# Sztuczna inteligencja – sprawozdanie projektu


| | |
|---|---|
| Pitas Jan | 160805 |
| Preugszas Dominikus Noah | 160752 |
| Przytarski Paweł | 160612 |


## Opis Problemu

Głównym celem projektu było napisanie algorytmu sztucznej inteligencji, która jest w stanie wykonywać najbardziej optymalne ruchy w grze Reversi (zwana też Othello).

Ideą było stworzenie struktury drzewiastej zawierającej, począwszy od danego stanu gry, jak najwięcej możliwych kolejnych stanów, na podstawie których za pomocą odpowiedniej heurystyki możliwe będzie wybieranie drogi, która prowadzi do zwycięstwa.

**Komentarz [D1]:** Na chwilę obecną nie przyszło mi nic lepszego do głowy, może macie jakiś pomysł

## Teoretyczny opis użytych metod

Został użyty (zmodyfikowany) algorytm minimax. Oryginalny algorytm polega na ocenie kolejnych liści (stanów) począwszy od liści od korzenia (stanu obecnego). Ocena ta jest miarą opłacalności doprowadzenia gry do tego stanu. Algorytm minimax „zakłada" z góry, że przeciwnik wykonuje tylko optymalne ruchy. Liście poszczególnych poziomów są, począwszy od korzenia, naprzemiennie oznaczane jako liście MAX, MIN, MAX itd. Liście MAX wybierają tylko wartość maksymalną pośród dzieci, a liście MIN tylko wartość minimalną. W ten sposób symulowany jest przebieg gry, w którym SI oraz przeciwnik (gracz) wykonują same optymalne ruchy, co pozwala na wybór optymalnej ścieżki.

**Komentarz [D2]:** Dla mnie osobiście nazwa minimax (minmax + i) podoba się lepiej. W całym dokumencie jest użyta nazwa minimax.

Na potrzeby projektu algorytm minimax musiał zostać wzbogacony o kilka własności, gdyż warunki w którym musi działać powodować mogą spore spowolnienie programu. Algorytm musi działać iteracyjnie, zawierać metodę α-β, dynamicznie reagować na zmiany[1] oraz działać na wielu wątkach.

Aby umożliwić dynamiczną reakcję oraz wielowątkowość algorytm musi działać iteracyjnie. Aby rozrywka nie cierpiała z powodu powolnego algorytmu, stosowana jest optymalizowana metoda α-β. Algorytm ze względu na poziom skomplikowania oraz przypuszczalnie dużej ilości danych do przetwarzania powinien posiadać możliwość korzystania z jak największej ilości zasobów systemu. Aby korzystać z możliwie największych zasobów algorytm musi być w stanie wykryć dostępne zasoby systemowe. Dynamiczne reagowanie na zmiany w stanie gry mają pozwolić na efektywniejsze wykorzystanie z dostępnej pamięci oraz czasu

**Komentarz [D3]:** Tutaj też nie miałem pomysłu na lepszy punkt. Chciałem, żeby to zdanie było w miarę krótkie, aby nie dostać mindfuck'a czytając je.

**Komentarz [D4]:** Jest opisane w części Pawła, nwm czy zostawić czy nie.

**Komentarz [D5]:** Opisać redukcję alfa-beta?

[1] **dynamicznie reagować na zmiany** - usuwać niepotrzebne części drzewa, uwzględnić nowe wygenerowane poziomy drzewa

procesora. Na przykład  kiedy gracz wykonuje ruch zwolniona zostanie część pamięci, w której znajdują się stany niepotrzebne.

Komentarz [D6]: Nie wiem, czy ten opis jest konieczny i dobry.

Oprócz wydajnego algorytmu generowania drzewa potrzebna jest odpowiednia dla Reversi heurystyka, która pozwoli na dokonanie dobrej oceny stanów oraz w konsekwencji dobrego wyboru kolejnego stanu (ruchu).

## Opis realizacji zadania

Komentarz [D7]: Postanowiłem wstawić część Pawła do opisu realizacji zadania. Nie wiem, czy to jest dobrze pomyślane przeze mnie

**Przekształcenia oryginalnego algorytmu:**

Oryginalny algorytm minimax w wersji rekurencyjnej został przekształcony na wersje iteracyjną przy pomocy drzewa. Algorytm w każdej iteracji przetwarza wszystkie aktywne liście (czyli takie, które nie podległy redukcji i mogą posiadać dzieci) zaczynając od korzenia. Dla każdego liścia wygenerowany zostaje zestaw dzieci, (który będzie przetworzony w następnej iteracji wraz z dziećmi pozostałych liści) oraz następuje propagacja nowej wartości stanu w drzewie (zgodnie z zasadą naprzemiennej maksymalizacji i minimalizacji aktualizuje wszystkich przodków na których wpłynie dodanie nowego liścia do drzewa) z uwzględnieniem algorytmu alfa-beta.

Komentarz [D8]: coś w stylu DFS<tutaj sprawdzić, czy to było to przeszukiwanie poziomami>

Taka modyfikacja algorytmu pozwoliła na wprowadzenie trzech modyfikacji do oryginalnego minimaxa.

**Generowanie nowych poziomów drzewa**

Pierwsza to „nieskończone” maksymalne zagłębienie – algorytm może cały czas pracować dodając kolejne poziomy do drzewa stanów, dzięki czemu im więcej czasu ma algorytm tym głębiej może przewidzieć rozwój gry. A w przypadku, gdy liczba stanów jest zbyt małą, by pokryć minimalne zagłębienie, wystarczy poczekać.

**Optymalizacja**

Druga modyfikacja polega na zachowywaniu poddrzewa – w momencie, gdy SI lub oponent dokona ruch, wybrane dziecko korzenia (wybrany ruch) jest przesuwane na miejsce korzenia (staje się obecnym stanem). Dzięki temu nie trzeba od nowa wyliczać całego drzewa, co przekłada się na szybkość algorytmu. Przy każdym ruchu drzewo zawsze traci tylko jeden poziom (bo SI zawsze wybiera poddrzewo najbardziej optymalne).

### Redukcja α-β

Nieco gorzej jest w przypadku ruchu gracza, który może dokonywać nieoptymalnych ruchów, wówczas drzewo może zostać skrócone do poziomu pojedynczego korzenia. Żeby się przed tym zabezpieczyć wprowadzono ograniczenie na używanie α-β redukcji (działa ona dopiero od pewnej wysokości drzewa). Dzięki czemu wysokość drzewa nigdy nie spada poniżej pewnego poziomu.

**Komentarz [D9]:** Tego nie było w nowszej wersji sprawozdania, ale uważam, że to jest ważne

### Wielowątkowość

Trzecia modyfikacja to wprowadzenie wielowątkowości. Dzięki przechowywaniu wszystkich liści do przetworzenia w postaci listy, można rozdzielić ich przetwarzanie pomiędzy kilka, a nawet kilkadziesiąt wątków (każdy wątek wylicza dzieci dla przetwarzanego liścia i dokonuje propagacji nowej wartości w drzewie), co pozwala na pełne wykorzystanie dostępnego sprzętu.

### Heurystyka

Użyta została heurystyka wartości pola , a dodatkowo funkcja sprawdza, czy dany stan gry jest stanem końcowym. Jeśli tak jest funkcja sprawdza, który gracz wygrał i zwraca wartość MAX lub -MAX. Dzięki temu SI dąży do najszybszego zwycięskiego zakończenia rozgrywki i unika sytuacji, gdy gra kończy się przegraną pomimo stosunkowo wysokiego wyniku (w porównaniu do stanów sąsiednich, które mogą być niskie ze względu na to, że algorytm nie obliczył wystarczającego głębokiego drzewa).

## Prezentacja oraz dyskusja osiągniętych wyników

**Komentarz [D10]:** Nie wiem czy na Git'cie jest najnowsza wersja, oraz jest późno w chuj, więc nie zacznę tej części jeszcze