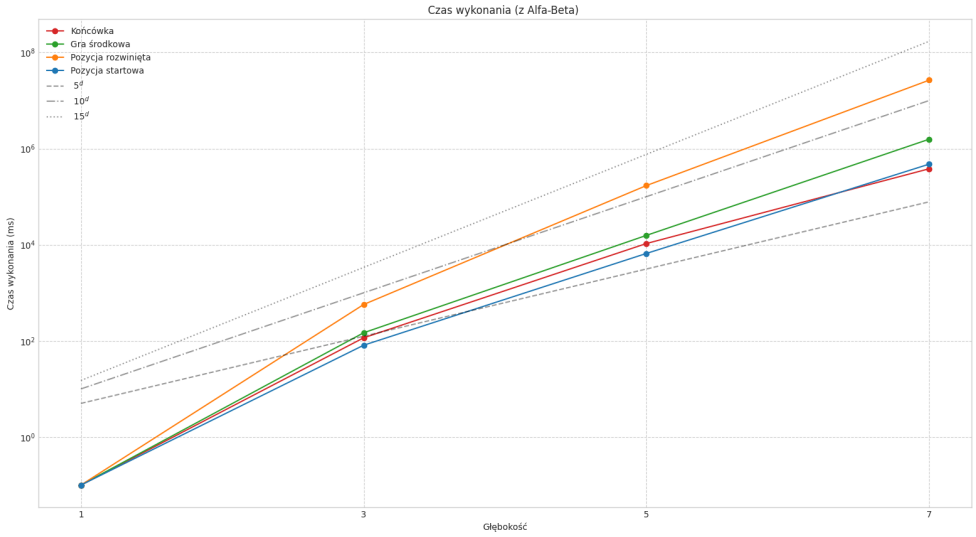
3.2 Czas Wykonania

Tabela 3: Czas wykonania w milisekundach dla algorytmu z Alpha-Beta Pruning

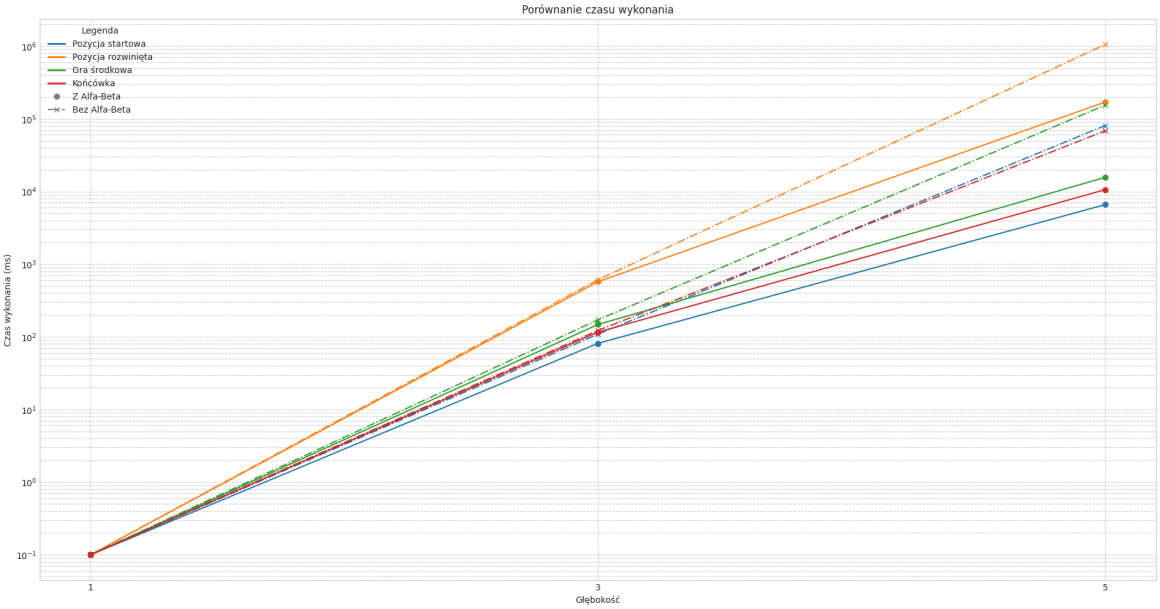
Pozycja	**Głębokość Przeszukania**			
	1	**3**	**5**	**7**
Pozycja startowa	0	81.2	6589.6	474761
Pozycja rozwinięta	0	571.0	170210.0	26531200
Gra środkowa	0	147.5	15652.6	1567220
Końcówka	0	115.8	10594.8	380512

Tabela 4: Czas wykonania w milisekundach dla algorytmu bez Alpha-Beta Pruning

Pozycja	**Głębokość Przeszukania**			
	1	**3**	**5**	**7**
Pozycja startowa	0	108.4	80700.8	x
Pozycja rozwinięta	0	610.2	1053540.0	x
Gra środkowa	0	171.2	154496.0	x
Końcówka	0	122.1	68435.2	x



Rysunek 11: Czas wykonywania algorytmu w milisekundach dla algorytmu z Alpha-Beta Pruning



Rysunek 12: Porównanie czasu wykonania dla danej planszy pomiędzy algorytmem z Alpha-Beta Pruning i bez niego

4 Wnioski

- Złożoność teoretyczna algorytmu min-max to $O(b^d)$, nasz algorytm rośnie wykładniczo wraz z głębokością przeszukiwania, przy czym dokładny stopień wzrostu jest trudny do aproksymacji.
- Alpha-Beta Pruning w algorymie Min-Max pozwala na ogromną oszczędność czasową, która tylko zwiększa się wraz z głębokością.
- Stan planszy znacząco wpływa na to jak będzie działał algorytm, im więcej dostępnych figur lub im bardziej "otwarta pozycja", tym większe będzie drzewo poszukiwań, a tym samym zwiększy się czas działania algorytmu.

- Trudność w wykonaniu projektu sprawił fakt, iż pomimo pozornie proste szachy mają dużo zasad, kilka specjalnych ruchów oraz przy ruchu króla zachodzi potrzeba kalkulacji czy przypadkiem nie poruszy się on na atakowane pole. Dodatkowo tworzenie interfejsu użytkownika po raz pierwszy przyspożyło dużo problemu.

5 Źródła

- `https://en.wikipedia.org/wiki/Alphabeta_pruning`
- `https://en.wikipedia.org/wiki/Minimax`
- `https://www.chessprogramming.org/Alpha-Beta`
- `https://en.wikipedia.org/wiki/Chess`