Politechnika Wrocławska

Projektowanie algorytmów i metody sztucznej inteligencji

Projekt
Termin: śr. 11:15

Projekt 3:
Gry & AI

Autor:
Michał Kuzemczak

Prowadzący:
dr inż. Łukasz Jeleń

5 czerwca 2019

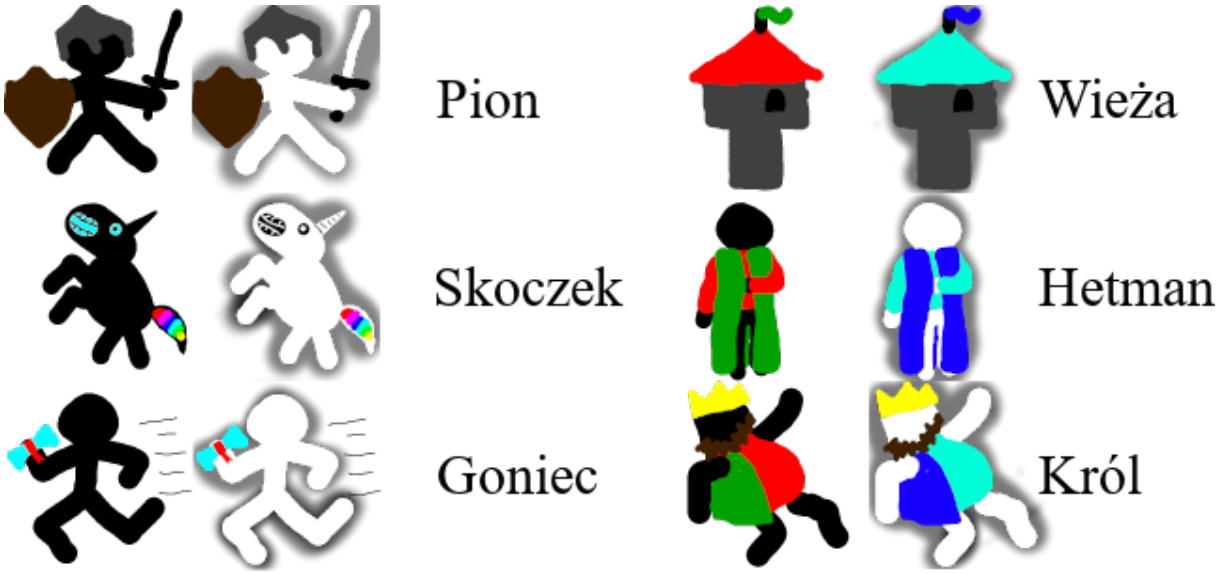
Politechnika
Wrocławska

1 Wprowadzenie

Sprawozdanie zawiera opis implementacji algorytmu MiniMax w napisanej w C++ grze, co pozwala by na granie przeciw komputerowi. MiniMax wzbogacono tutaj o algorytm Alfa-Beta.

2 Szachy

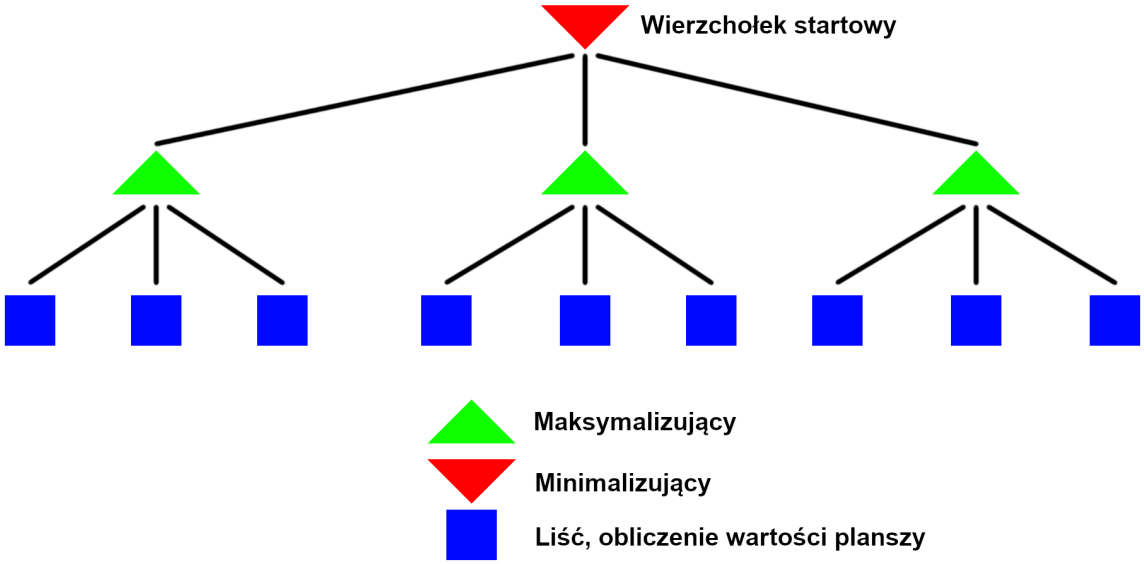
Tworzona gra to klasyczne szachy. **Rysunek 1** przedstawia wygląd figur.



Rysunek 1: Figury w grze.

3 Stosowane techniki SI

3.1 MiniMax



Rysunek 2: Struktura algorytmu MiniMax (poglądowa, w rzeczywistości więcej odnóg).

Algorytm MiniMax służy do rekursywnego przeszukiwania wszystkich możliwych ruchów na planszy i wybrania takiego, który może prowadzić do najlepszego możliwego wyniku dla drużyny wierchołka startowego.

Wynik gry określa się, sumując wartości liczbowe figur obecnych na planszy (nie zbitych):

Figura	Wartość liczbowa	
	Czarne	Białe
Pion	-10	10
Skoczek	-30	30
Goniec	-30	30
Wieża	-50	50
Hetman	-90	90
Król	-900	900

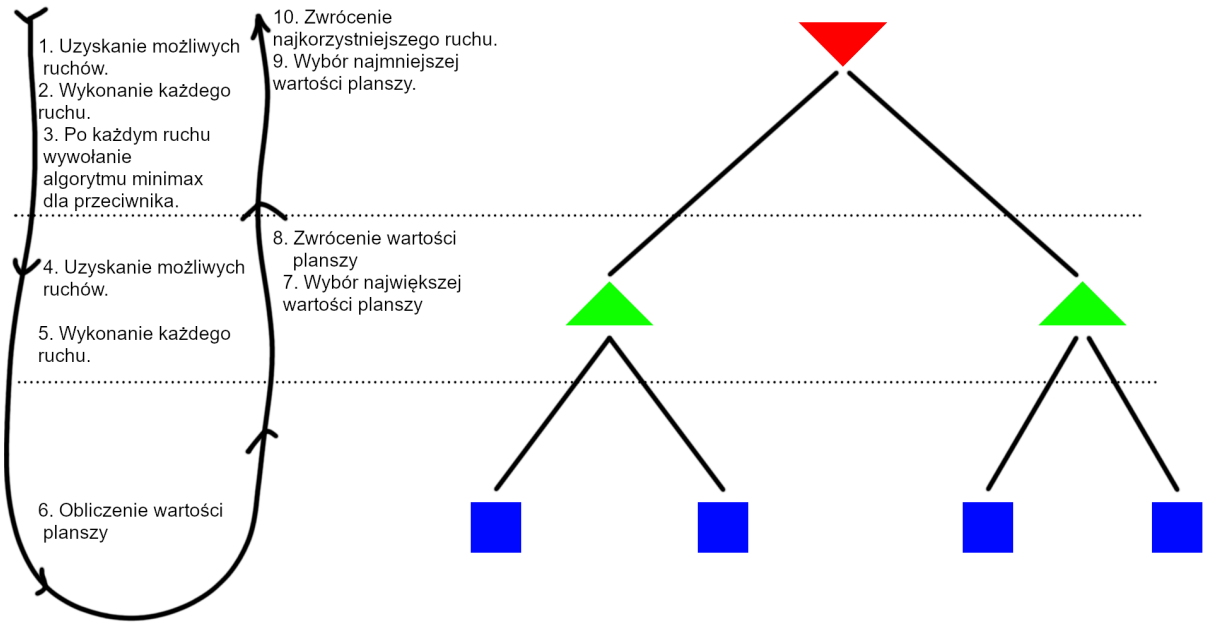
Tabela 1: Rozkład wartości figur

Król nie przypadkowo ma wartość bezwzględną większą niż suma pozostałych figur. Dzięki temu algorytm zawsze wybierze ruch, który nie zagraża królowi, nawet, jeśli miał by poświęcić inną figurę.

Przed zbiciem pierwszej figury w grze, wartość planszy, czyli suma wszystkich figur, wynosi 0.

Dzięki symetrycznemu rozkładowi drużyn względem zera, działanie algorytmu można oprzeć na maksymalizacji lub minimalizacji wartości planszy (zależnie od drużyny wierzchołka startowego).

Każdy wierzchołek, który nie jest liściem, to jedno rekurencyjne wywołanie funkcji minimax. W każdym takim wywołaniu algorytmu, w pierwszej kolejności uzyskuje wszystkie możliwe ruchy drużyny, którą wierzchołek reprezentuje. Następnie, w pętli, wykonuje każdy z tych ruchów i, jeśli wierzchołek-potomek nie jest liściem, następuje wywołanie minimax dla przeciwnej drużyny. Funkcja minimax zwraca najkorzystniejszą dla siebie wartość planszy. **Rysunek 3** przedstawia proces znajdywania najkorzystniejszego ruchu.



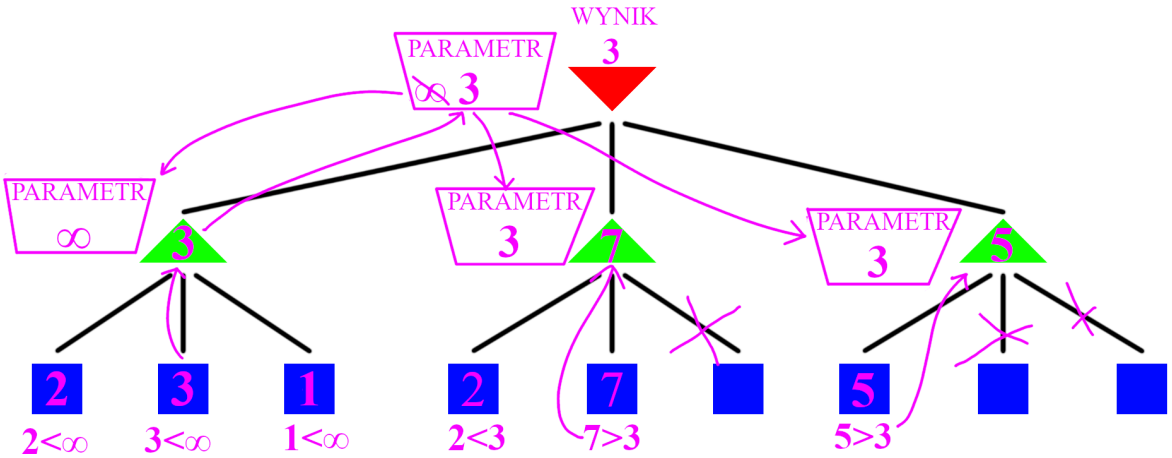
Rysunek 3: Proces znajdywania najlepszego ruchu. Głębokość: 2.

Głębokość rekurencji, czyli jednocześnie ilość przeszukiwanych ruchów w przód, określana jest przy wywołaniu funkcji. Parametr ten ma eksponencjalny wpływ na czas wykonywania algorytmu. Za jego pomocą określa się także poziom trudności gry.

3.2 Alfa-Beta

Algorytm Alfa-Beta to ulepszenie standaardowego minimax. Nie wpływa na wynik, poprawia jedynie czas wykonywania.

Alfa-Beta polega na ocenie, czy wierchołek-rodzic w ogóle wybierze wynik aktualnego wierzchołka. Jeśli nie, to nie ma sensu dalej przeszukiwać wierzchołków-potomków.



Rysunek 4: Wyjaśnienie przycinania Alfa-Beta.

Przebieg:

- W pierwszym kroku, parametr jest ustawiony na najgorszy dla wierzchołka startowego (minimalizującego), czyli $+\infty$.
- Następnie, parametr jest przekazywany do pierwszego potomka, wierzchołka maksymalizującego.
- Każde obliczenie wartości planszy w liściu pierwszego potomka jest mniejsze od parametru (wszystko jest mniejsze od nieskończoności), więc wszystkie liście są sprawdzane.
- Pierwszy potomek zwraca maksymalną wartość (3), która zastępuje parametr, ponieważ dla minimalizatora 3 jest lepsze niż ∞.
- Parametr (3) zostaje przekazany do drugiego potomka, maksymalizatora.
- W drugim liściu drugiego potomka, wartość planszy jest większa od parametru. Drugi potomek wie teraz, że zwróci co najmniej 7, bo jest maksymalizatorem. Jednak jego rodzic jest minimalizatorem, zatem będzie wolał 3 od 7. Drugi potomek na pewno nie zostanie wybrany, więc nie musi przeszukiwać trzeciego liścia. Zwraca 7.
- Podobna sytuacja występuje u trzeciego potomka. W pierwszym liściu dowiedział się, że zwróci co najmniej 5, a jego rodzic jako minimalizator będzie wolał 3 od 5. Niepotrzebne zatem jest przeszukiwanie pozostałych liści.

3.3 Dodatkowe ulepszenia

3.3.1 Porządkowanie ruchów

Dzięki zastosowaniu przycinania Alfa-Beta w sytuacji z **Rysunku 4**, ilość liści została zmniejszona z 9 do 6. Sytuacja była by jeszcze lepsza, gdyby, w pierwszym potomku najpierw został przeszukany liść o wartości 3, a w drugim wierzchołku liść o wartości 7. Ilość liści zmalała by z 9 do 3.

Z tego powodu w funkcji, która uzyskuje wszystkie dostępne ruchy dla danej drużyny, zastosowano **porządkowanie ruchów**. Ruchy, które biją najbardziej wartościowe figury, są ustawiane jako pierwsze, a ruchy ciche (nie bijące) jako ostatnie.

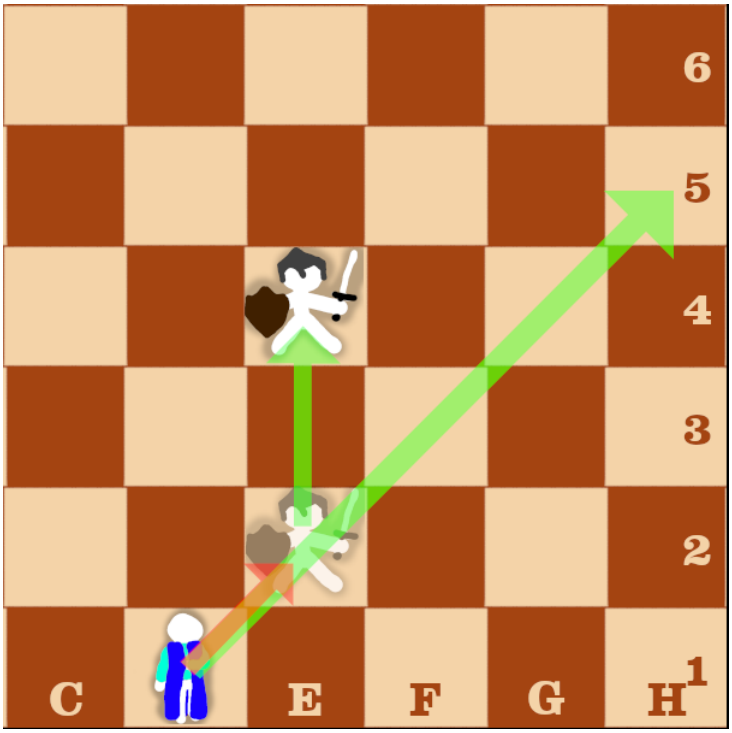
3.3.2 Usprawnienie generacji możliwych ruchów

W pierwszej wersji programu, każde wywołanie funkcji zwracającej listę możliwych ruchów wiązało się z pozyskiwaniem na nowo ruchów dla każdej figury z osobna. Było to powodem znacznie wydłużonego czasu działania algorytmu MiniMax, w którym każde wywołanie rekurencyjne wiąże się z pozyskiwaniem możliwych ruchów.

Optymalizacja w tym przypadku polegałą na tym, że stworzono strukturę danych, która przechowuje listę wszystkich ruchów według ich pól docelowych. Innymi słowy, stworzono listę, w której każdy rekord reprezentuje jedno pole, i do każdego takiego pola przypisana jest lista ruchów, które się na nim kończą.

Z użyciem takiej struktury danych, nie trzeba już za każdym razem, po każdym wykonanym ruchu, na nowo pozyskiwać wszystkich możliwych ruchów. Aktualizuje się tylko te, na które dana zmiana pozycji figury ma wpływ.

Przykładowo: Biały Pion wykonuje ruch z E2 na E4. Na polu D1 stoi Biały Hetman. Pion odblokował ruch promieniowy Hetmana, który teraz może się ruszyć na pola E2, F3, G4, H5 (**Rysunek 5**).



Rysunek 5: Przykład wpływu ruchu jednej figury na możliwe ruchy drugiej.

4 Podsumowanie

Minimax jest dobrym algorytmem do implementacji prostej SI. Atutem jest łatwość zmiany poziomu trudności poprzez głębokość przeszukiwania. Jest on jednak wrażliwy na wadliwą implementację mechanizmów szachowych. Wykonuje on bowiem znaczną ilość ruchów, i liczba ta rośnie wykładniczo wraz z głębokością. Z pomocą przychodzą modyfikacje takie jak przycinanie Alfa-Beta czy porządkowanie ruchów, które wielokrotnie skracają czas wykonywania.

5 Bibliografia

- Budowa szachów i minimax:
  https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/
  ostatnio odwiedzono: 1 czerwca 2019;

- Generacja ruchów, porządkowanie ruchów, minimax, struktury danych, figury, kodowanie ruchów:
  chessprogramming.org
  ostatnio odwiedzono: 1 czerwca 2019.