# Sprawozdanie

Projektowanie algorytmów i metody sztucznej inteligencji
Projekt 3

Mikołaj Saramonowicz 252964
Grupa środa, 13:15-15:00

# 1 Wprowadzenie

Sprawozdanie zapiera opis rozwiązań przygotowanych do zadania na ocenę bardzo dobrą, tj. omówienie zrealizowanej gry oraz zaimplementowanej metody wykorzystywanej przez przeciwnika komputerowego.

# 2 Wybrana gra

W zadaniu zaprojektowano grę w kółko i krzyżyk przeciwko sztucznej inteligencji. Gracz posiada możliwość wyboru: symbolu, rozmiaru planszy, ilości znaków w rzędzie potrzebnej do wygranej oraz trudności swojego przeciwnika. W przyjętym rozwiązaniu użytkownik zawsze rusza się jako pierwszy.

Program wykorzystując odpowiednie reguły sprawdza poprawność ruchu gracza.
Poprawny przebieg gry możliwy jest dzięki zastosowaniu specjalnie dobranych struktur danych.

# 3 Wykorzystywane struktury danych

## 3.1 Węzeł konfiguracyjny

W celu zrealizowania drzewa gry, każdemu z branych pod uwagę ruchów przypisano węzeł. Zawiera on wskaźniki na inny obiekt tej samej klasy (rodzica) oraz na ich listę (dzieci). Ponadto w przyjętym rozwiązaniu klasa ta przechowuje konfigurację danego ruchu (jako dynamiczną, dwuwymiarową tablicę znaków), rozmiar boku planszy (wykorzystywany przez konstruktor), ocenę ruchu oraz pomocniczą strukturę typu "Ruch", która umożliwia sprawne reprezentowanie ruchu (umożliwiającego przejście z poprzedniej badanej konfiguracji do tej przechowywanej w obecnym węźle) jako współrzędne oraz znak gracza.

Inicjalizacja obiektu dokonywana jest z użyciem konstruktora parametrycznego, który umożliwia odpowiednie ustawienie początkowe wskaźników oraz rezerwację właściwej ilości pamięci, odpowiadającej rozmiarowi planszy. Usuwanie węzła wykonywane jest przez zaprojektowany destruktor, zwalniający pamięć i usuwający wskaźniki.

Opisywana struktura wyposażona została w metody umożliwiające właściwe ustawianie oraz podawanie przechowywanych danych. Dla konfiguracji planszy możliwe jest wyświetlenie, dzięki czemu gracz może otrzymać graficzne informacje na temat stanu gry.

## 3.2 Drzewo gry

W celu reprezentacji zależności przyczynowo-skutkowych (rodzic-syn) pomiędzy możliwymi ruchami gracza lub komputera zaprojektowano strukturę przechowującą węzły konfiguracyjne, opartą na interfejsie drzewa.

Przygotowana klasa zawiera metody umożliwiające m.in. dodanie nowych węzłów jako możliwych ruchów (dzieci) wynikających z wybranej konfiguracji (rodzica).

Cechami wyróżniającymi zaprojektowane drzewo od typowego jest obecność dodatkowych metod i pól związanych z przebiegiem wybranej gry. Przykładowo, struktura zawiera odpowiadający aktualnemu stanowi gry wskaźnik "obecny_stan" oraz szereg różnych funkcjonalności umożliwiających m.in. sprawdzenie wyniku meczu lub wykonanie optymalnego ruchu przez komputer.

Konstruktor parametryczny tworzący drzewo wymaga podania pożądanego rozmiaru boku planszy gry, a także długości znaków w rzędzie potrzebnej do wygranej. Po jego wywołaniu następuje odpowiednie zapisanie wskazanych danych, a do struktury zostaje automatycznie dodany pierwszy węzeł (korzeń), zawierający początkowy stan gry.

Podczas projektowania destruktora klasy zastosowano przejście post-order, umożliwiające poprawne odwiedzenie każdego z węzłów i wykonanie operacji usuwania. Jako że podczas przebiegu gry nie ma potrzeby ręcznego usuwania elementów, to metoda usuwania ustawiona została jako prywatna - można ją wywołać jedynie poprzez destruktor drzewa; następuje wówczas usunięcie wszystkich elementów.

Jedna z metod związanych z ruchami sztucznej inteligencji opiera się na popularnym algorytmie MiniMax, który omówiony zostanie w następnej części sprawozdania.

Bardziej szczegółowe informacje na temat działania zaimplementowanych struktur zawarto w komentarzach kodu źródłowego dołączonego do sprawozdania.

# 4 Wykorzystywany algorytm SI - Minimax

## 4.1 Wersja podstawowa

W celu symulowania optymalnie myślącego przeciwnika przez komputer zastosowano algorytm Minimax. Działanie zastosowanej procedury opiera się na odpowiednich przeszukiwaniach w głąb drzewa gry do momentu napotkania węzła zewnętrznego (oznaczającego rozstrzygnięcie meczu). Następnie na podstawie informacji o tym, kto obecnie wykonuje ruch, komputer rekurencyjnie porównuje ze sobą możliwe konfiguracje i wybiera najlepszą z nich.

Każda z branych pod uwagę opcji badana jest pod kątem oceny sytuacji na planszy, gdzie -1 oznacza wygraną kółka, 1 wygraną krzyżyka, a 0 remis. Dla węzłów zewnętrznych wynik przepisywany jest do ich rodziców, po czym zostaje wykorzystany do dalszych porównań.

Efektem końcowym pracy algorytmu jest otrzymanie ruchu o najlepszej możliwej ocenie dla bieżącego gracza, biorąc pod uwagę jego dalsze konsekwencje w grze, przy założeniu optymalnych ruchów przeciwnika. Ilość przewidywanych ruchów odpowiada głębi, będącej jednym z parametrów algorytmu.

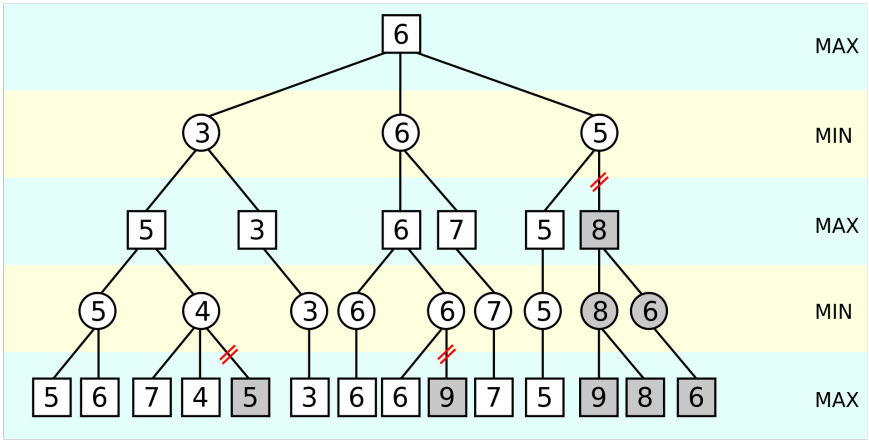
```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

Rysunek 1: Pseudokod ukazujący ogólną zasadę działania algorytmu

W zaprezentowanej formie algorytm na złożoność obliczeniową $O(b^m)$, i pamięciową $O(bm)$.

### 4.2 Wersja wykorzystująca alfa-beta cięcia

W przedstawionym rozwiązaniu wykorzystano metodę alfa-beta cięć, umożliwiającą skrócenie pracy algorytmu Minimax. Wykorzystuje ona odpowiednie porównania, umożliwiające ignorowanie gałęzi nie mających wpływu na optymalny wynik. Przykładowy sposób realizacji cięć przedstawiono na poniższej infografice:




Rysunek 2: Szare poddrzewa nie muszą być przeszukiwane przez algorytm

W zaimplementowanym rozwiązaniu cięcia są realizowane np. przez ignorowanie dalszych dzieci węzła odpowiadającego szukaniu wartości maksymalnej, jeżeli udało się już znaleźć wartość 1. Zastosowanie alfa-beta cięć umożliwia skrócenie złożoności obliczeniowej algorytmu do $O(b^{\frac{m}{2}})$.

## 5 Dodatkowe wnioski i spostrzeżenia

- Rozmiar planszy ma duże znaczenie dla czasu pracy algorytmu, zatem podczas implementacji należy odpowiednio ograniczać wartość głębi, aby pozostać w dziedzinie rozsądnych czasów odpowiedzi komputera.
- Podczas testowania programu zauważono, że nie wszystkie ruchy komputera wydają się być "ludzkie". Wynika to z wielu uproszczeń implementacji. Przykładowo, dla kilku ruchów o takiej samej ocenie komputer zawsze wybiera pierwszy analizowany, co teoretycznie nie powinno mieć wpływu na wynik meczu, ale może odróżnić go od człowieka.
- Jedną z cech wynikających z algorytmu Minimax jest założenie, że ludzki przeciwnik wykonuje same optymalne ruchy. Powoduje to poprawną rozgrywkę z kompetentnym graczem, jednak czasami objawia się w "poddawaniu się" komputera, jeżeli ten uzna, że przegra niezależnie od swoich dalszych ruchów. Ma to również związek z tym, że w przyjętym rozwiązaniu bot nie stara się dojść ani do najszybszego możliwego zwycięstwa, ani do najwolniejszej możliwej porażki.

# 6 Wykorzystana literatura


- dr inż. Łukasz Jeleń, prezentacje wykładowe i materiały dostępne na stronie kursu
- Minimax, Wikipedia, Aktualizacja 2 maja 2021,
  dostęp w Internecie:`https://en.wikipedia.org/wiki/Minimax`
- Alpha-Beta prunning, Wikipedia, Aktualizacja 12 czerwca 2021,
  dostęp w Internecie:`https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning`