Sztuczna inteligencja i inżynieria wiedzy

Algorytmy rozwiązywania gier

Sprawozdanie 3.

Ewa Skórska 212151

1. **Gra Stratego – zasady gry**

Gra Stratego polega na naprzemiennym kolorowaniu kratek planszy przez dwóch graczy, i zdobywaniu w ten sposób punktów. Punkty zdobywa gracz, który zamknie linię pomiędzy granicami planszy (w poziomie, pionie lub po skosie), niezależnie od koloru kratek w linii – liczy się tylko ostatni ruch. Jeśli zamalowanie jednej kratki zamyka kilka linii, punkty zliczane są osobno za każdą linię, a następnie sumowane. Gra kończy się po zamalowaniu ostatniej kratki.

Na potrzeby zadania przyjmuje się, że plansza jest kwadratem o dowolnej długości boku.

1. **Min-Max – opis algorytmu**

Algorytm Min-Max opiera się na założeniu, że dwóch graczy wykonuje zawsze najlepszy dla siebie ruch, co sprowadza się do tego, że z perspektywy oceny danego gracza on sam wykonuje ruch maksymalizujący zysk, natomiast jego przeciwnik wybiera ruch minimalizujący.

Do oceny każdego ruchu używa się przeszukiwania wgłąb drzewa do pewnej założonej głebokości (lub też w założonym czasie). Ponieważ zwykle przy dużej złożoności nie wystarcza czasu czy też mocy obliczeniowej na przeglądnięcie wszystkich możliwych ścieżek rozgrywki, na pewnym poziomie do oceny ruchu potrzebna jest heurystyka.

**Heurystyka funkcji oceny**

Za ocenę danego posunięcia przyjęto po prostu liczbę zdobytych dotychczas punktów co może bywać mylące, ponieważ punkty przynoszące największy zysk wykonywane są na samym końcu. W połączeniu jednak z pozostałymi elementami gry heurystyka ta wydaje się być wystarczająca, ponieważ bardzo cięzko jest wygrać z tym algorytmem,

**Heurystyka wyboru kolejnego ruchu**

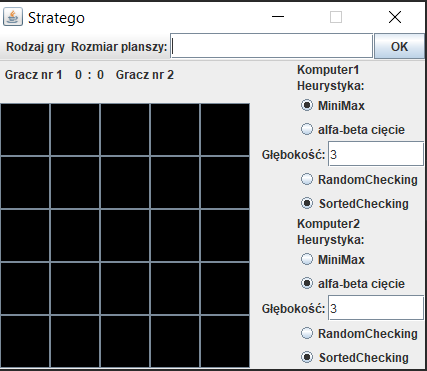
Tutaj zastosowano kilka heurystyk wykonywanych w zależności od sytuacji:

* Jeśli do końca gry pozostało więcej niż 9 ruchów (wyznaczone empirycznie ze względu na możliwosci obliczeniowe komputera) to:
* Wybierz punkt, który da ci najwięcej punktów w tym ruchu
* Jeśli za żaden ruch nie zarobisz punktu wybierz podzbiór takich, które są bezpieczne, co oznacza ruchy, po których wybraniu przeciwnik nie może zamknąć lini. Jeśli brak takiego ruchu wybierz ruch losowy
* Jeśli do końca gry zostało 9 lub mniej ruchów zastosuj min-max

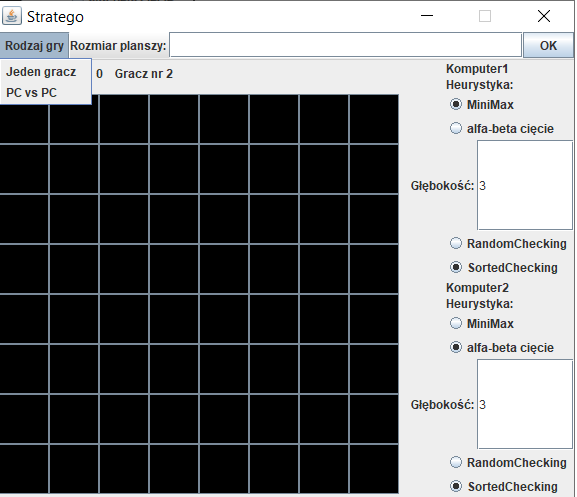
1. **Alfa-beta cięcie – opis algorytmu**

Ten algorytm jest ulepszoną wersją algorytmu min-max. W tym wypadku przechowujemy podczas eksploracji drzewa gry przechowujemy dwie wartości alfa – najlepszego znalezionego ruchu dla gracza maksymalizującego zysk (max) i beta – najlepszego ruchu dla gracza minimalizującego zysk (min). Jeśli beta (minimalizujące zysk) stanie się większe od alfa (które go maksymalizuje) ignorujemy dalszą eksplorację tej części drzewa. Algorytm ten w najgorszym przypadku jest równie efektywny co min-max, natomiast przy takim ułożeniu elementów drzewa, że najlepsze ruchy znajduje na początku (wersja optymistyczna), może w tym samym czasie zbadać dwukrotnie głębiej drzewo ruchów.

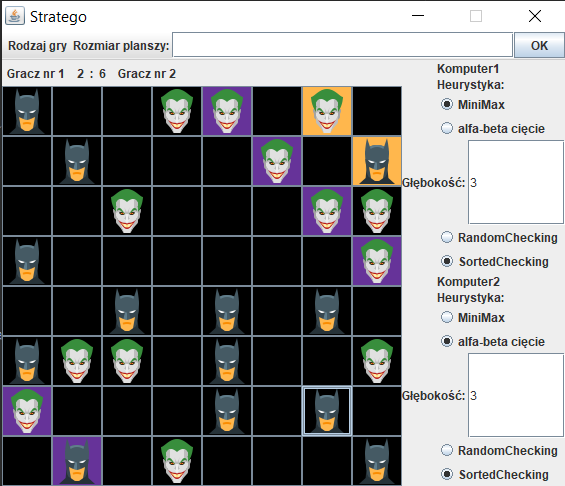
1. **Opis implementacji – GUI**



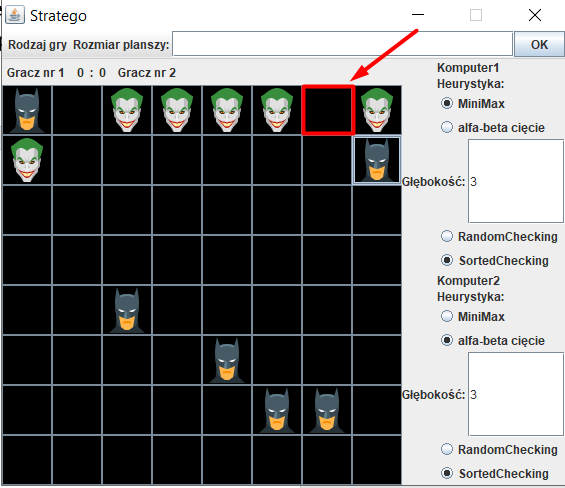
Po uruchomieniu programu pojawia się okienko z planszą, w którym możemy wybrać zarówno parametry gry dla obu komputerów, jak i rozmiar planszy (zatwierdzany przyciskiem „ok”). Domyślnie ustawione jest 5x5.



Grę rozpoczyna wybranie rodzaju gry. Przy wyborze gry przeciwko komputerowi zawsze rozpoczyna użytkownik. Można natomiast wybrać rodzaj gry komputera ustawiając parametry komputera1.



Na powyższym obrazku widzimy grę z opcją RandomChecking (opcja SortedChecking automatycznie ustawia się po rozpoczęciu gry). Na żółto oznaczone są pola zdobyte przez gracza pierwszego (użytkownika), a na fioleto przez gracza drugiego (komputer). Na pasku ponad planszą wyświetlona jest punktacja.



W zależności od wybranego trybu (na górze random checking, na dole sorted checking) komputer porusza się pseudolosowymi ruchami, lub wybiera każdą kolejną kratkę z zachowaniem opisanym powyżej. Jednocześnie możemy zauważyć, że w sytuacji kiedy gracz zagra ruch dzięki któremu komputer mógłby zdobyć kilka punktów ale pozwolić użytkownikowi zdobyć więcej w kolejnej turze, omija ten ruch.

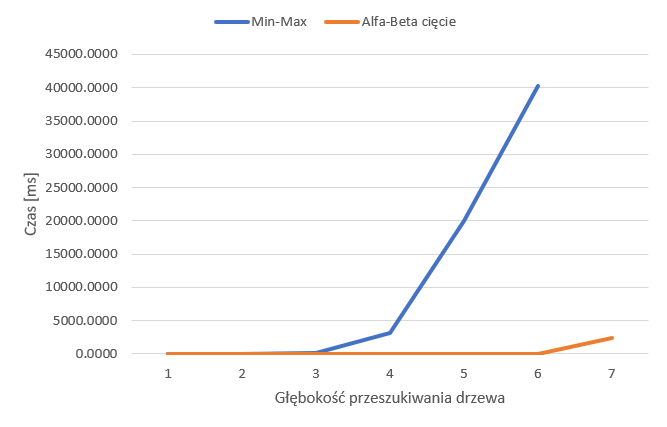
1. **Badania**

**Porównywanie czasów działania oraz liczby odwiedzonych węzłów algorytmów MinMax oraz AlfaBeta podczas 1 ruchu na planszy 7x7.**

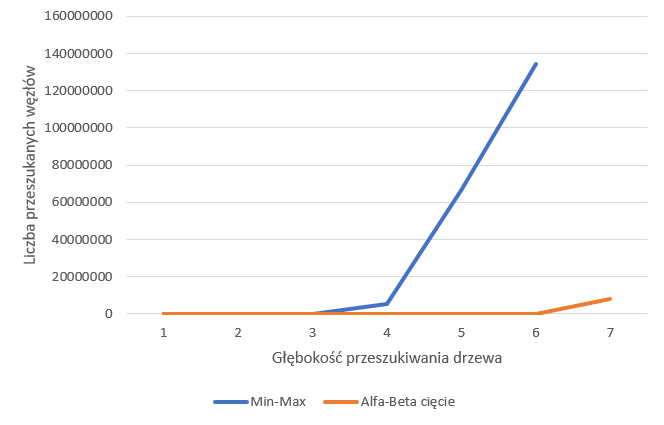
|  |  |  |
| --- | --- | --- |
| Głębokość przeszukiwania | MinMax [ms] | AlfaBeta |
| 1 | 0,742 | 0,4254 |
| 2 | 1,7651 | 0,8749 |
| 3 | 84,9781 | 1,5586 |
| 4 | 3128,9502 | 3,6538 |
| 5 | 20039,666 | 29,468 |
| 6 | 40231,2766 | 69,4898 |
| 7 | Brak wyników w rozsądnym czasie | 2356,0245 |

**Wykres czasu czasu obliczania pierwszego ruchu**

|  |  |  |
| --- | --- | --- |
| l. odwiedzonych węzłów | MinMax | AlfaBeta |
| 1 | 49 | 25 |
| 2 | 2401 | 73 |
| 3 | 112945 | 2195 |
| 4 | 5197969 | 6521 |
| 5 | 6693625 | 72860 |
| 6 | 134205625 | 181630 |
| 7 | Brak wyników w rozsądnym czasie | 7873392 |



**Wykres liczby odwiedzonych węzłów w pierwszym ruchu**



Z powyższych wykresów wynika, że choć początkowo wyniki obu algorytmów różnią się nieznacznie, to wraz ze wzrostem głębokości przeszukiwania drzewa Min-Max potrzebuje coraz więcej czasu, natomiast wykresy Alfa-Beta są niemal liniowe. Dzieję się tak ponieważ alfa-beta odcina węzły które nie mają szans na bycie wybranymi a tym samym rozpatruje o wiele mniej możliwości.

**Badanie skuteczności algorytmów**

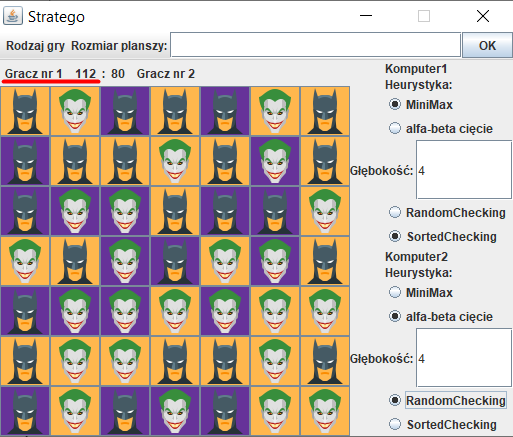
W poniższym badaniu sprawdza się skuteczność obu algorytmów w pojedynku przeciw sobie. Brane są pod uwagę różne parametry, nie badana jest natomiast głębokość powyżej 4 ponieważ sprzęt nie pozwala na tak dużą złożoność obliczeniową.

Dla obu algorytmów parametry są równe:

Rozmiar planszy: 7x7 – nieparzyste; Głębokość: 4

Min-Max vs Min-Max

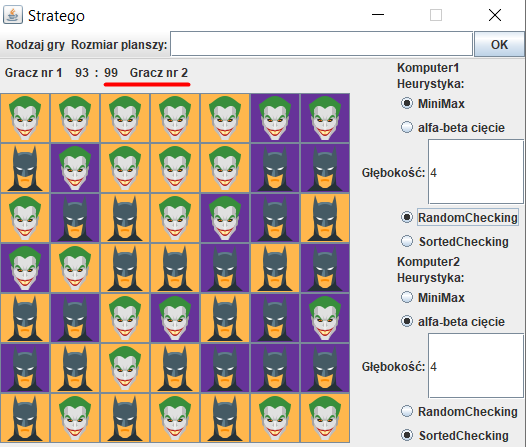
Player1: SortedChecking vs Play2: RandomChecking

 Zwycięzca: Min-Max sortedChecking

Rozmiar planszy: 7x7 – nieparzyste; Głębokość: 4

Min-Max vs Min-Max

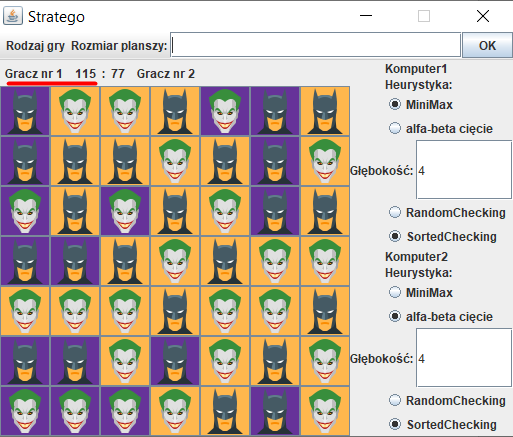
Player1: RandomChecking vs Play2: SortedChecking

 Zwycięzca: Min-Max sortedChecking

Rozmiar planszy: 7x7 – nieparzyste; Głębokość: 4,

rodzaj doboru pól: SortedChecking

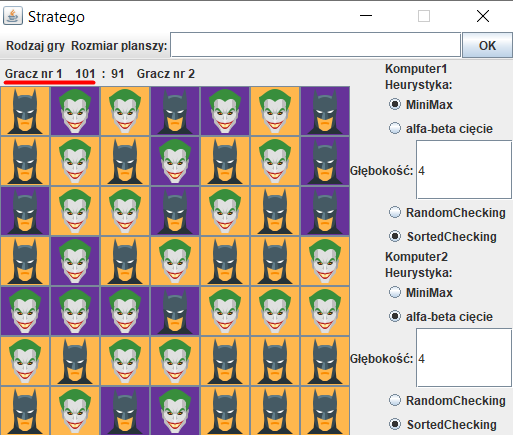
Player1: Min-Max vs Play2: Alfa-Beta

 Zwycięzca: Min-Max

Rozmiar planszy: 7x7 – nieparzyste; Głębokość: 4,

rodzaj doboru pól: SortedChecking

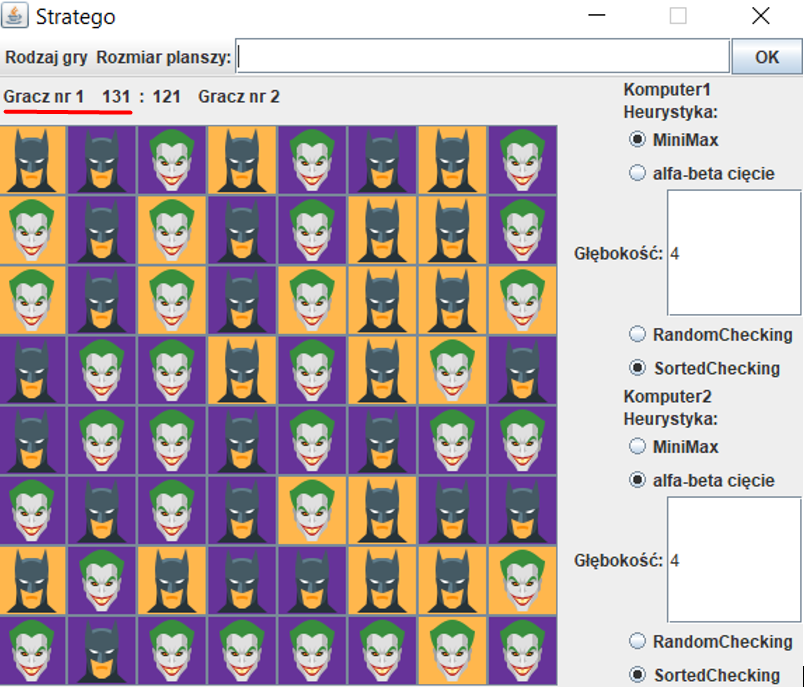
Player1: Alfa-Beta vs Play2: Min-Max

 Zwycięzca: Alfa-beta

Rozmiar planszy: 8x8 – parzyste; Głębokość: 4,

rodzaj doboru pól: SortedChecking

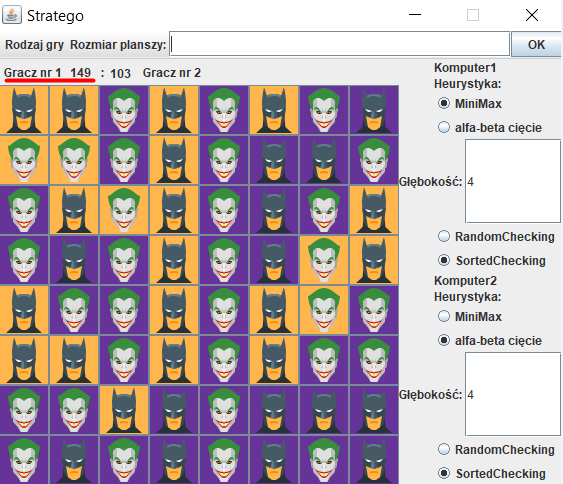
Player1: Min-Max vs Play2: Alfa-Beta

 Zwycięzca: Min-Max

Rozmiar planszy: 8x8 – parzyste; Głębokość: 4,

rodzaj doboru pól: SortedChecking

Player1: Alfa-Beta vs Play2: Min-Max

 Zwycięzca: Alfa-Beta

**Wnioski:**

W przypadku sortedChecking przeciwko randomChecking zawsze wygrywał algorytm sortujący, ponieważ miał on zadowalającą heurystykę.

Niezależnie od wielkości planszy wygrywał zawsze gracz 1 co da się całkiem łatwo wytłumaczyć. Oba algorytmy mają dokładnie takie same strategie i podejmują te same decyzje, a jedyną przewagą algorytmu alfa-beta cięcie jest czas wykonywania obliczeń. Ponieważ w tej implementacji każdy algorytm dostawał dowolną ilość czasu na obliczenia, algorytm min-max podejmował te same decyzje co alfa-beta, tylko że w dłuższym czasie.