Projekt 3 – Gry i AI

Nazwa kursu	
Projektowanie Algorytmów i Metody Sztucznej Inteligencji	
Autor	**Data**
Dominik Koperkiewicz 254023	08.06.2021
Prowadząca	**Termin zajęć**
Mgr inż. Marta Emirsajłow	Wt 15:15

Wprowadzenie

Wykonany przeze mnie projekt polegał na stworzeniu gry (Kółko i Krzyżyk) oraz sztucznej inteligencji przeciwko której będzie można zagrać.

Opis gry

Przebieg rozgrywki

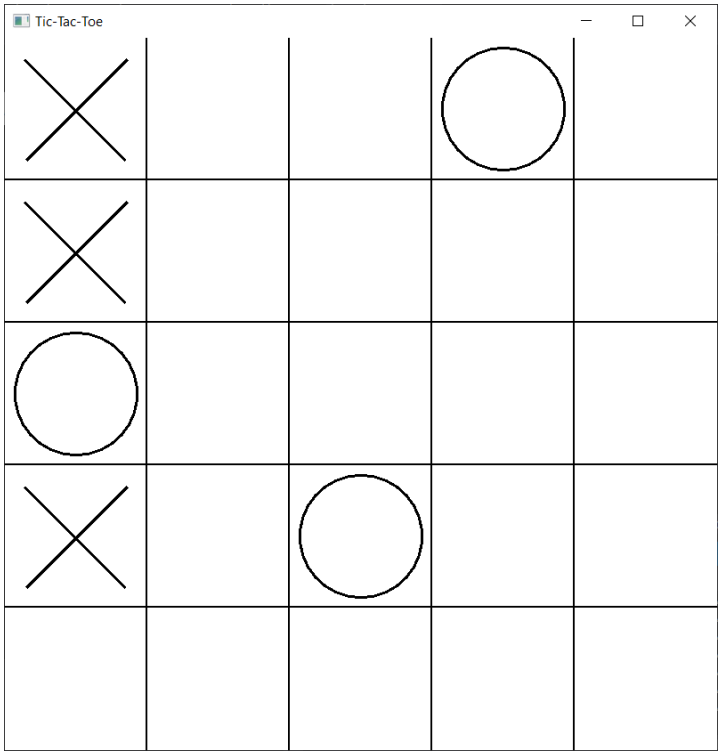
Po uruchomieniu gry, użytkownik jest pytany o rozmiar planszy, wymaganą liczbę znaków w rzędzie oraz tryb w jakim ma być uruchomiona rozgrywka:

Dostępne tryby:

- Gracz vs Gracz
- Gracz vs SI
- SI vs Gracz
- SI vs SI

Następnie w oknie zostaje wyrysowana plansza, a gracz wykonuje ruchy za pomocą lewego przycisku myszy klikając w odpowiednie pola.

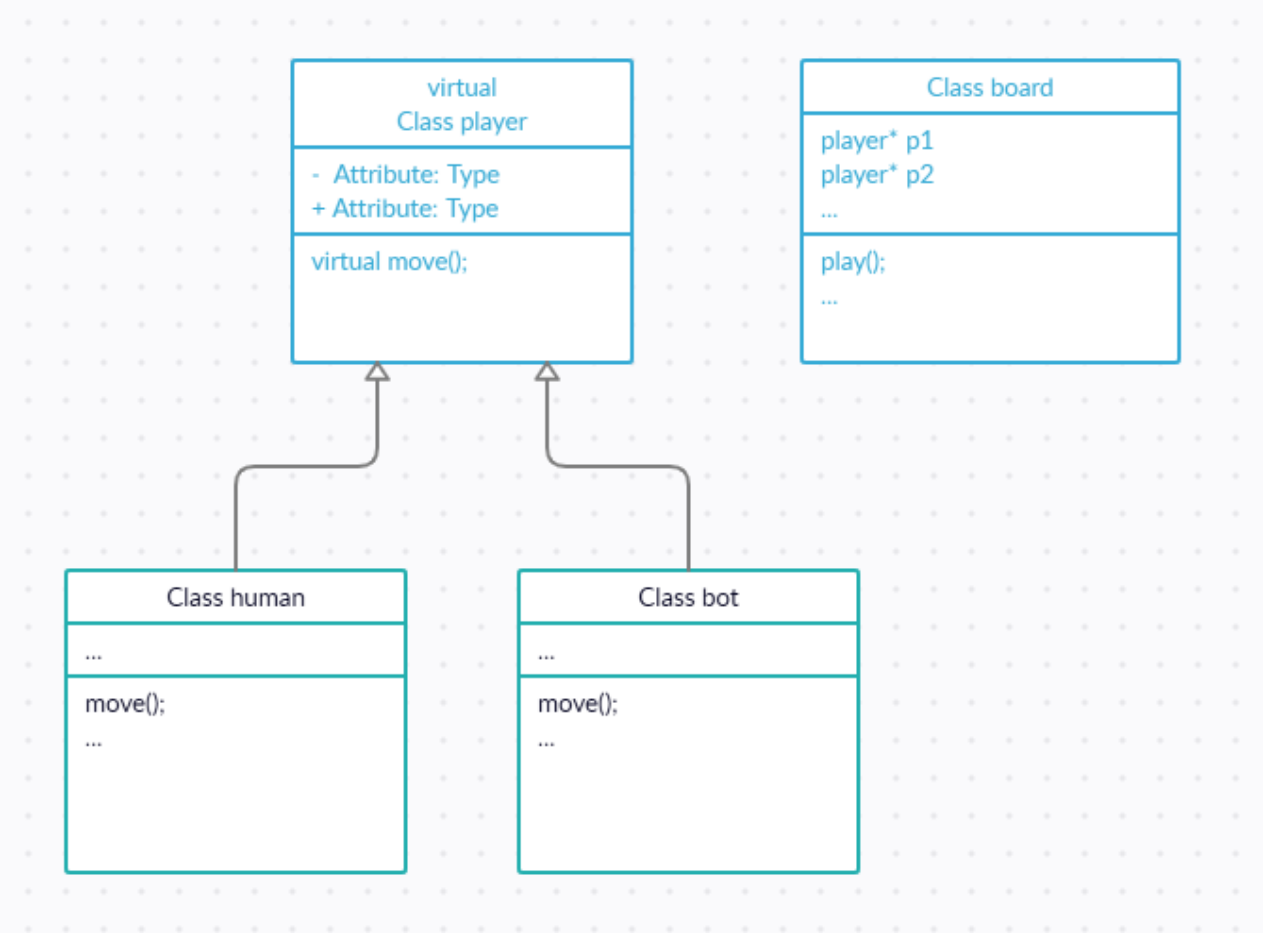
W przypadku zwycięstwa którejś ze stron w terminalu wypisany zostanie zwycięzca, a użytkownik zostanie poproszony o dane do kolejnej rozgrywki.



Budowa programu

Program został napisany w języku C++, a do obsługi grafiki posłużono się biblioteką **SFML.**

Na poniższym schemacie UML przedstawione są klasy zawarte w programie:



Za rozgrywkę głównie odpowiedzialna jest klasa **board**, to w niej przechowywany jest aktualny stan gry, sprawdzane są warunki zwycięstwa oraz wywoływane są ruchy graczy. Obiekty klasy **player** przekazują informację o tym jaki ruch chcą wykonać.

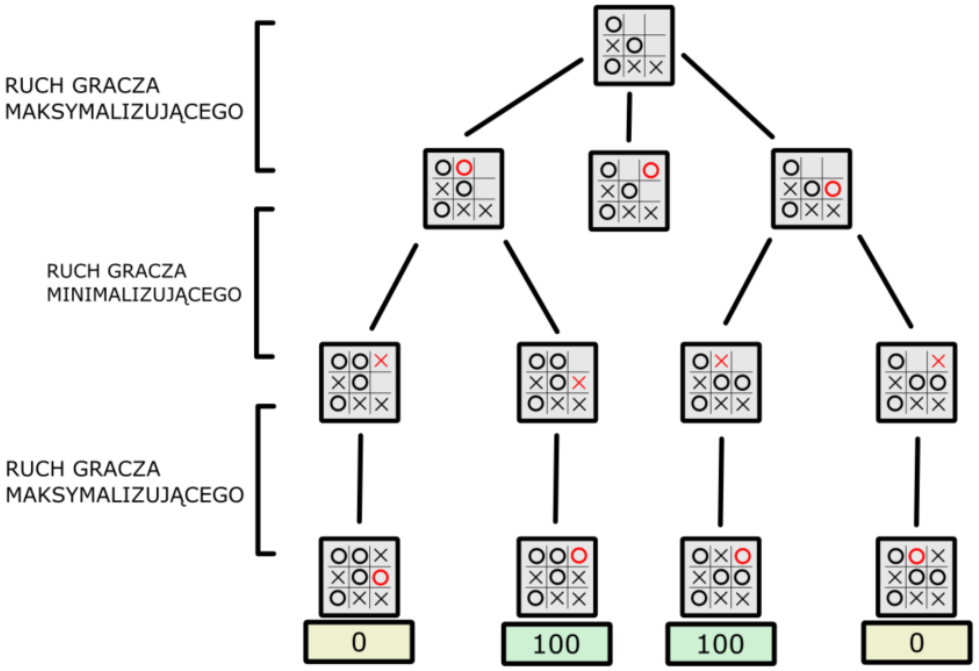
Zastosowane techniki SI

Klasa **bot** do podejmowania decyzji o tym jaki ruch wykona komputer stosuje algorytm *mini-max*. Algorytm za pomocą rekurencji sprawdza wszystkie możliwe ruchy do wykonania przez najbliższe kilka tur i przyznaje każdej możliwości ocenę mówiącą o tym jak korzystna będzie dla gracza dana sytuacja. Algorytm zakłada że przeciwnik będzie podejmował optymalne (według algorytmu) decyzję.

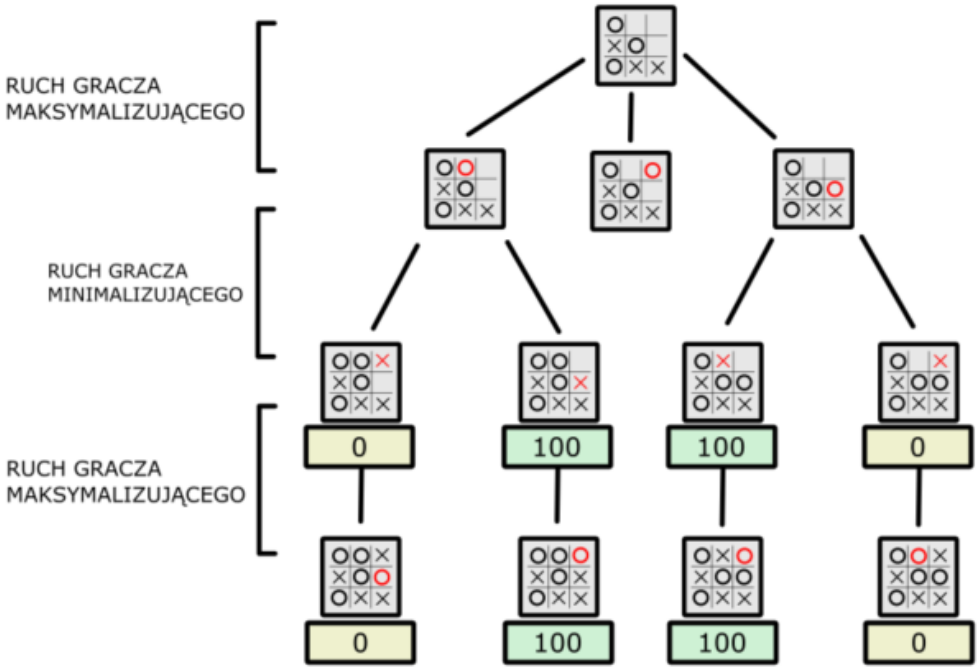
Funkcja wyznaczająca wartość danej sytuacji bierze pod uwagę jak wiele i jak długich rzędów posiadają gracze oraz przede wszystkim to czy któryś z graczy wygrywa w sprawdzanym przypadku.

Taki algorytm można przedstawić w formie drzewa:

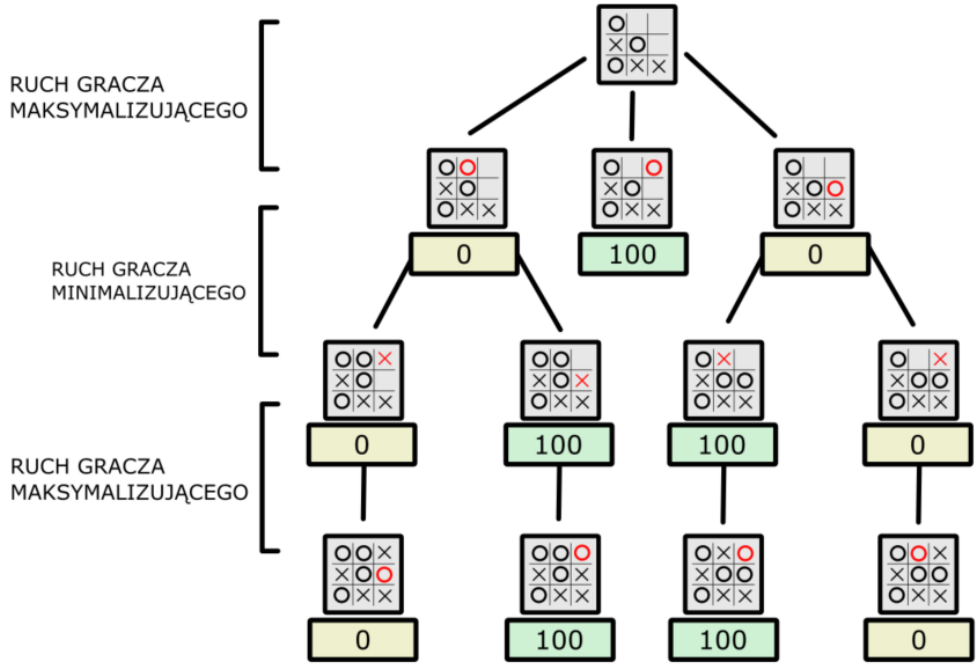
Rozrysowane zostały możliwości potoczenia się gry, a wszystkie ostatnie ruchy mają przypisany wynik dla gracza ‘O’



Ostatni ruch należy do gracza ‘O’ dlatego też wybiera on ruchy o najwyższym wyniku w danej gałęzi i przekazuje je wyżej (w tym przypadku ma tylko po jednym możliwym ruchu w każdej gałęzi).



Teraz sprawdzane są oceny przy ruchu gracza ‘X’. Jest to gracz minimalizujący, co oznacza że chce aby wynik gracza ‘O’ był jak najniższy, dlatego wybiera najniższe wartości i przekazuje je wyżej.



Pierwszy ruch należy do gracza maksymalizującego, więc wybiera on teraz ruch o najwyższym wyniku, czyli postawienie koła w prawym górnym rogu.

W podanym przykładzie wszystkie gałęzie kończyły się zwycięstwem gracza ‘O’ lub remisem. W przypadku gdyby ruch kończył się zwycięstwem gracza ‘X’ to przypisana by została wartość -100, natomiast gdyby po zejściu na daną głębokość gra nadal nie była rozstrzygnięta, to punktacja była by przypisana na podstawie rozłożenia znaków obu graczy, próbując oszacować czyja pozycja jest lepsza.

Podsumowanie i wnioski

Algorytmy *mini-max* są bardzo skuteczne, szczególnie w grach w których możemy łatwo określić który z graczy ma przewagę. Natomiast ich ogromną wadą jest złożoność obliczeniowa, ponieważ sprawdzanie wszystkich możliwych ruchów wymaga wielu powtórzeń algorytmu, co sprawia że taki algorytm nie byłby odpowiedni w przypadku gier o bardzo złożonych zasadach.

Poprzez powiększanie rozmiaru planszy bardzo szybko można zauważyć że czas potrzebny komputerowi na znalezienie odpowiedniego ruchu wyraźnie się zwiększa.

Literatura

- https://www.youtube.com/watch?v=trKjYdBASyQ
- https://www.sfml-dev.org/tutorials/2.5/
- https://pl.wikipedia.org/wiki/Algorytm_min-max