AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ GEOLOGII, GEOFIZYKI I OCHRONY ŚRODOWISKA

KATEDRA GEOINFORMATYKI I INFORMATYKI STOSOWANEJ

Praca dyplomowa

Projekt silnika szachowego z wykorzystaniem metod sztucznej inteligencji

Chess engine design using artificial intelligence methods

Autor:	**Bartłomiej Kózka**
Kierunek studiów:	*Inżynieria i Analiza Danych*
Opiekun pracy:	*Prof. dr hab. inż. Norbert Skoczylas*

Kraków, 2025

Spis treści

Spis rysunków

1. Wstęp

Sztuczna inteligencja (SI) stała się nieodłącznym elementem współczesnej technologii, wpływając na niemal każdy aspekt życia codziennego. Choć obecnie największą uwagę mediów i badaczy przyciągają generatywne modele językowe, nie należy zapominać o fundamentalnych metodach, które ukształtowały tę dziedzinę. Jednym z kluczowych obszarów, znajdującym szerokie zastosowanie w grach logicznych, jest przeszukiwanie przestrzeni stanów. Metody te stanowią podwaliny klasycznej sztucznej inteligencji, co zauważyć można już w pionierskich pracach Allena Newella i Herberta Simona. Stworzyli oni ”General Problem Solver”(GPS) – program zaprojektowany do rozwiązywania dowolnego problemu, który da się zdefiniować za pomocą skończonego zbioru reguł formalnych.

Szachy to deterministyczna, strategiczna gra planszowa o tzw. pełnej informowalności, pozbawiona elementów losowych. Jej geneza sięga VII wieku i indyjskiej gry Czaturanga. Po dotarciu do Europy gra ewoluowała, przyjmując współczesny system poruszania się figur pod koniec XV wieku, natomiast ostateczna standaryzacja reguł turniejowych nastąpiła w drugiej połowie XIX wieku. Obecnie szachy są jedną z najbardziej rozpowszechnionych gier logicznych na świecie, angażującą miliony graczy i stanowiącą idealne środowisko do badań nad sztuczną inteligencją.

Trafność wyboru tej gry jako domeny badawczej podkreślił rosyjski informatyk Alexander Kronrod, wypowiadając słynne zdanie: „Szachy są drozofilą sztucznej inteligencji”. Przez to porównanie do muszki owocowej – organizmu modelowego w genetyce – wskazał on na szachy jako idealne środowisko do izolowania i badania mechanizmów intelektualnych. Kronrod przestrzegał jednak, aby rozwój dziedziny nie zmienił się w „wyścigi muszek owocowych”, gdzie nacisk kładziony jest wyłącznie na wynik, a nie na zrozumienie procesów myślowych. Historia pokazała jednak, że to właśnie surowa siła obliczeniowa stała się kluczem do dominacji maszyn. Punktem zwrotnym w historii tej dyscypliny był rok 1997, kiedy to superkomputer ”Deep Blue”firmy IBM pokonał urzędującego mistrza świata, Garriego Kasparowa. Był to pierwszy przypadek w historii, gdy maszyna wygrała mecz z najlepszym szachistą globu w standardowych warunkach turniejowych. Wydarzenie to uznawane jest za kamień milowy w rozwoju informatyki, symbolicznie otwierający nową erę. Od tego momentu silniki szachowe przestały być postrzegane wyłącznie jako ciekawostka technologiczna, stając się potężnymi narzędziami analitycznymi, które dziś nie tylko przewyższają człowieka zdolnościami obliczeniowymi, ale także służą do treningu i podnoszenia poziomu gry zawodników.

Kluczowym wyzwaniem w projektowaniu współczesnych silników szachowych jest optymalizacja procesu decyzyjnego. Ze względu na tzw. eksplozję kombinato-

ryczną, liczba możliwych wariantów partii przekracza możliwości obliczeniowe jakiegokolwiek komputera, co uniemożliwia pełne przeszukanie drzewa gry. Z tego powodu nowoczesne silniki nie analizują każdej możliwości, lecz wykorzystują zaawansowane heurystyki, pozwalające na ”odcinanie”nieperspektywicznych gałęzi (pruning). Takie podejście umożliwia znalezienie optymalnego lub bliskiego optymalnemu ruchu w ściśle ograniczonym czasie, co jest warunkiem koniecznym w rozgrywkach turniejowych. Efektywność tych algorytmów decyduje o ”sile”silnika szachowego znacznie bardziej niż surowa moc obliczeniowa sprzętu.

Reasumując, zastosowanie metod sztucznej inteligencji, a w szczególności algorytmów przeszukiwania przestrzeni stanów w grach takich jak szachy, pozostaje kluczowym obszarem badań w informatyce. Gry te stanowią doskonałe środowisko do weryfikacji efektywności algorytmów, łącząc teorię z praktycznymi wyzwaniami optymalizacyjnymi. Projektowanie silnika szachowego nie jest zatem jedynie próbą stworzenia wirtualnego przeciwnika, lecz przede wszystkim ambitnym zadaniem inżynierskim, pozwalającym na praktyczne zmierzenie się z problemem wysokiej złożoności obliczeniowej.

2. Cel pracy

Celem niniejszej pracy jest zaprojektowanie i implementacja silnika szachowego wykorzystującego metody sztucznej inteligencji. Kluczowym elementem badań jest analiza wpływu wybranych algorytmów optymalizujących przeszukiwanie drzewa gry na wydajność obliczeniową programu. Ponadto, praca obejmuje porównanie różnych wariantów heurystycznych funkcji oceny pozycji oraz zbadanie, w jaki sposób zastosowane metody przekładają się na ostateczną siłę gry silnika.

Układ pracy odzwierciedla przyjętą metodologię badawczą i został podzielony na część teoretyczną oraz praktyczną. Część teoretyczna zawiera szczegółowy opis algorytmu Minimax, stanowiącego fundament działania silnika szachowego. W dalszej kolejności omówiono kluczowe algorytmy optymalizacyjne, takie jak odcięcia Alpha-Beta (Alpha-Beta Pruning), pogłębianie iteracyjne (Iterative Deepening) oraz przeszukiwanie wyciszeń (Quiescence Search). Rozdział ten zamyka charakterystyka funkcji heurystycznych wykorzystywanych do statycznej oceny pozycji.

Część praktyczna koncentruje się na weryfikacji eksperymentalnej zaimplementowanego rozwiązania. Siła gry silnika została zmierzona na podstawie serii turniejów rozegranych z innymi silnikami szachowymi. Przeprowadzono również analizę porównawczą, badającą wpływ poszczególnych metod optymalizacji oraz wariantów funkcji oceny na ostateczną skuteczność i ranking silnika.

3. Zasady gry w szachy (według specyfikacji FIDE)

Poniższe zestawienie reguł opracowano na podstawie oficjalnych Przepisów Gry w Szachy FIDE (FIDE Laws of Chess), obowiązujących w turniejach rangi mistrzowskiej. Stanowią one zbiór wymagań funkcjonalnych zaimplementowanych w silniku. https://www.fide.com/FIDE/handbook/LawsOfChess.pdf

3.1 Natura i cele gry w szachy

[Reguła 1.1] Gra w szachy toczy się między dwoma przeciwnikami, którzy na przemian wykonują posunięcia bierkami na kwadratowej planszy, zwanej szachownicą.

[Reguła 1.2] Celem każdego z graczy jest ustawienie króla przeciwnika ”pod atakiem”w taki sposób, aby przeciwnik nie miał żadnego legalnego posunięcia. Gracz, który doprowadzi do takiej pozycji, ”daje mata”i wygrywa partię.

[Reguła 1.3] Jeśli pozycja jest taka, że żaden z graczy nie jest w stanie ”dać mata”, następuje remis.

3.2 Pozycja początkowa i szachownica

[Reguła 2.1] Szachownica składa się z siatki 8×8, tworzącej 64 pola o jednakowych rozmiarach, naprzemiennie jasnych (pola białe) i ciemnych (pola czarne).

[Reguła 2.2] Na początku gry, jeden gracz posiada 16 białych pionków, a drugi 16 czarnych pionków.

[Reguła 2.3] Początkow pozycja pionków na szachownicy wygląda następująco:

Rysunek 3.1. Szachownica reprezentująca początkowe położenie pionków

[Reguła 2.4] Osiem pionowych kolumn pól nazywa się liniami (oznaczone literami od a do h). Osiem poziomych rzędów pól nazywa się rzędami (numerowane od 1 do 8). Każde pole jest jednoznacznie identyfikowane przez unikalną parę współrzędnych (np. e4).

3.3 Ruchy figur

[Reguła 3.1] Zgodnie z Artykułem 3, żadna bierka nie może stanąć na polu zajmowanym przez bierkę tego samego koloru. Jeśli bierka staje na polu zajętym przez przeciwnika, następuje ”bicie”(usunięcie bierki przeciwnika z planszy).

[Reguła 3.2 - Goniec] Goniec może poruszać się na dowolne pole wzdłuż przekątnych (diagonali), na których się znajduje.

[Reguła 3.3 - Wieża] Wieża porusza się na dowolne pole wzdłuż linii (pionowo) lub rzędu (poziomo), na którym stoi.

[Reguła 3.4 - Hetman] Hetman porusza się na dowolne pole wzdłuż linii, rzędu lub przekątnej, na której stoi. Łączy w sobie wektory ruchu Wieży i Gońca.

[reguła 3.5] Robiąc ruch Gońcem, Wieża lub Hetmanem, nie ma możliwości przeskakiwania nad innymi blokującymi figurami.

Reguła 3.6 - Skoczek Skoczek wykonuje ruch na jedno z pól najbliższych temu, na którym stoi, ale nie na tej samej linii, rzędzie czy przekątnej. Ruch ten definiuje się jako ”kształt litery L”(przesunięcie o 2 pola w poziomie/pionie i 1 pole prostopadle). Jest to jedyna bierka mogąca przeskakiwać nad innymi.

[Reguła 3.7 - Pion] Pion porusza się do przodu na pole bezpośrednio przed nim na tej samej linii, pod warunkiem, że jest ono wolne.

a) Standardowy ruch to przesunięcie o jedno pole wzdłuż linii.
b) Przy pierwszym ruchu danego piona możliwe jest przesunięcie o dwa pola.
c) Bicie odbywa się inaczej niż ruch: o jedno pole do przodu na ukos (diagonalnie).
d) [Bicie w przelocie (En Passant)] Pion atakujący pole, które minął pion przeciwnika wykonujący ruch o dwa pola, może zbić tego piona tak, jakby wykonał on ruch tylko o jedno pole. Bicie to jest możliwe wyłącznie w posunięciu następującym bezpośrednio po takim ruchu piona.
e) [Promocja] Kiedy pion osiągnie najdalszy rząd (ósmy dla białych, pierwszy dla czarnych), musi zostać natychmiast zamieniony na nową figurę (hetmana, wieżę, gońca lub skoczka) tego samego koloru.

[Reguła 3.8 - Król] Król ma możliwość poruszania się na dwa różne sposoby.

a) Król może poruszać się na dowolne sąsiednie pole, które nie jest atakowane przez jedną lub więcej bierek przeciwnika.
b) [Roszada] to ruch króla i jednej z wież tego samego koloru na tej samej linii. Jest traktowana jako jedno posunięcie króla. Wykonuje się ją przez przesunięcie króla o dwa pola w stronę wieży, a następnie postawienie wieży na polu, które król właśnie minął.
c) Uprawnienia do wykonania roszady są utracone kiedy:
   - jeśli król już się poruszył, lub
   - z wieżą, która już się poruszyła
d) Roszada jest tymczasowo niemożliwa, jeżeli:
   - pole, na którym stoi król, pole, które król musi przekroczyć, lub pole, które król ma zająć, jest atakowane przez jedną lub więcej bierek przeciwnika;
   - pomiędzy królem a wieżą, z którą ma zostać wykonana roszada, znajduje się jakakolwiek inna bierka.

[Reguła 3.9] Uznaje się, że król jest ”w szachu”, jeżeli jest atakowany przez jedną lub więcej bierek przeciwnika, nawet w przypadku, gdy bierki te nie mogą wykonać ruchu na to pole, ponieważ odsłoniłyby lub wystawiły własnego króla na szach. Nie wolno wykonać posunięcia, które wystawiłoby króla tego samego koloru na szach lub pozostawiło go pod szachem.

3.4 Zakończenie gry

[Reguła 5.1 - Wygrana] Partia zostaje wygrana przez gracza w dwóch przypadkach:

a) Gdy gracz zamatował króla przeciwnika. Kończy to partię natychmiast, pod warunkiem, że posunięcie prowadzące do pozycji matowej było wykonane zgodnie z przepisami.
b) Gdy przeciwnik gracza ogłosił rezygnację (poddanie się). To również natychmiast kończy partię.

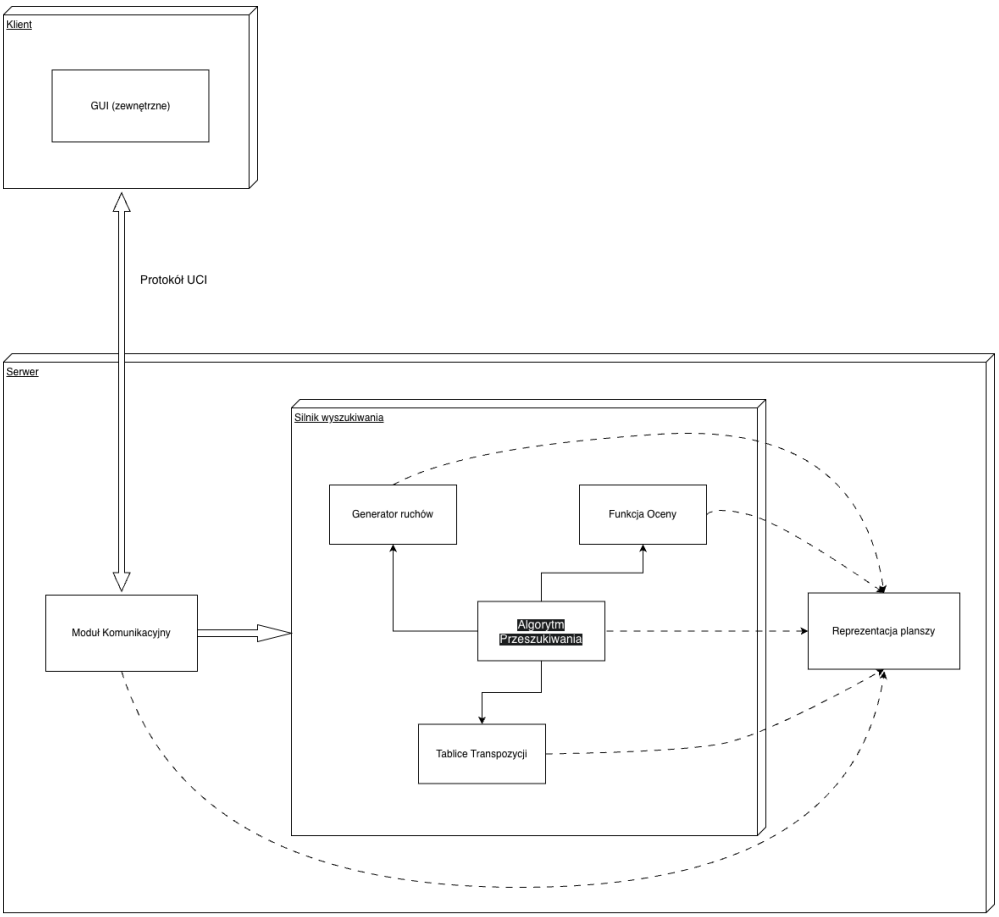
[Reguła 5.2 - Remis] Partia kończy się wynikiem nierozstrzygniętym (remisem) w następujących sytuacjach:

a) [Pat] Gracz, na którego przypada kolej posunięcia, nie może wykonać żadnego legalnego ruchu, a jego król nie znajduje się pod szachem.
b) Martwa pozycja: Powstaje pozycja, w której żaden z graczy nie może zamatować króla przeciwnika za pomocą jakiejkolwiek serii legalnych posunięć. Kończy to partię natychmiast (pod warunkiem legalności ostatniego ruchu).
c) [Porozumienie stron] Obaj gracze zgodnie ustalają remis w trakcie partii.
d) [Trzykrotne powtórzenie pozycji] Ta sama pozycja pojawiła się (lub ma się zaraz pojawić po zadeklarowanym ruchu) na szachownicy przynajmniej trzy razy.
e) [Reguła 50 ruchów] W ciągu ostatnich 50 posunięć wykonanych przez każdego z graczy nie nastąpiło żadne przesunięcie piona ani żadne bicie bierki.

4. Projekt silnika szachowego

4.1 Architektura systemu

System został zaprojektowany w oparciu o architekturę modułową, co zapewnia separację odpowiedzialności (ang. Separation of Concerns) i umożliwia niezależny rozwój oraz testowanie poszczególnych komponentów. Aplikacja funkcjonuje jako samodzielny program konsolowy, wykorzystujący standardowe strumienie wejścia/wyjścia (stdin/stdout) do komunikacji z otoczeniem. Całość realizuje model zbliżony do architektury klient-serwer, w którym silnik pełni rolę serwera obliczeniowego, natomiast klientem jest zewnętrzny graficzny interfejs użytkownika (GUI). Wymiana danych między warstwą prezentacji (GUI) a warstwą logiki (silnikiem) odbywa się za pośrednictwem ustandaryzowanego, tekstowego protokołu UCI (Universal Chess Interface).



Rysunek 4.1. Diagram UML przedstawiający ogólną architekturę projektu

4.2 Reprezentacja planszy

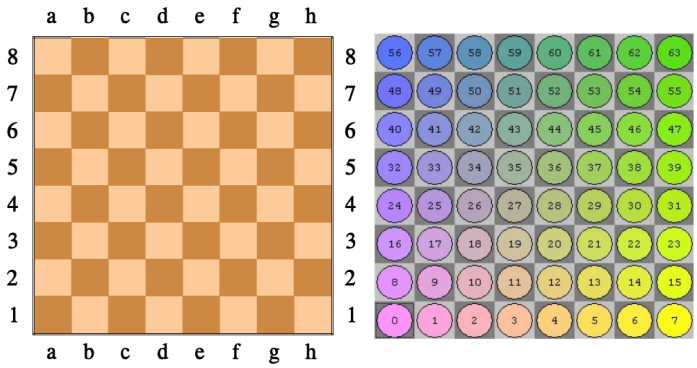
W programowaniu silników szachowych wyróżnia się dwa główne podejścia do reprezenracji stanu gry:

- zorientowane na bierki (ang. piece-centric)
- zorientowane na pola (ang. square-centric)

Metoda reprezentacji zorientowana na pola (np. *Mailbox*, 0x88) polega na przechowywaniu informacji o tym, jaka figura zajmuje każde z 64 pól planszy. Z kolei podejście zorientowane na bierki odwraca tę logikę, przechowując informację o lokalizacji wszystkich figur danego typu.

W projekcie zastosowano reprezentację zorientowaną na bierki, zaimplementowaną przy użyciu map bitowych (ang. *Bitboards*). Główną jej zaletą jest możliwość wykorzystania 64-bitowych rejestrów ogólnego przeznaczenia (GPR) współczesnych procesorów. Dzięki temu operacje na całych zbiorach figur nie wymagają iteracji pętlą po 64 polach planszy. Podstawowe operacje algebry Boole'a, takie jak koniunkcja bitowa (AND), alternatywa (OR) czy różnica symetryczna (XOR), są wykonywane przez jednostkę arytmetyczno-logiczną (ALU) procesora w ramach pojedynczej instrukcji maszynowej, co zazwyczaj przekłada się na jeden cykl zegara CPU. Przykładowo, nałożenie maski ataków na układ figur przeciwnika odbywa się w jednym kroku obliczeniowym, niezależnie od liczby bierek znajdujących się na szachownicy.

W projekcie przyjęto standardowe mapowanie pól znane jako LERF (Little-Endian Rank-File Mapping). W konwencji tej najmłodszy bit (LSB – index 0) słowa maszynowego odpowiada polu a1, kolejny polu b1, aż do najstarszego bitu (MSB – index 63) przypisanego do pola h8 (Rys. 3.x).



Rysunek 4.2. Reprezentacja szachownicy LERF (Little-Endian Rank-File Mapping)

Wybór ten nie jest przypadkowy i wynika z architektury współczesnych procesorów (x86-64), które operują w trybie Little Endian. Oznacza to, że mniej znaczące bajty zapisywane są pod niższymi adresami pamięci. Choć z punktu widzenia samej szybkości operacji bitowych (jak AND czy XOR) różnica wydajności między podejściem Little a Big Endian jest pomijalna, to zgodność logicznej reprezentacji planszy z fizyczną architekturą procesora ułatwia debugowanie, serializację danych oraz implementację niektórych operacji przesuwania bitów (bit-shifting).

4.3 Algorytmy przeszukania

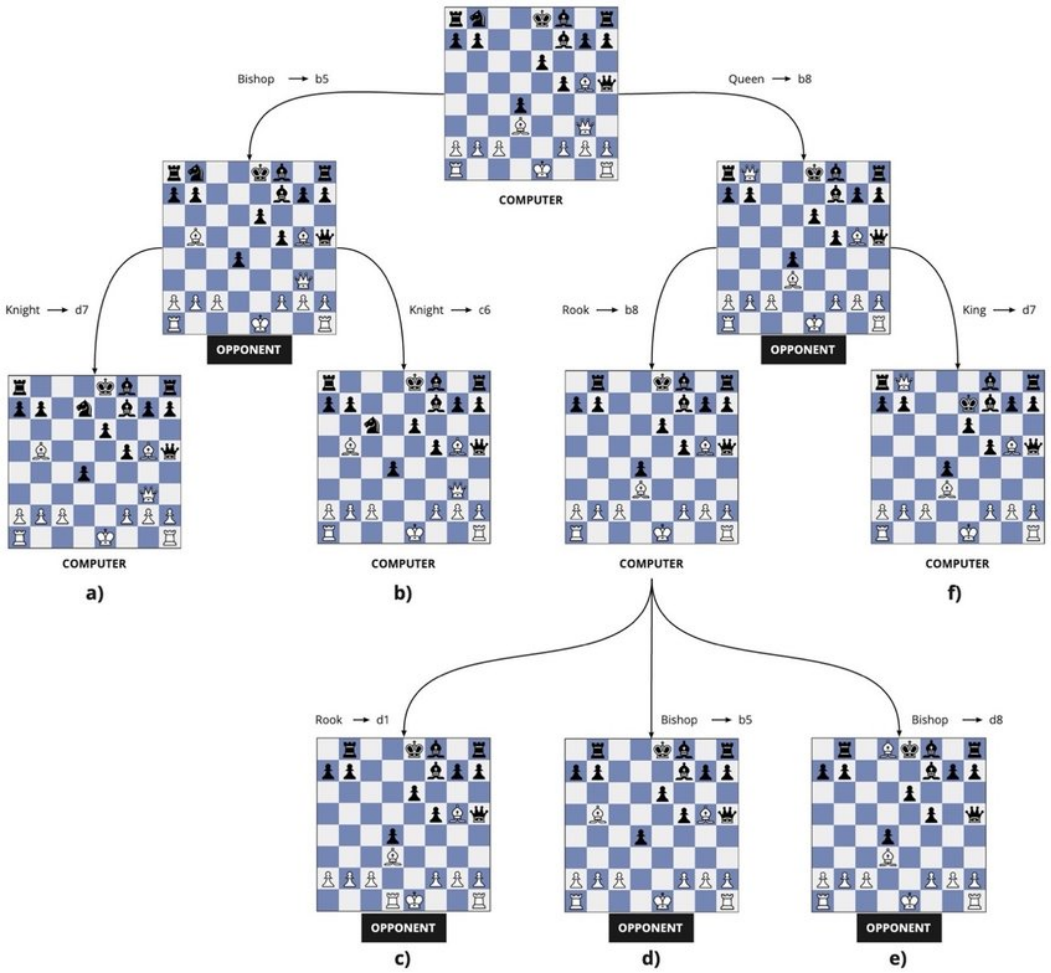
4.3.1 Drzewo gry

Drzewo gry jest fundamentalną strukturą danych wykorzystywaną w algorytmach przeszukiwania przestrzeni stanów dla gier o sumie zerowej z pełną informacją, takich jak szachy. Formalnie, przestrzeń wszystkich możliwych rozgrywek modelowana jest jako graf skierowany, który w kontekście algorytmów przeszukiwania (takich jak Minimax) przybiera postać drzewa rozpinanego od aktualnej pozycji.

Struktura drzewa gry składa się z trzech kluczowych elementów:

- **Korzeń (Root)** – wierzchołek początkowy reprezentujący aktualny stan gry (bieżącą konfigurację bierek na szachownicy), od którego silnik rozpoczyna analizę.
- **Krawędzie (Edges)** – reprezentują legalne posunięcia (akcje) możliwe do wykonania przez gracza, do którego należy ruch w danym węźle. Przejście krawędzią odpowiada wykonaniu ruchu i transformacji stanu gry.
- **Węzły (Nodes)** – wierzchołki potomne reprezentujące stany gry powstałe w wyniku wykonania posunięć. Węzły, które nie posiadają potomków w analizowanym fragmencie drzewa, nazywane są *liśćmi* (ang. *leaf nodes*) i to dla nich wywoływana jest funkcja oceny (ewaluacji).

W szachach, ze względu na naprzemienność wykonywania ruchów, drzewo to posiada strukturę warstwową, gdzie poziomy (głębokość) odpowiadają kolejnym posunięciom stron: Białych (MAX) i Czarnych (MIN).



Rysunek 4.3

4.3.2 Algorytm MiniMax

Algorytm Minimax jest podstawową metodą przeszukiwania drzewa gry, służącą do wyznaczenia optymalnej strategii poprzez symulację przyszłych stanów planszy. Jest to reguła decyzyjna mająca na celu minimalizację potencjalnych strat w scenariuszu pesymistycznym (ang. *worst-case scenario*).

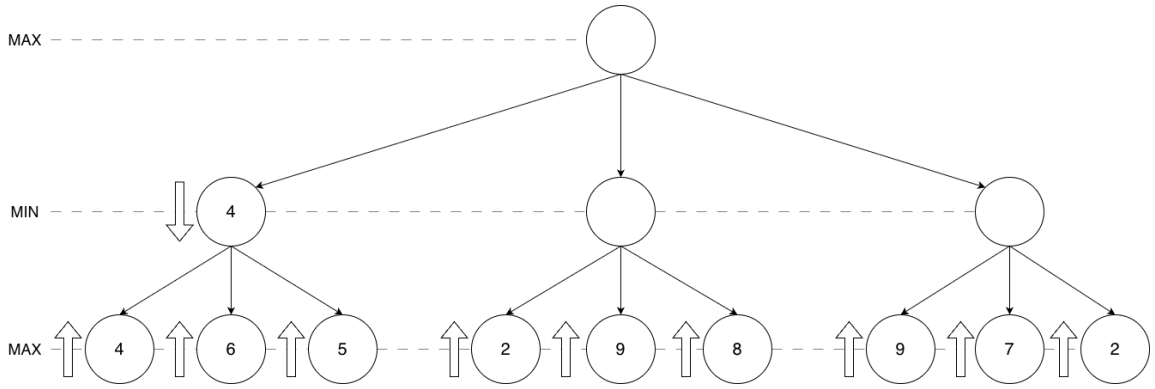
Zasada działania opiera się na symulacji rozgrywki w przód o zadaną liczbę posunięć (głębokość drzewa) i założeniu, że obaj gracze grają optymalnie:

- **Gracz MAX (Maksymalizujący):** Dąży do wyboru ruchu, który prowadzi do stanu gry o jak najwyższej ocenie punktowej (zazwyczaj utożsamiany z silnikiem/stroną wykonującą ruch).
- **Gracz MIN (Minimalizujący):** Przeciwnik, który dąży do wyboru ruchu prowadzącego do stanu o jak najniższej ocenie (najgorszego dla gracza MAX).

Przykładowy przebieg algorytmu (Opis algorytmu przygotowano w oparciu o książkę ”Wstęp do Sztucznej Inteligencji”autorstwa Mariusza Flasińskiego):

Załóżmy, że każdemu liściowi drzewa przedstawionemu na Rysunku 4.4 została już przypisana wartość liczbowa obliczona przez funkcję oceny. Zgodnie z założeniami algorytmu, gracz MAX dąży do maksymalizacji tej wartości, natomiast przeciwnik (gracz MIN) dąży do jej minimalizacji, wybierając scenariusz najmniej korzystny z naszej perspektywy.

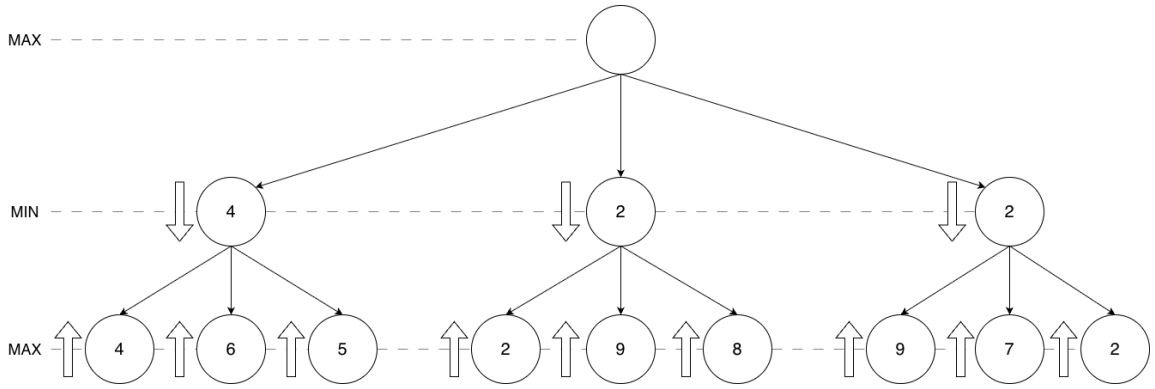
Proces wyznaczania wartości węzłów wewnętrznych odbywa się w drodze propagacji wstecznej, czyli od liści w górę do korzenia. Rozważmy lewe poddrzewo: liście posiadają wartości 4, 6 oraz 5. Ponieważ węzeł rodzicielski dla tych liści znajduje się na poziomie decyzyjnym gracza MIN (przeciwnika), wybierze on wartość najniższą. W tym przypadku jest to 4, co reprezentuje najlepszy możliwy ruch dla przeciwnika (a tym samym pesymistyczny wariant dla nas).



Rysunek 4.4

dentyczną logikę decyzyjną algorytm stosuje dla pozostałych gałęzi drzewa. W przypadku środkowego poddrzewa, spośród dostępnych ocen liści (2, 9, 8), gracz MIN wybiera wartość najniższą, czyli 2, co stanowi dla niego optymalny wybór w tej gałęzi.

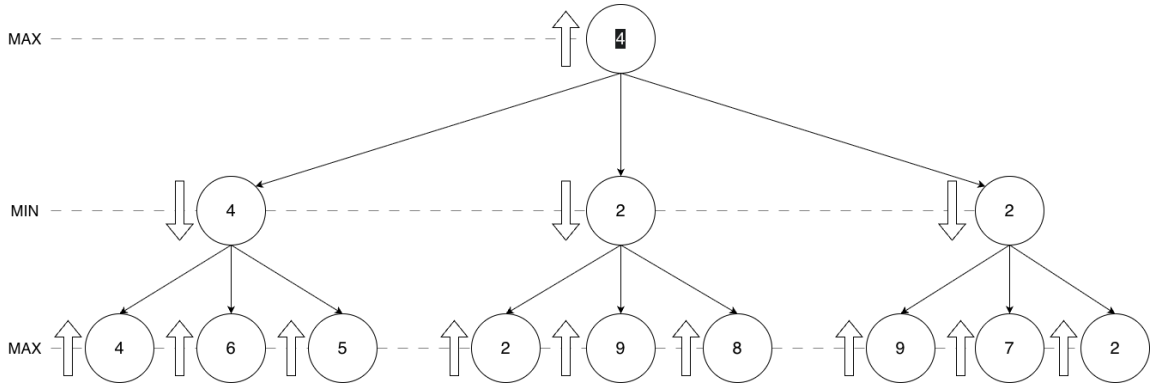
Podobnie w prawym poddrzewie, gdzie przeciwnik dysponuje wariantami ocenionymi na 9, 7 oraz 2. Zgodnie z założeniem o bezbłędnej grze rywala, do węzła rodzicielskiego propagowana jest wartość minimalna równa 2. W rezultacie, po zakończeniu analizy na tym poziomie głębokości, warstwa decyzyjna gracza MIN (reprezentowana przez węzły pośrednie) przyjmuje zestaw wartości 4,2,2.



Rysunek 4.5

Finalny etap algorytmu odbywa się w korzeniu drzewa (Rysunek 4.6). Węzeł ten należy do gracza maksymalizującego (MAX), którego celem jest wybór ruchu gwarantującego najwyższą ocenę punktową.

Analizując wartości propagowane z węzłów potomnych – odpowiednio 4,2,2 – algorytm dokonuje selekcji wartości największej, równej 4. Decyzja ta jest tożsama z wyborem posunięcia prowadzącego do lewego poddrzewa, które w rozpatrywanym wariancie zapewnia najlepszy możliwy wynik przy założeniu optymalnej gry przeciwnika.



Rysunek 4.6

4.3.3 Algorytm Alpha-Beta – optymalizacja przeszukiwania

Algorytm Alpha-Beta jest rozwinięciem metody Minimax, wzbogaconym o mechanizm odcinania gałęzi (ang. *pruning*), które nie mają wpływu na ostateczną decyzję. Jest to algorytm ściśle optymalizacyjny – gwarantuje znalezienie dokładnie tego samego ruchu co czysty Minimax, ale przy znacznie mniejszym nakładzie obliczeniowym.

Głównym celem algorytmu jest redukcja przestrzeni przeszukiwania. W scenariuszu optymalnym (gdy najlepsze ruchy są rozważane jako pierwsze), złożoność czasowa spada z wykładniczej $O(b^d)$ do $O(b^{d/2})$, gdzie b oznacza czynnik rozgałęzienia, a d głębokość drzewa. W ujęciu matematycznym oznacza to zredukowanie liczby odwiedzanych stanów do pierwiastka z liczby węzłów odwiedzanych przez standardowy algorytm Minimax ($\sqrt{N}$).

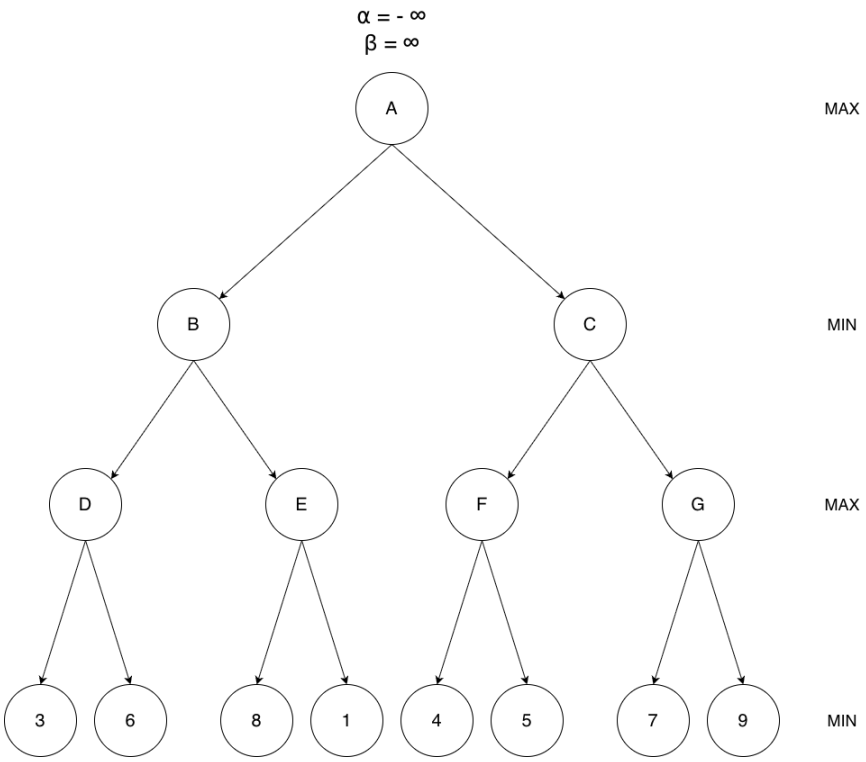
W celu realizacji mechanizmu odcięć wprowadza się dwie zmienne sterujące, definiujące przedział poszukiwań (tzw. okno alpha-beta):

- **Alpha (α)** – reprezentuje najlepszy (najwyższy) wynik, jaki gracz maksymalizujący (MAX) zdołał sobie dotychczas zagwarantować na obecnej ścieżce poszukiwań. Wartość początkowa to $-\infty$.
- **Beta (β)** – reprezentuje najlepszy (najniższy) wynik, jaki gracz minimalizujący (MIN) zdołał osiągnąć dla siebie (czyli najgorszy dla nas). Wartość początkowa to $+\infty$.

Zmienne te są przekazywane i aktualizowane w trakcie przeszukiwania drzewa. Jeżeli w dowolnym momencie analizy węzła okaże się, że wartość α jest większa lub równa wartości β ($\alpha \geq \beta$), następuje tzw. **odcięcie**. Oznacza to, że obecna gałąź nie musi być dalej przeszukiwana, ponieważ przeciwnik grający optymalnie nigdy nie dopuści do powstania tej pozycji (wybierze wcześniej korzystniejszy dla siebie wariant w innej gałęzi).

Poniżej przedstawiono przykładowe działanie mechanizmu odcięć Alpha-Beta:

- **Etap 1 (Inicjalizacja i zejście):** Proces rozpoczyna się w korzeniu drzewa (A). Inicjowane są zmienne okna przeszukiwania: α=−∞ oraz β=+∞. Następnie algorytm wykonuje rekurencyjne zejście w głąb lewego poddrzewa (zgodnie z porządkiem DFS), przekazując aktualne wartości α i β do kolejnych węzłów. Ścieżka wywołania prowadzi przez węzły A ☐ B ☐ D, aż do osiągnięcia rodzica skrajnie lewych liści (węzła D).



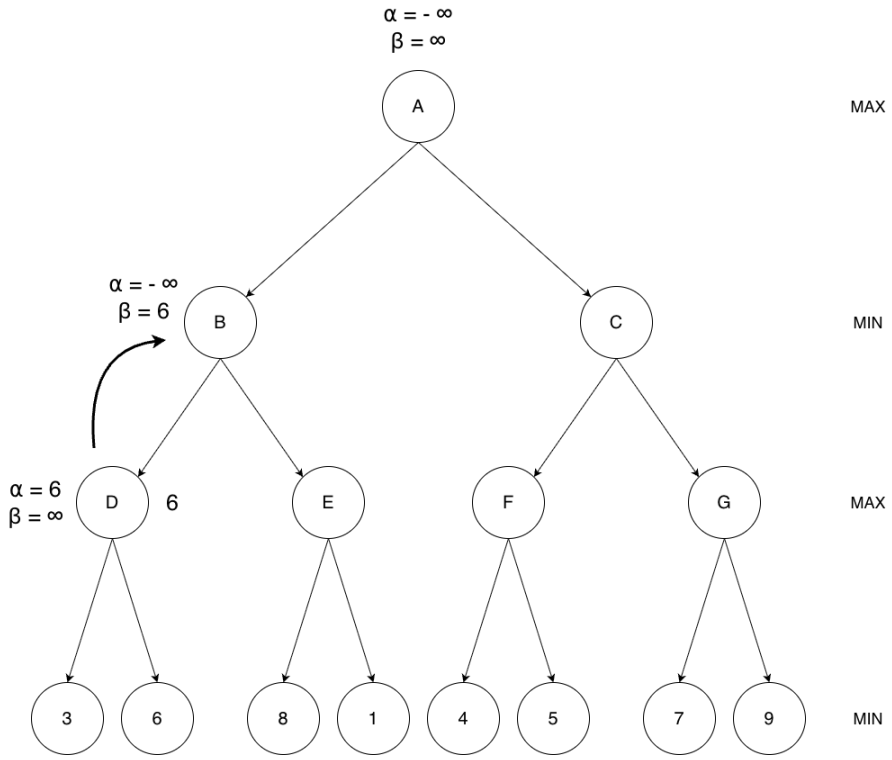
Rysunek 4.7

- **Etap 2 (Poziom MAX - węzeł D):** Węzeł D znajduje się na poziomie gracza maksymalizującego. Algorytm analizuje jego węzły potomne (liście) o wartościach $\{3, 6\}$. Celem jest wybór wartości najwyższej (maksymalizacja).
  - Początkowo analizowana jest wartość 3. Ponieważ $3 > \alpha$, wartość ta staje się tymczasowym najlepszym wynikiem.
  - Następnie analizowana jest wartość 6. Ponieważ $6 > 3$, algorytm aktualizuje wartość węzła D na 6.

  W rezultacie, zmienna α dla tego węzła przyjmuje wartość 6 (jako nowe dolne ograniczenie dla gracza MAX).

- **Etap 3:** Następnie propagujemy wybraną wartość 6 wstecz do węzła B, zgodnie z działaniem rekurencji, i tam aktualizujemy wartość β, iż jesteśmy na poziomie minimalizacji, czyli ruch wykonuje gracz minimaluzujący, a więc 6 < ∞, czyli β

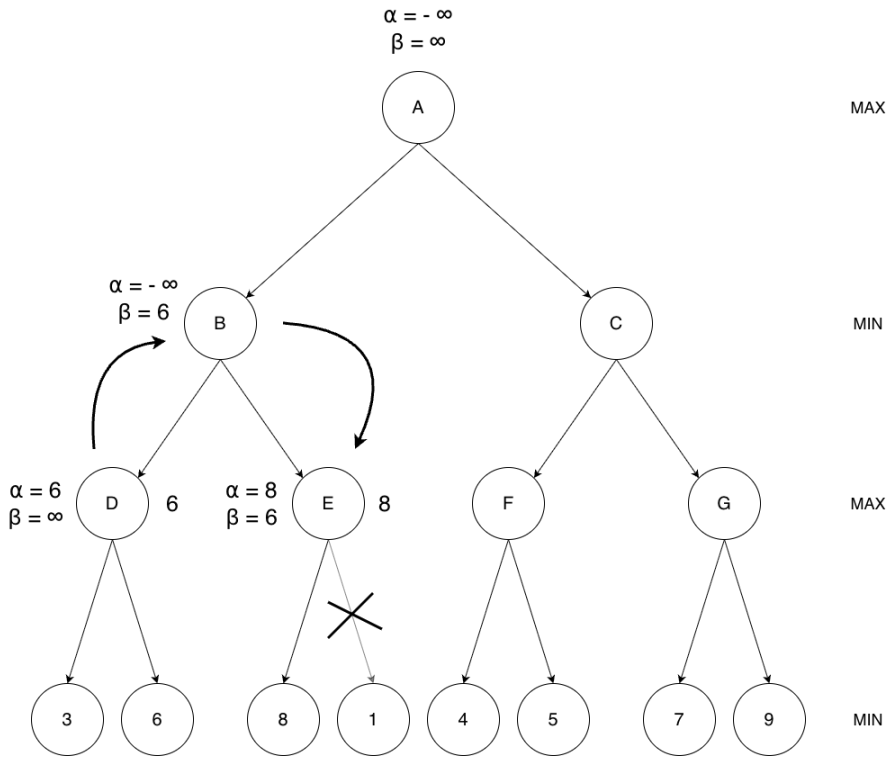
równa się 6. Dalej przechodzimy to prawego podrzewa od węzła B, gdyż lewe już oceniliśmy.



Rysunek 4.8

- **Etap 4 (Pierwsze odcięcie):** Algorytm przechodzi do węzła E (poziom gracza maksymalizującego), który otrzymał od rodzica (węzeł B) wartość β=6. Następuje analiza lewego węzła potomnego o wartości 8.
  - Ponieważ węzeł E dąży do maksymalizacji, aktualizuje swoją wartość tymczasową na 8, a tym samym przyjmuje nowe dolne ograniczenie α=8.
  - Następuje weryfikacja warunku odcięcia: α≥β (czyli 8≥6).

  Warunek jest spełniony (Prawda). W tym momencie następuje tzw. odcięcie Beta (ang. *Beta cut-off*). Prawa gałąź węzła E nie jest przeszukiwana. **Uzasadnienie:** Gracz minimalizujący (w węźle B) posiada już wariant gwarantujący mu wynik 6 (z węzła D). Skoro węzeł E gwarantuje wynik co najmniej 8 (lub wyższy), to dla gracza MIN jest to opcja gorsza niż ta, którą już posiada. Dlatego wartość pozostałych dzieci węzła E jest nieistotna dla decyzji podejmowanej w węźle B.

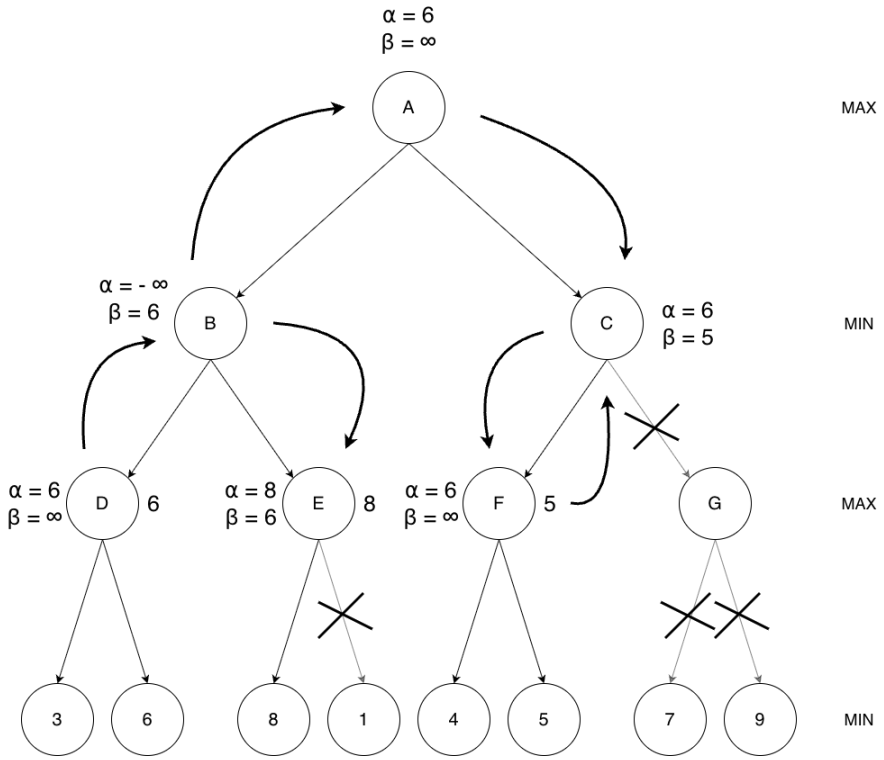


Rysunek 4.9

- **Etap 5 (Ostatnie odcięcie):** Następnie rozpoczyna się analiza prawego poddrzewa (węzeł C). Algorytm schodzi rekurencyjnie do węzła F, przekazując zaktualizowane okno przeszukiwania ($\alpha = 6, \beta = +\infty$). W węźle F sprawdzane są wartości liści:
  - Pierwsze dziecko ma wartość 4. Jest ona mniejsza od obecnego α (6), więc nie wpływa na wynik gracza maksymalizującego na wyższym poziomie.
  - Drugie dziecko ma wartość 5. Również jest ona mniejsza od α.

  Węzeł F zwraca do swojego rodzica (węzła C) swoją najlepszą wartość, czyli 5.

  Węzeł C (znajdujący się na poziomie minimalizacji) aktualizuje swoją wartość β na 5 (ponieważ $5 < \infty$). W tym momencie następuje sprawdzenie warunku odcięcia: $\alpha \geq \beta$ (czyli $6 \geq 5$). Warunek jest spełniony. Algorytm przerywa przeszukiwanie pozostałych gałęzi węzła C, uznając je za nieistotne, co kończy proces przeszukiwania dla tego fragmentu drzewa.



Rysunek 4.10

4.3.4 IPrzeszukiwanie z iteracyjnym pogłębianiem

5. Użyte technologie

6. Implementacja silnika

7. Analiza rezultatów i wydajności poszczególnych wersji silnika

8. Wnioski