Politechnika Wrocławska

Projektowanie i Analiza Algorytmów

Projekt 3

Mateusz Marko ISA nr 273168

3.06.2024

1.Wprowadzenie

- Cel zadania

Celem ćwiczenia było stworzenie gry kółko i krzyżyk z możliwością wyboru przez gracza rozmiaru planszy i warunku wygranej. Aby osiągnąć ten cel, zastosowano algorytm minimax z cięciami alfa-beta oraz funkcję ewaluacyjną, które zostaną opisane poniżej. Gra została wykonana w wersji graficznej przy użyciu biblioteki SFML, co pozwala na interaktywną i wizualnie atrakcyjną rozgrywkę.

- Algorytm Minimax

Algorytm ten służy do znajdywania najlepszego ruchu w grach dwuosobowych o sumie zerowej, takich jak kółko i krzyżyk. Przeszukuje drzewo możliwych ruchów do określonej głębokości, przypisuje wartości funkcji heurystycznej liściom drzewa, a następnie propaguje te wartości wstecz, wybierając ruch maksymalizujący wynik dla gracza maksymalizującego i minimalizujący wynik dla przeciwnika. Złożoność czasowa algorytmu wynosi $O((n^2 - 1)!)$, a złożoność pamięciowa $O(n^2)$, gdzie n to liczba wierszy lub kolumn planszy.

- Cięcia alfa-beta

Cięcia alfa-beta są techniką optymalizacji algorytmu minimax, która zmniejsza liczbę przeszukiwanych węzłów w drzewie gry. Używają dwóch wartości granicznych: alfa (najlepsza wartość dla gracza maksymalizującego) i beta (najlepsza wartość dla gracza minimalizującego). Cięcia te eliminują gałęzie, które nie mogą wpłynąć na ostateczną decyzję, co w najlepszym przypadku redukuje złożoność czasową do $O(b^{d/2})$, gdzie b to liczba możliwych ruchów, a d to głębokość drzewa.

- Sprzęt do badań

Program został napisany obiektowo z podziałem na pliki nagłówkowe. Sprzęt na którym były wykonywane badania miał 16GB RAMu, a także ośmiordzeniowy procesor AMD Ryzen 7 2700.

2. Przebieg Ćwiczenia

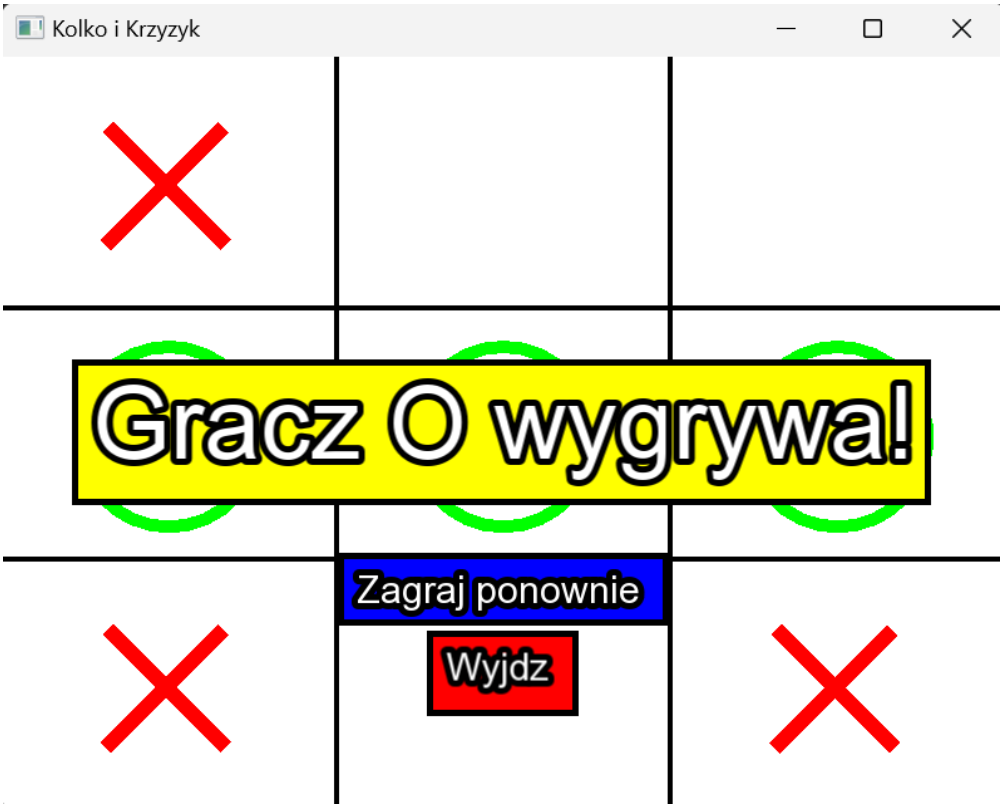
Celem ćwiczenia było stworzenie gry kółko i krzyżyk z możliwością wyboru przez gracza rozmiaru planszy i warunku wygranej. Do tego celu zastosowałem algorytm minimax z cięciami alfa-beta aby usprawnić działanie algorytmu. W zadaniu można było napotkać jednak parę problemów. Przy dużych rozmiarach planszy (np. 10x10), algorytm minimax potrzebował zbyt wiele czasu na przeszukiwanie wszystkich możliwych ruchów, co prowadziło do zawieszania się gry. Aby temu zaradzić, dodałem heurystyczne funkcje oceniające, które zapobiegały przeszukiwaniu całego drzewa możliwości. Funkcja ta ocenia aktualny stan planszy, przyznając punkty za różne układy znaków, co pozwala AI na szybsze podejmowanie decyzji. Dzięki temu a także dzięki zastosowaniu cięć alfa-beta gra stała się bardziej responsywna, nawet przy dużych rozmiarach planszy. W samej grze AI wykonuje ruchy na podstawie algorytmu minimax. Algorytm ten przeszukuje możliwe stany gry i wybiera ruch maksymalizujący szanse swojej wygranej, jednocześnie minimalizując szanse przeciwnika.

```
int Gra::ocen(std::vector<std::vector<char>>& plansza, char znak, char znakPrzeciwnika)
{
    //sprawdź wiersze
    for (size_t i = 0; i < rozmiar; i++) {
        int licznikAI = 0;
        int licznikPrzeciwnika = 0;
        for (size_t j = 0; j < rozmiar; j++) {
            if (plansza[i][j] == znak) {
                licznikAI++;
            }
            else if (plansza[i][j] == znakPrzeciwnika) {
                licznikPrzeciwnika++;
            }
        }
        if (licznikAI == warunekWygranej) {
            return 100;  //AI wygrywa
        }
        else if (licznikPrzeciwnika == warunekWygranej) {
            return -100;  //gracz wygrywa
        }
    }

    //sprawźd kolumny
    for (size_t j = 0; j < rozmiar; j++) {
        int licznikAI = 0;
        int licznikPrzeciwnika = 0;
        for (size_t i = 0; i < rozmiar; i++) {
            if (plansza[i][j] == znak) {
                licznikAI++;
            }
            else if (plansza[i][j] == znakPrzeciwnika) {
                licznikPrzeciwnika++;
            }
        }
        if (licznikAI == warunekWygranej) {
            return 100;  //AI wygrywa
        }
```

Rysunek 1 Kawałek kodu zawierający wspomniane heurystyczne funkcje oceniające mające na celu przyśpieszenie podejmowania decyzji

Do reprezentacji graficznej gry użyłem biblioteki SFML. Wybrałem ją ze względu na jej prostotę jako że jest to jedna z najbardziej podstawowych metod reprezentacji graficznej a także przez możliwość dynamicznego linkowania. SFML pozwoliło na stworzenie ładnego interfejsu użytkownika, w tym rysowanie planszy, znaków oraz komunikatów o wygranej czy remisie a także możliwość grania dalej bądź przerwania gry guzikami.



Rysunek 2 Interaktywny interfejs graficzny przygotowany w ramach gry kółko i krzyżyk biblioteką SFML

3. Wnioski

Algorytm minimax jest powszechnie głównie stosowany w grach dwuosobowych, takich jak „Kółko i Krzyżyk", jednak jego wysokie zapotrzebowanie na zasoby komputera przy dużych planszach powodowało problemy z płynnością gry nawet przy dobrym sprzęcie. W celu poprawy wydajności zastosowałem faktycznie skutecznie działające cięcia alfa-beta, które zredukowały liczbę przeszukiwanych węzłów i w najlepszym przypadku zmniejszają one złożoność czasową nawet do $O(b^{d/2})$. Dodatkowo, wprowadziliśmy funkcję oceny gry, która przyspiesza podejmowanie decyzji przez AI, oceniając stan planszy bez pełnego przeszukiwania drzewa możliwości zachowując podejmowanie najlepszej decyzji ruchu z możliwych maksymalizując jak to tylko możliwe szanse na wygraną. Użycie biblioteki SFML z kolei pozwoliło na łatwe i efektywne stworzenie atrakcyjnej graficznie wersji gry, co znacznie ułatwiło implementację i sam efekt końcowy dzięki swojemu interfejsowi. W efekcie, kombinacja tych technik umożliwiła stworzenie płynnie działającej i efektywnej gry Kółko i Krzyżyk.

4. Bibliografia

- Projektowanie i analiza algorytmów – Wykład/Wykłady/PAA_wyklad8i9.pdf
- https://www.geeksforgeeks.org/tic-tac-toe-game-in-cpp/
- https://www.simplilearn.com/tutorials/cpp-tutorial/game-in-cpp