SPRAWOZDANIE

PROJEKT 3

PROJEKTOWANIE ALGORYTMÓW I METOD SZTUCZNEJ INTELIGENCJI

AUTOR: EMILIA SZYMAŃSKA

PROWADZĄCY: DR INŻ. KRZYSZTOF HALAWA

GRUPA: WTOREK 13:15 – 15:00

1. Wstęp

   Celem projektu jest implementacja gry ,,Kółko i krzyżyk” wraz z algorytmem sztucznej inteligencji. Po uruchomieniu gry najpierw podajemy rozmiar planszy (która jest kwadratowa), ilość znaków w rzędzie/kolumnie/na przekątnej, która powoduje wygraną, znak gracza (użytkownika), znak przeciwnika (algorytmu) oraz wybieramy, kto zaczyna rozgrywkę. Plansza wyświetlana jest w konsoli, wpisując współrzędne pola użytkownik może wykonać swój ruch.

2. Omówienie gry oraz algorytmu

   Gra rozpoczyna się po podaniu wyżej wspomnianych danych wejściowych. Warto wspomnieć, że z punktu widzenia programu nie ma znaczenia to, czy znaki gracza i przeciwnika są takie same, gdyż macierz reprezentująca w kodzie planszę wypełniana jest ID odpowiednio gracza lub komputera, a nie znakami, jednak z punktu widzenia użytkownika gra stałaby się nieczytelna. Rozmiar planszy nie może być mniejszy niż 3, wygrywająca ilość znaków mniejsza niż 2 (choć sama wartość 2 nie ma zbyt wielkiego sensu, gdyż osoba zaczynająca prawie na pewno wygra). Warunki te są sprawdzane przy wczytywaniu danych. Przy wyborze zaczynającego rozgrywkę użytkownik musi wpisać wartość 1 lub 2 odpowiadające wskazaniu użytkownika lub komputera jako osoby zaczynającej, przy innych wartościach pojawia się komunikat o błędnym wejściu. Przy podawaniu współrzędnych pola także jest sprawdzany warunek, czy pole o takich współrzędnych istnieje i czy nie jest już zajęte.
   Komputer gra przy użyciu algorytmu MINMAX. Algorytm ten działa na zasadzie analizowania wszystkich możliwości wykonania ruchu gracza MAX, którego wygraną zwrócilibyśmy przez wartość 1. Jeśli nie mamy wyniku w postaci wygranej/przegranej/remisu, to analizujemy dalszy ruch, lecz tym razem gracza MIN, którego wygrana jest oznaczana przez wartość -1. Remis jest oznaczany jako 0. Jeśli nie ma wyniku, to analizujemy znowu potencjalny ruch gracza MAX. Następuje rekurencyjne wywołanie algorytmu (do określonego poziomu głębokości rekurencji), w którym zakładamy, że przeciwnik zawsze gra optymalnie. Ostatecznie algorytm zwraca nam wynik oraz ruch, który należy wykonać, by taki wynik osiągnąć. Maksymalizacja oznacza tu, że z ostatecznie zwróconej wartości bierzemy opcję z przypisaną wartością największą (bo chcemy osiągnąć wynik związany z wartością 1), minimalizacja wiąże się z wyborem opcji z wartością najmniejszą (gdyż wygrana gracza MIN wiąże się z wartością -1).
   Problemem tu jest głównie głębokość wywołania rekurencji. W przypadku planszy 3x3 nie jest on widoczny, gdyż nie ma wtedy aż tak wielu wywołań, by komputer sobie z nimi szybko nie poradził. Problem zaczyna się w przypadku planszy 4x4, gdzie bez ograniczenia głębokości komputer liczyłby rozwiązania bardzo długo. W tym celu stworzyłam tablicę maksymalnej głębokości wywołań funkcji dla plansz do rozmiaru 10x10. Powyżej tej wartości zawsze przyjmuję maksymalną głębokość jako 2. Dla $3 \leq N \leq 10$ obliczyłam maksymalne głębokości, przy których algorytm działa w rozsądnym czasie (użytkownik czeka ułamki sekundy na ruch komputera) i kod nie pozwala algorytmowi wejść głębiej.

3. Przykładowy wynik działania programu

```
emma@emma-Latitude-E6520:~/Dokumenty/TicTacToe$ ./tictactoe
Size of the board: 3
Number of elements in a row to win: 3
Type your sign: X
Type the sign of the AI: O
Who should start: you (type: 1) or AI (type 2)? 1
Type your move as: row column

   1  2  3
1  _  _  _
2  _  _  _
3  _  _  _

Your turn. Make your move: 2 2

   1  2  3
1  _  _  _
2  _  X  _
3  _  _  _

AI's turn

   1  2  3
1  O  _  _
2  _  X  _
3  _  _  _

Your turn. Make your move: 3 3

   1  2  3
1  O  _  _
2  _  X  _
3  _  _  X

AI's turn

   1  2  3
1  O  _  O
2  _  X  _
3  _  _  X

Your turn. Make your move: 1 2

   1  2  3
1  O  X  O
2  _  X  _
3  _  _  X

AI's turn

   1  2  3
1  O  X  O
2  _  X  _
3  _  O  X

Your turn. Make your move: 2 1

   1  2  3
1  O  X  O
2  X  X  _
3  _  O  X

AI's turn

   1  2  3
1  O  X  O
2  X  X  O
3  _  O  X

Your turn. Make your move: 3 1

   1  2  3
1  O  X  O
2  X  X  O
3  X  O  X

DRAW
```

```
emma@emma-Latitude-E6520:~/Dokumenty/TicTacToe$ ./tictactoe
Size of the board: 4
Number of elements in a row to win: 3
Type your sign: @
Type the sign of the AI: *
Who should start: you (type: 1) or AI (type 2)? 1
Type your move as: row column

   1  2  3  4
1  _  _  _  _
2  _  _  _  _
3  _  _  _  _
4  _  _  _  _

Your turn. Make your move: 1 1

   1  2  3  4
1  @  _  _  _
2  _  _  _  _
3  _  _  _  _
4  _  _  _  _

AI's turn

   1  2  3  4
1  @  *  _  _
2  _  _  _  _
3  _  _  _  _
4  _  _  _  _

Your turn. Make your move: 1 3

   1  2  3  4
1  @  *  @  _
2  _  _  _  _
3  _  _  _  _
4  _  _  _  _

AI's turn

   1  2  3  4
1  @  *  @  _
2  _  *  _  _
3  _  _  _  _
4  _  _  _  _

Your turn. Make your move: 3 2

   1  2  3  4
1  @  *  @  _
2  _  *  _  _
3  _  @  _  _
4  _  _  _  _

AI's turn

   1  2  3  4
1  @  *  @  _
2  _  *  *  _
3  _  @  _  _
4  _  _  _  _

Your turn. Make your move: 2 1

   1  2  3  4
1  @  *  @  _
2  @  *  *  _
3  _  @  _  _
4  _  _  _  _

AI's turn

   1  2  3  4
1  @  *  @  _
2  @  *  *  *
3  _  @  _  _
4  _  _  _  _

AI WON! :(
```

4. Wnioski

Wraz ze wzrostem rozmiaru planszy algorytm MINMAX działa dłużej. Bez ograniczenia głębokości wywołań musielibyśmy bardzo długo czekać na wynik działania komputera. W związku z tym bardzo ważne jest ograniczenie tej głębokości, co niestety (z punktu widzenia komputera) wiąże się z jakością działania algorytmu – rozpatruje on tylko część konsekwencji wykonania danego ruchu, więc tak jak w przypadku planszy 3x3 praktycznie nie da się z nim wygrać, tak przy większych planszach taka szansa już jest. Gdybyśmy chcieli sprawić, żeby algorytm był mniej nieomylny, należałoby tę głębokość jeszcze bardziej ograniczyć.
Plansza w teorii może mieć dowolny rozmiar, ale powyżej N=10 gra zaczyna tracić sens ze względu na bardzo szybko rosnący czas działania algorytmu.

5. Opis kodu

5.1. Algorytm MINMAX

Na implementację algorytmu składają się dwa pliki – MinMax.cpp i MinMax.hh. Działanie algorytmu zostało dokładnie omówione w punkcie 2., tutaj jest jego odzwierciedlenie. Należy wskazać, że algorytm wywołujemy z dodatkową informacją typu bool, która mówi, czy gramy jako MIN czy jako MAX.

5.2. Gracz

Struktura gracz znajduje się w pliku Player.hh. W strukturze tej znajduje się ID gracza, znak gracza oraz konstruktor.

5.3. Macierz

Implementacja macierzy jest w pliku Matrix.hh. Pola tej struktury to ilość kolumn, wierszy oraz dynamiczna macierz. Związane są z nią konstruktor oraz przeciążenie operatora ().

5.4. Warunek zwycięstwa

Warunek zwycięstwa realizuję w plikach WinCondition.hh i WinCondition.cpp jako klasę, której pole definiuje wygrywającą ilość znaków. Zaimplementowałam konstruktor oraz metodę Winner, która dla danej planszy i graczy zwraca informację o stanie gry (remis, wygrana danego gracza lub tymczasowy brak rozwiązania).

5.5. Plansza

W plikach Board.hh oraz Board.cpp można znaleźć implementację klasy oraz metod związanych z planszą. Sama plansza jest macierzą wypełnioną ID graczy. Metody,

które są tu zrealizowane, dotyczą wyświetlenia planszy, wykonania ruchu, cofnięcia ruchu, zresetowania planszy, zwrócenia wielkości planszy i samej macierzy oraz sprawdzenia, czy dany ruch jest możliwy.

5.6. Zarządzanie grą

Do zarządzania grą używam specjalnej funkcji z plików GameManagement.hh i GameManagement.cpp. Funkcja ta odpowiada za komunikację z użytkownikiem – proszenie o dane wejściowe, wyświetlanie odpowiednich komunikatów, dbanie o kolejność tur, wywoływanie metod oraz funkcji niezbędnych do realizacji gry.

5.7. Funkcja main

W funkcji main znajduje się jedynie wywołanie funkcji zarządzania grą. Zastosowałam takie rozwiązanie, aby potencjalne osoby (np. ja w przyszłości), które chciałyby wykorzystać w swoich większych projektach moją grę, mogły w funkcji main zawrzeć inne funkcjonalności obok wywołania gry (np. dodanie tam kilku innych gier do wyboru).

6. Literatura:

https://pl.wikipedia.org/wiki/Algorytm_min-max

https://thecodingtrain.com/CodingChallenges/154-tic-tac-toe-minimax.html