

Sztuczna inteligencja i inżynieria wiedzy

ALGORYTYMY: MIN-MAX, ALFA-BETA (STRATEGO)

MICHAŁ KANAK 206906

Spis treści

Wstęp	2
Cel ćwiczenia	2
Zasady gry	2
Algorytm Min-Max	2
Algorytm Alfa-Beta cięć	2
Metody optymalizacji – Heurystyki	2
Stan gry	2
Wyboru pola	2
Implementacja algorytmu, programu, GUI	3
Algorytm ogólny	3
Aplikacja	3
Pojedynek SI vs SI	4
Pojedynek SI vs Gracz	6
Podsumowanie	7

Wstęp

Cel ćwiczenia

Celem ćwiczenia była implementacja algorytmu Min-Max oraz Alfa-Beta cięć w grze „Stratego”. Funkcjonalność gry miała pozwolić na rozegranie jej w kombinacjach sterowania: Człowiek-Komputer oraz Komputer-Komputer. Umieszczona w aplikacji sztuczna inteligencja symuluje ruchy gracza, utrudniając zdobywanie punktów oraz maksymalizację swoich wyników. Oprócz tego miała być stworzona logika odpowiedzialna za turowość, wygląd i mechanikę gry.

Zasady gry

- Gracz może uzyskać punkty za zaznaczenie pola kończącego daną kolumnę, wiersz lub którąś z przekątnych. Kombinacje sumują się.
- Gracz na turę wykonuje tylko jeden ruch.
- Gra kończy się w momencie zaznaczenia wszystkich pól oraz zliczenia punktów.

Algorytm Min-Max

Metoda która umożliwia minimalizację maksymalnych możliwych strat (maksymalizacja minimalnego zysku). Stosuje się go do podejmowania decyzji o kolejnych ruchach w rozgrywkach między graczami. Zarówno przy grach turowych jak i tych gdzie decyzja jest równoczesna. Poprzez obliczenie drzewa stanów wszystkich możliwych kombinacji ruchów (do pewnej ustalonej głębokości), SI dokładnie wie co powinno zrobić w przypadku danego ruchu gracza.

Algorytm Alfa-Beta cięć

Algorytm wspomagający min max poprzez redukcję liczby węzłów, które mogłyby być przeszukane niepotrzebnie. Wykorzystywany w grach dwuosobowych takich jak kółko krzyżyk, stratego, szachy. Ten mechanizm pozwala oszczędzić czas i zasoby potrzebne na przeszukiwanie nieoptymalnych ruchów. Czas przeszukiwania ograniczony zostaje do przeszukania najbardziej obiecujących poddrzew, w związku z czym możemy zejść głębiej w tym samym czasie. Tak samo jak klasyczny min-max, algorytm należy do algorytmów wykorzystujących metody podziału i ograniczeń.

Metody optymalizacji – Heurystyki

Stan gry

- Uzyskanie największej ilości punktów – minimalizacja zysku gracza

Heurystyka dąży do minimalizacji zysku przeciwnika – zaznacza pola, które uniemożliwią jednoczesne zdobycie zbyt wielu punktów, prowadzi do tego że SI zdobywa pola, które spełniają więcej niż 1 możliwość zdobycia punktów.

- Wybór pól jak najdalej od ostatniego ruchu gracza jeśli ten może zdobyć dużo punktów

Jeśli SI wykryje, że gracz dąży do zdobycia konkretnego pola w kolejnych ruchach, wykonuje ruchy odwodzące go od tego pomysłu lub takie by zapanować nad kolejnością ruchów – by to SI miało ruch w momencie zamknięcia danego obszaru.

- Liczenie drogi do zdobycia największej liczby punktów w ograniczonym zakresie przeszukiwania

Przeszukiwanie całego drzewa przy większych planszach może być zbyt czasochłonne, SI planuje sekwencje ruchów, które doprowadzą do zdobycia większej ilości punktów.

Wyboru pola

- „Karanie” za głębokość drzewa przy wyborze kolejnego pola

Podczas implementacji algorytmu zauważyłem bardzo dużą złożoność obliczeniową podczas wyboru kolejnego pola, stąd potrzeba minimalizacji wyników zależna od głębokości przeszukiwania. Przy okazji wybieramy najbardziej punktowane pola w krótkim procesie wyszukiwania.

- Wybór pól posiadających zaznaczonych sąsiadów

Heurystyka promuje pola znajdujące się blisko innych pól – im więcej pól w pobliżu tym lepszy mają wynik.

- Wybór pól znajdujących się po ukosie od aktualnie zaznaczonego pola

Jest to dobry sposób na zdobycie punktów oraz rozszerzanie obszarów działań, w ten sposób algorytm porusza się bardziej inteligentnie, ponieważ ruchy po skosie zamykają też często inne poziomy punktowe.

Implementacja algorytmu, programu, GUI

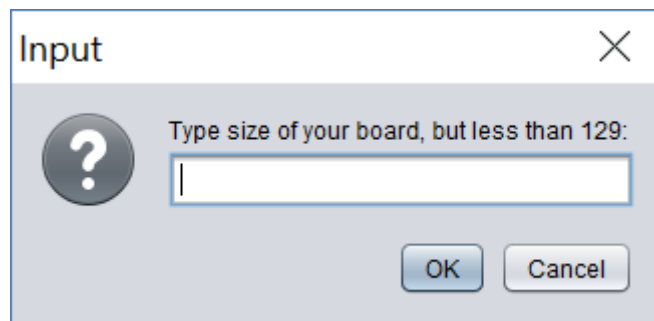
Program został napisany z użyciem języka Java w środowisku NetBeans 8.2 dla Javy. Gra została wyposażona w graficzny interfejs okienkowy. GUI składa się z przycisków oraz etykiet. Podczas rozgrywki SI vs SI konieczne było utworzenie osobnego wątku procesu, który umożliwia śledzenie zmian w grze. Gra pozwala na wybór pola o wielkości do 128 pól.

Algorytm ogólny

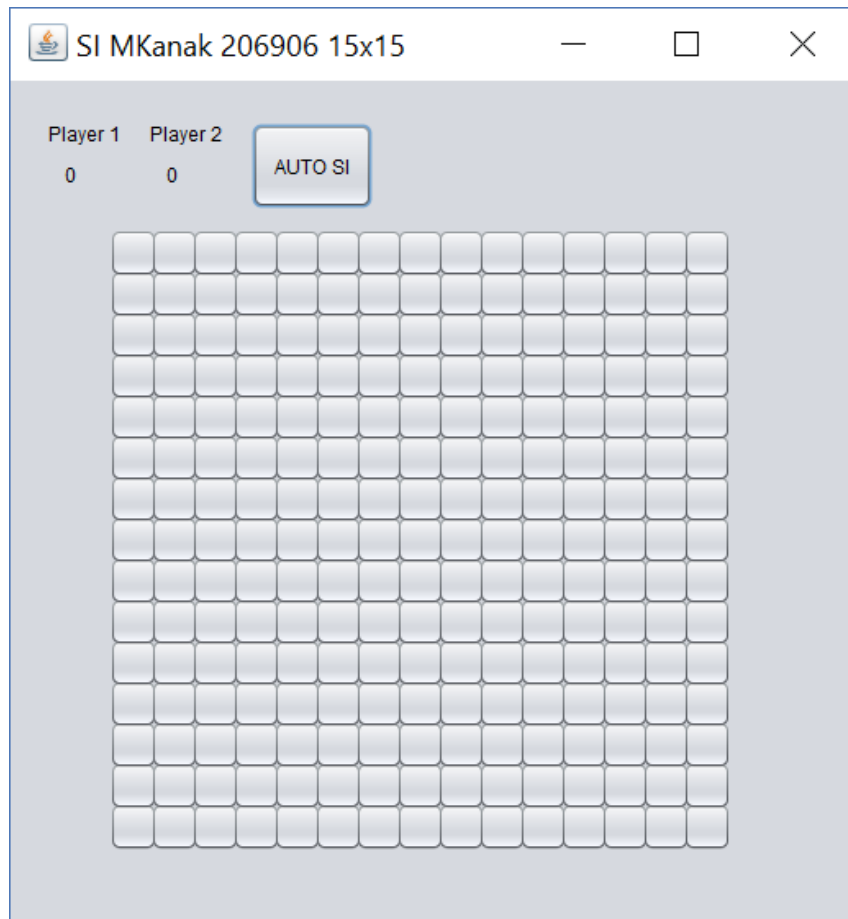
- Inicjalizacja interfejsu oraz ustalenie stanu gry na początkowy
- Utworzenie tablicy `ArrayList<ArrayList<Field>>`, która jest reprezentacją planszy gry.
- Po zaczęciu gry:
 - Zmiana tury na drugiego gracza oraz uruchomienie algorytmu SI:
 - Przekazanie aktualnego stanu gry (oraz stworzenie jego kopii)
 - Wybranie pola z pomocą Min-Max, Alfa-Beta + heurystyk(z przeszukiwaniem i symulacją zaznaczania pól)
 - Zaznaczenie pola wybranego przez algorytm
 - Zmiana tury na pierwszego gracza.

Aplikacja

Grę zaczynamy od podania rozmiaru planszy:



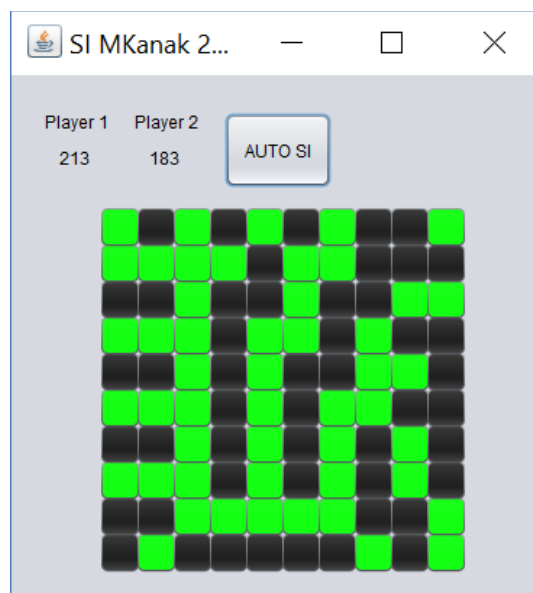
Stan początkowy planszy wygląda naępująco:

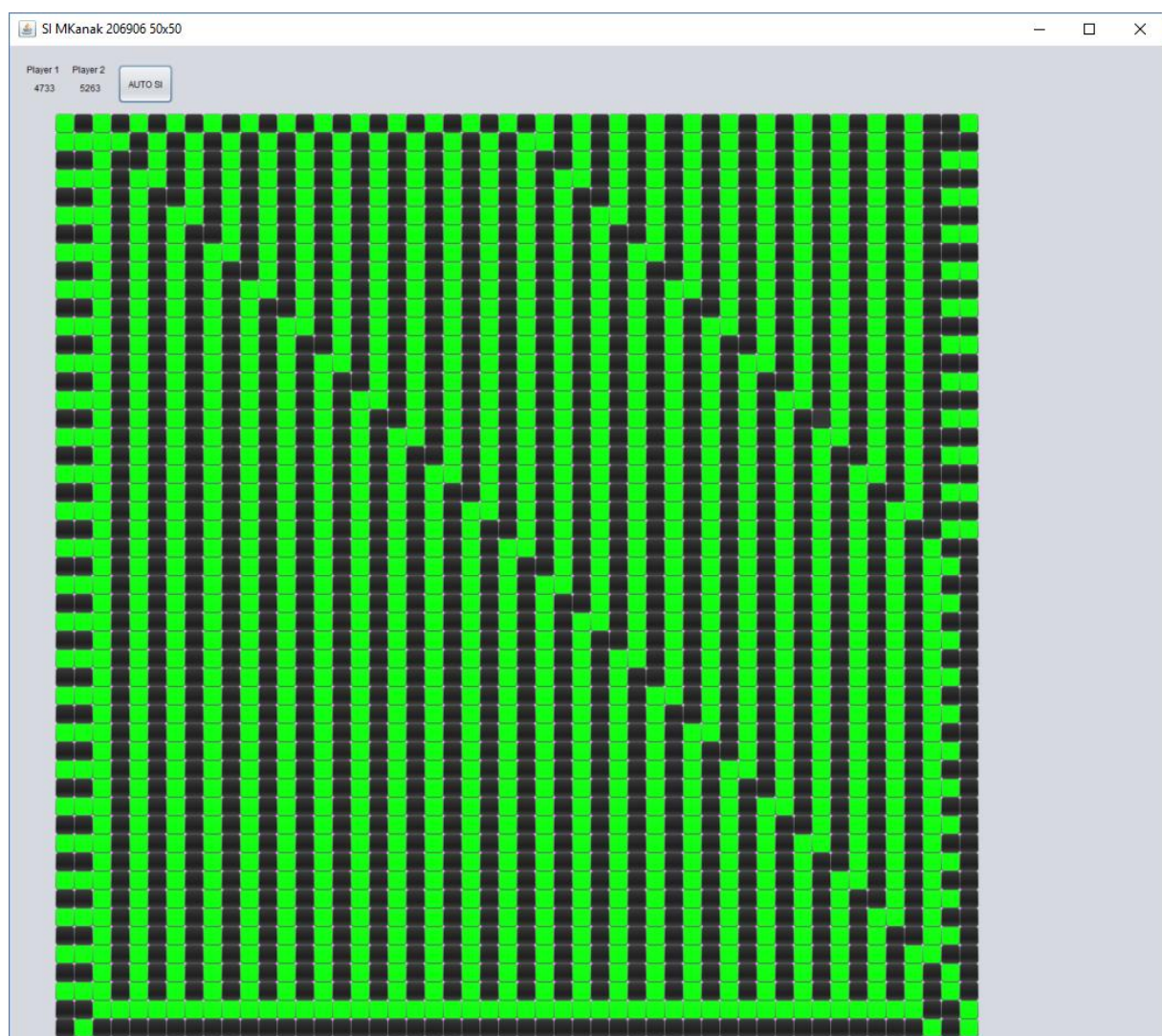
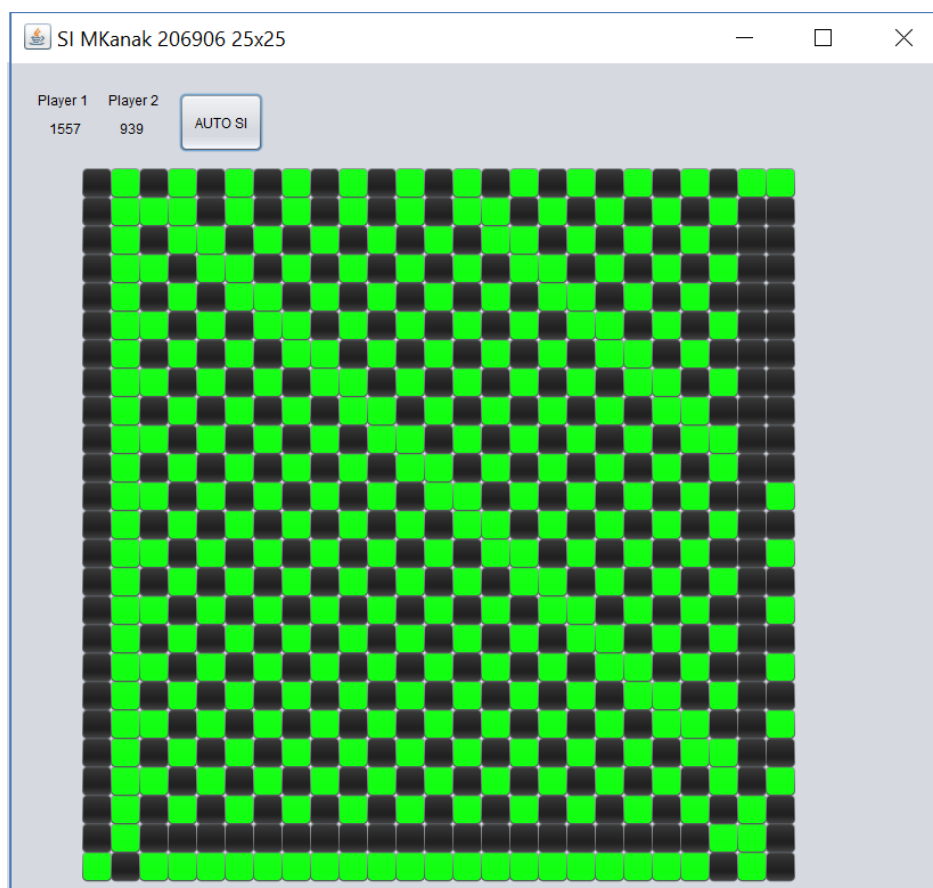


Plansza została zbudowana z komponentów JButton – wygenerowanych dynamicznie.

Pojedynek SI vs SI

Poniżej przedstawiam kilka przykładowych rozgrywek SI przeciwko SI.



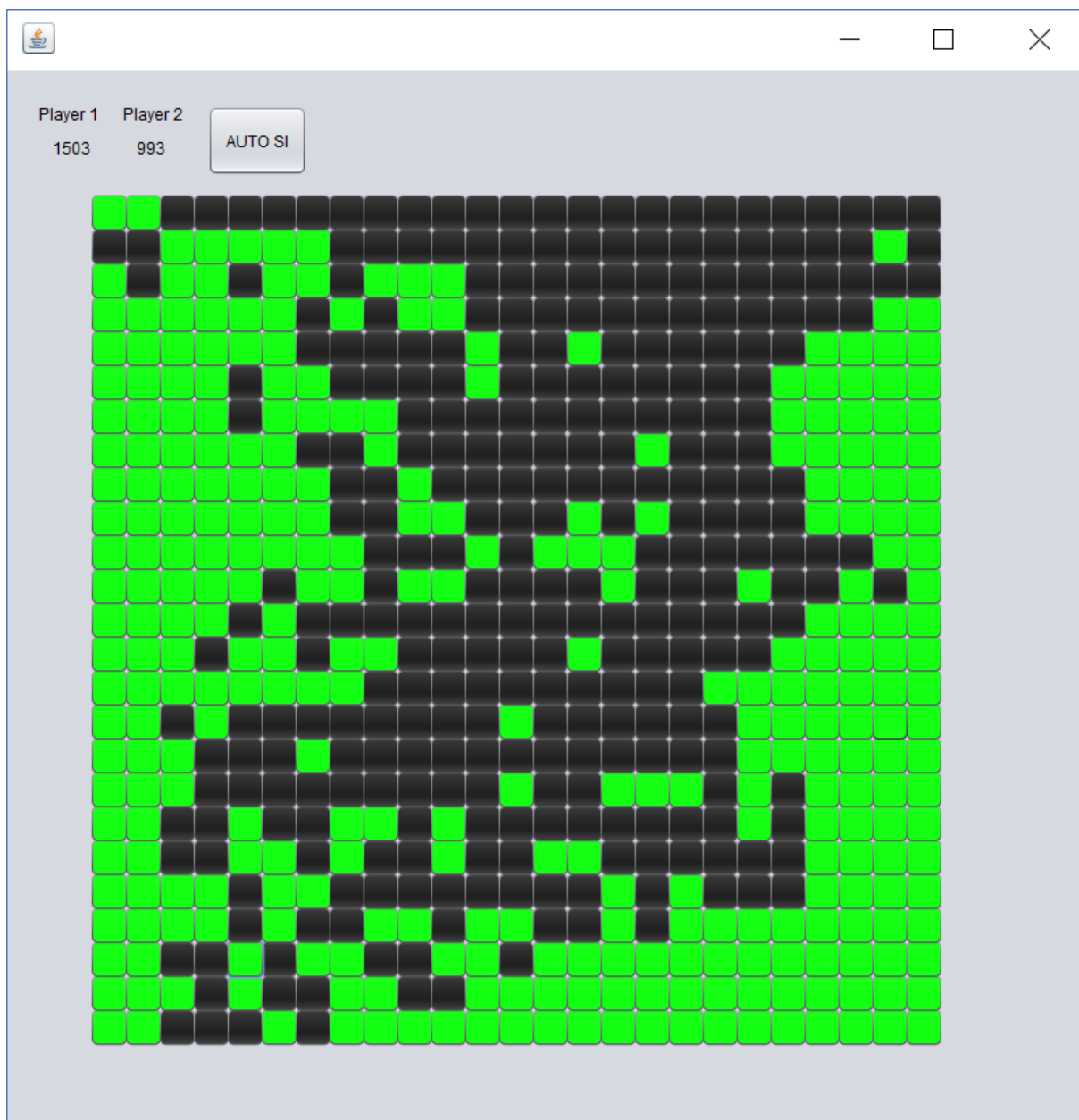
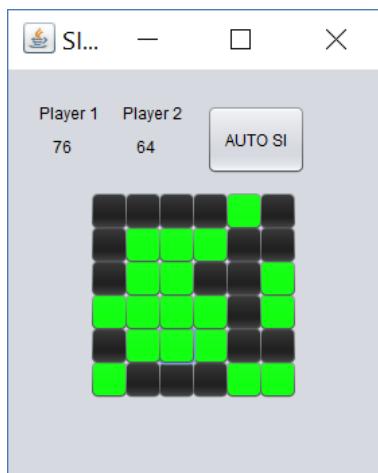


Co ważne czas wykonania programu nie rośnie w zastraszającym tempie.

Optymalizacja Alfa-Beta znacząco wpływa na szybkość decyzji SI dla większych plansz – zaczynających się od 6 pól wwyż. Ponieważ ucinamy zbyt głębokie przeszukania poprzez karanie ruchów głębokich.

Pojedynek SI vs Gracz

W trakcie gdy przeciw SI możemy wyraźnie zauważyć działanie algorytmu. SI gdy tylko ma możliwość zdobycia większej ilości punktów – robi to.



Si jest oznaczone czarnym kolorem.

Podsumowanie

Algorytmy zawarte w ćwiczeniu są średnio trudne do zaimplementowania, należy skupić uwagę na szczegółach implementacyjnych, tak by żaden z nich nie został pominięty. Zastosowanie ogranicza się do gier o sumie zerowej, jednak ich istnienie pozwala na przewidywanie ruchów oraz trenowanie osoby grającej np. w szachy. Czysty min max jest dość wolny i złożony dla plansz większych niż 6x6, przy planszy 8x8 jest to już 64! możliwych opcji, co znaczy że złożoność algorytmu oscyluje wokół $(x^2)!$. Co sprawia, że bez optymalizacji alfa-beta byłby niepraktyczny. Sama alfa-beta przyspieszyła realizację zadania o wiele tysięcy %. Dodatkowo stosując odpowiednie heurystyki dostosowane do aktualnego stanu gry, lub takie które typują kolejne pole do zaznaczenia – wpływają znacząco na „inteligencję” algorytmu i utrudniają poziom gdy. Dodatkowo dokładnie analizując dane układy figur (w szachach) jesteśmy w stanie zaprogramować najoptymalniejsze wybory – poza samym podstawowym wybieraniem pól i figur. Zastosowane algorytmy sprawiają że bardzo trudno jest wygrać z komputerem w tej rozgrywce. Przy walkach SI vs SI najczęściej wyniki są zbliżone do siebie. Jednak zdarzają się wyjątki przez stosowanie heurystyk oraz ograniczanie przeszukiwania drzewa. Realizacja zadania zajęła mi kilka dni, które spędziłem głównie na testowaniu kolejnych małych zmian w kodzie.