Projektowanie Algorytmów i Metod Sztucznej Inteligencji

Projekt 3: Kółko i Krzyżyk & AI

Maciej Gaik
241542

Grupa: środa 15:15
Prowadzący: Dr inż. Artur Rusiecki

1. Wprowadzenie

Algorytm MinMax wywodzi się z twierdzenia o grze o sumie stałej. Oznacza to, że jeśli dwie osoby grają przeciwko sobie, to poprawa sytuacji jednego z graczy oznacza proporcjonalne pogorszenie się sytuacji gracza drugiego. Dzięki tej informacji można przy użyciu odpowiedniej funkcji heurystycznej przypisać danej sytuacji na planszy konkretną wartość (dodatnią, jeśli jest korzystna dla gracza, dla którego ją rozpatrujemy lub ujemną w przeciwnym wypadku).

2. Opis tworzonej gry

2.1. Zasady ogólne

Klasyczna gra w „kółko i krzyżyk" posiada 9 pól (3x3), w które gracze na przemian wstawiają X lub O. Wygrywa ten gracz, który jako pierwszy uzyska sekwencje wygrywająca, czyli układ 3 swoich znaków w jednej linii (w tym przekątne). W swojej wersji gry daję możliwość zdecydowania z ilu pól ma się składać plansza (zawsze kwadrat) oraz jaka ilość znaków gracza w rzędzie ma wygrywać.

2.2. Budowa programu

Cała gra dzieli się na 4 pliki, w których korzystamy z 2 stworzonych klas: 'game' oraz 'window'. Klasa 'game' odpowida za mechanikę gry, funkcje związane z AI, sprawdzanie wolnych pól, czy sprawdzanie czy żądany ruch jest możliwy. Klasa 'window" otpowiada natomiast za interfejs graficzny gry, tworzenie okna, rysowanie planszy, pól oraz znaków X i O.

Pliki:

- main.py – główna pętla programu oraz definiowanie rozmiaru gry
- events.py – sprawdzanie wydarzeń np. kliknięcia przycisku myszy czy chęci opuszczenia rozgrywki
- game.py – zawiera klasę 'game' oraz metody składowe
- window.py – zawiera klasę 'window' oraz metody składowe

2.3. Oprawa graficzna

Do wersji graficznej gry wykorzystałem bibliotekę PyGame dla języka Python. Pozwala ona w łatwy i szybki sposób skonstruować oprawę graficznej tworzonej gry. Mechanika gry opiera się na tablicy o rozmiarze odpowiadającym rozmiarowy gry. Jeżeli w element tablicy zawiera 0 to znaczy, że pole gry jest puste, 1 komputer postawił swój znak oraz -1 gracz postawił swój znak. Odpowiednia funkcja sprawdza cały czas tę tablicę i jeżeli zaszła zmiana, rysuje odpowiedni znak w wyliczonym miejscu. Inna funkcja natomiast, sprawdza czy na planszy znajdę się sekwencja wygrywająca. Funkcja ta pobiera z generowanej na początku działa gry tablicy, współrzędne pól na, których musi znaleźć się znak gracza, żeby wygrał. Jeżeli wymagana ilość znaków znajduje się na wyliczonych kolejnych polach, gracz wygrał

2.4. AI

Algorytm MinMax jest metodą wybierania ruchu, który pozwala na minimalizację możliwych strat lub maksymalizację zysków. Jeśli dla danej sytuacji na planszy stworzymy drzewo wszystkich możliwych układów dla n-następnych ruchów i każdej przypiszemy wartość przy użyciu określonej funkcji oceniającej kto wygrał, to możemy wybrać taki ruch, który daje nam największe korzyści. Gracz, dla którego wywołany jest w/w algorytm będzie dążył do maksymalizacji zysków, natomiast przeciwnik będzie się starał te zyski zminimalizować. Algorytm MinMax dostaje jako aktualny stan gry, głębokość drzewa oraz znak gracza, dla którego przypada ruch. Następnie sprawdza, czy aktualny gracz dąży do maksymalizacji, czy do minimalizacji zysków i zależnie od tego wybiera ruch o największej / najmniejszej wartości. Aby to uzyskać, wywołuje się rekurencyjnie aż do osiągnięcia warunku podstawowego, dzięki któremu może zwrócić konkretną wartość, w tym wypadku element tablicy w którym należy postawić znak komputera.

2.5. Cięcia Alfa - Beta

Cięcia α, β nie są osobnym algorytmem, a modyfikacją do algorytmu MinMax. Zakładają one dodanie dwóch zmiennych, które przechowują minimalną β i maksymalną α wartość, jakie MinMax obecnie może zapewnić na danej głębokości drzewa. Przy starcie algorytmu $\alpha = -\infty$ i $\beta = \infty$. Wartości te są korygowane w dalszych etapach działania algorytmu. Na ich podstawie, jeśli przy sprawdzaniu kolejnego poddrzewa węzła minimalizującego α-β algorytm może odciąć dane poddrzewo (w tym przypadku cięcieβ), ponieważ już wie, że maksymalna wartość te-go poddrzewa przekroczy minimalną, którą węzeł może obecnie zagwarantować. Bardzo korzystnie wpływa to na złożoność algorytmu, natomiast nie gwarantuje sukcesu.

3. Wnioski

Algorytm radzi sobie bardzo dobrze w klasycznej wersji gry dla głębokości drzewa równej ilości wolnych pól. Ruch wykonuje niemal natychmiastowo, a zadanie gracza sprowadza się do tego, żeby nie przegrać, ponieważ algorytm skutecznie blokuje ruchy, ale nie zmarnuje żadnej okazji na wygraną. W wersji 4x4 i głębokości jak poprzednio, czas oczekiwania na pierwszy ruch komputera to parę sekund i z każdym ruchem ten czas maleje. Przy takiej głębokości drzewa algorytm nie dąży do wygranej, wstawia znak w kolejne wolne pola, jednak skutecznie blokuje każdą możliwą wygraną gracza. Dla wersji 5x5 i głębokości 8 odpowiedź komputera zajmuje do 2 minut, ale zdarzały się też dłuższe czasy. W przypadku wybrania mniejszej liczby znaków wygrywających niż rozmiar planszy, czasy te wydłużają się. Testowany był również, algorytm MinMax bez cięć α, β. Rezultatem było znaczne spowolnienie rozgrywki przy rozmiarach większych niż 3x3. Odpowiedź komputera zajmowała na kilkanaście lub kilkadziesiąt minut. Jedną z prawdopodobnych przyczyną może być sam język Python,

w których gra została napisana. Język ten jest, według różnych źródeł, ponad 3 razy wolniejszy od C++. Kolejnym powodem może być zastosowanie biblioteki Pygame, która zawiera wiele funkcji i możliwości do zrealizowania gier, jednak znacząco obciąża komputer.

4. Bibliografia

https://en.wikipedia.org/wiki/Minimax
https://github.com/Cledersonbc/tic-tac-toe-minimax
https://eduinf.waw.pl/inf/utils/001_2008/0415.php