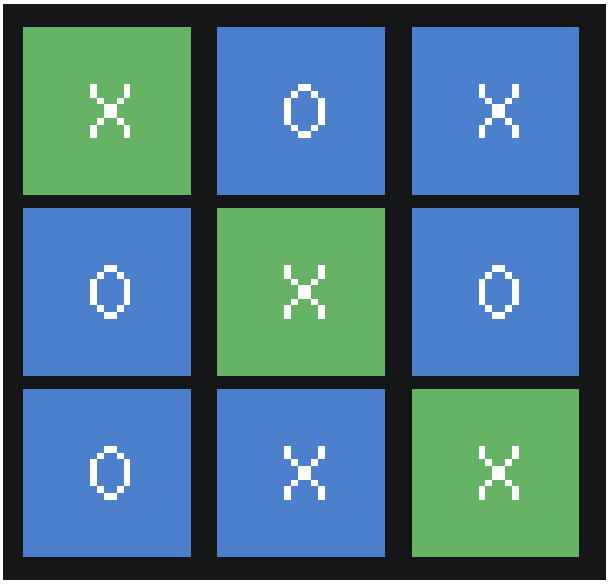
Projektowanie i analiza algorytmów	
Kierunek *Informatyczne Systemy Automatyki*	**Termin** *czwartek 11:15*
Imię, nazwisko, numer albumu *Iwo Chwiszczuk 280043*	**Data** *24 czerwca 2025*
Link do projektu `https://github.com/iwonieevo/tic-tac-toe`	

Raport

Kółko i krzyżyk z przeciwnikiem SI



Spis treści

1 Wprowadzenie

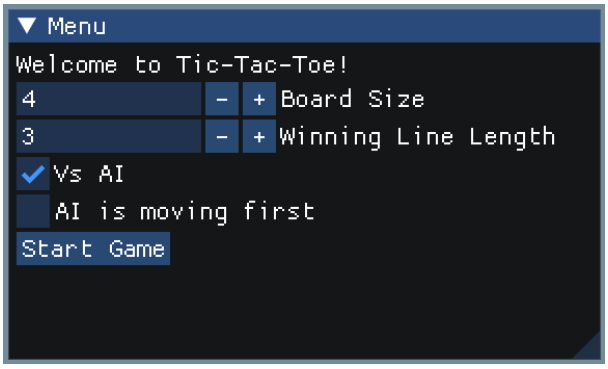
Celem projektu było stworzenie gry „Kółko i krzyżyk” (ang. *tic-tac-toe*) z komputerowym przeciwnikiem wykorzystującym algorytm minimax do podejmowania optymalnych decyzji. Gra została zaimplementowana w języku C++, z wykorzystaniem biblioteki ImGui do obsługi interfejsu graficznego.
Algorytm minimax jest klasycznym podejściem w teorii gier, stosowanym do wybierania najlepszego ruchu w grach dwuosobowych o zerowej sumie (w tym kółko i krzyżyk). Dzięki rekurencyjnemu przeszukiwaniu drzewa gry, minimax pozwala komputerowi na wybór ruchu minimalizującego maksymalną stratę lub też maksymalizującego minimalny zysk (stąd też nazwa).

2 Opis gry i zastosowanej techniki SI

2.1 Konfiguracja rozgrywki

Przed rozpoczęciem rozgrywki użytkownik może ustalić następujące parametry:

- **Rozmiar planszy** – kwadratowa plansza o boku od 3 w górę (np. 3×3, 4×4),
- **Długość zwycięskiej linii** – liczba symboli w linii potrzebna do wygranej (wartość z przedziału od 2 do rozmiaru planszy),
- **Tryb gry** – rozgrywka przeciwko SI lub dwóch graczy (naprzemienne tury),
- **Pierwszy ruch SI** – opcja dostępna tylko w trybie gry przeciwko SI, pozwala określić, czy komputer ma wykonać pierwszy ruch.



2.2 Zasady gry

Gra toczy się na kwadratowej planszy o wybranym rozmiarze. Zaczynający gracz (użytkownik lub SI) używa `X` jako swojego symbolu, a drugi korzysta z `O`. Wygrywa ten, kto pierwszy ustawi zadaną liczbę swoich znaków w linii (poziomo, pionowo lub na ukos).

2.3 Ruchy SI

Ruchy SI oparte są o algorytm minimax z przycinaniem alfa-beta oraz następującymi optymalizacjami:

- **Przetwarzanie równoległe** - każdy możliwy ruch jest analizowany w osobnym wątku przy użyciu `std::async`, co znacząco przyspiesza obliczenia dla większych plansz.
- **Dynamiczna głębokość przeszukiwania** - funkcja `calculateMaxDepth` dostosowuje głębokość rekurencji w zależności od stosunku wolnych pól do całkowitej liczby pól.
- **Heurystyczna funkcja oceny** - gdy algorytm nie może dotrzeć do stanu końcowego (wygrana/remis/przegrana), wykorzystuje funkcję `evaluateBoard`, która:
  - analizuje wszystkie linie (wiersze, kolumny, przekątne)
  - przyznaje punkty za prawie ukończone linie

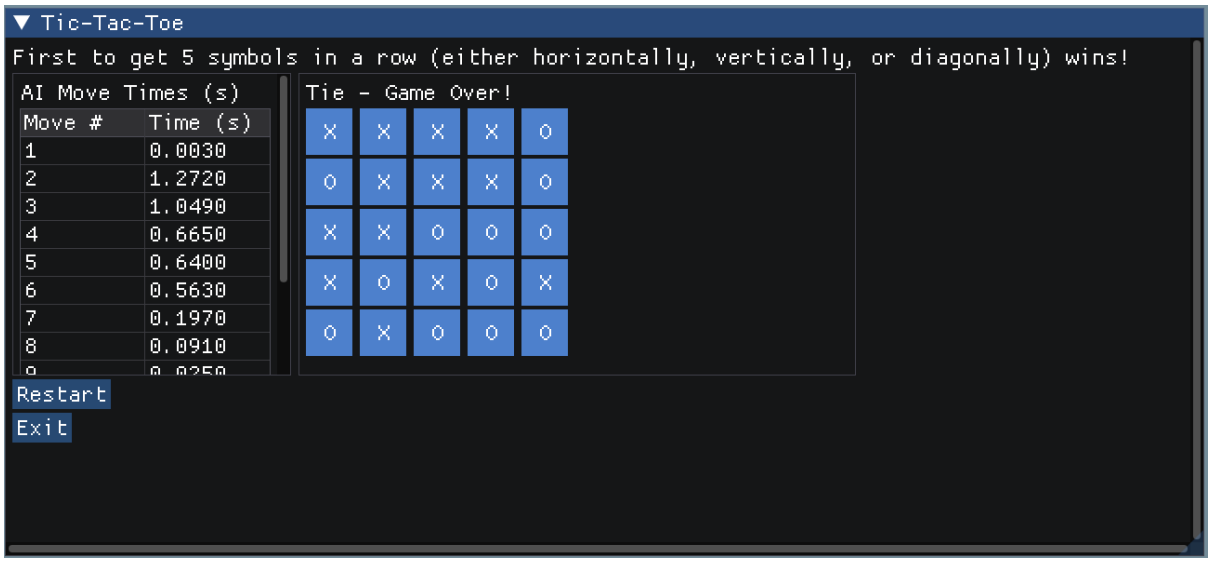
  - karze za groźne układy przeciwnika
  - wartościuje poszczególne sytuacje:
    * +1000/-1000 dla linii o 1 ruch od zwycięstwa
    * +100/-100 dla linii o 2 ruchy od zwycięstwa
    * +n/-n dla linii z n znacznikami (oraz brakiem znaczników gracza przeciwnego)

- **Optymalizacja pamięci** - każdy wątek tworzy swoją kopię planszy (`Board* board_copy`), co pozwala na bezpieczną równoległą analizę bez konieczności stosowania blokad.

Algorytm zawsze wybiera ruch o najwyższej ocenie, preferując:

- szybkie zwycięstwa (najwyższy priorytet)
- blokowanie zwycięskich linii przeciwnika
- budowanie własnych linii
- ruchy prowadzące do głębszych kombinacji (uwzględniane poprzez redukcję wartości wraz z głębokością rekurencji)

Dla standardowej planszy 3×3 algorytm jest w stanie znaleźć optymalne rozwiązanie w czasie rzeczywistym. Dla większych plansz (np. 4×4, 5×5) zastosowane optymalizacje pozwalają zachować rozsądny czas odpowiedzi przy zachowaniu wysokiego poziomu gry SI.



3 Podsumowanie i wnioski

Zaimplementowany algorytm minimax z przycinaniem alfa-beta oraz dodatkowymi optymalizacjami pozwolił na stworzenie silnego przeciwnika SI w grze kółko i krzyżyk. Główne osiągnięcia projektu obejmują:

- Poprawną implementację algorytmu dla różnych rozmiarów planszy
- Skuteczną funkcję oceny heurystycznej
- Równoległe przetwarzanie możliwych ruchów
- Dynamiczne dostosowywanie głębokości przeszukiwania

3.1 Wnioski

Podczas realizacji projektu zaobserwowano następujące istotne spostrzeżenia:

- **Kwadratowy wzrost złożoności**:
  - Liczba możliwych stanów rośnie wykładniczo wraz z rozmiarem planszy
  - Zastosowane optymalizacje (przycinanie alfa-beta, równoległość) znacząco redukują czas obliczeń
  - Mimo optymalizacji, fundamentalne ograniczenia algorytmu powodują odczuwalny wzrost czasu dla większych plansz
- **Możliwe kierunki rozwoju**:
  - Poprawa jakości wykorzystanej heurystyki oraz implementacja dodatkowych heurystyk dla wcześniejszego odcięcia mało obiecujących gałęzi
  - Wykorzystanie technik machine learning do przyśpieszenia oceny pozycji
  - Dalsze optymalizacje związane z bardziej złożonym obliczaniem maksymalnej głębokości
- SI ma trudności, żeby wygrać, jednak nigdy nie przegrywa

Podsumowując, projekt pokazał zarówno siłę algorytmu minimax w grach strategicznych, jak i jego fundamentalne ograniczenia przy zwiększaniu rozmiaru problemu. Zastosowane optymalizacje pozwoliły na praktyczne wykorzystanie algorytmu dla umiarkowanie dużych plansz, jednak dla większych rozmiarów konieczne byłyby bardziej zaawansowane techniki lub kompromisy w jakości obliczeń.

Źródła

[1] M.T. Goodrich, R. Tamassia, D. Mount, *Data Structures and Algorithms in C++*, 2nd ed., John Wiley & Sons, Inc., Hoboken, NJ, USA, 2011

[2] Russell, S., Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.

[3] Wikipedia, *Algorytm min-max*, dostęp: 24 czerwca 2025, `https://pl.wikipedia.org/wiki/Algorytm_min-max`

[4] Ocornut. (2014-2023). *Dear ImGui: Bloat-free Immediate Mode Graphical User interface for C++ with minimal dependencies* [Software]. GitHub. `https://github.com/ocornut/imgui` **Version:** v1.91.9b