Data: 26.05.2020
Imię i nazwisko: Karolina Głuszek
Nr albumu: 249034
Nazwa kursu: Projektowanie algorytmów i metody sztucznej inteligencji
Dane prowadzącego: mgr Marta Emirsajłow
Termin zajęć: piątek, 9.15

1. Wprowadzenie

Napisaną przeze mnie grą jest kółko i krzyżyk dające graczowi możliwość zdefiniowania rozmiaru planszy oraz ilości znaków w rzędzie gwarantujących wygraną. Gra została zaimplementowana w języku C++ i posiada wersję graficzną korzystającą z obiektów otwartej biblioteki programistycznej SFML. Gracz ma do wyboru rozgrywkę z innym graczem oraz ze stworzoną w programie sztuczną inteligencją decydującą o swoich ruchach na podstawie algorytmu minimax z odcięciami alfa-beta.

2. Opis stworzonej gry

a) Algorytm minimax

Algorytm minimax jest wykorzystywany do podejmowania decyzji w grach dwuosobowych o sumie zerowej, czyli takich, w których zysk jednego gracza skutkuje stratą drugiego gracza. Algorytm minimax sprowadza się do przeszukiwania drzewa rozwiązań gry w taki sposób, aby naprzemiennie rozważać najlepsze ruchy dla gracza, a następnie najlepsze ruchy przeciwnika, które tym samym są najmniej korzystne dla gracza. W przypadku gry kółko i krzyżyk do określenia, które ruchy są korzystne należy zaimplementować funkcję wartościującą ułożenie znaków na planszy zwracającą tym większe dodatnie wartości, im większa szansa wygranej gracza oraz tym mniejsze ujemne wartości, im większa szansa wygranej przeciwnika. Posiadając wyniki takiej funkcji bot stara się maksymalizować wartość swoich ruchów oraz minimalizować wartość ruchów przeciwnika symulując tym samym rozgrywkę dwóch inteligentnych graczy. Opisywany algorytm jest wywoływany rekurencyjnie do pewnej głębokości oznaczającej ile ruchów wprzód bot jest w stanie przewidzieć. W programie głębokość wywołania rekurencji została ograniczona w zależności od wybranej wielkości planszy według eksperymentalnie wyznaczonego wzoru $d = \frac{15}{s}$, gdzie d – maksymalna głębokość, s – wielkość planszy. Wyliczana głębokość zależy odwrotnie proporcjonalnie od wielkości planszy, ponieważ zwiększanie wielkości planszy bardzo wydłuża czas przeszukiwania drzewa rozwiązań, ze względu na jego większe rozgałęzienie. Zakładając, że rozgałęzienie drzewa jest stałe i wynosi b złożoność czasowa algorytmu minimax wynosi $O(b^d)$.

b) Przycinanie alfa-beta

Wykorzystany algorytm minimax został usprawniony przez dodanie tzw. odcięć alfa-beta. Technika ta sprowadza się do przerwania przeszukiwania drzewa rozwiązań w momencie, gdy obecnie analizowany ruch okaże się być gorszy od wcześniej zbadanych ruchów. Nie ma potrzeby analizowania całego poddrzewa ruchu, który dla gracza jest niekorzystny, więc rezygnując z dalszego przeszukiwania algorytm przyspieszy swoje działanie, nie wpływając przy tym na ostatecznie wybrane rozwiązanie. Cięcia alfa oznaczają porzucienie danego

węzła, jeżeli istnieją lepsze wartości w innej części grafu dla strony maksymalizującej, a cięcia beta odrzucają węzeł jeżeli istnieją lepsze wartości w innej części grafu dla strony minimalizującej. Zakładając, rozgałęzienie drzewa jest stałe i wynosi b, a najlepsze ruchy są rozpatrywane jako pierwsze, liczba przeszukanych pozycji wyniesie $O(b * 1 * b * 1 * ... * b)$ w przypadku nieparzystej głębokości lub $O(b * 1 * b * 1 * ... * b)$ dla parzystej głębokości, co oznacza, że złożoność czasowa algorytmu minimax z odcięciami alfa-beta wyniesie $O(b^{\frac{d}{2}})$.

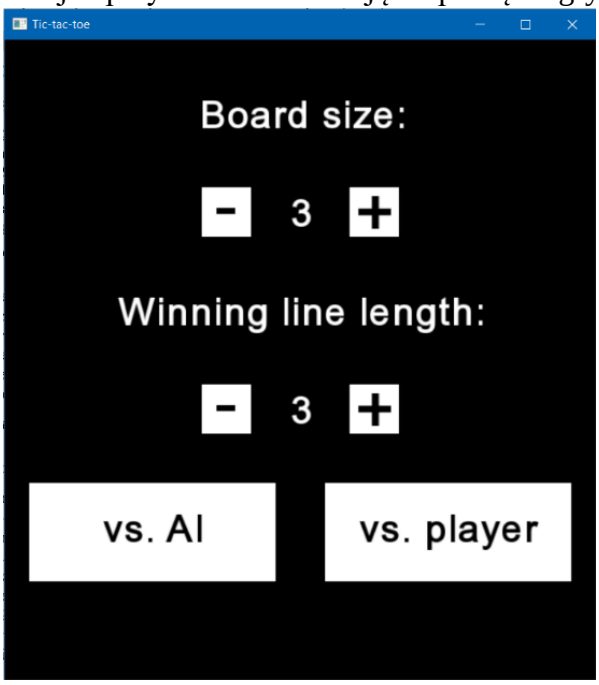
c) Funkcja heurystyczna wartościująca planszę

Jednym z najciekawszych zagadnień podczas pisania programu było wyznaczenie funkcji heurystycznej wartościującej możliwe ułożenia znaków na planszy. W tym celu na początku na całej planszy wyszukane zostają wszystkie ciągi długości wygrywającego ciągu n. Największym priorytetem bota powinno być wygranie gry. Dlatego jeśli istnieje fragment planszy, gdzie w jednej linii występuje $n - 1$ znaków bota oraz nie ma żadnego znaku gracza, takiej planszy zostaje przypisana maksymalna możliwa wartość. Aby bot blokował ruchy gracza wartość planszy jest duża wtedy, gdy gracz ma $n - 1$ swoich znaków w linii, a bot w tej samej linii ma conajmniej jeden znak. Dodatkowo im więcej znaków bota znajduje się w liniach wygrywającej długości, tym większy wynik oceny planszy jest zwracany. Dzięki temu bot preferuje stawianie znaków na samym środku planszy lub w jego okolicach, gdzie istnieje najwięcej możliwości wygrania we wszystkich kierunkach (pionowo, poziomo i na wszystkich przekątnych).

d) Interfejs graficzny

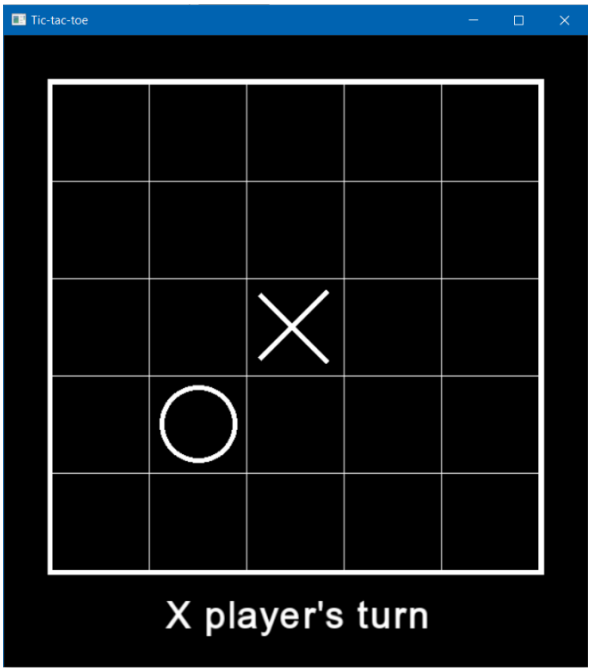
Interfejs graficzny został podzielony na trzy widoki wyświetlanie osobno w jednym oknie. W tym celu powstała abstrakcyjna klasa widoku *View* z metodami wirtualnymi oraz trzy klasy dziedziczące po niej i nadpisujące jej metody: *MenuView*, *GameView*, *ScoreView*. W danym momencie wyświetlany jest w oknie tylko jeden widok *View*, do którego przypisywane są odpowiednie obiekty klas podrzędnych w zależności od wywołanego zdarzenia.

Pierwszym widokiem obsługiwanym przez klasę *MenuView* jest widok głównego menu gry. Za pomocą czterech przycisków można wybrać dowolną wielkość planszy oraz dowolną długość wygrywającej linii pod warunkiem, że jest ona mniejsza niż wielkość planszy. Dwa kolejne przyciski umożliwiają rozpoczęcie gry odpowiednio z botem albo drugim graczem.



Widok głównego menu gry - MenuView

Następnym widokiem wyświetlanym w oknie po rozpoczęciu gry jest *GameView* wyświetlający aktualny stan planszy oraz komunikat o tym, czyjego gracza jest obecny ruch. Po zakończeniu gry program odczekuje 2 sekundy, a następnie przełącza widok na *ScoreView*.



Widok rozgrywki - GameView

Ostatnim widokiem jest *ScoreView*, który wyświetla wiadomość, gdy wygrał gracz X, O albo gra zakończyła się remisem. Dodatkowo w oknie znajduje się również przycisk, który zmienia widok spowrotem na widok głównego menu gry.

Widok wyniku gry – ScoreView

3. Podsumowanie i wnioski

Zgodnie z oczekiwaniami, im większa była głębokość rekurencyjnego wywoływania funkcji minimax, tym dłuższy był czas przeszukiwania drzewa rozwiązań. Z drugiej strony bot był wtedy w stanie lepiej reagować na zmianę sytuacji na planszy.

Dzięki zastosowaniu przemyślanej funkcji heurystycznej wartościującej sytuację na planszy głębokość rekurencyjnych wywołań algorytmu minimax nie musiała być duża, aby bot wykonywał poprawne ruchy. Nawet dla głębokości równej 1 i dużych plansz jak np. 10x10 bot poprawnie blokuje ruchy gracza. Dzięki temu gra zawsze kończy się remisem lub wygraną bota, jeżeli gracz popełni błąd.

Zaimplementowane usprawnienie algorytmu minimax w postaci odcięć alfa-beta znacznie przyspieszyło czas przeszukiwania drzewa rozwiązań przez program w przypadku niektórych sytuacji na planszy. Jednak w celu dalszej optymalizacji poszczególne węzły drzewa powinny zostać posortowane tak, aby najlepsze ruchy były rozpatrywane jako pierwsze. Wtedy wystąpiłaby maksymalna ilość cięć alfa-beta, a czasowa złożoność algorytmu wyniosłaby założone $O(b^{\frac{d}{2}})$.

4. Literatura

http://wazniak.mimuw.edu.pl/index.php?title=Sztuczna_inteligencja/SI_Modu%C5%82_8_-_Gry_dwuosobowe

http://pbeling.w8.pl/game_theory/Klewicka_Justyna_Alpha_Beta.pdf