**Możliwe parametry włączenia programu**

- black – czarne pionki sterowane przez gracza,

- white – białe pionki sterowane przez gracza,

- comps – rozgrywka dwóch komputerów

Domyślne włączenie tak jak w trybie white.

**Opis działania algorytmu**

SI przyjmuje obecny stan planszy i podejmuje decyzję o wykonaniu algorytmu minmax albo wybraniu ścieżki z systemu pamięci z prawdopodobieństwem proporcjonalnym do odsetku wygranych gier od danego ustawienia planszy. Przy wyborze tej pierwszej opcji dla każdego możliwego ruchu danego gracza ewaluowana jest wartość planszy wynikowej ruchu., aż do osiągnięcia maksymalnej głębokości lub warunku zwycięstwa. Najlepszy znaleziony ruch zostaje wykonany.

Aby ograniczyć ilość sprawdzanych możliwości wykorzystaliśmy obcinanie alfa-beta. Gdy jasne stanie się, że przeciwnik nie obierze ścieżki, która poprowadzi do sprawdzanego poddrzewa, wycofujemy się z niego. Dodatkowo, aby wprowadzić element losowości, jeżeli funkcja ewaluacyjna zwróci wartość równą obecnej, istnieje szansa na pozostanie przy obecnych wartościach albo zmianę dotychczasowego najlepszego ruchu.

Funkcja ewaluacyjna jest obliczana na podstawie kilku zmiennych w danym stanie:

* ile skoków dzieli króla od rogu – im więcej skoków, tym lepsza waga dla gracza czarnego, z górnym ograniczeniem ilości skoków wpływających na wynik
* ile pionków ma każdy z graczy,
* ile potencjalnych możliwości zbicia ma każdy gracz,
* ile pionków gracza jest otoczonych przez ile pionków przeciwnika.

Aby odnaleźć ilość skoków wykorzystaliśmy algorytm BFS, aby na pewno znaleźć drogę do najbliższego dostępnego wierzchołka.

**Opis działania systemu pamięci**

Oprócz standardowego podejścia zakładającego użycia algorytmu min/max i obcinania alfa/beta postanowiliśmy dodatkowo zamontować system uczenia się. Zakłada on zapamiętywanie dotychczas wykonanych ruchów i ich konsekwencji w celu późniejszego ich wykorzystania do skuteczniejszej gry.

**Założenie**

Każda rozgrywka składa się z kolejnych ustawień planszy. Po zakończeniu rozgrywki wiemy że ustawienie przedostatnie informuje gracza mogącego się wtedy poruszyć że dotychczas miała miejsce jedna sytuacja kiedy taki stan był kiedykolwiek osiągnięty i został zakończony wygraną, a ustawienie przed-przedostatnie informuje że była jedna sytuacja o takim stanie i zakończyła się przegraną dla znajdującego się w niej gracza. Idąc tak dalej raz obraną drogą możemy skomponować graf stanów planszy którego węzły będą wskazywać na stany kolejne, które miały miejsce oraz zawierać informacje o tym ile razy dany stan był osiągany w historii rozgrywek oraz ile z tych stanów kończyło się wygraną.

Początkowo taki graf nie będzie zbyt użyteczny gdyż bardzo często będą się pojawiać węzły opisujące stany które miały miejsce tylko raz i albo prowadziły do zwycięstwa lub przegrania, co nie uwzględnia ogromnej liczby innych możliwych posunięć, lecz z biegiem czasu kolejne stany będą uzupełniane o dodatkowe przejścia i ich konsekwencję w rezultacie czego nasza SI będzie mogła podejmować bardziej odpowiedzialne decyzje np.: kiedy dany stan miał miejsce 200 razy z czego 160 zakończyło się wygraną można zakładać, że jest to dobry ruch.

**Siła pomysłu**

Największą siłą takiego podejścia jest z pewnością to że nie jesteśmy ograniczani przez możliwości samej SI a przez umiejętności gracza z którym konkuruje. Nasza baza danych jest aktualizowana o ruchy SI oraz gracza ludzkiego w rezultacie czego może nauczyć się strategii tegoż gracza. Właściwie nie ma nawet potrzeby implementacji algorytmu min/max gdyż można wykonywać na początku losowe „głupie” posunięcia podczas gdy gracz ludzki będzie „mądrze” grał. Używając bazy danych z biegiem czasu posunięcia naszej SI będą coraz bardziej przemyślane. Należy jednak pamiętać że przy takim podejściu czas uczenia może zająć znacznie więcej czasu.

**Problem**

Należy pamiętać że dany stan może pojawić się w różnej kolejności w różnych rozgrywkach. Może to być zupełnie nowy stan albo już taki wielokrotnie odwiedzony. Potrzebujemy więc odpowiedniego sposobu na sprawdzenie czy dany stan miał miejsce i jeżeli go jeszcze nie było to stworzyć węzeł go identyfikujący (Przechowywanie wszystkich możliwych stanów oczywiście odrzucamy gdyż zajęło by to gargantuicznie dużo miejsca). Ważny jest też krótki czas sprawdzania czy dany stan pojawił się już czy nie.

**Rozwiązanie Problemu**

Zastosowanie struktury drzewa binarnego z algorytmem DSW wydaje się bardzo sensowne gdyż mamy logarytmiczny dostęp do danych czego nie odczujemy wraz z rozrostem pamięci (na czym nam z kolei zależy). Pozostaje sprawa odpowiedniego identyfikowania stanu w drzewie. To z kolei można rozwiązać przez odpowiednie generowanie Id n podstawie ułożenia planszy. Tak się złożyło że zastosowanie czterech intów wystarczy do takowej reprezentacji:

**Plansza = 9x9 = 81 pól; Narożniki = 4 pola; Środek = 1 pole (narożniki i środek nieistotny w obliczeniach)**

**Plansza – Narożniki –Środek = 76 pól**

**Każde pole może przyjmować 4 stany: 0-puste, 1-biały pionek, 2-czarny pionek, 3-król**

**Król jest tylko jeden więc możemy go potraktować jako biały pionek. Mamy więc 3 stany na pole.**

**3^20 = 3486784401; 2^32 = 4294967296 => 3^20 < 2^32 < 3^21; więc jednym intem możemy opisać 20 pól**

**Czwarty int opisuje tylko 16 pól co pozwala nam na pozostałych 4 zawrzeć informację o pozycji króla (3^4 = 81)**

Mamy więc złożony identyfikator który można porównywać z innymi i wstawiać do drzewa.

**Dodatkowo**

Za każdym razem kiedy odpalana jest gra plik z zapisaną bazą danych jest ładowany lub tworzony nowy w wyniku błędu, oraz zapisywany przy każdorazowym poprawnym zakończeniu rozgrywki.