Bartosz Średziński 18.01.2022

Dokumentacja

Kółko i krzyżyk z algorytmem minimax

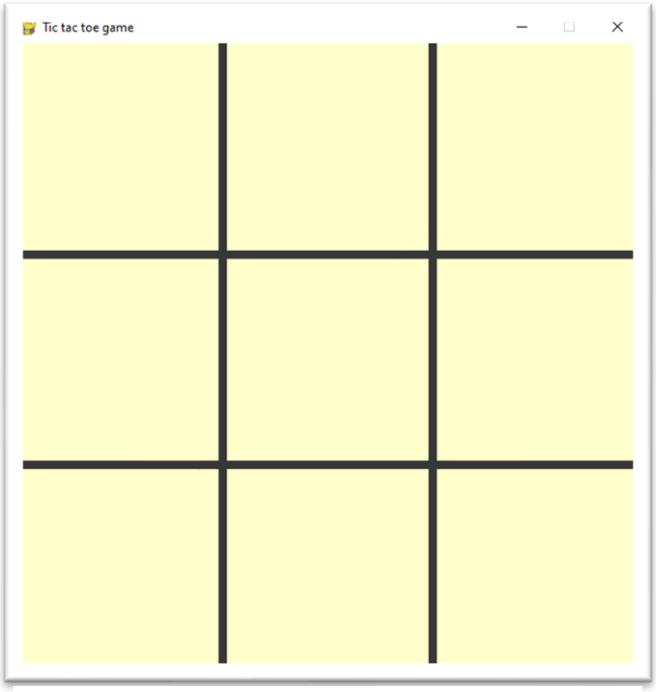
Część użytkownika

1. Uruchomienie

Program do uruchomienia potrzebuje pythona 3 (program napisany w pythonie 3.9), oraz biblioteki pygame.

2. Sposób działania

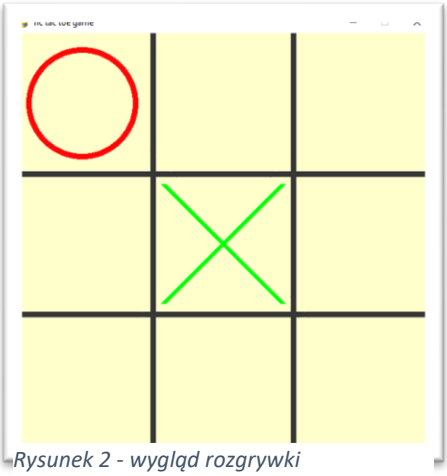
Po uruchomieniu programu pokaże się nam plansza gry.



Rysunek 1 - plansza gry

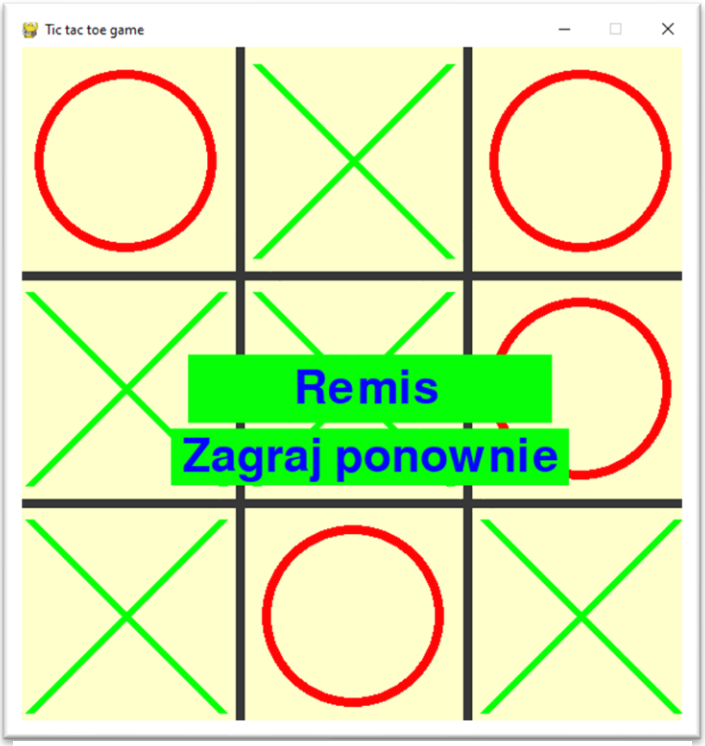
Zgodnie z zasadami gry w kółko i krzyżyk, gracz może wybrać jedno z pustych pól i w ten sposób klikając na danym polu postawi swój znak. Po wykonanym ruchu gracza, ruch wykonuje gracz komputerowy, który postawi symbol „O”.

W grze zawsze zaczyna użytkownik i stawia on „X”. Gra obsługuje wszystkie przypadki zwycięstw i remisów.



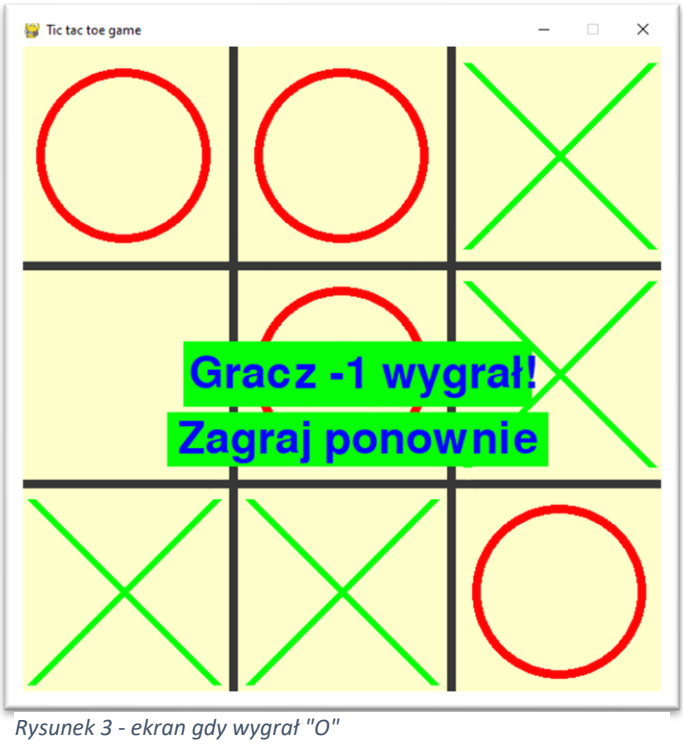
Rysunek 2 - wygląd rozgrywki

Po jednym z warunków zakończenia rozgrywki(remis/zwycięstwo któregoś z graczy), wyświetla się odpowiedni komunikat, a także przycisk, który resetuje rozgrywkę.



Rysunek 5 - ekran remis

Rysunek 4 - ekran gdy wygrał "X"



Rysunek 3 - ekran gdy wygrał "O"

Część techniczna

Kod programu został podzielony na 3 pliki:

main.py – plik zawiera funkcje i zmienne odpowiedzialne za wyświetlanie grafiki oraz główną pętle gry,

game_rules.py – plik zawiera funkcje odpowiedzialne za rozpoznawanie stanów końcowych gry

minima.py – plik zawiera implementacje algorytmu minimax na potrzeby gry kółko i krzyżyk

Stam gry jest trzymany w tablicy dwu wymiarowej. „X” jest reprezentowany przez 1, a „O” przez -1.

Najważniejsza część programu to pętla główna gry, która znajduje się na dole pliku main.py. Odpowiedzialna jest ona za działanie programu. Na początku pętli rysuje ona mapę oraz „X” i „O”. Następnie umożliwia wykonanie ruchu odpowiedniemu graczowi. Po wykonaniu ruchu w sprawdzane są warunki końcowe gry. Jeżeli któryś z warunków jest spełniony, wyświetlony zostanie ekran z odpowiednim komunikatem.

Opis wszystkich funkcji:

main.py:

draw_grid() – funkcja ustawia kolor tła planszy, oraz rysuje siatkę. Rozmiar ekranu jest ustawiony na 600x600 i funkcja jest pod te wymiary dostosowana.

draw_markers() – funkcja sprawdza tablice z stanem gry, jeżeli znajdują się w niej wykonane ruchy graczy, to funkcja odpowiednio rysuje na ekranie „X” lub „O”.

player_events(e) – funkcja obsługuje zdarzenia gracza. Jako parametr pobiera event z pętli głównej. Funkcja umożliwia zamknięcie programu oraz rysowanie „X” na planszy.

draw_winner() – funkcja na podstawie zmiennej „winner” ustawia odpowiedni komunikat o zakończeniu gry. Następnie rysuje ona komunikat na ekranie.

show_game_over_screen() – funkcja wywołuje draw_winner() i zapewnia możliwość rozpoczęcia gry na nowo.

make_best_move_enemy() – funkcja która wykonuje ruch przeciwnika. W pętli wykonuje wszystkie możliwe ruchy, dla każdego z nich oblicza przy pomocy algorytmu minima opłacalność. Po wykonaniu wszystkich możliwości, wybiera ona najlepszy ruch i go wykonuje.

game_rules.py:

is_tie(board) – funkcja sprawdza czy w podanej tablicy występuje remis(wszystkie pola zajęte). Zwraca True jeżeli remis.

check_winner(board) – sprawdza warunki zwycięstwa, dla gry kółko i krzyżyk, gdzie jeden gracz oznacza 1 a drugi -1. Sprawdza kolumny, wiersze oraz przekątne planszy. Zwraca numer gracza który wygrał lub 0 jeżeli nie ma żadnego zwycięstwa.

minimax.py:

minimax(depth, is_max, board) – funkcja która implementuje algorytm minima. Jako parametry przyjmuje, depth – głębokość drzewa (ile ruchów symulujemy), is_max – czy minimalizować wynik, board – tablica gry.

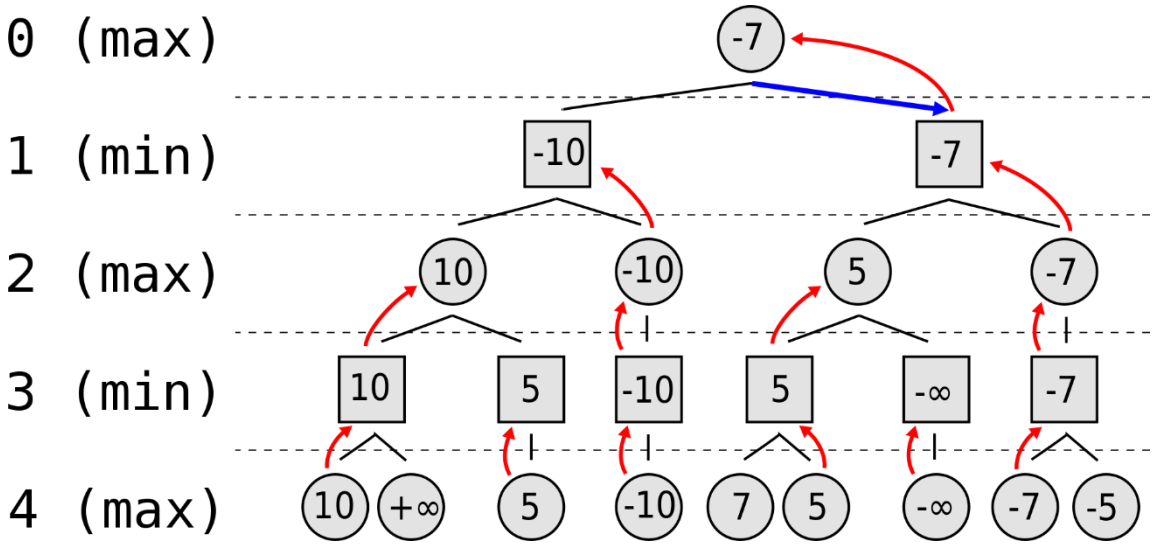
Algorytm min-max:

Funkcja rekurencyjnie tworzy drzewo stanów gry do głębokości depth lub stanu końcowego gry. Na przemiennie wykonuje ruchy jako gracz „X” oraz „O”. Algorytm maksymalizuje zysk i minimalizuje straty. Gałąź tworzonego drzewa może się skończyć wcześniej, gdy wystąpi warunek zwycięstwa. Algorytm odpowiednio nagradza za dane ruchy. Po dojściu do końca gałęzi, algorytm wraca i podlicza gałąź. Za wygranie gry daje 10 punktów, za przegranie daje -10 punktów, za remis lub dojście do depth daje 0 punktów. Dodatkowo algorytm uwzględnia głębokość drzewa dodając lub odejmując punkty. Za początkowe wartości stanu gry przyjmuje się $+\infty$, $-\infty$ lub też odpowiednio duże i małe wartości. Po wykonaniu algorytmu, otrzymujemy opłacalność dla każdego możliwego ruchu i wybieramy najlepszy(największa wartość).

Wadą algorytmu jest koszt utworzenia całego drzewa gry, szczególnie widać to przy większych grach niż kółko i krzyżyk. Przy pierwszym ruchu algorytm musi wygenerować drzewo o ilości węzłów większej niż 100’000. W praktyce widać, że im mniej pustych pól jest, tym algorytm działa szybciej. Algorytm można zoptymalizować np. poprzez ograniczenia głębokości drzewa(kosztem skuteczności), lub poprzez zastosowanie innych

bardziej optymalnych algorytmów jak alfa-beta(ulepszony algorytm min-max z odcinaniem nieopłacalnych gałęzi).

Przykładowe drzewo utworzone przez algorytm min-max:



Rysunek 6 - przykładowe drzewo gry utworzone przez algorytm

Źródła

https://en.wikipedia.org/wiki/Minimax

https://swistak.codes/algorytmika-gier-kolko-i-krzyzyk/

https://www.pygame.org/docs/